Mapping the Depths: Visualising Mussel Bed Data with SeaVis

Victor Emil Søe Rasmussen, Thomas Edwin O'Neill,

Janus Mohr Hovgaard, Emil Stald Pedersen, Ivan Mezinov

May 23, 2023

Figure 1: AI generated image of mussels

I

Abstract

SeaVis is a company that uses underwater drones to photograph the seafloor, collecting valuable data on the location and density of mussels. Unfortunately, they lack an effective way of visualising and communicating this data to their users, limiting the ability of mussel gatherers to optimise their harvest. This project aims to provide SeaVis with a user-friendly mapping client that displays their data on the location and density of mussels on the seafloor, as well as a means to convert their raw data into a form which can be readily displayed in the TimeZero software package. The mapping client includes features such as data security and better data management and representation, helping users optimise their harvest and reduce the time and energy required to gather mussels.

Contents

1 Introduction

This project aims to provide SeaVis with a way of transforming their data into a format that is suitable for mapping, as well as a way to display it. The data transformation process includes interpolation of data points into mapped fields. A primary requirement from SeaVis states that the data has to be compatible with the software package, TimeZero, which is Geographic Information System (GIS) software used by their costumers. However, due to our desire to create a holistic solution and introduce a potential alternative to TimeZero, a decision was made to create two standalone applications. Such solution provides an implementation where the first application does the calculation work, while another securely distributes and showcases this output to the end-user. The first application, named *KML Creator* (see Section 2), is responsible for processing the data into a KML file and thus fulfilling the request from SeaVis. It also allows SeaVis to configure which data gets processed and how it gets processed.

The second application referred to as the "Mapping Client" includes features and flexibility with displayed data that could be more convenient and user-friendly compared to ones represented in TimeZero. Such features include the ability to distribute data layer security, as well as better data management and representation. Overall, by providing a visual representation of the location and density of mussels on the seafloor, it can help users optimise their harvest and reduce the time and energy required to gather mussels.

The following working questions will be used as a primary focus of this project: **How can we provide SeaVis the means to convert their raw data into a form that can be displayed in the TimeZero software package? How can we create a user-friendly mapping client which is capable of displaying SeaVis' data on the location and density of mussels on the seafloor and can be accessed securely through a login service?**

1.1 SeaVis

SeaVis is a start-up that uses underwater drones to photograph the seafloor. There are a number of applications for which this technology can be used. One of these is helping mussel gatherers locate mussel beds, which can help the mussel gatherers save time and energy (both diesel and human energy), as well as minimise the damage on the seabed habitats. During our work process, we have been in contact with Bjørn Holm, who is the development engineer for SeaVis. We have held a number of meetings with Bjørn throughout the development process of the KML creator, to refine the requirements and get feedback on the process, as well as obtaining some information regarding SeaVis. When speaking to Bjørn regarding the need for such an application, he said:

"The fishermen are doing trades, and there's a growing market for protein, and the mussels and starfish are really rich in protein. The trawlers destroy the habitats actually. So what we want to do is, that we want to limit the amount of the area that they scrape and we want to help them catch them at the right time" [1]

When fully established, SeaVis plans on creating data visualisations of all relevant areas around Denmark, by capturing images from their underwater drones, where machine learning algorithms transform them into numerical data (GPS coordinates, oxygen level, mussel size, density etc.). In order to provide their data as a consumable product, and this is how they plan to execute that strategy: "So the plan is that we make the drone and then we take some pictures of the sea floor, and then we have a machine learning algorithm to map the biomass on the pictures" [1]

The current task for SeaVis is to create an optimal way of displaying their raw data as a visualisation in TimeZero for their clients. SeaVis is currently working with one client in Limfjorden, Denmark, but the vision for the company, is to expand to the rest of Denmark in 2024, and to go international by 2025. [1]

1.2   Mussels

Mussels cluster on rocky substrates or ropes, creating dense beds that are ecologically valuable for habitat and food [2]. Mussels are capable of movement and are harvested using various methods, such as on-bottom or rope culture. On-bottom culture is traditional but labour intensive, while the rope culture is more efficient but requires more investment [3]. Mussels can also act as bio-indicators of water quality, accumulating pollutants and contaminants that could affect human health and the ecosystem. They are capable of monitoring nutrient levels, such as nitrogen and phosphorus, which contribute to harmful algal blooms. By carefully monitoring mussel populations, researchers and environmental managers can gain insights into human-nature interactions.

Gathering mussels can be a time-consuming and expensive process, especially as the mussel beds are difficult to locate. With the solution we have created, SeaVis can help mussel gatherers better locate mussels and collect valuable data on the location and density of mussels on the seafloor, but lack an effective way to visualise and communicate this data to their users. This limits the ability of mussel gatherers to optimise their harvest, potentially leading to inefficient use of time and energy.

1.3 Drawbacks of Existing Solutions

This project follows on from a previous semester project, which attempted to solve SeaVis' problem. This earlier project (referred to as "Mussels in Limfjord", henceforth) [4], implemented a map-based data visualisation of mussel density, encased in a graphic user interface (GUI). Although the GUI allowed the user to filter data into categories, the software implementation as a whole had a number of shortcomings. For instance, the data visualisation itself was not visually appealing and gave the viewer a distorted view of the raw data. As far as solving SeaVis' issue, Mussels in Limfjord fell short.

1.4 Requirements From SeaVis

The starting and primary goal of this project has been to follow and satisfy the main requirement outlined by SeaVis. It includes the following:

Make a programme that is capable of converting a CSV-formatted data-set obtained from SeaVis into a file that is compatible with TimeZero – an application used to display the data for potential users.

As the project developed SeaVis provided some suggestions regarding certain aspects of the programme:

- An output file can be in the KML format, since it is one of the most convenient formats that could be imported into TimeZero
- The inclusion of forecasting what plotted data would look like after a certain amount of mussel growth.
- The option to display the data on a 3D map of the ocean floor.

During our collaboration with SeaVis, we had meetings with Bjørn to gather useful feedback and suggestions for improving our final product.

1.5 Our Requirements

Throughout the course of this project's development, it was decided to introduce additional requirements to expand the scope of the project. This was done both to satisfy the requirements of the course and to give us more of a challenge and an opportunity to learn.

Our primary requirement was to create a holistic solution that would ensure that raw data can be formatted, distributed securely, and displayed without the need to use TimeZero or other third-party tools.

More specifically, we outline the following requirements for the data conversion process:

- Implement the technique that would allow us to comprise discrete geographical data points during the data processing stage.
- Make an executable (.exe) formatted application compatible with Windows and Linux operation systems
- Make sure that the raw data can be customised and user's input can be saved
- Introduce various user-friendly GUI components

For the standalone mapping client the following requirements were introduced:

- Make a separate mapping client that would be able to display the processed data to provide an alternative for TimeZero.
- Within the mapping client implement user-friendly GUI
- Include user validation and data security features for the mapping client, to protect the intellectual property contained in the generated KML files.

During the development of the project, intermediate goals and requirements were constantly introduced and updated (see Section 6).

1.6   Summary of Content

Section 2 provides an in-depth analysis of the KML Creator application, exploring its features such as spatial interpolation, polygonisation, the MVP design pattern, and the Model's role in CSV to KML conversion. In Section 3, the focus is on the mapping client, diving into its interface, main routes, the authentication system with user registration, login, and token authentication using JSON Web Tokens, as well as the secure retrieval of KML files from a private S3 bucket. Section 4 offers an overview of the tools and third-party packages used in the project, including geospatial data manipulation libraries, spatial operation tools, and visualisation packages. Moving on to Section 5, it covers various aspects of testing, user perspective testing, compatibility and functionality testing of the KML Creator and mapping client, parameter testing, security testing, and unit and integration testing. Section 6 sheds light on the project's development process, discussing challenges faced, decisions made, workload distribution, and future development possibilities, while also acknowledging the reliance on third-party tools and libraries. The report concludes in Section 7 by highlighting the successful creation of two applications that both meets SeaVis' requirements as well as our own.

2 KML Creator

The KML Creator is the first of the two apps we have created. It is used to transform the data collected by SeaVis into a more suitable format for mapping.

The KML Creator consists of approximately 2500 lines of code [1], written in Python[2]. The code itself has the following features:

- The code is well commented and each method and class includes a doc string.
- Extensive type annotations are used. Although type annotations are not essential in a dynamically typed language like Python, we believe that there are valuable advantages to using type annotations, namely increased readability and better IDE support
- We have used the Pre-commit library [5] and a number of pre-commit hooks to ensure standardised formatting, allowing for easier-to-read code and easy development through minimising the size of GitHub diffs.
- Logging is used throughout the codebase to ensure easier bug and error detection.
- File paths are captured using Path objects form Python's pathlib module allowing the programme to run on Windows, Mac, or Linux.
- Rather than being hard coded throughout the codebase, capitalised global variables (e.g. "CSV_PATH", "TEST_PATH") are defined in config.py, allowing for standardisation and easier configuration.

The requirements that we used for the development of this app were that:

- it would be able to import CSV data
- there would be a way of splitting data into separate bins, with separate map layers relating to each bin. Each bin should have a lower and upper bound. There should be a way for a SeaVis user to configure these bins.
- it would be able to take the discrete points found in the CSV file and interpolate values for the areas in between each point, creating mapped fields.
- it would be able to create output that could be imported into TimeZero and our mapping client.

There are a few theoretical concepts that need to be explained, in order to fully grasp this process, we have devised to fulfil the requirements above. These concepts are explained in Section 2.1 through Section 2.4.

---

[1] Maximum line length of 88 characters. Including comments.
[2] The GitHub repository can be found at the following link: http://github.com/figgeous/kml-creator.

5

2.1 Spatial Interpolation

An interpolation algorithm is a mathematical algorithm used to estimate or calculate values between two known data points. In other words, it is a method used to approximate unknown values within a range of known values.
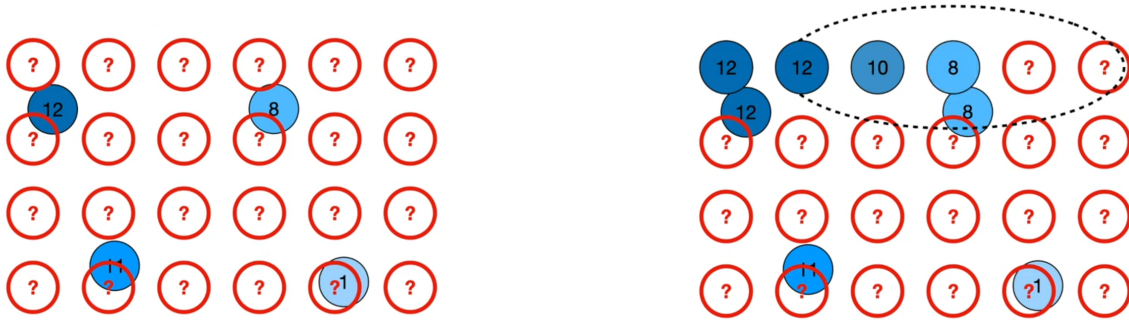
Figure 2: Illustration of the moving average algorithm [6]

In the figures above, the observed values are represented by the blue dots, and the unobserved values are represented by the red dots with the question mark. The search ellipse is represented by the black dotted ellipse, which include the number of observed values we want to include in the calculation. We then place the search ellipse over the observed values, with the unobserved value at the centre. We then calculate the average of the values within the window and use that average as our estimate for the unobserved value.
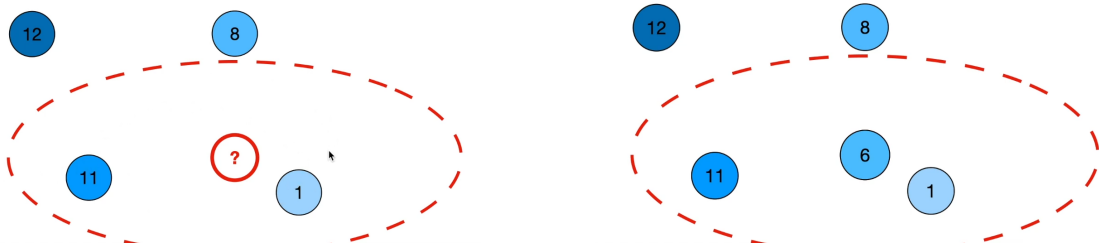
Figure 3: Illustration of the moving average algorithm [6]

In the second illustration of Figure 2, the search ellipse only contains one observed value, which has the value 8. Therefore, the value of the unobserved value also becomes 8. The same concept can be seen in the two illustrations above, in Figure 3. Here the search ellipse contains both 11 and 1, which averages to 6. Therefore, the unobserved value becomes 6. This process is then repeated for all unobserved values. Depending on the size of the data-set and the settings of one's algorithm, the calculations can take a very long time, as there can be millions or even billions of unobserved values.
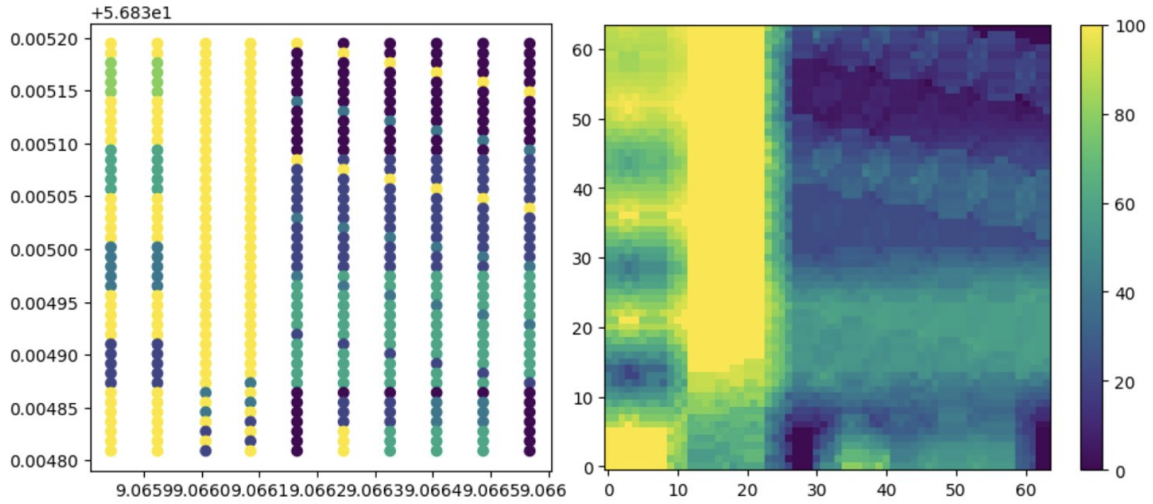
Figure 4: An illustration of our own fabricated data-set

There are different types of interpolation algorithms, including Linear interpolation, Inverse Distance interpolation, Moving Average interpolation, and Nearest Neighbour interpolation. In this project the moving average interpolation algorithm was used. This algorithm works by taking a search ellipse of observed values and averaging them to estimate the value the unobserved location. The size of the search ellipse can vary depending on the specific application and the desired level of accuracy.

As we are using GDAL for our project, the moving average algorithm is easy to implement into our code. However, using algorithms like moving average, has its limitations. For example, it assumes that the variable being interpolated varies smoothly across the space, which may not always be the case. Therefore it can also result in over- or under- smoothing, leading to inaccurate estimates.

2.2 Polygonisation

Polygonisation is a key step in converting GeoTIFF files back into vector data using GDAL. This process involves transforming raster data into polygonal representations for analysis and visualisation.
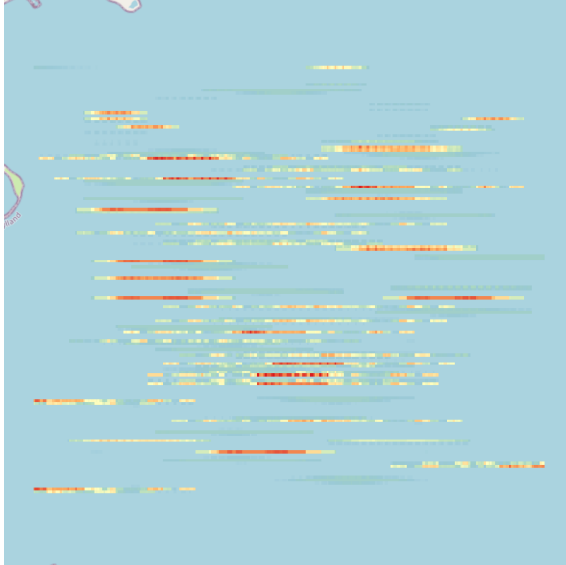
Figure 5: Illustration a GeoTIFF file before polygonisation
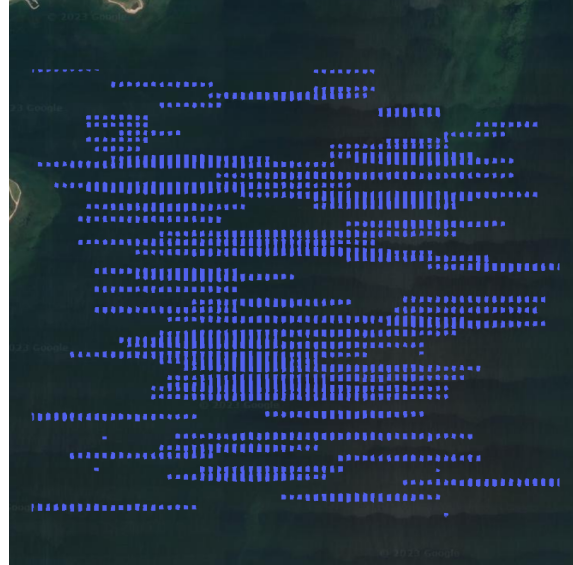
Figure 6: Illustration of a KML file made from the previous GeoTIFF file.

The GDAL polygonize utility [7] utilises various algorithms to convert raster data into vector polygons. While the specific algorithm used may vary depending on the GDAL version and the underlying libraries, the utility effectively identifies distinct regions within the raster and generates polygons to represent them. By analysing the raster data, the polygonize utility groups pixels with similar values or attributes into separate regions. These regions are then represented by polygons that describe their boundaries. The resulting vector polygons accurately capture the features present within the GeoTIFF file. The polygonisation process provides a convenient and efficient means of converting raster data into vector polygons. This vector data can then be further processed and analysed, enabling spatial analysis, measurements, and the generation of high-quality maps or visualisations within the mapping client.

2.3 Simplification and Polygon Unification

The simplify() method is part of the Shapely library (see Section 6). It is mainly used to get rid of points that may be unnecessary for representing the object and return a simplified version of the geometry object. The usual representation would be: object.simplify(tolerance, preserve_topology=True), where tolerance is the value stating the distance of the original geometry,

hence all points would be within this distance, and preserve topology that ensures that a slower algorithm is used to preserve topology. If the condition is set to false, the Douglas-Peucker algorithm is used, which is faster than the default one [8].
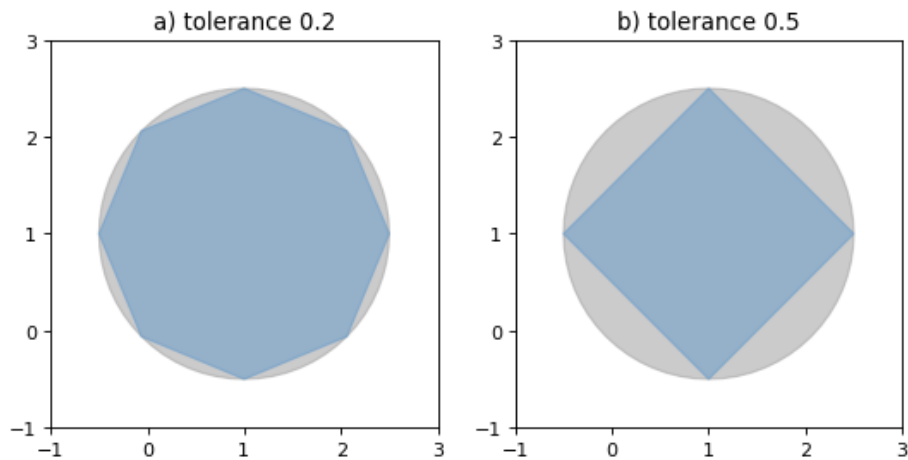
Figure 7: Simplification of a nearly circular polygon using a tolerance of 0.2 (left) and 0.5 (right) [8]

The other relevant segment of the Shapely library is the "unary_union" function. It is used to perform a union operation between two separate polygons. The function ignores None values, allowing the intersecting polygons to be merged into one polygon by combining common points and dissolving specific edges [9].

2.4 Model-View-Presenter

The KML Creator uses a Model-View-Presenter (MVP) design pattern, which separates logic into three distinct components [10]:

- The Model contains the data and business logic required by the View, although it is unaware of both the Presenter and the view. In the KML Creator the Model comprises three classes (Bin, Preper, Runner), which are capable of the main objective of the app: to convert a suitably formatted CSV file into a TimeZero-compatible KML file.
- The View, with its simple design, represents the presentation layer and does not contain control logic. It makes requests to the Presenter, from which it is updated. The View is aware of the Presenter, but is unaware of the Model. In the KML Creator, the user interacts most directly with the View class, which calls methods of the Presenter class, which in turn calls methods of the View class, updating its state.
- The Presenter mediates between the Model and the Presenter, linking the two separate components. In the KML Creator, the Presenter receives user input from the View, utilises the Model and updates the View through calling its methods.

An important feature of MVP is that there is a clear separation of concerns between the three distinct parts. That is, each of the three parts has its own distinct role and duty. This leads to numerous advantages and disadvantages: Advantages:

- it is easier to conduct unit tests on separate components, which are independent of one another.
- detection and location of bugs and errors is more straightforward, as they can be localised to a specific component.
- the app is more scalable and easier to maintain, as each part of three components can be modified and developed independently of the rest.
- code is more readable and easier to understand when there is clear separation of concerns.

Disadvantages:

- separating concerns into three components could add complexity, which you may want to avoid in smaller projects
- additional code often needs to be written in order to implement MVP, leading to an increase in code volume, which can have undesirable effects on maintainability.
- getting familiar with typical implementations of MVP involves a learning curve for developers who were previously unaware of it.
- there is a potential for over-engineering through excessive abstraction, leading to unnecessary complexity and also having a negative effect on maintainability.

Model-View-Controller (MVC) is another design pattern we considered using. The Controller of the MVC is similar to the Presenter of MVP, with the main difference being that in MVC the View is able to communicate directly with the Model. This makes the three components more interdependent and intertwined, which means that MVC does not enjoy the same advantages as MVP with regard to the clear separation of concerns. In our application there is no added benefit to the View being able to communicate with the Model, and thus we opted for MVP.

2.5 Model: from CSV to KML

Having explained some of the foundational theoretical concepts important to the KML Creator, we now explore the work flow the Model uses to convert CSV files into KML files. A sample KML can be found in Section 8.4.
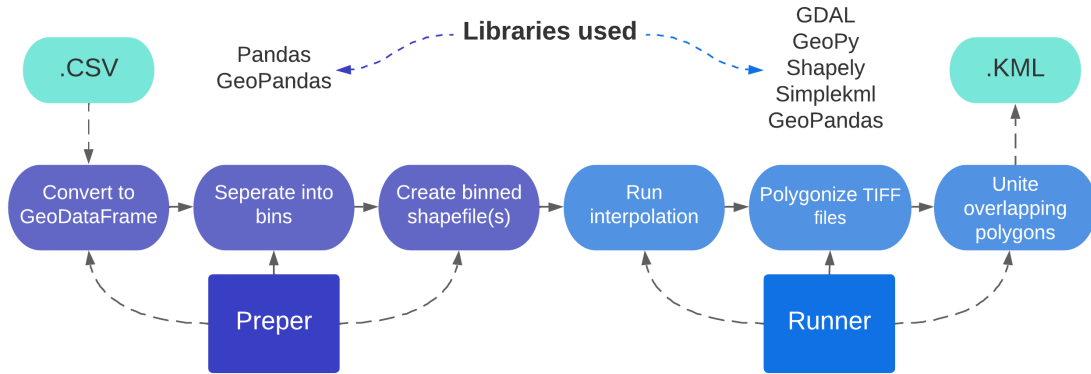
Figure 8: The six-step process to convert CSV to KML. Relevant libraries are included.

Figure 8 shows this six-step process. These steps are carried out by the Model's three classes: Bin, Preper, Runner. The steps will be described by exploring the capacities of each of these classes.
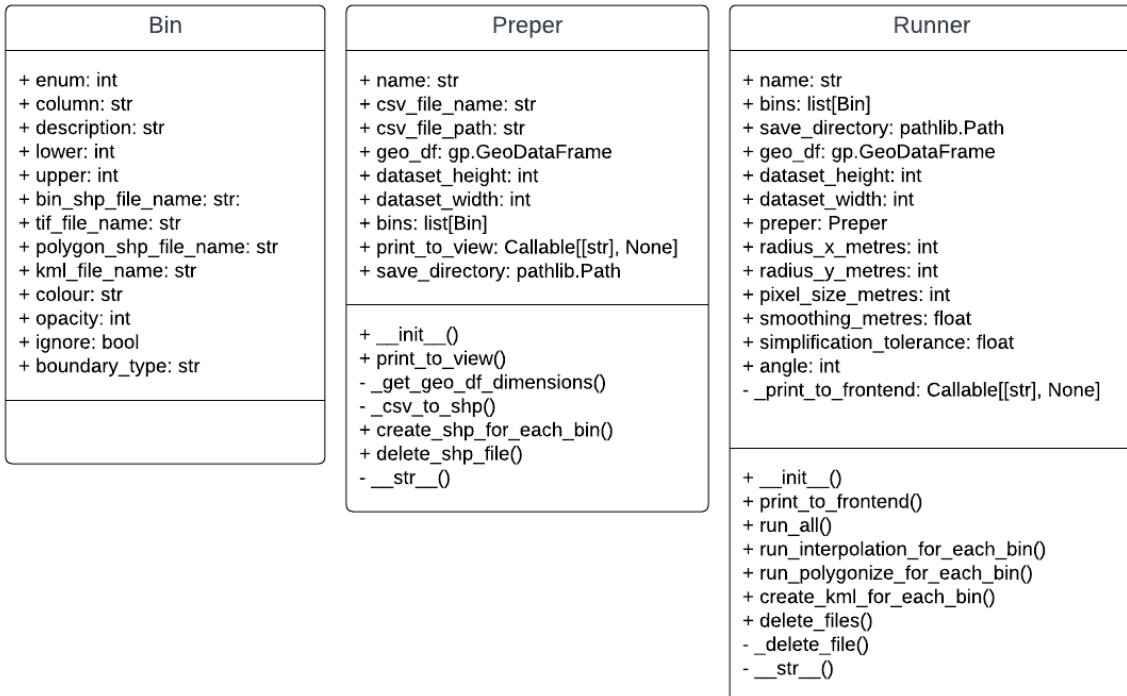
Figure 9: Class diagrams of the Bin, Preper and Runner classes. Here "gp" refers to Geopandas.

Bin   Bin is a Python data class, which acts as simple data container for the user-configured bins. Each bin represents a single layer generated by the Model. The most import attributes of the Bin are the lower and upper bound, which specify the boundaries for values included in the eventual map layer, captured by a single KML file. Its other attributes include: column name, description, colour, opacity, etc.[3]. The following code shows sample input for the initialisation of a Bin object:

```
Bin(
    enum=1,
    column="Blue Mussel Density",
    description="The density of Blue Mussels between 20 and 40",
    lower=20,
    upper=40,
    colour="3B9C17",
    opacity=50,
),
```

The map layer generated from this bin will be semi-transparent (opacity 50), green (hex colour "3B9C17") and show Blue Mussel density with values greater and equal to 20 and lesser and equal to 40.

During the conversion from CSV to KML, the bins are first populated by user input, then passed to the Preper, which updates the Bins' attributes before the bins are passed to the Runner. The Runner also updates the Bins' attributes for its own internal reasons. Most of these updates involve setting file paths for the various saved files created, allowing them to be later located and used. Another attribute of Bin is 'ignore', which is set to true when an empty data-set is created by any of the components. This signifies an empty Bin (no values between the lower and upper bounds) and the Bin is ignored for the remainder of processing.

Preper   The Preper is responsible for converting the data imported from the user-specified CSV file into a format that is more amenable to geospatial computations. This class imports the data from the user-specified CSV file, converts it to a GeoDataFrame (see Section 4.1.1) and saves it to the hard-disk as a compressed Shapefile. It then splits the data-set into separate GeoDataFrames, reflecting each user-configurable bin, containing data between (and including) the Bin's two bounds. These additional GeoDataFrames are also saved to disk, with their names set as the Bin's `bin_shp_file_name` attribute. The Preper computes the geographical dimensions of the data-set, for later use by the Runner.

---

[3]For a full list of attributes, see the code in Section 8.7

Looking at Figure 9, the Preper has two private methods [4]: _csv_to_shp converts the specified CSV file into a Shapefile and _get_geo_df_dimensions computes the geographic extents of the data-set. The crate_shp_for_each_bin method takes the Shapefile created by _csv_to_shp and creates separate Shapefiles pertaining to each Bin, containing a value within the Bin's lower and upper bound. The delete_shp_file method is called at the end of the main process to delete the Shapefiles generated by create_shp_for_each_bin.

Both the Preper and the runner have a print_to_view method, which is used by each class as well as the Model classes, in order to print text to the View's console window.

Runner The Runner takes the Shapefiles, prepared by the Preper, and performs four operations on them: interpolation, polygonisation, unison of overlapping polygons and creation of KML files. Each of these operations is carried out on each Bin object, taking the user-specified attributes of the bin into account. As described in Section 2.1, spatial interpolation is a technique used to estimate values for locations where data is not available. In this case, interpolation fills in the spatial areas between each given point for which we have no data. The interpolation process produces a GeoTIFF file. The next step, polygonisation, involves converting raster data (GeoTIFF) to vector data, creating an array of geometrical objects (polygons). Using the process detailed in Section 2.2 and 2.3, overlapping polygons are simplified and united and can be used to create our final product: the KML file. KML files are easily imported into Google Maps and TimeZero.

Looking at Figure 9, the Runner has methods for running the main process:

- run_interpolation_for_each_bin
- run_polygonize_for_each_bin
- create_kml_for_each_bin

All three of there method can be called with the simple run_all method. When the main process is completed, the delete_files method is called to delete the accumulated Shapefile and TIFF files. The delete_files method calls the private _delete_file method to delete individual files.

---

[4]Although Python does not have a strict access control mechanism, we use the convention of affixing an underscore to attributes and methods which are considered private

2.6 Presenter

The Presenter mediates between the View and the Model and controls the flow of logic in the application. The three classes of the Model are imported into the Controller and their various method called during the main process.

Presenter
+ view: KmlCreatorGui
+ __init()__ + clear_temp_files() + load_gui_stats() + get_target_columns_and_default_values() + initialize_and_run_process() + print_to_view_console() + run_process()
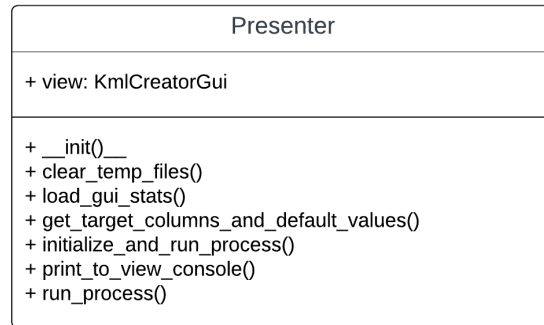
Figure 10: Class Diagram of the Presenter class

The Presenter has a method for saving the View's state to a JSON file and another for loading the same state. The clear_temp_files method clears temporary files which may have been left behind in a previous running of the application. The get_target_columns_and_default_values method fetches data from the user-specified data-set, in order to populate the View table's rows with data (target columns and min/max values for the lower and upper bounds). Using threading, the initialize_and_run_process starts a thread in which the run_process is called. The run_process method calls and initialises the Model classes and calls the method in order to convert a given CSV file into a KML file.

2.7 View

The view is responsible for the graphic user interface (GUI) and does not contain control logic itself. The user interacts with the View, which makes requests to the Presenter, updating the state of the View. The GUI can be seen in Figures 11 and 12.

14
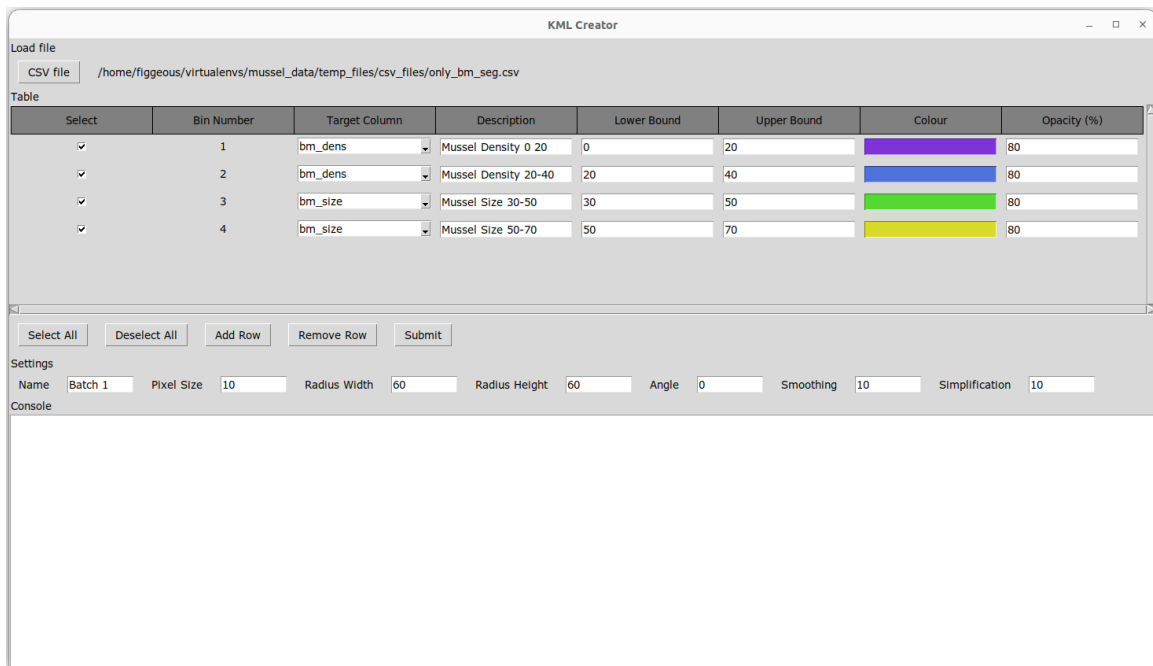
Figure 11: Screenshot of KML Creator with four bins entered. Before processing.
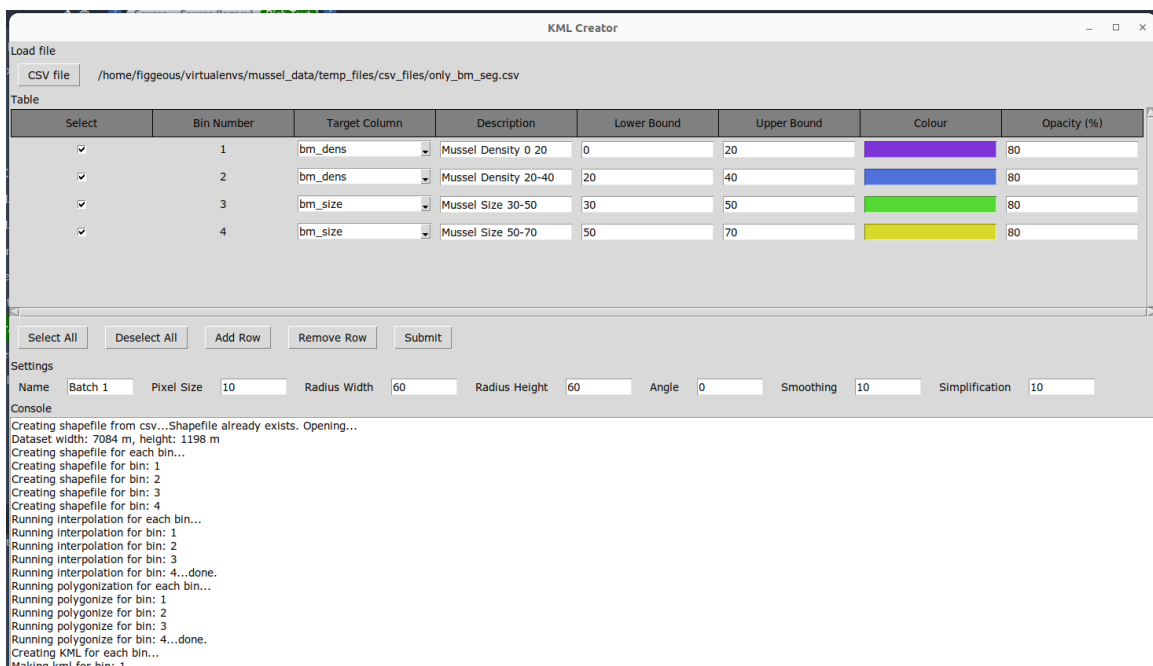
Figure 12: Screenshot of KML Creator with four bins entered. After processing.

As the View makes use of the tkinter library, the Widget object is the main component class. The Widget is the parent class for both the Frames that house the various buttons and entry objects, as well as many of the entry objects themselves. The main class in View is the KmlCreatorView class,

15

which inherits from Tk, which is tkinter's main toplevel widget, representing the main window of an application. The main window is split in five regions, which comprise of tkinter Frame objects, spatially stacked upon one another: Header, ButtonFrame, TableFrame, SettingsFrame, ConsoleFrame. These Frame objects have their own methods for handling their various buttons and entries.

KmlCreatorGui	TableFrame	TableRow
+ controller: Controller + settings: dict[str, str] + csv_file_path: str + saved_rows: dict[str, dict[str, str]] + header_frame: tk.Frame + table_frame: tk.Frame + console_frame: tk.Frame + button_frame: tk.Frame + settngs_frame: tk.Frame + submit: tk.Frame + submit_frame: tk.Frame	+ master: KmlCreatorGui + rows: list[TableRow] + width: int + cell_width: int + cell_height: int + button_width: int + button_height: int + target_columns: list[str] + saved_rows: dict[str, dict[str, Any]] + column_min_max: dict[str, tuple[float, float]] + canvas_frame: tk.Frame + canvas: tk.Canvas + title: str + scrollbar_v: int + scrollbar_h: int	+ master: TableFrame + check_box: CheckbuttonWithVar + bin_number_label: tk.Label + target_column_combo_box: TargetColumnCombobox + entry_colour: ColourEntry + opacity_entry: tk.Entry + description_entry: tk.Entry lower_bound_entry: LowerAndUpperBoundEntry upper_bound_entry: LowerAndUpperBoundEntry
+ __init__() + create_table_and_accessories() + ask_user_file_directory() + ask_user_path() + get_rows() + get_csv_file_path() + get_settings() + set_csv_file_path() + set_target_columns() + set_saved_rows() + set_col_min_max_values() + set_settings() + submit() + print_to_console() + on_close()	+ __init() + create_scrollbars() + create_headers() + add_row() + remove_rows() + get_rows() + select_all() + update_scrollregion()	+ __init_() + make_canvas_window() + create_entries() + get_row_data() + destroy_entries() + is_selected() + contains_empty_entry() - _convert_field_to_datatype()

Figure 13: Class Diagram of the KmlCreatorGui, TableFrame and TableRow. Here "tk" refers to tkinter. Only a selection of the View's classes are shown, as the ones that are not shown as fairly simple and do not require a lot of explaination.

KmlCreatorGui

The KmlCreatorGui, as shown in Figure 13, is the main class for the view. It contains methods for creating the View's various elements (create_table_and_accessories), asking for user input (ask_user_file_directory and ask_user_file_path). There are various methods that get and set data in the various frames: get_rows, get_csv_file_path, set_csv_file_path, set_target_columns, set_saved_rows, set_col_min_max_values and set_settings. The print_to_console method is used by the Presenter (and, indirectly, the Model) to output text to the View's console and the on_close method contains the code run when the programme is exited. When a user clicks the View's "Submit" button, the submit method is run, creating a Submit instance. Upon initialising, the console box is cleared, the "Submit" button disabled, the user is asked to select a save directory, data is collected from the all the View's entry widgets and passed to the Presenter's initialize_and_run_process method.

16

Header Contains the button and label for loading a CSV file. When the button is clicked a file dialog opens, asking the user to locate a CSV file. The class calls the Presenter's load_gui_state method is called with the CSV file path as an argument. The presenter loads the CSV file and updates the status of the View by calling the relevant View methods.

ButtonFrame A simple method-less class containing the "Select All", "Deselect All", "Add Row", "Remove Row" and "Submit" buttons.

TableFrame A Frame which contains the table headers and rows. Each row in represented by an instance of TableRow class. Its class diagram is found in Figure 13.

The TableFrame class has methods for creating table headers, the table's scrollbars and the select_all method is used to select or unselect all rows. Rows are added and removed with add_row and remove_rows and update_scrollregion is used to update the scroll bars with the additional or removal or rows. Adding rows involves creating a new TableRow object, whereas removing rows involves destroying the entries associated with a TableRow instance and then redefining the rows attribute. The get_rows method iterates through the rows attribute and calls the get_row_data method on each TableRow object. This is used by the Presenter to collect user-input prior to processing from CSV to KML. The get_rows method is also used upon closing the programme, when the state is saved to disk.

TableFrame: TableRow The TableRow represents a single row in the table. Its class diagram in Figure 13 The TableRow class has methods for creating and destroying entries, which are the cells that populate each row. The entries themselves are of varying type, some of which are inbuilt tkinter classes (Label, Entry), and others are classes we have defined (CheckbuttonWithVar, ColourEntry, LowerAndUpperBoundEntry). These entries are placed onto their own respective position in the GUI using the make_canvas_window method. The is_selected method returns boolean value indicating whether the checkbox for a given row is selected or not and contains_empty_entry returns a boolean value indicating whether all the entries of a given row are filled. When submitting user configuration for the processing of CSV to KML, these two methods allow the inclusion of selected rows and can ensure that these rows are filled with data.

SettingsFrame A simple class that contains the following settings

Setting name	Description
Name	Allows user to add their own text prefix to the output KML file. May be used to help identify the output of separate runnings of the programme (e.g. "Batch 1", "Batch 2", etc.).
Pixel Size	The width and height of the pixels generated by the interpolation process. Modifying this has a considerable effect on processing time.
Radius Width	The longitudinal distance (in metres) of the search ellipse used in interpolation.
Radius Height	The latitudinal distance (in metres) of the search ellipse used in interpolation.
Angle	The angle of the search ellipse used in interpolation.
Smoothing	The smoothing factor applied to the interpolation process. Higher values generate a smoother output.
Simplification	The simplification factor used in the polygonisation process. Higher values generate polygons with fewer points.

Other than the constructor class, SettingsFrame has a single method get_settings, which is used by the presenter to fetch the user-inputted settings from each of the entry fields.

ConsoleFrame A simple class that contains the console text and entry widgets. Sample output after processing can be found in Figure 14

```
Creating shapefile from csv...Shapefile already exists. Opening...
Dataset width: 7084 m, height: 1198 m
Creating shapefile for each bin...
Creating shapefile for bin: 1
Creating shapefile for bin: 2
Creating shapefile for bin: 3
Creating shapefile for bin: 4
Running interpolation for each bin...
Running interpolation for bin: 1
Running interpolation for bin: 2
Running interpolation for bin: 3
Running interpolation for bin: 4...done.
Running polygonization for each bin...
Running polygonize for bin: 1
Running polygonize for bin: 2
Running polygonize for bin: 3
Running polygonize for bin: 4...done.
Creating KML for each bin...
```

Figure 14: Sample output in the ConsoleFrame.

2.8 Extra Features

The KML Creator has three extra features which were implemented to enhance the usability of the app.

2.8.1 Save/Load State

As it can be a somewhat laborious task to input the values related to each bin into the View, we implemented a save and load state. When the user exits the programme, the View's on_close method is called, which in turn calls the Presenter's save_gui_state method. This Presenter collects user input by calling the View's get_csv_file_path, get_rows and get_settings methods, saving the returned data to a JSON file. When the programme is opened again, the Presenter's load_gui_state

method is called, which in turn calls the View's set_csv_file_path, set_rows and set_settings methods, populating the various user input entry boxes with saved data. If the JSON file does not exist or is unsuccessfully accessed the saved state is not restored and the user is shown a window without saved data.
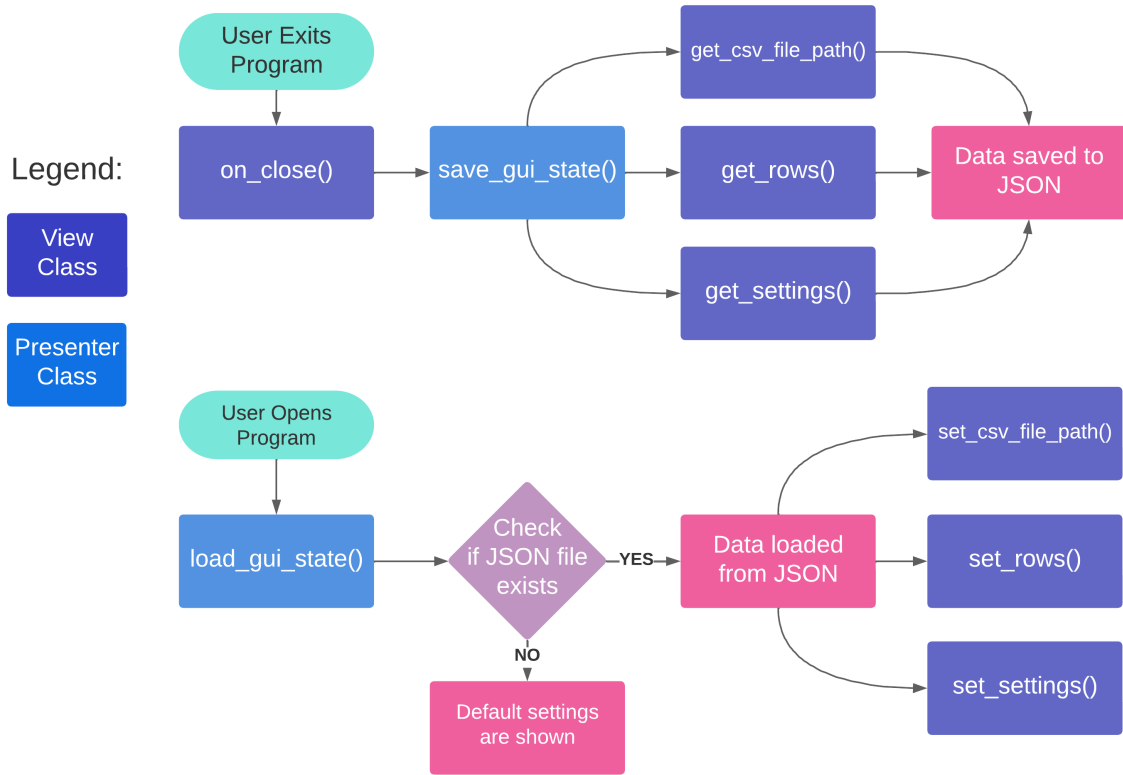
Figure 15: Flowchart of the save and load state

2.8.2 Preloaded Target Columns and Boundary Values

When a user selects and loads a CSV file, two data points are taken from this file, in order to generate content for the table's rows. These data points allow for the feature shown in Figure 16.

Figure 16: Demonstration of the minimum and maximum values for the "bm_dens" and "bm_size" target columns. These values were selected from the combo box, which is populated with the column names from the user-selected CSV file.

19

- The first datapoint is the column names. As the configuration of each bin pertains to the data found in a single column of the CSV file, the column name needs to be set for each row in the table. The column names fills the "Target Column" combo box, allowing for easy selection. The use of a combo box also limits the possibility for the user to input invalid column names and reduces the need for user input validation.

- The second datapoint is the minimum and maximum values for the given target column. When a column is selected from the "Target Column" combo box, the column's minimum and maximum values are shown in the "Lower Bound" and "Upper Bound" columns. This feature was requested by SeaVis during one of our final meetings and helps the user specify sensible boundary values.

2.8.3 Submission Spinner and Console Output
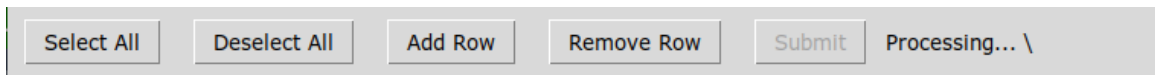
Figure 17: The SettingsFrame during processing. The spinner, situated to the right of the "Processing..." text, captured during its spin. The "Submit" button is disabled during processing.

After a user has selected a CSV file and input values into the table and settings, the click the "Submit" button to begin data processing. This process can take some time (in our experience, up to five minutes) and we thought it advisable to create a visual representation of the status of the processing. Consequently, we created a spinner, captured in its static state in Figure 17, which is active during processing. During this time the "Submit" button is disabled and various update messages are shown in the console (see Figure 18).

Figure 18: The ConsoleFrame during processing, containing sample output.

For the View not to freeze and allow for the operation of the spinner and the console, Python's in-built threading module to run processing. Threading allows simultaneous task handling by facilitating the execution of multiple concurrent threads within a single programme. The Presenter's initialize_and_run_process method starts the thread with the Presenter's run_process method and the thread exists until that method returns a value.

3 Mapping Client

The Mapping Client is the second of the two applications we have created. It is used to display the data transformed by our KML Creator on a map. The front end of the Mapping Client consists of approximately 1000 lines of code[5], which is written in JavaScript, HTML and CSS[6]. We then have approximately 800 lines of code[7] covering our Lambda functions which serves as our back end, these are written strictly in JavaScript. The code is well commented throughout and organised in a modular manner, separating each component based on its purpose.

The Mapping Client was built with the following requirements in mind:

- It should be able to display KML files on a interactive map in the sense that the user can move around freely on the map, zoom in and out, etc.
- The map should be integrated into a user-friendly GUI
- The data should be safe behind a user authentication system
- The KML files should be served to the end-user in a secure manner from an external point, protecting the intellectual property of SeaVis

During the development of the Mapping Client, the above requirements were broken into smaller tasks, which added up to the whole application. The Mapping Client allows the user to register and login through a GUI, as seen on Figure 19 and 20. When successfully logged into the Mapping Client, the user will be met by the actual client, which is only accessible when logged in with a valid token. The client page contains an interactive map, which has a button from which the user can choose which files they would like displayed on the map, this can be seen on Figure 21. These files are served from an external source, consisting of the KML files created using our KML Creator. The idea is that SeaVis themselves can update these files externally whenever new data-collections have been conducted. This allows SeaVis to control exactly which data is shown to the end-user in a secure manner.

---

[5]Excluding comments.

[6]The GitHub repository can be found at the following link: https://github.com/janusmh/SMP-CS-Mapping-Client-.
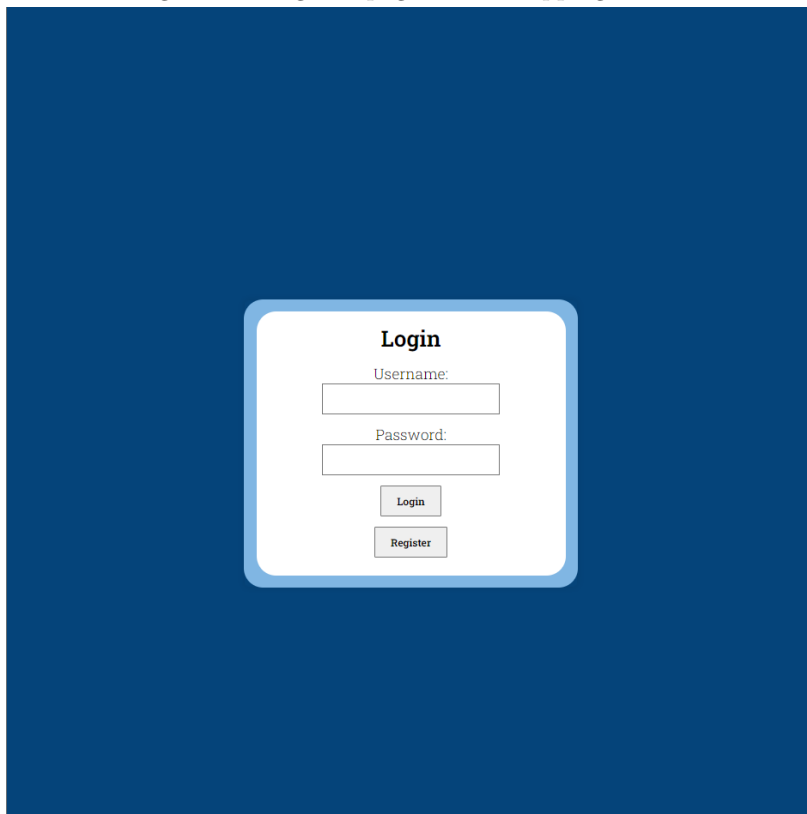
[7]Excluding comments.

Figure 19: Register page of the Mapping Client

Figure 20: Login page of the Mapping Client
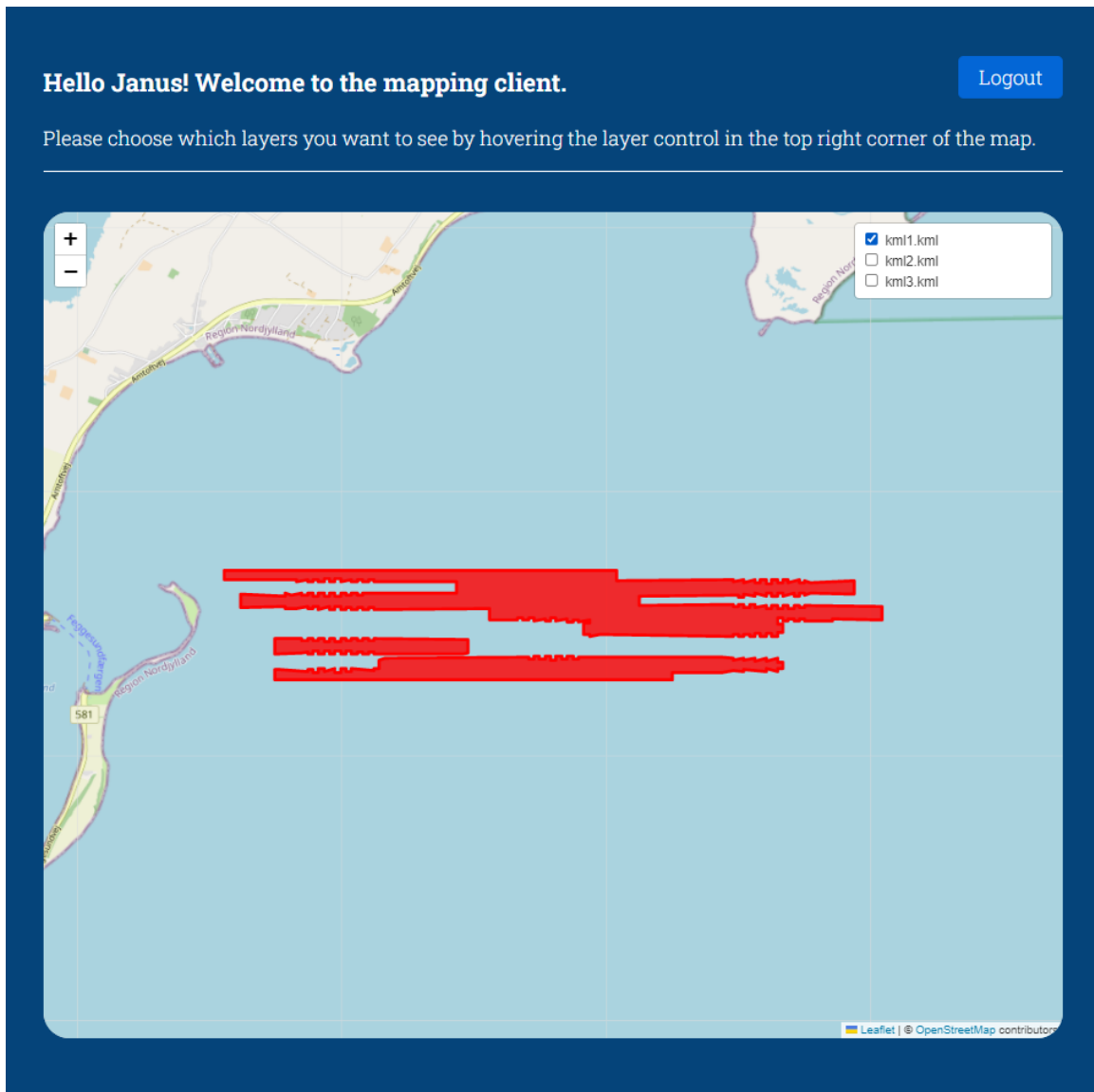
Figure 21: Client page of the Mapping Client, the red coloured polygon shows a layer imported from a sample KML file.

In the following sections, we explore the final product of the Mapping Client, going through the three main parts of the application:

- Interface
- Authentication
- File fetching

The different aspects of the three above parts are examined and thoroughly explained.

3.1 Interface

The interface of our Mapping Client is designed to be simple, intuitive and user-friendly. Its purpose is to allow SeaVis' users to access processed mussel data in a visual way, while keeping the data secure in order to protect the intellectual property of SeaVis. We have taken a modular approach, by setting up components for each page and then handling the routing using the React Router library. We have defined three main routes for navigating our components: the registration page, the login page and the private client. Each of the routes have assigned either PublicRoute or PrivateRoute, which are two custom components we've created to manage access. The two components verifies the authentication status of the user based on the presence of a token. In the coming subsections, we explore the three before-mentioned routes, which makes up the three pages of our application.

3.1.1 Use-case Diagram
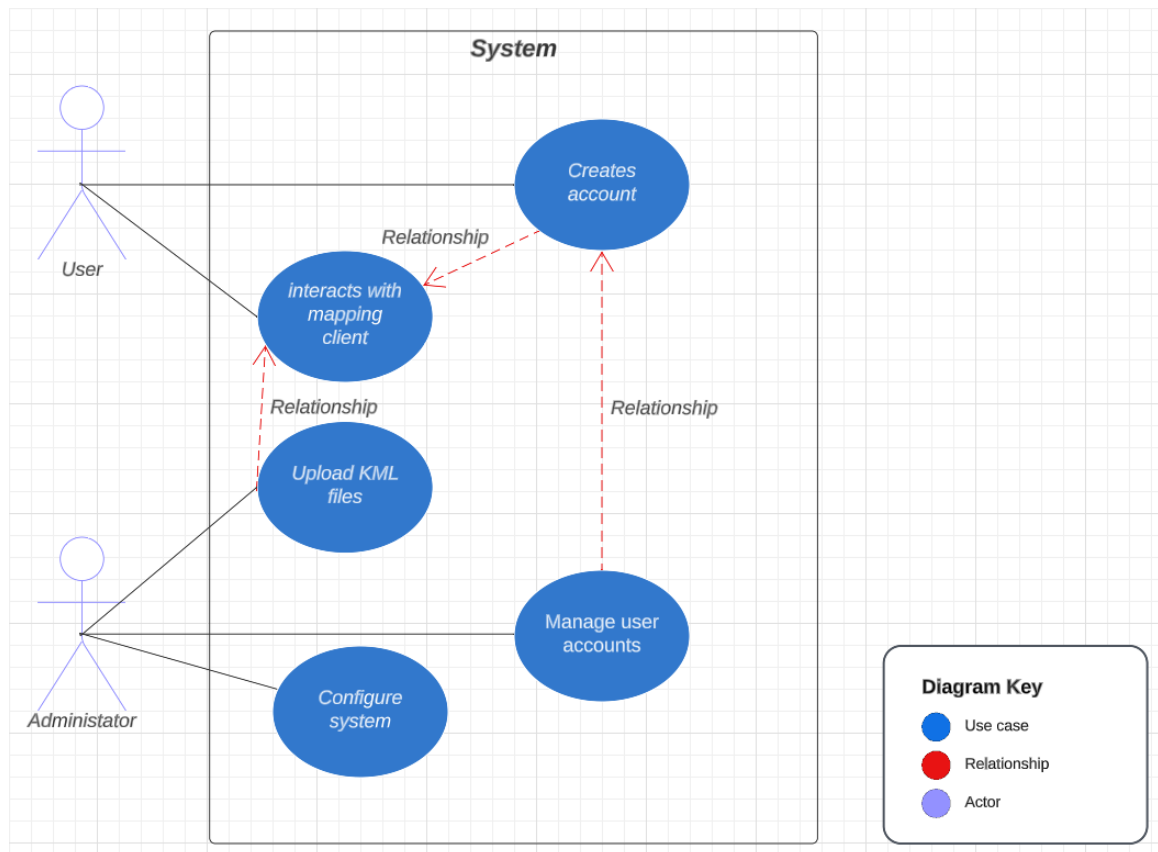
Figure 22: The use-case diagram presented here illustrates the key functionalities and interactions of the Mapping Client.

The use-case diagram on Figure 22 provides an overview of the system's behaviour from the perspective of different actors and highlights the main use cases supported by the application.

Actors:

- User: Represents the end-user of the mapping client. They interact with the system to view and interact with the map and the KML files displayed on the map.
- Administrator: Represents an administrative user of the mapping client, which is SeaVis. They have additional privileges and can manage user accounts, upload KML files, and configure the system.

Use cases:

- Display Map and KML Files: This represents the functionality of the mapping client to display maps and KML files. Both the User and administrator can access this use case to visualise the map and associated KML files.
- Interact with Map: This represents the ability of both the User and administrator to interact with the map. It includes functionalities such as zooming, panning, selecting features, and accessing additional information.
- Manage User Accounts: This is specific to the administrator and allows them to manage user accounts within the mapping client. They can perform actions like giving permission to register, modifying existing accounts, or deleting accounts.
- Upload KML Files: This is also exclusive to the administrator and enables them to upload KML files into the mapping client. This functionality allows the administrator to add new spatial data to the system for display and analysis.
- Configure System: This is also only restricted to the administrator and involves configuring the system settings of the mapping client. And it also allows the administrator to modify various parameters and preferences to customise the behaviour of the application.

3.1.2 Registration page

The registration page, as shown in Figure 19, allows new users to create an account by providing the necessary information for creating a user. The user data is filled out in a form, which has a submitHandler [11] function attached that first performs validation checks of the input and then makes a HTTP request, this is explained in more detail in Section 3.2.1. In our current product, the registration page is a part of the PublicRoute, making it accessible to anyone. This is merely for testing and developing purposes and in a final product it should be SeaVis themselves who chooses who has access to the registration path.

3.1.3 Login page

The login page, as show in Figure 20, is designed to authenticate the users of the Mapping Client, by validating credentials they have registered with. The credentials are filled in a form and the input is passed on via a HTTP request as described in detail in Section 3.2.2. When a user is successfully authenticated, they are redirected to our private client. If a user is logged in, we make sure that they can no longer access the login page and vice versa.

3.1.4 Client page

The private client page, as shown in Figure 21, is protected by our PrivateRoute element, making sure it is only accessible with a token. The client page holds a welcome message including the name of the user, followed by instructions of how the client is to be used. A map is also displayed using Leaflet, which has the KML files displayed on top as layers. The user has the option of turning different layers on and off, so that they can specifically which data they want displayed on the map. This way the Mapping Client allows the user to pick out what data they would like to focus on whether it being mussel density, mussel size, oxygen levels or something else.

The overall design of the Mapping Client is aimed to be simple, leaving room for implementing new features and paths in future versions. We have taken a modular approach to route management, using the React Router library, which makes it easier to implement new routes in the future, such as pages for user profiles, settings, data exploration, etc.

3.2 Authentication

Our authentication system can be split into three parts: user registration, user login and token verification. In the following subsections we go through each of those and discuss their purpose, functionality and safety.

3.2.1 User Registration

The first step of user registration happens on the front-end of our application, where the user is asked to fill out a user registration form consisting of inputting their name, e-mail, username and password. We have implemented several validation checks on our front-end using regular expressions [12]. This is done to check that the email is in the correct format, including "@" and ".". For the password we make sure that the user is creating a "strong" enough password including at least eight characters, one uppercase letter, one lowercase letter, one special character and one number. If the information submitted passes all validation checks, then a POST request is made to our API Gateway endpoint using Axios. The header of the request consists of our API key and the body of the information submitted by the user. By using a unique API key, we make sure that no outside

26

requests to our API endpoint can be made. The path of the API endpoint specifically points to our Lambda function for registration. A Lambda function is a function which is executed in response to events, in our case they are being triggered by HTTP requests via our APIs. In the case of a successful request our Lambda function is triggered, first validating that all fields has been filled out, then validating the username and email to make sure they are not already in use. If they are in use an error message is returned with an appropriate message. If the checked information is not already present in our DynamoDB table, then the function proceeds to add salt to the password, then hash it and our saveUser function is called to store it in our DynamoDB table along with the rest of the user information. Salting a password means appending a random and unique string of characters to the user's password before finally hashing it, in our case the salt consists of a randomly generated string of 10 characters. By adding salt before hashing the users password, we add an additional layer of security to avoid attacks such as Rainbow attacks, where attackers pre-compute the hash values of common passwords [13].
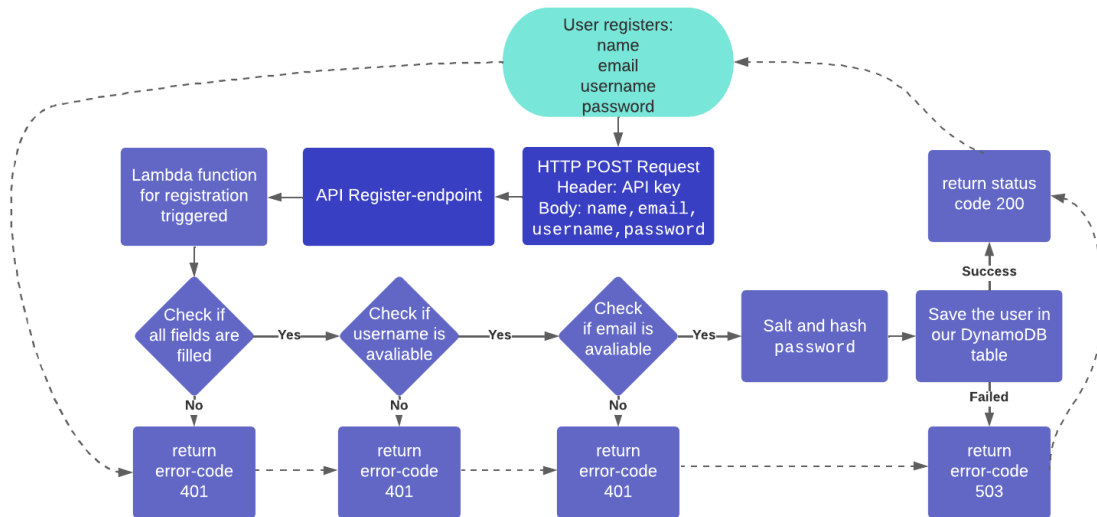
Figure 23: Registration flow showing the process of our Lambda function

3.2.2 User Login

For handling user logins, we take the same approach as with user registration. First the user is asked to fill in their username and password, upon submitting the login form we first check if all fields has been filled, if not then a error message is returned accordingly. If all fields are filled in correctly then a POST request is made using Axios. This time the request is aimed at our API endpoint for logging in, triggering the Lambda function accordingly. The Lambda function then retrieves the user data from our DynamoDB table based on the username input in the login form. Since hashing is a one-way function, when authenticating a user on login, the password input is

27

hashed including our original salt and then compared to the hash stored in our DynamoDB table[8]. If the password matches what is stored in our DynamoDB table with the username, then a JSON Web Token (JWT) is generated for the user. The JWT consists of a payload which includes the user's username and name, a secret key for signing the token and a expiry time of one hour.[14] The JWT is then returned to the client in the HTTP response, which is then saved in the session storage of the browser.
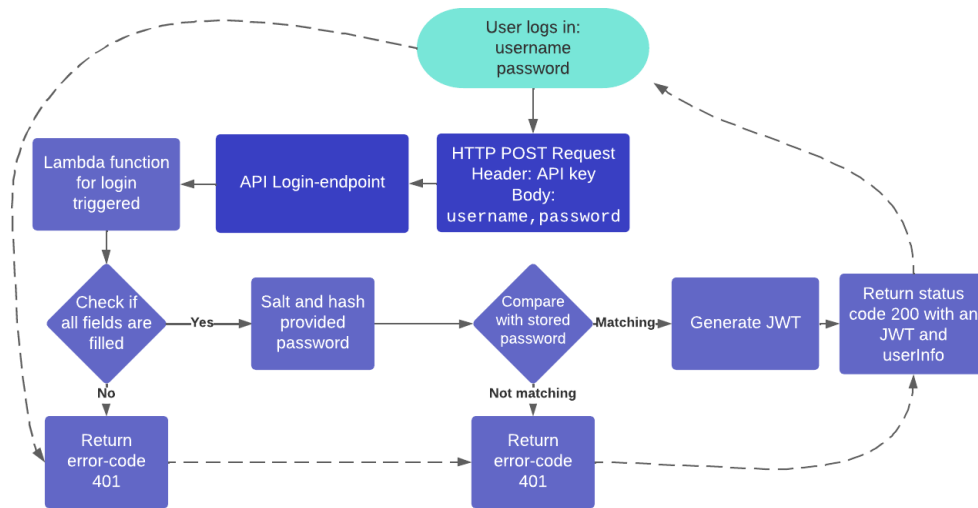
Figure 24: Login flow showing the process of our Lambda function

3.2.3 Token Authentication

When a user is successfully logged in our Lambda function will generate and return a JWT which is then stored in the browsers session storage. This is done to safely maintain user sessions across the mapping client. When the Mapping Client is initially opened by the user, we check if they have a valid JWT by making a POST request to our API endpoint for user verification. The POST request then triggers the specific Lambda function which verifies the information included in the POST request. We have made two functions for handling tokens in our Lambda functions: generateToken and verifyToken. As the names implies they are for generating tokens when users login and for verifying tokens on subsequent requests, respectively.

---

[8]DynamoDB functions as our user table for storing user information

28

function generateToken(userInfo)

This function is used when we need to generate a new JWT for a user. The JWT is based on the user's information by encapsulating it in the JWT. It takes userInfo as an argument, which consists of the user's name, email, username and password. If this is not provided then the function will return null and a JWT will not be generated. If the userInfo is provided, then we take use of jwt.sign, which is a part of the JWT library, to generate the token. jwt.sign lets us sign the token with the necessary arguments. This method takes three arguments: the payload (userInfo), a secret key and the expiration time of the JWT, which are then returned with the function. We have chosen an expiration time of 1 hour, which is simply based on finding a balance between user convenience and user safety.

function verifyToken(username, token)

This function is used for verifying the JWT of the user and making sure the JWT matches the provided username. The function takes two arguments, the username and token. We then use jwt.verify to verify the token, this function takes three arguments: the token that needs verification, the secret key and a callback function to handle the result of the verification. We then have different errors set up depending on the result of the verification, which all result in "verified" being set to false. If the token is valid, then "verified" is set to true with an appropriate message.

In conclusion, our authentication system is token based. By using JWT, we ensure that the user is validated continuously throughout their interaction with the mapping client. Another approach would've been to take use of Session Authentication, which stores the authentication details server-side instead of on the user-side [15]. All our APIs are secured using a unique API-key, in order to prevent any unauthorised outside requests from triggering our Lambda functions. We make sure to salt and hash the users password using bcryptjs, which makes up good protection for several types of attacks, such as rainbow and brute-force attacks [16].

3.3 File Fetching

One of our main requirements was to securely display the data provided by SeaVis and processed by the KML creator. These KML files needed to be securely stored and then served to the mapping client, also in a secure manner. We have accomplished this by taking the same approach as with our authentication system, by taking advantage of Lambda functions securely triggered by an API. In the following section we will explore how we safely manage the processed data and serve it to our mapping client.

3.3.1   Presigned URLs

Upon the user successfully being logged in, they will be redirected to the /client path of our application. When initialising the Leaflet map, we also have to fetch the KML files from the private S3 bucket in which they are stored. This is done by making a GET request to our API endpoint, which routes this request to a corresponding Lambda function that is then triggered. This function lists all the objects in our S3 bucket and generates presigned URLs for each object. A presigned URL provide temporary, secure access to the private objects in our S3 bucket, specifically our KML files in the format of a URL. We have set the presigned URLs to expire after 10 seconds upon being generated, leaving enough time to safely fetch them. If the object listing fails for any reason, we have set up error handling which will return a 500 error response, which indicates an execution error [17]. This allows us to securely serve these files without exposing them to unauthorised access.

4 Tools and Third-Party Packages

This section introduces the third-party tools and packages used throughout the working process, outlining their role in the project.

4.1 KML Creator

As the KML Creator involves spatial interpolation and manipulation of geospatial data, a number of libraries were used to work with such data and generate the KML files.

Below are the main libraries the KML Creator utilises:

- GeoPandas GeoDataFrame: A specialised data structure for handling geospatial data, facilitating manipulation and analysis.
- GDAL: A library for reading, writing, and manipulating geospatial data formats, used for interpolation and analysis.
- Shapely: Provides geometric objects for spatial operations within the KML Creator.
- GeoPy: Calculates accurate distances between points on the Earth's surface.
- Simplekml: Generates KML files from processed geospatial data.
- PyInstaller: Packages the KML Creator and its dependencies into a standalone executable for easy distribution

These are the main libraries that enables the KML Creator to efficiently process geospatial data, perform spatial operations, and create KML files for visualisation purposes.

4.1.1 GeoPandas GeoDataFrame

The fundamental data structure used in the KML Creator is the GeoDataFrame, which is a specialised type of Pandas Dataframe. Pandas is a Python library that is used for data manipulation and analysis [18]. With Pandas, data can be imported from many formats (CSV, JSON, etc.), and various data manipulation operations can be carried out: merging, filtering, cleaning, performing arithmetic operations, etc. Pandas is able to efficiently manipulate large data-sets as it uses vectorisation (as oppose to iteration) for many operations. This is important in the KML Creator, as SeaVis' data-sets could potentially be millions of rows in length. The primary data structure used in Pandas is the Dataframe, which is a two-dimensional, labelled and table-like structure.

While Pandas is suitable for general data manipulation, there are many other libraries that are more suitable for working with geospatial data. There are also more suitable data structures than the Dataframe. The GeoDataFrame is essentially a Pandas dataframe with an extra column of geospatial data. This extra "Geometry" column contains Point objects from the Shapely library [19]. A Point object takes two arguments: x (longitude) and y (latitude). GeoDataFrame has a method for saving

to a file, in a chosen format. Here we employed a popular format: the ESRI Shapefile [20]. The Shapefile actually consists of four files: the geometric data (.shp), an index file (.shx), an attribute table (.dbf) and a projection file (.prj). These are commonly found in a single zipped file with a ".shp.zip" extension. Most importantly, Shapefiles can be used as input and output for the main computational engine of the KML Creator: GDAL.

4.1.2 GDAL

The Geospatial Data Abstraction Library (GDAL) [21] is an open-source library that provides a set of tools and functionalities for reading, writing, and manipulating raster and vector geospatial data formats. The KML Creator uses GDAL twice:

- to perform interpolation and transform vector data (disparate points contained in a Geo-Dataframe) to raster data (GeoTIFF file)
- to convert the GeoTIFF file back into vector data, through the process of polygonisation (see Section 2.2)

4.1.3 Other geospatial libraries (Shapely and Geopy)

While GDAL provided the necessary algorithms to convert between raster and vector files, there were are a number of other geospatial libraries that fulfil minor roles:

In the KML Creator, the Shapely library is widely used for type annotations: for the Point objects used in the GeoDataframe and the Polygon and MultiPolygon objects used in the GeoDataframe imported from the polygonised Shapefile. Shapely performs a more functional role in the final creation of the KML file: the unary_union function unites polygons which overlap.

GeoPy is used to calculate the distance from one point (latitude and longitude) to another. This is a non-trivial calculation to make accurately as the globe is not perfectly spherical. We have used the GeoPy.distance module to calculate distance using the World Geodetic System (WGS-84), which is said to be the most accurate model for any location in on the globe [22]. GeoPy uses its own Point objects, which are distinct from the Shapely Point objects used in GeoDataframe.

4.1.4 Simplekml

Keyhole Markup Language (KML) is an XML-based file format used for representing and visualising geographic data, such as points, lines and polygons [23]. We use the Simplekml library to create KML files from the GeoDataframe containing polygons and MultiPolygons. A sample KML file can be found in Section 8.4.

4.1.5 PyInstaller

PyInstaller is a tool used to convert the KML Creator and its dependencies into a standalone executable. By tying up the script and dependencies, PyInstaller simplifies the distribution process, removing the need for a separate Python installation and compiler. With this, the user can easily run the KML Creator on different machines without worrying about Python or dependency management [24].

PyInstaller automatically detects and includes the necessary components into the executable or directory, making it a self-contained environment for end-users. Within PyInstaller, customisation options are available, which allowed us to fine-tune the packaged executable for the KML Creator. With the extra options, additional files and or directories can be added, modules can be excluded, and runtime options can be set for optimal performance.

4.1.6 Other tools

There are a number of tools we used throughout the process, which are not intended to be included in the final product. Chief amongst these is Pre-commit, which enables the setup of pre-commit hooks, which are automated checks that run before committing code to GitHub. We were able to ensure a certain level of code quality and adherence to coding standards by using the following pre-commit hooks: trailing-whitespace, requirements-txt-fixer, trailing-whitespace, end-of-file-fixer and jupyter-black. In addition to these tools, we also made use of Matplotlib [25] to visualise data during development.

4.2 Mapping Client

In the following section, we discuss the various tools and packages used in the development of the mapping client. These include:

- React: A component-based architecture for creating complex UIs.
- Leaflet: An open-source JavaScript library designed for building web-based mapping applications, providing interactive and responsive maps.
- Leaflet KML: Parses and displays KML files on a Leaflet map.
- Axios: an HTTP client used in the front-end for accessing APIs triggering Lambda functions, and facilitating tasks like user authentication and secure file retrieval.
- AWS Lambda: A server-less computing service that handles user authentication in the Mapping Client and serves KML files from a private S3 bucket.
- API Gateway: Facilitates secure information between the front-end and back-end through RESTful APIs.

- DynamoDB: a NoSQL database used for secure storage and verification of user information during registration and login in the Mapping Client.
- S3: A storage service used to securely store and retrieve KML files in the mapping client.
- bcryptjs: JavaScript library used in the Mapping Client, to securely hash and compare user passwords during registration and login.
- JSON Web Token: Used for secure user authentication in the Mapping Client.
- React Router: A package used to handle routing of the Mapping Client

These are the main tools used in the mapping client, that enables it to efficiently visualise and securely store KML files, while ensuring a secure login process.

4.2.1 React

One of the key benefits of using React is its component-based architecture. React components are modular, reusable building blocks that allow you to create complex user interfaces with ease. Each component encapsulates its own state and logic, making it easy to manage and update the application's UI.

Additionally, React has a large and active community, which means there are many resources available to help you learn and troubleshoot any issues you may encounter while building your application. There are also many third-party libraries and tools available that can enhance your React application, such as Redux for state management and React Router for managing application routing.

4.2.2 Leaflet

Leaflet is a widely-used open-source JavaScript library that is specifically designed for building web-based mapping applications. It provides a comprehensive range of tools and features that allow developers to create interactive and responsive maps with ease. With Leaflet, we will be able to create an engaging and user-friendly interactive map to visualise the mussel data, displaying the mussel density at different locations in the sea. The map will be an essential component of the mapping client, helping users to make informed decisions based on the latest data [26].

4.2.3 Leaflet KML

Leaflet KML is a package that lets you parse and display KML files on a Leaflet map, since KML isn't a supported format by Leaflet [27]. This is done by parsing the KML data to corresponding Leaflet layers. We have used it for parsing our KML files from our KML Creator and display them on a Leaflet map in our mapping client.

4.2.4 Axios

Axios is a promise-based HTTP Client for JavaScript, allowing us to make HTTP requests from our front-end [28]. We have used Axios for making POST and GET requests to our APIs depending on the need. Axios serves as a crucial part of accessing our APIs and triggering our Lambda functions when either authenticating a user or fetching files securely from our S3 bucket.

4.2.5 AWS Lambda

AWS Lambda is a server-less computing service provided by Amazon Web Services (AWS). It allows its users to execute code on their back-end without having to create and manage their own servers, hence it's referred to as server-less [29]. The Lambda functions set up can be triggered by HTTP requests coming from the front-end of your application and the user only pays for the requests made, making it cost-effective.

AWS Lambda serves as a key component to handle user authentication of the mapping client. We have developed three primary functions, which handle user registration, user login, and token verification (see Section 3.2). These functions are triggered based on which of our API endpoints that receives a HTTP request from our front-end. We have also used AWS Lambda for serving KML files stored in a private S3 bucket to our mapping client. For this we have developed a Lambda function, which generates presigned URLs that gives the mapping client temporary access to the files in our bucket.

4.2.6 API Gateway

Amazon API Gateway is a service for creating, deploying, and maintaining APIs. Being a part of AWS, it's fully compatible with many of the other services provided by Amazon, such as AWS Lambda, DynamoDB and S3 [30]. We have used API Gateway for creating RESTful[9] APIs, which are HTTP based allowing the use of HTTP requests, such as GET, POST, PATCH and DELETE. This provides a safe way of exchanging information between our front- and back-end.

Specifically, we have implemented three POST methods corresponding to the Lambda functions previously described: user registration, user login and token verification. Each API endpoint is secured with a unique API key, which ensures that only valid requests from our front-end can interact with our back-end services through the API. We have also implemented a GET method for fetching the KML files directly from our private S3 bucket.

---

[9]A RESTful API is a framework for an application program interface (API) that utilises HTTP requests to access and manipulate data. [31]

4.2.7 DynamoDB

Amazon DynamoDB is a NoSQL database, making it flexible as its designed to not rely on predefined tables and fixed column definitions [32]. We are using DynamoDB to safely store and check user information when registering and logging in. Upon registering we store the username, name, e-mail, and password input by the user. When a user logs into the mapping client, our Lambda function fetches the user data from DynamoDB using the provided username. The hashed password stored in DynamoDB is then compared with the provided password, which is also hashed, and if they match, the user is authenticated.

4.2.8 S3

Amazon S3 is a storage service provided by Amazon, which allows its users to safely store objects remotely [33]. We have used Amazon S3 to store and retrieve our KML files that are to be displayed on the Leaflet map when the user is successfully signed into the mapping client. The KML files are stored as objects within an S3 bucket each object is associated with a key, which is unique.

As previously explained, we securely fetch the KML files by triggering a Lambda function through a HTTP request that is made once a user is successfully logged in. Our Lambda function lists all the objects in our S3 bucket, generates a presigned URL for each file, and then returns these URLs in the response body. By retrieving a list of all our stored objects, it allows us to easily update our S3 bucket with new KML files once new data has been processed.

4.2.9 bcryptjs

bcryptjs is a JavaScript implementation of the bcrypt password-hashing function [34]. In our mapping client, bcryptjs is utilised for hashing user passwords during registration before storing it in our DynamoDB table. However, what is special about bcryptjs is that it adds salt to the password before hashing it.

4.2.10 JSON Web Token

JSON Web Token (JWT) is an open standard, allowing for securely transmitting information between parties [35], such as an applications front-end and its servers back-end. In our mapping client, we are using JWT to authenticate users when they are successfully logged into the application. This is done by returning a JWT in the the response, when a user is logged in. The token is then stored in the session storage and checked every time the user makes a request to the server that needs to be authenticated, such as accessing the mapping client itself.

4.2.11 React Router

React Router is a library for React, which provides tools for handling the navigation of an application [36]. In the Mapping Client, we have used React Router to define the different paths which leads to the different components of the application. This allows us to define which components are rendered depending on the route.

4.2.12 Other tools

During the development of the Mapping Client we also took use of a few tools, which are not intended to be used in our final product. This mainly concerns the use of Mocha, Chai and Axios Mock Adapter.

- Mocha is is a test framework based on JavaScript and running on Node.js [37].
- Chai is an assertion library for JavaScript, which provides a way of defining tests [38].
- Axios Mock Adapter allows for making mock HTTP requests in the style of Axios [39].

We have used the above libraries for performing assertion tests of our front-end password validation and testing for the return of different status codes from our APIs depending on the request made.

4.3 TimeZero

TimeZero is a software company that was established to cater to the unique needs of the recreational sailing, cruising, and regatta industries. The company has developed a comprehensive platform that provides a range of solutions to the marine industry. With over 25,000 installations worldwide TimeZero is also offered in many languages such as Danish.

One of the standout features of TimeZero is their mapping client. This client is fully interactive and provides users with a range of tools and functionalities that allow them to easily navigate waterways. TimeZero has developed a flexible approach that allows third-party companies to create their own data sets that are compatible with the mapping client. This approach enables these companies to offer tailored services to their users, providing a more personalised experience that meets their specific needs. One such third-party company is SeaVis, which has integrated mussel data into the mapping client to offer an more enhanced experience to their users [40].

5 Testing

When testing the clients individually, focus was placed upon how a user might make a mistake or how the client might display the data wrongfully. So the testing process was executed from the perspective of a potential user of the clients. By making sure that the correct error messages was displayed when an error occurred, and decrease the chances for a user to make mistakes such as entering the wrong password when registering an account.

5.1 TimeZero Mapping Client

When adapting TimeZero into our product solution, we needed to conduct test it in two different parts of our project. Firstly regarding the software's capabilities and compatibility, in order to create a product that would work symbiotically with it. Secondly when we finished developing our programme for TimeZero. Here we needed to ensure, that TimeZero would portray our KML file correctly with all perimeters, such as smoothing, and layer colour.

5.1.1 Testing TimeZero

When testing the capability of TimeZero, we first formatted the SeaVis data to an XYZ file, to test if this file type was compatible with our data and TimeZero. When uploaded, the data was not projected and we therefore had find another file type to visualise the data. After trail and error, we attempted with a KML file, which portrait the the data correctly, This let us to the creation of our KML creator. When testing our KML creator in TimeZero, all parameters were displayed correctly, in exception for the bin colour. This mistake was later corrected in our KML creator.

5.2 KML Creator

The testing of our KML creator plays a big role in reassuring the reliability and correctness of our KML creator. We do that by verifying the functionally of the application through testing. This also helps us identify and fix any issues or bugs.

5.2.1 Black-box testing KML creator

When testing the KML creators functionality, we developed sets of tests, that covers our bins, preper and runner classes, as well as our generating of KML files. the initialisation tests on our bin, preper and runner has been implemented in each classes. These tests can be run with the following command : "python -m unittest tests.test_backend" and ensure that the classes can be instantiated correctly and that their properties are set as intended. The functionality of the application is thoroughly tested through a series of tests that simulate the entire process from CSV to KML generation.

these test includes Creating the shapefile for each bin, Running interpolation for each bin, running polygonisation for each bin and crating KML for each bin.

5.2.2 Parameter Testing

In this section we test the different parameters of the KML Creator. Testing the different parameters is a crucial aspect of optimising our application, and making sure the parameters work as intended. We will discuss the assessment of several key parameters, namely the height and width of the search ellipse, simplification, pixel size and angle. We have aimed to understand how these parameters affect the performance of the application and the outcome of the KML file that is created. To test the different parameters we made several KML files of the same CSV file, and then proceeded to compare different values of the parameters. Below is the KML file with the default settings of the KML Creator.
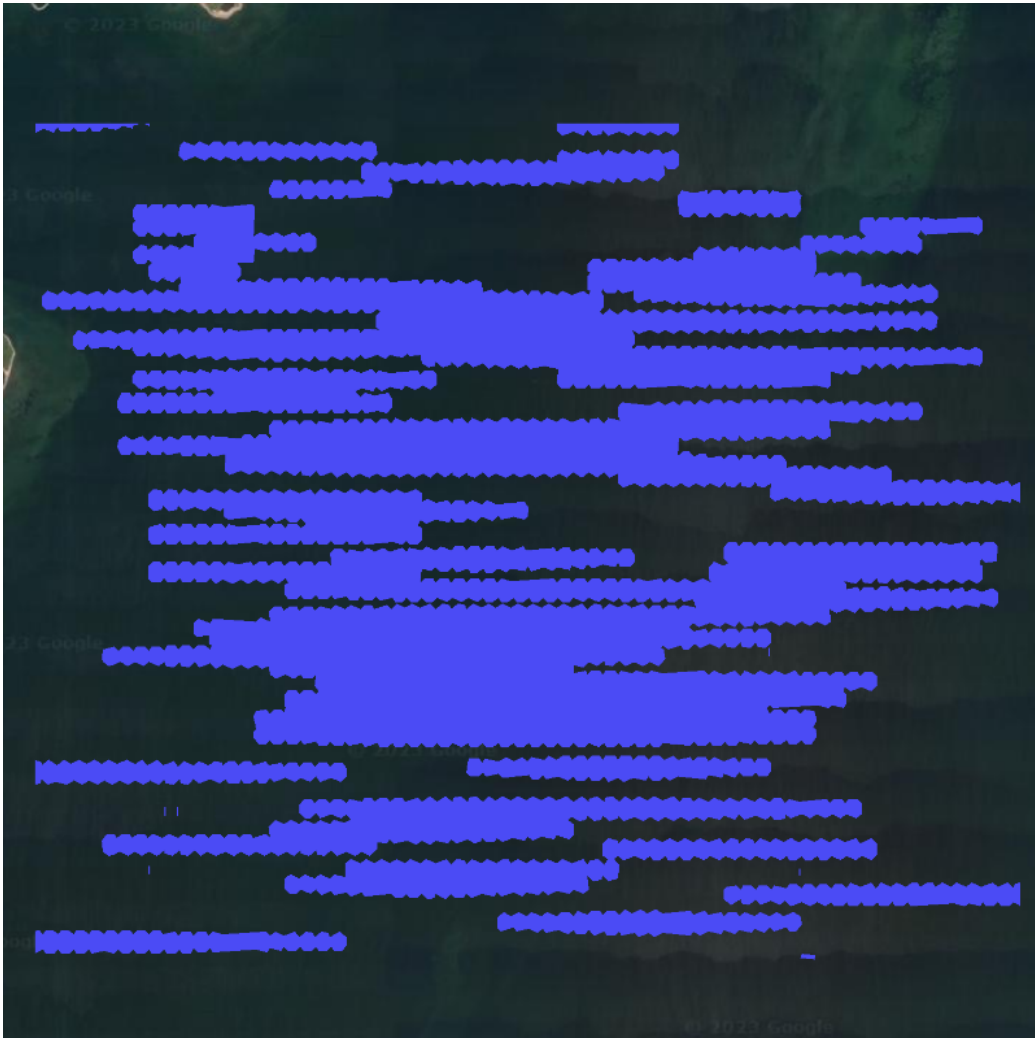
Figure 25: Output KML with default settings of KML Creator:
Pixel Size = 10, Radius Width & Height = 60, Smoothing = 10, Angle = 0

The first parameter that was tested, was the height and the width of the search ellipse. As seen in the figure below, there is a drastic change in how the KML looks. As anticipated, the search ellipse operates in the manner described in Section 2.1, thus validating the expected outcome.
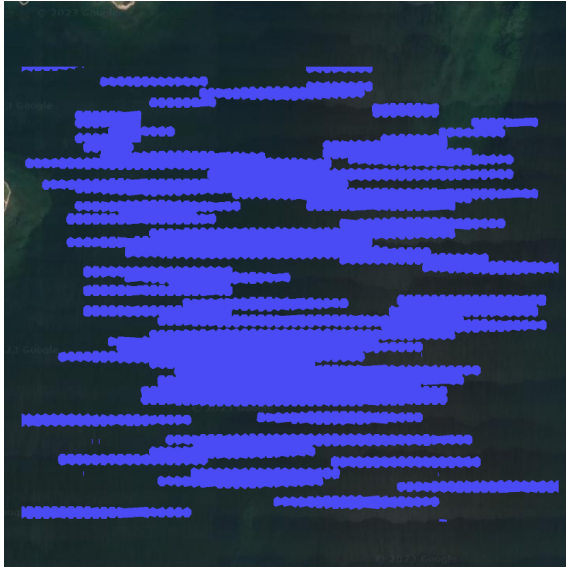
Figure 26: Search Ellipse Height and Width = 50 metres
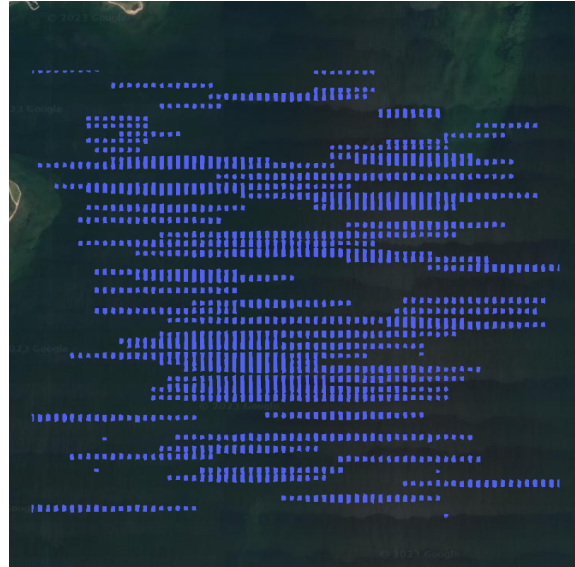
Figure 27: Search Ellipse Height and Width = 30 metres

Next, we examined the simplification parameter. The figure below illustrates the noticeable impact of this parameter on the edges of the data points. As simplification decreases, the edges become rougher and more pixelated, resulting in a less streamlined appearance. It is also important to note that this decrease in simplicity can potentially affect performance, as larger amounts of data are required, subsequently leading to larger KML file sizes.
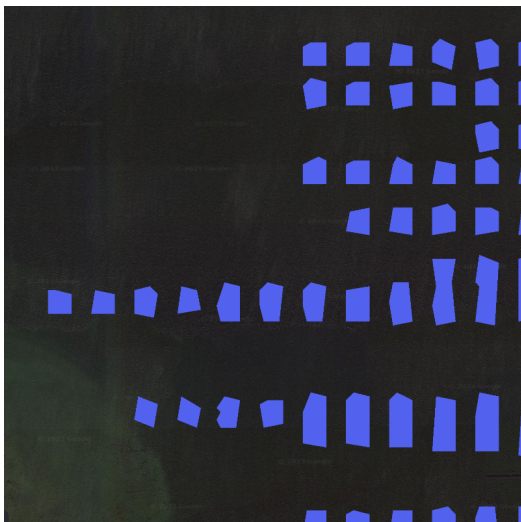
Figure 28: Simplification = 10
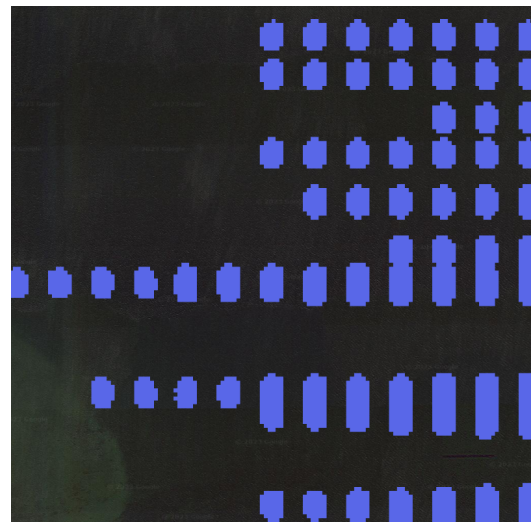
Figure 29: Simplification = 0

In the next test, we focused on adjusting the pixel size, ranging from 10 to 50 metres. This adjustment had profound impact on the resulting KML file. As seen on the figures below, the data points have transformed into larger pixels, as all the interpolated data points have clustered into several big data points. This change in appearance brought a notable performance enhancement, specifically in terms of the KML file size reduction, as fewer coordinates are required.
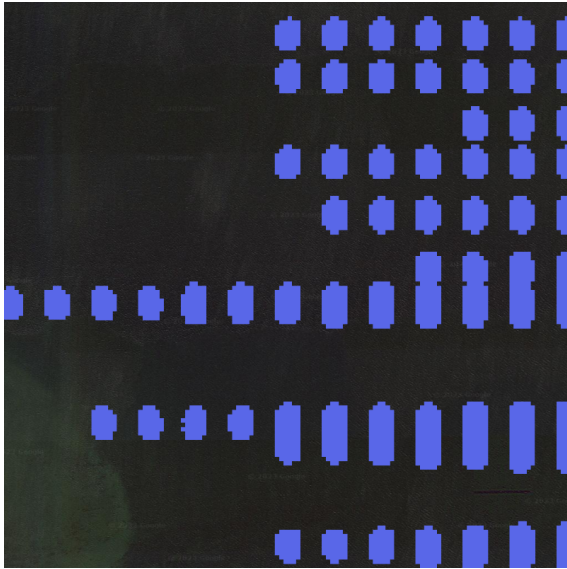
Figure 30: Pixel size = 10 metres

Figure 31: Pixel size = 50 metres

In the final test, we examined the impact of adjusting the angle of the search ellipse from 0 to 45 degrees. Some changes were observed in the resulting KML file, which is surprising considering that in this case the search ellipse is a circle, which ought to be identical for any angle of rotation. One plausible explanation for the differing output has to do with the curvature of the earth. In the Runner's run_interpolation_for_each_bin method the search ellipse width and height is converted to degrees, using a non-spherical model of the earth. When this search ellipse is rotated, it may not take into consideration the not-perfectly-spherical model of the earth, leading to different algorithmic output.

Figure 32: Angle = 0 degrees   Figure 33: Angle = 45 degrees

5.3 Standalone Mapping Client

The mapping client provides the user with multiple functions and services, such as a login function that guarantees SeaVis that the intellectual properties of the client is protected, an interactive client, where the user can zoom in and out, as well as move around on the map.

5.3.1 Black-box Testing Mapping Client

When testing the standalone mapping client, we focused on the security aspect of the client. The purpose of the security is to protect the intellectual properties of the client. To confirm that the client is secure, we've had to detect and prevent ways of bypassing the login function, and thereby getting direct access to the client without a valid token. such as bypassing the registration by changing the URL to the URL of the client. This was prevented by redirecting the user to the registration if an attempt to access the client without a valid token has occurred. another feature regarding our security system, is the requirement that the user repeat their password twice To make sure that the user has entered the correct information and reduce the probability of text error, we decided that the user had to enter the password twice. An additional login feature was the error messages when a user attempts to use an email that dose not contain valid email requirements, the way we set up the requirements was that we separated the email into three part. The email must contain letters and numbers prior and after the '@' the second parameter set upon the email, is that there must be between 2-4 characters after the '.'. Further parameters was set for the password, such as number inclusion, uppercase letter and as mentioned before, a correct repeat of password.
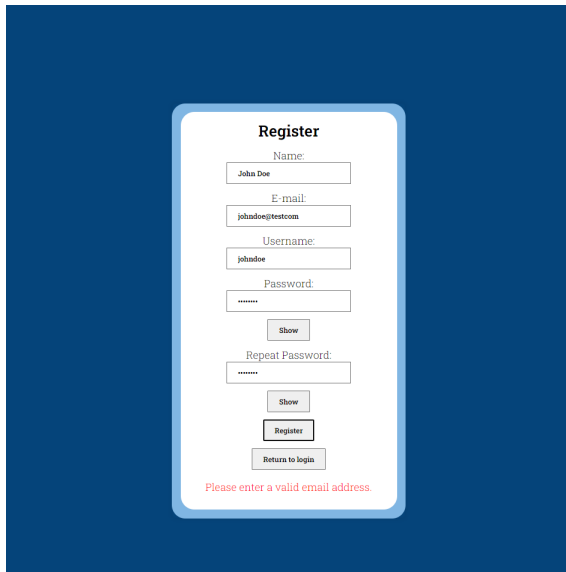
Figure 34: Invalid email
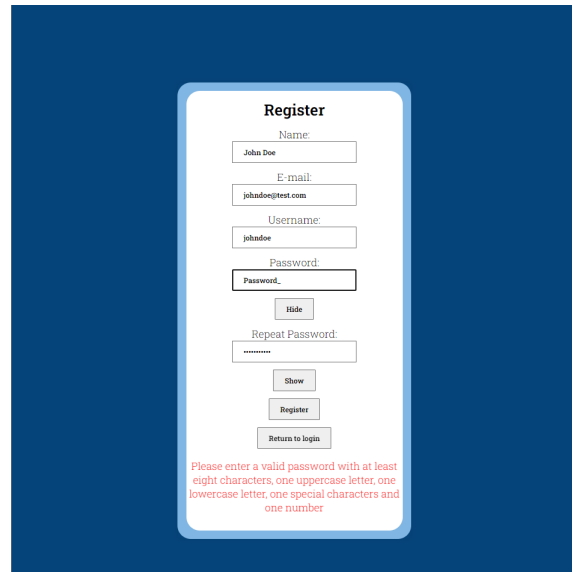
Figure 35: Invalid password

after integrating these functions in our client, we estimate that the chance for creating faulty password is minimised drastically.

5.3.2 Unit Tests

We have written several unit tests for our "ValidationService" module, testing all the functions used. This is done using Mocha and Chai as explained in Section 4.2.12, more specifically we take use of Chai's "expect" function, which lets us make assertions about the expected result of each test. We have specifically written tests for validation of name, email, username, password and repeat password, testing for both valid and invalid input. This is done to ensure that the logic of our validation tests work as intended. The results of our tests can be seen on Figure 36.

5.3.3 Integration Tests

We have written a few integration tests, to check if our API handles user input as expected. The tests consist of two inputs, "validUser" and "invalidUser" corresponding to a valid user input and a invalid user input. We take use of Chai's "expect" function, which again lets us make assertions about the returned response. The tests consists of POST requests to the API endpoints of registration and login, containing the valid and invalid user input. We then assert the returned response based on the input. The results of our integration tests can be seen in Figure 36, along the results of our unit tests.

Figure 36: Results of our unit and integration tests

5.4 Product Testing

Throughout the course of developing both clients, white-box and black-box testing were performed to ensure that the final product is as refined as possible.

5.5 Challenges of Using Fabricated Data

The following section discusses the challenges encountered while working with fabricated data and the measures taken to address these challenges in the development of the data interpolation client. The project involved a request for data from SeaVis, but since they could not provide real-life data, we had to work with a fabricated data-set consisting of roughly 500,000 data points that were designed to simulate real data. The use of fabricated data presented a significant challenge, as we had no prior knowledge of how the actual data should look, making it impossible to confirm whether the interpolation methods and parameters applied to the fabricated data would work with actual data.

We then interpolated the data using various methods and parameters (see Section 2.1), producing the final product shown in Figure 25, which depicts the transformed and interpolated data. However, the resulting image appeared unrealistic as it seems too orderly and as if the regions with mussels were simply drawn by the human hand, on top of the map. As we are not mussel experts, we cannot confirm whether this is how it would actually look in real life.

To further explore the limits of the interpolation method, we generated synthetic data-sets solely for testing purposes. The synthetic data-sets were not intended to replicate any real-life systems, but instead aimed to evaluate the accuracy of the interpolation algorithm under different conditions.

To address the challenges of working with fabricated data, we added noise to the data and evaluated the output of the interpolation algorithm. We also assessed the algorithm's performance using different types of synthetic data to confirm its effectiveness under different conditions. These measures enabled us to overcome the challenges posed by working with fabricated data and to develop an effective interpolation client. When presented with the visual result of the fabricated data, Bjørn responded with:

"I think a lot of (...) the issues that I have with the result is because of the [fabricated] data set. More than it's because of your programme "[1].

In addition to the data visualisation, we also presented the data processing client, used to convert data into layers of .KML files. The client contains a variety of parameters to define the data layer, such as data category, size, colour, etc. Based on the feedback from the presentation, there was a suggestion to add the lower and upper bound in from of percentages within the data-set inputs. This would provide a clearer understanding of the range of values associated with the data-sets.

"So if you look at the software you have (...) especially with the [mussel] density, that I know this is going to be, I would like this to be some kind of percentage" [1]

6 Project Development and Discussion

In this section, we will outline the difficulties faced and decisions made during the development of the project.

6.1 Interpolation Algorithm

When selecting an interpolation algorithm for our project, we evaluated several algorithms provided by GDAL. Within their library and the gdal.grid function, we could choose from Inverse Distance, Moving Average, Nearest Neighbour, and Linear interpolation algorithm [41]. After careful consideration, we decided to use the Moving Average algorithm (see Section 2.1), due to its suitability and performance for our data-set.

The Moving Average algorithm yielded satisfactory results with minimal unobserved data points during the interpolation process (see Figure 37). While alternative algorithms like Inverse Distance or Nearest Neighbour appeared more visually appealing (see Figure 38), they did not provide the desired accuracy when converting the resulting GeoTIFF files into KML files using the KML Creator.
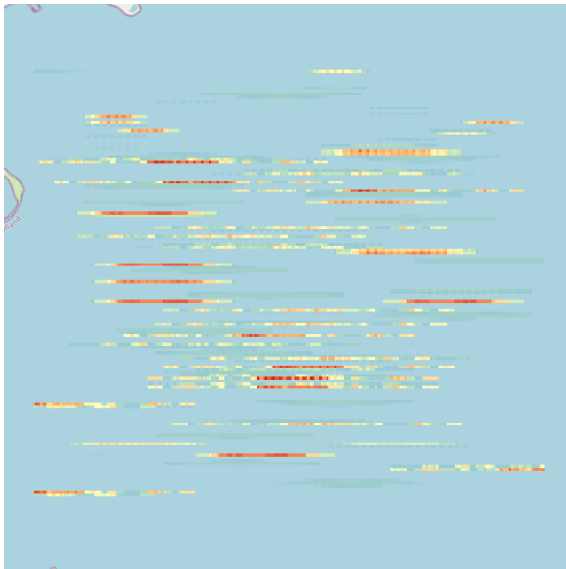
Figure 37: Moving Average interpolation algorithm
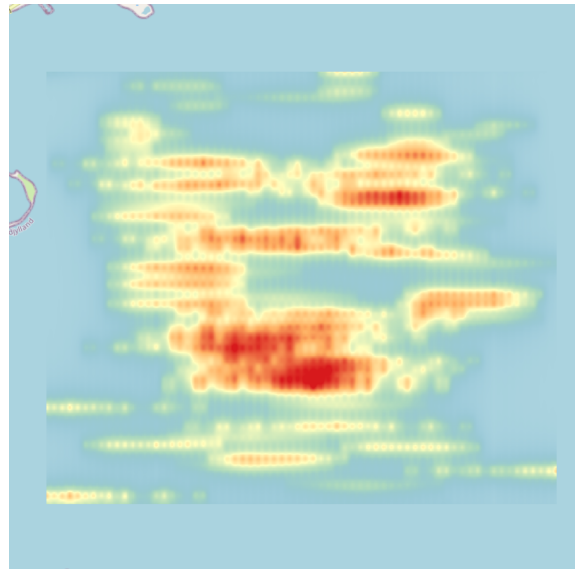
Figure 38: Inverse Distance interpolation algorithm

By choosing the Moving Average algorithm, we achieved a balance between accuracy and visual representation. The algorithm effectively estimated values between known data points while also maintaining the integrity of the original data points. Taking this approach ensured reliable interpolated results that aligned well with the characteristics of our geospatial data-set.

6.1.1   GUI vs Command-line

When designing our application, we faced the decision of whether to create a command-line interface or a graphical user interface (GUI). After careful consideration, we decided to create a GUI for our application for several reasons.

- A GUI is more intuitive and user-friendly compared to command-line interfaces.
- A GUI allows users to interact with the application visually, eliminating the need to remember and type specific commands.
- a GUI provides a more immersive experience for users, as they can see real-time reflections of their actions on the screen.
- a GUI enhances users' sense of control and satisfaction when using the application.
- Creating a GUI helps showcase the application's functionality in a visually appealing manner.
- a GUI Makes it easier to highlight important features and make easier for users to navigate and our application

Using a GUI also has its disadvantages. Firstly, generally GUIs are very resource-intensive, requiring more memory and processing power, making it challenging to work on older systems. Additionally, GUIs often have limited flexibility. A GUI provides a predefined set of options and interactions, which therefore limits the flexibility and customisation available to users.

6.2   Electron

In the early stages of developing the Mapping Client we wanted it to be a desktop application. For that we used Electron, which is a runtime framework that allows developers to create cross-platform desktop applications using web technologies such as HTML, CSS, and JavaScript [42]. Electron resembles the architecture of a modern website, however, it's contents will be rendered as a desktop application. Later in development we decided to design the Mapping Client solely as a web application and switched to the use of React. The old version of the Mapping Client before we started using React can be seen on Figure 39. The decision was based on making the application more efficient when it comes to deploying updates, such as adding new data to display, and improving availability. We also made sure that our web application could be rendered using Electron, meaning the option for making it a desktop application would always be available even when sticking to React. This is done by creating a wrapper for our React application to be rendered using Electron [43].
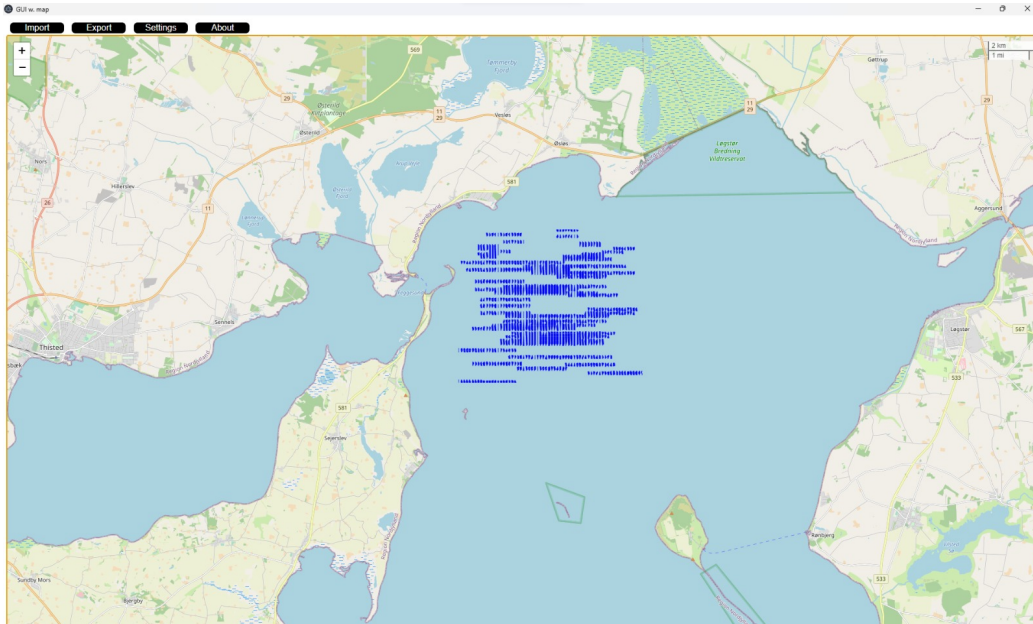
Figure 39: Old version of the Mapping Client

The above illustration was one of the earliest versions of the Mapping Client, which instead of fetching the files externally, they were imported locally by the user.

6.3   Difficulties of Using PyInstaller

When we set out to distribute our Python programme, we wanted to make it as easy as possible for end users to install and run it without having to worry about installing Python or any dependencies. After some research, we decided to use PyInstaller, which a Python library that can package Python applications as a standalone executable. Installing PyInstaller was straightforward using pip, however, we ran into some difficulties during the process of creating the executable.

As newcomers to PyInstaller, we found it challenging to determine which commands to use and when. We eventually decided to try auto-py-to-exe, a package that integrates PyInstaller with a user-friendly GUI interface. Although this made the process easier, we still encountered issues, such as ensuring all the necessary files were in the correct location. Fortunately, PyInstaller provided helpful error messages that made it easier to troubleshoot and fix these issues. One of the biggest challenges we faced was packaging the GDAL module, which was difficult to install and even harder to make into an executable. We spent a considerable amount of time researching and testing different approaches, and ultimately were able to include GDAL in the final executable (see Section 8.2). Overall, PyInstaller was a helpful tool that simplified the installation process for end users by eliminating the need to install Python or any dependencies. We tested the executable on different operating systems, and it worked seamlessly in each case [24].

6.4 Third-party Tools and Libraries

Since both the KML Creator and the mapping client are relatively complex applications a significant amount of third-party tools and libraries is utilised due to various reasons. First of all, it was done to safe time and effort instead of building certain components from scratch, as well as to ass new features and capabilities to the code. Another reason was to improve the overall quality of the code and reduce the risk of introducing errors as the libraries and tools we used are reliable and have been thoroughly tested. Relying on the 3rd party tools and libraries also allows us to fill in the expertise gap, since it would be unrealistic to obtain a stable and operational final product. However, there is a potential risk of over-reliance that can make it difficult to fully understand the code. Therefore, it may be harder to potentially troubleshoot problems or make changes to the code. Third-party tools and libraries may not always be compatible with the specific environment or technology. This can lead to conflicts or errors that can be difficult to resolve [44]. Another issue is installation and integration of 3rd party tools and libraries that can be quite challenging. A particular examples within our project include GDAL and PyInstaller. Overall, due to the complexity of this project, the used amount of third-party tools and libraries is justified since those provide functionalities that are important, but are not essential for the implementation of the general concept both for KML Creator and mapping client applications.

6.5 Dual Applications: Data Formatting and Mapping Client

One of the most significant decisions made while developing this project was an idea to split our effort into 2 general directions: developing a way of formatting the data from SeaVis into a suitable format and creating a standalone mapping client that could display the formatted data. There were certain reasons for such a decision: as the project work started, the information we were getting from SeaVis was limited and there was no guarantee that we would get access to TimeZero in order to test the outcome and quality of the KML Creator. The other reason was our aspiration to expand the project beyond the scope of the SeaVis's requirements, while learning about and implementing certain aspects such as functioning GUI and the elements of data security. It also allowed us to distinguish our work from the project that has been done before (see Section 1.3). Overall, while decentralising our working efforts, we managed not only to fulfil the primary requirement and goal of this project, but also to explore and implement new aspects and features, as well as to provide a suitable alternative for TimeZero mapping client.

6.6   Workload Distribution and Organisation

Throughout the course of the project, every week two group meetings were conducted. Primarily utilised for sharing progress and keeping every group member up to date, as well as to brainstorm certain aspects of the project and plan out further work. The workload was divided into 3 major areas: data processing application, mapping client as well as report writing and documentation.

GitHub was one of the primary tools used in this project. Used for both data storage and sharing purposes, it also provided a space to distribute the workload via the usage of Kanban boards. The code and other relevant data were stored in 2 repositories, one for KML Creator and another for mapping client.

6.7   Requirements

Overall, the primary requirement from SeaVis was successfully fulfilled. The current version of KML Creator is capable of converting the raw data into the type that can be successfully imported into TimeZero. We also fulfilled some of the SeaVis' recommendations such as making sure that KML is the format of the output file. Suggestions regarding the visual aspects of GUI were also taken into account. However, some recommendations were not fulfilled. The options of forecasting the mussel growth and representing the data as 3D were disregarded, due to the limitations that TimeZero provided. It was decided not to include those options in the mapping client, as we wanted to pursue other aspects such as data security. As for our own requirements, we managed to fulfil all of the major ones, both general and specific ones for KML Creator and mapping client. Additionally we showcased the final version of the KML Creator to SeaVis's representative and got positive feedback confirming that the we have indeed fulfilled their requirements.

6.8   Future Directions

While our final product is sufficient, it also provides a base for further development. Some general improvements such as, bug fixing, seeking solutions to make the programme execution faster and more efficient, as well as improving the visual and technical representation could be implemented. However, it is also possible to outline a sample of more specific directions for the further development of both programs.

KML Creator The KML Creator fulfils its task well, but there are some ways that it could be improved.

- Submission is not possible when any of the table's values are blank and the user receives an error message pop-up box. However, this is the only user input validation used, which means that the user could input none-sense values and crash the programme. We did not prioritise user input validation for the View because we knew that a single user (Bjørn from SeaVis) would be using the programme and that the scope for issues would thus be limited. More thorough user validation would be a good idea, nonetheless, and could be easily added.

- Once a submission is underway, it is not possible to halt it, unless you force quit the whole. Ideally, there should be a way of interrupting the thread containing the running process, bringing it to a stop.

- Although the console box (in ConsoleFrame) is scrollable (you can use the wheel on your mouse to navigate up and down), the scrollbar itself has not been properly implemented. A scrollbar would aid usability

For the KML Creator, one can expand the number of file types that can be obtained after processing the raw CSV file and provide a user with the option to choose which file type is desirable as an output. Another potential direction could be the implementation of predicting the relevant data based on the imported data-set via the implementation of machine learning aspects. An example of such could be predicting the change of mussel density over time.

Mapping Client For the mapping client, different ways of data representation such as 3D view could be introduced. Providing a user with more tools for interacting with the data could be another viable expansion path. For example, if the mapping client could construct a route that results in the most energy-efficient and effective route for mussel harvesting based on the imported data. Making the application more informative, but keeping it relatively simple and user-friendly can also be useful to improve the user experience and increase the adoption of the mapping system. Additionally, the inclusion of the search and filter functions, can allow a user to quickly locate specific information and analyse the mussel data in a more meaningful way.

Overall, both the KML Creator and the mapping client can be expanded in various directions.

7   Conclusion

In conclusion, we have successfully managed to create two standalone applications, fulfilling all of the requirements stated by SeaVis and ones that we implemented ourselves. KML Creator allows SeaVis to convert raw data into a format that can be displayed in TimeZero, while the mapping client provides a user-friendly way of displaying the formatted data, showcasing the mussels on the seafloor and including the data security features such as the login service. Therefore, we successfully answered our research questions outlined in the introduction of this report. Throughout the development of this project, wide-ranging research has been conducted on many topics, such as interpolation, polygonisation, and data security. Extensive testing and bug fixing was constantly being applied throughout the development stage. The final product is comprehensive and self-sufficient holistic solution with two applications that are used to format and display the input data. Both applications contain a significant amount of features, and provide an extensive base for further development. Overall, the project allowed us to learn a lot of new concepts and most importantly to understand and go through the different stages of app development process.

References

[1] T. E. O'Neill and E. S. Pedersen, *SeaVis Interview*, Full interview can be found in the Appendix (8.6), 2023.

[2] C. F. C. Saurel, *DTU Project Bank*, Accessed: May 8, 2023. [Online]. Available: https://projektbank.dtu.dk/en-us/Pages/BulletinView.aspx?EntityId=bcda62b3-a939-e811-8116-005056a057de.

[3] River Cottage Reunited, *Discover the benefits of rope-grown mussels*, Accessed: May 8, 2023. [Online]. Available: https://www.learningwithexperts.com/river-cottage-reunited/projects/discover-the-benefits-of-rope-grown-mussels.

[4] M. H. Trojahn, M. V. J. Puggaard, and M. J. V. Jørgensen, "Mussels in Limfjorden - A Data Visualisation Software of Mussel Density," 2022.

[5] Pre-commit, *pre-commit*, Accessed: May 22, 2023. [Online]. Available: https://pre-commit.com/.

[6] Making Sense Remotely, *Spatial interpolation — moving average*, Accessed: April 26, 2023, 2021. [Online]. Available: https://www.youtube.com/watch?v=OfC3KpL4PRw.

[7] F. Warmerdam and E. Rouault, *GDAL_polygonize.py*, Accessed: May 14, 2023. [Online]. Available: https://gdal.org/programs/gdal_polygonize.html.

[8] S. Documentation, *object.simplify*, Accessed: May 18, 2023. [Online]. Available: https://shapely.readthedocs.io/en/stable/manual.html#other-transformations.

[9] S. Documentation, *unary_union*, Accessed: May 21, 2023. [Online]. Available: https://shapely.readthedocs.io/en/stable/reference/shapely.unary_union.html.

[10] M. Gharbi, A. Koschel, and A. Rausch, *Software Architecture Fundamentals: A Study Guide for the Certified Professional for Software Architecture®–Foundation Level–iSAQB compliant*. dpunkt. verlag, 2019.

[11] Kent C. Dodds, *How to type a React form onSubmit handler*, Accessed: May 23rd, 2023. [Online]. Available: https://epicreact.dev/how-to-type-a-react-form-on-submit-handler/.

[12] W3Schools, *JavaScript RegExp Reference*, Accessed: May 23rd, 2023. [Online]. Available: https://www.w3schools.com/jsref/jsref_obj_regexp.asp.

[13] GeeksforGeeks, *Understanding rainbow table attack*, Accessed: May 21, 2023, 2023. [Online]. Available: https://www.geeksforgeeks.org/understanding-rainbow-table-attack/.

[14]   Auth0, *Introduction to json web tokens*, Accessed: May 21, 2023, 2023. [Online]. Available: https://jwt.io/introduction.

[15]   GeeksforGeeks, *Session vs token-based authentication*, Accessed: May 21, 2023, 2022. [Online]. Available: https://www.geeksforgeeks.org/session-vs-token-based-authentication/.

[16]   R. Karthika, "Providing password security by salted password hashing using bcrypt algorithm," 2015.

[17]   Mozilla Developer Network, *500 Internal Server Error*, Accessed: May 23rd, 2023. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/500.

[18]   NumFOCUS, Inc., *Pandas*, Accessed: May 14, 2023. [Online]. Available: https://pandas.pydata.org/about/index.html.

[19]   GeoPandas Developers, *GeoPandas*, Accessed: May 13, 2023. [Online]. Available: https://geopandas.org/en/stable/about.html.

[20]   Environmental Systems Research Institute (ESRI), *Shapefile Technical Description*, Accessed: May 12, 2023, 1998. [Online]. Available: https://www.esri.com/content/dam/esrisites/sitecore-archive/Files/Pdfs/library/whitepapers/pdfs/shapefile.pdf.

[21]   F. Warmerdam and E. Rouault, *GDAL*, Accessed: May 13, 2023. [Online]. Available: https://gdal.org/index.html.

[22]   Sphinx, *geopy.distance module - geopy 2.2.0 documentation*, Accessed: May 14, 2023. [Online]. Available: https://geopy.readthedocs.io/en/stable/#module-geopy.distance.

[23]   Google Developers, *KML Reference*, Accessed: May 14, 2023. [Online]. Available: https://developers.google.com/kml/documentation/kmlreference.

[24]   D. Cortesi, *PyInstaller*, Accessed: May 13, 2023. [Online]. Available: https://pyinstaller.org/en/stable/.

[25]   Matplotlib Contributors, *Matplotlib*, Accessed: May 15, 2023, 2023. [Online]. Available: https://matplotlib.org/.

[26]   Leafletjs, *Leaflet*, Accessed: May 23rd, 2023. [Online]. Available: https://leafletjs.com/index.html.

[27]   npm, *leaflet-kml*, Accessed: May 23rd, 2023, 2019. [Online]. Available: https://www.npmjs.com/package/leaflet-kml.

[28] Axios, *What is Axios?* Accessed: May 23rd, 2023. [Online]. Available: https://axios-http. com/docs/intro.

[29] Amazon Web Services, *AWS Lambda: Developer Guide*, Accessed: May 9, 2023, 2023. [Online]. Available: https://docs.aws.amazon.com/pdfs/lambda/latest/dg/lambda-dg.pdf.

[30] A. W. Services, *Amazon API Gateway: Developer Guide*, Accessed: May 9, 2023, 2023. [Online]. Available: https://docs.aws.amazon.com/pdfs/apigateway/latest/developerguide/apigatewa y-dg.pdf.

[31] A. S. Gillis, *Rest api (restful api)*, Accessed: May 23rd 2023, 2020. [Online]. Available: https: //www.techtarget.com/searchapparchitecture/definition/RESTful-API#:~:text=A%5C% 20RESTful%5C%20API%5C%20is%5C%20an,deleting%5C%20of%5C%20operations%5C% 20concerning%5C%20resources..

[32] Amazon Web Services, *Amazon DynamoDB Developer Guide*, Accessed: May 23rd, 2023. [On-line]. Available: https://docs.aws.amazon.com/pdfs/amazondynamodb/latest/developerguid e/dynamodb-dg.pdf.

[33] Amazon Web Services, *What is Amazon S3?* Accessed: May 23rd, 2023. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html.

[34] npm, *bcrypt.js*, Accessed: May 23rd, 2023, 2017. [Online]. Available: https://www.npmjs.com/ package/bcryptjs.

[35] JSON Web Tokens, *Introduction to JSON Web Tokens*, Accessed: May 23rd, 2023. [Online]. Available: https://jwt.io/introduction.

[36] React Router, *Main Concepts*, Accessed: May 22, 2023. [Online]. Available: https://reactrout er.com/en/main/start/concepts.

[37] *Mocha*, Accessed: May 21, 2023. [Online]. Available: https://mochajs.org/.

[38] *Chai*, Accessed: May 21, 2023. [Online]. Available: https://www.chaijs.com/.

[39] *Axios-mock-adapter*, Accessed: May 21, 2023. [Online]. Available: https://www.npmjs.com/ package/axios-mock-adapter.

[40] MyTimeZero, *TimeZero Navigation*, Accessed: May 8, 2023. [Online]. Available: https:// mytimezero.com/about.

[41] F. Warmerdam and E. Rouault, *gdal_grid*, Accessed: May 15, 2023. [Online]. Available: https: //gdal.org/programs/gdal_grid.html.

[42] Electron, *What is electron?* Accessed: May 22, 2023. [Online]. Available: https://www.electr onjs.org/docs/latest.

[43] Anubhav Omar, *Building an App with Electron React*, Accessed: May 23rd, 2023, 2023. [Online]. Available: https://www.scaler.com/topics/react/electron-react/.

[44] Distinguished, *3rd party tools and libraries*, Accessed: May 22, 2023. [Online]. Available: https: //distinguished.io/blog/third-party-software-pros-and-cons.

[45] Cambridge University Press, *Polygon definition*, Accessed: May 17, 2023. [Online]. Available: https://dictionary.cambridge.org/dictionary/english/polygon.

[46] W3Schools, *Interpolation*, Accessed: May 17, 2023. [Online]. Available: https://www.w3schoo ls.com/python/scipy/scipy_interpolation.php.

[47] QGIS, *Raster data*, Accessed: May 17, 2023. [Online]. Available: https://docs.qgis.org/3.28/ en/docs/gentle_gis_introduction/raster_data.html.

[48] Britannica, *Graphical User Interface*, Accessed: May 22, 2023. [Online]. Available: https:// www.britannica.com/technology/graphical-user-interface.

[49] P. Magazine, *Definition of bucket*, Accessed: May 22, 2023. [Online]. Available: https://www. pcmag.com/encyclopedia/term/bucket.

[50] A. W. Services, *Application Programming Interface*, Accessed: May 22, 2023. [Online]. Available: https://aws.amazon.com/what-is/api/.

[51] GeeksforGeeks, *Hypertext Transfer Protocol*, Accessed: May 22, 2023. [Online]. Available: htt ps://www.geeksforgeeks.org/http-full-form/.

[52] GeeksforGeeks, *White Box and Black Box testing*, Accessed: May 22, 2023. [Online]. Available: https://www.geeksforgeeks.org/differences-between-black-box-testing-vs-white-box-testing/.

[53] P. Magazine, *Comma-Separated Values*, Accessed: May 22, 2023. [Online]. Available: https: //www.pcmag.com/encyclopedia/term/csv.

[54] Fileformat, *Keyhole Markup Language file*, Accessed: May 22, 2023. [Online]. Available: https: //docs.fileformat.com/gis/kml/.

[55] F. Warmerdam and E. Rouault, *Geospatial Data Abstraction Library (GDAL) - GeoTIFF*, Accessed: May 14, 2023. [Online]. Available: https://gdal.org/drivers/raster/gtiff.html.

8 Appendix

8.1 Nomenclature

This section contains definitions and technical terms used throughout the paper.

- A polygon is a geometric shape with straight sides and enclosed by a closed path. The sides of a polygon are always straight [45].

- Interpolation is a technique used to estimate or predict values within a given set of data points based on the known values. It involves filling in the gaps between the existing data points to create a continuous function or curve [46].

- Raster data is a type of geospatial data that represents information in a grid or cell-based format. Each cell corresponds to a specific location on the Earth's surface [47].

- GUI stands for Graphical User Interface. It allows users to interact with a computer system or software application through visual elements. GUIs provide a more intuitive and efficient way for users to interact with a program [48].

- A bucket is a concept that refers to a container or directory used to organize and store files or objects. Buckets are typically associated with object storage systems, such as cloud storage services [49].

- API stands for Application Programming Interface. It is a set of rules, protocols, and tools that defines how different software components or systems should interact with each other [50].

- HTTP stands for Hypertext Transfer Protocol. It is an application-layer protocol used for communication and data transfer on the World Wide Web. HTTP defines the format and rules for how web browsers, web servers, and other web clients communicate with each other to request and deliver web resources [51].

- Whitebox testing, is a testing technique where the tester has knowledge of the internal workings of the system being tested. The tester has access to the source code, architecture, and design of the software. The primary objective of whitebox testing is to identify defects in the software's internal logic, code structure, and design [52].

- Blackbox testing is a software testing technique that involves testing the functionality of a system without any knowledge of its internal workings. The testing is conducted based on inputs and outputs and the expected behaviour. The objective of blackbox testing is to identify defects in the system's functionality, including its user interface, database, network communication, and other external interfaces. [52].

- CSV (Comma-Separated Values) is a plain text file format commonly used for storing and exchanging tabular data. In a CSV file, each line represents a row of data, and the values within each row are separated by commas [53].

- A KML file (Keyhole Markup Language file) is an XML-based file format used to display

57

geographic data. KML files contain information about geographical features, such as points, lines, polygons, and images.[54].

- A GeoTIFF file is a Tagged Image File Format (TIFF) file that contains both image data and additional geospatial metadata, allowing the image to be located and aligned in a geographic space. The metadata includes information on the coordinate reference system , projection, extent and pixel resolution [55].

8.2 PyInstaller Command

Here is the PyInstaller command to create the executable for the KML Creator. Please note that "path/to/" is used for file and directory paths instead of the actual path, as it represents the paths from the computer where the command was executed.

```
pyinstaller --noconfirm --onedir --windowed
--name "KML Creator" --hidden-import "geopandas"
--paths "path/to/mussel_data/venv/Lib/site-packages" --hidden-import "osgeo"
--paths "path/to/mussel_data/venv/Lib/site-packages/osgeo"
--paths "path/to/mussel_data/venv/Lib/site-packages/osgeo_utils"
--hidden-import "gdal" --hidden-import "tkinter" --hidden-import "tkinter.ttk"
--hidden-import "tkinter.colorchooser"
--add-data "path/to/mussel_data/src/controller;controller/"
--add-data "path/to/mussel_data/src/model;model/"
--add-data "path/to/mussel_data/src/view;view/"
--hidden-import "geopy" --hidden-import "geopy.distance"
--hidden-import "simplekml" --hidden-import "osgeo_utils"
--hidden-import "osgeo_utils.gdal_polygonize"
--add-data "path/to/mussel_data/src/presenter;presenter/"
--add-data "path/to/mussel_data/temp_files;temp_files/"
--add-data "path/to/mussel_data/temp_files/app.log;temp_files/"
"path/to/mussel_data/__main__.py"
```

8.3 KML Creator File Structure

In this section we examine the file structure of the Mussel Data Client. The client has the following directories, which we outline here and will then examine in turn.

```
mussel_data
├── README.md
├── LICENSE.txt
├── requirements.txt
├── config.py
├── setup.py
├── __main__.py
├── temp_files
│   ├── saved_state.json
│   ├── app.log
│   ├── binned_shp_file
│   ├── csv_files
│   ├── kml_files
│   ├── polygon_shp_files
│   ├── shp_files
│   └── tif_files
├── tests
│   ├── tests_backend.py
│   ├── tests.log
│   └── context
│       ├── bins_for_testing.py
│       ├── expected.kml
│       ├── expected.tif
│       ├── expected_bin.ship.zip
│       ├── expected_polygons.shp.zip
│       └── test.csv
└── src
    ├── __init__.py
    ├── controller
    │   ├── __init__.py
    │   └── controller.py
    ├── model
    │   ├── __init__.py
    │   ├── bin.py
    │   ├── preper.py
    │   └── runner.py
    └── view
        ├── __init__.py
        ├── gui_button_frame.py
        ├── gui_console.py
        ├── gui_header.py
        ├── gui_main.py
        ├── gui_settings.py
        ├── gui_submit.py
        ├── gui_table.py
        ├── gui_table_row.py
        └── gui_table_row_fields.py
```

The root folder contains the following files:

File name	Description
README.md	Instructions about how to run the app
LICENSE.txt	Open-source license of the app
Requirements.txt	List of dependencies
config.py	Global variables for the configuration of the app's backend
setup.py	setup.py configures project packaging, distribution, and installation
__main__.py	The __main__.py file is the entry point of our app
temp_files	Contains temporary files
test	Contains files and directories for testing
src	Contains the source code for the project

The **temp_files** folder contains the following files and directories.

File name	Description
saved_state.json	Saves last state of application
app.log	Logs of applications
binned_shp_file	Contains the binned shape files
csv_files	Contains temporary CSV files
kml_files	Contains temporary KML files
polygon_shp_files	Contains temporary polygon SHP files
shp_files	Contains temporary SHP files
tif_files	Contains temporary TIF files

The **test** folder contains the following files and directories.

File name	Description
test_backend.py	This script tests class and method functionality.
tests.log	Logging of tests
context	Contains files for testing purposes

The **context** folder contains the following files.

File name	Description
bins_for_testing.py	Bin objects are defined for testing the functionality of the class.
expected.kml	The expected KML file outcome
expected.tif	The expected TIF file outcome
expected_bin.shp.zip	The expected SHP file outcome
expected_polygons.shp.zip	The expected polygon SHP file outcome
test.csv	A CSV file for testing

The **src** folder contains the following files and directories.

File name	Description
__init__.py	Used for packaging purposes
controller	Contains the necessary files for the controller module
model	Contains the necessary files for the model module
view	Contains the necessary files for the view module

The **controller** folder contains the following files.

File name	Description
__init__.py	Used for packaging purposes
controller.py	Manages GUI state and facilitates communication between the view and the model

The **model** folder contains the following files.

File name	Description
__init__.py	Used for packaging purposes
bin.py	Represents a range of values with associated attributes for grouping and visualisation purposes.
preper.py	Creates SHP files, calculates data-set dimensions, and handles data conversions.
runner.py	Handles data processing tasks, like SHP creation, KML generation, interpolation, etc.

The **view** folder contains the following files.

File name	Description
__init__.py	Used for packaging purposes
gui_button_frame.py	A frame with buttons for the GUI
gui_console.py	A frame the a GUI that contains a console for users to view the program's output
gui_header.py	The header frame in the GUI
gui_main.py	Main GUI window with CSV file handling, table creation, data submission, etc.
gui_settings.py	A frame with stored settings
gui_submit.py	Collects data from GUI and submits it to the controller
gui_table.py	A frame containing a table for the GUI. Allows adding, removing and retrieving data.
gui_table_row.py	Represents a row in the table for the GUI
gui_table_row_fields.py	Defines custom widget classes for enhancing GUI table rows

8.4 Sample KML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2"
xmlns:gx="http://www.google.com/kml/ext/2.2">
    <Document id="1">
        <Placemark id="3">
            <Style id="8">
                <PolyStyle id="9">
                    <color>ff9d96d#4</color>
                    <colorMode>normal</colorMode>
                    <fill>1</fill>
                    <outline>0</outline>
                </PolyStyle>
            </Style>
            <name>MultiPoly</name>
            <MultiGeometry id="2">
                <Polygon id="4">
                    <outerBoundaryIs>
                        <LinearRing id="6">
                            <coordinates>
                            8.96033868454661,56.975680617394964,0.0
                            8.9592654407161,56.975499745798324,0.0
                            8.958728818800846,56.97577105319328,0.0
                            8.957521419491524,56.975499745798324,0.0
                            </coordinates>
                        </LinearRing>
                    </outerBoundaryIs>
                </Polygon>
            </MultiGeometry>
        </Placemark>
    </Document>
</kml>
```

8.5 Mapping Client File Structure

In this section, we examine the file structure of the Mapping Client. The client has the following directories, which we outline here and will then examine in turn.

```
mapping client
├── .babelrc
├── .env
├── package.json
├── public
│   └── index.html
├── src
│   ├── App.js
│   ├── Client.js
│   ├── index.css
│   ├── index.js
│   ├── LeafletMap.css
│   ├── Login.js
│   ├── Register.js
│   ├── route
│   │   ├── PrivateRoute.js
│   │   └── PublicRoute.js
│   └── service
│       ├── AuthService.js
│       └── ValidationService.js
└── test
    ├── registerValidation.test.js
    └── registerApi.test.js


User authentication
├── index.js
├── package.json
├── service
│   ├── login.js
│   ├── register.js
│   └── verify.js
└── utils
    ├── auth.js
    └── util.js


Generate presigned URLs
├── index.js
└── package.json
```

The **Mapping client** folder contains the following files and directories.

File name	Description
.babelrc	Babel set-up for our unit and integration tests
.env	Used to store secret variables, such as API Keys and endpoints
package.json	Node.js relevant file for managing metadata and dependencies of the application
public	Directory that contains the files served directly by the server
src	Directory where our source code is stored
test	Directory where out unit and integration tests are stored

The **public** folder contains the following files.

File name	Description
index.html	Main HTML file for the Mapping Client, which render the contents of our React component

The **src** folder contains the following files and directories.

File name	Description
App.js	Main component of the Mapping Client, responsible for rendering based on routing
Client.js	The component which holds the contents of our client page
index.css	The style-sheet for the whole Mapping Client
index.js	Main entry point of the application responsible for rendering App.js
LeafletMap.css	The style-sheet for our Leaflet map
LeafletMap.js	Leaflet component initialising the map and make API calls to fetch presigned URLs for KML files
Login.js	The component which holds the contents of our Login page
Register.js	The component which holds the contents of our Register page
route	Directory that contains our routing components
service	Directory that contains our service functions

The **route** folder contains the following files.

File name	Description
PrivateRoute.js	The component that checks if a user should have access to our private routes
PublicRoute.js	The component that checks if a user should have access to our public route

The **service** folder contains the following files.

File name	Description
AuthService.js	Module which contains functions for managing the user session
ValidationService.js	Module which contains functions for user input validation

The **test** folder contains the following files.

File name	Description
registerValidation.test.js	Unit tests of the validation functions used during registration
registerApi.test.js	Integration tests of registration and login requests

The **User authentication** folder contains the following files and directories.

File name	Description
index.js	Entry point for our API, which routes incoming requests to the appropriate service
package.json	Node.js relevant file for managing metadata and dependencies of the Lambda functions
service	Directory that contains our main Lambda functions for authentication
utils	Directory that contains our utility functions for authentication

The **service** folder contains the following files.

File name	Description
login.js	Lambda function to handle user login
register.js	Lambda function to handle user registration
verify.js	Lambda function to handle token verification

The **utils** folder contains the following files.

File name	Description
auth.js	Functions to generate and verify tokens
util.js	Function to build response for API Gateway

The **Generate presigned URLs** folder contains the following files and directories.

File name	Description
index.js	Lambda function listing all objects and returns presigned URLs
package.json	Node.js relevant file for managing metadata and dependencies of the Lambda function

8.6 Interview with Bjørn from SeaVis

Thomas: This is my, uh, my fellow group member Emil.

Emil: Hi.

Thomas: Um, yeah, thanks for, thanks for meeting, but today we're just, just a short meeting, um, where we could show you the software we've created and also, um, Emil has some questions about, um, Seavis as a company cause, um, so perhaps that sounds okay. We can Yeah. Get started with you, Emil.

Emil: Yeah. So first we're just going to start by asking some very simplistic questions about Seavis, just, uh, for the report part, so we have some sort of scourses. Um, and first, if you just briefly tell us, uh, what is Seavis?

Bjørn: Um, actually right now it's not a company, uh, because it hasn't been founded yet. Um, but we are quite certain of the name and, uh, going to found it here in August this year. Um, so the plan is that we make the drone and then we take some pictures of the sea floor, and then we have, uh, a machine learning algorithm to map the biomass on the pictures. And from that, we would like to have these data maps that you have been working on. So we expect that we can generate some kind of data that looks like the one that we have, uh, given you to work on. And then we are going to sell, like in the beginning, the blue muscle maps to the fishermen's organization, "Blue muscle fishers organization" in Limfjorden.

Emil: Okay. Great. Is there any mission for expansion or is it, is it primarily gonna be in Limfjorden?

Bjørn: We start Limfjorden because we have a customer there. But we plan to expand to the rest of Denmark, uh, like to start up other, uh, other places in Denmark in 2024. And then we would like to include like all relevant areas in 2025, and then as well expand beyond the borders at that time. But it also depends on how many whistles we can have, because one thing is the drone, the drone is quite cheap. Uh, and then we need a boat to drag the drone. Um, and then we would like to drag about 6 or 10 drones at a time. And each drone is going to cost about 70,000 dkk to produce, and we expect our boat to cost about 100 to 150 thousands dkk to repurpose, an old fishing boat.

Emil: So if you could tell us a little bit about the history of seas. How did, how did it start? Um,

Bjørn: Um, it started out by, Claus had have a contact, um, with some ma marine biologist, uh, doing arctic, research at the North Pole. And they were mapping macro seaweed. Um, and there, what they did was that, I think what they did was that they took a door from the boat and then attached barrels to it, and then dragged that after the boat with a camera on it. And then Claus was like, we can do this better. And then they made something. and then that was an older version. I can see if I can find a picture if you want to see it.

Emil: Yeah, that would be great. And also just, Claus, do you have his full name?

Bjørn: Uh, Claus Melvad.

Emil: Okay, great.

Bjørn: Yeah, I will just share some of it on the screen, because if it's the business part, this might make more sense because we actually have a presentation on the business part.

Emil: Okay.

Bjørn: So like this, we can see it. Yeah. So, um, this is the problem. right now, the fishermen is I'm doing trials, um, and there's a growing market for the protein, uh, which these, uh, muscles and, and all starfishes are really like, uh, rich in protein. Uh, um, and then the trawlers are destroy the habitats actually. Um, so we want to do is that we want to limit the, uh, amount of the area that they, they scrape and we want to Yeah. To help them catch them at the right time. Uh, there's no pictures of the old one. No, that was a bad one. Nope, just a sec. Yeah, here we go. Yeah, this is, the old drone

Emil: Great.

Bjørn: You see it?

Emil: Yeah.

Bjørn: So this one was, uh, built for, I think this was actually built for the artic research and then repurposed for muscle, uh, pictures. Um, and then I heard about it because, uh, I had a mechatronics subject at, uh, course at Aarhus university, and they pitched this idea and I was like, yeah, it would be nice to do something that actually like would amount or something and with the possibility to, to be part of a startup. Uh, so in our, in that course, we went and we built that one. which is, which is actually the, like a bigger version of this one. So this was like a prototype test, uh, for composite on, on that.

Thomas: Have you been out in the swimming pool trying to pull, pull the prototype through with a string and..

Bjørn: this one?

Thomas: Yeah.

Bjørn: Uh, not this one, but we did that with the next one. Ah, I don't know if we have pictures of that one. Oh, yeah, there's picture of that one there. And that is build up on a lot of tests, uh, in, in a swimming stadium. Um, and then they, they ended up with this design For the final one. And this, this is the one that we are billing on right now. And now we are doing the next iteration, uh, to make it more eco-friendly and easier to produce. Okay. So, yeah, and here you can see the, the

67

people in the company. So we have claus Melvad uh, that is the ceo, and then we have, uh, Matt that, you know Yeah, right. Uh, he's for business development. And then yeah, we have Nick that he did the first one that I showed you up here. He built that one. Um, and then we have Henrik, he is a fisherman. Yeah. And then we have me and Jalte and he normally is sitting next to me and, and doing a lot of, um, the mechanical stuff and electronics. And I do more of the, what you say, the organizational part of the work and like planning. And I also do all the software.

Emil: Okay.

Bjørn: Yeah. Does that answer your question?

Emil: It does, yeah. I think we have, uh, the things we need for the report now, so Yeah. That's great. Unless you have something to add, Thomas?

Thomas: Just add a more curiosity question. You said that that, that a one of these boats was aiming to pull was said two to six drones behind it, and those drones are each gonna be a hundred meters apart from each other. So then the whole

Bjørn: No, no, no, no.

Thomas: Okay.

Bjørn: Uh, in that case, we would just, uh, have a way wider picture. Yeah. Okay. Um, but you are right, it's going to be really wide because, uh, they are 1.2 meters in width. So to have the 10 that we will, that we aim for, we are going to have like, we are going to have like a band of drones for eight meters on each side of the boat or something like that. Right. So that would be like 20 meters in width that we are going to, to take.

Thomas: So you're covering with 20 meters in width. You're co you're covering that with eight drones with I think that would be 10, but yeah, 10 drone. Okay. So it's actually very, very detailed. So the data set that we got, that was the sort of these columns, almost like these vertical lines with, uh, which are a hundred meters apart, but actually in, in practice these lines will be a lot closer together.

Bjørn: Well then we will have, like you would have 10 lines that are really close.

Thomas: Yeah.

Bjørn: And then you will have a hundred meters, and then you'll have 10 lines that are really close.

Thomas: Okay. Yeah. Uh, alright. Why, why the extra drones? What, what, what benefit does that give you? Why not just one?

Bjørn: It's just to have more coverage Because it's going to be a sale point on like, it's going to be a negotiation with the customer of how many percentage of the area that we cover that we actually

have like documentation of.

Thomas: Right. What stops you putting the drones out really wide then? Like saying the boat like 50 meters on either side. Like is it the construction of the drones? They can't, cause obviously I must have, there must be some sort of steering issues there, right? Because you're pulling the boat at an angle to the direction of the drone. You want the drone to go.

Bjørn: Uh, actually the only thing that stops us is, uh, the array system in some way. And then there is something that is, uh, the drag force Yeah. And the construction of the brick that we drag the drones with. Right. If you get it. Yeah. Yeah. Okay. I see. Because there's, um, there's a lot of force in, in this tracking things through the water.

Thomas: Right Yeah. Okay. Nice. Um, perhaps we can move on now to showing you the software. And we have managed to package the software down into EXE file, which you can run for Windows, uh, just by clicking on it. Um, we're very happy about that because of the whole process of installation. Uh, the, the dependencies for the software is quite, uh, quite a little bit involved. So we're happy that we can package it down into this packaging. We can send that to you next week and where you can have like a play around with the software yourself and maybe give us some, we, we'd appreciate some, maybe a little bit of written feedback on it to include in the report or, or, or just, uh, another, another call where you can tell us about what you think.

Thomas: Um, but for now, maybe we can just show you the software and get you just some initial thoughts about it, if you have any immediate ideas that we could make improvements on. And then we can go away and make those changes before we send you the EXEfile. Um, so here we go. There we go. Can you see my screen?

Emil: Yes.

Bjørn: Yep. Yeah.

Thomas: Good. All right. So we start the thing off and it opens up like this, just a small window inviting you to open up a CSV file's, open up this one. And we have this pop up here. We have these, this area here corresponds to like the bins that we've spoken about. So from zero to 20, for example, 20 to 40, 40 to 60 muscle density or size. Yeah. And the CSV file, you choose, uh, the, the columns of that get populate this list here.

Bjørn: Okay.

Thomas: Of the target columns. So you might also choose like the, the raw data set you sent us has, you know, many other columns. Yeah. Um, and then I think I'll just go with this one slightly smaller data set. Um, and then you can put a description in here. I don't know, um, test. And let's say from zero to 20 and then select a color and opacity. So how transparent,

Bjørn: Hmm.

Thomas: The layer. The layer should be, um, we can add another row if you like. Um, for the next bi, uh, select all deselect all then select columns to remove. And when you're ready, when you, and when you've kind of put all your bins in, look at the settings down here. Um, Pixel size is, we'll put a bit more details here about what these actually mean. Uh, but pixel size, meaning in meters, how, how big should one pixel be?

Bjørn: Hmm. So yeah,

Thomas: At the moment this is one pixel for, for 10 meters, um, radius width. This is the, the, the search ellipses we use for the algorithm. And at the moment it sets a 60, uh, we in width, um, 10 is, is in height. Is is good. Right. To minimize the, in the, in kind of inaccuracy, we, we found that it needs to be above 50, right? Because otherwise we've got a hundred meters between each row, each column. You're not gonna get proper interpolation unless it's over 50. So 60 is just what we've been testing it with. And then smoothing is, um, you know, these after inter interpolation, uh, the resulting rester image file is polygon. So it's made into a vector vile with these shapes, these polygons, these fields. And they can have a certain amount of smoothing. So some of the edges can be more kind of, um, simplified, like some of the sharp edges can be removed.

Bjørn: Mm.

Thomas: And that's also in meters. But, uh, we'll put a bit more information here about, as I said, what these, um, what these fields actually mean. And that could just be like a little pop-up box that comes up to show, show more information about what What they mean.

Bjørn: yeah. Um,

Thomas: then once all that's looking good, we can just go submit and it'll, oh, well it's not doing anything moment at the moment because it's not selected. It's like that.

Bjørn: Then what happens if you have two columns?

Thomas: Two columns? Yeah. So let's do that. So density and size. Oh, it's there. Yeah. Also 20. Just spink like that. Like this, yeah. Then submit, choose a directory to save the KML files to this will be fine. And then here is where the status see the status.

Bjørn: Hmm.

Thomas: So, this is the small data set we than the one you sent. It's just covering this area, Little process bar with the spinning spinner, ¡laugh¿ ¡laugh¿. And I can hear my computer starting to pick up, that the fans are starting to turn on there. Uh, yep. So inspiration polygon organization and then creating the KML files, um, and then de deleting the temporary files and then done. Hold

on. Yeah, there's a few things that need to be repetition here, but, uh, but it does say here that the selection that we made zero to 20, there are no, there's, there's nothing in that bin. So it just skipped and it didn't make any KML files in the end. You can see here bin bin one is empty, ignoring bin two is empty ignoring. Yeah.

Bjørn: Hmm.

Thomas: So we wouldn't have, we didn't get anything out of that. But uh, when you do select bins, which you know, actually have, uh, content, um, then you should get something which looks like, you know, these, which you've, we've sent you before. Yeah. This sort of thing. Um, and this is with the 60 by 60 radius. 60 meter by 60 meter radius. And this is with like the, this, the smoothing is not optimal here. We still need to do some work on that, cuz you can see it's quite kind of jagged, the edges. Um, and you can, I mean you can really see that 60 by 60 cuz you can almost see like the circles, like the circles almost, you know, like one circle here, another circle here.

Bjørn: Yeah.

Thomas: But this is, yeah, basically the output, and you've seen this already. Yeah. Um, this is with a 60, 60, this is with a 60 10, so 60 meters wide, 10 meters height radius and with some different smoothing. Um, that's another one. And these all were the opacity hundred. So I, it, it would be better ready to have the opacity like as or 80. Right. So then you can see like the layering Better.

Bjørn: Yeah.

Thomas: Um, and yeah, and we, we we're quite confused about this data, like how it's in these, uh, these perfect, uh, horizontal lines. Uh, you said this is fabricated, like made up data? Yeah. Yeah. So I guess this is a copy paste, so that's why we get these, these Yeah. Horizontal. Yeah.

Bjørn: Have you seen the, the raw data, like how it's built?

Thomas: Um, what I've, I mean, I've looked at the data set sent us, which is called raw data version one thing. Yeah. Um, but oh, yes. In a setting, I just real, I realize a setting that we need to add here is, uh, angle. So at the moment, the, you know, the search lips both has width and height, but also angle. So if you were to have to do these. At the moment, everything's vertical, like all the lines that you sent us. But if they were at an angle, then you'd want the search lip sources to be at that the same angle, right?

Bjørn: Yeah. So yeah. But in reality they will be, they will not be like, I don't think they're going to be like we have a grid of straight lines. Hmm. It's going to be, we have a line here.

Thomas: Yeah.

Bjørn: And then we have a line with a little angle and then Yeah. Because everything moves in the

71

real world. Right, right. Um, yeah.

Thomas: So there could be, there were, there were live parallel with each,

Bjørn: we will try to make that, but it's, yeah, it will not be completely straight.

Thomas: No, of course not. No.

Bjørn: Well, we hope, we hope to that, you know, using, having access to these settings would help you kind of account for different, you know, you could lay around and see what works.

Thomas: Um, yeah. So anything, does anything stick out to you as needing, needing attention?

Bjørn: Uh, not really.

Thomas: Mm-hmm. um, either with, either with the software or with the, the result.

Bjørn: Yeah. I think this, the result looks stranges, but I think that's because of the way I made the data set.

Yhomas: Mm-hmm.

Bjørn: And I think a lot of it will be like, a lot of the issues that I have with the result is because of the data set. Right. More than it's because of your program. Um, because yeah.

Thomas: Yeah, we've also struggled with this, these lines. Cause we, it does feel very artificial for us. Um, we're also going to test in within the next week test the, the data set you, you sent us, but with some noise added to it. So we're gonna take the, the different points then like introduce a bit of noise, so the shift around a bit within a a within a ellipse, I might reduce something interesting too. Good. All right. And do you think that this seems okay to like reasonably easy to work with this software?

Bjørn: Yeah. Yeah. I was thinking like, would it be possible to have an indication of the lower bound and the upper bound in the dataset when you, uh, put in the row?

Thomas: Yeah. So at the moment, this description box does nothing outside of this software. It doesn't do anything to the KML file that you get at the end at this box here. But what I think is what is possible is that, um, when you click on this, and I think this is also true in time zero, when you click on the layer, you get this box with, uh, at the moment it's just a multi polygon, but you can customize what goes in here. So it might be possible to put the description in there and then like bin, bin information.

Bjørn: Oh, that was not, uh, what I was talking about. Yeah. So if you look at the software you have, yeah. Uh, and here we have the density. Yeah. And especially with the density that I know this is going to be, I would like this to be some kind of percentage, I think. Hmm. We have put in

the data, right.

Thomas: It's, it could be percentage. I haven't seen any values above a hundred. Yeah.

Bjørn: Um, but it, it should be in the top column of the, of the data set that I sent you.

Thomas: Yeah. I can try and open that up.

Bjørn: Um, I have it here, so Yeah, no, that is, uh, uh, kilograms per square meter. Yeah. But it could also be have been, um, it could have been percentage. And if it's percentage, then again, the question is, is it, uh, between zero and one? Yeah. Or is it between zero and 100? So if you, when you get the data set loaded into it could get an indication of what are the values for like lower and upper value. Ah-huh. Okay. I think that would be nice.

Thomas: Yeah. So if it's, if there's no, if it's, yeah, if there's no value above one, right. Then it's percentage, I guess if zero, for example, 0.5 is 50 percent, is that what you mean?

Bjørn: Or, yeah. So is that when, when I, when I, when I choose a tag of column Yeah. Then it, it tells me that, well, the smallest number is this and the largest number is this. Yeah. Uh, in relation to place in my lower and upper bound. But I think it's a nice to have

Thomas: Yeah.

Bjørn: Not a need to have.

Thomas: Right. Right. Okay. Cause then, then you could potentially have the density column, which is related, which is a percentage, and then the size column, which is not a percentage. So you'd have to have, be able to accommodate both. It would be easier if they're all percentages. Right. Cause then we could just change the, the header to a little percentage sign here. Um, yeah, just wondering how, like in a term, in user interface sense, how do we make it so that each row can be, have either be a percentage or just the number?

Bjørn: Um, um, might be well as well if it's percentage, I, I don't, I don't care if it's between, uh, if the percentage is between one and a hundred and zero and a hundred or is be, or is it between zero and one?

Thomas: Right.

Bjørn: Uh, it's just that if, if this, like if next to the lower bound, there was like a little grade out box where I could see this is the lower bound, this is the lowest, uh, value in the data set, that would be enough. Okay. Yeah. Because then I can look at the number and I can see like, okay, if it's between everything is less than one, it's, I know kind of the scale.

Thomas: How about if it was, so you can, you can, for the, all these fields that you see here, you can give them default values or you can give them like a, um, like a grade out default value. So if

you, if there's no value here, it would in gray just give whatever text you want. Yeah. So here you could have as a default when you add a add a new row, it could have like, okay, BM dense, and then here it would, it would give in like a grade out font, uh, text what the lower bound is like, what the lowest, lowest minimal value is. Maybe that would be a way to do it.

Bjørn: That would be perfect.

Thomas: Yeah. Yeah. Okay.

Bjørn: But again, it's a, it's a, it's a nice to have. It's not a need to have because Sure. Of course. I should also be aware of like how the data that I put into it is built up. Yeah. But Okay.

Thomas: No, that's, I think we can do that probably, uh, at the moment we're spending a lot of time just testing it, trying to make it like resilient and bug free.

Bjørn: Yeah. So, yeah. Um, yeah. And I think it looks great. Good. What is the language that you wrote it in?

Thomas: It's, um, so Python,

Bjørn: um, okay.

Thomas: So yeah. Yeah. Using a, a library called the, the interfaces library called to kinter. Yeah. That's good. Yeah. You, you have some programming experience. I, I understand all.

Bjørn: A little. A little. Yeah.

Thomas: But you know, the package that we, we delivered to you will be, will have all the Python files in it. Um, yeah. So, and as well as the EXE files. So any future development can happen quite, quite easily. Yeah. Good. All right. Uh, so we'll send you the EXE file next week and it'd be great if you could have a, have a play with it and see what you think. Yeah,

Bjørn: Yeah. If you would, uh, like to see it, I can show you the Excel file that I used.

Thomas: Yeah.

Bjørn: Because this is the data that I sent you. Right. One second. This is what I sent you. that is built on this.

Emil: Oh, okay.

Bjørn: That doesn't look like it's in lines. That's strange. No, but the thing is that this is squid together. Yeah. So every column here is super, is uh, 10 times the width. Ah, yeah. Um, right. But still, because these are like, yeah, these are like, uh, how you say, these are the one that shift with, uh, a hundred meters. Yeah. Each column is after meter shifts with one meter.

Thomas: Yeah.

Bjørn: Apart. What is the right here? That is 17 pixels. Uh,

Thomas: I don't understand how you're running the nexel without a crashing. ¡laugh¿ the high half a million data points like Excel should be crying. ¡laugh¿.

Bjørn: It did. Yeah. ¡laugh¿. So this one should be 700. Ah-huh. Yeah. So that is the data set that you, that you're looking at. Yeah. Yeah. Okay. Well, I just think it's easier to figure out when it's like this.

Thomas: It is, yeah. And I'm still, I, I'm a little bit still bit puzzled why it doesn't look more like lines. Um, but yeah. Okay.

Bjørn: Well we can look at the muscle density. You looked at the muscle density, right?

Thomas: Yeah. Yeah. And size, yeah.

Bjørn: Basically, yeah. So, and I tried there because it's much wider than in real life. I, I also try to make these one like narrowed down. Yeah. So that would, they will always be longer than wider. Okay. Uh, to try to get something that was kind of round, but yeah. I don't know why it is like, like you are also saying like why it is like these lines.

Thomas: Yeah. It's weird. Yeah. Cause we, we've, we've kind of confirmed this in various ways that it, that the data set is like that because we've done it both with the polygon, like the interpolation algorithm, but we've also just broken the data set into bins and then plotted them onto a graph simply. And we've also noticed the lines Yeah. For each bin are there. So it's, yeah. Confusing. Another thing which just, just as you're showing it as the data set there, something we've, been a bit of a struggle is the column names. Um, you know, computer programs don't, they don't really like, um, brackets and dashes, hyphens and things, um, spaces and, this is likely to be how it's gonna be? like with, with these column names. Is this similar to the

Bjørn: No, the column names are probably going to be different. Different each time.

Thomas: Okay.

Bjørn: not each time, but it's probably going to be computer friendly.

Thomas: Okay.

Bjørn: So, it's going to be like case no strange letters. Yeah. Uh, no signs. Um, but then I would, yeah, I think I would just like destroy them when they came into the program. Like remove them.

Thomas: Right. You can and yeah, because at the moment we've got it set up to, to, you mean you can, the target column can be anything, right. Any of the columns. That's fine. Um, but it's also, it drops any latitude and longitude columns. So if that but it, any, the latitude and longitude need to have a certain name for that to work. Right.

Bjørn: Okay. Yeah. Yeah.

Thomas: So little details like that. Um, at the moment it, they wouldn't drop these columns because it's latitude and then space and then these brackets ns, um, I then maybe that should be a setting that you can set in the program that just allows you to specify what the latitude and longitude columns are called. So they get filtered out of the target columns.

Bjørn: Yeah, yeah, yeah. it depends because we also need to have another program to, to combine the data from the machine learning with the data from the drone, because there's going to be a lock with the GPS data and then an offset.

Thomas: Okay. So that's like a data science kind of task.

Bjørn: Yeah. Yeah, exactly. Yeah.

Thomas: Well we can talk about that after we hand in this submission. If it's, you need help with that, maybe we can do something there. Yeah. Alright, well good. Uh, yeah, I'll, I'll send you an email with, uh, the package in it and, uh

Bjørn: Okay. Any other que comments? A Emil questions?

Emil: No, I think we're good.

Thomas: Yeah. Yeah. Yeah. Great. We'll see you!

Bjørn: ¡laugh¿. Have a good weekend.

Emil: Bye.

Thomas: Bye.

8.7 Code: KML Creator

```
#####################
/__main__.py
#####################
import logging
from config import DEVELOPMENT
from config import TEMP_FILES
from src.view.gui_main import KmlCreatorGui


if DEVELOPMENT:
    logging.basicConfig(
        filename=str(TEMP_FILES / "app.log"),
        filemode="w",
        format="%(name)s - %(levelname)s - %(message)s",
        level=logging.DEBUG,
    )
logger = logging.getLogger(__name__)
logger.info("Starting app")


if __name__ == "__main__":
    view = KmlCreatorGui()
    view.mainloop()


#####################
/config.py
#####################
from pathlib import Path


# Folder paths
SRC_FOLDER = Path("src")
TEMP_FILES = Path("temp_files")
CSV_PATH = TEMP_FILES / "csv_files"
SHP_PATH = TEMP_FILES / "shp_files"
BINNED_SHP_PATH = TEMP_FILES / "binned_shp_files"
POLYGON_SHP_PATH = TEMP_FILES / "polygon_shp_files"
TIF_PATH = TEMP_FILES / "tif_files"
KML_PATH = TEMP_FILES / "kml_files"
SAVED_STATE = TEMP_FILES / "saved_state.json"
```

```
# Test path and file names
TEST_PATH = Path("tests")
TEST_CONTEXT_PATH = TEST_PATH / "context"
TEST_CSV_NAME = "test"
TEST_CSV_DIMENSIONS = (430, 2)
TEST_KML_NAME = "expected"
TEST_SHP_NAME = "expected_bin"
TEST_TIF_NAME = "expected"
TEST_POLYGON_SHP_NAME = "expected_polygons"


# GUI Spinner speed
SPINNER_SPEED = 100


# Assorted backend constants
COLUMNS_TO_DROP = ["index", "lat", "lon", "long", "latitude", "longitude"]
ELLIPSOID = "WGS-84"
PROJECTION = "EPSG:4326"
SHAPEFILE_DRIVER = "ESRI Shapefile"
INTERPOLATION_OUTPUT_FORMAT = "GTiff"
INTERPOLATION_OUTPUT_TYPE = "Byte"
INTERPOLATION_ALGORITHM = "average"


# When True, logging is enabled
DEVELOPMENT = True


####################
/setup.py
####################
from setuptools import find_packages
from setuptools import setup


with open("README.md") as f:
    readme = f.read()


with open("LICENSE") as f:
    license = f.read()


with open("requirements.txt") as f:
```

```
    requirements = f.read().splitlines()


setup(
    name="Seavis: Mussel Data",
    version="1.0.0",
    author="Thomas O'Neill",
    author_email="thomas.oneill@gmail.com",
    long_description=readme,
    packages=find_packages(),
    install_requires=requirements,
    entry_points={
        "console_scripts": [
            "your-command=your_package.module:main_function",
        ],
    },
)


####################
/src/__init__.py
####################


####################
/src/view/gui_console.py
####################
import tkinter as tk
from typing import TYPE_CHECKING


if TYPE_CHECKING:
    from view.gui_main import KmlCreatorGui



class ConsoleFrame(tk.Frame):
    """
    A frame that contains a console for the user to view the output of the program
    :param master: The parent widget
    """

    master: "KmlCreatorGui"
    console_title: tk.Label
```

```python
    console_text: tk.Text
    scrollbar: tk.Scrollbar


    def __init__(self, master: "KmlCreatorGui"):
        super().__init__()
        self.parent = master

        # Destroy the old console if it exists
        if hasattr(master, "console_frame"):
            self.parent.console_frame.destroy()

        # Create a title for the console and anchor it to the left.
        self.console_title = tk.Label(self, text="Console", anchor=tk.W)
        self.console_title.pack(side=tk.TOP, fill=tk.X)

        # Create a text widget for the console. Readonly.
        self.console_text = tk.Text(self, state=tk.DISABLED)
        self.console_text.pack(side=tk.TOP, fill=tk.X)

        # Create a scrollbar for the console
        self.scrollbar = tk.Scrollbar(self, command=self.console_text.yview)
        self.scrollbar.pack(side="right", fill="y")

        # Configure the console to use the scrollbar
        self.console_text.config(yscrollcommand=self.scrollbar.set)


####################
/src/view/gui_main.py
####################
import logging
import tkinter as tk
from pathlib import Path
from tkinter import filedialog
from typing import Any


from presenter.presenter import Presenter
from view.gui_button_frame import ButtonFrame
from view.gui_console import ConsoleFrame
from view.gui_header import Header
```

```python
from view.gui_settings import SettingsFrame
from view.gui_submit import Submit

from .gui_table import TableFrame


logger = logging.getLogger(__name__)


class KmlCreatorGui(tk.Tk):
    """
    The main GUI window.
    """

    presenter: "Presenter"
    settings: dict[str, int]
    csv_file_path: Path = Path()
    saved_rows: list[dict[str, Any]] = []
    header_frame: Header
    table_frame: TableFrame
    console_frame: ConsoleFrame
    button_frame: ButtonFrame
    settings_frame: SettingsFrame
    submit: Submit
    submit_frame: Submit

    def __init__(self):
        # Call the parent constructor
        super().__init__()

        # Set the title of the window
        self.title("KML Creator")

        # Set the default font, size and window size
        self.option_add("*Font", "Verdana 10")
        self.geometry("400x75")

        # Create the header
        self.header_frame = Header(master=self)
```

```python
        # Create a Presenter object to facilitate communication between the view and
        # the model. The presenter will
        # also load the GUI state from the saved_state JSON file.
        self.presenter = Presenter(view=self)
        self.presenter.load_gui_state()


        # Set up callback for when the window is closed. When this happens, the GUI
        # state is saved to the saved_state
        # JSON file.
        self.protocol("WM_DELETE_WINDOW", self.on_close)


    def create_table_and_accessories(self) -> None:
        """
        Create the table and the accessories (buttons, settings, console).
        """
        if not hasattr(self, "csv_file_path"):
            return


        # Create the table
        self.table_frame = TableFrame(
            parent=self,
        )


        # Create the buttons ("Select all", "Deselect all", "Add row", "Delete row",
        # "Submit")
        self.button_frame = ButtonFrame(master=self)
        self.button_frame.pack(side=tk.TOP, fill=tk.Y, anchor=tk.W)


        # Create the settings ("Name", "Pixel Size", "Radius width", Radius height",
        # "Smoothing", "Angle")
        self.settings_frame = SettingsFrame(master=self)
        self.settings_frame.pack(side=tk.TOP, expand=True, fill=tk.BOTH)


        # Create the console
        self.console_frame = ConsoleFrame(master=self)
        self.console_frame.pack(side=tk.TOP, expand=True, fill=tk.BOTH)


        # Set the window size to fit the table
        self.geometry(f"{self.table_frame.width + 10}x800")
```

```python
@staticmethod
def ask_user_file_directory() -> Path | None:
    """
    Open a file dialog window to allow the user to select a directory.
    """
    file_path_str: str = filedialog.askdirectory()
    # If user cancels, return None
    if file_path_str == ():
        return None
    return Path(file_path_str)


@staticmethod
def ask_user_file_path(filetypes=None) -> Path | None:
    """
    Open a file dialog window to allow the user to select a file. The file must
    When the user cancels the file dialog, file_path is an empty tuple. Return if
    :param filetypes: The file types to allow the user to select. Defaults to CSV
    files.
    """
    if filetypes is None:
        filetypes = [("CSV files", "*.csv")]
    file_path: str = filedialog.askopenfilename(filetypes=filetypes)
    if file_path == ():
        return
    return Path(file_path)


def get_rows(
    self, ensure_filled=True, selected_only=False
) -> dict[str, Any] | None:
    """
    Get the rows from the table.
    :param ensure_filled: If True, displays a message box if any required fields are
     empty, and returns None.
     :param selected_only: If True, only return rows that have been selected
     (checkbox is ticked).
    """
    if hasattr(self, "table_frame"):
        return self.table_frame.get_rows(
```

```python
            ensure_filled=ensure_filled, selected_only=selected_only
        )


def get_csv_file_path(self) -> Path | None:
    """
    Get the path to the CSV file.
    """
    if hasattr(self, "header_frame"):
        return self.header_frame.csv_file_path


def get_settings(self) -> dict[str, int | str] | None:
    """
    Get the settings from the settings frame ("Name", "Pixel Size", "Radius width",
    Radius height", "Smoothing",
    "Angle").
    """
    if hasattr(self, "settings_frame"):
        return self.settings_frame.get_settings()


def set_csv_file_path(self, file_path=None) -> None:
    """
    Set the path to the CSV file.
    :param file_path: The path to the CSV file.
    """
    if file_path is None:
        file_path: Path = self.ask_user_file_path()
    # Update the file label with the path to the selected file
    self.header_frame.csv_file_path_label.config(text=str(file_path))
    # Convert to pathlib object
    self.header_frame.csv_file_path = file_path


def set_target_columns(self, target_columns) -> None:
    """
    Set the target columns.
    :param target_columns: The target columns fetched from the column names in the
    CSV file. Certain columns are
    excluded (e.g. "lat", "lon").
    """
    self.target_columns = target_columns
```

```python
def set_saved_rows(self, rows) -> None:
    """
    Set the saved rows.
    :param rows: The rows fetched from the saved_state JSON file.
    """
    self.saved_rows = rows


def set_col_min_max_values(self, column_min_max) -> None:
    """
    Provide the minimum and maximum values for each column in the CSV file.
    :param column_min_max: The minimum and maximum values for each column in the
    CSV file.
    """
    self.column_min_max = column_min_max


def set_settings(self, settings) -> None:
    """
    Set the settings in the settings frame ("Name", "Pixel Size", "Radius width",
    Radius height", "Smoothing",
    "Angle").
    :param settings: The saved settings fetched from the saved_state JSON file.
    """
    # TODO: Set settings.
    pass


def print_to_console(self, text) -> None:
    """
    Print text to the console.
    :param text: The text to print to the console.
    """
    self.console_frame.console_text.configure(state="normal")
    self.console_frame.console_text.insert(tk.END, text)
    self.console_frame.console_text.configure(state="disabled")


def submit(self):
    """
    The method that is called when the "Submit" button is clicked. Creates a Submit
    rows from the table, and then calls the submit method in the presenter. A
```

```python
        submission is in progress.
        """
        self.submit = Submit(master=self)


    def on_close(self) -> None:
        """
        The method that is called when the window is closed. Saves the GUI state to a
        JSON file before closing.
        """
        try:
            self.presenter.save_gui_state()
        except Exception as e:
            logger.error(e)
        self.destroy()
```

####################
/src/view/gui_table_row_fields.py
####################

```python
import tkinter as tk
from dataclasses import dataclass
from tkinter import ttk
from tkinter.colorchooser import askcolor
from typing import Any
from typing import TYPE_CHECKING


if TYPE_CHECKING:
    from view.gui_table_row import TableRow



@dataclass
class RowField:
    col_name: str
    widget: tk.Checkbutton | tk.Entry | tk.Label | ttk.Combobox
    datatype: Any



class TargetColumnCombobox(ttk.Combobox):
    """
    A combobox that displays the target columns. If the user has loaded in a saved row,
```

```
    the combobox will display the target column that was selected when the row was
    saved.
    :param parent: The parent widget.
    :param values: The values to display in the combobox.
    """


    def __init__(self, parent: "TableRow", values: list):
        super().__init__(parent.canvas, values=values)
        self.parent = parent
        if parent.saved_row_data and (col := "target_column") in parent.saved_row_data:
            self.current(parent.target_columns.index(parent.saved_row_data[col]))
        else:
            self.current(0)
        self.bind("<<ComboboxSelected>>", self._update_min_and_max_bounds)


    def _update_min_and_max_bounds(self, *args):
        self.parent.lower_bound_entry.update_to_default_value()
        self.parent.upper_bound_entry.update_to_default_value()




class LowerAndUpperBoundEntry(tk.Entry):
    """
    A text entry box that displays the min and max bounds of the target column. The
    and is erased when the user clicks on the entry box. If the user clicks away from
    anything, the default value is restored.
    :param parent: The parent widget.
    """


    _default: str


    def __init__(self, parent, **kwargs):
        super().__init__(parent.canvas, **kwargs)
        self.parent: TableRow = parent
        self._get_default_value()
        self.bind("<FocusIn>", self.erase_default_value)
        self.bind("<FocusOut>", self.restore_default_value_if_empty)


    def _get_default_value(self):
        min_and_max = self.parent.target_col_min_max[
```

```python
            self.parent.target_column_combo_box.get()
        ]
        self._default = f"Min: {min_and_max[0]}, Max: {min_and_max[1]}"


    def update_to_default_value(self):
        self._get_default_value()
        self.delete(0, tk.END)
        self.insert(0, self._default)
        self.config(fg="grey")


    def erase_default_value(self, event=None):
        if self.get() == self._default:
            self.delete(0, tk.END)
        self.config(fg="black")


    def restore_default_value_if_empty(self, event=None):
        if self.get() == "":
            self.update_to_default_value()




class ColourEntry(tk.Entry):
    """
    A text entry box that displays the colour picked from the colour picker dialog.
    :param parent: The parent widget.
    """


    def __init__(self, parent, **kwargs):
        super().__init__(parent.canvas, **kwargs)
        self.parent: TableRow = parent
        self.bind("<Button-1>", lambda event: self._choose_color())


    def set_colour(self, colour):
        """
        Sets the colour of the entry box to the colour passed in.
        """
        self._choose_color(colour=colour)


    def _choose_color(self, colour: str = None):
        """
```

```python
        Displays the colour picker dialog and sets the text in the entry box to the hex
        :param colour: The colour to set the entry box to. If None, the colour picker
        """
        if not colour:
            colour = askcolor()[1]  # returns a tuple (None, '#ffffff')
        # If a color is selected, set the text in the text entry to the hex value of
        # the colour
        self.delete(0, tk.END)
        self.insert(0, str(colour))
        # set entry box colour to the colour picked
        self.config(bg=colour, fg=colour)


class CheckbuttonWithVar(tk.Checkbutton):
    """
    A checkbutton that has a BooleanVar associated with it. The BooleanVar is set to
    True when the checkbutton is checked, and False when it is unchecked.
    """

    var: tk.BooleanVar

    def __init__(self, *args, **kwargs):
        self.var = tk.BooleanVar()
        super().__init__(*args, variable=self.var, **kwargs)

    def get(self):
        return self.var.get()


####################
/src/view/gui_submit.py
####################
import logging
import tkinter as tk
from typing import TYPE_CHECKING


from config import SPINNER_SPEED


if TYPE_CHECKING:
    from view.gui_main import KmlCreatorGui
```

```python
logger = logging.getLogger(__name__)


class Submit:
    """
    The class that collects the data from the GUI (table and settings) and submits it
    to the presenter. Also displays a spinner while the process is running.
    :param master: The parent widget.
    """

    master: "KmlCreatorGui"
    spinner_title: tk.Label
    spinner: tk.Label

    def __init__(self, master: "KmlCreatorGui"):
        self.master = master

        # Clear parent.console_frame.console
        self.master.console_frame.console_text.config(state=tk.NORMAL)
        self.master.console_frame.console_text.delete("1.0", tk.END)
        data = {}

        # Get the save directory from the user
        save_directory = master.ask_user_file_directory()
        # If user presses cancel, return.
        if not save_directory:
            logger.info("User cancelled file dialog")
            return
        logger.info(f"Save directory: {save_directory}")

        # Get the row data from the table
        try:
            if bins := master.get_rows(ensure_filled=True, selected_only=True):
                data["bins"] = bins
            else:
                return
        except ValueError as e:
            logger.error(e)
```

```python
        self.master.print_to_console(
            "There was a problem collecting the data from the table."
        )
        return


    # Get the data from the settings
    data["settings"] = master.settings_frame.get_settings()
    logger.info(f"Settings: {data['settings']}")


    if data:
        self.submit_start_spinner()
        master.presenter.initialize_and_run_process(
            data=data, save_directory=save_directory
        )


def submit_start_spinner(self) -> None:
    """
    Start the spinner animation.
    """
    # Disable the Submit button so the user can't click it again
    self.master.button_frame.submit_button.config(state="disabled")


    # Create the spinner title
    self.spinner_title = tk.Label(self.master.button_frame, text="Processing...")
    self.spinner_title.pack(side=tk.LEFT, pady=10)
    # Create the spinner
    self.spinner = tk.Label(self.master.button_frame, text="--")
    self.spinner.pack(side=tk.LEFT, pady=10)
    # Start the spinner animation
    self.submit_animate_spinner()


def submit_animate_spinner(self) -> None:
    """
    Animate the spinner. The animation is done by rotating the spinner 30 degrees
    every number of milliseconds set by config.SPINNER_SPEED.
    """
    # Rotate the spinner by 30 degrees
    text = self.spinner.cget("text")
    spin_dict = {"--": "\\", "\\": "|", "|": "/", "/": "--"}
```

```python
        self.spinner.config(text=spin_dict[text])

        # Schedule the next rotation after SPINNER_SPEED milliseconds
        if self.master.button_frame.submit_button.cget("state") == "disabled":
            self.master.table_frame.after(SPINNER_SPEED, self.submit_animate_spinner)


    def submit_finish(self) -> None:
        """
        Finish the submit process. This is called by the presenter when the process is
        finished.
        """
        # Remove the spinner and spinner title
        self.spinner_title.pack_forget()
        self.spinner.pack_forget()

        # Re-enable the submit button
        self.master.button_frame.submit_button.config(state="normal")


####################
/src/view/gui_button_frame.py
####################
import tkinter as tk
from typing import TYPE_CHECKING


if TYPE_CHECKING:
    from view.gui_main import KmlCreatorGui



class ButtonFrame(tk.Frame):
    """
    The frame containing the buttons for the GUI. Any existing buttons are destroyed
    before the new ones are created.
    :param master: The parent widget.
    """

    master: "KmlCreatorGui"
    select_all_button: tk.Button
    deselect_all_button: tk.Button
    add_button: tk.Button
```

```python
    remove_button: tk.Button
    submit_button: tk.Button


    def __init__(self, master: "KmlCreatorGui"):
        super().__init__(master)
        self.parent = master

        # Destroy the buttons if they already exist
        if hasattr(self.parent, "button_frame"):
            self.parent.button_frame.destroy()

        button_data: list[tuple[str, callable]] = [
            ("Select All", lambda: master.table_frame.select_all(True)),
            ("Deselect All", lambda: master.table_frame.select_all(False)),
            ("Add Row", master.table_frame.add_row),
            ("Remove Row", master.table_frame.remove_rows),
            ("Submit", master.submit),
        ]

        for text, command in button_data:
            button = tk.Button(self, text=text, command=command)
            setattr(self, text.lower().replace(" ", "_") + "_button", button)
            button.pack(side=tk.LEFT, padx=10, pady=10, anchor=tk.W)


####################
/src/view/gui_header.py
####################
import tkinter as tk
from pathlib import Path
from typing import TYPE_CHECKING

if TYPE_CHECKING:
    from view.gui_main import KmlCreatorGui



class Header(tk.Frame):
    """
    Header frame for the GUI.
    :param master: The parent widget.
```

```python
        """

        master: "KmlCreatorGui"
        csv_file_path: Path = Path()
        load_csv_text: tk.Label
        load_csv_button: tk.Button
        csv_file_path_label: tk.Label

        def __init__(self, master: "KmlCreatorGui"):
            super().__init__()
            self.master = master
            self.pack(side=tk.TOP, expand=True, fill=tk.BOTH)

            self.load_csv_text = tk.Label(self, text="Load file", anchor=tk.W)
            self.load_csv_text.pack(side=tk.TOP, fill=tk.X, expand=True)

            self.load_csv_button = tk.Button(
                self,
                text="CSV file",
                command=self.load_csv_button,
            )
            self.load_csv_button.pack(side=tk.LEFT, padx=10, pady=5)

            self.csv_file_path_label = tk.Label(self, text="No file selected")
            self.csv_file_path_label.pack(side=tk.LEFT, padx=10, pady=5)

        def load_csv_button(self) -> None:
            """
            Callback for the load_csv_button. If the user presses cancel, the function
            returns and nothing happens.
            """
            csv_file_path = self.master.ask_user_file_path()
            if csv_file_path in [None, ""]:
                return
            self.master.presenter.load_gui_state(csv_file_path=csv_file_path)
```

####################
/src/view/gui_settings.py
####################

```python
import tkinter as tk
from typing import TYPE_CHECKING


if TYPE_CHECKING:
    from view.gui_main import KmlCreatorGui



class SettingsFrame(tk.Frame):
    """
    The settings frame. The settings are stored in a dictionary, with the key being the
    being a tuple containing the default value and the datatype of the value. Any
    the new ones are created.
    :param master: The parent widget.
    """

    master: "KmlCreatorGui"
    settings: dict[str, tuple[str | int, type]]
    settings_title: tk.Label
    name_label: tk.Label
    name_entry: tk.Entry
    pixel_size_label: tk.Label
    pixel_size_entry: tk.Entry
    radius_width_label: tk.Label
    radius_width_entry: tk.Entry
    radius_height_label: tk.Label
    radius_height_entry: tk.Entry
    angle_label: tk.Label
    angle_entry: tk.Entry
    smoothing_label: tk.Label
    smoothing_entry: tk.Entry
    simplification_label: tk.Label
    simplification_entry: tk.Entry

    def __init__(self, master: "KmlCreatorGui"):
        super().__init__(master)
        self.master = master

        # Destroy the old settings if they exist
        if hasattr(self.master, "settings_frame"):
```

```python
            self.master.settings_frame.destroy()


        # Settings with default values
        self.settings = {
            "name": ("Batch 1", str),
            "pixel_size": (10, int),
            "radius_width": (60, int),
            "radius_height": (60, int),
            "angle": (0, int),
            "smoothing": (10, float),
            "simplification": (10, float),
        }


        # Create a title for the other settings and anchor it to the left
        self.settings_title = tk.Label(self, text="Settings", anchor=tk.W)
        self.settings_title.pack(side=tk.TOP, fill=tk.X)


        # Create the settings and pack them
        for setting in self.settings:
            label = setting.replace("_", " ").title()
            setattr(
                self,
                setting + "_label",
                tk.Label(self, text=label),
            )
            getattr(self, setting + "_label").pack(side=tk.LEFT, padx=10, pady=5)
            setattr(self, setting + "_entry", tk.Entry(self, width=10))
            getattr(self, setting + "_entry").pack(side=tk.LEFT, padx=10, pady=5)
            getattr(self, setting + "_entry").insert(0, self.settings[setting][0])


def get_settings(self) -> dict[str, int | str]:
    """
    Get the settings from the settings frame.
    """

    settings = {}
    for setting in self.settings:
        datatype = self.settings[setting][1]
        value = getattr(self, setting + "_entry").get()
        if datatype == int:
```

```python
                settings[setting] = int(value)
            elif datatype == str:
                settings[setting] = value  # Already a string
            elif datatype == float:
                settings[setting] = float(value)
            else:
                raise TypeError(f"Type {self.settings[setting][1]} not supported")
        return settings
```

####################
/src/view/gui_table_row.py
####################

```python
import logging
import tkinter as tk
from typing import Any
from typing import TYPE_CHECKING

from view.gui_table_row_fields import CheckbuttonWithVar
from view.gui_table_row_fields import ColourEntry
from view.gui_table_row_fields import LowerAndUpperBoundEntry
from view.gui_table_row_fields import RowField
from view.gui_table_row_fields import TargetColumnCombobox

if TYPE_CHECKING:
    from view.gui_table import TableFrame

logger = logging.getLogger(__name__)


class TableRow:
    """
    A row in the table. Each row contains the columns reflected in TableFrame.headers.
    :param master: The parent widget.
    :param saved_row_data: The saved row data, if any.
    """

    master: "TableFrame"

    # Entry widgets
```

```
check_box: CheckbuttonWithVar
bin_number_label: tk.Label
target_column_combo_box: TargetColumnCombobox
entry_colour: ColourEntry
opacity_entry: tk.Entry
description_entry: tk.Entry
lower_bound_entry: LowerAndUpperBoundEntry
upper_bound_entry: LowerAndUpperBoundEntry


# Saved row data
saved_row_data: dict[str, Any]


# Cell and button dimensions
cell_width: int
cell_height: int
button_width: int
button_height: int


def __init__(self, master: "TableFrame", saved_row_data=None):
    self.row_num = len(master.rows)
    self.saved_row_data = saved_row_data


    # Get the parent attributes
    self.parent = master
    self.cell_width = master.cell_width
    self.cell_height = master.cell_height
    self.canvas = master.canvas
    self.target_columns = master.target_columns
    self.target_col_min_max = master.column_min_max


    # Create entry widgets
    self.row_fields = []
    self.create_entries()


def make_canvas_window(self, num, item) -> None:
    """
    Create a canvas window for the given widget.
    :param num: The column number of the widget.
    :param item: The widget to create a canvas window for.
```

```python
        """

        canvas_y_padding: int = (
            5  # 5 pixels of padding on the left and right of the widget
        )
        canvas_x_padding: int = (
            5  # 5 pixels of padding on the top and bottom of the widget
        )
        canvas_x_offset: int = 10  # 10 pixels of offset from the right edge of the cell
        x1: int = num * self.cell_width
        y1: int = (self.row_num + 1) * self.cell_height
        self.canvas.create_window(
            x1 + canvas_y_padding,
            y1 + canvas_x_padding,
            width=self.cell_width - canvas_x_offset,
            window=item,
            anchor="nw",
        )


    def create_entries(self) -> None:
        """
        Create the entry widgets for this row. First, the widgets are created, setting
        value or a default value. Then, the canvas windows are created for each widget
        the row_fields list.
        """
        # Assign the entry widgets to the row
        self.check_box = CheckbuttonWithVar()
        self.bin_number_label = tk.Label(self.canvas, text=f"{self.row_num + 1}")
        self.target_column_combo_box = TargetColumnCombobox(
            parent=self, values=self.target_columns
        )
        self.description_entry = tk.Entry(self.canvas)
        self.lower_bound_entry = LowerAndUpperBoundEntry(
            parent=self,
        )
        self.upper_bound_entry = LowerAndUpperBoundEntry(
            parent=self,
        )
        self.entry_colour = ColourEntry(parent=self)
        self.opacity_entry = tk.Entry(self.canvas)
```

```python
# Make the canvas window for each widget and append each widget to the
# row_fields list.
# Select column
if self.saved_row_data and (col := "select") in self.saved_row_data:
    self.check_box.var.set(self.saved_row_data[col])
else:
    self.check_box.var.set(False)
self.make_canvas_window(num=len(self.row_fields), item=self.check_box)
self.row_fields.append(
    RowField(col_name="select", widget=self.check_box, datatype=bool)
)
# Bin Number column
self.make_canvas_window(num=len(self.row_fields), item=self.bin_number_label)
self.row_fields.append(
    RowField(col_name="bin_number", widget=self.bin_number_label, datatype=int)
)


# Target Column column
self.make_canvas_window(
    num=len(self.row_fields), item=self.target_column_combo_box
)
self.row_fields.append(
    RowField(
        col_name="target_column",
        widget=self.target_column_combo_box,
        datatype=str,
    )
)


# Description column
if self.saved_row_data and (col := "description") in self.saved_row_data:
    if self.saved_row_data[col]:
        self.description_entry.insert(0, self.saved_row_data[col])
self.make_canvas_window(num=len(self.row_fields), item=self.description_entry)
self.row_fields.append(
    RowField(
        col_name="description", widget=self.description_entry, datatype=str
    )
```

```python
        )

        # Lower and Upper Bounds columns
        for col, widget in [
            ("lower_bound", self.lower_bound_entry),
            ("upper_bound", self.upper_bound_entry),
        ]:
            if self.saved_row_data and col in self.saved_row_data:
                if self.saved_row_data[col] is not None:  # allow 0 as a valid value
                    widget.insert(0, str(self.saved_row_data[col]))
                else:
                    widget.update_to_default_value()
            else:
                widget.update_to_default_value()
            self.make_canvas_window(num=len(self.row_fields), item=widget)
            self.row_fields.append(RowField(col_name=col, widget=widget, datatype=int))


        # Colour column
        if self.saved_row_data and (col := "colour") in self.saved_row_data:
            if colour := self.saved_row_data[col]:
                self.entry_colour.set_colour(colour)
        self.make_canvas_window(num=len(self.row_fields), item=self.entry_colour)
        self.row_fields.append(
            RowField(col_name="colour", widget=self.entry_colour, datatype=str)
        )


        # Opacity Column column
        if self.saved_row_data and (col := "opacity") in self.saved_row_data:
            if self.saved_row_data[col] is not None:  # allow 0 opacity
                self.opacity_entry.insert(0, str(self.saved_row_data[col]))
        self.make_canvas_window(num=len(self.row_fields), item=self.opacity_entry)
        self.row_fields.append(
            RowField(col_name="opacity", widget=self.opacity_entry, datatype=int)
        )


def get_row_data(self) -> dict[str, Any]:
    """
    Returns a dict containing the data from the row. A try/except block is used to
    datatype. If this fails, the field is not added to the row_data dict, as the
```

```python
        invalid data (e.g. a string in a field that should contain an int).
        """
        row_data = {}
        bin_number = None
        for row_field in self.row_fields:
            # The bin number uses a different function to get the data from the widget
            if row_field.col_name == "bin_number":
                bin_number = row_field.widget.cget("text")
                continue

            row_data[row_field.col_name] = self._convert_field_to_datatype(row_field)
        return {bin_number: row_data}


    def destroy_entries(self) -> None:
        """
        Destroys all the entry widgets in the row_fields list.
        """
        for entry in self.row_fields:
            entry.widget.destroy()
        self.row_fields = []


    def is_selected(self) -> bool:
        """
        Returns True if the row is selected, False otherwise.
        """
        return self.row_fields[0].widget.var.get()


    def contains_empty_entry(self) -> bool:
        """
        Returns True if any of the entry widgets in the row_fields list are empty, False
         otherwise. Ignores the
        Bin Number and Select columns.
        """
        for row_field in self.row_fields:
            if row_field.col_name in ["bin_number", "selected"]:
                continue
            if row_field.widget.get() is None:
                return True
            row_field_value = self._convert_field_to_datatype(row_field)
```

```python
            if row_field_value is not None:
                continue
            else:
                return True
        return False


    def _convert_field_to_datatype(self, row_field: RowField) -> Any:
        """
        Converts the data in the row_field to the correct datatype. If the conversion
        fails, None is returned.
        """
        if row_field.col_name == "bin_number":
            return row_field.widget.cget("text")


        value = row_field.widget.get()
        value = None if value == "" else value
        try:
            if isinstance(row_field.datatype(), int):
                value = int(value)
            elif isinstance(row_field.datatype(), float):
                value = float(value)
        except ValueError:
            logger.exception(
                f"Failed to convert {row_field.col_name} to {row_field.datatype()}"
            )
            value = None
        finally:
            return value


###################
/src/view/gui_table.py
###################
import logging
import tkinter as tk
from tkinter import messagebox
from typing import Any


from view.gui_table_row import RowField
from view.gui_table_row import TableRow
```

```python
from view.gui_table_row_fields import CheckbuttonWithVar


logger = logging.getLogger(__name__)



class TableFrame(tk.Frame):
    """
    The frame containing the table for the GUI. Any existing table is destroyed before
    If there are no saved rows (loaded in from the Presenter), a single empty row is
    :param master: The parent widget.
    """

    headers: list[str] = [
        "Select",
        "Bin Number",
        "Target Column",
        "Description",
        "Lower Bound",
        "Upper Bound",
        "Colour",
        "Opacity (%)",
    ]
    rows = []
    width: int
    cell_width: int = 180
    cell_height: int = 35


    target_columns: list[str]
    saved_rows: dict[str, dict[str, Any]]
    column_min_max: dict[str, tuple[float, float]]
    rows: list[TableRow]
    canvas_frame: tk.Frame
    canvas: tk.Canvas
    title: tk.Label
    scrollbar_v: tk.Scrollbar
    scrollbar_h: tk.Scrollbar


    def __init__(self, parent):
        # Call the parent constructor
```

```python
        super().__init__()

        self.parent = parent
        self.rows = []

        # Destroy the table if it already exists
        if hasattr(self.parent, "table_frame"):
            self.parent.table_frame.destroy()

        # Pack the table frame into the parent
        self.pack(side=tk.TOP, expand=True, fill=tk.BOTH)

        # Get the target columns, saved rows, and column min/max from the parent
        self.target_columns = parent.target_columns
        self.saved_rows = parent.saved_rows
        self.column_min_max = parent.column_min_max

        # Create a title for the table and anchor to the left
        self.title = tk.Label(self, text="Table", anchor="w")
        self.title.pack(side=tk.TOP, fill=tk.X)

        # Create a canvas to hold the table, scrollbars, and headers
        self.canvas = tk.Canvas(self)
        self.canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
        self.create_scrollbars()
        self.create_headers()

        # Draw the rows
        if self.saved_rows:
            # If there are saved rows, draw them
            for row in self.saved_rows.values():
                # Check that each saved row has all the headers
                try:
                    self.add_row(row)
                except Exception as e:
                    logger.error(e)
        else:
            # Add a row if there are no saved rows
            self.add_row()
```

```python
        self.update_scrollregion()


def create_scrollbars(self) -> None:
    """
    Create the vertical and horizontal scrollbars for the table.
    """
    scrollbar_v = tk.Scrollbar(self, orient=tk.VERTICAL, command=self.canvas.yview)
    scrollbar_v.pack(side=tk.RIGHT, fill=tk.Y)
    self.canvas.config(yscrollcommand=scrollbar_v.set)
    scrollbar_h = tk.Scrollbar(
        self, orient=tk.HORIZONTAL, command=self.canvas.xview
    )
    # scrollbar_h.pack(side=tk.BOTTOM, fill=tk.X)
    scrollbar_h.place(relx=0, rely=1, relwidth=1, anchor="sw")
    self.canvas.config(xscrollcommand=scrollbar_h.set)


def create_headers(self) -> None:
    """
    Create the column headers for the table.
    """
    # Draw the table headers
    for i, header in enumerate(self.headers):
        x1 = i * self.cell_width
        y1 = 0
        x2 = (i + 1) * self.cell_width
        y2 = self.cell_height
        self.canvas.create_rectangle(x1, y1, x2, y2, fill="gray", outline="black")
        self.canvas.create_text((x1 + x2) / 2, (y1 + y2) / 2, text=header)

    # Set required width and height of the table
    self.width = len(self.headers) * self.cell_width + 10  # +10 for scrollbar


def add_row(self, saved_row=None) -> None:
    """
    Add a row to the table and update the scroll region.
    """
    new_row = TableRow(master=self, saved_row_data=saved_row)
```

```python
        self.rows.append(new_row)

        self.update_scrollregion()


    def remove_rows(self) -> None:
        """
        Remove the selected rows from the table. If no rows are selected, an error
        Once the rows are removed, the remaining rows are re-drawn and the scroll
        """
        rows_to_remove = []
        for row in self.rows:
            if row.is_selected():
                row.destroy_entries()
                rows_to_remove.append(row.row_num)
        if not rows_to_remove:
            messagebox.showerror("Error", "Please select a row to remove")
            return

        rows_to_remove.sort(reverse=True)
        for row_num in rows_to_remove:
            self.rows.pop(row_num)

        remaining_rows = self.get_rows(ensure_filled=False, selected_only=False)

        # Destroy all rows
        for row in self.rows:
            row.destroy_entries()
        self.rows = []

        # Re-draw the rows
        for row in remaining_rows.values():
            self.add_row(row)

        self.update_scrollregion()


    def get_rows(
        self, ensure_filled=True, selected_only=False
    ) -> dict[str, Any] | None:
        """
```

```
    Get the data from the table rows and return it as a dictionary. If
    will be displayed to the user if any fields are empty. If selected_only is
    selected rows will be returned.
    :param ensure_filled: If True, ensure all fields are filled before returning
    :param selected_only: If True, only return the data from the selected rows
    """

    if selected_only:
        rows = []
        for row in self.rows:
            if row.is_selected():
                rows.append(row)
    else:
        rows = self.rows


    # Ensure all fields are filled
    if ensure_filled:
        for row in rows:
            if row.contains_empty_entry():
                messagebox.showerror("Error", "Please fill in all fields")
                return


    # Create a dictionary of the table data where the key is the bin
    # number and the value is a dictionary of the row data
    data = {}
    for row in rows:
        data.update(row.get_row_data())
    return data


def select_all(self, select_all: bool) -> None:
    """
    Select or deselect all rows in the table.
    :param select_all: If True, select all rows. If False, deselect all rows.
    """

    for row in self.rows:
        row: TableRow
        select_box: RowField = row.row_fields[0]
        assert isinstance(
            select_box.widget, CheckbuttonWithVar
        )  # helps type checker
```

```python
        if select_all:
            select_box.widget.select()
            select_box.widget.var.set(True)
        else:
            select_box.widget.deselect()
            select_box.widget.var.set(False)


    def update_scrollregion(self) -> None:
        """
        Update the scroll region of the canvas to fit the table.
        """
        region = self.canvas.bbox(tk.ALL)
        region = (region[0], region[1], region[2], region[3] + 20)
        self.canvas.config(scrollregion=region)
```

####################
/src/view/__init__.py
####################
```python
# Empty init file for view package. Causes Python to treat the directory as a package.
```

####################
/src/presenter/presenter.py
####################
```python
import json
import logging
import os
import threading
from pathlib import Path
from typing import Any
from typing import TYPE_CHECKING

import pandas as pd
from model.bin import Bin
from model.prepar import Preper
from model.runner import Runner

from config import BINNED_SHP_PATH
from config import COLUMNS_TO_DROP
from config import KML_PATH
```

```python
from config import POLYGON_SHP_PATH
from config import SAVED_STATE
from config import TIF_PATH


if TYPE_CHECKING:
    from view.gui_main import KmlCreatorGui


logger = logging.getLogger(__name__)




class Presenter:
    """

    Presenter class handles the communication between the view and the model. The view
    presenter by calling methods from the presenter. The presenter communicates with
    from the view, and vice versa. Likewise, the presenter communicates with the model
    model, and vice versa.
    When the Presenter is instantiated, it clears any temporary files that may have
    session. It also loads the GUI state from the saved state JSON file, if it exists.
    :param view: The view that the presenter is controlling.
    """


    view: "KmlCreatorGui"


    def __init__(self, view):
        self.view = view
        self.clear_temp_files()


    def clear_temp_files(self) -> None:
        """

        Clear any temporary files that may have been left over from a previous session.
        Uses a list of folders to iterate through and delete all files in each folder,
        except for ".gitkeep". .gitkeep is used to keep the folders in the repository,
        but is not needed for the program to run. Note that the SHP_PATH folder is not
        included in the list of folders to clean up, as this contains the shapefiles
        that the user might want to keep for later use.
        """
        # List of folders to clean up
        folders = [BINNED_SHP_PATH, POLYGON_SHP_PATH, TIF_PATH, KML_PATH]
```

```python
    for folder in folders:
        for file_path in folder.iterdir():
            if file_path.name != ".gitkeep":
                if file_path.is_file():  # Not a folder
                    file_path.unlink()
                    logger.info(f"Deleted file {file_path}")


def load_gui_state(self, csv_file_path: Path = None) -> None:
    """
    Load the GUI state from the saved state JSON file. If the saved state JSON file
    saved state JSON file does not contain a csv_file_path, then the GUI state is
    longer exists or if there is an error loading the saved state JSON file, then
    :param csv_file_path: The path to the CSV file to load the GUI state from. If
    is loaded from the saved state JSON file.
    """
    rows, settings, file_path = [], {}, None
    if csv_file_path:
        file_path = csv_file_path
    else:
        # Open the saved state file
        try:
            with open(str(SAVED_STATE), "r") as f:
                state = json.load(f)
        except FileNotFoundError:
            logger.exception("No saved state found")
            return
        try:
            saved_file_path = state.get("csv_file_path", None)
            if saved_file_path in [None, "", "."]:
                return
            # Check if csv file specified in saved state still exists
            if not os.path.exists(saved_file_path):
                return
            file_path = Path(saved_file_path)
            rows = state.get("rows", [])
            settings = state.get("settings", {})
        except FileNotFoundError:
            logger.exception("No saved state found")
        except Exception as e:
```

111

```python
            logger.exception(e)

    if file_path is None:
        return

    target_columns, column_min_max = self.get_target_columns_and_default_values(
        csv_file_path=file_path
    )
    # Target columns are mandatory so we need to check if they exist before loading
    # the state
    if target_columns in [None, []]:
        return

    # Load the GUI state
    self.view.set_target_columns(target_columns=target_columns)
    self.view.set_col_min_max_values(column_min_max=column_min_max)
    self.view.set_csv_file_path(file_path)
    self.view.set_saved_rows(rows=rows)
    self.view.set_settings(settings=settings)
    self.view.create_table_and_accessories()

def save_gui_state(self) -> None:
    """
    Save the GUI state to the saved state JSON file. If the saved state JSON file
    created. If the saved state JSON file does exist, then it is overwritten. The
    are used to get the GUI state (e.g. csv_file_path, rows, settings, etc.). If
    they are not saved.

    """
    state: dict[str, str | dict[str, str]] = dict()
    if csv_file_path := self.view.get_csv_file_path():
        state["csv_file_path"] = str(csv_file_path)
    if rows := self.view.get_rows(ensure_filled=False):
        state["rows"] = rows
    if settings := self.view.get_settings():
        state["settings"] = settings

    # Save the GUI state to the saved state JSON file
    try:
```

```python
            with open(str(SAVED_STATE), "w") as f:
                json.dump(state, f)
            logger.info("Saved state to JSON file")
        except FileNotFoundError:
            logger.exception("No saved state found")
        except Exception as e:
            logger.exception(e)


    def get_target_columns_and_default_values(
        self, csv_file_path: Path
    ) -> (list, dict):
        """
        Get the target columns and column min/max values from the CSV file. The target
        the user can select in the GUI's combobox. The column min/max values are the
        each column in the CSV file. These values are used for the lower and upper
        :param csv_file_path: The path to the CSV file to get the target columns and
        """
        # When loading from a saved state, the file may no longer exist
        if not os.path.exists(csv_file_path):
            raise FileNotFoundError
        try:
            with open(csv_file_path, "r") as f:
                # load csv file into pandas
                first_line = f.readline()
                if ";" in first_line:
                    csv_df = pd.read_csv(csv_file_path, index_col=0, sep=";")
                else:
                    csv_df = pd.read_csv(csv_file_path, index_col=0, sep=",")
        except Exception as e:
            logger.exception(e)
            raise e


        # Define columns to drop. They should only be dropped if they exist in the csv
        # file
        cols_to_drop = [col for col in COLUMNS_TO_DROP if col in csv_df.columns]


        csv_df = csv_df.drop(columns=cols_to_drop)
        column_names: list[str] = csv_df.columns.to_list()
        columns_min_and_max: dict[str, tuple[int, int]] = dict()
```

```python
        # Get min and max values for each column
        for column_name in column_names:
            columns_min_and_max[column_name] = (
                csv_df[column_name].min(),
                csv_df[column_name].max(),
            )
        return column_names, columns_min_and_max


        # Try to load the file


def initialize_and_run_process(self, data, save_directory) -> None:
    """
    Initialize and run the process in a separate thread. This is done so that the
    GUI does not freeze while the process is running.
    :param data: The data to pass to the process (row data, settings, etc.).
    :param save_directory: The directory to save the output kml files to.
    """
    t = threading.Thread(target=self.run_process, args=(data, save_directory))
    t.start()
    return


def print_to_view_console(self, text: str) -> None:
    """
    Print text to the view's console.
    """
    logger.info(text)
    self.view.print_to_console(text=text)


def run_process(self, data, save_directory) -> None:
    """
    Run the process. This is done in a separate thread so that the GUI does not
    running. Bin objects are created from the data and then a Preper and Runner are
    a shapefile for the Runner, which then creates further shapefiles (one for each
    interpolation, running polygonization and converting the polygon shapefiles to
    :param data: The data fetched from the GUI (row data, settings, etc.).
    :param save_directory: The directory to save the output kml files to.
    """
    bins_dict: dict[str, Any] = data["bins"]
    settings_dict: dict[str, Any] = data["settings"]
```

```python
bins: list[Bin] = []

# Create bins
for k, v in bins_dict.items():
    bins.append(
        Bin(
            enum=k,
            column=v["target_column"],
            description=v["description"],
            lower=v["lower_bound"],
            upper=v["upper_bound"],
            colour=v["colour"],
            opacity=v["opacity"],
            ignore=False,
            boundary_type="[]",
        )
    )


# Create Preper and Runner
preper = Preper(
    name=settings_dict["name"],
    save_directory=save_directory,
    csv_file_path=self.view.get_csv_file_path(),
    bins=bins,
    print_to_view=self.print_to_view_console,
)
runner = Runner(
    preper=preper,
    print_to_frontend=self.print_to_view_console,
    radius_width_metres=settings_dict["radius_width"],
    radius_height_metres=settings_dict["radius_height"],
    smoothing=settings_dict["smoothing"],
    pixel_size_metres=settings_dict["pixel_size"],
    angle=settings_dict["angle"],
    simplification_tolerance=settings_dict["simplification"],
)

self.print_to_view_console("\nCreating shapefile for each bin...")
preper.create_shp_for_each_bin()
```

```python
        self.print_to_view_console("\nRunning interpolation for each bin...")
        runner.run_interpolation_for_each_bin()
        self.print_to_view_console("...done.")


        self.print_to_view_console("\nRunning polygonization for each bin...")
        runner.run_polygonize_for_each_bin()
        self.print_to_view_console("...done.")


        self.print_to_view_console("\nCreating KML for each bin...")
        runner.create_kml_for_each_bin()
        self.print_to_view_console("...done.")


        self.print_to_view_console("\nDeleting temporary files...")
        runner.delete_files()
        self.print_to_view_console("...done.")


        self.view.submit.submit_finish()
        self.print_to_view_console("\nFinished.")
```

```
####################
/src/presenter/__init__.py
####################
# This file causes Python to treat the directory as a package.


####################
/src/model/bin.py
####################
import dataclasses



@dataclasses.dataclass
class Bin:
    """
    A class to represent a bin. A bin is a range of values that are grouped together.
    0 - 10, 10 - 20, 20 - 30, etc. The bin class is used to store the information about
    description, the range of values, the colour, etc. The attributes ending in
    Preper or the Runner during the processing of the data.
    :param enum: The bin number. This is used to identify the bin in the code.
```

```
    :param column: The column name that the bin is associated with.

    :param description: The description of the bin.

    :param lower: The lower bound of the bin.

    :param upper: The upper bound of the bin.

    :param bin_shp_file_name: The name of the bin shapefile.

    :param tif_file_name: The name of the tif file.

    :param polygon_shp_file_name: The name of the polygon shapefile.

    :param kml_file_name: The name of the kml file.

    :param colour: The colour of the bin.

    :param opacity: The opacity of the bin.

    :param ignore: Whether or not the bin should be ignored.

    :param boundary_type: The boundary type of the bin. This is used to determine

    are inclusive or exclusive. The first character is the lower bound and "[" means

    The second character is the upper bound and "]" means inclusive and "[" means

    that the lower bound is inclusive and the upper bound is exclusive.
    """

    enum: int

    column: str

    description: str

    lower: int

    upper: int

    bin_shp_file_name: str = None

    tif_file_name: str = None

    polygon_shp_file_name: str = None

    kml_file_name: str = None

    colour: str = "D10000"

    opacity: int = 100

    ignore: bool = False

    boundary_type: str = "[["


####################
/src/model/runner.py
####################
import logging

import os

from typing import Callable

from typing import TYPE_CHECKING
```

```python
import geopandas as gp
import geopy.distance
import shapely as sp
import simplekml
from osgeo import gdal
from osgeo_utils.gdal_polygonize import gdal_polygonize


if TYPE_CHECKING:
    from . import Bin
from .prepar import Preper
from config import (
    BINNED_SHP_PATH,
    INTERPOLATION_OUTPUT_FORMAT,
    INTERPOLATION_OUTPUT_TYPE,
    INTERPOLATION_ALGORITHM,
)
from config import KML_PATH
from config import POLYGON_SHP_PATH
from config import TIF_PATH


logger = logging.getLogger(__name__)



class Runner:
    """
    Creates individual shapefiles for each bin, interpolates them, polygonizes them,
    Files are saved in the temp_files directory and deleted when done. The run_all()
    methods in order.
    :param preper: Preper object that contains data pointing to the shapefile to be used
    :param radius_width_metres: Search ellipse width in metres
    :param radius_height_metres: Search ellipse height in metres
    :param pixel_size_metres: Pixel width/height in metres
    :param smoothing: Smoothing tolerance for the interpolation. Greater values result
    :param simplification_tolerance: for the polygonization. Greater values result in
    :param angle: Angle in degrees. Measured counter-clockwise from the positive
    :param print_to_frontend: Function to print to the frontend
    """

    preper: Preper
```

```python
    radius_x_metres: int
    radius_y_metres: int
    pixel_size_metres: int
    smoothing: int
    simplification_tolerance: int
    angle: int
    print_to_frontend: Callable[[str], None]


    def __init__(
        self,
        preper: Preper,
        radius_width_metres: int,
        radius_height_metres: int,
        pixel_size_metres: int,
        smoothing: int,
        simplification_tolerance: int = 10,
        angle: int = 0,
        print_to_frontend: Callable[[str], None] = None,
    ):
        # Get attributes from Preper
        self.name = preper.name
        self.bins = preper.bins
        self.save_directory = preper.save_directory
        self.geo_df = preper.geo_df
        self.dataset_height = preper.dataset_height
        self.dataset_width = preper.dataset_width
        self.bins = preper.bins
        self.preper = preper


        # User-defined attributes
        self.radius_x_metres = radius_width_metres
        self.radius_y_metres = radius_height_metres
        self.pixel_size_metres = pixel_size_metres
        self.smoothing = smoothing
        self.simplification_tolerance = simplification_tolerance
        self.angle = angle


        self._print_to_frontend = print_to_frontend
```

```python
def print_to_frontend(self, text: str) -> None:
    """
    Prints to the frontend if the print_to_frontend attribute is not None
    :param text: Text to print
    """
    logger.info(text)
    if self._print_to_frontend:
        self._print_to_frontend(text)


def run_all(self, delete_files_when_done=True) -> None:
    """
    Runs all the methods in order
    :param delete_files_when_done: Whether to delete the files when done
    """
    self.run_interpolation_for_each_bin()
    self.run_polygonize_for_each_bin()
    self.create_kml_for_each_bin()


    if delete_files_when_done:
        self.delete_files()


def run_interpolation_for_each_bin(
    self,
    radius_x_metres=None,
    radius_y_metres=None,
    pixel_size_metres=None,  # in metres
    smoothing=None,
) -> None:
    """
    Runs the interpolation for each bin. If bin.ignore is True, it is skipped. When
    search ellipse, the radius_x_metres and radius_y_metres are used. These are in
    to degrees using the centre of the dataset as the reference point. This is
    earth, which is not perfectly spherical.
    This method contains run_interpolation() which is the method that actually runs
    is called for each bin.
    """

    def run_interpolation(
        *,
```

```python
    input_shp_name: str,
    target_column: str,
    output_tif_name: str,
    output_format: str = INTERPOLATION_OUTPUT_FORMAT,
    output_type=INTERPOLATION_OUTPUT_TYPE,
    dataset_width: int = 0,
    dataset_height: int = 0,
    z_increase=None,
    z_multiply=None,
    output_bounds: list = None,
    algorithm: str = INTERPOLATION_ALGORITHM,
    power: int = None,
    smoothing: float = None,
    radius: float = None,
    radius_width: float = None,
    radius_height: float = None,
    angle: int = None,
    max_points: int = None,
    min_points: int = None,
    max_points_per_quadrant: int = 0,
    min_points_per_quadrant: int = 0,
    nodata: float = None,
    where: str = None,
    sql: str = None,
) -> str:
    """
    Runs the gdal.Grid function to interpolate a raster from a shapefile.
    :param input_shp_name: The name of the shapefile to be interpolated.
    :param target_column: The column in the shapefile to be interpolated.
    :param output_tif_name: The name of the output tif file.
    :param output_format: The format of the output file.
    :param output_type: The type of the output file. Default is Byte.
    :param dataset_width: The width of the output file.
    :param dataset_height: The height of the output file.
    :param z_increase: The amount to increase the z values by. Z values are the
    :param z_multiply: The amount to multiply the z values by. Z values are the
    :param output_bounds: The bounds of the output file.
    :param algorithm: The algorithm to use for interpolation. Options are
    :param power: The power to use for the inverse distance to a power
```

```
        :param smoothing: The smoothing to use for the average distance algorithm.
        :param radius: The radius to use for the average distance algorithm.
        :param radius_width: The radius1 to use for the average distance algorithm.
        :param radius_height: The radius2 to use for the average distance
        :param angle: The angle to use for the average distance algorithm. This is
        :param max_points: The max_points to use for the average distance algorithm.
        :param min_points: The min_points to use for the average distance algorithm.
        :param max_points_per_quadrant: The max_points_per_quadrant to use for the
        :param min_points_per_quadrant: The min_points_per_quadrant to use for the
        :param nodata: The nodata value to use for the average distance algorithm.
        :param where: The where clause to use for the average distance algorithm.
        :param sql: The sql to use for the average distance algorithm.
        """


    def _get_output_bounds() -> list:
        """
        Returns the output bounds for the gdal.Grid function. 18 decimal places
        nanometres. This is unneccessarily precise, but why not? Gdal.Grid will
        Gdal accepts the bounds in the order min_x, max_x, min_y, max_y.
        """
        min_x, min_y, max_x, max_y = output_bounds
        return [
            "%.18g" % min_x,
            "%.18g" % max_x,
            "%.18g" % min_y,
            "%.18g" % max_y,
        ]


    def _get_algorithm_str() -> str:
        """
        Returns the algorithm string for the gdal.Grid function.
        """
        s = f"{algorithm}:"
        s += f"power={power}:" if power else ""
        s += f"smoothing={smoothing}:" if smoothing else ""
        s += f"radius={radius}:" if radius else ""
        s += f"radius1={radius_width}:" if radius_width else ""
        s += f"radius2={radius_height}:" if radius_height else ""
        s += f"angle={angle}:" if angle else ""
```

```python
    s += f"max_points={max_points}:" if max_points else ""
    s += f"min_points={min_points}:" if min_points else ""
    s += (
        f"max_points_per_quadrant={max_points_per_quadrant}:"
        if max_points_per_quadrant
        else ""
    )
    s += (
        f"min_points_per_quadrant={min_points_per_quadrant}:"
        if min_points_per_quadrant
        else ""
    )
    s += f"nodata={nodata}:" if nodata else ""
    return s


new_options = []
if output_format is not None:
    new_options += ["-of", output_format]
if output_type is not None:
    new_options += ["-ot", output_type]
if dataset_width != 0 or dataset_height != 0:
    new_options += ["-outsize", str(dataset_width), str(dataset_height)]
if output_bounds is not None:
    new_options += ["-txe"] + _get_output_bounds()
if algorithm is not None:
    new_options += ["-a", _get_algorithm_str()]
if target_column is not None:
    new_options += ["-zfield", target_column]
if z_increase is not None:
    new_options += ["-z_increase", str(z_increase)]
if z_multiply is not None:
    new_options += ["-z_increase", str(z_multiply)]
if sql is not None:
    new_options += ["-sql", str(sql)]
if where is not None:
    new_options += ["-where", str(where)]


# Create the grid options object
grid_options = gdal.GridOptions(options=new_options)
```

```python
        dest_name = str(TIF_PATH / (output_tif_name + ".tif"))
        src_ds = str(BINNED_SHP_PATH / (input_shp_name + ".shp.zip"))


        logger.info(
            "Running interpolation on: {} \nOptions: {} \nSaving to: {}".format(
                str(src_ds), str(new_options), str(dest_name)
            )
        )


        # Run the interpolation
        gdal.Grid(destName=dest_name, srcDS=src_ds, options=grid_options)


        return output_tif_name


# If no values are passed, use the values from when Runner was initialized
radius_x_metres = radius_x_metres or self.radius_x_metres
radius_y_metres = radius_y_metres or self.radius_y_metres
pixel_size_metres = pixel_size_metres or self.pixel_size_metres
smoothing = smoothing or self.smoothing


# Convert the radius from metres to degrees using centre of the dataset as
# the reference point
min_x, min_y, max_x, max_y = self.geo_df.total_bounds
centre_of_geo_df = (
    ((max_x - min_x) / 2 + min_x),
    ((max_y - min_y) / 2 + min_y),
)  # (longitude, latitude)


# dest_coord_x is radius_x_metres metres eastwards. Point() object is
# (latitude, longitude)
distance_x = geopy.distance.distance(meters=radius_x_metres)
dest_coord_x: geopy.Point = distance_x.destination(
    (centre_of_geo_df[1], centre_of_geo_df[0]), bearing=90  # East
)
radius_x_degrees = dest_coord_x[1] - centre_of_geo_df[0]  # longitude
# dest_coord_y is radius_y_metres metres northwards. Point() object is
# (latitude, longitude)
distance_y = geopy.distance.distance(meters=radius_y_metres)
```

```python
    dest_coord_y: geopy.Point = distance_y.destination(
        (centre_of_geo_df[1], centre_of_geo_df[0]), bearing=0  # North
    )
    radius_y_degrees = dest_coord_y[0] - centre_of_geo_df[1]  # latitude


    # Run the interpolation for each bin
    for b in self.bins:
        if b.ignore:
            continue

        msg = "\nRunning interpolation for bin: " + str(b.enum)
        self.print_to_frontend(msg)

        tif_file_name = run_interpolation(
            input_shp_name=b.bin_shp_file_name,
            target_column=b.column,
            output_tif_name=b.bin_shp_file_name,
            algorithm="average",
            radius_width=radius_x_degrees,
            radius_height=radius_y_degrees,
            smoothing=smoothing,
            dataset_width=int(self.dataset_width / pixel_size_metres),
            dataset_height=int(self.dataset_height / pixel_size_metres),
            angle=int(self.angle),
        )
        b.tif_file_name = tif_file_name


def run_polygonize_for_each_bin(
    self,
    simplification_tolerance: int = None,
) -> None:
    """
    Runs the polygonization of each bin. If bin.ignore is True, it is skipped.
    """
    # Run polygonize for each bin
    for b in self.bins:
        if b.ignore:
            continue
```

125

```python
        self.print_to_frontend("\nRunning polygonize for bin: " + str(b.enum))

        output_shp_name = b.tif_file_name + "_polygons"

        input_tif_path = TIF_PATH / (b.tif_file_name + ".tif")

        output_shp_path = POLYGON_SHP_PATH / (output_shp_name + ".shp.zip")

        gdal_polygonize(
            src_filename=str(input_tif_path),
            dst_filename=str(output_shp_path),
            band_number=1,
            dst_layername=None,
            dst_fieldname=None,
            mask="default",
            connectedness8=False,
            options=None,
            quiet=True,
        )
        logger.info(
            "Polygonized " + str(input_tif_path) + " to " + str(output_shp_path)
        )

        b.polygon_shp_file_name = output_shp_name


def create_kml_for_each_bin(self) -> None:
    """
    Creates a kml for each bin. If the ignore of bin is True, it is skipped. The
    before the kml is created. The simplification tolerance is in metres and
    are simplified. This method contains the
    to create the kml from a shapely Polygon or MultiPolygon.
    """

    # Convert the simplification tolerance from metres to degrees
    x1, y1 = self.geo_df.total_bounds[0:2]  # lon, lat
    # Get the coordinate a certain distance away. Returns a tuple of (lat, lon)
    dest_coord = geopy.distance.distance(
        meters=self.simplification_tolerance,
    ).destination(
```

```
            (y1, x1), bearing=90  # 90 degrees is east
        )
        x2 = dest_coord[1]  # lon
        simplification_tolerance_degrees = x2 - x1


        # Create kml for each bin
        for bin in self.bins:
            if bin.ignore:
                continue
            self.print_to_frontend("\nMaking kml for bin: " + str(bin.enum))


            # Open the polygon shapefile using geopandas
            polygon_shp_file_path = POLYGON_SHP_PATH / (
                bin.polygon_shp_file_name + ".shp.zip"
            )
            polygon_df: gp.GeoDataFrame | None = None
            if not os.path.exists(polygon_shp_file_path):
                logger.exception("File does not exist: %s", polygon_shp_file_path)
                continue
            try:
                polygon_df = gp.read_file(
                    str(polygon_shp_file_path), driver="ESRI Shapefile"
                )
                logger.info(
                    f"Opened shapefile from {polygon_shp_file_path}. "
                    f"Shape (rows, columns): {self.geo_df.shape}"
                )
            except Exception as e:
                logger.error("Can't open shapefile. %s", e)
                self.print_to_frontend("A problem occurred. Please try again.")


            if polygon_df is None:
                self.print_to_frontend(
                    f"The polygon shapefile for bin {bin.enum} is empty. Skipping"
                )
                continue


            # Remove the polygons with zero values. These are not of value to the user.
            polygon_df = polygon_df[polygon_df["DN"] != 0]
```

```python
if polygon_df.empty:
    msg = "\nBin {} is empty. Ignoring...".format(bin.enum)
    logger.info(msg)
    self.print_to_frontend(msg)
    bin.ignore = True
    continue


# Unify and simplify the polygons
united: sp.Polygon | sp.MultiPolygon = sp.unary_union(
    polygon_df["geometry"]
)
united = united.simplify(tolerance=simplification_tolerance_degrees)


def _get_kml_colour(b: "Bin") -> str:
    """
    KML colour uses the format AABBGGRR, where AA is the alpha value, BB is
    value, and RR is the red value. The alpha value is the opacity of the
    transparent and FF is fully opaque. Values are specified in
    scaled from 0-100 to 0-255 and converted to hexadecimal before being
    """
    from simplekml import Color

    colour_without_hash = b.colour[1:]
    alpha = hex(int(b.opacity / 100 * 255))[
        2:
    ]  # first two characters are 0x
    return Color.hexa(colour_without_hash + alpha)


# Create the kml object and a MultiGeometry object
bin.kml_file_name = bin.polygon_shp_file_name
kml = simplekml.Kml()
multi_geometry: simplekml.MultiGeometry = kml.newmultigeometry(
    name=bin.description
)
# Polygon or MultiPolygon objects can be added to the MultiGeometry object
if isinstance(united, sp.Polygon):
    united = multi_geometry.newpolygon(
        name=bin.description,
        outerboundaryis=list(united.exterior.coords),
```

128

```python
                )
                united.style.polystyle.color = _get_kml_colour(bin)
                united.style.polystyle.outline = 0
            elif isinstance(united, sp.MultiPolygon):
                for polygon in united.geoms:
                    pol = multi_geometry.newpolygon(
                        name=bin.description,
                        outerboundaryis=list(polygon.exterior.coords),
                    )
                    pol.style.polystyle.color = _get_kml_colour(bin)
                    pol.style.polystyle.outline = 0
            else:
                raise TypeError("Polygon must be a shapely Polygon or MultiPolygon")


            # Save the kml file
            if save_directory := self.save_directory:
                file_path = str(save_directory / (bin.kml_file_name + ".kml"))
            else:
                file_path = str(KML_PATH / (bin.kml_file_name + ".kml"))


            try:
                kml.save(file_path)
            except Exception as e:
                logger.exception("Can't save kml file. %s", e)
                self.print_to_frontend("A problem occurred. Please try again.")
            logger.info("Saved kml file to " + file_path)


def delete_files(
    self,
    delete_binned_shp_files: bool = True,
    delete_polygon_shp_files: bool = True,
    delete_tif_files: bool = True,
    delete_kml_files: bool = False,
) -> None:
    """
    Deletes the files created by the Runner.
    :param delete_binned_shp_files: If True, deletes the binned shp files
    :param delete_polygon_shp_files: If True, deletes the polygon shp files
    :param delete_tif_files: If True, deletes the tif files
```

```python
        :param delete_kml_files: If True, deletes the kml files
        """
        # Remove the temporary files
        for bin in self.bins:
            if bin.ignore:
                continue

            self.print_to_frontend(
                "\nRemoving temporary files for bin: " + str(bin.enum)
            )

            if delete_binned_shp_files and hasattr(bin, "bin_shp_file_name"):
                binned_shp_file_path = (
                    str(BINNED_SHP_PATH / bin.bin_shp_file_name) + ".shp.zip"
                )
                self._delete_file(file_path=binned_shp_file_path)

            if delete_tif_files and hasattr(bin, "tif_file_name"):
                tif_file_path = str(TIF_PATH / bin.tif_file_name) + ".tif"
                self._delete_file(file_path=tif_file_path)

            if delete_polygon_shp_files and hasattr(bin, "polygon_shp_file_name"):
                polygon_shp_file_path = (
                    str(POLYGON_SHP_PATH / bin.polygon_shp_file_name) + ".shp.zip"
                )
                self._delete_file(file_path=polygon_shp_file_path)

            if delete_kml_files and hasattr(bin, "kml_file_name"):
                kml_file_path = str(KML_PATH / bin.kml_file_name) + ".kml"
                self._delete_file(file_path=kml_file_path)

    def _delete_file(self, file_path: str) -> None:
        """
        Deletes the file at the given file path.
        """
        if os.path.exists(file_path):
            os.remove(file_path)
            logger.info("Removed file: " + file_path)
        else:
```

```python
            logger.info("File not found: " + file_path)


    def __str__(self):
        return (
            f"Runner class for: {self.name}, using csv_file: "
            f"{self.preper.csv_file_name}"
        )
```

####################

/src/model/prepar.py

####################

```python
import logging
import os
from pathlib import Path
from typing import Callable


import geopandas as gp
import pandas as pd


from .bin import Bin
from config import BINNED_SHP_PATH
from config import COLUMNS_TO_DROP
from config import CSV_PATH
from config import ELLIPSOID
from config import KML_PATH
from config import PROJECTION
from config import SHAPEFILE_DRIVER
from config import SHP_PATH


logger = logging.getLogger(__name__)



class Preper:
    """

    Prepares the data for the Runner. This includes:

    - Creating a shapefile from the csv if one does not exist

    - Creating seperate shapefiles for each bin

    - Getting the width and height of the dataset in meters
    """
```

```python
    name: str
    csv_file_name: str
    csv_file_path: Path
    geo_df: gp.GeoDataFrame
    dataset_height: float
    dataset_width: float
    bins: list[Bin]
    _print_to_gui: Callable[[str], None]

    def __init__(
        self,
        name,
        bins: list[Bin],
        save_directory: Path = None,
        csv_file_name: str = None,
        csv_file_path: Path = None,
        print_to_view: Callable[[str], None] = None,
    ):
        assert (
            csv_file_name or csv_file_path
        ), "Must provide either a csv_file_name or a csv_file_path"
        self.name = name
        self.save_directory = save_directory or KML_PATH
        self._print_to_view = print_to_view
        self.csv_file_name = csv_file_name
        self.csv_file_path = csv_file_path
        self.geo_df = self._csv_to_shp()
        self._get_geo_df_dimensions()
        self.bins = bins

    def print_to_view(self, text: str) -> None:
        """
        Prints text to the view's console if self._print_to_gui is not None.
        """
        logger.info(text)
        if print_to_view := self._print_to_view:
            print_to_view(text)
```

```python
def _get_geo_df_dimensions(self) -> None:
    """
    Gets the width and height of the dataset in meters and saves them to
    self.dataset_width and self.dataset_height.
    """
    import geopy.distance

    min_lon, min_lat, max_lon, max_lat = (
        self.geo_df.total_bounds[0],
        self.geo_df.total_bounds[1],
        self.geo_df.total_bounds[2],
        self.geo_df.total_bounds[3],
    )

    lower_left = geopy.point.Point(longitude=min_lon, latitude=min_lat)
    upper_left = geopy.point.Point(longitude=min_lon, latitude=max_lat)
    lower_right = geopy.point.Point(longitude=max_lon, latitude=min_lat)
    self.dataset_width = geopy.distance.geodesic(
        lower_left, lower_right, ellipsoid=ELLIPSOID
    ).m
    self.dataset_height = geopy.distance.geodesic(
        lower_left, upper_left, ellipsoid=ELLIPSOID
    ).m

    self.print_to_view(
        f"\nDataset width: {round(self.dataset_width)} m, height: "
        f"{round(self.dataset_height)} m"
    )


def _csv_to_shp(self) -> gp.GeoDataFrame:
    """
    Creates a shapefile from a csv file with the name csv_file_name. If the
    it is loaded as a geopandas.GeoDataFrame and returned. If the shapefile does
    and returned. If an error occurs when trying to open an existing SHP file, a
    shapefile is done by converting the csv to a pandas.DataFrame, adding a
    objects, and converting the pandas.DataFrame to a geopandas.GeoDataFrame. The
    saved as a shapefile.
    """
    from shapely.geometry import Point
```

```python
        logger.info("Creating shapefile from csv")
        self.print_to_view("Creating shapefile from csv...")


        if self.csv_file_name:
            self.csv_file_path = CSV_PATH / (self.csv_file_name + ".csv")
        else:
            # Select filename from Path (e.g. "/path/to/file.csv" -> "file")
            self.csv_file_name = self.csv_file_path.stem


        # Give shp_file_name the same name as the csv_file_name
        self.shp_file_path = SHP_PATH / (self.csv_file_name + ".shp.zip")


        # Check if the shapefile already exists. If it does, load it and return it.
        if self.shp_file_path.exists():
            logger.info("Shapefile already exists. Opening...")
            self.print_to_view("Shapefile already exists. Opening...")
            try:
                geo_df = gp.read_file(str(self.shp_file_path))
                logger.info(f"Shapefile opened. Shape: {geo_df.shape}")
                return geo_df
            except Exception as e:
                logger.error(
                    "Could not open geopandas.GeoDataFrame from "
                    + str(self.shp_file_path)
                )
                logger.error(e)
                self.print_to_view("Problem opening shapefile. Creating new one...")


        # Create a new shapefile
        # Create a pandas.DataFrame from the csv
        try:
            input_df = pd.read_csv(self.csv_file_path, sep=",", index_col=0)
            logger.info(
                "Opened csv from %s. Shape: %s", self.csv_file_path, input_df.shape
            )
        except pd.errors.ParserError:
            # If the csv is not separated by commas, try semicolons
            input_df = pd.read_csv(self.csv_file_path, sep=";", index_col=0)
```

```python
            logger.info(
                "Opened csv from %s. Shape: %s", self.csv_file_path, input_df.shape
            )
        except Exception as e:
            logger.error("Could not open csv from %s. Error: %s", self.csv_file_path, e)
            raise e


        # Create a geometry column with shapely.Point objects
        input_df["geometry"] = input_df.apply(
            lambda x: Point((float(x.lon), float(x.lat))), axis=1
        )
        geo_df = gp.GeoDataFrame(input_df, geometry="geometry", crs="EPSG:4326")
        geo_df = geo_df.reset_index()
        cols_to_drop = [col for col in COLUMNS_TO_DROP if col in geo_df.columns]
        geo_df = geo_df.drop(columns=cols_to_drop)


        # Save the geopandas.GeoDataFrame as a shapefile
        try:
            geo_df.to_file(
                self.shp_file_path,
                driver=SHAPEFILE_DRIVER,
                projection=PROJECTION,
            )
            self.print_to_view("\nShapefile created.")
        except Exception as e:
            logger.error(
                "Could not save geopandas.GeoDataFrame to " + str(self.shp_file_path)
            )
            logger.error(e)
            self.print_to_view("An error occurred. See log for details.")
        return geo_df


    def create_shp_for_each_bin(self) -> None:
        """
        Creates a shapefile for each bin. If the ignore attribute of bin is True or it
        the shapefile already exists, it is skipped.
        """
        for b in self.bins:
            if b.ignore:
```

```python
            logger.info("Bin {} is ignored. Skipping.".format(b.enum))
            continue
        shp_file_name = "{}-{}-bin_{}".format(self.name, b.column, str(b.enum))
        shp_file_path = str(SHP_PATH / (shp_file_name + ".shp.zip"))

        # if the file exists, open it
        if os.path.exists(shp_file_path):
            msg = "\nShapefile for bin {} already exists. Skipping...".format(
                b.enum
            )
            self.print_to_view(msg)
            b.bin_shp_file_name = shp_file_name
            return

        self.print_to_view("\nCreating shapefile for bin: " + str(b.enum))

        # Create a new dataframe with only the rows that fall within the bin.lower
        # and bin.upper. Boundary type
        # determines whether the lower and upper bounds are inclusive or exclusive.
        temp_df: gp.GeoDataFrame | None = None
        if b.boundary_type == "[[":  # Inclusive lower, exclusive upper
            temp_df = self.geo_df[
                (self.geo_df[b.column] >= b.lower)
                & (self.geo_df[b.column] < b.upper)
            ]
        elif b.boundary_type == "[]":  # Inclusive lower, inclusive upper
            temp_df = self.geo_df[
                (self.geo_df[b.column] >= b.lower)
                & (self.geo_df[b.column] <= b.upper)
            ]
        else:
            raise NotImplementedError(
                "Boundary type not implemented: " + b.boundary_type
            )

        # If the dataframe there must be no data so skip this bin
        if temp_df.empty:
            self.print_to_view("\nBin {} is empty. Ignoring...".format(b.enum))
            b.ignore = True
```

```python
                continue

            binned_shp_file_path = BINNED_SHP_PATH / (shp_file_name + ".shp.zip")
            temp_df.to_file(
                filename=str(binned_shp_file_path),
                driver=SHAPEFILE_DRIVER,
                crs=PROJECTION,
            )
            logger.info(f"Saved geo_df to {shp_file_path}")
            b.bin_shp_file_name = shp_file_name


    def delete_shp_file(self) -> None:
        """
        Deletes the shapefile created by _csv_to_shp()
        """
        os.remove(self.shp_file_path)


    def __str__(self):
        return f"Preper for {self.name}, using csv: {self.csv_file_name}"
```

####################
/src/model/__init__.py
####################
```python
# This file causes Python to treat the directory as a package.
```

####################
/tests/test_backend.py
####################
```python
import filecmp
import logging
import os
import shutil
import sys
import unittest
from pathlib import Path

from src.model.bin import Bin
from src.model.prepar import Preper
```

```python
from src.model.runner import Runner
from tests.context.bins_for_testing import single_bin
from tests.context.bins_for_testing import test_bins


mussel_data_folder = Path(os.path.abspath(".")) / Path("src")
sys.path.insert(0, str(mussel_data_folder))


from config import CSV_PATH, BINNED_SHP_PATH, POLYGON_SHP_PATH, TEST_PATH
from config import KML_PATH
from config import TEST_CONTEXT_PATH
from config import TEST_CSV_DIMENSIONS
from config import TEST_CSV_NAME
from config import TEST_KML_NAME
from config import TEST_POLYGON_SHP_NAME
from config import TEST_SHP_NAME
from config import TEST_TIF_NAME
from config import TIF_PATH


logging.basicConfig(
    filename=str(TEST_PATH / "tests.log"),
    filemode="w",
    format="%(name)s - %(levelname)s - %(message)s",
    level=logging.DEBUG,
)
logger = logging.getLogger(__name__)
logger.info("Starting app")


class SetupTest(unittest.TestCase):
    """
    Setup and teardown for tests
    """

    def setUp(self):
        source_csv_path = str(TEST_CONTEXT_PATH / (TEST_CSV_NAME + ".csv"))
        dest_csv_path = str(CSV_PATH / (TEST_CSV_NAME + ".csv"))
        logger.info(f"Copying {source_csv_path} to {dest_csv_path}")
        shutil.copy(source_csv_path, dest_csv_path)
        self.copied_csv_path = dest_csv_path
```

```python
        self.assertTrue(os.path.exists(self.copied_csv_path))


    def tearDown(self):
        # Delete copied csv file
        os.remove(self.copied_csv_path)
        logger.info("Remove csv path: " + self.copied_csv_path)



class TestObjectInitialization(SetupTest):
    """
    Test that the models initialize correctly
    """


    def test_bin_class_initializes(self):
        """
        Test that the Bin class initializes
        """
        bin = Bin(
            enum=1,
            column="value",
            description="test description",
            lower=0,
            upper=666,
        )
        self.assertEqual(bin.enum, 1)
        self.assertEqual(bin.column, "value")
        self.assertEqual(bin.description, "test description")
        self.assertEqual(bin.lower, 0)
        self.assertEqual(bin.upper, 666)
        # Test default values
        self.assertEqual(bin.boundary_type, "[[")
        self.assertEqual(bin.colour, "D10000")
        self.assertEqual(bin.opacity, 100)
        self.assertEqual(bin.ignore, False)


    def test_preper_class_initialize(self):
        """
        Test that the Preper class initializes
        """
```

```python
        preper = Preper(
            name="test name",
            csv_file_path=CSV_PATH / str(TEST_CSV_NAME + ".csv"),
            bins=test_bins,
        )
        self.assertEqual(preper.name, "test name")
        self.assertEqual(preper.csv_file_name, TEST_CSV_NAME)
        self.assertEqual(preper.bins, test_bins)


    def test_runner_class_initializes(self):
        """
        Test that the Runner class initializes
        """
        preper = Preper(
            name="test name",
            csv_file_path=CSV_PATH / str(TEST_CSV_NAME + ".csv"),
            bins=test_bins,
        )
        runner = Runner(
            preper=preper,
            radius_width_metres=60,
            radius_height_metres=60,
            pixel_size_metres=10,
            smoothing=10,
        )
        self.assertEqual(runner.name, "test name")
        self.assertEqual(runner.preper.csv_file_name, TEST_CSV_NAME)
        self.assertEqual(runner.preper.bins, test_bins)
        self.assertEqual(runner.preper.name, "test name")
        self.assertEqual(runner.preper.geo_df.shape, TEST_CSV_DIMENSIONS)



class TestWholeProcess(SetupTest):
    preper: Preper
    runner: Runner
    name: str


    """
    Test the whole process, from csv to kml. Each Runner method is tested in turn.
```

```python
    """

    def setUp(self):
        """
        Set up the test by copying the test csv file to the csv folder
        and creating a Preper and Runner object
        """
        super().setUp()
        self.name = "test_name"
        self.preper = Preper(
            name=self.name,
            csv_file_path=CSV_PATH / str(TEST_CSV_NAME + ".csv"),
            bins=single_bin,
        )
        self.runner = Runner(
            preper=self.preper,
            radius_width_metres=60,
            radius_height_metres=60,
            pixel_size_metres=10,
            smoothing=10,
        )

    def tearDown(self):
        """
        Tear down the test by deleting the files created by the Preper and Runner object
        """
        self.preper.delete_shp_file()
        self.runner.delete_files(delete_kml_files=True)
        super().tearDown()

    def test_expected_shp_generated(self):
        """
        Test that create_shp_for_each_bin creates the expected binned shapefile
        """
        self.preper.create_shp_for_each_bin()
        created_shp_name = single_bin[0].bin_shp_file_name
        path_to_created_shp = str(BINNED_SHP_PATH / (created_shp_name + ".shp.zip"))
        path_to_expected_shp = str(TEST_CONTEXT_PATH / (TEST_SHP_NAME + ".shp.zip"))
        comparison = filecmp.cmp(
```

141

```
        path_to_created_shp, path_to_expected_shp, shallow=False
    )
    # Compare two files are roughly the same size
    size_diff = abs(
        os.path.getsize(path_to_created_shp) - os.path.getsize(path_to_expected_shp)
    )
    self.assertTrue(size_diff < 50)  # 50 bytes


def test_expected_tif_generated(self):
    """
    Test that run_interpolation_for_each_bin creates the expected tif file
    """
    self.preper.create_shp_for_each_bin()
    self.runner.run_interpolation_for_each_bin()
    created_tif_name = single_bin[0].tif_file_name
    path_to_created_tif = str(TIF_PATH / (created_tif_name + ".tif"))
    path_to_expected_tif = str(TEST_CONTEXT_PATH / (TEST_TIF_NAME + ".tif"))
    comparison = filecmp.cmp(
        path_to_created_tif, path_to_expected_tif, shallow=False
    )
    self.assertTrue(comparison)


def test_expected_polygon_shp_generated(self):
    """
    Test that run_polygonize_for_each_bin creates the expected polygon shapefile
    """
    self.preper.create_shp_for_each_bin()
    self.runner.run_interpolation_for_each_bin()
    self.runner.run_polygonize_for_each_bin()
    created_shp_name = single_bin[0].polygon_shp_file_name
    path_to_created_shp = str(POLYGON_SHP_PATH / (created_shp_name + ".shp.zip"))
    path_to_expected_shp = str(
        TEST_CONTEXT_PATH / (TEST_POLYGON_SHP_NAME + ".shp.zip")
    )

    # Compare two files are roughly the same size
    size_diff = abs(
        os.path.getsize(path_to_created_shp) - os.path.getsize(path_to_expected_shp)
    )
```

```python
        self.assertTrue(size_diff < 50)  # 50 bytes


    def test_expected_kml_generated(self):
        """
        Test that run_all creates the expected kml file
        """
        self.preper.create_shp_for_each_bin()
        self.runner.run_interpolation_for_each_bin()
        self.runner.run_polygonize_for_each_bin()
        self.runner.create_kml_for_each_bin()
        created_kml_name = single_bin[0].kml_file_name
        path_to_created_kml = str(KML_PATH / (created_kml_name + ".kml"))
        path_to_expected_kml = str(TEST_CONTEXT_PATH / (TEST_KML_NAME + ".kml"))

        # Compare two files are roughly the same size
        size_diff = abs(
            os.path.getsize(path_to_expected_kml) - os.path.getsize(path_to_created_kml)
        )
        self.assertTrue(size_diff < 50)  # 50 bytes


if __name__ == "__main__":
    unittest.main()


####################
/tests/context/bins_for_testing.py
####################
from src.model.bin import Bin


single_bin = [
    Bin(
        enum=1,
        column="value",
        description="Test description 1",
        lower=40,
        upper=60,
        colour="3B9C17",
        opacity=50,
    ),
```

```python
    ]
    test_bins = [
        Bin(
            enum=1,
            column="value",
            description="Test description 1",
            lower=0,
            upper=20,
            ignore=True,
        ),
        Bin(
            enum=2,
            column="value",
            description="Test description 2",
            lower=20,
            upper=40,
            colour="5CFF21",
            opacity=50,
        ),
        Bin(
            enum=3,
            column="value",
            description="Test description 3",
            lower=40,
            upper=60,
            colour="3B9C17",
            opacity=50,
        ),
        Bin(
            enum=4,
            column="value",
            description="Test description 4",
            lower=60,
            upper=80,
            colour="3B9C17",
            opacity=100,
        ),
        Bin(
            enum=3,
```

```
            column="value",
            description="Test description 3",
            lower=80,
            upper=100,
            colour="3B9C17",
            opacity=50,
            boundary_type="[]",
        ),
    ]


####################
/tests/context/__init__.py
####################
```

-3cm-3cm

8.8   Code: Mapping Client

```
####################
/public/index.html
####################
<!DOCTYPE html>  <!-- Defines the document type -->
  <head>
  <!-- Imports the Leaflet CSS stylesheet, which is used for interactive map functionalities -->
    <link
            rel="stylesheet"
            href="https://unpkg.com/leaflet@1.6.0/dist/leaflet.css"
            integrity="sha512-xwE/Az9zrjBIphAcBb3F6JVqxf46+CDLwfLMHloNu6KEQCAWi6H
            cDUbeOfBIptF7tcCzusKFjFw2yuvEpDL9wQ=="
            crossorigin=""
    />
    <title>Mapping Client</title> <!-- Specifies the title of the document -->
  </head>
  <body> <!-- Contains the content of the document -->
    <div id="root"></div> <!-- This is where our React application will be mounted -->
  </body>
</html>


####################
/src/Client.js
####################
import React from 'react';
// Import 'getUser' and 'resetUserSession' from AuthService.
// 'getUser' retrieves the user object from the session storage,
// while 'resetUserSession' removes the user and token data from the session storage.
import { getUser, resetUserSession } from './service/AuthService';
// Import 'LeafletMap' component which renders an interactive map using the Leaflet library.
import LeafletMap from './LeafletMap';


// The client component is used to render the mapping client itself once logged in
const Client = (props) => {
    // Get the user object from the session storage
    const user = getUser();


    // Extract the users name if the user object is not undefined
```

```
        const name = user !== "undefined" && user ? user.name : '';


        // Define a logoutHandler function that resets the user session and
        // redirects the user to the login page.
        const logoutHandler = () => {
            resetUserSession();
            // After resetting the session, navigate the user back to the login page
            props.history.push("/login");
        };


        return (
            // Render the greeting with the user's name, a logout button,
            // and the LeafletMap component,
            // which holds our interactive map and layercontrol for the KML files
            <div className="container">
                <div className="header">
                    <span className="welcome-text">Hello {name}! Welcome to the mapping client.
                    </span>
                    <br />
                    <br />
                    <span className="instruction-text">
                        Please choose which layers you want to see by hovering the layer control
                        in the top right corner of the map.
                    </span>
                    <input
                        type="button"
                        value="Logout"
                        onClick={logoutHandler}
                        className="logout-btn"
                    />
                </div>
                <LeafletMap />
            </div>
        );
};
// Export the Client component
export default Client;


####################
```

```
/src/index.css
####################
/* Import Leaflet CSS for map */
@import url('leaflet/dist/leaflet.css');
/* Import Roboto Slab font with 300 and 600 weight*/
@import url('https://fonts.googleapis.com/css2?family=Roboto+Slab:wght@300;600&display=swap');


/* Global body styles */
body {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: 'Roboto Slab', serif;
    font-weight: 300;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
    background-color: #05447A;
}


/* Header styles */
.header {
    margin: 1em 0 2em 0;
    padding: 1em 0;
    border-bottom: 1px solid;
    border-color: #FFFFFFFF;
}


/* Container for positioning elements */
.container {
    position: relative;
}


/* Welcome text styles */
.welcome-text {
    font-size: x-large;
    color: #FFFFFFFF;
    font-weight: 600;
}
```

```css
/* Instruction text styles */
.instruction-text {
    color: #FFFFFFFF;
    font-weight: 300;
}


/* Logout button styles */
.logout-btn {
    font-family: 'Roboto Slab', serif;
    position: absolute;
    top: 10px;
    right: 0;
    background-color: #0366D6;
    color: #FFFFFFFF;
    border: none;
    padding: 8px 20px;
    border-radius: 5px;
    cursor: pointer;
    font-size: 1em;
}


/* Logout button hover styles */
.logout-btn:hover {
    font-weight: 600;
}


/* Authentication container styles */
.auth-container {
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    background-color: #05447A;
}


/* Authentication form title styles */
.auth-container h5 {
    margin-top: 0.5em;
```

```css
    margin-bottom: 0.5em;
    font-size: 1.5em;
}


/* Authentication form styles */
.auth-container form {
    position: relative;
    max-width: 400px;
    width: 100%;
    padding: 15px;
    border-radius: 5px;
    background-color: #05447A;
    box-shadow: 0 0 10px #00000019;
    text-align: center;
}


/* Authentication form frame styles */
.auth-container form:after {
    content: "";
    display: block;
    position: absolute;
    top: -1px;
    left: -1px;
    right: -1px;
    bottom: -1px;
    z-index: 1;
    background-color: #80B6E3;
    border-radius: 25px;
}


/* Authentication form inner styles */
.auth-container .form-inner {
    position: relative;
    background-color: #FFFFFFFF;
    padding: 1px 1px 10px;
    z-index: 2;
    border-radius: 25px;
}
```

```css
/* Authentication form input and button styles */
.auth-container label,
.auth-container input[type="text"],
.auth-container input[type="password"],
.auth-container input[type="submit"],
.auth-container button {
    display: block;
    margin-bottom: 1em;
    margin-left: auto;
    margin-right: auto;
    font-family: 'Roboto Slab', serif;
    font-weight: 600;
    padding: 10px 20px;
}


/* Authentication form input and button hover styles */
.auth-container input[type="submit"]:hover,
.auth-container button:hover {
    cursor: pointer;
    opacity: 0.9;
}


/* Content styles */
.content {
    margin: 2em;
    font-size: 1.2em;
}


/* Message styles */
.message {
    color: #0000FF;
}



/* Error message styles */
.error-message {
    color: #FF0000FF;
}
```

```
####################
/src/index.js
####################
// Import the React library for building user interfaces
import React from "react";
// Import ReactDOM library for DOM manipulations in React
import ReactDOM from "react-dom";
// Import the CSS file for global styles
import "./index.css";
// Import the main application component
import App from "./App";


// Render the React application
// ReactDOM.render method is used to render React components to the DOM
// Here we're rendering the App component inside a div element
// with the id "root" in the HTML file
// React.StrictMode is a wrapper component that checks for potential problems in the application
// during the development build
ReactDOM.render(
    <React.StrictMode>
        <App />
    </React.StrictMode>,
    document.getElementById("root")
);


####################
/src/LeafletMap.css
####################
.leaflet-map {
    width: 100%;
    height: 800px;
    border-radius: 25px;
}


.leaflet-control-layers {
    width: 175px;
    height: 75px;
}
```

```css
.leaflet-control-layers-expanded {
    width: 175px;

    height: auto;

    max-height: 400px;
}


.leaflet-control-layers-list {
    width: 100%;

    height: 100%;
}
```

####################
/src/App.js
####################

```javascript
// This is the main component of the application.

// It is responsible for rendering the header and the content of the application

// Import the BrowserRouter, NavLink, Route and Switch components

// from the react-router-dom library

import {BrowserRouter, Redirect, Route, Switch} from "react-router-dom";

// Import the Home, Login, Register, and Client components

import Login from "./Login";

import Register from "./Register";

import Client from "./Client";

// Import the PublicRoute and PrivateRoute components for managing access to routes

import PublicRoute from "./route/PublicRoute";

import PrivateRoute from "./route/PrivateRoute";

// Import the useState and useEffect hooks from the React library

import { useState, useEffect} from "react";

// Import the getUser, getToken, setUserSession

// and resetUserSession functions from the AuthService module

import { getUser, getToken, setUserSession, resetUserSession } from "./service/AuthService";

// Import axios to make API requests

import axios from "axios";


// Define the verify token API endpoint URL

const verifyTokenURL = process.env.REACT_APP_API_GATEWAY_VERIFY;


// Create our main App component

function App() {
```

```javascript
// Create a state variable for authenticating the user
// and a function to update the state variable
const [isAuthenticating, setAuthenticating] = useState(true);
const [isLoggedIn, setIsLoggedIn] = useState(!!getToken());
// Use an effect hook to verify the user token on initial render
useEffect(() => {
    // Check if the token is undefined, null or empty,
    // if so return and do not make an API request
    const token = getToken();
    if (!token) {
        setAuthenticating(false);
        return;
    }
    // Create a requestConfig object that will be used to set the API key header
    const requestConfig = {
        headers: {
            // The API key is stored in the .env file
            "x-api-key": process.env.REACT_APP_API_KEY
        }
    }
    // Create a RequestBody object that will contain the user data and token
    // The token will be used to verify if the user is authenticated
    const RequestBody = {
        user: getUser(),
        token: token
    }
    // Make a POST request to the verifyTokenURL with the RequestBody
    // and requestConfig objects
    // response.data contains the new token and user data (if the token is valid)
    axios.post(verifyTokenURL, RequestBody, requestConfig).then(response => {
        // If the token is valid, set the user session and stop authenticating
        setUserSession(response.data.token, response.data.user);
        setIsLoggedIn(true);
        setAuthenticating(false);
    }).catch(()=> {
        // If the token is invalid, reset the user session and stop authenticating
        resetUserSession();
        setIsLoggedIn(false);
        setAuthenticating(false);
```

```
        })
    }, []);
    // Check if the user is authenticating and
    // if the token is valid, if so render the loading message
    const token = getToken();
    if (isAuthenticating && token) {
        return <div className={"content"}>Loading...</div>


    }
    // If the user is not authenticating anymore, render the application
    return (
        <div className={"App"}>
            <BrowserRouter>
                <div className={"content"}>
                    <Switch>
                        <Route exact path="/">
                            {isLoggedIn ? <Redirect to="/client" /> : <Redirect to="/login" />}
                        </Route>
                        <PublicRoute exact path="/login" component={props =>
                            <Login {...props} onLogin={() => setIsLoggedIn(true)} />} >
                        </PublicRoute>
                        <PublicRoute path="/register" component={Register} />
                        <PrivateRoute path="/client" component={Client} />
                    </Switch>
                </div>
            </BrowserRouter>
        </div>
    );
}
    // Export the App component
export default App;


####################
/src/Login.js
####################
// Import React and useState hook from the React library
import React, {useState} from "react"
// Import axios for making API requests
import axios from "axios"
```

```javascript
// Import setUserSession function from AuthService for managing user sessions
import { setUserSession } from "./service/AuthService"
// Import useHistory from 'react-router-dom' for programmatic navigation
import { useHistory } from "react-router-dom"


// Define the login API endpoint URL
export const loginUrl = process.env.REACT_APP_API_GATEWAY_LOGIN;


// Create a functional Login component with 'props' as its parameter
const Login = (props) => {
// Create a state variable for username and password with useState hooks
const [username, setUsername] = useState("");
const [password, setPassword] = useState("");
const [errorMessage, setErrorMessage] = useState(null);
const history = useHistory();


// Create a submitHandler function that will be called when the user submits the form
const submitHandler = (event) => {
    // Prevent the default behavior of the form
    event.preventDefault();
    // Check if username or password is empty, if so set an error message and return
    if (username.trim() === "" || password.trim() === "") {
        setErrorMessage("Please fill in all fields");
        return;
    }
    // If username and password are not empty, set the error message to null
    setErrorMessage(null);
    // Create a requestConfig object that will be used to set the API key header
    const requestConfig = {
        headers: {
            "x-api-key": process.env.REACT_APP_API_KEY // The API key is stored in the .env file
        }
    }
    // Create a RequestBody object that will be used to send the username and password to the API
    const RequestBody = {
        username: username,
        password: password,
    }
    // Make a POST request to the login API endpoint with the RequestBody
```

```
    // and requestConfig objects
    axios.post(loginUrl, RequestBody, requestConfig).then(response => {
        // If the request is successful, call the setUserSession function from AuthService
        // and pass the token and user data
        setUserSession(response.data.token, response.data.user);
        props.onLogin();
        // Redirect the user to the client page
        props.history.push("/client");
    }).catch(error => {
        // If the request fails, check if the error is a 401 or 403 error
        // and set the error message accordingly. If the error is not a 401 or 403 error,
        // set the error message to "Something went wrong. Please try again later."
        if (error.response.status === 401 || error.response.status === 403) {
            setErrorMessage(error.response.data.message);
        } else {
            setErrorMessage("Something went wrong. Please try again later.");
        }
    })
}

    // handleRegisterButtonClick function navigates to the Register page
    // and passes the current username value as state
    const handleRegisterButtonClick = () => {
        history.push({
            pathname: "/register",
            state: { username: username }
        });
    }


    // Return the login form with our state variables and submitHandler function
    return (
        <div className={"auth-container"}>
            <form onSubmit={submitHandler}>
                <div className={"form-inner"}>
                <h5>Login</h5>
                Username: <input type="text" value={username}
                onChange={event => setUsername(event.target.value)}/>
                Password: <input type="password" value={password}
                onChange={event => setPassword(event.target.value)}/>
                <input type="submit" value="Login" />
```

```
                <button onClick={handleRegisterButtonClick}>Register</button>
                {errorMessage && <p className="error-message">{errorMessage}</p>}
                </div>
            </form>
        </div>
    )
}
// Export the Login component
export default Login;


####################
/src/LeafletMap.js
####################
// Import necessary libraries and styles
import React, { Component } from 'react';
import L from 'leaflet';
import 'leaflet-kml';
import 'leaflet/dist/leaflet.css';
import './LeafletMap.css';
import axios from 'axios'; // Import Axios for making HTTP requests


// Define the LeafletMap component
class LeafletMap extends Component {
    // When the component is mounted
    componentDidMount() {
        // Initialize the map with OpenStreetMap as the base layer
        this.map = L.map('map', {
            layers: [
                L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
                    attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">
                    OpenStreetMap</a> contributors',
                }),
            ],
        });

        const baseMaps = {}; // Object to hold base layers
        const overlayMaps = {};


        // Function to load all KML files and add them to the map
```

```javascript
const loadKmlFiles = async () => {

    // Fetch the list of KML files from the API using Axios
    // The request is made to the API endpoint stored in the .env file
    // and passes an API key as a header
    const response = await axios.get
    ('${process.env.REACT_APP_API_GATEWAY_PRESIGNEDURL}', {
        headers: {
            // The API key is stored in the .env file
            'x-api-key': process.env.REACT_APP_API_KEY_2,
        }
    });
    const files = response.data;

    // For each file returned from the HTTP request, create a new layer group,
    // add it to the overlayMaps object,
    // and start loading the KML file into it
    // This creates an array of promises for each KML file that is being loaded
    const promises = files.map(({ file, url: signedUrl }) => {
        const layerGroup = L.layerGroup();
        overlayMaps[file] = layerGroup;
        return this.loadKML(signedUrl, layerGroup);
    });

    // Wait for all the KML files to be loaded before adding the layers to the map
    await Promise.all(promises);
    L.control.layers(baseMaps, overlayMaps).addTo(this.map);
    L.control.scale({ position: 'bottomleft' }).addTo(this.map);
};
// Call the loadKmlFiles function to start the process of loading the KML files
loadKmlFiles();
}


// When the component is about to be unmounted
componentWillUnmount() {
    // Remove the map from the DOM
    this.map.remove();
}
```

159

```
    // Fetch and parse KML file, then add it to the map as a layer
    loadKML = async (signedUrl, layerGroup) => {
        // Fetch the KML content using the signed URL with Axios
        // 'signedUrl' contains the presigned URLs that was returned by the HTTP request,
        // and it gives us access to the KML file in the S3 bucket
        const kmlResponse = await axios.get(signedUrl);
        const kmlText = kmlResponse.data;

        // Parse the KML content using a DOMParser, which turns the KML text into a DOM Document,
        // that we can read and manipulate
        const parser = new DOMParser();
        const kml = parser.parseFromString(kmlText, 'application/xml');
        // Use the 'leaflet-kml' library to convert the KML Document into a Leaflet layer
        const track = new L.KML(kml);

        // Add the KML layer to the map
        layerGroup.addLayer(track);
        this.map.addLayer(layerGroup);
        this.map.fitBounds(track.getBounds());
    };

    // Render the map container
    render() {
        return <div id="map" className="leaflet-map"></div>;
    }
}


// Export the LeafletMap component
export default LeafletMap;

####################
/src/Register.js
####################
// Import React and useState hook from the React library
import React, {useState, useEffect} from "react";
// Import axios for making API requests
import axios from "axios";
// Import useHistory from 'react-router-dom' for programmatic navigation
import { useHistory } from "react-router-dom";
```

```
// Import validation functions from the ValidationChecks module
import { passwordsMatch, isValidPassword, isValidUsername, isValidEmail, isValidName}
from "./service/ValidationService";



// Define the register API endpoint URL
export const registerUrl = process.env.REACT_APP_API_GATEWAY_REGISTER;


// Create a Register component
const Register = (props) => {
    // Create state variables for the form fields with useState hooks
    const [name, setName] = useState("");
    const [email, setEmail] = useState("");
    const [username, setUsername] = useState(props.location.state?.username || "");
    const [password, setPassword] = useState("");
    const [repeatPassword, setRepeatPassword] = useState("");
    const [showPassword, setShowPassword] = useState(false);
    const [showRepeatPassword, setShowRepeatPassword] = useState(false);


    const [message, setMessage] = useState(null);
    const [errorMessage, setErrorMessage] = useState(null);



    const history = useHistory();


    useEffect(() => {
        // If the username was passed from another component, update the username state
        if (props.location.state) {
            setUsername(props.location.state.username);
        }
    }, [props.location.state]);


    // Create a submitHandler function that will be called when the user submits the form
    // The function includes validations to ensure the entered credentials
    // are as per the required criteria
    const submitHandler = (event) => {
        // Prevent the default behavior of the form
        event.preventDefault();
        // Validation of the users entered name, email, username and password
```

```
if (!passwordsMatch(password, repeatPassword)) {
    setErrorMessage("Passwords do not match.");
    return;
}


if (!isValidName(name)) {
    setErrorMessage("Please enter a valid name with at least 2 characters.");
    return;
}


if (!isValidEmail(email)) {
    setErrorMessage("Please enter a valid email address.");
    return;
}


if (!isValidUsername(username)) {
    setErrorMessage("Please enter a valid username with at least 4 characters.");
    return;
}


if (!isValidPassword(password)) {
    setErrorMessage("Please enter a valid password with at least eight characters,
    one uppercase letter, one lowercase letter, one special characters and one number");
    return;
}


setErrorMessage(null);

// Create a requestConfig object that will be used to set the API key header
const requestConfig = {
    headers: {
        // The API key is stored in the .env file
        "x-api-key": process.env.REACT_APP_API_KEY
    }
}
// Create a RequestBody object that will be used to send the name,
// username, email and password to the API
const RequestBody = {
    name: name,
```

```
            username: username,
            email: email,
            password: password,
        }
        // Make a POST request to the register API endpoint with the RequestBody
        // and requestConfig objects
        axios.post(registerUrl, RequestBody, requestConfig).then(response => {
            // If the request is successful, set the message to "Successfully registered"
            setMessage("Successfully registered");
        }).catch(error => {
            // If the request fails, check if the error is a 401 or 403 error
            // and set the error message accordingly
            if (error.response.status === 401 || error.response.status === 403) {
                setErrorMessage(error.response.data.message);
            } else {
                setErrorMessage("Something went wrong. Please try again later.");
            }
        })
    }
    const handleReturnButtonClick = () => {
        history.push("/login");
    };
    // Toggle the visibility of the password in the password input field
    const toggleShowPassword = () => {
        setShowPassword(!showPassword);
    };
    // Toggle the visibility of the repeated password in the repeatPassword input field
    const toggleShowRepeatPassword = () => {
        setShowRepeatPassword(!showRepeatPassword);
    };
// Return the form
    return (
        <div className={"auth-container"}>
            <form onSubmit={submitHandler}>
                <div className={"form-inner"}>
                    <h5>Register</h5>
                    Name: <input type="text" value={name}
                    onChange={event => setName(event.target.value)}/>
                    E-mail: <input type="text" value={email}
```

```
                    onChange={event => setEmail(event.target.value)}/>
                    Username: <input type="text" value={username}
                    onChange={event => setUsername(event.target.value)}/>
                    Password:
                    <input
                        type={showPassword ? "text" : "password"}
                        value={password}
                        onChange={event => setPassword(event.target.value)}
                    />
                    <button type="button" onClick={toggleShowPassword}>
                    {showPassword ? "Hide" : "Show"}</button>
                    Repeat Password:
                    <input
                        type={showRepeatPassword ? "text" : "password"}
                        value={repeatPassword}
                        onChange={event => setRepeatPassword(event.target.value)}
                    />
                    <button type="button" onClick={toggleShowRepeatPassword}>
                    {showRepeatPassword ? "Hide" : "Show"}</button>
                    <input type="submit" value="Register" />
                    <button onClick={handleReturnButtonClick}>Return to login</button>
                    {message && <p className="message">{message}</p>}
                    {errorMessage && <p className="error-message">{errorMessage}</p>}
                </div>
            </form>
        </div>
    )
}
// Export the Register component
export default Register;
####################
/src/route/PublicRoute.js
####################
// Import React, Redirect and Route components from the react-router-dom library
import React from "react";
import { Redirect, Route } from "react-router-dom";

// Import getToken function from the AuthService module to check for the user's token
import { getToken } from "../service/AuthService";
```

```
// Define a PublicRoute component
// The PublicRoute component takes in a component and other properties (denoted by '...rest')
const PublicRoute = ({ component: Component, ...rest }) => {
    // Return a Route component
    return (
        <Route
            {...rest}
            // Render function decides what to render based on the user's token
            // If the user does not have a token (i.e., is not authenticated),
            // render the requested Component
            // If the user has a token (i.e., is authenticated),
            // redirect the user to the "/client" route
            render={props => {
                // Check if the user does not have a token (i.e., the user is not logged in)
                return !getToken() ? (
                    // If the user does not have a token, render the Component
                    // (i.e., the public route)
                    <Component {...props} />
                ) : (
                    // If the user has a token (i.e., the user is logged in),
                    // redirect the user to the "/client" route
                    <Redirect to={{ pathname: "/client" }} />
                );
            }}
        />
    );
};


// Export the PublicRoute component
export default PublicRoute;


####################
/src/route/PrivateRoute.js
####################
// Import React, Redirect and Route components from the react-router-dom library
import React from "react";
import { Redirect, Route } from "react-router-dom";
```

```
// Import getToken function from the AuthService module to check for the user's token
import { getToken } from "../service/AuthService";


// Define a PrivateRoute component
// The PrivateRoute component takes in a component and other properties (denoted by '...rest')
const PrivateRoute = ({ component: Component, ...rest }) => {
    // Return a Route component
    return (
        <Route
            {...rest}
            // Render function decides what to render based on the user's token
            // If the user has a token (i.e., is authenticated), render the requested Component
            // If the user does not have a token, redirect the user to the "/login" route
            render={props => {
                // Check if the user has a token
                return getToken() ? (
                    // If the user has a token, render the Component (i.e., the private route)
                    <Component {...props} />
                ) : (
                    // If the user does not have a token (i.e., the user is not logged in),
                    // redirect the user to the "/login" route
                    <Redirect to={{ pathname: "/login" }} />
                );
            }}
        />
    )
}


// Export the PrivateRoute component
export default PrivateRoute;


####################
/src/service/ValidationService.js
####################
// Function for checking if the passwords match
const passwordsMatch = (password, repeatPassword) => {
    return password === repeatPassword;
};
```

```javascript
// Function for checking if the name is valid based on a minimum length
const isValidName = (name) => {
    const minLength = 2;
    return name && name.length >= minLength;
};


// Function for checking if the email is valid based on a regular expression
const isValidEmail = (email) => {
    const regex = /^[a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
    return email && regex.test(email);
};


// Function for checking if the username is valid based on a minimum length
const isValidUsername = (username) => {
    const minLength = 4;
    return username && username.length >= minLength;
};


// Function for checking if the password is valid based on a minimum length
// and a regular expression
const isValidPassword = (password) => {
    const minLength = 8;
    const regex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&_-])[A-Za-z\d@$!%*?&_-]{8,}$/;
    return password && password.length >= minLength && regex.test(password);
};


// Export the functions
module.exports = {
    passwordsMatch,
    isValidName,
    isValidEmail,
    isValidUsername,
    isValidPassword
}


####################
/src/service/AuthService.js
####################
// This module contains functions for managing the user session.
```

```javascript
// The functions are used to get the user data and token from the session storage,
// set the user session and reset the user session.


// Import the sessionStorage object from the browser
module.exports = {
    // Retrieve the user object from sessionStorage
    // If the user object is empty, return null
    // If the user object is not empty, parse the JSON string and
    // return the user object as a JavaScript object
    getUser: function() {
        const user = sessionStorage.getItem("user");
        if (!user ) {
            return null;
        } else {
            return JSON.parse(user);
        }
    },
    // Retrieve the authentication token for the current user session from sessionStorage
    getToken: function() {
        return sessionStorage.getItem("token");
    },
    // Set the user session in sessionStorage by storing the authentication token
    // and user object
    // The user object is stored as a JSON string
    setUserSession: function(token, user) {
        sessionStorage.setItem("token", token);
        sessionStorage.setItem("user", JSON.stringify(user));
    },
    // Reset the user session by removing the authentication token
    // and user object from sessionStorage
    resetUserSession: function() {
        sessionStorage.removeItem("token");
        sessionStorage.removeItem("user");
    }
}



####################
/test/regsiterApi.test.js
```

```
####################
import axios from "axios";

import AxiosMockAdapter from "axios-mock-adapter";

import { expect } from "chai";

import { registerUrl } from "../src/Register";

import { loginUrl } from "../src/Login";


// Create a new instance of AxiosMockAdapter to intercept and mock axios requests
const mock = new AxiosMockAdapter(axios);


// Define a valid user object to be used in tests
const validUser = {
    name: "John Doe",
    email: "john@example.com",
    username: "johndoe",
    password: "Password123!",
};


// Define an invalid user object by spreading the valid user object
// and changing the email and password
const invalidUser = {
    ...validUser,
    email: "john@example",
    password: "Pass123",
};


// Main describe block for API tests
describe("API", () => {
    // Reset the mock adapter after each test to ensure no interference between tests
    afterEach(() => {
        mock.reset();
    });


    // Registration tests
    describe("Registration", () => {
        // Test for successful registration with valid data
        it("should successfully register with valid data", async () => {
            // Mock a successful registration response when the registerUrl
            // endpoint is called with validUser data
```

```javascript
        mock.onPost(registerUrl, validUser).reply(200);


        // Make the actual API call and store the response
        const response = await axios.post(registerUrl, validUser);


        // Check if the response status is 200 (success)
        expect(response.status).to.equal(200);
    });


    // Test for failed registration with invalid data
    it("should fail registration with invalid data", async () => {
        // Mock a failed registration response when the registerUrl
        // endpoint is called with invalidUser data
        mock.onPost(registerUrl, invalidUser).reply(400);


        // Try making the actual API call with invalid data and catch the error
        try {
            await axios.post(registerUrl, invalidUser);
        } catch (error) {
            // Check if the error response status is 400 (bad request)
            expect(error.response.status).to.equal(400);
        }
    });
});


// Login tests
describe("Login", () => {
    // Test for successful login with correct credentials
    it("should successfully login with correct credentials", async () => {
        // Mock a successful login response when the loginUrl endpoint
        // is called with validUser's username and password
        mock.onPost(loginUrl, { username: validUser.username,
        password: validUser.password }).reply(200);


        // Make the actual API call with correct credentials and store the response
        const response = await axios.post(loginUrl, { username: validUser.username,
        password: validUser.password });


        // Check if the response status is 200 (success)
```

```
                expect(response.status).to.equal(200);
        });


        // Test for failed login with incorrect credentials
        it("should fail login with incorrect credentials", async () => {
            // Mock a failed login response when the loginUrl endpoint
            // is called with validUser's username and wrong password
            mock.onPost(loginUrl, { username: validUser.username,
            password: "wrongpassword" }).reply(401);


            // Try making the actual API call with incorrect credentials and catch the error
            try {
                await axios.post(loginUrl, { username: validUser.username,
                password: "wrongpassword" });
            } catch (error) {
                // Check if the error response status is 401 (unauthorized)
                expect(error.response.status).to.equal(401);
            }
        });
    });
});


####################
/test/registerValidation.test.js
####################
import { expect } from "chai";
import {isValidName, isValidEmail, isValidUsername, isValidPassword, passwordsMatch}
from "../src/service/ValidationService";


// Tests for the validation functions used during registration
describe("Validation functions", () => {

    // Tests for the name validation function
    describe("Name validation", () => {

        // Check if the function returns true for valid names
        it("should return true for valid names", () => {
            expect(isValidName("John Doe")).to.be.true;
        });
```

```javascript
    // Check if the function returns false for invalid names
    it("should return false for invalid names", () => {
        expect(isValidName("J")).to.be.false;
    });
});


// Tests for the email validation function
describe("Email validation", () => {

    // Check if the function returns true for valid email addresses
    it("should return true for valid email addresses", () => {
        expect(isValidEmail("example@example.com")).to.be.true;
    });


    // Check if the function returns false for invalid email addresses
    it("should return false for invalid email addresses", () => {
        expect(isValidEmail("example@example")).to.be.false;
    });
});


// Tests for the username validation function
describe("Username validation", () => {

    // Check if the function returns true for valid usernames
    it("should return true for valid usernames", () => {
        expect(isValidUsername("username")).to.be.true;
    });


    // Check if the function returns false for invalid usernames
    it("should return false for invalid usernames", () => {
        expect(isValidUsername("usr")).to.be.false;
    });
});


// Tests for the password validation function
describe("Password validation", () => {

    // Check if the function returns true for valid passwords
```

```javascript
    it("should return true for valid passwords", () => {
        expect(isValidPassword("Password123!")).to.be.true;
    });


    // Check if the function returns false for various types of invalid passwords
    it("should return false for invalid passwords", () => {
        expect(isValidPassword("Pass123")).to.be.false;
        expect(isValidPassword("password")).to.be.false;
        expect(isValidPassword("PASSWORD")).to.be.false;
        expect(isValidPassword("Password")).to.be.false;
        expect(isValidPassword("Password123")).to.be.false;
        expect(isValidPassword("Password!")).to.be.false;
    });
});
// Tests for the password match validation function
describe("Password match validation", () => {

    // Check if the function returns true for matching passwords
    it("should return true for matching passwords", () => {
        expect(passwordsMatch("Password123!", "Password123!")).to.be.true;
    });


    // Check if the function returns false for non-matching passwords
    it("should return false for non-matching passwords", () => {
        expect(passwordsMatch("Password123!", "Password123")).to.be.false;
    });
});
});
```

-3cm-3cm

8.9 Code: Lambda Functions for Authentication

```
####################
/index.js
####################
// Import the required services and utility functions
const registerService = require('./service/register');
const loginService = require('./service/login');
const verifyService = require('./service/verify');
const util = require('./utils/util')

// Define the API paths
const healthPath = '/health';
const registerPath = '/register';
const loginPath = '/login';
const verifyPath = '/verify';

// Main Lambda function handler
exports.handler = async(event) => {
    // Log the incoming request event
    console.log('Request Event: ', event);

    let response;
    // Determine which service to call based on the HTTP method and path
    switch(true) {
        // Health check endpoint
        case event.httpMethod === 'GET' && event.path === healthPath:
            response = util.buildResponse(200);
            break;

        // User registration endpoint
        case event.httpMethod === 'POST' && event.path === registerPath:
            const registerBody = JSON.parse(event.body);
            response = await registerService.register(registerBody);
            break;

        // User login endpoint
        case event.httpMethod === 'POST' && event.path === loginPath:
            const loginBody = JSON.parse(event.body);
```

```
                response = loginService.login(loginBody);
                break;


            // Token verification endpoint
            case event.httpMethod === 'POST' && event.path === verifyPath:
                const verifyBody = JSON.parse(event.body);
                response = verifyService.verify(verifyBody);
                break;


            // Default case for unknown paths
            default:
                response = util.buildResponse(404, '404 Not found');
    }


    // Return the appropriate response
    return response
};


####################
/service/login.js
####################
// Import AWS SDK and configure the region
const AWS = require('aws-sdk');
AWS.config.update({
    region: 'eu-north-1'
});


// Import utility functions, bcrypt for password comparison, and auth for generating tokens
const util = require('../utils/util');
const bcrypt = require('bcryptjs');
const auth = require('../utils/auth');


// Import DocumentClient for DynamoDB
const { DocumentClient } = require('aws-sdk/clients/dynamodb');
const dynamodb = new DocumentClient();


// Define the DynamoDB table name
const userTable = 'mapclient-users';
```

```
// Main function to log in a user
async function login(user) {
    const username = user.username;
    const password = user.password;

    // Validate the input fields
    if (!user || !username || !password) {
        return util.buildResponse(401, {
            message: 'Username and password are required'
        });
    }

    // Get the user from the database using their username
    const dynamoUser = await getUser(username.toLowerCase().trim());
    if (!dynamoUser || !dynamoUser.username) {
        return util.buildResponse(403, { message: 'Username or password is wrong' });
    }

    // Compare the provided password with the stored hashed password
    if (!bcrypt.compareSync(password, dynamoUser.password)) {
        return util.buildResponse(403, { message: 'Username or password is wrong' });
    }

    // Create a userInfo object and generate a token for the authenticated user
    const userInfo = {
        username: dynamoUser.username,
        name: dynamoUser.name
    };
    const token = auth.generateToken(userInfo);

    // Construct a response object with the userInfo and token
    const response = {
        user: userInfo,
        token: token
    };

    // Return a success response with the userInfo and token
    return util.buildResponse(200, response);
}
```

```javascript
// Function to get a user by username from the database
async function getUser(username) {
    const params = {
        TableName: userTable,
        Key: {
            username: username
        }
    };

    // Query the database for the user and return the result
    return await dynamodb.get(params).promise().then(response => {
        return response.Item;
    }, error => {
        console.error('There is an error getting user: ', error);
    });
}

// Export the login function
module.exports.login = login;
```

####################
/service/register.js
####################
```javascript
// Import AWS SDK and configure the region
const AWS = require('aws-sdk');
AWS.config.update({
    region: 'eu-north-1'
});

// Import utility functions and bcrypt for password hashing
const util = require('../utils/util');
const bcrypt = require('bcryptjs');

// Import DocumentClient for DynamoDB
const { DocumentClient } = require('aws-sdk/clients/dynamodb');
const dynamodb = new DocumentClient();

// Define the DynamoDB table name
```

```
const userTable = 'mapclient-users';

// Main function to register a new user
async function register(userInfo) {
    // Extract user information from the input
    const name = userInfo.name;
    const email = userInfo.email;
    const username = userInfo.username;
    const password = userInfo.password;

    // Validate the input fields
    if (!username || !name || !email || !password ) {
        return util.buildResponse(401, {
            message: 'All fields are required'
        })
    }

    // Check if the username is already taken
    const dynamoUser = await getUser(username);
    if (dynamoUser && dynamoUser.username) {
        return util.buildResponse(401, {
            message: 'Username is already taken. Please choose a different one.'
        })
    }

    // Check if the email is already taken
    const dynamoEmail = await getEmail(email);
    if (dynamoEmail && dynamoEmail.email) {
        return util.buildResponse(401, {
            message: 'Email is already taken. Please choose a different one.'
        })
    }

    // Hash the user's password
    const hashedPW = bcrypt.hashSync(password.trim(), 10);

    // Create a user object with the input data
    const user = {
        name: name,
```

```
            email: email,

            username: username.toLowerCase().trim(),

            password: hashedPW

        }


        // Save the new user to the database

        const saveUserResponse = await saveUser(user);

        if (!saveUserResponse) {

            return util.buildResponse(503, {

                message: 'Server Error. Please try again later.'

            });

        }


        // Return success response with the registered username

        return util.buildResponse(200, {username: username});

}


// Function to get a user by username

async function getUser(username) {

    const params = {

        TableName: userTable,

        Key: {

            username: username

        }

    }


    // Query the database for the user and return the result

    return await dynamodb.get(params).promise().then(response => {

        return response.Item;

    }, error => {

        console.error('There is an error getting user: ', error);

    })

}


// Function to get a user by email

async function getEmail(email) {

    const params = {

        TableName: userTable,

        FilterExpression: "#email = :email",
```

```
            ExpressionAttributeNames: {
                "#email": "email"
            },
            ExpressionAttributeValues: {
                ":email": email
            }
        };


        // Scan the database for the user with the given email and return the result
        const queryOutput = await dynamodb.scan(params).promise();


        if (queryOutput.Items && queryOutput.Items.length > 0) {
            return queryOutput.Items[0];
        } else {
            return null;
        }
    }


    // Function to save a new user to the database
    async function saveUser(user) {
        const params = {
            TableName: userTable,
            Item: user
        }


        // Add the user to the database and return the result
        return await dynamodb.put(params).promise().then(() => {
            return true;
        }, error => {
            console.error('There is an error saving user: ', error)
        });
    }


    module.exports.register = register;


    ###################
    /service/verify.js
    ###################
    // Import utility functions and auth for verifying tokens
```

```javascript
const util = require('../utils/util');
const auth = require('../utils/auth');


// Main function to verify a user's token
function verify(requestBody) {
    // Validate the input fields
    if (!requestBody.user || !requestBody.user.username || !requestBody.token) {
        return util.buildResponse(401, {
            verified: false,
            message: "Incorrect request body"
        });
    }


    // Extract user and token from the request body
    const user = requestBody.user;
    const token = requestBody.token;


    // Verify the token using the auth.verifyToken function
    const verification = auth.verifyToken(user.username, token);


    // If the token is not verified, return an error response
    if (!verification.verified) {
        return util.buildResponse(401, verification);
    }


    // If the token is verified, return a success response with the user and token
    return util.buildResponse(200, {
        verified: true,
        message: "Verified",
        user: user,
        token: token
    });
}


// Export the verify function
module.exports.verify = verify;


###################
/utils/auth.js
```

181

```
####################
// Import JSON Web Token (JWT) library
const jwt = require('jsonwebtoken');


// Function to generate a JWT token for the user
function generateToken(userInfo) {
    // If userInfo is empty, return null
    if (!userInfo) {
        return null;
    }


    // Sign the userInfo with the JWT secret and set the expiration to 1 hour
    return jwt.sign(userInfo, process.env.JWT_SECRET, {
        expiresIn: "1h"
    });
}


// Function to verify a JWT token
function verifyToken(username, token) {
    // Verify the token with the JWT secret
    return jwt.verify(token, process.env.JWT_SECRET, (error, response) => {
        // If there's an error, return an invalid token response
        if (error) {
            return {
                verified: false,
                message: "Invalid token"
            };
        }
        // If the response's username does not match the given username,
        // return an invalid user response
        if (response.username !== username) {
            return {
                verified: false,
                message: "Invalid user"
            };
        }

        // If the token is valid, return a verified response
        return {
```

```
                verified: true,
                message: "Verified"
            };
        });
}


// Export the generateToken and verifyToken functions
module.exports.generateToken = generateToken;
module.exports.verifyToken = verifyToken;


####################
/utils/util.js
####################
// Function to build a response object for API Gateway with the given statusCode and body
function buildResponse(statusCode, body) {
    return {
        // Set the HTTP status code of the response
        statusCode: statusCode,

        // Set response headers for CORS and content type
        headers: {
            // Allow cross-origin resource sharing (CORS) with any domain
            'Access-Control-Allow-Origin': '*',

            // Set the content type of the response to JSON
            'Content-Type': 'application/json'
        },

        // Convert the response body to a JSON string
        body: JSON.stringify(body)
    }
}


// Export the buildResponse function
module.exports.buildResponse = buildResponse;
```

-3cm-3cm

8.10   Code: Lambda Function for generating presigned URLs

```
####################
/index.js
####################
// Import the AWS SDK and create an S3 client.
const AWS = require('aws-sdk');
const s3 = new AWS.S3({
    region: 'eu-north-1', // Our bucket's region
});


// The main Lambda handler function.
exports.handler = async (event) => {
    // Set parameters for listing the objects in the bucket.
    const params = {
        Bucket: 'kml-data-bucket', // Name of our bucket
    };


    try {
        // List all objects in the bucket.
        const data = await s3.listObjectsV2(params).promise();


        // Extract the filenames from the list of objects.
        const files = data.Contents.map((object) => object.Key);


        // Generate pre-signed URLs for each file.
        const urls = await Promise.all(files.map(async (file) => {
            const params = {
                Bucket: 'kml-data-bucket',
                Key: file,
                Expires: 10,
            };


            const url = await s3.getSignedUrlPromise('getObject', params);
            return { file, url };
        }));


        // Return the filenames and URLs in the response body.
        return {
```

```
            statusCode: 200,
            headers: {
                "Access-Control-Allow-Origin": "*",
                "Content-Type": "application/json"
            },
            body: JSON.stringify(urls),
        };
    } catch (err) {
        // Log any errors that occur during object listing.
        console.log(err);

        // Return a 500 error response if object listing fails.
        return {
            statusCode: 500,
            body: 'Failed to list objects in bucket',
        };
    }
};
```

-3cm-3cm