



**CS-A1121**

# **Y2: Project Reservation System**

**Hetijetti Jet Ski Rental**

Project By:

**Tatu Rouhiainen**

**100810721**

**Informaatioteknologia**

**12.05.2023**

---

tatu.rouhiainen@aalto.fi



# Table of Contents

<b>I. General Description</b>	<b>3</b>
<b>II. Instructions for the User</b>	<b>5</b>
<b>III. External Libraries</b>	<b>7</b>
<b>IV. Structure of the Program</b>	<b>9</b>
<b>V. Algorithms</b>	<b>14</b>
<b>VI. Data Structures</b>	<b>19</b>
<b>VII. Files</b>	<b>22</b>
<b>VIII. Testing</b>	<b>26</b>
<b>IX. The Known Shortcomings and Flaws in the Program</b>	<b>29</b>
<b>X. 3 Best and 3 Worst Areas</b>	<b>32</b>
<b>XI. Changes to the Original Plan</b>	<b>35</b>
<b>XII. Realised Order and Scheduled</b>	<b>37</b>
<b>XIII. Assesment of the Final Result</b>	<b>39</b>
<b>XIV. References</b>	<b>41</b>
<b>XV. Attachments</b>	<b>44</b>



# I

## General Description

---

Tatu Rouhiainen | [tatu.rouhiainen@aalto.fi](mailto:tatu.rouhiainen@aalto.fi)

# General Description

## Context

In the context of digitalization and the constant pursuit of providing the best user experience, this project entailed the creation of a robust and user-friendly jet ski reservation system for the company, Hetijetti. Hetijetti is a Lahti-based jet ski rental business that was initially established in the Helsinki metropolitan area in May 2022. After a promising start and subsequent expansion to the Lahti region, the business model was revised in 2023 to focus solely on operations in Lahti with two jet skis, with plans to acquire a third one.

The project's genesis was rooted in the observation of the initial reservation system's limitations, which, despite serving its purpose, lacked a certain professional finesse and was lacking in terms of customer experience. Thus, the primary objective was to devise a reservation system that encapsulated the essence of Hetijetti's business model while also being efficient, easy to navigate, and visually appealing.

## The Features of the Project

The primary features of this reservation system include a graphical user interface for seamless user experience, the ability to choose from three different jet skis (products), variable reservation lengths (1h, 2h, 4h, 8h, 24h), and prices commensurate to the reservation length. Moreover, the system also offers the possibility to order additional services such as life vests, wet suits, and snacks & beverages, enhancing the overall customer experience.

In addition to the reservation process, the system also includes an automatic confirmation email upon successful reservation. This provides users with clear communication and ensures that they are well-informed about their reservation details.

## The Difficulty of the Project

The project has been implemented at a high level of difficulty, which was intentionally chosen to create a system that is suitable for commercial use, and presents a challenge that encourages innovative solutions. The intricacies of designing a system that is simultaneously efficient, user-friendly, and capable of handling real-world complexities have made this a challenging yet rewarding endeavor.





## II

# Instructions for the User

# Instructions for the User

## Step-by-step instructions

The developed jet ski reservation system is designed to be as user-friendly and intuitive as possible, with the aim to provide a fast, pleasant, and smooth customer experience. The application primarily relies on user interaction through clicking, with the exception of the personal details section where user input is required.

- **Launch the Program:** Start the application running the *main.py* file. You will be greeted with an intuitive graphical user interface.
- **Select the Reservation Length:** The program offers five different reservation lengths. You can select the desired length by clicking on one of the available options.
- **Select a Date:** Pick the date for your reservation by clicking on a specific date from the displayed calendar.
- **Select a Jet Ski:** The program provides three different jet skis to choose from. Select the jet ski you wish to rent.
- **Select a Time Slot:** Based on the availability, choose a time slot for your reservation by clicking on one of the free time slots displayed on the screen.
- **Choose Additional Services (Optional):** If you want additional services like life vests, wet suits, or snacks & beverages, click on the respective buttons to add them to your reservation. If you wish to remove them from your reservation, you can click the same button to do so.
- **Enter Personal Details:** Fill in your personal details in the respective fields. Make sure to provide accurate information to ensure a smooth reservation process.
- **Confirm the Reservation:** Before confirming, view your total price and review your order details to ensure everything is as per your requirements. If all the required fields are filled correctly and you are satisfied with the details of your reservation, confirm the reservation by clicking on the confirmation button at the end.
- **Close the Program:** You can view the cancellation policy or safety guidelines on the last page, as well as your reservation number and instructions moving forward. The program is closed by clicking "READ MORE ON THE WEBSITE" button.





**III**

## External Libraries

---

Tatu Rouhiainen | [tatu.rouhiainen@aalto.fi](mailto:tatu.rouhiainen@aalto.fi)

# External Libraries

It is important to note that the use of these libraries was carefully considered to ensure that they facilitate, rather than hinder, the learning of programming basics. The emphasis throughout the project has been on developing programming skills and understanding, rather than relying excessively on libraries to provide pre-made solutions.

## PyQt6

This is the primary library used for building the user interface of the application. PyQt6 is a set of Python bindings for The Qt Company's Qt application framework, which provides a comprehensive set of GUI libraries and tools for building desktop applications. It was chosen for its ability to create a variety of GUI elements such as windows, buttons, and text boxes, and its support for event-driven programming, which allows user actions like button clicks to trigger specific functions.

## Google Sheets API

This external library is used to store and retrieve reservation data. It provides the application with the ability to interact with Google Sheets, storing reservation data and loading it when needed to verify the availability of jet skis. The use of Google Sheets API was approved during the project planning phase due to the practical advantages it offers. Storing data in Google Sheets provides easy access for managers and workers of Hetijetti and enhances data accessibility by leveraging cloud technology.

## Python Standard Libraries

- **csv:** This module is used for reading and writing data to CSV files, providing functionality for handling CSV-formatted data.
- **datetime:** The datetime module supplies classes for manipulating dates and times, and it is used in this project for handling and calculating dates for reservations.
- **socket:** This module provides access to the BSD socket interface. It is used for network-related tasks in the project.
- **smtplib and email.message:** These modules are used for sending emails from within the application. The smtplib module defines an SMTP client session object that can be used to send email to any machine running an SMTP server, while the email.message module is used for creating and manipulating email messages.







# **IV**

## **Structure of the Program**

---

Tatu Rouhiainen | [tatu.rouhiainen@aalto.fi](mailto:tatu.rouhiainen@aalto.fi)

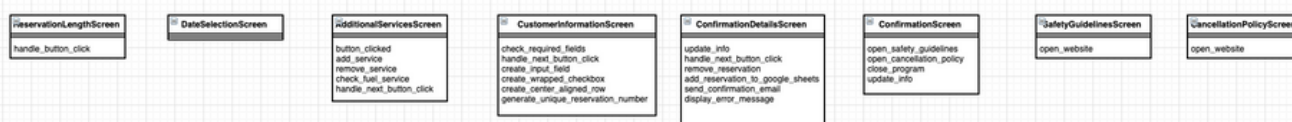
# Structure of the Program

## Overview

### Three Main Categories of Classes

The program can be conceptually divided into three main categories: screen-based classes, graphic-based classes and object-based classes. The screen-based classes model the different user interface screens encountered during the reservation process, while the object-based classes model the entities or objects such as the customer, jet ski, reservation data, and additional services. The graphic-based classes model different original graphical elements designed specifically for this project.

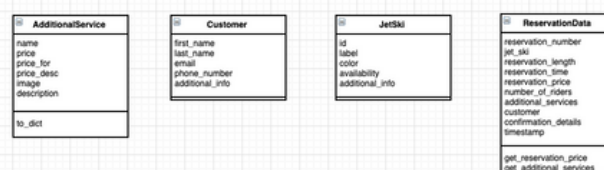
### Screen-Based Classes



### Graphic-Based Classes



### Object-Based Classes



# Structure of the Program

## Screen-Based Classes

### Screen-Based Classes

Screen-based classes define the user interface and user interactions within the system. They encapsulate the logic for each screen in the reservation process. These classes are essentially PyQt widgets, each representing a different window or screen in the application. This division into different screens makes the application's flow intuitive and user-friendly, guiding the customer through the reservation process in a structured manner.

Each screen-based class is responsible for a specific part of the reservation process. For instance, **CustomerInformationScreen** captures the user's personal information, **ConfirmationDetailsScreen** presents the final reservation details to the user, and **AdditionalServicesScreen** allows the user to add optional services to their reservation. The user navigates these screens in a sequence that mirrors the flow of making a reservation, from choosing a jet ski and reservation length, to providing personal details and confirming the reservation.

The **myCalendar** module, incorporated in the **DateSelectionScreen**, exemplifies the use of customized PyQt classes to create interactive calendar and time slot selection features. The module enhances the standard PyQt functionalities with additional user interaction capabilities, such as hover effects and clickable time slots.

- **ReservationLengthScreen:** This screen allows the user to select the length of their reservation.
- **DateSelectionScreen:** This screen, which imports classes from the **myCalendar** module, allows the user to select a jet ski, date, and time slot for their reservation.
- **AdditionalServicesScreen:** This screen allows the user to select additional services for their reservation.
- **CustomerInformationScreen:** This is the screen where the user inputs their information.
- **ConfirmationDetailsScreen:** This screen displays the total price and reservation details to the user.
- **ConfirmationScreen:** This is the final screen of the reservation process, showing whether the reservation was successful and providing the user an opportunity to view the cancellation policy or safety guidelines.
- **SafetyGuidelinesScreen:** This screen displays the safety guidelines to the user.
- **CancellationPolicyScreen:** This screen displays the cancellation policy to the user.



# Structure of the Program

## Object-Based Classes

### Object-Based Classes

The object-based classes, namely JetSki, Customer, ReservationData, and AdditionalService, model the real-world entities involved in the reservation process. These classes encapsulate the data and behaviors associated with the respective entities, contributing to the overall object-oriented design of the system.

For instance, the JetSki class models a jet ski and its associated properties, such as id, label, color, and availability. The Customer class represents a customer and contains personal details like name, email, and phone number. ReservationData is a comprehensive class that gathers all information related to a specific reservation, including the customer, the selected jet ski, the reservation length and time, the selected additional services, and the total price.

These classes interact with the screen-based classes to complete the reservation process. The data entered or selected by the user on different screens are used to create instances of these classes, which are then manipulated and stored as required.

- **JetSki:** This class models a jet ski, encapsulating its properties such as id, label, color, availability, and additional information.
- **Customer:** This class models a customer, capturing the customer's personal details such as first name, last name, email, phone number, and additional information.
- **ReservationData:** This class models a reservation, encapsulating all necessary data related to a reservation, such as reservation number, selected jet ski, reservation length, date, time, price, number of riders, additional services selected, customer details, confirmation details, and timestamp.
- **AdditionalService:** This class models an additional service offered during the reservation process, capturing properties such as name, price, price description, image, and description of the service.



# Structure of the Program

## Graphic-Based Classes

### Graphic-Based Classes

The graphic-based classes are responsible for creating and managing the visual elements of the reservation system. They enhance the user interface and contribute to the overall user experience. These classes include the `myCalendar`, `ProgressBar`, `BottomBar`, and the `MainWindow` class in the `main.py` file. The `general.py` module also plays a crucial role in setting up some of the system's visual aspects.

The graphic-based classes are responsible for creating and managing the visual elements of the reservation system. They enhance the user interface and contribute to the overall user experience. These classes include the **`myCalendar`**, **`ProgressBar`**, **`BottomBar`**, and the **`MainWindow`** class in the **`main.py`** file. The **`general.py`** module also plays a crucial role in setting up some of the system's visual aspects.

Classes include:

- **MainWindow:** the primary class that brings all these visual elements together. It sets up the main window of the application and adds all the different screen classes to a stacked widget. This stacked widget is then added to the main window, enabling the user to navigate through different screens as they progress through the reservation process.
- **ProgressBar:** creates a graphical element that indicates the user's progress through the reservation process. It serves as a visual guide, helping users understand how far they've progressed and how much is left to complete.
- **BottomBar:** complements the `ProgressBar` by providing navigational elements in the form of arrow buttons. These buttons allow the user to move to the next or previous screen, providing a seamless navigation experience.

A file that still plays a role in the structure of the program, but isn't a class itself, is the **`general.py`** module that is integral to the overall appearance of the application. It initializes the fonts used throughout the program and provides other general functions that are used across multiple parts of the program.

DISCLAIMER: **`general.py`** only provided more issues and is not in use in the final version of the reservation system. More on this in section IX.





# V Algorithms

---

Tatu Rouhiainen | [tatu.rouhiainen@aalto.fi](mailto:tatu.rouhiainen@aalto.fi)

# Algorithms

In this project, I developed multiple algorithms to ensure a seamless user experience throughout the reservation process. From calculating the total cost of a reservation to managing the calendar view and navigating through the reservation system, these algorithms are designed to streamline operations and improve customer satisfaction.

## The Calendar View

The Calendar View Algorithm is another key component of the reservation process and deserves special attention due to its complexity and critical role in the reservation process. I made the choice to build a custom calendar widget using PyQt6 instead of utilizing a ready-made calendar widget to provide a customised high-level, ensuring that the widget aligns with the overall theme of the reservation system and offers a superior customer experience. Developing this was almost a project in itself as it was by far the most time-consuming part of the project.

It displays a GUI showing each day of the week or month that the customer can rent the jet skis. Based on the selected date, the algorithm displays the available times for the chosen jet ski, taking into account the length of the reservation and existing bookings. The algorithm also considers several rules such as the earliest possible start time (10 am), the latest possible reservation end time (10 pm). The algorithm reads and uploads data into Google Sheets, using the Google Sheets API.

The complexity of this algorithm lies not only in the variety of functions and their interactions but also in the need to ensure a smooth and intuitive user experience. Each function is designed with a specific purpose in mind, and their collaborative work results in a comprehensive and user-friendly calendar view.

To save time, I could have chosen to use the calendar widget offered by the PyQt6, but I did not like at all how it looked or how it behaved. It wouldn't have fit the vibe of the reservation system and it felt cheap. Building a calendar widget from scratch was also a great chance for me to display my programming skills.



# Algorithms

The Calendar View algorithm operates as a complex system that organizes and presents key information for the user in a way that is intuitive and efficient. It allows navigation through different weeks in the calendar, showing future availability or previous selections. The calendar view updates automatically with any change in date, jet ski selection, or week navigation. It presents visual elements such as images of each reservable jet ski and the available time slots for the chosen reservation length.

The algorithm works in sync with Google Sheets, pulling current reservation data to determine the availability of each jet ski. It handles various user interactions such as clicking on a date, selecting a jet ski, or progressing to the next step in the reservation process. The active time slot is highlighted for the user's benefit when a specific slot is chosen. Any alterations in the data or selection results in the calendar view being reset and refreshed. This system ensures a smooth and user-friendly experience, making the reservation process easy and efficient.

## **Additional Services Algorithm**

The Additional Services Screen algorithm is primarily designed to manage the addition or removal of extra services to a reservation. When a service button is clicked, the algorithm checks whether the service has already been added to the reservation. If it has, the service is removed; if not, the service is added.

The addition of a service involves updating the visual cues on the button, such as text and color, to reflect that the service is now part of the reservation. This service is also added to a list of added services for future reference. Conversely, the removal of a service reverses these actions, returning the button to its original state and removing the service from the list of added services.

A special case is considered for a "Fuel Service". If the reservation length is more than 5 hours, this service becomes unavailable. The algorithm identifies the "Fuel Service" button and description label among other GUI elements, and updates them accordingly to reflect the unavailability of the service.

When the user decides to proceed to the next screen, the algorithm stores the list of added services into the reservation data for future use. This approach was chosen over others due to its ability to handle service selections in a flexible, user-friendly manner while seamlessly integrating with the existing reservation data structure. The customizability of this method allowed for the consideration of special cases and user feedback, enhancing the overall user experience.





# Algorithms

## Reservation Price

The Reservation Price Calculation Algorithm plays a crucial role in determining the total cost of a reservation. Unlike most systems that use a simple hourly rate, our algorithm calculates the total cost based on individually set prices for each reservation length, jet ski, and additional service. This approach incentivizes customers to book longer reservations by making them more cost-effective. The formula used is:

$$\text{TotalCost} = \text{JetSkiRentalPriceForTheSelectedLength} + \text{AdditionalServicesCost}$$

This algorithm is primarily used in the `ConfirmationDetailsScreen` class to display the final price to the customer before confirming the reservation.

## Checking the Necessary Fields of Customer Information

This algorithm ensures that all the necessary fields are filled before allowing the customer to proceed to the next step. It can be considered as a form validation algorithm which is quite essential in any application that involves user input.

The algorithm is mainly implemented in the **`check_required_fields`** function of the **`CustomerInformationScreen`**. Here's a brief overview of how this algorithm works:

- This function checks if all required fields are filled out and whether the terms and conditions checkbox is checked. It does this by evaluating the truthiness of the text inside each `QLineEdit` and the state of the checkbox. If a `QLineEdit` is empty, its `text()` method will return an empty string, which is considered `False` when evaluated in a boolean context. Similarly, the `isChecked()` method of a `QCheckBox` returns `True` if the checkbox is checked, and `False` otherwise.
- Based on the result of these checks, the function then enables or disables the 'Next' button. If all required fields are filled and the checkbox is checked, the button is enabled and its style is updated to reflect its enabled state. If any of these conditions are not met, the button is disabled and its style is updated to reflect its disabled state.

Furthermore, each input field and checkbox is connected to this function using signals and slots. This means that every time the text in an input field changes or the state of the checkbox changes, the `check_required_fields` function is called, ensuring that the state of the 'Next' button is always up-to-date.



# Algorithms

This form validation algorithm ensures that customers cannot proceed to the next step without providing all the necessary information and agreeing to the terms and conditions. This not only improves the user experience by providing immediate feedback when something is missing or incorrect, but also prevents potential issues down the line by ensuring that all necessary data is collected before proceeding.

## Unique Reservation Number Algorithm

The function `generate_unique_reservation_number()` in `CustomerInformationScreen` is an algorithm designed to generate unique reservation numbers for each booking made in my system. It begins by capturing the current Unix timestamp, which denotes the number of seconds that have elapsed since January 1, 1970. This timestamp is used as the base of the reservation number, as it always increases and is unique for each second. The algorithm then generates a random alphanumeric string of length 5 by selecting random characters from a pool of uppercase letters and digits. This random string is appended to the timestamp with a hyphen as a separator, forming the complete reservation number. This combination of timestamp and random string ensures the uniqueness of each reservation number, minimizing the chances of collision.

## Screen Navigation Algorithm

In the `MainWindow` class, I implemented an algorithm for seamless navigation between different screens in the reservation system. The `QStackedWidget` widget from `PyQt6` was employed as a central widget in the main window. Each screen of the reservation system, such as `ReservationLengthScreen`, `DateSelectionScreen`, `AdditionalServicesScreen`, etc., was added to this stacked widget as separate widgets.

The navigation between these screens is handled by an index-based system. Each screen is assigned a unique index when it's added to the stacked widget, starting from 0 and incrementing by 1 for each subsequent screen. When the user clicks a button to proceed to the next screen, the `setCurrentIndex(self.page_index + 1)` method is invoked. This method, present in each screen class, increments the current index by 1, effectively moving the view to the next screen in the sequence. The initial screen is set to the first in the stack (`self.stacked_widget.setCurrentIndex(0)`), and subsequent screens are displayed as the user navigates through the reservation process.





# VI

# Data Structures

---

Tatu Rouhiainen | [tatu.rouhiainen@aalto.fi](mailto:tatu.rouhiainen@aalto.fi)

# Data Structures

The reservation system for jet skis relies on a variety of data structures to store and manipulate data efficiently. Key among these are arrays, lists, and dictionaries, which have been chosen for their flexibility, ease of use, and ability to handle complex data types.

## Arrays and Lists

Arrays and lists are used extensively in this program for storing and manipulating similar types of data. For instance, the list of available time slots for a reservation, the list of added services, and the list of available jet skis are all handled using arrays and lists. The decision to use these data structures was made based on their flexibility and ease of use.

These data structures are mutable, meaning their contents can be altered after they have been created. This is a critical feature for our reservation system, as the availability of time slots and services will change constantly based on user interactions.

## Dictionaries

The Python dictionary is another data structure that has been put to use in this reservation system. Dictionaries are used to store data in key-value pairs. This is especially useful for storing complex data, like the details of a reservation, where each piece of data can be accessed using a unique key. For example, the reservation length objects are created with a for loop iterating through a dictionary with reservation lengths mapped to their respective prices.

## Other Possibilities

Other data structures that could have been used in this application for different uses include lists, tuples, or sets. For instance, lists could have been used to store reservation lengths and prices, with corresponding indices used to link the two. However, this approach would be less efficient and more error-prone, as maintaining the connection between two separate lists can be cumbersome and confusing.



# Data Structures

Tuples, being an immutable data structure, could have served as an alternative to dictionaries for storing reservation lengths and prices if these values were constant and not expected to change. However, the need for flexibility and the ability to update the reservation lengths and prices at runtime led to the decision to use dictionaries, which are mutable. Tuples could have been used to store the start and end times of a reservation, as well as the date selected by the customer since tuples are immutable, which means that their size and values can't be changed. They could have ensured that the start and end times of a reservation are not accidentally modified.

Sets, another data structure, could have been considered if the application needed to store a collection of unique elements, but they were not required for this project as there was no need to enforce uniqueness in the data being handled.

Python's predefined data structures were primarily used in this application, with lists being the most prevalent due to their easiness and user-friendliness in handling structured data. There was no need to create a custom data structure, as the built-in Python data structures provided all the necessary functionality for the reservation system. This choice simplified the implementation and enhanced the maintainability and readability of the code.





# VII

## Files

# Files

## CSV --> Google Sheets

The reservation system was initially designed to interact with Comma Separated Values (CSV) files. CSV is a simple file format used to store tabular data, such as a spreadsheet or database. These are essentially text files where each line represents a row in the table and the individual values in that row are separated by commas. This format makes it easy to read and write data from a program and is also human-readable, which is beneficial for debugging and manual data entry or analysis.

At the start of the project, the program was designed to read reservation data from a CSV file and to write new reservation data back to a CSV file. This design made sense as it allowed us to keep a persistent record of reservations across multiple runs of the program. It also provided a simple way to view and modify the reservation data outside of the program if necessary. Here is an example of how the data was structured in the CSV file:

```
date,time_slot,Sea-Doo Spark,Sea-Doo GTI 130,Sea-Doo GTI 170
01/05/2023,10:00-11:00,,,
01/05/2023,11:00-12:00,,,
01/05/2023,12:00-13:00,,,
01/05/2023,13:00-14:00,,,
```

Empty cells in each column under the jet skis represented available time slots. If the time slot was reserved the reservation number would have been saved on to the time slot.

However, as the project progressed, it became clear that a more scalable and collaborative solution was required. This led to the transition to using Google Sheets for the storage and management of reservation data. Google Sheets provides similar tabular data storage as a CSV file but with added benefits of being accessible from anywhere, allowing for multiple simultaneous users, and having built-in functions for data manipulation and analysis.

The transition was relatively straightforward as Google Sheets can also be interacted with as a CSV file. The Google Sheets API allows for reading data from a sheet in a similar way to reading lines from a CSV file and writing data to a sheet is analogous to appending lines to a CSV file. This meant that much of the code for handling CSV files could be reused with minor modifications to interact with the Google Sheets API instead.



# Files

The data in the Google Sheet is structured in the same way as it was in the CSV file, making it easy for anyone familiar with the original system to understand and work with the new system. The first row of the sheet contains the headers and each subsequent row represents a reservation.

While the switch to Google Sheets required an internet connection and additional setup to authenticate with the Google Sheets API, the benefits of increased scalability, accessibility, and collaboration made this a worthwhile trade-off.

## **PNG & JPG Files**

Our reservation system utilizes PNG (Portable Network Graphics) and JPEG (Joint Photographic Experts Group) files to manage and display images. The choice between the two file types depends on the purpose of the image. PNG files support transparency, making them ideal for certain graphic elements such as icons, transparent figures of jet skis, or the brand's logo. This ability to handle transparency allows PNG images to blend seamlessly with different backgrounds or overlays in the user interface.

On the other hand, JPEG images are typically smaller in file size compared to PNG images. This is due to the lossy compression algorithm used by JPEG, which reduces file size at the cost of some image quality. This efficiency in storage makes JPEG suitable for larger, complex images such as backgrounds where transparency isn't required. The smaller the file size, the faster it loads, which contributes to the overall performance and responsiveness of the application.

## **TrueType Font (.ttf) Files**

To enhance the aesthetics of the user interface, the reservation system incorporates custom fonts, specifically Montserrat-Light and Montserrat-Bold. These fonts are stored as TrueType Font (.ttf) files. TrueType is a scalable font technology that ensures the font maintains high quality when resized. The use of Montserrat-Light and Montserrat-Bold provides a clean, modern look that improves readability and the overall user experience.





# Files

## HTML

In order to provide a more personalized and visually appealing confirmation email to the customers, the system sends HTML-based emails. HTML, or Hypertext Markup Language, is the standard markup language for documents designed to be displayed in a web browser. In our case, the HTML file allows us to customize the layout, colors, and font styles of the confirmation email, making it more engaging and professional than a plain text email. This allows us to incorporate elements such as headers, tables, images, and links directly into the email, providing the user with all the necessary reservation details in a structured and attractive format.





# VIII

# Testing

---

Tatu Rouhiainen | [tatu.rouhiainen@aalto.fi](mailto:tatu.rouhiainen@aalto.fi)

# Testing

## Testing Plan

The program passes all the tests laid out in the project plan. Each of these tests was designed to examine the key functionalities of the program, ensuring that it behaves as expected under different scenarios. When a new reservation is added, the system correctly checks and confirms the availability of the requested jet ski. Additionally, it accurately calculates the total price of a reservation based on the number of jet skis rented and the rental duration.

During the development process, the program was consistently tested to identify and rectify any issues or bugs. This was accomplished through the use of unit tests. Unit tests were written to verify the functionality of individual components or methods in isolation.

One of the main areas where unit tests were utilized was in the creation and management of customer objects. By creating mock customer data and running it through the relevant functions, it was possible to verify that the system handles customer information correctly.

Despite the thorough testing, there were a few gaps in the test planning. For example, the testing process did not initially account for the handling of edge cases in the reservation process, such as bookings that span the closing and opening times of the rental service. Also, the system's response to unexpected user input or system behavior was not thoroughly tested in the initial plan. These gaps were identified and addressed during the development process, further strengthening the reliability of the program.

## Add Reservation

The "Add Reservation" functionality was tested extensively due to its pivotal role in the system. Test cases included valid reservations with all required fields, reservations with missing or incorrect data, and reservations conflicting with existing ones. The program successfully handled all these scenarios, adding valid reservations, and generating appropriate error messages for invalid or conflicting ones. Unit testing after adding the Google Sheets API turned out to be an issue, which is why the unit tests regarding adding the reservation had to be removed, but the function was thoroughly tested otherwise.



# Testing

## Get Available Start Times

The "Get Available Start Times" function was tested for different scenarios, including valid dates and lengths with available start times, valid dates and lengths without available start times, and invalid dates or lengths. The system correctly returned available start times or error messages as expected based on the input conditions.

## Creating A Customer Object

Creating a Customer object was tested for scenarios like an empty dictionary of customer data, a dictionary with all required information, and a dictionary missing some required information or containing invalid information. The system correctly created new customer objects for valid data and returned error messages for invalid or incomplete data.

## Creating A Reservation Data Object

Creating a reservation data object was tested for initialisation, creating and unique reservation number and setting reservation data. The system correctly created new reservation data objects for valid data and returned unique reservation numbers.

## Continuous Testing

During the creation process, the program was constantly tested to ensure it behaves as expected. This included unit tests for individual components and integration tests for the program as a whole. It allowed for early identification and correction of potential issues.





## **IX**

# **The Known Shortcomings and Flaws in the Program**

# Shortcomings and Flaws

## Technical Issues

- **Font Inconsistency:** While the custom fonts used in the application enhance its aesthetic appeal, they occasionally fail to load properly. Often times I have changed nothing, but the fonts will just simply not load even though the path has been verified as correct, and the program recognising the correct path. This may be due to a variety of reasons, including local machine issues, caching problems, or issues with the way the font files are being accessed. For now, there is no fix.
- **Unreliable mail server:** The mail server currently used for sending confirmation emails is not always responsive. This could be due to server overload, connectivity issues, or problems with the server itself. Exploring other mail server options or using a dedicated email service provider could help resolve this issue. I have implemented an error handling to notify the user when an email fails to send, but it will not stop the system from confirming the reservation.

## Shortcoming

- **Multiple Jet Ski Reservations:** The program currently does not support the reservation of multiple jet skis at once. This could be improved by modifying the date selection screen to allow for adding another jet ski after selecting the first one. This could have led to impulse buying. This doesn't seem like a big change as the rest of the program would work the same even though we added two jet skis to the reservation data instance.
- **Weekend Reservations:** The program does not support weekend reservations due to the complexity of the existing calendar system even though I originally had planned so. The calendar system was made for hourly reservations, and I couldn't think of a realistic way to offer the 72h reservation. In real life, you would sell a weekend reservation even if there was a few 1h reservations because you could reschedule the customers or move them to another jet ski, but the system would recognise the weekends as reserved. In real life it would be better to sell the weekend reservations separately too, with direct contact between the customer and the sales. I could only imagine making a whole separate calendar for the weekend reservation logic, and with how much time the hourly-based system took, I couldn't even imagine it.



# Shortcomings and Flaws

- **Limited Calendar View:** Limited calendar view: The current calendar widget only allows viewing of 7 days at a time. Even though I had planned a monthly view at the start, it went soon out of the window after how much the weekly view took time. While a monthly view could provide a broader overview, the current weekly view works well as jet skis are rarely reserved much in advance.
- **Insufficient Error Feedback:** The current customer detail screen does not provide specific error messages when a field is missing or incorrect. Enhancing this feature to provide detailed error feedback would improve the user experience and make the program more user-friendly.
- **More Customer Information:** The program currently does not collect address information from customers. While this was a design decision to simplify the reservation process, it could potentially lead to difficulties in the event of a no-show. Collecting a billing address is important, as in the event of a no-show or a late cancellation the customer would have to be billed. The decision to leave it out was made to save time, and adding more text boxes wouldn't really improve the difficulty of the project. The screen already proves I can handle text boxes, user input and the data collected from them.
- **Confirmation Email:** The current confirmation email does not include a detailed breakdown of the total price and has a couple alignment issues. Improving the design of the email to include this information would provide a more comprehensive overview for the customer. However, I thought HTML coding and design would be irrelevant for this project, thus I didn't spend too much time on it.
- **Timeslot Confirm Button:** The height of the confirm button on the date selection screen changes based on the button text. Ensuring UI consistency by fixing the height of the button regardless of the text would improve the aesthetic and usability of the program.





**X**

## **3 Best & 3 Worst Areas**

---

Tatu Rouhiainen | [tatu.rouhiainen@aalto.fi](mailto:tatu.rouhiainen@aalto.fi)



# The Best Parts

## Original Calendar Widget

The creation of a custom calendar widget has proven to be one of the significant strengths of this program. Not only does it provide a seamless and intuitive user interface, but it is also tailored specifically to meet the needs of a jet ski reservation system. This customization enhances the user experience and contributes to the overall functionality of the program.

## User Interface

User Interface: The UI design is another strong area of the program. Its clean, user-friendly design aligns perfectly with the existing Hetijetti brand and the website (that also was designed and developed by me, so the branding is unique and not copied), providing a coherent and consistent user experience. The interface is easy to navigate, visually appealing, and seamlessly integrates various components of the program.

## Realism

The integration of the Google Sheets API adds a level of realism to the program that sets it apart. This feature considers the management side of the operations, allowing for real-time updates and data management. Additionally, the confirmation email feature further enhances the user experience by providing users with immediate confirmation of their reservation details. The whole project has been made as realistic as possible, nothing has been made up. These jet skis exist in real life, this brand exists in real life - hopefully one day this reservation system exists in real life.



# Areas for Improvement

## Unreliability of Email and Fonts

The unreliability of the email server is a significant area for improvement. This issue not only affects the confirmation email feature but can also impact the overall user experience. Exploring other email server options or integrating with a more reliable email service provider could significantly improve this aspect of the program.

The occasional failure of the custom font to load can negatively impact the aesthetics of the user interface. While the program can function with the default font, the inconsistent appearance can create an unprofessional impression. This issue could be addressed by investigating the root cause of the loading problem and implementing a more reliable method for loading the custom font.

## Error Handling and Feedback

An area that could be improved is the lack of specific error messages when user input is incorrect or missing. Providing detailed feedback to the user not only improves the user experience but also reduces potential user frustration and confusion.

## Unit Testing

During the development process, I implemented some unit tests, particularly at the beginning of the project, but they weren't extensive enough to cover all the complexities and functionalities of the system. A more comprehensive suite of unit tests would have allowed for better detection of potential issues, ensuring that each component of the program works as expected independently and when interacting with other components.





# XI

## Changes to the Original Plan

# Changes to the Original Plan

## A Learning Experience

The initial plan provided a good starting point for the project, but as the development progressed, several modifications were made to accommodate the evolving requirements and unforeseen challenges. One significant departure from the original plan was the time allocation. The project took almost double, if not triple, the initially planned time. This was primarily due to the complexity of the project and the continuous addition of new features to improve the overall functionality and user experience.

Originally, the plan included a separate screen for jet ski selection. However, during the development phase, it was decided to merge this functionality with the date selection screen for a more streamlined user experience. This decision led to the removal of the initial jet ski selection screen.

Additionally, two new screens were introduced: the cancellation policy screen and the safety guidelines screen. These were added towards the end of the project to provide essential information to the users, enhancing transparency and ensuring user safety.

The implementation order mostly followed the original plan, but some components were developed concurrently to meet the project's evolving needs. The need for agile development became apparent as the project progressed, and some flexibility in the implementation order was necessary to manage interdependencies between different parts of the project effectively.

In hindsight, it's clear that the initial time estimate was optimistic. The complexity of creating a custom calendar widget, developing a user-friendly interface, and ensuring robust functionality took longer than anticipated. This project has been a learning experience in terms of time management and planning for development projects. It's evident that in software development, it's crucial to account for the possibility of overruns and unexpected complications in the initial project plan.

Despite these changes, the final project effectively meets its objectives, providing a functional and user-friendly jet ski reservation system. The deviations from the original plan, in fact, helped shape a better end product, emphasizing that adaptability is an asset in the field of software development.





## **XII**

# **Realized order and schedules**

# Realized order and schedules

## Order of Operations

The project's implementation was carried out in the following order:

- **Planning and Design (20th of February 2023 – 3rd of March 2023):** This phase followed the initial plan accurately. Understanding the project requirements, identifying necessary features, and planning the system's classes and methods were done during this time. The user interface was also sketched out.
- **Implementing Core Functionality (4th of March 2023 – 23rd of March 2023):** The implementation of the core functionality was initiated as planned. However, the process took longer than anticipated due to the complexity of the project. Also, a significant change was made in this phase. Initially, a separate screen was planned for jet ski selection, but it was later merged with the date selection screen for a more streamlined user experience.
- **User Interface Implementation (24th of March 2023 – 14th of April 2023):** This phase also took longer than planned. The process of developing user interface components using PyQt, implementing logic to connect UI components with system functions, and ensuring a pleasant and functional user interface was time-consuming.
- **Testing and Debugging (15th of April 2023 – 30th of April 2023):** Testing and debugging was an ongoing process throughout the project, but it was more intensely focused on in this phase. Unit tests were run on system components and methods, and the system's workflow and user interface were manually tested.
- **Additional Features and Final Touches (1st of May 2023 – 12th of May 2023):** After the core system was functional, additional features like the cancellation policy screen and safety guidelines screen were added. This phase also included finalizing the user interface and performing comprehensive testing and debugging.

The project did not strictly follow the initial schedule due to the underestimation of the time required for each phase. The complexity of the tasks and the continuous addition of new features led to this time overrun. There was also a deviation from the plan in terms of the implementation of some features. However, these changes led to a more efficient and user-friendly system.





## **XIII**

# **Assesment of the Final Result**

# Assessment of the Final Result

## I Am Very Proud of What I Have Achieved

Looking at the final product, I am proud of what I have achieved with the Hetijetti Jet Ski Reservation System.

The program's strongest feature is undoubtedly its user interface. The interface is clean, intuitive, and aligns with the existing Hetijetti brand, which I also created. This consistency enhances the user experience and brings a sense of professionalism and credibility to the system. I also believe the integration with Google Sheets API was a successful addition, enabling easy management of reservations. I cannot forget to mention the custom calendar widget, which was a complex component to develop but worked out very well. It allows users to easily see available time slots and make reservations accordingly. The program's realism, including features like sending a confirmation email to the user, is another strong point. It makes the system more practical for potential real-world use.

However, like any project, it does have its weaknesses. The most significant technical issue is the inconsistency of the email server. Sometimes the confirmation email doesn't send, and this is definitely a flaw that needs to be addressed. It might be due to an issue with the mail server itself, and exploring alternative options could be a solution.

There are also several shortcomings, mostly opportunities for added functionality that were missed due to time constraints. For example, the system doesn't currently support multi-jet ski reservations or weekend reservations. These are functionalities that could be added in the future. Some user interface details could also be improved, such as better error messaging on the customer details screen and a more comprehensive confirmation email. Additionally, there's an unidentified issue with the font loading that affects the aesthetics of the user interface.

As for the code structure, I believe it is suitable for making changes and expanding. The classes are defined clearly and their responsibilities are well divided. The code is modular and methods are well-encapsulated, which promotes code reusability and maintainability.

Overall, despite its limitations, I believe the Jet Ski Reservation System is a solid project that can be expanded and improved in the future. The foundation is strong, and with some additional work, it could become a fully functional, real-world application.







# **XIV**

## **References**

# References

## **<https://hetijetti.fi>**

hetijetti.fi website developed by me, the not-so-hypothetical company of this project

## **UML Class Diagram Creation**

<https://gitmanager.cs.aalto.fi/>

## **W3Schools**

general knowledge about python syntax and OOP

<https://www.w3schools.com/python/>

## **PyQt General Information**

general knowledge about pyqt

<https://riverbankcomputing.com/software/pyqt/>

## **Wikipedia**

Creation of the plan and document, information about datatypes and what they stand for

<https://www.wikipedia.org>

## **Establish Strategic Partnerships**

Collaborate with local businesses, events, tourist attractions, and public transportation providers to increase Dibs Electric's presence and demonstrate the brand's commitment to the community.

## **A+, Y2 course material**

<https://plus.cs.aalto.fi/y2/2023/?hl=en>

## **MyCourses, Y2 course material**

<https://mycourses.aalto.fi/my/>

## **RealPython**

reading about data structures

<https://realpython.com/>

## **PyQt6 Documentation**

<https://www.riverbankcomputing.com/static/Docs/PyQt6/>

## **Python Documentation**

Python documentation

<https://docs.python.org/3/>

## **Google Sheets API**

<https://developers.google.com/sheets/api/quickstart/python>



# References

## **SMTP for Python**

<https://docs.python.org/3/library/smtplib.html>

## **Unit Testing Framework**

<https://docs.python.org/3/library/unittest.html>

## **DateTime Module**

<https://docs.python.org/3/library/datetime.html>

## **StackOverflow**

<https://stackoverflow.com/>

## **GitHub**

<https://github.com/>

## **Reddit: Python Community**

<https://www.reddit.com/r/Python/>





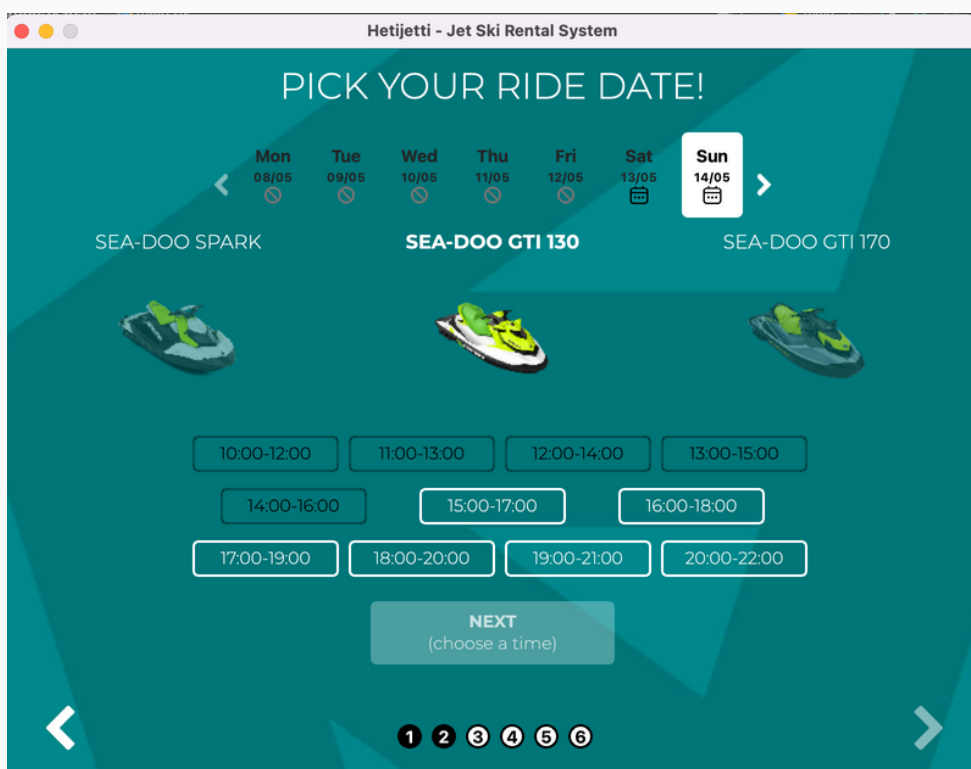
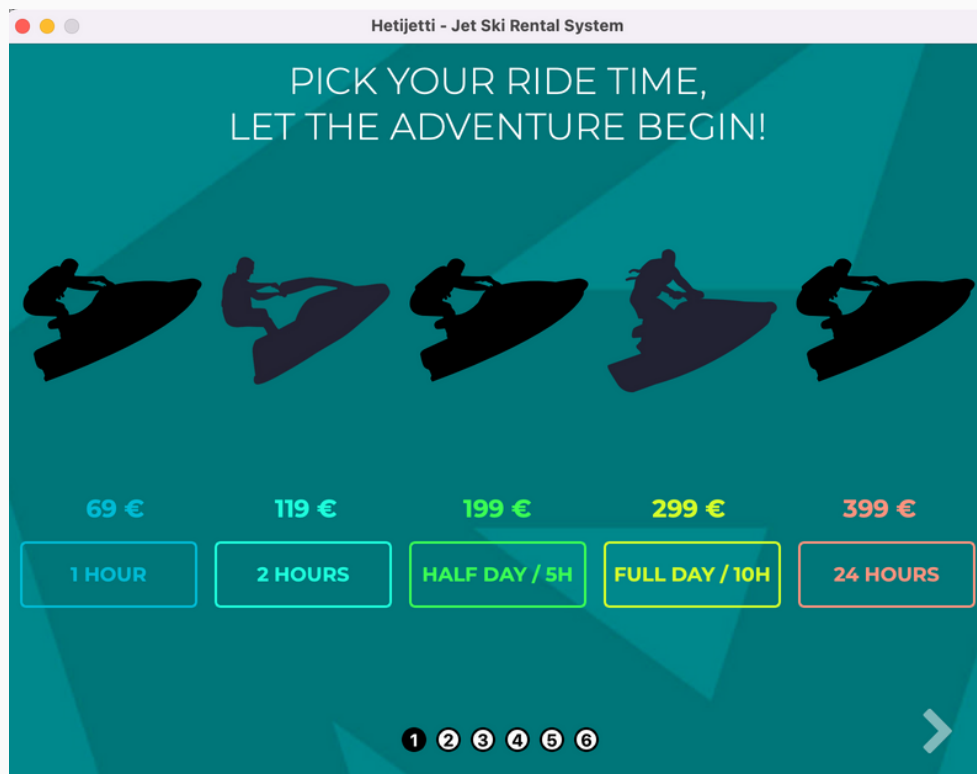
# **XV**

## **Attachments**

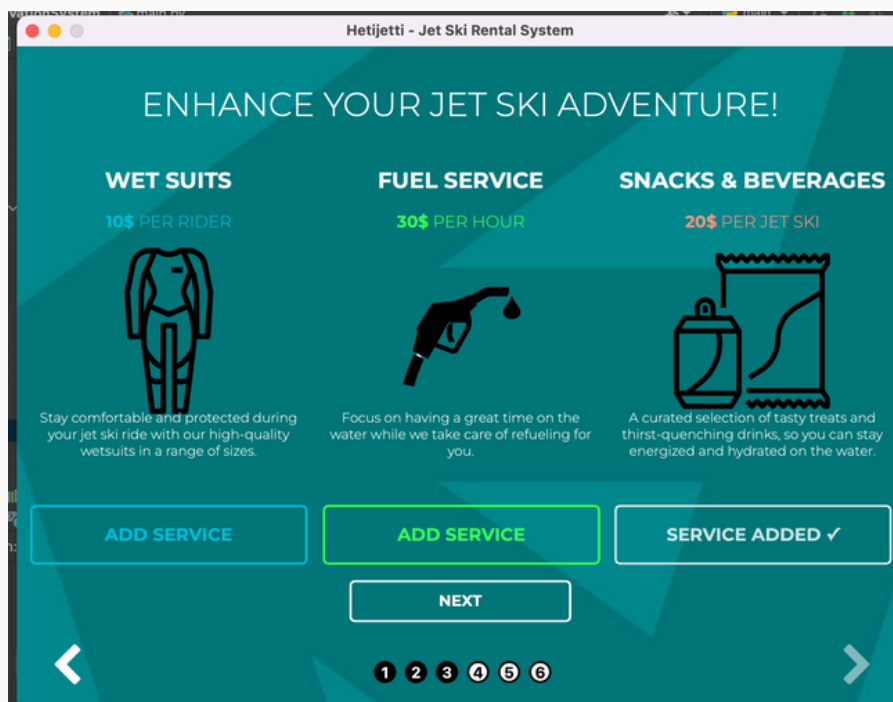
---

Tatu Rouhiainen | [tatu.rouhiainen@aalto.fi](mailto:tatu.rouhiainen@aalto.fi)

# Screen Flow



# Screen Flow



# Screen Flow

Hetijetti - Jet Ski Rental System

## HANG TIGHT! SHARE YOUR DETAILS TO RIDE


First Name \*

Last Name \*

Email \*

Phone Number \*

Additional Information

1  Number of Riders (1-3) \*

☐ I confirm that I am at least 18 years old


Fields marked with an \* are required.

NEXT

1 2 3 4 5 6

Hetijetti - Jet Ski Rental System

## ALMOST THERE! CONFIRM YOUR JET SKI ADVENTURE



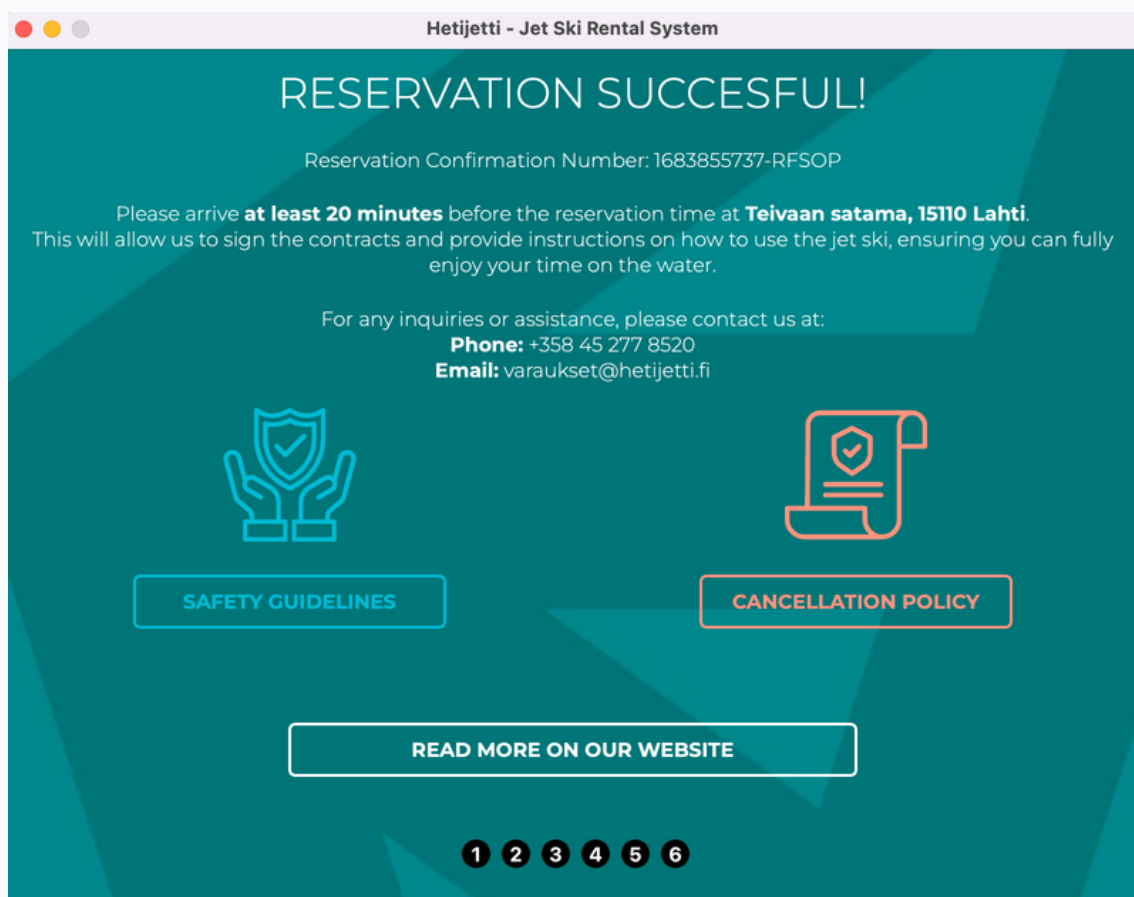
Jet Ski:	Sea-Doo GTI 130	
Reservation Date:	25/05/2023	
Reservation Time:	11:00-13:00	
Reservation Length:	2 hours	119 \$
Riders on Reservation:	2	
Additional Services:	Snacks & Beverages	20 \$
Total price:		139 \$
Customer:	Tatu Rouhiainen tatu.rouhiainen@gmail.com 0452778520 tdaykdja	

CONFIRM RESERVATION

1 2 3 4 5 6



# Screen Flow







Project By:

**Tatu Rouhiainen**

**100810721**

**Informaatioteknologia**

**12.05.2023**

---

tatu.rouhiainen@aalto.fi