Design Report - Solar Boat Twente

Tessa van Belois (s2510332), Oliver Davies (s2159449),
Matthew Pieper (s2131455), Darrell Tufto (s1935941),
Tibet Tugay (s2489384)
Supervised by: Champika Epa Ranasinghe

April 2024

1 Abstract

A strategy tool for the student team Solar Boat Twente has been developed as an assignment for the Design Project module given by the University of Twente. This tool helps the team decide when to charge the boat and when to use the energy in the battery. This report details the overall process of that project from the initial requirement-gathering phase until the final product.

Contents

2 Introduction

Our project group was tasked with developing a strategy tool for the **Solar Boat Twente** team, a student team that achieved third place in the endurance race at the Monaco World Championships in 2022. The Solar Boat Twente team is a diverse group of students at the University of Twente. They design and build a solar-powered boat to participate in races against other student teams, including the national and world championships of solar boat endurance races. During an endurance race, the team has a set amount of time to sail as far as possible. To do so, the team wants to optimize the boat's average speed. The most efficient way to sail fast is above a certain speed, when the forces on the foil of the boat lift the boat out of the water, reducing drag while increasing speed and efficiency. This is called foiling (see figure 1). The current solar boat still loses more energy while foiling than it gains from its solar panels, meaning it can only foil for a certain amount of time until it has to start charging its battery again. The team needs a strategy to select when to foil and charge optimally.

Figure 1: Foiling solar boat

Our project aimed to develop a tool for the solar boat team that gives them this strategy. This tool needs to display the relevant information and shorten the time the team strategist takes to make decisions. To achieve this, we have designed and implemented a dashboard that takes the raw data provided by the boat and displays the relevant processed data. Throughout this paper, we will discuss and review our design and development process, design choices, and project evaluation. Moreover, this paper aims to explain how the product operates and how to run it.

3 Learning Goals and Responsibilities

While all of us are computer science students, we are also a diverse group with different skills, backgrounds, and interests. Ahead of this module, we discussed our learning goals within the group to see how the work should be divided and which person should be responsible for the possibility of achieving those learning goals. This section lists those learning goals, resulting in a division of responsibilities.

Tessa

Having recently become interested in UI design, I wish to gain more practical experience in realizing a design in code. TCS contains a lot of group projects, but for most of these, the project scope is defined beforehand. The project we have been assigned for this module is somewhat ambiguous and open to interpretation, so we will probably spend considerable time on requirement engineering. I am mainly interested in designing the front-end in collaboration with the client. My learning goals are:

- Learning a new front-end framework (Vue);
- Further develop requirement engineering skills and implementing requirements into a design;
- Working from a fully-fledged design to code for the front-end, improving practical coding skills;
- Improving project soft skills, like working with teammates and clients.

Oliver

While Technical Computer Science has taught me many theoretical aspects of Computer Science, and my extracurricular experiences at the university have given me a good grip on my soft skills, such as planning and time management, I have minimal practical experience in developing robust software for clients. In this module, I want to learn:

- Familiarizing myself with a practical front-end framework (in this case Vue);
- Familiarizing myself with a practical back-end framework (in this case Django);
- Getting started with front- and back-end communication;
- Furthering my UI design skills with HTML and CSS.

Matthew

As a student whose preference lies in back-end development, my heart lies with implementing the system functionalities. An essential aspect of this project is understanding the available data and figuring out how to query and process it. Specifically, time series data is the primary data type used by the Solar Boat Twente team. This involves learning how to work with time series databases (in this case, InfluxDB), which are a little different from relational databases. Furthermore, I lack experience in working with clients in real-life scenarios. This project is a perfect opportunity to work on this soft skill. Keeping close ties with the client to communicate realistic goals and ask for the needed resources is another crucial aspect of this project. To summarize, my learning goals are to develop my skills in:

- understanding and processing data to implement physics models;
- working with time series databases (InfluxDB);
- client communication.

Darrell

As someone already quite familiar with programming and the frameworks used in our project, I will try to familiarize myself with the role of a DevOps engineer. In this module, I want to learn:

- Docker;
- Continuous Integration/Continuous Development environments;
- Code Quality Assurance;
- Deployment.

Tibet

Even though I have gained quite a lot of practical experience throughout the projects within our study, I am unfamiliar with the frameworks we will use in this project. By the end of this module, I want to be more confident with the systems we use and enlarge my scope on the entire web app creation process instead of knowing parts from different sections. I plan to achieve this by realizing the following goals:

- Familiarizing myself with the front-end framework of our choice (Vue);
- Familiarizing myself with GraphQL as an alternative to REST;
- Adopting better Version Control practices in my workflow (such as PRs) and using terminal commands for VC;
- Play an active role in analyzing the data we receive from the client and the respective calculations.

Person	Responsibilities
Tessa	UX/UI
	Energy display cell
Oliver	Planning
	Information display cell
Matthew	State-of-charge cell
Darrell	Version control
	Dev env
	Back-and frontend communication
Tibet	Time series forecasting
	SoC quick view cell

Table 1: Divison of Responsibilities per member

Responsibilities

Based on these learning goals, our responsibilities were set, for which an overview is given in table 1. In the context of our project, *Being responsible* for something does not mean that the person had to implement a specific feature solely. This person was responsible for the group finishing this segment promptly. Note that not all responsibilities have proved relevant to the project; responsibilities that did not make the final design are indicated in red.

4 Planning

At the start of the project, we created a comprehensive planning with specific deadlines to adhere to, so that the group members' expectations are aligned and the deliverables and deadlines are clear to everyone. In the following section, we will outline the original timeline that we planned to follow and explain any possible deviations from it.

4.1 Timeline

In this section, we will revisit the planning we originally made for developing the Solar Boat Twente strategy tool project. For simplicity, we will refer to the deadlines regarding project week numbers defined in table 4.1.

In this timeline, we define three phases of the project: the project proposal, the development, and the presentation. Gantt chart 4.1 gives an overview of the timing of these phases in weeks and the critical project deadlines.

4.1.1 Project proposal

The project proposal for the Solar Boat Twente strategy tool will be created in the first three weeks. To make a comprehensive proposal, one or more meetings will be had with the client in the first couple of weeks to gather the requirements

Week no.	dates
1	February 5th - February 11th, 2024
2	February 12th - February 18th, 2024
3	February 19th - February 25th, 2024
4	February 26th - March 3rd, 2024
5	March 4th - March 10th, 2024
6	March 11th - March 17th, 2024
7	March 18th - March 24th, 2024
8	March 25th - March 31st, 2024
9	April 1st - April 7th, 2024
10	April 8th - April 14th, 2024
11	April 15th - April 21st, 2024

Table 2: Week numbers and the dates they represent

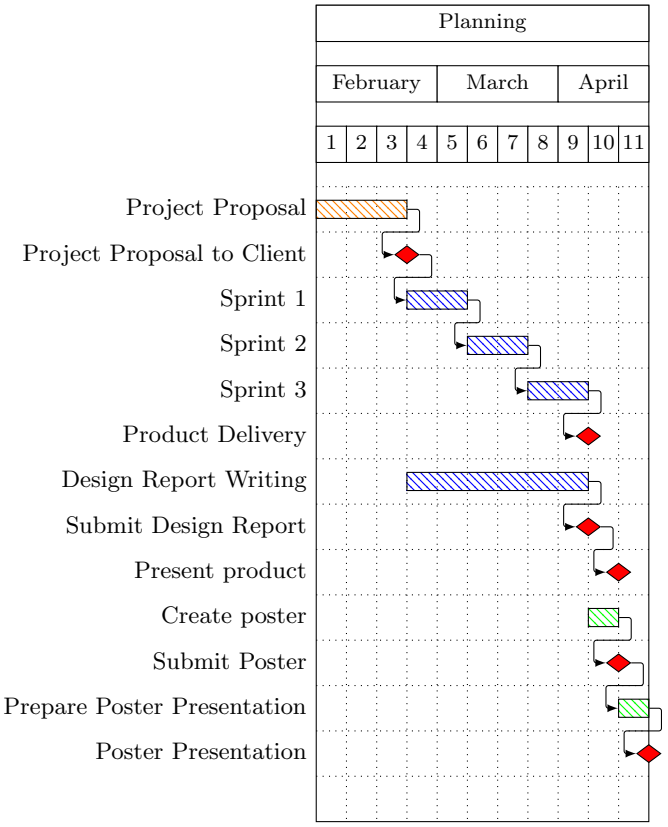

Figure 2: Gantt chart of the project timeline

for the product. In the first week, we will also find a supervisor and have an initial meeting about the procedure and deadlines of the project. Based on these initial meetings, a proposal will be created with an initial design for the product, containing the features we deem achievable and valuable for the client.

4.1.2 Development

Once the client has approved the project proposal, development can start at the beginning of week four. The development is divided into three sprints of two weeks; at the end of each sprint, the client will be consulted to help plan the next sprint. The responsibilities for the working of the product's different features have been divided among group members as specified in section 3 to ensure that each feature is completed on time. During this phase, regular meetings will be planned with the client and the project supervisor to maintain a quick feedback loop so that desired changes to the final product can be made quickly and easily. During this phase, the product report will be updated every week. At the end of week nine, the final product will be delivered to the client, and the final report will be delivered to the project supervisor.

4.1.3 Presentation

When the product has been delivered to the client, we will present the product to the client and project supervisor in week ten. We will also spend week ten preparing the poster presentation of the product. In the final week, we will participate in the Design project poster presentation.

4.2 Changes in planning

The planning as specified in figure 4.1 was followed mainly well by the project group. However, the project proposal's first version was not entirely reflective of the client's wishes, so a co-design session with a previous strategist was planned as described in section 5.4 to refine the original product design. The proposal was adapted based on this session and finalized at the end of week 4. Consequently, the first sprint was a week shorter than initially planned.

5 Design Methodologies

This section details the design process of this project. Firstly, we discuss the methodology used in the initial requirement engineering process, where we created an initial lo-fi design for our client. Secondly, the development process, an adapted version of Scrum, is discussed, which fits the module's structure.

5.1 Requirement Engineering

At the start of the project, we were provided with a brief assignment description that left a lot of room for interpretation and creativity. The general premise was that they need a tool that calculates and shows the optimal foiling strategy and other essential data. How these things are done was up to us, and the client did not have a clear idea of what they wanted. None of us had interacted with a Solar Boat before, so we first had to learn how a boat works, what data could be significant, and what an optimal strategy could look like. All these factors meant that the requirement engineering and design process took considerable time. The lack of access to the actual data delayed the process further; despite requesting the data, we were provided the database structure instead.

5.2 Development cycles

The first three weeks of the project were planned for creating the Project Proposal. We spent the first two weeks brainstorming amongst the group and together with the client to find out what functionalities the application should have and what data it should calculate and show. We created a list of (non-)functional requirements and categorized them in a session with the client based on the MoSCoW method of prioritization. Since the members of the Solar Boat Team change every year, and the current team has not competed in any races yet, we thought it would be useful to ask the previous strategist for their opinion. After this co-design session, we finalized the requirements and the design and started the development phase of our project.

5.3 Initial design

Figure 3 shows the initial design we made based on the information we thought was relevant to the strategist. The main focus of the dashboard is to show a foiling strategy, which we decided to show in a graph titled the State of Charge (SoC) graph. The graph (in the middle of the screen) shows the past SoC of the last 20 minutes and the predicted future SoC, both if you keep foiling - which uses more energy than you gain, and if you stop foiling to charge. This graph is still present in the final design, with the minor change of showing less past data and more of the predicted.

To calculate the SoC, the dashboard needs to know the approximate solar power they will gain with the solar panels on the boat during the race. Through discussions with our clients, we figured out that using actual weather data would

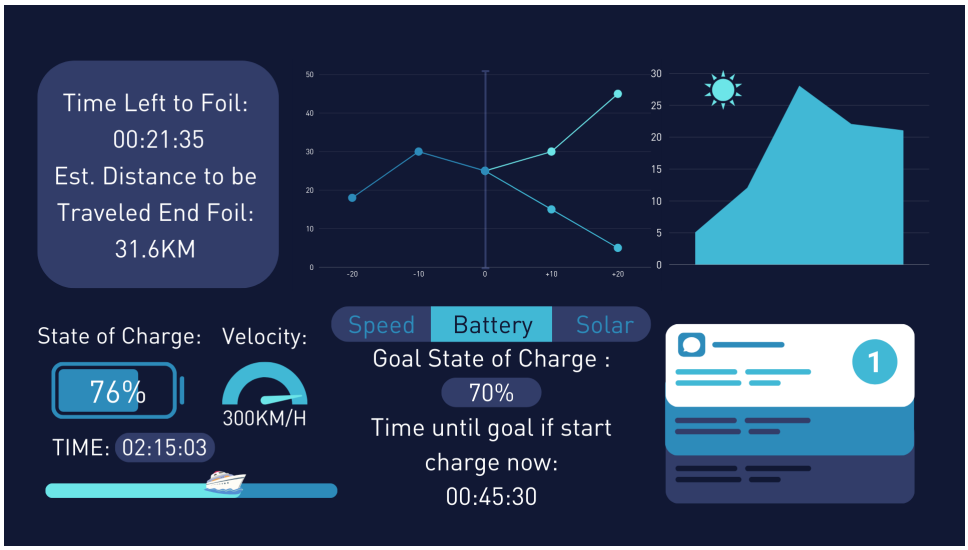

Figure 3: Initial front-end design

not be within the scope of our project, so we decided to remove the graph (top right) and instead add an input field (the sun in the final design) so the strategist can fill in an approximation themselves. We also removed the real-time data, like the current speed of the boat, as the team has a separate application to show them.

5.4 Co-design Session

When the first draft of the proposal was finished, we sat with the strategist of the last year for a co-design session, as shown in figure 5. We first asked the strategist to walk us through their thought process during a race to try to gain a clearer understanding of what factors impact decision-making. We went through the requirements and features of the dashboard; the strategist only noted that seeing a prediction of the number of kilometers the boat will sail throughout the race would be an excellent addition.

The strategist individually drew a dashboard of what they would like to see, after which our initial design, shown in figure 3, was revealed. Discussing the differences resulted in figure 4.

The biggest addition from the co-design session was the addition of the foiling and charging time until the end of the race. The strategist reasoned that there might be factors during the race that the dashboard cannot include, which the strategist will use to make decisions. If they then know how much time they need to be at least charging at some point during the race, they can decide for themselves at what point in the race they will be doing that.

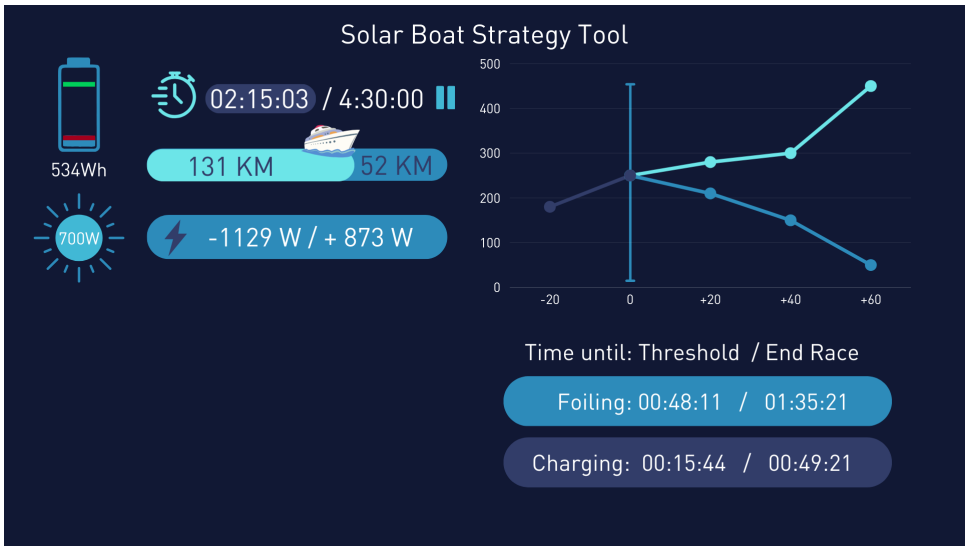

Figure 4: Final Lo-fi design of UI

Figure 5: Co-design session where last year's strategist helped us develop the UI.

6 Requirements

In this section, we detail our functional and non-functional requirements, as well as the validation and prioritization of these requirements and how they changed during the project.

6.1 Requirement Gathering

As mentioned in section 5.1, we did not have prior knowledge of solar boats and solar boat races. To resolve that, the client was interviewed to acquire insights into the field and determine the project scope. The problem statement, "getting the optimal times to foil during an endurance race," and the current solution, an Excel sheet to calculate how long the boat had to charge during a race, were also discussed. At this point, the strategist, our contact person from Solar Boat Twente, was determined to be the main stakeholder. A full stakeholder analysis can be found in Appendix A. After this, we brainstormed a list of user stories. These user stories formed the basis for the project's requirements, which can be split into functional and non-functional requirements. A summarized list for both types is detailed in the next two sections.

6.2 Functional Requirements

The functional requirements describe the interactions the system should provide to the user.

As a user/maintainer, I want to be able to ...

- ... change the motor controller before a race
- ... input and change the race time left in the race
- ... start, pause and resume the race timer
- ... input and change the estimated solar cell power
- ... view the past, present and future battery capacity of the boat
- ... view the time left foiling and charging until their respective thresholds are reached
- ... change the foiling and charging thresholds
- ... view the total energy used/gained throughout the race
- ... view the total distance traveled throughout the race

6.3 Non-Functional Requirements

The non-functional requirements describe specific criteria the system should hold to.

As a user/maintainer, I want ...

- ... the system to be written in Python or C++
- ... the system to be an installable app or deploy-able software
- ... the interface to be clear and concise
- ... to see the amount of energy in the battery in an instant
- ... the interface to fit a desktop screen
- ... the strategy to refresh at the latest every 15 minutes, and if just the Excel sheet strategy is implemented every 5 seconds
- ... all displayed information to be secured behind authentication

6.4 Validation and Prioritization

After the user stories were constructed, the team validated them with the strategist. The team inquired what requirements were good and what requirements were still missing. At the same time, both functional and non-functional requirements were prioritized using the MoSCoW framework. Appendix B shows the MoSCoW prioritization as defined at that time, together with whether the requirement was finished.

6.5 Changes during the project

It was difficult to find the exact requirements together with the client since they did not have a clear idea of what they wanted. During development, we encountered some minor problems and clarity issues regarding the requirements. We discussed these with the client to get more clarity. In turn, the client narrowed down what they wanted, which led to us updating our requirements. Below is the list with all the changed user-stores and the reasons for the change.

As a user, I want ...

- ... to get a warning when the boat's battery is dangerously low.
  The initial design had a notification system, which was scrapped after the co-design session with the client.
- ... to be able to adjust the system to a different boat, specifically to the energy efficiency curve used to choose optimal sailing speeds.
  During the project, we found that the efficiency curve provided by the client was incorrect. Fixing this was deemed not in the scope of this project by the client.

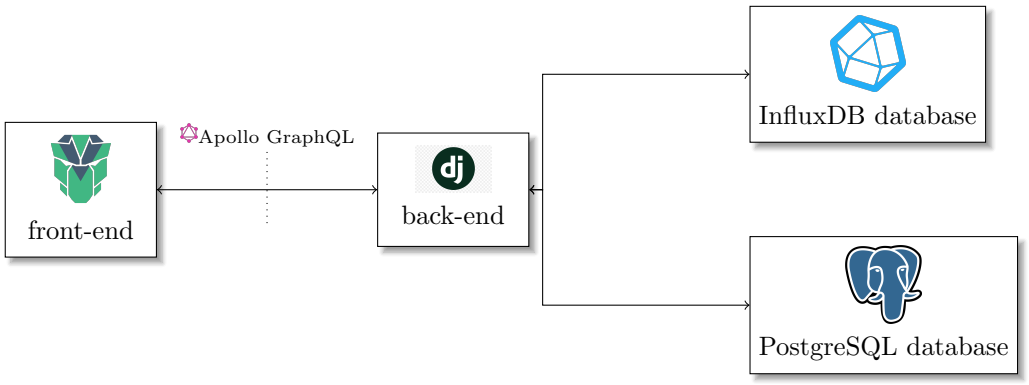

Figure 6: Top-level overview of the designed infrastructure.

... to know the confidence interval of a given strategy.
Initially, we wanted to suggest speeds other than the "ideal" foiling speed if there was enough battery capacity left. However, this was also deemed out-of-scope because of the previously mentioned issues with the efficiency curve.

The final change was less of a user story and more that the Solar Boat team wanted us to help them with the project deployment instead of doing the last user stories we had left.

7 Infrastructure

We could design our infrastructure based on these requirements, which are expanded upon in the following subsections. A top-level overview that shows our different components and how they relate to each other is given in figure 6. In the rest of this section, all the design choices regarding these components are explained in detail.

7.1 Front-end

We decided to make our interface a web app that can be used on any device without installation. In the end, we decided to use Vue[1], as according to blogs on the internet, it has a low learning curve, is maintained often, and fits our needs of showing different components and rendering them reactively. Because we did not want to reinvent the wheel, we looked for a templating library that fits the needs according to our design, eventually ending up with PrimeVue[2]. PrimeVue and other libraries we use are managed through our package manager Yarn[3]. This front end is automatically built through Vite[4], making the deployment more straightforward for our client.

7.2 Back-end

One requirement was that we use Python, as the client is already familiar with that programming language, and they wanted to be able to improve the product themselves after our project was finished. Because of this and our need for a second database, as explained in section 7.4, we are using Django[5]. A secure back-end framework that gives access to our database of choice through an ORM. It is an easy choice as we could focus on the product instead of worrying about security issues like SQL injections, which are automatically protected against. Like in the front-end, we are not trying to reinvent the wheel, and if an open-source and well-maintained library is available for some functionality, we will use it. To ensure that everyone's development environment stays up-to-date. We are making use of Poetry[6], which is a Python package manager.

7.3 Communication layer

The GraphQL framework connects the front and back-end. GraphQL is an alternative to REST with the following advantages:

- You can query only relevant fields on a model and thus not receive unnecessary fields in your return;
- It only makes use of a single API endpoint, making managing the API easier;
- GraphQL included subscriptions: if data changes on the server, the front-end will automatically receive new data;
- Every field is strongly typed, ensuring a higher code quality.

While RESTful services are better at caching, GraphQL fits our project better because we are not retrieving past data from the back-end at any point. The specific implementation we use is Apollo GraphQL[7], which is well-integrated into our front-end library Vue in the form of Apollo composable[8].

7.4 Databases

The solar boat team supplied us with an export of their database, which is an Influxdb database. Influxdb databases are time-series databases without relational mapping between different "tables." As such, what usually would be called a "table" is called a measurement here. Since non-relational databases cannot store accounts, which is one of the requirements, we have supplemented the system with a PostgreSQL database. This choice was made because it is one of the officially supported databases, with the added benefit that our client, a computer science student, has used PostgreSQL in their studies. Section 7.5.2 explains why we are saving extra fields.

7.5 Data

7.5.1 Data used from the Influxdb

Solar Boat Twente's Influxdb time series database has several measurements, but only a couple were relevant to our product. The measurements used are as follows:

- 'Observer': this measurement has the field 'velocity,' which contains the measured speed in meters per second. The velocity is used to calculate the total distance traveled by integrating the velocity over the time it is measured. The velocity is also used to calculate the total energy spent by multiplying the velocity with a set motor efficiency coefficient in Joules per meter, yielding the power used in Watts, and then integrating the power over time to produce a total energy consumption in Joules.
- 'BMS': this measurement has field 'pack_current,' which contains the current in Ampère running through the battery management system, and field 'pack_voltage,' which includes the voltage in Volts of the battery management system. These fields are multiplied to produce the power in Watts that the system is currently spending or producing. The power is then integrated over the time measured to create the total amount of energy in Joules gained or lost in the given time. This value, combined with the formerly mentioned energy consumption over the same time stamp, gives us our net energy gain or loss from which we calculate the battery level over time.

7.5.2 Saving Data

To ensure we can keep showing new results every minute, we need to save the results of our previous calculations, as querying hours of data will make our system too slow. For every calculation we make, we continue from the result of our last calculation. An example is the state of charge of the battery. Figure 7 shows the entire data flow through our web app. We make a request through Apollo Composable for new data. Django graphene, the graphql endpoint in our back-end, then combines the last minute of data from the InfluxDB with the data that is already present in the PostgreSQL database. This way, our calculations can be made quickly enough to keep serving new data to the front end every minute. This means that our calculations become slightly less precise, as we only store the intermittent results by two decimals, which is insignificant to our application.

8 UI Components

The dashboard is divided into 4 parts, called the cells. The design of the dashboard is supposed to be simple, so the strategist only has to glance at it to gain the relevant information. For this purpose, the design contains as little text

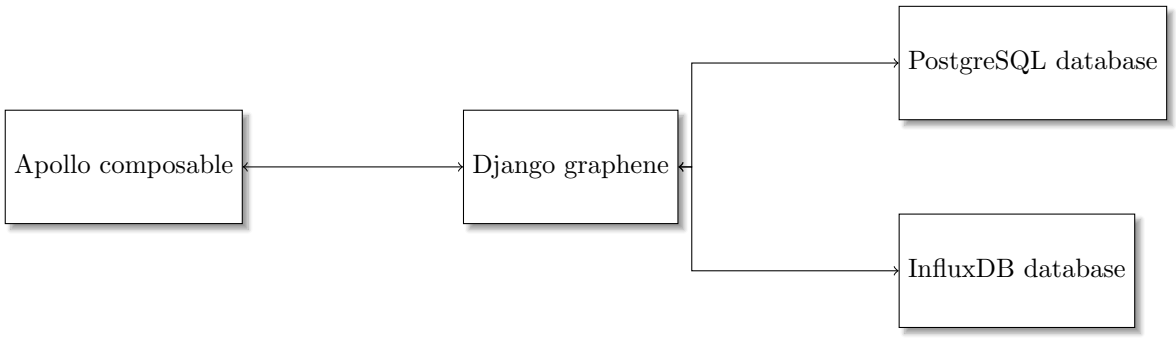

Figure 7: Dataflow of information requests

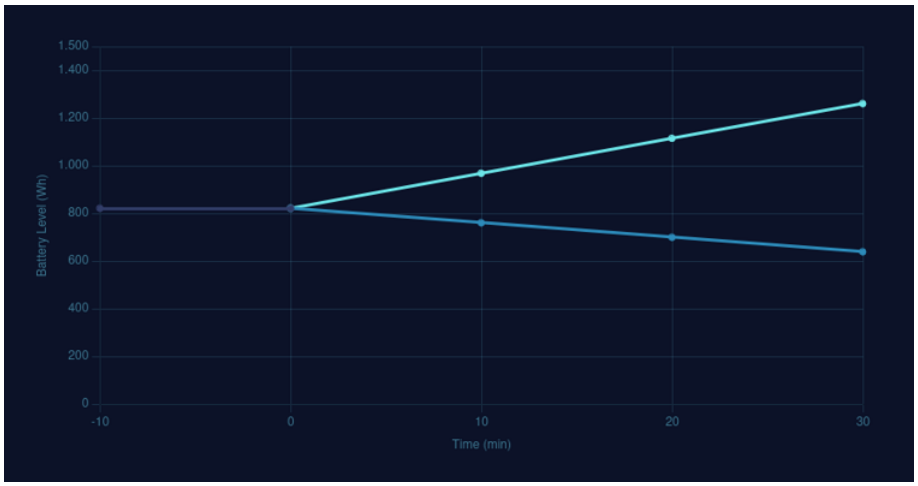

Figure 8: Charge of Charge Cell

as possible. This does mean that there is a slight learning curve to using the dashboard, which the client accepts, and even welcomes. The reasoning is that the strategist will use the dashboard often enough to learn what the elements mean, and a competing team would not be able to understand the strategy, were they able to take a quick look. All the items that were deemed necessary to include in the dashboard fit in 3 out of the 4 cells, so we conferred with the client to leave the fourth cell empty, as they want to implement whatever they need in the cell after they use it in the test races.

8.1 State-of-Charge cell

The state of charge cell is on the top right of the screen. It contains the State of Charge graph, which displays the SoC of the battery over time, displaying the past, as well as a prediction of the future, both if you foil and if you charge.

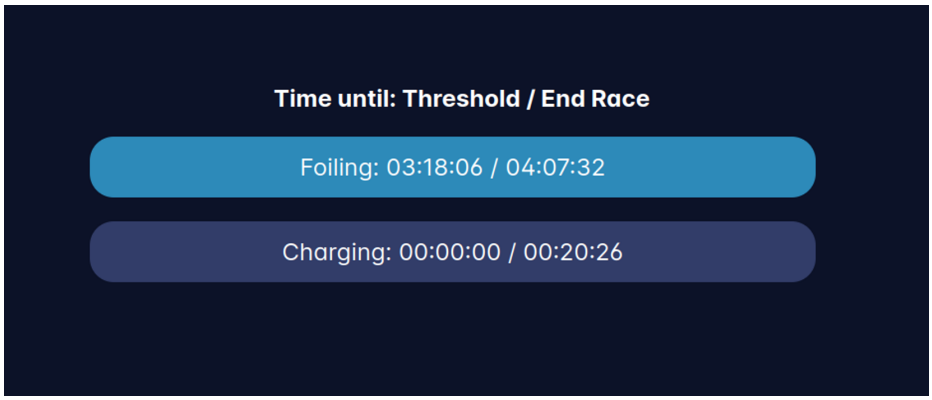

Figure 9: Charge of Charge Quickview

The graph displays the battery level from 10 minutes ago, the current battery level, the battery level if you were foiling and charging in 10, 20, and 30 minutes. The display of 10 minutes ago is just taken from the Postgres database. For the estimations, the calculations take in the solar power provided by the client (in Watts), the current battery level(Watt-hours), and the velocity the boat is moving with while charging/foiling(meters/second). It calculates the power used for the said velocity according to the directions also provided by the client. Lastly, this prediction, the combination of the solar power(Watt-hours) and the power used to move at that velocity(Watt-hours), is added to the current battery level for charging and subtracted from it for foiling, resulting in the graph showing the estimated battery in Watt-hours.

8.2 State-of-Charge Quick View Cell

Below the SoC graph, the data is displayed numerically as well. There is a lower threshold (non-zero) where the boat should stop foiling, so there is still enough energy for the boat to be on and sail instead of fully turning off. There is also an upper threshold at which the charging is less efficient, and the boat should not be charging above that point. The thresholds can be changed in the Information display cell, when you click on the text below the battery. For both foiling and charging, the cell shows the time until reaching the threshold, and the total time the boat will foil/charge for until the end of the race.

The calculations for the quick view cell is fairly similar to the SoC graph since its purpose is to show what the SoC graph shows in a more intuitive way and for a broader scope. In contrast to the graph, which aims to aid the strategist in short-term decision-making by showing them what they can expect the battery level to be, the quick view aims to guide the strategist with the most efficient strategy, which is to foil until the boat can't then charge until the upper threshold and let the boat foil for the longest time again.

The strategy calculates how much the boat can foil until the threshold by

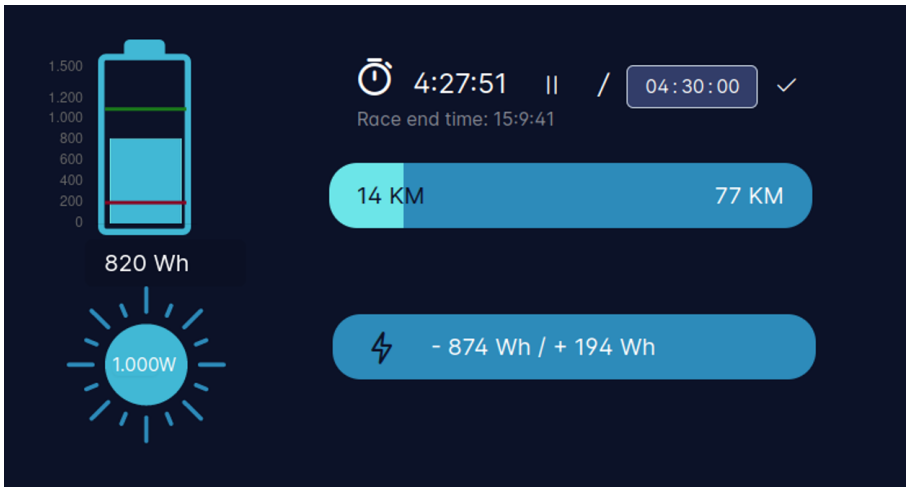

Figure 10: Information Display Cell

taking the solar power(Watts), the current battery level(Watt-hours), and the foiling velocity(meters/second). The same is done for charging. Both of these calculations are the same as the SoC graph calculations, the only difference being that these use the predictions to calculate time until the threshold instead of showing the battery levels on the graph.

The end-race functionalities are then calculated according to the foiling/charging ratio. These calculations account for the initial foil to start the race with and apply the ratio to the rest of the race, resulting in an accurate description of how long the boat should foil and charge for during the entire race.

8.3 Information Display Cell

The top left displays the rest of the information. The battery graph shows the State of Charge(Watt-hours). The thresholds are displayed as well. If you click the text under the graph, a pop-up field shows up, where you can change these thresholds. The calculations for the quick-view use these thresholds as mentioned earlier.

On the right of that, the Race Timer is displayed. The timer shows the end of the race and the actual time at which the race will end. Furthermore, there is a field for changing the time left in the race, in case the officials decide to stop the race earlier. Moreover, there is a pause button on the right side of the timer. The client assumed this will not be used that often but it is still a nice feature to have in case of an unlikely pause.

Below the Race Timer is the distance view. It shows the total distance already traveled, and the estimated distance to still be traveled(in KM for both). The distance traveled calculates the integral of the boat's velocity to time. On the other hand, the estimated distance to travel uses the total foiling and

charging times from the SoC quick view to calculate the estimated velocity of the boat for the rest of the race after which, it uses both the time and the velocity to calculate the distance the boat will travel.

Below the Battery graph, the sun is an input field for the Solar Power the strategist thinks the boat will gain during the race(in Watts). This amount is used to estimate the future SoC of the boat, which is used in both the SoC graph and the SoC quick view cells.

Next to the solar power input field, the total energy gained and used throughout the race is displayed(in Watt-hours for both). The energy consumed is calculated by multiplying the velocity(meters/second) with the power guide for the said velocity(Joule/meter) the client has given us. The energy gained is calculated by adding the battery capacity difference(Watt-hours) and the energy consumed(Watt-hours). This calculation of the energy gained is independent of the estimated solar power gain input by the client and could serve as a tool to cross-reference the solar power estimation with.

9 Testing

This section details the testing done for this project. The subsections explain how the back and front end are tested separately using various methods.

9.1 Back-end

There are two types of testing done in the back end: unit tests for separate functions, which have clear input and output, and API tests, which mock requests sent by the front end to ensure that all API endpoints return the correct response. Both of these types of tests are run in pytest[9]. We chose pytest over regular python unit tests because there is less setup required, resulting in more time for tests being written.

9.1.1 Results

Appendix C shows the export of the coverage report for these tests. Notably, there are no misses and the coverage is 100% for everything in the back-end. The report is trimmed to only show files where there were statements found in this report, as otherwise, *__init__.py* files were also included.

9.2 Front-end

Due to time constraints the front-end is not tested in the same manner as the back-end. Although we have set up the testing suite, using Vitest for unit tests, and Cypress for end-to-end tests, we have not implemented any end-to-end tests. Our unit tests have a 100% coverage for the few front-end calculations we do.

9.3 GitHub

We have set up strict GitHub branch protection rules to ensure that only the working code reaches our main branch. To be allowed to merge code to main, it has to:

- Pass a series of linting and security vulnerability checks;
- Pass all front-end tests;
- Pass all back-end tests;
- Gain the approval of at least one reviewer.

Using GitHub actions, which run all tests in the codebase, we can ensure that code changes do not break the behavior of previously written related code. Then, the reviewer's job is to ensure that untested code does not gain an approving review. By adhering to this process for every pull request, we can guarantee the quality present in the codebase.

9.4 Edge Case Mitigation

A developer and reviewer might not always catch certain edge cases, so even if a function is tested, it might be possible that a piece of code does not work as intended. Because of this, a connection with Sentry has been set up. Sentry is a SaaS product that catches exceptions in a production environment and relates them to the code or database queries that produced the exception. While this product is not delivered as being deployed, the sentry account will be transferred to the client so they can handle edge cases unforeseen by the development team.

10 Project Evaluation

10.1 Delivered Product

At the start of the project, we defined requirements with the client of what features and functionalities they wanted their interface to have. As such, the best way to evaluate the quality of the product is to review the requirements. Appendix B specifies an overview of the product requirements. The MoSCoW methodology prioritizes them. Appendix B indicates whether the requirement has or has not been implemented, with a checkmark or a cross, respectively. As can be seen, all of the 'Must' requirements have been implemented. Of the 'Should' requirements, only one requirement has not been fully implemented:

- As a strategist, I want to be able to adjust the system to a different boat, specifically to the energy efficiency curve used to choose optimal sailing speeds, so that the system will work with new boats.

The client sent us a polynomial approximation of the efficiency curve to implement this requirement. Unfortunately, after implementing this polynomial, we noticed that the client's approximation was incorrect. The client did not have time to deliver a correct polynomial, so this requirement should be implemented in a future update. For now, we have created our calculations with hard-coded speed and efficiency values in a configuration file. These values can be easily edited if the optimal values change. The method that retrieves these values is also set up to accept an efficiency polynomial for when the team finds a correct approximation, so this connection can be made almost immediately.
Of the 'Could' requirements, only two requirements were not implemented:

- As a strategist, I want to know the confidence interval of a given strategy so I can decide whether I should follow that strategy.
- As a strategist, I want the strategy to be updated when the team makes changes to the adaptable fields.

Since no alternative strategies are calculated in this version of the product, calculating the confidence interval of a strategy is not in the scope of this project. Regarding saving changes in strategy to the database immediately, at the moment, this is done once per minute. Although it was not implemented due to our own time constraints and priorities, it is not difficult to add by the solar boat team itself.

10.2 Methodology

As mentioned in the report, the project's requirement engineering and design section initially took more time than expected. At the beginning of our project, finding the exact requirements together with the client was difficult. This could mostly be attributed to us not having access to the data. This was not the client's fault either since they were not allowed to share the data format of their motor controller as it falls under their non-disclosure agreement. We decided to use the data format of an old race, which does not fall under this NDA, instead. After that, the client can adapt the code to fit the new motor controller. Another aspect was that a large part of the time we spent on the was figuring out exactly what the project was about, what an 'optimal foiling strategy' would even entail, and how we would calculate and visualize it. There was a lot of information we had to figure out, which took us a lot of time. While our method worked well, it is hard to say whether another method worked better. Contacting the Strategist of last year's team was a good addition - he had practical experience with races where the boat was able to foil. During the co-design session, he gave valuable insights for the project.

10.3 Learning Goals

At the beginning of the project, we all defined individual learning goals that we wanted to achieve during this module as seen in section 3. In this section, we reflect on these learning goals.

Tessa

My main goal for this module was to gain more practical experience working in the front-end throughout the process of requirement engineering to practical implementation. Collecting the requirements took more time than I thought it would initially, which makes sense as it is maybe the most critical step in a project - it is important to get it right. Designing the front-end also took quite a few iterations, as I made designs whilst we were still figuring out what elements would be necessary. I feel like I have learned a lot of things during this process that I will be able to use in future projects as well. Working with other team members with more experience with the framework we were using sped up the learning process as I was able to ask them whenever I had questions. I have good understanding of the (front-end) frameworks we used.

Oliver

My learning goals mainly focused on gaining valuable practical experience in implementing robust software for clients. During the project, I spent the first weeks working on front-end components, which I am happy to say gave me a good grip on the basics of Vue and a further aptitude with HTML. After finishing those components, I connected them to the back-end, for which I needed to learn queries and mutations with GraphQL. For the rest of the project, I focused on making back-end calculations, testing them, and connecting them to other calculations and databases, in which I furthered my skills with Python. I learned a lot about how to write API and unit tests properly and what the back-end architecture of a robust product looks like.

Matthew

My learning goals were focused on data querying and processing and client communication. I got quite accustomed to working with the influx database and Flux (the database's functional data scripting language). I hope to be able to use this skill for future projects.

I first found communicating with the client quite challenging. However, because the client was approachable, I saved a lot of time by validating my data understanding and calculations with their business knowledge, which we considered the ground truth.

Lastly, I learned a lot about using Git. I feel confident with the main Git processes: merging branches, making pull requests, and giving reviews. I'm really glad I got to work with people who taught me how Git works since it is such an invaluable skill in my field.

Darrell

My learning goals were initially set around some topics that I knew were related to dev-ops engineering. In the end, I am happy to say that I learned more than initially planned. My largest contribution was that my team members could

work on their parts of the project without any major disruptions while we set a robust infrastructure for future work to improve on.

Tibet

My learning goals were focused on learning more about Vue, GraphQL, and the analysis of the data, alongside establishing better version control practices for myself. I was able to dive into all of these topics throughout this project. In the beginning, I worked on the front-end, learning more about Vue and its components. However, the part I enjoyed the most was after that; I worked on connecting the back-end and front-end through GraphQL queries(mostly) and mutations. Lastly, I moved on to the calculations on the back-end. Even though I initially wanted to get involved in the complete analysis of the data, I changed my mind throughout the project. I focused on the data analysis only relevant to the calculations. I believe this was a good decision, as I initially wanted to focus mostly on the back-end, not InfluxDB, and we needed more time on the calculations than the analysis of the data in the end. Throughout the project, I learned as much as possible from my peers regarding better VC practices. The main two that I learned are using the command line for git commands and using PRs. I am quite happy about my progress in general.

10.4 Future development

We have developed a piece of software for Solar Boat Twente, which shows them an intuitive interface with important information and projections as described in Appendix E so that they can make real-time, informed strategy decisions. The features described in this document is described as-is, and the project group will only help the solarboat team with general questions and deployment in the future. However, more features can be added to account for more external factors and to consider potential alternative strategies. Some possible future endeavors are described below.

10.4.1 Efficiency curve integration

When hydrofoils go above a certain speed, the forces on the foil of the boat cause the boat to lift out of the water. Due to the reduced water friction on the hull, the power efficiency of the boat at these high speeds is improved as shown in the graph in figure 11. As is evident in this figure, there is a 'sweet spot' where the boat is way more efficient at high speeds than at other speeds.

Currently, the program uses an optimal 'foiling speed' and an optimal 'charging speed' of 22 km/h and 6 km/h, respectively, as given to us by Solar Boat Twente. However, whether these speeds are precisely optimal is taken for granted for now and can change with different iterations of the boat. For a future update, an integration can be made with efficiency graphs like this one so that the software can easily read and recommend the optimal foiling and charging speeds. If the efficiency graph is integrated, the system could even try

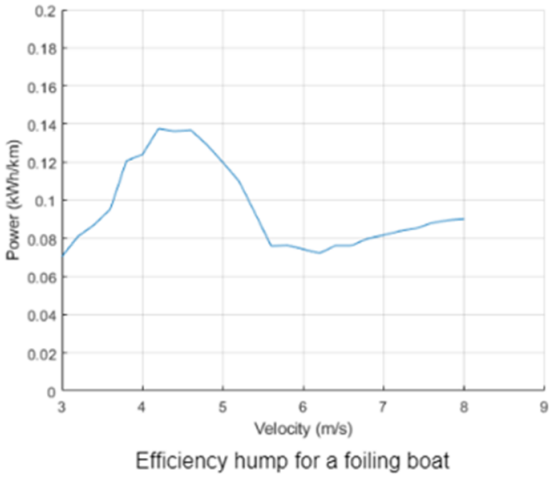

Figure 11: Efficiency curve of the solar boat, with the efficiency in kWh/km measured against the velocity in m/s

to make calculations with sub-optimal speeds to propose potential alternative strategies (as expanded in section 10.4.2) that may give better results in certain conditions.

10.4.2 Alternative strategies

Currently, the interface considers one 'best' strategy that the Solar Boat team has employed in previous years: the boat should first foil until the battery is almost empty, then it should charge its battery until it has enough battery to foil until the end of the race, and the final stretch should be spent foiling until the battery is empty. This strategy is a good way to ensure that all the energy gained is used as efficiently as possible, but it is not necessarily relevant for every race scenario. For example, there may be situations where the boat has a higher battery capacity than first anticipated close to the end of the race; in that case, it would be nice if the system suggests foiling at a higher speed than the optimum to gain more distance in total. And if weather conditions get cloudy during the originally planned charging period, it might make sense to charge for longer at the start of the race so that the boat can foil for the longest time possible given the circumstances. Future updates of the system could take the efficiency curve (as mentioned in section 10.4.1) and interesting external factors (as explained in section 10.4.3 on board to calculate a number of possible strategies to optimize the boat performance. The system would use time series forecasting to project which strategy is expected to perform better in the race conditions and would give a confidence interval with every proposed strategy.

10.4.3 Accounting for external factors

A team's performance at a solar boat race depends on the boat's quality and many external factors. The product could implement features to account for and calculate strategies based on the following external factors:

- Weather: Currently, the Solar Boat team manually inputs the expected solar energy the boat gains as an educated guess based on experiences in previous years and tests. This is obviously not entirely accurate. A weather API could be integrated into the system in the future to automate this expectation across the whole race, and strategy calculations could be made to account for at which times the sun is at its strongest or weakest during the race.
- Wave height: if waves reach a certain height, races can be finished earlier than usual due to safety reasons. If forecasts of this are available, these could be integrated with the system. Otherwise, possible strategies should at least account for a potential earlier finish of the race.

10.4.4 Miscellaneous features

This section describes a number of future features that the client expressed as possible wishes for the product but did not prioritize. These features could also be implemented in a future update.

- The application could, in the future, make strategy calculations every time a value, such as the solar power or desired battery level, is changed by the team, rather than simply calculating once every minute.
- The application could, in the future, refresh the calculations every time the user requests the system to do so.
- The application could, in the future, be made to be nicely visible on a phone.
- The application could, in the future, contain a field showing the performance of each individual solar cell so that the solar team can know when one of the solar cells is not picking up the expected amount of energy.

References

[1] E. You, "Vue," 2023. [Online]. Available: https://vuejs.org/

[2] PrimeTek, "Primevue," 2023. [Online]. Available: https://primevue.org/

[3] Y. Contributors, "Yarn package manager," 2024. [Online]. Available: https://yarnpkg.com/

[4] E. Y. . V. Contributors, "Vite," 2024. [Online]. Available: https://vitejs.dev/

[5] D. S. Foundation, "Django," 2024. [Online]. Available: https://www.djangoproject.com/

[6] S. Eustace, "Poetry," 2024. [Online]. Available: https://python-poetry.org/

[7] A. Inc., "Apollo graphql," 2024. [Online]. Available: https://www.apollographql.com/

[8] G. Chau, "Apollo composable," 2024. [Online]. Available: https://apollo.vuejs.org/

[9] H. Krekel and pytest-dev team, "pytest," 2024. [Online]. Available: https://docs.pytest.org/en/8.0.x/

A Stakeholder Analysis

Figure 12 shows the stakeholder analysis done for this project. The red circle is the most important one related to our project, as it contains the stakeholders who were critical to getting an in-depth business understanding of the solar boat team, with the strategists and system maintainers being the most important ones.

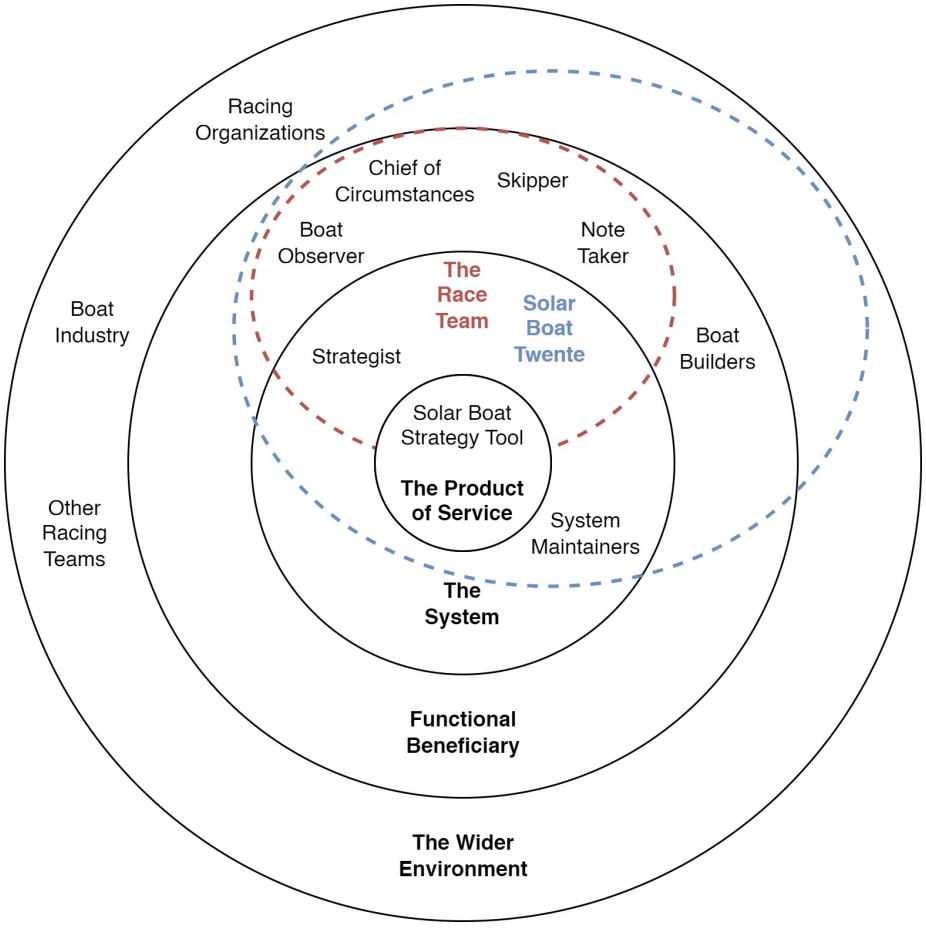

Figure 12: Stakeholder Onion Diagram for the Solar Boat Strategy Tool

B MoSCoW of requirements

B.1 Musts

- As a system maintainer, I want the system to be mainly written in Python or C++ since I am experienced in these languages and can therefore more easily maintain the system. ✔
- As a strategist, I want to know how long the boat can foil before the battery depletes so I know when I have to tell the pilot to charge the battery again. ✔
- As a strategist, I want to know how long the boat needs to charge until it can foil so I know when I can tell the pilot to foil. ✔
- As a strategist, I want to be able to tell the system how long a race is and when a race starts so that the dashboard can show how much time there is left in the race and adapt the strategy. ✔
- As a strategist, I want the interface to be clear and concise, so I can easily see all the relevant information at once and make quick decisions. ✔
- As a strategist, I want the interface to fit a desktop screen so I can view it on my laptop during a race. ✔
- As a strategist, I want the battery to never run out during the race, meaning that there is a threshold at which the system says the boat needs to stop foiling. ✔
- As a strategist, I want to be able to fill in an estimation of the amount of solar power the boat will gain during the race, and for the strategy to adapt to this amount. ✔
- As a strategist, I want to see the amount of energy in the battery in an instant, so I have an easy insight into the amount of energy the boat has left. ✔
- As a strategist, I want the system to be either an installable application or a deployable web app so that we can run it on our systems. ✔

B.2 Shoulds

- As a strategist, I want to be able to swap between the motorcontroller in use from the front-end, for the system to adapt the way the data from the database is used, as the different controllers have different fields. ✔
- As a strategist, I don't want others to be able to access the strategy dashboard so they don't have any insight into our strategy. ✔

- As a strategist, I want to be able to adjust the system to a different boat, specifically to the energy efficiency curve used to choose optimal sailing speeds, so that the system will work with new boats. ?
- As a strategist, I want to see how long it would take to charge the battery until the goal percentage, so I know how long the boat needs to not foil. ✔
- As a strategist, I want the strategy to refresh at the latest every 15 minutes and if just the excel sheet strategy is implemented every 5 seconds. ✔
- As a strategist, I want the threshold to stop foiling to be adjustable in the front-end. ✔
- As a strategist, I want to be able to change the predicted solar energy the boat will gain during the race, and for the strategy to adapt to this new value. ✔
- As a strategist, I want to see the total energy the boat has used throughout the race, based on the amount of energy the motors have used and the amount of energy the solar panels have gained. ✔
- As a strategist, after the race, I want to see how much energy was used and gained in total. ✔
- As a strategist, I want to be able to edit the amount of time left in the race, as sometimes the race gets stopped earlier, so the forecast can adapt and it is accurate. ✔
- As a strategist, I want to be able to rerun the data from the boat after the race so I can see the history of the system's predictions to evaluate racing decisions. ✔

B.3 Coulds

- As a strategist, I want to know the confidence interval of a given strategy so I can decide whether I should follow that strategy. ✖
- As a strategist, I want to input the goal of the battery charging estimation, so I can adapt the time of charging needed in between foils. ✔
- As a strategist, I want the real-time data from the boat (energy used, state of charge) to be at the latest from 5 seconds in the past. ✔
- As a strategist, I want to see the total amount of KM that the boat has travelled during the race. ✔
- As a strategist, I want the strategy to be saved on moments where the team made changes in the dashboard to the adaptable fields. ✖

B.4 Won'ts

- As a strategist, I want the system to consider future weather conditions so the strategy becomes more precise. ✖
- As a user, I want the interface to be viewable on a phone. ✖
- As a strategist, I want to be able to refresh the prediction to be of the exact time I need it. ✖
- As a strategist, I want the data from the MBPT sensors to be shown on the dashboard to see the amount of charge each individual cell (8 total) is getting, to see if one is under performing. ✖

C Backend coverage

Name	Stmts	Miss	Cover
solarboat/authentication/factories.py	11	0	100%
solarboat/authentication/schema.py	10	0	100%
solarboat/battery/capacity.py	9	0	100%
solarboat/battery/factories.py	9	0	100%
solarboat/battery/migrations/0001_initial.py	7	0	100%
solarboat/battery/migrations/0002_powerrecording.py	6	0	100%
solarboat/battery/models.py	12	0	100%
solarboat/battery/schema.py	38	0	100%
solarboat/battery/tests/test_battery_api.py	16	0	100%
solarboat/battery/tests/test_capacity.py	27	0	100%
solarboat/battery/tests/test_total_energy_gained.py	18	0	100%
solarboat/battery/tests/test_total_energy_used.py	17	0	100%
solarboat/battery/total_energy.py	12	0	100%
solarboat/distance/calculations.py	11	0	100%
solarboat/distance/migrations/0001_initial.py	7	0	100%
solarboat/distance/models.py	7	0	100%
solarboat/distance/schema.py	13	0	100%
solarboat/distance/tests/test_distance_estimated.py	12	0	100%
solarboat/distance/tests/test_distance_traveled.py	17	0	100%
solarboat/examples/test_example.py	8	0	100%
solarboat/influxdb/fakedb.py	20	0	100%
solarboat/influxdb/tests/test_fakedb.py	23	6	100%
solarboat/race/factories.py	16	0	100%
solarboat/race/migrations/0001_initial.py	7	0	100%
solarboat/race/migrations/0002_race_motor_controller_race_reference.py	4	0	100%
solarboat/race/migrations/0003_race_solar_power.py	4	0	100%
solarboat/race/migrations/0004_race_end_date_race_unique_user_day.py	6	0	100%
solarboat/race/migrations/0005_race_lower_threshold_race_upper_threshold.py	4	0	100%
solarboat/race/models.py	25	0	100%
solarboat/race/schema.py	59	0	100%
solarboat/race/tests/test_api.py	27	0	100%
solarboat/schema.py	17	0	100%
solarboat/settings.py	26	0	100%
solarboat/strategy/config.py	4	0	100%
solarboat/strategy/foil_charge_calculations.py	37	0	100%
solarboat/strategy/schema.py	53	0	100%
solarboat/strategy/tests/foil_charge_calculations_test.py	39	0	100%
solarboat/urls.py	5	0	100%
TOTAL	643	0	100%

Table 3: Backend test coverage report.

D Installation manual

The installation manuals are provided in the codebase in README files with runnable code blocks for ease of use. The text below is this readme converted into a PDF. Refer to the codebase instead for a better reading experience and help to install the stack.

How to start the backend

N.B. If you are looking for an installation guide, it is in the bottom of this file! ## Running the backend **N.B.** if you are working in a poetry shell, `poetry run` is not needed in these commands.

Make sure all your packages are up-to-date

```
poetry install
```

Run the postgres and influxdb

```
docker compose up -d
poetry run python manage.py influx
```

Populate both databases

```
poetry run python manage.py seed
```

Now run the server with

```
poetry run python manage.py runserver
```

Installing the backend

Tools needed

Refer to the official documentation and installers for the following tools:
`Python` - version >=3.11
`Python Poetry` - Link to documentation
`Docker compose` - Link to documentation

Poetry shell

We recommend you to use a poetry shell, which runs in a virtual environment. This way all changes you make, will only apply to the project. If you are using PyCharm, go to
`file > settings > project > python interpreter > add new interpreter > poetry environment`
By selecting your poetry and python executables, pycharm will automatically set up a poetry shell in your terminal.
If you are using a different IDE, refer to the documentation.

Automated Linting

To be allowed to merge your commits, a series of linting checks have to be passed. Installing pre-commit makes sure that you pass these checks by doing the same checks locally. You can install pre-commit by running the following in your poetry shell.

```
pip install pre-commit
```

How to start the frontend

N.B. If you are looking for an installation guide, it is in the bottom of this file!

Running the frontend

Make sure all your packages are up-to-date

```
yarn install
```

Now run the server with

```
yarn dev
```

Installing the frontend

Tools needed

Refer to the official documentation and installers for the following tools:
`NodeJS` - version >=18
`Yarn` - Link to documentation

Automated Linting

The frontend also has automated linting. For the installation of this component, read the README in the backend folder! It will work simultaneously.

Your frontend is now fully installed, refer to the top of the document to start it!

E Product manual

The Solar Boat Twente strategy interface will help the user make decisions justified by incoming data during a Solar Boat race. This manual serves to explain how to set up and use the interface so that strategy decisions to optimize performance can be made by the user in real-time.

E.1 Setup

Upon starting the application, the user needs to authenticate and specify a few conditions of the application before the interface can be properly displayed.

E.1.1 Logging in

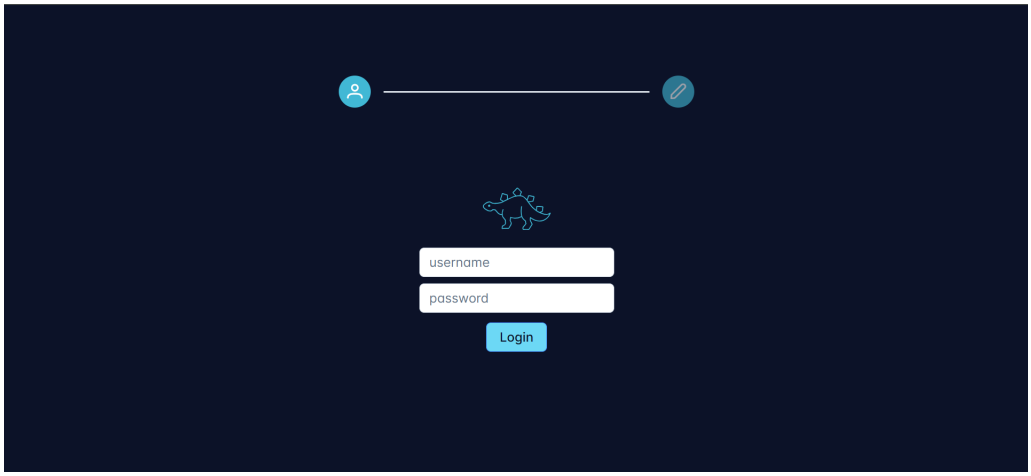

Figure 13: The strategy tool authentication page

First, the user must log in with a username and password, as seen in figure 13. Once the 'login' button is pressed, the application shows the next screen if the username and password are correct.

E.1.2 Setting race conditions

Once the user is authenticated, the original race conditions must be set, as seen in figure 14:

- In the "Race title" text field, a name for the race must be set.
- The motor controller which will be used during the race must be selected. This can be either Jasmine or Vesc.
- The duration of the race in hours, minutes and seconds must be set.

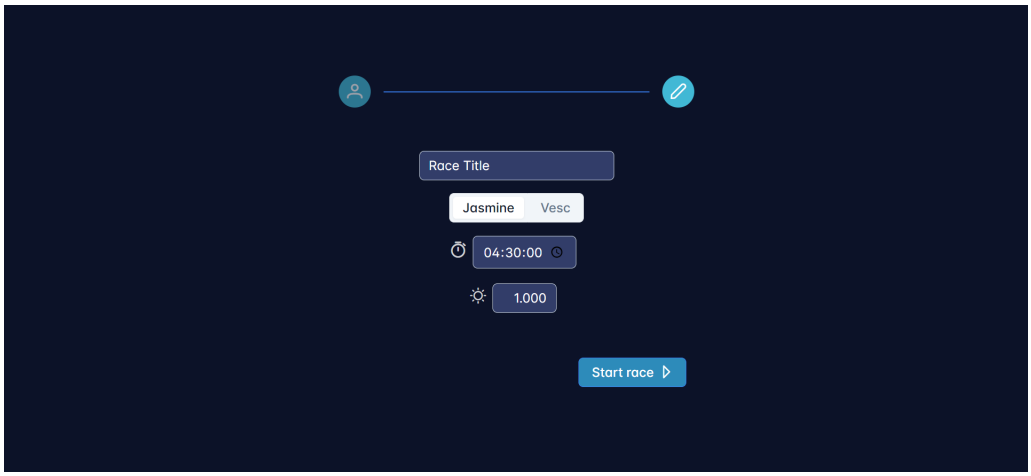

Figure 14: The race setup page

- The user should specify the estimated amount of solar energy in W that they expect to gather from the sun during the race.

Once the conditions have been set, press the 'Start race' button at the bottom of the screen. If the formats of all the input fields are valid, the race interface will be shown. The race timer will immediately start running when the 'Start race' button is pressed, so press this button exactly when the starting shot is fired!

E.2 Race interface

Once the original race conditions have been set, the race interface will be shown, as seen in figure 15. The race interface is divided into three components: a general race information display, a graph projecting the battery's state of charge, and an overview of the remaining foiling and charging times the team should employ during the race. The interface makes calculations based on real-time data once every minute, and updates the interface accordingly. In the following sections, every component of the interface will be explained.

E.2.1 General information display

In the top left of the interface, the important general information of the race is displayed. This consists of the following components:

- A battery displaying the current boat's battery level in Wh. The red line shows the minimal battery level in Wh that the user wants the boat to have. The green line shows the maximal battery level in Wh that the user wants the boat to contain. Either of these thresholds can be changed: to change these, press the battery. In the popup, change either of the values

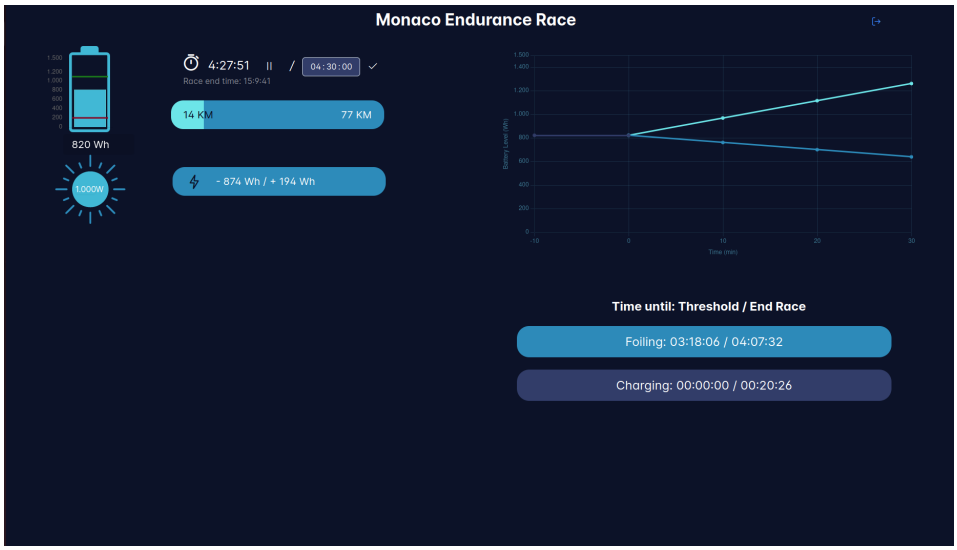

Figure 15: The race interface

to the desired value in Wh, and save the changes. The next calculations will include the adjusted threshold values.

- A race timer showing the remaining time of the race and the calculated end time of the race. The timer immediately starts running when the interface is created. If the race is paused, press the pause button to stop the timer counting. When the race is resumed, press the play button again, and the timer will resume with the end time updated accordingly. If the end time of the race is changed, fill in the remaining race time in the input field to the right of the timer and press the check mark; the timer will be adjusted accordingly.

- A display of the solar power in W that the boat is expected to collect. This value is equal to the value input upon setting the original race conditions. This value can be changed by clicking on the sun icon, and adjusting the value to the new expected value in W.

- A display of the distance travelled so far in kilometres and the expected distance in kilometres that will be travelled from now until the end of the race.

- A display of the energy in Wh used by the boat so far, and a display of the energy in Wh gained by the solar panels so far.

E.2.2 State-of-charge graph

The top right component shows a graph with the time in minutes on the x-axis, and the battery level in Wh on the y-axis. The trend in battery level from 10 minutes in the past until now is shown. Then, for the next half an hour into the future, two lines are shown with two possible projections: the projection of the battery level if the boat foils for the next half an hour, and the projection of the battery level if the boat charges for the next half an hour.

E.2.3 Foiling-charging overview

The bottom right component gives an overview of the amount of foiling and charging that can still be done in the race. It is divided into two sections: foiling and charging.

- The left-hand side of the foiling section shows the amount of time that the boat can still foil until it reaches the critical battery level, as indicated with the red line on the battery level display. The right hand side of the foiling section shows the calculated time that the boat is able to foil until the end of the race.
- The left-hand side of the charging section shows the amount of time that the boat can still charge until it reaches the desired battery level, as indicated with the green line on the battery level display. The right hand side of the charging section shows the calculated time that the boat has to charge until the end of the race.

Note that the right hand sides of both sections added together make up the total remaining race time. So, if the charging time until the end of the race is zero, it will be possible to foil until the end of the race.

List of Figures

List of Tables