Online Platform for Virtual Electric Vehicle Grand Prix

The Process of Improving a Virtual Race Application

Andy Li Bas Filius Vladimir Rullens Maroje Luka Borsic
Jan Domhof

June 20, 2022

TU Coach: Stefan Hugtenburg
TA: Naval Tapan Bhagat
Client: Global EEE

Preface

In this report, the team would like to describe the hard work, process and results of our group over the duration of the Software Project as students of the Technical University of Delft (TU Delft). The field of Computer Science is a field with limitless options and paths to choose from. As a group of Bachelor Students from TU Delft, the group has been given the opportunity to choose one of these paths through a ten-week project called the Software Project.

The team has chosen the project ”Online Platform for Virtual Electric Vehicle Grand Prix” provided by the client, Global EEE. What makes this project stand out from the others is the interesting topic that the project covers and the knowledge the group can gain by participating in this project. Most of the group members have shown interest previously in both racing car competitions, but also learning new frameworks and programming languages such as React, Laravel and PHP which are used in this project. Any computer scientist should be interested in these languages and frameworks as they have proven to be the future of technology, being the basis for some of the most popular websites in the world, such as Instagram, Netflix and DropBox (C. Brewster, 2022.). Readers interested in the topic of programming languages can find more about the use of programming languages in this project in section 4 of the report. Readers interested in the topic of ethics can find more about the implications of ethics in our project in section 7 of the report. Future groups who plan to work on this project might be interested in the process of creating the improved application and can find more about this process in sections 4 and 5 of the report. The group has also created recommendations for future groups regarding this project that can be found in section 8 in subsection 8.2.3.

By completing this project, the team hopes to impress their client with the new implementations and functions added to the application, while also allowing a better experience for all users of this platform. By reading this report, the team gives an insight to the audience to have an insight into the background of the project, what were the initial problems, how the team planned to approach these problems, what the outcomes were and what can be concluded from this project. Because of the complexity of the topic, this report has been written for people with a background in Computer Science and people who plan to manage this platform in the future in mind. The report might be especially useful to future development teams who will work on the application.

We hope that through this report, the audience can appreciate all of the hard work and efforts put into the project. Finally, we would like to give a special thank you to the Teaching Assistant Ms. Bhagat, the coach Mr. Hugtenburg, the client Mr. Bedewi and the previous developer Mr. Peterse for all their support and guidance during the project.

Delft, 08 June 2022
Andy Li, Bas Filius, Vladimir Rullens, Maroje Luka Borsic and Jan Domhof

Summary

In this project, an application was improved by a group of students who cooperated with the client, Global EEE. Global EEE is a non-profit organization that used to host a collegiate competition between students, challenging their technological, innovative and engineering skills by designing, building and racing high-performance formula-style electric vehicles. Due to the outbreak of the pandemic, Global EEE was forced to switch to an online alternative which is now called the Virtual Electric Vehicle Grand Prix Platform or VEVGPP for short. The original application or version of the VEVGPP was developed by an outside developer.

The goal of our Software Project was to improve the VEVGPP in such a fashion that the application is more user-friendly with more functionalities and better maintainability for the future. These functionalities include a new spectator mode for both the admin and the public, an improved admin view, and the handling of unsaved data during the race. Certain parts of the codebase have been refactored. This is because components were too intertwined, which made it hard to test for existing bugs in the application. With the refactor, tests have been written for the newly implemented portion of the code. The new codebase is better structured, contains fewer bugs and is more user-friendly. Through regular meetings and the midterm presentation, the client has confirmed to be satisfied with the current improvements and has approved of the plans for moving on and using the new version of the application for future competitions. With this new version of the application, GlobalEEE will be able to host races where the flow of the race will not be broken due to a bug or lack of features, enabling a better quality of life experiences for both the users and admins of the application. The application now serves the purpose of a substitute for real-life competitions, but also as practice for users to learn and prepare for the physical race event.

The project included both front-end (client) and back-end (server) code; i.e. full-stack development. The proof-of-concept development included implementation of JavaScript, React (front-end framework), PHP, Laravel (back-end framework) and D3 (JavaScript library for visualization of tracks, data displays, etc.). The group has learned a lot regarding the use of these languages and frameworks, but also about working in a more team-oriented environment. The group has also come up with propositions for the client if the application was to be further developed in the future. Some of the recommendations include refactoring a majority of the codebase, documenting the codebase to a more extensive proportion and focusing more on the testability of the current codebase rather than focusing on more new implementations.

Contents

1 Introduction

Ever since the outbreak of the COVID-19 pandemic, at least 5800 businesses have been forced to switch to online alternatives (Bartik et al., 2020) to which Global EEE is no exception (Global EEE, 2022). Global EEE is a non-profit organization that hosts a collegiate competition between students, challenging their technological, innovative and engineering skills by designing, building and racing high-performance formula style electric vehicles. However, due to the pandemic the focus of the organization has transitioned to an online alternative called "Virtual Electric Vehicle Grand Prix Platform" or VEVGPP for short. Global EEE tasked an outside developer to create an initial version of VEVGPP and the application is up and running. However, the application is filled with bugs and has a lot of room for improvement. This is where our software project comes into play.

The goal of this platform is to hold physics-based simulations, allowing students to configure race cars, go through technical inspections and race head-to-head on the track. As the company has set high standards during the in-person Grand Prix's and has many dedicated supporters, it is important for the company to live up to the high expectations. As of this moment, the application is filled with bugs, it contains large files that are not modular and the User Interface (UI) might be confusing for first-time users. The goal of our project is to extend the current version of the platform and deliver a well-structured, bug-free, user-friendly application with the necessary requirements suitable to use for students and satisfying the demands of our client. The requirements include having a landing page, integrating the technical inspection questionnaire page and having multiple racetracks available for races. It must be kept in mind that in the future, the client would like to use the application as a form of test run to make sure that the students are prepared to step into real-world racing and have the necessary knowledge to care about both their own and other students' safety.

On the other hand, the goal of the report is to show that research was done extensively, all limitations were identified, possible solutions were proposed and relevant results and conclusions regarding the goal were formulated. Here it was decided that it would be best to use techniques such as hashing out the requirements and using the MoSCoW prioritization technique as the group was most familiar with these. Also, as much as possible was done to get familiar with the current application, such as researching the rules and regulations of the Grand Prix and hosting a Grand Prix using the current application in order to spot any flaws and limitations as early as possible.

The report is presented in the following structure. Chapter 2 will provide the analysis of our problem and talk about what we think will be our greatest limitations and what will be the final goals for this project. Chapter 3 will contain all about the approach towards the requirements elicitation and further elaboration on the use of the MoSCoW prioritization method. Chapter 4 describes the overall approach to the project itself and goes into more detail about the specifics and technicalities of the VEVGPP. Chapter 7 will go in depth regarding the moral implications of the use of VEVGPP by minors. Finally, in Chapter 8 the results, conclusions and key takeaways regarding the project will be laid out.

2 Problem Analysis: Improving an Existing Race Application

Improving an application can be difficult, especially when there are limitations due to the design of the application. This chapter will analyze potential problems when improving the existing race application. Chapter 2.1 describes the problems that came up, considering the stakeholders of the application. Therefore, the goals of the project based on those problems, will be elaborated on in chapter 2.2.

2.1 Problem Statement: Two Potential Problems of Improving VEVGPP

In order to state any potential problems, the stakeholders of the application need to be considered. The stakeholders for this project include the client (Global EEE), the users (high-school students participating in the race) and the sponsors of the electrical vehicle race event. After the problem analysis, the group found that the client's problems could be categorized into two parts.

Firstly, after a few test sessions, it became apparent that the initial version of the VEVGPP was full of bugs that hindered the racing experience. The client also told the group about the process of hosting and monitoring a race. This process contained a lot of limitations and excessive work. The client used to host video calls with all participants and needed to slowly switch between them if they needed help or if the client wanted to check on them. Furthermore, the client did not have direct access to the back-end of the application, where a lot of helpful information is stored.

Another problem that arose was that the code quality of the code base was not up to standards. The code base mainly consisted of some code files that were quite large and complex with no existing documentation for the code base. Having such complex and large code files means that the code base is far from modular. Many methods and functions were intertwined and interdependent, leading to lots of confusion and issues when both refactoring and documenting the application. Furthermore, there were no tests written for the entire code base. Because of that, there was no way of measuring the reliability of the application. One of the wishes of the client was for the application to be developed in the coming years. To make it easier and faster for groups to pick up the project in the future, the code base needs to be made future-proof.

2.2 Project Goals for Improving VEVGPP

Improving an application is a tough challenge, since it requires a deep understanding of the existing code. Only then, is it possible to know what to improve or even if improvements can be made. The client, Global EEE, desires improvements to the existing application (VEVGPP) whilst adding new features. However, this needs to result in a functioning application ready for future development. The two project goals for improving VEVGPP are the following:

First and foremost, a new iteration of the application needs to be developed. This new iteration needs to be fully functional and have at least the same functionalities as the current iteration extended with new features desired by the client. It also needs to be easily extendable, such that future developers have a smooth experience of understanding the code and can easily add new functionalities. The best way to take this into account is with code smells. With code smells, it can be seen what code might need refactoring or can be improved by standard metrics. However, according to a study regarding maintenance of code (Sjoberg et al., 2013), to reduce maintenance effort, a focus on reducing code size and the work practises that limit the number of changes may be more beneficial than refactoring code smells. The new iteration made by us is aimed at being fully implemented and usable by the end of the project.

Furthermore, a questionnaire designed by an external developer was planned to be integrated as one of the client requirements. However, the finalization of the questionnaire has run into some problems and the external developer is not able to share a working version of the questionnaire, resulting in this requirement not being viable to implement.

3 Client Requirements for Improving VEVGPP

When talking about software development, requirements elicitation is the process of extracting and researching the requirements of a system or application from either the client, user, or any alternative stakeholder. It is very important to carry out this process, which will be laid out in this chapter, before starting to code and implement any new features. In chapter 3.1 the requirement elicitation procedure will be discussed. The approach to prioritizing the requirements will be discussed in chapter 3.2.

3.1 Requirement Elicitation Procedure

Since there was already an application, understanding its purpose as a participant was the first step of the requirement elicitation procedure. This first step was done by getting familiar with the rules of the physical race event. This gave insight as to how the application functions, what goes through the mind of a race participant and what problems might arise during a race.

All this insight could be useful later when implementing our functions, since we would then be mindful of the users' point of view and we would know how to go about handling it. We made sure to spend time both in the practice and race modes in order to be able to distinguish the differences and what is expected from each view. After the procedure, we got introduced to the rules of the application which simulates the physical event as well as the rules which will come in handy. This is because some of the issues are connected to the ruling aspect of the race, so race knowledge will be needed to implement these requirements. Then, we got to experience the racing application as racers ourselves. Finally, we got to run through the code together with the client to see the current structure and logic of the entire code base. This helped us visualize what we were expected to research in the following weeks, how the code base communicates internally, how we are going to set-up our working environments etc.

During this process, we naturally came across some bugs or missing features we experienced as participants. As a solution to this problem, we decided to use the MoSCoW prioritization technique, which will be further discussed in Chapter 3.2.

3.2 Using the MoSCoW Method

We have chosen the MoSCoW framework, which separates requested features into four categories based on importance. Namely, 'must-haves', 'should-haves', 'could-haves' and 'won't-haves' This is utilized with the SCRUM methodology, which focuses on periodic planning of what has been done, and still needs to be done. We chose the MoSCoW prioritization method as it is the most familiar to us, is known for its rapid application development and is one of the best methods to use when working on a project with a fixed deadline, such as in our case. It is often used in agile software development. For example, SCRUM, which will be more touched upon in the processes chapter found at 6.2, is the approach our group has taken since we are familiar with it, and we think we can be most effective with this approach.

Through regular meetings and communication with our client, we were able to create a list of requirements using the MoSCoW technique as seen in Appendix B. To ensure that the client and we, as the developers, are on the same page regarding the (priority of the) requirements, we had weekly meetings with the client to discuss our progress and if we possibly needed to shift our focus and attention on different requirements. Going over the MoSCoW method: In our case, the 'must-have-requirements' are features that are a necessity to our client. Without these features the application would be deemed incomplete, hence these requirements are our primary goal in this project. The 'should-have-requirements' are features our client believes will improve the functionality of the overall application.

We then have the 'could-have-requirements' whose main purpose is to improve the quality of the user experience. These are nice to have but are viewed as more of a bonus to the application. Since our client expects the development of this software to continue in the future, it is our goal to take into consideration future-proofing the current application in a way that will allow the implementation of the could have requirements in the future. Finally, the 'won't-have-requirements' are seen as requirements that are either outside of the scope of the project or our expertise and will therefore not be implemented.

4 Design of VEVGPP

This chapter will describe the several distinct elements that will need to be implemented or changed within the existing code base. In chapter 4.1 there is an explanation regarding the languages used and their reasoning. In chapter 4.2 insight is given regarding the client-server connection and concurrent logins, after which 5.1 will go over the new functionality which is present in our application. Finally, 4.4 will go over the way code refactoring and documentation have been handled.

4.1 Programming Languages Used

Due to the fact we will be working with an already existing code base, we will be utilizing languages that are already in use, in this case: HTML, JavaScript (utilizing React) and PHP (utilizing Laravel). React is currently the most used JavaScript library (McKeachie, 2021), while Laravel is the most used PHP framework (Sagar, 2019), resulting in a large number of learning sources and support, which covers our lack of knowledge and experience with these tools. The question may be raised of whether or not these programming languages are the best for this application, especially with React having some close contenders. On React's side, React focuses on a more stable and secure performance by sacrificing speed, which cannot be said about other frameworks (Xu, 2021). Since fast decision-making is not too important for these races, but a stable performance is, we believe React to be the most suitable choice. On Laravel's side, there is one other potentially interesting framework named Symfony. It allows for more scalability through modularity and does not have nearly as big of a risk as Laravel when it comes down to code-breaking after an update, but it is harder to learn and does not contain as reusable code as Laravel (Asper Brothers, 2019). Whether or not Symfony is better than Laravel depends on the future of this application and its developers, as Symfony might be a better option if the application becomes a lot larger with developers who stay with the application longer.

4.2 VEVGPP Client-Server Communication

One of the aspects of the current implementation of VEVGPP that needs improvement is the communication between server and client, as well as what is stored, handled and computed where.

First there are concurrent logins, where if an account has recently (i.e., within the last two seconds) contacted the server, it will be deemed in use. Whenever a user attempts to log into an in-use account or access the race page of one, this action will now take precedence over the old login. Meaning, the old user gets logged out automatically.

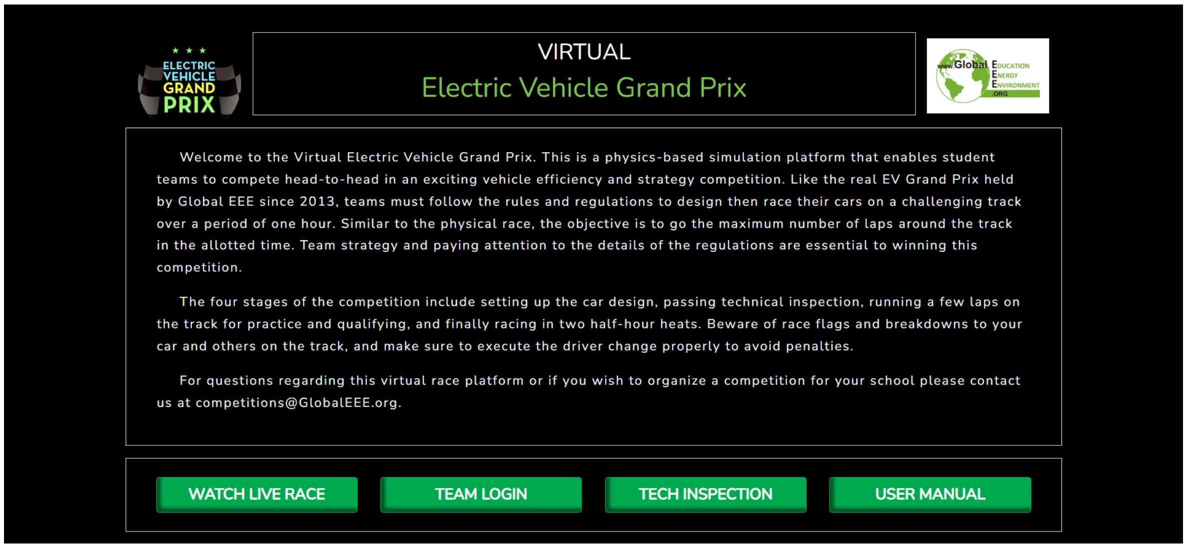
A continuous stable connection between client and server cannot be guaranteed. Therefore, the client will attempt to keep the server updated with the latest information that it has available. When reconnecting, the player will retrieve the data back from the server after which their client will calculate and thereby update their current position, which the server will then once again obtain.

4.3 New Functionality for VEVGPP

In addition to the client-server changes, VEVGPP is in need of some new functionality which needs to be added to the application. In this chapter these features will be discussed. First the landing page will be discussed in 4.3.1, followed by the track-editor in 4.3.2 and the spectator mode in 4.3.3. Then, the feature of textual description for unclickable buttons will be discussed in 4.3.5 and finally, the race settings saving feature upon refresh will be discussed in 4.3.4. See Appendix A and Appendix B for an exhaustive list of requirements.

4.3.1 VEVGPP Landing Page

One of the features requested is a new landing page for VEVGPP, which can be found as number eleven in Appendix B. In the previous iteration of VEVGPP, the start of the application redirects users to the login page. There needs to be a page that welcomes the user and where the user can navigate to different sections of the website, such as the spectator mode to view active races, which will be further described in chapter 4.3.3. The user also needs to be able to navigate to the login page. As well as to the technical inspection page, where participants answer a questionnaire before the race. Finally, there needs to be a button where users can find the race manual, which describes the rules of the race. A suitable landing page already exists, which serves as a basic template for the improved landing page. Below, the figure can be found with the updated landing page.


VIRTUAL

Electric Vehicle Grand Prix

Welcome to the Virtual Electric Vehicle Grand Prix. This is a physics-based simulation platform that enables student teams to compete head-to-head in an exciting vehicle efficiency and strategy competition. Like the real EV Grand Prix held by Global EEE since 2013, teams must follow the rules and regulations to design then race their cars on a challenging track over a period of one hour. Similar to the physical race, the objective is to go the maximum number of laps around the track in the allotted time. Team strategy and paying attention to the details of the regulations are essential to winning this competition.

The four stages of the competition include setting up the car design, passing technical inspection, running a few laps on the track for practice and qualifying, and finally racing in two half-hour heats. Beware of race flags and breakdowns to your car and others on the track, and make sure to execute the driver change properly to avoid penalties.

For questions regarding this virtual race platform or if you wish to organize a competition for your school please contact us at competitions@GlobalEEE.org.

Figure 1: The updated landing page for VEVGPP.

As can be seen in the figure, the user gets welcomed with text describing the purpose of the application. The user also sees four green buttons which they can use to navigate to different sections of the website.

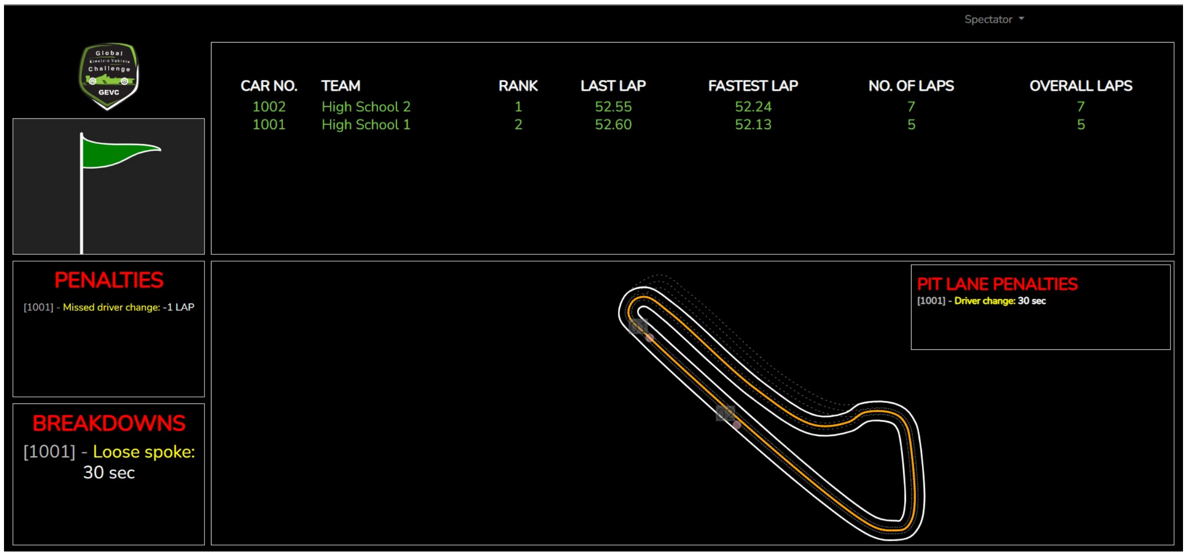
4.3.2 VEVGPP Track Editor

The application's tracks are currently hard coded using arrays of 2D-coordinates. To make this more user-friendly, we will use D3: a JavaScript library whose primary function is to manipulate documents based on data (Bostock, 2021). For our purpose, D3 has a function which allows you to draw lines, or create curves using data points. These data points can then be connected to an element which can be manipulated utilizing a drag and drop functionality (Pereira, 2018). The data points of these lines and curves can then be used to convert the created D3 representation of the racetrack into a 2D array of coordinates which are used by the application. This conversion can also be done backwards, by letting the application create these texture elements utilizing the stored 2D array. The track editor could also be used to import tracks. Since one of the could-have requirements is to add four new tracks to the application. The client might find this import feature useful as they can utilize it themselves. There is already a demo showcasing the way these curves could be edited (d3indepth, 2019).

4.3.3 VEVGPP Spectator Mode

Within the application there exist two types of spectator modes, one for the admin and one for the public to view active races. For the admin spectator mode, the host of the event, who will act as the admin, is able to view screens of players within the VEVGPP. This requires an important thing: there needs to be an interactive element of the UI which 'sends' the host to a certain player's screen. This view will retrieve the most up-to-date information from the server regarding the screen of the player and display it to the host. The interactive UI has been achieved through the React library, which will then result in information being retrieved from the server through PHP. This information is then used either directly or in calculations for all elements on the screen.

Furthermore, there is the public spectator mode, which is open to the public. This means that no account is needed for VEVGPP for the use of this mode. It is as simple as clicking the button "Watch Live Race" on the landing page, which can be found in figure 1. Below, the figure can be found when pressed on the "Watch Live Race" button.



CAR NO.	TEAM	RANK	LAST LAP	FASTEST LAP	NO. OF LAPS	OVERALL LAPS
1002	High School 2	1	52.55	52.24	7	7
1001	High School 1	2	52.60	52.13	5	5

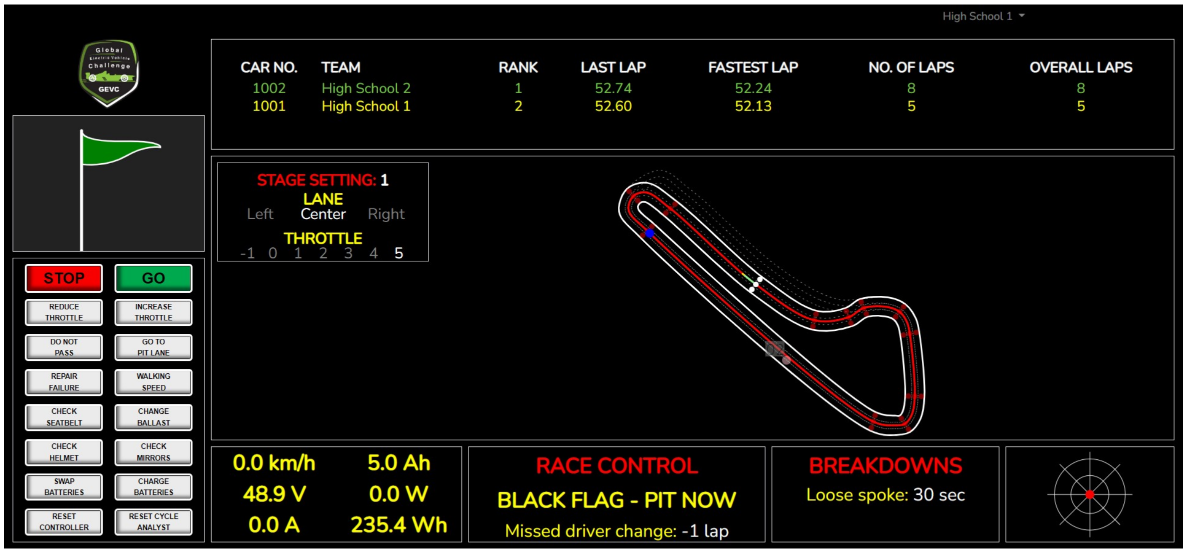
PENALTIES
[1001] - Missed driver change: -1 LAP

BREAKDOWNS
[1001] - Loose spoke: 30 sec

PIT LANE PENALTIES
[1001] - Driver change: 30 sec

Figure 2: The public spectator view for VEVGPP.

As can be seen in the figure, the user gets to see the 'spectator' view. They are able to see the scoreboard with the race participants and their (total) laps, current rank and lap times. They are also able to see the penalties and potential breakdowns. For comparison, the figure can be found below for when the user has logged in via the "Team Login" button.



CAR NO.	TEAM	RANK	LAST LAP	FASTEST LAP	NO. OF LAPS	OVERALL LAPS
1002	High School 2	1	52.74	52.24	8	8
1001	High School 1	2	52.60	52.13	5	5

STAGE SETTING: 1
LANE
Left Center Right
THROTTLE
-1 0 1 2 3 4 5

0.0 km/h 5.0 Ah
48.9 V 0.0 W
0.0 A 235.4 Wh

RACE CONTROL
BLACK FLAG - PIT NOW
Missed driver change: -1 lap

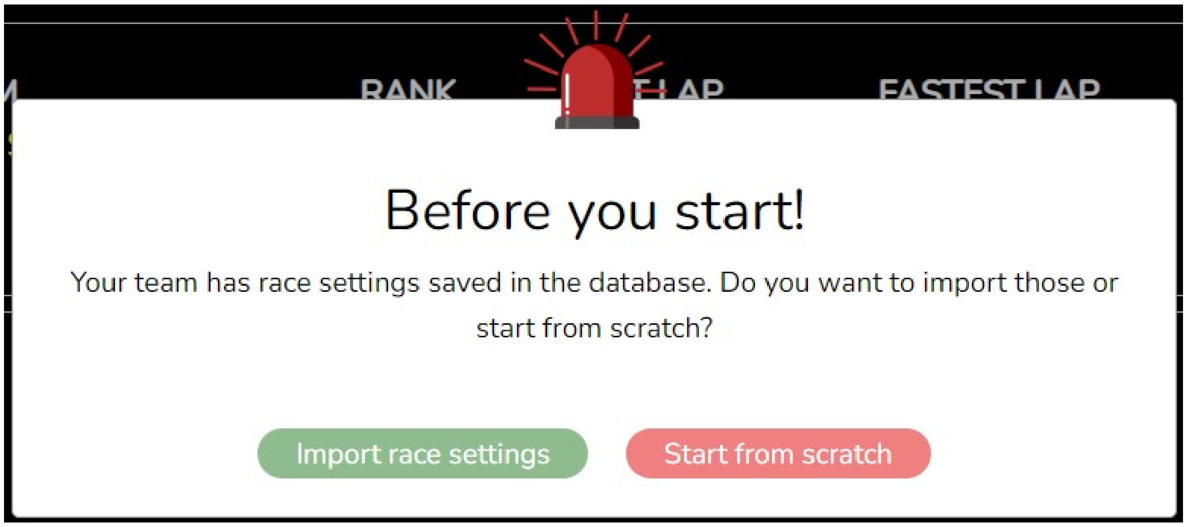
BREAKDOWNS
Loose spoke: 30 sec

Figure 3: The race participant view for VEVGPP.

As can be seen in the figure, the user gets to see the 'race participant' view. They are able to see the scoreboard with the race participants (including themselves) and their (total) laps, current rank and lap times. They are also able to see their own penalties and potential breakdowns. Furthermore, on the left-hand-side, the control buttons for the race car can be found. As well as the car physics information on the bottom left.

4.3.4 Race Settings Saved Upon Refresh of Webpage

Another problem for race participants was that their race settings were reset when disconnected or when refreshing the page. The race settings were stored only on the client, so if a user lost connection to the internet, accidentally refreshed their page or their browser froze, it would result in their race settings being lost. They would need to go over their race line configurations once more and set the throttle correctly for every section of the race track. This is not the worst problem during a practice session, but if this happens during a race, it could cost the participating team valuable minutes setting everything up again. The new version of VEVGPP now stores the race settings, as shown below.



Before you start!

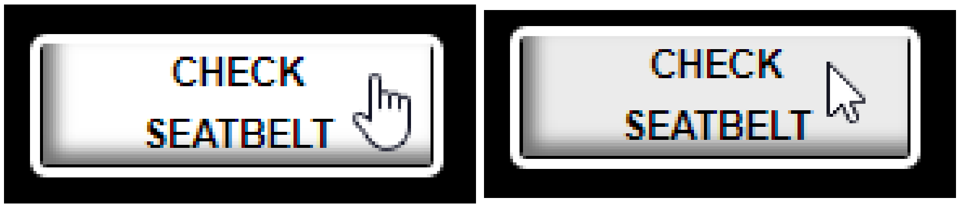
Your team has race settings saved in the database. Do you want to import those or start from scratch?

Figure 4: Race settings popup.

When the user joins the race, they get a popup asking them if they want to import the race settings that were stored, or if they want to start from scratch. If they click 'Import race settings', they can continue from where they left off, and if they want to start over, they can click 'Start from scratch'.

4.3.5 Textual Description for Unclickable Buttons

A common problem we had as participants, during the requirement elicitation process, was the fact that we did not always know the reason behind buttons being unclickable. We discussed this problem with the client and out of that conversation came 'should-have-requirement' number four, which can be found in Appendix B. Below the figures can be found for when hovering over a button.



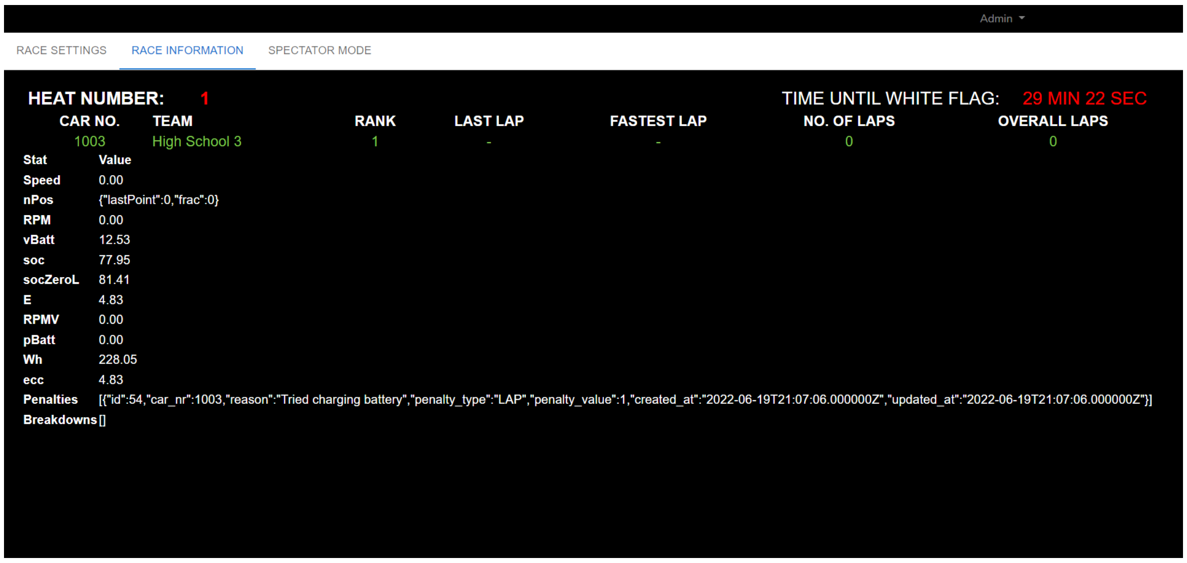
(a) When the button is clickable. (b) When the button is unclickable.

Figure 5: The 'check seatbelt' button when hovered over with mouse.

In figure 5a, it can be seen that the button is clickable due to the mouse being a hand icon, indicating that the button is clickable. Whereas in figure 5b, it can be seen that the button is unclickable due to the mouse being a pointer icon, but what is missing, is the reason for the button being unclickable.

4.3.6 Administrator View of the Race Details

Originally, there was no way for the administrator to easily investigate problems during the race. The administrator view only contained one page for making API requests to control certain aspects of the race. To give the administrator more information about the race without needing to switch between spectating multiple participants, the administrator page was extended with two tabs, a spectator mode as mentioned in 4.3.3, and a tab called race information, as shown in the figure below 6. In this tab, the administrator can select a participant and receive in-depth race details. These race details contain information that normally would not be visible. This includes statistics like the 'state of charge' of a battery, which means the percentage of charge left in the battery. This information was previously only available through the back-end of the application. The administrator can also see the penalties and the breakdowns of all the participants, including the time stamps to give the administrator a more in-depth look at the situation of a participant. The race information page contains a lot of rapidly changing numbers. Therefore, the choice was made to keep the page as simple as possible to not be overwhelming.



HEAT NUMBER: 1

TIME UNTIL WHITE FLAG: 29 MIN 22 SEC

CAR NO.	TEAM	RANK	LAST LAP	FASTEST LAP	NO. OF LAPS	OVERALL LAPS
1003	High School 3	1	-	-	0	0

Stat	Value
Speed	0.00
nPos	{"lastPoint":0,"frac":0}
RPM	0.00
vBatt	12.53
soc	77.95
socZeroL	81.41
E	4.83
RPMV	0.00
pBatt	0.00
Wh	228.05
ecc	4.83
Penalties	[{"id":54,"car_nr":1003,"reason":"Tried charging battery","penalty_type":"LAP","penalty_value":1,"created_at":"2022-06-19T21:07:06.000000Z","updated_at":"2022-06-19T21:07:06.000000Z"}]
Breakdowns	[]

Figure 6: The administrator view of the race details.

4.4 Code Refactoring and Documentation of VEVGPP

The current codebase requires some heavy restructuring and refactoring to make the code maintainable, testable and organizable. The code base currently has large JavaScript files which need to be refactored into smaller pieces. On top of this, the code base needs to be refactored to improve, or sometimes even allow testability. For this we will be using the free version of Better Code Hub, which is an open-source web-based source code analysis service that supports both JavaScript and PHP (Software Improvement Group, n.d.), which will give us insight into the above-mentioned issues, after which we will be able to add tests for JavaScript using Jest and tests for PHP using Codeception, these frameworks include both unit and integration tests, while also supporting React and Laravel respectively (Facebook, Inc., 2017; Codeception, n.d.).

In order for future developers to get accustomed to the code base in a timely manner, documentation has been added in three forms. First, there is the 'Readme' file. This file has been expanded by introducing the testing frameworks within the application and how they can be run, as well as a mention of a batch script which has been added to allow developers to immediately run the four terminals in the background which had to be run manually previously. The getting started section now also supports windows computers, as some elements did not yet have windows counterparts. Next, there is the 'developerTips' file which mentions elements that took us time to understand. This includes an overview of when all the interactive buttons during the race are available or not. Finally, there is the 'fileNavigation' file which gives a short explanation of what is contained per file.

5 Implementation of VEVGPP

The previous chapter discussed the design of VEVGPP with a general description of newly added features, this chapter will dive into the details of the implementation of the new features. Implementation details for the new functionality of VEVGPP can be found in chapter 5.1.

5.1 Implementation Details for the New Functionality of VEVGPP

As read in the previous chapter, VEVGPP was in need of new functionality which had to be added to the application. This chapter will describe the implementation details. First, the landing page will be discussed in 5.1.1. Secondly, the track-editor in 5.1.2 and the spectator mode in 5.1.3. Followed by the feature of textual descriptions for unclickable buttons will be discussed in 5.1.5. Then, the race settings saving feature upon refresh will be discussed in 5.1.4. Finally, the addition of extra race details to the administrator view in 5.1.6.

5.1.1 VEVGPP Landing Page Implementation

A general description and appearance of the updated landing page can be found in the previous chapter, 4.3.1. The figure of the landing page can be found in figure 1. There are four clickable buttons which help the user navigate the application.

The first button ”Watch Live Race” logs the user in as a ”Spectator” user so they can see ongoing races through a spectator view, which is further described in chapter 5.1.3.

The second button, ”Team Login”, sends the user to the login page by redirecting the website route to the login route, where the user can enter their login credentials.

The third button, ”Tech Inspection”, sends the user to the technical inspection page, which is http://evgp-test.globaleee.org/. On this page, participants can take part of the technical inspection questionnaire before the race.

In the figure below, it shows a PDF, which can be found after having pressed the ”User Manual” button.

Electric Vehicle Grand Prix

2020 Official Drivers Manual

Figure 7: The screen seen when pressed on ”User Manual” button.

It redirects the user to the public folder ’manuals’ and opens a PDF file describing the rules of the race, which the user can download if needed.

5.1.2 VEVGPP Track Editor Implementation

Due to time constraints, the track editor was not implemented. However, the addition of new tracks is present. This has been done manually by researching how current tracks were created and becoming more familiar with the variables and racing lines included in the race track files. Going into detail, each racetrack consists of seven different lanes. There are the left/right borders, the left/right lanes, the centerleft/centerright lanes and the main center lane. The distance between the center lane and the centerleft/centerright lanes is at all times 2.5 units, between the center lane and the left/right lanes is at all times 5.0 units and between the center lane and the left/right borders is at all times 7.5 units. The left, center and right lanes are lanes in which the car can drive. The left and right border lanes serve as a barrier and outline the limits of the race track. The centerleft and centerright lanes serve an aesthetic purpose to visibly distinguish between the three drivable lanes. In order to change lanes, there are control points. Control points are points on the track where the user can decide in which lane they want to drive in. Because of this, cars always encounter groups of three control points, one to switch into the left lane, one for the center lane and one for the right lane. On top of this, there is also the pit lane. The pit lane is a special section of the track which the user must enter in order to switch drivers and serve penalties. This is a very delicate component of the race track and, for the purpose of simplicity, it was decided not to change the functionality or layout of the pit lane on all new tracks.

Now that the components of the race track have been explained, we can go over how new race tracks have been created. The first step is to decide on the shape of the new race track. We graph the shape of the race track as a sketch and we pick out 20-25 points on the track. For each point, the x and y coordinates are noted and recorded in a new JavaScript file. This set of points will serve as the center lane for the new track. From the chosen points of the center lane, an outer ring is created that surrounds the center lane and is displaced at a distance of 2.5 units away from the center lane. This will be the centerleft lane if the track is driven in a clockwise direction, or it will be the centerright lane if the track is driven in a counterclockwise direction. This process is repeated for distances of 5 units and 7.5 units away from the center lane. At this point, there are four lanes. To get the remaining three inner lanes, we simply reflect each lane around the center lane. To show a specific example, if the center lane coordinates were [355.14331, 91.35708] and the coordinates of the right lane were [352.88432, 86.89603], we do the following: 355.14331 - 352.88432 = 2.25899, 355.14331 + 2.25899 = 357.4023. Therefore, 357.4023 is the x coordinate of the left lane for this specific point. This process is repeated for all points of all lanes, which will then give a complete table of the seven lanes that are needed.

The final step in creating the race track is to assign the control points around the track, and import the race track JavaScript file into the 'Track.js' and 'TrackController.js' files. We assign the control points within the race track JavaScript file in such a way that the user has the opportunity to switch lanes at entries to a turn and exits of the turn. This will give the user the highest possible degree of control and strategy over their race. Once we have finalized the race track file and imported it within the necessary controller files, we edit the admin view file, giving the admin the option to choose and render the new tracks.

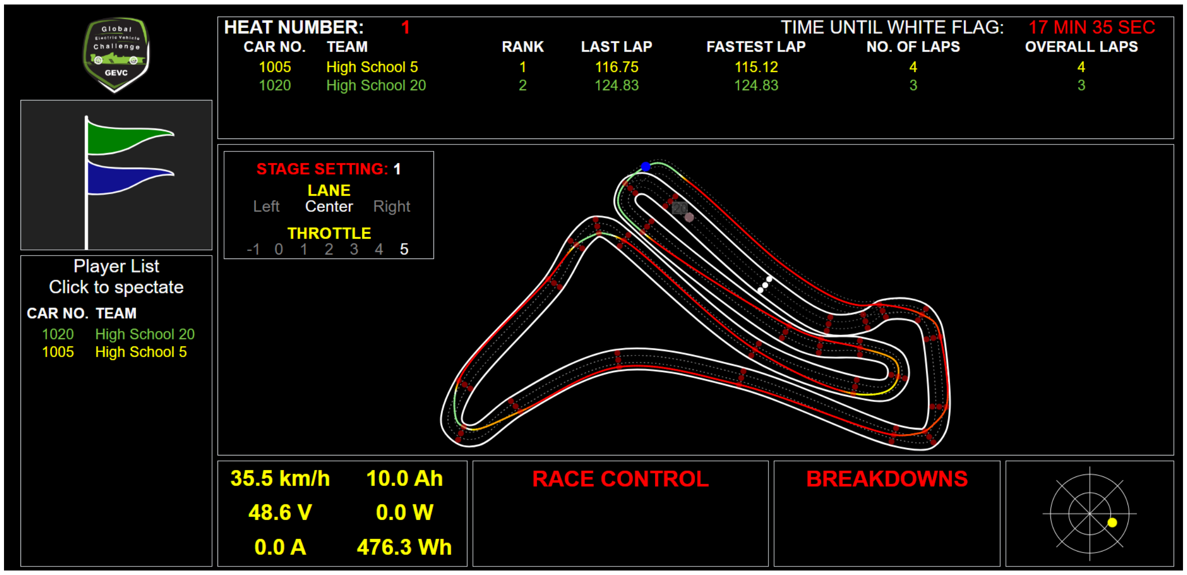
5.1.3 VEVGPP Spectator Mode Implementation

Within the application there now exist two types of spectator modes, one for the admin and one for the public to view active races.

First there is the public spectator mode, which is open to the public. This means that no account is needed for VEVGPP to use this mode. It is as simple as clicking the button "Watch Live Race" on the landing page, which can be found in the previous chapter in figure 1.

The public spectator mode has been done by creating a "Spectator" user which sees a modified view of the race participant view. This includes the race track, but without the ability to change race settings or 'participate' in the race. This is because the spectator user is not active on the scoreboard nor do they have an active blue car dot for themselves and other participants. Finally, they are also able to see penalties and breakdowns for all the race participants.

In the admin spectator mode, the admin is able to view screens of other players in a race. The following figure showcases this spectator view.


HEAT NUMBER: 1 TIME UNTIL WHITE FLAG: 17 MIN 35 SEC

CAR NO.	TEAM	RANK	LAST LAP	FASTEST LAP	NO. OF LAPS	OVERALL LAPS
1005	High School 5	1	116.75	115.12	4	4
1020	High School 20	2	124.83	124.83	3	3

Player List
Click to spectate

CAR NO.	TEAM
1020	High School 20
1005	High School 5

STAGE SETTING: 1
LANE
Left Center Right
THROTTLE
-1 0 1 2 3 4 5

35.5 km/h 10.0 Ah
48.6 V 0.0 W
0.0 A 476.3 Wh

RACE CONTROL

BREAKDOWNS

Figure 8: The admin spectator view.

This screen looks similar to the screen a player sees. The main difference component-wise is the "Player List" in the bottom left, replacing the interactive buttons. This is an interactive Player List where the admin can select any player by clicking on their team name. This is also indicated in the text right below "Player List" as "Click to spectate" to guide the user. Selecting a user results in changes to what is visible on the screen.

First of all, the player becomes highlighted in yellow, both in the Player List and the Scoreboard at the top, to indicate they are being spectated. The current progress of all players will be retrieved in real-time, with the select player's car being highlighted in blue. Their analyst data, shown in the bottom left next to the Player List, as well as the G-Force panel, shown in the bottom right, are also updated in real-time. In order to make spectating the G-Force and Analyst data possible, additional variables have been stored in the database. The G-Force data is calculated locally with this extra information to reduce the load on the server. Furthermore, their control points will be retrieved, which is possible thanks to the control point retrieval feature showcased in 5.1.4. The player's penalties and breakdowns will also be shown on screen, in the "RACE CONTROL" and "BREAKDOWNS" panels, respectively.

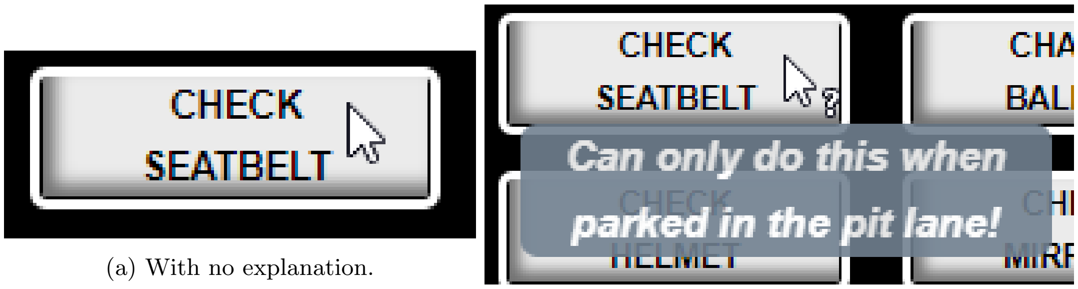
The penalties, breakdowns, and control points are retrieved both when the user is clicked and periodically every ten seconds. This time was chosen since these elements are rarely updated, but are still of importance to the spectator. Meanwhile, the current location of all cars, their analyst data, their G-Force data, and the scoreboard are still kept up-to-date in real time. This is because these are the main elements of the race and are therefore most important to keep the race going smoothly on the admin's screen.

5.1.4 Implementation of Race Settings Saved Upon Refresh of Webpage

The new version of VEVGPP, developed during this project, persists more data to the server than the previous version. One of those things are the race settings. When a user changes their race settings, a post request is sent to the database. The controller that handles this request, uses the *createOrUpdate* method, one of Laravel's methods for server interaction. As the name suggests, this method first checks if there already exists an entry in the database that matches the keys and values specified in the first parameter of the function call. If not, a new entry is created. However, if that entry does already exists, it updates that entry with only the keys and values specified in the second parameter of the function call. A user can save multiple race settings, namely one for every race track, and can therefore even use their race settings from the previous time that they raced on that same track.

5.1.5 Implementation of Textual Description for Unclickable Buttons

It is necessary for race participants to see the actual reason behind a button being unclickable. This has been done by adding a textual reason in the code where the logic of the buttons is defined for being unclickable. This reason is only shown when two conditions are met. Namely, the mouse needs to be hovering over a button and that button needs to be unclickable. Only then will the reason be displayed for the user. Below, the figure can be found which contains an image when hovering over a button.



(a) With no explanation.

(b) With explanation.

Figure 9: The 'check seatbelt' button when being unclickable and hovered over with a mouse.

As seen in figure 9b, the reason can now be found when hovering over the button. The mouse icon with the question mark serves as a visual indication of a button being unclickable. This improves the user experience, but it is also helpful for developers. As the developers are now able to see a textual description of the logic behind the buttons. Which is helpful for changing or expanding functionality of the buttons, as the logic behind them is now displayed in text format.

5.1.6 Implementation of the Administrator View of the Race Details

There previously was no way for the administrator to have any insights into the race data of the participants. Therefore, two new pages were added to fulfill this purpose. To accommodate the two new pages in the administrator view a tab bar was added to the top of the page. The tab bar was implemented using a React component library called MUI (Material UI SAS, 2022). In the race information tab, an overview of all participants is shown in a scoreboard-like fashion. Upon clicking one of the participants, live data is shown about all available race information. After a couple of seconds, an API request is made to the server for the current penalties and breakdowns. The timing of the API requests is based on server ticks to avoid any complications with client-side timers. The design of the race information tab can be found in figure 6.

5.2 Manual Tests as Replacement of Automated Integration tests

A testing issue occurred during the course of this project. While there were functions which could be separated for a test, it was not possible to test full elements of the application, or how one element interacted with another. A temporary solution was created to deal with this issue, namely manual tests. The next subsections describe how manual testing is used as a solution to the testing problem, it gives an example of such manual tests and finishes up with an explanation of how these tests can be transformed into automated tests.

5.2.1 Manual Testing

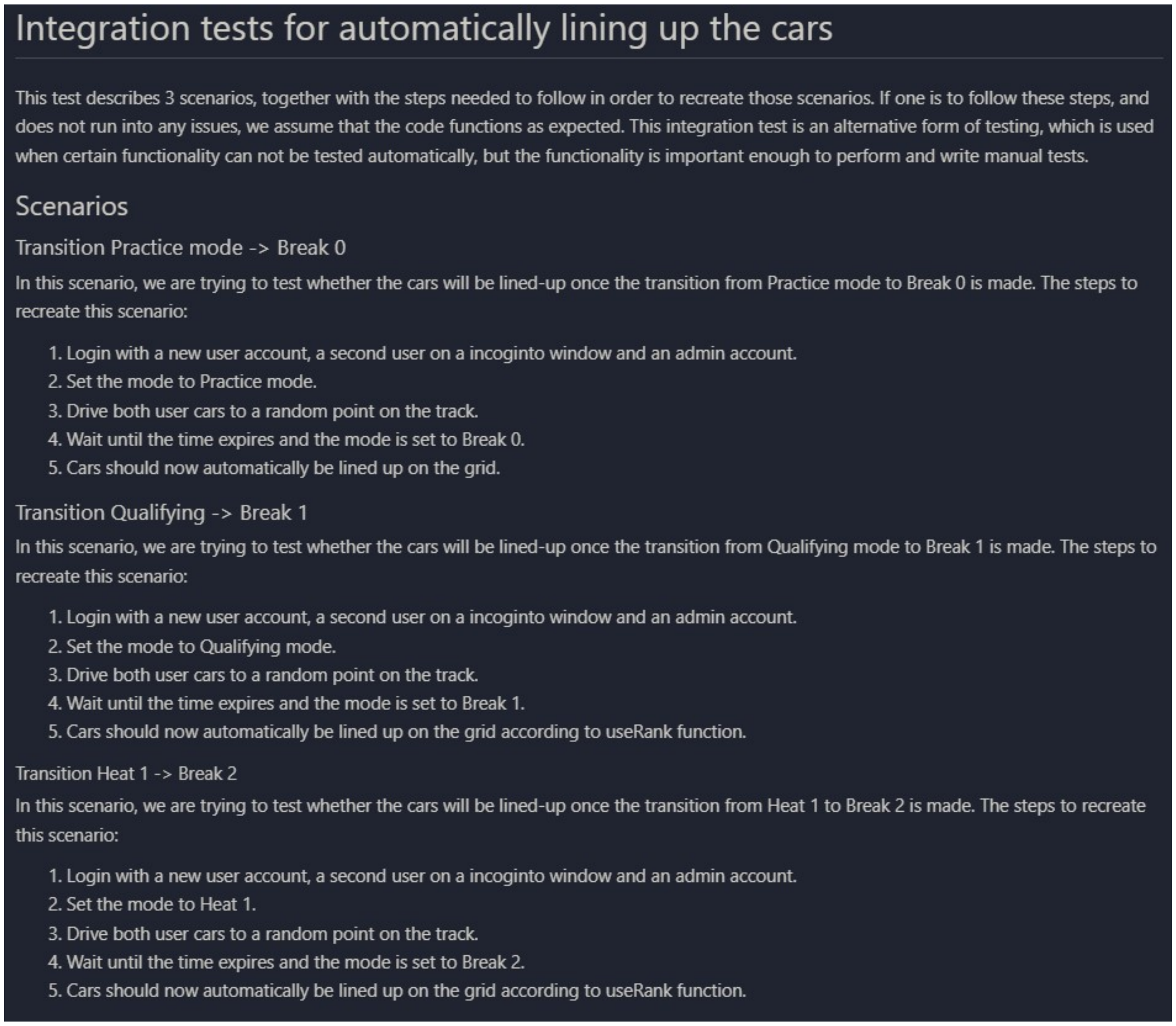
Within the confines of this project, considering the time constraint combined with the conflicting demands of the client and requirements of the TU Delft, testing became a difficult roadblock to overcome. It was deemed impossible to test the existing application, create new features, and test those features all within this project. Manual testing can be done to ensure functionality while continuing development. This is a good alternative for now, with the idea in mind that these tests are to be transformed into automated tests as much as possible in the future. A big restructuring could facilitate this conversion if that is something that the client, GlobalEEE, wishes to do so. This restructuring is more touched upon in 8.2.1, which explains how the code base could be restructured to improve testability and modularity. Wikipedia (Wikipedia Contributors, 2019) defines the function of integration tests as follows:

> *Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.*

The input modules for the integration tests for the VEVGPP are not, and can not yet, be tested automatically to fulfill this requirement. It was allowed to assume, during this project, that the input modules are functioning, if the manual representations of these tests pass. That is, it is assumed that the application works in all circumstances, if the manual tests are performed and the application works as described by the tests.

5.2.2 A Manual Test Example

The following figure shows an example of a list of manual tests for a feature, contained in a markdown file.



Integration tests for automatically lining up the cars

This test describes 3 scenarios, together with the steps needed to follow in order to recreate those scenarios. If one is to follow these steps, and does not run into any issues, we assume that the code functions as expected. This integration test is an alternative form of testing, which is used when certain functionality can not be tested automatically, but the functionality is important enough to perform and write manual tests.

Scenarios

Transition Practice mode -> Break 0

In this scenario, we are trying to test whether the cars will be lined-up once the transition from Practice mode to Break 0 is made. The steps to recreate this scenario:

1. Login with a new user account, a second user on a incoginto window and an admin account.
2. Set the mode to Practice mode.
3. Drive both user cars to a random point on the track.
4. Wait until the time expires and the mode is set to Break 0.
5. Cars should now automatically be lined up on the grid.

Transition Qualifying -> Break 1

In this scenario, we are trying to test whether the cars will be lined-up once the transition from Qualifying mode to Break 1 is made. The steps to recreate this scenario:

1. Login with a new user account, a second user on a incoginto window and an admin account.
2. Set the mode to Qualifying mode.
3. Drive both user cars to a random point on the track.
4. Wait until the time expires and the mode is set to Break 1.
5. Cars should now automatically be lined up on the grid according to useRank function.

Transition Heat 1 -> Break 2

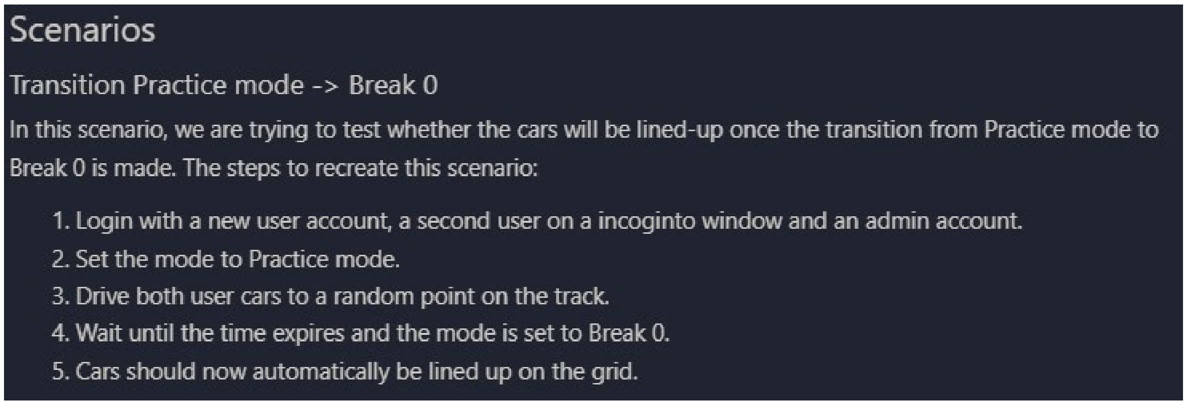
In this scenario, we are trying to test whether the cars will be lined-up once the transition from Heat 1 to Break 2 is made. The steps to recreate this scenario:

1. Login with a new user account, a second user on a incoginto window and an admin account.
2. Set the mode to Heat 1.
3. Drive both user cars to a random point on the track.
4. Wait until the time expires and the mode is set to Break 2.
5. Cars should now automatically be lined up on the grid according to useRank function.

Figure 10: An example of a manual test file.

The markdown file starts with a brief description of what the test describes and that the code is assumed to function correctly if all the manual tests succeed. It then proceeds to talk about three different scenarios. These scenarios are supposed to capture every possible scenario that might occur within the usage of the application. The tests in figure 10 test whether cars automatically line up when the race goes from one stage to another. That is, from practice mode to break one, testing the functionality when the race goes from a practice stage to a break stage. Another scenario would be going from a break stage to a heat stage. This functionality is captured in the second scenario. The third race tests the functionality when the race goes from a heat stage back to a break stage. For certain functionality, more scenarios can be thought of, but those might be unlikely and thus unnecessary to test manually. In this case, we could also test the scenario when the race goes from a heat stage to a practice stage, but this will simply never occur.

The following image takes a closer look at one of the earlier mentioned scenario's.



Scenarios

Transition Practice mode -> Break 0

In this scenario, we are trying to test whether the cars will be lined-up once the transition from Practice mode to Break 0 is made. The steps to recreate this scenario:

1. Login with a new user account, a second user on a incoginto window and an admin account.
2. Set the mode to Practice mode.
3. Drive both user cars to a random point on the track.
4. Wait until the time expires and the mode is set to Break 0.
5. Cars should now automatically be lined up on the grid.

Figure 11: A manual test scenario example.

This scenario lists out all the steps, from logging in to the final result, making this test accessible and workable to anyone with a basic understanding of the application and the admin view.

5.2.3 Integration Test Conversion

Automatic tests are recommended to be created from manual tests once the code base has been modified in a way that allows this. These can then be kept alongside each other. While manual tests confirm that the desired functionality is there, automated tests are needed to guarantee functionality in any scenario, and will automatically detect failure when code breaks. With only manual tests, some problems might stay undetected until these tests are performed.

Figure 11 shown above in 5.2.2 will be used as an example. In this example, the automatic car line-up feature was tested. In order to test this feature automatically, there are classes which need to be mocked or mounted, meaning they get instantiated for a test (Russom, 2018). This way, it can be checked if the code behaves as expected. In this case, this would be one of the classes used by the TrackController class, which sets up the racing view one would see during a race. Here, functions exist which deal with the line-ups, which are necessary for this test.

Within the steps of the manual test there are two different elements which need to be handled. The first element is the setup; the first two lines indicate that two users need to be logged in, while using the Practice mode. In a refactored code base this could be handled during the mounting process; two users being logged in could be achieved by mounting the class with a fake user while keeping in mind another car already being on the track. The practice mode describes the race track, which could be massively simplified to a smaller one. The second element is the test itself; lines three and four describe what needs to occur during the race and these could be executed through steps. Cars being at a random point on a track could be done by using predetermined points, which will determine the outcome. The timer expiring can be negated, as the thing this leads to, the next mode, can be called manually. At the end, five of the current positions of the car can be compared to see if the line-up happened correctly, keeping in mind their initial locations.

6 Development Process of VEVGPP

In this chapter, the development plan for the remainder of this project will be given. First, a mention of the roles distributed among the team will be given in section 6.1, after which an explanation of the process frameworks is in section 6.2. After this, the development and collaboration tools will be explained in section 6.3, followed by the project schedule in section 6.4. Finally, the task distribution of the MoSCoW requirements will be discussed in section 6.5.

6.1 Role Distribution During Development of VEVGPP

Roles were created for the development of the second iteration of VEVGPP. This was done so no-one on the team was overwhelmed by the scope of the project, while also giving a point of contact for the client. For this, five roles were created which were distributed among the team. The roles are as follows:

1. **Team Leader (Vladimir Rullens).** The team leader was responsible for delivering the code at the end of the week, as well as keeping track of, and re-distributing the current issues as necessary, making sure that everyone has enough to do, but does not get overworked. Finally, they were also the point of contact for the client in case of any questions or updates.

2. **Software Tester (Andy Li).** The software tester was responsible for formalizing the existing requirements in a live document. These requirements were not requirements for the project, but the requirements the existing program already satisfied. This list of requirements were appended with the new features and requirements completed during this project, such that another person or group can continue to work on this project using these already satisfied requirements. The software tester was also responsible for tests to make sure that the requirements are satisfied. Lastly, they were responsible for refactoring the code to improve testability.

3. **Quality Manager (Bas Filius).** The main tasks for the quality manager were to set up continuous integration for quality control, such as incorporating 'Better Code Hub' into our GitLab CI/CD (continuous integration and continuous deployment) pipeline, as well as refactoring the existing code base to improve its readability. This way groups working on the application can be more efficient.

4. **Track Engineer (Maroje Borsic).** The track engineer's responsibility was for a feature to be created which allows the admin to easily create racetracks. At the start of this project two racetracks were already hard coded, one for practicing, and the other for the official race.

5. **Back-end Visualizer (Jan Domhof).** The race steward needs to see the race data from all the cars, including physics, vehicle configurations and game state (penalties, flags, breakdowns). This data should also be available after the race for analysing purposes. The implementation of what is described above was the main responsibility of the back-end visualizer.

Being responsible for a certain feature, task or quality control meant keeping track of the distribution of work done within that task. On top of this, these roles assigned responsibility for code written by the group regarding specific aspects of the code, resulting in a better overview of how the team was progressing in each part of the project. In order to stay consistent, these roles were not rotated.

6.2 Processes Used During the Development of VEVGPP

Processes were utilized in this project to improve the overall efficiency and quality of work. The following gives a description of the processes which have been used during the development process of VEVGPP:

- **SCRUM.** The SCRUM (Drumond, 2018) development framework was utilized during this project. SCRUM sprints had a 1-week duration. These sprints started with a planning period, where a meeting was held which discussed the issues that would be handled during that week, the time estimate for how long it would take to complete the issue, as well as which project member would be responsible for the delivery of this issue. There was one team member assigned as SCRUM master for the respective week, supervising the group, and making sure that the group was on track with the weekly SCRUM plan. This role was rotated to make sure one member did not consistently do more work than others. At the end of each sprint, a sprint retrospective meeting was held, which discussed the task progress, as well as the tasks that still needed to be worked on. Here the team also reflected upon what went well and what could have been done better in the next sprint. The SCRUM master changed on a weekly basis depending on the set schedule that had been set up.

- **Testing.** The code base needed to be improved, as it contained no tests before the start of the project. It was up to the team to test the existing code base, as well as the new features implemented. The software tester had to make sure that the tests for the existing code base were distributed evenly amongst the group and that the tests were well documented. Testing the newly implemented features was done right after the implementation by every group member, making sure to catch most of the issues early on. This prevented spending time on debugging and refactoring our own code.

6.3 Tools Used During the Development of VEVGPP

Tools were utilized in this project to aid the team's communication with anyone related to this project. The following tools have been used during the development process of VEVGPP:

- **GitLab.** To facilitate collaboration and version control, we used GitLab. GitLab's milestone feature is an excellent tool to keep track of the weekly sprint progress, as it organizes the issues and boards such that the SCRUM master can quickly see who is on track and who is falling behind. On Monday in week 3, the 2nd of April 2022, we had our first sprint planning meeting, during which we created two issue boards. The first board contains a list of issues, where each issue corresponds to a requirement, containing the importance of the requirement as well as the requirement description. The second board contains the issues for the current sprint, which could be marked with the corresponding sprint milestone. This board consists of 6 columns: backlog, to-do, doing, testing, review and done.

  The requirements to be assigned from the first board were decided at the beginning of every sprint, after which they were linked to the requirement issue and added to the sprint board. Every issue was initially placed in the backlog column. This column was meant for unassigned issues that belonged to the current sprint. After assigning a group member to a certain issue, the issue would advance to the to-do column. If some issues remained in the backlog column, they could be picked up by group members once their assigned issues were completed. For the remainder of the sprint, each group member had to keep track of their issues and make sure they were in the correct column corresponding to the status of the issue.

  Our branch structure consisted of two protected branches, main and dev, in which no project member was allowed to work in. Instead, someone could branch out of dev for every issue they would be working on. Since there would be many issues and branches, a naming convention was agreed upon to start the branch name with the issue number from GitLab. The dev branch was isolated from merge conflicts, by rebasing onto the dev branch, and resolving the rebase conflicts locally. Only after the issues were resolved and the pipeline had passed, a merge request could be created.

  To maintain high code quality, we would work with code reviews for merge requests throughout every sprint. We set a hard requirement of two approvals for every merge request. This made sure that at least two group members knew what and how features were being implemented, while they also checked that the code that would be merged was merged correctly and conformed to the quality standards.

- **Discord.** Discord was our main way of communication within the group. We created multiple channels for the maintenance and clarity of information. For example, there were channels dedicated to merge requests and questions which were to be asked to our teaching assistant (TA). This ensured that the chats remained uncluttered. Discord was also our preferred platform for online meetings and to communicate on days we were not physically working together. We also created a discord webhook and connected this to GitLab, allowing us to receive push notifications within discord once a group member pushed, created, merged or commented on a merge request or created an issue.

- **WhatsApp.** We used WhatsApp to keep in touch with our client and to maintain a quicker way of communication. This made it easier for both parties to reach each other and to ask questions.
- **Mattermost.** Mattermost was mainly used for communication with the TA and Coach regarding questions and information about the project, as well as to keep up to date with the course-wide announcements.
- **SharePoint.** TU Delft's SharePoint includes One Drive, which we used as a cloud collection for all the documents related to the project assignments we worked on during the project.

6.4 Project Schedule

The seven-week long development process started in week three (2nd of May 2022 till 8th of May 2022). By then, the list of requirements was completed, as well as the project plan and the role division. The first sprint planning meeting was held on Monday the 2nd of May 2022 in week three (9th of May 2022 till 15th of May 2022), which marked the start of development for the first 'must-have-requirements'.

Two software test runs were planned to test the functionality of the software. For these runs, friends and family were invited to join in on a race using the application. We planned to hold an initial test run at the start of the development. However, medical complications arose resulting in this one being cancelled. A final test run was held near the end of development on Tuesday 14th of June 2022 in week nine (13th of June 2022 till 19th of June 2022). During this test run, the team members were the race supervisors. This meant looking at the back-end through the admin view and public spectator views designed in this project, shown in 4.3.3. This resulted in a better understanding of what exactly went on during a race, while also potentially encountering any bugs. The point of the cancelled first race was to give a better indication of any potential features that we could add to improve the user's experience. The second race gave a good indication of the progress made and what bugs needed to be addressed.

There were also two presentations, which were held for the client, coach and TA. The first presentation took place in week six on the 23rd of May 2022, and the final presentation took place in week ten on the 22nd of June 2022. The former being an indication of the progress of the project, and the latter being the result which will be delivered to the client.

6.5 Task Distribution Approach

To ensure a balanced workload, the tasks were distributed evenly among the project members. The tasks were split when starting with a new category of the MoSCoW method. The 'must-have-requirements' were distributed considering the strengths and weaknesses of each project member. After this, the 'should-have-requirements' were distributed among team members who were able to complete their 'must-have-requirements'. The amount of requirements were attempted to be distributed equally, but some issues turned out to be more difficult than others, which meant some group members finished more issues than others. The distribution of requirements can be found in Appendix C for the must-have-, should-have- and could-have-requirements. The check mark represents that the issue has been assigned to a certain developer and finished and the cross represents that the issue has been assigned to a developer but has not been finished (or abandoned) whereas the lack of a check mark/cross represents the fact that that specific issue has not been assigned to anyone.

7 Moral Implications with the Use of VEVGPP by Minors

It is important to be aware of any moral implications when considering high-school students as the end-user. We aim to lay out two concerns for minors by using our application. In section 7.1, we examine some behavioral concerns for VEVGPP when used by minors. Then in section 7.2, we examine the safety concerns of personal data for VEVGPP.

7.1 Behavioral Concerns of the Use of VEVGPP by Minors

It is important to keep behavioral concerns in mind when working on VEVGPP. It is even more important, since children are the end-users, and their brain functions are more susceptible to change (Rosyati et al., 2020). Thus the use of VEVGPP can influence their behavior quite significantly.

An immediate concern that comes to mind when considering a 'gaming application' is how addictive the application is. Gaming addiction has become an increasing topic of interest and saw a significant increase in the number of empirical studies examining various aspects of video game addiction (D. Griffiths et al., 2012). Another study showed that particularly younger people are more susceptible to highly-time consuming gaming behavior and even gaming addiction (King Delfabbro, 2020). Treatment for Gaming Disorder (GD) has been generally adapted from addiction treatments, showing that gaming addiction is a serious problem as this can affect behavior.

This issue has been minimized by the principles of how VEVGPP works. The 'official race' can only be hosted by an official, meaning the end-user is not able to participate unless there is an event scheduled. This also holds for the 'practice mode' where users can race freely on a practice track to get a feel of the racing mechanics. Thus, the end-user is not able to spend their free time using the application.

Furthermore, VEVGPP does not make use of excessive visually stimulating elements such as animations, vibrant colors, and images. The role of gameplay loops which is the role of regular incentivizing feedback loops in addictive (gaming) behavior has also been taken into account when analyzing the addictiveness of VEVGPP. The application has been designed to show only the necessary information which minimizes the chances of minors getting addicted to using VEVGPP.

7.2 Safety Concerns of the Storage of Data Within VEVGPP

Safety of storing personal data is a concern for any application, as a lack thereof can lead to users with malicious intent to i.e., leak stored (private) information. Therefore, it is of utmost importance to be mindful of such an important aspect in the creation of this project.

One of the requested 'should-have-requirements', the first 'should-have' in Appendix B, for VEVGPP is the ability for the admin to change pictures or text on the website to suit their needs. However, this can be very dangerous, since a malicious user could take advantage of the vulnerabilities of personal data by logging into the website using the credentials of the admin. They can make images extremely graphic, exposing minors to graphic content, or do any other illegal activities, replacing what was originally there all while under someone else's name.

It is important to reduce the chances of this happening as much as possible, as complete mitigation is never truly guaranteed (Finney, 2018). Therefore, we will primarily put our focus on two aspects to mitigate the problem:

First of all, the account of the admin needs to be protected from getting hacked. This will be done by improving the encryption algorithm, to make sure their credentials cannot be easily decrypted. It is recommended to change the password of the admin periodically, to counter potentially unknown data leaks. On top of this, the password should be improved to make sure it cannot be taken through a brute-force attack (Esheridan, 2022).

Furthermore, it is important to be able to back up previous versions, by storing these on the server's side. This however comes with glaring costs. Namely, that it would be possible to store too many backups, to the point of overloading the server. A fix for this would be to limit the number of times one can upload to the server. This allows for a timely response to the problem.

Storage of private personal data on the other hand is not of concern within our application. The only information which is stored, is the information of a car that is driving, and registration details, which include team name, car number, username, and password. As long as the username and password are stored securely, there is no need to worry about private information. This is because the account's only purpose is to drive in the race, with no added information. The only concern would then be secure storage, which is already handled in VEVGPP through Laravel's hashing algorithm.

8 Conclusion

This chapter contains a final conclusion, as well as a recommendation for future groups that will work on this project.

8.1 Conclusion

The goal of our project group is to extend the current version of the platform and deliver a well-structured, bug-free, user-friendly application with the necessary requirements suitable to use for the students and satisfy the demands of our client. To achieve this, a problem analysis was done. Based on this analysis and meetings with the client a set of requirements was elicited. Research has been done on the right use of programming languages and frameworks and the moral implication of the use of VEVGPP by minors.

Through an elicitation process with the client and a prioritization technique called the MoSCoW method certain requirements were established. The 'must-haves', the minimal set of finished requirements, are all finished and working as well as some of the second most important category of requirements: the 'could-haves'. Some of these requirements were bugs that were found during the test sessions. All these bugs and several more that have been found during development have been resolved. On top of that, a lot of effort has been made during development to make the code base as future-proof as possible. Documentation was added within the code to explain the dependencies and functionality of certain methods. This was mostly done with methods that the group had worked on and some other important classes. Documentation was also added outside the code in the form of a getting started guide.

During the development, the group encountered some problems regarding the testing of the application. The application was never developed with extensive testing in mind. This resulted in a highly dependent and connected code base, which could have been solved by refactoring the whole or large parts of the code base. It was, however, decided that this would not be the best way to spend our time during this project. It was possible to write unit tests for some parts of the code, however, the remainder required a different answer. The solution for this was to write manual tests. A manual test looks like a set of instructions that a tester can perform to test certain functionality. So through extensive testing during development and the creation of these manual tests, a certain degree of bug-free code could now be guaranteed.

The client seemed overall satisfied with the delivered work. The group has obtained weekly feedback during the meetings with the client and the midterm presentation. The client said the features lived up to his expectations in how they looked and in terms of functionality. With this, it can safely be concluded that this project was a success, even though not all requested features have been implemented. Ultimately, this report has shown the improvements which have been made to the code base, containing both new features as well as better documentation and testing. Future groups are encouraged to improve upon this code base further, so the application has a bright future.

8.2 Recommendations for Future Teams

Based on this conclusion and our development process we have a few recommendations for future developers of this project. In this section, recommendations are given for the code structure and the frameworks that VEVGPP is built upon, such that future groups can take these recommendations into account when further developing this software.

8.2.1 Refactoring the code base of VEVGPP

Something to always keep in mind when refactoring VEVGPP is that you can never refactor everything. There are a lot of functions with high dependency on other functions and variables. If one would extract such a method, it would require them to rewrite that method such that it gets all those dependencies as arguments. This would only decrease code quality, defeating the purpose of refactoring in the first place. Therefore, some functionality will always remain untested, which is fine. Extensive integration testing can fill those gaps and give somewhat of a guarantee that that functionality will work in most cases that occur during the use of the application. Nevertheless, this does not mean that one can not extract a significant amount of functionality to still increase testability, modularity, and code quality.

A good place to start is removing duplicate code from all files. For example, there is a JavaScript file called *apiRequests.js*, that includes all API requests. While this file currently includes many different API requests, this can be reduced by creating a single method for POST requests, which receives the body and the API path as arguments. Many methods within the current code base are simply copied and pasted. Going over the entire code, extracting and generalizing methods, and testing those methods separately would increase the code readability and quality dramatically.

8.2.2 Choice of Frameworks

Because many different groups will work on this project successively, it is of high importance to choose frameworks not only on their performance but also on their ease of use and learning curve. The two major frameworks that are used, Laravel and React, are excellent choices for this project because of those exact reasons. Firstly, Laravel is easy to learn and use and thus allows future development to quickly pick up where a previous group left off, even without any PHP experience. Furthermore, it is flexible, has easy integration, and has good compatibility. In the future, this code base could transform, perhaps even a complete restructuring, where Laravel's flexibility and easy integration could play a vital role in saving valuable time and effort. Similarly, React is great for groups that want to pick up this project and quickly want to start implementing new features. It is a relatively simple framework, with a fair amount of documentation, tutorials, and training resources. In the future, a group or single developer can decide to stay with the project for a longer period of time. In that case, they could consider Symfony as an alternative, considering the advantages mentioned in 4.1.

The actual game part of the VEVGPP is another topic of discussion. There are many game engines, ranging from advanced real-time 3D engines like Unreal Engine to a simple point-and-click game creation tool like Scratch. While a game engine could be useful for simple web games, it is not clear if it would be beneficial in this case because of two main reasons. Firstly, keeping the number of frameworks, languages, engines, and tools to a minimum helps with the continuation of the development of the platform. Learning React, Laravel, and a game engine all in a single project could be overwhelming and would therefore only work against this platform's development. Secondly, game engines generally have their own physics. The vehicles that are used in these races have different physics than most of the game engines' built-in physics models. This would require the incorporation of custom physics into an existing game engine, which would then overcomplicate things for this simple web game. This clever native React implementation allows the use of custom physics, which can be changed according to rule changes or new vehicle designs.

8.2.3 Final Recommendation

To make this project more future-proof, the code structure needs to be improved, as well as the code quality. Taking plenty of time to improve on this will save groups many hours later on. This is the biggest point of action for this project in the near future. Perhaps even making that improvement a short project on its own could be a great idea. Only then, can the code even be tested, and allows more features to be added with ease. As for the frameworks, a solid choice was initially made when designing this project. If there comes a time when this project is rebuilt from scratch, the development team could look into game engines that can take this platform to a higher level, making the experience feel more like a game rather than a simulation. However, without a definite incentive, there is no direct reason to pick different frameworks to implement the same functionality.

References

Asper Brothers (2019). *Laravel vs Symfony in 2022 – Which PHP Framework Choose For Your Project?* [online] Available at: https://asperbrothers.com/blog/laravel-vs-symfony/.

Bartik, A.W., Bertrand M., Cullen Z.B., Glaeser E.L., Luca M., Stanton C.T., (2020). *HOW ARE SMALL BUSINESSES ADJUSTING TO COVID-19?* [online] Available at: https://www.nber.org/system/files/working_papers/w26989/w26989.pdf.

Bostock, M. (2021). *D3.js - Data-Driven Documents.* [online] D3js.org. Available at: https://d3js.org/.

Brewster, C. (2022). *12 Examples of Successful Companies Using React Native in 2022* [online] Available at: https://www.trio.dev/blog/companies-use-react-native.

Codeception (n.d.). Codeception - PHP Testing framework - PHP unit testing, PHP e2e testing, database testing. [online] Available at: https://codeception.com/.

d3indepth (2019). D3 curve explorer. [online] Available at: http://bl.ocks.org/d3indepth/b6d4845973089bc1012dec1674d3aff8

Drumond, C. (2018). Scrum – what it is, how it works, and why it's awesome. [online] Available at: https://www.atlassian.com/agile/scrum.

Esheridan. (2022). Blocking Brute Force Attacks. [online] Available at: https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks

Facebook, Inc. (2017). Jest. Delightful JavaScript Testing. [online] Available at: https://jestjs.io/.

Finney, G. (2018). The Illusion Of Perfect Cybersecurity. [online] Available at: https://www.forbes.com/sites/forbestechcouncil/2018/03/27/the-illusion-of-perfect-cybersecurity/?sh=623cfaee11f9

GlobalEEE.org. (2022). Global Education Energy Environment. [online] Available at: https://globalEEE.org/

Griffiths, M.D., King, D.L. and Kuss, D.J. (2012). "Video Game Addiction: Past, Present and Future.". *Current Psychiatry Reviews.* vol. 8. pp.308-318. DOI: 10.2174/157340012803520414.

King, D.L., Delfabbro, P.H. and Essau, C.A. (2020). "Chapter 7 - Video game addiction". *ScienceDirect.* Academic Press. pp.185-213. Available at: https://www.sciencedirect.com/science/article/pii/B9780128186268000074

McKeachie, C. (2021). JavaScript Frameworks in 2022. Accelebrate. Available at: https://www.accelebrate.com/blog/javascript-frameworks-2022.

Material UI SAS. (2022). MUI: The React component library you always wanted. [online] Available at: https://mui.com/.

Pereira, G. (2018). D3.js Tutorial: Drag and Drop. [online] octoperf.com. Available at: https://octoperf.com/blog/2018/04/18/d3-js-drag-and-drop-tutorial/#d3js-click-and-drag-events-combined.

Rosyati, T., Purwanto, M.R, Gumelar, G and Yulianti, R.T. (2020). Effects of Games and How Parents Overcome Addiction to Children. *Journal of Critical Reviews.* vol. 7. pp.65-67.

Russom J. (2018). Integration Testing in React [online] Available at: https://medium.com/expedia-group-tech/integration-testing-in-react-21f92a55a894

Sagar, P. (2021). Top 7 PHP Frameworks for Web Development in 2022. [online] Excellent Webworld. Available at: https://www.excellentwebworld.com/best-phpframeworks/

Sjoberg, D.I.K., Yamashita, A., Anda, B.C.D., Mockus, A. and Dyba, T. (2013). Quantifying the Effect of Code Smells on Maintenance Effort. IEEE Transactions on Software Engineering, 39(8), pp.1144–1156.

Software Improvement Group (n.d.). Better Code Hub / frequently asked questions. [online] BetterCodeHub. Available at: https://bettercodehub.com/docs/faq.

Wikipedia Contributors. (2019). Integration testing. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Integration_testing

Xu, W. (2021). Benchmark Comparison of JavaScript Frameworks React, Vue, Angular and Svelte.

Appendix A : Non-Functional Requirements

1. The system should be **loosely coupled, modular** and allow for **easy integration with other systems (API).**
2. The system should be written in **HTML, JavaScript (utilizing React) and PHP (utilizing Laravel).**
3. All interactions with the systems are handled through their respective **APIs**.
4. The development process must use **continuous integration.**
5. To determine and improve the code quality, **Better Code Hub** should be used.
6. The system should have at least **75%-line coverage** for new features/code.
7. The system should have at least **50%-line coverage** for existing code.
8. For integration tests **Jest** and **Codeception** should be used.
9. For unit tests **Jest** and **Codeception** should be used.
10. The source code must have a clear general format for documentation.
11. The connection between the front-end and back-end should be more stable.
12. There must be two official race sessions in which we test the framework on real users (volunteers).
13. There must be a 'Getting started guide' which is easy to edit. It needs to include the development setup for setting up all the servers to make the race possible as a developer. As well as document where to find what in the source code.

Appendix B : Functional Requirements

Must-have:

1. **Users** *must be* integrated with the technical inspection questionnaire page prior to a race.
2. **Users'** car *must* automatically be lined up after every racing break.
3. **Users** *must be able to* keep their race settings after being disconnected or if the page has been refreshed.
4. The application (server) *must* be updated of the car progress upon reconnect.
5. **Users** *must be able to* always change their race settings (e.g., not be blocked by other players' cars) [bug fix].
6. **Users** *must be able to* always see the lap counter and it must only be updated when a car passes the finish line. [bug fix]
7. **Users'** car *must* automatically stop during a pit stop.
8. **Admins** *must be able to* view screens of other racers as a spectator.
9. The application must contain a new administrator view for the current race details (i.e., breakdowns, yellow flags, penalties, car data)
10. **The race** *must* automatically continue to the next racing stage without the need of manual interference from an administrator.
11. **The website** *must* have a landing page for users that are not logged in yet.

Should-have:

1. **Admins** *should be able to* change any website images or logo's by uploading them.
2. **Admins** *should be able to* change the introduction page text preferably through the GUI.
3. **Admins** *should be able to* select a different track for both the practice race and the real race.
4. **Users** *should be able to* see a textual description upon hovering an unclickable button explaining why the button is unclickable.
5. **Users** *should not be able to* log into the same account at the same time.
6. **Users** that are not participating *should be able to* spectate the race from a general view.
7. **The application** *should* save relevant information on the server of the drivers.

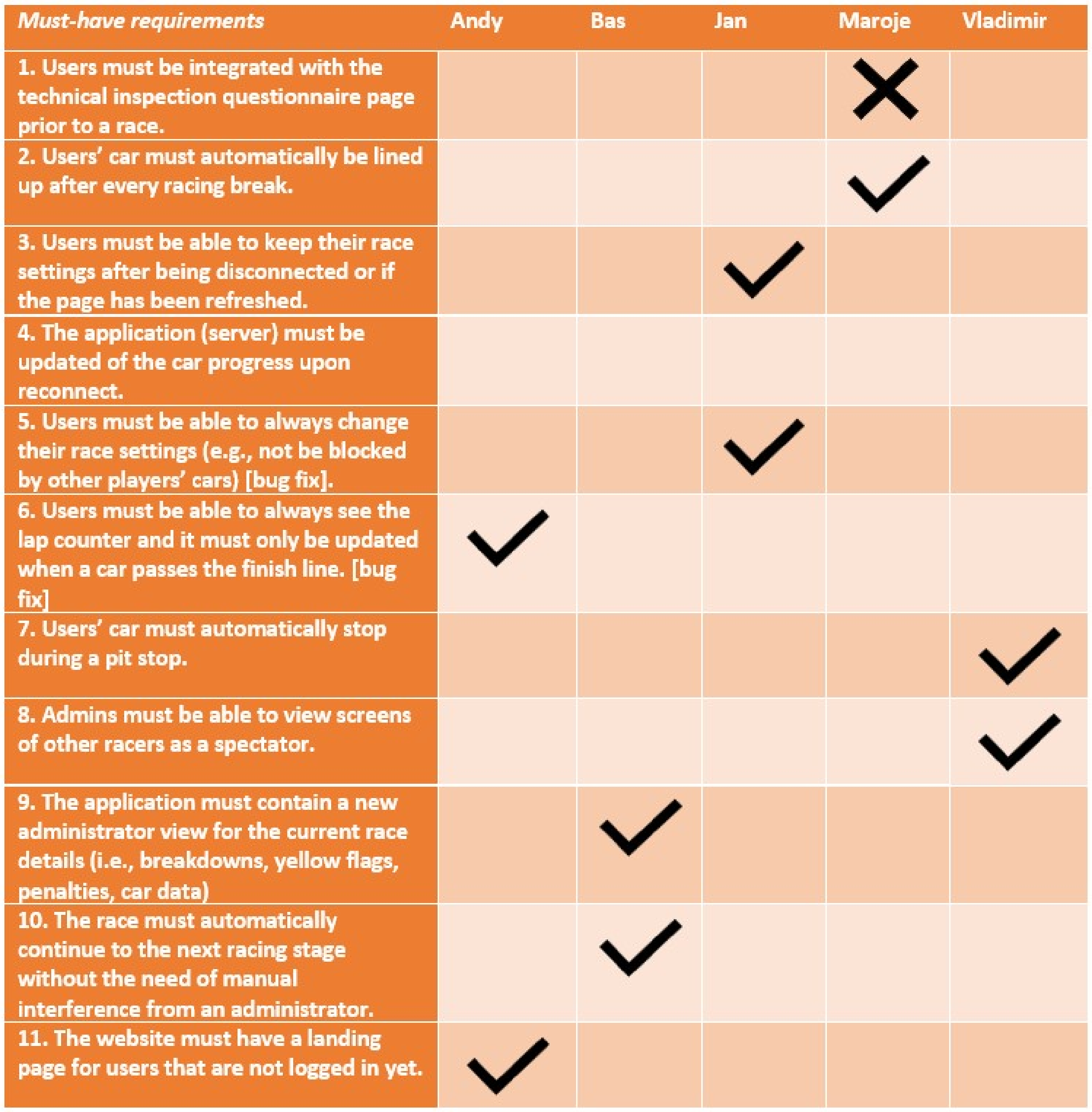
Could-have:

1. **Admins** *could* create a new racing track on the administrator page.
2. **Admins** *could* edit an existing racing track on the administrator page.
3. **The application** *could* contain four unique racetracks.
4. **The application** *could* have options for multiple log ins and controls over the car.
5. **The GUI** *could* have a visual overhaul.

Won't-have (for our development cycle):

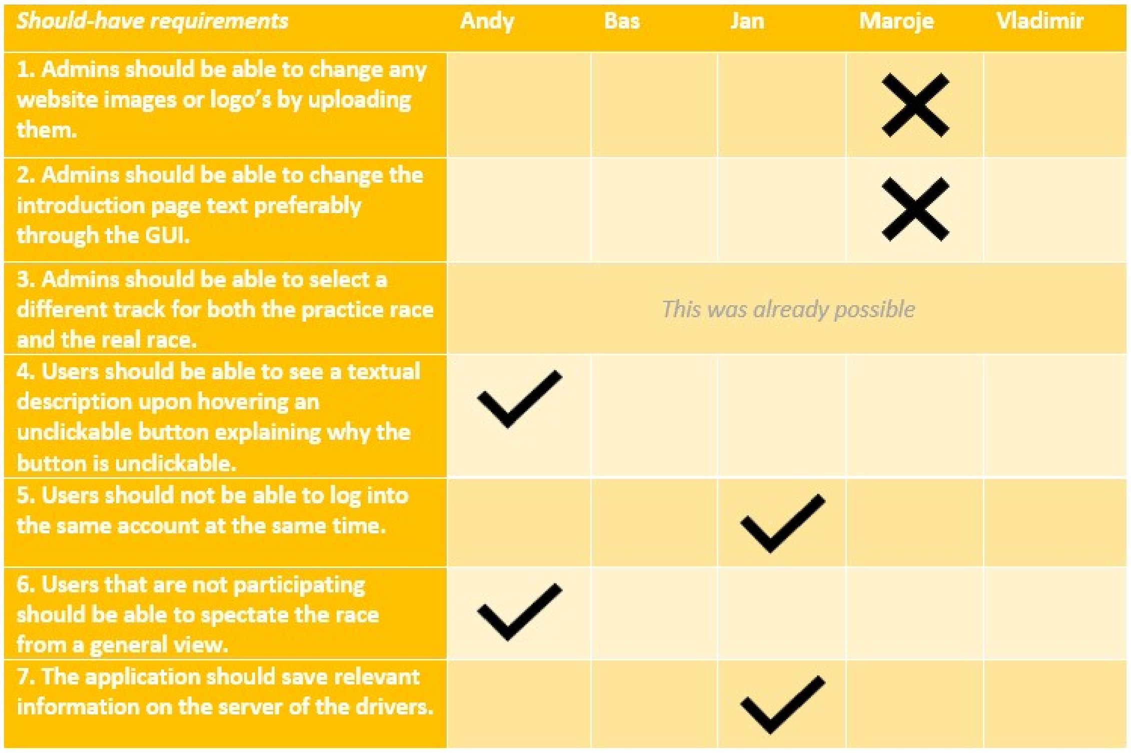
1. **The application** *won't have* a 3D-simulation.

Appendix C : Task Distribution of MoSCoW Requirements



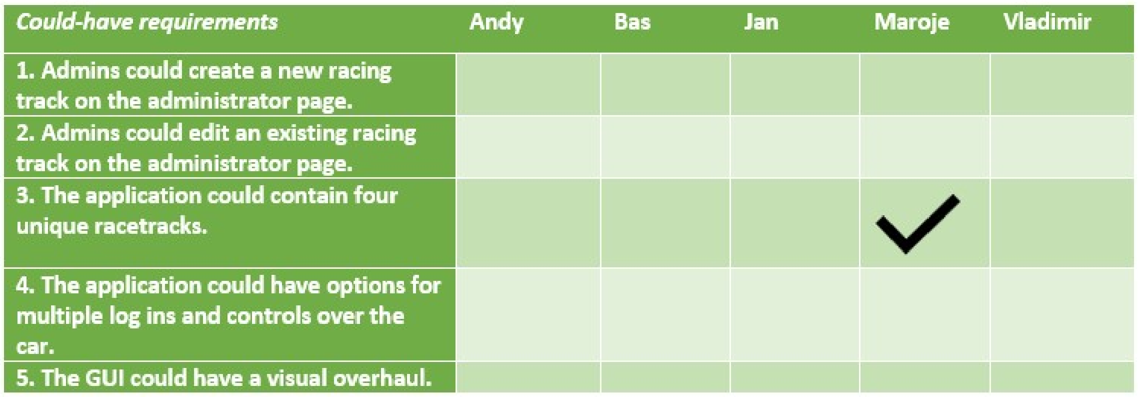
Must-have requirements	Andy	Bas	Jan	Maroje	Vladimir
1. Users must be integrated with the technical inspection questionnaire page prior to a race.				✗	
2. Users' car must automatically be lined up after every racing break.				✓	
3. Users must be able to keep their race settings after being disconnected or if the page has been refreshed.			✓		
4. The application (server) must be updated of the car progress upon reconnect.					
5. Users must be able to always change their race settings (e.g., not be blocked by other players' cars) [bug fix].			✓		
6. Users must be able to always see the lap counter and it must only be updated when a car passes the finish line. [bug fix]	✓				
7. Users' car must automatically stop during a pit stop.					✓
8. Admins must be able to view screens of other racers as a spectator.					✓
9. The application must contain a new administrator view for the current race details (i.e., breakdowns, yellow flags, penalties, car data)		✓			
10. The race must automatically continue to the next racing stage without the need of manual interference from an administrator.		✓			
11. The website must have a landing page for users that are not logged in yet.	✓				

Figure 12: The distribution of the 'must-have-requirements'.



Should-have requirements	Andy	Bas	Jan	Maroje	Vladimir
1. Admins should be able to change any website images or logo's by uploading them.				✗	
2. Admins should be able to change the introduction page text preferably through the GUI.				✗	
3. Admins should be able to select a different track for both the practice race and the real race.	*This was already possible*				
4. Users should be able to see a textual description upon hovering an unclickable button explaining why the button is unclickable.	✓				
5. Users should not be able to log into the same account at the same time.			✓		
6. Users that are not participating should be able to spectate the race from a general view.	✓				
7. The application should save relevant information on the server of the drivers.			✓		

Figure 13: The distribution of the 'should-have-requirements'.



Could-have requirements	Andy	Bas	Jan	Maroje	Vladimir
1. Admins could create a new racing track on the administrator page.					
2. Admins could edit an existing racing track on the administrator page.					
3. The application could contain four unique racetracks.				✓	
4. The application could have options for multiple log ins and controls over the car.					
5. The GUI could have a visual overhaul.					

Figure 14: The distribution of the 'could-have-requirements'.