



[www.github.com/Khazoda/tournamint](https://www.github.com/Khazoda/tournamint)

UNIVERSITY OF SURREY  
FACULTY OF ENGINEERING AND PHYSICAL SCIENCES  
COMPUTER SCIENCE BSC

COM3001  
FINAL YEAR PROJECT REPORT  
2021 — 2022

Title | Building an open source, easily deployable  
web application for organizing tournaments

Author | June Faleiro  
URN | 6532489  
Supervisor | Dr. Robert Granger

### *Declaration of Originality*

I confirm that the submitted work is my own work and that I have clearly identified and fully acknowledged all material that is entitled to be attributed to others (whether published or unpublished) using the referencing system set out in the programme handbook. I agree that the University may submit my work to means of checking this, such as the plagiarism detection service Turnitin® UK. I confirm that I understand that assessed work that has been shown to have been plagiarised will be penalised.

## *Acknowledgements*

I would like to take this opportunity to thank my supervisor, Dr. Robert Granger for his invaluable support and guidance during this project.

I would also like to express my gratitude toward the Computer Science department staff who have offered their support over the last four years.

Lastly, I wish to show my appreciation towards my family, my friends and my cat, who have all supported me throughout my degree.

## Abstract

With the release of the game Space Invaders in 1980 came the first large scale e-sports tournament. With 10,000 total attendees, this event marked the beginning of a cultural shift in sports entertainment. 40 years later in 2021, the League of Legends World Championship finals match boasted a staggering 74 million concurrent viewers.<sup>[1]</sup>

Tournaments are hugely popular across many different videogames, with participants showcasing their competitive spirit, and fans supporting their favourite teams and players. With League of Legends' biggest tournaments reaching viewership numbers comparable to the Super Bowl<sup>[2]</sup> and these numbers only predicted to grow in the following years, it's abundantly clear that demand for competitive tournaments is high and ever increasing.

As of the time of publication of this document, there are three main avenues for players & teams to compete in competitive league of legends tournaments:

- Large scale, professional tournaments run by Riot Games, the developers behind League of Legends (e.g., LCS,LEC,LCK,MSI,Worlds)<sup>[3]</sup>
- 3<sup>rd</sup> Party tournaments with smaller prize pools, run independently with support from Riot Games
- League of Legends' Clash (in-client random-opponent tournaments with no monetary prize pool)

The large-scale tournaments are only accessible to the best teams from each region in the world, and the two other options only allow a player to form their team with 4 other players and then compete against random individuals.

This therefore leaves two subsets of people; Those that wish to participate in a competitive tournament format but wish to be matched up against people they know, and tournament organizers that don't want to be subjected to the rigid rules that 3<sup>rd</sup> party tournaments need to adhere to. These people can be grouped as "casual tournament players".

Herein lies the use case for what this project aims to achieve — A functional web app, which is easily deployable, cheap to run and allows players to form teams, create tournaments and participate in said tournaments.

This dissertation will therefore lay out the design and implementation of a web app that meets these criteria, with a focus on web technologies used to design and deploy the application in addition to ease of set-up and use.

# TABLE OF CONTENTS

<i>Declaration of Originality</i> .....	2
<i>Acknowledgements</i> .....	3
<i>Abstract</i> .....	4
Non-Standard Terms .....	9
Acronyms & Abbreviations .....	9
<u>1 INTRODUCTION .....</u>	<u>10</u>
1.1 Background .....	10
1.2 Aims .....	10
1.3 Objectives .....	10
1.3.1 Research, Design Objectives .....	10
1.3.2 Implementation Objectives .....	10
1.4 Project Limitations / Risks.....	11
1.5 Success Criteria .....	12
1.6 Project Sections .....	13
<u>2 LITERATURE REVIEW.....</u>	<u>15</u>
2.1 Current Applications .....	15
2.2 Data Privacy .....	17
2.3 Web Frameworks.....	17
2.3.1 NextJS / ReactJS .....	18
2.3.2 Sapper / Svelte.....	18
2.4 Web Technologies .....	18
2.4.1 TypeScript .....	18
2.4.2 SASS .....	19
2.4.3 Axios .....	19
2.4.4 TailwindCSS, DaisyUI & React-Icons.....	20
2.4.5 Moment .....	21
2.5 Web Solutions.....	21
2.5.1 Vercel.....	21
2.5.2 Redis & Upstash.....	21
2.5.3 Riot Games API .....	21
2.5 Literature Review Conclusion .....	22
<u>3 PROBLEM ANALYSIS &amp; SPECIFICATION.....</u>	<u>22</u>
3.1 Current Systems.....	22
3.2 Requirements .....	24
3.2.1 Functional Requirements.....	24

3.2.2 Non-Functional Requirements.....	25
3.3 Feasibility Study .....	25
3.3.1 Technical Feasibility .....	26
3.3.2 Temporal Feasibility.....	26
3.3.3 Operational Feasibility.....	27
3.3.4 Feasibility Conclusion .....	27
3.4 Analysis & Specification Conclusion .....	28
<u>4 WORKPLAN .....</u>	<u>29</u>
<u>5 APPLICATION DESIGN .....</u>	<u>31</u>
5.1 Overview.....	31
5.2 Activity Flow .....	33
5.3 System Architecture .....	37
5.3.1 Choice of Platform .....	37
5.3.2 Wider System Architecture.....	38
5.3.2.1 System Security.....	39
5.4 Front-End Application Design .....	39
5.4.1 Overview.....	40
5.4.2 Landing Page.....	40
5.4.3 Tournamint Assets .....	42
5.4.3.1 Initial Logo Ideas .....	42
5.4.3.2 Further Branding.....	43
5.4.3.3 Colour Scheme.....	43
5.4.4 Profile Page .....	44
5.4.5 Settings Page.....	47
5.4.6 Team Page .....	47
5.4.6.2 Main Page (Initial).....	47
5.4.6.1 Team Creation .....	48
5.4.6.1 Team Joining.....	49
5.4.6.2 Main Page (Team Information).....	50
5.4.7 Tournament Pages.....	51
5.4.7.1 Main Page .....	51
5.4.7.2 Creating a Tournament.....	52
5.4.7.3 Joining a Tournament .....	53
5.4.7.3.1 Private Tournaments .....	53
5.4.7.3.2 Public Tournaments.....	53
5.4.7.4 Tournament Waiting Period .....	54
5.4.7.5 Tournament Seeding Phase.....	55

5.4.7.5 In-Progress Tournament .....	57
5.6 Data Storage Design .....	57
5.7 API Design .....	59
5.6.1 Next API .....	59
5.6.2 Axios .....	60
5.8 Programming Architecture .....	62
5.7.1 Object Oriented vs. Functional .....	62
<u>6 APPLICATION IMPLEMENTATION.....</u>	<u>63</u>
6.1 Application Implementation Choices.....	63
6.1.1 Platform .....	63
6.1.2 Framework & Language.....	63
6.1.3 Web Solutions.....	63
6.1.4 Web Technologies .....	63
6.1.5 Version Control .....	64
6.1.6 Development Environment .....	64
6.2 Global Data .....	64
6.2.1 Data Dragon & LoL API.....	65
6.2.2 TypeScript Interfaces .....	66
6.3 Front-End Implementation .....	67
6.3.1 Landing Page.....	68
6.3.2 Profile Page.....	72
6.3.3 Settings Page.....	75
6.3.4 Team Page .....	76
6.3.4.1 Main Page (Initial).....	76
6.3.4.2 Team Creation .....	77
6.3.4.3 Team Joining.....	78
6.3.4.4 Main Page (Team Information).....	79
6.3.5 Tournament Pages.....	82
6.3.5.1 Main Page (Initial) – No state .....	83
6.3.5.2 Create Tournament Page – No state .....	86
6.3.5.3 Join Private Tournament Page – No state .....	88
6.3.5.4 Join Public Tournament Page – No state .....	89
6.3.5.5 Main Page (Waiting Phase) – FILLING_UP, FULL.....	91
6.3.5.6 Main Page (Seeding Phase) – SEEDDED .....	95
6.3.5.7 Main Page (Tournament In-progress) - ONGOING .....	97
6.4 Tournament Logic.....	101
6.4.1 MatchTidbits.....	101

6.4.2 16-team Tournaments .....	102
6.5 Implementation Conclusion .....	108
<u>7 APPLICATION TESTING .....</u>	<u>109</u>
7.1 Functional Requirement Tests .....	109
7.2 Non-Functional Requirement Tests .....	110
7.2.1 Lighthouse Score.....	111
7.2.2 Deployment stability.....	114
<u>8 STATEMENT OF ETHICS .....</u>	<u>116</u>
8.1 LSEP Considerations.....	116
8.1.1 Legal Issues .....	116
8.1.1.2 Data Protection Act .....	116
8.1.2 Ethical Issues.....	116
8.1.2.1 Addictive Design & Algorithmic Bias.....	116
8.1.2.2 Data Confidentiality & Ownership of Personal Data .....	117
8.1.3 SAGE.....	117
<u>9 PROJECT EVALUATION &amp; CONCLUSION .....</u>	<u>118</u>
9.1 Evaluation .....	118
9.1.1 Areas of improvement.....	119
9.2 Future Work.....	119
9.3 Conclusion .....	122
9.3.1 Academic contributions.....	122
9.3.2 Final Word .....	123
References .....	124



## NON-STANDARD TERMS

Term	Description
Riot Games	The development studio that created and runs League of Legends
League of Legends	A competitive online multiplayer PC video game
Wild Rift	A competitive online multiplayer mobile video game based off League of Legends
Tournamint	The name of this project's proposed web application
Vercel	Name of a company and its' web hosting platform
Upstash	A data storage solution using Redis

## ACRONYMS & ABBREVIATIONS

Acronym/Abbr.	Meaning
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
TS	TypeScript
SASS	Syntactically Awesome Style Sheets
DOM	Document Object Model
OOP	Object Oriented Programming
API	Application Programming Interface
JSON	JavaScript Object Notation
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UI / UX	User Interface / User Experience
CORS	Cross-Origin Resource Sharing
SQL	Structured Query Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
SSR	Server-side Rendering
UML	Unified Modelling Language
SPA	Single Page Application
QOL	Quality of Life
LoL	League of Legends
NPM	Node Package Manager
PC	Personal Computer
CRUD	Create, Read, Update, Delete
XML	eXtensible Markup Language
JSX	JavaScript XML
SVG	Scalable Vector Graphic
IDE	Integrated Development Environment
VSCoDe	Visual Studio Code
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
PUUID	Player Universally Unique Identifier
CDN	Content Delivery Network
RegEx	Regular Expression
FIFO	First In First Out
SEO	Search Engine Optimization
CLS	Cumulative Layout Shift

# 1 INTRODUCTION

## 1.1 BACKGROUND

The existing tournament scene for League of legends caters only to professional teams and players that wish to be matched up against random opponents. There is a lack of free tools for players to set up tournaments to play with their friends rather than random opponents, and a lack of easily accessible, open-source League of Legends tournament-creation tools in general.

To address these shortcomings, this project aims to provide a solution in the form of an easily deployable, cheap to run and accessible web application that lets players create and play in tournaments with their friends.

To this end, all players hoping to be directly involved in a competitive League of Legends tournament environment will be able to do so.

## 1.2 AIMS

The primary aim of this project is to produce a functional, easily deployable, and cheap web application that anyone with an internet connection and web browser can set up. This application should let players create accounts, form & join teams, and create & join tournaments.

The following criteria give a broad overview of the aims of this project:

- Investigate web solutions that allow for easy deployment of a web application.
- Research web frameworks & technologies that will create a streamlined development process and polished end product.
- Create a project workplan in order to identify the timescale and risk associated with the project.
- Develop a web application using the web framework(s), technologies and web solutions selected from research.
- Test the functionality of the web application against this project's objectives.
- If possible, collect user feedback on the web application

## 1.3 OBJECTIVES

The following list of objectives serves as a checklist of goals that the project should meet. This list will be compared against in this document's evaluation. It will also extensively be used to build and supplement the project's requirements in the Problem Analysis & Specification section.

### 1.3.1 Research, Design Objectives

1. Research into web solutions for easy & cheap deployment present
2. Research around web framework options & web technologies for optimal development workflow and end-product usability and quality present
3. UML Class diagram indicating data dependency/ownership design present

### 1.3.2 Implementation Objectives

1. Web application can be deployed with minimal problems
2. Web application loads successfully in Google Chrome & Firefox
3. User can register account using their lol summoner name
4. User can log into their account using lol summoner name and passcode
5. User can log out of their account by clicking the log out button
6. User should be guided by the application into creating/joining a team
7. User should be guided by the application into creating/joining a tournament
8. User's league of legends account data (level, summoner icon, ranked information) should be displayed for them
9. User can change their personal username, biography, and favourite champion
10. Changing their favourite champion should change their profile background to said champion's splash art

11. Users' tournament statistics should be displayed
12. User should be able to toggle between light and dark mode
13. User should be able to change settings
14. Changing light/dark mode or settings should persist between browser sessions
15. User can create a team with customizable name, icon, and colour
16. User can join a team using team tag and invite code
17. User can leave a team
18. User can create a tournament with customizable name, size, start date/time, privacy, ID code and join code
19. User can join a private tournament using ID code and join code
20. User can join a public tournament from a list of public, non-full tournaments
21. Tournament members can see how much time is left until tournament start
22. Tournament creator can seed the brackets of a tournament once it's full
23. Tournament members can see bracket seeds once the creator has seeded the brackets, and it's an hour or less before the tournament begins
24. Tournament members can see the current state of brackets once the tournament is underway
25. Tournament creator can select which team wins a match, which updates this information for all tournament members
26. Tournament members can see match winners/losers
27. Tournament members can see which team they will be fighting next
28. User's team icon, summoner icon, level and username should be displayed in the navbar

These aims and objectives, along with the success criteria while also considering the project limitations, will be used to determine the project's success in the testing and evaluation sections.

## 1.4 PROJECT LIMITATIONS / RISKS

While developing any modern web application, it's extremely important to select a web stack consisting of technologies that empower the developer, minimize total bundle size and offer flexibility while maintaining a readable codebase. Due to the staggering variety of combinations of web technologies available, it is very likely that a web development project will run into problems and limitations at some point during the project development lifecycle.

Due to the nature of this project, there is also a significant time constraint which limits the scope of what the application could become with a sole developer.

With that in mind, this project will likely have the following limitations:

- **Polish:** In an ideal world, a perfect design that takes into account all web technologies, UML class diagrams and data flow in advance would lead to a robust, polished web application that isn't prone to failure or bugs. Of course, in reality the development process almost always leads to bugs and problems, some of which require large structural changes to the application. Due to the time constraint, if such problems arise it may be more beneficial to work on completing objectives rather than implementing large fixes for bugs.
- **Time:** A web application of this complexity with multiple interlinking forms of data that need to be accounted for and synced on the server side and client side may be too complex to complete in time.
- **3<sup>rd</sup> Party Availability:** This project aims to research and use a web technology / web technologies that offer a server-side service to the user deploying their distribution of this web application. If for any reason their pricing changes or they go out of business, or any other factor that affects the availability of their service, this project would need to be adapted to change to a different provider. To mitigate this risk, the project aims to use an abstraction of a specific server side web technology, so that adaptation to a new service built on top of the underlying web technology is possible and takes minimal effort. To assist in mitigating this risk, making the project open source can extend its' longevity and allow maintainers to update the project as time goes by.

## 1.5 SUCCESS CRITERIA

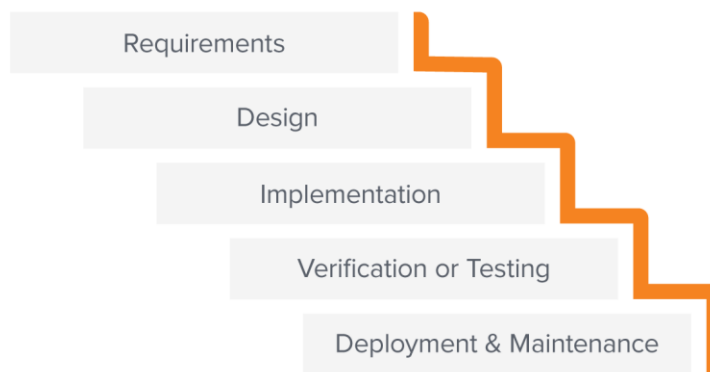
While taking into account the aims, objectives and project limitations, this section defines the high-level criteria the project must meet. This list will be used in the evaluation section to determine whether the project is considered a success.

- A functional, deployable, open-source web application has been developed which can let users sign up and create and participate in tournaments.
- Most objectives have been met, with an emphasis on implementation objectives which focus on the functionality of the application.
- Most application tests pass, and user feedback, if collected, is overall positive of the application.

## 1.6 PROJECT SECTIONS

Each main section of this report represents a stage of the project. This project will take a waterfall-based project lifecycle methodology approach, loosely following the structure below:

### The Waterfall Method



[1] *Waterfall Project Lifecycle Method*

Using the waterfall project lifecycle method as a guide, the following is the structure that this report will use:

---

#### 1 Introduction

The introduction introduces the reasoning for the project alongside its' aims and objectives. It sets out the project's limitations and the criteria for success.

---

#### 2 Literature Review

Research into existing technologies and solutions related to tournament organization applications is carried out in the literature review and brief justification is provided for the existence of a new tournament planning application in relation to the existing market.

---

#### 3 Problem Analysis & Specification

Analysis of the project involves creating functional and non-functional project requirements that can be evaluated against in the project evaluation section. These requirements are based on aims and objectives but are more low-level with a higher level of specificity. The technical feasibility of the project will also be determined in this section.

---

---

#### 4 Work Plan

**The work plan sets out a timeline of events for this project. It follows a similar structure to this list in Gantt chart form, blocking out a timescale for the project to be completed in.**

---

#### 5 Application Design

In order to develop a functional web application that meets the project's requirements, adequate research into web solutions, frameworks and technologies needs to be carried out. Additionally, design of data structures and data flow needs to be considered and presented in this section in order to support the application's implementation.

---

#### 6 Application Implementation

With the research and design work completed in the previous section, the application implementation section will focus on the development process and highlight how different parts of the application design have been translated into an actual application. Changes to the design will also be documented in this section, along with the considerations that led to them.

---

#### 7 Application Testing

In order to ensure that the application meets the project's requirements, as well as its' general functionality being adequate, testing must be performed. This section will include white and black box testing, and the results then used in the project evaluation section.

---

#### 8 Statement of Ethics

In software development, legal and ethical considerations are always important. This section will explain what considerations have been made to meet ethical and legal challenges, and why it's important that those challenges are met.

---

#### 9 Project Evaluation & Conclusion

The project evaluation will determine whether the final web application meets the requirements of the project based on the testing results. Further improvements to the application and considerations of the project's limitations will be discussed here. The conclusion will broadly review how the project went, what the next steps would be and any final words.

---

## 2 LITERATURE REVIEW

This section has two distinct parts; The first will focus on existing products/applications in the tournament planning space, as well as Riot Games' own implementation of tournament match making, Clash<sup>[5]</sup>. The second part will analyse web frameworks, technologies and solutions that would fit this project best.

### 2.1 CURRENT APPLICATIONS

In this section, 1<sup>st</sup> Party refers to Riot Games directly, and the services they provide directly. 3<sup>rd</sup> Party refers to independent groups/organizations that may or may not have direct approval to run tournaments by Riot Games. Approval refers to the application process that developers must go through to use certain parts of Riot Games' API. If Riot deems an application to be suitable to obtain access to tournament code functionality, it is a 3<sup>rd</sup> party Riot Games approved application, otherwise it is a 3<sup>rd</sup> party application without Riot Games' approval. Not obtaining Riot Games' approval for an application does not mean it goes against their terms of service to hold tournaments, it only means that the application won't have access to the tournament code API functionality.

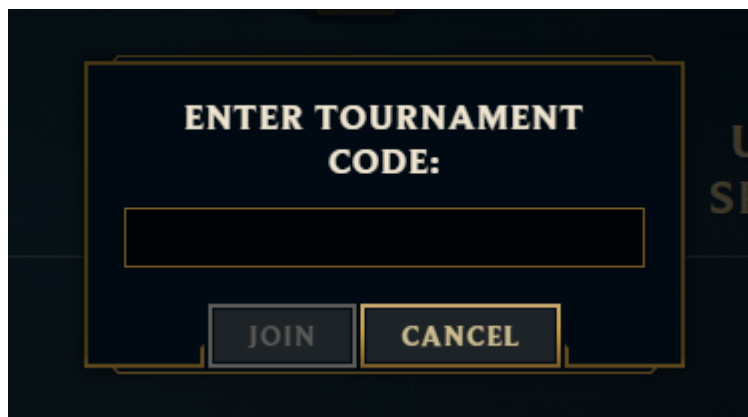
#### 1<sup>st</sup> Party Tournaments (Riot Games)

League of Legends' first world championship tournament was held in 2011 in Sweden<sup>[6]</sup>, and every year since there have been tournaments held for many regions including north America, Europe, and South Korea. These tournaments are held in-person with live audiences.

Riot Games uses a modified version of their tournament code feature for their tournaments in order to streamline their tournaments and make them failure-proof. The League of Legends client accessible to all players has the functionality to join a tournament using a tournament code, but this is for 3<sup>rd</sup> party tournaments only.

#### 3<sup>rd</sup> Party Tournaments (Independent Organizers, Riot Games approved)

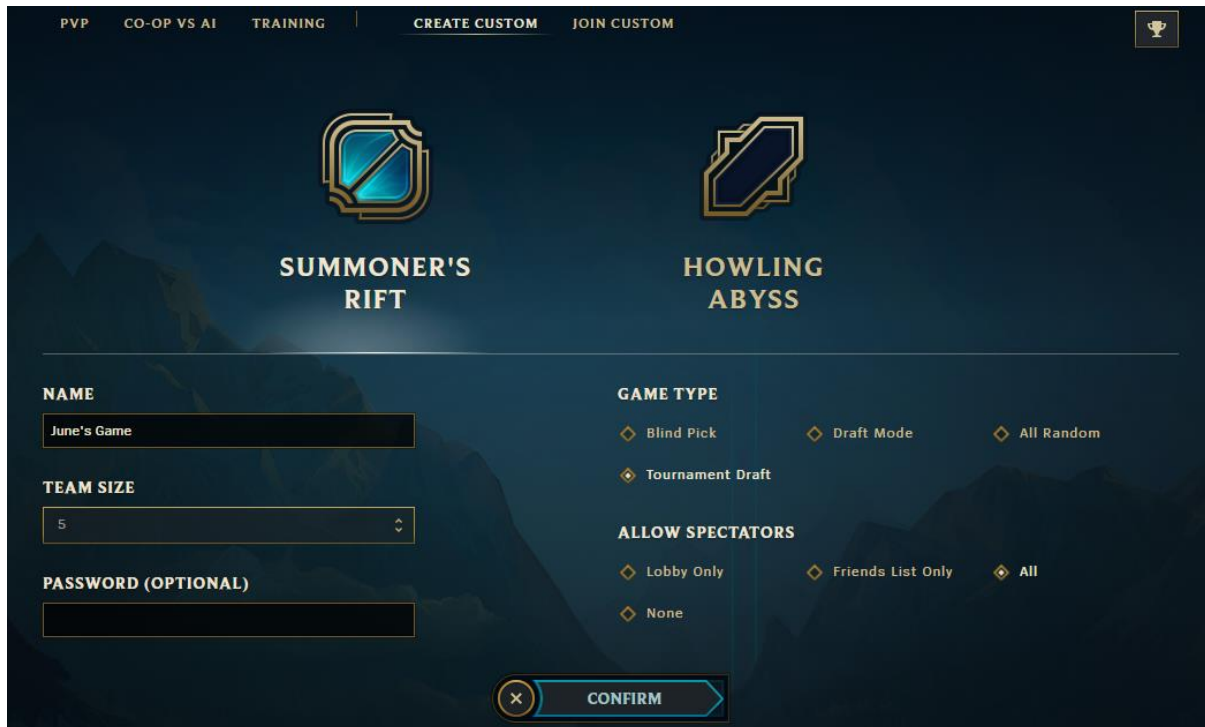
3<sup>rd</sup> party tournaments utilize either tournament codes or custom lobbies. Sites like challengermode.com<sup>[7]</sup> provide tournaments with small prize pools that use Riot Games' tournament code feature. Once a developer is approved, they can use Riot Games' League of Legends API to create tournaments that work with this code system.



[2] Tournament Code feature, League of Legends Client

### 3<sup>rd</sup> Party Tournaments (Independent Organizers, Not Riot Games approved)

If the tournament organization application is not approved by riot games, simply due to time or not meeting the requirements, then the only option is to use the League of Legends client's in-built custom game feature:



[3] Custom Game feature, League of Legends Client

The custom game feature even includes “Tournament Draft” as a game type, which allows players to use the tournament style picking/banning phase present in 1<sup>st</sup> party and other tournament code tournaments. This solution is used by sites such as [challonge.com](http://challonge.com)<sup>[8]</sup>, that let users make custom friendly tournaments, similar to what this project aims to accomplish.

### 1<sup>st</sup> Party Clash Tournaments (Riot Games)

The last option for playing League of Legends tournaments is with Riot Games’ Clash feature. Clash is a tournament style experience in which a player is able to form a team of 5 people, and Riot’s matchmaking algorithms will create a tournament and seed the brackets based on the team’s combined skill level. It’s a very sophisticated system, but it is randomized matchmaking. The only people a player can choose to play with are their four teammates, other teams will always be random. An additional drawback is that clash is only held every two weeks at the weekend. If a player wanted to compete in a tournament on any other day, they’d need to use a different solution.

The following table collates the differences between the mentioned tournament solutions that are currently available. The key is created to show how close a tournament solution matches up with this project’s aims, with Green/Circle/True meaning closely matching, and Red/Square/False meaning not matching closely.



Key | ● = True    ★ = Sometimes True    ■ = False    — = Not Applicable

Type of Tournament Solution	Accessible to all?	Players able to be matched with their friends?	Anyone can create a tournament?	Is the software open source?	Can anyone host their own distribution of the software?
1 <sup>st</sup> Party Tournament	■	—	—	■	■
Riot Approved 3 <sup>rd</sup> Party Tournament	●	★	★	■	■
Non-Riot Approved 3 <sup>rd</sup> Party Tournament	●	★	★	■	■
1 <sup>st</sup> Party Clash Tournament	●	■	■	■	■

From the table it's clear that, while good solutions to running your own tournament do exist, they fall short in key areas that this project aims to provide functionality in. This project's philosophy is, that if an independent group, organization, or individual wishes to host its' own tournament creation web application, then they should be able to. The nature of open-source web applications means that if the hosting party wishes to modify assets or functionality of the application, they are free to do so. In order to facilitate this kind of modification and distribution, this project will later go over software licenses and which one would best reflect these values.

## 2.2 DATA PRIVACY

In the current tech climate and as a direct result of capitalist interests, data is the most valuable resource that can be gathered on the web. In order to offset hosting and data storage costs, companies often collect data and sell it. Along with data collection, advertising provided by services such as Google Ads also helps offset these costs and lets these businesses turn a profit.

Both of these practices however are detrimental to the end user's experience. Advertisements are often intrusive, and the general population is becoming ever more security conscious when it comes to their data and how it is used. By making this web application open source and independently deployable, this project aims to remove these problems by leveraging potential free hosting and data storage solutions, as well as having no intention of generating profit.

All too often however does open-source code and software get taken to be used by corporations for profit. A recent example being amazon web services providing open-source projects as a paid service.<sup>[9]</sup> In this particular case it was legal, but the philosophy of open source software is always under attack by these kinds of actions. This is a real concern for a project like this, where the primary intention is to have a copyleft license with little restriction when it comes to modification and redistribution. As the project is intended to be deployed by anyone who wishes to, and have a license which permits source alterations, a profit-minded organization/company may take the software, replace the branding with their own and implement advertisements and data collection.

## 2.3 WEB FRAMEWORKS

Arguably the most important decision when creating a modern web application is the framework. The current trend, and best supported frameworks are JavaScript based. Using a web framework cuts down on workspace setup and development time. Many frameworks are opinionated, meaning that they employ certain conventions, styles and methodologies when it comes to the web technologies they come with, as well as folder structure, naming conventions, and of course programming languages.

Some frameworks also come with in-built API routing and other bells and whistles that let a developer focus on getting their application running and functional, rather than installing many smaller packages to manage different tasks. For this particular project, API routing in particular will be especially useful as data retrieval and transmission between both the Riot Games League of Legends API and the persistent server-side data storage will be important.

A huge benefit to popular frameworks in particular is that they are usually free to use as well as rigorously tested. Two popular JavaScript frameworks have been analysed for this project that have wide adoption and a feature-rich set of functions that would take a long time to replicate at a lower level.

### 2.3.1 NextJS / ReactJS

Next.js is a hugely popular JavaScript framework built on top of React.js, a framework originally devised by the team at Facebook, and maintained by an immense number of people through open-source software practices. While a minimal HTML/JS/CSS tech stack separates the three languages into separate files, with awkward dependency and script inclusion, React and Next allow developers to bundle all of a page's prescriptive (HTML), descriptive (CSS), and logic/interaction (JS) into one file using the JSX format. Next automatically manages routing for pages by detecting the application's file structure, making it much easier and faster to write interoperable code. It also allows developers to swap JavaScript for TypeScript, which is automatically compiled and hotswappable, which can be used by entering a single terminal command.

### 2.3.2 Sapper / Svelte

Sapper is a lesser-known framework built on top of the svelte, a hugely popular framework for web development in itself, but with far less wide adoption than React and Next. However, the Sapper framework's website actually touts itself as an improvement to Next (which it praises highly). Its' main improvement over Next involves bundle size and loading speed. In larger web applications this can be hugely important, but for a project of this size, it becomes less so. Both Next and Sapper offer client hydration with serverside rendering, SPA-focused design principles, and can both be easily deployed to Vercel, the free hosting platform.

Sapper however has some drawbacks specific to this project. Unlike Next, it doesn't offer an in-built API routing system, which can be remedied by using Axios or another JavaScript library that offers this functionality. Sapper also uses the svelte language, paradigm and conventions, which the developer for this project has less experience with than Next & React's conventions, including React components and hooks. Due to time constraints, it would not be prudent to learn svelte and sapper when Next can achieve an application with the same or better quality and functionality than Sapper.

## 2.4 WEB TECHNOLOGIES

Although a developer's choice of framework is very important, the importance of building a tech stack around said framework that fits the needs of the web application by utilizing other web technologies cannot be overstated. These technologies can range from language and function abstractions that empower developers and compile code at run/build time back into their original language to component libraries that offer boilerplate code for reusable parts of a web application and much more.

This project will examine a few web technologies that can improve the development experience and end product.

### 2.4.1 TypeScript

TypeScript, created by Microsoft who maintain it to this day, is a high-level programming language that compiles to JavaScript at run/build time, and is ubiquitous with the modern web. Almost all large-scale web projects that use JavaScript will be using TypeScript during the development process. While JavaScript has a reputation of being prone to bugs due to its' limited error checking feature set, resulting in many hard-to-pinpoint runtime errors, TypeScript addresses this problem by introducing a whole suite of tools that empower developers to write more readable, testable, structured code.

A hugely useful feature of TypeScript for example, is its' inclusion of interface structures. Commonplace in other programming languages such as Java and C#, interfaces allow data to be described, and

compilers can perform error checking according to these interfaces. For this project, interfaces will be invaluable to describe each data structure and its' relation to other data structures (e.g. User→Team→Tournament).

Interfaces would not be as useful as they are without another TypeScript feature that is also commonly found in languages like Java and C#: static typing. “Statically typed languages perform type checking at compile time”<sup>[10]</sup>, meaning that the developer must define data types before their code is run, rather than the compiler or interpreter assigning a type at runtime based on what data is stored in the variable. This leads to errors being easier to catch inside the IDE instead of resorting to runtime browser development tools, as well as leading to better code readability.

#### 2.4.2 SASS

SASS stands for “Syntactically Awesome Style Sheets” and has been a staple of web developers' toolkits since its' inception in the mid-late 2000s. SASS has many features, but two that stand out are its' variables, and nested styling.

SASS variables are similar to CSS variables in that a value can be assigned to them, and then by referring to the variable later in code, the value can be reused in multiple places. SASS variables have features such as value mapping where dynamic styling options can be defined and then used, but by and large they exist for a single purpose: Make reused CSS code easier to read and faster to write. CSS variables allow for mutation, while SASS variables do not. This makes SASS variables perfect for imperative variable declaration, i.e. variables that don't change. If a developer then wishes to add declarative variables, they can use CSS variables, which are less readable, but visually distinct from SASS. Segmenting these two types of variables is advantageous as they have distinct purposes, and in a project such as this which aims to be open-source, code readability and ease-of-understanding is important.

The second useful feature of SASS is nested styling. In CSS, all styling rules exists at the top level of a .css file. Child styling rules can be declared, but the code is much wordier. SASS solves this problem by allowing nested styles that affect a parent's children:

```
.parent {
  background: black;
}

.parent.child {
  background: yellow;
}
```

[4] Vanilla CSS

```
.parent {
  background: black;
  .child {
    background: yellow;
  }
}
```

[5] SASS

This not only improves code readability by having child dependents easily accessible, but it also cuts down on the amount of class name/id text that's needed to declare styles for a child. This becomes very apparent when a child is nested multiple levels down and that level of specificity is required.

#### 2.4.3 Axios

As discussed in 2.3 *Web Frameworks*, API calls will be an essential part of this project due to its' requirement of needing to communicate with both a backend server and Riot Games' League of Legends API. Axios is a solution which offers API calls using JavaScript. It utilizes CORS (Cross-Origin Resource Sharing), an HTTP mechanism that as the name implies, allows a program to fetch a resource from a different origin to that which it is hosted on. This will be useful for the Riot Games API calls, as all requests to said API will be HTTP GET requests.

Axios can be used for other kinds of API calls too of course, but POSTing, PATCHing and GETting from the serverside database may be better suited to a solution such as Next.js's in-built API calling

functionality. In this way the API call logic could be segmented between the two resource locations with Axios and the inbuilt API calling of Next.js, improving readability due to the logic segmentation.

Axios also features helper functionality such as a `.getAll()` function which gets multiple forms of a resource, likely with different identifiers at the same location. This could be useful for example when retrieving user data for all members of a team in this project.

#### 2.4.4 TailwindCSS, DaisyUI & React-Icons

Front-end development can be a very time-consuming process for developers; Page layouts need to be constructed with html elements, styled with CSS, populated with components and data, and much more. Thankfully many tools exist to make this process less time intensive and let the developer focus on getting their application running. TailwindCSS is a perfect example of a time saving tool. By writing shorthand class names for components, CSS classes are automatically applied. For example, to make a div that uses the `display: 'relative'` property that fills its container's size, instead of writing the css code on the left, the developer simply writes the class names shown on the right to achieve the same effect, but much faster:

```
.element {
  display: 'relative';
  width: '100%';
  height: '100%';
}
```

[6] Vanilla CSS code

```
<div class='relative w-full h-full'></div>
```

[7] TailwindCSS code

TailwindCSS also comes with many other features that make development quicker, including a system to implement different themes, a custom plugin system that allows developers to customize TailwindCSS's default property values to their own needs and much more. While other solutions such as BootstrapCSS and MaterialUI have more opinionated sets of components with premade styling, TailwindCSS sets itself apart by giving the developer total design freedom, while still decreasing the time it takes to build their application tremendously.

Two other tools that fit into the umbrella of decreasing the time it takes a developer to create styled elements are DaisyUI and React-Icons. DaisyUI is a component library, and React-Icons is an icon library. A component library offers components, that is styled pieces of HTML+CSS code that can be invoked with a single line of code. React component libraries specifically, which DaisyUI is, often allow for modifiers to their properties, passed in as React props, offering an additional level of customizability. Using a component library such as this would be hugely advantageous to this project, as a lot of components need to be created in a short amount of time, and DaisyUI is a sleek looking modern React component library.

React-Icons on the other hand is much more of a granular solution. Instead of scouring the web for high quality images of icons, or svgs that fit with one-another stylistically, an icon library such as React-Icons offers sets of icons that can be added to a project with a single line of code as a React component, similar to DaisyUI's components. These icons can receive props for size and styling, and are exported only when used in the application, reducing overall bundle size.

Together, these three web technologies make the development process much quicker and easier, and should certainly be considered for this project.

### 2.4.5 Moment

Moment is a more niche library that may become very useful for this project in particular. JavaScript has data structures (`Date()`) and functions for said structure that allow for date and time reading and manipulation, but code for this is often verbose and prone to errors due to the global nature of dates, times, time zones and date/time formats.

Moment on the other hand is a comprehensive library that can convert from and to JavaScript `Date()` objects, and perform manipulation and conversion on these objects. For this project, where tournaments will have a start date and time, which needs to be communicated to all members of the tournament, a library like this that simplifies and makes more robust the process of dealing with dates and times is a must.

## 2.5 WEB SOLUTIONS

2.3 *Web Frameworks* and 2.4 *Web Technologies* focused on tools that empower the development experience, but web solutions takes a slightly different direction by exploring tools and solutions that can help this project reach its' functional aims and objectives. From web hosting to server side data storage and retrieval to the Riot Games API, these solutions will become vital for this project to reach its' potential.

### 2.5.1 Vercel

G2.com describes Vercel as 'a cloud platform for static sites and Serverless Functions that fits perfectly with your workflow. It enables developers to host websites and web services that deploy instantly, scale automatically, and requires no supervision, all with no configuration.'<sup>[11]</sup> This description summarizes well Vercel's purpose in the market, and why it may be such a good fit for this project.

This project aims to be a fully open-source, cloud hosted, easily deployable non-profit solution for an individual or group. Vercel offers a powerful free plan to all with an incredibly easy deployment process; Interested parties simply have to fork this open-source project to their own GitHub account, select it on Vercel's dashboard, and Vercel will take care of the rest. It even provides serverless functions, a solution which could prove incredibly useful for having a cloud-based server to store user, team, and tournament data. Vercel will offer the hosted site a domain (`site-name.vercel.app`) which can be accessed by anyone with an internet connection, but it can also be redirected to a user's own domain name. Domain names are very cheap to purchase and maintain, with prices ranging from around £2 -£15/year. This would be a fantastic solution for anyone wanting to host this project themselves.

### 2.5.2 Redis & Upstash

Redis describes itself as 'The open source, in-memory data store used by millions of developers as a database, cache, streaming engine, and message broker.'<sup>[12]</sup> It essentially is a language that developers can use to store, retrieve, and modify data. As it is a low-level solution, it has been widely adopted and modified by different parties and companies that offer services using Redis as an underlying technology.

One such company is Upstash, who offer 'Serverless Data' for different technologies including Redis. By using Redis' syntax, developers can use HTTP requests to communicate with an Upstash serverless data store. This acts as a quasi-serverside-data store, leaving structure and convention up to the developer. Thanks to this philosophy, it is possible to create a linked list database using client side API call logic coupled with the data storage on Upstash, and the best part is that like Vercel, it offers a completely free solution – up to 10,000 API calls/day.

### 2.5.3 Riot Games API

Riot Games offers developers a comprehensive API that can interface with many of their games' social features. For League of Legends, player names, rank information, profile picture information and level can all be accessed using API calls. The application process to become a Riot certified developer is simple, and the number of API calls allowed is astronomically high.

In order to make the user experience feel more personal for this project, as well as show other people who they will be playing against, utilizing the Riot Games League of Legends API will be essential, as it lets other players see ranks and statistics of their opponents before their match begins.

## 2.5 LITERATURE REVIEW CONCLUSION

There exist multiple different tournament organization applications that target a wide range of people, but all of them are either centralized, closed source implementations with a profit incentive, or 1<sup>st</sup> party implementations that do not have full accessibility to all players or are only available at certain times. It is therefore justifiable for this project to result in the creation of a tournament planning web application which unlike its 'competitors' is open source, free to redistribute and modify, easy and cheap to host, and lets players create tournaments that all their friends or members of their community can participate in.

By utilizing a web framework, web technologies and solutions, this project can meet these requirements and be completed in a shorter timespan without sacrificing quality, and in many case, improving upon it. Using web solutions such as Vercel and Upstash with Redis will permit easy, free deployment of the application to anyone, and the Riot Games League of Legends API will take care of a lot of user information, which they would otherwise need to enter themselves.

## 3 PROBLEM ANALYSIS & SPECIFICATION

Clearly identifying the problem with the current systems by analysing their capability in comparison to this project's aims and objectives is an important step to take before the design process. Building a set of requirements as a specification for this project can then be achieved by asking how the current's systems' problems can be solved, and how this project intends to do so. These functional and non-functional, testable requirements based on the problem analysis as well as the aims and objectives of the project can then be evaluated for being met during the evaluation section of the report.

As this project aims to create a web application based on open-source principles and easy, free/cheap deployment, the current systems discussed in *2.1 Current Applications* will be evaluated, and requirements for this project will be directly influenced by their shortcomings.

Finally, a feasibility study will be conducted to determine whether this project can be completed on time and to a satisfactory standard.






### 3.1 CURRENT SYSTEMS

There are 3 current systems (excluding 1<sup>st</sup> party large-scale tournaments) that all have shortcomings that this project intends to address and improve upon. These are

- The 1<sup>st</sup> Party in-client Clash tournament system
- The 3<sup>rd</sup> Party Riot-Approved tournament systems
- The 3<sup>rd</sup> Party Non-Approved tournament systems

For the sake of simplicity and to avoid excessive verbosity, these tournament systems will be referred to as Clash, Riot-Approved and Non-Approved in this section.

#### Clash

<b>Accessible to all?</b>	<b>Players able to be matched with their friends?</b>	<b>Anyone can create a tournament?</b>	<b>Is the software open source?</b>	<b>Can anyone host their own distribution of the software?</b>
				

Clash tournaments are an enjoyable, first party experience accessible to all league players. People can form a team with 4 of their friends and be matched by Riot's matchmaking algorithms with other teams of similar skill level to compete for in-game rewards. Clash tournaments are held twice a month across Saturday and Sunday.








Compared to the aims and objectives of this project, clash performs quite badly. It is a first party, closed source service with limited time availability and no control over the teams players will be fighting against. Tournaments are automatically created with no oversight – everything is managed by Riot Games.

This project on the other hand aims to be a third-party solution that is open-source and freely deployable by anyone. Users should be able to make tournaments whenever they want, and players should be able to invite their friends to fight in tournaments against them. Tournaments in this project will be overseen by the creator of the tournament, instead of an automated algorithm.

Though most of what makes clash unique will not be helpful to adopt in this project, the team formation is something worth drawing inspiration from. Clash teams allow the team leader to customize the team's tag, team name and team icon from a set of premade icons. This lets each team differentiate itself from another.

#### Riot-Approved

<b>Accessible to all?</b>	<b>Players able to be matched with their friends?</b>	<b>Anyone can create a tournament?</b>	<b>Is the software open source?</b>	<b>Can anyone host their own distribution of the software?</b>
				






Riot-Approved tournaments are often websites that allow users to sign up and participate in tournaments with small prize pools. While this solution is accessible to all, and potentially lets people play against their friends and create tournaments, it is not always the case, and these solutions are not open-source, deployable applications, but rather for-profit services provided by a company.

Compared to the aims and objectives of this project, riot-approved tournament solutions fare better than Clash due to their relative flexibility, round-the-clock availability and the ability to sometimes create tournaments.

Where riot-approved tournament solutions fall short compared to the aims and objectives of this project, however, are in their business philosophy. They are never open-source, deployable solutions, and are usually for-profit, whereas this project aims to be open-source, deployable and not for profit. Riot-approved tournament sites often sell user data (legally), and display ads to generate revenue.

While revenue-generating tactics are not something this project wishes to emulate, the format of riot-approved tournaments is something to draw inspiration from. The phases of team creation, pre-seeding, seeding and then the tournament running in an asynchronous fashion are all things this project should aim to emulate as they function well as a robust system as well as being user friendly and easy to understand.

#### Non-Approved

<b>Accessible to all?</b>	<b>Players able to be matched with their friends?</b>	<b>Anyone can create a tournament?</b>	<b>Is the software open source?</b>	<b>Can anyone host their own distribution of the software?</b>
				

Non-approved tournament solutions encompass 3<sup>rd</sup> party tournament applications that are not approved by riot, meaning that they cannot use the tournament code feature. This project will therefore most likely resemble a 3<sup>rd</sup> party non-approved tournament application, with some key differences.

Although existing non-approved tournament solutions are accessible to all and share traits with riot-approved tournament solutions such as being able to be matched with friends and creating tournaments sometimes, there are still no solutions which offer open-source, deployable distributions.

This project aims to be a close match to non-approved tournament solutions by requiring the individual or group who are setting up their distribution of this web application to only need a few things:

- Riot API key (obtainable for free in under 3 days)
- Free Vercel account to host their distribution
- Free Upstash account to host the Redis serverless backend data storage

## 3.2 REQUIREMENTS

With the analysis of existing solutions complete, functional and non-functional requirements can be extrapolated and built that will be used further in the project to compare against, and determine whether the project has been successful or not.

From this point forward, the word “Tournamint” will also be used to reference the web application that this project is building. The name will come up in branding for the project and will be used during the design and implementation phase.

### 3.2.1 Functional Requirements

Functional requirements refer to functionality of this project’s web application. These testable requirements are specific and measurable and are listed in the following table. Their order follows a general user interaction flow, meaning the order of operations which a typical user would perform when using Tournamint.

<b>Requirement</b>	<b>Description / Outcome</b>	<b>ID</b>
Register Account	User can register account using their League of Legends summoner name and a 6-digit passcode	01
Log In to Account	User can log into their account using League of Legends summoner name and previously set 6-digit passcode	02
Log Out of Account	Once logged in, user can log out of their account by clicking the log out button	03
Assist user in creating/joining a team	When not in a team, user should be guided by the application’s interface into creating/joining a team	04
Assist user in creating/joining a tournament	When not the owner or participant of a tournament, user should be guided by the application into creating/joining a tournament	05
Display LoL account data	User's league of legends account data (level, summoner icon, ranked information) should be displayed for them in the navbar and profile page	06
Editable user information	On the profile page, user can change their personal username, biography, and favourite champion	07
Persistent user information	On the profile page, user’s personal information should persist between browser sessions and across different browsers if they log in in a new location	08
Favourite champion splash art change	On the profile page, changing their favourite champion should change the user’s profile background to said champion's splash art	09
Tournamint statistics shown	On the profile page, users' tournamint statistics should be displayed	10
Theme Swappable	User should be able to toggle between light and dark mode with a single button press	11
Changeable settings	On the settings page, user should be able to change settings by clicking toggle switches	12
Persistent QOL data	Changing light/dark mode or settings should persist between browser sessions but not across different browsers	13
Create team	User can create a team with customizable name, tag, icon, and colour	14
Join team	User can join a team using a team’s tag and invite code	15



Leave team	User can leave a team by clicking a button on the teams page	16
Create tournament	User can create a tournament with customizable name, size, start date/time, privacy, ID code and join code	17
Join private tournament	User can join a private tournament using ID code and join code	18
Join public tournament	User can join a public tournament from a list of public, non-full tournaments	19
See time until tournament start	Tournament members can see how much time is left until their tournament begins	20
Seed tournament	Tournament creator can seed the brackets of a tournament once it's full and 1hr or less before tournament start	21
Brackets visible (seeding)	Tournament members can see bracket seeds once the creator has seeded the brackets, and it's an hour or less before the tournament begins	22
Brackets visible (running)	Tournament members can see the current state of brackets once the tournament is underway	23
Update tournament state	Tournament creator can select which team wins a match, which updates this information for all tournament members	24
See tournament state	Tournament members can see match winners/losers	25
See next opponent	Tournament members can see which team they will be fighting next	26
Team information visible	User's team icon, summoner icon, level and username should be displayed in the navbar	27

### 3.2.2 Non-Functional Requirements

Non-functional requirements exist in a broader sense than functional ones. They describe and are used to evaluate the overall operation of this project and other external requirements (e.g., legal). The following table lists these non-functional requirements:

Requirement	Description	ID
Deploy web app	Web application can be deployed using Vercel in a short amount of time	01
Load web app	Web application loads successfully in Google Chrome & Firefox	02
Loading speed adequate	Lighthouse score which encompasses First Contentful Paint, Largest Contentful Paint and Cumulative Layout Shift should be above 85.	03
Adequate deployment accessibility	Uptime of a deployment of Tournamint should be high with minimal downtime. Updating the deployment to a new version of Tournamint should also have minimal/no downtime.	04
Legal requirement	The web application should comply with GDPR guidelines and legislation.	05

## 3.3 FEASIBILITY STUDY

Masanja describes a feasibility study as “a systematic plan and analysis of the sustainability of a project”<sup>[13]</sup>. In other words, an exploration of the viability of a proposed project. The benefits of such a study include identifying the positive and negative outcomes of completing a project. This information lends itself to producing a cost-benefit analysis, weighing up the merits and demerits of a project so that an informed decision on its’ production can be made.

Due to the nature of this project, a cost-benefit analysis will not be performed in a financial sense, as no investment is required for the project, and due to its open-source nature, no returns on any investment would be expected. One resource, however, must be used to produce the Tournamint web application – time. A feasibility study then, will examine the limitations and risks of the project, and weigh the potential of project failure in comparison to its’ successful outcome.

In summary, this project's feasibility analysis will focus on the technical, temporal and operational aspects.

### 3.3.1 Technical Feasibility

Web applications that are developed with multiple web technologies are always subject to the potential of incompatibility, failure and a myriad of possible bugs at both the compilation and runtime stages.

Technical feasibility examines whether there is a possibility of catastrophic failure in the combination of web technologies, solutions and underlying framework that could lead to the project resulting in failure.

Firstly, the framework should be examined. The two possibilities mentioned in *2.3 Web Frameworks* are Sapper/Svelte and NextJS/React. When examining Sapper, the conclusion was made that Next would be a better fit as a framework for this project due to previous experience as well as a fully formed feature set perfect for this project's web application. With this reasoning, going forward Next will be used in further analysis as the framework that this project aims to utilize.

Next has a long history, and the added benefit of being able to utilize the entire React and Node ecosystem, including the Node package manager (npm). All the web technologies mentioned in *2.4 Web technologies* are distributed via npm, making them all directly compatible with the Next framework. Next has sophisticated in-browser debugging support and error checking, which coupled with Typescript's conventions and structure can lead to a streamlined development process with fewer bugs and roadblocks.

As a preliminary test, Next has been installed and using npm, all the aforementioned web technologies have been added to a project. After a test page was created, it has been determined that there are no direct incompatibilities between these technologies and Next. Therefore, technical feasibility is positive.

### 3.3.2 Temporal Feasibility

This web application has a sole developer and a time constraint. In order to produce a product that meets this project's requirements, feature implementation must occur within its' time constraint. Temporal feasibility in this case refers to the viability that Tournamint can be completed on time with a feature set that meets most requirements, and is functional.

With prior experience working on other projects that use Next, along with a similar number of 3<sup>rd</sup> party web technologies to this project's proposed number, and a time constraint of around 2 months of development, it is entire feasible for Tournamint to be completed on time to a good standard. After a preliminary investigation into the largest time sinks during development time, it's been assessed that the stages of development that will likely be the most costly are as follows:

- Class structure and interoperability
- Connecting functionality to serverless data storage
- Interfacing with the Riot Games API

#### Class structure and interoperability

Tournamint will have three distinct classes that interact with one another; The user, team and tournament classes. While object-oriented programming is slowly falling out of favour for many developers, it is still incredibly useful when segmenting logic and distinct data groups, then linking them in a form of relational database. Further atomization of classes can be achieved for a tournament, as each tournament will have rounds and matches as child classes that it will be related to. Correctly designing and implementing the structure required for this functionality will be time consuming.

#### Connecting functionality to serverless data storage

The class structure and associated data structures need to be persistently stored in a serverless data storage environment, and then correctly handled by the user's browser client when reading and writing to the store. This functionality needs to be implemented so that data faults don't occur that could corrupt the persistent storage. Successfully implementing this logic will also be a time-consuming task.

### Interfacing with the Riot Games API

Riot Games' League of Legends API is a new technology and foray into 3<sup>rd</sup> party API connections that this project's developer is not yet familiar with. Learning how to correctly access each API route to retrieve the correct data, and then converting said data to JSON, interpreting it, displaying it to users in the client and potentially storing this data in the persistent data storage will all take a significant time to understand and correctly implement.

### Temporal Feasibility Conclusion

The three main concerns when it comes to temporal feasibility are real challenges to overcome, but are by no means the only factors that could affect the feasibility of the project when it comes to time spent designing and developing them. The front-end design of Tournamint, as well as functional page routing, modal logic implementation, account login/registering and much more are all going to play a large part in the viability of this project. After careful deliberation however, all the features mentioned and more should be completable within the 2-month timeframe.

#### 3.3.3 Operational Feasibility

One of Tournamint's core design principles is its open source, freely deployable nature. Using web solutions explored in section 2.5, Tournamint source code should be able to be forked, modified and deployed with free/cheap rates. Operational feasibility therefore refers to the viability that this web application can meet these expectations, and the situations/risks in which it may not be able to.

After preliminary research into both Redis and Upstash' implementation of Redis, it seems due to the user-friendly nature of their product that implementing a user's own deployment should not be difficult. Upstash allows login using Google, Facebook and other well-established accounts, and by using the API key that they provide to the user and placing it in the user's distribution of Tournamint, the web application should function correctly. This minimal-effort process is very accessible and shouldn't create a roadblock when concerning the persistent data storage of the application.

The developer of this project has previous experience working with Vercel's free cloud hosting solution, and it can therefore be said that it is a very simple to use, accessible product that only requires a user to fork the repository of Tournamint, change whatever code they wish, and then deploy it directly to Vercel using their GitHub interoperability. Any future changes to their forked code, or the original code of Tournamint, once handled using git best practices, can be redeployed to Vercel automatically, and pushed to the main release of their distribution without any input from the user themselves, if they so wish.

The main risks associated with operational feasibility arise due to third party shortcomings. If Vercel or Upstash no longer offer their service due to operational costs or business changes in the company, then Tournamint would no longer function. Thankfully, due to its' open-source nature, a new service provider can be found and Tournamint can be adapted to work with the new service provider. The original distribution, maintained by the original developer and potentially contributors, can then have this new provider added as to the original repository through pull requests by the community.

Due to the flexibility of Tournamint's design, and the ease of setup and use of Vercel and Upstash, Tournamint is in an advantageous position when it comes to operational feasibility.

#### 3.3.4 Feasibility Conclusion

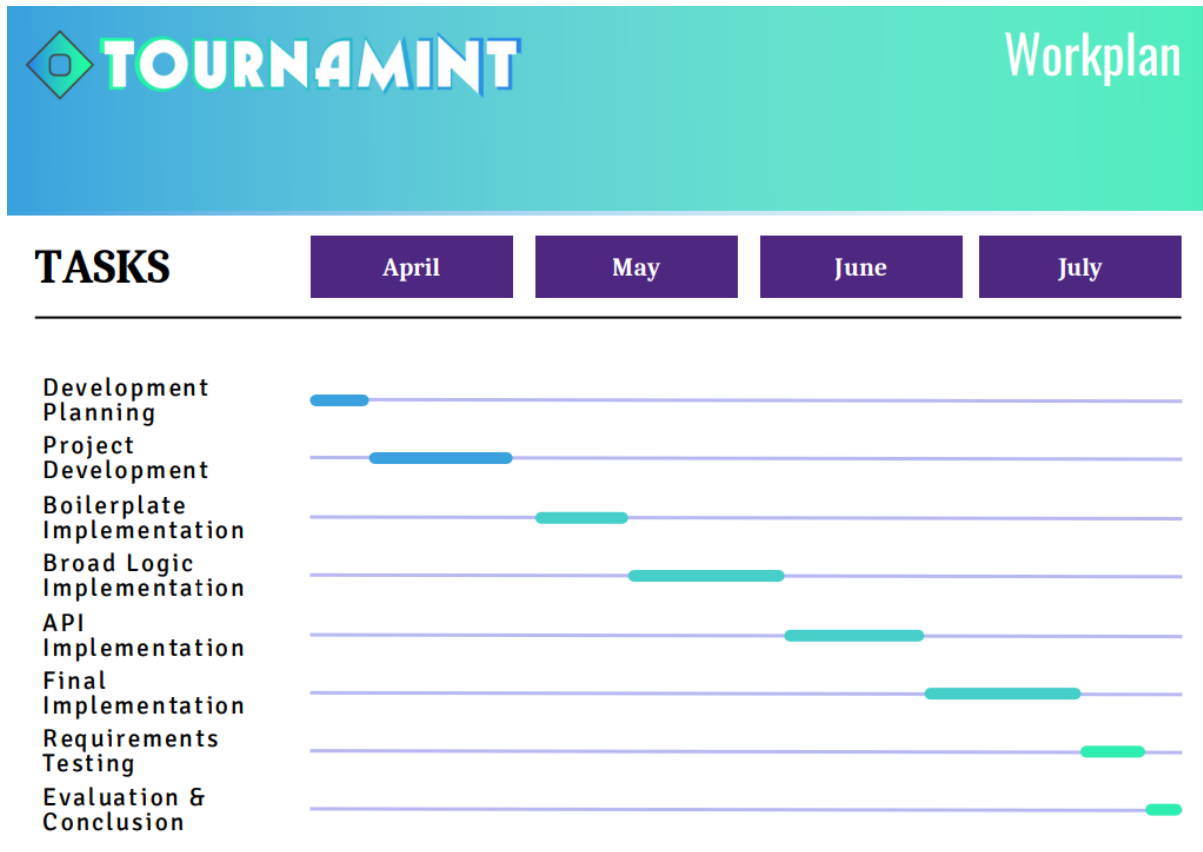
Taking into account the technical, temporal and operational risks to the project's viability, it's clear that while the risks are present and real, there are solutions and mitigations which lead to the project being feasible to complete on time, to a good standard and with all operational technologies in place, functional, and able to be used by anyone that wishes to create and manage their own Tournamint distribution.

### 3.4 ANALYSIS & SPECIFICATION CONCLUSION

After analysing the shortcomings of current tournament creation and participation systems in relation to the aims and objectives of this project, a set of functional and non-functional requirements has been constructed that lay the groundwork for the application's design and subsequent implementation. With these new requirements, a feasibility study has been conducted to determine the viability of this project in relation to three distinct factors. The feasibility study suggests that the project will be able to be completed to a satisfactory standard and within the loose time constraints. With the viability of the project being confirmed, a workplan should now be constructed to determine what stages of the project should be completed by when in order to stay on track for completion within the time limit.

## 4 WORKPLAN

An initial workplan in the form of a Gantt chart is laid out in the following figure. This workplan details what work should be done and by what time from this point in the project onward. In order to meet the time constraint and produce a functional web application, this workplan includes broad task descriptors that serve as a broad overview of each section of the development/implementation/testing/evaluation process.



[8] Tournamint Workplan

Planning for the project's development section and the development section itself is aimed to be completed by the end of April. Following from the rough 2-month time scale given to the implementation of the web application, this work plan allocates ~2.5 months from May-mid July for the total implementation, broken up into stages.

Boilerplate implementation refers to a rough outline of the web application with mainly HTML and CSS, with placeholder pages, routes and some minimal interaction elements such as opening and shutting modals.

Broad logic implementation encompasses a saturation of the boilerplate. This is a time-consuming section that involves writing JavaScript code and using aforementioned web technologies to build the model classes and integrate them into the application. Interaction with the site is the main focus of this time period.

API Implementation runs off the back of the broad logic, and involves both the League of Legends API being implemented into the application, as well as the logic surrounding the Upstash data storage. Lastly, the final implementation section ties the previous sections together by interconnecting disconnected

functionality, and making certain that the application functions correctly in relation to the project's requirements.

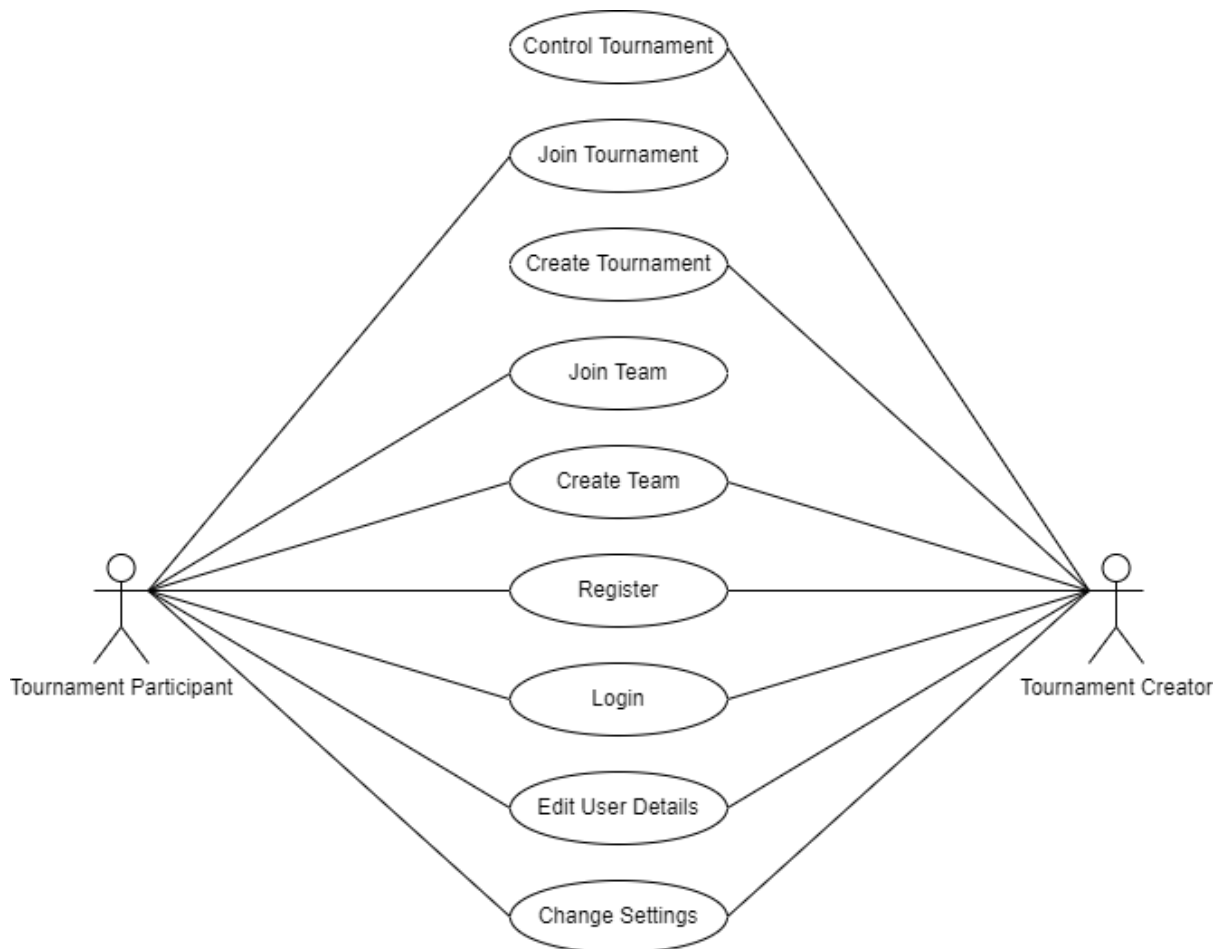
## 5 APPLICATION DESIGN

The design of the application is the last stage before its' implementation can begin. Components that comprise the functionality of the project's web application are designed, and functionality planned. The design structure of the persistent data storage will also be created and shown.

### 5.1 OVERVIEW

To formulate a comprehensive design, a high-level overview of the application's systems is initially required. As Tournamint has a high degree of user interaction, a use case diagram can be helpful to illustrate the types of actors using the application, and the functions that they perform.

Below is such a use case diagram. The two actors that use Tournamint are participants and creators. As the names suggest, tournament participants partake in the tournaments that the creators create and run. There is of course also commonality between the functions of each actor, such as account registration and team creation/joining.



[9] High Level Use Case Diagram

Though much of Tournamint's functionality is accessible to both creator and participant, their actions deviate from one another after the team creation/joining phase. The tournament creator will create a tournament with their custom values, while the tournament participant will either search for a public tournament from a list to join, or join a private tournament using the tournament's ID code and lobby access code.

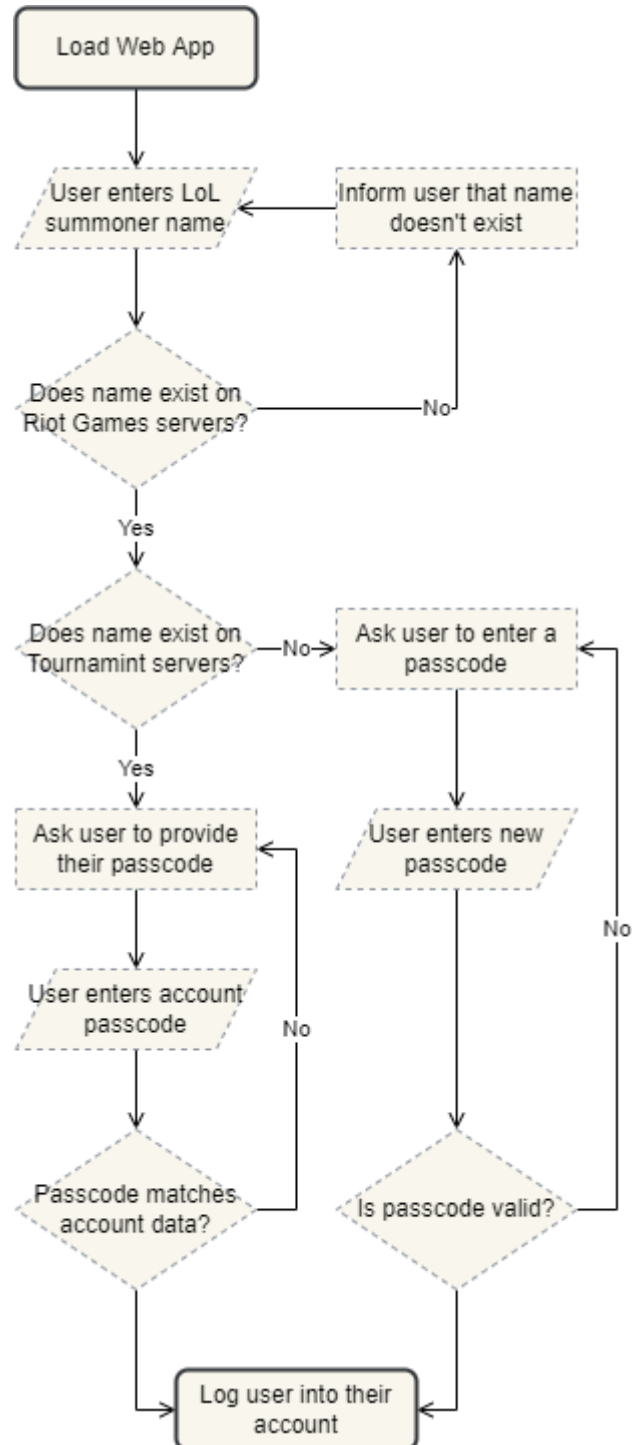
Tournament participants have a minimal user input during the tournament running phase itself, with most operational input being performed by tournament creators in terms of creation and subsequent updates to rounds and matches as a tournament progresses. This does not however, mean that participants have no CRUD (Create, Read, Update, Delete) operations to perform in the application. All Tournamint users can

create, read and update their account information, and read some account information from members of the same tournament. Additionally, users who create a team are able to delete the team once created. All of these operations should be reflected in the data store on Upstash.



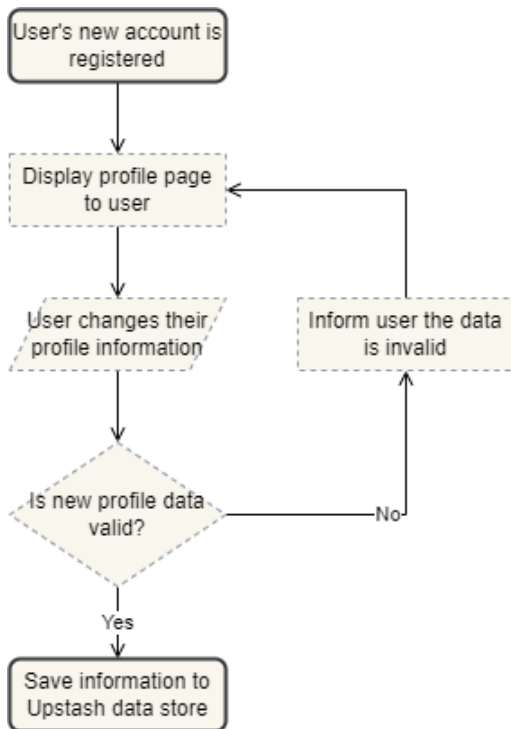
## 5.2 ACTIVITY FLOW

Tournamint will be a considerably user-involved application. From account creation/management to team creation/joining and the tournaments themselves, there is a large amount of user input that affects the overall system and what other users see. To visualize the 'flow' of activity that a user may perform, multiple activity diagrams are presented. The first describes the initial actions a user will take when loading Tournamint. They are greeted with the Login/Register page and are encouraged to either log in or register for a new account. The processes and checks performed by the web application when signing in consider both the Upstash data store and the Riot Games' API information:



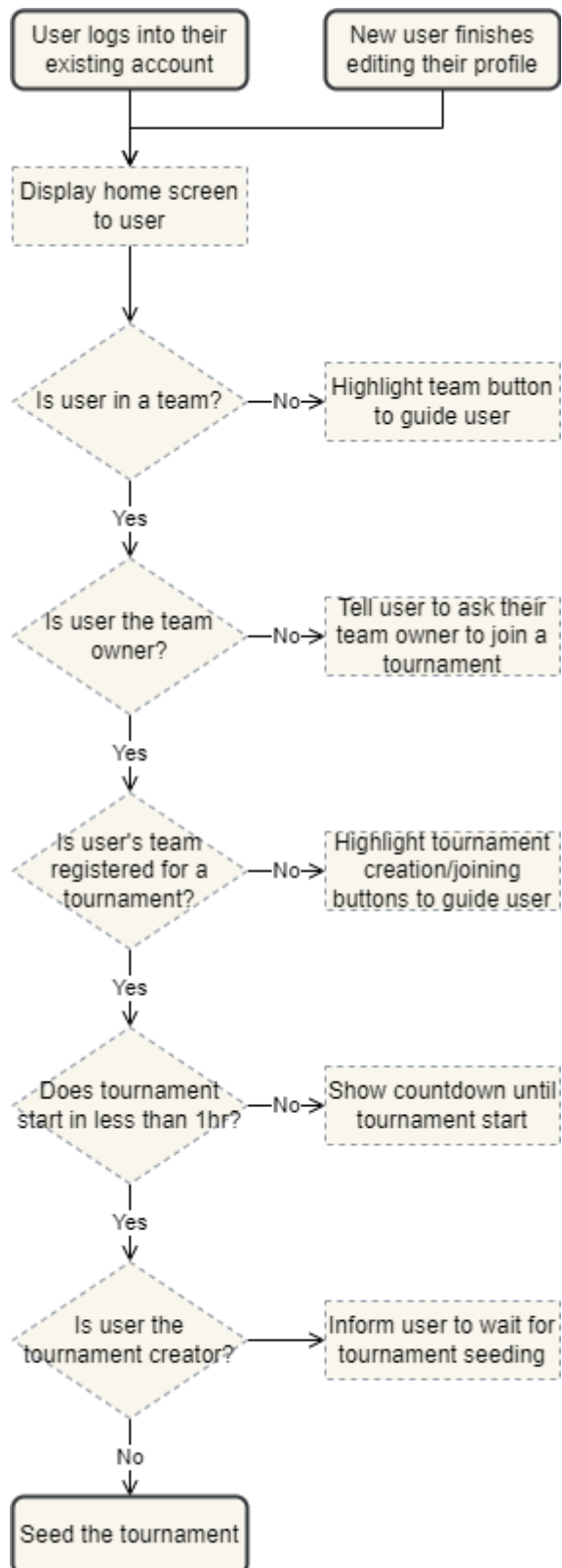
[10] Initial Activity Flow

Once a user has registered for, or logged into an account, they are greeted with one of two pages; The homepage if a user just logged in, and the profile page if they registered. The philosophy behind this decision is that a newly registered user would likely wish to edit their personal information, such as username, biography and favourite champion. The profile page is also shown so that users are aware of its' existence, should they wish to return to it in future to see/edit their information or check their statistics. With this thought process in mind, the following diagram shows the short user activity flow of a newly registered user.



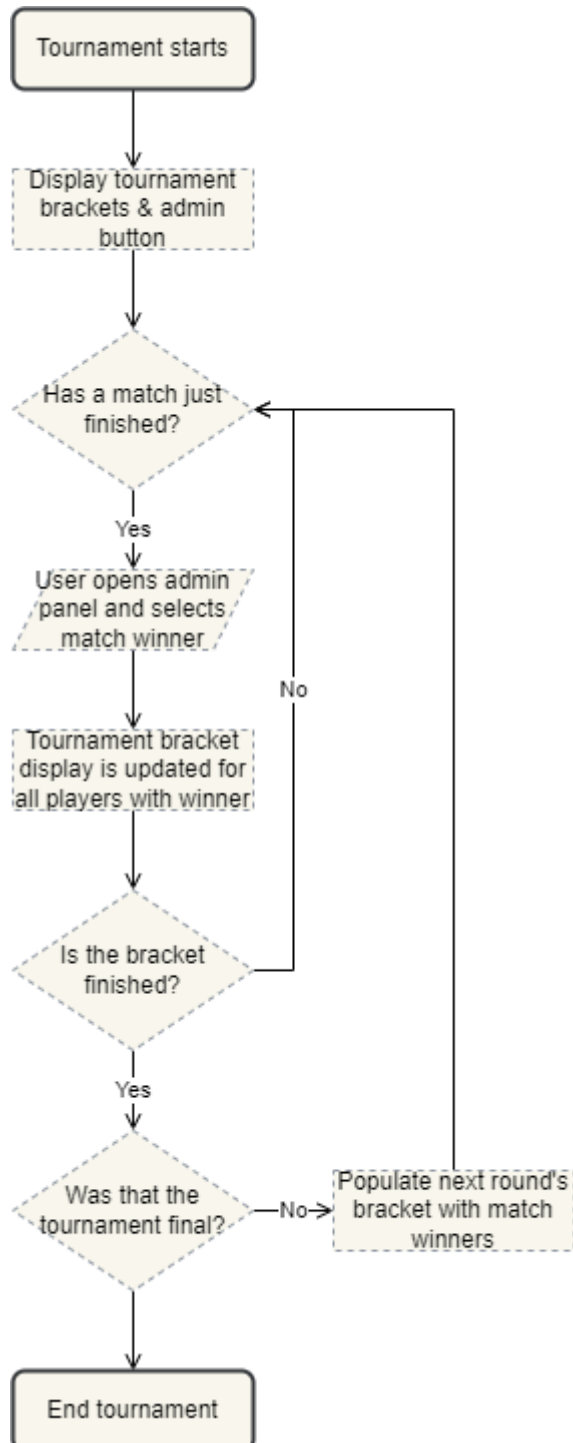
[11] New User Activity Flow

Once a user has either completed the activity flow by saving their new information, or simply left the profile page to visit the home page instead, they are faced with new options to pursue. The next diagram accounts for these options, which are available to both the registered user who has left the profile page, and a user who has just logged in.



[12] Pre-Tournament User Activity Flow

The previous activity flow showed the steps a user would take to reach the point at which a tournament is seeded and ready to begin. The next diagram picks up where the prior one left off, and shows the activity flow of a tournament creator once their tournament starts. Crucially, there is no activity flow diagram present for tournament participants, because there is no user input on their part. Once the tournament starts, the outcome of matches is controlled by the creator, and participants see live updates of the state of the tournament.



[13] Tournament Creator Activity Flow

These activity flow diagrams describe almost all of Tournamint's planned functionality involving user interaction. Noticeably they are quite high level and from a user's perspective, without exploring the intricacies of API access and data storage/structure. From this foundation more specific functionality can be designed that works around the principles explored through this map of user interaction.

## 5.3 SYSTEM ARCHITECTURE

When considering the architecture of a proposed system, it's important to evaluate what platforms would be most beneficial to the end-user. For this project, there is an argument to be made for both a focus on desktop/laptop/tablet and on smartphones. The rationality behind these two groupings lies not in the way a user physically interacts with an application, but rather the size and aspect ratio of their device.

### 5.3.1 Choice of Platform

While desktop computers, laptops and tablets have three different potential input methods (mouse,touchpad,touchscreen), in real world use cases this isn't as large a difference as one would expect when referring to web based applications.

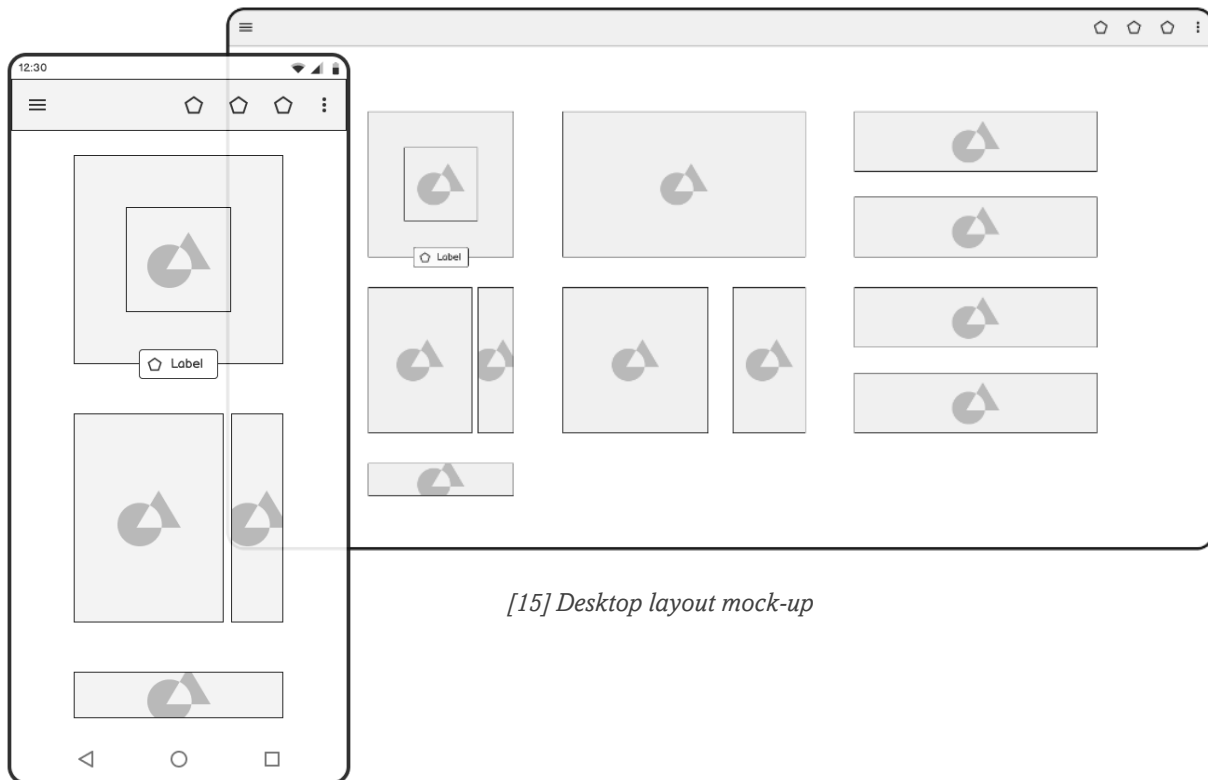
Most web applications are controlled through data input forms which have been designed to adapt to the user's input method, buttons/links that involve a simple click or tap, and scrolling, which is performed with ease on all three platforms, and requires no additional work to be implemented on the side of the developer. While platform specific guidelines and best practices do exist in the school of web design, for the purposes of this project which focuses on functionality more than the design side of the application, they will not be explicitly considered.

So then, with the first group of input devices considered, what then is the benefit of web applications on a smartphone, and how does a developer need to adapt their application to be suited to the smaller aspect ratio and screen size?

While desktops/laptops/tablets usually have a horizontal aspect ratio, smartphones are usually used vertically. This allows for far less horizontal screen space for elements, and forces a developer to redesign their application to work for mobile. Due to the challenges that arise with converting a horizontal, standard web application to a vertical format, many developers and guidelines agree that developing mobile-first is always the better option.

In his article, "How to Take the Right Approach to Responsive Web Design"<sup>[14]</sup>, Kevin Powell breaks down the multiple reasons behind a mobile-first approach, not least of which is that mobile layouts tend to be simpler than desktop layouts, making them a good starting point for a developer putting down their ideas. This simple starting point is also helpful when subsequently developing for desktop, as a developer can, by using media queries for different screen size/ratio breakpoints, enlarge elements and allot different spaces for them as the screen size becomes bigger.

In order to illustrate this point, below are two mock-ups for Tournamint's profile page. The mobile mock-up is distinctly simpler than the desktop setup, as the elements of the page are placed below one another in a single line; This can be achieved with minimal CSS. Having larger elements on mobile also helps with being able to tap elements, as user's precision on mobile devices is limited compared to desktop due to the smaller screen and touch-based input.



[15] Desktop layout mock-up

[14] Mobile layout mock-up

Tournamint is a web application whose primary userbase is people who play League of Legends. The game is exclusively for PC, so the system architecture of Tournamint should logically cater to that platform. However, considering the advice from the wider web development community, developing mobile-first and adapting to desktop is even more advantageous than only developing for desktop. This reason alone is enough to warrant dual-platform development of Tournamint; There is an obvious advantage to also developing the web application for mobile however – being able to quickly glance at information.

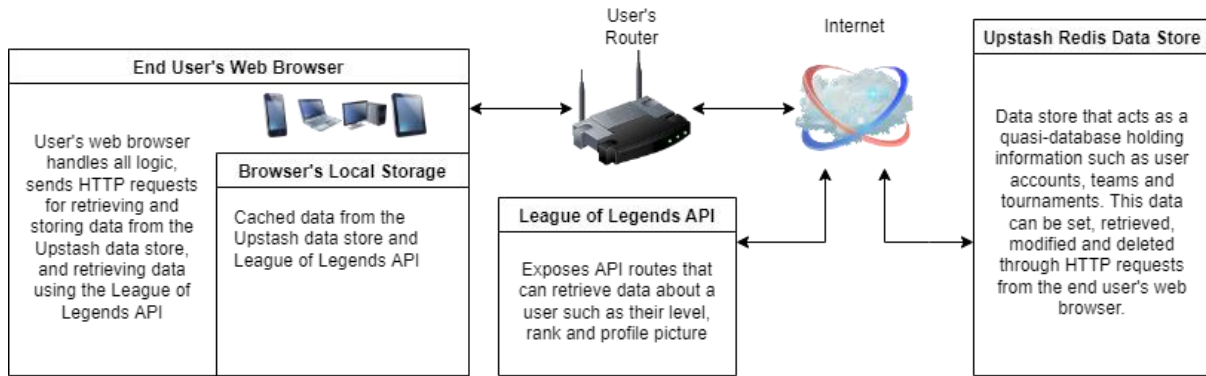
If a user has signed up to a tournament for example, they may wish to quickly check when that tournament is and who is hosting it. If they are not at their computer at the time, it would be advantageous to easily allow the user to check the information from their phone.

With these considerations in mind, Tournamint will be a mobile & desktop web application, developed mobile first and then adapted to desktop using media queries and other techniques.

As an aside, although League of Legends (PC) has been playable and in active development since 2009, Wild Rift, the mobile version of the PC game was released in 2020. This project will not cater to Wild Rift however, as the API access to the game is still in its' infancy, and many systems are different to the PC version. The choice of supporting Wild Rift or League of Legends PC is down to numbers. Although Wild Rift boasts an impressive monthly player count of 20 million players<sup>[15]</sup>, League of Legends PC is currently sitting at 180 million<sup>[16]</sup>.

### 5.3.2 Wider System Architecture

With the choice of both mobile and desktop web application as the platform to develop for, work can now be done regarding the design of the web application's structure of data, data storage and data transportation. The following is a rudimentary diagram detailing the movement of data from the user's web browser to the two data sources/sinks, namely the Upstash data store and Riot Games' servers using the League of Legends API. Of note is also the use of Local Storage, a technology superseding cookies that is present in all modern web browsers under a unified design language using JSON that can be used by Tournamint to cache data for use on the client.



[16] Broad architecture diagram

Important to note is the lack of traditional security, usually in the form of SSL encryption. Data is sent and retrieved from the Upstash data store without encryption, but there is a rationale behind it.

### 5.3.2.1 System Security

Tournamint is intended to be a freely deployable, freely modifiable piece of software. This means that by its' very nature it cannot be guaranteed to have adequate security. Although it is technically and legally feasible to implement encryption (as an example using TLS with Upstash) and requiring said encryption to remain unchanged within the software's license, it is never guaranteed that any party deploying this application will adhere to the guidelines, sometimes due to malicious intentions, but often simply due to negligence or lack of knowledge/ability. A simple design principle to combat this problem is therefore to design the application in such a way that data encryption is not needed in the first place.

By adopting this principle, Tournamint will not be susceptible to dangerous data breaches or man-in-the-middle attacks as the data is not valuable. To achieve this end-goal, a set of rules should be followed:

- The application should use publicly available data from the League of Legends API wherever possible to supplement user-provided data.
- User provided data should be cosmetic or trivial in nature, i.e., custom display name, biography, favourite champion, not home address, legal name etc.
- User should be clearly informed that a passcode for their account is not stored securely, and to not use a passcode that they use anywhere else.

In a typical web application that offers account functionality, a user must set a username, provide an email address and set a password. An unfortunate reality is that the password is often re-used from another site, and their email address is information that must be encrypted. Tournamint instead aims to use a user's existing League of Legends username (which is available publicly) along with a passcode that adheres to strict requirements (e.g., 6 characters, all numbers), to prevent a user using a password that they have used before. Due to the nature of this application, an email address won't be required.

Rather counter-intuitively, this approach will make the application safer, as a user is informed about how their data is stored, instead of them blindly trusting a system that may mismanage their data or be susceptible to data breaches.

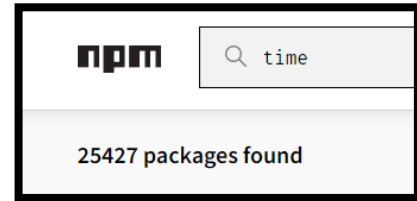
## 5.4 FRONT-END APPLICATION DESIGN

Arguably the most impactful aspect of a web application regarding the end-user's experience is the design and implementation of its' front-end. Over the last decade alone, the rise in JavaScript frameworks and the staggering abundance of technologies that work with them has drastically changed the landscape of the web. Unified design languages such as Material Design by Google have shaped the way we interact with the web on a daily basis, and advancements in the underlying web technologies of popular browsers mean that more is possible in the realm of web development than ever before.

This rapid advancement has not come without its' drawbacks, however. A report by Alex Zito-Wolf in March 2022 talks about JavaScript bloat, citing that "Since 2017, JavaScript file weight has increased 78 percent in total kilobytes for desktop sites, while the number of JS requests is up 17 percent"<sup>[17]</sup>. While

these numbers can partially be attributed to an increase in overall complexity of websites over the last 5 years, much of it is due to bad practice and package bloat. There are so many packages in fact, that a simple search with the keyword “time” returns over 25,000 results.

Thankfully, many packages include code that bundles only the components/code that a developer has actually imported in the final release of their application. As Tournamint will only use a small number of packages, bloat is not a main concern. Packages such as react-icons, which was examined in section 2.4.4, are very explicit in their bundling, a developer must manually import each icon they intend to use, and only those icons will be bundled during the build step. DaisyUI, a component library also explored in the same section also exhibits this functionality.



[17] Screenshot showing npm packages

It is then no surprise, that with this immense popularity in developing frameworks, packages and other technologies that the front-end development of an application holds great importance.

#### 5.4.1 Overview

This section of the report focuses on the front-end design of Tournamint – that is, what the end-user sees, and how they interact with what they see. Mock-up designs of each of Tournamint’s pages are shown, along with an explanation of how a user will interact with the elements on the page. Although front-end best practices are hugely important for commercial projects, and are also used in this project, they are not the main focus, as they trend more towards web design than development. This section will also not dive into specific algorithmic solutions either, focusing mainly on the presentation of the application, and the black-box user interaction. During the explanation of the 5.4.7 *Tournament Pages* section however, there will be a more comprehensive discussion of algorithms due to the interconnected nature of the tournament’s logic and the rest of the front-end that supports it.

#### 5.4.2 Landing Page

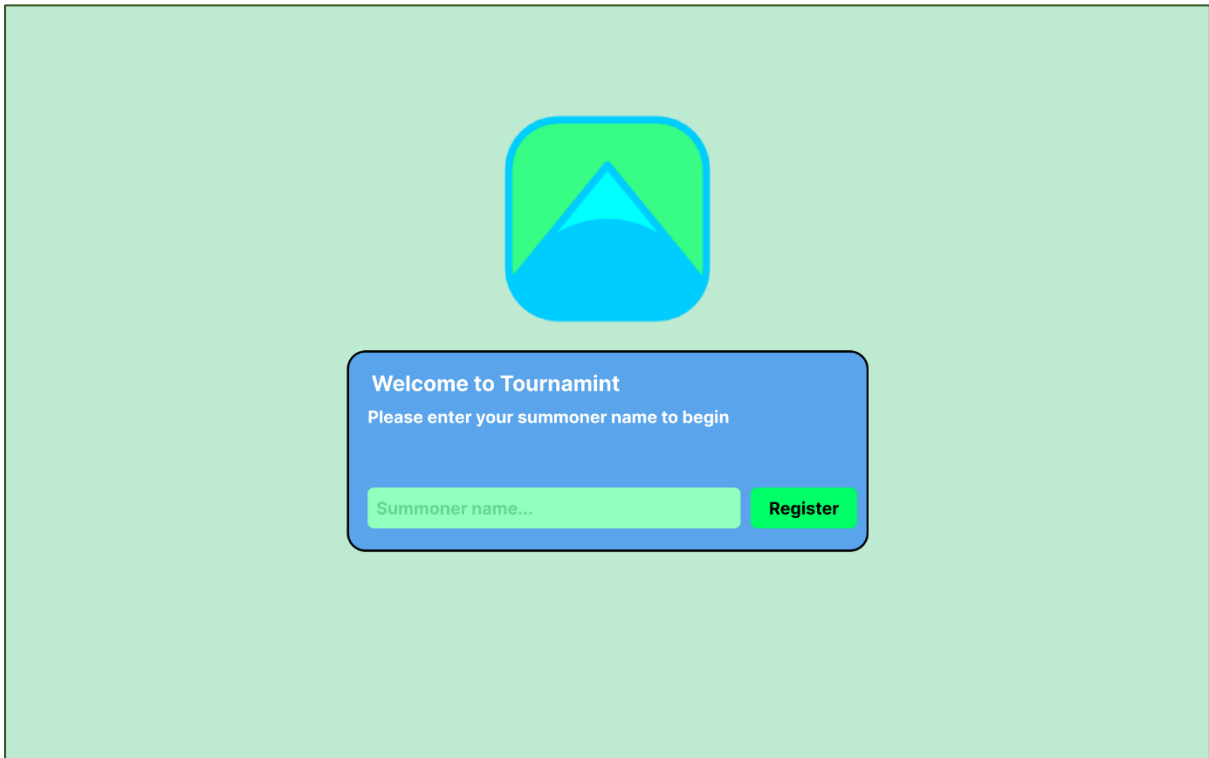
A landing page is the first thing a user sees when visiting a website. Its’ purpose is to clearly show the website’s branding if it has any, and draw the user in. It shouldn’t be overcomplicated or filled with information, but rather entice a user to take the first step in using the website. This is commonly referred to as a “Call to Action”, an easy-to-spot user input of some sort that, once interacted with, lets the user access the rest of the website and its’ functionality. This call to action can be anything from a short sign-up form to a simple button that shows the rest of the website.

Tournamint requires only one piece of information from the user first and foremost; Their summoner name. This is the username that is displayed to everyone when they play League of Legends. By requesting this single piece of information, the web application can perform API requests to access the user’s information and start creating an account for them before they have performed any other action.

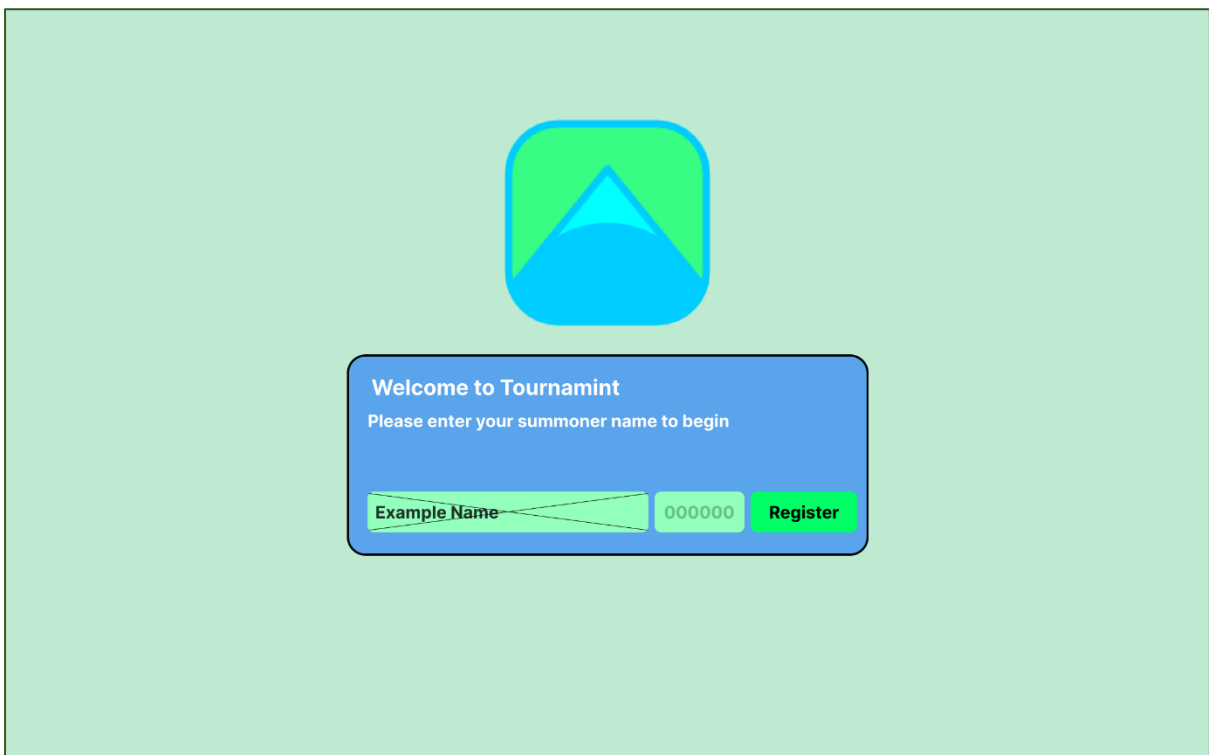
There is however one other piece of information the web application will need from the user, a passcode. This 6-digit numeric value will let a user log into their account once they have claimed their username. Many modern websites have adopted a strategy to having this type of form-first access to their site. It’s surprisingly simple, and only involves showing a single input at a time. What this means is that initially, the passcode input element will be hidden from the user, and only revealed once the user has entered their summoner name. This way, the user only has one action that they can, and need to perform at once. Not only is this approach user friendly, it also helps with user retention, as they are more likely to enter a passcode as they have already spent time entering their summoner name, and exiting the website at that point would feel like a waste.



With these design principles in mind, a mock-up of the landing page is shown below.



[18] Initial tournamint landing page mock-up



[19] Tournamint landing page 'enter passcode' mock-up

The first mock-up shows the initial landing page that a user is greeted with, while the second shows how a passcode box is shown *after* the user has entered their summoner name. The X across the 'Example Name' denotes the element becoming disabled, that is non-interactable, maintaining the philosophy that only one element should have the user's focus at once on the landing page.

### 5.4.3 Tournamint Assets

If it wasn't clear from the two landing page mock-ups, the Tournamint logo is a draft version. Therefore, before proceeding further with the front-end application design of each page, a more finished set of branding assets will be developed. As mentioned in the overview, the front-end design of the application is not this project's focus, but is still considered throughout the design phase and will be used during the implementation of the web application too.

For brand assets though, it's important to first consider what the application aims to be, and how its' name and message will be received. For this project, "Tournamint" was chosen as a name. It is a combination of the words tournament and mint. While simple, the easily recognizable words instantly bring imagery to anyone who hears the name. The word tournament instantly tells them what the application aims to be, which is a tournament application of some sort. The word "mint" is more abstract in its' function, and in-fact has nothing directly to do with the application's function. It exists only to drive the branding of the application. "Mint" evokes other words such as "fresh, clean" and colours such as "white, green, blue". The site can be designed in such a way that complements these ideas, and brand assets can be constructed with an identity formed from the colours and ideas evoked by the name.

#### 5.4.3.1 Initial Logo Ideas

As seen in the landing page mock-up, the green-blue colours will be the main brand colours for Tournamint. Taking this idea further, to the right is an evolution of the Tournamint logo over many iterations.

The first logo design takes the "M" from the word mint, and combines it with the colour blue. This is a very early initial idea, but it has helped inform the second design.

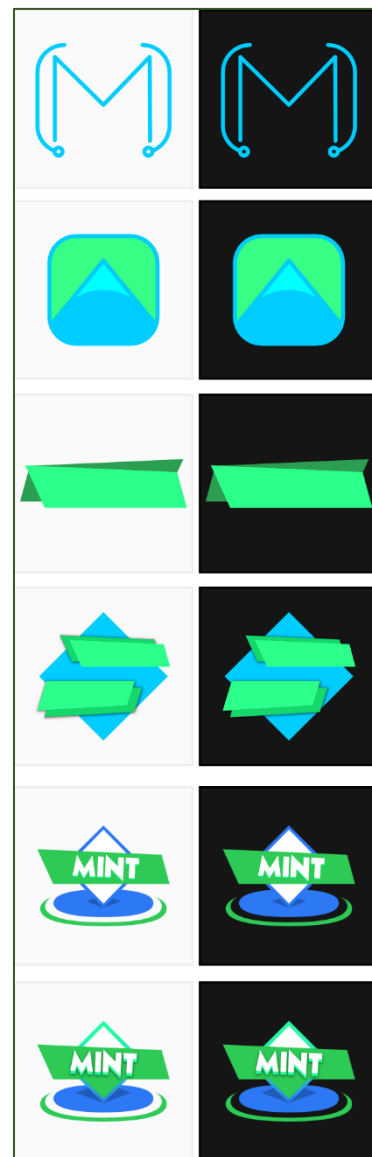
The mountain design actually implements the first draft, flipping the "M" upside-down to create the distinction between the mountain peak and the green sky. This logo doesn't feel distinct enough, however, and doesn't communicate the "tournament" part of Tournamint clearly enough.

Starting with a fresh idea, the third design takes inspiration from tournament bracket designs for its' sharp-edged rectangle design resembling a single match.

Taking the previous design further, two match-rectangles combines with the blue colouring create a primitive yet more distinct logo.

Extrapolating the base elements of the previous design as well as adding a rounder base gives this logo. It's distinct, easy to read and combines the colour language of Tournamint.

To polish the logo, a gradient between minty green and blue fading into the circle, as well with a more distinct drop-shadow on the "MINT" is added. This logo is now ready to be used.

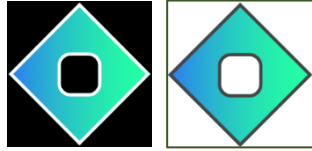


[20] Tournamint logo designs

### 5.4.3.2 Further Branding

The logo isn't the only part of branding that needs to be created for this application, however. Other variations of the logo for smaller elements are required, as well as a long form version that includes the application's full name. These assets are now easier to create thanks to the logo's final design.

Firstly, the web-specific, smaller logo design. As a web application is viewed on many different devices with varying display size and resolution, the logo must be simple yet easy to recognize. To this end, the logo on the left takes the green-blue gradient idea from the diamond in the main logo and applies it to the shape's background instead of border. The border then is a single colour, dependent on the background. This is due to the different themes that tournamint will implement: light and dark. Not only is this an accessibility feature that is prevalent in most web applications today, but it also gives the user more



[21] Mini Tournamint logos

choice. The harsh light mode can often give headaches and eye strain with prolonged exposure, while dark mode can often do the same for certain individuals with visual disabilities or those prone to migraines. Dark mode also affects people with astigmatism negatively, creating a sort of 'halo effect' where text becomes blurred and washy, leading a user to undergo further eye strain attempting to read the text<sup>[18]</sup>. Therefore, it's best to give users a choice in the matter, and therefore multiple web-ready logos are required.

Additionally, a longer form logo for external branding is useful. For Tournamint, two such logos are made, both in light and dark variants. The first logo mimics the original logo with only the word "MINT" with its' green drop-shadow.



[22] Tournamint logo variations

The second logo contains the entire word "Tournamint". This logo would be used for banners and other locations on the web where a longform logo containing the entire application's name would be useful. Due to the increased length, the green drop shadow is now also a green-blue gradient, mirroring the smaller logo.

With all this brand material created, the front-end application design can be completed and match the vision of what Tournamint should become with far better accuracy. Even though this project is intended as an open-source solution, branding is important to make the application distinctive and recognizable.

### 5.4.3.3 Colour Scheme

In order to achieve brand recognizability as well as achieve a pleasant experience for the user, a colour scheme should be used. For Tournamint, this palette is chosen from the colours present in the logo, that being various shades of green, blue and cyan, combined with a stand-out accent colour of 'pear', and a greyscale shade as the neutral tone.



[23] Tournamint colour scheme

```

{
  mytheme: {
    primary: "#00ff88",
    secondary: "#00a2ff",
    accent: "#d2eb45",
    neutral: "#3d4451",
    "base-100": "#ffffff",
  },
},
},

```

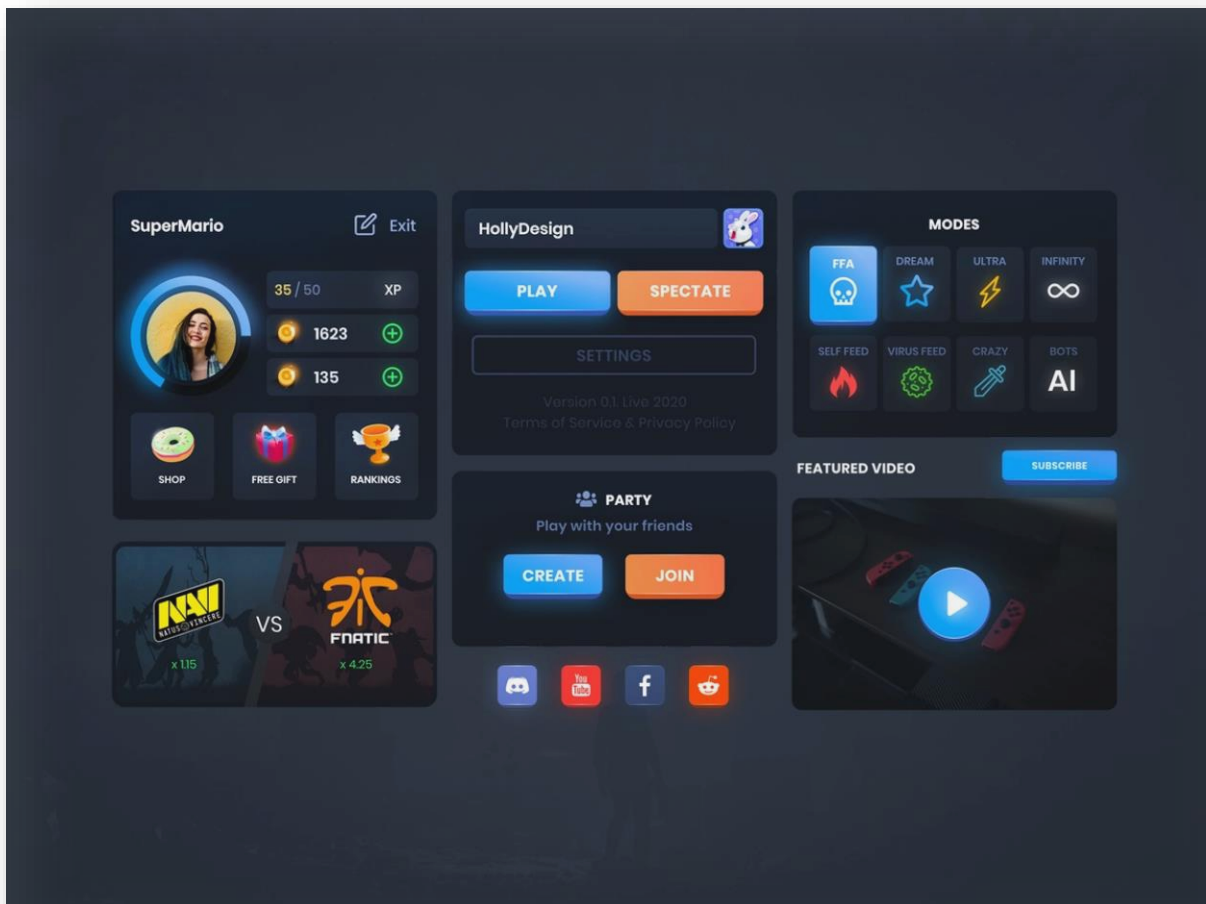
[24] Tailwind theme

When implementing this colour scheme in the application itself, TailwindCSS has a useful theming feature. After defining a colour scheme in `/tailwind.config.js`, it can be used with Tailwind's class-based styling. For example, to change the text colour of an element to the primary colour, the class `text-primary` should be added. For changing the background colour of an element to the secondary colour, `bg-secondary` is used. An additional benefit of this system is that for custom tournament distributions, if the colour scheme needs to be changed it only involves changing values in one location, instead of updating every element that has been styled with a colour.

#### 5.4.4 Profile Page

According to the activity flow diagram in section 5.2 concerning newly registered users, the profile page is the first thing they see after registering for a new account. This page is important as it shows the user the information that Tournamint has collected from their public League of Legends account, as well as their statistics within Tournamint, and their own little editable profile. This page should have a form for the user to change their display name, biography and favourite champion.

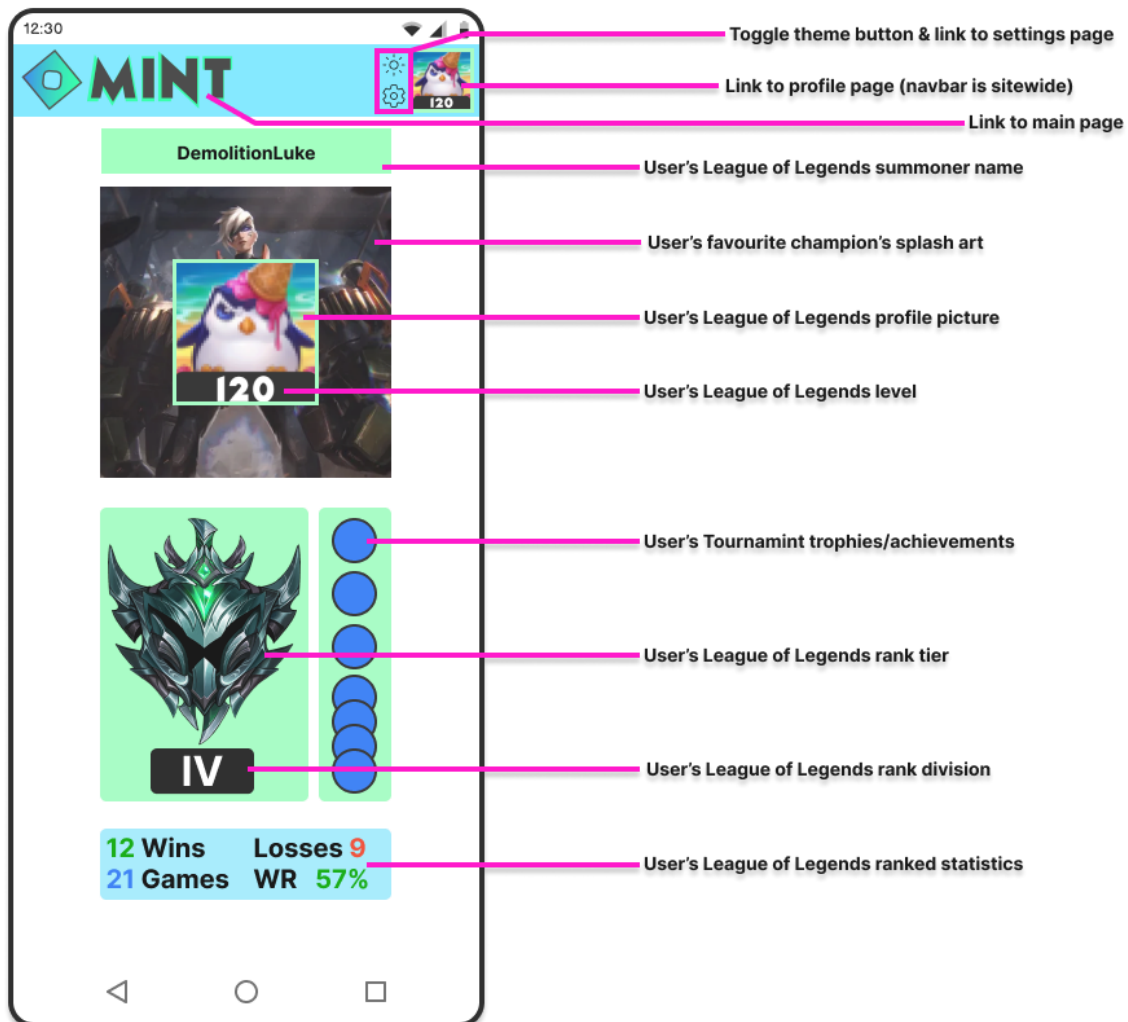
In similar design fashion to 5.3.1 *Choice of Platform*, there should be a mobile-accessible version of the application as well as desktop. Inspiration for the layout of the profile page is taken from the following image design created by Anton Olashyn<sup>[19]</sup>



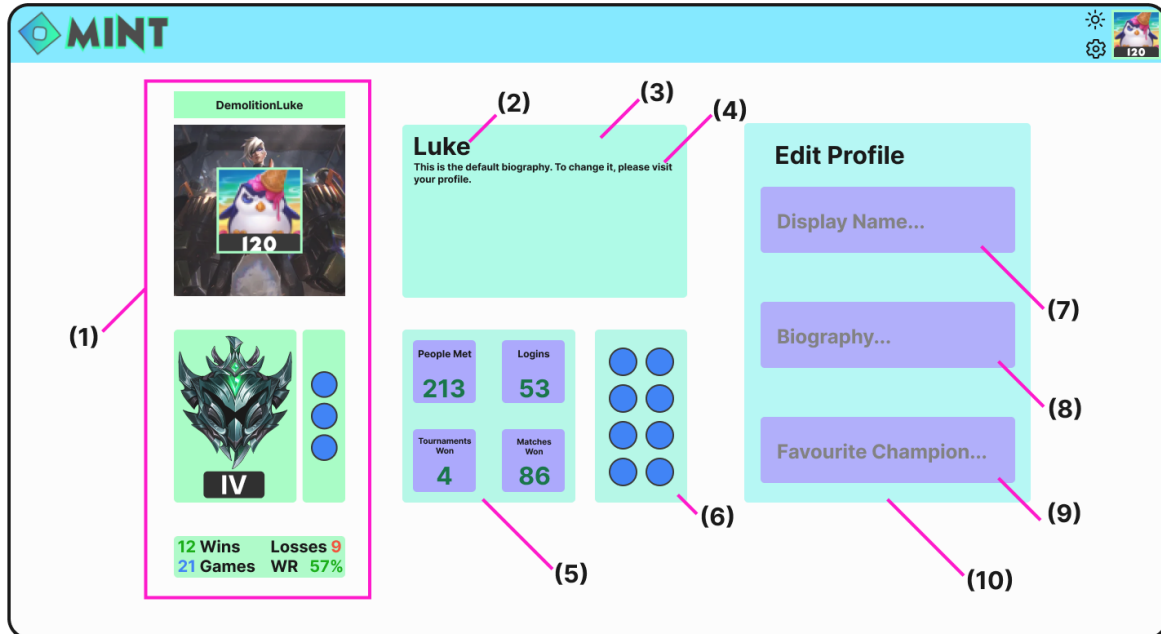
[25] Profile design by Anton Olashyn<sup>[19]</sup>

The design of this dashboard is perfect for Tournamint. Due to its card-grid system it is modular, and therefore can be manipulated to fit different layouts using media queries. It isn't oversaturated with information, and clearly segments logic by column, with the most important information in the left column, the user-interactable information in the centre, and other information in the right column.

Using this layout philosophy as inspiration, the following designs have been created for both mobile and desktop:



[26] Mobile design mock-up



[27] Desktop design mock-up

Due to the size limitations of this document, in order to achieve better readability for the desktop mock-up, the descriptions of each element have been numbered. Each number corresponds to the following list:

(1) Same section as seen in the mobile mock-up. The rest of this page is rendered in a grid using media queries that change based on screen size / aspect ratio.

(2) User's display name

(3) User profile information display

(4) User's biography

(5) User's Tournamint statistics

(6) User's trophy overflow

(7) User's display name input. Affects the profile information display.

(8) User's biography input. Affects the profile information display.

(9) User's favourite champion input. Affects their profile splash art.

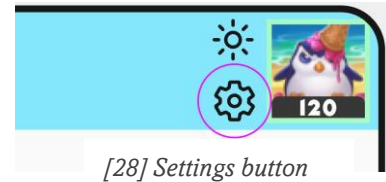
(10) User's favourite champion input. Affects their profile splash art.

There is clear evidence of mobile first design in these mock-ups. The single column list of elements in the mobile design can be seen clearly translated to the desktop version. The additional information in the grid system of the desktop design is not visible in the mobile mock up for a reason. Taking this design approach leaves the designer with freedom in what information to display to each platform. For example, it may be sensible to omit information that a user won't be using as much on mobile as on desktop. Although mobile performance has made great strides in the last decade, mobile data usage is still a source of worry for many people, and reducing the number of elements an application renders will cut down on this usage over time.

Contrarily however, should the data prove useful to being displayed on mobile also, then it can simply be appended below the initial elements in the single column. Scrolling on mobile is a very simple, intuitive gesture.

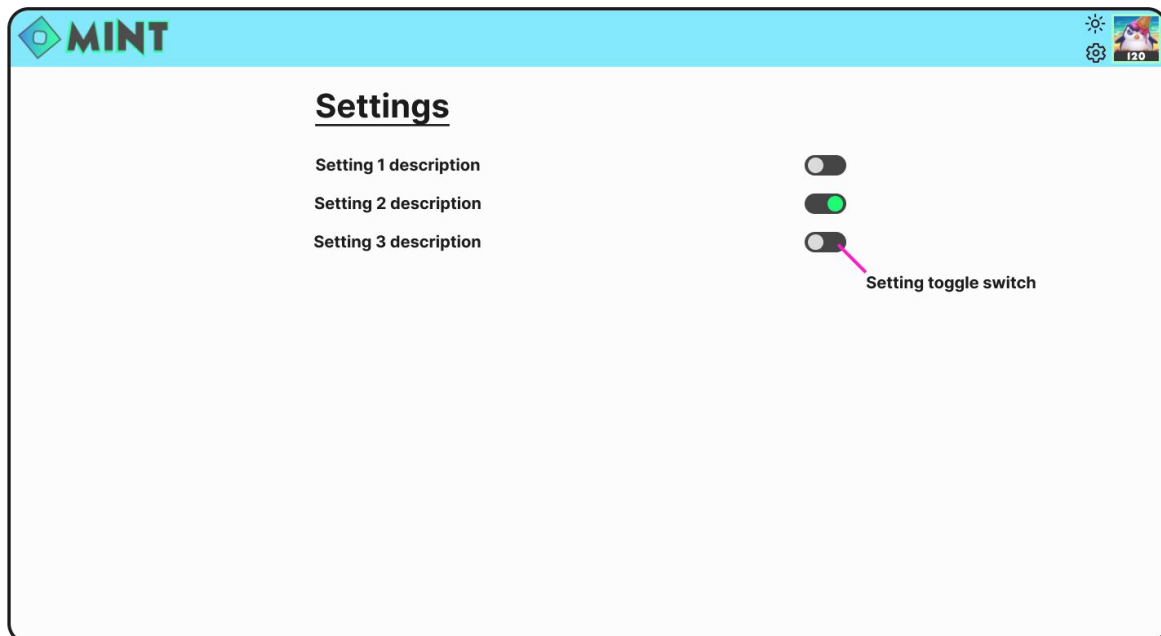
### 5.4.5 Settings Page

Once a user has finished editing their profile and viewing their information, they may wish to change some settings. Access to this page should be simple and available everywhere. For this reason, the button to navigate to the settings page is on the navigation bar. Using Next's layout feature, a component containing the navbar jsx code can be inserted at the same position on every page of the application without the developer adding it manually. This helps create a unified design language across the site, helping a user achieve familiarity with the application quicker and easier.



The settings page itself should be simple in design. The user needs to see a description of what each setting changes, as well as a toggle switch/slider that they can manipulate to change that setting's value and subsequent effect on the application. A design mock-up for the settings page is shown below.

The lack of mobile mock-ups from now on is due to their surplus nature. As explored in 5.4.4 Profile Page, they are a starting point for desktop development, meaning that showing the more complex desktop design mock-ups that contain all the information possible is the better choice.



[29] Settings page design mock-up

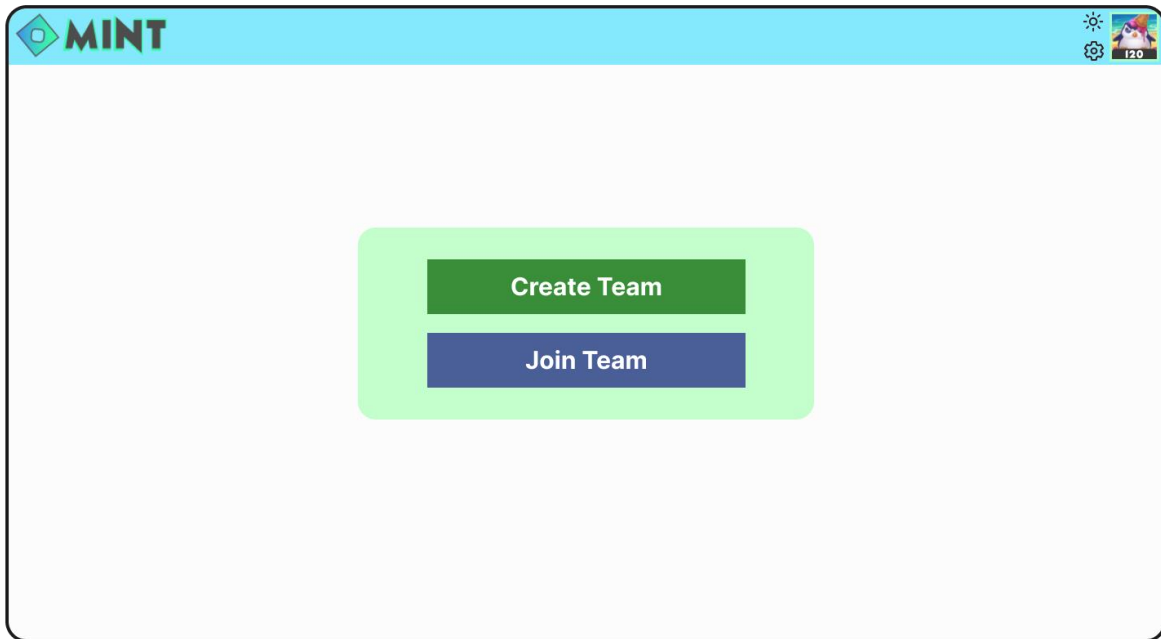
### 5.4.6 Team Page

Compared to the settings page, the team page and its' associated functionality is considerably more complex. Due to this, it's constituent parts have each been given their own section. All of team creation, joining and viewing is initially accessible via a single button press however, as team creation and team joining pages are handled through the use of modals; These components overlay an existing page, moving it out of focus, and them into focus. Typically, a modal is used for forms or extra information that doesn't warrant an entire new page to be rendered. This approach is perfect for team creation and joining, as these actions both affect the main team page directly.

#### 5.4.6.2 Main Page (Initial)

When initially visiting the teams page, a user should be presented with two options – Create Team and Join Team. This approach to simplicity in user interaction and not overburdening the user is borrowed from the design of 5.4.2 Landing Page, which implements a similar philosophy. A mock-up for this page is

shown below.



[30] Initial team page design mock-up

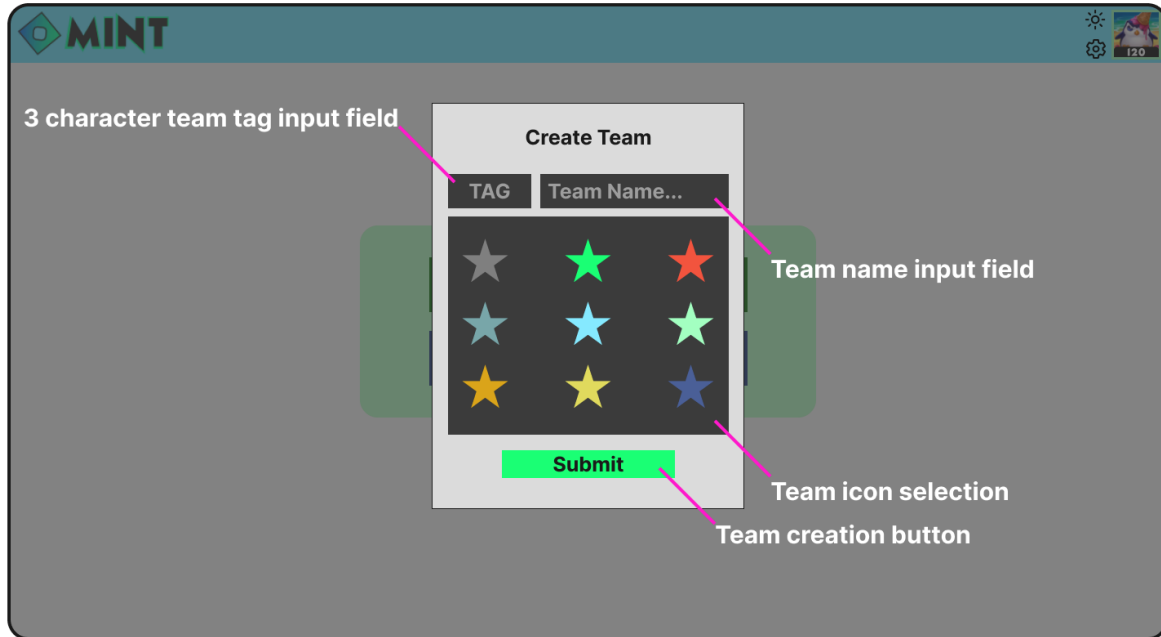
If the user clicks the Create Team button, a modal is shown overlaying the current page with a form for team creation (see 5.4.6.1 Team Creation).

If the user clicks the Join Team button, a similar modal is shown, but with a shorter form that a user fills out to join an existing team (see 5.4.6.2 Team Joining).

#### 5.4.6.1 Team Creation

Team creation is the process that a team leader goes through. This option is accessible if the user is not in a team. A team has a name, a 3-character tag (e.g., TEA, FPX), and an icon (represented in the mock-up as different coloured stars). These features let a team stand out and be easily recognizable and distinct from other teams in the tournament view on the main page of the application. This format for team information is inspired by League of Legend's Clash mode (2.1 Current Applications), so it may feel familiar to users who have participated in a Clash tournament before.





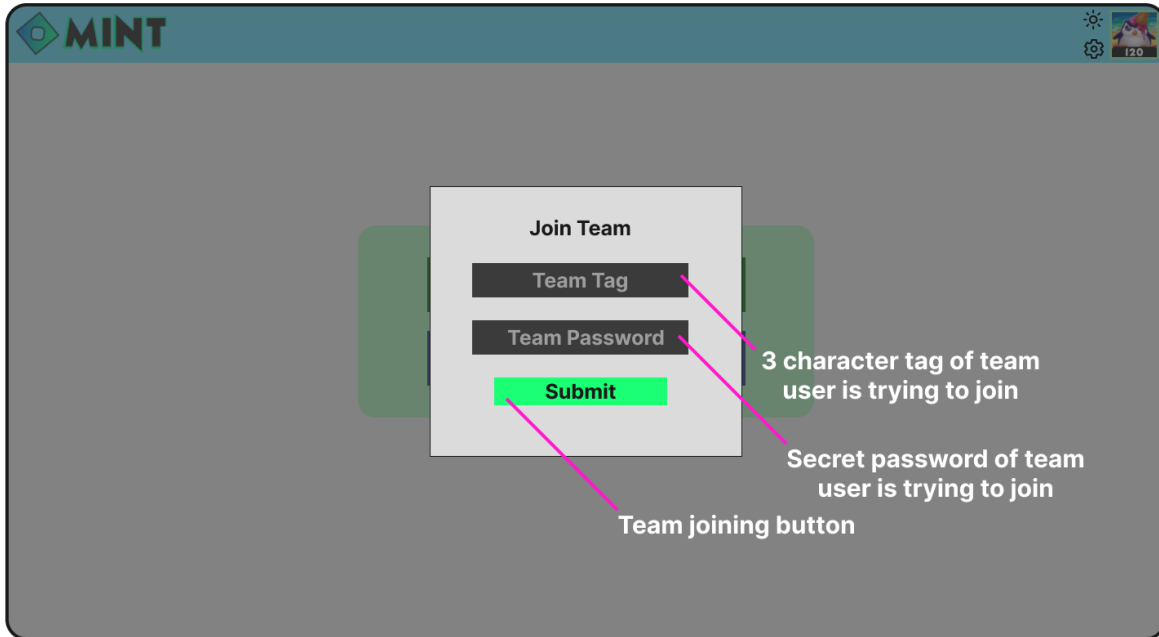
[31] Team creation design mock-up

Once the Submit button has been pressed, and the application has checked that the information entered is valid, the team is formed. The user's account data is updated with the new team in Local Storage and in the Upstash data store, and the team is added to the list of teams in the data store too. The modal closes, and the user sees the initial team page again, but now it's populated with their team information.

From now on, whenever this user visits the team page, even after logging out and in, their team information is displayed. For a more comprehensive examination of this page post-team creation please see 5.4.6.2 *Main Page (Team Information)*.

#### 5.4.6.1 Team Joining

Joining a team requires two pieces of information – the team's 3-character tag, defined by the team creator/leader, and the team's secret password, which is automatically generated, and visible to team members of said team. A mock-up of the form to accept these pieces of information is shown below. This view is seen when clicking the Join Team button on the initial team main page, and is opened in the form of a modal, in the same fashion as the team creation modal.

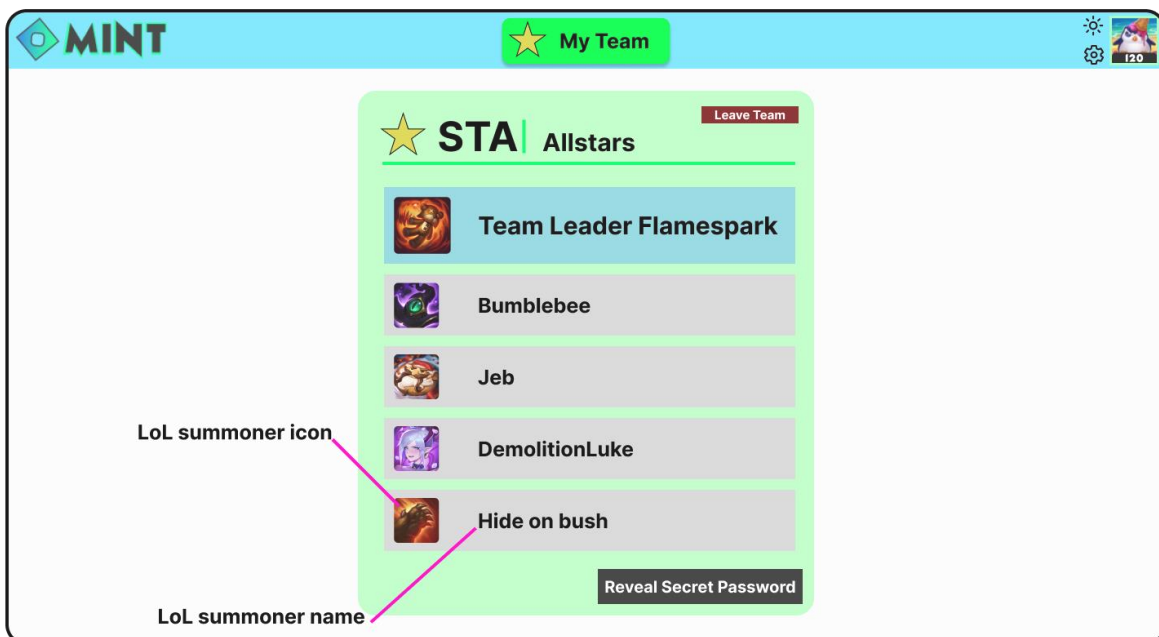


[32] Join team design mock-up

Once a user has entered the two pieces of required information, the program validates the inputs and if they are found valid, adds the user's account details to the team data in the Upstash data store. The user's account is also updated with the new team information in Local Storage and the data store. Similar to team creation, clicking the Submit button will also close the modal, showing the user the main page with their new team information (5.4.6.2 Main Page (Team Information))

#### 5.4.6.2 Main Page (Team Information)

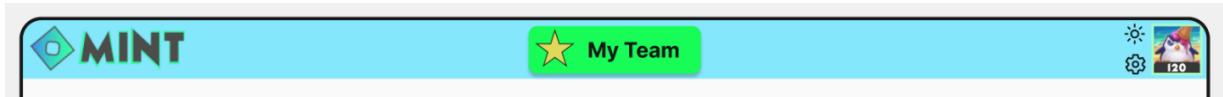
This version of the main team page is the one users will see most of the time. Once they have created or joined a team, the information of that team is displayed on this page, along with a button to leave the team, and a hidden field that displays the team's secret password, used for joining. This page lets team members see their teammates in the form of their LoL summoner name, icon and level. A mock-up of this design is shown below.



[33] Team information design mock-up

By hovering their mouse over “Reveal Secret Password”, a user will be shown the secret password used to join the team, and by clicking it, will be able to copy it to their clipboard automatically, so that they can easily share the code with their friends who wish to join their team.

One of the most important aspects of the team page has however not yet been discussed – how the user initially navigates to the page. So far for the profile page, a user is redirected automatically after registration, and subsequently can revisit the page by clicking their icon in the navbar. To access the settings page, a user clicks the small cogwheel, also in the navbar. It therefore makes sense for the team page button to also be shown in the navbar. As the user’s team is their central point around the entirety of tournament, their collective identity with their team mates even, this button would be best suited in a distinct location. No location on a navbar is more distinct than the centre. It is where a user’s eye is naturally drawn. The button will look something like the following in the navbar at the top of every page:



[34] Team button navbar design mock-up

If possible, having the user’s team’s icon (in this case a yellow star) shown on the button would be a nice cosmetic feature, but is not essential.

#### 5.4.7 Tournament Pages

Once a user has set up their profile, altered their settings to their liking, and either created or joined a team with their friends, it’s time participate or host a tournament. Tournaments come in two categories: Public and Private. Public tournaments are accessible to any team registered on that distribution of Tournamint, while private tournaments require an access code, and are not publicly listed.

As the process of creating, joining, seeding and participating in tournaments through different views and dynamic components is complex, this section will break down each distinct phase of a user’s journey to participating in a tournament into its’ own section.

##### 5.4.7.1 Main Page

Tournamint’s main page is instrumental in how the entire application flows. In *5.2 Activity Flow*, the diagram titled *Pre-Tournament User Activity Flow* depicts multiple functions of the main page that are carried out before a user has even joined a team or a tournament such as “Highlight team button” and “Highlight tournament creation/joining buttons” at different stages of use. To effectively pull off the large number of different ways the main page should be presented to the user, jsx conditional rendering can be used. This method of rendering specific components based on one or multiple conditions is especially useful for an application such as this. It allows code to be simplified and all written within one file, rather than spread out over multiple pages, and in the case of a Next application, routes (leading to slower loading times as well as limited data access and subsequent risk of prop tunnelling).

The following example of conditional rendering shows how it is possible to render elements conditionally with very little code. The first JavaScript line’s left hand side expression evaluates to true, therefore

```
<div>
  {true || false && <p>Hello World! </p>}
  {true && false && <p>Goodbye World...</p>}
</div>
```

[35] Conditional rendering example

rendering the `<p/>` element. The second line’s left hand side expression evaluates to false, therefore not rendering the `<p/>` element. Any valid JavaScript expression can be passed to this jsx code, and elements therefore conditionally

rendered. This approach will be used in the

implementation of Tournamint’s main page.

Once a user has created or joined a team, as per *5.2 Activity Flow*, the main page should guide them to create or join a



[36] Tournament buttons design mock-up

tournament. This would best be in the form of a box containing buttons that has an animation to catch the user's attention such as fading in and out or its' scale becoming slightly larger and smaller. A mock-up of this box is shown to the right:

'Create Tournament' takes the user to the page where they can fill out the form to create a tournament. 'Join Tournament' similarly takes the user to a page where they can fill out a form to join a private tournament. 'Find Tournament' also takes the user to a new page, but unlike the other pages this one will feature a grid of public tournaments in this Tournament distribution, along with small amounts of information about each, such as the tournament name, organiser's summoner name, tournament size and current number of teams signed up.

#### 5.4.7.2 Creating a Tournament

When a user clicks on the "Create Tournament" button, rendered conditionally only when they are in a team, they are redirected to the tournament creation page. This page features a form with multiple varied input types such as textual inputs, sliders, toggle switches and date/time pickers. A mock-up design of this page is shown below.

The mock-up shows a form for creating a tournament. The form includes the following elements:

- Tournament Name:** A text input field.
- Number of Teams:** A draggable slider input with values 4, 8, and 16.
- Start Date & Time:** A date/time picker input.
- Private/Public:** A toggle switch input.
- Tournament ID:** A constrained text input.
- Secret Passcode:** A constrained text input.
- Create Tournament:** A green button.

[37] Tournament creation page design mock-up

This page can, similar to the main page, make use of conditional rendering. A secret passcode is only required when a tournament's type is set to private. Public tournaments do not require a passcode. Therefore, toggling the toggle switch's state to public can trigger a variable change, which drives the conditional rendering of the Secret Passcode input element, causing it to fade out and in based on the variable's value.

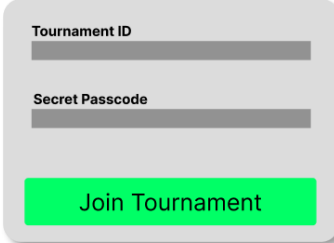
Once all required information for a new tournament has been provided and the "Create Tournament" button clicked, the application should update the Upstash data store with this new tournament, as well as the user's team information as participating in this tournament. The user should then be redirected to the main page, where they will be presented with a new view showing the teams that are currently signed up to the tournament. This will initially only be the user's team, but every time a new team signs up the conditionally rendered view updates to show the new team's information, including their icon and tag.

### 5.4.7.3 Joining a Tournament

If a team does not wish to host a tournament, but instead participate in an existing one, then there are two options: public tournaments and private tournaments. While the tournament's execution does not differ between the types, the method of joining each one does.

#### 5.4.7.3.1 Private Tournaments

Private tournaments follow the more familiar form-based style, requesting the user to enter the Tournament ID and Secret Passcode to join (shown to the right), whereas public tournaments are handled differently.

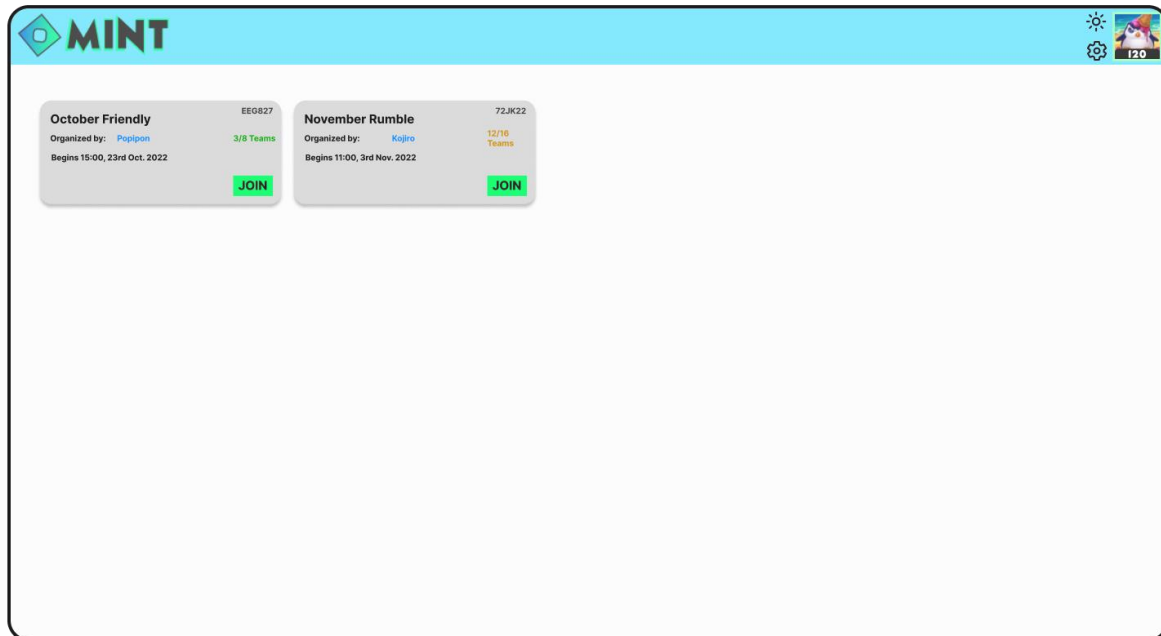


Tournament ID  
Secret Passcode  
Join Tournament

[38] Tournament joining design mock-up

#### 5.4.7.3.2 Public Tournaments

To join a public tournament, a user should click the “Find Tournament” button on the main page. This will redirect them to a new page that has a list of available tournaments, that is, tournaments whose type is public, that starts in the future and has not reached its team limit. A design mock-up of this page is provided below showing two available tournaments.

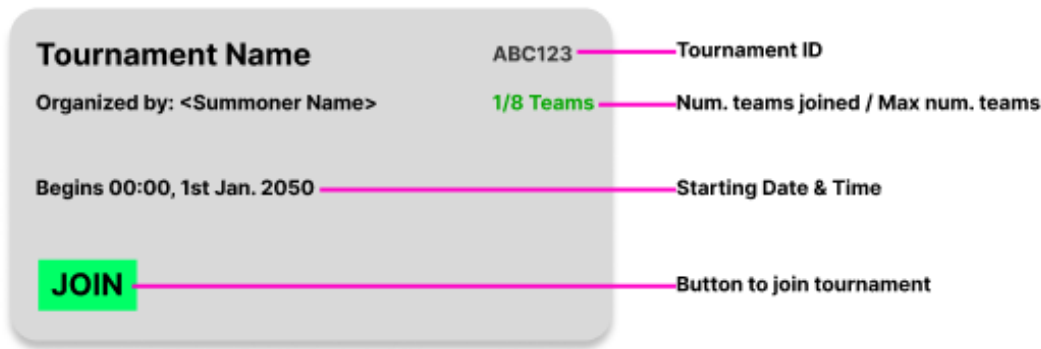


[39] Public tournament grid design mock-up

This page holds a grid containing a block for each valid public tournament, along with the following information:

- Tournament name
- Tournament organizer
- Tournament start date/time
- Tournament ID
- Tournament size & number of teams already signed up
- Join tournament button

A clearer look at one of the blocks is shown below:



[40] *Public tournament block design mock-up*

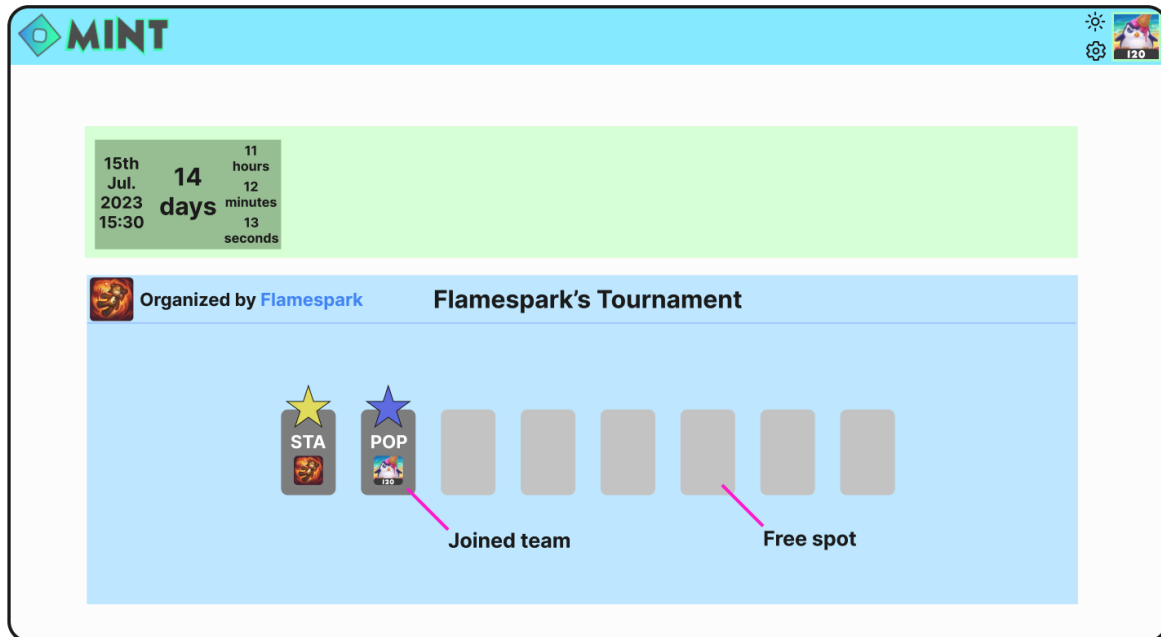
Once a user finds a tournament that their team wishes to participate in, they click the “Join” button on that block. In the same fashion as joining a private tournament, the application then updates the Upstash data store’s team hash map with the new tournament, and update’s the tournament’s list of teams that are signed up by adding the user’s team. This will naturally increment the number of teams that have joined the tournament by one, shown in the public tournament list.

#### 5.4.7.4 *Tournament Waiting Period*

Once a user has either created a tournament or signed up to either a private or public tournament, the waiting period begins. This phase is defined as the block of time after a user signs up to a tournament, but before the tournament begins minus one hour. This one hour before a tournament starts comes after the waiting period, and is named the seeding phase, as it is when tournament brackets are seeded. Seeding can happen anytime during this one-hour period, as it is the last chance for the tournament to fill up before beginning. If a tournament is not full by the time the seeding phase is over, the tournament should be automatically cancelled.

The waiting phase brings new conditionally rendered elements to the main page for users that are signed up to the tournament. The first element is a countdown timer, denoting the start of the tournament. As discussed in 5.3.1 *Choice of Platform*, the countdown timer is an element that should be easily accessible to both desktop and mobile users of the application. Mobile users that log into their Tournamint account will be greeted by the main page first, and if they are in a tournament that is currently in the waiting phase, the countdown timer will clearly show when the tournament is slated to begin.

The second new component shown to the user is the list of teams that have signed up to the tournament, and the amount of space left. With these new components, the main page should look similar to the following mock-up:



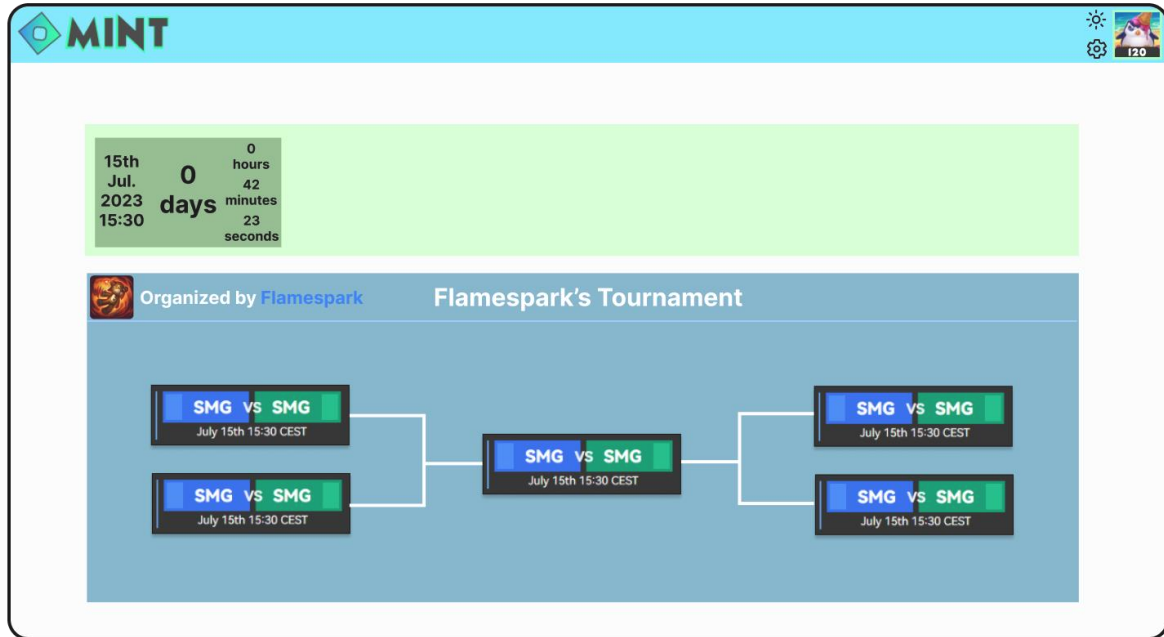
[41] Tournament waiting period design mock-up

#### 5.4.7.5 Tournament Seeding Phase

An hour before the tournament begins, the waiting period comes to a close and the seeding phase begins. During this phase, once the tournament is full and the tournament creator logs in, the tournament brackets are seeded with their initial data. This process randomly assigns teams participating in the tournament into matches for round one.

An addition to this process, which is not in the scope of this project, but would be a good future goal to work towards if Tournamint is further developed past launch, is weighted seeding. Although multiple different algorithms exist for this process, the core idea is the same: Teams that perform better overall are weighted higher than those that do not. Highly weighted teams are more likely to be matched against other high performing teams, while lower performing teams are more likely to be matched with other low performing teams. This creates a tournament landscape that is as fair as possible.

Once a tournament is seeded, all members of teams participating in the tournament will see a conditionally rendered display of all the tournament brackets resembling the following mock-up (match display information is placeholder):

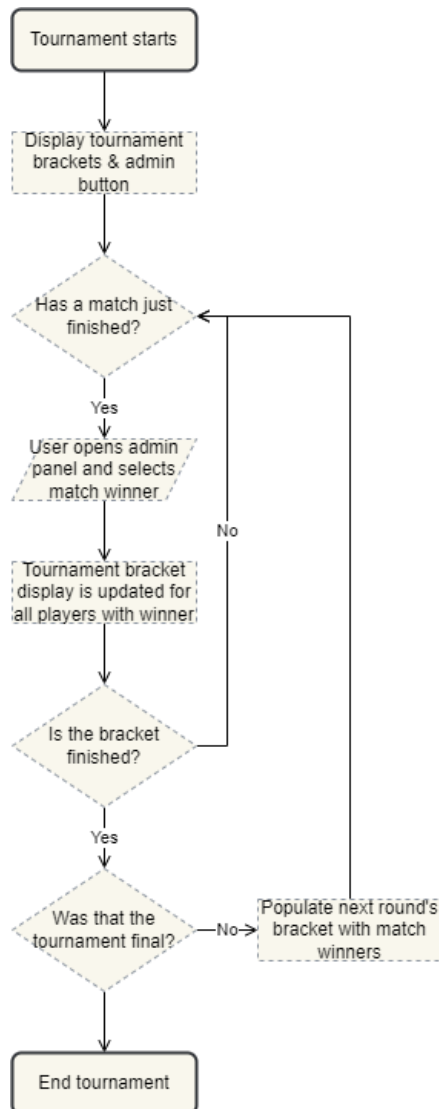


[42] Tournament seeding design mock-up



#### 5.4.7.5 In-Progress Tournament

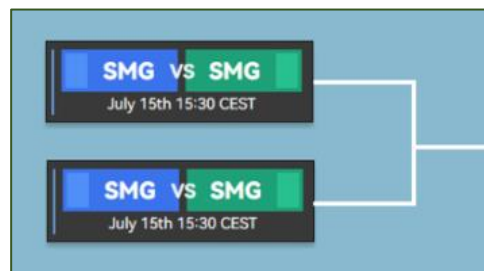
Once the one hour seeding phase of the tournament is finished, the tournament's first round begins. In 5.2 Activity Flow, an activity flow diagram depicting the steps that are carried out by the tournament creator during an in-progress tournament. For convenience, this diagram is shown again.



[43] Tournament activity flow diagram

The first step the tournament creator should take is to open their administrator panel. This is done by clicking the administration button, which is conditionally rendered to only the creator of the tournament. This button opens a modal that contains a list of every currently ongoing match. By clicking on a team within a currently ongoing match, the tournament creator signifies that this is the team that won the match. Graphically, this will update for all tournament participants as the winning team of the match being highlighted, and the losing team greyed out. This way participants can keep track of the status of the tournament, and who they may end up fighting.

Once a winner of a match is declared, the Upstash data store's tournament hash map is updated with the match winning team, and the application then checks whether the bracket is finished. A bracket is defined as a set of two matches. A bracket has 4 total teams participating, and two winners, who go on to fight in the subsequent round:



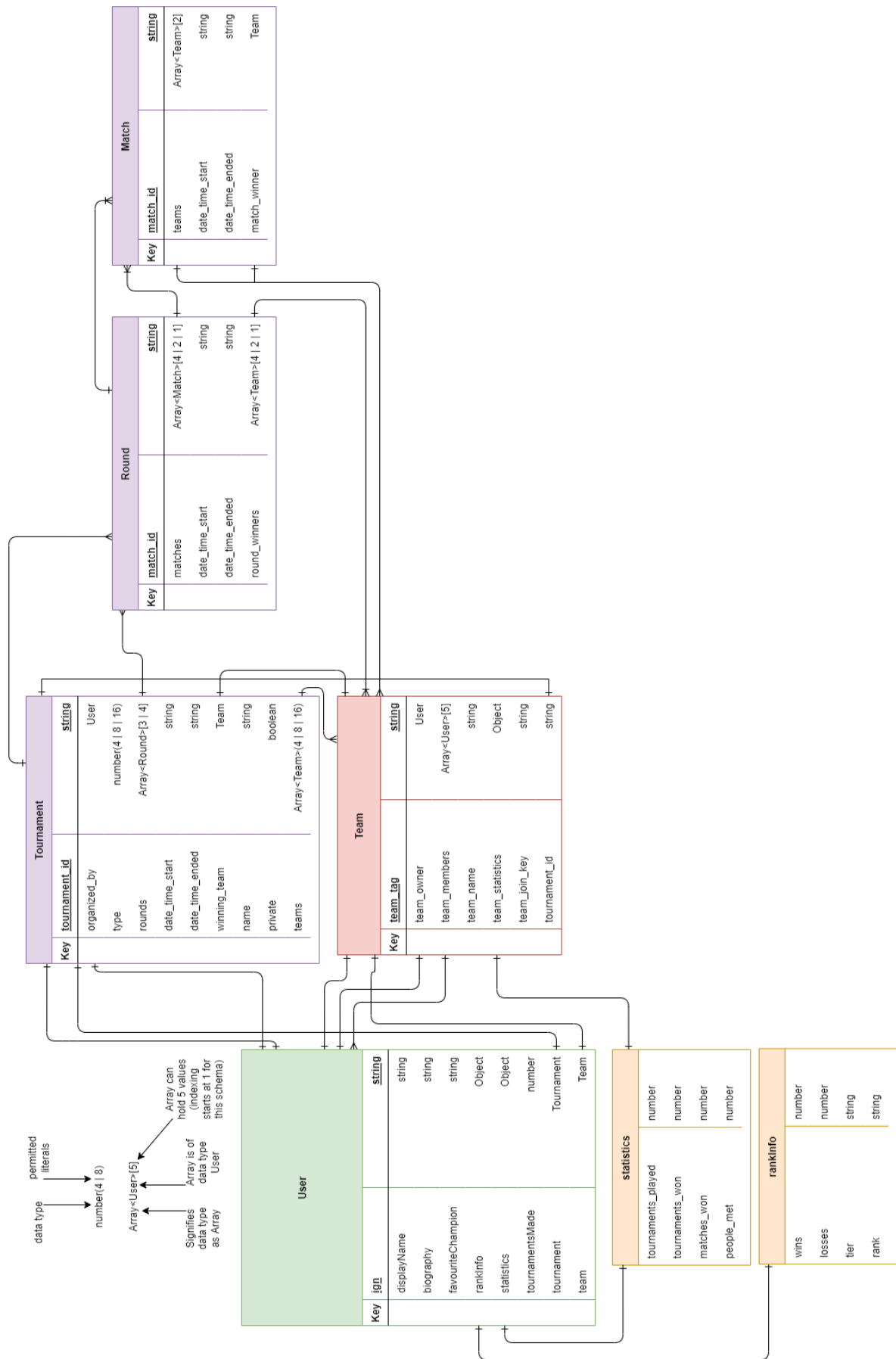
[44] Tournament bracket design mock-up

This process is repeated by the tournament creator until all brackets have finished, and only the final match remains. Once the winner of the final has been declared by the tournament creator the tournament ends, and the winner is recorded in the data store. All participants are then free to resume their normal use of Tournamint, creating, joining and leaving teams, and creating and joining public and private tournaments.

## 5.6 DATA STORAGE DESIGN

Facilitating the kinds of algorithms that will serve this application's function requires a robust design of the underlying data. As this application will use Upstash' Redis-based data storage, an object-oriented quasi-database design would be best, as a hash map can be translated to and from JSON, which is used in the front-end of the application.

To represent the objects and their relationships, a UML Class diagram can be used. Due to the size of the diagram it has been placed at 90° of its' original rotation:



[45] Tournament class diagram

This Class diagram contains multiple classes and their associated fields and relationships to other classes. For example, a single User has a 1-1 relationship with a Team. One Tournament can have multiple teams, but only one creator (User). A Tournament has multiple Rounds, which in turn has multiple Matches, but one Match is only a part of one Round, and a Round is only ever a part of one Tournament. Visualizing these relationships, as well as the classes' fields and their associated data types is instrumental in designing the backend schema and how it relates to the data being shown and retrieved to and from the front-end.

The different colours (Green, Purple, Red) denote different top-level classes. For example, although Rounds and Matches are distinct classes with their own data, they are only ever found within a Tournament class. This means that these top-level classes will each have their own hash map within Upstash's data store.

With the data store representation of Tournament's different data classes complete, what is now left is a way to interact with this data. For this the application must use different API calls.

## 5.7 API DESIGN

To access the data in the Upstash data store to perform CRUD operations on, as well as access user data on Riot Games' servers, API calls must be performed. As discussed in sections 2.4.3 & 2.5, this project would benefit from logic segmentation between different API calling technologies, namely Next's inbuilt API functionality, and the third-party library Axios.

### 5.6.1 Next API

Next's API system is clever in its design. All API calls are designed in a separate file structure to jsx pages, under /pages/api. The file structure for an API call that queries the Upstash data store for login details for example may look like the following image:

A screenshot of a file explorer showing a breadcrumb path: client &gt; pages &gt; api &gt; TS login.ts &gt; ...

The benefits of this system are clear – naming API call files after the action that the API performs, and segmenting the code into separate files makes maintaining the codebase, and initially understanding it far easier than searching for API call designs within a jsx page.

[46] Example API directory route

For Tournament, these API call files will have a structure resembling the following:

A screenshot of a code editor showing a TypeScript API handler function. The code is as follows:

```
export default async function handler(  
  // handler method parameters  
  req: NextApiRequest,  
  res: NextApiResponse<any>  
) {  
  // handler method body  
  if (req.method === 'POST') {}  
  if (req.method === 'PATCH') {}  
  if (req.method === 'GET') {}  
}
```

[47] API usage example

Each API file exports a default handler that takes a user's request. Due to the flexible modularity of this Typescript-based system, a developer can add or remove any HTTP method they wish in these API files, and handle the responses for incorrect inputs too. To make a call to an API file, one needs to call it using either a 3<sup>rd</sup> party solution such as Axios, or more commonly use JavaScript's in-built fetch() method. By calling this method with a parameter containing the router to the API file as well as the method

parameters for the API handler to interpret, Next can make the API call with very little code written by the developer. An example for the `/api/login` route is shown below, with a string passes for the handler's method parameter:

```
await fetch('/api/login?' + new URLSearchParams({ ign: name_input })))
```

[48] *JavaScript fetch() function example*

This code, when combined with JSON conversion and interpretation algorithms, will send a request to the login API file, and ask it to check for the existence of the `name_input` value in the Upstash data store. From then the application can use the response to determine whether an account with that name exists, and can retrieve data from said account if it does.

### 5.6.2 Axios

In order to segment logic between Tournamint's two external data source/sinks, the third-party library Axios will be used for retrieving data from Riot Games' League of Legends API. Axios functions in a similar manner to Next's API solution, but has different syntax and additional methods, one of which will be very useful for a section of this program's development.

This Axios function, aptly named `.all()`, permits a developer to make multiple API calls in sequence, with the data being collected by the function for further manipulation or display. This function is particularly useful when aggregating similar data each requested with a different set of method body or header parameters.

One use case of this particular function is calling the League of Legends API for each member of a team to display their data. In this case the same API call is being performed, but with a different summoner name for each call as a parameter. In the following example, Axios is used to first execute multiple API calls of the API route `/api/team_data`, and then manipulate the data it receives to extract only the useful parts into the `tempTeamMembersData` variable. This variable can then be used by the program to display each team member's information.

```
// Fire multiple API calls in sequence
axios
  .all([
    axios.get('/api/team_data', {
      params: { ign: team_members[0] },
    }),
    axios.get('/api/team_data', {
      params: { ign: team_members[1] },
    }),
    axios.get('/api/team_data', {
      params: { ign: team_members[2] },
    }),
    axios.get('/api/team_data', {
      params: { ign: team_members[3] },
    }),
    axios.get('/api/team_data', {
      params: { ign: team_members[4] },
    }),
  ])
  // Extract useful data
  .then(
    axios.spread((m1, m2, m3, m4, m5) => {
      tempTeamMembersData = [
        {
          ign: team_members[0],
          level: m1.data.summoner_level,
          icon_id: m1.data.icon_id,
        },
        {
          ign: team_members[1],
          level: m2.data.summoner_level,
          icon_id: m2.data.icon_id,
        },
        {
          ign: team_members[2],
          level: m3.data.summoner_level,
          icon_id: m3.data.icon_id,
        },
        {
          ign: team_members[3],
          level: m4.data.summoner_level,
          icon_id: m4.data.icon_id,
        },
        {
          ign: team_members[4],
          level: m5.data.summoner_level,
          icon_id: m5.data.icon_id,
        },
      ]
    })
  )
)
```

[49] Axios multiple API call example

This methodology is not required for the Upstash data store due to its' inherent top-down structured design meaning that collections of data can be retrieved by executing an API call to access the parent

node of the data rather than multiple individual children. This is another reason to segment the API call logic with Axios being used for the League of Legends API (which has a rigid unchangeable structure due to it being a third-party data source) and Next's inbuilt API functionality for the Upstash data store.

## 5.8 PROGRAMMING ARCHITECTURE

Programming languages vary considerably in how they are written, what best practices one should follow and how logic and algorithms are structured. However, many programming languages don't have a 'best way' in which they should be used. A prime example of this is JavaScript, the main language that this project used (Wrapped by TypeScript). The Mozilla Foundation describes JavaScript as "a prototype-based, multi-paradigm scripting language that is dynamic, and supports object-oriented, imperative, and functional programming styles"<sup>[20]</sup>. With so much choice in how to use the language, it can be easy to become overwhelmed with options.

### 5.7.1 Object Oriented vs. Functional

There is thankfully usually a 'best' paradigm to use for each use case of a language, however. For example, this project uses Next.js, which is built on React.js. React has a history spanning longer than a decade<sup>[21]</sup>, and has undergone many iterations during this time. Initially, object-oriented design was seen as the gold standard of how to structure applications with React, and its' guidelines and architecture reflected this. React made use of encapsulation, inheritance and polymorphism with its' classes and interfaces. However, recently object-oriented design has started to be replaced in popularity with a more functional approach.

Where previously, React would manage state within a class's inherent state variable, since 2018<sup>[22]</sup> with the release of React 16.8, a new feature titled 'Hooks' has been added. Hooks work seamlessly with both class based React applications and applications that are purely functional in nature. They give developers an import-based opt-in set of features that work in conjunction with routes and nested components to make an application more *reactive*. For this project, a few hooks stand out as being particularly useful:

- `useEffect`: This hook, when given no callback parameter, will execute its' method body once upon page load, ideal for modifying & setting initial dynamic data values. When given a callback parameter in the form of a variable, the hook will execute each time said variable is updated. This way certain code can be executed when a dynamic event occurs in the application.
- `useState`: This hook acts as an easier-to-use, less code intensive version of React's previous class-based state. To create an application that is truly dynamic, i.e., responds to user input without URL changes by executing JavaScript within the same page, this `useState` hook can be used.
- `useRouter`: This hook, exclusive to Next.js, gives a developer direct access to Next's router. Usually, routes are handles automatically by Next, but by using functions provided by the `useRouter` hook, this functionality can be replicated in custom JavaScript. This can be useful when needing to modify the user's page route when they click a button or enter data for example.
- Custom routes: Although React and Next provide many different routes to a developer, there is still much functionality that doesn't exist due to it not being generic enough. For this project, an overarching 'useUser' may be very helpful to keep track of a user's state throughout their use of the application. For example, if they change their user information on their profile, and then open the team page to view their team, the data must be updated for display. This can be done one of two ways: Retrieve the data from the Upstash data store, or instead simply retrieve the data from the custom `useUser` hook. Not only is the second option faster as it doesn't require an API call, it also cuts down on the number of API calls performed, which is important as Upstash only permits 10,000 free API calls/day

So then, with React's hooks system and the flexibility of functional programming for the web, it seems like the clear choice for this project. Furthermore, TypeScript is actually able to mimic OOP concepts such as interfaces, and statically type variables. This way the logic of Tournamint can be written with functional programming concepts and React hooks, while the class-based object structure of the user account/team/tournament data can be represented and manipulated using a combination of TypeScript types, interfaces and JSON structure.

## 6 APPLICATION IMPLEMENTATION

In section 5, a good foundation for the design of Tournamint was presented. This section will focus on the implementation of those design choices to create a functional web application. Tournamint's source code is available on GitHub at the following link: <https://github.com/Khazoda/tournamint> and is also provided separately to this report as a compressed file. As this section will not cover every implementation decision and part of the code, instead opting to explain functionally significant sections, exploring the source code for specific details is a good idea.

### 6.1 APPLICATION IMPLEMENTATION CHOICES

Before documenting the implementation of the application, this section will revisit the choices of tools, solutions and technologies that were made in earlier sections of this report, as these components will be essential in the development of the application.

#### 6.1.1 Platform

In section 5.3.1 *Choice of Platform*, it was reasoned that the web application should be developed to both support desktop and mobile environments, by being made compatible with their respective screen sizes, aspect ratios and input methods.

To implement this, The web application will be developed mobile first with an emphasis on displaying content and making user input easy to understand. Then, using media queries and potentially JavaScript alterations, the content of the page will be scaled and rearranged to work well for desktop environments. Some content of the application should be desktop-only, achieved through the use of JSX conditional rendering.

#### 6.1.2 Framework & Language

In order to develop Tournamint for the web, a framework that enables the rapid development of its' features is needed. In 2.3 *Web Frameworks*, two possible frameworks were examined, and a conclusion was drawn to use Next.js, the framework created by Vercel, which is built on the React framework.

Next is a modern, heavy-duty yet fast SPA-enabled web framework that includes API calling out of the box, useful for Tournamint's communication with the Upstash data store,

Being built on React, Next offers everything that React does, including TypeScript support to create a quasi-statically typed JavaScript application and Hooks, a powerful feature that enables developers to create dynamic, reactive data and algorithms.

#### 6.1.3 Web Solutions

Functional out of the box with an application built with Next, Vercel is a free cloud-hosting solution that supports serverless functions and data. Explored in section 2.5 *Web Solutions*, Vercel can integrate neatly with Upstash, providing the least hassle and most user-friendly experience to an individual or group who wish to deploy their own Tournamint application.

Vercel allows users to deploy straight from a GitHub repository. As Tournamint will be hosted using GitHub, this makes it ideal for further streamlining the deployment process.

Upstash with Redis will be used as the back-end data storage solution. User accounts, teams and tournament data will all be stored here persistently, so that all users of the web application can access the same data.

Like Vercel, Upstash offers generous free solutions, and the deployment process is also relatively simple.

Finally, the Riot Games API for League of Legends will provide the data from their servers for League of Legends players, allowing Tournamint to retrieve a user's public information including their summoner name, level and ranked history.

#### 6.1.4 Web Technologies

Web technologies, listed in section 2.4 *Web Technologies*, encompass third party libraries that will enable the application to be developed easier and to a better standard.



Typescript is an essential requirement for the implementation of this application. It allows static typing in JavaScript, a dynamically typed language, provides the functionality to create interfaces, which will be instrumental when dealing with the data stored on Upstash, and much more. For creating the visual aspects of Tournamint, SASS, TailwindCSS, DaisyUI & React-Icons will be used. These libraries' functionality ranges from overhauling how a developer writes their CSS styles to component libraries containing essentially prefabricated HTML/CSS components, and unified SVG icons.

To access the League of Legends API, Axios will be used (5.6.2 *Axios*), and for date and time representation and manipulation, this application will use Moment.

### 6.1.5 Version Control

Although many version control technologies exist, many proprietary, the most popular technology is Git. There are many different providers that build on top of Git to create cloud-based version control systems, such as GitLab, BitBucket and GitHub. Although all solutions have their merits and demerits, this project will use GitHub. GitHub is perfect for independent developers not part of a larger team or company, which this project fits perfectly. It also has good ease-of-use and popularity, meaning that the open-source community may be more active on this platform. Due to this ease-of-use, GitHub is also a better solution for individuals or groups intending to deploy their own Tournamint distribution.

GitHub offers great customizability to a repository's main page, license, releases and much more. It even provides 'Insights' into repository activity.



[50] GitHub commit graph

With future potential contributions to the project from the open-source community, GitHub also provides visualisations of network graphs and a list of forks.

### 6.1.6 Development Environment

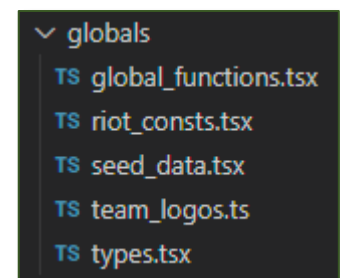
In order to develop the application using the myriad of different technologies and tools, a versatile integrated development environment (IDE) is needed.

For web development, Visual Studio Code or JetBrains WebStorm are often recommended. Due to this project developer's previous experience with VSCode, that will be the IDE used. It offers extensions for every language, as well as useful refactoring, syntax highlighting, linting and debugging, as well as first class Next.js support through Vercel's own 1<sup>st</sup> party extension.

## 6.2 GLOBAL DATA

Tournamint has many data types, interfaces and enumerations, as well as static data such as team logos and URIs pointing to static external data. As most of this global data is static, having a separate directory is advisable. Therefore, a top-level /global directory will be created with the following files:

Filename	
global_functions.tsx	Contains exported commonly used functions such as a string capitalizer, custom ID creator and tailored time converter.
riot_consts.tsx	List of constants containing both the distribution deployer's API key as well as the current data dragon URI (see 6.2.1 <i>Data Dragon</i> )
seed_data.tsx	Some constants containing data to initially seed into dynamic data positions.
team_logos.tsx	Contains a single JS constant export containing logo indices, their URI path and JS import relation.
types.tsx	A list of globally used TypeScript data structure interfaces, such as IUser, ITeam and ITournament (see 6.2.2 <i>TypeScript Interfaces</i> ).



[51] Global data pages



### 6.2.1 Data Dragon & LoL API

Data Dragon is the name given by Riot Games to their CDN containing League of Legends assets. Data present in the data dragon ranges from summoner icons to ranked crests and champion splash art. Essentially, if there is an image that a developer is trying to retrieve from the League of Legends API, they will need to use Data Dragon.

The way this works in practice, is that an API call to request certain data is sent to the League of Legends API. In the case of retrieving a user's summoner icon, the application must first construct an API call to the following URL with the user's summoner name (as of September 2022):

```
const PUUID = await getPlayerUUID(ign)
const API_CALL = encodeURI("https://euw1.api.riotgames.com/" + "lol/summoner/v4/summoners/by-puuid/" + PUUID + "?api_key=" + API_KEY)
```

*[52] PUUID calculation API call*

The constant "PUUID" (highlighted in green) has a value calculated by the following function:

```
function getPlayerUUID(ign: string | string[]) {
  return axios.get(encodeURI("https://euw1.api.riotgames.com/" + "lol/summoner/v4/summoners/by-name/" + ign + "?api_key=" + API_KEY))
    .then(response => {
      return response.data.puuid
    }).catch(err => err)
}
```

*[53] Player data GET API call*

Essentially, the application is making two API calls to Riot's servers. The first obtains a user's PUUID from their summoner name, and the second call obtains their public user information from their PUUID.

This two-stage process is necessary due to how the API functions, as it's not possible to obtain the required amount of user data using their summoner name directly, the player's UUID must be received first. This is likely due to summoner names not being global, but rather region specific, while PUUIDs are globally unique.

Once a user's data has been obtained, it needs to be trimmed so that only the useful data is extracted. Useful data is data that will be used by this application. This can include summoner name, summoner level, summoner icon and ranked information (although obtaining ranked information is a separate API call route).

Referring back to the data dragon, its' use is once a user's information has been received through the API call. Rather than transferring, for example, a summoner icon directly from an HTTP GET request, Riot's servers return an identifier for the icon instead. This code can then be automatically matched to a URI that is served by the Data Dragon.

The following code is an example of this, picking up from obtaining a user's information from their PUUID. Once the user's data has been retrieved, it is stored in a variable accessible by the custom `userData` hook (6.1.2 *Framework & Language*, 5.7.1 *Object Oriented vs. Functional*). This `userData` hook can be accessed globally throughout the application, even directly inside an HTML element (in JSX code):

```

<Image
  src={
    userData.profileIconId === undefined
      ? '/images/spinner.svg'
      : DD_PREFIX + 'img/profileicon/' + userData.profileIconId + '.png'
  }
  ...
>
</Image>

```

[54] Conditional Image tag props

The important parts to note here are the `DD_PREFIX` constant, and the `userData.profileIconId`. By combining the ID with the Data Dragon prefix to create a URI that points to the Data Dragon CDN, the application can access the summoner icon directly for display. The `DD_PREFIX` constant (6.2 Global Data) is located in the global folder:

```
export const DD_PREFIX = 'http://ddragon.leagueoflegends.com/cdn/12.16.1/'
```

[55] Data Dragon URL

Another technique used in displaying this profile image is the use of conditional rendering. This incredibly useful tool permits shorthand code that returns a different result based on a condition. In this case, if `userData.profileIconId` is undefined, the default spinner vector graphic is displayed instead of the summoner icon. This is best practice, and shows a user that their icon is still loading, whereas normally nothing would be shown initially. As CDN requests can sometimes take a few seconds, this is quite important in the wider scope of the application, where multiple different requests may be taking place within a short amount of time.

### 6.2.2 TypeScript Interfaces

To represent the data seen in the class diagram in section 5.6, TypeScript interfaces work wonderfully. To represent the team class for example, the code would be as follows:

```

export interface ITeam {
  team_icon_path: number
  team_tag: string
  team_colour_hex: string
  team_owner: string
  team_members: string[]
  team_name: string
  team_statistics: IStatistics
  team_join_key: string
  tournament_id: string
}

```

[56] ITeam interface

Each field of the `ITeam` interface has a statically defined data type, which can also be another interface (e.g., `team_statistics` using `IStatistics`). VSCode's TypeScript compiler will flag any type-related issues involving these fields in uses of the `ITeam` interface.

This aids the developer greatly in identifying bugs before the compilation stage inside the editor itself, rather than running into runtime errors.

Therefore, all the classes from section 5.6's class diagram are represented along with their parent-child relations through interface inheritance in the `types.tsx` file in the `/global` directory.

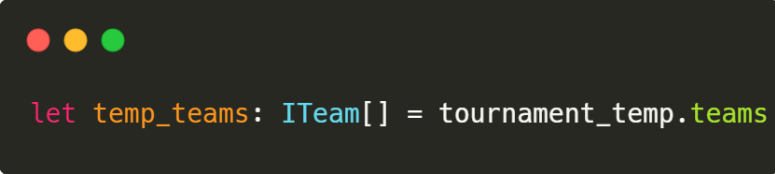
To use an interface, it simply needs to be imported like any other

JavaScript import like so:

```
import { ITeam } from '../..../globals/types'
```

[57] ITeam interface JS import

Then the interface can be used within that file, and the TypeScript compiler will automatically adapt its error checking to include it. This interface can be subjected to a change in morphology such as becoming an Array of the interface type, and the TypeScript compiler will adapt as such:



```
let temp_teams: ITeam[] = tournament_temp.teams
```

[58] *ITeam array example*

## 6.3 FRONT-END IMPLEMENTATION

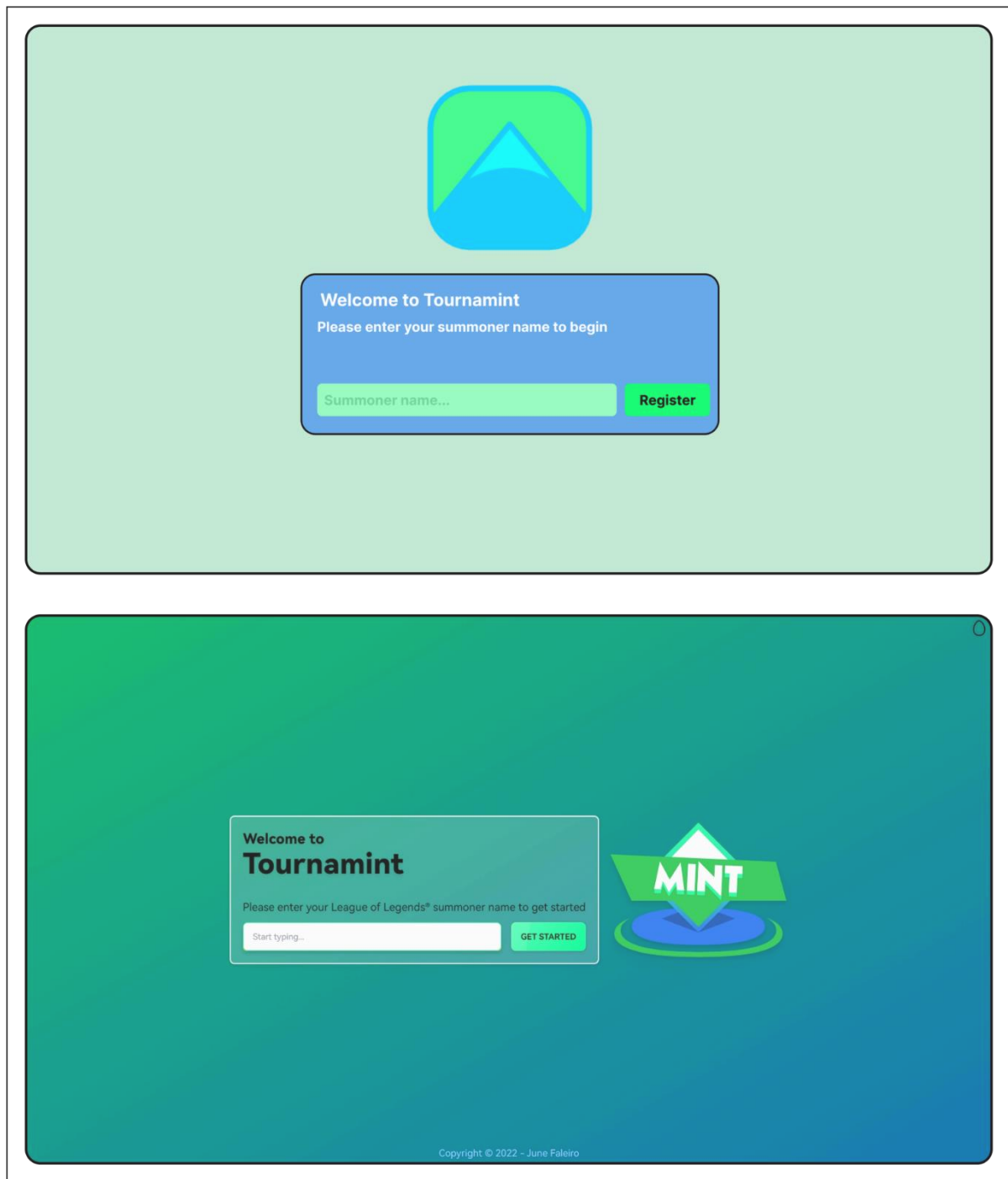
Since the global functionality and typings, communication with the Riot Games LoL API and the implementation of the Upstash data store have been completed, serving as the bulk of the ‘back-end’ code functionality, the focus can now be turned to the implementation of the Tournamint front-end.

Each subsection is of a page/page grouping in the Tournamint application. While following the same structure as *5.4 Front-End Application Design*, this section will compare each implemented page to its’ design counterpart, and discuss challenges that arose during development. Particular functionality will also be explained. Page element identifiers/explanations have been removed from the images, but the rest of the design is identical to those shown in the design section.

Additionally, screenshots of the final implementation have been edited with a bevel and border to better resemble the design mock-ups, but the content of the images themselves have not been edited at all.

### 6.3.1 Landing Page

#### Initial View Comparison



[59] Landing page comparison

The landing page is the first view a user sees when loading Tournamint. Faithful to the design specification, the implementation sticks to a single input at once methodology. The user is first asked to enter their summoner name. If the summoner name exists in the Upstash data store (Next API call), then the user is asked to log in. If it does not, they are asked to register. These stages have been documented below. Important to note is that once the summoner name field has been filled and the user has either pressed the 'Enter' key or clicked the 'Get Started' button, the summoner name field is greyed out and disabled. This way the number of user inputs at one time remains one.

The code that drives the conditional logic surrounding the landing page input fields is all activated through the green button whose initial value is “Get Started”:

```
<button
  onClick={() => {
    if (passcode_showing && db_account_exists) {
      loginAccount(name_input, pass_input)
    }
    if (passcode_showing && !db_account_exists) {
      createAccount(name_input, pass_input)
    }
    if (!passcode_showing) {
      checkName( )
    }
  }}
  className="btn glass bg-[#00ff95] text-black-600"
>
  {passcode_showing && db_account_exists ? 'Log In' : null}
  {passcode_showing && !db_account_exists ? 'Register' : null}
  {!passcode_showing ? 'Get Started' : null}
</button>
```

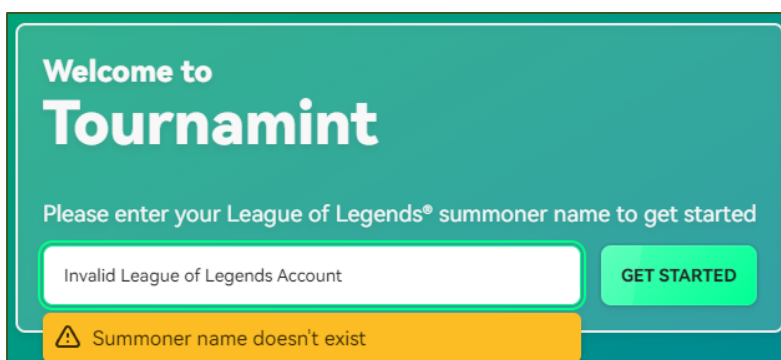
[60] Landing page conditional rendering

The loginAccount and createAccount functions drive the API calls to check validity of entered data and adjust the view to the user.

#### Summoner name not found in Riot Game’s servers

League of Legends has multiple regions. For this project, only the EUW (Europe West) region is supported to simplify API logic and reduce errors, but in future the option for multiple regions would be developed.

The first check the application makes when a user enters their summoner name, is that it exists on Riot Game’s servers. It does this utilizing Axios and the League of Legends API. If the summoner name is valid, the application will proceed to asking a user to enter a passcode, but if it is not valid, the user is shown the following view:

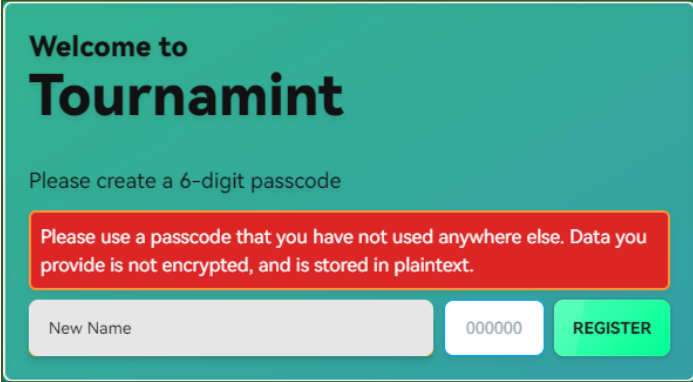


[61] Invalid summoner name

The notification disappears after a few seconds (by using a JavaScript setTimeout function), and the user is free to attempt to enter a valid name any time.

### Summoner name not found in Upstash data store

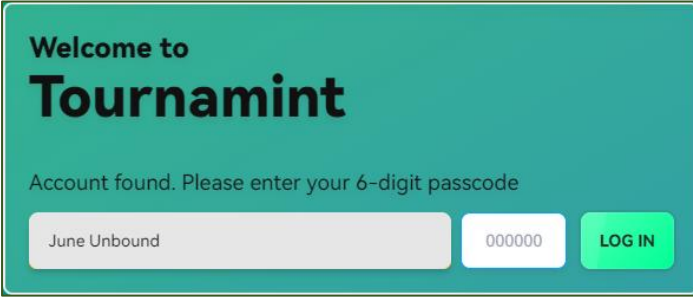
This view is shown to the user when registering for a new account. The data security message discussed in section 5.3.2.1 *System Security* is also shown at this stage.



[62] *New account - registration*

### Summoner name found in Upstash data store

If the user has already registered an account, this is the view that is shown them. It is displayed when the summoner name entered matches a summoner name in the Upstash data store.



[63] *Existing account - login*

### Invalid passcode entered for new account

In order to validate the passcode input for a new account, RegEx can be used. A passcode must be a 6-digit numeric string. The following expression validates a string that is 6 characters long and is made up of digits from 0-9:

```
const regex = new RegExp('([0-9]*[0-9]){6}')
```

[64] *Passcode RegEx string*

By comparing the passcode string the user has entered with the regex object using JavaScript's `.test()` function, either a true or false boolean result is returned. If the result is true, the user's account is created with the passcode they entered. If it is false, the user will be informed that their input was invalid. While this could be done with a custom element coupled with JavaScript code driving a `useState` hook for dynamic DOM updates, it is simpler and more efficient to pass the regular expression to the HTML input element instead:

```

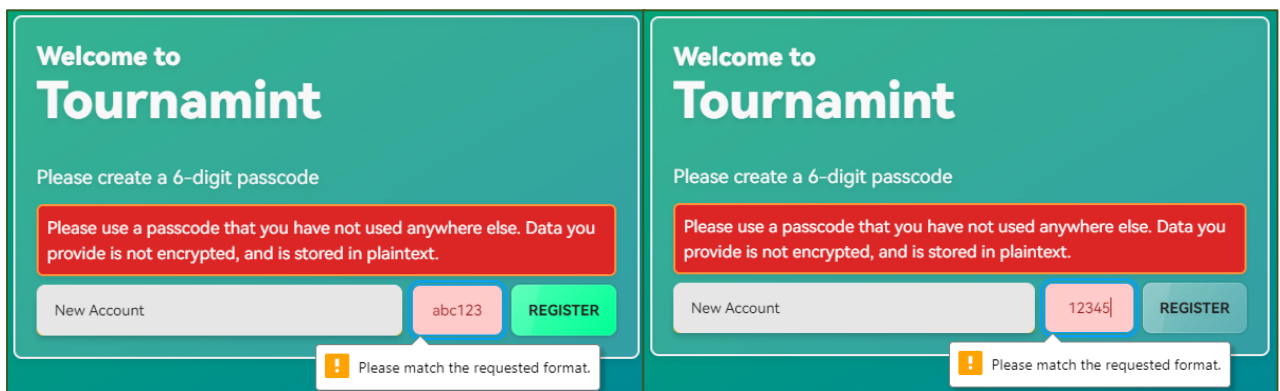
<input
  ref={passInputRef}
  type="text"
  inputMode="numeric"
  pattern="([0-9]{6})"
  placeholder="000000"
  maxLength={6}
  ...
/>

```

[65] HTML DOM regex usage example

HTML input elements support many different parameters to validate data in fact. Alongside the regular expression, the input element can also be capped at 6 characters, which physically prevents a user from entering more, an `inputMode` of 'numeric', so that non-numeric characters cause specific styling to occur (For this application, the developer has chosen to make the background and text shades of red to signify invalidity). A placeholder of '000000' also clearly shows the user what kind of input the field is asking for.

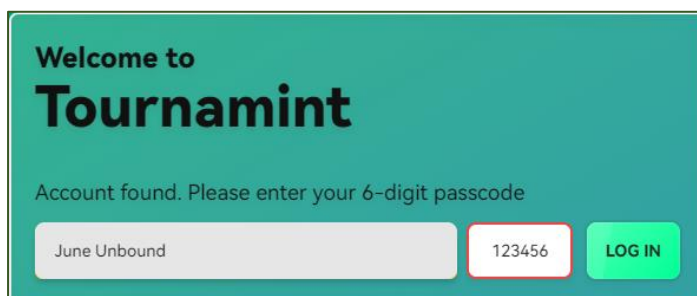
Together, these elements cause the following view to be shown to the user when they enter an invalid passcode while registering for a new account with Tournamint:



[66] Invalid passcode format

### Passcode entered while logging in does not match Upstash data store

When the user enters a passcode, it is compared with the passcode stored in the account hash for the user's summoner name. If the passcode matches, the user is logged in and is shown the main page of the application. If it however does not match, they are shown this view.

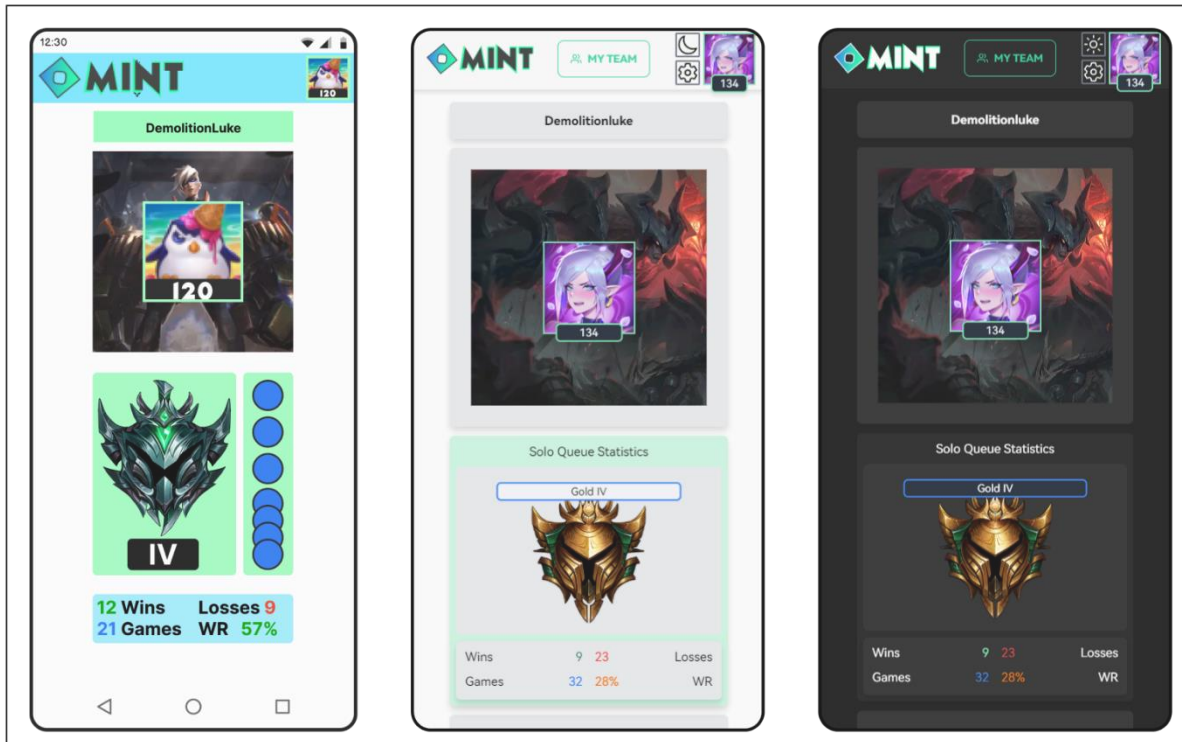


[67] Incorrect passcode

The red outline around the passcode tells the user that they entered a valid, yet incorrect passcode, urging them to try again with the passcode that matches their account.

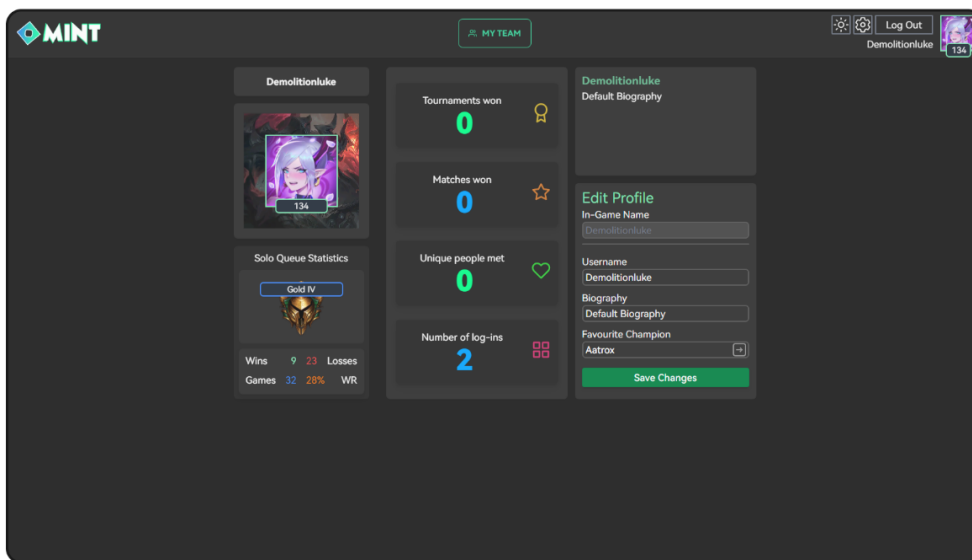
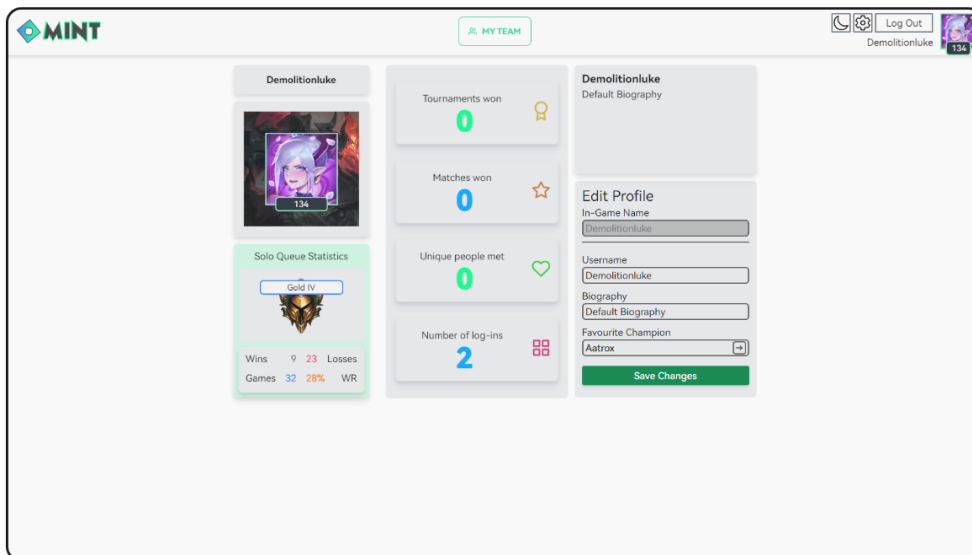
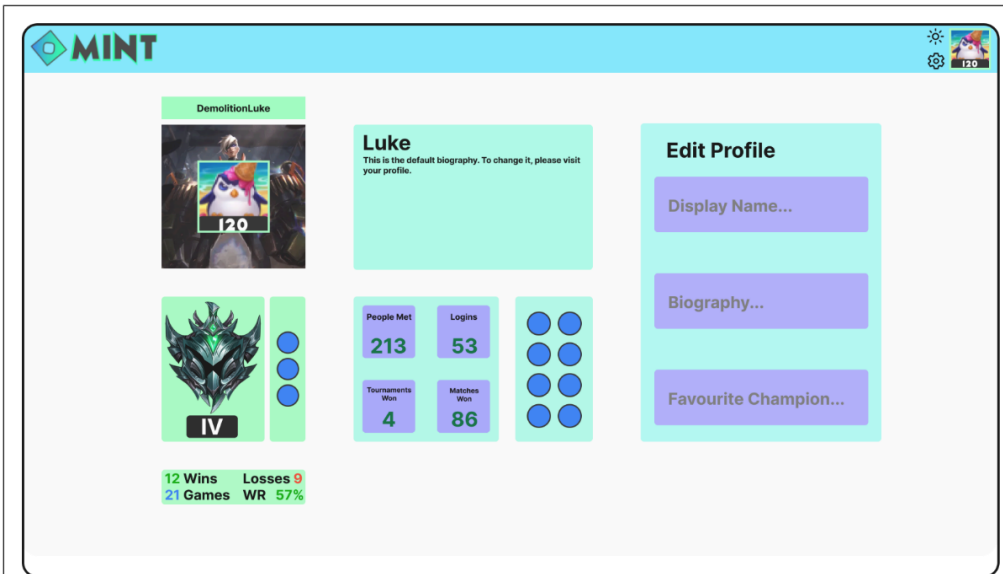
### 6.3.2 Profile Page

Once a user has registered for a new account with a valid summoner name and passcode, they are shown the profile page. As in the design section, this is the only page that will be shown in this section in both profile and desktop versions. It is also the only page that will be shown in both light and dark mode. Future screenshots will all be taken in dark mode due to its higher popularity among users and starker colour contrast, which will aid in viewing in a small format such as this report.



[68] Mobile profile comparison





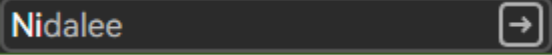
[69] Desktop profile comparison

Elements worthy of note are the theme-adaptive MINT logo (reasoning discussed in 5.4.3 *Tournamint Assets*), the lack of tournament trophies, the new segmentation of data and favourite champion logic and custom autocomplete feature.

While trophies viewable in a user's profile are a nice novelty and definitely a positive feature for this application, due to time constraints as well as implementation difficulties the idea has been tabled for potential future implementation.

The new layout, i.e., segmentation of data, is down to two reasons. In the design mock-up the profile editing section and the display of said data which it affected were in different columns. In the implementation, the centre column has been dedicated to a user's statistics, the left column solely to data gathered from the League of Legends servers, and the right column solely to Tournamint data. Although the favourite champion field affects the splash art of the lefthand column, the favourite champion data itself is stored in the Upstash data store.

In order to get the favourite champion selection working correctly with all 161 League of Legends champions, an algorithm was devised that can facilitate autocomplete in the DOM as well as the data itself. The results are as follows:



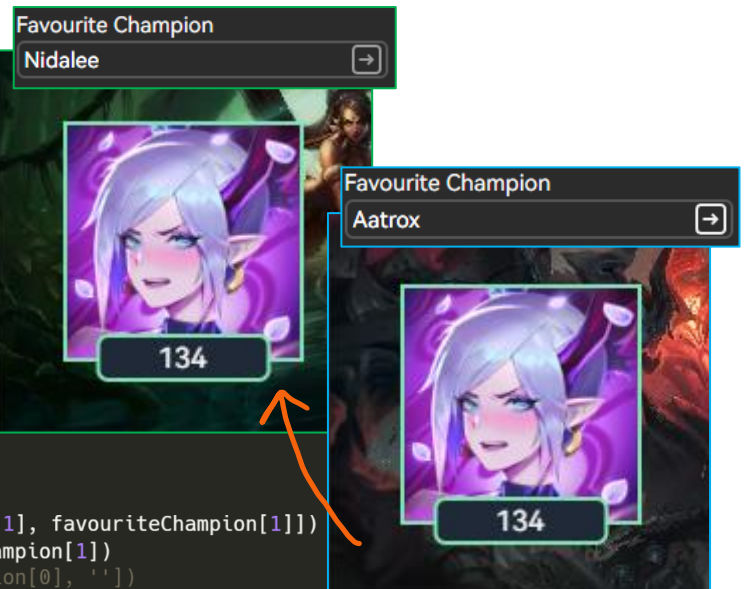
[70] *Favourite champion autocomplete*

By entering part of the name of a champion, the custom algorithm checks its' constant array list of champions and by using JavaScript's .filter function, matches the champion that resembles the user input the closest.

```
// Capitalize first letter
e.target.value = Capitalize(value)
if (e.target.value !== '') {
  let match = CHAMPIONS.filter((event) =>
    event.startsWith(e.target.value.toLowerCase()
  ))[0]
  setFavouriteChampion([value, match])
  if (value?.toLowerCase() ==
  match?.toLowerCase()) {
    triggerSplashUpdate(match)
  }
}
```

```
switch (key) {
  case 'Enter':
    e.preventDefault()
    saveUserDetails()
    break
  case 'ArrowRight':
    e.preventDefault()
    setFavouriteChampion([favouriteChampion[1], favouriteChampion[1]])
    e.target.value = Capitalize(favouriteChampion[1])
    // setFavouriteChampion([favouriteChampion[0], ''])
    triggerSplashUpdate()
    break
  ...
}
```

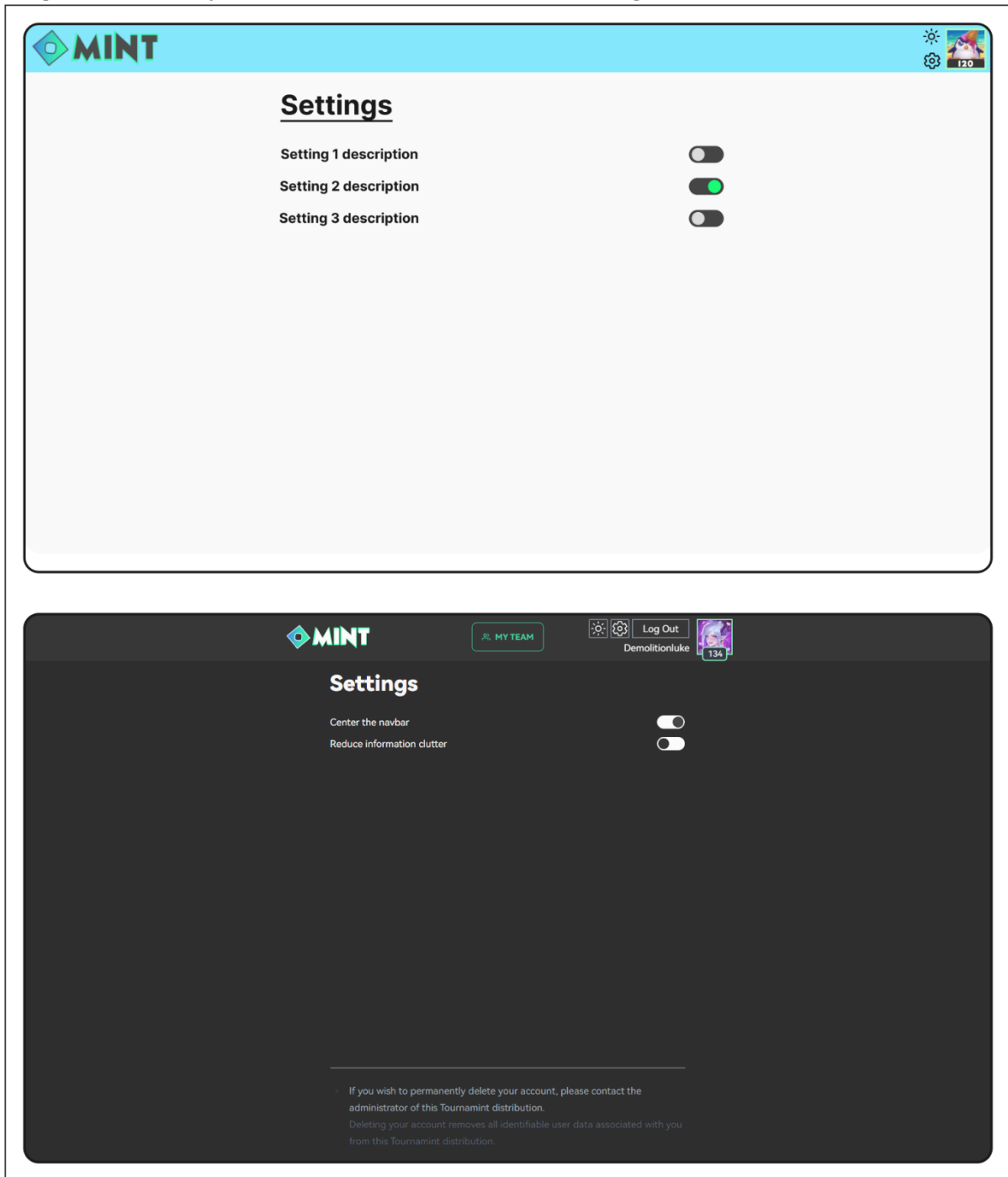
The code automatically capitalizes the user's input, shows them the suggested champion based on their input and when the user presses the right arrow key, the algorithm automatically fills the input with the champion it was suggesting. When the user then presses the enter key, the splash art is updated with the champion they selected.



[71] *Favourite champion splash change documentation and examples*

### 6.3.3 Settings Page

The settings page, accessible through the cog button in the navigation bar, is a useful, but not yet fully realised feature of Tournamint. As the application grows and receives new features, the settings page will become far more developed and useful than it is at this stage, but the groundwork has been fully completed. It currently has a functional “Center the navbar” setting.



[72] Settings page comparison

In `types.tsx`, the global file that contains many TypeScript interfaces, there is an addition called `ISettings`. This interface is future-proof and expandable. When a new setting is required for the application, it should initially be added here for use in subsequent code.

```
export interface ISettings {  
  centered_navbar: boolean  
}
```

Algorithms in the settings page save each setting in Local Storage, the useUser hook and the Upstash data store. This way settings are saved within a session, from session to session, and also when logging in from a new browser or computer/mobile device.

[73] *ISettings interface*

Additionally, information at the bottom of the settings page reads as follows:

'If you wish to permanently delete your account, please contact the administrator of this Tournamint distribution.'

'Deleting your account removes all identifiable user data associated with you from this Tournamint distribution.'

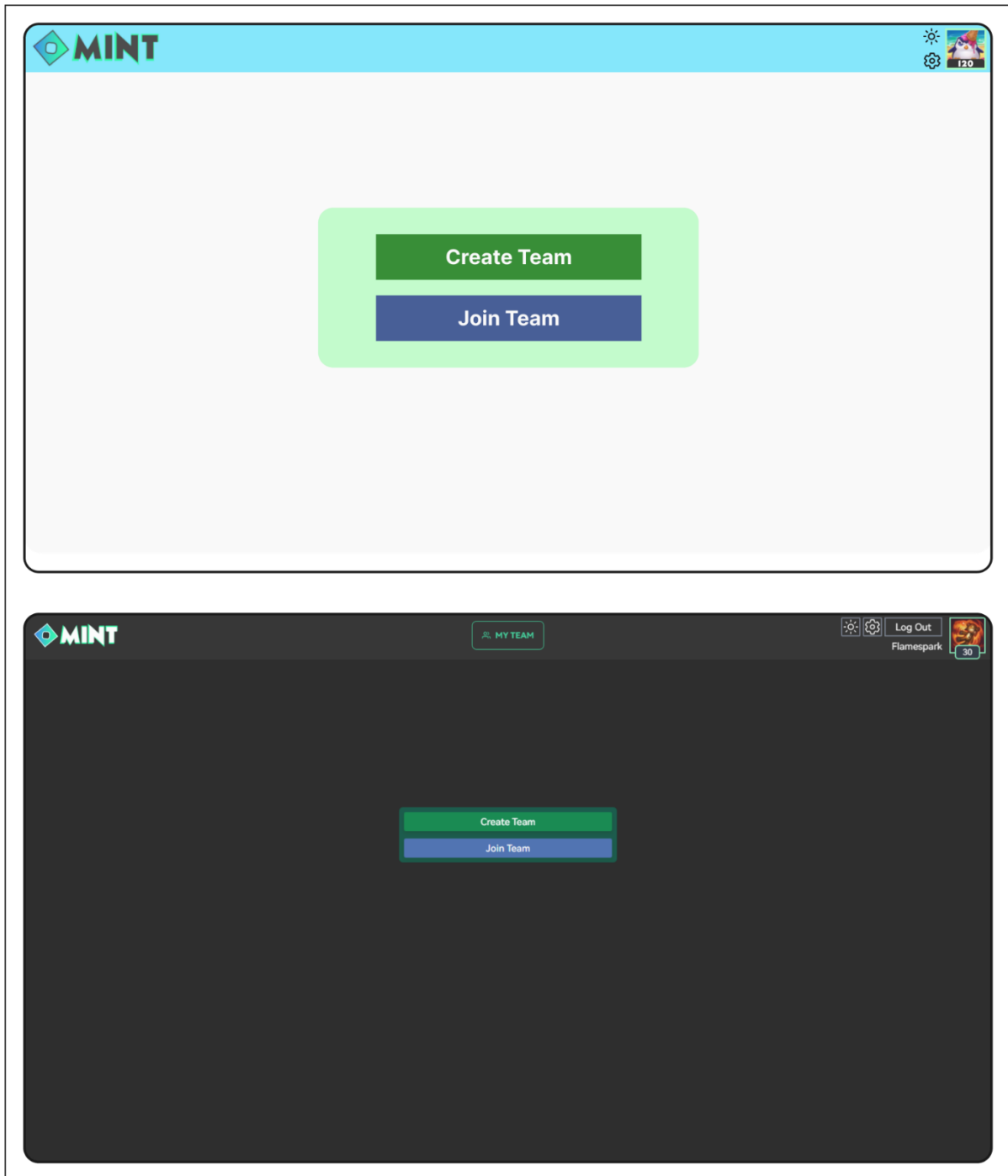
As Tournamint is an open-source, deployable application, a distribution may implement its' own account deletion algorithms. However, at this stage of development to avoid unforeseen problems relating to account deletion this feature has been made manual, so that it is up to the distribution's administrator to handle.

#### 6.3.4 Team Page

The team page, as laid out during the design sections, has only one main page, but features modals and conditional rendering to present the user with different information based on what stage they are at in team creation/joining. Each stage has been separated into a different subheading with its' own design comparison.

##### 6.3.4.1 Main Page (Initial)

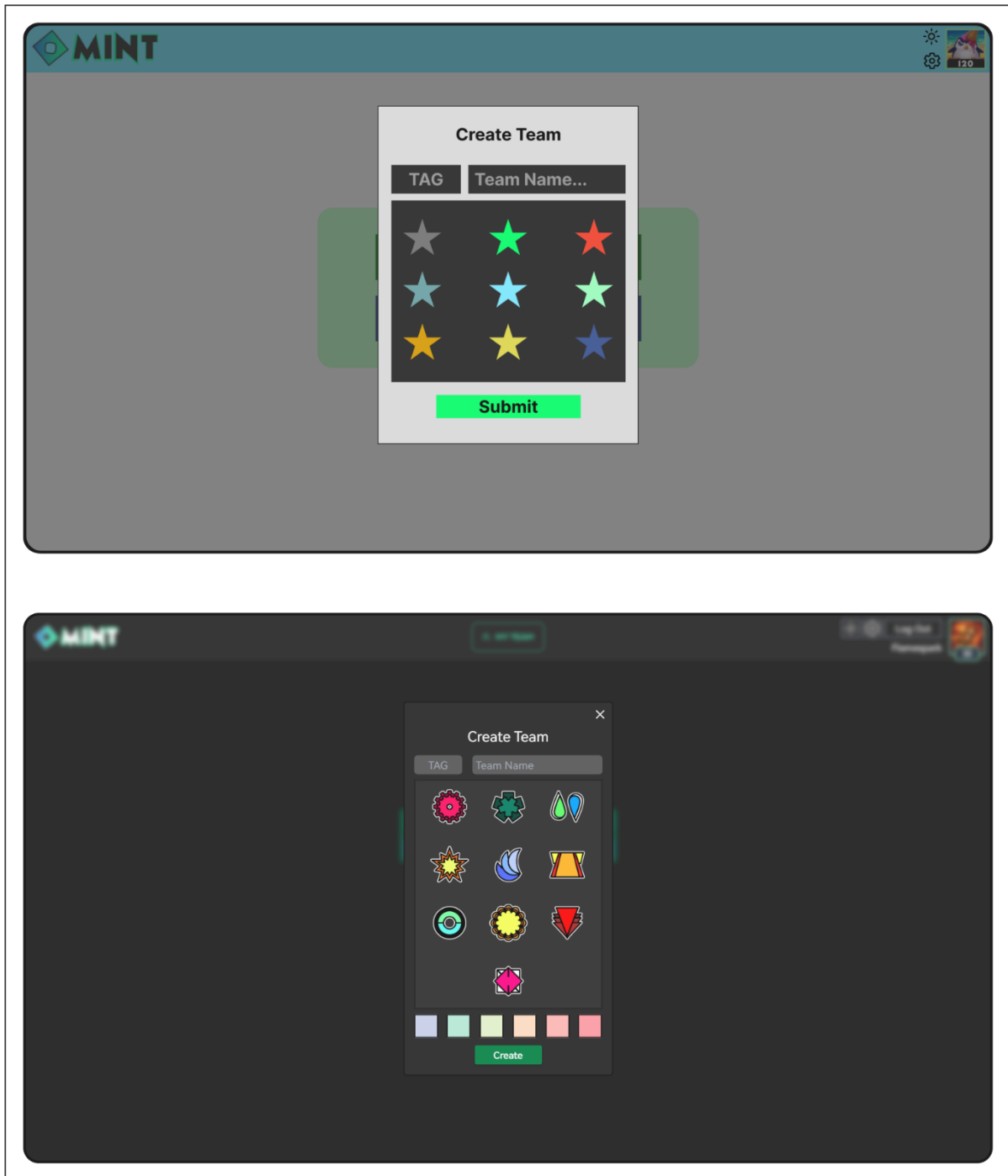
The initial view of the main page is shown when a user is not part of a team. Akin to the landing page, very few options are shown to the user, namely Create Team and Join Team. Very little has changed from the design mock-ups, with each button toggling a variable that conditionally shows a DaisyUI modal containing the a form.



[74] Team initial page comparison

#### 6.3.4.2 Team Creation

When clicking “Create Team”, a modal opens where a user can enter the team tag, name, icon, and – an addition to the design mock-up -, a colour. Much in the same realm as the team icon, a team colour is another customization option that changes how the team’s information is displayed on the team page, and during the tournament phase. The reasoning behind this addition is that with many teams, 10 icons is not enough to differentiate between teams. Adding 6 possible team colours to choose from changes the number of unique team customizations from 10 to 60.

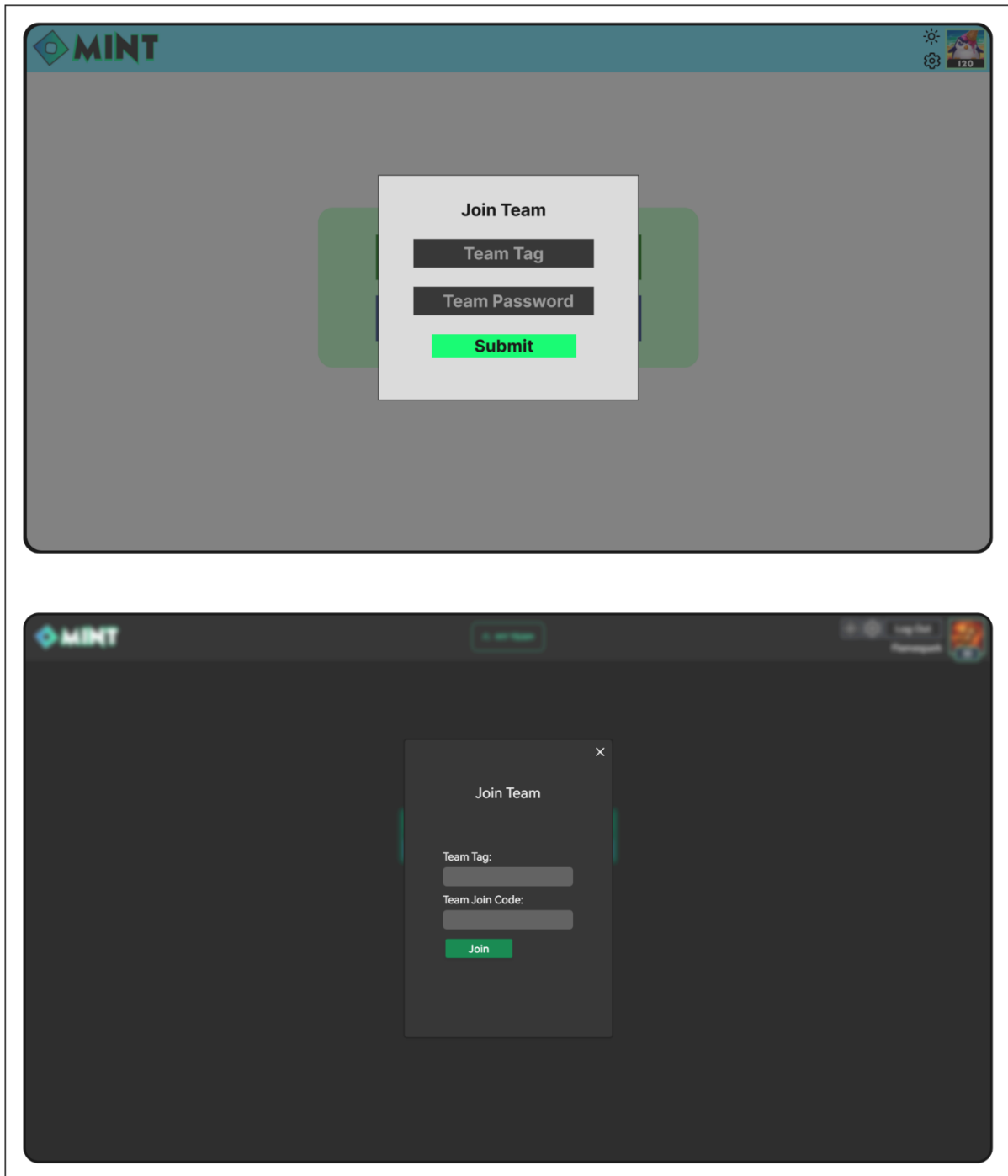


[75] Team creation comparison

Upon clicking the “Create” button, the application creates a team with the selected information, saving the data to Upstash in the Team hash map, and under the user’s account hash and in Local Storage and the useUser hook.

#### 6.3.4.3 Team Joining

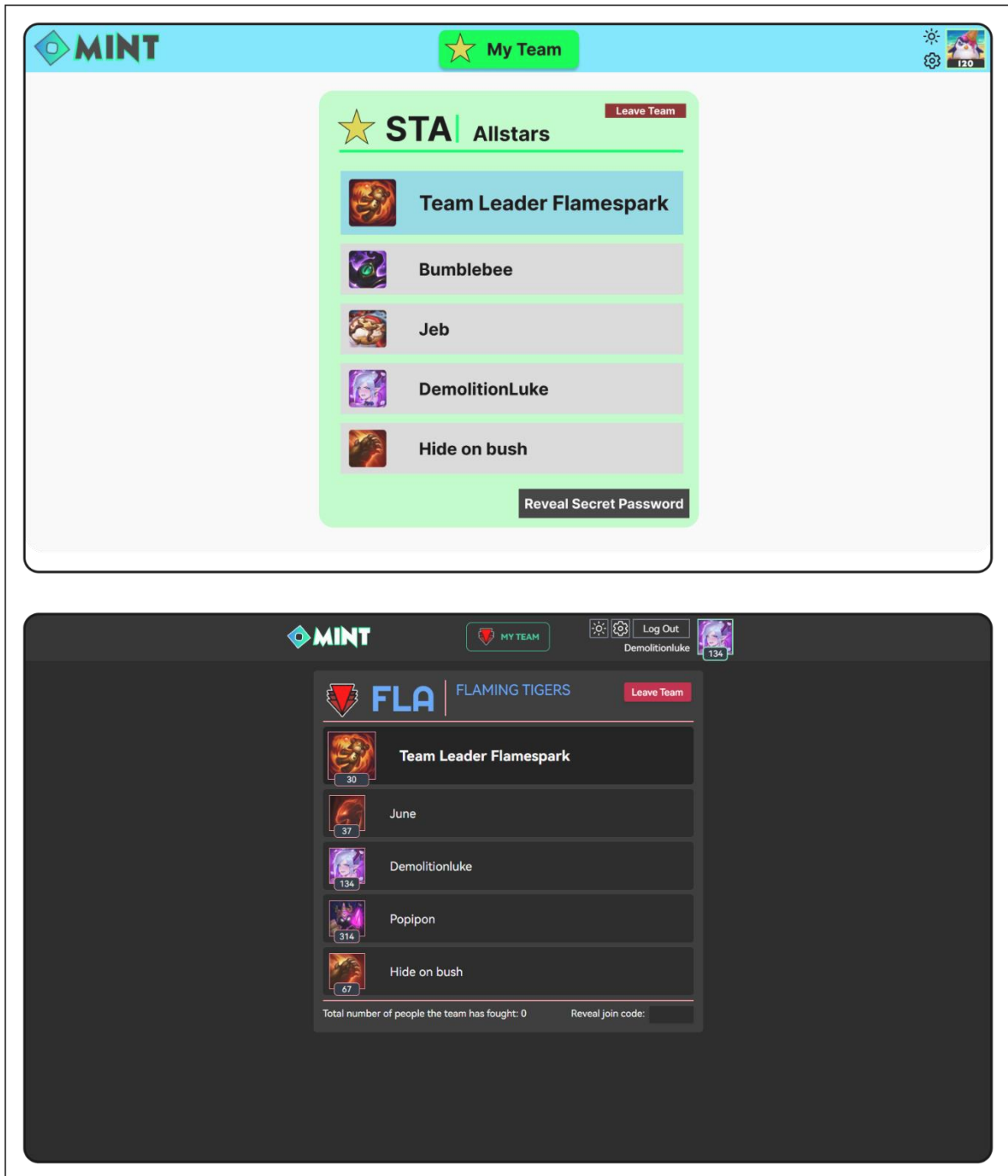
Similarly to team creation, the team joining modal is opened by clicking the “Join Team” button on the initial team page. This modal is much simpler than the team creation modal, only requiring a user to enter two pieces of information – The team’s tag and join code. This modal follows the design mock-up exactly.



[76] Team joining comparison

#### 6.3.4.4 Main Page (Team Information)

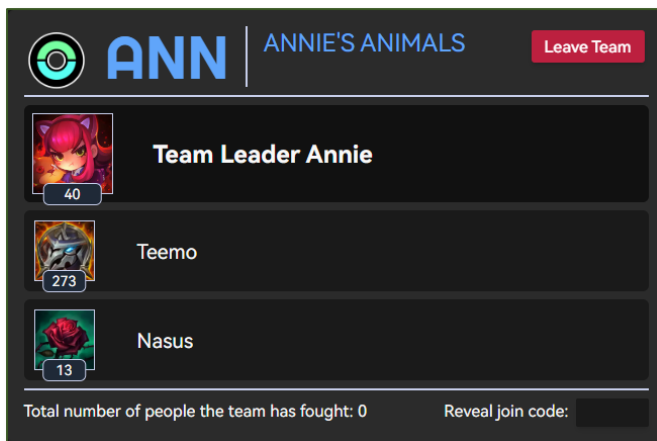
Once a user has created or joined a team, they are shown the team information every time they visit the main team page. The information displayed here is very similar to the design mock-up, with the only addition being a counter for the number of people that the team has fought. During a tournament, when a match completes, this number is incremented by 5. It's a flavour feature for a team to see how much they have played together.



[77] Team information comparison

The team roster fills up as more members join a team. For example, this is how the team information looks when a team has three members:

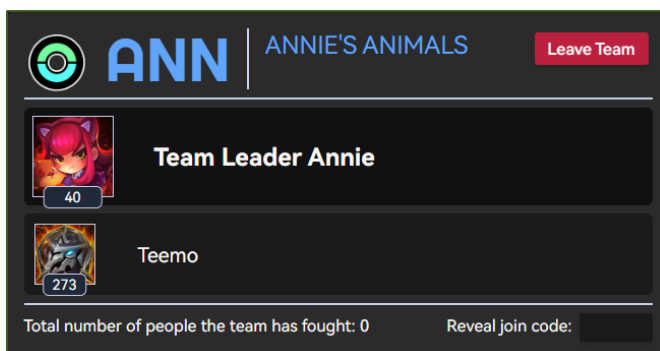




[78] Team information 3 members

Smaller notable additions to the original design include the colour of the straight-line borders around the team information, which matches the team's colour, selected in the team creation phase, and the display of team member's levels inside League of Legends.

Leaving a team is also possible by clicking the "Leave Team" button in the top-right hand corner. This will use an HTTP DELETE API request for the Upstash data store, as well as remove the information linking the user and the team from Local Storage and the useUser hook. To illustrate, here the player named 'Nasus' decided to leave the ANN team. When the team leader 'Annie' logged in, her team main page looked like so:



[79] Team information 2 members

### 6.3.5 Tournament Pages

As with the design section, the tournament phase is broken into multiple stages, namely the initial view, creating a tournament, joining a private/public tournament, the waiting phase, the seeding phase and the in-progress phase. To create the functionality for the tournament, conditional rendering was used generously. To organize the tournament states, an enumeration has been created, 'TOURNAMENT\_STATE'.

```
enum TOURNAMENT_STATE {
  BUFFERING,
  FILLING_UP,
  FULL,
  SEEDED,
  ONGOING,
  ENDED
}
```

This enum has 6 different states. Buffering is the standby, default state. When this state is active, the tournament display section will have a spinner similar to summoner icon pictures.

The FILLING\_UP & FULL states are sub-states of the waiting phase. FILLING\_UP is active when a tournament is longer than 1hr from starting and has not got a full roster of teams. FULL is active when a tournament is longer than 1hr from starting and has got a full roster of teams.

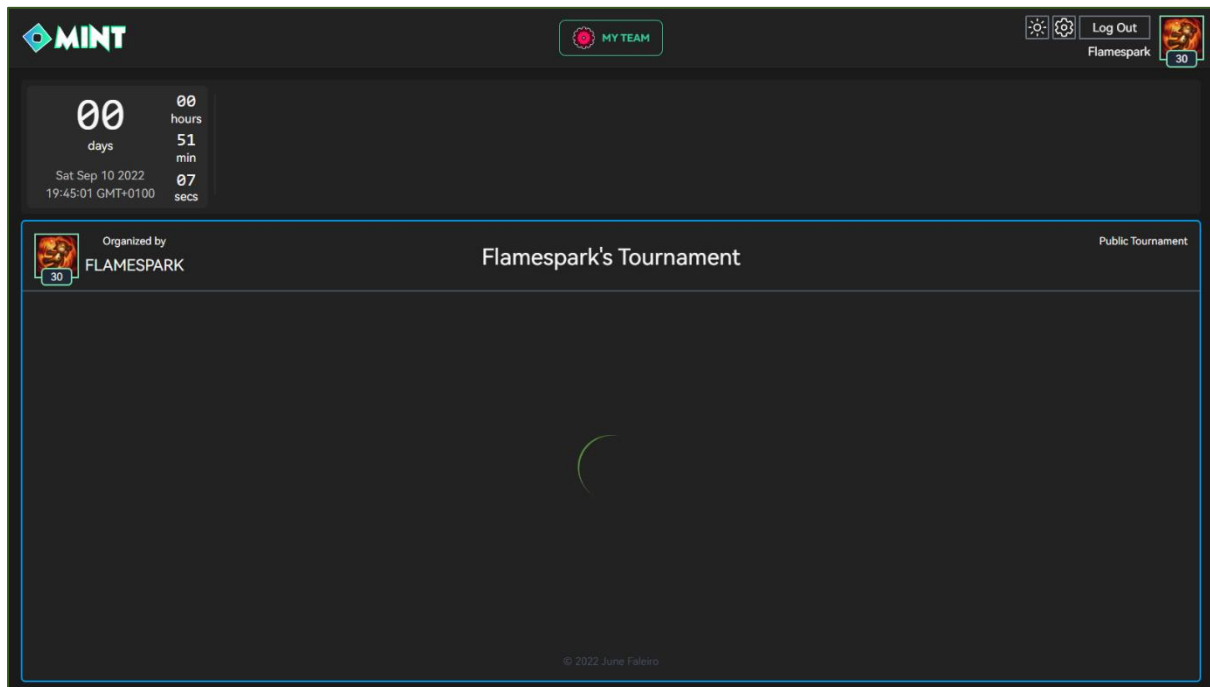
[80] Tournament state enumeration

The SEEDED state is active if a tournament is less than 1hr from starting, the tournament creator has logged in to Tournament, and the tournament has a full roster.

The ONGOING state is active when a tournament is in progress, meaning that the time is past the tournament start time, the tournament was successfully seeded, and the final match has not completed yet.

The ENDED state is active once the final match has completed.

The following is a screenshot of the tournament display section being in the BUFFERING state. This state is only active when other states are not, and indicates to the user that the application is waiting for data:



[81] Tournament buffering state

To use the TOURNAMENT\_STATE enum data to conditionally render the tournament data, the following code block is used within the main page's JSX:

```

{
  tournament_state == TOURNAMENT_STATE.BUFFERING && (
    </img>
  )
}
{
  tournament_state == TOURNAMENT_STATE.FILLING_UP && (
    <TournamentFillingUp
      team={team}
      tournament={tournaments}
    ></TournamentFillingUp>
  )
}
{
  tournament_state == TOURNAMENT_STATE.FULL && (
    <TournamentFull team={team} tournament={tournaments}></TournamentFull>
  )
}
{
  tournament_state == TOURNAMENT_STATE.SEEDDED && (
    <TournamentSeeded team={team} tournament={tournaments}></TournamentSeeded>
  )
}
{
  tournament_state == TOURNAMENT_STATE ONGOING && (
    <TournamentDisplay team={team} tournament={tournaments}></TournamentDisplay>
  )
}
}

```

[82] Tournament state change conditional rendering logic

Important to note here is the `<TournamentFillingUp/>`, `<TournamentFull/>`, `<TournamentSeeded/>` and `<TournamentDisplay/>` components. These React component take team and tournament parameters (of types `ITeam` and `ITournament`), and produce a view to the user based on the type of state and the data it is given. These components dynamically display data based on these parameters.

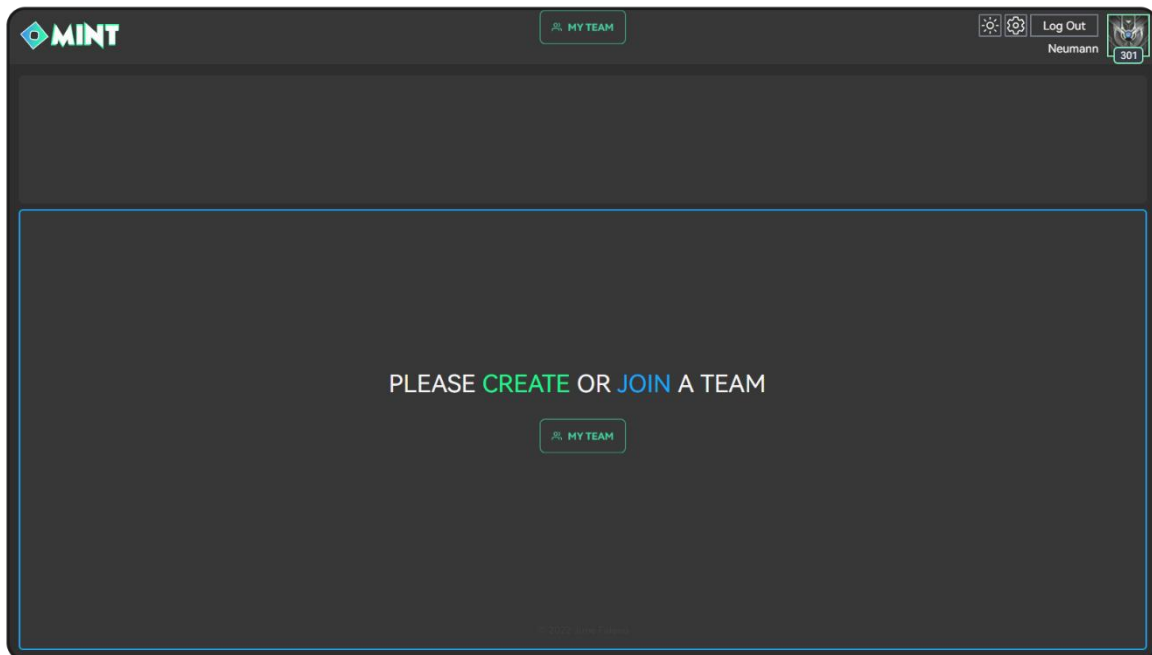
With this in mind, the following sections are named primarily after the design section’s naming convention, but also include the tournament’s state in their titles.

#### 6.3.5.1 Main Page (Initial) – No state

The initial view of the main page is dependent on the user’s current progress in the application. While design mock-ups weren’t created for the different views a user would receive, this section will focus on the application’s guidance to the user to help them use Tournamint.

##### No Team

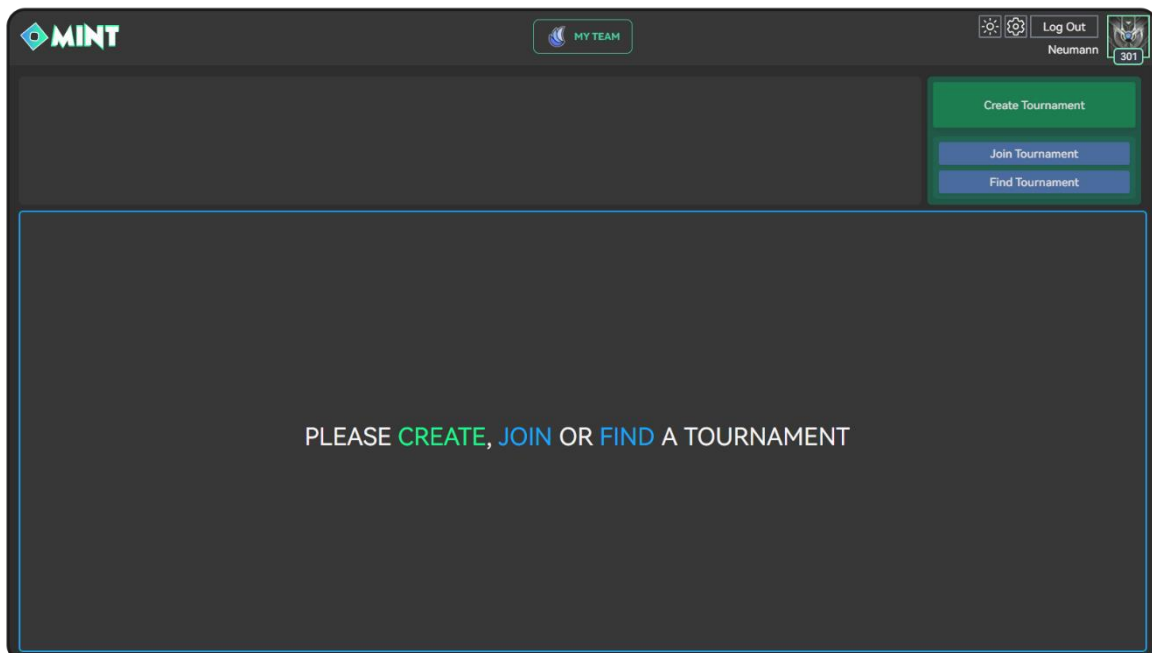
If the user is not yet part of a team, then the main page of the application will show the following view. It urges the user to either create or join a team, and gives them a functional “My Team” button in the center of their screen. This button has a small animation (which is not visible in a static report), drawing the user’s eye to it. It also has the same design as the “My Team” button in the navbar, helping the user link the two for future use.



[83] Main page, stateless, no team

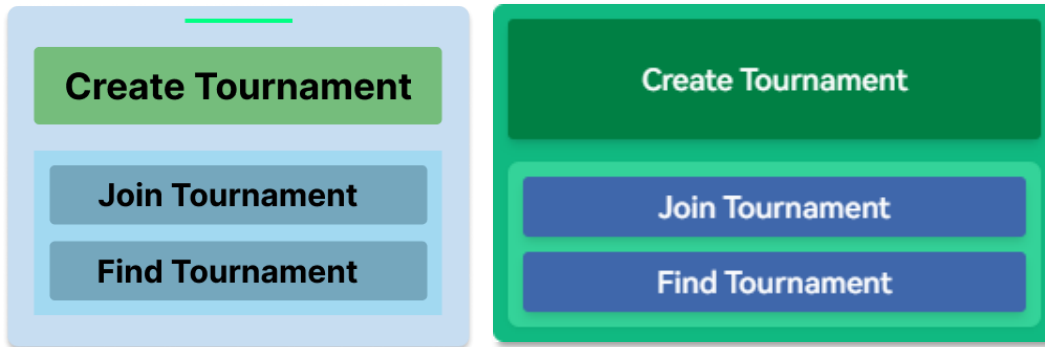
### No Tournament

Once the user is part of a team, the main page will urge them to either create, join or find a tournament. A noticeable block is also visible in the top right of the screen, which has a light pulsing animation, drawing the user's eye. The intense splash of colour that these buttons have in comparison to the rest of the application's main page is intentional, as is the linking of the colour green for create, and blue for join and find. These design choices help guide a user subconsciously to picking the options the application is trying to help them to.



[84] Main page, stateless, no tournament

In comparison to the design mock-up, the create and find tournament button box has changed very little, except for styling such as drop-shadows and colour decisions:



[85] *Tournament buttons comparison*

### 6.3.5.2 Create Tournament Page – No state

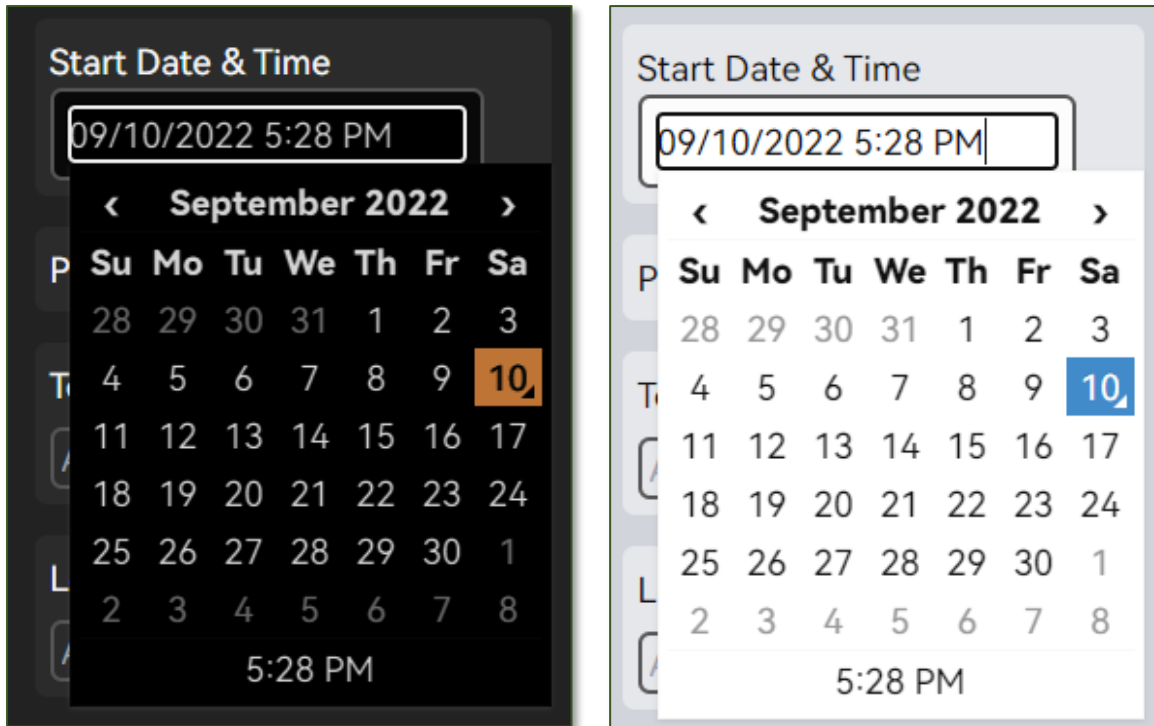
If a user chooses to create a tournament, they are redirected to the tournament creation page.

The top screenshot shows a design mock-up of the tournament creation page. It features a light blue header with the 'MINT' logo on the left and a settings icon and user profile on the right. The form is centered and contains the following fields: 'Tournament Name' (text input), 'Number of Teams' (range slider from 4 to 16), 'Start Date & Time' (text input), 'Private' (radio button) and 'Public' (radio button) options, 'Tournament ID' (text input), and 'Secret Passcode' (text input). A green 'Create Tournament' button is at the bottom.

The bottom screenshot shows the actual implementation of the page in a dark theme. The header includes the 'MINT' logo, a 'MY TEAM' button, and a 'Log Out' button next to the user profile 'Neumann' with the ID '301'. The form is filled with the following data: 'Tournament Name' is 'Worlds Group Cup 2022', 'Number of Teams' is set to 8, 'Start Date & Time' is '09/10/2022 5:07 PM', 'Private' is selected, 'Tournament ID Code' is 'ABC123', and 'Lobby Code' is 'ABC123'. A green 'Create' button is at the bottom.

[86] Tournament creation comparison

While the content of the tournament creation page differs little from the design mock-up, the functionality is of course implemented. Much like creating a team, creating a tournament will send all the data a user enters to the Upstash data store for the tournament hash, and if the user makes the tournament public, it will be added to the list of public tournaments. Of note is the date/time picker, which is a fully styled, functional HTML date & time picker:

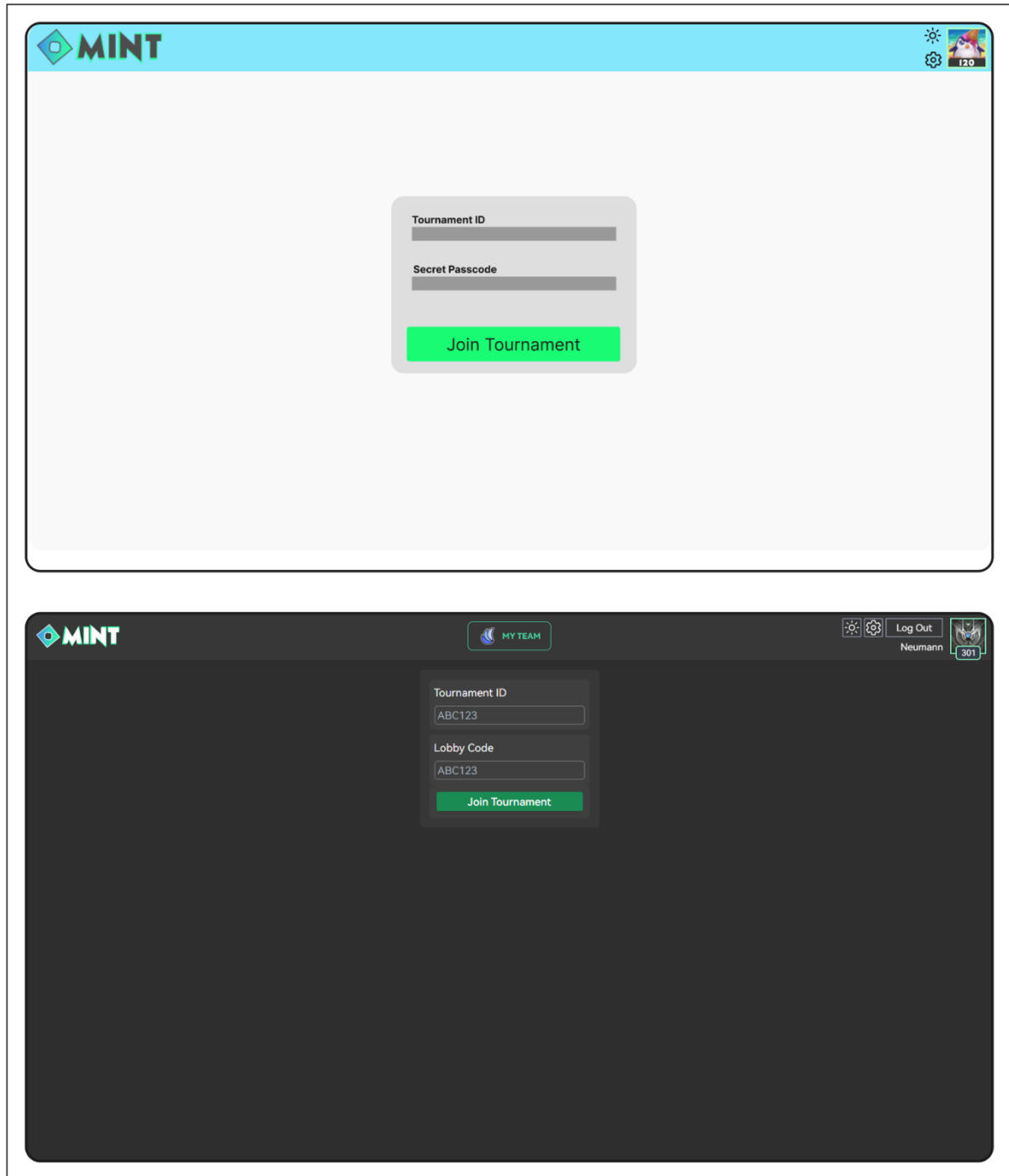


[87] Tournament creation date/time picker light & dark themes

When a date and time is entered here, using the Moment 3<sup>rd</sup> party library it is converted into a Moment object, which can be used and modified by the rest of the application.

### 6.3.5.3 Join Private Tournament Page – No state

Joining a private tournament is very similar in function to joining a private team. The tournament ID is used as the key for the tournament hash to locate the correct tournament, and the secret passcode is compared against the passcode stored in the hash. If there is a match, the user's team joins the private tournament. If not, the user must try a different secret passcode.



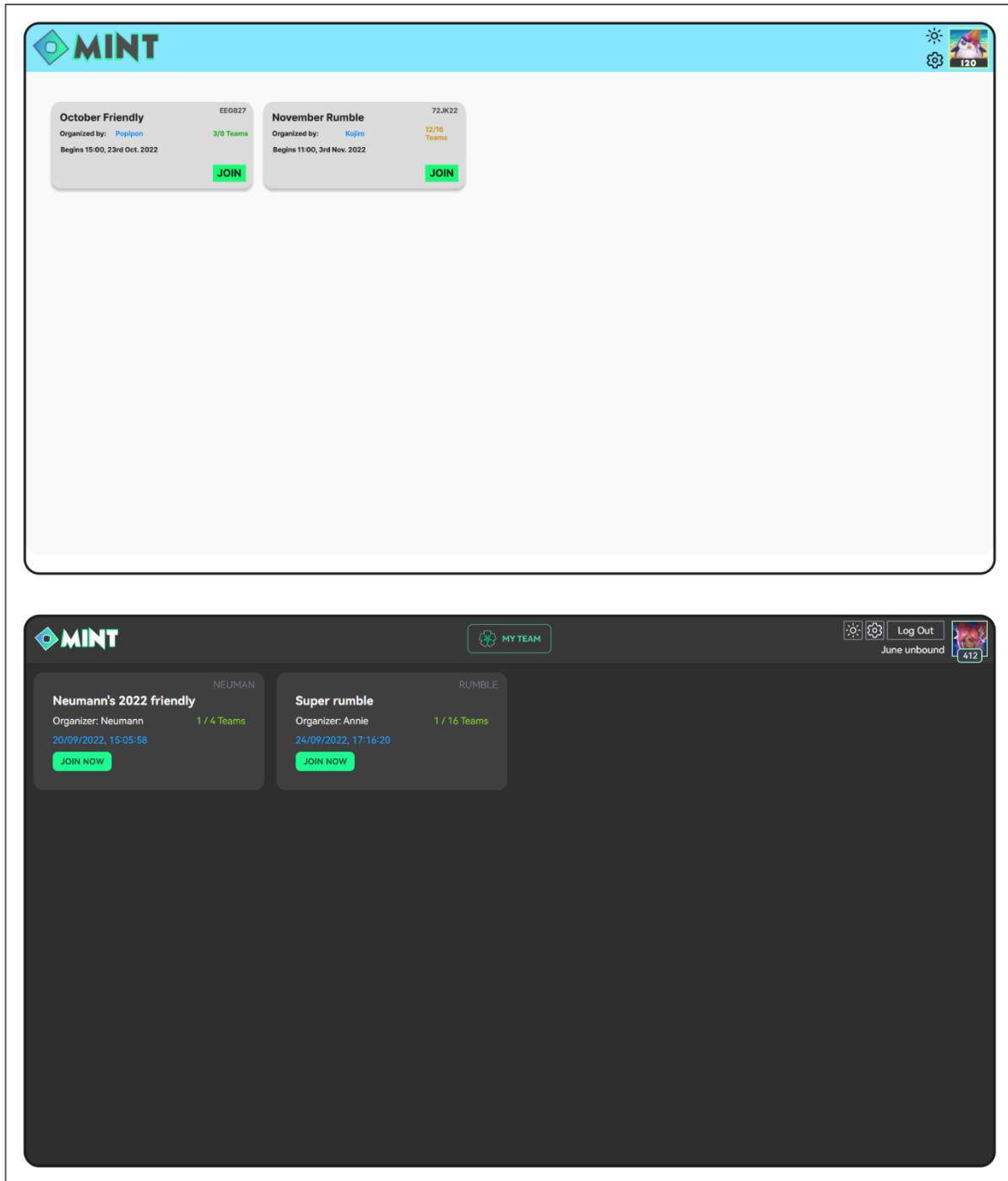
[88] Join private tournament comparison

Compared to the original design mock-up very little has changed in proposed and implemented function, as well as design.



#### 6.3.5.4 Join Public Tournament Page – No state

The ability to join a public tournament is exercised when clicking the “Find Tournament” button on the main page. A new page is loaded for the user containing a grid (or column on mobile) list of valid public tournaments. This means tournaments that have fewer than the max number of teams for that tournament, as well as their start time being later than the current time.



[89] Public tournament display comparison

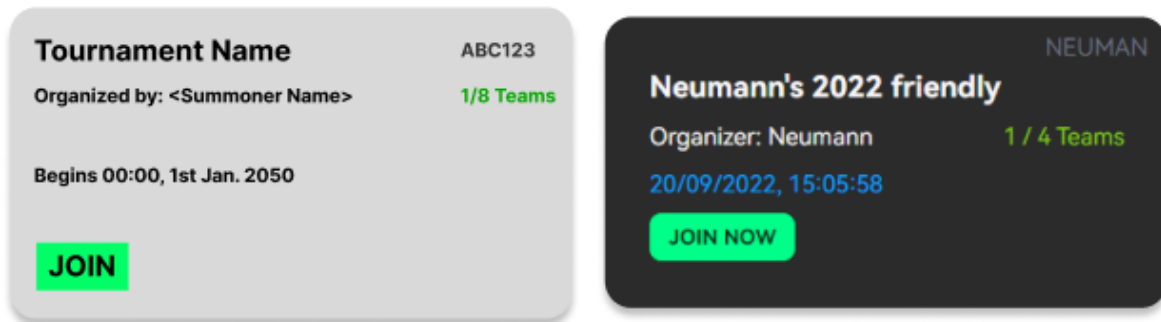
The initial design mock-up was heavily used for the final implementation, with DaisyUI card components being used for each tournament. Like in the design, conditional rendering is used for the colour of the team counter, with  $\leq 50\%$  full being green, and greater than  $50\%$  full being orange. The JSX code to achieve this is as follows:

```
<p
  className={` ${e.teams.length / e.type <= 0.5 ? 'text-lime-500' : ''} ${
    e.teams.length / e.type > 0.5 ? 'text-orange-500' : ''
  } flex justify-end`}
>
  {e.teams.length} / {e.type} Teams
</p>
```

[90] Conditional text render for tournament fullness

As the list of public tournaments is rendered using a JavaScript `.map` function, each tournament object is represented by the variable `e`. This variable's data, e.g., the tournament's teams, type and creator are accessible by the JSX code, being dynamically rendered.

To see a public tournament card more clearly, here is a comparison of the design mock-up and final result:

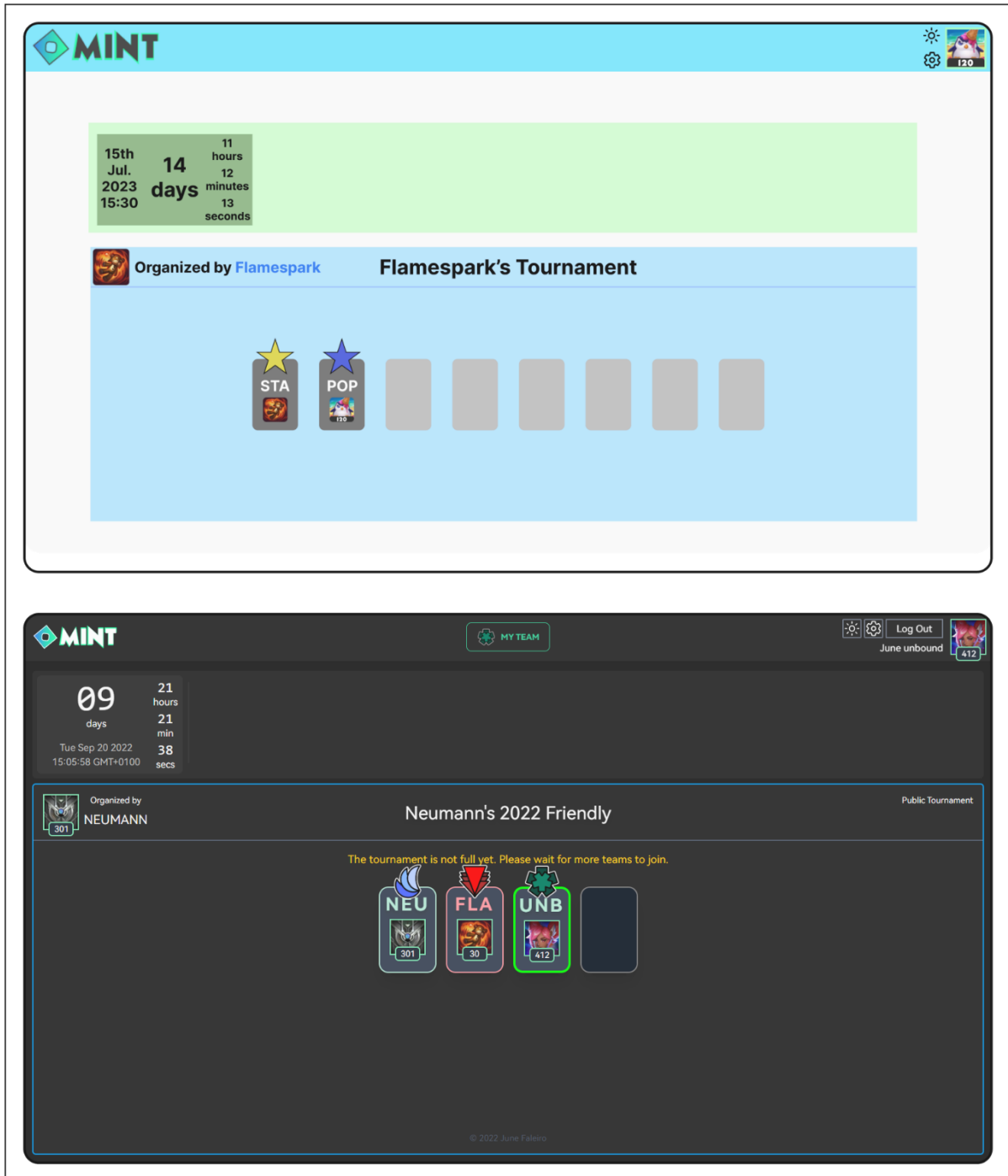


[91] Public tournament block comparison

By clicking the “JOIN NOW” button, a user's team is added to the tournament's data, and the team's data is updated with the tournament in Local Storage, the `useUser` hook and on the Upstash data store.

### 6.3.5.5 Main Page (Waiting Phase) – FILLING\_UP, FULL

Once a team owner has signed their team up to a tournament using either the code-based private tournament joining, or card grid-based public tournament list, they are witness to a phase in the tournament, that is, a condition-based dynamic view of certain tournament information. The first of these phases is the waiting phase, which encompasses the tournament's FILLING\_UP and FULL states.



[92] Tournament filling up comparison

During this phase, a user in the tournament's main Tournamint page is filled with two new elements, the countdown component and the team list component.

The countdown component shows a timer that counts down until the start of the tournament. This is achieved with a slick, eye catching animation provided by the DaisyUI component library, with the data provided by an algorithm using the Moment package. The code to drive the function of the countdown is

shown below. Firstly, the `useState` hook is used to provide reactive, dynamic data concerning the countdown. This data is split into days, hours, minutes and seconds, as these are all separate components on the page:

```
const [countdown_s, setCountdown_s] = useState < number > 0
const [countdown_m, setCountdown_m] = useState < number > 0
const [countdown_h, setCountdown_h] = useState < number > 0
const [countdown_d, setCountdown_d] = useState < number > 0
```

*[93] Countdown state hook initialization*

A `useState` hook has the immutable value, e.g., `countdown_s`, and a function which updates this value, e.g., `setCountdown_s`. This is used further in the code when calculating the values for the countdown component:

```
const countdownInterval = setInterval(function () {
  var now = new Date().getTime()
  var left = countDownTime - now
  if (tournaments) {
    left = moment(tournaments.date_time_start).toDate().getTime() - now
  }

  setCountdown_d(Math.floor(left / (1000 * 60 * 60 * 24)))
  setCountdown_h(Math.floor((left % (1000 * 60 * 60 * 24)) / (1000 * 60 * 60)))
  setCountdown_m(Math.floor((left % (1000 * 60 * 60)) / (1000 * 60)))
  setCountdown_s(Math.floor((left % (1000 * 60)) / 1000))
}, 1000)
```

*[94] Countdown calculation logic*

Using the modulus operator, each part of the countdown component can be accurately calculated to derive the precise number of days, hours, minutes and seconds until the start of the user's tournament.

As the code is wrapped in a 1-second repeating `setInterval` function, there is the possibility of a memory leak occurring on page refreshes or state updates. To combat this, the function is wrapped in a `useEffect` hook that supports memory clean-up as a callback:

```
return () => {
  if (countdownInterval) {
    clearInterval(countdownInterval)
  }
}
```

*[95] Countdown memory leak handler*

The second new element during this phase is the team list component. Rendered conditionally by the following code, this component dynamically displays information about all the teams participating in the tournament, such as their team tag, team icon, team colour and team owner's summoner icon and level.

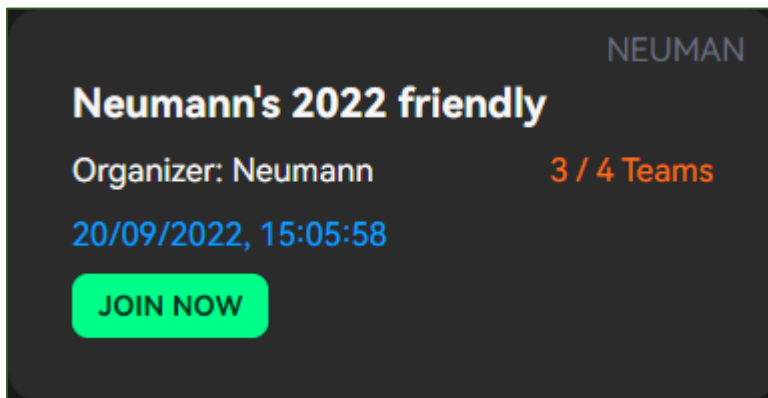
```
<TournamentFillingUp
  team={team}
  tournament={tournaments}
></TournamentFillingUp>
```

[96] *Tournament filling up component and props*

Based on whether the tournament is full or not, it will also inform the user in orange lettering what they should do. While the tournament is still filling up during the waiting phase, this message will read “The tournament is not full yet. Please wait for more teams to join.”

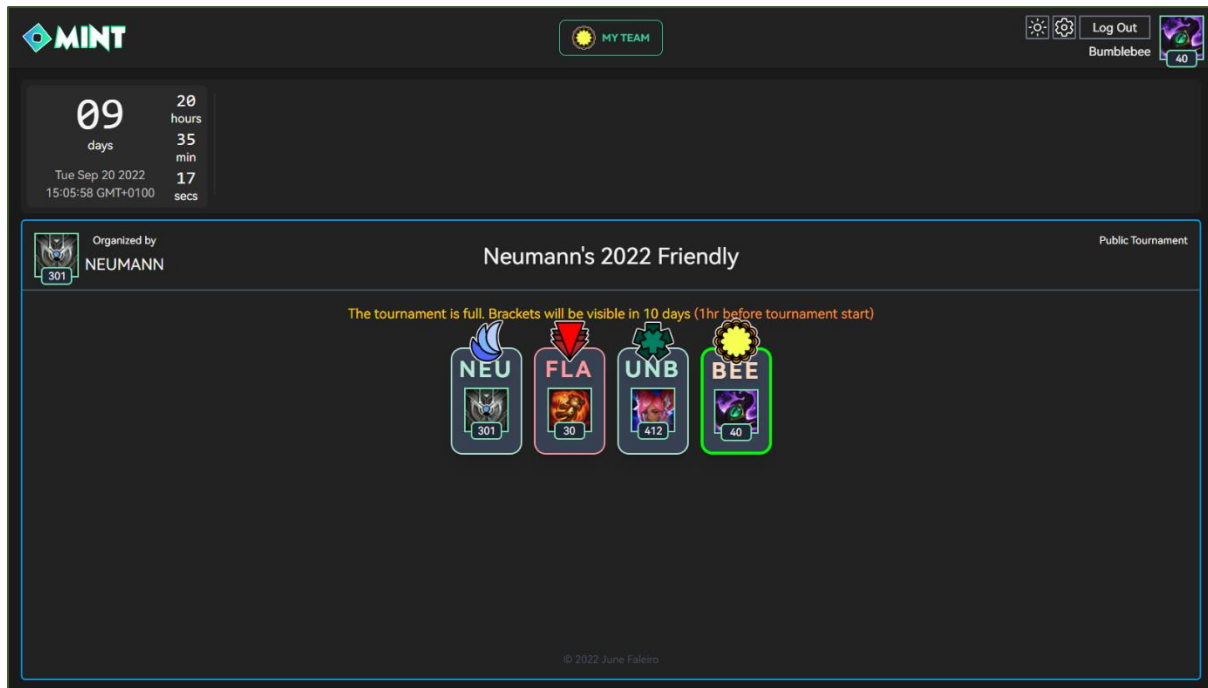
Once the tournament team roster has filled up, the tournament's state will change, and a new component will be rendered instead of `<TournamentFillingUp/>`.

As a short example, a new team with the tag BEE is looking to join a tournament. They browse the list of public tournaments and see that there is one spot left in Neumann's 2022 Friendly:



[97] *Public tournament block example*

The team leader of BEE, Bumblebee clicks “JOIN NOW”. Their team is entered into the tournament and Bumblebee's client is redirected to the main page of Tournamint. The view they now see is as follows:



[98] Waiting period main page display example

Although at first glance this screen seems very similar to the `<TournamentFillingUp/>` component, it is in fact a `<TournamentFull/>` component. While at first it may seem odd to create a new component when the only visible change is that the status message has changed from “The tournament is not full yet. Please wait for more teams to join.” To “The tournament is full. Brackets will be visible in X days (1hr before tournament start)”, there is a good reason.

Segmenting these states into separate components allows anyone wanting to create their own distribution of Tournamint to modify each state’s view separately. Quasi-cloning the filling up component to create the full component creates a tiny amount of extra space requirement, while the benefit of flexibility and separation of states outweighs that factor.

The status message for the `<TournamentFull/>` component is derived from the tournament’s start time using Moment. Moment even has readability-friendly functions that convert a time difference into human-readable language, i.e. (‘In 10 days’).

### 6.3.5.6 Main Page (Seeding Phase) – SEEDED

Once a tournament's roster has filled up and the time until tournament start is 1hr or less, the seeding phase begins. To start it, the tournament creator must log into Tournamint, where the client will automatically populate the tournament brackets with the teams that have signed up.

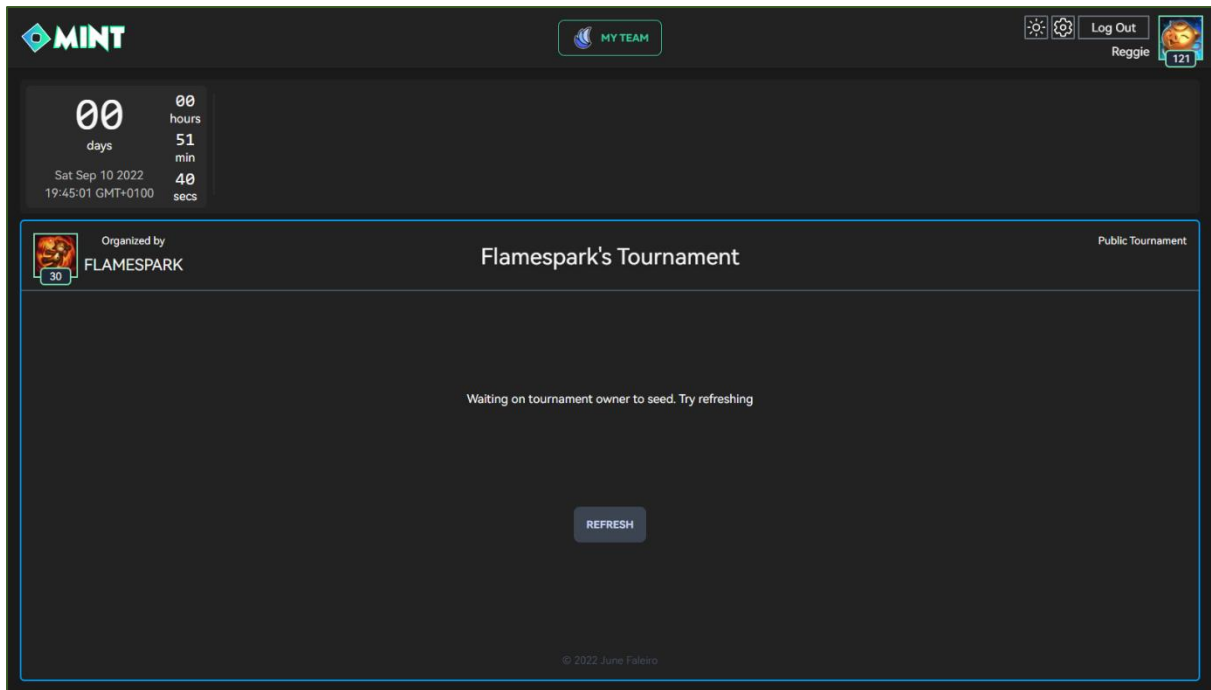
The image displays two screenshots of the MINT application interface during the seeding phase of a tournament.

**Top Screenshot:** Shows the tournament page for "Flamespark's Tournament". At the top, a green banner indicates the time until the start: "15th Jul. 2023 15:30" with a large "0 days" and a timer showing "0 hours, 42 minutes, 23 seconds". Below this, the tournament is organized by "Flamespark". The main content area shows a bracket of four matches, all labeled "SMG vs SMG" for "July 15th 15:30 CEST".

**Bottom Screenshot:** Shows the same tournament page after seeding. The timer now shows "00 days, 32 min, 11 secs" for "Sat Sep 10 2022 19:45:01 GMT+0100". The tournament is still organized by "FLAMESPARK" and is labeled as a "Public Tournament". A message states: "The tournament is seeded. Please see the initial brackets below. The tournament will start in 33 minutes". The bracket shows two matches: "STA vs FLA" and "FIL vs BOW".

[99] Seeded tournament comparison

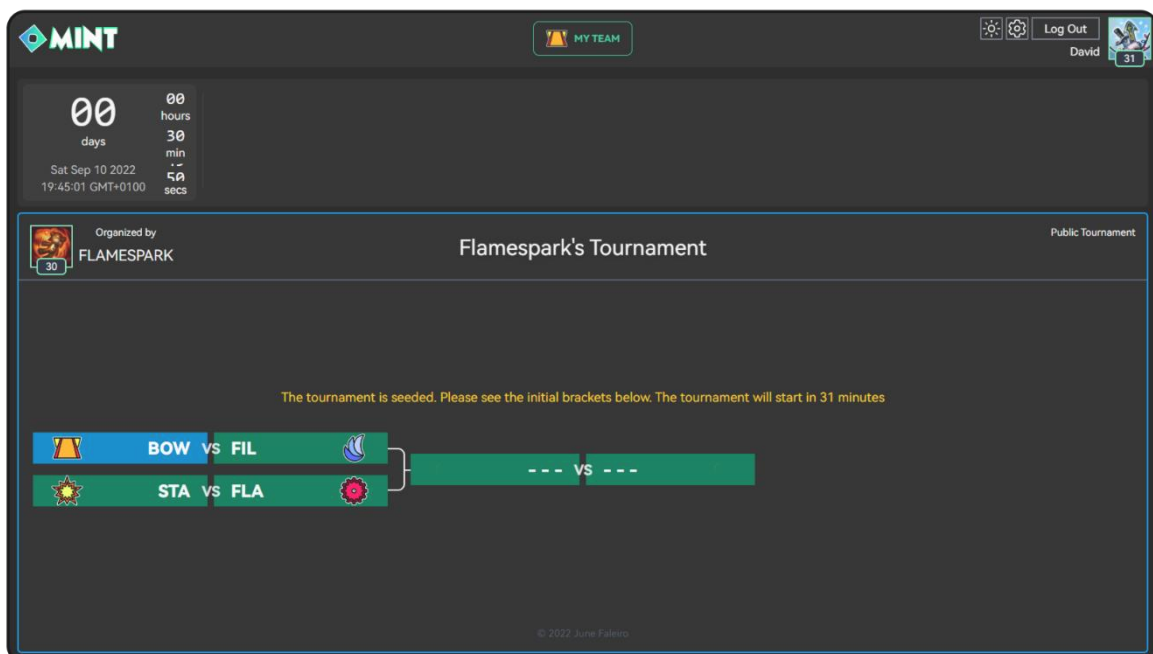
The view seen above is once the tournament is seeded by the creator. However, if a tournament participant logs in to their account before it is seeded, they will be met with the following view:



[100] Waiting for tournament seeding

The seeding view itself merely displays the tournament brackets, with no additional functionality. The user is also told the precise time (using the Moment library) of when the tournament starts.

The user's team is also highlighted in blue, compared to the rest of the teams being green. In the first image Flamespark's team was highlighted blue as she was logged in. When David, the team leader for team STA logs in though, his view is different, highlighting his team:

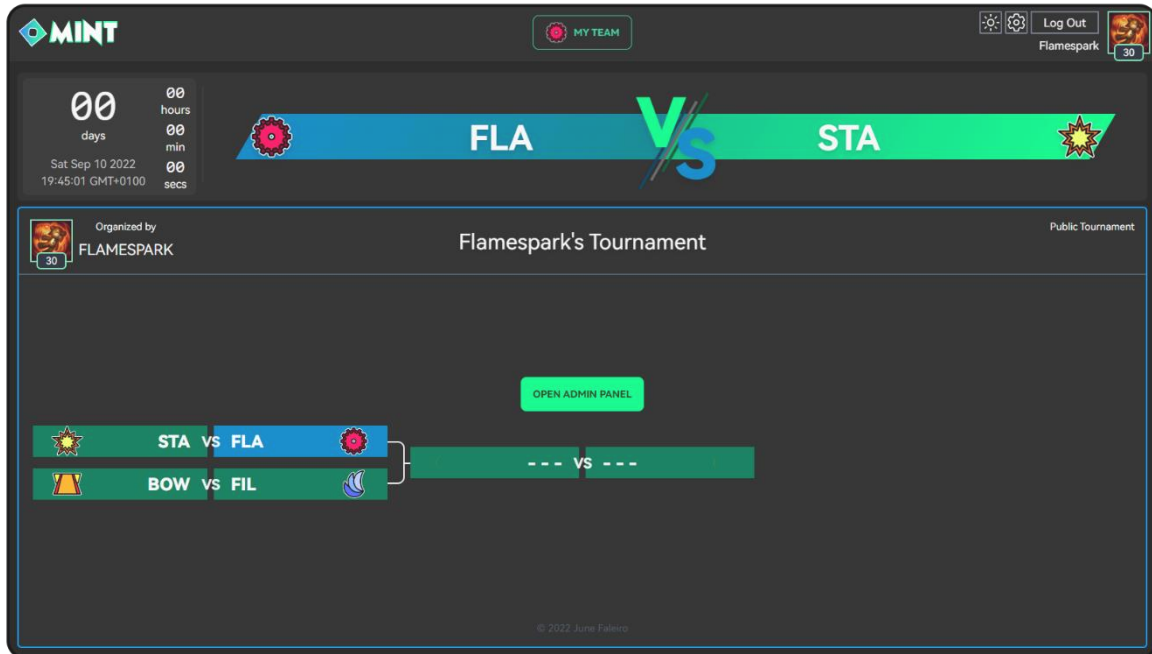


[101] Seeded tournament



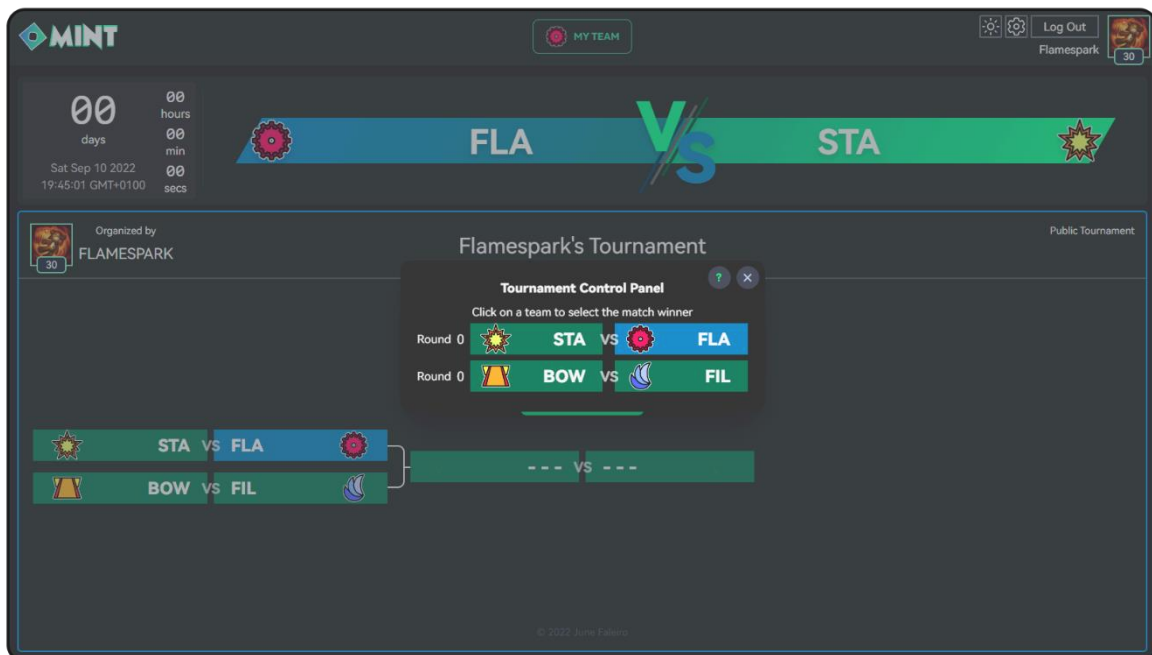
### 6.3.5.7 Main Page (Tournament In-progress) - ONGOING

The penultimate phase of the tournament, marked by the ONGOING state, is heavily based on the SEEDING state, with a key difference, the admin panel. This panel allows the tournament creator to select the teams that win their matches, while the application handles the obfuscated logic that deals with the subsequent display of the tournament's outcome.



[102] Tournament start

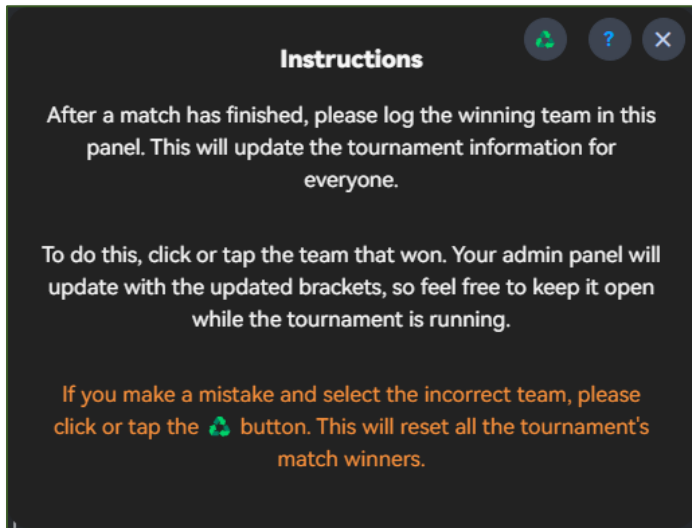
This admin panel is conditionally rendered to the tournament creator only, not even members in their team. Upon clicking the “OPEN ADMIN PANEL” button, a modal is shown to the tournament creator:



[103] Tournament creator admin/control panel

Before speaking further on the admin panel, a quality-of-life feature present only during the ONGOING state must be highlighted – The VS component. This component is located to the right of the countdown timer, and shows a user's team's current opponent during the tournament. For example, in the screenshot above it shows the user's team, FLA's current opponent is STA. While not an essential feature, it is a fully functioning component that makes it easier to see who the user is currently up against, as with higher team counts it can be confusing looking only at the tournament brackets.

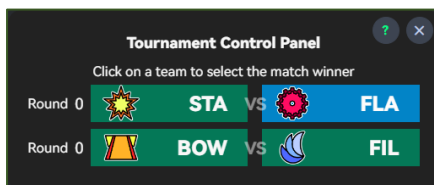
Exploring the admin panel further though, it serves one main function of controlling the tournament's outcome. In order to help a first-time user, a tutorial is included, which can be accessed by clicking the "?" button next to the close button:



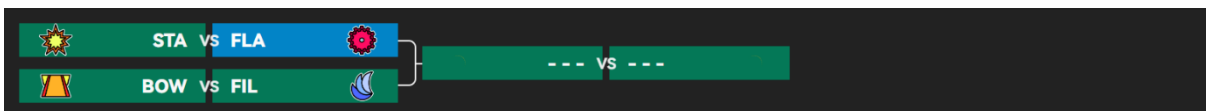
[105] Admin panel tutorial/instructions

In order to illustrate how the admin panel functions, the following set of screenshots will document the process of teams winning matches until the tournament is won.

### Initial State

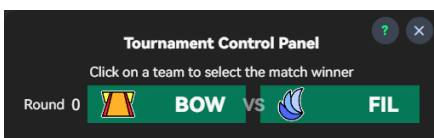


[106] Admin panel initial state

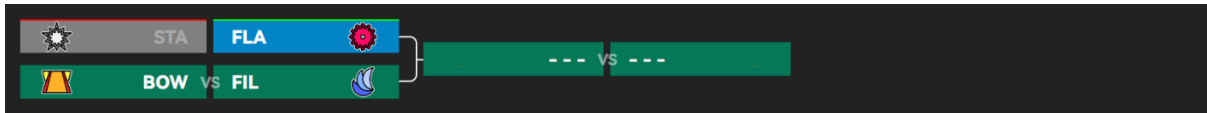


[107] Tournament display initial state

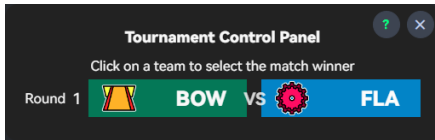
### FLA wins their match



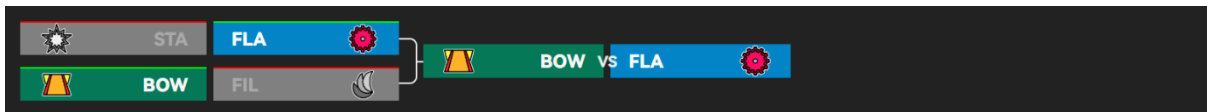
[108] Admin panel state



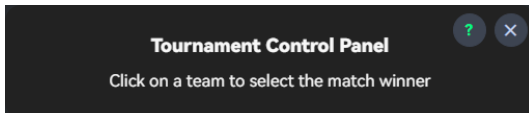
[109] Tournament display state

BOW wins their match

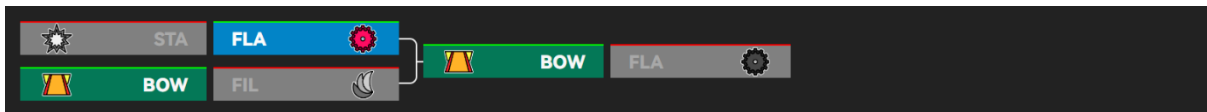
[110] Admin panel state



[111] Tournament display state

BOW wins the final match

[112] Admin panel final state



[113] Tournament display final state

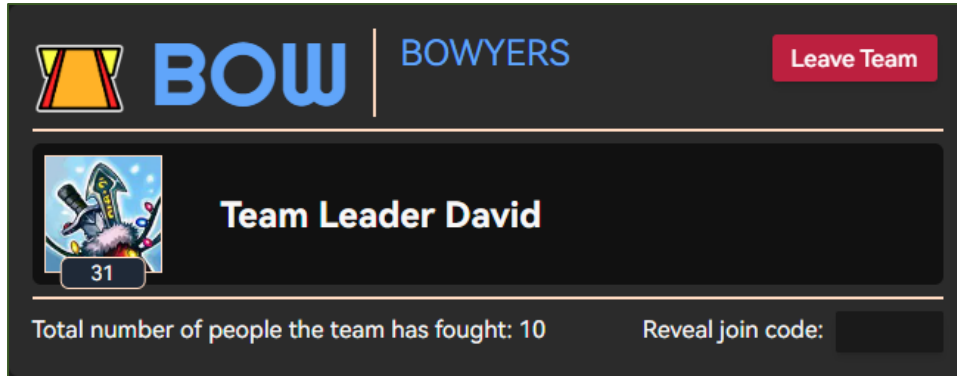
The last image is the final state of the tournament, BOW has won both their matches and with that, the tournament, changing the state of the tournament to ENDED.

Although the state change has been handled, due to time constraints for this project, the code required to safely delete the tournament and update both team and user hash-map data in the Upstash data store as well as Local Storage and the useUser hook has not yet been completed. This is definitely an objective to meet as early as possible in future work, as ending a tournament and allowing teams to join another is a necessary feature. Unfortunately, due to the complexity of the task, this has not been completed in-time.

That being said, the state change code will not pose an issue due to it's modular design, only the API accessing and subsequent testing.

Data that does update during a tournament however, is the team's total number of people they have fought counter rising. For example, in the tournament shown above, David's team BOW fought two

different teams. Each team has 5 members, incrementing David's team's count to 10:



[114] Team information display showing incrementation of # people fought

## 6.4 TOURNAMENT LOGIC

Although the process of a tournament has been documented in the *6.4.5 Tournament Pages* section, a lower-level view at some of the processes is what this section documents.

### 6.4.1 MatchTidbits

The TournamentDisplay component, which drives the ONGOING state stage of the tournament, is 1024 lines long. This is principally due to the conditional rendering of the tournament components. For example, the following is the code to render a tournament with a team size of 4:

```

output = (
  <div>
    {tournament.organized_by_ign == ign && adminButton}
    {adminPanel}

    <div className="grid grid-cols-3 p-5">
      <div className="mr-4 flex flex-col justify-between">
        <div className="relative mb-2 after:absolute after:-right-6 after:top-1/2 after:h-full after:w-6
after:rounded-tr-[9.6px] after:border-r-[2px] after:border-t-[2px] after:border-black-600 dark:before:border-white-300
dark:after:border-white-300">
          {team != null && (
            <MatchTidbit
              winner={getMatchWinner(getTag(0, 0, 0), getTag(0, 0, 1), 0)}
              team_1_tag={getTag(0, 0, 0)}
              team_1_icon_index={getIconIndex(0, 0, 0)}
              team_2_tag={getTag(0, 0, 1)}
              team_2_icon_index={getIconIndex(0, 0, 1)}
            ></MatchTidbit>
          )}
        </div>
        <div className="relative mt-2 after:absolute after:-right-6 after:bottom-1/2 after:h-full after:w-6
after:rounded-br-[9.6px] after:border-r-[2px] after:border-b-[2px] after:border-black-600 dark:before:border-white-300
dark:after:border-white-300">
          {team != null && (
            <MatchTidbit
              winner={getMatchWinner(getTag(0, 1, 0), getTag(0, 1, 1), 0)}
              team_1_tag={getTag(0, 1, 0)}
              team_1_icon_index={getIconIndex(0, 1, 0)}
              team_2_tag={getTag(0, 1, 1)}
              team_2_icon_index={getIconIndex(0, 1, 1)}
            ></MatchTidbit>
          )}
        </div>
        <span className="bg-white-200"></span>
      </div>
      <div className="mx-4 flex flex-col justify-center">
        <div className="relative before:absolute before:-left-2 before:top-1/2 before:h-0 before:w-2 before:border-b-
[2px] before:border-black-600 dark:before:border-white-300 dark:after:border-white-300 ">
          {team != null && (
            <MatchTidbit
              winner={getMatchWinner(getTag(1, 0, 0), getTag(1, 0, 1), 1)}
              team_1_tag={getTag(1, 0, 0)}
              team_1_icon_index={getIconIndex(1, 0, 0)}
              team_2_tag={getTag(1, 0, 1)}
              team_2_icon_index={getIconIndex(1, 0, 1)}
            ></MatchTidbit>
          )}
        </div>
      </div>
    </div>
  )
  ...

```

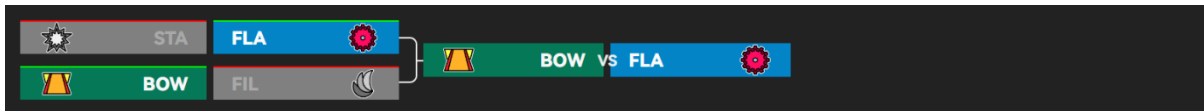
*[115] MatchTidbit component examples*

The important sections of this code are the `<MatchTidbit/>` components. Each `MatchTidbit` is passed parameters for a team's tag and icon, as well as a winner parameter, that is either null, or the tag of a team that has won the match. To keep the logic static and separated, three-dimensional functional matrices are used, namely `getTag()` and `getIconIndex()`. Due to the structure of the tournament object in JSON, teams are always ordered in a specific manner once the tournament is seeded. This way, their 'positions' in a tournament will always remain the same across all clients participating.

By leveraging this fact, the `getTag` and `getIconIndex` functions collect the dynamic data they require to display the correct team information from universally static data sources in the tournament data, retrieved from the Upstash data store. The matrix's syntax follows a distinct pattern:

- The first dimension is the round index. This number begins at 0 for round 1, and increments each time a new round is generated.
- The second dimension is the match index. Each round has a distinct number of matches based on the tournament type, with each bracket containing 2 matches.
- The third dimension is the team index. This number alternates between 0 and 1. Index 0 refers to the left-hand side team, while index 1 refers to the team on the right hand side.

As an example, in the following tournament setup, the `MatchTidbit` on the right, denoting the final match would use the following matrix for team FLA: (1,0,1). This can be verified in the code supplied above. The bottom-most `MatchTidbit`'s `team_2` data uses the matrix (1,0,1).



[116] *MatchTidbit tournament display matrix example*

Unsurprisingly, the code for these matrices and the `MatchTidbits` gets slightly more intense for higher team roster numbers. For a tournament that has 16 teams for example, the rendering code spans 224 lines.

#### 6.4.2 16-team Tournaments

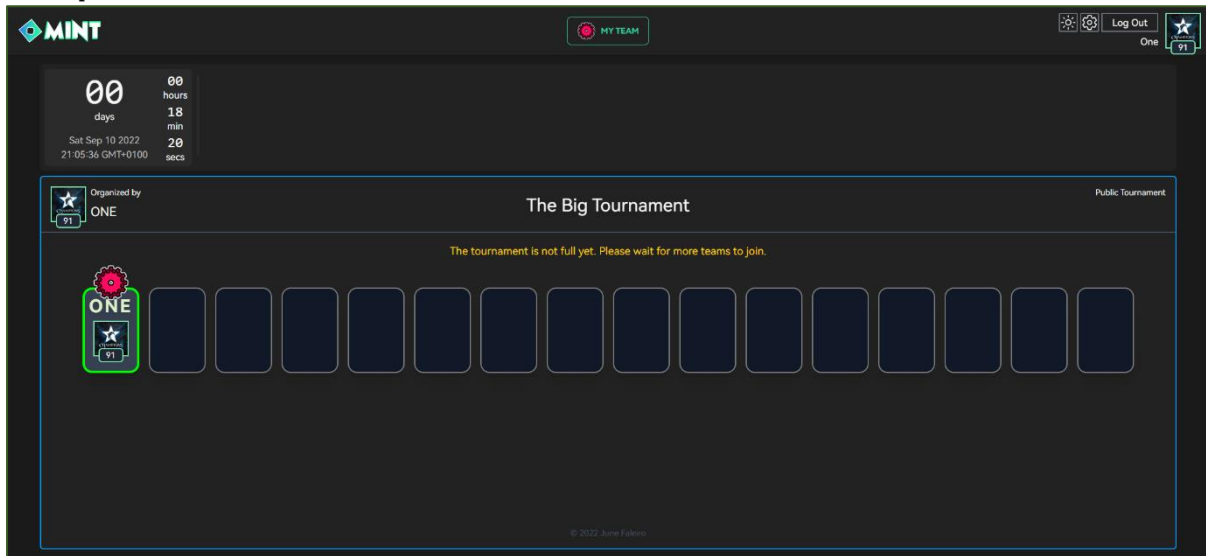
A 16-team tournament looks more impressive than a 4-team tournament, but setting one up takes time and commitment from all teams involved. To put numbers into perspective, a 16-team tournament consists of 80 unique player participants. This tournament type has 4 distinct rounds, and 15 total matches. To simplify the process of account creation and team creation to provide this example, account and team names are named after numbers.

An addition to improve a user's experience is functional arrows due to the tournament display being quite wide. These arrows alternate between the right-hand side of a tournament, and the left. If a user has a wide enough screen however, this is not an issue, and they will be able to view the entire tournament at once.

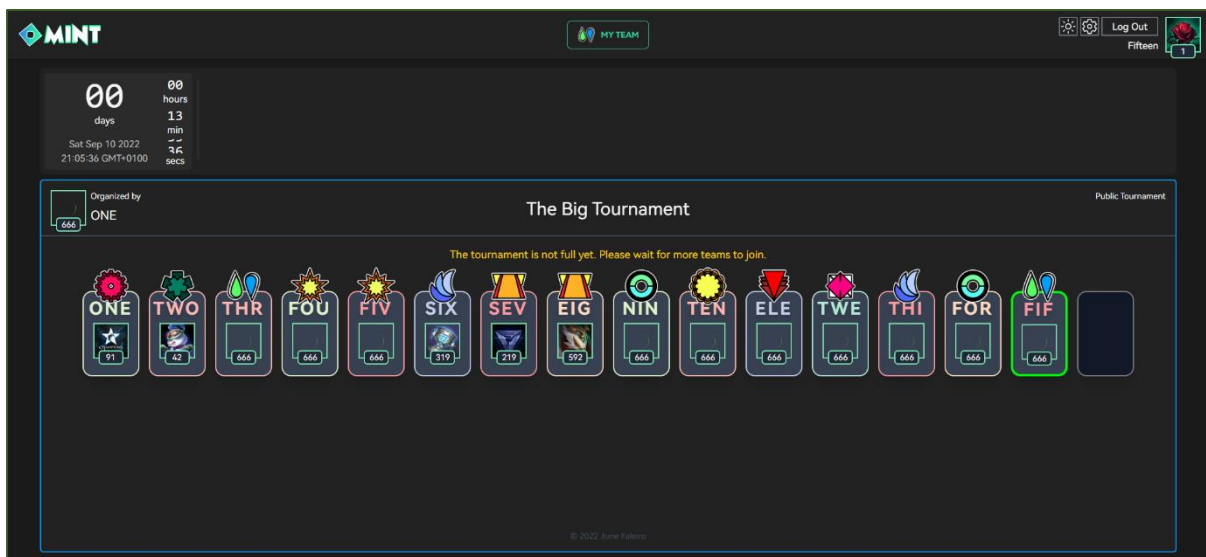
To showcase the process of putting together and running a 16-team tournament, each step will be shown below.

## Tournament Formation

The waiting phase of the tournament is identical to a 4-team tournament, but naturally there are more team spots that must be filled:



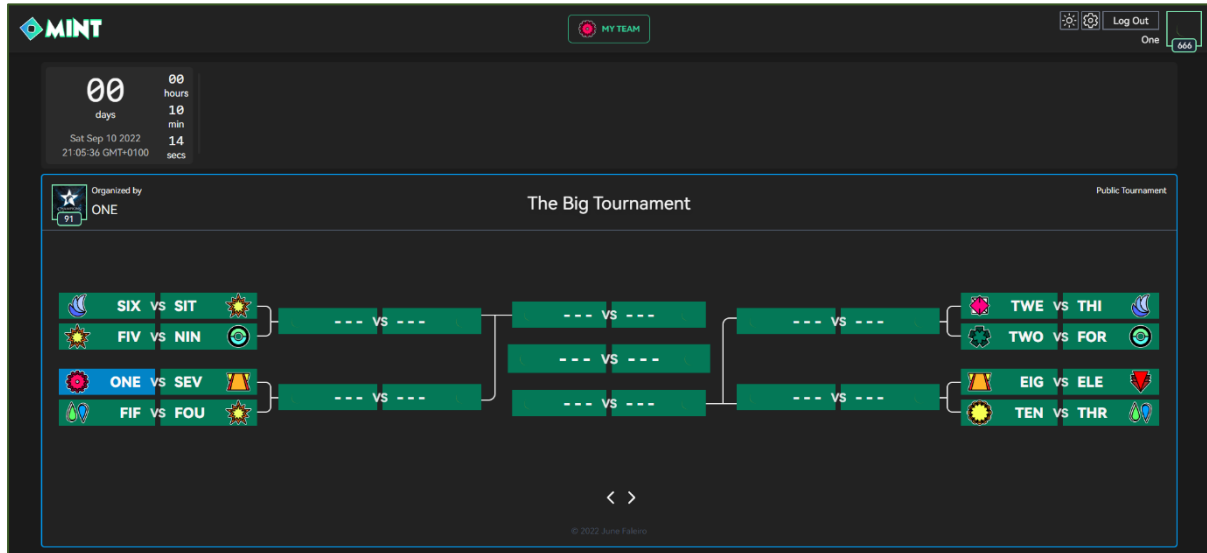
[117] Initial 16-team tournament formation



[118] 16-team tournament 15 team formation

### Seeding Phase

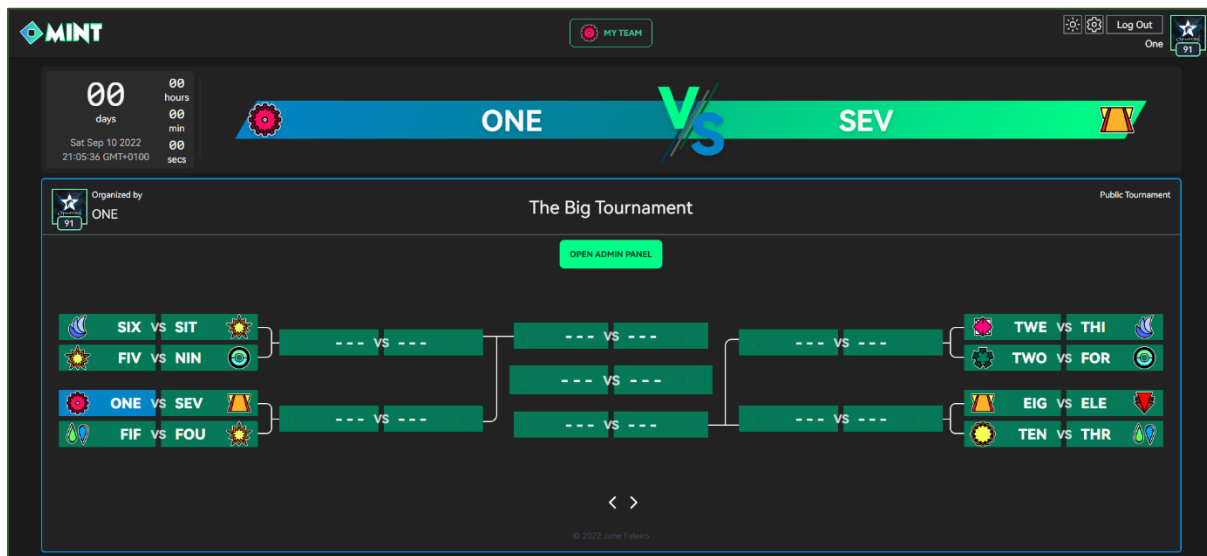
Once the tournament has all 16 spots filled by teams and the tournament creator (One) logs into the application, the tournament is seeded. All participants are then shown the following view with randomly created brackets and matches. Unfortunately, due to traffic throttling, some summoner icons don't load in these screenshots. For demonstration, many accounts are being made in a short amount of time, and the tournament participants are being rendered many times. In a normal setting, this would likely not be a problem due to traffic being spread out over a longer amount of time.



[119] 16-team tournament seeded brackets

### Tournament Start

Once the tournament starts, the creator is given the admin panel to control the outcome of each match:

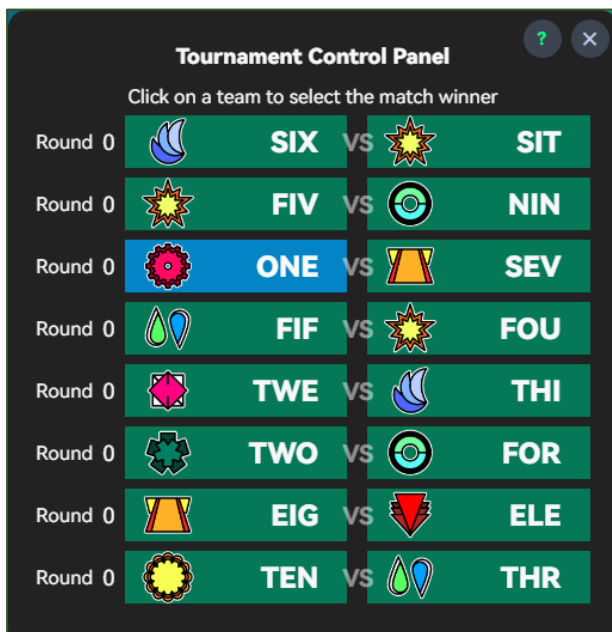


[120] 16-team tournament creator view, tournament start

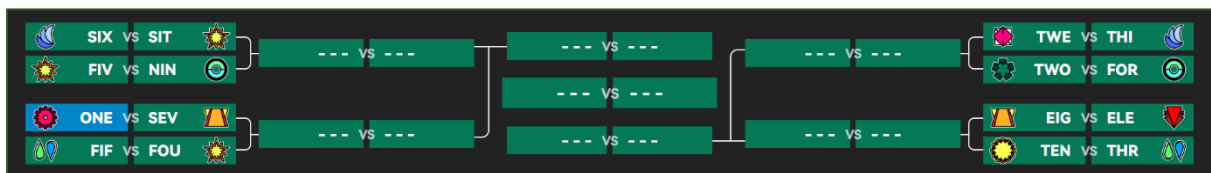


### Round 1 Start

The admin panel for round 1 looks as follows, with each match and its' teams listed in index order according to the matrix used by the MatchTidbits:



[121] Admin panel initial state

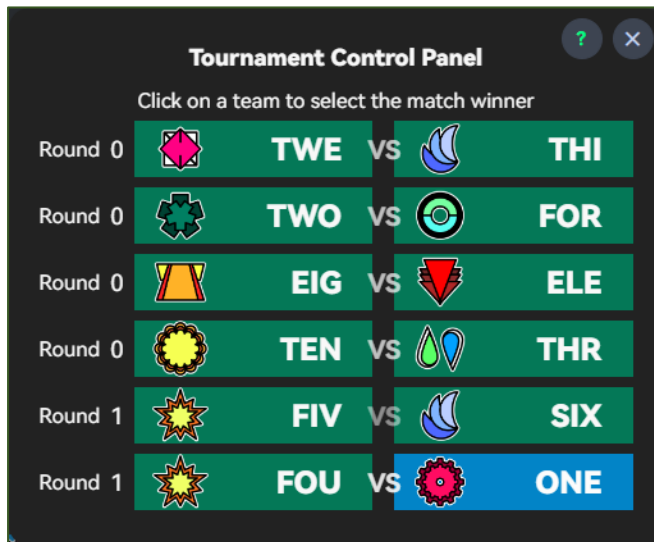


[122] Tournament display initial state

The beauty of this asynchronous way of handling the running of the tournament is that matches can finish whenever, and the algorithm waits for them. For example, the entire right-hand side of the tournament can reach the final before the left-hand side has even started, without any issues. For this example however, each round will be completed before the next.

### Round 1 Half-Way

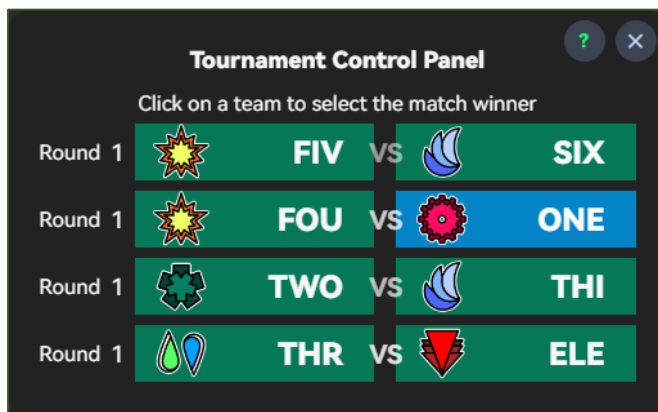
Due to the way that rounds and matches have been designed to be indexed, the admin panel displays information in the way that makes the most sense. For instance, half-way through round 1, there are two matches from the second round that are taking place. However, rather than them being sorted in a FIFO structure, they are inserted at the end of the match list. As it is more likely that round 1 matches will finish before round 2, this gives the tournament creator a better view of the structure of the tournament.



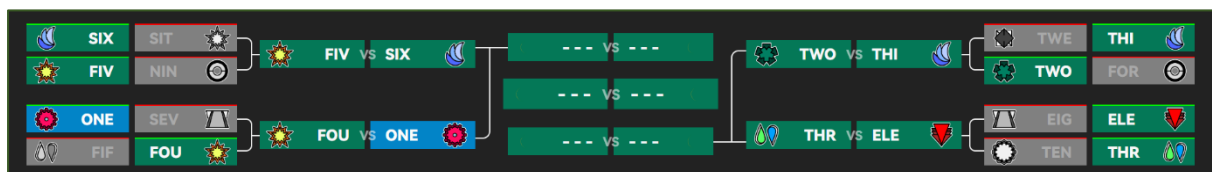
[123] Admin panel state

### Round 1 Outcome / Round 2 Start

At the beginning of round 2, there are now half the number of matches as round 1, as half the teams have been eliminated. Once again, these matches are sorted according to the matrix structure, so that the tournament creator doesn't become confused.

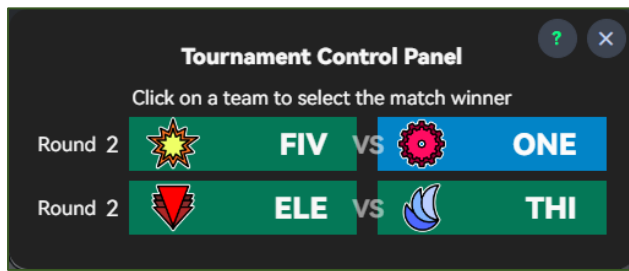


[124] Admin panel state



[125] Tournament display state

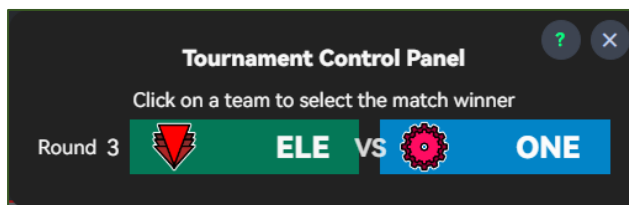
As in the 4-team tournament, match winners are highlighted green, match losers are highlighted red and greyed out using a CSS filter, and the user's team is highlighted blue.

Round 2 Outcome / Round 3 Start

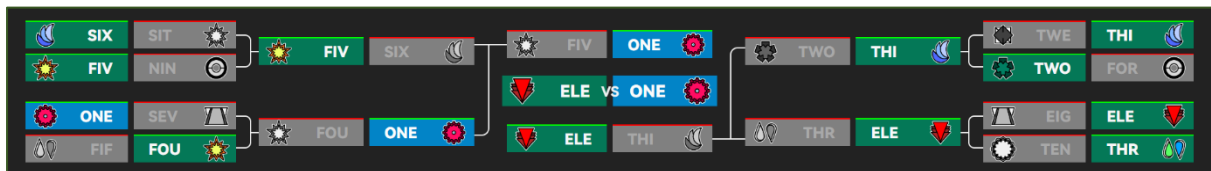
[126] Admin panel state



[127] Tournament display state

Round 3 Outcome / Final Round Start

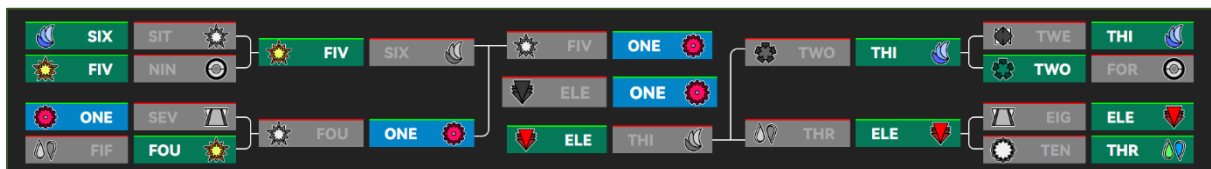
[128] Admin panel state



[129] Tournament display state

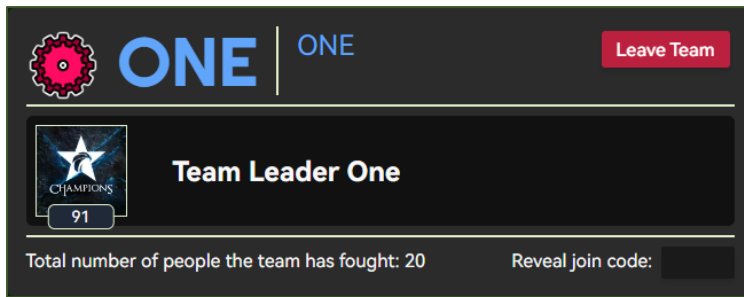
Final Outcome

As discussed in section 6.4.5.7, the post-tournament implementation is not finished due to time constraints. However, the outcome of the tournament is still visible, showing team ONE as the winner:



[130] Final tournament display state

Team ONE's number of people fought counter has also gone up to reflect the number of people their team played against:



[131] Team information display # people fought for 16-team tournament

## 6.5 IMPLEMENTATION CONCLUSION

Throughout the implementation of Tournamint there were many challenges to overcome in order to meet the design specification, aims and objectives. While the final outcome is rich in distinct features and interconnected logic spanning many React components, coupled with 2 different API sources, a custom quasi-database design and conditional logic that makes use of the SPA-features of Next, while fully utilizing all the 3<sup>rd</sup> party libraries and packages that this project requires, there are some sections of the final implementation that are lacking or unfinished. These will be discussed in section 9.2 *Future Work*.

## 7 APPLICATION TESTING

Application testing encompasses multiple testing types and methodologies. In order to evaluate this project's success, different types of tests must be performed.

First, tests against the project's requirements from section 3.2 will be undertaken. These tests do not use unit testing methods, but rather rely on observation of function and completion.

To evaluate the success of each requirement test, a colour-based key will be used:

- - Test has passed, all of the description's requirements have been met
- ★ - Parts of the test passed, or passed to some degree, while other parts may not
- - Test has failed

In order for a requirement test to be considered a success, its' outcome must be a green circle ●.

### 7.1 FUNCTIONAL REQUIREMENT TESTS

Requirement	Description	Result	ID
Register Account	User can register account using their League of Legends summoner name and a 6-digit passcode	●	01
Log In to Account	User can log into their account using League of Legends summoner name and previously set 6-digit passcode	●	02
Log Out of Account	Once logged in, user can log out of their account by clicking the log out button	●	03
Assist user in creating/joining a team	When not in a team, user should be guided by the application's interface into creating/joining a team	●	04
Assist user in creating/joining a tournament	When not the owner or participant of a tournament, user should be guided by the application into creating/joining a tournament	●	05
Display LoL account data	User's league of legends account data (level, summoner icon, ranked information) should be displayed for them in the navbar and profile page	●	06
Editable user information	On the profile page, user can change their personal username, biography, and favourite champion	●	07
Persistent user information	On the profile page, user's personal information should persist between browser sessions and across different browsers if they log in in a new location	●	08
Favourite champion splash art change	On the profile page, changing their favourite champion should change the user's profile background to said champion's splash art	●	09
Tournamint statistics shown	On the profile page, users' tournamint statistics should be displayed	★	10
Theme Swappable	User should be able to toggle between light and dark mode with a single button press	●	11
Changeable settings	On the settings page, user should be able to change settings by clicking toggle switches	●	12
Persistent QOL data	Changing light/dark mode or settings should persist between browser sessions but not across different browsers	●	13
Create team	User can create a team with customizable name, tag, icon, and colour	●	14
Join team	User can join a team using a team's tag and invite code	●	15
Leave team	User can leave a team by clicking a button on the teams page	●	16

Create tournament	User can create a tournament with customizable name, size, start date/time, privacy, ID code and join code	●	17
Join private tournament	User can join a private tournament using ID code and join code	●	18
Join public tournament	User can join a public tournament from a list of public, non-full tournaments	●	19
See time until tournament start	Tournament members can see how much time is left until their tournament begins	●	20
Seed tournament	Tournament creator can seed the brackets of a tournament once it's full and 1hr or less before tournament start	●	21
Brackets visible (seeding)	Tournament members can see bracket seeds once the creator has seeded the brackets, and it's an hour or less before the tournament begins	●	22
Brackets visible (running)	Tournament members can see the current state of brackets once the tournament is underway	●	23
Update tournament state	Tournament creator can select which team wins a match, which updates this information for all tournament members	●	24
See tournament state	Tournament members can see match winners/losers	●	25
See next opponent	Tournament members can see which team they will be fighting next	●	26
Team information visible	User's team icon, summoner icon, level and username should be displayed in the navbar	●	27
Tournament end statistics update	Once a tournament final has ended, team and user statistics should be updated	★	28
Tournament deletion	Once a tournament final has ended and its' outcome recorded in persistent data storage, it should be removed from the application.	■	29

## 7.2 NON-FUNCTIONAL REQUIREMENT TESTS

Non-functional requirements exist in a broader sense than functional ones. They describe and are used to evaluate the overall operation of this project and other external requirements (e.g., legal). The following table lists these non-functional requirements:

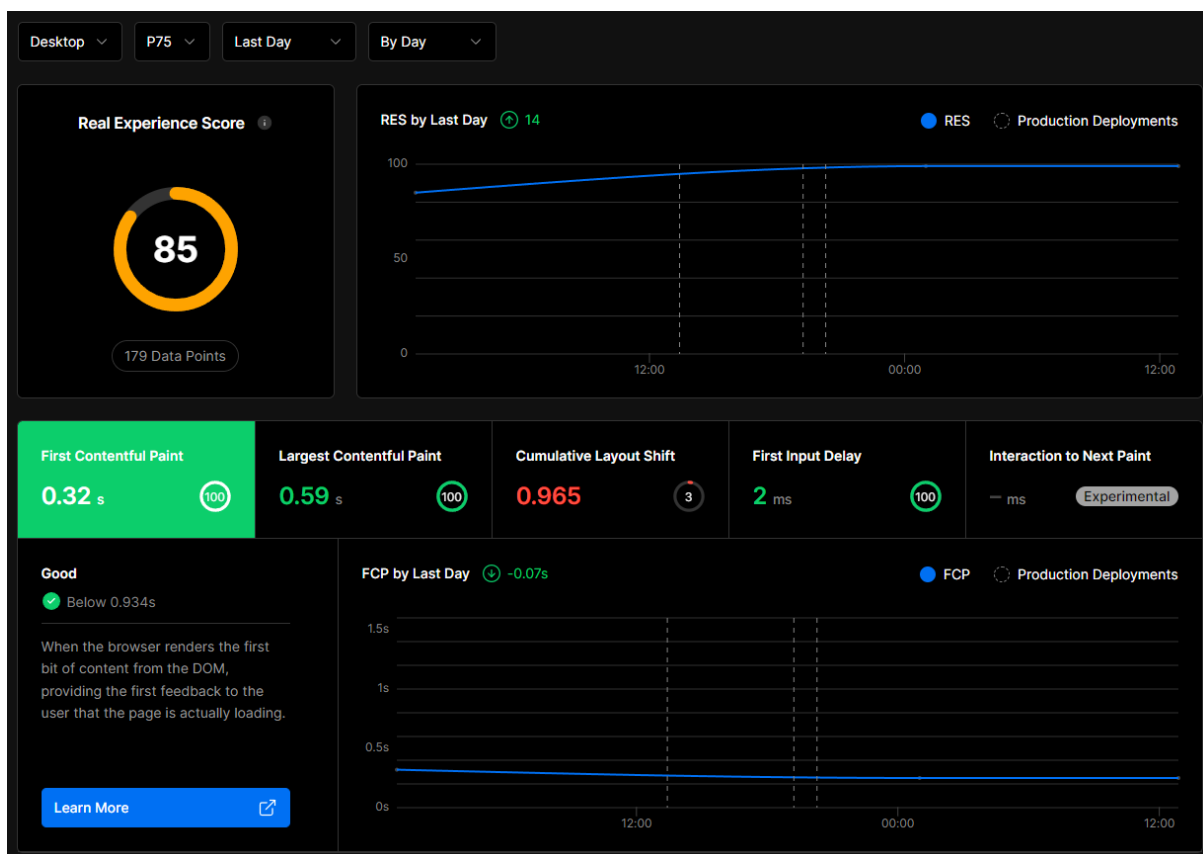
Requirement	Description	Result	ID
Deploy web app	Web application can be deployed using Vercel in a short amount of time	●	01
Load web app	Web application loads successfully in Google Chrome & Firefox	●	02
Loading speed adequate	Lighthouse score which encompasses First Contentful Paint, Largest Contentful Paint and Cumulative Layout Shift should be above 85.	●	03
Adequate deployment accessibility	Uptime of a deployment of Tournamint should be high with minimal downtime. Updating the deployment to a new version of Tournamint should also have minimal/no downtime.	●	04
Legal requirement	The web application should comply with GDPR guidelines and legislation.	★	05

### 7.2.1 Lighthouse Score

On the modern web, there are many factors that influence how successfully a web application will perform. A system was devised to quantify the ‘quality’ of a website called Lighthouse. Any website that is crawled by a search engine uses this system, or a variation on it. These websites are automatically “evaluated with a score from 5 categories: Performance, Accessibility, Best Practices, SEO and PWA” and then “given a score between 0 – 100.”<sup>[23]</sup>

In order to achieve the best user experience as well as rank higher in SEO results, a Lighthouse score should be maximised as much as possible. Thankfully, Vercel, the deployment platform used by this project offers Lighthouse score calculation for free over a 24hr time period.

After implementation and generating 179 data points for the score to be based off, the results are as follows:



[132] Vercel lighthouse score audit

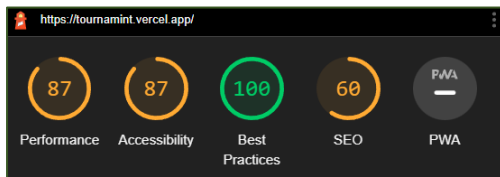
In non-functional requirement test 03, it was stated that for the test to pass, the total score should be above 85. As seen in the screenshot the overall score is exactly 85 (although this number fluctuates with different devices using the site). Upon further examination however, it’s clear that the score is being dragged down by one factor in particular, the CLS (Cumulative Layout Shift).

Cumulative layout shift is described as being a measure of ‘visual stability’. It ‘helps quantify how often users experience unexpected layout shifts<sup>[24]</sup>. This, then at first seems like an important issue that this project has failed to address. Cumulative layout shift can be very damaging in certain situations. Web.dev<sup>[24]</sup> gives an example of a complete/cancel order set of buttons being moved when a user does not expect them to, leading them to click the wrong button.

What has likely happened with Tournamint however, is that due to it’s conditional rendering adding and removing many elements based on state, which the lighthouse algorithm has picked up on, the score has been dramatically lowered.

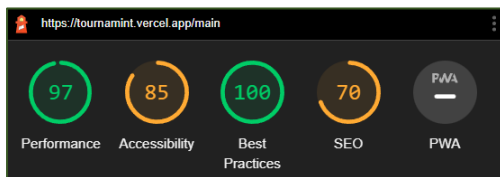
This theory can be supported by manual testing also. Microsoft's Edge browser devtools offers developers the ability to perform a lighthouse score audit within their browser. The results for each page are as follows:

### Landing Page



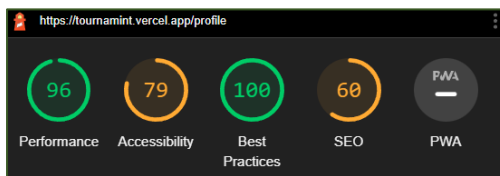
[133] Microsoft Edge lighthouse score audit for landing page

### Main Page (Tournament in progress)



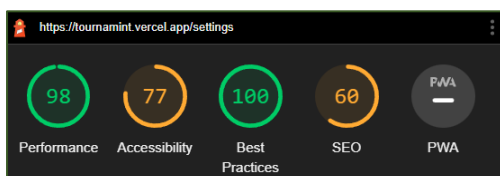
[134] Microsoft Edge lighthouse score audit for main page

### Profile Page



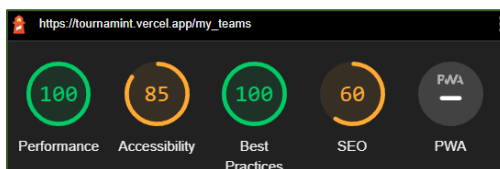
[135] Microsoft Edge lighthouse score audit for profile page

### Settings Page



[136] Microsoft Edge lighthouse score audit for settings page

### Team Page (User is in a team)



[137] Microsoft Edge lighthouse score audit for team page

While performance is overall very high (avg. score 95.6), and best practices are at 100 score across the board, accessibility has some shortcomings. Edge's devtools lighthouse generates a report for each page which contains detailed information on the greatest offenders in each tested section. In terms of accessibility, there are many small instances of shortcomings on the part of the developer, such as certain image elements missing alt attributes, some links not having 'discernible names' and insufficient contrast ratios between some fore- and background elements.



This Microsoft Edge lighthouse report does not calculate cumulative layout shift, so specific measures are not able to be taken. To address this issue, future work in diagnosing the sources of CLS issues and determining whether they require fixing should be undertaken. It may be the case that many causes of the CLS-flagged behaviour is intended, and subsequent alterations to the source code should only address those issues which do impact the user's experience.

However, overall the lighthouse scores across both Vercel and Microsoft Edge's offerings are high, especially in the performance category. For this reason the requirement test has been deemed a success.

### 7.2.2 Deployment stability

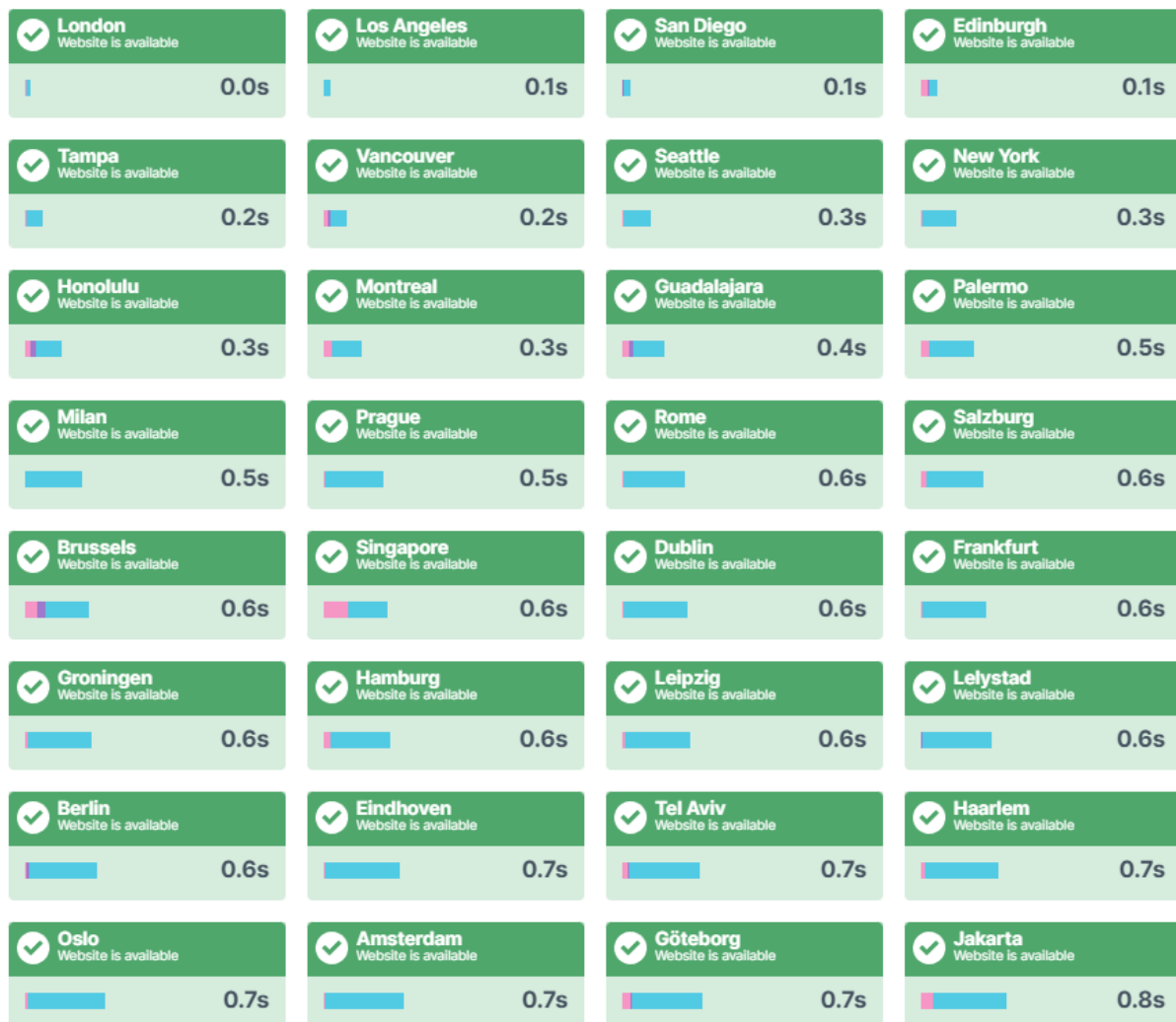
An important factor to consider when recommending or requiring a certain platform for the deployment of an application, is its' stability. Stability of a platform in this case refers to the length of uninterrupted uptime of the web application, it's global availability and initial response times, as well as its' performance when running the web application.

As determined in 7.1.1 *Lighthouse Score*, the performance of the application is more than satisfactory, with an average lighthouse score of 100 for the Vercel system, and 95.6 for the Edge system.

After running multiple versions of a Tournamint deployment using Vercel over the course of 2 months, not a single instance of site downtime was reported.

In order to monitor the application's downtime once it has been deployed to a live user base, a 3<sup>rd</sup> party uptime-monitoring implementation can be used. As many directly integrate with a site without needing to access the site's source code or private information, this can be left to the discretion of each distribution administrator.

In terms of global availability, many 3<sup>rd</sup> party pinging tools are also available. For instance, Uptrends reports the following data for a ping of the project's current Tournamint distribution:



[138] Global pings of Vercel Tournamint deployment

Due to the volume of results, some have been omitted. The longest time taken to ping the application was in Tokyo, with a 1.5s response time, and Beijing, which was unable to access the site, likely due to China's policy on internet freedom.

In conclusion, using Vercel leads to a very stable deployment. Further, larger scale testing with many more entry points is useful, but that cannot be performed at this time.

For these reasons, non-functional requirements test 04 has been marked as successful.

## **8 STATEMENT OF ETHICS**

Throughout the project, legal and ethical considerations have been made. This section will explore the ways in which these aspects of the application have been considered and catered for during development. Social and professional issues are not pertinent to this application, and have therefore not been documented. A SAGE form has also been completed in compliance with University of Surrey requirements, which is appended to the end of this section.

### **8.1 LSEP CONSIDERATIONS**

#### 8.1.1 Legal Issues

This project was developed within the UK, but is currently able to serve users in the entirety of western Europe. Therefore, this project must comply with UK and European law. Due to Brexit, the UK is no longer a part of the European union. However, the GDPR (General Data Protection Regulation) act, which became law in 2018 before Brexit, is still part of UK law under the Data Protection Act 2018.<sup>[25]</sup>

##### *8.1.1.2 Data Protection Act*

The data protection act 2018, which incorporates the legislation introduced by the EU's GDPR regulation, building on the UK's previous 1998 data protection act, is in effect as of the writing of this report and final implementation of Tournamint.

Data protection act legislation requires stored data to be subject to certain conditions involving legality, privacy, security and accountability/governance. These must be met by all "organizations" that process user data. Although in practice for this specific case consultation from legal council would be required, an open-source project such as this, which processes no identifiable user information at any point, is not classed as an organization, and would therefore not be subject to compliance with the data protection act, due to a lack of relevant data to protect. This does not mean, however, that special care hasn't been taken in the design and philosophy of the application in relation to a user's data. More on this topic can be found in *8.1.2.2 Data Confidentiality & Ownership of Personal Data*.

#### 8.1.2 Ethical Issues

Ethical issues in computer science and software development impact the everyday life of billions of people across the planet. In today's society, ethical issues surrounding data are becoming ever more pertinent, as large corporations work tirelessly to retain users and extract as much data as possible. Unethical application design is rampant, and unfortunately the societal systems that people have created and fostered in the previous millennia have impacted the way that software is developed.

This project therefore has throughout its' entire conception, design and development paid close attention to how it handles user data, and to its' design principles. George Lawton describes in a paper some examples of unethical design choices in software, such as 'addictive design', 'corporate ownership of personal data' and 'algorithmic bias'<sup>[26]</sup>.

##### *8.1.2.1 Addictive Design & Algorithmic Bias*

At the start of the ongoing 2020 Covid-19 pandemic, a certain app quickly rose to massive success – TikTok. This short-form video sharing application dominated app store charts in a time when many were stuck inside during lockdowns, needing entertainment or distraction.

In an academic paper published in 2021<sup>[27]</sup>, Ciarán O'Connor explains that far-right extremism runs rampant on TikTok, leveraging its' elusive algorithm which decides what video a user will see. By "making use of hashtags related to general political discussions and trends on TikTok, using hashtags like #Conservative, #Politics, #BLM [Black Lives Matter], #Republican and #Trump in large numbers", uploaders make their videos more likely to be recommended to a user who often consumes similar content, and who make like and share these videos, strengthening the algorithmic bias towards this type of content.

TikTok's algorithm is so good at its' job of retaining users by any means necessary, including algorithmic bias that can serve hate speech to it's audience of mostly impressionable teenagers in fact, that it is, as of 2022, the 6<sup>th</sup> most used app in the world, sporting over 1billion active monthly users<sup>[28]</sup>.

It was important, then, for Tournamint to avoid the principles which drive TikTok's business strategy by creating an application that is enjoyable to use, but not addictive, and that doesn't feature any algorithmic bias.

As of the final implementation of the algorithm, it is safe to say that Tournamint meets both these criteria in their fullest. Although Tournamint's landing page features a design principle intended to capture a user's attention, this is not present throughout the application. Tournamint exists only to serve a purpose: Organizing and joining tournaments with your friends.

The second criteria, algorithmic bias, is also not present in Tournamint. The only algorithm that affects a user is the random initial seeding of teams in a tournament. No personal data is collected except for publicly available data from League of Legends, which cannot be used to identify a user, making a targeted algorithm that acts upon a user's personal data impossible.

#### *8.1.2.2 Data Confidentiality & Ownership of Personal Data*

During design and development, the confidentiality of user's data has been cause for concern. Due to the open-source, freely deployable nature of the application, user's data is always in danger of either being misused or mishandled, whether out of inaction or malice.

The approach to tackling this issue lies in the fundamental design of Tournamint's data systems themselves. The application primarily utilizes existing public data from Riot Games' servers as the basis for account information, such as profile pictures, account names and skill information.

The only 'sensitive' information a user enters is their account passcode, which is stored in plaintext. Although a hashing algorithm can be implemented in future for storing this passcode, it may do more harm than good; An independent user creating a Tournamint distribution could modify the hashing code to obtain user passcodes regardless of initial security considerations. Therefore, making the user fully aware when entering the passcode that it is stored 'unsafely' in plaintext can make the application safer, as they are far less likely to use a passcode that they have used before.

This transparent approach to handling user data extends to the entirety of Tournamint's philosophy – the application collects no user data to sell to advertisers, and isn't made to generate profit. Considerations have been specifically made in choice of technology (Vercel,Upstash) as well as target users (small groups/collectives of people who wish to have a competitive tournament environment without a business generating profit off them) in order to keep the application free to host and maintain. This way a user is under no false impressions when using the application.

Tournamint's data confidentiality considerations therefore fall under a simple mantra – 'Instead of treating the problem, remove the problem in the first place'. There will be no incentive to steal a user's account information, as no data worth stealing is stored in it.

#### 8.1.3 SAGE

The following attached SAGE form, which serves as a list of ethical checks that must be performed for a project, has determined that all checks pass, signifying clearance of the project.



**SAGE Form**  
**Tournamint June Fa**

## 9 PROJECT EVALUATION & CONCLUSION

This section evaluates the overall outcome of the project, contrasting it to the initial objectives, implementation and test results. Tournamint is an open-source web application, meaning that its' potential development does not end with this report. Objectives that were not met, or met to a poor standard, as well as future features that would improve the application will be discussed in *9.2 Future Work*.

### 9.1 EVALUATION

As shown in section 7.1 and 7.2, the outcome of almost all requirements tests first derived in 3.2 *Requirements* were a resounding success, with every required feature being implemented in the final application, bar the algorithms that are executed when a tournament's final match ends.

In sections 1.2 and 1.3, the project's aims and objectives were initially created. Although the objectives were used to derive the project's requirements, which have been tested against, the broader aims of the project have not yet been compared to the final outcome. Therefore, the following table details each aim, and discusses whether this aim was met.

Aim	Discussion	Aim Met?
Investigate web solutions that allow for easy deployment of a web application.	In <i>2.5 Web Solutions</i> , Vercel was identified as the perfect, free web solution for hosting a Tournamint distribution, in part thanks to its' direct compatibility support with Upstash' Redis data store.	✓
Research web frameworks & technologies that will create a streamlined development process and polished end product.	In <i>2.3 Web Frameworks</i> , Next, React and Sapper were explored as potential solutions for this project. It was concluded that Next, built on React would be the perfect framework for the application due to a number of factors including its' interoperability and level of support with Vercel and by extension the Upstash data store. <i>2.4 Web Technologies</i> dove into the plethora of 3 <sup>rd</sup> party packages that would enable Tournamint to be developed as efficiently as possible to produce the best looking and performing result.	✓
Create a project workplan in order to identify the timescale and risk associated with the project.	A project workplan was devised in section 4, detailing a timescale for each section of this project.	✓
Develop a web application using the web framework(s), technologies and web solutions selected from research.	Tournamint was fully implemented and documented in section 6, lacking only one major feature that will be delegated to future work. All the proposed web technologies, solutions and framework were used in the final implementation, utilizing their feature set to its' fullest.	✓
Test the functionality of the web application against this project's objectives.	The web application was tested against both the project's functional and non-functional requirements, which are based off the project's objectives in section 1.3.	✓
If possible, collect user feedback on the web application	Unfortunately, due to factors outside the control of this project, user feedback on the final implementation of the application was not able to be collected. However, user feedback in the form of a dedicated feedback form integrated into the web application, as well as targeted forms requesting specific data from users can be devised in future	✗

Most of the project's broad aims have been fully met with the exception of collecting user feedback. Before detailing the future work that can be performed in section 9.2, a broad overview of areas of improvement will be shown.

### 9.1.1 Areas of improvement

The following numbered list of eight broad improvements to be made to Tournamint will be referenced in 9.2 *Future Work*.

1. Tournament ending functionality
2. Application front-end polish
3. Trophy system implementation
4. Simplifying deployment customizability
5. Including multiple LoL regions
6. Diagnosing and fixing CLS issues
7. Improving lighthouse score with targeted accessibility improvements
8. Collecting user feedback & implementing bug reporting

## 9.2 FUTURE WORK

Although the application implementation for this project is finished, there are many areas that would benefit from future improvement or additions. This section explores eight of these areas.

### 1 Tournament ending functionality

Originally a functional requirement, this objective involves a tournament's relevant data being stored before it is removed from a distribution of tournamint. After a tournament has ended, teams should be able to join a new tournament. However, as of the current final implementation of the application, this functionality has not yet been included due to time and complexity constraints.

This core feature of the application should be implemented as soon as possible in terms of future work. The winning team data from a tournament should be recorded by the application within the team's data as well as the user accounts that are part of the team at the time. The application should then remove references to the tournament from team and user account data in the Upstash data store, Local Storage and the useUser hook.

Once this action has been completed, teams will be able to create and join new tournaments.

### 2 Application front-end polish

Although the application functions well, with careful attention being paid to the design language of many elements, there is room for improvement. Further improvements to the colour scheme, unification of some design elements and user feedback improvements are all a good first step.

An immediate change that would be good to implement is more state indicators, that is visual elements that communicate back-end changes to a user. For example, the landing page sometimes requires time to set-up the initial connection with the Upstash back-end. While this is happening there is currently no visual indicator to tell the user that the program is processing information, unlike with the spinner SVGs for summoner icons and tournaments. For the landing page specifically, a spinner or loading bar shown at the top of the screen may be useful, though the merits of each system should be analysed before committing to one.

### 3 Trophy system implementation

This system would be a flavour addition to the tournament ending functionality. Once a tournament ends, the winning team should be awarded a trophy for said tournament. This trophy should have metadata such as the tournament name, date and size. These trophies should be stored both within the team's data, as well as the account data for all members of the team.

By storing data in the user accounts as well as team data, it can be displayed in team member's profiles, similar to the original design mock-ups in section 5.4.4. This flavour feature would enrich a user's experience with the application, giving them rewards to work towards. A further addition to this system

could be in the form of tournament creator customization – trophies could be customized with icons, colours and shapes that are unique to each tournament, so that they stick out on profile pages better and are more recognizable.

#### 4 Simplifying deployment customizability

Tournamint is open source under a permissive license, meaning that code modification is not only allowed, but encouraged. Users are free to create forks of the application and apply their own design choices and features.

Currently, unification of these designs is seen in the TailwindCSS theme customization file as well as some unified props across components. Future work in this area should examine what parts of an application would benefit from a unified set of changeable data. As an example, tournament brackets currently use hard-coded colours for MatchTidbits. Making this data easily editable a user wishing to create their own distribution would benefit them greatly.

Together with the new proposed customizability improvements should come documentation. Due to Tournamint's design and features being in an early stage in terms of code readability and comment documentation there is currently no formal documentation to accompany the application. Once Tournamint's features and design language have matured, documentation for these features and customizability improvements should be written and separately hosted, potentially using GitHub pages.

#### 5 Including multiple LoL regions

Tournamint currently caters to one region – Europe West. This was done early during development due to potential code breakage and League of Legends API constraints. Once Tournamint is deemed fully functional as a live application, more regions should be accommodated, with a switch passing variable data to the Axios API calls based on user selection of their region.

#### 6 Diagnosing and fixing CLS issues

In section 7.2.1 *Lighthouse Score*, the cumulative layout shift of the application was deemed to be critically negative. Although much of this score is due to the way in which Tournamint functions, it is important to diagnose which elements and associated JavaScript code is causing the low score.

Not only does a low score negatively impact the website's SEO, but it can indicate genuine problems with the application. For this reason, if a diagnosis deems that an element is negatively impacting a user, the issue should be addressed with code fixes or design changes.

Further research into CLS and SEO score should also be performed in order to ascertain whether the score can be improved in certain areas without resorting to sweeping design changes of the application.

#### 7 Improving lighthouse score with targeted accessibility improvements

CLS was not the only issue found during the non-functional requirement tests. Another shortfall in the application's implementation lies in its' accessibility score. This score is determined by a myriad of factors relating to accessibility requirements and best practices, such as alternate text for elements, page metadata and screen-reader compatibility checks.

Future work should work on these areas in order to improve the accessibility lighthouse score, and by extension, the user experience.

#### 8 Collecting user feedback & implementing bug reporting

A broad aim of this project was to collect user feedback on Tournamint's implementation. Due to a number of factors this has not been possible in the time constraint. Collecting user feedback can be done in multiple ways thanks to the medium of the web. The simplest way is to create a form using an application like Microsoft or Google Forms, which is a questionnaire style user response system.

However, a better solution may be to integrate feedback directly into the application. By creating a separate page within the application, specific user data in any form the developer wishes can be obtained.



A link to this form could be displayed at the end of a tournament for example, asking users how they would rate the experience and suggest improvements.

Helpfully, bug reporting can also be implemented in this fashion. When a user encounters a bug using Tournamint, the last thing they want to do is scour the GitHub repository looking for a bug-reporting form. Users who are developers themselves may use GitHub's issue system to raise a problem, or submit a pull request themselves addressing the bug, but more casual users will likely prefer an easily accessible bug reporting form inside the application itself. The link to this can even be placed on the navbar in the same button style as the theme swapping and settings buttons. This way the link is unobtrusive but always accessible, no matter where in the application a user is.

## 9.3 CONCLUSION

This project has concluded, but Tournamint still has an open path ahead of it. As detailed in *9.2 Future Work*, there are numerous improvements that can be undertaken to help Tournamint reach its' full potential.

### 9.3.1 Academic contributions

In order to develop Tournamint to the standard that it has been, a wide breadth of knowledge surrounding web development was required. During the application developer's time at the University of Surrey, there have been multiple modules which have spurred an interest and understanding of application development.

#### Web and Database Systems COM1025

As first module at the University which taught web development concepts, COM1025 was instrumental in fostering a newfound appreciation for the breadth and intricacy of the web. It was taught using Ruby on Rails, an unfamiliar framework using an unfamiliar programming language, Ruby. This fresh perspective into developing the web helped inspire later research and work into backend technologies.

#### Parallel Computing COM2039

Year 2's Parallel Computing module introduced many new ways of thinking about how to process data. Parallelization was a foreign concept at the time, but the way of thinking about algorithms in a parallelized fashion, rather than serial, helped tremendously during this project when programming with asynchronous function calls and data handling, especially when dealing with data from API requests.

#### Project Management MAN3104

Although Project Management was not a computer science module, its' abstraction of the core concepts of the process of completing a project was invaluable for the planning and execution of this project. Implementing a realistic project development lifecycle plan and effectively managing time, scope and expectation was helped in no small part by this module.

#### Advanced Challenges in Web Technologies COM3014

The final web development module undertaken at the university was COM3014. This module allowed the developer to showcase their talents and hone their front-end skills that they would subsequently use to develop this project. As a group module, it heavily relied on teamwork and using Git with effective use of branches, pull requests and merging. Although the latter two have not been used in this project, using Git, and by extension GitHub has been made significantly easier by this module, as well as using concepts and technologies for the front-end of Tournamint that were used in Com3014.

### 9.3.2 Final Word

Tournamint was, and is a project developed to accommodate users who wish to partake in tournaments with their friends and acquaintances in a competitive, yet fun and relaxed environment. This project has reached almost all of its' objectives and requirements, and has plenty of room to grow in the future.

This report has showcased how an open-source, deployable application centred around providing users with the best possible experience at the lowest cost available is not only possible, but a process to be emulated. Too much of our software these days is created for and by large corporations to reap a profit by selling user data and displaying targeted advertisements while using unethical practices to keep users hooked.

Unfortunately, it is this project developer's belief that this trend will likely continue into the distant future under current economic and governmental systems, but by creating software such as this project's application, with an intent to improve user's experiences without the detrimental effects, and by applying this philosophy in one's daily life, conditions can improve.

In future once Tournamint has been polished, and features from *9.2 Future Work* implemented, user feedback can be gathered around its' function and iterative, open-source community involved development can begin.

## REFERENCES

- [1] [www.lolesportsmedia.com](http://www.lolesportsmedia.com) *League of Legends Esports Media Center - 'League of Legends Esports Breaks World Championship Viewership Record'*. [online]  
Available at: <https://www.lolesportsmedia.com/League-of-Legends-Esports-Breaks-World-Championship-Viewership-Record#:~:text=This%20year%27s%20Finals%20match%20saw>  
[Accessed 27 Jun. 2022]
- [2] Pei, A. (2019). *This esports giant draws in more viewers than the Super Bowl, and it's expected to get even bigger*. [online] CNBC.  
Available at: <https://www.cnbc.com/2019/04/14/league-of-legends-gets-more-viewers-than-super-bowlwhats-coming-next.html>  
[Accessed 28 August 2022]
- [3] Wikipedia. (2020). List of League of Legends leagues and tournaments. [online]  
Available at: [https://en.wikipedia.org/wiki/List\\_of\\_League\\_of\\_Legends\\_leagues\\_and\\_tournaments](https://en.wikipedia.org/wiki/List_of_League_of_Legends_leagues_and_tournaments)  
[Accessed 28 August 2022]
- [4] [www.workfront.com](http://www.workfront.com). (n.d.). Waterfall Methodology - A Complete Guide | Adobe Workfront. [online]  
Available at: <https://www.workfront.com/en-gb/project-management/methodologies/waterfall>  
[Accessed 28 August 2022]
- [5] League of Legends. (n.d.). Clash - League of Legends' tournament mode for teams. [online]  
Available at: <https://euw.leagueoflegends.com/en/featured/clash#/>  
[Accessed 28 Aug. 2022].
- [6] <https://www.buff.game/>. (2021). League of Legends World Championship History - BUFF. [online]  
Available at: <https://www.buff.game/league-of-legends-world-championship-history/>  
[Accessed 28 Aug. 2022].
- [7] Challengermode. (n.d.). League of Legends. [online]  
Available at: <https://www.challengermode.com/lol>  
[Accessed 28 Aug. 2022].
- [8] [challonge.com](http://challonge.com). (n.d.). Challonge. [online]  
Available at: <https://challonge.com/>  
[Accessed 28 Aug. 2022].
- [9] Salil Deshpande [www.techcrunch.com](http://www.techcrunch.com) (2018) – The crusade against open-source abuse [online]  
Available at: <https://tcrn.ch/2Q3vE2z>  
[Accessed 28 Aug. 2022].
- [10] [docs.oracle.com](http://docs.oracle.com). (n.d.). Dynamic typing vs. static typing. [online]  
Available at:  
[https://docs.oracle.com/cd/E57471\\_01/bigData.100/extensions\\_bdd/src/cext\\_transform\\_typing.html](https://docs.oracle.com/cd/E57471_01/bigData.100/extensions_bdd/src/cext_transform_typing.html)  
[Accessed 28 Aug. 2022].
- [11] [g2.com](http://g2.com) Vercel Overview [online]  
Available at: <https://www.g2.com/products/vercel/reviews>  
[Accessed 29 Aug. 2022].
- [12] Redis (n.d.). Redis. [online] [redis.io](http://redis.io).  
Available at: <https://redis.io/>  
[Accessed 29 Aug. 2022].
- [13] Masanja, Ndalaha. (2020). A PRACTICAL GUIDE TO WRITING A FEASIBILITY STUDY.  
Available at:  
[https://www.researchgate.net/publication/341134813\\_A\\_PRACTICAL\\_GUIDE\\_TO\\_WRITING\\_A\\_FEASIBILITY\\_STUDY](https://www.researchgate.net/publication/341134813_A_PRACTICAL_GUIDE_TO_WRITING_A_FEASIBILITY_STUDY)  
[Accessed 30 Aug. 2022]

- [14] freeCodeCamp.org. (2020). How to Take the Right Approach to Responsive Web Design. [online] Available at: <https://www.freecodecamp.org/news/taking-the-right-approach-to-responsive-web-design/> [Accessed 6 Sep. 2022].
- [15] Anon, (2020). League of Legends: Wild Rift Live Player Count and Statistics. [online] Available at: <https://activeplayer.io/league-of-legends-wild-rift/> [Accessed 6 Sep. 2022].
- [16] Spezzy (2020). How many people play League of Legends? - UPDATED 2020. [online] LeagueFeed. Available at: <https://leaguefeed.net/did-you-know-total-league-of-legends-player-count-updated/> [Accessed 6 Sep. 2022].
- [17] builtin.com. (n.d.). Why Is the Internet So Slow? (And How We Can Fix It). | Built In. [online] Available at: <https://builtin.com/software-engineering-perspectives/fix-javascript-bloat> [Accessed 6 Sep. 2022].
- [18] Locke, H. (2021). Why dark mode causes more accessibility issues than it solves. [online] Medium. Available at: [https://medium.com/@h\\_locke/why-dark-mode-causes-more-accessibility-issues-than-it-solves-d2f8359bb46a](https://medium.com/@h_locke/why-dark-mode-causes-more-accessibility-issues-than-it-solves-d2f8359bb46a) [Accessed 6 Sep. 2022].
- [19] Olashyn, A. (2020). Game Menu Dashboard UI Design. [Digital] Available at: <https://dribbble.com/shots/9993801-Game-Menu-Dashboard-UI-Design> [Accessed 6 Sep. 2022].
- [20] MDN Web Docs. (2019). About JavaScript. [online] Available at: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/About\\_JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript) [Accessed 9 Sep. 2022].
- [21] Wikipedia. (2022). React (JavaScript library). [online] Available at: [https://en.wikipedia.org/wiki/React\\_\(JavaScript\\_library\)#:~:text=It%20was%20first%20deployed%20on](https://en.wikipedia.org/wiki/React_(JavaScript_library)#:~:text=It%20was%20first%20deployed%20on) [Accessed 9 Sep. 2022].
- [22] reactjs.org. (n.d.). Introducing Hooks – React. [online] Available at: <https://reactjs.org/docs/hooks-intro.html#:~:text=Hooks%20are%20a%20new%20addition> [Accessed 9 Sep. 2022].
- [23] Liddell, A. (2021). Lighthouse Scores - What are they and why are they important? [online] MCD helps customers transform their businesses by harnessing the power of digital technology. We design and build digital products across mobile and web. Available at: <https://mcdsystems.co.uk/lighthouse-scores-what-are-they-and-why-are-they-important/#:~:text=What%20is%20a%20lighthouse%20score> [Accessed 11 Sep. 2022].
- [24] web.dev. (n.d.). Cumulative Layout Shift (CLS). [online] Available at: <https://web.dev/cls/>. [Accessed 11 Sep. 2022].
- [25] HM Government (2018). Data Protection Act 2018. [online] Legislation.gov.uk. Available at: <https://www.legislation.gov.uk/ukpga/2018/12/contents/enacted>. [Accessed 11 Sep. 2022].
- [26] Lawton, G. (2020). 5 examples of ethical issues in software development. [online] SearchSoftwareQuality. Available at: <https://www.techtarget.com/searchsoftwarequality/tip/5-examples-of-ethical-issues-in-software-development>. [Accessed 11 Sep. 2022].

[27] Hatescape: An In-Depth Analysis of Extremism and Hate Speech on TikTok Ciarán O'Connor. (n.d.). [online]

Available at: <https://www.politico.eu/wp-content/uploads/2021/08/24/ISD-TikTok-Hatescape-Report-August-2021.pdf>.

[Accessed 11 Sep. 2022].

[28] Dixon, S. (2022). Global Social Media Ranking 2021. [online] Statista.

Available at: <https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/>.

[Accessed 11 Sep. 2022].