Tour de icapps

Realization report

Internship ITFactory

Luca Van Genechten 3APPAI01

Academic Year 2022-2023

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

LID VAN ASSOCIATIE KU LEUVEN

THOMAS MORE

TABLE OF CONTENTS

1 INTRO

In this document, you will find the explanation of all my accomplishments which I realized during my internship at icapps. This internship took place from the 28th of Februari to the 27th of May of 2023.

To start off, I will be giving a small explanation detailing the company I took my internship at. Afterwards, I will be introducing the internship project itself, as well as an overview of the architecture and the technologies used to realize the project. I will also be giving insight into the internship and how I experienced it.

But before we continue, I would like to thank icapps again for not only their wonderful guidance throughout this internship, but the amazing opportunity they have given me.

2 INTERNSHIP COMPANY

icapps is a prominent digital agency based in Antwerp, Belgium, and it holds a notable position within the Cronos Group, a leading technology and innovation hub. As part of the Cronos Group, icapps benefits from a strong network of expertise and resources, enabling them to deliver cutting-edge solutions to their clients.

With a team of over 100 highly skilled professionals, icapps has successfully completed numerous projects, ranging from small-scale applications to large enterprise solutions. Their portfolio boasts a diverse range of clients from various industries, including finance, healthcare, retail, and more. This extensive experience allows icapps to understand the unique requirements and challenges specific to each sector, ensuring tailored and impactful digital solutions. Some of icapps' customers include widely known companies, such as Monizze, KBC, Telenet, Orange, Port of Antwerp, Luminus, and many more.

icapps has offices in both Antwerp and Mechelen, but my internship will take place at the office in Antwerp. I will be starting within Squad 42, a dedicated team of developers who specialize in one particular framework: Flutter.

3 THE PROJECT

3.1 Reason

Within icapps, sports is a true lifestyle. Many employees engage in daily cycling, friendly football competitions, darts, or spinning. This passion for sports led Hannes Van den Berghe, the current team lead of Squad 42 and an avid cyclist, to create a friendly cycling competition within icapps. This annual competition, known as "Tour de icapps," coincides with the Tour de France.

The Tour de icapps consists of multiple stages which happen during the span of a tournament. Employees can be asked whether or not they want to participate in the tournament, after which all participants will be divided among teams. Each stage is composed of multiple challenges, some examples being “Most kilometers cycled per team” or “Highest average top speed per team”. Participants then can do as many activities as they want during each stage.

Hannes has always acted as the judge of this competition and, as a result, had to meticulously record all activities performed by the participants using an Excel sheet and the Strava application. However, this process was time-consuming and burdensome for Hannes.

3.2 Goal of the project

The goal of this project is to develop a Minimum Viable Product (MVP) that Hannes can use to track all activities performed by players and automate scoreboards for the entire tournament.

Initially, all employees are asked to participate and can either accept or decline the invitation. Following that, teams are randomly created with the participating employees and assigned names. These teams are then stored and utilized within the application.

Participants will be able to complete activities during the tournament, which will be recorded on Strava. Strava is an internet service and application used for tracking physical exercise, offering various social network features. Participants will be able to view and upload all their tournament activities to the application.

Once an activity is uploaded, it will be linked to a stage. A stage is defined by two dates and has specific challenges associated with it. These challenges may vary in purpose, such as determining which team rode the most kilometers during a stage, which team achieved the most elevation meters, and so on.

To ensure the accuracy of uploaded activities, challenges and stages typically have requirements attached to them. For instance, in a stage focused on a time trial, participants must have exercised a distance greater than 25 kilometers. Alternatively, in a longer stage spanning multiple days, a requirement could be that the total distance covered by a team must exceed 250 kilometers. Activities that do not meet the requirements at the challenge or stage level are not eligible for point calculation.

Finally, once all challenges are completed and the tournament concludes, a ranking displaying the scores of all teams should be visible both on the stage and tournament levels.

4 ARCHITECTURE

Flutter is a powerful cross-platform framework developed by Google that allows developers to build beautiful and performant applications for various platforms, including web browsers. When it comes to Flutter for Web, it utilizes a unique architecture that combines the best of Flutter's reactive UI framework with the capabilities of the web platform.

At the core of Flutter for Web is the Flutter engine, which is responsible for rendering and updating the user interface. The engine leverages the Skia graphics library to draw and manipulate UI elements. Unlike traditional web development approaches that rely on HTML, CSS, and JavaScript, Flutter for Web utilizes Dart, a statically-typed programming language, to describe the UI components and logic. This enables developers to write a single codebase for both mobile and web applications, reducing development time and effort.

To enable Flutter applications to run in web browsers, Flutter for Web introduces a unique concept called "Hummingbird." Hummingbird is a framework that translates Flutter's UI rendering and layout semantics into optimized JavaScript code. It bridges the gap between Flutter's rendering engine and the web platform, allowing Flutter apps to be executed in a browser environment. This architecture ensures that Flutter for Web applications maintain the same high-performance and visually rich user experience that Flutter is known for, while seamlessly adapting to the web platform's capabilities.

Overall, Flutter for Web's architecture provides a streamlined and efficient way to develop cross-platform applications that run on the web. By combining the power of Flutter's reactive UI framework with the flexibility of the web platform, developers can create visually stunning and responsive web applications with ease. Whether it's building a simple website or a complex web app, Flutter for Web offers a robust architecture that simplifies development and enables consistent user experiences across different platforms.

During the development of Tour de icapps, I made use of the MVVM architecture as well. The Model-View-ViewModel (MVVM) architecture is a design pattern commonly used in Flutter applications. In MVVM, the Model represents the data and business logic, the View represents the user interface, and the ViewModel acts as a mediator between the Model and the View. The ViewModel exposes data and commands that the View binds to, allowing for separation of concerns and improved testability. In Flutter, the Model can be represented by data models or state management solutions like Provider or Riverpod. The View corresponds to the UI widgets and layout, while the ViewModel can be implemented using classes that manage the state and provide methods and properties for the View to consume. By following the MVVM architecture, Flutter developers can create scalable, maintainable, and testable applications with clear separation between the UI and business logic.

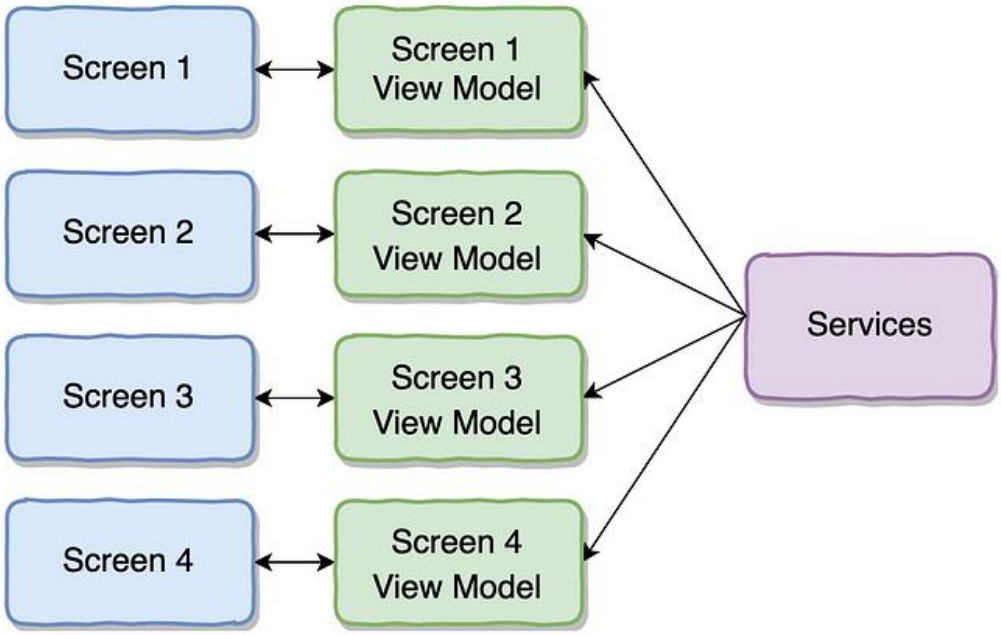

Figure 1: example of the MVVM architecture

4.1 Technologies

Tour de icapps is considered a full stack project, with multiple technologies responsible for keeping the front-end and back-end running. In the next paragraphs, I will describe the technologies and features used for each part of the project.

4.1.1 Back-end

The back-end of the project consists of a few parts. First off, let's talk about Flutter. icapps uses a custom template for all of their projects, which includes a variety of features. Some of these features include automatically generated service classes, DAO classes for data access as well as a package called Drift which can be used to create a local SQLite database. During the project, the service and DAO classes were some of the most used classes.

The service classes made use of 2 insanely handy packages to make API calls, namely Dio and Retrofit. The Dio and Retrofit packages are popular networking libraries for Flutter that provide convenient and efficient ways to make HTTP requests and handle API communication. Dio is a powerful package that offers a simple yet feature-rich API client for handling network requests. It supports various request methods, such as GET, POST, PUT, and DELETE, and provides features like interceptors, request cancellation, and form data handling. Dio also supports advanced features like file uploading and downloading, request timeout configuration, and request/response logging.

On the other hand, Retrofit is a package inspired by the popular Java library of the same name. It simplifies the process of defining RESTful API endpoints in Flutter by using annotations and automatic code generation. Retrofit allows developers to define the API interface with custom request methods and parameters, and it automatically generates the necessary code for making the requests. Retrofit supports query parameters, request headers, and

request/response serialization with various data formats like JSON and XML. It also supports different HTTP clients, allowing flexibility in choosing the underlying networking library.

Both Dio and Retrofit provide efficient and flexible solutions for networking in Flutter applications. They help streamline API communication, handle various types of requests, and offer advanced features for managing network interactions. Whether you prefer a more manual approach with Dio or a declarative and code-generation-based approach with Retrofit, these packages simplify the process of working with APIs and enhance the development experience in Flutter.

Usually, Flutter applications will use these backend packages to communicate with API's that can provide them with data from a service or a database. However, in Tour de icapps, this functions a little bit differently in some regards. Tour de icapps utilizes the Dio and retrofit packages in order to communicate with the Strava application. This is first seen in the login screen, where a communication is made between the Tour de icapps and Strava using single sign-on, or SSO. This communication is then used for further purposes, which will be explained later in this document.

What the backend of Tour de icapps does not do however, is use an API to communicate with its database of choice, Firestore. Instead, a direct communication is made between Tour de icapps and Firestore using the cloud_firestore package, allowing for instant retrieval of data without the need of an intermediary API.

To integrate Firestore in the backend, developers need to set up the necessary authentication and authorization mechanisms using Firebase Authentication. This ensures that the server has the required credentials and permissions to access the Firestore database. Once the authentication is configured, the cloud_firestore package allows developers to perform a wide range of operations, such as creating and updating collections, adding and modifying documents, and executing queries.

4.1.2 Front-end

The front-end of a Flutter application is built using a combination of widgets, layouts, and styles. Flutter follows a declarative UI programming model, which means that the user interface is described using code rather than through a visual editor. At the core of Flutter's front-end development is the concept of widgets. Widgets are the building blocks of the user interface, representing everything from buttons and text fields to complex layouts and animations.

Developers can use a wide range of pre-built widgets provided by Flutter, such as Container, Column, Row, Text, Image, and more. These widgets can be composed and nested to create complex UI structures. Flutter also offers the flexibility to create custom widgets by subclassing existing ones or creating entirely new ones. Widgets in Flutter are reactive and stateful, meaning they can be updated and respond to changes in the application's state.

To design the layout of the user interface, Flutter provides a flexible system called the widget tree. The widget tree represents the hierarchical structure of the widgets, with each widget having a parent and potentially multiple children. Flutter offers different layout widgets like SizedBox, Expanded, and Flexible that allow for precise control over the arrangement and sizing of widgets within a layout.

Styling and theming in Flutter are achieved through the use of properties and the ThemeData class. Properties like color, font size, padding, and margin can be customized to achieve the desired visual appearance. Flutter also supports hot-reloading, enabling developers to see immediate changes in the UI as they make modifications to the code.

Overall, the front-end of a Flutter application is built by composing and nesting widgets to create the desired user interface, defining layouts to structure the widgets, and applying styles and theming to achieve a visually appealing and consistent look. The declarative nature of Flutter's UI programming model, along with its rich set of pre-built widgets and flexible layout system, makes front-end development in Flutter efficient and powerful.

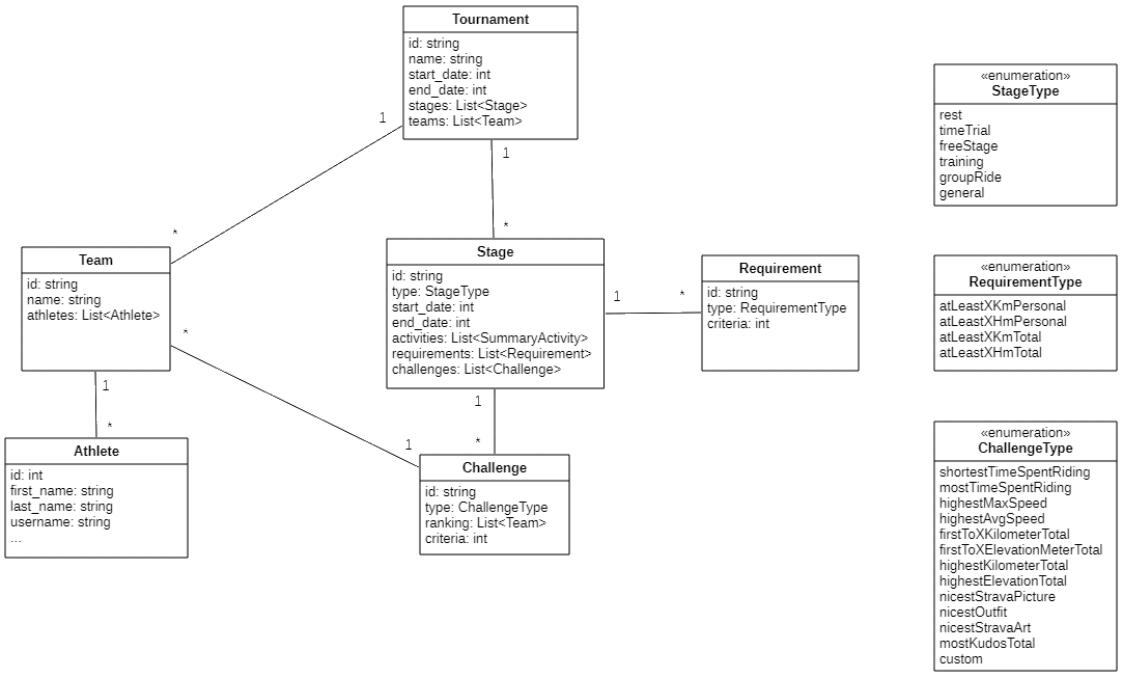

Figure 2: class diagram of the application

5 OVERVIEW OF ALL PAGES

In the next section of the document, I will be going over each page individually and explaining its purpose, different features and how it works. To start off, we will take a look at the first page a user will see once they navigate to the website of the application.

5.1 Login screen

The login screen is the very first page a user can see. In order to fully use the application, a user needs to be able to log into Strava in order to retrieve activities and upload them. That's where this page comes into play. The only piece of interaction a user will have with Strava will happen on this page. Once they click on the Authorize button, a screen will pop up where a user can log into Strava using whichever method they prefer, whether this is Google, Facebook or via email. Strava and the application then use a process called single sign-on (SSO) to securely log in the user. This process is explained on a later page. When the authentication process is completed, the user will then be referred to the team selection page, or the home page if the user had joined a team prior.

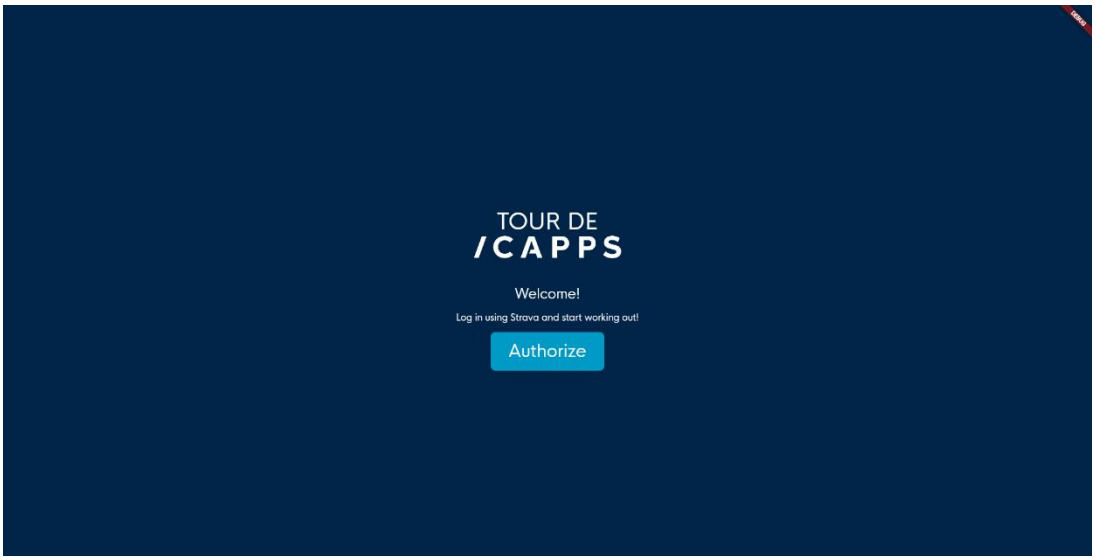

Figure 3: login screen

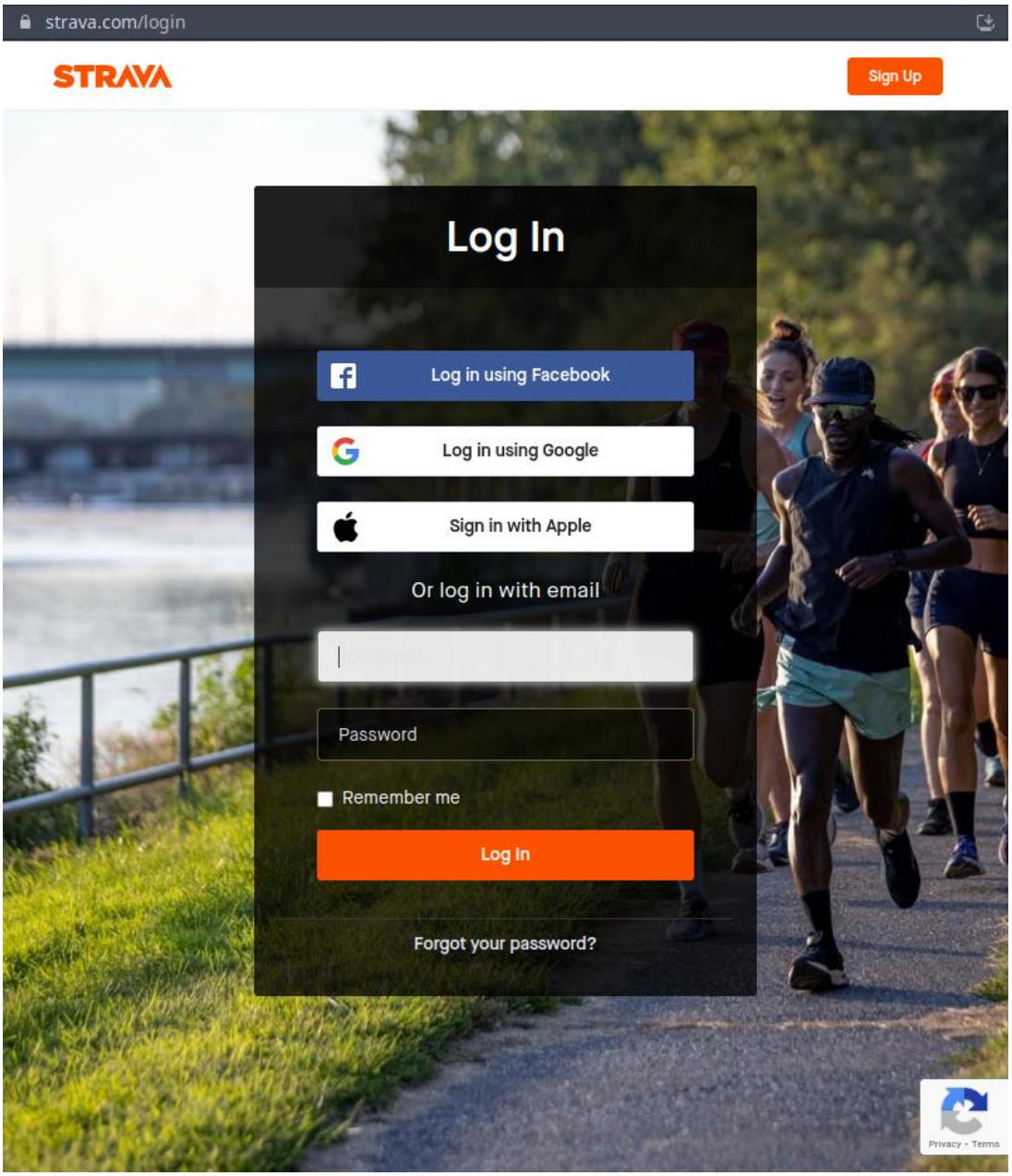

Figure 4: pop-up to authenticate the user

5.2 Team selection screen

Once the user has successfully authenticated using SSO, they are then redirected to the following page. This page's sole purpose is so the user can select a team to join once they have logged in. This page is only shown once, when a user logs in and their Strava ID is not yet associated with a team. Here, a user will be able to choose which team to join, with a choice between all registered teams in the application. Once a team is selected, they can confirm their choice and continue to the home screen of the application.

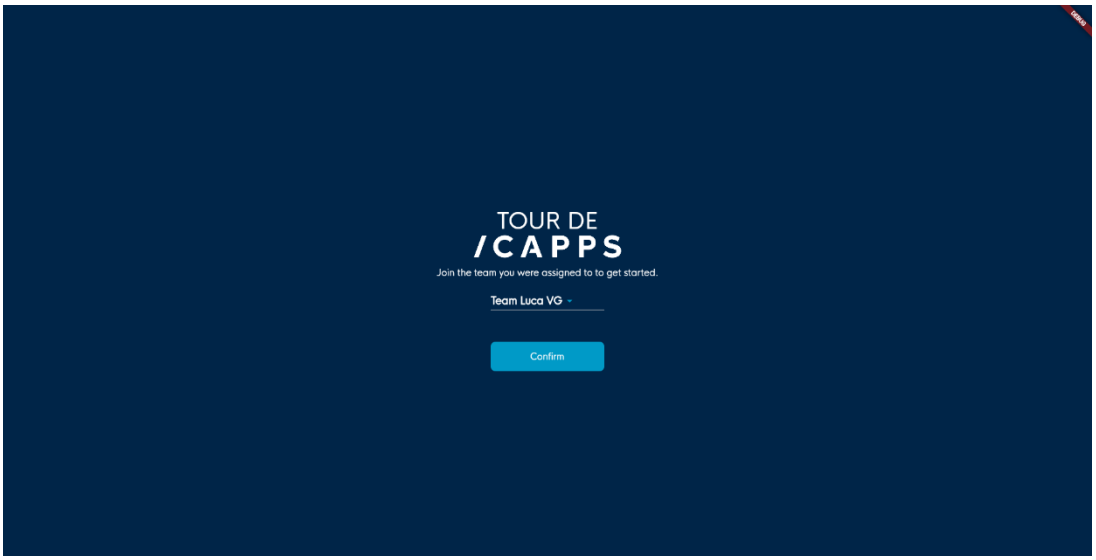

Figure 5: team selection screen

5.3 Home screen

The home screen is the screen that all users will see once they have logged in and have selected a team to join. The home screen utilizes a few widgets to display a variety of information. When a user logs in before a tournament has started, they will be greeted with a countdown timer on the left-hand side of the screen. This timer indicates how many days, hours, minutes and seconds are left until the tournament starts. Once this timer reaches 0, it will automatically be replaced with a text message indicating that the tournament has started.

Below the countdown, a button can be seen saying “View stages”. Clicking on this button will redirect the user to the next screen of the application, the tournament screen.

On the right-hand side of the screen a ranking I shown indicating the scores each team has received in total over the course of the tournament. These scores are calculated by calculating a ranking for each challenge in the tournament. For each challenge, a score can be received from 3 to 1, with first place receiving 3 points, second place receiving 2 points, and third place receiving 1 point. Only the top 3 teams per challenge can receive points. The total scores per team are then calculated and shown here in a ranking.

Another thing we can see on this screen is the navigation bar at the top of the screen. On the current page, we have 2 buttons: the logo of the app in the top left and the “log out” button in the top right. The logo will redirect the user back to this page and is accessible on each page of the application. The “log out” button is only accessible on the home page of the application.

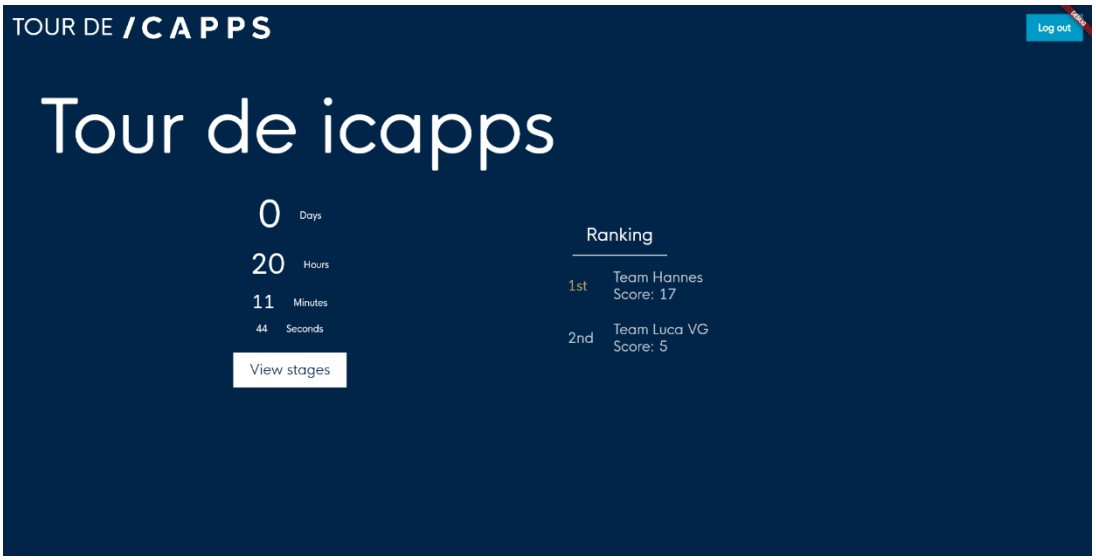

Figure 6: home screen with countdown

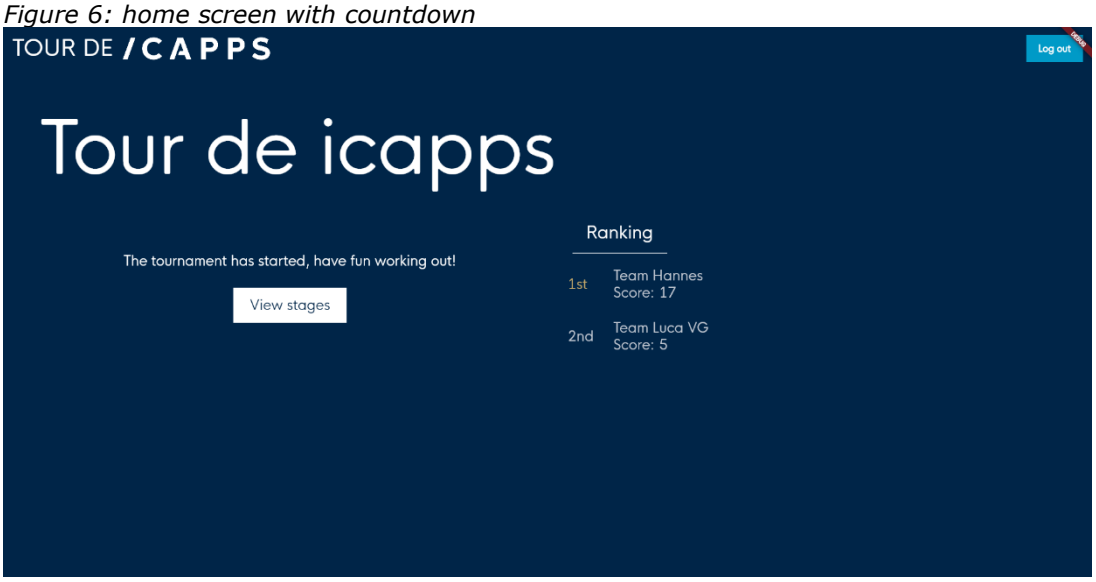

Figure 7: home screen without countdown

5.4 Tournament screen

The tournament screen is the where all information will be shown regarding uploaded activities, challenges per stage, as well as the rankings for said challenges. When we first enter this screen, we are greeted with a tab view of our tournament. Each stage of the tournament is represented with its own tab, where all uploaded activities per stage are shown in a scrollable list. All stages are shown here, except for stages that have been marked as being "rest" stages, where no activities need to be done. By clicking on a tab at the top of the screen, the user can navigate between stages.

On the left-hand side of the screen, we can see each individual activity uploaded per user. Each activity also shows relevant information, such as the distance of the activity, the time spent moving, the amount of elevation meters done, as well as the average and maximum speed at which the activities were done. This is all information which is retrieved directly from Strava.

On the right-hand side of the screen, we can see rankings for each challenge in a stage. On the example below, we can see that this stage in particular only has 2 challenges associated with it: "Highest max speed" and "Shortest moving time to 22km".

On the navigation bar at the top of the screen, we can now see that the logo of the application remains to navigate to the home screen, but a title for the current page is also added. On the right side of the bar, we can see that the "log out" button of the previous page is now gone and instead shows an "upload" button. Clicking on this button will allow us to navigate to the next page.

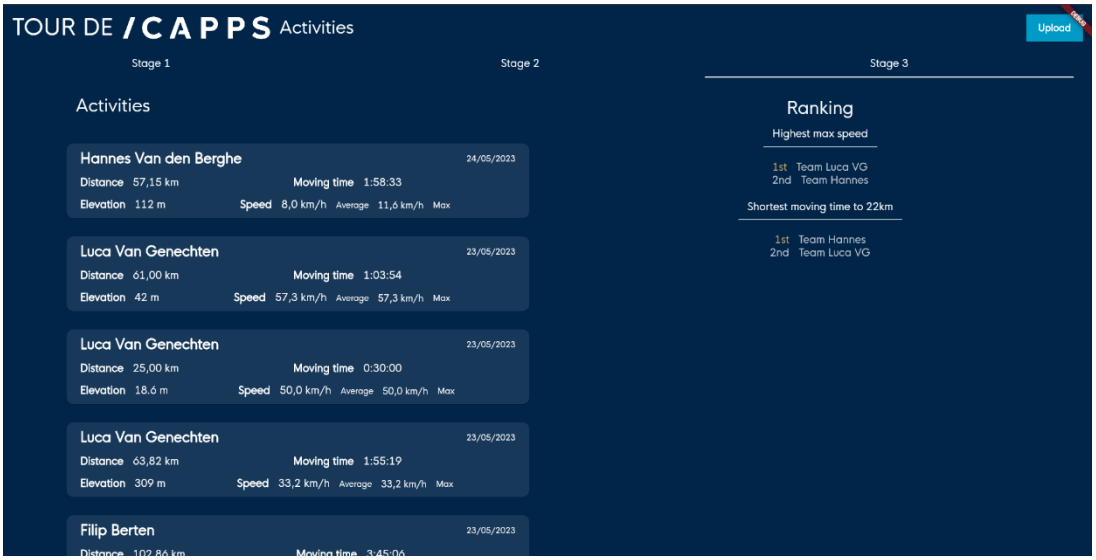

Figure 8: tournament screen

5.5 Upload screen

When we enter the next page, we can see that we have now reached the uploadzone of the application. On this page, a user has a variety of interactions. At the left-hand side of the screen, we are shown a list of all bicycle activities that have been done by the user during the time period of the current ongoing tournament. The application only shows "Ride", "Gravel Ride" or "Mountain Bike Ride" activities, as those are the only ones that are valid for this tournament. All of these activities have been retrieved from Strava using various API calls.

By clicking on each activity, we can see that the checkbox will become checked and the activity will be added on the right hand side. Using the handy "Select All" button, we can instead select or deselect all activities in the list.

Once the user has chosen which activities they want to upload, they can click on the "Upload" button on the right-hand side of the screen. This will begin the process of adding each chosen activity to their corresponding stage. If an error occurs during the process, this will also be reflected in a Toastr pop-up at the bottom of the page.

When uploading an activity, the application will check the date of when the activity has been done. If the activity falls between the start and end date of a stage in the tournament, the activity will be added to said stage. If a user is late in uploading their activity however, they will have 3 days of leeway. If an activity which happened during a stage is uploaded within 3 days after said

stage has ended, it will still be uploaded correctly. Otherwise the user will receive an error.

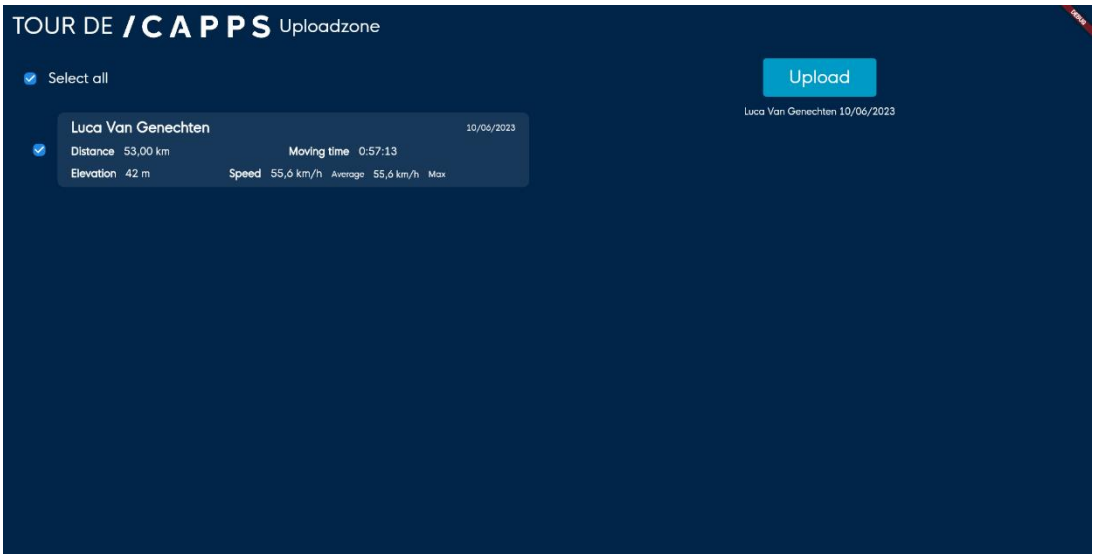

Figure 9: upload screen with activities

When this screen is reached without there being any record of activities during the time period of the tournament, the following message will be shown. The user will not be able to upload any activities in this case.

Figure 10: upload screen without activities

Whenever the user attempts to upload an activity which did not occur between the start and end date of a stage, the following error message will be shown to the user and the activity in question will not be uploaded.

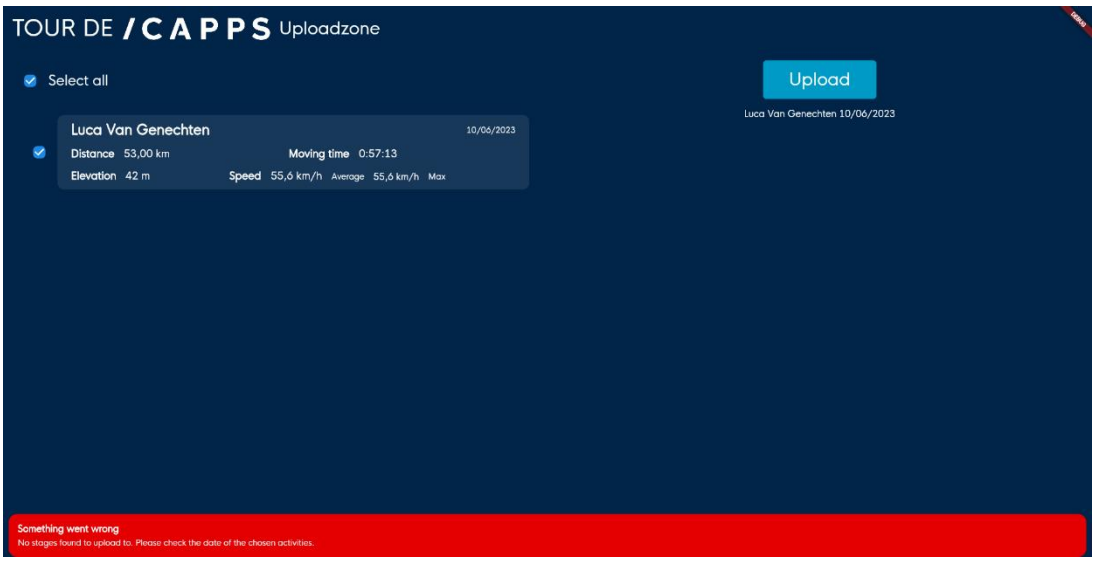

Figure 11: upload screen with error

6 INTEGRATIONS

6.1 Single Sign-On (SSO)

When a user initiates the login process in the Flutter app, it redirects them to the Strava OAuth authentication page. The app sends a request to Strava's authorization server, including the necessary credentials and scopes indicating the requested permissions. These scopes are determined by the developers of the app and determine how much access the application will have to the data of the authenticated user. In case of the Tour de icapps, the application will have access to all public profile data, as well as all activities.

Strava then presents the user with a login page, where they can enter their Strava credentials. Once the user successfully logs in, Strava's server validates the credentials and generates an authorization code. This code is then passed back to the Flutter app's backend server or directly to the app, depending on the implementation.

The Flutter app can then exchange the authorization code with Strava's token endpoint to obtain an access token and a refresh token. The access token represents the user's authorization to access their Strava data, while the refresh token allows the app to obtain a new access token without requiring user interaction.

With the access token, the Flutter app can make authorized API requests to Strava's servers on behalf of the user. This enables the app to retrieve activity data, upload new activities, and perform other Strava-related operations. The access token has an expiration time, but it can be refreshed using the previously obtained refresh token to maintain ongoing access.

On the next page, a graph is shown visualizing the authentication flow of an SSO session using Strava.

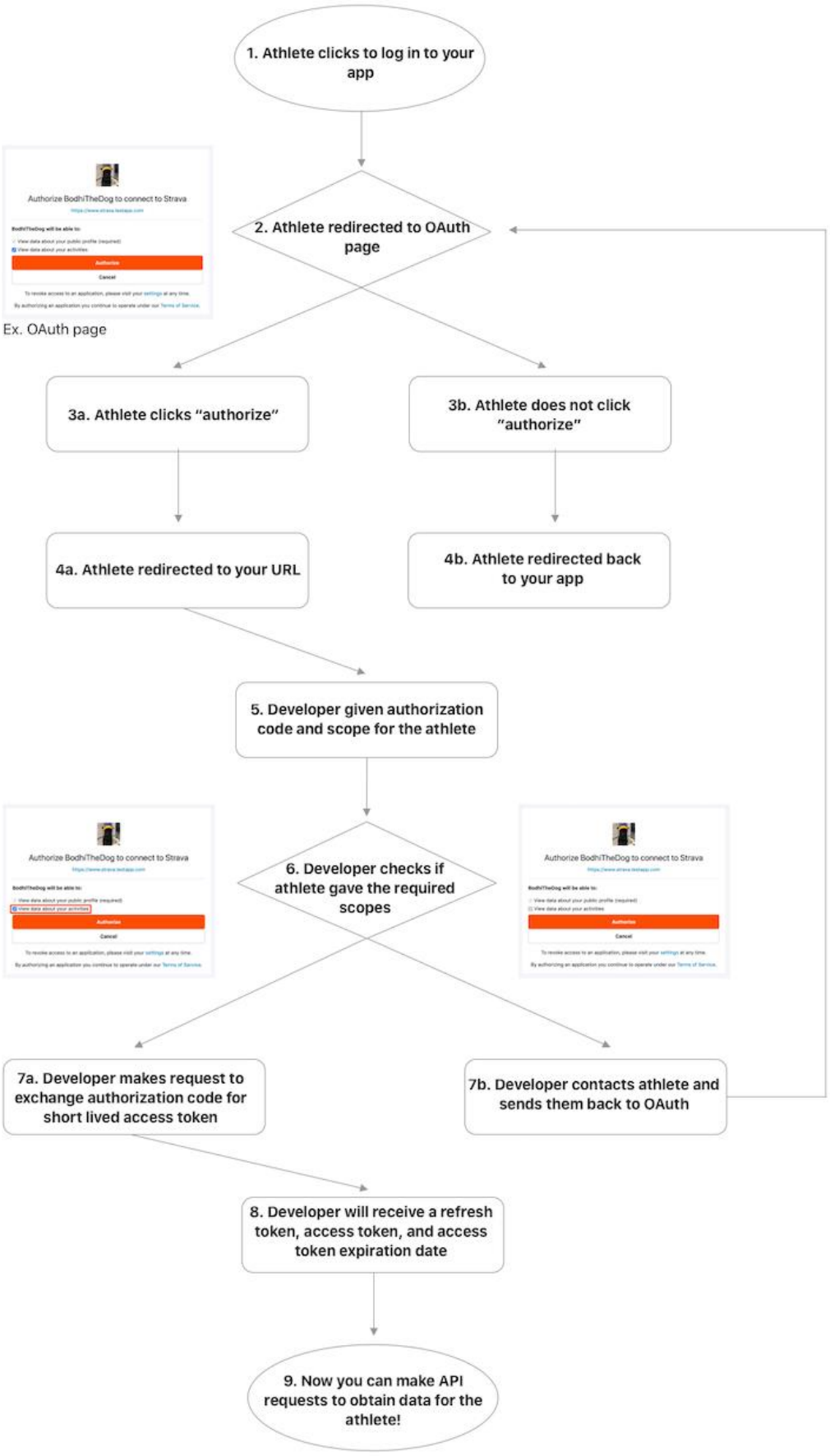

Figure 12: Strava SSO authentication flow

6.2 Firestore

In the Tour de icapps, Firebase is a core part of the application. More specifically, Firestore database was used as the main method of storage for this project. Firestore will store all user data, as well as serve as a configuration point for creating, updating and deleting data related to a tournament. In the upcoming section, I will be going over the Firestore database where all data of the application is stored and explain each collection in depth.

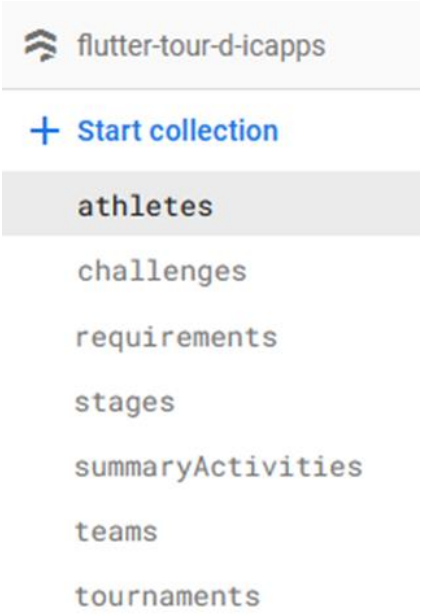

Figure 13: all collections stored in Firestore

6.2.1 Collection tables

In the section below, all existing collections within the Firestore database will be shown with their respective names, fields, data types per field and the description detailing which function each field serves. There are certain collections which are generated automatically or contain data received from public API's like Strava, such as the athletes table. A link to the data models corresponding to these collections will be included.

In these collections, there will be several cases of Lists being utilized. Because Firestore is not a relational database and is made in NoSQL, these lists cannot be directly declared. Instead, Firestore utilizes a technique called "references". Each reference declares a connection between documents stored in Firestore, defined by the field type "reference" and a document path declared as "collection/documentId".

For example, to link a team within the collection "teams" with document ID "team001" to a tournament, you can create a field in a tournament document of the type "array", followed by a field of the type "reference" with the value "teams/team001", as shown below.

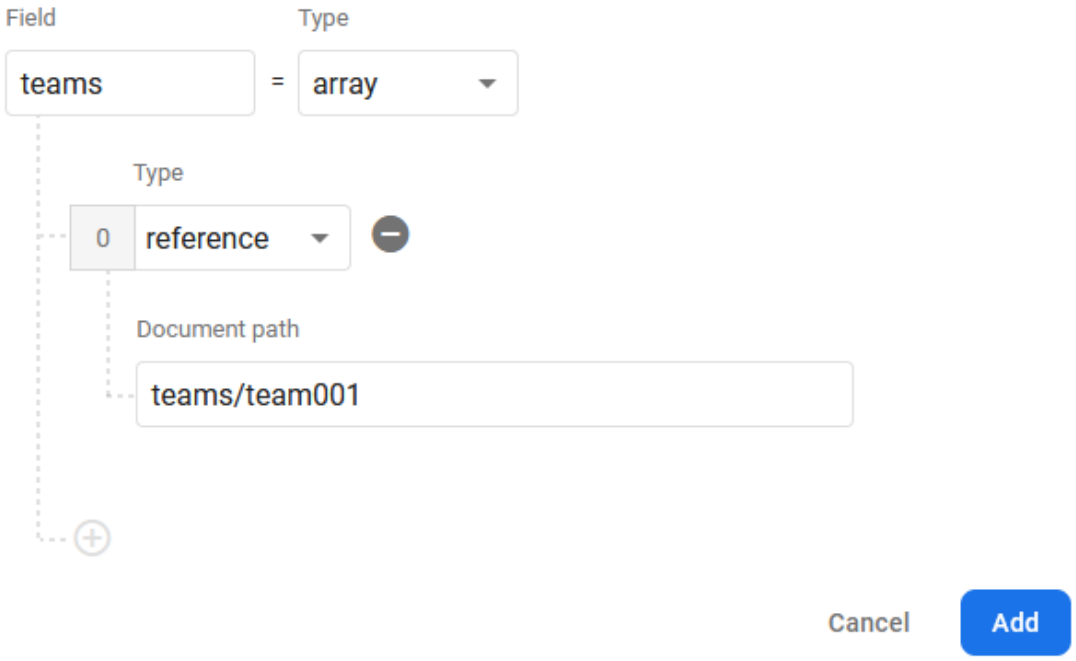

Figure 14: example of declaring a reference between documents

Collection name	Field name	Data type	Description
Athletes	A detailed explanation of the DetailedAthlete data objects stored in this collection can be found on the official Strava documentation.		
SummaryActivities	A detailed explanation of the SummaryActivity data objects stored in this collection can be found on the official Strava documentation.		
Tournaments	ID	String	A unique identifier, autogenerated by Firestore.
	Name	String	A name given to a tournament
	Stages	List<Stage>	A list of stages linked to a tournament.
	Teams	List<Team>	A list of teams linked to a tournament.

	Start_date	Int	An epoch timestamp designating the start date of the tournament.
	End_date	Int	An epoch timestamp designating the end date of the tournament.
Stages	ID	String	A unique identifier. Can be autogenerated.
	Type	Enum/String	An enum typing detailing what type of stage the athletes will have to do: Rest: a rest stage indicates there will be no competition on that day Time Trial: a time trial stage indicates that there will be a competition of speed Free Stage: a free stage indicates that the athletes will be free to ride as they wish, there will be no specific goals Training: a training stage indicates a stage where athletes can ride mostly for fun, as to prepare for upcoming stages. This usually has some fun challenges attached. Group Ride: a special type of stage which indicates all athletes participating will ride together General: a general stage is generally seen as a starting stage, with

			warm-up challenges attached
	Start_date	Int	An epoch timestamp indicating the start date of the stage
	End_date	Int	An epoch timestamp indicating the end date of the stage
	Activities	List <SummaryActivity>	A list of all activities and their athletes linked to a stage
	Challenges	List<Challenge>	A list of all challenges linked to a stage
	Requirements	List<Requirements>	A list of all requirements linked to a stage
Requirements	ID	String	A unique identifier, can be autogenerated
	Criteria	Int	An integer which can be passed along with the requirement
	Type	Enum/String	An enum typing detailing what type of requirement an activity needs to meet before being eligible for score calculation. This type is further explained below.
Challenge	ID	String	A unique identifier, can be autogenerated

	Criteria	Int	An optional integer which can be passed along with the challenge
	Ranking	List<Team>	A list of teams detailing the current ranking of all teams within a challenge.
	Type	Enum/String	An enum typing detailing what type of challenge an activity needs to meet before being eligible for score calculation. This type is further explained below.
Teams	ID	String	A unique identifier, can be autogenerated by Firestore
	Name	String	A name given to a group of athletes
	Athletes	List<Athlete>	A list of athletes belonging to a Team

6.2.2 Requirement type

The requirement type is an enumerable which represents which calculation must be done in order to deem an activity, or list of activities, as valid. Once these activities are deemed valid, they can then be used to calculate the rankings for each challenge. In the table below, each type is described in detail.

type	description
atLeastXKmPersonal	In order for a single activity to be deemed valid, the total distance travelled during the activity must be at least X kilometers
atLeastXHmPersonal	In order for a single activity to be deemed valid, the total elevation meters done during the activity must be at least X meters
atLeastXKmTotal	All activities linked to a stage are first grouped per team, after which the sum total of distance travelled by the entire team is calculated. If the sum total

	exceeds the criteria X, then that list of activities is deemed valid.
atLeastXHmTotal	All activities linked to a stage are first grouped per team, after which the sum total of elevation meters done by the entire team is calculated. If the sum total exceeds the criteria X, then that list of activities is deemed valid.

6.2.3 Challenge type

The challenge type is an enumerable which represents which calculation must be done in order to calculate the rankings for each challenge. Before each calculation is done, all activities in a stage are grouped per team. Afterwards, each list of activities per team is put through the calculator and then placed in a ranking.

type	description
shortestTimeSpentRiding	The average of all time spent riding per activity. The team with the lowest average time is ranked first.
mostTimeSpentRiding	The sum total of all time spent riding per activity. The team with the longest time is ranked first.
highestMaxSpeed	The average of all maximum speeds per activity. The team with the highest maximum speed is ranked first.
highestAvgSpeed	The average of all average speeds per activity. The team with the highest average speed is ranked first.
firstToXKilometerTotal	First, the sum total of all distance travelled per activity is calculated. Afterwards, the time spent riding is calculated. If the sum total distance travelled reaches the criteria X, the team with the least time spent riding is ranked first.
firstToXElevationMeterTotal	First, the sum total of all elevation meters done per activity is calculated. Afterwards, the time spent riding is calculated. If the sum total distance travelled reaches the criteria X, the team with the least time spent riding is ranked first.
highestKilometerTotal	The sum total of all distance travelled per activity. The team with the highest amount of kilometers is ranked first.
highestElevationTotal	The sum total of all elevation meters done per activity. The team with the highest amount of elevation meters is ranked first.

nicestStravaPicture	A custom challenge. This has no ranking calculation attached, but the winner is declared by vote.
nicestOutfit	A custom challenge. This has no ranking calculation attached, but the winner is declared by vote.
nicestStravaArt	A custom challenge. This has no ranking calculation attached, but the winner is declared by vote.
mostKudosTotal	The sum total of all kudos collected per activity. The team with the most amount of kudos is ranked first.

6.3 Remote Config

Firebase Remote Config is a powerful feature of the Firebase platform that enables developers to dynamically customize the behavior and appearance of their mobile or web applications without requiring app updates. By utilizing Remote Config, developers can define key-value pairs in the Firebase console, which are then fetched by the app during runtime.

This allows for on-the-fly configuration changes, such as adjusting feature flags, content variations, or UI settings, without the need for recompilation or redeployment. With Firebase Remote Config, developers can efficiently perform A/B testing, deliver personalized experiences, and rapidly iterate on app configurations to enhance user engagement and satisfaction. In case of Tour de icapps, it can be used to change the current active tournament ID on the fly.

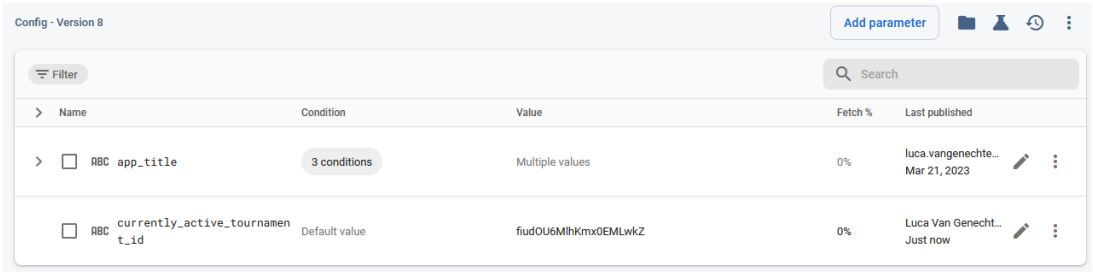

Figure 15: screenshot of Firebase remote config

6.4 Crashlytics

Firebase Crashlytics is a robust crash-reporting tool provided by Firebase that helps developers monitor and analyze app crashes in real-time. When integrated into a mobile or web application, Crashlytics automatically captures crash reports and provides comprehensive insights into the root causes of crashes.

It offers detailed crash logs, stack traces, and relevant metadata, enabling developers to quickly identify and prioritize issues for debugging and resolution. Crashlytics can be used to track crash occurrences, understand the specific conditions leading to crashes, and ultimately improve app stability and user experience.

However, Crashlytics does not work on Flutter for Web because it is designed specifically for native mobile platforms. Flutter for Web allows developers to build web applications using Flutter framework, and it relies on different underlying technologies compared to native mobile platforms. As a result, Crashlytics does not have direct support or integration for Flutter-based web applications, and alternative crash-reporting solutions may need to be considered for web-specific environments.

7 CI/CD

7.1 Bitbucket

Bitbucket is a web-based version control platform designed to simplify and streamline the management of source code repositories. It provides a collaborative environment for software development teams to store, track, and manage code changes effectively.

Bitbucket supports both Git and Mercurial, allowing developers to choose their preferred version control system. With Bitbucket, developers can create repositories to store their code, branch off to work on new features or bug fixes, and create pull requests for code reviews.

It offers powerful collaboration features such as inline commenting, integrated issue tracking, and integration with popular development tools. Bitbucket can be used for version control, code collaboration, and managing the software development lifecycle. It enables teams to work together efficiently, track changes, ensure code quality, and facilitate the continuous integration and delivery of software projects.

7.2 Firebase

Firebase Hosting is a powerful hosting service provided by Firebase that simplifies the deployment and hosting process for web applications. With Firebase Hosting, developers can easily deploy their static or dynamic web content, including HTML, CSS, JavaScript, and assets, to a global content delivery network (CDN) with a single command. The CDN ensures fast and reliable content delivery to users worldwide, resulting in low latency and improved performance.

Firebase Hosting also provides features like custom domain setup, SSL certificate management, and URL redirects. It can be used to host both single-page and multi-page applications, making it suitable for a wide range of web projects.

In terms of Flutter for Web, Firebase Hosting works seamlessly and efficiently. It supports hosting Flutter-based web applications, allowing developers to deploy their Flutter for Web projects effortlessly and take advantage of the scalability and performance benefits provided by Firebase Hosting.

7.3 Jenkins

Jenkins is an open-source automation server that provides a robust framework for building, testing, and deploying software projects. It enables developers to automate various stages of the software development lifecycle, such as building code, running tests, and deploying applications.

Jenkins works based on a distributed architecture, where jobs or tasks are executed on remote servers called agents or slaves. It supports a wide range of plugins and integrations, allowing seamless integration with various development tools, version control systems, and deployment platforms. Jenkins can be used for continuous integration (CI), enabling developers to integrate their code changes frequently and detect issues early.

Additionally, Jenkins facilitates continuous delivery and continuous deployment (CD), automating the release and deployment of software. With its extensive plugin ecosystem and flexibility, Jenkins is a versatile automation server that can be tailored to meet the specific needs of software development teams.

7.4 Compatibility

The compatibility between Bitbucket, Jenkins, and Firebase Hosting allows for the seamless formation of a robust CI/CD pipeline. Bitbucket serves as the version control platform, enabling teams to manage and collaborate on their source code repositories.

Jenkins, as an automation server, integrates with Bitbucket to trigger builds, tests, and deployments based on code changes. Jenkins pipelines can be set up to monitor Bitbucket repositories, automatically initiate the CI/CD workflow when changes are detected, and perform tasks like building and testing the code.

Once the code passes the required tests, Jenkins can seamlessly deploy the application to Firebase Hosting. Firebase Hosting provides a reliable and scalable hosting platform for web applications, ensuring fast and global content delivery through its content delivery network (CDN). By combining the strengths of Bitbucket, Jenkins, and Firebase Hosting, developers can establish an end-to-end CI/CD pipeline that simplifies the development process, improves code quality, and accelerates the deployment of their web applications.

8 SCHEDULE

During the first 3 weeks of my internship, my main focus will be on building up my knowledge base. I will aim to absorb as much as possible about Flutter for web and gain a clear understanding of what lies ahead during the realization phase. This period will include learning about icapps as a company and working on a starter project called the Flutter Beer App.

The Flutter Beer App serves as a starting point for new hires and interns to familiarize themselves with the Flutter framework. It also teaches the use of various tools, scripts, and templates offered by icapps. I have been given three weeks to explore this sandbox project, with specific small goals in mind to learn basic Flutter techniques.

Once I have a good grasp of these techniques, I will move on to setting up the Tour de icapps project. icapps provides a starter template for all their new applications, which includes a comprehensive toolkit featuring utilities such as a debug screen, project structure, and a local storage database. The Beer App is built upon the same starter project.

Upon completing the first 3 weeks, I will begin working on the Tour de icapps project itself during the realization phase. The realization phase will be divided into five sprints over the course of 10 weeks, focusing on the following areas:

1. General project setup:
    - Configuration of Strava integration
    - Firebase setup
    - Single Sign-On implementation
2. Strava interaction:
    - Retrieving activities from logged-in users
3. Stages & tournaments setup:
    - Defining the tournament structure in Firebase
    - Displaying the tournament structure in the application
4. Home screen & score calculation:
    - Creating a homepage to greet users and display a countdown until the next tournament
    - Implementing a ranking system to show scores for the entire tournament
    - Developing a ranking system to show scores per stage
5. Calendar & landing:
    - Implementing a calendar feature to display all stages, their requirements, and challenges
    - Making final adjustments to ensure the project lands securely
    - Completing documentation for the project

These five sprints will form the core of the realization phase and guide the development of the Tour de icapps project.

9 CONCLUSION

Upon looking back at this project, I am left with a very pleasant feeling. The end result of internship is a fully functional web application developed entirely with Flutter, with a pleasant yet professional interface. Most of the desired functionalities are present and the feedback I received along the way helped me shape this application into a fully fledged Flutter for Web application.

I personally think this project was a great finisher for my high school career, as it allowed me to prove myself as an independent and professional developer. The purpose of this project was not only to provide Hannes with a great alternative to manual labor using Strava, but also to serve as a font of knowledge. It became an opportunity to not only learn more about Flutter for Web and its many applications, but to spread the knowledge I gained along the way.

This project has taught me many new things about Flutter that I previously had no idea about, and I am excited to be able to work with this technology more in the future. Besides various hard skills being improved upon, this internship proved to be a precious learning experience for various soft skills as well, as this was one of the first experiences I've had as a Flutter developer in a large company. Because of the amazing guidance I received from my supervisors and the various developers around me, as well as the plentiful teambuilding exercises along the way, it has truly helped me grow.

Of course, I am not the only person who needs to be happy with the final result, the client, Hannes, needs to be content as well. I truly hope I was able to produce a viable product which can be used for many years to come within icapps.