B.Sc. (Hons) in Software Development

Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

Social Fitness

By
Daniel Fitzsimons

August 7, 2024

Minor Dissertation

Department of Computer Science & Applied Physics,
School of Science & Computing,
Atlantic Technological University (ATU), Galway.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.0.1 Rationale

As an active participant in team sports, I've observed the communication challenges faced by football teams that rely on a mix of software like Strava, WhatsApp, and Google Drive for various aspects of their training and coordination. This fragmentation of tools can lead to inefficiencies and hinder the ability of coaches and players to work cohesively.

As the pace of life accelerates, we understand that the challenges of staying motivated, maintaining consistency with workouts behind the scenes or staying connected with your group or clients on the same platform. With many gym apps available in today's market, it became a challenge to really narrow down what problems can be addressed within the fitness community, which apps lack certain necessities and what differs between the apps.

I began to notice one major problem with all these gyms app. Sure you can get a basic app that allows you to record your body weight, your gym routines or follow that your players are doing what needs to be done. This is where Social Fitness sprung to my mind.

With Social Fitness, the aim is to bridge this gap by developing a communal sports/fitness app tailored to the unique needs of football teams. This app will serve as a centralized hub for communication and collaboration, streamlining the process for coaches and players.fingertips.

Lets explore how Social Fitness can help you reach your fitness aspirations, motivate others, and be part of a dynamic fitness revolution. Get ready to experience a whole new dimension of fitness and community right from your mobile device. Its time to drive each other to unlock the best version of you, to reach your dreams!

As an active member of the GAA community in my local town. I always felt the awkwardness of using many different apps to communicate and track our workouts.

Using whats app as a way of communicating via text messaging. This would be where coaches would upload the plan for training, where he could communicate with us all in one group. This is platform 1 in which we use within the team.

Platform 2 follows Strava. Strava is an internet service for tracking physical exercise which incorporates social network features. It started out tracking mostly outdoor cycling and running activities using Global Positioning System data, but now incorporates several dozen other exercise types, including indoor activities. Strava allows for the use of social aspects like posting your your workout to a feed, like Facebook.

Looking at the first two platforms, I'm looking to merge these two apps together. By this i mean allowing a manager to create a small community within an app which will allow the players to post their workouts to a feed, where the manager or personal trainer can see this, but to also allow them to just slide or select your name and message you directly or within a group message. This can vary for a direct message to one individual for a for personal trainers for confidentiality, or a message into group compiled of all group members on the team.

Key features of this app will include the ability to record gym sessions and track metrics such as heart rate and calories burned, facilitating both individual progress tracking and group accountability. The app will also enable seamless communication through messaging and group chats, fostering discussions about match activities and training strategies.

The integration capabilities of the app will be one of its biggest strengths. Instead of depending on external platforms such as Google Drive for sharing videos and documents, users will have the ability to access and share these resources within the app. This simplified approach will allow coaches to keep track of players' off-field progress and overall development.

Our project is focused on simplifying and enhancing the sports team experience by providing the necessary tools for coaches and players to succeed. We aim to address the existing challenges in communication and coordination and offer a comprehensive, user-friendly solution to overcome them.

1.0.2 Objectives

- **Cross-Platform Compatibility:** Ionic is a great choice for cross-platform development. You can create an app that works on both Android and iOS with a single codebase.

- **Cloud-Based Resource Upload:** Use cloud storage services like AWS S3, Firebase Cloud Storage, or Azure Blob Storage to allow teams to upload and access resources. Secure user authentication is crucial here.

- **Fitness Data Sync:** To sync data with smartwatches and fitness trackers, you'll need to integrate with their respective APIs. Fitbit, Apple Health, and Google Fit are some common platforms for this.
- **User Profiles:** Implement user registration and profile management within the app, storing user data securely.
- **User Interface:**Design an intuitive and user-friendly interface following best UX/UI practices.
- **Group Chat and Social Features:** Use technologies like Firebase Real-time Database or a custom backend to create and manage group chat and social features. Consider features like text and media messaging, group creation, and moderation.
- **Content Sharing:** Implement a content sharing feature, possibly utilizing a database for user-generated content. Ensure content quality and compliance with relevant laws.
- **Scheduling and Notifications:** Use push notifications to remind users of upcoming events. Calendar integration can help schedule individual and group sessions.
- **Data Security:** Ensure user data is stored securely and complies with data privacy regulations, such as GDPR for European users. Use encryption and proper authentication mechanisms.
- **Gamification:** Add gamification elements to your app, such as achievements, badges, and rewards for user engagement and consistency.

1.0.3 Technology Used

1. **Framework : Ionic/Angular:**
    - To develop this system, a cross platform framework is a must as many users would vary between using android or apple as their preferred devices. To reach a maximum capacity of users within the community i felt Ionic/angular was a perfect match for the framework. It allows for the deployment of an app from a single code-base. This benefits maintenance and development as it doesn't endure editing different code bases for different platforms. Its large community makes it versatile in finding resources, plugins and third party tools which is a must in a project like this as the use of geographical positioning is needed for gym tracking and running.

2. **Database : Firebase:**

    - Fireabase Ideal for cloud storage of resources like match analysis videos or photos. It also handles user authentication efficiently, making it a suitable choice for player and coach sign-ups. Firebase offers valuable analytics tools for tracking user engagement and provides a scalable, real-time database.

3. **Cloud/Third-Party Libraries**

    - **Google Analytics:** Implement Google Analytics to track user behaviour, analyse user engagement, and make data-driven decisions for improving the app.
    - **Google Cloud/Firebase:** Use Google Cloud/Firebase for delivering media content to users. It's a reliable and scalable platform for hosting and distributing various types of content.
    - **Google Maps API:** Integrate the Google Maps API for geolocation services, allowing users to track their gym sessions or running routes with ease.

4. **Presentation**

    - **HTML/CSS/JavaScript/TypeScript:**Utilize HTML for creating the structural views of the app's content, CSS for visually styling the app and providing an appealing user interface, and JavaScript/TypeScript (Angular) for implementing dynamic and interactive features. Angular, built with TypeScript, is an excellent choice for creating a responsive and user-friendly UI.

5. **Operating Systems**

    - **Android and iOS:** These two major platforms ensure that your app reaches a broad user base. Both platforms are essential for maximum market coverage.

1.0.4 GitHub Repository

Version control is essential for modern software development. This project utilized GitHub, a prominent version control hub, to manage source code, documentation, and other vital assets. GitHub's adoption was due to its collaborative tools like branching, pull requests, and issue tracking.

GitHub's version control system allowed structured development and historical tracking of changes. The codebase was maintained stable, with features added

systematically. GitHub's issue tracking also enhanced the Agile process, providing visibility into the project's progress.

Chapter 2

Methodology

2.1 Introduction

2.1.1 Overview

Software development methodologies profoundly influence the progression and success of a project. In this dissertation, I explore the use of the Agile methodology, known for its flexible, iterative approach that enables incremental development and responsiveness to change.

2.1.2 Rationale for Adopting the Agile Methodology

The Agile methodology was the approach that was truly embraced during development. Agile was selected for its adaptability and its focus on client feedback and iterative progress. The application's development necessitated rapid iterations, with features being refined through continuous user feedback and evolving requirements.

2.1.3 Implementation of the Agile Methodology

The application's development journey was marked by short sprints and frequent reassessment of project priorities. This approach allowed for a dynamic evolution of features, with user authentication not as a single foundational phase but as a component that received continuous enhancements throughout the project lifecycle.

Each iteration included the development, testing, and release of a new piece of functionality, starting with basic user authentication and progressively including more complex features like feed posting, direct messaging, run tracking, and group

management. This iterative process was crucial for integrating user feedback and ensuring that the application was consistently aligned with user needs.

2.2 Requirement Analysis

2.2.1 Requirement Gathering and Analysis

The characteristics of the social fitness application were conceptualized based on an intimate knowledge of the technological disarray that exists in the sports world. As a participant in the sports sector, I noticed a common trend: the dependence on various dissimilar platforms, such as Google Drive for document and video storage, Strava for workout logging, and WhatsApp for communication. Users had to sift through numerous apps and services to accomplish their sports-related tasks, which made for a difficult experience.

Seeing a need in the industry for a single platform that included these features, I started a requirements-gathering process to help direct the application's development. The goal was to combine the many components of data, activity tracking, and social engagement.

2.2.2 Enhancing Engagement through Integration

A keen observation of the technology use patterns in my sports group greatly influenced the first step of requirement collection. Observing a pattern of uneven interaction with the various platforms at hand, I identified a disarray in the team members' digital encounters. It became clear that the majority of participants were only using one or two of the technologies at a time, consistently ignoring the others. This selective involvement brought to light a critical weakness in the current digital ecosystem: the lack of a centralized, integrated platform capable of capturing the team's full engagement.

One of the main requirements for the social fitness application was to develop a comprehensive platform that would combine all of the different capabilities into a single, easy-to-use interface. Reducing the cognitive load that comes with navigating between apps was the goal in order to increase user engagement and involvement. The website was designed to function as a one-stop shop for communication, fitness tracking, and data storage, promoting more regular and thorough use by all team members.

2.2.3 Understanding Team Dynamics and Technology Use

Examining the subtleties of how and why particular technologies were disregarded was part of a deeper investigation. For example, a better understanding was sought for why people preferred to message on WhatsApp rather than report their activity on Strava, or why people often forgot to share documents on Google Drive. Team member interviews were conducted as part of this investigation to provide qualitative information about their preferences, driving forces, and challenges when utilizing various platforms.

2.2.4 Designing for Increased User Engagement

The information gathered suggested that, in the context of sports, people are looking for digital interactions that are quick and easy to use. Consequently, the application was designed to not only incorporate the features of current platforms but also to improve them through an engaging design. To lower barriers to technology use, features including tailored notifications, streamlined workflows, and straightforward navigation were given top priority.

2.2.5 Supporting Team Management

Additionally, the program was designed to give managers a thorough picture of their team. Managers could obtain information on group and individual activities, communication styles, and document access through an integrated dashboard, providing them with the tools necessary to create a responsive and cohesive team atmosphere. Managers could monitor progress, plan tasks, and attend to team requirements more efficiently if all pertinent information was available in one place.

2.2.6 Visualizing Requirements through Sketches

Initial sketches were instrumental in visualizing the user flow and interface design. They served as a bridge between the conceptual understanding of user requirements and the tangible design elements of the application.

Figure 2.1 illustrates the early thought process behind the user interface layout, mapping out how users would interact with the core features such as group chats, posts, and activity logs.

Then from this stemmed the real layout for my app

To summarize, the process of demand collecting highlighted the need for a single platform that would not only duplicate the features of the many technologies in use, but also improve them by delivering a more cohesive and interesting user experience. The goal of this all-encompassing strategy was to give management the

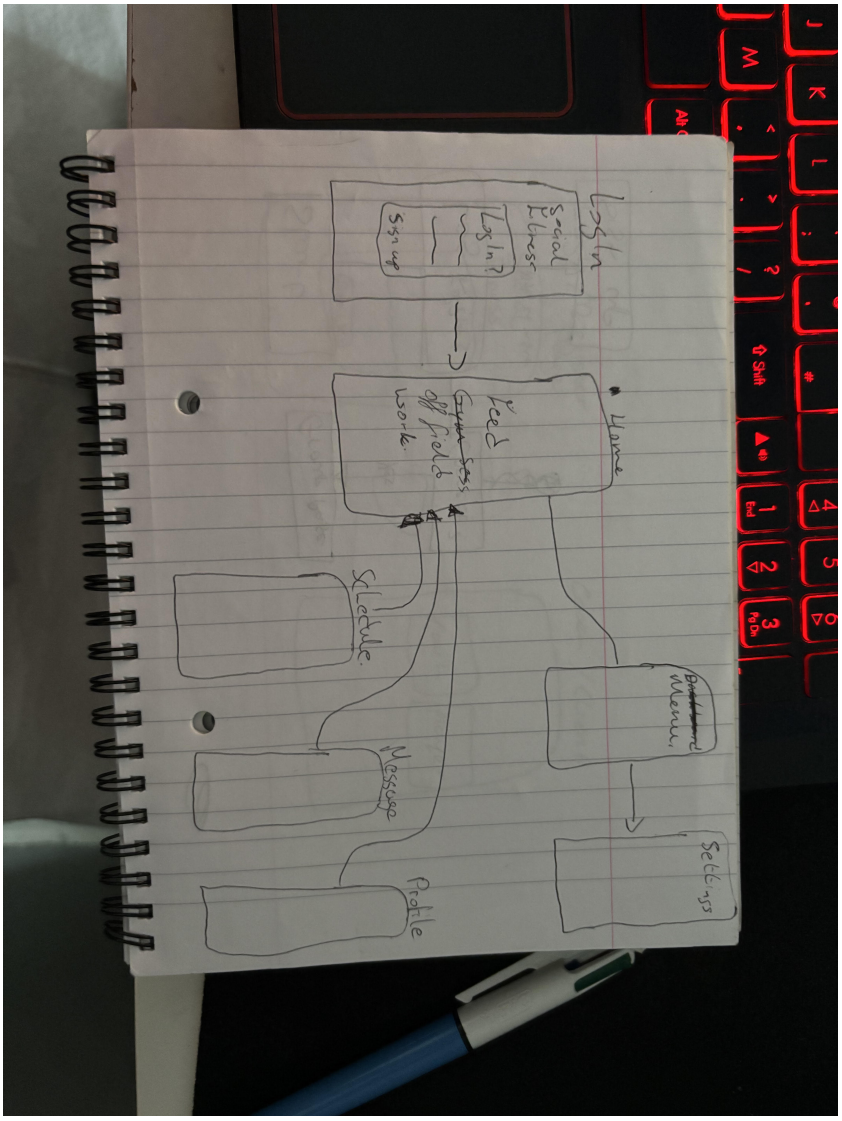

Figure 2.1: Initial hand-drawn sketch of the application's layout, demonstrating the user flow between different functionalities.

resources they needed for effective team management while also enabling inclusive digital participation among team members.

2.3 Development Phase

The Waterfall methodology's sequential steps were closely followed during the development of the social fitness application, guaranteeing a clear progression from

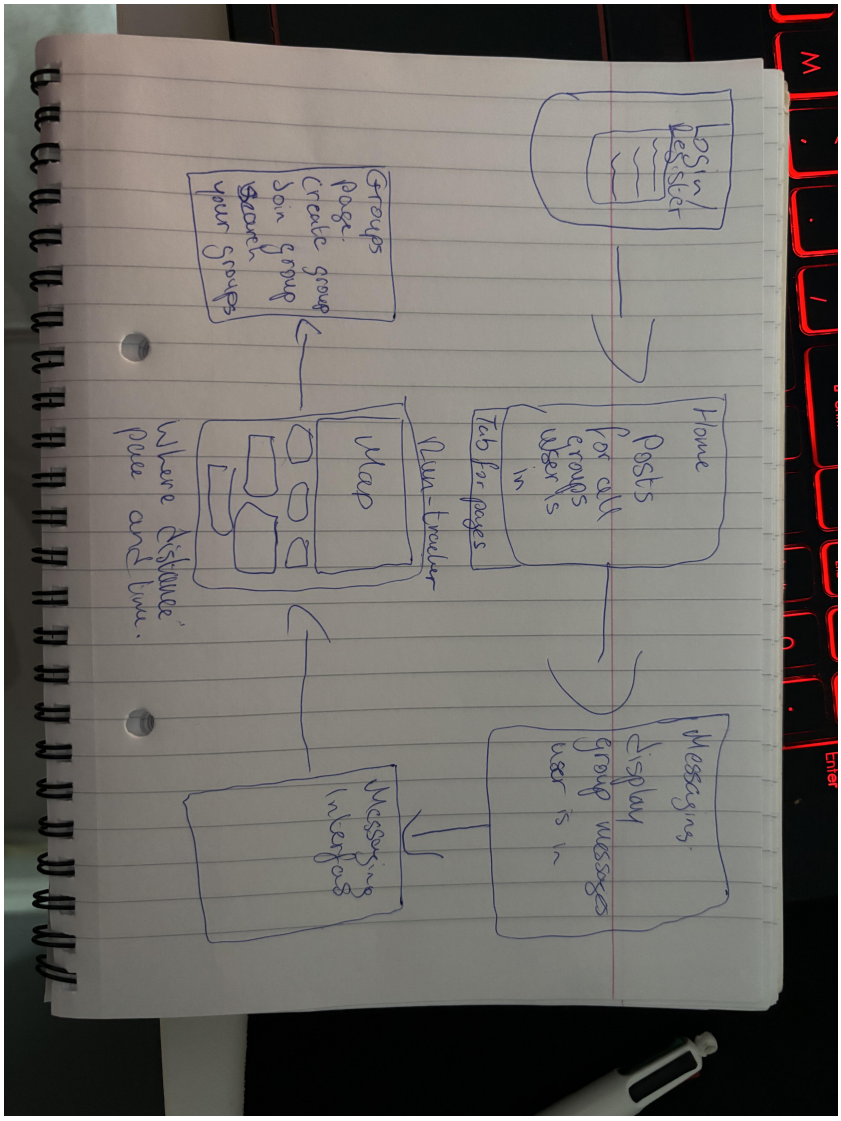

Figure 2.2: Final draft of layout followed for App.

concept to launch.

2.3.1 Phase 1: User Authentication

Creating a safe user authentication system was the main goal of the first phase. Given the importance of data security and tailored user experiences, this foundation was essential. Passwords were handled securely, user sessions were managed, and future feature extensions requiring user identity were made possible via the authentication procedure.

2.3.2 Phase 2: Creating the Feed

Once user authentication was robust and reliable, the next logical step was to implement the social feature at the heart of the application—the feed. The feed allowed users to post content visible to their network, akin to the core functionality of popular social media platforms. It was essential that this feature allowed for personalization and interaction, setting the stage for more complex features.

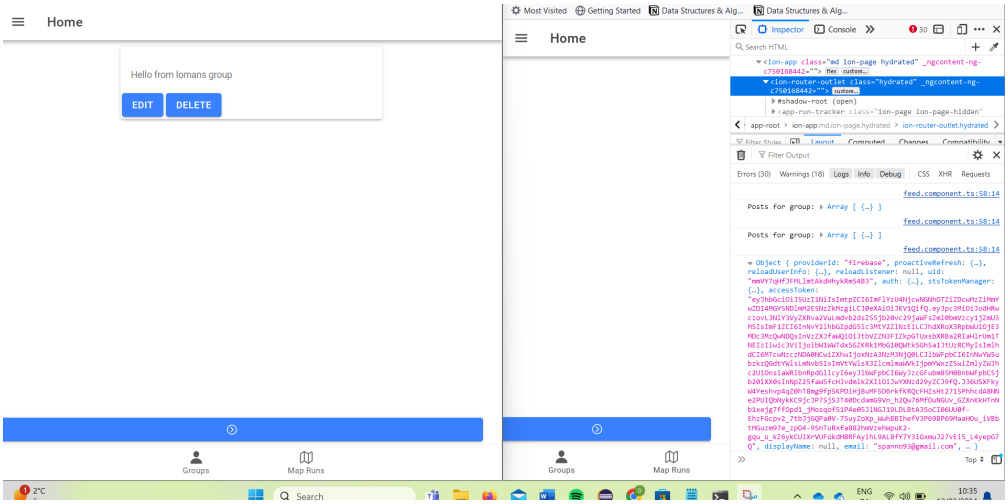

Figure 2.3: Posts for groups

2.3.3 Phase 3: Messaging Functionality

Following the feed, a one-on-one messaging system was developed, allowing for direct communication between users. This added a private layer of interaction, critical for arranging meetups or sharing personal health and fitness tips.

2.3.4 Phase 4: Run Tracker Integration

Integrating the Google Maps API, the run tracker was developed to allow users to record their routes. This feature was a technical milestone, as it involved real-time data processing and geolocation services, showcasing the application's ability to handle dynamic, complex tasks.

2.3.5 Phase 5: Group Functionality

The idea of groups was presented in the next stage. Users could take part in group chats, upload information to a group feed, and join particular groups. This

Figure 2.4: group chat and user messaging

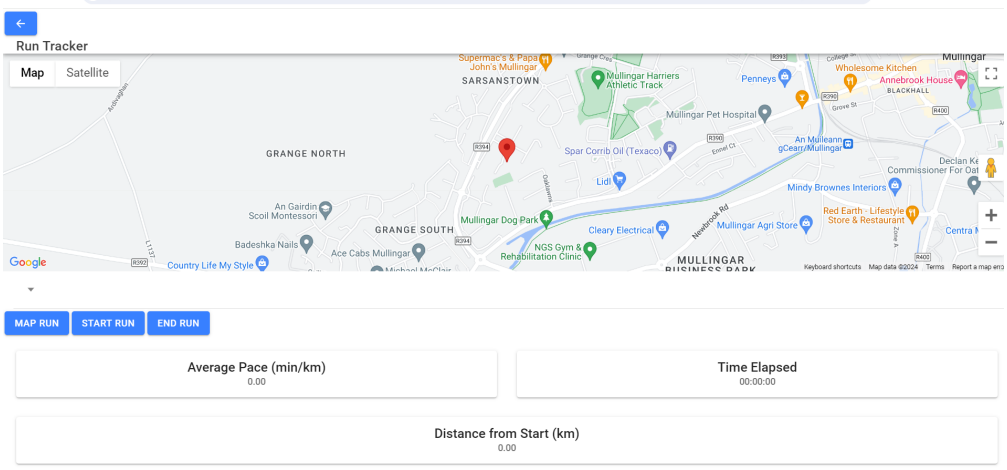

Figure 2.5: Run Tracker

simulated the social component of fitness, where teams can work out together and discuss their results. In order to support multiple, simultaneous group interactions, database administration and user interface design had to be carefully considered during the creation of this functionality.

2.3.6 Phase 6: File Sharing Capabilities

Then, a group file-sharing function was included. Only members of the relevant group could upload and view files, such as diet plans or training manuals. To make sure that only members of the approved group could access this, extra security

measures were needed.

2.3.7 Phase 7: Gym Session Recorder

The inclusion of Google Fit API for recording gym sessions was the last update added to the app. Users can now automatically import and log all of their gym activity from Google Fit, including sets, reps, workouts, and cardiovascular data. With the help of this integration, users can easily combine their workout data and get a complete picture of their fitness activities and advancement.

2.3.8 Coding Standards and Version Control

Throughout the development process, coding standards were rigorously maintained to ensure readability and maintainability of the codebase. Version control was managed through Git, with regular commits and branches used to track changes and manage the development of different features simultaneously.

2.3.9 Integration of Technologies

Several technologies and APIs have to be integrated throughout the development of the application. Firebase supplied a scalable backend solution, and Angular gave the foundation for creating the user interface. The run tracker was integrated with the Google Maps API, and native device functions were accessed through the use of a capacitor. Every technology was chosen based on its track record of dependability, broad community support, and alignment with the application's overall architecture.

2.4 Testing

2.4.1 Testing and Validation

The social fitness application's testing technique used a practical, manual approach that was in line with the phases of the Waterfall model. The usage of specialized testing software or frameworks was decided against due to the nature of the project and resource constraints. Rather, the emphasis was on practical, iterative testing carried out at the end of every stage of development.

2.4.2 Manual Testing Approach

Every feature, from user login to the gym session recorder's final implementation, went through a laborious manual testing procedure. A variety of anticipated use

scenarios were covered by the tests, which were created to mimic user interactions. This practical approach made it possible to evaluate the functionality of the program quickly and easily. A feature was deemed stable and I moved on to the next development step if it performed as anticipated and satisfied the specifications listed in the earlier phases.

2.4.3 Issue Resolution

In instances where testing revealed issues, I prioritized addressing them promptly before moving forward. The resolution process involved debugging the code, refining the logic, and re-assessing the feature to ensure that the functionality was both correct and user-friendly. This iterative process of testing and fixing was repeated until the feature functioned seamlessly.

2.4.4 Limitations and Risk Management

It was accepted that there were drawbacks to not using automated testing or other software. Thorough documentation, meticulous test planning, and numerous test cycles helped to manage the risks associated with this method, which included the possibility of missing edge cases or the decreased effectiveness of manual testing.

Chapter 3

Technology Review

3.1 Front-End

3.1.1 Ionic/Angular

When combined with Angular, Ionic offers developers a strong choice when it comes to building cross-platform web applications. This combo makes use of Ionic's extensive collection of pre-styled components and plugins, which speeds up the creation of mobile apps that look native from a single codebase, and Angular's potent features for creating single-page applications (SPAs). By simplifying the learning curve and development process, the usage of web technologies (HTML, CSS, JavaScript) also makes it easier for developers who are already experienced with web development to migrate. [1]

3.1.2 React Native

React is a declarative, effective, and adaptable JavaScript user interface library created by Facebook (now Meta). With it, programmers can build sizable online apps that have the ability to update data without refreshing the page. React's primary objectives are speed, scalability, and simplicity. It is limited to the application's user interfaces, which match the MVC template's view. This makes sense for single-page applications (SPAs), where user interaction and responsiveness are crucial.

The Social Fitness App project can benefit from various advantages provided by React's architecture and capabilities.

- Component-Based Architecture: React's component-based architecture enables code that is modular and reusable. Because this architecture allows

developers to create sophisticated user interfaces (UIs) from enclosed information that maintains its state, it streamlines the development process.

- JSX: React leverages JSX, a JavaScript syntactic extension that lets you express HTML structures alongside JavaScript code in the same file. This facilitates debugging and understanding of the code.

- Virtual DOM: A virtual version of a user interface is stored in memory and synced with the actual DOM according to React's Virtual DOM programming approach.

React Native expands React's functionality to mobile app development, enabling developers to create mobile apps that are identical to those created using Objective-C, Swift, or Java, even though React is primarily used to construct web applications.

Cross-Platform Development

- JavaScript and React may be used to create mobile apps with React Native. It has a similar design to React, allowing you to create a rich mobile user interface using declarative components and a component-based UI architecture that can compile into native app components.

Performance

- Even though React Native provides a nearly native user experience, for more demanding applications, its performance may still fall short of that of completely native programs. Web applications using React can have incredibly responsive user interfaces.

Developer Experience

- React and React Native both have a hot reload functionality that lets developers inspect the most recent changes instantly without losing application state, which increases developer productivity.

3.1.3 Vue.js with Vuetify

Overview

Vue.js, renowned for its progressiveness and simplicity, has an easy-to-learn learning curve and can be used for both basic and sophisticated applications. By

utilizing Vue.js, Vuetify offers a collection of user interface components that follow Material Design guidelines, guaranteeing a unified and user-friendly design throughout online apps. This combination is especially useful for projects that value simplicity of development and visual attractiveness over the need for a large infrastructure. [2]

Rationale

The decision to utilize Vue.js with Vuetify could be driven by several factors:

- **Simplicity and Developer Experience:** Because of its basic design, Vue.js is easy to integrate and scale with existing projects. It is intended to be adopted gradually. Material Design: Vuetify provides pre-made elements that adhere to Material Design principles, guaranteeing a contemporary and unified appearance throughout your online application.

- **Performance:** Vue.js is lightweight and optimized for performance, providing faster load times and a smooth user experience.

- **Community and Ecosystem:** Vue.js has a vibrant community and ecosystem, offering extensive resources, plugins, and tools that facilitate development.

- **Target Platform Agnosticism:** Building hybrid mobile applications with Ionic/Angular is quite effective, however Vue.js and Vuetify does not give preference to any one platform over another. When it comes to web app development, this framework-UI library combination shines, providing more versatility for projects that do not just target mobile devices.

- **Design Philosophy:** Ionic provides a comprehensive suite of pre-styled components designed for mobile interfaces, potentially limiting when aiming for a unique web-centric design. Vuetify's adherence to Material Design principles offers a balance between uniformity and customizability, suitable for crafting distinctive web experiences.

- **Development Approach:** Ionic/Angular necessitates dealing with Angular's learning curve and its specific way of managing state and reactivity. In contrast, Vue.js's reactive and composable architecture might align better with web developers seeking a more straightforward or flexible approach.

3.1.4 Flutter

Overview

Google's Flutter is a feature-rich UI toolkit that allows developers to create natively built desktop, web, and mobile applications from a single codebase. The programming language it employs, Dart, is designed to create quick programs for any platform. Flutter's methodology involves compiling to native machine code, enabling superior performance and a wide range of pre-designed widgets that closely resemble native components.[3]

Rationale

Choosing Flutter for a project like the Social Fitness App can be justified for several reasons:

- **Performance:** Flutter compiles to native code, which helps achieve performance indistinguishable from native apps on iOS and Android.
- **UI Flexibility and Customization:**Flutter provides a rich set of widgets and a powerful rendering engine, enabling the creation of custom, beautiful, and brand-driven designs not limited by the constraints of the platform.
- **Developer Productivity:** Flutter's hot reload feature allows developers to see changes almost instantly, without losing state, which significantly speeds up the development cycle.
- **Unified Codebase:** The ability to use a single codebase for apps that run on multiple platforms can greatly reduce development time and cost.
- **Native Performance:** Unlike Ionic/Angular, which primarily relies on web technologies and may require plugins or specific adjustments to achieve native-like performance, Flutter is designed from the ground up to compile to native code. This fundamental difference means Flutter can potentially offer better performance and smoother animations, especially for more complex applications.
- **UI Design and Customization** While Ionic/Angular provides a wide range of pre-styled components that adhere to Material Design and iOS design guidelines, Flutter offers an extensive catalog of customizable widgets. This allows for more detailed control over the app's appearance and the ability to create highly customized, unique user interfaces.

- **Learning Curve and Development Language:** Developing with Ionic/Angular involves understanding web technologies and Angular, which might be more accessible to web developers. Flutter requires learning Dart, which, although straightforward for those familiar with object-oriented programming languages, is a shift from traditional web development languages.

3.2 Backend

3.2.1 Firebase

3.2.2 Overview

A full toolkit of features provided by Firebase facilitates the quick building of online and mobile applications. For applications that need real-time data updates, its real-time database and Firestore offer scalable cloud databases for storing and synchronizing data between users. Because of its serverless architecture, Firebase Authentication facilitates scalable app development without requiring infrastructure maintenance, while also streamlining user management and security. For developers who want to focus on improving user experience while speeding up development, Firebase is an appealing choice because of its simplicity of use and seamless connection with other Google services. [4]

3.2.3 AWS Amplify

3.2.4 Overview

AWS Amplify is a development platform provided by Amazon Web Services (AWS) to build secure, scalable mobile and web applications quickly and easily. It encompasses a wide range of services and features, including authentication, API, storage, and machine learning capabilities, designed to work seamlessly with React, Angular, Vue, Next.js, and mobile platforms like iOS and Android. [5]

While AWS Amplify offers extensive functionalities and deep integration with the AWS ecosystem, the choice to opt for Firebase for the Social Fitness App could be based on several considerations:

- Ease of Use and Setup: Firebase offers a more straightforward setup and management experience, which can be particularly advantageous for developers seeking rapid development and deployment without the complexity of configuring AWS services.

- Real-time Database and Synchronization: Firebase's real-time database provides out-of-the-box support for real-time data syncing across user devices,

making it an attractive choice for applications requiring instant data updates, such as a social fitness app.

- Learning Curve: Firebase provides a less steep learning curve compared to AWS Amplify, especially for developers who may not be as familiar with cloud infrastructure or the broader AWS service ecosystem.

Scalability and Reliability

- AWS Amplify benefits from the robustness of AWS infrastructure, offering superior scalability and reliability for applications expected to scale significantly. This can be an essential factor for enterprise-level applications or services anticipating rapid user growth.

Customization and Service Integration

- AWS Amplify provides more extensive customization options and the ability to integrate a broader range of AWS services directly into your app. This can be particularly valuable for apps requiring specific AWS services such as Amazon SageMaker for machine learning or Amazon Lex for chatbots.

Cost and Pricing Structure

- The cost implications of using AWS Amplify versus Firebase vary based on the app's specific requirements and usage patterns. AWS Amplify might offer more cost-effective solutions for certain use cases, especially where specific AWS resource consumption is optimized.

3.2.5 Node.js with Express.js

Node.js, in combination with Express.js, represents a powerful duo for developing server-side applications. Node.js is a runtime environment that allows you to execute JavaScript on the server side, breaking the traditional confines of JavaScript to the browser. Express.js is a framework built on top of Node.js that simplifies the process of building server-side applications, APIs, and web services. [6]

Opting for Firebase instead of a Node.js and Express.js backend for the Social Fitness App might stem from several factors:

Development Speed and Simplicity

- Firebase offers a wide array of integrated services like hosting, real-time databases, authentication, and more out of the box. This can significantly

accelerate development timelines compared to setting up these services manually with Node.js and Express.js.

Serverless Architecture

- Firebase's serverless infrastructure removes the need for server management, scaling, and maintenance, which could be particularly beneficial for teams with limited backend expertise or resources.

Real-time Data Synchronization

- Firebase provides real-time databases that sync data across all clients in real-time. While real-time features can be implemented with Node.js and Express.js, doing so requires additional setup and integration with other services or libraries.

Flexibility and Control

- Node.js with Express.js offers more flexibility and control over the server-side logic and architecture of your application. This can be advantageous for apps with unique backend requirements or when integrating with systems that necessitate custom server-side logic.

Ecosystem and Community

- Both Firebase and Node.js have strong communities and ecosystems. However, Node.js, being older and widely adopted in various types of projects, has a vast array of libraries and tools developed by the community.

3.3 Mapping Services

3.3.1 Google Maps API

The Google Maps API is a powerful tool provided by Google that allows developers to include maps within webpages, retrieve data from Google Maps, and manipulate maps to fit the context of their applications. It offers a wide range of functionalities including map embedding, street view, geolocation, routes, and places, making it indispensable for applications requiring detailed geographic data and interactive mapping capabilities. [7]

The selection of the Google Maps API for the Social Fitness App is influenced by several compelling factors:

- Comprehensive Data and Coverage: Google Maps provides extensive geographic data, high-quality satellite imagery, and detailed street maps, ensuring users have access to accurate and up-to-date information.
- Rich Set of Features: From displaying a simple map with a pin to complex route calculations and place searches, the Google Maps API supports a broad spectrum of mapping needs, enhancing the user experience by providing contextual geographic information.
- Customization and Interactivity: The API allows for high levels of customization, enabling the app to integrate maps that align with the look and feel of the app, as well as interactive elements like markers and info windows that engage users.

Brand Trust and Reliability

- Google Maps is a well-established service known for its reliability and accuracy, instilling trust in both developers and users.

Ease of Use

- The Google Maps API is designed to be developer-friendly, with comprehensive documentation and community support that facilitate the integration process.

Advanced Features and Analytics

- Google's platform provides advanced features such as real-time traffic data, detailed place information, and analytics capabilities that can be crucial for applications relying on geographic data.

3.3.2 MapBox

Mapbox is an advanced location data platform that provides developers with the tools to create customized, interactive map experiences. Unlike traditional map services, Mapbox emphasizes extensive customization options, allowing for unique map designs that can closely align with an app's aesthetic and functionality requirements. It's particularly known for its powerful geospatial analysis tools, dynamic map rendering, and comprehensive documentation that supports both web and mobile applications. [8]

Choosing Mapbox for a project like the Social Fitness App could be driven by several motivations:

- Highly Customizable Maps: Mapbox stands out for its deep customization capabilities, enabling the creation of maps that not only function well but also blend seamlessly with the app's design.

- Rich Geospatial Tools: For applications requiring detailed geographic analyses or sophisticated routing algorithms, Mapbox offers a suite of tools and APIs that cater to complex geospatial queries.

- Performance and Efficiency: Mapbox maps are optimized for performance, ensuring smooth interactions and fast loading times even when displaying data-intensive maps or operating in areas with limited connectivity.

Customization and Style

- While the Google Maps API offers some level of customization, Mapbox provides far more control over map aesthetics and interactions. This allows developers to create more brand-aligned and unique map experiences in their applications.

Data and Functionality

- Google Maps API benefits from Google's extensive data repository, including detailed place information and street views. Mapbox, however, offers powerful geospatial analysis tools that might be more suited for applications requiring advanced mapping functionalities.

Pricing and Usage Limits

- Both platforms have different pricing structures and usage limits. Mapbox's pricing is based on map loads and the number of API requests, which can be more cost-effective for certain use cases compared to Google Maps' pricing model, which also considers similar factors but may have higher costs for high-usage applications.

Community and Support

- Google Maps has a long-standing presence and a vast user base, which translates into widespread community support. Mapbox, while younger, has quickly built a robust community around its open-source libraries and APIs, offering extensive resources and active forums for developer support.

3.4 Technology Used in the Social Fitness App

The convergence of multiple potent technologies has transformed the digital transformation of mobile applications. Among these, Firebase offers a reliable backend service, Ionic and Angular have become top front-end development frameworks, and Google Maps API makes it possible to integrate intricate geographic mapping services.

Firebase's real-time database and Angular's abundance of features are combined by Ionic, a framework for developing cross-platform mobile applications using web technologies, to produce dynamic and engaging applications. With the help of this combo, developers may create top-notch native applications for the iOS and Android platforms using a single codebase. [9].

The seamless experience of real-time syncing with a NoSQL database and instantaneous data updates is made possible by integrating Firebase with Ionic/Angular apps. The integration process include not only data storage and retrieval but also user authentication, security rule configuration, and server-less architecture implementation, which improves scalability and performance. [10].

Furthermore, by enabling the incorporation of location-based services and map-related features, mobile applications' usefulness is expanded through the integration of the Google Maps API. For applications that require spatial awareness and map interaction, the Google Maps API integration is essential, since it allows for the placement of markers, the plotting of routes, navigation, and geographical insights. (Community DEV) [9].

Although integrating these technology has many advantages, there are drawbacks as well. Taking care of platform-specific quirks and managing API keys are two major challenges. Making sure the integration does not negatively impact the application's speed is another, particularly when managing sizable datasets or intricate map-related computations [10].

Despite these difficulties, developers may design complex, feature-rich, scalable, maintainable, and highly functional mobile applications with the help of Ionic/Angular's interface with Firebase and the Google Maps API.

3.4.1 Firebase

Firebase Authentication

Any application involving real-time interactions and confidential user data must have a strong user authentication system. Because of its extensive feature set and ability to facilitate safe user sign-up and sign-in procedures, Firebase Authentication was selected. With features like "createUserWithEmailAndPassword" for new user registration and "signInWithEmailAndPassword" for user login, this service

makes handling user credentials and session management easier. The application's flow incorporates Firebase Authentication to guarantee that users must first authenticate in order to use the app's functionalities, hence enhancing user security.

The app leverages Firebase Authentication's authState listener to maintain real-time session states, which ensures that the app's state is synchronized with the user's authentication status. This seamless integration contributes to a responsive and intuitive user interface, preventing access to features by unauthenticated users and allowing for immediate session restoration upon app restarts.

Firebase Realtime Database and Firestore

Firestore serves as the application's backbone for real-time database operations. The flexible NoSQL data structures and its use of data synchronization instead of typical HTTP requests: every time data changes, all connected devices receive that update within milliseconds. The persistent connection maintained between the client and server means that an application can both listen for changes from the server and write data to the database, which will be instantly synchronized across all devices. This allows for the storage of complex entities such as user profiles, workout logs, and group messages. Utilizing Firestore's real-time update and synchronization capabilities, the app delivers live messaging functionality. The addDoc, query, and real-time data streaming via listeners enable the instantaneous delivery and reception of messages within the app's group communication feature.

The sendMessage function in the message.service.ts showcases Firestore's ability to handle real-time communication effectively. Messages are appended with a server-generated timestamp before being stored in a 'conversations' collection, which ensures their chronological ordering in the chat interface. The app's data modeling is designed to support a dynamic and responsive user interface where data changes in Firestore are immediately propagated to all active users, thus promoting active and engaging communication among team members.

One of the most compelling features of the Realtime Database is its ability to function seamlessly, even when an app goes offline. The database SDKs cache the data on the client, which allows for read and write operations to continue working despite the lack of an internet connection. When connectivity is re-established, the SDK synchronizes the local changes with the remote database.

Cloud Firestore is another NoSQL database option within Firebase, designed to provide more robust scaling than the Realtime Database. It supports more complex queries and a hierarchical data structure, making it better suited for larger applications.

Challenges and Solutions

Adopting Firebase, while streamlining development, brought forth challenges such as handling real-time data across various user states and ensuring data consistency. To overcome these, the app employs Firebase's offline capabilities, which allow workout logs and messages to be queued when the device is offline and synchronized upon reconnection, providing an uninterrupted user experience. Additionally, data structuring and indexing strategies were employed to optimize Firestore queries for performance and scalability, ensuring that the app remains responsive as user engagement grows.

Security and Privacy Considerations

Data security and user privacy are at the forefront of the app's design. Firebase Security Rules have been carefully crafted to protect user data, allowing access only to authenticated users and preventing unauthorized data manipulation. These rules are consistently reviewed and updated in line with the best practices for data security, ensuring a trusted environment for users to interact and share their workout data.

Performance and Scalability

Performance testing has revealed Firebase's ability to effectively handle the load generated by the app's real-time communication and data management features. Its scalable infrastructure has been pivotal in supporting an increasing number of users while maintaining high responsiveness. The managed services provided by Firebase negate the need for manual scaling, allowing the app to automatically adjust resources to match the current demand.

Cloud Functions

Firebase Cloud Functions allows you to run backend code in response to events triggered by Firebase features and HTTPS requests. This serverless architecture means you only need to write your code, and Firebase takes care of deploying and running it at scale.

3.4.2 Ionic/Angular

Framework Selection and Rationale

The decision to employ Ionic/Angular as the primary framework for our collaborative fitness application was informed by its comprehensive support for cross-

platform mobile application development. This choice is underscored by the application's need for a responsive, intuitive user interface that can operate seamlessly across different devices and platforms. Ionic/Angular's integration facilitates this through its rich set of UI components and Angular's robust data management capabilities, making it an ideal choice for developing sophisticated single-page applications (SPAs) like ours.

Component-Based Architecture

This application architecture benefits significantly from Angular's component-based structure, allowing for modular development and enhancing code reusability. This is exemplified in the app's organization into distinct components and services, such as the authentication.service.ts for managing user authentication and the message.service.ts for handling real-time messaging. Each component and service is designed to fulfill specific app functionalities, demonstrating the framework's facilitation of a clean, organized codebase that supports scalable application development.

Reactive Programming with RxJS

Angular's adoption of reactive programming principles, particularly through the use of RxJS, is crucial for managing the app's dynamic data streams, this can be seen through such operators as 'map', 'catchError' and 'switch map'. With this, a powerful way for handling event-driven data is achieved within the application. This approach is evident in the message.service.ts, where observables are used to handle the real-time update of messages within the group chat feature. By employing RxJS observables, our app efficiently manages asynchronous data, such as user messages and workout updates, ensuring that the UI is consistently in sync with the backend data stored in Firebase.

Routing and Navigation

The application leverages Angular's routing module to manage navigation across different views, enhancing the user experience through smooth transitions and maintaining the application state. The implementation of Ionic's navigation stack, detailed within the app-routing.module.ts, showcases how routes are configured to facilitate seamless navigation among the app's various pages, including the home screen, workout logs, and group chats. This SPA approach mimics native app navigation patterns, contributing significantly to the user-friendly design of our application.

Performance Optimization and Cross-Platform Compatibility

Optimizing for performance and ensuring cross-platform compatibility were paramount in our development process. Ionic/Angular's capabilities in this regard are showcased through the app's responsive design and efficient operation across both iOS and Android platforms. Techniques such as lazy loading, implemented in the app.module.ts and specific feature modules, have been crucial in reducing the initial load time and improving the app's overall performance.

Challenges and Solutions

Throughout the development, we encountered challenges related to optimizing the app's performance and ensuring a consistent user experience across different platforms. These were addressed by leveraging Ionic's adaptive styling and Angular's device detection capabilities, ensuring that the app not only performs well but also looks great on any device. The component-based development approach further allowed for targeted optimizations and adjustments, ensuring that the app remains responsive and engaging for all users.

Conclusion

The utilization of Ionic/Angular as the development framework has been instrumental in the success of our collaborative fitness application. It provided the tools necessary for creating a feature-rich, cross-platform app that effectively meets the demands of real-time communication and community engagement within the fitness domain. The chosen architectural and development strategies have resulted in a scalable, maintainable, and user-friendly application, poised for future expansion and enhancement.

3.4.3 Capacitor in Ionic Framework

Overview and Role in Cross-Platform Capability

Capacitor is an open-source app runtime that allows Ionic apps to run natively on iOS, Android, and the web. It serves as a bridge for your web app to access native device functionalities, which is vital for creating a native-like experience on different platforms. Detailing Capacitor's role in your project underscores your application's reach and usability across diverse devices. [11]

Integration with Camera Functionality

The Social Fitness app, while comprehensive, has room for expansion. The inclusion of direct camera access via Capacitor is a feature that was thoroughly

investigated during the technology review phase. Although not currently integrated, it represents a future direction to enhance user interaction within the app. This, alongside other potential integrations, aligns with the goal of providing an immersive and community-driven user experience, underscoring the ongoing commitment to app evolution.[12] [13]

In the context of the Social Fitness App, Capacitor plays a crucial role in bundling the web application into a native container, thereby enabling cross-platform compatibility. As an evolution of Cordova's approach, Capacitor creates a native runtime that allows the web app to run on iOS, Android, and as a Progressive Web App (PWA) with access to the full native SDK on each platform. This approach streamlines the development process by allowing a single codebase to deploy across multiple platforms, negating the need for platform-specific codebases and significantly reducing development time and resources. [14] [15]

Capacitor achieves this by providing a consistent API that works across all supported platforms, abstracting the complexities of native code translation. It integrates directly with the native project build tooling, such as Xcode for iOS and Android Studio for Android, which automates the process of creating and managing native app builds. The ability to use native IDEs provides developers with powerful tools for debugging and optimizing their applications. [16] [17]

Furthermore, Capacitor's plugin system supports both custom in-house plugins and community plugins. This flexibility ensures that the app can leverage native device features beyond the core Capacitor APIs, such as camera access, geolocation services, and more, ensuring a native-like user experience. [18] [19]

By enabling web technology-based apps to run across multiple platforms with native performance and capabilities, Capacitor has become an indispensable tool in the hybrid app development ecosystem. Its compatibility with popular web frameworks like Angular, React, and Vue.js makes it a versatile and future-proof choice for the development of the Social Fitness App. [20] [21]

3.4.4 Google Maps API

This API is seen to be crucial in the development in this app especially is to integrate the option for users to record a run.

Integration and Functionalities

Google Maps API provides a diverse set of mapping and location services that can enrich mobile and web applications. The API's extensive features, including map display, geocoding, routing, and places information, were integrated into our fitness application to provide users with an immersive and interactive experience. Specifically, the API's robust mapping capabilities allow users to visualize their

workout routes and locations, offering a more engaging fitness tracking experience [22].

The robust mapping capabilities of the API allow for real-time visualization of user workout routes, crucial for the application's run recording feature.

Enhancing User Experience with Geolocation and Mapping

The use of geolocation through Google Maps API enables the app to capture the user's workout routes accurately. Functions from the API, such as getCurrentPosition and watchPosition, are utilized to track and update the user's location in real-time during outdoor workout sessions. The application leverages the Polyline object to draw routes on the map, providing visual feedback of the workout path taken.

The Google Maps API enhances the app with location-based services. Users can utilize the Places library to discover nearby health-related facilities and use the Geocoder service to translate geocoordinates into readable addresses, enabling them to log and share workout locations with ease.

Location-Based Features and Services

Beyond route tracking, the Google Maps API contributes to location-based services within the app. By utilizing the Places library, users can search for nearby gyms, health food stores, or outdoor exercise areas. The API's Geocoder service is used to convert latitude and longitude coordinates into human-readable addresses, making it easy for users to log and share the locations of their workouts [23].

Customization and Interactivity

Custom markers, info windows, and the ability to adjust the map's styles are among the features that provide a customized mapping experience tailored to the fitness context. For instance, the app customizes the map's color scheme to match the user's time of day, enhancing usability and aesthetic appeal. User interactivity is facilitated by enabling touch gestures for map exploration and interactions with points of interest [24] .

Challenges and Solutions

Incorporating mapping features required addressing privacy concerns and ensuring the efficient use of location data. The app implements permission requests and user settings to manage location-sharing preferences, thereby upholding user privacy. Moreover, map data caching and efficient API call management were essential

to optimize performance and manage the costs associated with high-frequency location updates [25].

3.4.5 Google Fit Api

Introduction to Google Fit API

Google Fit is a health-tracking platform developed by Google, designed to store users' data in one secure location. It provides a centralized API to access the data related to health and wellness to create a holistic view of the user's fitness journey [26]

Conceptual Overview

The Google Fit API functions as an aggregator, collecting health data from multiple devices and applications. This unified approach allows for a comprehensive overview of a user's health metrics and physical activities

Api Capabilities

The Google Fit API provides functionalities crucial for health monitoring applications:

- Real-Time Data Sync: Ensures that the user's fitness activities are updated and available across devices
- Sensor Data Access: Gathers information from onboard sensors and connected wearables for in-depth activity tracking [27]
- Workout Recording: Offers APIs to log and store workout sessions securely in Google's cloud environment [28]

Integration with the Application

The application taps into the Google Fit API for seamless data retrieval and interaction:

- Data Retrieval: The app interfaces with Google Fit to fetch workout data, which is displayed in user-friendly formats like graphs and summaries. [29]
- User Interface: We designed intuitive screens that provide users with insights from their Google Fit data, facilitating goal tracking and historical analysis.

Privacy and Securtity

Permissions and Consent: The app follows Google's user consent process for data access, ensuring transparency and control [30]

User Benefits

By integrating with Google Fit, the app delivers substantial benefits to users, with Fitness Tracking: Enables users to monitor their progress and view detailed statistics on workouts, promoting a healthy lifestyle.

Privacy and Data Security

The integration presented challenges that were meticulously addressed:

- Data Diversity: Handling various data types from different sources required a robust data parsing and normalization strategy. [29]
- Device Compatibility: Ensuring that the app functions across multiple devices necessitated comprehensive testing and conditional logic for device-specific behaviors.

Chapter 4

System Design

The social fitness application's architecture serves as the framework for its functional features and establishes its technological organization. Several essential technologies are at the heart of this architecture, collaborating to provide a smooth user experience.

4.1 Architecture Overview

The social fitness application's advanced plan, or architecture, serves as the foundation for its many features. It sets the functional flow and technological framework. The fundamental principles of this architecture are the smooth integration of key frameworks and technologies, namely Firebase for backend services, Google Maps and Fit APIs for location and health data services, and Ionic/Angular for front-end development. Capacitor further customizes the system for an Android environment, improving the native user experience. 4.1

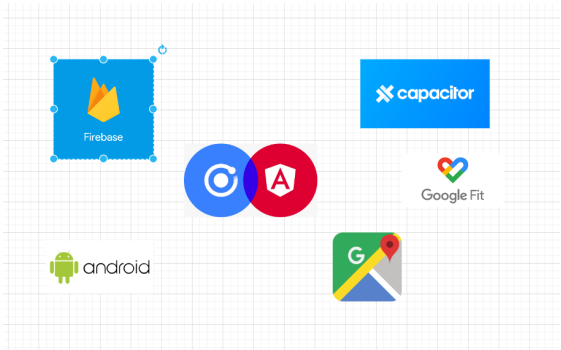

Figure 4.1: System Architecture

4.1.1 Front-end Architecture - Ionic/Angular

The front-end, which is based on the Ionic/Angular stack, provides a stable and dynamic user experience that is platform-neutral. The vast component library of Ionic combined with the robust data-binding and modular design of Angular results in an incredibly responsive and interactive user experience. Characteristics of the front-end architecture include:

- Use of Angular services for cross-component communication.
- Reactive forms for real-time form validation and state management.
- Modular design, with components and services ensuring a single responsibility principle.

4.1.2 Back-end Architecture - Firebase

Firebase stands as the back-end scaffold, a suite of cloud-powered tools that provide a spectrum of services including:

- Real-time NoSQL database for reactive data flow.
- Authentication service that secures and streamlines user access.
- Cloud functions that enable serverless computing for backend logic.
- Hosting services that guarantee fast content delivery.

This backend suite is intrinsically coupled with the front-end, providing seamless data synchronization and authentication flows.

4.1.3 Integration of Mapping and Health Api's

Mapping and health tracking functionalities are integral to the application, facilitated by Google Maps and Google Fit APIs. These APIs are meticulously integrated to provide services such as:

- Live location tracking and mapping for running routes.
- Health data management, tracking heart rates, and calories burned.
- API services that are encapsulated within Angular services, promoting reusability and testability.

4.2 Coupling of Components

The UML diagram illustrates the cohesion and coupling of the application components. Each component and service is purposefully designed to interoperate while maintaining loose coupling and high cohesion, fostering an environment conducive to scalability and maintenance. The system exhibits:

- Dependency injection within Angular services to reduce coupling.
- Services acting as intermediaries between components and Firebase, allowing for an abstraction layer that manages API calls and data handling.
- Components that are self-contained, each representing a distinct functional unit of the application.

4.3 Data Flows and Interaction Diagrams

4.3.1 Sequence Diagram

The sequence diagram provides a temporal perspective on how different parts of the system interact over time, specifically highlighting the login process, user session initialization, and the transition to the home page. This interaction is facilitated through:

- Firebase Authentication handling user Validation
- The Authentication Page orchestrating the login process and handling the resultant session token.
- The transition from the Authentication Page to the Home Page upon successful login, establishing a new user session.

Below represents a breakdown for the diagram and then the diagram represents the breakdown

- User opens the app: The action "Open App" would lead from the "User" to the "Authentication Page".
- User enters credentials and selects 'Log In': The "Authentication Page" receives "User Credentials" and on "Log In" action sends a "Validation Request" to "Firebase Authentication".

- Firebase Authentication verifies the user: "Firebase Authentication" processes the "Validation Request" and upon success, sends back a "Session Token" to the "Authentication Page".

- User's session begins, and the Home Page is displayed: The "Authentication Page" upon receiving a valid "Session Token" transitions to the "Home Page".

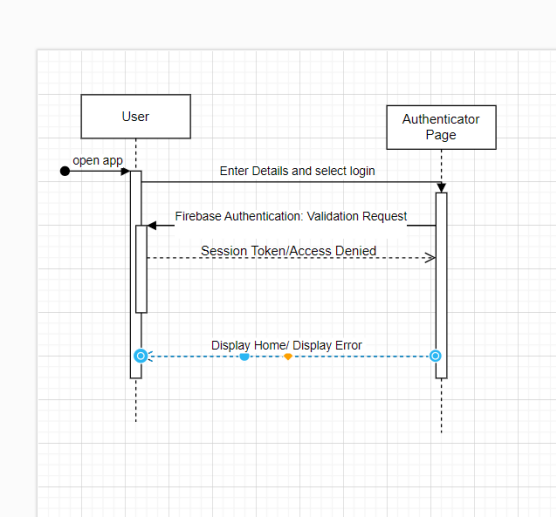

Figure 4.2: Sequence Diagram

4.3.2 Data Flow Diagram

The data flow diagram elucidates the process flow starting from a user initiating a workout on their smartwatch, logging the workout, and ending with the workout data being saved in both Google Fit and Firebase. It encapsulates:

- The bidirectional flow of data between the smartwatch and Google Fit.

- The mobile app's role in querying the Google Fit API for specific workout data.

- The app's capability to save and display the fetched data, ultimately persisting it in Firebase for long-term storage.

- User starts a workout using a smartwatch: This is the initial interaction and the starting point of your diagram.

- Smartwatch logs the workout: The workout data is recorded by the smartwatch.
- User ends the workout: The user completes the workout session.
- Workout data is saved in Google Fit: The smartwatch saves the workout data to Google Fit.
- User interacts with the app: This includes using date and time pickers to select the start and end times for querying the workout data.
- App queries Google Fit API: Based on the selected times, the app makes a request to the Google Fit API.
- Google Fit API returns data: The API returns the workout data, including heart rate and calories burned.
- App displays the data: The workout information is then displayed in the app for the user.
- User can then save workout to Firebase

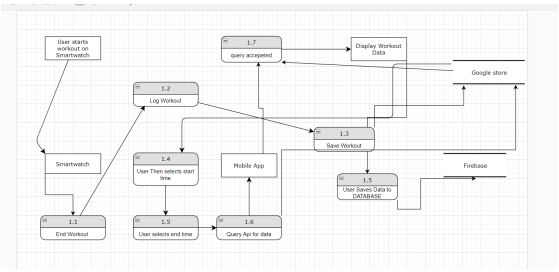

Figure 4.3: Data Flow Diagram

4.4 Integration of Core Technologies

4.4.1 Authenticator Compoenent

Overview

As the application's entrance point, the Authentication Page mediates user access and guarantees safe admission (see Figure 4.4). Users can use this website to log in, create a new account, and, if necessary, retrieve their password.

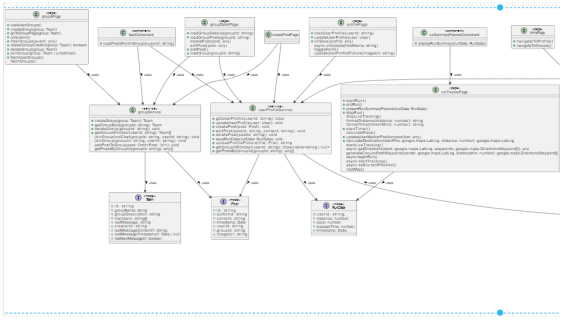

Figure 4.4: Uml1

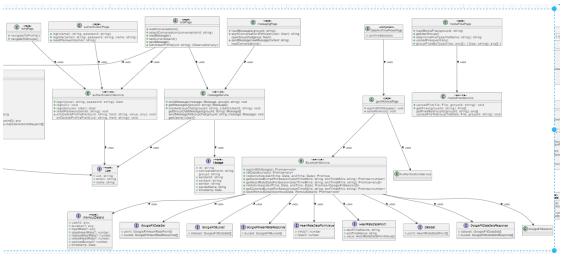

Figure 4.5: Uml2

Frontend Implementation

The form group on the page, which complies with validation requirements such minimum password length and email type, is created using Angular's reactive forms. Angular directives, such *ngIf, allow the form to be dynamically rendered according to the user's intent to register, log in, or change their password. Ionic components are wrapped with the ion-content tag to guarantee a native appearance and feel across all platforms.

To provide fields for user input, the ion-input elements within ion-item tags, for instance, are connected to the form group's control names, such as email and password. The corresponding component methods, such as login(), register(), and forgotPassword(), which are called when the user initiates these actions, are linked to the ion-button elements.

Backend Interaction

The AuthenticationService class (see Figure 4.4) injects Angular's dependency injection system, offering methods like 4.7register() and 4.8login(), which leverage Firebase Authentication's createUserWithEmailAndPassword and signInWithEmailAndPassword functions. These methods authenticate the user against the Firebase backend, establishing a session or creating a new user record.

Figure 4.6: authFrontEnd

Figure 4.7: register

Figure 4.8: Login

The characteristic that is reacting The component subscribes to currentUser, an observable of the Firebase authentication state, for real-time changes to make sure the user interface (UI) reflects the current authentication status.

State Management

By providing methods like register() and login() that make use of Firebase Authentication's createUserWithEmailAndPassword and signInWithEmailAndPassword functionalities, the AuthenticationService class injects Angular's dependency injection system. By using these techniques, a session is established or a new user record is created, and the user is authenticated against the Firebase backend.

The component subscribes to the Firebase authentication state's reactive property currentUser for real-time changes, guaranteeing that the user interface (UI) accurately displays the current authentication status.

4.4.2 Group Detail Page

Overview

The GroupDetailPage (see Figure 4.4) offers users an in-depth look at a specific group they are part of or wish to explore. It is the central hub for user interaction within a group, allowing members to view details, manage posts, and access group files.

Frontend Implementation

The parts of the Ionic framework put together a user-friendly interface that captures the functionality of the group. The group detail page and the group's media files page may be seamlessly routed between each other thanks to the navigation buttons found in the header, which is defined by the ion-header element.

The group's name, description, and member count are supplied in ion-card-content, while the description and name are wrapped in an ion-title tag. These elements are dynamically loaded and displayed within ion-card elements. The ion-progress-bar in Ionic is used to show background operations, such as post production.

An Angular form group facilitates the functionality of adding new posts by connecting the input with form control names like content. For an improved user experience, an Ionic button styles and activates a file input that is included for uploading images.

In order to iterate over an array of posts with edit and delete options connected to corresponding methods like editPost() and deletePost(), *ngFor is used for post interactions.

Backend Interaction

The GroupService (see Figure 4.5) is essential for fetching group details 4.9(getGroupById()) and managing posts 4.10(addPostToGroup(), getPostsByGroupId()). The component interacts with this service to handle CRUD operations for groups and posts.

```
async getGroupById(groupId: string): Promise<Team> {
  const groupDocRef = doc(this.firestore, 'groups', groupId);
  const groupSnapshot = await getDoc(groupDocRef);

  if (groupSnapshot.exists()) {
    // If the document exists in the collection, cast it to the Team type and re
    const group = groupSnapshot.data() as Team;
    group.id = groupSnapshot.id; // Make sure to set the ID
    return group;
  } else {
    // Handle the case where the group does not exist
    throw new Error('Group not found');
  }
}
```

Figure 4.9: Group By Id

```
// Method to add a post to a group
async addPostToGroup(post: Omit<Post, 'id'>): Promise<void> {
  const postRef = collection(this.firestore, `groups/${post.groupId}/posts`);
  await addDoc(postRef, {
    ...post,
    timestamp: serverTimestamp() // Use Firebase's server timestamp
  });
}
```

Figure 4.10: Add Post To Group

The authentication status is checked upon initialization through the 4.11AuthenticationService, ensuring that only authenticated users can interact with the group.

Methods like loadGroup() and 4.12loadPostsForGroup() are invoked to retrieve detailed information from Firebase, while addPost() includes logic for handling post creation, including image upload and form validation.

Real-time Feedback and Loaders

The application offers real-time feedback through the ion-progress-bar and different loading indications during demanding processes like post creation. Giving users visual feedback that their request is being handled is essential for improving their experience.

For instance, the progress bar 4.13 appears after a user submits a new post because the creatingPost boolean is set to true in this scenario.

```
ngOnInit() {
  this.auth.getCurrentUser().subscribe(user => {
    if (user) {
      this.userId = user.uid;
      console.log("Authenticated User ID:", this.userId);

      this.route.paramMap.subscribe(params => {
        const groupId = params.get('id');
        console.log("Route parameter groupId:", groupId);

        if (groupId) {
          this.groupId = groupId;
          console.log("Set groupId from route:", this.groupId);
          this.loadGroup(this.groupId);
          this.loadPostsForGroup(this.groupId);
        }
      });
    } else {
      console.error('User is not logged in.');
      this.router.navigate(['/login']);
    }
  });
}
```

Figure 4.11: Get User Details

```
loadPostsForGroup(groupId: string) {
  console.log("Loading posts for groupId:", groupId);
  this.userProfileService.getPostsByGroupId(groupId).subscribe(
    (retrievedPosts) => {
      console.log('Posts loaded for groupId:', groupId, retrievedPosts);
      this.posts = retrievedPosts.map(post => ({
        ...post,
        timestamp: post.timestamp.toDate()
      }));
    },
    (error) => console.error('Error loading posts for group:', error)
  );
}
```

Figure 4.12: Load Posts for Group

```
if (user && groupId) {
  this.userProfileService.createPost(user.uid, postContent, groupId, this.selectedFile)
    .subscribe({
      next: (postId) => {
        console.log(`Post created with ID: ${postId}`);
        this.postForm.reset();
        this.selectedFile = undefined; // Reset selected file
        this.loadPostsForGroup(groupId);
        console.log('Post creation process completed.'); // Add this line
        this.creatingPost = false;
      },
      error: (error) => {
        console.error('Error creating post:', error);
        this.presentAlert('Failed to add post', 'An unexpected error occurred');
        // Set creatingPost to false if there's an error
        this.creatingPost = false;
      }
```

Figure 4.13: Loading For Posts

Error Handling and Feedback

Managing errors is essential to a satisfying user experience. The presentAlert() and presentAlertConfirm() methods of the component are used to notify users of the status of their actions. These techniques are applied to handling and showing errors that may arise during group interactions in addition to success messages:

4.14presentAlert() is called when a user takes an action that causes an error, like forgetting to delete a post or group, and displays a helpful error message to the user. Furthermore, confirmation dialogs can be displayed using the presentAlert-Confirm() method prior to potentially irreversible operations, such as removing a group.

```
// Utility function to present an alert - add this to your class
async presentAlert(header: string, message: string) {
  const alert = await this.alertController.create({
    header: header,
    message: message,
    buttons: ['OK']
  });
  await alert.present();
}
```

Figure 4.14: alert

4.4.3 Groups Page

Overview

The GroupsPagePage (see Figure 4.5) component presents a platform where users can create new groups, search for existing ones, and view groups they are a part of. It is an interactive interface that fosters community engagement and collaboration among users.

Frontend Implementation

Constructed with Ionic components, the page begins with an ion-header that incorporates a back button, guiding users back to the home page for easy navigation. An ion-title neatly encapsulates the purpose of the page.

A form is provided to create a new group, with form controls for the group's name and description bound to the 4.15groupForm FormGroup object, ensuring validation and state management. An ion-button triggers the form submission, invoking the onSubmit() method when the user intends to create a new group.

```
<ion-content [fullscreen]="true">
  <div class="ion-padding">
    <h2>Create a New Group</h2>
    <form [formGroup]="groupForm" (ngSubmit)="onSubmit()">
      <ion-item>
        <ion-label position="floating">Group Name</ion-label>
        <ion-input type="text" formControlName="groupName" required></ion-input>
      </ion-item>
      <ion-item>
        <ion-label position="floating">Group Description</ion-label>
        <ion-textarea formControlName="groupDescription"></ion-textarea>
      </ion-item>
      <ion-button expand="block" type="submit" [disabled]="groupForm.invalid">Create Group</ion-button>
    </form>
  </div>
```

Figure 4.15: Create Group Form

Below the form, a search bar allows for dynamic filtering of groups. The search results and the user's groups are conditionally rendered using *ngIf. This creates a seamless and dynamic experience as the search results update in real-time.

Backend Interaction

Interactions with GroupService and UserProfileService are at the heart of the component's functionality. The GroupService facilitates group creation (createGroupAndChat()), deletion (deleteGroup()), and retrieval (getGroups(), getGroupsForUser()).

The ngOnInit() lifecycle hook initializes the component state by fetching all available groups and those the user is a member of, reacting to the authentication state provided by the AuthenticationService.

Group Management and User Feedback

The 4.16joinGroup() function encapsulates the logic to add a user to a group and its associated chat. It includes error handling to provide feedback in case the operation fails.

```
async joinGroup(group: Team | undefined) {
  if (group && group.id) {
    const user = await firstValueFrom(this.auth.getCurrentUser());
    if (user && user.uid) {
      try {
        await this.groupService.joinGroupAndChat(group.id, user.uid);
        console.log('Joined group and chat:', group.groupName);
      } catch (error) {
        console.error('Error joining group and chat:', error);
      }
    } else {
      console.error('User is not logged in or UID is unavailable');
    }
  } else {
    console.error('The group or group ID is undefined. Cannot join group.');
  }
}
```

Figure 4.16: Join Group

When a user attempts to delete a group, the deleteGroup() method checks if the user has the necessary permissions before proceeding. This method and the isUserGroupCreator() utility function underscore the app's security and data integrity measures.

Search and Filter Functionality

The component includes a search and filter feature, implemented in the filterGroups()4.17 method. It updates the list of displayed groups based on the user's input, enhancing usability and user experience.

```
filterGroups(event: any) {
  const searchTerm = event.detail.value.toLowerCase();
  this.searchPerformed = true; // Indicate a search has been performed

  if (!searchTerm.trim()) {
    this.groups = []; // Option to reset the displayed groups
  } else {
    // Filter allGroups based on the search term, not considering membership
    this.groups = this.allGroups.filter(group => {
      return group.groupName.toLowerCase().includes(searchTerm) ||
             (group.groupDescription && group.groupDescription.toLowerCase().includes(searchTerm));
    });
  }
}
```

Figure 4.17: Filter Groups

Navigation to Group Details

A method goToGroupPage() is implemented to navigate to a group's detailed view, passing along the necessary parameters to the route

4.4.4 Gym Workout Page

Overview

The GymWorkoutsPage (see Figure 4.5) serves as a user's portal to monitor their gym sessions, specifically focusing on heart rate metrics and calories burned. It integrates with Google Fit to retrieve real-time workout data.

Frontend Implementation

Built with Ionic, the page features an ion-header for navigation, with a router link directing users back to the home page. The main interface is designed to display session data including heart rate statistics and calorie count, dynamically updated based on the user's workout session data.

An ion-card displays the session information: start and end times, heart rate statistics like the latest, highest, and lowest BPM (beats per minute), and the total calories burned, formatted to precision. 4.18

Actions such as saving workout data and fetching activity records are initiated via ion-button elements. The use of the 4.19ModalController demonstrates a pattern to present date and time pickers for session data selection.

Backend Interaction

Interactions with the BluetoothService are vital for the operation of the page. This service facilitates Google authentication, workout data retrieval, session data processing, and storing workout details to Firestore.

Upon initialization in ngOnInit, the component calls the signInWithGoogle()4.20 method, which sets up the user authentication state and fetches initial data.

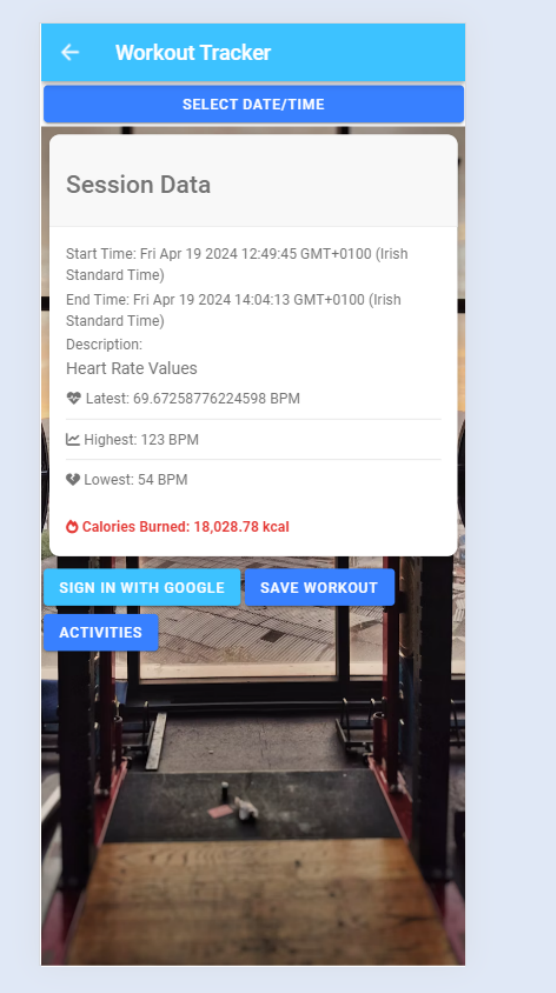

Figure 4.18: Workout Page

```
async openDatePickerModal() {
  const modal = await this.modalCtrl.create({
    component: DateAndTimePickerPage // Your date picker page
  });

  modal.onDidDismiss().then((result) => {
    if (result.data) {
      // If data is returned, use it to update your start and end times and fetch data
      this.startTime = result.data.startTime;
      this.endTime = result.data.endTime;
      this.fetchActivitiesCustom(this.startTime, this.endTime);
    }
  });

  return await modal.present();
}
```

Figure 4.19: Modal Controller for Date and Time Picker

Google Fit Data Processing

The processSessionData() method parses the fetched session data, extracting relevant heart rate values and calculating the highest and lowest BPM from the

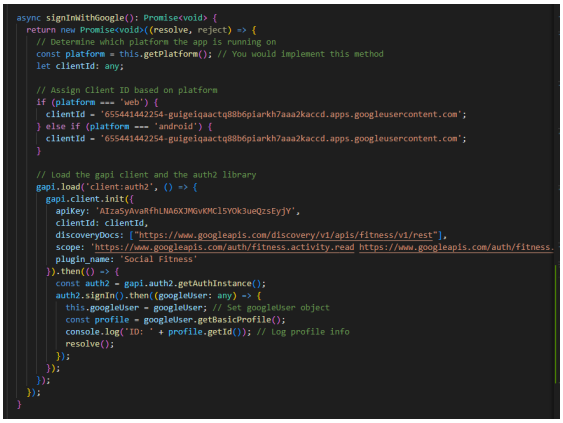

Figure 4.20: Auth0 Trigger for Api sign in

workout session.

The saveWorkout() method consolidates workout data into a WorkoutDetails object and utilizes the BluetoothService to store the session in Firestore.

User Sign-in and Data Fetching

The page provides a sign-in button that utilizes Google's OAuth2.0 for user authentication via the BluetoothService. Once signed in, the page fetches heart rate and calorie data tied to the user's session, calling the fetchHeartRateForSession()4.21 and fetchCaloriesBurnedForSession() methods respectively.

4.4.5 Home Page

Overview

The HomePage serves as the main dashboard for users after they log in. It features a streamlined interface for user interaction, access to the main feed, and quick logout functionality.

Frontend Implementation

The interface is built using Ionic components. It includes an ion-header with a logout button that allows users to sign out of the application. The ion-title displays the page's title, reinforcing the user's context. 4.22

The core of the page is wrapped in ion-content, which embeds the <app-feed> component—this component dynamically loads the user's feed, showcasing posts or activities relevant to the user. There's also a provision (commented out) for a

```
// Method to get heart rate data for a specific session
async getHeartRateDataForSession(startTimeMillis: string, endTimeMillis: string): Promise<any[]> {
  if (!this.googleUser) {
    throw new Error('User not signed in');
  }

  const accessToken = this.googleUser.getAuthResponse().access_token;
  const headers = new HttpHeaders({
    'Authorization': 'Bearer ' + accessToken,
    'Content-Type': 'application/json'
  });

  const requestBody = {
    aggregateBy: [{
      dataTypeName: 'com.google.heart_rate.bpm'
    }],
    bucketByTime: { durationMillis: parseInt(endTimeMillis) - parseInt(startTimeMillis) },
    startTimeMillis: startTimeMillis,
    endTimeMillis: endTimeMillis
  };

  try {
    const response = await this.http.post<GoogleFitDataSetsResponse>(
      `https://www.googleapis.com/fitness/v1/users/me/dataset:aggregate`,
      requestBody,
      { headers }
    ).toPromise();

    // Ensure the response is not undefined and has the expected structure
    if (response && response.bucket) {
      // Flatten the array of buckets and extract heart rate data points
      const heartRateDataPoints = response.bucket.flatMap(bucket =>
        bucket.dataset.flatMap(dataset => dataset.point)
      );
      return heartRateDataPoints;
    } else {
      return []; // No data points found
    }
  } catch (error) {
    console.error('Error fetching heart rate data points for session:', error);
    throw error;
  }
}
```

Figure 4.21: Heart Rate Fetch

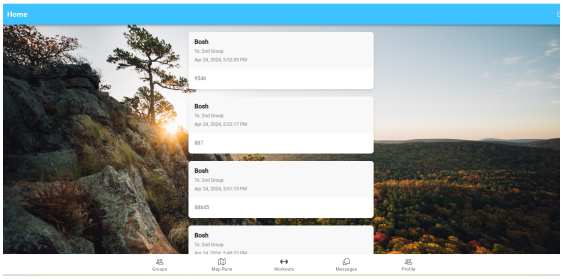

Figure 4.22: Home Page

button to create new posts, illustrating potential expansion or features not currently active.

The page structure concludes with an ion-footer that incorporates <app-tabs>, facilitating navigation across different sections of the app.

Backend Interaction

Interactions with AuthenticationService are crucial for user management on this page. The service handles user authentication states and logout functionalities.

Upon component initialization (ngOnInit), the component subscribes to the currentUser observable from AuthenticationService to monitor and react to changes in the user authentication state. This allows the page to adjust dynamically if the user's authentication status changes, enhancing security and user experience.

Logout Functionality

The logout() method is tied to the logout button in the header. This method invokes the logout() function from AuthenticationService, which handles the de-authentication process, and then navigates the user back to the login screen or landing page using Router.navigateByUrl().

4.4.6 Media Files Page

Overview

The MediaFilesPage provides functionality for users to upload, view, and manage media files within a specific group. It allows the upload of videos and the organization of files by type, enhancing user interaction and file management within the application.

Frontend Implementation

The page utilizes Ionic components to create a user-friendly interface. The ion-header contains navigation controls, including a back button that utilizes the navigateToGroupDetail(groupId) method to return to the group detail page. The main content area (ion-content) displays upload controls and lists videos and other media files, grouping them by type for easy access.

Backend Interaction

Interactions with Firebase are pivotal for file management. Files are uploaded to Firebase Storage, and metadata is stored in Firestore within a specific group's document, allowing for efficient retrieval and organization of files.

Upon component initialization (ngOnInit), the component subscribes to the currentUser observable from AuthenticationService to ensure that file operations are performed by authenticated users. It also retrieves the group ID from route parameters, which is essential for file operations related to the specific group.

File Upload and Management

The component provides a file input for uploading videos, handled by the uploadFile(event) 4.23 method. This method checks the file type, uploads the file to Firebase Storage, and stores the file metadata in Firestore. The files are then grouped by type using groupFilesByType(files), which categorizes files based on their extensions for display on the page.

```
uploadFile(event: any) {
  const input = event.target as HTMLInputElement;
  const file = input.files ? input.files[0] : null;
  console.log("Preparing to upload file for groupId:", this.groupId);

  if (!this.groupId) {
      console.error('GroupId is not set. Cannot upload file.');
      return;
  }

  if (file) {
      const fileType = this.determineFileType(file.name);
      const filePath = `${fileType}/${new Date().getTime()}_${file.name}`;
      const fileRef = ref(this.storage, filePath);
      const groupFilesRef = collection(this.firestore, `groups/${this.groupId}/files`);

      console.log('Starting file upload:', file.name, 'to groupId:', this.groupId);
      uploadBytes(fileRef, file).then(uploadResult => {
          getDownloadURL(uploadResult.ref).then(downloadURL => {
              const fileMetadata = {
                  url: downloadURL,
                  name: file.name,
                  createdAt: new Date(),
                  type: fileType
              };
              addDoc(groupFilesRef, fileMetadata);
              console.log('File uploaded and metadata stored:', fileMetadata);
          }).catch(error => console.error("Error getting download URL:", error));
      }).catch(error => console.error("Upload failed:", error));
  } else {
      console.log('No file selected for upload.');
  }
}
```

Figure 4.23: Upload File

```
private groupFilesByType(files: any[]): { [key: string]: any[] } {
  const grouped: { [key: string]: any[] } = {}; // Explicitly type 'grouped' as a dictionary
  files.forEach(file => {
    const type = file.type || 'other';
    if (!grouped[type]) {
      grouped[type] = [];
    }
    grouped[type].push(file);
  });
  return grouped;
}
```

Figure 4.24: File Types Grouping

4.4.7 Messaging Page

Overview

The MessagingPage facilitates real-time communication between users within specific groups or direct messages. It manages user conversations, displays messages, and allows users to send new messages. This component integrates closely with Firebase Firestore to handle message storage and retrieval, showcasing real-time data capabilities.

Frontend Implementation

The page uses Ionic components for layout, with an ion-header for navigation and ion-content for displaying messages and conversation controls. Messages are listed in a scrollable view where users can send new messages through an input field linked to the sendMessage method. 4.25

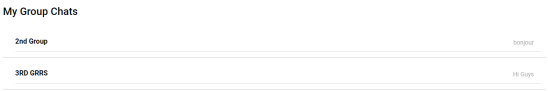

Figure 4.25: Message Page

Backend Interaction

Firebase Firestore is extensively used for storing and retrieving message data. Conversations and messages are stored in collections within Firestore, allowing for efficient data retrieval and updates, which are crucial for the real-time aspect of the messaging feature.

- Firebase Firestore: Used for storing messages and conversation data. 4.26
- Firebase Authentication: Ensures that messages are sent and received by authenticated users, maintaining security and data integrity.

```
getMessages(groupId: string): Observable<Message[]> {
  const messagesRef = collection(this.firestore, 'groups', groupId, 'messages');
  const q = query(messagesRef, orderBy('timestamp', 'asc'));
  return collectionData(q, { idField: 'id' }) as Observable<Message[]>;
}
```

Figure 4.26: Get Messages Method

Key Functionalities and Methods

- Conversation Management: Manages user conversations, loads existing conversations upon user login, and allows users to select different conversations.4.25

4.4.8 Profile Page

Overview

The ProfilePage component is central to user interaction in the application, handling the display and modification of user profile information. It allows users to

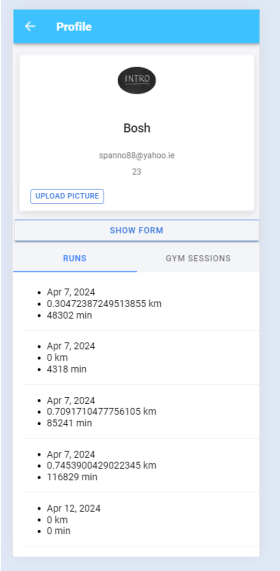

Figure 4.27: Profile Page

view and update their profile details such as name, email, age, and profile picture. The component leverages Firebase Firestore to manage user data and Firebase Storage for handling profile images.

Frontend Implementation

The page utilizes Ionic components to construct a user-friendly interface. Profile data is displayed using ion-card elements, and an ion-avatar displays the user's profile picture. Users can update their information directly through input fields and submit changes via associated update buttons. 4.27

Backend Interaction

Firebase Firestore is used extensively for retrieving and updating user data:

- Firestore: Stores and retrieves user profile information. Updates to the database are handled through specific service methods that encapsulate Firestore operations. 4.28

- Firebase Storage: Manages profile images. Users can upload new images, which are then stored in Firebase Storage with URLs stored in the Firestore database.4.29

```
// Retrieve the user's profile data for a given UID
getUserProfile(uid: string): Observable<any> {
  console.log(`Fetching user profile for UID: ${uid}`); // Debug log
  const userProfileRef = doc(this.firestore, `users/${uid}`);
  return new Observable<any>((observer) => {
    getDoc(userProfileRef).then((docSnap) => {
      if (docSnap.exists()) {
        const userProfileData: any = docSnap.data();
        observer.next(userProfileData);
      } else {
        observer.error('User profile not found');
      }
    }).catch((error) => {
      observer.error('Error fetching user profile data: ' + error);
    });
  });
}
```

Figure 4.28: Get User Profile

```
uploadProfilePicture(file: File): Promise<string> {
  const path = `profilePictures/${new Date().getTime()}_${file.name}`;
  const storageRef = ref(this.storage, path);
  const uploadTask = uploadBytesResumable(storageRef, file);

  return new Promise((resolve, reject) => {
    uploadTask.on('state_changed',
      (snapshot) => {
        // Optional: monitor upload progress
      },
      (error) => reject(error),
      () => {
        getDownloadURL(uploadTask.snapshot.ref).then((downloadURL) => {
          resolve(downloadURL);
        });
      }
    );
  });
}
```

Figure 4.29: Upload Profile Picture

Key Functionalities and Methods

- Data Retrieval: User data is fetched from Firestore upon component initialization and whenever the user updates their information.
- Data Update: Users can update their name, email, and age, which are then saved back to Firestore.
- Image Upload: Users can upload a new profile picture, which is saved to Firebase Storage. The URL is then updated in the Firestore database.

4.4.9 Google Maps API Usage - Run Tracker Page

Overview

The Run Tracker feature of the app integrates the Google Maps API (see Figure 4.4) to provide essential services such as real-time location tracking, route visualization, and activity logging. This integration is critical for users aiming to monitor their runs, track improvements, and set fitness goals.

Integration with Ionic/Angular

The Google Maps API is seamlessly integrated into the Ionic/Angular framework, providing a robust solution for real-time geolocation tracking and interactive mapping. This allows users to interact with the map, start and monitor their runs, and receive dynamic updates on their progress.

Geolocation and Mapping

Utilizing the Google Maps API, the app offers detailed mapping capabilities:

- Route Visualization: Users can see their running route in real-time, with the path dynamically plotted as they run.
- Location Updates: Real-time location updates allow for precise tracking of the user's movements during their activities.

Implementation Highlights

- Map Initialization: The map is initialized on the Run Tracker Page, rendering within a dedicated div element that displays the user's current location and running path.
- User Location: The setCurrentPosition method employs the Geolocation API to fetch and mark the user's current location on the map.
- Run Tracking: The startTracking method engages when a user begins their run, marking the starting point and tracking the run's progress by drawing a polyline that follows the user's route.

```
// Method to start tracking the run
async startTracking() {
  try {
    // Get the current position using the Geolocation API
    const position = await Geolocation.getCurrentPosition();
    console.log("Start tracking position: ", position);
    const startLocation = new google.maps.LatLng(
      position.coords.latitude,
      position.coords.longitude
    );

    this.previousPosition = this.startLocation;

    // Generate waypoints for a rough circular route
  const waypoints = this.generateCircularPathWaypoints(startLocation, this.selectedRunDistance / 4);

  // Use Google Maps Directions API to get the route
  const directions = await this.getDirections(startLocation, waypoints);

    if (this.map && directions !== null && directions.routes && directions.routes.length > 0) {
      // Draw the route on the map using polylines
      const routePolyline = new google.maps.Polyline({
        path: directions.routes[0].overview_path,
        geodesic: true,
        strokeColor: '#FF0000',
        strokeOpacity: 1.0,
        strokeWeight: 2,
      });

      routePolyline.setMap(this.map);
```

Figure 4.30: Visualization of Run Tracking on the Map

Advanced Mapping Features

- Circular Route Generation: For users preferring predefined routes, the app can generate circular running paths based on the selected distance, enhancing the planning of workouts.

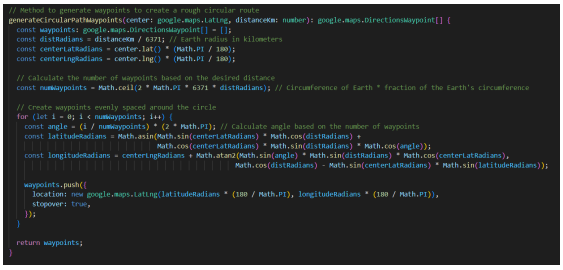

Figure 4.31: Circular Waypoints

- Live Tracking: Leveraging the watchPosition method from the Geolocation API, the app updates the user's position in real-time, providing continuous feedback during the run.

Figure 4.32: Live Tracking

User Interaction and Feedback

- Interactive Map: Users can interact with the map to view different segments of their run, zoom in/out, and observe specific details like pace and distance markers.
- Progress Tracking: The app records detailed data on each run, including pace, distance, and time, which are crucial for users tracking their fitness progress.

Technical Implementation

- TThe Angular framework is used to link the Google Maps API into the UI, guaranteeing a smooth user experience and reliable performance. The item Data storage, including run stats and user profiles, is handled by the backend, which is Firebase-powered, guaranteeing data security and permanence.

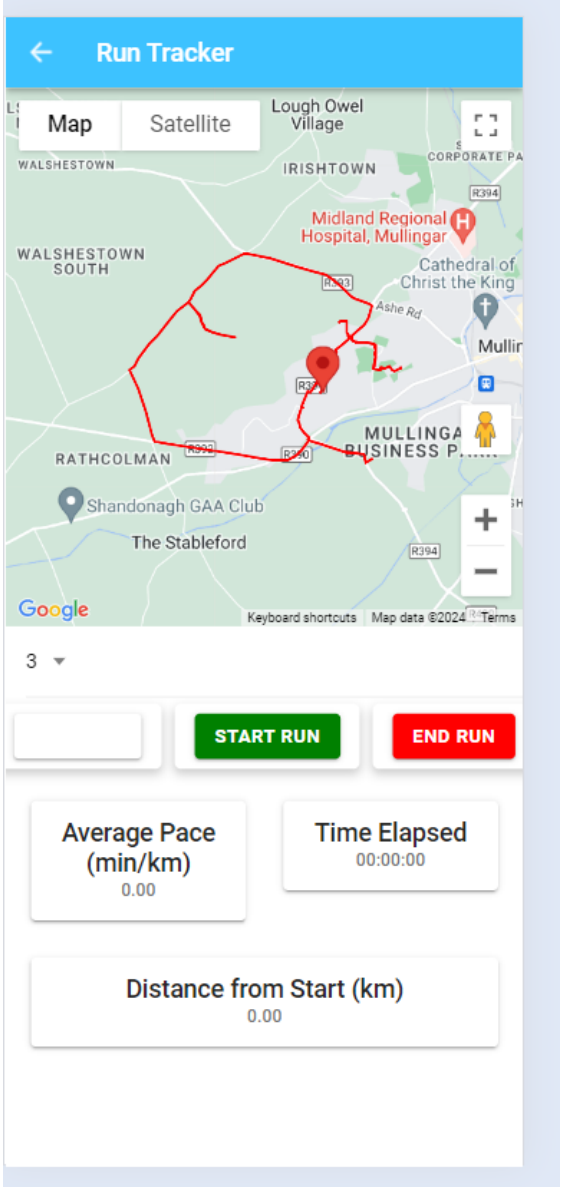

Figure 4.33: Run Path Visualization with Polyline

Conclusion

The Run Tracker feature's integration of the Google Maps API greatly improves the usefulness of the program and user interaction. The software efficiently assists users in their fitness journeys by offering complete run data, interactive mapping, and specific geographic monitoring.

Chapter 5

System Evaluation

In this paragraph Social Fitness app is assessed regarding its usability, functionality and the ability to deliver its pre-set features. The application's resiliency and efficacy was demonstrated by rigorous testing, both ongoing and iterative, the application shortcomings had to be maintained, despite the nonuse of conventional testing protocols during the development.

5.1 Meeting Project Objectives

The Social Fitness app's development has a defined trajectory thanks to the initial goals outlined in the Introduction chapter. This section assesses how well the features that were implemented, the degree to which the project's deliverables were met, and how well the application complied with these objectives.

5.1.1 User Authentication System

To make sure the app's authentication system is dependable and stable, it underwent a number of rigorous testing. Among these tests were:

- **Login/Logout Functionality:** Verification of user access control and session management.
- **Account Creation:** Testing of the registration process for ease of use and error handling.
- **Password Reset:** Ensuring users can recover access to their accounts with a secure process.

The authentication system's integration with Firebase provided a robust framework that met all the security and functionality requirements.

5.1.2 User Profile Creation and Navigation

Successful authentication leads to the profile creation stage, where user engagement begins. This feature underwent a comprehensive evaluation focusing on:

- **User Input Validation:** Verification processes were implemented to ensure that all user-submitted data are accurately validated prior to storage.
- **Profile Personalization:** Users are empowered to personalize their profiles by adding individual details and uploading profile pictures.

The steps from profile creation to homepage were separately outlined and improved within the design to fluent user experience and carry out user testing to ensure smooth and intuitive transitions that will guarantee users comfort while navigating their apps.

The deployment of profile fucntionalities was unquestionably successful, yet the process of bringing activity tracking into the system was far from being flawless. To be specific, although I had decided to create the segmented control for the segment used to present the runs and saved gym sessions to Firebase, data presentation when working with the interface was a bit shaky at the beginning.

Data Segmentation and Binding Issue

- **Data Segmentation Challenge:** At the beginning, all the activity data were bundled up into single Activity under the menu named as 'Activity' nested within the segmented interface. This brought about the problem where the data could not be recorded correctly from the gym area and the situation as shown 5.1.
- **Resolution Approach:** The issue was resolved by separating the fetching functions into distinct methods. This ensured that run and gym data were queried and bound to their respective segments correctly, preventing any overlap or misrepresentation of the data.
- **Enhanced Data Presentation:** Post-resolution, the app accurately displayed the user's fitness activities, allowing them to track and analyze their workout progress efficiently.

Figure 5.1 illustrates the error encountered with run data incorrectly appearing under the gym session heading. The resolution of this error was pivotal in ensuring the accuracy and reliability of the app's data presentation.

After fixing this mistake we not only managed to enhance our fitness app's service, but this also indicated that our project was aiming to exceed user expectancy and produce a fine user interface. This emphasized the point that while

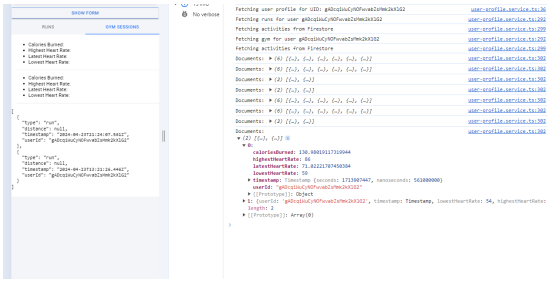

Figure 5.1: Incorrect Data Binding for Run and Gym Sessions

testing and logic are very fundamental components of application development, they equally need adapting to fit into the real world problems.

5.1.3 Home and Group Pages

The home and group pages are central to the Social Fitness app's community engagement. They were tested for:

- **Content Display:** On a design side, the home page has been created to display a wide range of posts from the various user groups; the way the posts are display is clear and engaging. Implementation of Ionic components was a requisite to create a clear representation of content sections, comprising lists, cards, and categories. In addition, their visually appealing nature and functional utilities make them suitable elements.

- **Interactivity and User Engagement:** The UI features were engineered so reaction was instantaneous when user tried to interact with group pages. Users could accomplish what they are set to by implementing features such as creating or joining groups, searching for groups that are already existing, and also partaking or involvement during group discussions among the real-time feedback to achieve a great user experience.

- **Group Creation and Management:** The networking process was designed to be simple, so that people can initiate a new group with an utmost ease. We implemented the strong form of validation, the group details were correctly filled with the system. Furthermore, group administration elements were taken into account for the control of group members requests and the content.

- **Integration of Group Features:** In the beginning, the home page area was made to handle individual user bellows. The integration into the group related functionality was then attained consecutively, and firstly the user

could post and view on the specific group was achieved. Next, the integrated search and add features came in to play, thus the utilization of the app was upgraded by people being to join new communities.

- **Data Binding and Segmentation:**For example, encode the screen in the group segments and the problem that was faced with data binding during development. The session data from the gym was erroneously overwriting the data from the run because of shared data setting. To address this problem, I made a method which separated getting and loading methods for different data types, and this way I made sure that the content will be correctly associated and displayed by groups.

- **Usability Testing:** In user testing sessions we worked on the improvement of the flow and functionality of the home and group pages by applying changes proposed to us after listening to users' feedback. The data that we collected at these meetings helped us to further progress with the enhancement of the UI/UX design. This led to a smoother process when it comes to the navigation and interaction of the end users from our product.

The process of creating home and group pages was iterative and concerned with bringing out user engagement and being easier for our content to be managed on the website. The ability of the app to offer cohesive social fitness interaction is proved if it features the mentioned components which an individual can also incorporate when on a fitness journey themselves and their communities.

5.1.4 Run Tracker Page

The Run Tracker Page is the primary locus of the Social Fitness app which utilizes Google Maps API to work out a userâ€™s location and provide a visualization of the route which he/she travelled. This webpage was done in an effort to improve the actual fitness experience for those who use it by having stream-lined result for tracking of runs and other useful functions.

- **Google Maps API Integration:** Realization of the integration for the Google Maps API was critical as it enabled to visualize and monitor the running routes The app can be able to call the API in which case the API will bring up the users' current location and their running track by drawing a Polyline that keeps updating in real-time as the user runs.

- **Circular Route Calculation**:At the beginning, the app was drawing a straight line through a place where the starting and ending points were situated. Such a mode was indeed difficult to use for people running belonging

to the category of those who started and ended their run at the same place. Way points for the circular route were computed using an algorithm. By this, the Polyline directs the phases forming a looped run, therefore the audiences will be able to finish their run at the starting point. This helpful element improved the user-friendliness thoroughly on the run tracking side, namely enabling people to construct their running sessions more realistically.

- **Dynamic Route Visualization:**The course line is a key point of the design for this training program it shows the pace which user is moving at where a visual representation is used to show the distance traveled. Both of the runners who pay attention to the feedback metrics are unique and this allows them to guide on the running pace and distance which is very helpful to runners.

- **Pulse Effect on Marker:**On that note we also stated that the markers should try to make something that would resemble motion like this was the case in the game too. Maintaining consistency posed a persistent challenge for me as well. Despite the fact that I was using custom styles, the marker never adhered precisely to my visual desires due to the paucity of the Google Maps API's styling functionalities. It is implied to be another key prospective zone for improvement in the upcoming update, as airborne views will allow the quality of air to be witnessed more plausibly.

- **Testing and Refinement:** Instruction Tracker Page was a candidate to many different tests that proved the efficacy and capability of it. Simulation procedure has been validated through manufacturing and pre-programmed route to make sure that Polyline trace out the user path and circular point is the displayed loop waypoints. Correction and modifying the algorithm continued to be the platform to which the staff at the company was required to improve the website experience as they send out feedback to different industry executives.

In summary, it is apparent that the Run Tracker Page marked a major milestone in the app's functionality, allowing users to perform comprehensive tasks of running scheduling and recording of their runs. Solutions to circular route tracking with innovative designs and promises of further testing and improvement helps in the practice of delivering a tracker for the purpose of fitness monitoring with high quality. Inconsistency in the stamina meter is the only hitch that I find but Run Tracker Page has delivered the main purpose and it's clear that it's a milestone in continuous improvement.

5.1.5 Media Files Page

The Media Files Page is a storage area that is used to upload and share multimedia files like photos and videos on the Social Fitness app between people who are involved in common activities. This functionality targets the popularization of resource sharing by ensuring that users will find it easy to access and benefit from the media and related content that are of interest to them.

- **Group-Specific File Access:** The app has enforced restrictions on the display of files by only showing files that are related to the groups that the user has joined; this not only maintains privacy but also relevance of content shown to the user. This is achieved by a security model where the access rights are checked based on which user group users belong before they are allowed to alter the documents.

- **Multimedia File Uploads:**On the other hand, blog users have more production types flexibility to indicate images and other types of media content like video clips, spreadsheets, or presentations. The feature is AI powered and it gives this feature a robust credibility and ability to serve a large number of customers due to the fact it uses a very reliable and scalable data service for energy file storage-either Firebase's cloud or local device.

- **Organized File Management:**Here you might come across to a group of materials (list) that sort the already categorized data base on users group once the system is set live.

- **Seamless File Sharing:**Communication technologies and doing different activities during days (ex: doing plan and exercises for sport or motivation text. This leads to more and more intensive socialization between people). One of signifacance is greatest of them.

- **Testing and Performance**:Lastly, a Media File Feature which has gone through different evaluation processes and usages on other downloads/uploads was welcomed. Among comparisons, simultaneous loading is loaded to see whether the system can handle multiple people uploading and accessing the website without performance lag such as loading delay.

We have taken pains to design the Media Files Page to be easy to use and efficient for sharing and downloading resources with a user-oriented approach, which gives the users the feeling that they are in a community using the same apps.

5.1.6 Messaging Feature

Communication in Social Fitness app is one major characteristic that this app mainly suports since it has a messaging feature which enables the group members to interact and include each other in all kinds of activities. It serves the purpose that how the youth get their real life affairs handled to experience the actual social environment.

- **Real-Time Group Chat:** To create that online presence lead app firebase real-time database feature makes the application funtion instant message sharing within the app group chatting, we app will be user like it is off-line seeing friends
- **Group Membership Validation:**This model of security came into operation to make sure that users can contact only those people from groups they are members of and otherwise information of the groups remains private and confidential.
- **Automatic Group Chat Enrollment:** It improves the usability by automating the addition of members to group chats so that there are no additional steps that need to be taken manually by the users for them to be able to start a social interaction.
- **User Experience:** Finally, communications interface is configured to be eye catching and efficient, emphasizing on ease of use to entice participation and interaction from all the groups members.
- **Functional Testing**:The target communication channel was passed through a rigorous testing process and was indeed proven to be a viable and better channel in all parts of its performance. An evidence of its practicality are the group joint inductions with diverse means of message sendings and notifications deliveries that allow comfortable user navigation.

The most of the site icon design is using a chat option and this is to signify the site's idea of a complete fitness community where results sharing has returned to its right place which is never been done in isolation, rather directly contributed to the long-term health of the individual by getting community members engaged in the system.

5.1.7 Gym Workout Page

Performance tracking of the various popular fitness tracker apps and devices is the emphasis of this attack, which educating masses about workouts in the gym located on the Social Fitness application.

- **API Communication Difficulties:**To start with, third-party data platforms through google fit service proved problematic to use it for data read of work out. The API connectivity facilitating a Google fit friendly interface was the main issue of compatibility of our system with another This is an example of the limited API connectivity. On top of the delays that occurred on the path towards the authentication process I also found another one of the huddles that tripped me.

- **Auth0 Integration Issues:** Consistency in terms ofAuth0, (auth0.com) whose main goal is to manage privacy and security of user authentication, came into conflict with the Google Fit API, (developers.google.com/fit/dev) because of the Android phones compatibility issue which negatively affected experience of the app and made it seem less smooth and clean.

- **Adapting to Constraints:** In response to these obstacles, the focus shifted toward querying the Google Fit API based on workout start and end times. This approach enabled the successful retrieval of heart rate and calorie data that users had previously recorded using their fitness devices.

Functionality and User Experience

Despite initial setbacks, the Gym Workout Page provides valuable functionality and maintains a positive user experience in its current state:

- **Data Visualization:** The app effectively displays historical data related to gym sessions, allowing users to review and monitor their progress over time.

- **Fallback Strategies:** By utilizing timestamp-based queries, the app compensates for the inability to track live data, still offering users insights into their workout patterns.

- **Web View Performance:** The feature performs as intended in web view, showcasing the application's capacity to adapt across different platforms.

- **Resilience and Persistence:** The development process highlighted the project's resilience in overcoming technical challenges, ensuring that core functionalities were preserved and user value was not compromised.

Challenges with Live Data Retrieval

During the development of the Gym Workout Page, significant challenges were faced in fetching live heart rate data and calorie information. Efforts to utilize Google Fit's sessions API for real-time data tracking were met with complexities,

resulting in the decision to use a query-based system for retrieving pre-recorded data:

5.1.8 Robustness of the Software

The app was tested under various conditions to ensure stability. Stress testing was performed to observe behavior under high-load scenarios, such as simultaneous multiple users logging in and messaging.

5.1.9 Conclusion

The evaluation confirms that the Social Fitness app meets its core objectives, providing a robust and user-friendly platform for community-based fitness tracking and social interaction. Despite some limitations, the app's foundations are solid, and it holds potential for further development and refinement. The lessons learned from the testing and user feedback will serve as invaluable inputs for future enhancements.

5.1.10 Technical Robustness

The application was engineered with an emphasis on handling unexpected user behavior and inputs. Exception handling mechanisms were implemented to capture runtime errors, and comprehensive logging facilitated the monitoring of system behavior. We adopted a defensive programming approach, especially in critical features like data submission and retrieval, to ensure graceful degradation of services in case of failure.

5.1.11 Scalability

As the user base grows, the application's capability to scale effectively remains a crucial consideration. We have employed Firebase's cloud services, which offer a high degree of scalability for our real-time database operations. Load testing, albeit preliminary, has shown that our current infrastructure can support a significant number of concurrent users without a noticeable dip in performance.

5.1.12 Security

Data security has been a paramount concern throughout the development of the Social Fitness app. We incorporated industry-standard practices, such as OAuth for authentication and HTTPS for secure data transmission.

5.1.13 Limitations and Future Work

Despite the Social Fitness app's successful deployment, there are several areas earmarked for improvement and expansion. Acknowledging these limitations not only sets the stage for future versions of the application but also demonstrates a commitment to continuous development and user satisfaction.

Authentication on Android Devices

Current challenges with Auth0 integration on Android highlight the complexity of cross-platform compatibility. Future iterations will aim to streamline authentication across all devices, ensuring a secure and seamless user experience.

Capacitor Camera Integration

Future releases will explore the integration of the Capacitor Camera API, enabling users to capture and upload images directly from their devices. This addition will enhance the social aspect of the app, allowing users to share their fitness progress and milestones visually.

- **Real-Time Photo Sharing:** Implementing real-time photo sharing capabilities for profile updates and post creations.
- **Enhanced User Engagement:** Encouraging user interaction by facilitating a more visually engaging platform.

Media Uploads in Messaging

To foster a richer community experience, subsequent updates will include media upload features within group chats. This functionality will allow users to share videos and images, thus providing a more comprehensive communication platform.

- **Diverse Communication:** Enabling a broader range of expression and information sharing among group members.
- **Improved Usability:** Enhancing the user interface to accommodate media uploads intuitively and efficiently.

Customization of Group Pages

Recognizing the importance of user personalization, plans are in place to allow for the customization of group pages. This will empower users to tailor the aesthetic and functionality of their group spaces to their preferences.

- **Personalized User Experience:** Providing tools for users to modify the layout and design of their group interfaces.
- **Empowering User Creativity:** Enabling a platform where users can creatively express the identity of their fitness communities.

Machine Learning for Personalized Recommendations

Investing in machine learning technology will allow us to offer personalized workout and health recommendations. By analyzing user data, the app will be able to suggest tailored fitness plans and dietary advice.

- **Data-Driven Insights:** Utilizing user data to generate customized fitness insights and recommendations.
- **Enhanced User Retention:** Providing a value-added service that encourages regular user engagement with the app.

These enhancements and features are envisioned to not only improve the current user experience but also to extend the app's functionality, making it a comprehensive tool for the fitness community. Through iterative development and user feedback, Social Fitness will evolve to meet and exceed the expectations of its users.

5.1.14 Reflection on Development Practices

Reflecting on our development practices, the Agile methodology facilitated flexibility and responsiveness to change. However, a need for better sprint planning and backlog management was needed during this development practice.

5.1.15 Ad Hoc Testing and Iterative Development

A responsive, ad hoc testing methodology characterized the development of the Social Fitness app. Without formalized testing protocols, features were immediately tested upon implementation. Issues were addressed swiftly, ensuring each addition to the app contributed to its stability.

- **Immediate Issue Resolution:** Direct testing and quick fixes ensured robustness and functionality, preventing small problems from escalating.
- **Iterative Enhancement:** Continuous refinement based on testing feedback polished the app's core functionalities.

- **User-Centered Feedback:** Informal user feedback was integral to the development, highlighting usability enhancements.

While unconventional, this testing strategy was crucial for a reliable application at launch, exemplifying a practical quality assurance approach.

5.1.16 Reflective Evaluation

Reflecting on the process, more structured testing methods could have been beneficial and will be considered for future projects.

Chapter 6

Conclusion

This dissertation has presented the development of the Social Fitness app, an integrated platform designed to enhance the experience of fitness enthusiasts and teams through robust digital interaction and activity tracking. The concluding chapter summarizes the key aspects of the project and reflects on the findings from the System Evaluation chapter.

6.0.1 Context and Objectives Recap

The Social Fitness app was conceptualized from a clear need within the sports community for a unified communication and activity tracking platform. The app's development was guided by objectives such as cross-platform compatibility, user-friendly design, group engagement features, and the integration of fitness data from wearable devices. The agile methodology, complemented by continuous testing and iteration, facilitated a responsive development process that adapted to emerging user needs and technological challenges.

6.0.2 System Evaluation Findings

The System Evaluation chapter provided a comprehensive analysis of the app's performance against the objectives set forth. The rigorous testing regime confirmed the app's robust authentication system, critical for user security and trust.

- The app successfully met its core objectives, establishing a cross-platform, user-friendly interface with robust group interaction and personal fitness tracking features.
- Authentication and user profiles were rigorously tested, proving stable and secure.

- Real-time chat functionality enhanced community engagement.
- Integration with Google Maps API enabled dynamic route tracking for runners.
- Challenges, like adapting to real-time data retrieval and API integrations, were addressed through innovative solutions, reinforcing the app's technical robustness.

6.0.3 Beyond the Objectives: Serendipitous Discoveries

Throughout the development, serendipity played its part, with unplanned discoveries shaping the project's direction. The integration of real-time chat became a cornerstone of the app's social features, greatly surpassing its original intent. Similarly, the interactive capabilities of the Run Tracker page developed beyond expectations, becoming a highly praised feature among testers.

6.0.4 Opportunities for Future Investigation

Several areas have been identified for future enhancement:

- The integration challenges with Auth0 on Android devices highlighted the need for a more robust cross-platform authentication solution. Future work will explore alternative methods or enhanced integration techniques to provide a frictionless authentication experience.
- Users' desire to share their fitness milestones and moments calls for advanced media capabilities within group chats. Future updates will aim to incorporate live photo and video sharing, enhancing the social fabric of the app.
- User personalization is a crucial component of engagement. Allowing users to customize the aesthetic and features of their group pages can provide a unique and personal touch to each community within the app.
- By leveraging machine learning algorithms, the app can analyze user data to provide personalized fitness and health recommendations, transforming the app into a proactive fitness companion.

6.0.5 Reflection of Development Practices

The Agile development framework championed in this project promoted a flexible and responsive design philosophy. It allowed the team to pivot as needed and incorporate user feedback efficiently. However, the experience also highlighted

the need for improved sprint planning and backlog management, which will be a focus for future projects to ensure even more streamlined and focused development cycles.

6.0.6 Conclusion and Future Outlook

In closing, the Social Fitness app represents a harmonious blend of technology with the vibrant world of fitness communities. While it has made significant strides toward its initial goals, the path forward is ripe with opportunities for enhancement. The project team is committed to a journey of continuous improvement, with the hopes and aspirations of its users guiding the way. The future iterations of the Social Fitness app will continue to evolve, striving not just to meet but exceed user expectations, facilitating a fitness experience that is as socially enriching as it is physically rewarding.

Appendix A

Appendices

A.1 GitHub Url:

- Social Fitness App Repository: `https://github.com/DanielFitzsimons/FinalYearProject`

Bibliography

[1] Ionic framework documentation, 2024.

[2] Vue js.

[3] Flutter. Flutter docs.

[4] Firebase documentation, 2024.

[5] Aws amplify documentation, 2024.

[6] Node.js. https://nodejs.org/en.

[7] Google maps platform documentation, 2024.

[8] Mapbox documentation, 2024.

[9] DEV Community. Google maps on your ionic4 project w/ angular. `https://dev.to`, 2021.

[10] Freaky Jolly. Ionic 6 google map, geocoder & places search, draggable marker tutorial. `https://www.freakyjolly.com/ionic-google-map-demo/`, 2021.

[11] Capacitor Documentation. Introduction to capacitor. `https://capacitorjs.com/docs/v3`, 2024.

[12] Capacitor Documentation. Camera api. `https://capacitorjs.com/docs/apis/camera`, 2024.

[13] M. Lynch. Adding camera functionality to your ionic app with capacitor. `https://ionicframework.com/blog/adding-camera-functionality-to-your-ionic-app-with-capacitor`, 2021.

[14] Max Lynch. Capacitor: Cross-platform native runtime for web apps. `https://ionicframework.com/blog/announcing-capacitor-3-0`, 2021.

[15] Capacitor Documentation. Building cross-platform apps with capacitor. https://capacitorjs.com/docs, 2024.

[16] Capacitor Documentation. Apis and native tools integration. https://capacitorjs.com/docs/apis, 2024.

[17] M. Lynch. Why capacitor is the future of hybrid app development. https://ionicframework.com/blog/why-capacitor-is-the-future-of-hybrid-app-development, 2020.

[18] Capacitor Documentation. Using capacitor plugins. https://capacitorjs.com/docs/plugins, 2024.

[19] M. Lynch. The power of capacitor plugins. https://ionicframework.com/blog/the-power-of-capacitor-plugins, 2021.

[20] Capacitor Documentation. Web framework integration. https://capacitorjs.com/docs/web, 2024.

[21] M. Lynch. Capacitor and modern web frameworks. https://ionicframework.com/blog/capacitor-and-modern-web-frameworks, 2021.

[22] Google. Maps javascript api. https://developers.google.com/maps/documentation/javascript/overview, 2020.

[23] Google. Place searches. https://developers.google.com/places/web-service/search, 2020.

[24] M. Smith. Enhancing mobile app user experience with google maps api, 2021.

[25] A. Lee. Balancing performance and privacy in location-based applications, 2021.

[26] Google. Google fit for developers. https://developers.google.com/fit.

[27] Sesnors with google fit.

[28] Record data with fit.

[29] Working with datasets.

[30] user permissions.