

Development of the WeShre Platform for Enabling Real-Life Social Connections

Auteur : Altaha, Yaman

Promoteur(s) : Mathy, Laurent

Faculté : Faculté des Sciences appliquées

Diplôme : Master en ingénieur civil en informatique, à finalité spécialisée en "intelligent systems"

Année académique : 2023-2024

URI/URL : <http://hdl.handle.net/2268.2/20443>

Avertissement à l'attention des usagers :

Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.

Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.



MASTER THESIS

UNIVERSITY OF LIEGE - FACULTY OF APPLIED SCIENCES

CIVIL ENGINEERING - MASTER IN COMPUTER SCIENCE

Development of the WeShre Platform for Enabling Real-Life Social Connections

Thesis conducted for obtaining the Master's degree in Computer
Science and Engineering by Yaman Altaha

Author: **Yaman ALTAHA**

Supervisor: **Prof. Laurent MATHY**

Academic year 2023-2024

Acknowledgements

First and foremost, I would like to express my sincere gratitude to WeShre for providing me with the invaluable opportunity to undertake this thesis.

I extend my sincere thanks to Mr. Simon, the CEO of WeShre, for his guidance and direction throughout this project.

Special thanks to Mr. Nizeyimana, the CTO, for his insightful supervision and constant presence throughout the entire development and completion of this thesis.

I am particularly grateful to Professor Mathy for agreeing to be my thesis advisor and for his wise guidance and precise feedback.

Additionally, I would like to acknowledge the use of AI tools, in particular ChatGPT, which assisted with brainstorming, rephrasing, translation, and coding aspects of this project. These tools helped to improve the clarity and quality of my work. However, all the content and final edits were reviewed and approved by me.

Finally, I would like to thank my family and friends for their encouragement and support during the course of this thesis.

Abstract

Human communication is one of the most important factors influencing our daily lives and determining the success of our relationships. With the development of digital devices as well as social media, which have become dominant in the world, the importance of face-to-face interaction has never been more critical. The WeShre app is a distinctive platform designed to facilitate real human interaction and bridges the gap between the virtual and the real-world experiences, encouraging real-life encounters and drives interactions between individuals.

This master's thesis project aims to develop and adapt the WeShre platform for both web-based and mobile environments, with a specific focus on the participant side. The application is designed to include a homepage that provides users with immediate access to all events, news, and the ability to search for specific events. The profile page allows users to view their personal details, events, and groups, while a dedicated login page ensures secure access to the platform. The events page offers detailed information about each event, providing a comprehensive overview of the event. Additionally, the calendar page keeps users up-to-date by displaying event dates attractively for each month. The group page provides users with information about the group and its members, as well as details of any events organised by the group.

Table of Contents

1	Introduction	1
2	Context	2
2.1	Presentation of the company	2
2.2	Problem statement and Objectives	2
3	Frontend Application architecture and used technologies	4
3.1	Unifying Mobile and Web Development through Expo	4
3.2	Workflow Optimisation with Expo	5
3.3	Using Expo Go for development and testing	7
3.4	File System Routing and Navigation with Expo Router	8
3.5	Optimising Styling and Build Processes with Metro	12
3.6	Leveraging Tamagui Library for Efficient and Scalable Design	13
3.7	Utilising Basic Tamagui Components and Providers	13
3.8	Custom Themes and Responsive Design Implementation with Tamagui	15
3.9	Integrating Tamagui Design System with Babel for UI Enhancement	17
4	Backend System Architecture and Data Migration Process	19
4.1	Introducing Supabase for Efficient and Scalable Backend Solutions	19
4.2	Supabase Architecture and Core Components	20
4.3	Advanced Backend Configuration with Supabase	21
4.4	Evaluating Firebase and Supabase Before Migration process	23
4.5	Detailed Steps in Database Migration from Firebase to Supabase	24
5	Database Architecture and Schema	26
5.1	Overview of Database Architecture	26
5.2	Detailed Database Schema	26
5.2.1	Table Descriptions	26
5.2.2	Entity-Relationship Diagram (ERD)	28
6	Design and Implementation of User Interfaces in WeShre	30
6.1	System Overview with High-Level Architecture	30
6.2	Responsiveness and Platform Adaptability	30
6.3	Login Page Interface	32
6.3.1	Page Overview	32
6.3.2	Web and Mobile Implementation	32
6.4	Home Page Interface	32
6.4.1	Page Overview	32

6.4.2	Web and Mobile Implementation	34
6.5	Calendar Page Interface	35
6.5.1	Page Overview	35
6.5.2	Web and Mobile Implementation	35
6.6	Profile Page Interface	38
6.6.1	Page Overview	38
6.6.2	Web and Mobile Implementation	39
6.7	Event Page Interface	39
6.7.1	Page Overview	39
6.7.2	Web and Mobile Implementation	41
6.8	Group Page Interface	43
6.8.1	Page Overview	43
6.8.2	Web and Mobile Implementation	43
7	Testing	46
7.1	Testing with Cypress	46
7.2	End-to-End Test Execution and Results	47
7.3	Interpreting Test Outcomes	48
8	Deployment	50
8.1	Webmin Configuration and Virtualmin Setup	50
8.2	Project Deployment Process	50
8.3	Verifying Deployment Success	51
9	Conclusion	52
9.1	Project Conclusion	52
9.2	Limitations	53
9.2.1	Migration challenge	53
9.2.2	Tamagui Limitations	53
9.2.3	Expo Compatibility Issues	54
9.3	Future work	54
9.3.1	Future Interface Additions	54
9.3.2	Expanding Authentication Methods	55
9.3.3	Adding Notification System	55
9.3.4	Integrating Ratings and Comments	55
Appendix A	Appendix	59
A.1	Responsive Web Interfaces on Mobile Devices	59

List of Figures

3.1	Expo Logo Medium	4
3.2	React Native Compilation Flow Retool	5
3.3	Terminal commands to create and open the 'weshre-app' project using Expo	5
3.4	Screenshot of Initial project setup with Expo project template	6
3.5	Expo Go QR code in Visual Studio Code to run 'weshre-app' for mobile testing.	7
3.6	Project running in Expo Go app Retool	8
3.7	Expo Go Flow Retool	8
3.8	Expo Go application interface for running WeShre app	9
3.9	Screenshot of the File System Routing in the project	10
4.1	Screenshot of Supabase Table View Interface	20
4.2	Supabase Architecture and Core Components	21
5.1	Entity-Relationship Diagram (ERD) for the WeShre database	29
6.1	High-Level Architecture Diagram of the WeShre Platform	31
6.2	Mobile version of the login page.	33
6.3	Web version of the login page.	34
6.4	Web version of the home page with large event and blog cards.	35
6.5	Mobile version of the home page with a single card view and horizontal scrolling.	36
6.6	Mobile calendar page with individual event cards and horizontal scrolling.	37
6.7	Web calendar page with a wide view and multiple event cards.	38
6.8	Mobile profile page showing user info, activities, sports skills, and more details.	39
6.9	Web profile page displaying user info, activities, sports skills, and more details.	40
6.10	Mobile version of the event page displaying event details.	41
6.11	Web view of the event page displaying event details.	42
6.12	Mobile version of the group page displaying group information, events, and members list.	43
6.13	Web group page displaying group information, events, and members list.	45
7.1	Cypress Application	47
7.2	Cypress Folder Structure in Project Directory	48

7.3	Cypress Test Runner Output showing successful end-to-end test execution from the testing project.	49
8.1	Webmin File Manager interface after uploading the project files to the server.	51
A.1	Responsive Login page design for smaller screens.	60
A.2	Responsive design of the home page on smaller screens.	61
A.3	Responsive calendar page design for smaller screens.	62
A.4	Responsive design of the profile page for smaller screens.	62
A.5	Responsive event page design for smaller screens.	63
A.6	Responsive group page design for smaller screens.	63

List of Tables

4.1	Comparison of Firebase and Supabase Features. Adapted from Flatirons Development and White Lotus Corporation	24
5.1	Detailed Summary of Database Tables and Key Fields in the WeShre Platform	28

List of Abbreviations

API	Application Programming Interface.
UI	User interface.
UX	User Experience.
SDK	Software Development Kit.
iOS	Operating System for Apple's mobile devices.
NPM	Node Package Manager.
NPX	Node Package Execute.
QR	Quick Response Code.
LAN	Local Area Network.
JSON	JavaScript Object Notation.
API	Application Programming Interface.
CLI	Command-Line Tool.
TS	Type Script.
CSS	Cascading Style Sheets.
LAN	Local Area Network.
ESN	Erasmus Student Networks.
SSL	Secure Sockets Layer.
DNS	Domain Name System.

1 | Introduction

Face-to-face communication is crucial for building strong relationships and enhancing understanding. The fast-paced evolution of information and communication technologies facilitates on one hand the virtual interaction between people, but on the other hand, it leads to weaker real-life interaction.

Gathering people in a well-organised event and in an optimised environment can be tiresome for many. This is where the idea of WeShre app [59] came from. The app serves as a powerful, advanced tool for those looking for meaningful experiences and real encounters in the physical world. Through this app, users can participate in parties, unique experiences, festivals, cultural events, and small private sporting gatherings. These events are managed by the event organiser and accessible to the app users, allowing them to easily meet new people, as well as organising their own events and share them on the app. In addition, users can join groups within the app to keep up to date with group-related events and members.

The objective of this thesis is to develop a unified cross-platform application for the WeShre platform that facilitates real-life social interactions. Using the Expo platform [13], the goal is to create a new version that enhances and extends the capabilities of the existing mobile application, providing seamless functionality on web browsers, iOS, and Android devices. This will ensure a consistent user experience, regardless of whether the application is accessed through a web browser or a mobile device.

Throughout this thesis, the development of a multi-platform application starting from scratch will be presented. A later section will demonstrate in more detail how the Expo platform facilitates this development.

This project shows how cutting-edge technologies like Expo, React [32], and React Native [30] can be used to create a comprehensive application that works across different platforms simultaneously. These technologies were chosen to ensure that the application remains flexible, scalable, and easy to maintain. It is important to note that all the technologies used in this thesis were chosen by the company. This decision was made to align with the company's strategic goals and to build a robust foundation for future development.

2 | Context

2.1 Presentation of the company

WeShre is a Belgian startup offering an all-in-one platform designed to find, create and manage various types of events. The platform also facilitates connections among users within their community and beyond.

WeShre collaborates with universities, student associations, and student networks across Belgium, Spain, the Netherlands, and Quebec. These partnerships allow the company to provide tailored solutions that meet the specific needs of these groups.

The company was co-founded by Hugo Simon and Bill Nizeyimana. Their vision was to create a tool that brings people together and makes event management easier.

2.2 Problem statement and Objectives

The WeShre platform has demonstrated success in facilitating real social connections by meeting users' desires to either participate in or organise a variety of events, including celebratory, sports, and also cultural gatherings.

The main challenge addressed in this thesis is the limitations imposed by the platform's exclusive availability on mobile devices, which restricts user accessibility and participation.

This project aims to overcome these barriers by expanding the platform's reach to include not only mobile devices but also web browsers. This will provide a more inclusive and accessible way for users to connect. Despite the successes of the existing application, meeting the needs of the Erasmus Student Networks (ESN) has been a major challenge. These users, who play a crucial role in organising and participating in events specifically tailored for Erasmus students, have expressed the importance and necessity of extending the current application to include web access.

Although the existing mobile application functions well, extending access to a web-based interface will increase convenience, speed and accessibility for Erasmus students wishing to participate in ESN events. Therefore, developing a responsive, efficient, and user-friendly version of the platform that operates on both web and mobile devices has become essential. This approach will ensure that the

WeShre platform can effectively meet the diverse needs of all users. The aforementioned developments lead to the following objectives:

- **Improve accessibility:** Create a web platform that is easy to navigate and accessible from any device.
- **Increase the number of users:** Provide a web platform for users who have limited storage space on their devices, offering an alternative to downloading the mobile app.
- **Maintain consistency:** Ensure that the web version mirrors the functionality and user experience of the mobile application, so that the experience is standardised and consistent across all platforms.
- **Design for all users:** Implement a responsive design for all platform users, focusing on accessible and user-friendly interfaces that make it easier to discover, attend, and organize events.
- **Unified codebase:** Create a joint codebase for both web and mobile platforms, allowing WeShre to replace its current application, which cannot be extended to the web. This unified codebase will facilitate the addition of new features and improve the platform over time.

Ultimately, WeShre aims to become an integrated tool that meets the expectations of all users, including the Erasmus community, who are frequent users of the application. This expansion will not only improve the user experience but also enhance the platform's potential to strengthen social connections among different user groups.

3 | Frontend Application architecture and used technologies

3.1 Unifying Mobile and Web Development through Expo

Expo [13] is an open free source toolset for building universal native apps on iOS, Android and Web with a single React codebase. As Expo is a JavaScript framework built on top of the React Native framework, it builds React Native apps much faster and more efficient. Companies such as Facebook, Walmart and Airbnb are already using Expo in production. Figure 3.1 shows the Expo logo, which represents the technology we used to build the application.

Using Expo, there will be no specific folders for Android and iOS within the files (missing iOS and Android directories), therefore Expo will handle all the configuration. Expo has developed a technique called Continuous Native Generation [6] where native code is generated predictably from a set of inputs, such as application settings, platform-specific configurations and so on, ensuring automatic code generation for iOS and Android, that simplifying the development process.

As it is a set of tools and services for React Native, it has largely improved the developer experience. It provides a list of native libraries [55] that can be used without the need to install them. On top of that, it allows building an application from start to finish that runs on any computer and phone in a reasonable time, which is what most developers are looking for. Simply, Expo is easier to use, faster to test, and produces neater code.



Figure 3.1: Expo Logo [Medium](#)

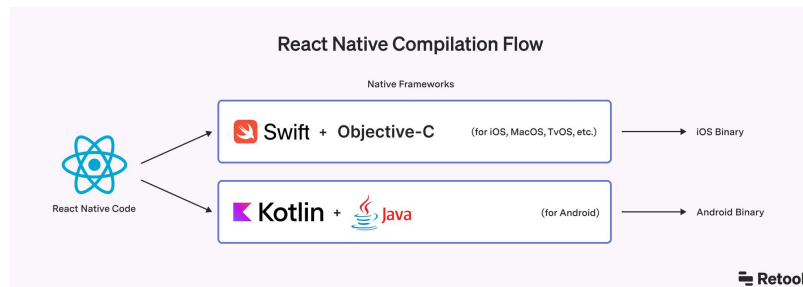


Figure 3.2: React Native Compilation Flow [Retool](#)

```
# create an expo app called weshre-app
npx create-expo-app weshre-app

# navigate to the newly created project folder
cd weshre-app

# open project folder on VS code
code .
```

Figure 3.3: Terminal commands to create and open the 'weshre-app' project using **Expo**.

In general, React Native compiles JavaScript into two codebases: Swift, Objective-C codebase and Kotlin, Java codebase. These two codebases are then natively compiled into iOS and Android binaries, after which they become usable on smartphones. Figure 3.2 illustrates this process.

React Native solves this issue by using a JavaScript bridge that allows React components to represent native UI components, However, React Native apps can offer a more native experience than traditional native apps because they tend to rely less on web views. Moreover, one of the main reasons is the necessity to use third-party libraries (which will be discussed in Section 3.3), since the core React Native libraries are fairly limited in scope. This is where Expo comes in.

3.2 Workflow Optimisation with Expo

This thesis project was built from scratch. It was initially created using Expo's command-line, specifically the "npx create-expo-app my-app" command. By running this command-line, it generates a basic project template and handles all configuration to start writing code immediately. Figure 3.3 shows the commands executed in the terminal to generate a basic project structure and open it in Visual Studio Code for file and folder exploration.

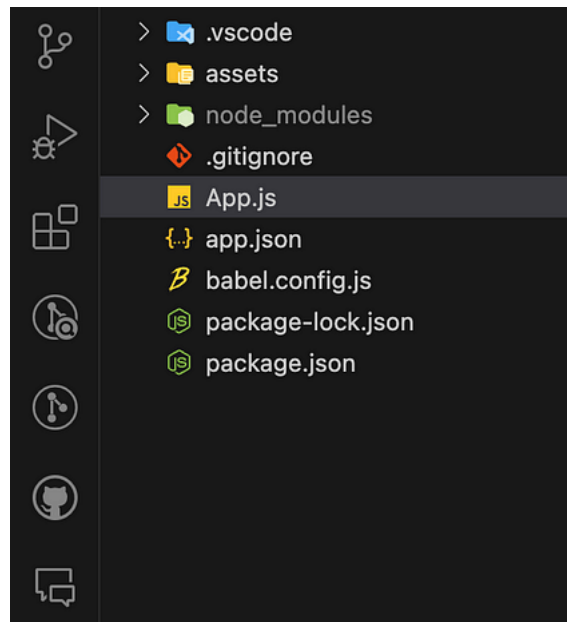


Figure 3.4: Screenshot of Initial project setup with Expo project template

The default codebase created by Expo when the project is started from scratch is shown in Figure 3.4.

Let's take a look at some of the main initial files and folders that are created:

- **assets [23]**: This directory contains the application splash icon and static assets related to the application icon.
- **App.js**: The initial point of entry for the Expo React Native application.
- **app.json [12]**: contains the configuration for the Expo build setup.
- **babel.config.js**: Babel configuration file for converting and packaging JavaScript code. Its function and configuration in our project will be examined in Section 3.9, where we will discuss its integration with the Tamagui library.
- **package.json**: Contains package information and scripts for run, test, and build commands. The integration of the Tamagui library through dependencies listed in this file will also be explored in Section 3.9.

By simply running `npm start` command it can instantly run the app on iOS, Android and the web. On the code side, if any changes are made to the user interface, the app will immediately reload to reflect the changes. Additionally, the app can be tested in a sandbox environment through the use of tools such as Expo Go, which will be explained in the following next section.

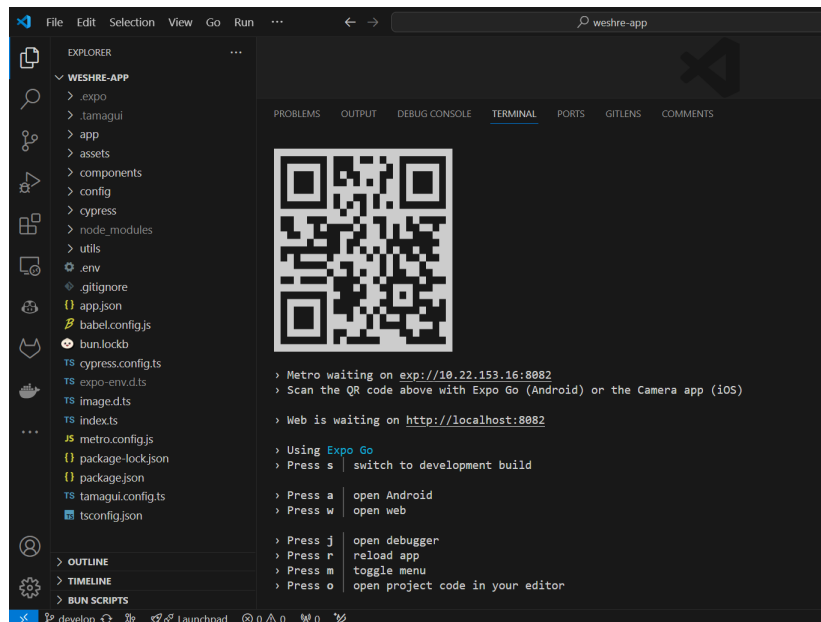


Figure 3.5: Expo Go QR code in Visual Studio Code to run 'weshre-app' for mobile testing.

3.3 Using Expo Go for development and testing

Expo Go [14] is an open-source application that is publicly available on the App Store and Play Store. It is the ideal solution for testing apps, as it enables the use of a scannable QR code, such as the one displayed in the Visual Studio Code terminal as shown in Figure 3.5. This provides immediate access to the application on a mobile device through Expo Go. The application is available on the official mobile app store and includes all the core Expo SDK libraries that enable common mobile functionality such as the camera, calendar, keyboard, contacts, video, audio, etc. Furthermore, the underlying native libraries of these custom SDK libraries are included in the Expo Go Configuration, thus ensuring their compatibility and functionality with Expo Go. Through a QR code, the project can be easily linked and accessed. In other words, the Expo Go application runs an Expo project by importing a build over a local area network (LAN) or localhost connection, as shown in Figure 3.6.

In general, after setting up the project, the application is typically run using `npm start`. This will open Expo DevTools in the browser, where a QR code can be observed, and by scanning this QR code using the Expo Go app, which is downloaded on the physical device, the live app can be viewed directly on the device.



Figure 3.6: Project running in Expo Go app [Retool](#)

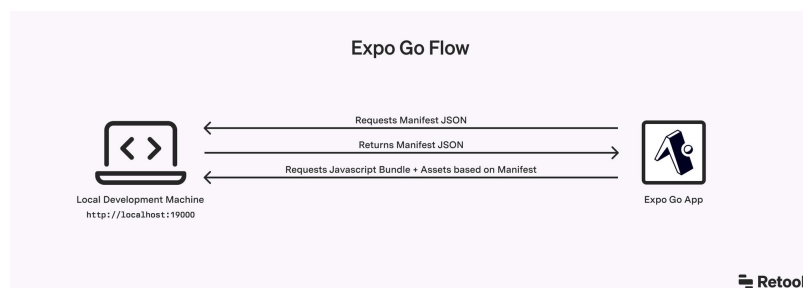


Figure 3.7: Expo Go Flow [Retool](#)

However, the question remains as to how this process operates. In fact, after scanning the QR code, Expo Go connects to the Expo dev server of the local development machine and requests a `manifest.json`. Afterward, Expo Go downloads the JavaScript bundle and any assets required to display the React Native application. Figure 3.7 shows how this process works.

Once changes are saved on the local development device, the Expo Go app automatically loads them via wireless technology. Figure 3.8 illustrates a screenshot from the Expo Go application, which is an effective solution for running and testing the application and provides a visual representation of the application interface used during development.

3.4 File System Routing and Navigation with Expo Router

The Expo code is unique in that it uses File System Routing, similar to Dynamic Routes, as shown in Figure 3.9, which is very similar to the React Framework. This allows us to create screens that are automatically deep linkable on mobile devices, web applications, and we can easily navigate between them in the UI with the link component. In Expo, the Router module [22] provides a straightforward

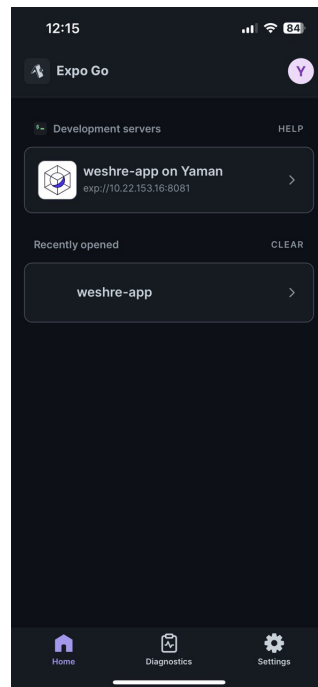


Figure 3.8: Expo Go application interface for running WeShre app

and intuitive way to control this behaviour.

However, Expo Router is a file-based router designed for React Native and web applications. It allows the management of navigation between screens in the application, so that users can move seamlessly between different parts of the application's user interface using the same components across multiple platforms, such as Android, iOS, and web. The code snippet from our Expo Router is presented in Listing 3.1.

```
1 const Router = () => {
2   const navigation = useNavigation();
3   const routes: RouteItem[] = [
4     { path: "index", screen: <Login {...{ navigation:
5       navigation }} /> },
6     { path: "event/[event]", screen: <Event /> },
7   ];
8   return (
9     <SafeAreaView style={{ height: "100%",
10      backgroundColor: "white" }}>
11     <Stack>
12       {routes?.map((item, key) => (
```

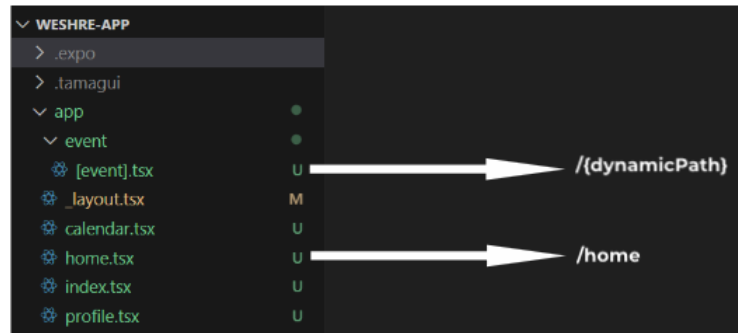


Figure 3.9: Screenshot of the File System Routing in the project

```

11     <Stack.Screen key={key} name={item?.path} />
12     ))}
13   </Stack>
14   </SafeAreaView>
15   );
16 };
17 export default Router;

```

Listing 3.1: Expo Router Code Snippet

In addition, the Expo router can also use a layout route [1] to share the user interface across multiple pages, as well as represent more advanced routing patterns. Since the files are organised into folders and a layout route is created, which adds shared UI elements such as stack bars and tab bars, and is built on top of React navigation [31], we can access all of the same powerful features, such as shared element transitions or native stack navigators. Besides, the Root Layout (app/_layout.js) can be used to add providers which can be accessed by any route in the application. The implementation of this functionality is demonstrated in Listing 3.2. The `TamaguiProvider` component, which is used in this layout, will be discussed in Section 3.7.

```

1 export default function RootLayout() {
2
3   const [loaded] = useFonts({
4     Inter: require("@tamagui/font-inter/otf/Inter-
      Medium.otf"),

```

```
5     InterBold: require("@tamagui/font-inter/otf/Inter-
        Bold.otf"),
6   });
7
8   if (!loaded) return null;
9
10  return (
11    <Provider>
12      <TamaguiProvider config={config}>
13        <Main/>
14      </TamaguiProvider>
15    </Provider>
16  );
17 }
```

Listing 3.2: Expo Router Layout Code

Expo Router offers more than just navigation. It is also highly beneficial for TypeScript. While Expo CLI [11] already has exceptional TypeScript capabilities, TypeScript files can be added at any stage during the development process. Additionally, Expo CLI will automatically set up and install TypeScript dependencies, configure the TS configuration, and augment the React native types so that they are compatible with web.

To illustrate how Expo CLI effortlessly integrates TypeScript into our project, consider the `tsconfig.json` file in Listing 3.3, which is taken from our project's code. This file is automatically generated and configured by Expo CLI when TypeScript files are introduced.

```
1 {
2   "extends": "expo/tsconfig.base",
3   "compilerOptions": {
4     "strict": true
5   },
6   "include": [
7     "**/*.ts",
8     "**/*.tsx",
9     ".expo/types/**/*.ts",
10    "expo-env.d.ts",
11    "components/mobileMap/MobileMap.native.js",
12    "components/map/WebMap.web.js"
13  ]
}
```

14 }

Listing 3.3: Customised tsconfig.json with Expo CLI Defaults and Manual Additions

3.5 Optimising Styling and Build Processes with Metro

In addition to TypeScript integration, Expo CLI also extends the styling capabilities of our project. The key component in this process is the `metro.config.js` file [15], which is automatically created by Expo. It configures the Metro bundler [7], which is responsible for packaging the source code (including JavaScript, TypeScript, JSON, and other files used by the application) and assets (such as images and fonts) for the project.

Further, Metro is a JavaScript bundler specially designed for React Native applications, including those created with Expo. It can be considered as a tool that organises all the necessary components into a single easily accessible package for the application to understand and display.

The `metro.config.js` file we have in the project directory, shown in Listing 3.4, is specifically configured to enable CSS styling for web applications. This is important since it allows us to use CSS directly in our Expo project. This feature is not enabled by default; however, it can be enabled through the configuration file, showing how Expo CLI makes the development experience more flexible and web-friendly.

```
1 // Enables CSS support in Metro for web-specific
  styling.
2 const { getDefaultConfig } = require("expo/metro-
  config");
3
4 const config = getDefaultConfig(__dirname, {
5   isCSSEnabled: true, // Turns on CSS support.
6 });
7
8 module.exports = config;
```

Listing 3.4: Metro configuration enabling CSS support for web projects within the Expo framework

By enabling CSS support (`isCSSEnabled: true`), this configuration allows us to style our application using CSS, which is useful when creating or optimising the application for web platforms.

3.6 Leveraging Tamagui Library for Efficient and Scalable Design

Having explored and discussed the functionality of Expo and its important and effective role in developing our cross-platform application, we still have an important aspect of our structure to focus on, which is user interaction and how to make the application usable, allowing users to interact with it, ensuring that it works well, and providing a robust user experience. This leads us to the incorporation of the Tamagui [47] library, which serves as a single codebase for creating cohesive and responsive designs across both web and mobile platforms. Thus, in order to create a scalable application, special attention and effort were given to designing the UI and UX using the Tamagui library.

Tamagui is a user interface framework that offers a lightweight design system that easily adapts to different screen sizes and devices. It also includes a collection of optimised components, and a style library with a series of themes, media queries, animations, responsive and typed inline styles, and other features.

The easiest way to install Tamagui is to use the following starter template:

```
npm create tamagui@latest
```

Furthermore, the optimising compiler in Tamagui flattens the component tree and outputs minimal CSS. This library allows developers to write once and deploy high quality designs everywhere, and integrates well with Expo's cross-platform features, ensuring that the app looks consistent overall.

Since the goal of our project is to provide users with an effective and seamless experience, regardless of the platform they are using. Therefore, using the Tamagui library fits well with the requirements of our project. Using the library's ability to design themes and style props allows us to be able to maintain design consistency and take advantage of library performance optimisations for faster rendering times.

In the following sections, we will look in more detail at how Tamagui has been used in our various application components, showing how it has influenced the development process and overall user experience.

3.7 Utilising Basic Tamagui Components and Providers

Our application's user interfaces were developed using the basic components that Tamagui provided, which allowed us to modify them to meet the needs of our

project. Although Tamagui provides `YStack` and `XStack` for horizontal and vertical alignment, the versatile **View** component was mostly employed during the design phase, making it more compatible with React Native practices.

One of the advantages of this library is also its ability to simplify the code and improve readability by offering concise shortcuts for props [50]. Some of the shortcuts that were frequently used when building the application are as follows:

- `f` for `flex`, defines the layout flexibility of a component,
- `jc` for `justifyContent`, aligns within a container along the main axis,
- `ai` for `alignItems`, aligns along the cross axis,
- `p` for `padding`, to specify the padding within a container,
- `m` for `margin`, to set the space around components,
- `maw` for `maxWidth`, to control the maximum width of an element.

Additionally, all Tamagui components, such as `paragraph`, `button`, `text`, etc., aim to provide a unified styling across platforms and render the appropriate native components on both web and native platforms. On the other hand, one of the key components in our application is the **TamaguiProvider** [50]. This component wraps all the other components to effectively integrate the settings of the Tamagui library. This is done by importing the **TamaguiProvider** and applying it as shown in our `RootLayout` function in Listing 3.5. This ensures that our design system, including the fonts and attributes defined in `tamagui.config.ts` (which will be explained in detail in the upcoming sections), is applied consistently throughout the application. With this configuration, we can ensure that the entire application has a uniform look and feel.

```
1 import { TamaguiProvider } from "tamagui";
2 ...
3 export default function RootLayout() {
4   ...
5   return (
6     <Provider>
7       <TamaguiProvider config={config}>
8         <Main/>
9       </TamaguiProvider>
10    </Provider>
11  );
12 }
```

Listing 3.5: Integrating **TamaguiProvider** in `RootLayout`

3.8 Custom Themes and Responsive Design Implementation with Tamagui

To further improve the design of our application, we used Tamagui's advanced customisation features to improve the design. This included using the styled library [51] for component customisation. This approach was applied to all components in the project.

For example, we customised a component to create a CardInfo layout, which organises and displays information in a card layout. This customisation was achieved using Tamagui's styled library as shown in Listing 3.6.

```
1   const CardInfo = styled(Stack, {
2     display: 'flex',
3     flexDirection: 'column',
4     justifyContent: 'flex-start',
5     marginLeft: 10
6   })
```

Listing 3.6: Customising the CardInfo Component Using Tamagui Styled Library.

To ensure a consistent theme throughout the application, Tamagui offers a predefined set of named Colour tokens [49] as part of its theme construction capabilities, ensuring uniformity in the colour scheme. Besides, by customising the default Tamagui configuration, we can also define our own custom colour tokens to match our desired theme.

The Tamagui library requires a configuration file [50] in order to handle themes and be flexible. Thus, the `tamagui.config.ts` file shown in Listing 3.7 sets up the default behaviour for Tamagui and comes with many predefined values. It also allows us to define the basic setup for Tamagui, providing the ability to specify the necessary default settings and customisations for the project requirements.

```
1   const config = createTamagui({
2     light: {
3       color: {
4         background: "gray",
5         text: "black",
6       },
7     },
8     defaultFont: "body",
9     animations,
10    shouldAddPrefersColorThemes: true,
```

```
11  themeClassNameOnRoot: true,  
12  shorthands,  
13  fonts: {  
14    body: bodyFont,  
15    heading: headingFont,  
16  },  
17  themes,  
18  tokens,  
19  media: createMedia({  
20    xs: { maxWidth: 660 },  
21    sm: { maxWidth: 800 },  
22    ...  
23    xl: { maxWidth: 1420 },  
24    xxl: { maxWidth: 1600 },  
25    ...  
26    gtLg: { minWidth: 1280 + 1 },  
27    short: { maxHeight: 820 },  
28    ...  
29    pointerCoarse: { pointer: "coarse" },  
30  } ),  
31 });  
32  
33 export default config;
```

Listing 3.7: Setting Up Tamagui Configuration for Theme and Responsiveness.

Within this configuration file (`tamagui.config.ts`), Tamagui provides extensive customisation, including defining global font styles and animations, setting up responsive breakpoints, and defining a light theme with predefined background and text colours. Following this, we can incorporate media queries into the UI elements of our application. In fact, in our project, we extensively used the **useMedia** [54] hook from Tamagui to apply responsive styles to our user elements, this way we were able to dynamically adapting the design based on the screen size of the device, ensuring an adaptable UI across different devices.

One example of using media queries in our project is shown in Listing 3.8

```
1  import { ..... , useMedia } from 'tamagui';  
2  
3  const media = useMedia();  
4  
5  const Card = styled(Stack, {  
6  marginBottom: 20,
```

```
7  marginTop: media?.sm || Platform.OS !== 'web' ? 20 :
    10,
8  display: 'flex',
9  flexDirection: 'row',
10 justifyContent: 'flex-start',
11 alignItems: 'flex-end',
12 });
```

Listing 3.8: Using the useMedia Hook for Responsive Styles

This is only one of many examples where we used the useMedia hook to ensure that our entire application has a responsive design.

3.9 Integrating Tamagui Design System with Babel for UI Enhancement

In order to incorporate the design system properties defined in the `tamagui.config.ts` file discussed earlier, the Babel plugin acts as a bridge through the `babel.config.js` file shown in Listing 3.9, loading and applying the configurations and customisations defined for the UI components within our application.

```
1  module.exports = function (api) {
2    api.cache(true);
3    const plugins = [];
4
5    plugins.push([
6      "@tamagui/babel-plugin",
7      {
8        components: ["tamagui"],
9        config: "./tamagui.config.ts",
10     },
11   ]);
12
13   plugins.push("expo-router/babel");
14
15   return {
16     presets: ["babel-preset-expo"],
17     plugins,
18   };
19 };
```

Listing 3.9: Babel Configuration for Tamagui and Expo Router Integration

Within this configuration file `babel.config.js`, we can see that Tamagui's base components are included in the `components` array, which corresponds to the architecture of our application that utilises the core UI elements provided by the library. In addition, the `babel.config.js` file highlights the integration with Expo to streamline development processes and improve application performance across platforms.

However, as can be seen from the snippet of our `package.json` file shown in Listing 3.10, the inclusion of certain packages in this file is necessary to integrate this library and its tools into our project. These dependencies guarantee that our project has access to the most recent features provided by the Tamagui library.

```
1 {
2   "name": "weshre-app",
3   "version": "1.0.0",
4   "scripts": {
5     "start": "expo start",
6     "android": "expo start --android",
7     "ios": "expo start --ios",
8     "web": "expo start --web"
9   },
10  "dependencies": {
11    "expo": "~49.0.11",
12    ...
13    "@tamagui/animations-react-native": "^1.79.2",
14    "@tamagui/avatar": "^1.79.13",
15    "@tamagui/font-inter": "1.74.8",
16    "tamagui": "^1.79.11",
17    ...
18  },
19  "devDependencies": {
20    "@babel/core": "^7.20.0",
21    "@tamagui/babel-plugin": "1.74.8",
22    ...
23  }
24 }
```

Listing 3.10: **package.json** - Essential Dependencies for Tamagui Integration and Other Tools

4 | Backend System Architecture and Data Migration Process

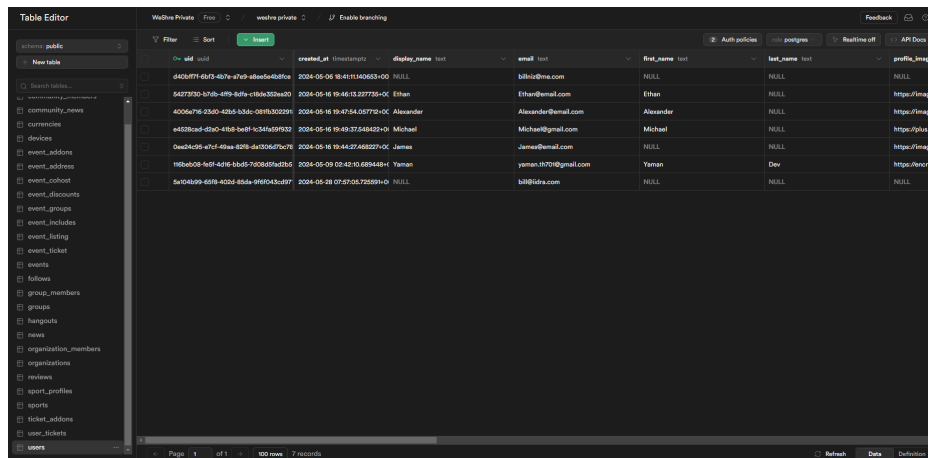
4.1 Introducing Supabase for Efficient and Scalable Backend Solutions

Supabase [33] is a complete backend for web and mobile applications based entirely on free open-source software. In general, the biggest challenge when building an application is designing a complete system that works effectively at scale. Products like Firebase, for example, have addressed this barrier, but the big issue is that they lock the user into proprietary technology on a specific cloud platform.

Supabase was created in 2019 specifically as an open-source alternative to Firebase [16], providing users with an all-in-one backend. Supabase is known as a backend-as-a-service, which means that it provides a full range of different backend services at a high level, such as:

- Database cloud functions authentication [37] for implementing granular access to data.
- Edge functions [40] running in the cloud to implement custom server-side functionality
- Database file storage [44] for efficient file storage and access, using an extensive server network to ensure faster data retrieval and optimal media handling.
- Realtime database updates [42] to enable real-time communication between clients and servers, allowing clients to be updated instantly as database changes occur.
- AI integration [34] for enabling advanced search and data handling capabilities.

However, on the user interface side, we have client-side SDKs that can easily connect this infrastructure to the front end of the user's choice, such as JavaScript frameworks, React Native, and many other platforms. These front-end applications can then simply connect to the back-end services we previously mentioned. In other words, Supabase is responsible for the majority of the processing on the back end. As a result, Supabase allows users to create a full-stack application with a fully functional and scalable backend.



The screenshot shows the Supabase Table Editor interface for a table named 'users'. The table has the following columns: id, email, created_at, display_name, first_name, last_name, and profile_image. The data is as follows:

id	email	created_at	display_name	first_name	last_name	profile_image
6403771-613-617e-4769-8be6e6b8f0ca	bllhiz@me.com	2024-05-06 18:41:14.0363+00	NULL	NULL	NULL	NULL
5477330-57db-4f9-8d9c-c18de32bea20	Ethan@gmail.com	2024-05-16 19:46:13.227735+00	Ethan	NULL	NULL	https://image
A006476-2340-4265-b3dc-087b320291	Alexander@gmail.com	2024-05-16 19:47:54.02772+00	Alexander	Alexander	NULL	https://image
e4528cad-d7a2-418a-b6ff-4c351a599231	Michael@gmail.com	2024-05-16 19:49:37.648422+00	Michael	Michael	NULL	https://pics.u
0ae24c95-47cf-49aa-8278-da356a976778	James@gmail.com	2024-05-16 19:44:27.668227+00	James	NULL	NULL	https://image
1f6b608-4d1-4216-bb65-7d0685d6285c	yaman1970@gmail.com	2024-05-09 02:42:10.689448+00	Yaman	Yaman	Dev	https://enjoy
5a104899-4919-4034-85da-9f6f043c937	bllhiz@me.com	2024-05-28 07:57:05.720599+00	NULL	NULL	NULL	NULL

Figure 4.1: Screenshot of Supabase Table View Interface

4.2 Supabase Architecture and Core Components

Supabase uses an SQL database [38] with PostgreSQL [27], a robust database system that allows complex queries to be made using SQL, which deals with rows and columns rather than documents, unlike a NoSQL database such as the one used by Firebase, and the functions provided by Supabase client library for interacting with the database reflect SQL terminology. In addition, Supabase provides an intuitive user interface for managing the database and generating REST [35] and GraphQL [41] APIs. On the other hand, the PostgreSQL database can also be managed through an easy-to-understand user interface that automatically generates REST and GraphQL APIs for use in our code.

Furthermore, Supabase uses only open source technologies and provides a user-friendly dashboard that can be accessed without the need to download any other applications or extensions. Through the user interface shown in Figure 4.1, tables can be easily viewed, as well as quickly sort data and preview records. Hence, allowing data resources to be easily created and scaled for applications.

Looking at Figure 4.2, which clearly shows the Supabase architecture [36], we can see that Supabase is built around the PostgreSQL core, enhanced with a number of additional tools to improve functionality and ease of use. The main components are as follows:

- **PostgreSQL [28]:** Manages the relational database system, providing robust and scalable data storage.
- **GoTrue [39]:** Handles secure user authentication system.

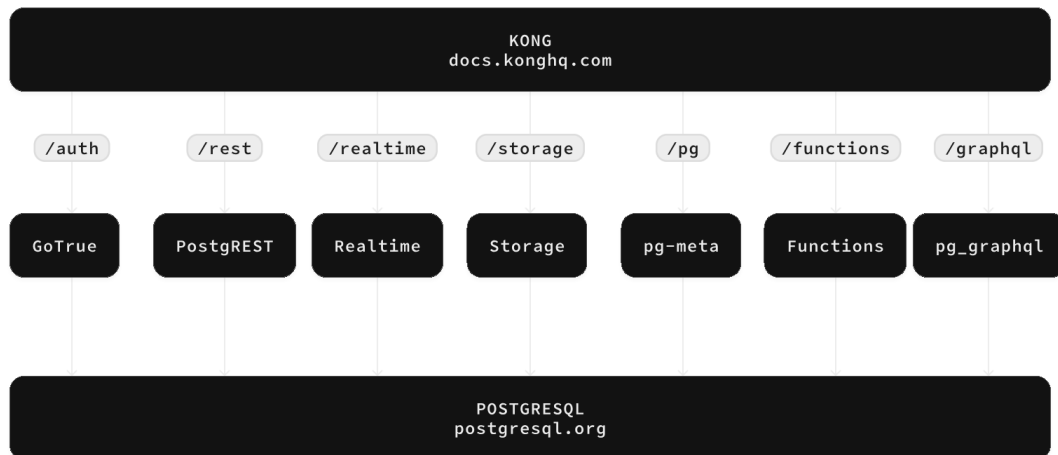


Figure 4.2: [Supabase Architecture](#) and Core Components

- **PostgREST [29]**: Creates a web API that allows to interact with a PostgreSQL database using standard HTTP requests.
- **Realtime [42]**: Enables real-time changes and receive updates instantly.
- **Storage API [43]**: Handle files, including managing file uploads and downloads.
- **Deno (Edge Functions) [9]** : Executes serverless functions closer to the user, helping to reduce latency and improve performance.
- **Supavisor [46]**: Efficiently manages database connections, ensuring optimal performance and resource utilisation.

This architecture provides a robust backend solution that is easy to use and built on open-source technologies.

4.3 Advanced Backend Configuration with Supabase

One of the most important aspects of our backend configuration is setting up the Supabase client in our application. This setup is important to enable secure interactions between the frontend of our application and the Supabase backend. As can be seen, the `supabase.ts` file in our project, shown in Listing 4.1, initialises the authentication mechanism and is also useful for how the application handles session data across different platforms. Additionally, we can observe that the environment variables for the Supabase URL and the anonymous key are used to protect the authentication details, while "AsyncStorage" is used to securely

manage the user session across devices and store data locally. In other words, this configuration is an important part of integrating the front-end and back-end capabilities of Supabase.

```
1 const supabaseUrl = process.env.  
  EXPO_PUBLIC_SUPABASE_URL as string;  
2 const supabaseAnonKey = process.env.  
  EXPO_PUBLIC_SUPABASE_ANON_KEY as string;  
3  
4 export const supabase = createClient(supabaseUrl,  
  supabaseAnonKey,  
5  {  
6    auth: {  
7      ...(Platform.OS !== 'web' ? { storage:  
        AsyncStorage } : { }),  
8      autoRefreshToken: true,  
9      persistSession: true,  
10     detectSessionInUrl: false,  
11   },  
12 }  
13 );
```

Listing 4.1: Initialising Supabase Client with Environment Variables and Session Management

In terms of the actual code, as evidenced by Listing 4.2, which is derived from our code, it is unnecessary to write raw SQL code for the database. Instead, we can utilise SDK queries tailored to our database and use dot notation to grab the data we require and then perform operations on it.

```
1 const { data: users, error } = await supabase  
2   .from("users")  
3   .select("displayName, photoURL, photo, id")  
4   .in("id", ids)  
5   .range(0, 5);
```

Listing 4.2: Supabase Query Example to Fetch User Data

Furthermore, as illustrated in Listing 4.3, it is possible to authenticate a user with just a few lines of code, while returning an error as an object without having to wrap it within a try-catch block in order to catch errors.

```
1 const { data, error } = await supabase.auth.  
  signInWithPassword({
```



```
2   email: 'example@email.com',  
3   password: 'example-password',  
4 })
```

Listing 4.3: Supabase Authentication example using Email and Password

In addition, a listener can be set up with Supabase to handle real-time authentication state changes, logging the event type and session information when a user logs in and logs out, as shown in Listing 4.4.

```
1 const { data } = supabase.auth.onAuthStateChange((  
2   event, session) => {  
3   console.log(event, session)  
4 })
```

Listing 4.4: Handling Authentication State Changes with Supabase

4.4 Evaluating Firebase and Supabase Before Migration process

In the context of application development, choosing the optimal backend can be challenging. Especially for applications that need to be extensible and scalable, a robust backend is essential to facilitate data storage, user management, and overall operational efficiency. In this regard, platforms such as Firebase [17] and Supabase offer a large number of tools that handle these processes. Therefore, the question arises as to which of these two options is more appropriate. In order to answer this question, it is necessary to conduct a detailed analysis of the two options presented, and take a closer look at these two backend platforms from several perspectives:

- **Authentication:** Both platforms are intuitive. Supabase's GoTrue-based authentication integrates with PostgreSQL's security features, while Firebase Authentication [18] is known for its ease of use with other Firebase services.
- **Database and Data Management:** Firebase's Firestore [19] is a NoSQL database, whereas Supabase uses PostgreSQL, a relational database with powerful SQL queries. Generally, relational databases offer more advantages than NoSQL databases.
- **Real-time Capabilities:** Each platform provides real-time features, enabling applications to instantly update and synchronize data across devices.

- **Cloud Functions:** Both Firebase and Supabase offer serverless functions, allowing custom code to run on the server without the need to manage the servers.

For a concise overview of the differences and similarities between Firebase and Supabase, Table 4.1 presents a comparison of their main features.

Features	Supabase	Firebase
Relational Database	Offers SQL, real-time updates	NoSQL, real-time updates
Deployment	Hosted on Amazon Web Services	Hosted on Google Cloud
Authentication	Built-in, supports social logins	Built-in, with social login integrations
Custom Functions	Full SQL support, more customisable	Limited to Firebase services (Cloud Functions)
Community	Growing rapidly, open-source community, but smaller compared to Firebase	Established platform with a larger and active community
Scalability	Scales with PostgreSQL, suitable for complex queries and large datasets	Scales with Firestore, optimised for real-time data and simple queries

Table 4.1: Comparison of Firebase and Supabase Features. Adapted from [Flatirons Development](#) and [White Lotus Corporation](#)

4.5 Detailed Steps in Database Migration from Firebase to Supabase

Database migration was part of the application development process. The company decided to move from Firebase to Supabase because Supabase is more flexible and open source, which fits with the company's long-term goals.

The migration process from Firestore to Supabase took place in two phases. The first was transferring authentication data, and the second was migrating the database content.

After following the steps outlined in the Supabase documentation for migrating

authentication data from Firebase [24], the process was successfully completed, and all authentication data was transferred to Supabase. This procedure is necessary to ensure that existing user accounts, including their authentication states, and user continuity are maintained.

In sum, the authentication migration involved several key actions:

- **Exporting Users from Firebase:** The following command was used to extract users from Firestore and export them to a JSON file, with the option of defining the output filename and the number of users per batch:

```
node firestoreusers2json.js [<filename.json>] [<batch_size>]
```

- **Formatting the data:** The command below formats exported data to match Supabase authentication requirements, importing from a JSON file into Supabase tables with an optional batch size for efficient processing:

```
node import_users.js <path_to_json_file> [<batch_size>]
```

As a consequence, the database content will be the main focus of the next phase. By following the instructions provided in the Supabase documentation on how to migrate the database from Firestore data [25], the Firebase NoSQL Firestore data was converted to a relational format that was compatible with PostgreSQL.

In other words, the objective was to convert Firestore collections to PostgreSQL tables and to ensure that the NoSQL documents were adapted to a relational schema suitable for the Supabase environment. This phase is important in order to ensure that the integrity and structure of the data is maintained.

The following steps detail the main procedures involved in the data migration process:

- **Extracting Firestore Data:** Data were extracted from Firestore collections and saved in JSON format for transition to a relational database structure. The following command was used, with optional parameters for batch size and document limit:

```
node firestore2json.js <collectionName> [<batchSize>] [<limit>]
```

- **Importing Data into Supabase:** The JSON data was imported into a Supabase database using the command below with optional parameters for the primary key strategy:

```
node json2supabase.js <path_to_json_file> [<primary_key_strategy>]
```

5 | Database Architecture and Schema

5.1 Overview of Database Architecture

This thesis initially considers the existing database architecture created by the company. The original database, part of the back-end system, was designed to support the operational needs of the WeShre platform.

While developing the backend for this project, it was necessary to ensure seamless integration with the existing database architecture and to adapt the system in order to support all functionalities.

Despite these efforts, the migration from Firebase to Supabase presented significant challenges, which will be summarised in the limitations section. These challenges included the complexity of redefining unique identifiers and restructuring relationships between data tables, which ultimately led to the company's decision to create a new database directly through Supabase. This new database was designed to be fully compatible with the front-end components and to ensure proper linkage and functionality.

In the next sections, we will explain the existing structures that were initially used and review the final design of the database after the migration.

5.2 Detailed Database Schema

5.2.1 Table Descriptions

In this subsection, we will provide a detailed overview of all the tables in the WeShre platform database that were used and that met our needs during the development process. Due to the challenges encountered during the migration from Firebase to Supabase, as previously mentioned, the current schema reflects the new database structure designed for optimal compatibility.

To clearly present the information about these tables, we use a tabular format. Table 5.1 lists all the names of these tables, along with their purpose, main functions, and key fields associated with each table. This format provides a clear view of the database structure and its role in supporting the functionality of the platform.

Table Name	Description
users	Contains basic information about each user registered on the WeShre platform. It contains comprehensive profile information, including fields such as displayName, photoURL, photo, id, display_name, first_name, last_name, profile_image, gender, about, ratings, rate, birthdate, email, school, languages, and job . The table supports many functions, such as retrieving user details for profile views, managing user connections by fetching friends' data, and supporting event-related features by providing user details when creating or participating in events.
events	Used to manage event data. It stores details about each event, including fields such as id, creator, title, description, start_date, image, end_date, type, languages, location, map, status . It also includes relationships to other tables to provide more detail about events. This table is used to retrieve full event information for individual event pages, to facilitate event discovery, and to provide event summaries for user profiles.
groups	Used to manage group data, including creator, name, description, photo, and uid fields. It also includes relationships to other tables to manage memberships and associated events. It supports many features, such as retrieving group details for display on group pages, linking groups to their creators, and providing summaries of group activity for user profiles.
news	Manages news blog posts on the platform. It stores information about each news item, including fields such as id, image, and title . These attributes are used to display news summaries in small blog cards on the homepage.
sport_profiles	Manages users' sports-related information. It includes fields like sport_id and rank , and references the <code>sports</code> table to display users' sports profiles, showing their activities and ranking.
sports	Contains information about different sports, including fields such as id, name, and icon . The table provides sport details to the <code>sport_profiles</code> table, allowing sport names and icons to be displayed in user profiles.
event_ticket	Manages ticket information for events, including fields such as id, event, price, and currency . It references the <code>currencies</code> table for currency details. This table helps display ticket prices and currency information on event pages.

currencies	Stores currencies information used within the platform, including id and symbol . It provides currency details to <code>event_ticket</code> table, allowing ticket prices to be displayed accurately with the appropriate currency symbols.
event_includes	Manages details about what is included in events. Fields in this table are event , id , type , and title . The table helps to display additional information about each event's offerings.
user_tickets	Records the tickets that users buy for events. Key fields include id , event . This table references <code>users</code> table for details like display_name and profile_image .
reviews	Holdes reviews for events. Important fields are id , event , comment , and rating . It links to the <code>users</code> table to include reviewer details such as display_name , profile_image , ratings , and rate .
group_members	Tracks group members. It has group and user fields, and references the <code>users</code> table to get user information such as uid , display_name , and profile_image .
event_groups	Links events to groups. It contains event and group fields, and references the <code>events</code> table to provide event details such as id , title , description , start_date , end_date , image , and type .
follows	Manages user follow relationships using follower and followed fields. This table tracks which users follow others in order to display user connections on the platform.

Table 5.1: Detailed Summary of Database Tables and Key Fields in the WeShre Platform

5.2.2 Entity-Relationship Diagram (ERD)

Having described each of the tables used in the WeShre database in the previous section, it is important to comprehend the interrelationships between these tables and how they interact within the overall database architecture. To illustrate this, we will present an Entity Relationship Diagram (ERD) that shows the relationships between these tables. This diagram clarifies the dependencies and connections between the tables, and helps to understand the WeShre database design. The ERD in Figure 5.1 clearly shows the database architecture and highlights key entities.

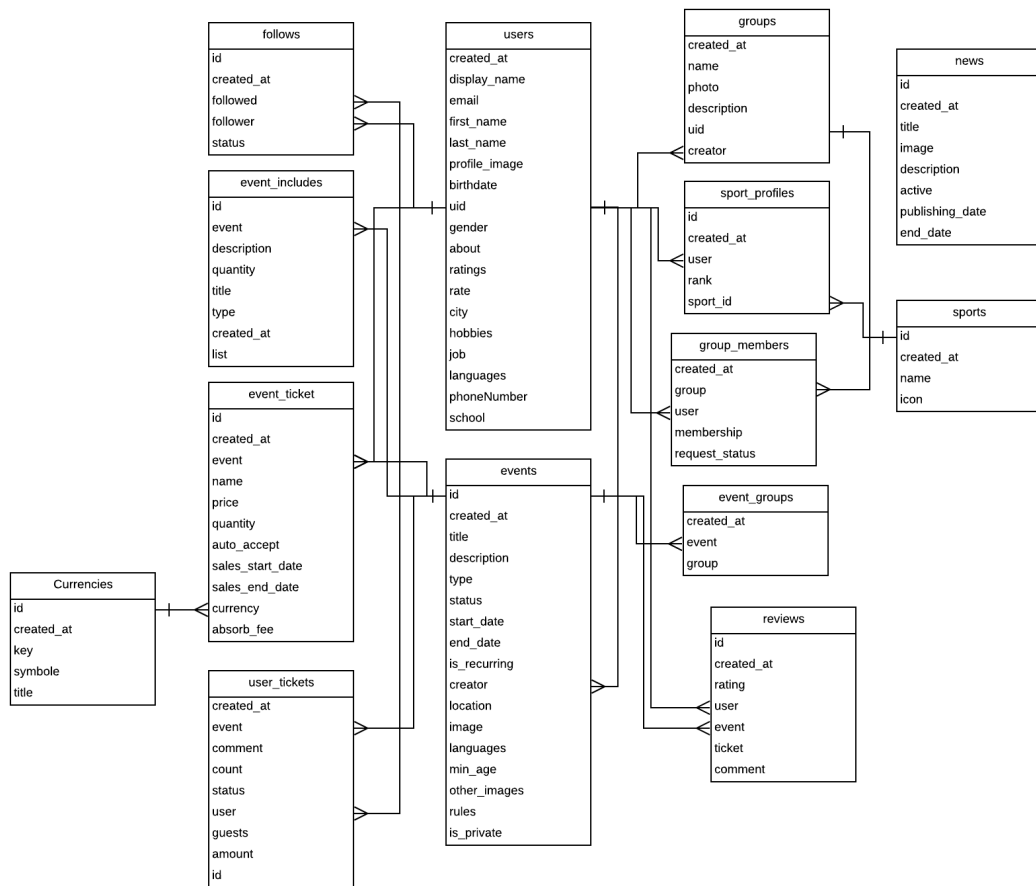


Figure 5.1: Entity-Relationship Diagram (ERD) for the WeShre database

6 | Design and Implementation of User Interfaces in WeShre

6.1 System Overview with High-Level Architecture

Before delving into the explanation of our application interfaces in the following sections, it is best to provide a general and clear overview of the main features offered by the application built during this thesis. In this section, we present a high-level architecture diagram of the WeShre platform, through which we can illustrate the interconnectivity and flows between different components and screens of our system, facilitating a comprehensive understanding of the platform's structure. Figure 6.1 provides a high-level overview of the core functionality implemented throughout this thesis.

6.2 Responsiveness and Platform Adaptability

First of all, before explaining the interfaces that were designed and developed during this project in the following sections, it is important to clarify that each interface was carefully designed to ensure optimal performance and to meet all user needs. In particular, each interface was crafted to maintain consistency across both web and mobile platforms and across different screen sizes on various devices.

The web versions were designed to efficiently use the space available on large screens, while maintaining a clean and organised appearance. Conversely, the mobile versions were designed to be compact, easy to navigate, and suitable for mobile device screens, ensuring that all elements are touch-friendly.

Furthermore, in order to demonstrate the versatility of the interfaces across different platforms, it is worth noting that the web platform interfaces are also responsive, ensuring full functionality when accessed from smaller screens, such as smartphone browsers. Screenshots of the web versions of each interface adapted for smaller screens are provided in the appendix of this thesis. This is intended to ensure the effective user experience offered by the WeShre platform, regardless of the device type.

Specific adaptations and design considerations for each page will be detailed in the sections dedicated to each.

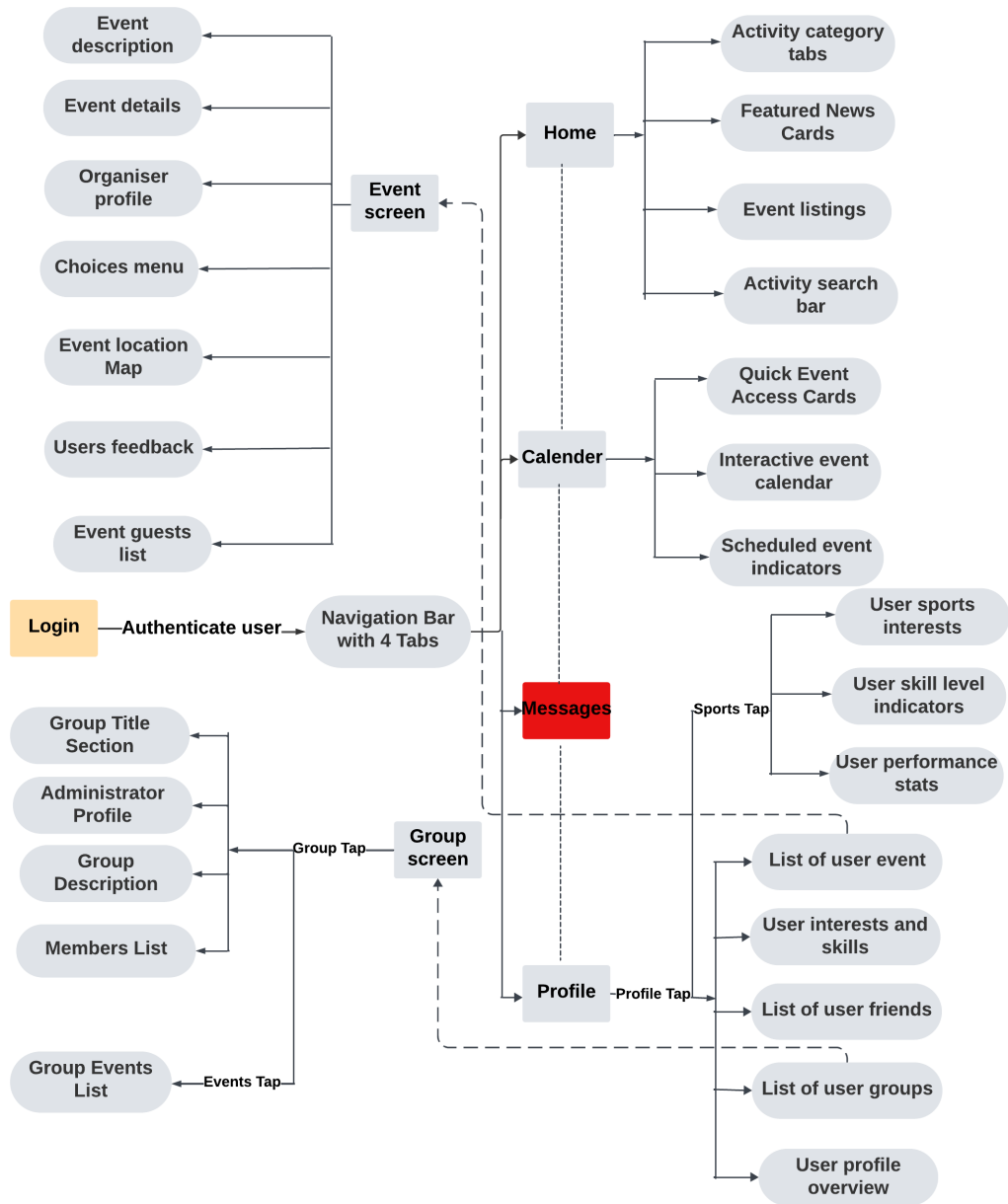


Figure 6.1: High-Level Architecture Diagram of the WeShre Platform

6.3 Login Page Interface

6.3.1 Page Overview

The Login page is the first page designed in this project, through which participants can log in to the WeShre platform. This page was designed to be a welcoming and straightforward page to ensure that users can easily access the platform. Users can log in using their email and password. It should be noted that the Google, Apple ID, and phone number login options are outlined but not yet operational. These are planned additions for future enhancements to improve accessibility.

6.3.2 Web and Mobile Implementation

The implementation of the login interface ensures consistency across web and mobile platforms as follows:

- **Mobile Design:** The mobile version of the login page, as shown in Figure 6.2, has a user-friendly design optimised for smaller screen sizes.
- **Web Design:** The web version, shown in Figure 6.3, effectively utilises the space available for larger screens while maintaining an attractive design.

When the login button is clicked, the system checks the associated Supabase database to verify if the entered email is registered in the authentication data, ensuring secure access to the platform.

Screenshots showing how the web interface adapts to smaller screens are provided in the appendix (Figure A.1).

6.4 Home Page Interface

6.4.1 Page Overview

Following a successful authentication process on the login page, users are directed to the home page. The homepage represents the main interface of the WeShre platform. Through this interface, users can perform the following:

- Discover all available events and activities offered by the platform.
- Filter activities based on categories such as "**Party**," "**Sports**," or "**Experience**."
- Use the search capability to easily find specific events or activities.

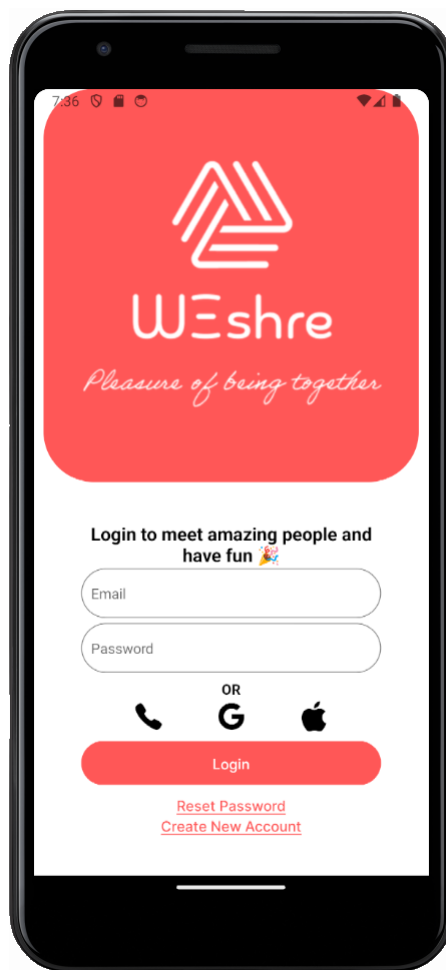


Figure 6.2: Mobile version of the login page.

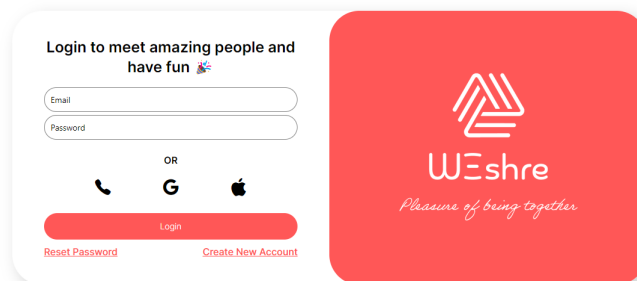


Figure 6.3: Web version of the login page.

- View attractive display of multiple event cards with scrolling and navigation to explore all activities on the platform. Users can click on a card to access event pages. Each card displays a summary of the event, including date, time, and user ratings.
- View blog cards that summarise the latest news and updates on the platform. Each card includes an image and title, allowing users to quickly show the latest news.

6.4.2 Web and Mobile Implementation

In the **web** version, the home page uses a wide layout with large clickable event cards, as shown in Figure 6.4. The design focuses on prominently displaying the event cards, blog cards, and key features such as search bars, in order to ensure they are clearly visible on large screens.

The **mobile** version of the homepage, as shown in Figure 6.5, is designed to adapt to smaller screens while maintaining the functionality and aesthetics of the web version. The mobile design shows a single blog or event card view within the page, with the ability to easily scroll horizontally to get new cards. In addition, the search bar is designed to be touch-friendly and adaptable. The layout supports vertical scrolling, thus making it easy for users to explore content with simple swipes and taps.

The appendix to the thesis includes screenshots of the web interface adapted for smaller screens when accessed from mobile browsers are included (Figure A.2), demonstrating the responsive capabilities of the platform.

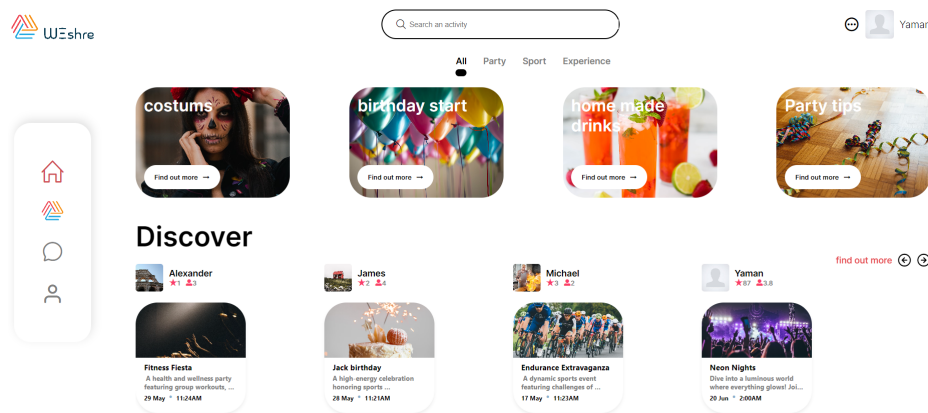


Figure 6.4: Web version of the home page with large event and blog cards.

6.5 Calendar Page Interface

6.5.1 Page Overview

The purpose of the Calendar page is to show users upcoming events, both events they are interested in and events they have committed to attend. It displays events in a monthly calendar format, making it simple for users to keep track of upcoming events for each month. Each event is marked with its own date, making it easy for users to determine when an event is taking place. In addition, by clicking on a date within the calendar, users can view a brief summary of each event scheduled for that day.

6.5.2 Web and Mobile Implementation

- **Mobile Design:** The mobile version of the calendar page, shown in Figure 6.6, where we can see that the event cards are displayed as individual items that users can scroll through horizontally by swiping left or right, or by clicking on the circular arrow buttons. In addition to displaying the calendar component in a compact and ideal way, which is the main goal of this page.
- **Web Design:** The web version, shown in Figure 6.7, offers a wider view of the calendar component and displays more event cards at the same time, taking advantage of the larger screen space.

Each event is indicated by a red dot within the calendar, indicating that these days contain events. Clicking on these dates brings up a brief summary of the event, providing the user with a quick overview of the activities on those days.

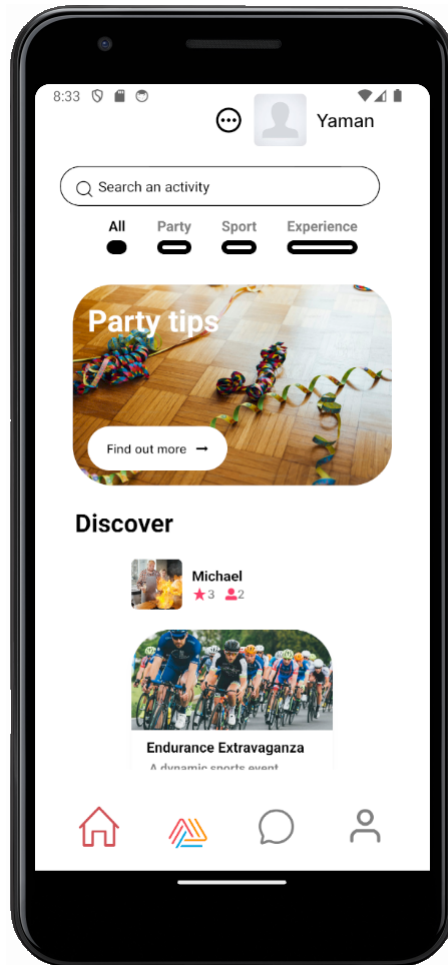


Figure 6.5: Mobile version of the home page with a single card view and horizontal scrolling.

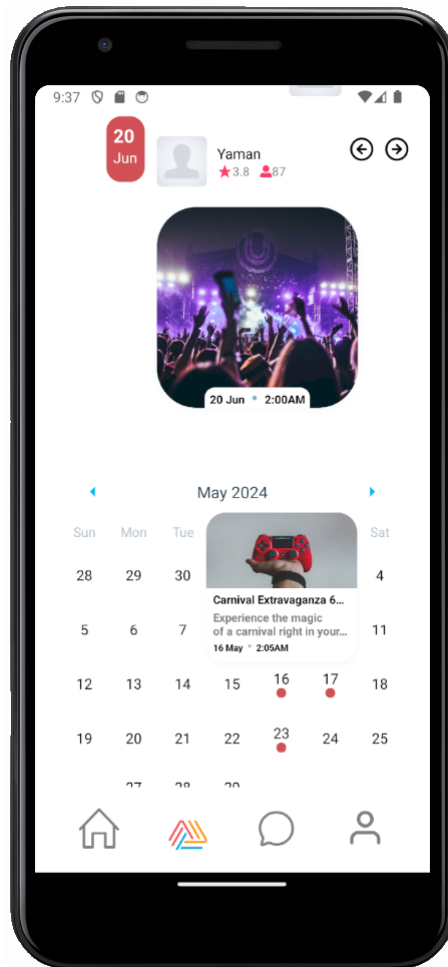


Figure 6.6: Mobile calendar page with individual event cards and horizontal scrolling.

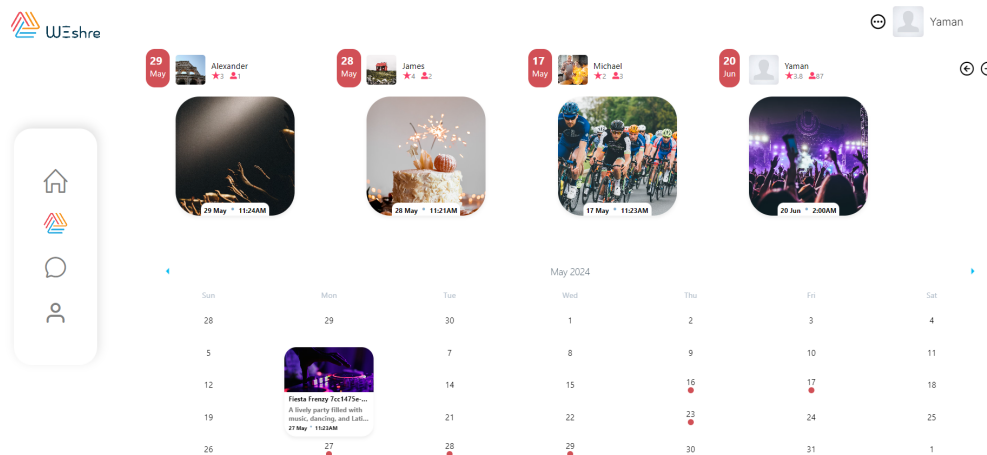


Figure 6.7: Web calendar page with a wide view and multiple event cards.

Furthermore, care has also been taken to ensure that the design of the calendar web interface is optimised for smaller screen sizes when accessed from mobile browsers. Screenshots are included in the appendix (Figure A.3), to highlight the responsive design of the platform.

6.6 Profile Page Interface

6.6.1 Page Overview

The profile page is the main interface through which users can display and interact with their personal details, activities, and groups on the WeShre platform. The page consists of two main tabs: the `Profile` tab and the `Sport` tab.

The `Profile` tab displays the user's personal information and activities, while the `Sport` tab displays the user's sporting interests and achievements.

In summary, the profile page has the following key elements:

- **User Information:** Displays the user's name, profile picture, ratings, and friends list.
- **Activities and Groups Section:** Provides a list of the user's recent activities and groups, allowing quick access to the user's event and group pages.
- **Sports Skills and Details:** Displays icons for each sport the user participates in, their achievements, along with their respective skill indicators for each sport (such as `Beginner`, `Expert`, **etc.**).

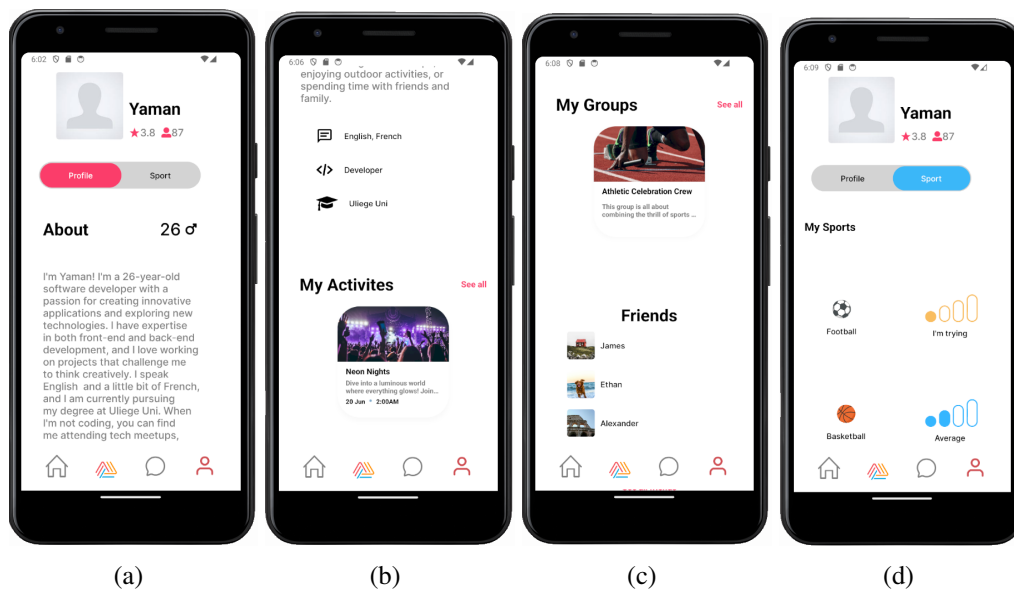


Figure 6.8: Mobile profile page showing user info, activities, sports skills, and more details.

6.6.2 Web and Mobile Implementation

- **Mobile Interface:** The mobile design of the profile page, shown in Figure 6.8, ensures that the page elements are properly sized and arranged for ease of use on smaller screens. The interface allows users to scroll through their information and access different functions easily.
- **Web Interface:** The web version of the profile page, shown in Figure 6.9, utilises the available space to display information in a more readable format.

The web version of the profile page is also responsive. Screenshots showing how the web interface adapts to smaller screens are included in the appendix (Figure A.4).

6.7 Event Page Interface

6.7.1 Page Overview

The Event page shows details of specific events on the WeShre platform and can be accessed by clicking on their respective event card. The page provides an overview of the event, including the date, time, and price. It also includes an 'About' section that provides a brief summary of the event and a section that

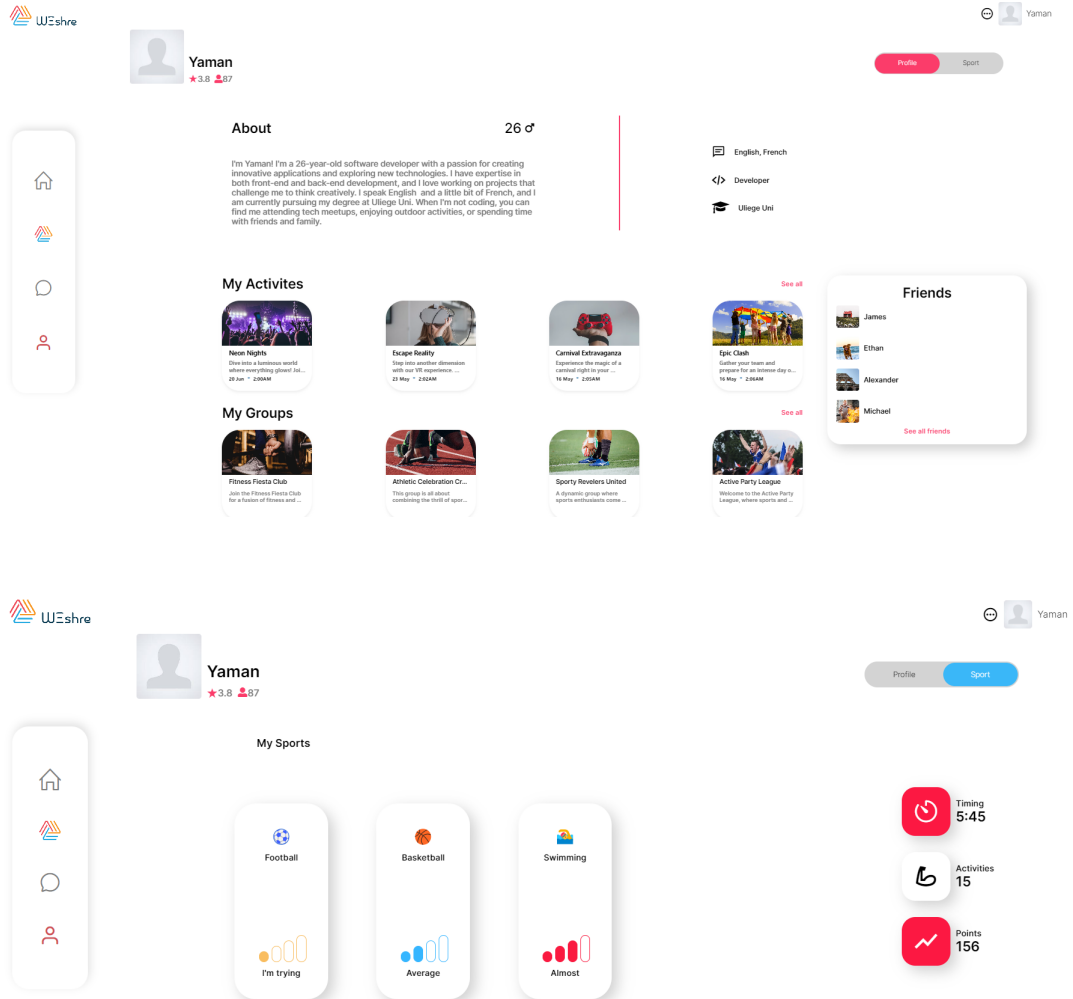


Figure 6.9: Web profile page displaying user info, activities, sports skills, and more details.

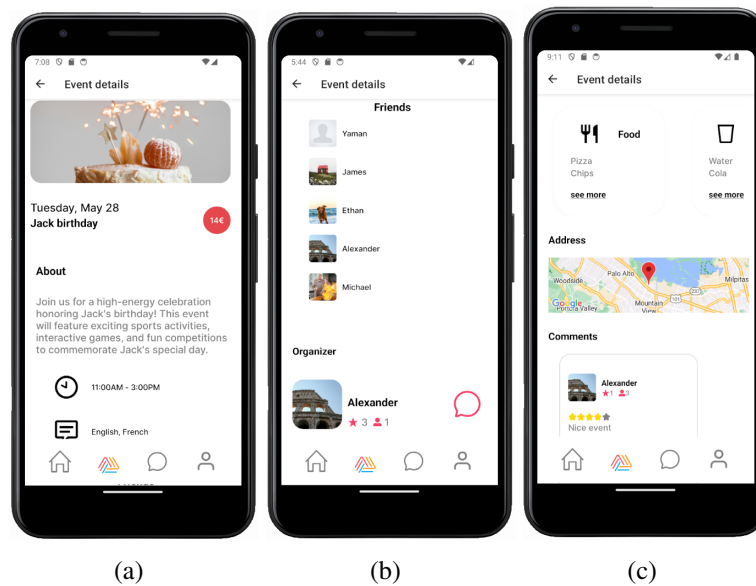


Figure 6.10: Mobile version of the event page displaying event details.

identifies the event organiser. There is also a list of what the event includes (such as food and drinks), an interactive map showing the location, and a section for user comments and ratings.

6.7.2 Web and Mobile Implementation

We have ensured visual and functional consistency across both mobile and web platforms by carefully adapting each component of this page as follows:

- **Mobile Implementation:** Figure 6.10 shows the layout of the event page on a mobile device. The design uses vertical scrolling to make the best use of the small space and to facilitate navigation, allowing users to easily explore event details.
- **Web Implementation:** Figure 6.11 shows the event page layout on the web. Here, we can clearly see that the event details are more widely distributed, taking advantage of the additional screen space for a more detailed display.

Screenshots in the appendix (Figure A.5) are provided to demonstrate the responsiveness of the web interface, showing how the event page adapts to different screen sizes when accessed via the web.

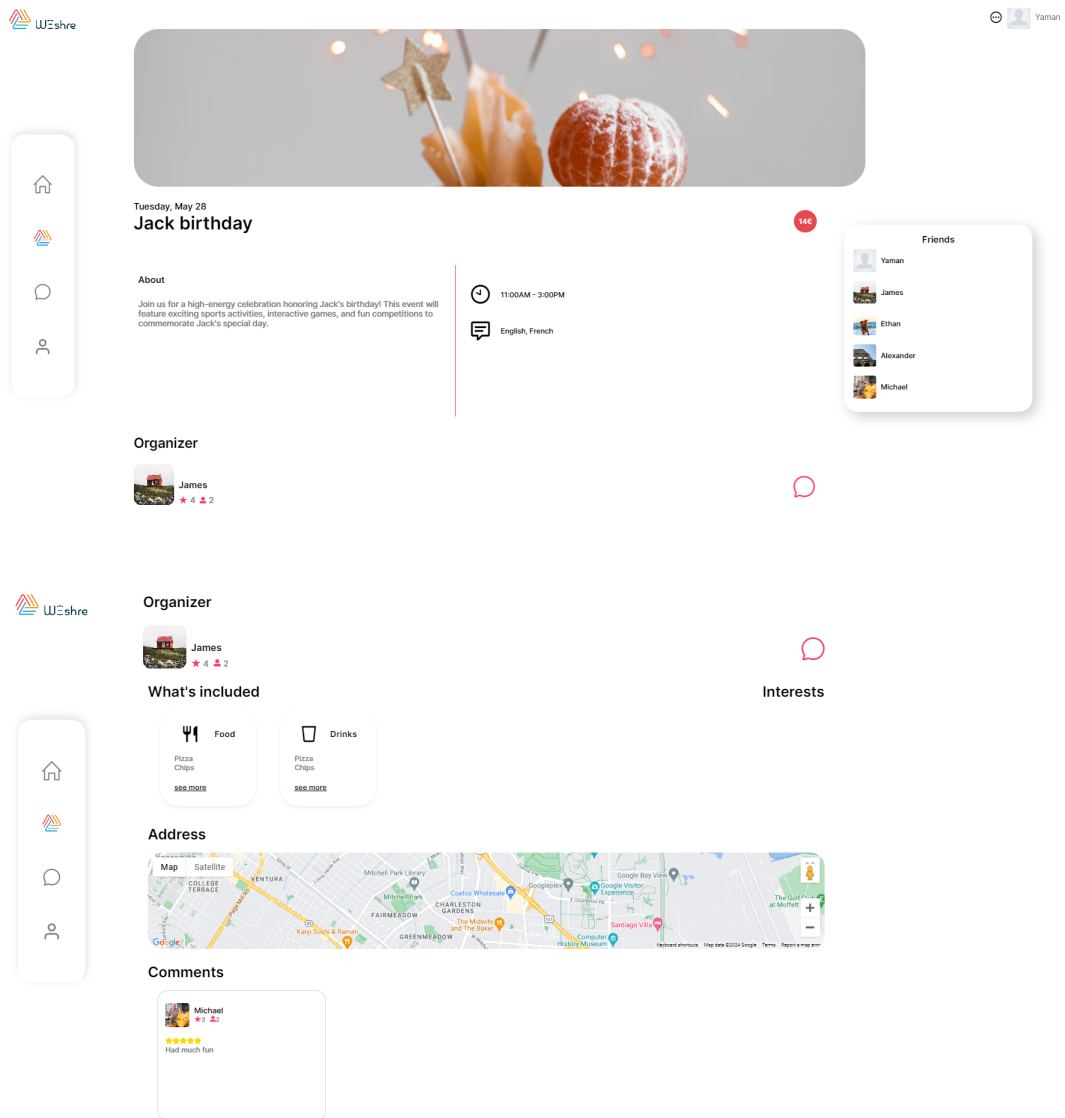


Figure 6.11: Web view of the event page displaying event details.

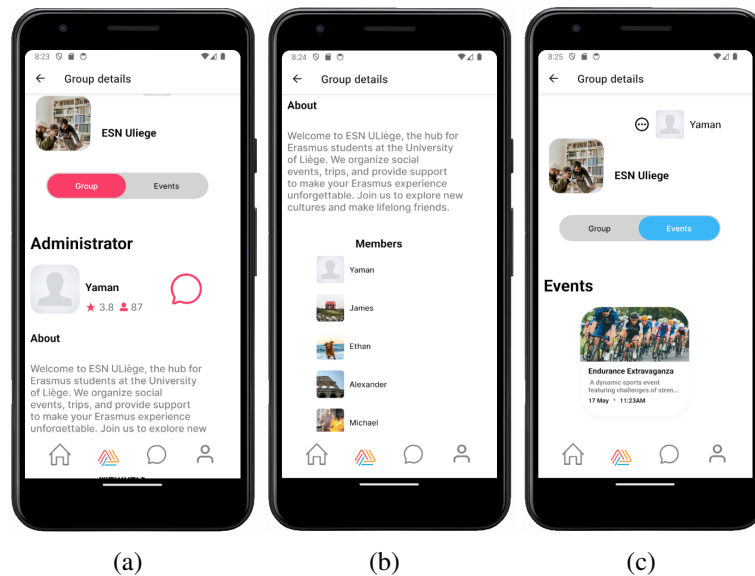


Figure 6.12: Mobile version of the group page displaying group information, events, and members list.

6.8 Group Page Interface

6.8.1 Page Overview

The group page provides users with information about the group and its members, as well as displaying group-related events. It has two main tabs: `Group` and `Events`.

Key elements include:

- **Group and Admin Information:** Displays the group name, cover image, description, and the group administrator.
- **Events section:** Lists upcoming events related to the group with names, dates, and times.
- **Members List:** Displays the current members of the group.

6.8.2 Web and Mobile Implementation

- **Mobile Design:** The mobile version of the group page, shown in Figure 6.12, provides a compact and easy-to-touch interface, where users can easily scroll through the information.

- **Web Design:** The web version of the group page, shown in Figure 6.13, shows that the design makes effective use of the available screen space.

Screenshots in the appendix (Figure A.6) demonstrate how the web interface adapts to accommodate the narrower screen width, showcasing the responsive design of the platform.

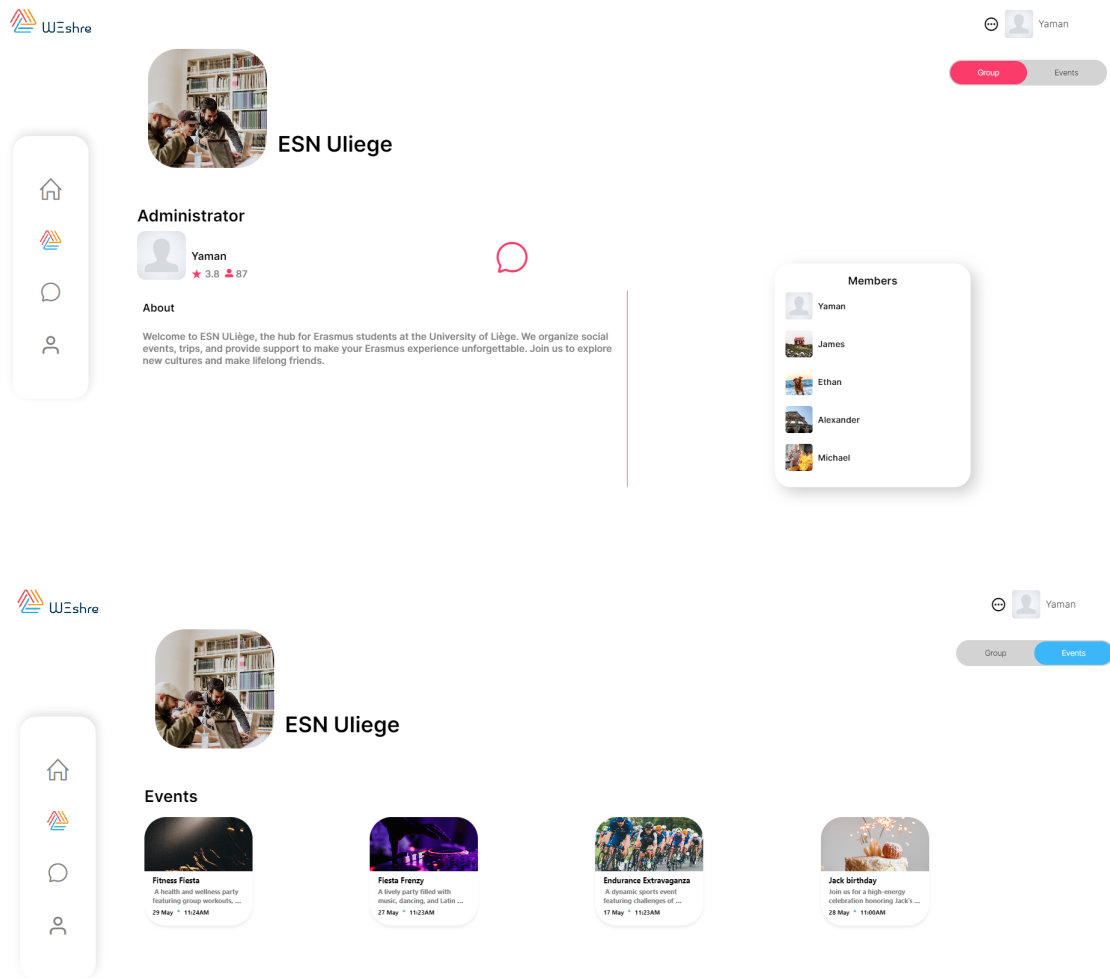


Figure 6.13: Web group page displaying group information, events, and members list.

7 | Testing

7.1 Testing with Cypress

The testing process is one of the most important processes in ensuring the success and integrity of the project and verifying the accuracy of the functions implemented during development.

After reviewing several testing methods and trying to find a suitable test for this thesis project, Cypress [8] was selected as the primary tool for end-to-end testing. Cypress is a JavaScript-based testing framework that allows testers to validate web applications. It was chosen due to its ease of use and powerful features that align with and simulate real user interactions in our project.

Cypress provides an open-source browser-based test runner that can simulate the experience of an end-user browsing the website. This is achieved by simulating the entire project execution, manually navigating the project from the login form to the user's home page and so on until all components of the project have been navigated and tested. Each action performed during the simulation is automatically recorded, and a snapshot is saved at each step, as we will see later when it is applied to our project. Ultimately, this testing methodology allows us to examine the end-user experience in detail and identify any areas where the code may be failing.

The Cypress application [21] is available for use with the Cypress software, which assists in the creation of tests. The application can be accessed by entering the following command:

```
npx cypress open
```

There are two main testing options available: end-to-end testing [10] and component testing [26] as shown in Figure 7.1. Cypress enables the implementation of two different types. End-to-end testing is the process of loading the web application in the browser and involves the simulation of an entire browser environment using automated tools to ensure that the inputs, button presses, form filling, and actions work correctly. This allows for an overall assessment of the application's functionality and performance. The testing procedures performed in this thesis focused on end-to-end testing, as it provides the most comprehensive solution offered by Cypress.

Cypress automatically generates a folder in the root of the project named "cypress" which contains all of the testing code, as shown in Figure 7.2. The "fixtures" directory is where the mock data is stored, while the "support" directory is for the

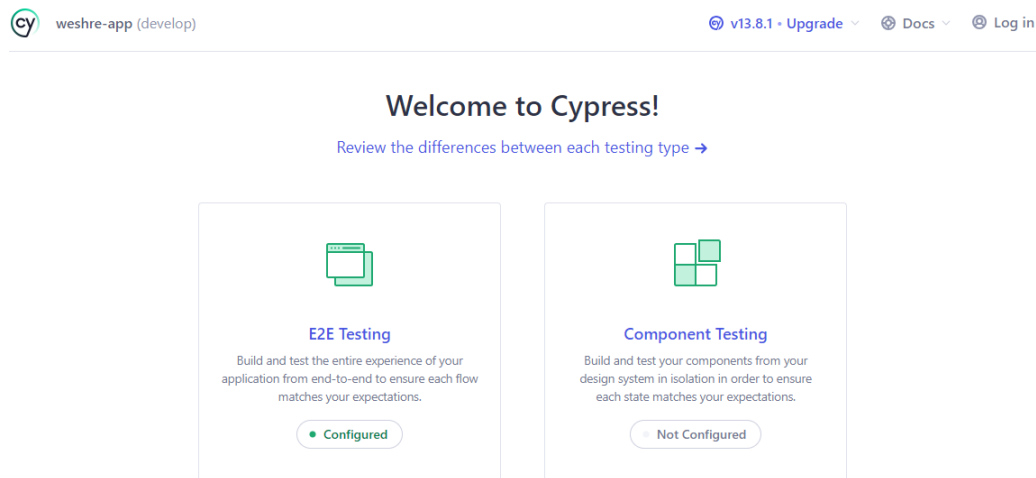


Figure 7.1: Cypress Application

global configuration.

The main functionality of Cypress is its ability to perform end-to-end testing using an easy-to-understand syntax. A sample test file, such as `spec.cy.ts`, can be found in the cypress integration directory. Each test suite starts with a `'describe'` block that defines the purpose of the suite. Listing 7.1 shows an example of the main steps of the test case.

```
1 describe('template spec', () => {
2   it('passes', () => {
3     cy.visit('http://localhost:8081')
4   })
5 })
```

Listing 7.1: Cypress test script for checking the functionality of a local server.

7.2 End-to-End Test Execution and Results

The Cypress test was executed using the Cypress Test Runner. The test runner was opened using the same command that is used to access the application. Next, the test file `spec.cy.ts`, which was previously described, was selected and the test was executed. The test navigated to `http://localhost:8081` and verified that the application loaded correctly. It also involved navigating through all the

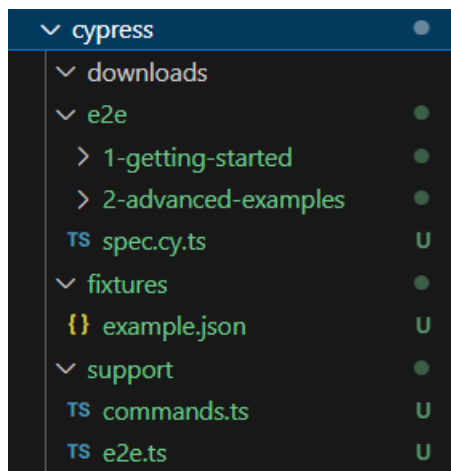


Figure 7.2: Cypress Folder Structure in Project Directory

different sections of the application, including the profile page, calendar, home page, individual event pages, and group page.

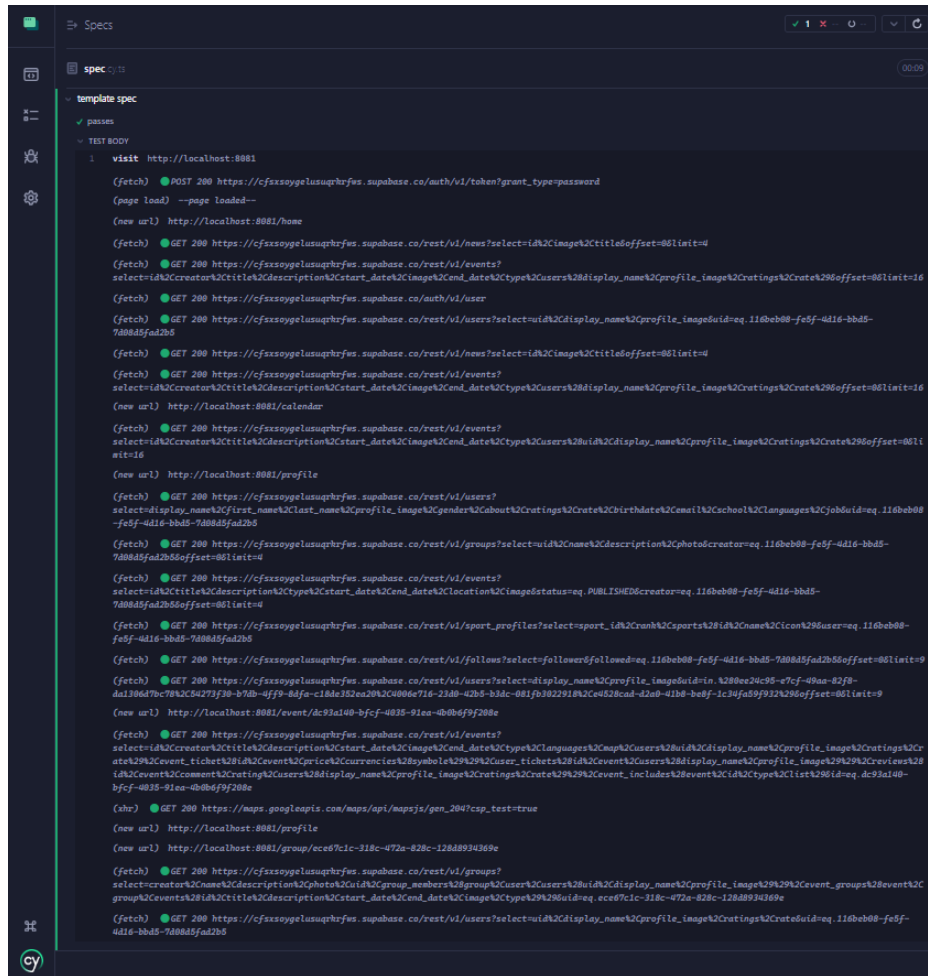
During the test, several network requests were made to fetch key data such as user profiles, groups, events, and calendar entries. Each of these requests returned a status of 200, indicating successful data retrieval. The details of these requests, taken from our thesis project, are shown in the test runner output in Figure 7.3.

7.3 Interpreting Test Outcomes

After successfully implementing this test, Cypress indicated that the web application was functioning correctly. In particular, the application was able to:

- Authenticate the user.
- Display home page information.
- Retrieve and display user profile information.
- Load group and event data.
- Navigate between different sections such as profile, calendar, home, individual event pages, and group pages.

In summary, the results of this test confirm that the key functions of the web application are working properly. Using Cypress proved to be an effective tool for end-to-end testing and our test results proved that the core functionality of the application works as expected.



```
Specs
spec
template spec
  passes
  TEST BODY
    visit http://localhost:8081
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/auth/v1/token?grant_type=password
    (page load) --page loaded--
    (new url) http://localhost:8081/home
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/rest/v1/news?select=id%2Cimage%2Ctitle%2Coffset%0%limit%4
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/rest/v1/events?select=id%2Ccreator%2Ctitle%2Cdescription%2Cstart_date%2Cimage%2Cend_date%2Ctype%2Cusers%28display_name%2Cprofile_image%2Cratings%2Crate%29%offset%0%limit%16
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/auth/v1/user
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/rest/v1/users?select=id%2Cdisplay_name%2Cprofile_image%2Cuid%eq.116beb00-fe5f-4d16-bbd5-7888d5fad2b5
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/rest/v1/news?select=id%2Cimage%2Ctitle%2Coffset%0%limit%4
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/rest/v1/events?select=id%2Ccreator%2Ctitle%2Cdescription%2Cstart_date%2Cimage%2Cend_date%2Ctype%2Cusers%28display_name%2Cprofile_image%2Cratings%2Crate%29%offset%0%limit%16
    (new url) http://localhost:8081/calendar
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/rest/v1/events?select=id%2Ccreator%2Ctitle%2Cdescription%2Cstart_date%2Cimage%2Cend_date%2Ctype%2Cusers%28display_name%2Cprofile_image%2Cratings%2Crate%29%offset%0%limit%16
    (new url) http://localhost:8081/profile
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/rest/v1/users?select=display_name%2Cfirst_name%2Clast_name%2Cprofile_image%2Cgender%2Cabout%2Cratings%2Crate%2Cbirthdate%2Cemail%2Cschool%2Clanguages%2Cjob%2Cuid%eq.116beb00-fe5f-4d16-bbd5-7888d5fad2b5
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/rest/v1/groups?select=id%2Cname%2Cdescription%2Cphoto%2Ccreator%eq.116beb00-fe5f-4d16-bbd5-7888d5fad2b5%offset%0%limit%4
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/rest/v1/events?select=id%2Ctitle%2Cdescription%2Ctype%2Cstart_date%2Cend_date%2Clocation%2Cimage%2Cstatus%eq.PUBLIC%2Ccreator%eq.116beb00-fe5f-4d16-bbd5-7888d5fad2b5%offset%0%limit%4
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/rest/v1/sport_profiles?select=sport_id%2Crank%2Csports%28id%2Cname%2Cicon%29%user%eq.116beb00-fe5f-4d16-bbd5-7888d5fad2b5
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/rest/v1/follows?select=follower%2Cfollowed%eq.116beb00-fe5f-4d16-bbd5-7888d5fad2b5%offset%0%limit%9
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/rest/v1/users?select=id%2Cprofile_image%2Cuid%in.%280ea26c95-e7cf-49aa-82f8-dal106d7bc78%2C5a273f30-b7db-4ff9-84fa-c18a352ea20%2C0086e716-234b-42b5-b3dc-081fb3022918%2Ce528cad-42a0-41b8-ba8f-1c3fa5f932%29%offset%0%limit%9
    (new url) http://localhost:8081/event/dc9a1a0-bfcf-4035-91ea-4b0b6f9f208e
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/rest/v1/events?select=id%2Ccreator%2Ctitle%2Cdescription%2Cstart_date%2Cimage%2Cend_date%2Ctype%2Clanguages%2Cmap%2Cusers%28uid%2Cdisplay_name%2Cprofile_image%2Cratings%2Crate%29%2Cevent_ticket%28id%2Cevent%2Cprice%2Ccurrency%2Csymbol%29%29%2Cuser_tickets%28id%2Cevent%2Cusers%28display_name%2Cprofile_image%2Cratings%2Crate%29%29%2Creviews%28id%2Cevent%2Ccomment%2Crating%2Cusers%28display_name%2Cprofile_image%2Cratings%2Crate%29%29%2Cevent_includes%28event%2Cid%2Ctype%2Clist%29%id%eq.dc9a1a0-bfcf-4035-91ea-4b0b6f9f208e
    (xhr) GET 200 https://maps.googleapis.com/maps/api/mapsjs/gen_204?csp_test=true
    (new url) http://localhost:8081/profile
    (new url) http://localhost:8081/group/ee67c1c-318c-472a-826c-128d8934369e
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/rest/v1/groups?select=creator%2Cname%2Cdescription%2Cphoto%2Cuid%2Cgroup_members%28group%2Cuser%2Cusers%28uid%2Cdisplay_name%2Cprofile_image%29%29%2Cevent%2Cgroup%28events%28id%2Ctitle%2Cdescription%2Cstart_date%2Cend_date%2Cimage%2Ctype%29%29%2Cuid%eq.ee67c1c-318c-472a-826c-128d8934369e
    (fetch) GET 200 https://cfxxoygelusuphrfes.supabase.co/rest/v1/users?select=id%2Cdisplay_name%2Cprofile_image%2Cratings%2Crate%2Cuid%eq.116beb00-fe5f-4d16-bbd5-7888d5fad2b5
```

Figure 7.3: Cypress Test Runner Output showing successful end-to-end test execution from the testing project.

8 | Deployment

8.1 Webmin Configuration and Virtualmin Setup

To manage the deployment of our website, we used Webmin [56], a web-based interface for system administration. Webmin facilitates the management of various system services such as web servers, databases, DNS, and user accounts. It offers a user-friendly interface accessible from any modern web browser and supports SSL encryption for secure connections, which ensures secure communications between the client and the server. Additionally, Webmin provides a file management feature that was particularly beneficial for uploading our website.

After setting up Webmin and logging in, we used the Virtualmin [57] module, which is an extension of Webmin, to create and manage virtual servers. Virtualmin facilitates the process of setting up domains and managing web server configurations by allowing the direct creation of virtual hosts. Webmin's modular design allowed us to customise our server environment efficiently to meet our specific needs through the use of Virtualmin.

We created a virtual server with the domain name `rewarding.today`. This involved configuring the necessary settings to host our website, such as setting the document root to the directory where the website files would be stored and enabling SSL for secure communications.

8.2 Project Deployment Process

To deploy our web project, we first exported the project files using the following command:

```
npx expo export --platform web
```

This command generated a `dist` folder containing the static files necessary for our web project.

We then used the File Manager [58] module in Webmin to upload the contents of the `dist` folder to the server, as shown in Figure 8.1. The File Manager in Webmin allowed us to easily transfer the project files to the appropriate directory on the server.

After uploading the project files, we properly configured the web server to serve the website from the uploaded files. This configuration ensures that when `rewarding.today` is accessed, the web server correctly serves the project files from the specified document root.

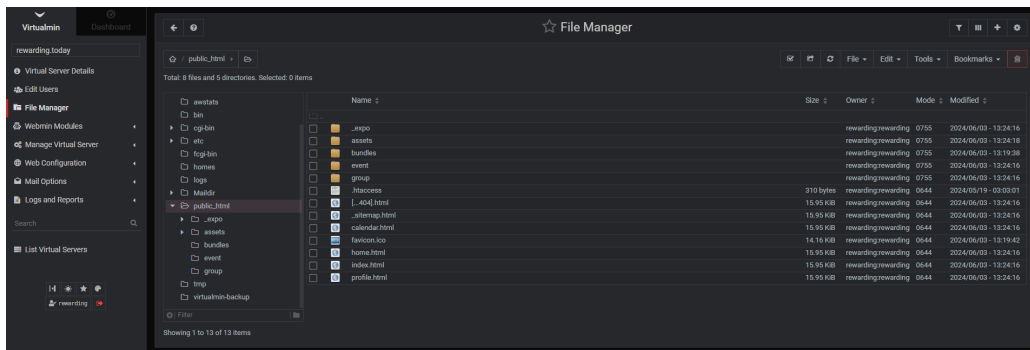


Figure 8.1: Webmin File Manager interface after uploading the project files to the server.

8.3 Verifying Deployment Success

The deployed project can be tested by navigating to the following link: <https://rewarding.today> To log in and test the functionality, the following credentials can be used:

- Email: `yaman.th701@gmail.com`
- Password: `123456`

These credentials can be used to verify that the deployment is successful and that the website is functioning as expected.

Deploying our project with Webmin was straightforward. Webmin simplifies server management, while Virtualmin handles virtual hosts efficiently. By exporting the project files, uploading them via the File Manager, and configuring the web server, we successfully deployed our web project.

9 | Conclusion

9.1 Project Conclusion

In conclusion, during this thesis, we developed and extended the WeShre platform to better meet the needs of its users. The overarching goal of this work was to enhance the reach and utility of the platform, with a particular focus on ensuring its unified and accessible operation across different environments.

We addressed several development tasks by leveraging advanced technologies. The Expo framework was chosen for the front-end due to its simplicity of deployment across iOS, Android, and web platforms. Additionally, React and React Native were key components of the front-end. React was used for building web interfaces, while React Native was used for mobile interfaces and their integration. Finally, the Tamagui library was employed to implement a consistent and responsive cross-platform design system, simplifying the management of styles and themes.

Regarding the backend, the transition from Firebase to Supabase was of paramount importance. Supabase, an open-source alternative to Firebase, efficiently provides all necessary backend services, including authentication, real-time databases, storage, and other notable features.

This project successfully fulfilled the company's objectives and addressed the limitations of the previously used technologies, which prevented the existing application from being expanded to include web functionality.

The project built during this thesis represents a strong and important starting point for WeShre in achieving its goal of unifying its application across both web and mobile environments. This unification will provide the opportunity to add future features by developing on a single codebase that supports all platforms simultaneously.

Furthermore, the use of these company-chosen technologies provided valuable learning experiences. Working with Expo, Supabase, and Tamagui allowed me to gain new skills and insights into advanced development practices, thereby enhancing my experience in creating cross-platform applications. Learning these new technologies was both challenging and rewarding at the same time, and it greatly contributed to my professional growth.

9.2 Limitations

Throughout the development of this thesis project, I did my best to adhere closely to the design specifications and operational requirements provided by the company at the beginning of the project. Interfaces were carefully designed, and functionalities were developed to align as closely as possible with the company's interests, adhering to and following the feedback and guidance provided by the company as the project progressed. This strong commitment was aimed at achieving the desired requirements and vision of the project.

Despite my best efforts, I was unable to implement all the planned features and elements due to time constraints. Important pages and features were prioritised, but these constraints, along with other limitations, significantly impacted the scope of the project, leading to the incompleteness of some planned pages and features.

The details of these unimplemented aspects are presented in the "**Future Work**" section 9.3, which provides a clear roadmap for subsequent improvements and future development of WeShre's functionality.

9.2.1 Migration challenge

Migrating the database from Firebase to Supabase was a big challenge. Transitioning from a NoSQLB database to a PostgreSQL relational database introduced complexities in redefining unique identifiers and restructuring relationships between data tables. Adapting the database required substantial changes to the original schemas, which posed challenges in maintaining data integrity and functionality. This complexity made it difficult to connect all front-end components to this data.

Ultimately, the decision was made by the company to create a new database directly through Supabase. This new database was structured to be compatible with the front-end elements, ensuring proper linkage and functionality, and thereby addressing the migration challenges.

9.2.2 Tamagui Limitations

The use of the Tamagui library introduced several limitations. Despite its effectiveness in developing responsive and cross-platform interfaces, we encountered many constraints related to this library, especially when implementing interfaces and adapting them to the desired design. This library lacks comprehensive documentation and many real-world examples compared to other libraries, which made it difficult to solve issues encountered during the project's implementation. In ad-

dition, this library is relatively new, still under development, and lacks stability compared to other libraries.

Although Supabase provides real-time data capabilities, it was difficult to ensure that the components of this library interacted correctly with these real-time changes and often resulted in performance delays. Lastly, due to the regular updates released by Expo, it was difficult to keep up with these updates while integrating Tamagui, and often required corresponding changes to the integration of this library.

9.2.3 Expo Compatibility Issues

Despite the overall strength of the Expo platform, we encountered several challenges and limitations during this project. One of these challenges was managing compatibility across different platforms and its inconsistent support for certain components that do not have standardised cross-platform support. A notable example from our project was the map component used on the event page. While this component is supported on mobile devices, it lacks similar support on the web. This led to the implementation of many different solutions for the same feature to ensure it worked across all platforms, which was time-consuming. This discrepancy is just one example of separate implementations required to get the same feature to ensure functionality on both platforms, which subsequently increased the development time for this project.

9.3 Future work

To conclude this thesis, we list below potential improvements and new features that could be integrated into the existing system to enhance its overall capabilities.

9.3.1 Future Interface Additions

Sign Up Interface Create a sign-up interface to enable new user registrations. In fact, this page was not implemented because the primary focus of this thesis was designing interfaces for existing members (i.e. those who already have accounts on the platform). Therefore, the creation of this page was not considered a high priority in the context of this thesis.

Chat Interface Implement a chat interface to allow users to communicate within the platform. Adding a chat feature in future updates will enable event organisers and attendees to communicate directly thus facilitating event organisation and planning.

News Interface Create a simple news page that can be accessed by clicking on the blog cards on the home page. This page will only display the full information of the news items. Similar to the sign-up interface, the design of this page was not prioritised during the thesis as the focus was on other main interfaces.

9.3.2 Expanding Authentication Methods

Add Apple ID, phone number, and Google login capabilities in feature updates. This will make the platform easier to access and provide users with a variety of authentication options.

9.3.3 Adding Notification System

Develop a notification system to alert users to new events, chat messages, and other activities, thereby enhancing the application's capabilities and user experience.

9.3.4 Integrating Ratings and Comments

Implement additional features that allow users to rate and comment on events directly from the interfaces, allowing the platform to become more integrated.

Bibliography

- [1] *Advanced configuration: Root layout in expo router*, <https://docs.expo.dev/router/advanced/root-layout/>.
- [2] *Balancing between platforms: Native and web app development with tamagui*, <https://www.themorrow.digital/blog/balancing-between-platforms-native-and-web-app-development-with-tamagui>.
- [3] O. Bilgili, *Exploring expo in react native: A comprehensive guide to cross-platform app development*, <https://omurbilgili.medium.com/exploring-expo-in-react-native-a-comprehensive-guide-to-cross-platform-app-development-45e6a3bfa111>, Dec 22, 2023.
- [4] *Chatgpt*, <https://chatgpt.com/>.
- [5] *Comparison of firebase vs. supabase*, <https://www.whitelotuscorporation.com/firebase-vs-supabase/>.
- [6] *Continuous native generation*, <https://docs.expo.dev/workflow/continuous-native-generation/>.
- [7] *Customizing metro in expo projects*, <https://docs.expo.dev/guides/customizing-metro/>.
- [8] *Cypress: Web testing built for developers*, <https://www.cypress.io/>.
- [9] *Deno*, <https://deno.com/>.
- [10] *End-to-end test with cypress*, <https://docs.cypress.io/guides/end-to-end-testing/writing-your-first-end-to-end-test>.
- [11] *Expo cli documentation*, <https://docs.expo.dev/more/expo-cli/>.
- [12] *Expo configuration documentation*, <https://docs.expo.dev/workflow/configuration/>.
- [13] *Expo development platform*, <https://expo.dev/>.
- [14] *Expo go: Quick access to expo tools and resources*, <https://expo.dev/go>.
- [15] *Expo metro configuration*, <https://docs.expo.dev/versions/latest/config/metro/>.
- [16] *Firebase*, <https://firebase.google.com/>.
- [17] *Firebase*, <https://firebase.google.com/>.
- [18] *Firebase authentication documentation*, <https://firebase.google.com/docs/auth>.
- [19] *Firebase firestore documentation*, <https://firebase.google.com/docs/firestore>.

-
- [20] *Firestore vs. supabase: A detailed comparison*, <https://flatirons.com/blog/firebase-vs-supabase/#:~:text=Firestore%20takes%20a%20more%20abstracted,using%20PostgreSQL's%20PL/pgSQL%20language..>
 - [21] *Getting started with cypress: Opening the app*, <https://docs.cypress.io/guides/getting-started/opening-the-app>.
 - [22] *Introduction to expo router*, <https://docs.expo.dev/router/introduction/>.
 - [23] *Managing assets in expo*, <https://docs.expo.dev/guides/assets/>.
 - [24] *Migrating to supabase: Firebase authentication*, <https://supabase.com/docs/guides/resources/migrating-to-supabase/firebase-auth>.
 - [25] *Migrating to supabase: Firestore data*, <https://supabase.com/docs/guides/resources/migrating-to-supabase/firestore-data>.
 - [26] *Overview of component testing with cypress*, <https://docs.cypress.io/guides/component-testing/overview>.
 - [27] *Postgresql*, <https://www.postgresql.org/>.
 - [28] *Postgresql documentation*, <https://www.postgresql.org/docs/current/index.html>.
 - [29] *Postgres documentation*, <https://postgrest.org/en/v12/>.
 - [30] *React native: A framework for building native apps using react*, <https://reactnative.dev/>.
 - [31] *React navigation: Routing and navigation for react native apps*, <https://reactnavigation.org/>.
 - [32] *React: A javascript library for building user interfaces*, <https://react.dev/>.
 - [33] *Supabase*, <https://supabase.com/>.
 - [34] *Supabase ai guide*, <https://supabase.com/docs/guides/ai>.
 - [35] *Supabase api guide*, <https://supabase.com/docs/guides/api>.
 - [36] *Supabase architecture overview*, <https://supabase.com/docs/guides/getting-started/architecture>.
 - [37] *Supabase authentication guide*, <https://supabase.com/docs/guides/auth>.
 - [38] *Supabase database overview*, <https://supabase.com/docs/guides/database/overview>.
 - [39] *Supabase documentation: Generates an email action link*, <https://supabase.com/docs/reference/self-hosting-auth/generates-an-email-action-link>.

-
- [40] *Supabase functions guide*, <https://supabase.com/docs/guides/functions>.
 - [41] *Supabase graphql guide*, <https://supabase.com/docs/guides/graphql>.
 - [42] *Supabase realtime guide*, <https://supabase.com/docs/guides/realtime>.
 - [43] *Supabase self-hosting storage: Getting started*, <https://supabase.com/docs/reference/self-hosting-storage/start>.
 - [44] *Supabase storage guide*, <https://supabase.com/docs/guides/storage>.
 - [45] *Supabase vs firebase: Comprehensive comparison*, <https://www.restack.io/docs/supabase-knowledge-supabase-vs-firebase-comparison>.
 - [46] *Supervisor documentation*, <https://supabase.github.io/supervisor/>.
 - [47] *Tamagui*, <https://tamagui.dev/>.
 - [48] *Tamagui and react native: Create faster design systems*, <https://blog.logrocket.com/tamagui-react-native-create-faster-design-systems/>.
 - [49] *Tamagui colour tokens*, <https://tamagui.dev/docs/intro/colors>.
 - [50] *Tamagui documentation: Core configuration and shorthand properties*, <https://tamagui.dev/docs/core/configuration>.
 - [51] *Tamagui documentation: The core styled functionality*, <https://tamagui.dev/docs/core/styled>.
 - [52] *Testing applications with cypress and xray*, <https://www.getxray.app/blog/testing-applications-with-cypress-xray>.
 - [53] M. J. Uddin, *Building your first mobile app with react native and expo: A comprehensive guide*, <https://medium.com/devsorigin/building-your-first-mobile-app-with-react-native-and-expo-a-comprehensive-guide-249d2a61a265>, Dec 3, 2023.
 - [54] *Usemedia*, <https://tamagui.dev/docs/core/use-media>.
 - [55] *Using libraries with expo*, <https://docs.expo.dev/workflow/using-libraries/>.
 - [56] *Webmin*, <https://webmin.com/>.
 - [57] *Webmin*, <https://webmin.com/virtualmin/>.
 - [58] *Webmin*, <https://webmin.com/docs/modules/file-manager/>.
 - [59] *Weshre: A platform for real human interaction*, <https://www.weshre.com/>.

A | Appendix

A.1 Responsive Web Interfaces on Mobile Devices

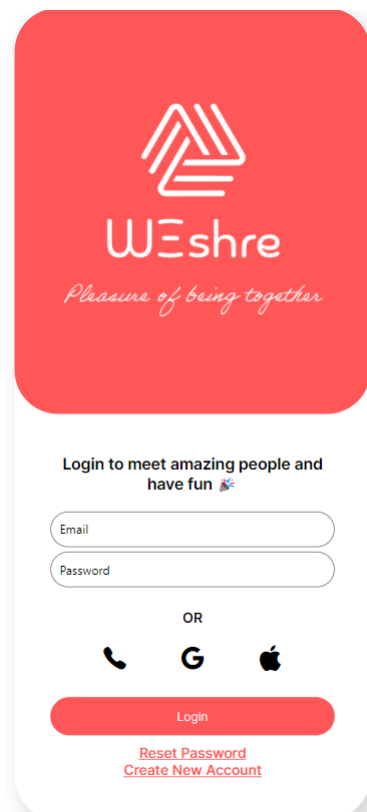


Figure A.1: Responsive Login page design for smaller screens.

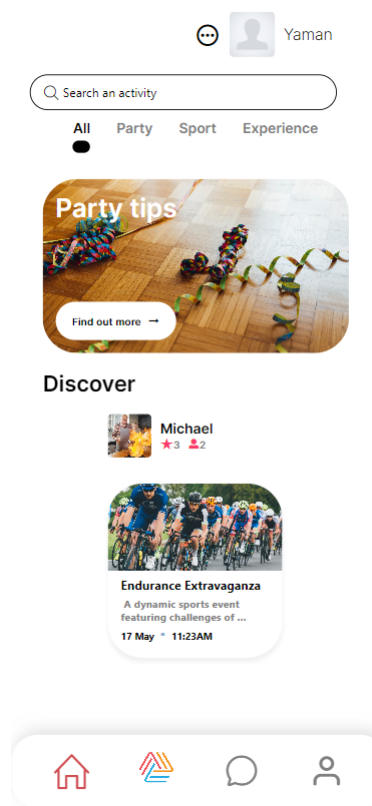


Figure A.2: Responsive design of the home page on smaller screens.

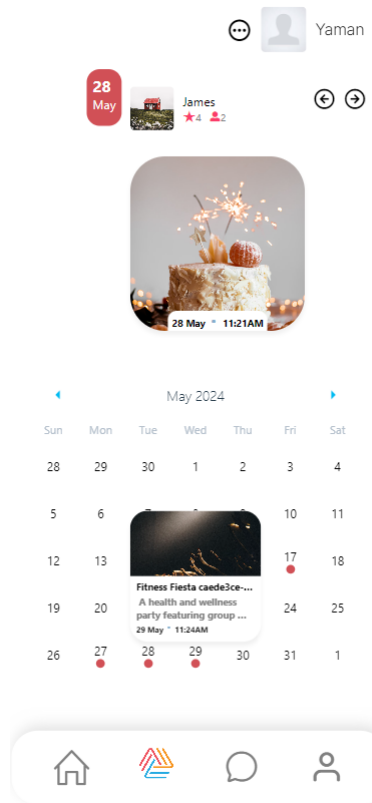


Figure A.3: Responsive calendar page design for smaller screens.

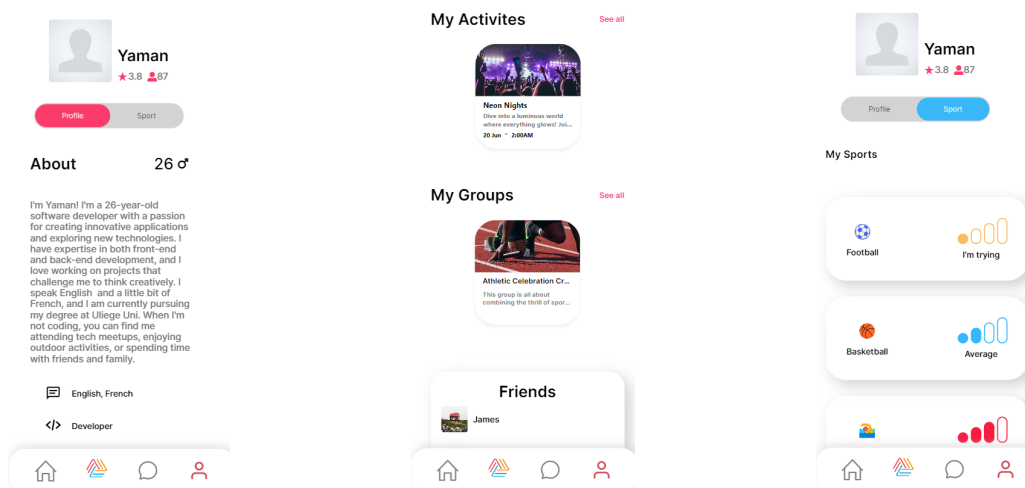


Figure A.4: Responsive design of the profile page for smaller screens.

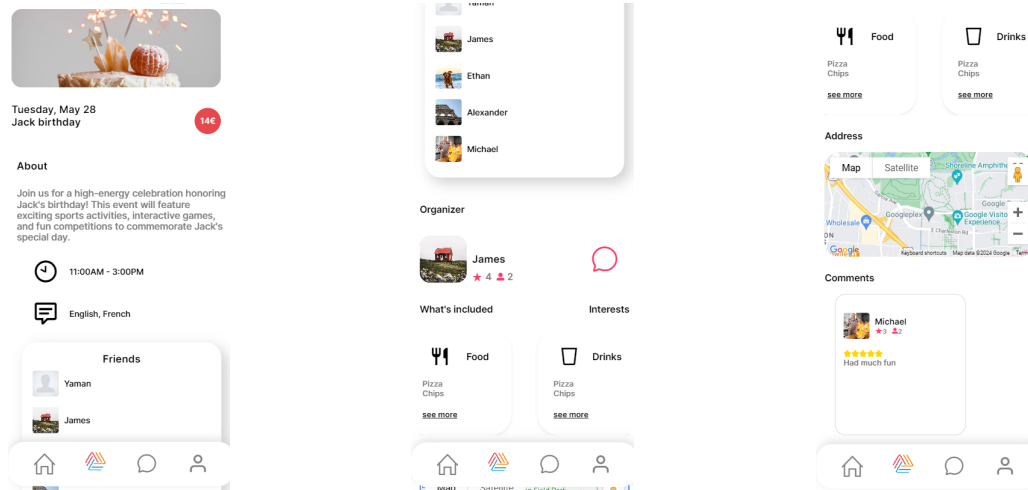


Figure A.5: Responsive event page design for smaller screens.

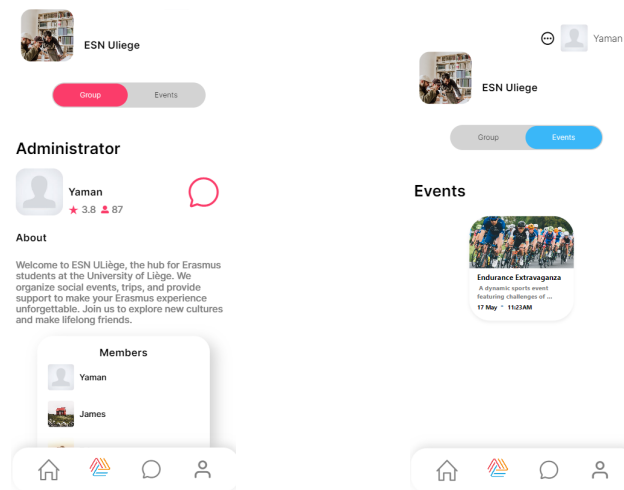


Figure A.6: Responsive group page design for smaller screens.