Mobile App Development for Youth Association Management

Thomas Biselx

Haaga-Helia University of Applied Sciences
Degree Programme in Business Information Technology
Bachelor's Thesis
2025

Abstract

Author(s)
Thomas Biselx

Degree
Bachelor's Degree in Business Information Technology

Report/Thesis Title
Mobile App Development for Youth Association Management

Number of pages and appendix pages
54 + 7

In Switzerland, young people living in villages outside urban areas often establish youth associations to foster community engagement, organize festive events, and support local initiatives. These associations play a vital role in preserving village traditions and strengthening social cohesion. However, limited financial resources often hinder their ability to adopt digital tools for managing memberships, finances, and events effectively.

This thesis examines the development of a mobile application using React Native to provide a cost-effective solution for youth association management. The development process followed a predefined structure, with planning, system architecture, and core functionalities, including member management, financial tracking, document storage, and a shared calendar, specified from the outset. Firebase was chosen as the backend to ensure secure data handling and real-time synchronization. While implementation was guided by initial specifications, user feedback may be incorporated where possible to inform refinements and future improvements.

The outcome is a functional prototype tailored to the administrative needs of small, community-based associations. In addition to improving operational efficiency, the study explores the viability of sustaining the application using free-tier services and modern development frameworks. The findings contribute to the digitalization of grassroots organizations and present a scalable model that may be adapted to similar contexts.

Key words
Mobile application, React Native, Firebase, User Experience, Youth associations

Table of Contents

1 Introduction

Small youth associations are emerging rapidly, particularly in rural areas such as villages in Switzerland. These organizations, often run by volunteers with limited resources, play a crucial role in fostering community engagement. However, as they grow, managing their operations becomes increasingly complex without efficient digital tools. Administrative tasks such as membership tracking, financial management, and document storage require structured and secure solutions to ensure smooth operations. Many associations lack the financial means to invest in expensive management software, forcing them to rely on manual processes or generic digital tools that, while functional, are often not cost-effective for their scale.

The cost, data privacy, and customization of digital solutions are major concerns for these associations. Existing software often requires high subscription fees, making it unaffordable for small organizations with limited budgets. Additionally, youth associations manage sensitive personal and financial data, making privacy and security critical factors. Many commercially available solutions store data on external servers without offering proper control over user access and information security, which can pose a risk of unauthorized access or data misuse. Furthermore, most off-the-shelf solutions are not customizable, limiting the ability of associations to tailor the system to their membership structures, yearly admission fee tracking, and document organization.

This thesis explores the design and development of a mobile application that addresses these challenges by offering a cost-effective, customizable, and privacy-focused digital solution. The proposed application seeks to streamline administrative processes, reduce manual workload, and enhance operational efficiency while maintaining strong data security measures. The system will incorporate role-based access control and encrypted data storage to ensure that only authorized members can access sensitive information.

With the increasing reliance on mobile technology (Nath & Mukherjee 2015), this project presents a unique opportunity to create a platform that not only enhances operational management but also fosters better communication and engagement among members. By focusing on affordability, privacy, and customization, this study aims to bridge the gap between the administrative needs of youth associations and the limitations of existing software. In the context of rural Swiss villages, where associations rely heavily on volunteers and often lack dedicated IT support, a secure and adaptable management tool can significantly improve organizational efficiency and member engagement.

1.1 Context and Background

Having been involved in the management of a youth association for the past five years, valuable firsthand experience has been gained in understanding the challenges these organizations face. The increasing use of mobile phones in professional environments (Nath & Mukherjee 2015) has made them an indispensable tool for youth associations, which often operate with limited resources and lack dedicated office spaces. Managing administrative tasks efficiently remains difficult, as many associations rely on manual processes or scattered digital tools, which can lead to inefficiencies and mismanagement.

Youth associations often face difficulties in keeping track of their members, including monitoring yearly admission fee payments and maintaining an updated database. Financial management is another challenge, as recording income, tracking expenses, and handling invoices without a proper system can become overwhelming. Additionally, storing and accessing essential documents, such as meeting minutes and financial records, is often disorganized, making it difficult for committee members to retrieve necessary information. Planning events and coordinating schedules also require structured tools to ensure smooth communication among members. Furthermore, as these associations rely heavily on volunteers, any digital solution must be intuitive and require minimal technical expertise to ensure broad adoption.

Previous studies and practical experiences highlight the importance of digital transformation in non-profit organizations (Nikita et al. 2024). Secure, efficient, and accessible management systems play a crucial role in improving organizational efficiency while reducing paperwork and administrative overhead. Many existing tools are either too expensive or lack the flexibility needed by small youth associations, further emphasizing the need for an affordable and adaptable solution.

By leveraging mobile technology, this project aims to develop a practical and user-friendly application that addresses these challenges. The proposed solution seeks to streamline administrative tasks, improve financial transparency, ensure secure document storage, and enhance communication among members. Ultimately, the goal is to empower youth associations with a cost-effective digital tool that supports their growth, fosters engagement, and facilitates better resource management.

1.2 Objectives and Minimum Viable Product (MVP)

The primary objective of this thesis is to design and develop a mobile application that addresses the administrative needs of youth associations. The application will integrate key functionalities such as member management, accounting, invoice tracking, document storage for example meeting minutes and PDFs, and a shared calendar to enhance organizational efficiency. The main research question guiding this thesis is:

"How can a mobile application effectively streamline the administrative processes of youth associations to enhance operational efficiency and member engagement?"

To explore this question, the research will examine sub-questions related to user needs, system architecture, security considerations, and usability testing. The study focuses on small to medium-sized youth associations, ensuring that the application remains scalable, adaptable, and practical within the given constraints. The research does not extend to large-scale enterprise solutions but may provide insights applicable to other community-based organizations.

The MVP (Minimum Viable Product) aims to determine whether the application effectively simplifies administrative tasks and enhances association management. The core functionalities must be fully operational, demonstrating a clear improvement over existing manual process. To evaluate its impact, the MVP will be tested in real-world conditions with youth association representatives, assessing its ability to reduce administrative workload, improve organization, and enhance accessibility for members.

The initial deployment will prioritize Android, specifically on a Samsung Galaxy S24 Plus, ensuring smooth operation on a widely used device. Additionally, efforts will be made to release a beta version on both the App Store and Play Store, facilitating early user feedback and iterative improvements. The focus is on validating the application is a viable, user-friendly, and effective tool for youth association management before expanding its functionality or platform support.

1.3 Scope and Delimitations of the Research

This study focuses on developing a cross-platform mobile application for iOS and Android, using React Native as the primary framework. Firebase will manage authentication, database management, and real-time data synchronization. The research is limited to small to medium-sized youth associations, excluding larger organizations with more complex administrative needs.

The application will integrate core administrative functionalities, including membership management, yearly admission fee tracking, financial monitoring, invoice handling, document

storage, and event scheduling. However, advanced features such as AI-driven analytics and deep integration with third-party services are beyond the scope of this phase.

To ensure feasibility, unit testing and end-to-end (E2E) testing are not included in the initial development cycle. The focus remains on delivering a functional prototype that demonstrates the application's practical benefits while maintaining data security and long-term usability.

Certain features will not be implemented due to time constraints. Bank integration will be excluded from the MVP, and while system data transfer to a new platform is an important feature for long-term adaptability, it falls outside the scope of this project. Instead, priority is given to developing an efficient, structured digital solution that optimizes administrative workflows and enhances operational efficiency.

1.4 Research Problem

The primary research problem addressed in this thesis is the lack of an efficient, user-friendly, and secure digital solution for managing youth association activities. Sub-problems to be explored include:

1. How can member management be effectively streamlined within the application?
2. What are the best practices for secure financial tracking and invoice management?
3. How can document storage be optimized to ensure easy access and organization?

1.5 Research Method

This thesis will employ a qualitative research approach, utilizing methods such as interviews and usability testing to gather insights from target users. The development process will follow a structured Waterfall methodology, requiring detailed planning and execution in sequential phases. While the methodology emphasizes planning everything in advance, user feedback will still be incorporated at the final stages to refine and improve the application. The research will involve analysing user requirements, designing and developing the application, and conducting thorough testing to ensure it meets the intended object.

1.6 Key Concepts

Several key concepts are central to understanding the objectives and scope of this thesis. These include:

- User-Centered Design: A design methodology that focuses on the needs, expectations, and experiences of end-users, creating intuitive and accessible applications.
- Cross-Platform Development: The use of frameworks like React Native to ensure compatibility across multiple operating systems (Android and iOS).
- Cloud-Based Backend Services: Leveraging Firebase for secure authentication, database management, and real-time synchronization.

1.7 Planned Tools and Technologies

To achieve the project objectives, several tools and technologies will be utilized, including:

- React Native for cross-platform mobile application development.
- Firebase as the backend with services like authentication and data storage.
- Figma for UI/UX design and prototyping.
- GitHub for version control.

2 Theoretical Framework

2.1 Cross Platform Development

Cross-platform development is the process of developing software applications that are designed to function seamlessly on various operating systems. These applications are developed using a unified codebase, enabling them to be compatible with multiple operating systems. This approach facilitates the development of code that is compatible with multiple platforms, including Android and iOS, thereby reducing the need for repeated coding, development time and costs while ensuring consistency in user experience. (Latif et al. 2017, 1).

Cross-platform mobile development frameworks allow developers to create platform-independent applications using web standards like HTML, JavaScript, and CSS. However, some functionalities still require platform-specific code, as frameworks function as middleware that provides native API implementations. Accessing device features such as the camera, accelerometer, and storage often necessitates platform-specific integration. Frameworks like PhoneGap/Cordova expose these native features through JavaScript APIs, wrapping web-based code in a thin native container. Additionally, achieving native performance and a consistent user experience may require custom optimizations, as platform-specific APIs and UI conventions need to be incorporated (Amatya 2013).

Despite these platform-specific requirements, a single codebase with minimal modifications per platform is often sufficient, making cross-platform development an efficient and cost-effective solution. Hybrid apps, which combine web and native technologies, can be distributed via app stores while leveraging native capabilities. Even with the need for some platform-specific code, cross-platform toolkits ensure that most of the code remains reusable, maintaining scalability and efficiency in mobile application development (Amatya 2013).

2.2 React Native

React Native is a prominent open-source JavaScript framework introduced by Facebook in March 2015, designed for the development of cross-platform mobile applications. Based on the React framework, it is widely adopted for its simplicity and efficiency. React Native enables developers to build mobile applications using JavaScript and React, while leveraging native components to ensure optimal performance and responsiveness (Keshari et al. 2023, 1572; Wenhao 2018).

Google introduced Flutter in late 2016 as a direct competitor to React Native. Both frameworks, backed by major technology companies, have benefited from continuous improvements, strong community support, and long-term viability (Wenhao 2018).

Older frameworks such as Cordova, a cross-platform framework, utilize web technologies and depend on a WebView component to render the user interface. While this approach ensures broad compatibility, the WebView component introduces additional overhead, which can impact performance. A key strength of Cordova is its plugin architecture, which allows access to native device functionalities through JavaScript interfaces, enabling developers to integrate features such as camera access, GPS, and file storage (Uniyal et al. 2023).

Cordova also provides seamless integration with various development tools and supports web development frameworks, making it a flexible option for web developers transitioning to mobile applications. Its cross-platform capabilities simplify both development and maintenance, reducing the need for separate native applications. Additionally, applications built with Apache Cordova may exhibit lower CPU and memory usage compared to those developed with React Native, depending on application complexity and processing demands (Uniyal et al. 2023).

While some developers encounter challenges or opt for alternative solutions, React Native remains a compelling choice for this project due to its ability to streamline cross-platform development. One of its key advantages is the use of a single code base for both iOS and Android, significantly reducing development time and effort by allowing developers to reuse a substantial portion of the code across different platforms. However, despite these benefits, additional work may be required to refine the app’s styling and optimize its performance for each platform to achieve a truly native-like experience (React Native, 2025).

Furthermore, React Native streamlines both development and maintenance, offering additional savings in time and effort. It also employs native components to deliver a smoother user experience and benefits from a substantial and engaged developer community. While React Native addresses a wide range of requirements, certain advanced functionalities may still necessitate supplementary native development for full integration (React Native, 2025).

Although React Native applications perform well compared to other cross-platform frameworks, they are often outperformed by fully native applications (Nilsson, 2022, 26). Despite this, React Native remains the most suitable choice for this project due to its efficiency in development and maintenance, along with its strong support for the core features needed in the youth association management application. However, this does not mean that React Native is the ideal solution for all mobile applications; the choice of technology should always be guided by the specific needs and constraints of each project.

2.3 TypeScript

TypeScript was designed with several key goals to enhance JavaScript development. Microsoft aimed to statically identify errors by introducing strong typing and static type checking at compile time, reducing potential runtime issues. As a superset of JavaScript, TypeScript ensures high compatibility with existing JavaScript code while adding object-oriented features such as classes, interfaces, and modules for better structuring and maintainability. It imposes no runtime overhead, as type annotations and TypeScript-specific features are removed during compilation, producing clean JavaScript code. Additionally, it is a cross-platform tool, open-sourced under the Apache license, making it accessible across major operating systems. (Jansen, Vane, Wolff 2016, Chapter 1)

For this project, TypeScript has been selected over JavaScript due to its strong typing system, early error detection, and enhanced tooling support, which contribute to writing more reliable and structured code. Unlike JavaScript, which is loosely typed and can lead to unexpected runtime errors, TypeScript catches errors at compile time, reducing debugging time and improving overall code quality. (Islam 2023, 20-30)

2.4 Youth Associations

A youth association is an organization run by and for young people, aiming to foster socialization, autonomy, and active citizenship. These associations provide spaces where young individuals can develop leadership skills, engage in collective projects, and contribute to their communities. (Many 2007, 15).

However, they often struggle with inadequate funding and institutional support. Many governments prioritize youth protection policies over empowerment initiatives, leaving these associations underfunded. This lack of resources limits their ability to reach broader audiences, sustain long-term projects, and fully support youth participation in society. (Many 2007).

The stakeholders of this project are the association committees, which play a crucial role in governance. According to Article 69(1) of the Swiss Civil Code (CC, SR 210): "The committee is entitled and obliged according to the powers that it is granted under the articles of association to manage and represent the association.” This project aims to support committees in fulfilling their duties effectively by providing them with the necessary resources and guidance to strengthen youth participation.

2.5 Key Functional Areas

In this section, the main aspects of the application will be discussed, and the Minimum Viable Product (MVP) will be outlined based on these key points.

2.5.1 Member Management

According to Article 61a of the Swiss Civil Code (CC, SR 210): “1 Associations that must be entered in the commercial register shall keep a list of their members, including their first and surnames, business names and addresses. 2 They shall keep the list so that it may be accessed at any time in Switzerland.” To comply with these legal requirements, the application will integrate a secure and accessible member database, enabling association committees to efficiently manage and update member records while ensuring transparency and legal compliance.

According to Article 6 of the Swiss Federal Act on Data Protection (FADP), the processing of personal data must be lawful, conducted in good faith, and proportionate to its purpose. Data collection is restricted to specific and recognizable purposes, and any further processing must remain compatible with these objectives. Once the data is no longer needed, it must be deleted or anonymized. Additionally, organizations must ensure that stored data is accurate and take appropriate measures to correct or remove any incorrect information. When consent is required, it must be freely given and based on informed decision-making. In cases involving sensitive data or high-risk profiling, explicit consent is mandatory (Art. 6 LPD).

According to the Federal Data Protection and Information Commissioner (FDPIC): “The association committee is responsible for ensuring that member data is used lawfully. The committee may only request association members to provide personal data that is directly related to the purpose of the association as set out in its statutes.” This means that only necessary data will be collected, and it will be used exclusively for management purposes.

For these reasons, a member management system will be integrated into the application. This feature will allow authorized users to view the member list, update information, add new members, and remove existing ones, ensuring compliance with Swiss data protection regulations. The system will adhere to FDPIC guidelines, collecting only the necessary data and using it exclusively for administrative purposes, as outlined in the association's statutes. This approach enhances data security, transparency, and compliance with Swiss legal requirements.

2.5.2 Financial Tracking

According to Article 69a of the Swiss Civil Code (CC, SR 210): “The committee shall maintain the association’s business ledgers. The provisions of the Code of Obligations on commercial bookkeeping and accounting apply mutatis mutandis.” To ensure compliance with financial management obligations, the application will include a financial tracking module. This feature will assist in recording transactions and generating financial reports. By integrating these tools, the application will support associations in maintaining transparent and accurate financial records in line with Swiss legal requirements.

2.5.3 Document Storage

According to Article 64 of the Swiss Civil Code (CC, SR 210): “1 The general meeting of members is the supreme governing body of the association. 2 The general meeting is called by the committee.” To maintain a written record of discussions during general meetings, minutes must be prepared, especially when registering the association with the commercial register. The Ordinance on the Commercial Register (ORC), Article 90a, specifies that the registration application must include supporting documents such as minutes of the general meeting. These minutes should confirm the adoption of the statutes and the appointment of management members.

To facilitate document management and legal compliance, the application will include a dedicated storage space for these official records, ensuring accessibility and traceability of key association documents.

2.5.4 Shared Calendar

There are no specific legal requirements mandating the use of a shared calendar for associations. However, the best practices recommend implementing such a tool to enhance collaboration and knowledge sharing within organizations. Research highlights that shared calendars and emails help staff stay informed about each other’s schedules and activities, improving coordination and efficiency (Wynne 2011).

For this reason, the application will include a shared calendar feature that allows users to add, modify, and delete events, ensuring better planning and communication within the association.

2.6 User Experience (UX) in Mobile Programming

User Experience (UX) is a crucial factor in mobile application development, directly influencing user adoption, retention, and satisfaction (Pinchot 2020, 202). Unlike traditional software, mobile applications present unique interaction challenges, such as limited screen space, touch-based navigation, and varying device capabilities (Punchoojit & Hongwarittorrn 2017).

UX has evolved from its early foundations in usability and cognitive psychology (Nielsen 1993, 115) to an increasingly comprehensive discipline that deepens our understanding of user-centered design, incorporating advanced research in human psychology, accessibility, and cognitive load theory while expanding into new areas like emotional design (Norman 2013).

A well-structured UX ensures that users can navigate effortlessly, complete tasks efficiently, and engage meaningfully with the application.

2.6.1 Human-Computer Interaction (HCI) and Its Role in UX

Human-Computer Interaction (HCI) is a research field that emerged in the early 1980s, focusing on enhancing interactions between humans and computers to improve usability, efficiency, and overall user satisfaction. The field emphasizes usability, ensuring that digital systems are intuitive, easy to learn, and accessible, thereby enhancing the user experience (Carroll 2009, 27).

HCI adopts an interdisciplinary approach, drawing on knowledge from computer science, psychology, design, and cognitive science to develop more intuitive and user-friendly systems. It has evolved from early studies in ergonomics and cognitive psychology to accommodate increasingly sophisticated user interfaces, transitioning from command-line interactions to graphical user interfaces (GUIs) and, more recently, to touch-based, multimodal, and emerging interaction paradigms (Dix 2017; Norman 2013).

2.6.2 Application of HCI in UX

HCI principles fundamentally shape UX design practices, with core concepts like affordances helping users intuitively recognize interface functionality through visual cues (Norman 2013). Design is guided by established usability heuristics, including error prevention, system feedback, and visibility of system status (Nielsen 1993). Modern interfaces increasingly incorporate multimodal interaction through touch, voice, and gestural inputs to enhance accessibility and engagement (Turk 2013). These principles collectively contribute to creating more intuitive and efficient user experiences.

2.6.3 User-Centered Design (UCD)

User-Centered Design (UCD) is a design approach that relies on the active involvement of users to better understand their needs and task requirements (Mao et al. 2005). Unlike traditional system-centric approaches that prioritize technical capabilities, UCD emphasizes iterative design processes driven by user feedback and continuous usability evaluation (Gulliksen et al. 2003, 402). This methodology has become fundamental in modern technology development, particularly in domains like e-commerce where user experience directly impacts business success (Mao et al. 2005).

2.6.4 UX Research Methods in UCD

UCD employs various methods to understand and address user needs. The design process begins with observing potential users in their natural environments to understand their interests, motivations, and needs. Iterative design is crucial to refining products, involving systematic cycles of observation, idea generation, prototyping, and testing. This iterative approach, particularly through rapid prototyping, allows teams to efficiently identify and address usability issues before final implementation (Norman 2013).

2.7 Firebase

In developing a mobile application for managing a youth association, Firebase offers a comprehensive Backend-as-a-Service (BaaS) solution that accelerates development and enhances functionality. Its suite of services, including user authentication, real-time data synchronization, file storage, and push notifications, addresses the core requirements of the project. Firebase provides a variety of built-in functionalities that significantly reduce development time, such as ready-made authentication systems, real-time databases, and high-speed hosting (Tarif 2024).

With Firebase's Realtime Database and Cloud Firestore, data is automatically mirrored across all clients in real-time. This ensures that all users have consistent and up-to-date information, which is crucial for features like shared calendars and member management. Additionally, Firebase supports offline capabilities, allowing data to be stored locally and synchronized when the device reconnects to the internet (Tarif 2024). Firebase Authentication offers a secure and straightforward method for managing user sign-ins, supporting various authentication methods such as email/password and social media logins. This simplifies the process of implementing user authentication, enhancing security and user experience (Tarif 2024).

3 Empirical part

3.1 Project overview

The system architecture, as illustrated in Figure 1, is based on a React Native mobile application that interacts with multiple backend services through Firebase. The React Native App serves as the primary interface for users to manage association activities such as member registrations, financial tracking, and document storage.

When a user interacts with the application, requests are directed to the corresponding Firebase services:

- Firebase Authentication manages user login and authentication, ensuring secure and controlled access to the system.
- Firestore Database handles the storage of dynamic data, including member information, event schedules, financial records, and other association-related data.

This architecture ensures a seamless and efficient system where users can interact with the mobile application while Firebase services manage authentication, real-time data synchronization, and secure storage operations in the background.

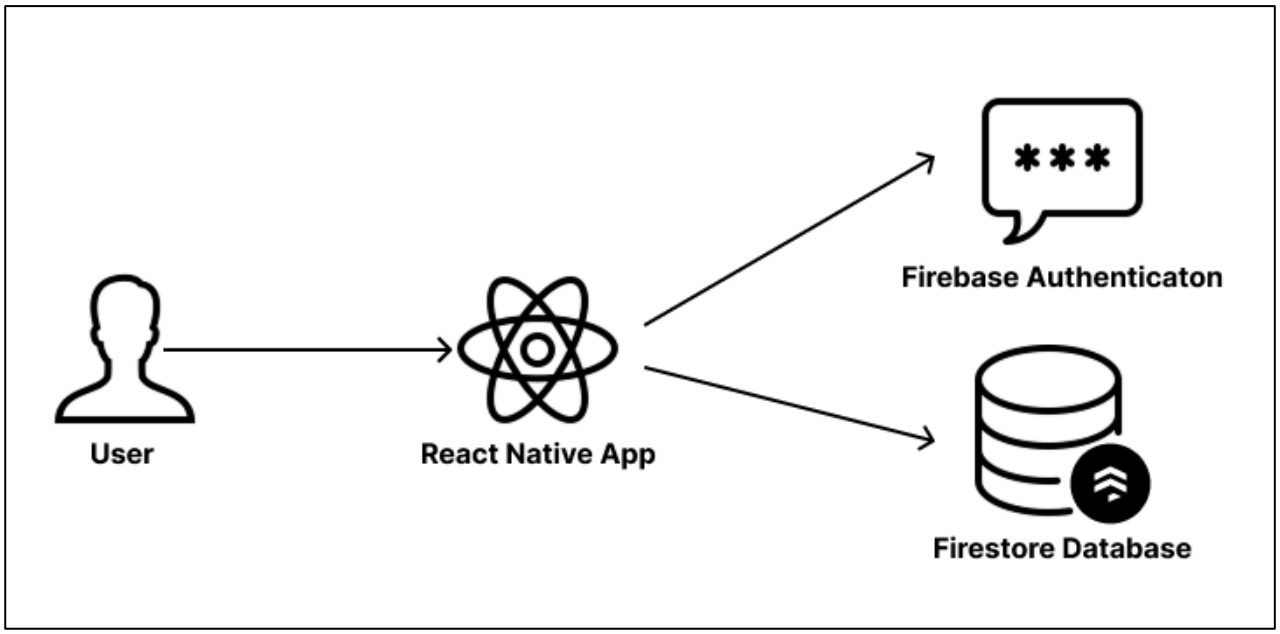


Figure 1 System Architecture Diagram

3.2 Project phasing

3.2.1 Prototyping with Figma

The project will begin with a prototyping phase, where the primary objective is to design the user interface and validate the user experience before proceeding with development. Wireframes and interactive mock-ups will be created using Figma, focusing on key features such as member management, financial tracking, document storage, and a shared calendar.

User testing will be conducted on the prototype to gather feedback and improve the design based on usability insights. This step ensures that the interface is intuitive and meets user expectations before the development phase begins.

3.2.2 Backend Development with Firebase

The backend will be implemented using Firebase, ensuring scalability, security, and real-time data synchronization. Several Firebase services will be utilized, including:

- Firebase Authentication for user login and role management.
- Firestore Database for storing association data (members, finances, documents).

Security rules and access controls will be defined to restrict actions based on user roles (e.g., admin vs. standard member). The frontend will then be connected to the Firebase backend, allowing seamless data synchronization across devices.

3.2.3 Frontend Development with React Native & TypeScript

Once the design is validated, the frontend development phase will commence using React Native and TypeScript. The initial setup will involve configuring the development environment and integrating necessary libraries such as React Navigation, Redux, and Tailwind CSS.

The main components to be developed include:

- User authentication (registration, login).
- Dashboard displaying an overview of finances and upcoming events.
- Member management (adding, editing, and removing members).
- Financial management, including tracking income and expenses, associating transactions with specific events, marking membership fees as paid or unpaid, uploading receipts for expense verification, and comparing recorded finances with the association's bank account balance.
- Document storage and access (minutes, invoices, PDFs).
- Integration of a shared calendar for event management.

3.2.4 Deployment to App Store & Play Store

Before deployment, the application will be optimized for performance, reducing file sizes and fixing potential bugs. The project will then be compiled for Android and iOS, ensuring compatibility across different devices.

The deployment process will involve:

- Creating developer accounts on Google Play Store and Apple App Store.
- Generating builds for both platforms.
- Submitting the application for approval, including preparing a description, screenshots, and compliance documentation.

3.2.5 User Testing & Feedback

After deployment, a user testing phase will be conducted with a group of association members to assess usability and performance. Feedback will be collected through interviews, identifying areas for improvement.

Based on user insights, updates will be planned to fix bugs, enhance user experience, and optimize features. A post-launch update will be scheduled to incorporate improvements and address any issues identified during real-world usage.

3.3 Project implementation

3.3.1 Phase 1: Prototyping with Figma

The application was initially designed to begin with a straightforward login page, as shown in Figure 2. Users are required to sign in before accessing the platform's functionalities. However, before logging in, users must first create an association and provide a valid email address.

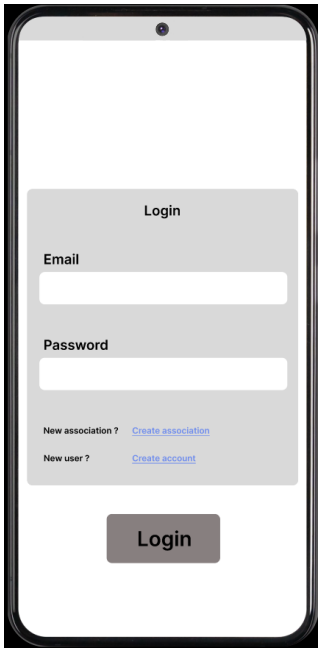


Figure 2 Login Prototype Page

The process of creating an association is illustrated in Figure 3, where users enter essential information such as the association's name, city, postal code, and a contact email. Once the association is created, new users can proceed to create their personal account, as shown in Figure 4.

Upon registration, a confirmation email is automatically sent to the provided email address to verify its validity. Furthermore, the association owner is granted the authority to accept or reject membership requests from individuals who wish to join their association. This mechanism ensures an initial layer of control and security over the platform's access management.

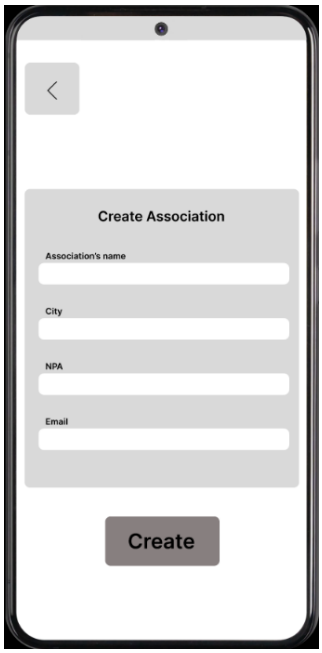


Figure 3 Create Association Prototype Page

Figure 4 Signup Prototype Page

The home page design, as illustrated in Figure 5, displays the name of the association alongside key practical information that users frequently need. This includes the total number of registered members and the current balance of the association's bank account, which can be updated by the treasurer.

Additionally, the page highlights upcoming events and displays the most recently accessed documents, providing users with a quick overview of important activities and resources. Functioning as a general dashboard, the home page centralizes access to essential information.

Users can also view their personal account details and have the option to log out of the application. A navigation menu located at the bottom of the screen allows users to seamlessly switch between different sections of the application, promoting ease of use and intuitive navigation.

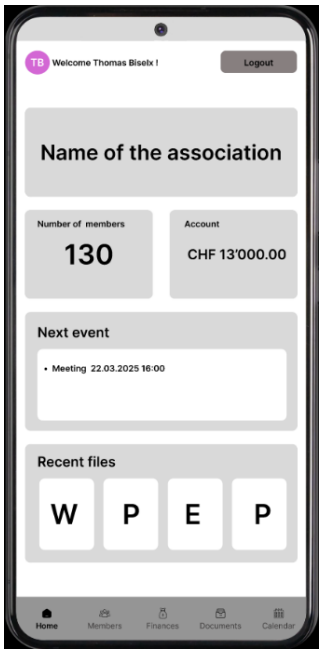


Figure 5 Home Prototype Page

The member management interface, as shown in Figure 6, enables users to add new members, remove existing ones, and edit member information efficiently. An option to export the member list into CSV or Excel format is provided, facilitating the creation of attendance lists for various events.

Each member row is selectable, allowing for quick actions such as edits or deletions. The page also integrates search and filter functionalities to simplify navigation and make the management of large member lists more user-friendly.

Additionally, this interface serves as the point where membership fee payments are tracked. Users can mark whether each member has paid or not for the current year. At the bottom of the page, key summary statistics are displayed, including the total number of registered members and the number of members who have paid their annual fees.

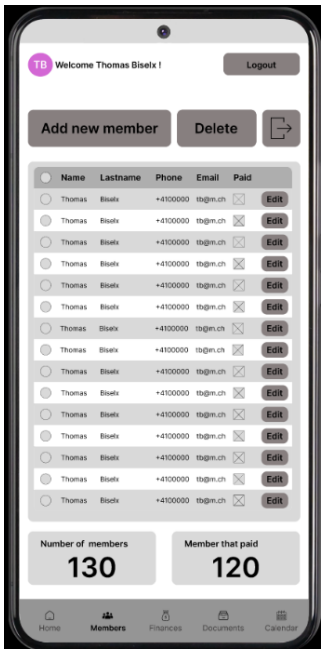


Figure 6 Member Management Prototype Page

The financial management interface, as presented in Figure 7 and 8, provides users with the ability to manage and review the association's financial activities on a yearly basis.

On the financial page (Figure 7), users can select the year they wish to view. They are required to add various events corresponding to that year, either by creating new events directly or by selecting existing ones previously created on the calendar page. Membership fees are automatically imported from the member management module to ensure data consistency.

For each event, users can input details such as profits and expenses, and they have the option to upload supporting images, such as receipts, to serve as proof of transactions. The system aggregates all financial entries and calculates the financial results for the selected year.

In Figure 8, the interface shows the financial summary, comparing the calculated financial results with the manually entered bank account balances. Since direct integration with bank accounts is not implemented, manual updates are necessary to reflect actual bank records, ensuring that the financial status remains accurate and verifiable.

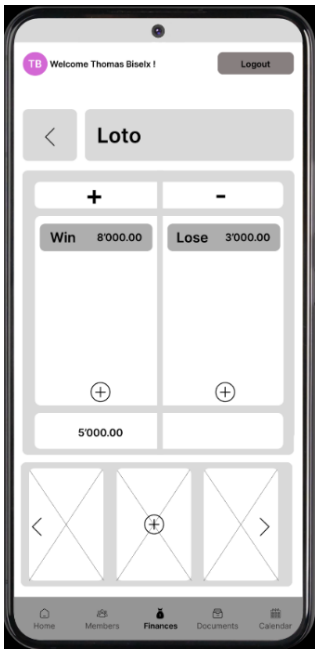


Figure 7 Financial Prototype Page

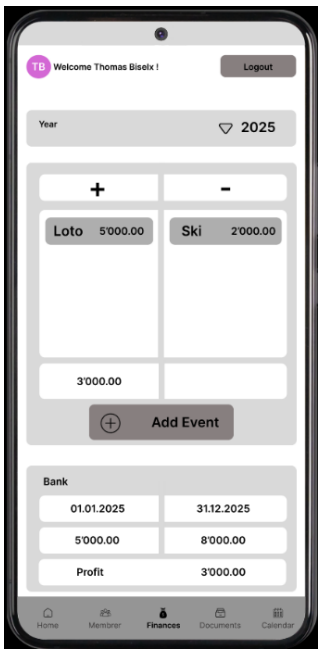


Figure 8 Event Management Prototype Page

The document storage module enables associations to organize and securely store various types of documents. In Figure 9, users can create different folders to categorize documents based on their purpose, such as meeting minutes, statutes, financial reports, and other administrative files. This structured approach facilitates easier access and better organization of critical association documents.

Figure 9 Document Management Prototype Page

Within each folder, as illustrated in Figure 10, users can upload and view files, including Word documents, Excel spreadsheets, PowerPoint presentations, PDFs, and image files. The layout provides a simple and efficient way to manage digital records, ensuring that all members with the appropriate access rights can retrieve the necessary documents when needed.

Figure 10 File Listing Prototype Page

The event management module allows users to view all upcoming events within a shared calendar that is accessible to all members of the association. In Figure 11, the calendar displays the events organized by month and year, providing a clear and intuitive overview of scheduled activities. Users can quickly navigate through the calendar to find upcoming events.

Figure 11 Calendar Prototype Page

When creating a new event, as shown in Figure 12, users are required to input essential information such as the event's name, date, location, and a brief description. Additionally, it is possible to select specific members who will participate in the event if it does not concern all association members.

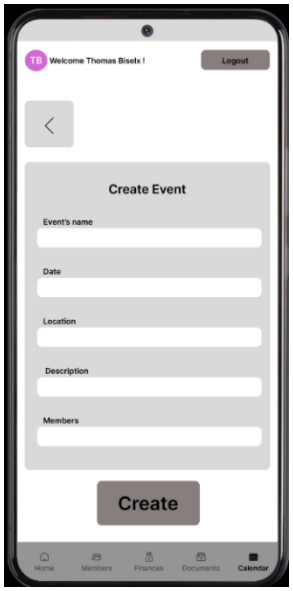


Figure 12 Create Event Prototype Page

Prototype Testing and User Feedback Analysis

The prototype was sent to a youth association in Switzerland, and here is their feedback on the application (Appendix 1) :

First, they immediately understood that it was an application designed for association management and that only the committee members would have access to it. This is a positive aspect for the thesis, as it confirms that the application's purpose is clear without requiring prior knowledge of the project.

Secondly, they found that the application provided almost all the necessary information for managing an association and was very easy to use. This is particularly important, as simplicity and accessibility were key concerns of this thesis.

However, they suggested some feature improvements, such as adding a to-do list to help the committee keep track of tasks. This feature will be integrated into the home page without requiring a separate dedicated page.

The main issue raised during testing was related to the calendar page. The users did not fully understand its purpose and requested a list view of the events. To address this, the calendar page will be redesigned in the development phase to improve clarity. A toggle button will be added to switch between calendar and list views, and the page will be renamed Events to better reflect its function.

Overall, the prototyping phase has been successful, and the backend development phase can now begin.

3.3.2 Phase 2: Backend Development with Firebase

Database Structure

The database is modeled using a collection-based NoSQL approach with Firebase Firestore. Two primary collections are used: users and associations. Each document in the users collection stores user-specific information such as first name, last name, email, and references to their joined associations.

Associations are stored as individual documents in the associations collection. Each association contains general metadata (e.g., name, description, city, postal code) and maintains relationships with other entities such as events, members, finances, folders, documents, settings, and todos, which organize the association's operational data.

The appMembers entity manages the connection between users and associations by storing user IDs and their assigned roles, enabling role-based access control. This relational structure ensures flexibility and scalability, allowing a user to participate in multiple associations, each with distinct permissions.

The overall data model is illustrated in Figure 13, which presents the entities and relationships governing the application's database.

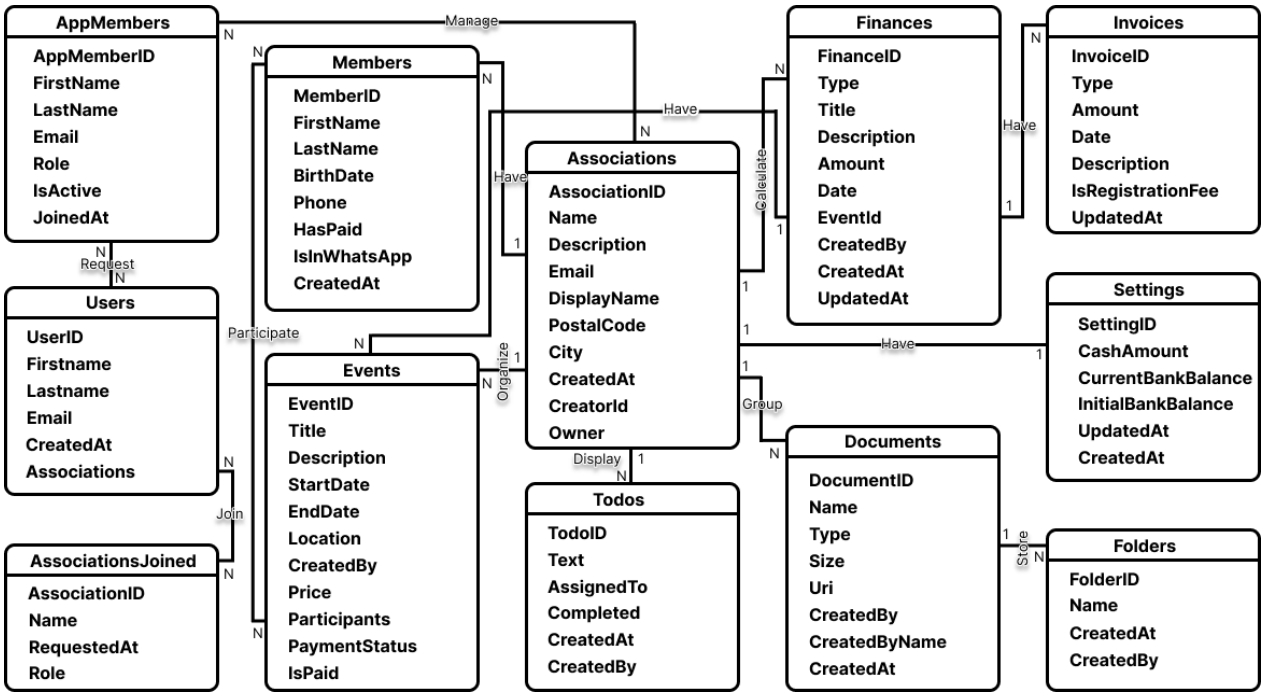


Figure 13 Database Structure

Authentication Service

The authentication service in the application is implemented using React Context and TypeScript. It defines a context named AuthContext that centralizes all authentication-related logic, such as login, logout, password reset, and user or association registration. This context provides a type-safe interface AuthContextType and manages state variables such as the current user, error messages, and loading status. By encapsulating this logic in a context, the application ensures modularity and reusability, allowing components to access authentication state and functions through a custom hook, useAuth. This implementation is illustrated in Figure 9, which presents the structure of the context and the defined authentication functions.

```
29  type AuthContextType = {
30    user: User | null;
31    loading: boolean;
32    error: string | null;
33    login: (email: string, password: string) => Promise<any>;
34    logout: () => Promise<void>;
35    registerUser: (userData: any) => Promise<any>;
36    registerAssociation: (associationData: any) => Promise<any>;
37    resetPassword: (email: string) => Promise<void>;
38    clearError: () => void;
39  };
40
41  const AuthContext = createContext<AuthContextType>(initialState);
42  export const useAuth = () => useContext(AuthContext);
```

Figure 14 Authentication Context Definition

To handle real-time authentication state changes, the app uses Firebase's onAuthStateChanged method. As shown in Figure 10: Observer Pattern, a useEffect hook initializes an observer that listens for authentication status updates. This observer automatically updates the application state whenever a user logs in or out. The unsubscribe() function ensures proper cleanup when the component is unmounted, preventing memory leaks and unnecessary updates.

```
50    useEffect(() => {
51      const unsubscribe = onAuthStateChanged(auth, (currentUser) => {
52        setUser(currentUser);
53        setLoading(false);
54      });
55
56      return () => unsubscribe();
57    }, []);
```

Figure 15 Authentication State Observer Pattern

Figure 11 illustrates how the login function leverages Firebase Authentication to sign in users with their email and password. After authentication, it checks whether the user's email is verified. If not, the user is signed out immediately, and an error message is displayed. This ensures that only verified users can access the application. In addition, proper error handling and loading state management are implemented to improve the user experience during the sign-in process.

```
65  const login = async (email: string, password: string) => {
66    try {
67      setLoading(true);
68      const userCredential = await signInWithEmailAndPassword(auth, email, password);
69
70      if (!userCredential.user.emailVerified) {
71        await signOut(auth);
72        setError('Veuillez vérifier votre email avant de vous connecter');
73        throw new Error('Veuillez vérifier votre email avant de vous connecter');
74      }
75
76      return userCredential;
77    } catch (error) {
78      if (error instanceof Error) {
79        setError(error.message);
80      } else {
81        setError('Erreur de connexion');
82      }
83      throw error;
84    } finally {
85      setLoading(false);
86    }
87  };
```

Figure 16 Login Function with Email Verification

The function illustrated in Figure 12 demonstrates how a new association is created in the Firestore database. It uses setDoc to store key metadata such as the association's name, description, city, postal code, and email. The name field is automatically formatted to be URL-friendly. The user ID is saved as both the owner and creatorId, and a creation timestamp is recorded.

```
151      const associationRef = doc(db, "associations", userId);
152      await setDoc(associationRef, {
153        id: userId,
154        displayName: associationData.name,
155        name: associationData.name.toLowerCase().replace(/\s+/g, '-'),
156        description: associationData.description,
157        postalCode: associationData.postalCode,
158        city: associationData.city,
159        email: associationData.email,
160        owner: userId,
161        creatorId: userId,
162        createdAt: new Date().toISOString()
163      });
```

Figure 17 Association Creation Function

Managing Associations and Members

Once a new association is created, the user who created it is automatically registered as a member with the role of owner. As shown in Figure 13, a new user document is created in the “users” collection using setDoc. This document contains the user’s email, creation date, and an array of associations the user is linked to. The associations field includes the association ID, name, role, and the date the user joined.

```
168     const userRef = doc(db, "users", userId);
169     await setDoc(userRef, {
170       email: associationData.email,
171       firstName: '',
172       lastName: '',
173       createdAt: new Date().toISOString(),
174       associations: [{
175         id: userId,
176         name: associationData.name,
177         role: 'owner',
178         joinedAt: new Date().toISOString()
179       }]
180     });
```

Figure 18 Owner Association Registration

For other users to join an association, they must first request access using the association’s ID. As shown in Figure 14, the requestToJoinAssociation function collects the user’s basic information and creates a pending request inside the sub-collection pendingRequests of the target association.

Additionally, as illustrated in Figure 15, the same function updates the corresponding document in the “users” collection by adding the association to the user's associations array, with the role set to 'pending'. This setup allows the owner of the association to later review and either approve or reject incoming membership requests.

```
93  export const requestToJoinAssociation = async (
94    userId: string,
95    associationId: string,
96    userData: {
97      email: string;
98      firstName: string;
99      lastName: string;
100   }
101 ) => {
102   const associationRef = doc(db, "associations", associationId);
103   const userRef = doc(db, "users", userId);
```

Figure 19 Join Request to an Association

```
131     const pendingRequestsRef = collection(db, `associations/${associationId}/pendingRequests`);
132     await addDoc(pendingRequestsRef, {
133       userId,
134       email: userData.email,
135       firstName,
136       lastName,
137       requestedAt: new Date().toISOString(),
138       associationName: associationData.displayName
139     });
140
141     await updateDoc(userRef, {
142       associations: arrayUnion({
143         id: associationId,
144         name: associationData.displayName,
145         role: 'pending',
146         requestedAt: new Date().toISOString()
147       })
148     });
```

Figure 20 Join Request to an Association (Data Insertion and Update)

When a user submits a request to join an association, the owner can either approve or reject the request. As shown in Figure 16, the function approveMemberRequest retrieves the pending request data and adds the user to the appMembers sub-collection of the target association. The user's role is set to 'admin', and their membership is marked as active.

If the request is rejected (Figure 17), the function rejectMemberRequest simply deletes the pending request document. This flow ensures that only approved users are granted access and assigned roles, while rejected requests are safely discarded.

```
192 export const approveMemberRequest = async (associationId: string, requestId: string) => {
193   try {
194
195     const pendingRequestRef = doc(db, `associations/${associationId}/pendingRequests/${requestId}`);
196     const requestSnap = await getDoc(pendingRequestRef);
197
198     if (!requestSnap.exists()) {
199       throw new Error("Demande non trouvée");
200     }
201
202     const requestData = requestSnap.data();
203
204     const appMemberRef = doc(db, `associations/${associationId}/appMembers`, requestData.userId);
205
206     await setDoc(appMemberRef, {
207       userId: requestData.userId,
208       email: requestData.email,
209       firstName: requestData.firstName || '',
210       lastName: requestData.lastName || '',
211       role: 'admin',
212       joinedAt: new Date().toISOString(),
213       isActive: true
214     });
```

Figure 21 Approve Member Request Function

```
231  export const rejectMemberRequest = async (associationId: string, requestId: string) => {
232    try {
233
234      const pendingRequestRef = doc(db, `associations/${associationId}/pendingRequests/${requestId}`);
235      const requestSnap = await getDoc(pendingRequestRef);
236
237      if (!requestSnap.exists()) {
238        throw new Error("Demande non trouvée");
239      }
240
241      try {
242        await deleteDoc(pendingRequestRef);
243      } catch (deleteError) {
244        console.error("Error during deleteDoc:", deleteError);
```

Figure 22 Reject Member Request Function

Event Management and Finance

This function adds a new event to the events sub-collection of a given association. If the event is marked as paid (isPaid), a corresponding finance entry is also created in the finances sub-collection. The financial record includes the event title, amount, type, and timestamps for tracking. This logic ensures that financial data is automatically linked to events, reducing manual input and improving consistency in record management, as shown in Figure 18.

```
32  export const addEvent = async (
33    associationId: string,
34    eventData: Omit<Event, "id">
35  ): Promise<{ success: boolean; eventId?: string }> => {
36    try {
37      const eventsRef = collection(db, `associations/${associationId}/events`);
38      const docRef = await addDoc(eventsRef, eventData);
39      const eventId = docRef.id;
40
41      if (eventData.isPaid) {
42        const financeEventRef = doc(db,`associations/${associationId}/finances/${eventId}`);
43        await setDoc(financeEventRef, {
44          title: eventData.title,
45          date: eventData.startDate,
46          type: "income",
47          amount: 0,
48          description: eventData.description || "",
49          createdBy: eventData.createdBy,
50          createdAt: eventData.createdAt,
51          updatedAt: new Date().toISOString(),
52          eventId: eventId,
53        });
```

Figure 23 Add Event and Link to Financial Record

The function depicted in Figure 19 processes a list of financial events to calculate the annual balance of an association. It classifies events into categories such as "income", "expense", and "membership", then aggregates the values accordingly. The final balance is determined by subtracting total expenses from the sum of incomes and membership fees. This approach offers a concise financial snapshot of the association's activities, improving transparency and aiding in strategic financial planning.

```
33  export type FinancialEvent = {
34    id?: string;
35    title: string;
36    date: string;
37    type: "expense" | "income" | "membership";
38    amount: number;
39    description?: string;
40    eventId?: string;
41    createdBy?: string;
42    updatedAt?: string;
43    isMembershipFee?: boolean;
44  };
45
46  export const calculateBalance = (events: FinancialEvent[]): YearlyBalance => {
47    const balance: YearlyBalance = {
48      incomeTotal: 0,
49      expenseTotal: 0,
50      membershipTotal: 0,
51      balance: 0
52    };
53
54    events.forEach(event => {
55      const amount = Number(event.amount) || 0;
56
57      if (event.type === 'income') {
58        balance.incomeTotal += amount;
59      } else if (event.type === 'expense') {
60        balance.expenseTotal += amount;
61      } else if (event.type === 'membership') {
62        balance.membershipTotal += amount;
63      }
64    });
65
66    balance.balance = balance.incomeTotal + balance.membershipTotal - balance.expenseTotal;
67    return balance;
68  };
```

Figure 24 Calculate Yearly Balance Function

As shown in Figure 20, this function synchronizes financial data with event participation records. When executed, it retrieves a specific event and verifies whether it is marked as paid. It then calculates the total amount collected by checking which registered participants have completed their payment. If payments exist, the function creates or updates a corresponding financial entry in the finances sub-collection. It uses the eventId to ensure the link between the financial document and the original event is maintained. This document includes event details such as the title, date, amount, and metadata like creator ID and timestamps.

Additionally, the function registers a summary in the nested invoices sub-collection, storing the number of paid registrations and a description for traceability. This process plays a critical role in ensuring data consistency between the event and finance modules. It allows the application to track income directly related to participation, automate financial record generation, and avoid discrepancies between event attendance and revenue tracking. By doing so, it reduces manual input, strengthens financial transparency, and supports better accountability within the association.

```
98  export const syncEventWithFinances = async (
99    associationId: string,
100   eventId: string,
101   userId: string
102 ): Promise<void> => {
103   try {
104     const eventRef = doc(db, `associations/${associationId}/events/${eventId}`);
105     const eventDoc = await getDoc(eventRef);
106     if (!eventDoc.exists() || !eventDoc.data().isPaid) return;
107     const eventData = eventDoc.data();
108     const price = eventData.price || 0;
109     const paymentStatus = eventData.paymentStatus || {};
110     const participants = eventData.participants || [];
111     let paidCount = 0;
112     Object.entries(paymentStatus).forEach(([id, status]) => {
113       if (status === true && participants.includes(id)) {
114         paidCount++;
115       }
116     });
117     const totalPaid = paidCount * price;
118     const financeEventRef = doc( db, `associations/${associationId}/finances/${eventId}`);
119     await setDoc(
120       financeEventRef,
121       {
122         title: eventData.title,
123         date: eventData.startDate,
124         type: "income",
125         amount: totalPaid,
126         description: eventData.description || "",
127         createdBy: eventData.createdBy,
128         createdAt: eventData.createdAt,
129         updatedAt: new Date().toISOString(),
130         eventId: eventId,
131       },
132       { merge: true }
133     );
134
135     const invoicesRef = collection(financeEventRef, "invoices");
136     const registrationQuery = query( invoicesRef, where("isRegistrationFee", "==", true));
137     const querySnapshot = await getDocs(registrationQuery);
138     const invoiceData = {
139       description: `Inscriptions payées (${paidCount}/${participants.length} participants)`,
140       amount: totalPaid,
141       type: "income",
142       date: eventData.startDate,
143       isRegistrationFee: true,
144       updatedAt: new Date().toISOString(),
145     };
```

Figure 25 Synchronization Between Events and Finances

Firestore Security Rules

The rules define access control across the database. Three helper functions are used to simplify permissions: isAuthenticated checks if the user is logged in, isCreator verifies if the user created the association, and isMember checks if the user belongs to the association. Users can create their profile and access only their own data. Associations can be created and read by any authenticated user, but only creators or members are allowed to update them.

Within each association, sub-collections such as appMembers, members, pendingRequests, finances, and invoices are protected using strict access control. Permissions to these documents are granted only to the creator or members of the respective association, except for join requests, which can be initiated by any authenticated user. A fallback rule also provides read and write access to other nested collections, ensuring flexibility while preserving role-based access restrictions. This security structure enables scalable growth of the data model while maintaining reliable access control, as shown in Appendix 2.

Conclusion of the Backend Section

The backend of this application, primarily built using Firebase, plays a crucial role in enabling and securing the core functionalities. It manages user authentication, ensures real-time data synchronization across association members, and provides a secure and scalable structure for storing important data such as documents, events, and user roles. By leveraging Firebase Authentication and Cloud Firestore, the backend architecture remains both flexible and maintainable.

The implemented backend functions effectively handle interactions between members and administrators, supporting a smooth and responsive user experience. As such, the backend serves as the backbone of the system, ensuring the application's reliability and operational efficiency.

3.3.3 Phase 3: Frontend Development with React Native & TypeScript

The frontend of the application follows a modular design, where each screen is dedicated to a specific feature. In this case, the focus is on the Member Screen, which allows users to view and manage members within a selected association. As shown in Figure 21, the MemberScreen component retrieves the currently active association ID via a custom hook useAssociation and passes it as a prop to two child components: MembersStats and MemberTable.

The screen is structured using React Native's SafeAreaView and ScrollView components to ensure proper display across various devices and screen sizes. This design pattern promotes clarity and separation of concerns:

- The MembersStats component is responsible for summarizing key statistics related to the association's members.
- The MemberTable component handles the rendering of detailed member information in a tabular format.

By organizing the user interface in this way, the frontend ensures better maintainability, improved readability of the code, and greater scalability as new features are added. This architecture also enables clear integration of backend data, making the interface both dynamic and user-centered.

```
7   export default function MemberScreen() {
8     const { currentAssociationId } = useAssociation();
9
10    return (
11      <SafeAreaView style={globalStyles.container}>
12        <ScrollView>
13          <MembersStats associationId={currentAssociationId} />
14          <MemberTable associationId={currentAssociationId} />
15        </ScrollView>
16      </SafeAreaView>
17    );
18  }
```

Figure 26 Member Screen

To manage the current working context of the user within the application, a custom hook named useAssociation is implemented. This hook centralizes the logic related to the selected association and provides real-time access to relevant data such as the association's name, description, and total number of members.

Internally, the hook relies on a context provider that maintains the state of the currently active association. It enables the application to:

- Retrieve and store the ID of the currently selected association,
- Automatically update the interface when the user's selected association changes,
- Display key metadata of the association (e.g., name, description),
- Count the number of active members dynamically using a real-time listener.

In addition to data retrieval, the hook exposes methods to switch associations, leave the current one, and refresh the associated data. It also includes internal error management and loading indicators to ensure a smooth and user-friendly experience.

This structure abstracts complex Firestore interactions and state management behind a simple and reusable interface, reinforcing the modular architecture of the frontend. As a result, components consuming the useAssociation hook remain clean, maintainable, and focused solely on rendering data, rather than handling data logic.

As illustrated in Figure 22, the MembersCounts component is responsible for retrieving and monitoring the list of members belonging to a given association. It receives the associationId as a prop and uses the Firestore function onSnapshot to establish a real-time listener on the members sub-collection.

This implementation ensures that any changes in the member list, such as additions, deletions, or updates, are immediately reflected in the UI without requiring manual refreshes. A loading state is also implemented to enhance user experience during data fetching. Error handling is included to catch and report any issues related to data retrieval.

This reactive data-fetching mechanism strengthens the connection between the backend and the frontend, offering both responsiveness and data consistency while minimizing latency in user interactions.

```
22  export default function MembersCounts({ associationId }: MembersCountsProps) {
23      const [members, setMembers] = useState<Member[]>([]);
24      const [loading, setLoading] = useState<boolean>(true);
25
26      useEffect(() => {
27          if (!associationId) {
28              setMembers([]);
29              setLoading(false);
30              return;
31          }
32          setLoading(true);
33
34          const membersRef = collection(db, `associations/${associationId}/members`);
35          const unsubscribe = onSnapshot(
36              membersRef,
37              (snapshot) => {
38                  const fetchedMembers = snapshot.docs.map(doc => ({
39                      id: doc.id,
40                      ...doc.data()
41                  } as Member));
42
43                  setMembers(fetchedMembers);
44                  setLoading(false);
45              },
46              (error) => {
47                  console.error("Erreur lors de l'écoute des membres:", error);
48                  setLoading(false);
49              }
50          );
51          return () => unsubscribe();
52      }, [associationId]);
```

Figure 27 Member Counts

Figure 23 presents the second part of the MembersCounts component, which is responsible for generating and rendering statistical insights based on the list of retrieved members. Several key metrics are calculated: the total number of members, the number of members who have completed their payment hasPaid, and the number of members associated with a WhatsApp contact isInWhatsApp.

These statistics are then displayed using visually distinct UI blocks, each styled using the application's global styles. Conditional rendering is applied to indicate loading states before the data becomes available. This interface design supports intuitive reading of statistics and promotes an at-a-glance understanding of key membership metrics.

The implementation not only enhances the user's ability to track member engagement but also reflects a faithful transformation of the association's data into actionable visual insights. These dynamic statistics serve both administrative needs and member transparency.

```
55      const totalMembers = members.length;
56      const paidMembers = members.filter(m => m.hasPaid).length;
57      const whatsappMembers = members.filter(m => m.isInWhatsApp).length;
58
59      return (
60          <View style={globalStyles.statsContainer}>
61              <View style={globalStyles.statCard}>
62                  <Text style={globalStyles.statNumber}>{loading ? "..." : totalMembers}</Text>
63                  <Text style={globalStyles.statLabel}>Total</Text>
64              </View>
65              <View style={globalStyles.statCard}>
66                  <Text style={[globalStyles.statNumber, { color: '#4CD964' }]}>
67                      {loading ? "..." : paidMembers}
68                  </Text>
69                  <Text style={globalStyles.statLabel}>Cotisations</Text>
70              </View>
71              <View style={globalStyles.statCard}>
72                  <Text style={[globalStyles.statNumber, { color: '#25D366' }]}>
73                      {loading ? "..." : whatsappMembers}
74                  </Text>
75                  <Text style={globalStyles.statLabel}>WhatsApp</Text>
76              </View>
77          </View>
78      );
79  }
```

Figure 28 Display of member counts

Frontend Result and differences with mock-ups

The current version of the login interface, shown in Figure 24, introduces several improvements compared to the initial mock-ups. One of the major conceptual changes was the removal of the option to create an association directly from the login screen. This choice was made to ensure better data consistency and to separate the responsibilities of user identity from association management. Instead, the updated flow clearly distinguishes between account creation and association registration. Users are now required to create a personal account and log in with verified credentials before being allowed to perform association-related operations.

To enhance security and ensure account legitimacy, users must confirm their email address after registration. The verification process is illustrated in Figure 25, which shows the automated email sent by Firebase Authentication containing the validation link.

Additional features have also been implemented in the login interface:

- A "Remember Me" checkbox to improve user experience on subsequent logins,
- A password reset link for users who have forgotten their credentials,
- A cleaner layout aligned with mobile usability best practices.

These adjustments contribute to a more robust and user-friendly authentication system, which aligns better with both technical constraints and user expectations.

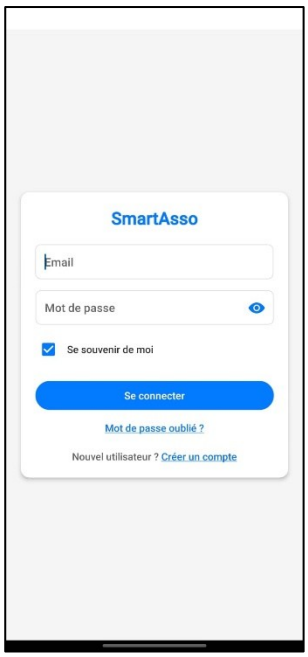


Figure 29 User Login Interface

Figure 30 Email Verification Message

After a successful login, the user is redirected to a landing page where they are presented with three options, as shown in Figure 26. If the user is already a member of one or more associations, they are displayed in a list with their role clearly indicated (e.g., administrator or member). If the user is not yet associated with any organization, they may choose to either create a new association or request to join an existing one.

To create a new association, users can navigate to the form shown in Figure 27, where they are required to provide essential metadata such as the name, description, postal code, city, and email address of the association. Upon creation, the user is automatically registered as the owner, ensuring immediate administrative access to all related features.

Alternatively, users can join an existing association by entering a unique code provided by the association's owner, as shown in Figure 28. This invitation-only mechanism ensures a first level of access control and protects the integrity of the association’s membership. Once the request is submitted, the association owner retains the right to approve or reject the application, as described in the backend logic.

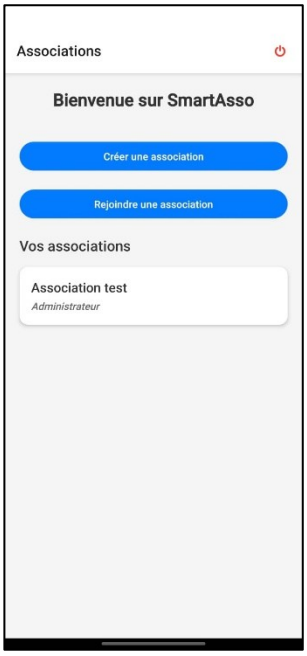


Figure 31 Association Landing Page

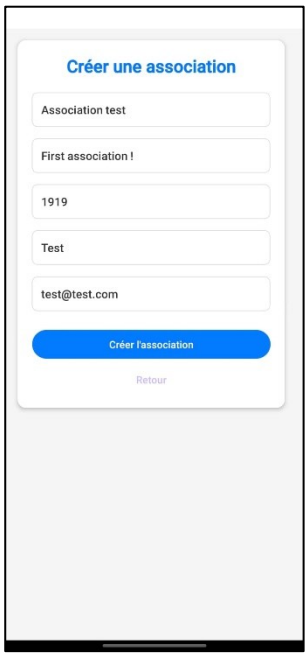


Figure 32 Association Creation Form

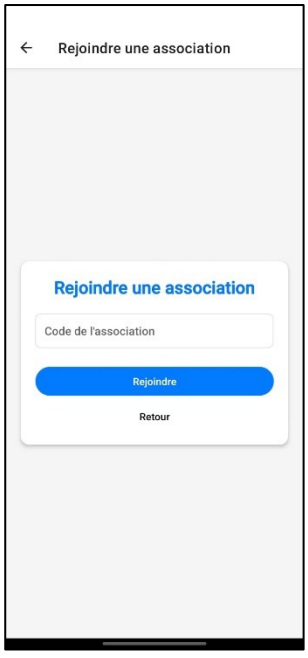


Figure 33 Join Association Form

Once a user submits a request to join an association using the unique code, the owner of the target association receives a notification, as illustrated in Figure 29. This interface allows the owner to view the applicant's information, including their name, email address, and the date of the request.

The owner is then given the option to either approve or reject the request using clearly identifiable buttons. This validation step acts as a second layer of security, ensuring that only authorized users are granted access to the association's internal data and functionalities.

This mechanism supports a controlled and secure onboarding process, enabling association administrators to maintain full oversight of their member base and enforce appropriate access rights before any user can interact with the system's core features.

Figure 34 Incoming Membership Request Interface

The home page of the application, shown in Figure 30, serves as the central dashboard for users after selecting an association. It provides a quick overview of the association's most relevant information, including the association name and description, the number of active members, the current financial balance, the next scheduled event, and a summary of tasks.

While the overall layout remains consistent with the original design prototype, one notable change was made based on user feedback during the prototyping phase. The “Recent Files” card initially planned was replaced by a To-Do List, as requested by members of the pilot association (Appendix 1).

This new component enables users to:

- Add, modify, or delete tasks,
- Assign responsibility for a task to a specific member,
- Mark tasks as completed once done.

Each user can view their personal tasks as well as the completed tasks of the entire group, fostering collaboration and accountability within the association. This change reflects a more operationally focused interface that better aligns with the actual needs of users in a real-world association setting.

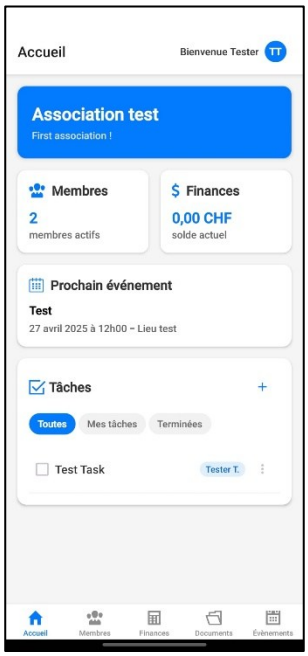


Figure 35 Association Home Screen

The member management interface, presented in Figure 31, provides an overview of all current members of the selected association. The layout closely follows the original mock-up, preserving the top-level statistics that display the total number of members, the number of members who have completed their payments, and those linked to a WhatsApp contact.

Some adjustments have been made to optimize the user experience. For example, email addresses are no longer displayed directly in the list view but can still be accessed by selecting an individual member. This change was introduced to improve the visual clarity and readability of the interface.

The page also features a search bar and a streamlined table layout, offering essential controls for viewing and managing member information. This approach ensures a user-friendly experience by presenting only the most relevant data upfront, while keeping detailed information easily accessible when needed.

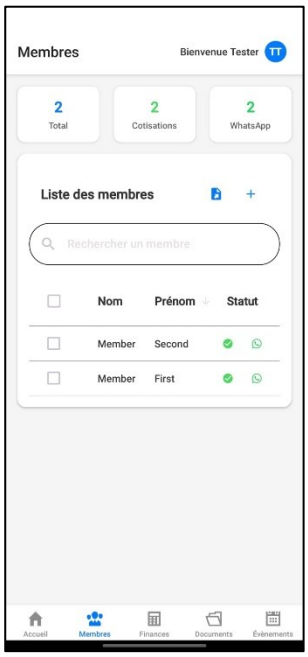


Figure 36 Member Management Page

The frontend implementation of the financial module allows associations to manage their finances in a clear and organized manner. As shown in Figure 32, users can view a summary of profits, losses, and the current balance. A dedicated section also enables manual input of bank and cash balances. Using this data, the system automatically checks whether the accounting is balanced, helping users identify any discrepancies.

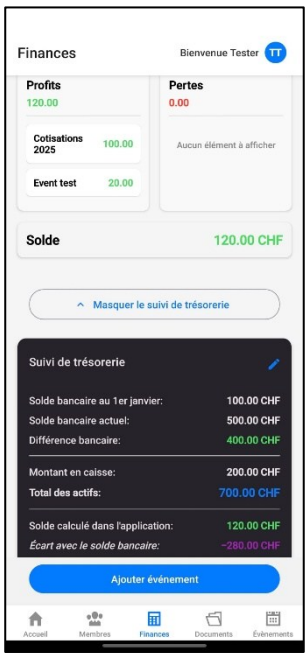


Figure 37 Finance Overview Page

In Figure 33, users can access a breakdown of transactions related to a specific event. This includes revenues and expenses categorized per event, with the ability to add operations dynamically. The total for each event is automatically calculated and reported in the main finance overview, ensuring accurate aggregation of financial data.

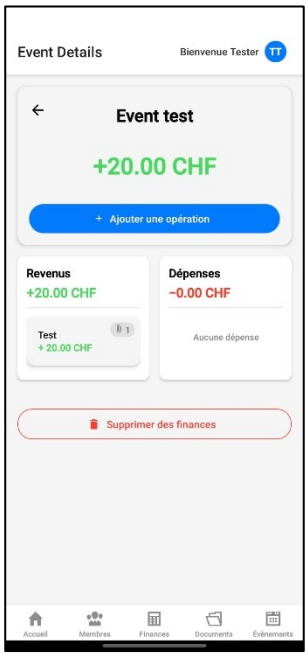


Figure 38 Event Details View

Lastly, Figure 34 shows the detailed information of a financial entry. Each billing record may include a description, amount, and an optional image such as a receipt or invoice. Users can edit or delete these entries, and the inclusion of visual proof supports transparency and accountability.

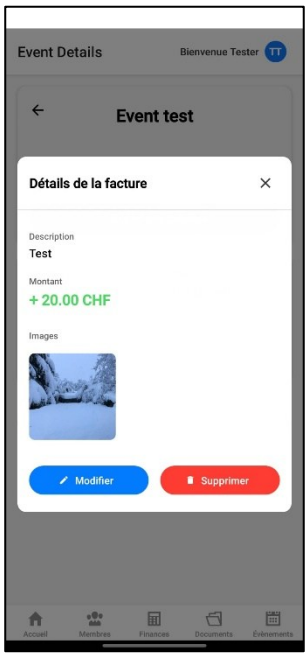


Figure 39 Billing Detail View

The document management feature remains largely consistent with the initial prototype. It provides a straightforward interface where users can organize documents by creating folders linked to specific events or themes, such as general meetings, annual reports, or financial reviews.

As shown in Figure 35, users can create and name folders, which are timestamped upon creation. Within each folder, as presented in Figure 36, users can upload files in various formats, including images, PDFs, Word documents, and spreadsheets. Each file entry displays its name, upload date, and size, along with an option to delete it.

This module enhances collaboration and traceability by offering centralized access to key documents while maintaining a clean and intuitive interface adapted for mobile use.

Figure 40 Folder Management Page

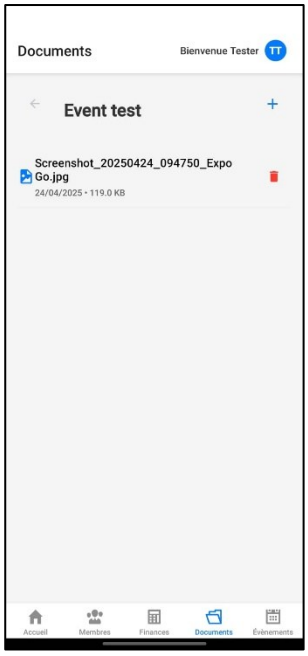


Figure 41 File Listing Page

Following feedback received during the prototype testing phase (Appendix 1), a new feature was added to the event module: the ability to switch between calendar view and list view. As shown in Figure 37, the interface now allows users to visualize scheduled events within a monthly calendar, enhancing the clarity of upcoming activities. The toggle button at the top enables switching to the list format.

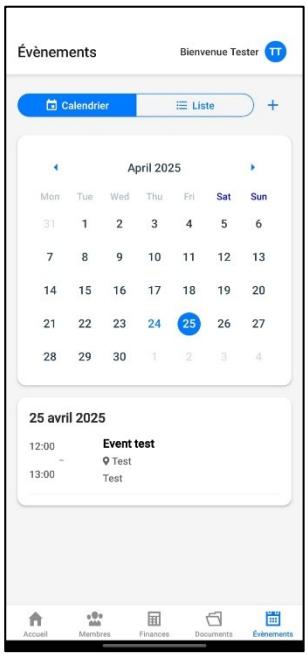


Figure 42 Event Calendar View

In Figure 38, events are displayed in chronological order, offering an alternative mode of consultation better suited for users who prefer a linear overview. This improvement aligns with the needs expressed by users seeking a simplified and accessible way to track participation.

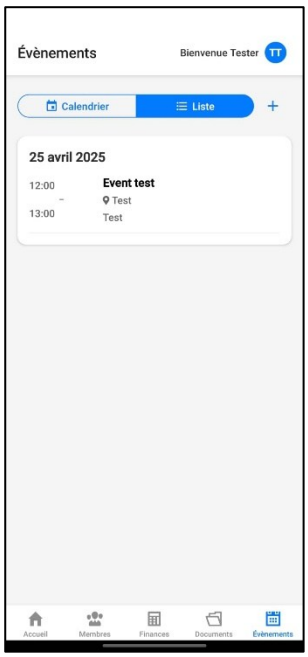


Figure 43 Event List View

Further details are accessible by selecting a specific event, as illustrated in Figure 39. When an event includes a participation fee, the interface allows administrators to track payments and

associate received amounts with specific users. Paid events are automatically integrated into the finance module, ensuring real-time synchronization between event data and financial reporting.

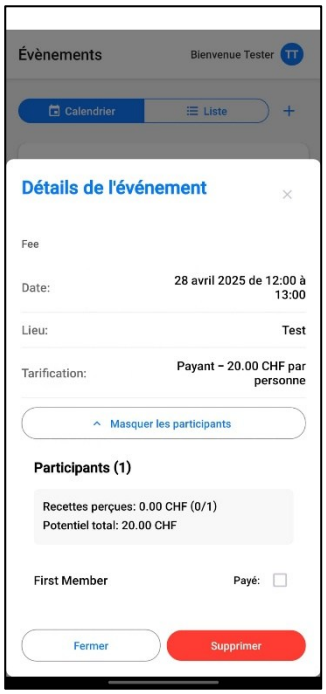


Figure 44 Event Detail View

Among the additional features implemented in the final version of the application is the Association Information Page, shown in Figure 40. This screen provides access to all key metadata related to the selected association, such as name, description, postal code, city, email, and date of creation.

One important element displayed here is the invitation code, which allows users to join the association. This code can be easily copied and shared by administrators with members who wish to request access.

In addition, the page offers an administrative function that enables the current owner to transfer ownership to another member. This feature ensures the continuity of administrative access, for example, in cases where the president of the association steps down and wishes to hand over responsibilities to a successor.

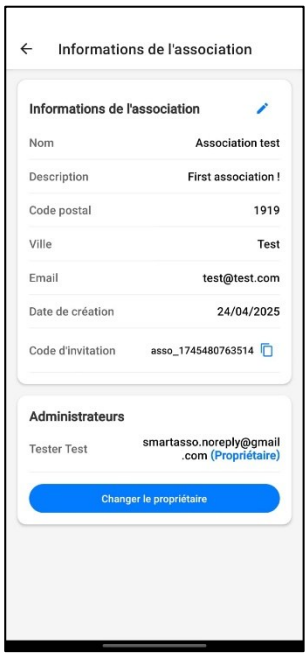


Figure 45 Association Details View

The User Information Page, shown in Figure 41, allows each user to view and manage their personal data. The interface displays basic information such as first name, last name, email address, registration date, and optionally, phone number and physical address if provided. From this page, users can also access the password change function, providing a secure and user-friendly way to update their credentials when needed.

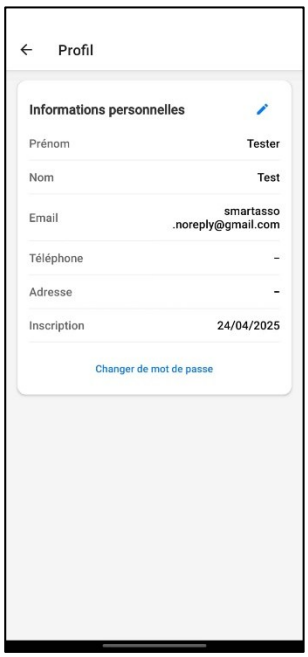


Figure 46 User Profile Page

Phase 4: Deployment to App Store & Play Store

Before the official deployment to the App Store (iOS) and Play Store (Android), a local testing phase was conducted to verify that the application functioned properly in a production-like environment.

The Android version of the application was successfully built using the EAS (Expo Application Services) build system. The application ran correctly on development servers, which allowed for functional tests to be carried out.

Among the testers selected for the evaluation phase, including association members and committee participants, all were iOS users. This made iOS testing particularly relevant. However, generating a preview build for iOS via EAS required enrollment in the Apple Developer Program, which involves an annual cost of 99 USD. Due to limited budget and time constraints within the scope of this academic project, registration for the Apple Developer Program was not feasible.

As a practical alternative, a detailed video demonstration was recorded using an Android device. This video walkthrough presented the core functionalities of the application, navigation structure, and user interface design. It was shared with the testers, who provided qualitative feedback based on visual observation and the accompanying explanations. While this approach did not allow for hands-on experience on iOS, it enabled valuable feedback collection regarding usability, logic flow, and overall design.

At this stage, official deployment to the App Store and Play Store is not planned. The current focus remains on validating the app’s utility and relevance through user feedback. These evaluations will help assess whether the application effectively meets the needs of youth associations and whether further development, including full-scale publishing, would be justified. In future development cycles or potential commercial projects, enrolling in the Apple Developer Program and ensuring publication on both platforms would be essential to ensure broad accessibility.

Phase 5: User Testing & Feedback

Through the demo video, it was possible to collect a wide range of feedback in a short period of time. The test was shared with 13 individuals aged between 18 and 26, representing diverse genders and professional or educational backgrounds. Out of these, seven participants provided detailed responses.

According to the feedback from User 1, the ability to monitor finances without requiring access to the e-banking platform is a significant advantage, especially for associations where financial responsibilities are limited to a few members. This functionality was described as both practical and professional, drawing comparisons to workplace tools like Odoo. Additionally, the app was perceived as easy to use, provided it includes clear explanations alongside its features. (Appendix 3)

User 2 emphasized the practicality of having a centralized platform to track association activities, noting that it offers a more efficient solution than relying on WhatsApp. The user described the design as simple and clear, and found the application intuitive and easy to use even at first glance (Appendix 4).

User 3 pointed out that the application successfully addresses the essential needs of managing a youth association and praised its ease of use and clarity. Suggestions included introducing role-based access, particularly to distinguish general members from committee roles, and the addition of an archive section for storing documents like meeting minutes. The current “Folders” feature already answers this need, confirming its importance. The user also proposed the ability to add an association logo during the creation process and recommended a built-in discussion space to replace external messaging tools. Overall, the app was seen as promising and well-suited for real-world use, with high potential for adoption among youth groups (Appendix 5).

User 4 enthusiastically praised the event management system, describing it as highly practical. They also highlighted the usefulness of the finance-related features, particularly for tracking payments like membership fees, which they found made accounting tasks much easier to handle. This combination of features was seen as a strong point of the application (Appendix 6).User 5 praised the design and found the app easy to use, without offering further suggestions (Appendix 7).

User 6 found the application very impressive and expressed genuine enthusiasm for its features. The design was described as clear, accessible, and easy to use, particularly for recurring tasks within a committee. One standout element was the ability to validate financial transactions with

photos, which made managing documentation much more efficient by centralizing everything in one place.

Feedback included a suggestion to implement restricted-access profiles for volunteers, allowing them to view and register for events without having visibility into more sensitive data. Another improvement proposed was the addition of a search bar to easily locate individuals during payment validation, especially useful in situations involving large groups such as 120 volunteers.

Overall, the user was very positive, highlighting the app’s practicality and expressing eagerness to use it in real scenarios (Appendix 8).

User 7 found the application very practical, particularly for managing busy schedules and overloaded agendas. The user also appreciated the design, describing it as pleasant and visually appealing. (Appendix 9)

4 Project Results and Discussion

4.1 Summary of User Feedback and Usage Data

User feedback highlighted that the application was generally well received for its clarity, simplicity, and relevance to the everyday needs of youth associations. Many testers appreciated how the app brought together core administrative features such as member management, financial tracking, event planning, and document storage in a single, accessible platform. This centralization was seen as a major improvement over using separate tools like WhatsApp or spreadsheets. The ability for committee members to access financial data without relying on e-banking was considered both practical and secure.

Some areas for improvement were also identified. Several users mentioned the need for differentiated access rights, especially to limit what volunteers or regular members can view. There were also suggestions to expand the app's functionality by including features such as volunteer registration for events, a searchable list of users for validating payments more efficiently, and a discussion space to improve internal communication. The option to upload the association's logo during setup was also proposed to strengthen group identity.

Overall, the feedback confirmed that the application addressed a real need and was easy to use. It also provided valuable insights for future updates, which could further improve user experience and expand the app's impact.

4.2 Lessons Learned and Project Challenges

One of the most significant challenges encountered during this project was transitioning from development mode to a stable production environment. While development within Expo Go allowed for rapid testing and iteration, preparing the application for real-world deployment using EAS builds introduced various technical and logistical difficulties. The integration of Firebase services in both web and native environments required a deep understanding of different SDKs and their compatibility with React Native. Debugging environment-specific issues, especially related to authentication and Firestore access, proved time-consuming and complex.

Another major constraint was the inability to test the application directly on iOS devices. Due to the cost of an Apple Developer Program subscription and the limited scope of this academic project, a proper iOS deployment could not be completed. This limited the testing phase to Android builds only, although most test users were iOS users. To overcome this, a detailed walkthrough video of the application was recorded, which enabled feedback collection despite the platform restrictions.

5 Conclusion and Recommendations

5.1 Key Outcomes of the Project

The main outcome of the project was the successful development of a working prototype of a mobile application designed to support the daily administrative tasks of youth associations. The app was implemented using React Native and TypeScript, with Firebase as the backend for real-time data management, authentication, and storage. The application includes essential modules such as member management, event creation, document storage, and finance tracking, features that were directly inspired by the needs of real associations.

Importantly, the app was built to operate entirely free of charge, fulfilling the objective of providing an affordable digital solution for small youth associations with limited budgets. All core functionalities, including data synchronization and user authentication, are powered by free tiers of the chosen technologies, meaning the app can be used without incurring costs.

While the application has not yet been fully deployed to official distribution platforms and is not operating in a production environment, it functions reliably in development mode. All implemented features are operational in this environment, enabling comprehensive user testing in a simulated setting. This setup made it possible to demonstrate and validate most core functionalities effectively.

Feedback collected from test users highlighted a strong appreciation for the app’s clarity, intuitive interface, and the practicality of having all necessary tools in one place. Suggestions from testers also brought attention to possible future enhancements such as role-based permissions, a communication system for members, and better search functionality.

Overall, the project met its initial goals by providing a solid foundation and a development-ready prototype that could be extended into a full product with further refinements and deployment preparation.

5.2 Recommendations for Future Work

One of the key takeaways from this project is the importance of allocating more time specifically for the building and deployment phases. Unexpected changes, such as the update from Expo SDK 52 to 53, can significantly impact dependencies, libraries, and the stability of the project. Allowing more time would make it easier to adapt and resolve issues that may arise late in the development cycle.

In terms of what remains to be done, several important steps are still pending before the app can be considered production ready. First, the deployment process needs to be completed, particularly for iOS, which requires an Apple Developer account. Backend stability and testing in a real production environment must also be ensured. Additionally, several features suggested by users, such as role-based access control, volunteer-specific views, an integrated chat or discussion space, and a search function in user lists, should be planned for future iterations.

Sources

Amatya, S. 2013. Cross-platform mobile development: An alternative to native mobile development. Master's thesis. Linnaeus University Sweden. Degree Project. Department of Computer Science. URL: https://www.diva-portal.org/smash/get/diva2:664680/FULLTEXT01.pdf. Accessed: 17 February 2025.

Apple Developer Program 2025. Terms and payment. URL: https://developer.apple.com/programs/enroll/. Accessed: 8 May 2025.

Article 61a and 69(1) of the Swiss Civil Code (CC, SR 210)

Article 6 of the Swiss Federal Act on Data Protection (FADP)

Article 90a of The Ordinance on the Commercial Register (ORC)

Carroll, J. M. 2009. Human computer interaction (HCI). Encyclopedia of Human-Computer Interaction. Aarhus: The Interaction Design Foundation. URL: https://snoopedu.com/app/uploads/2022/03/Reading1_HCI.pdf. Accessed: 5 February 2025.

Dix, A. 2016. "Human–computer interaction, foundations and new paradigms," Journal of Visual Languages & Computing, 42, pp. 122–134. doi:10.1016/j.jvlc.2016.04.001. URL: https://linkinghub.elsevier.com/retrieve/pii/S1045926X16300088. Accessed: 7 February 2025.

Federal Data Protection and Information Commissioner (FDPIC). 2024. Data protection in clubs and associations. URL: https://www.edoeb.admin.ch/en/data-protection-in-clubs-and-associations. Accessed: 2 February 2025.

Gulliksen, J., Göransson, B., Boivie, I., Blomkvist, S., Persson, J., & Cajander, Å. 2003. Key principles for user-centred systems design. Behaviour and Information Technology, 22(6), 397-409. URL: https://www.tandfonline.com/doi/epdf/10.1080/01449290310001624329?needAccess=true. Accessed: 9 February 2025.

Islam, S. 2023. JavaScript alternative (TypeScript) and its effectiveness in web development. Bachelor's thesis. Tampere University of Applied Sciences, Degree Programme in Software Engineering.
URL: https://www.theseus.fi/bitstream/handle/10024/795522/Islam_Saiful.pdf?sequence=2&isAllowed=y. Accessed: 4 February 2025.

Jansen, R.H., Vane, V. & Gabe de Wolff, I. 2016. TypeScript: Modern JavaScript Development. In Part 1 (Module 1). Introducing TypeScript, Chapter 1. Packt. Birmingham, Mumbai. E-book. Accessed: 4 February 2025.

Latif, M., Lakhrissi, Y., Nfaoui, E.H. & Es-Sbai, N. 2017. Review of mobile cross platform and research orientations. IEEE. Fez. URL: https://ieeexplore.ieee.org/document/7934674. Accessed: 1 February 2025.

Many, B. 2007. Jeunesse d'aujourd'hui et organisations de jeunesse de demain. De Boeck Supérieur Section: Sociologie. URL: https://shs.cairn.info/revue-pensee-plurielle-2007-1-page-9. Accessed: 2 February 2025.

Nielsen, J. 1993. Usability engineering. Academic Press. Boston. URL: https://dl.acm.org/doi/pdf/10.5555/2821575. Accessed: 7 February 2025.

Nath, A., & Mukherjee, S. 2015. Impact of Mobile Phone/Smartphone: A pilot study on positive and negative effects. International Journal, 3(5), 294-302. URL: https://d1wqtxts1xzle7.cloudfront.net/45348395/Impact_of_Mobile_Phone_Smartphone-libre.PDF. Accessed: 9 February 2025.

Nikita, N. A., Azim, K. S., Jafor, A. H. M., Shayed, A. U., Hossain, M. A., & Khan, O. U. 2024. Digital Transformation in Non-Profit Organizations: Strategies, Challenges, and Successes. AIJMR-Advanced International Journal of Multidisciplinary Research, 2(5). URL: https://www.aijmr.com/research-paper.php?id=1097. Accessed: 27 January 2025.

Nilsson, A. 2022. Performance and feature support of Progressive Web Applications: A performance and available feature comparison between Progressive Web Applications, React Native applications and native iOS applications. URL: https://www.diva-portal.org/smash/get/diva2:1653815/FULLTEXT01.pdf. Accessed: 14 March 2025.

Norman, D. 2013. The design of everyday things: Revised and expanded edition. Basic books. New York. URL: https://dl.icdst.org/pdfs/files4/4bb8d08a9b309df7d86e62ec4056ceef.pdf. Accessed: 7 February 2025.

Pinchot, J. 2020. USER EXPERIENCE (UX) DESIGN CONCEPTS FOR MOBILE APP DEVELOPMENT COURSES. Robert Morris University. URL: https://iacis.org/iis/2020/4_iis_2020_202-211.pdf. Accessed: 5 February 2025.

Punchoojit, L. and Hongwarittorrn, N. 2017. "Usability Studies on Mobile User Interface Design Patterns: A Systematic Literature Review," Advances in Human-Computer Interaction, 2017, pp. 1–

22. doi:10.1155/2017/6787504. URL: https://onlinelibrary.wiley.com/doi/epdf/10.1155/2017/6787504. Accessed: 07 February 2025.

React Native Documentation 2025. Platform-Specific Code. URL: https://reactnative.dev/docs/platform-specific-code. Accessed: 14 March 2025.

Tarif, Z. 11 October 2024. Top Benefits of Using Firebase for App Development. back4app blog. URL: https://blog.back4app.com/advantages-of-firebase/. Accessed: 10 February 2025.

Turk, M. 2013. “Multimodal interaction: A review,” Pattern Recognition Letters, 36, pp. 189–195. doi:10.1016/j.patrec.2013.07.003. URL: https://dl.acm.org/doi/pdf/10.1145/3308561.3354632. Accessed: 7 February 2025.

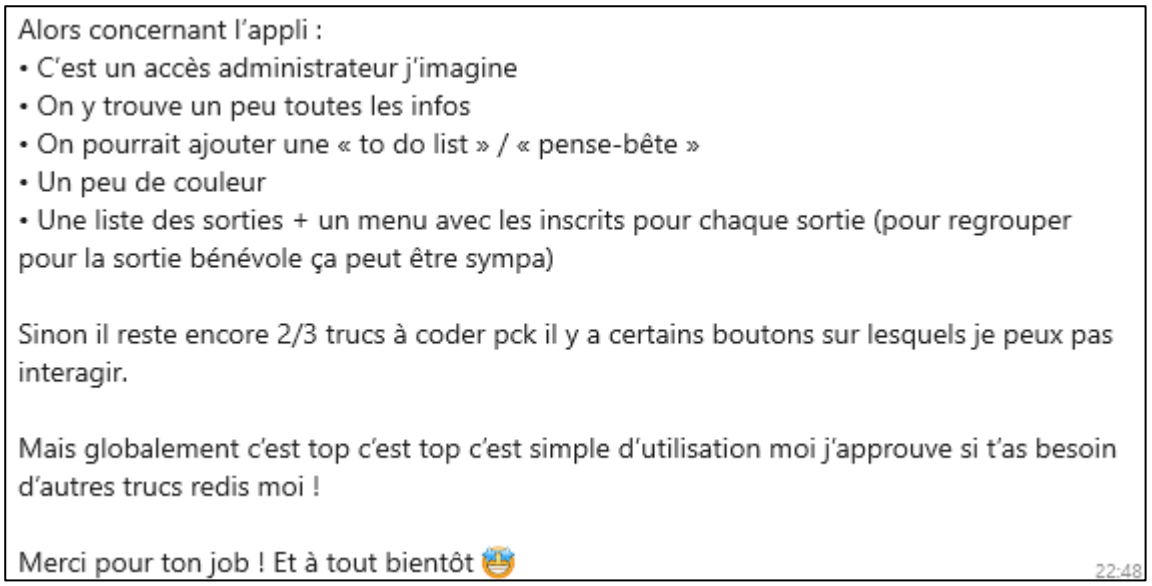
Uniyal, S. P., Joshi, K., Singh, V. K., Aggarwal, A., Chhabra, G., & Kumar, A. 2023, August. Comparative Analysis of App Size Variations between React Native and Apache Cordova Powered Android Applications. In 2023 Second International Conference on Augmented Intelligence and Sustainable Systems (ICAISS) (pp. 1697-1702). IEEE. URL: https://ieeexplore.ieee.org/document/10250551/. Accessed: 17 February 2025.

Wenhao, W. 2018. React Native vs Flutter, cross-platform mobile application frameworks. Bachelor's thesis. Metropolia University of Applied Sciences, Degree Programme in Engineering Information technology. URL: https://www.theseus.fi/bitstream/handle/10024/146232/thesis.pdf. Accessed: 3 February 2025.

Wynne, B.G. 2011. Managing knowledge in small non-profit organizations: A toolkit and research report. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=914845955db915bd270f909bd428e66200eb032c. Accessed: 3 February 2025.

Appendices

Appendix 1. User Feedback on Prototype



Alors concernant l'appli :
• C'est un accès administrateur j'imagine
• On y trouve un peu toutes les infos
• On pourrait ajouter une « to do list » / « pense-bête »
• Un peu de couleur
• Une liste des sorties + un menu avec les inscrits pour chaque sortie (pour regrouper pour la sortie bénévole ça peut être sympa)

Sinon il reste encore 2/3 trucs à coder pck il y a certains boutons sur lesquels je peux pas interagir.

Mais globalement c'est top c'est top c'est simple d'utilisation moi j'approuve si t'as besoin d'autres trucs redis moi !

Merci pour ton job ! Et à tout bientôt

22:48

English translation :
So about the app:
- It's an administrator access I guess
- You can find all the information you need
- We could add a "to-do list" / "reminder".
- A bit of colour
- A list of outings + a menu with registrations for each outing (to group together for the volunteer outing could be nice).

Otherwise, there are still 2/3 things to code, because there are some buttons I can't interact with.

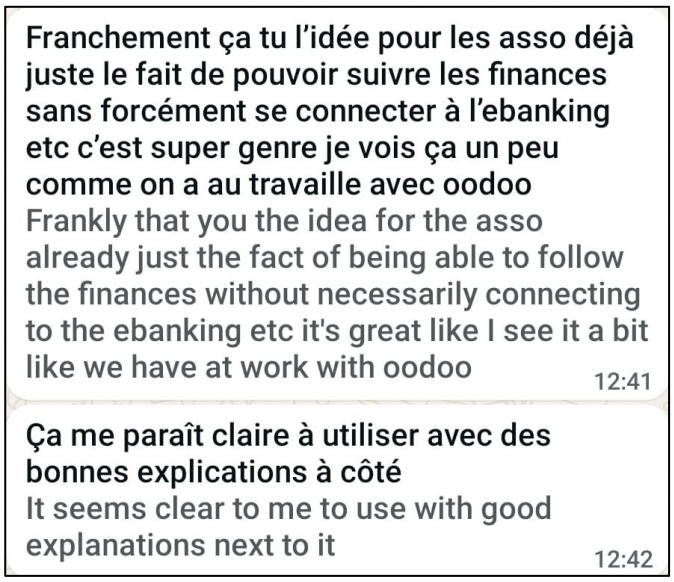
But overall, it's great, it's great, it's easy to use, and I'm all for it. If you need anything else, let me know!

Thanks for your work! See you soon!

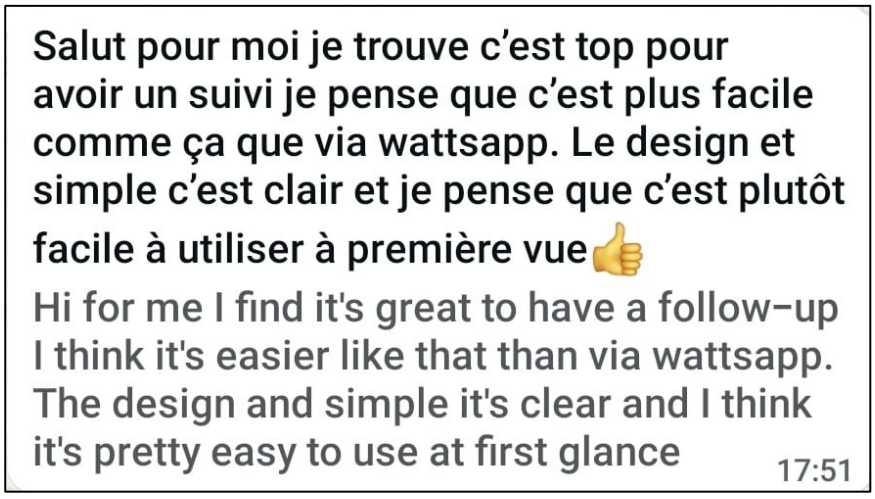
Appendix 2. Firestore Rules for Access Control

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    function isAuthenticated() {
      return request.auth != null;
    }

    function isCreator(associationId) {
      let association = get(/databases/$(database)/documents/associations/$(associationId));
      return association != null && association.data.creatorId == request.auth.uid;
    }

    function isMember(associationId) {
      return exists(/databases/$(database)/documents/associations/$(associationId)
      /appMembers/$(request.auth.uid));
    }

    match /users/{userId} {
      allow create: if isAuthenticated();
      allow read, update: if isAuthenticated() && request.auth.uid == userId;
    }

    match /associations/{associationId} {
      allow create: if isAuthenticated();
      allow read: if isAuthenticated();

      allow update: if isAuthenticated() && (isCreator(associationId) || isMember(associationId));

      match /appMembers/{memberId} {
        allow read: if isAuthenticated();
        allow write: if isAuthenticated() && (isCreator(associationId) || isMember(associationId));
      }

      match /members/{memberId} {
        allow read, write: if isAuthenticated() && (isCreator(associationId) || isMember(associationId));
      }

      match /pendingRequests/{requestId} {
        allow create: if isAuthenticated();
        allow read, delete: if isAuthenticated() && (isCreator(associationId) || isMember(associationId));
      }

      match /finances/{eventId} {
        allow read, write: if isAuthenticated() && (isCreator(associationId) || isMember(associationId));

        match /invoices/{invoiceId} {
          allow read, write: if isAuthenticated() && (isCreator(associationId) || isMember(associationId))
        }
      }

      match /{collection}/{docId} {
        allow read, write: if isAuthenticated() && (isCreator(associationId) || isMember(associationId));

        match /{subcollection}/{subdocId} {
          allow read, write: if isAuthenticated() && (isCreator(associationId) || isMember(associationId))
        }
      }
    }
  }
}
```

Appendix 3. User 1 Feedback on Final App



Appendix 4. User 2 Feedback on Final App



Appendix 5. User 3 Feedback on Final App

Alors j'ai Check ton tuto à l'instant donc mes retours :

• On voit que ça fait quelques années que t'es bien investi dans la Jeunesse pck pour moi on retrouve vraiment tous les trucs à gérer
• Ça m'a l'air simple, efficace et facile à prendre en main
• Est-ce que t'as un système hiérarchique en mode le caissier prime sur les finances, président sur l'organisation etc avec aussi des comptes « visiteurs » avec les infos essentielles pour les membres de la Jeunesse ?
• Il manquerait une zone « archives » pour notamment déposer les PV des séances (ainsi que scanner les documents reçus autres que des factures)
• Est-ce qu'il est possible lors de la création d'une association d'ajouter un Logo qui ensuite serait un peu intégré au Design ? Ça rendrait l'application un peu « unique » pour chaque utilisateur sans avoir à faire de la programmation pour chacun
• Peut-être une espace « discussions » qui remplacerait un groupe whatsapp dans lequel on pourrait faire un canal « Jeunesse - Infos », « Jeunesse ouvert à tous » et « Jeunesse - comité » ?

Voilà c'est un peu tout mais je trouve ça vraiment top, le jour où tu la publies on peut tester ça en vrai et après je pense qu'elle risque d'être utilisée par pas mal de Jeunesses ou autres société moi je trouve que c'est bcps trop bien comme appli !

Courage pour la fin du prijrt et merci pour le job ! 12:05

English translation:

I just checked out your tutorial and here are my thoughts:

You can clearly see that you've been involved in the Jeunesse for a few years now because everything that needs to be managed is really well covered.

It looks simple, effective, and easy to use.

Do you have a hierarchical system, like the treasurer having priority over financial aspects, the president managing organization, etc., along with "visitor" accounts containing essential information for Jeunesse members?

There's a missing "Archives" section, where you could upload meeting minutes (PVs) and scan documents other than just invoices.

Is it possible to include a feature during association creation to add a logo that would then be integrated into the design? That would make the app feel a bit more "unique" for each user, without them needing to do programming.

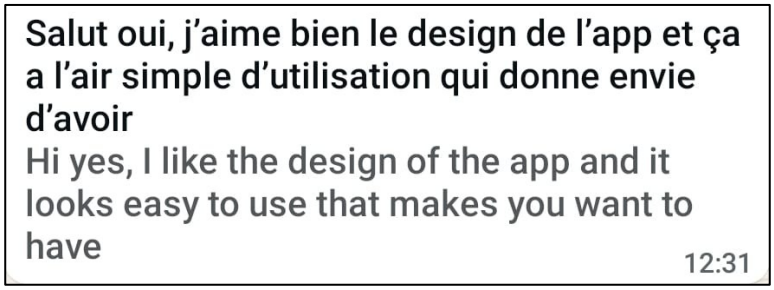
Maybe consider a "Discussions" space that could replace a WhatsApp group, with channels like "Jeunesse - Info," "Jeunesse - Open to All," and "Jeunesse - Committee"?

That's about it, but honestly, I think it's really great. The day you release it, we should definitely test it live, and I really think it has potential to be used by many Jeunesse groups or other societies. Personally, I think it's a very, very good app!

Good luck with the end of your project and thanks for the great work!

Appendix 6. User 4 Feedback on Final App

Appendix 7. User 5 Feedback on Final App



Appendix 8. User 6 Feedback on Final App

C’est vraiment trop cool ce que tu as fait, j’ai adoré ! Concernant tes questions sur le design, je trouve l’interface assez claire, facile d’accès et simple à utiliser. Les fonctionnalités sont bien pensées, surtout celles qu’on utilise souvent en comité.

J’ai trouvé ça très pratique de pouvoir valider les finances avec des photos : tout est centralisé au même endroit, on n’a pas besoin de chercher dans 15 000 documents. C’est super efficace d’avoir tout sur une seule application.

Une idée que j’ai eue : ce serait peut-être intéressant de créer un accès dédié pour les bénévoles, avec des droits limités. Par exemple, ils pourraient voir uniquement les événements et s’y inscrire, sans avoir accès aux autres données. Je sais que c’est beaucoup de travail, donc ce serait peut-être pour plus tard, mais je trouve que ce serait une super évolution.

En regardant la vidéo, j’ai aussi pensé à quelque chose : quand tu valides les paiements des bénévoles pour un événement, ce serait vraiment utile d’avoir une barre de recherche pour trouver quelqu’un par son nom. Parfois, on a jusqu’à 120 bénévoles à gérer, donc pouvoir les chercher rapidement rendrait l’utilisation beaucoup plus pratique.

En tout cas, j’ai vraiment hâte d’utiliser l’appli, elle est super bien faite et tu as fait un super boulot. Bravo !

English translation:
What you’ve done is really impressive, I loved it! Regarding your questions about the design, I found the interface clear, accessible, and easy to use. The features are well thought out, especially the ones we use daily as a committee.

I found it very practical to validate financial transactions with photos, everything is centralized in one place, and we don’t have to dig through tons of documents. It’s super-efficient to have it all in one app.

One idea that came to mind: it might be useful to have a dedicated access level for volunteers, with limited permissions. For example, they could only see and register for events, without access to other sensitive data. I know that’s a lot of extra work, so maybe that’s something to consider for later, but it would be a great feature.

While watching the video, I also thought it would be really helpful to add a search bar when validating who paid for an event. Sometimes we have up to 120 volunteers to manage, and being able to search by name would make things much easier and more practical.

In any case, I’m really looking forward to using the app. It looks great and you’ve done an amazing job. Congratulations!

Appendix 9. User 7 Feedback on Final App