MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Implementation of OSINT functionality and feature extension for a contact management application

Bachelor's Thesis

ROBIN CHMELÍK

Brno, Spring 2024

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Implementation of OSINT functionality and feature extension for a contact management application

Bachelor's Thesis

ROBIN CHMELÍK

Advisor: Ing. Lukáš Grolig

Department of Computer Systems and Communications

Brno, Spring 2024


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby, I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during the elaboration of this work are properly cited and listed in complete reference to the due source. During the preparation of this thesis, I used several AI tools to enhance my writing and speed up the development. These tools are Grammarly for improved writing style, Github Copilot for faster programming, ChatGPT 4 for a better structure of the thesis, and Midjourney for the generation of images used in the application. I declare that I used these tools in accordance with the principles of academic integrity. I checked the content and took full responsibility for it.

Robin Chmelík

Advisor: Ing. Lukáš Grolig

Acknowledgements

I would like to thank my advisor, Ing. Lukáš Grolig, for his continuous support and guidance regarding this thesis. Furthermore, I want to say my thanks to the Technology Transfer Office of Masaryk University and the South Moravian Innovation Centre for mentoring and funding this project.

Abstract

Hybrid mobile applications created using web technologies typically struggle to replicate a native-like user experience. This struggle is mostly caused by limited access to native device features and not following platform-specific user interface guidelines. This thesis performs a comprehensive rewrite of a hybrid mobile application for personal relationship management. It aims to improve the native user experience with the mobile UI framework called Ionic and to implement new features like open-source intelligence (OSINT) capabilities using the Google Programmable Search Engine. The first part of the thesis also explores different types of mobile applications and the GDPR in the context of contact management solutions. The overhauled application offers a significantly improved native user experience and more tools for advanced contact analysis. The final product was released and made publically available on App Store and Google Play.

Keywords

Capacitor, Ionic, Firebase, PRM, GDPR, Nuxt.js, OSINT, Vue.js, hybrid application

Contents

Introduction

As the world becomes more interconnected with the advance of social networks and the internet in general, so does the complexity and the scope of our social circles. This advancement eventually led to the popularization of contact management applications, which strive to organize and aggregate information about given contacts to help users to keep up with people around them. These applications became very popular, especially in the form of Personal or Customer Relationship Managers (PRMs and CRMs).

This bachelor thesis discusses a complete rewrite of a PRM mobile application called Attachi. It also implements several new features, such as Open Source Intelligence (OSINT) capabilities for searching for new information about contacts, auto-grouping of contacts based on similarities, or design changes for a more native user experience.

This work starts by introducing different types of mobile applications that can currently be found on distribution platforms such as Google Play or App Store. This introduction focuses primarily on hybrid applications and challenges commonly faced during the development of mobile applications with web technologies. Hybrid applications are explored more in-depth since this project uses hybrid technologies, and it is necessary to first understand the underlying principles of this approach before delving into more specific topics. The thesis then continues with ethical and regulatory considerations made during the development due to the highly personal nature of data manipulated by this application. Since the primary purpose of personal relationship managers is to manage private information about individuals, this part focuses on GDPR compliance and employed mechanisms to ensure user privacy. The next part of this work then continues to analyze the initial state of the application, identify its flaws, and propose several goals for this rewrite. Before delving into particular challenges faced during the development, this thesis introduces the used technologies and the system architecture of the whole solution to get a better grasp of the employed environment. The last part of this work explores several notable changes and additions to the system made during this rewrite, followed by the introduction of utilized testing and deployment strategies.

The outcome of this thesis is a hybrid mobile PRM application publicly available on the App Store and Google Play. The application now provides OSINT functionalities and a more native user experience thanks to the integration of Ionic framework. The frontend of the application was rewritten entirely to Vue3 and converted to use TypeScript instead of JavaScript. Finally, several new features were added, like an automatic smart grouping of contacts, recovery codes, passcodes, or in-app notifications.

1 Types of mobile applications

Since the release of iPhone in 2007, the popularity of mobile applications has been increasing rapidly, recently reaching more than four million available applications on Google Play and App Store [1].

This growth was largely made possible by the emergence of cross-platform and hybrid applications as an alternative to traditional native applications. These alternatives can run on multiple operating systems, like Android or iOS, using a single codebase, which makes them much cheaper and faster to develop [2]. The following chapter will describe different types of mobile applications, focusing on the hybrid ones since this application uses for hybrid technologies.

1.1 Native mobile applications

Native applications are designed and developed for a specific operating system like Android and iOS, allowing them to fully leverage the hardware capabilities of the smartphone. They use programming languages natively supported by the device's operating system, such as Objective-C and Swift for Apple devices or Java and Kotlin for Android devices.

This platform-specific design and direct access to hardware features without additional levels of abstraction allow highly optimized performance, enabling a significantly smoother and more responsive user experience than web-based applications. These advantages, however, come at a notable cost, as the development of native applications is usually longer and more expensive than the development of hybrid or cross-platform applications [2].

This increase in required resources is mainly caused by the need to develop and maintain different codebases for each platform, which requires more funding and usually more developers specialized for the given platform. For this reason, native applications are typically developed by larger companies with more teams and resources.

Some well-known products that were developed as native applications are, for example, Netflix, Duolingo, and Uber.

1.2 Cross-platform mobile applications

Cross-platform mobile applications are similar to native ones but with one major difference. As the name suggests, these applications share a single codebase across all platforms. A single codebase significantly reduces the development cost and makes shipping of new features faster [2] as they do not have to be programmed multiple times for each platform.

Utilizing programming languages that are not native to targeted operating systems is another distinction from native applications. Cross-platform applications use frameworks that allow developers to use other languages that are then compiled into a native code specific to each platform. A well-known example of such frameworks can be Flutter, which uses a programming language called Dart; .NET MAUI, which uses C#; or React Native, utilizing JavaScript.

The performance of cross-platform applications is usually close to native ones; however, in high-demand scenarios, they might struggle with resource management and require a significant amount of memory [3]. This might be caused by many factors, such as the need to communicate over the bridge between JavaScript and the native layer in React Native or the need to render UI[1] elements with a custom render engine in Flutter.

Some famous examples of cross-platform mobile applications are Facebook, Discord, and Skype [4].

1.3 Hybrid mobile applications

The last type of mobile applications found in app stores are so-called hybrid applications. Hybrid mobile applications combine native and web applications by wrapping a webview into a native container or shell. Since the webview is just a built-in browser window without the browser UI, it is possible to render any website as a standalone mobile application. This makes it possible to use web technologies such as Vue.js or Angular for mobile application development, which would otherwise not be feasible.

1. User Interface

Hybrid mobile applications employ several layers of abstraction; the only native component they typically use is the webview displaying the web application. This abstraction negatively affects their performance, most notably the startup time, which takes around two seconds on average, showing the difference between native applications with an average startup time of 200ms [5].

Furthermore, these applications usually have limited access to native functionalities, making them unsuitable for projects that require tightly integrated hardware and native features.

These downsides are, however, well compensated by significantly reduced complexity, as any existing web application can be converted to a mobile one without the need to create a new codebase. This allows to cover more channels and reach a wider audience with a single solution while keeping a consistent brand identity. An exemplary showcase of such omnichannel consistency achieved with the hybrid application can be demonstrated with Burger King, which reused web UI components for Android and iOS applications [6].

Among native and cross-platform applications, hybrid applications are the cheapest and fastest to develop, making them a suitable choice for projects with little resources and limited time for development.

1.3.1 Native user experience of hybrid applications

Hybrid mobile applications usually struggle to replicate native-like user experience since all UI components are rendered as HTML[2] elements in a browser [7]. Furthermore, native applications traditionally have non-linear routing and rich page transitions, which is not typical for web applications. For this reason, hybrid solutions mostly employ Single Page Application (SPA) bundles that do not require reload on routing.

Some application stores even have guidelines that prevent submitting hybrid applications that do not have a certain level of native user experience, as they might be regular websites only wrapped in a native runtime. An excellent example is an App Store guideline 4.2 for Minimum Functionality, which states that '*Your app should include features, content, and UI that elevate it beyond a repackaged website. If your*

2. HyperText Markup Language

app is not particularly useful, unique, or "app-like," it doesn't belong on the App Store' [8].

There are many web-based UI frameworks like Ionic, Framework7, or Onsen UI that try to combat this lack of native user experience connected to hybrid applications. These frameworks automatically recognize the current platform and try to mimic the native UI components of this platform with HTML and CSS[3]. These web-based replicas are often almost unrecognizable from the original.

1.3.2 Access to native APIs with hybrid applications

Hybrid applications resolve the earlier-mentioned limited access to native functionalities with so-called native bridges. Native bridges are a core mechanism of hybrid applications that facilitates communication between the website running in the webview and the native layer of the device. The bridge acts as a mediator between these two layers, enabling the web applications to use all native device features, like haptic feedback, geolocation, or access to the camera and the file system [9] using so-called native bridge bindings.

Native bridge bindings can be configured manually to fit virtually any specific need of the developer. The creation of custom bridge bindings would, however, remove the main benefit of hybrid applications - a single codebase. This is due to the fact that bridge bindings must be configured not only on the web layer but also on the native layer, which requires another project in Android Studio or Xcode.

Hybrid applications solve this problem using so-called cross-platform native runtimes such as Capacitor.js, Apache Cordova, or Tauri Mobile, which provide many ready-to-use pre-configured native bridge bindings so developers do not have to configure everything manually and can keep the benefit of a single codebase. These cross-platform native runtimes provide JavaScript libraries that allow invoking common native functionalities directly from the web layer. Subsequently, these libraries can automatically generate the necessary Android and iOS project files with prepared bridge bindings and a pre-configured webview for displaying the SPA bundle.

3. Cascading Style Sheets

If needed, this process allows to further manually extend and modify the generated project files by opening them in Android Studio or Xcode. This makes it possible to access native tools and features, such as IDE[4]-specific debugging capabilities, that might be needed to test the performance of the application or to verify the implementation of custom bridge bindings. These project files are also necessary for subsequent build configuration to prepare the application for distribution.

By automating the generation of platform-specific project files, the native runtime frameworks significantly streamline the development processes by using a single codebase for all platforms. The hybrid model, therefore, combines the speed of web development with access to native features.

1.4 Web native applications

Since the end of 2020, the Ionic framework has promoted a new name for a sub-category of hybrid applications. This new sub-category is called **web native**, an umbrella term for modern hybrid applications that take full advantage of native bridge bindings and mobile UI frameworks, making them very similar to native and cross-platform applications [10].

This new name is being promoted to escape several bad connotations connected to hybrid applications made with PhoneGap and Apache Cordova, as these frameworks are often associated with poor performance and bad native user experience. However, since these technologies are more than 15 years old now, the technological landscape has significantly changed. Nowadays, modern hybrid applications using new technologies like Capacitor.js and Ionic framework can be compared to native applications as the difference is often hard to recognize. Therefore, to distinguish themselves, modern hybrid mobile applications sometimes call themselves web native applications.

4. Integrated development environment

1.5 Progressive web applications

The last type of applications found in several app stores are Progressive Web Applications or PWAs. PWAs are very similar to traditional web applications as they run only in a browser without any native wrapper. They offer several advantages over web applications, such as the ability to function offline or the ability to access native device features like biometric authentication, haptic feedback, or geolocation [11]. These additional capabilities are made possible by taking advantage of service workers, web manifests, and web APIs[5].

Service workers run on a separate thread from the main browser thread and act as proxy middleware between the requested resources and the application itself. By intercepting requests, service workers can cache the request responses and subsequently use this cache to serve the application when no internet connection is available. Furthermore, service workers also provide support for background activities, such as listening for push notifications.

The web manifest file, on the other hand, provides a configuration to determine how to display the PWA. This file includes information like the name of the application, a link to the splash screen and icons that will be used upon saving the PWA on the device, or a preferred screen orientation of the application.

Finally, web APIs, or browser APIs provide access to many native device capabilities, such as haptic feedback, geolocation, screen capture or Bluetooth [12]. Web APIs are similar to native bridge bindings used in hybrid applications. However, unlike more consistent and predictable native bridge binding, some web APIs might not be supported on all devices and browsers [13], which currently makes them slightly unreliable.

Similarly to hybrid applications, PWAs suffer from poor native user experience and restricted access to some native features compared to cross-platform and native applications. Since PWAs are not wrapped in any native runtime, they are closer to web applications than to mobile ones, making their level of native immersion even lower than that of hybrid applications.

5. Application Programming Interface

For this reason, some distribution platforms, such as the App Store, do not allow PWAs at all [14, 8]. Currently, the only way to publish PWA on the App Store is to wrap it in a native runtime using a tool like Capacitor or PWABuilder, essentially making it a hybrid application. On the other hand, Google Play allows and even promotes the usage of PWAs on their platform. Google provides tools like Bubblewrap, which turns the PWA into TWA, or Trusted Web Activity, which can then be published to Google Play [15]. TWA extends and builds on top of so-called Chrome Custom Tabs that are functionally very similar to webviews used by hybrid applications. These technologies allow PWAs to be easily published on distribution platforms like Google Play, Microsoft Store, or Samsung Galaxy Store.

PWAs are generally more suitable for extending the capabilities of web applications rather than for the development of standalone mobile applications. However, due to their simplicity and popularity, they are a viable option for developing mobile applications that target only Android devices and do not heavily rely on native features.

2 GDPR and Personal Relationship Managers

Since the primary goal of personal relationship managers is to manage personal information, it is necessary to carefully consider data privacy regulations. Specifically, because this application operates in the European Union, it falls under the General Data Protection Regulation or GDPR. Not following the guidelines set by this regulation could result in many legal repercussions and suspension of the application from distribution platforms. Since this topic was influential in the overall design of the application, the following chapter will discuss aspects of GDPR in the context of personal relationship managers.

2.1 What is GDPR

General Data Protection Regulation, or GDPR, is currently one of the most comprehensive data privacy laws in the world. It came into effect on 25. May 2018 and aims to govern how personal data of individuals within the European Union (EU) and the European Economic Area (EEA) are collected, processed, and protected. These regulatory rules help to enhance the privacy rights of individuals and impose stricter rules on all entities that are somehow manipulating personal data.

Furthermore, GDPR applies not only to entities within the EU but also to those outside the region if they operate with any personal data of individuals located in the EU or EEA. This means that if, for example, US companies want to operate within the EU, they must comply with the GDPR. The cross-border scope of the GDPR significantly extends its reach, as many organizations cannot afford to exit the European market, which is one of the largest economies in the world [16, 17].

2.2 Key GDPR definitions

Since GDPR uses several key definitions to better classify regulatory requirements, it is necessary to first define and understand these terms as they will be used throughout this chapter. Each definition will also contain an explanation in the context of this application.

2.2.1 Personal data

In GDPR, personal data refers to any information that can be used to identify an individual, either directly or indirectly. This means that any information such as name, email address, location, or physical appearance is considered to be personal data.

In the context of this application, the definition of personal data is complicated since the users mostly manage the personal data of their contacts. However, since PRM applications are typically meant for purely personal purposes, this activates so-called Household Exemption under certain conditions, which partially solves this issue. More about this exemption in chapter 2.5.1.

2.2.2 Data subject

GDPR refers to any identifiable natural person as a data subject. Data subjects can be identified by collecting and processing their personal data. One of the main goals of GDPR is to give data subjects greater control over their personal data.

In the context of this application, the data subjects would be users who create an account and provide their personal data, such as a name and email address.

2.2.3 Data controller

A data controller is any entity that determines how and why the collected personal data will be processed. The data controller is typically an organization, public authority, or any other body that provides services that require personal data.

In the context of this application, the data controller would be the author of the thesis who decides the purpose and means of processing provided personal data.

2.2.4 Data processor

GDPR refers to any entity that processes personal data on behalf of the data controller as a data processor. GDPR defines processing as any operation over collected personal data such as structuring, storing,

altering, or erasing. Data processors are typically third-party services the data controllers use to further process collected data.

In the context of this application, there are several data processors that are somehow processing provided user data. Some notable data processors used by this application are, for example, Firebase for backend services or Proxycurl for scraping LinkedIn.

2.3 Core principles of GDPR

At its core, GDPR is structured around seven fundamental principles that together create a comprehensive framework for enhanced privacy.

The first principle, ***Lawfulness, fairness, and transparency***, ensures that individuals are informed about how their personal data will be used.

The second principle, ***Purpose limitation***, restricts the processing and collecting of personal data only to explicitly stated legitimate purposes. Any additional collection and processing for any purpose, other than the original requires new consent.

The third principle, ***Data minimization***, allows only the collection of personal data necessary for the given task to prevent the collection of extra information that is not actually needed for the agreed purpose.

The fourth principle, ***Accuracy***, dictates that all personal data should be kept up-to-date. Furthermore, it requires data processors to provide means to erase and modify all inaccurate personal data to avoid any possible risk to the given individual.

The fifth principle, ***Storage limitation***, states that all collected personal data should not be stored longer than necessary for their agreed purpose.

The sixth principle, ***Integrity and confidentiality***, makes data processors responsible for the security of the collected data. This very broadly defined principle requires protection from accidental deletion or unauthorized access and generally a good cybersecurity posture to prevent incidents.

Finally, the last seventh principle, ***Accountability***, requires all data controllers to be able to fully prove their compliance to prevent them from untruthfully claiming GDPR compatibility.

2.4 Rights of data subjects

As mentioned in the introduction of this chapter, GDPR strives to enhance the privacy rights of individuals. This is achieved with eight fundamental privacy rights for all data subjects. These fundamental rights follow the described core principles of the GDPR and aim to give data subjects more control over how their personal data are being used.

Right to be informed allows all data subjects to be informed about what personal data is being collected about them by the data controller and for what purpose. This right furthermore requires the data controller to be able to provide a data retention period of the collected data and a way for the data subject to file a complaint.

Right for access to personal data states that if data subjects request a copy of their personal data or an information about how and for what purpose their personal data are being processed, the data controllers must comply with their request.

Right to rectification declares that all data subjects must be allowed to correct their inaccurate or incomplete personal data collected by any data controller. Data controllers are required to correct all reported inaccurate information as soon as possible.

Right to be forgotten or right to erasure allows data subjects to request a complete erasure of all their personal data collected by the data controller. This right applies under several conditions, such as withdrawal of data subject consent or if collected personal data is no longer necessary for its original purpose. Right to erasure also applies to all data processors, meaning that the data controller must contact all third parties that process collected personal data and request their erasure, making the compliance significantly more complex.

Right to restriction of processing states that data subjects can request the data controllers to stop processing their personal data if these data are inaccurate, unlawfully processed, or no longer needed.

Right to data portability declares that data controllers should provide means for data subjects to request their collected personal data in a structured, machine-readable format, allowing them to transfer these data to another data controller. By implementing this right, GDPR tries to reduce and discourage vendor lock-in practices [18].

Right to object the processing in certain conditions allows data subjects to object to processing their personal data. The typical example where individuals can use this right is to object to processing their personal data for marketing purposes. By objecting against such profiling, the data controller is obligated to immediately stop such processing of the personal data of the given data subject.

Rights in relation to automated decision-making and profiling allow data subjects to withdraw from being a subject in any decision-making without human involvement. Some examples where this right could be exercised are automatic assessments of performance or automatic evaluation of CVs[1]. In recent years, this right has significantly raised in importance due to recent advancements in artificial intelligence, raising concerns about its current applicability [19].

2.5 Compliance of this application

GDPR compliance in the context of this application is a very complicated topic as it operates not only with the personal data of the user but notably also with the personal data of the contacts provided by the user. This complexity is due to the nature of personal relationship managers, as their primary function is to structure and aggregate information, typically personal data, about given contacts.

Under normal circumstances, collecting personal data about contacts saved by the user would require consent from these contacts. A typical example of this can be seen in customer relationship managers. These applications structure and organize information about current and potential customers to generate leads. However, CRMs must first obtain consent from these contacts to store their personal data.

In the case of personal relationship managers, the situation slightly changes due to the different purpose of the data collection. While CRMs gather personal data with commercial intentions, PRMs typically do so for purely personal purposes. This is a significant distinction in the context of GDPR since it allows the use of so-called *Household exemption*, allowing users to store personal data about the saved contacts without their consent.

1. Curriculum vitae

2.5.1 Household exemption

The household exemption defined in Article 2(2)(c) is another important concept from the GDPR that exempts purely personal or household activities from the regulations outlined in GDPR. This exemption was created to allow data subjects to manage their personal matters without the burden of compliance [20].

In the context of personal relationship managers, the household exemption may be applied as users are expected to use the application for managing information about their friends, family, or colleagues purely for personal use, such as keeping track of birthdays, contact information, character typologies, or other similar personal activities. However, since GDPR does not provide a specific definition of what is considered to be a household or personal activity, it may be challenging to decide when this exemption applies.

Specifically, in the case of this project, the household exemption is applied to storing information about contacts created by the user to avoid the need to collect consent from these contacts. Furthermore, all contact data are encrypted with the password of the user, and it is not possible to share these data as they are meant for personal purposes only [20].

In any case, it is essential to understand that even if the household exemption is applied to the contacts of the users, the GDPR still applies to all data controllers and processors providing services for these personal activities. This means that this application still needs to fully comply with the GDPR, and therefore, it follows the described core principles and respects all rights of the data subjects.

2.5.2 Limitations of household exemption

It is essential to understand that there are certain limitations on the scope of household exemption. The absence of a clear definition of what is and is not considered a personal or household activity makes it difficult to distinguish between situations where a household exemption may or may not be applied. For example, personal relationship managers might sometimes offer capabilities for a more sophisticated and systematic approach to contact analysis that goes beyond what is typically considered a "personal" or "household" use.

This application, for example, provides two features for such advanced analysis that might not be classifiable as for personal or household needs. These capabilities that might not fit into household exemption are web scraping for gathering more public data about the given contacts and AI[2] analysis of contacts data with a Large Language Model (LLM). Therefore, users should receive consent from the contacts they want to analyze with the web scraper or LLM.

2.5.3 Consent management

Due to the household exemption limitations, this application requires users to explicitly state that they received consent from the given contact before they are allowed to use web scraping or LLM tools. This confirmation of received consent is, however, only on a best-effort basis since the application does not provide any comprehensive consent management and verification system. This means that users could potentially provide fake consent and bypass the restrictions. Furthermore, this solution does not fully comply with the Accountability principle of GDPR, which states that data controllers are responsible for ensuring compliance with GDPR and must be able to demonstrate such compliance. However, without consent management and verification mechanisms, the compliance could not be currently fully demonstrated.

A potential solution for this complication could be implementing an enforcing mechanism to verify given consent from the contacts or altogether emitting these problematic features from the application. As said before, the current solution relies solely on transparency and trust toward the users.

2. Artificial intelligence

3 State of the application and proposed changes

This application was initially developed in the period of three months as a demo for the "*CTT MU Start Your Business*" competition in 2023 by the author of this thesis. It was created only as a prototype with a strict deadline, and as a result, the final product had many technical shortcomings. This initial proof-of-concept demo version was significantly limited in its functionality and lacked some core features like a password reset. To better understand the capabilities of the original application, the figure 3.1 shows its use case diagram.

In summary, the prototype allowed users to create contacts and contact groups to better organize information about given people. Each contact consisted of premade questions about this person. Answers to these questions were then compiled and sent to the OpenAI API endpoint for GPT3.5, which analyzed the provided contact. The application then allows the use of premade or custom prompts to ask the AI any questions about the person. According to the received feedback from the users, the most common use case of this capability was a generation of personalized briefing before meeting someone, or recommendation of personalized gifts and activities the analyzed contact could like.

3.1 Analysis of the original application

Frontend

The frontend layer of the prototype was developed using Vue2 in combination with Nuxt2, reflecting its original design as a web application. Later, in an advanced stage of development, the focus of the project shifted more towards mobile application development. However, because of the limited time for implementation, it was decided to continue using Vue instead of rewriting the frontend to a native framework. This decision enabled fast shipping of new features and continuous implementation of jury feedback instead of rewriting the existing functions.

On the other hand, the continuous usage of web technologies has caused complications regarding the native user experience. Because

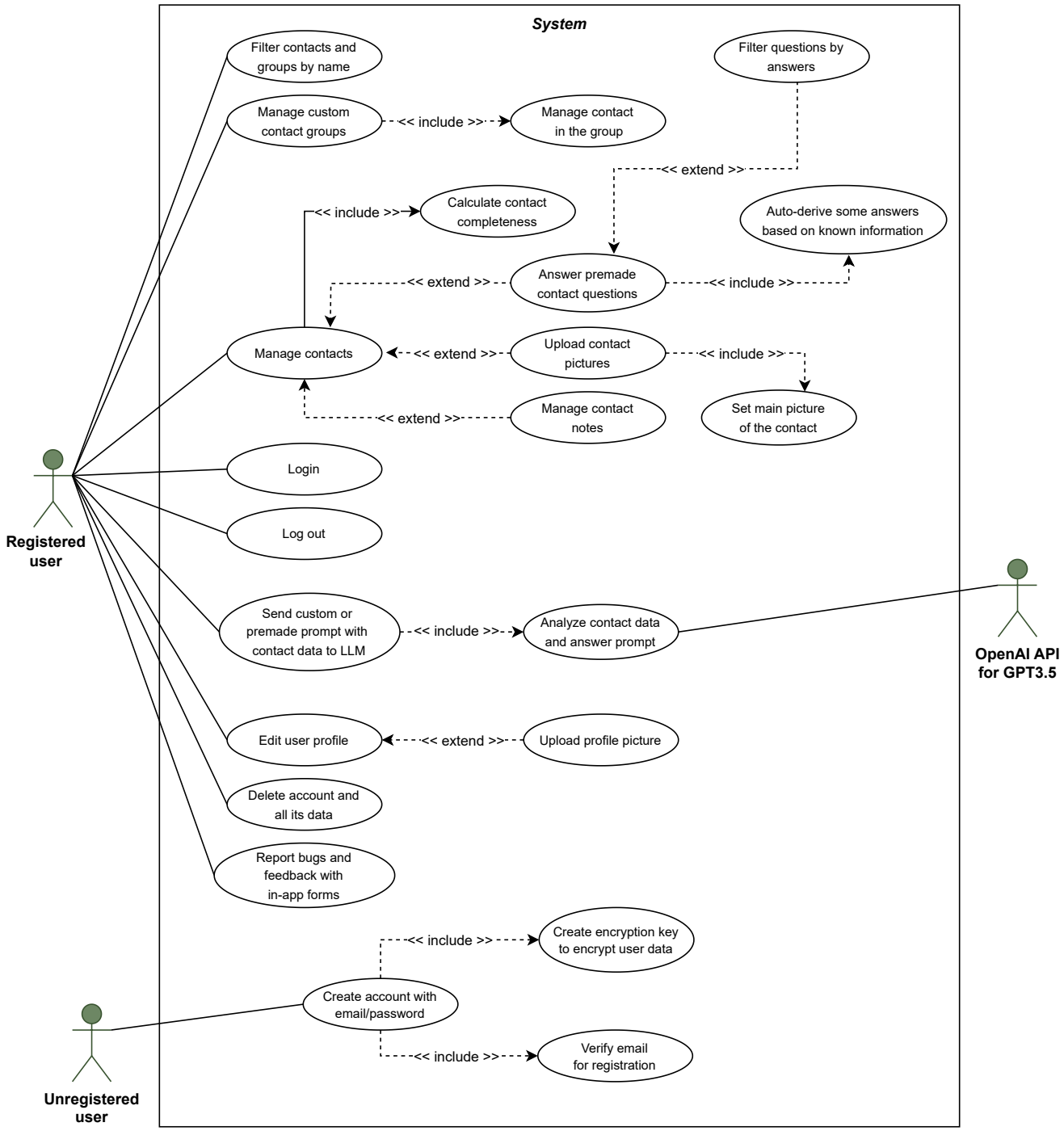

Figure 3.1: Use case diagram of the original application.

the application was designed only with custom CSS and several components from BootstrapVue, it felt more like a regular website than a mobile application. This lack of native user experience was caused by the absence of any mobile UI framework, such as Ionic or Framework7, that would otherwise compensate for the UX[1] drawbacks of hybrid applications.

Furthermore, using Vue version 2 proved to be a wrong decision because this version reached its End-Of-Life (EOL) on December 31st, 2023. Running production on an unsupported version of any framework would raise many security and functional concerns. These concerns eventually led to a decision to upgrade the frontend to Vue3 as a part of this thesis.

Backend

The backend architecture of the original application was initially written in plain PHP that was connected to the MySQL database with PHP Data Objects (PDO). However, due to the rising complexity of functional requirements, it was later migrated to Node.js hosted on Firebase. This transition to the backend-as-a-service (BaaS) solution outsourced many common functionalities such as authentication, storage buckets, or image compression. This outsourcing significantly improved the development speed but at the cost of reduced control over some features and increased operating expenses (OpEx).

The increase in OpEx was particularly noticeable because each backend endpoint was deployed as an individual cloud function. Most of these functions were constantly running to prevent potential cold starts; however, this accumulated hefty computational fees over time. Furthermore, the database was also notably moved to Supabase since Firebase does not offer any production-ready service for relational database hosting.

1. User Experience

3.2 Proposed changes for the rewrite

The original prototype had many shortcomings both in design and in implementation. Specifically, the most notable problems were a general lack of native user experience, low implementation quality, cost-ineffective backend architecture, fundamentally flawed encryption system, and a fairly limited amount of available features. Since all these problems require significant changes in the codebase, this thesis largely focuses on rewriting the application and not only extending it with new features. The following sections address these individual flaws in the original design and propose suitable solutions to implement as part of this rewrite.

Integration of Ionic framework

Due to the usage of traditional web technologies and lack of any mobile UI framework, the level of immersion was closer to PWA rather than to standalone mobile applications. To mitigate this issue, the first proposition for this rewrite is to achieve a more native user experience by implementing a UI framework called Ionic. This framework targets hybrid mobile applications such as this one by replicating native UI components with web technologies.

Transition to TypeScript and upgrade to Vue3

Another area for improvement in the prototype was a low level of implementation quality since all logic was written in JavaScript without any types or JSdoc. With the increasing size of the project, the maintenance of its codebase was becoming unsustainable. For this reason, the second proposition is to transition from JavaScript to TypeScript to make the project more maintainable. This proposition also encompasses upgrading deprecated frameworks such as Vue2 and Nuxt2 to the latest available versions.

Cost optimization of backend services

As mentioned in chapter 3.1, the backend architecture of the prototype could have been designed more effectively, causing high OpEx and

poor performance due to the cold starts of some cloud functions. This rewrite proposes a redesign of the backend architecture to reduce operational expenses and response time.

Improvement of the encryption system

The next significant flaw in the prototype that needs to be addressed is the design of an encryption system. The original application did not follow the best practices by, for example, reusing salt and initialization vectors (IVs) for all users. Reworking this encryption system to adhere to the best practices while keeping backward compatibility with the old system is another proposition for this rewrite.

Addition of new features

Since the prototype was only a demo version, it skipped many originally planned features, such as open-source intelligence (OSINT) capabilities or smart grouping of contacts. As part of this thesis, the following new features and improvements in table 3.1 will be implemented.

Feature or improvement	Type
Smart auto-grouping of contacts based on similar traits	New
Passcode system for opening the application	New
Password complexity indicator and bruteforce estimator	Rework
System for changing user password	New
System for resetting user password via recovery codes	New
System for generating and managing recovery codes	New
Screen for information about the last update	Rework
Native Google and Apple ID sign-in	New
In-app notification system	New
Haptic feedback for selected actions	New
Improved loading screen animations	Rework
Filter based on multiple properties, not only contact name	Rework
Native animations for route transitions	Rework
Support for hardware back buttons	New
Support for swipe-back gestures	New
OSINT system for scraping public user data	New
Credit system for data scraping	New
Conflict resolver for scraped data	New
Confirmation system for derived data	New
UI improvements	Rework
Dynamic safe area	Rework
Tabbed navigation with multiple navigation stacks	New
Live updates of SPA bundle	New

Table 3.1: Table of proposed features and improvements for implementation.

4 Implementation technologies

Identifying and choosing the appropriate technologies for the project's use cases is an integral part of the development process, with far-reaching implications. Choosing the wrong technologies can lead to slow development, poor application performance, or potential financial loss due to the need for a comprehensive project rewrite. This chapter introduces technologies used for the implementation of this project and also gives reasons for choosing them.

4.1 Frontend

Frontend is a presentational layer for websites and applications that users can interact with. It encompasses everything that users can see on their screens as well as the client-side behavior of the application. A well-written frontend aims to create an intuitive, responsive, and performant user interface (UI) to ensure a good user experience (UX). The following sections describe the frontend technologies used in this project.

4.1.1 Vue.js

Vue is a well-established open-source JavaScript framework for building interactive user interfaces for the web. It was created by Evan You in 2014 as a lightweight alternative to Angular.js for rapid prototyping. According to the State of Javascript 2022 report, Vue.js places in the top three frontend JavaScript frameworks alongside React and Angular [21], which showcases its rise in popularity since its creation.

This popularity is based mainly on simplicity, fast development, and the ability to be embedded into any existing code base as a standard web component, even if that codebase uses a different framework like React or Angular [22].

Reasons for choosing Vue.js

This project was initially meant to be a web application, not a mobile one, which is the main reason why a web framework was used to

develop a mobile application instead of more typical tools for this task, like React Native or Flutter.

Specifically, Vue was selected because it is ideal for quick prototyping, which allowed fast implementation of new feature requests and a faster reflection of feedback from the competition jury. Since the need to convert the application from web to mobile came at a later stage of the development, the decision to keep using Vue was made to prevent the need to rewrite the entire frontend codebase to a different framework.

Another reason for choosing Vue was the author's long-term interest in exploring the contemporary capabilities of web technologies aside from React Native in the development of native applications.

4.1.2 Nuxt.js

Nuxt.js is an open-source web framework for more structured, efficient, and faster development with Vue.js. It simplifies the development process by handling common configuration aspects, like middleware setup, automatic file-based routing, auto-imports of Vue components, or server-side rendering (SSR). These functionalities would otherwise require complex setups in plain Vue.js projects, but with Nuxt, most of this process is automated, which allows for a better developer experience (DX).

Nuxt also enforces standardized project structure to maintain best practices for web development [23]. Nuxt.js could be compared to a more known framework called Next.js, which is functionally very similar but was designed to extend the capabilities of React.

Reasons for choosing Nuxt.js

This project uses Nuxt.js, as it significantly reduces boilerplate code by abstracting and automating many repetitive configuration tasks, making the code much more readable and easier to understand. Furthermore, the opinionated project structure makes the application easier to maintain as it grows, which would be significantly more complicated if the project was written only with plain Vue.

4.1.3 Ionic

Ionic is an open-source UI toolkit for building native-like user interfaces for mobile applications using Angular, React, or Vue from a single codebase. This framework comes with ready-to-use UI components like sliders, modals, alerts, or toolbars, which are styled to exactly match their native counterparts from Android or iOS [24].

Besides the UI components, Ionic tries to replicate the native user experience with features like tabbed navigation or swipe-back gestures without resetting the DOM[1].

Reasons for choosing Ionic

Initially, this project did not use any UI framework, significantly reducing the native user experience as it was easy to recognize that the application was only an embedded webview. Achieving a native UX without a dedicated UI framework would be a significant challenge [25], leading to a decision to integrate Ionic as a part of this rewrite.

The decision to use Ionic and not other mobile UI frameworks like OnsenUI was made primarily due to its excellent compatibility with the utilized native runtime Capacitor.js, made by the same team as Ionic. Furthermore, Ionic is currently the most popular and active among web-based mobile UI frameworks, further solidifying the decision to use it.

4.1.4 Sass

Sass is a pre-processor that complies the SassScript or Sassy Cascading Style Sheets (SCSS) to a traditional CSS. This extension of CSS notably reduces code repetitiveness by declaring CSS variables and allowing the nesting of element selectors in the same way as HTML does.

Reasons for choosing Sass

Even though the project uses Ionic for core UI components, a large portion of the interface is still made with custom Vue components that require custom styling. The decision to use Sass with Vue was mainly

1. Document Object Model

based on the need for a consistent visual identity, achieved with CSS variables for colors, radiuses, or font sizes.

4.1.5 TypeScript

TypeScript is a strongly typed superset of JavaScript that introduces optional static typing and some features from object-oriented programming (OOP), like interfaces or abstract classes. These extensions help to better structure the data and allow early detection of potential runtime errors with improved intellisence. The simplicity and fast development make TypeScript the fourth most popular programming language according to the Stack Overflow 2023 Developer survey [26].

Reasons for choosing TypeScript

Using JavaScript over TypeScript might be reasonable for prototyping. However, when using web technologies to build larger codebases meant for production, TypeScript is now an industry standard. As the complexity and size of the original application started to grow, it became clear that going forward without TypeScript, or at least JSDoc, would cause many problems in the future. Since this rewrite further extends the size of the project, TypeScript is a natural choice going forward.

4.2 Backend

While the frontend technologies run on the user's devices, the backend runs on a server. It encompasses the underlying architecture for data management, storage, and processing. The backend typically provides an Application Programming Interface (API), which facilitates communication with the frontend part of the application. This connection allows users to access persistent data in the database, authenticate to the application, or perform complex operations that would otherwise require a significant amount of memory on the user's device.

The following sections describe the backend technologies used in this project.

4.2.1 Firebase

Firebase is a comprehensive set of tools for faster development and scaling of mobile and web applications. It is part of Google Cloud Platform (GCP), making it another cloud offering similar to AWS by Amazon or Azure by Microsoft. Firebase provides many tools for building the backend infrastructure and also for deploying and monitoring the application after its release. The following list mentions some of the core services Firebase provides.

- Serverless Cloud Functions for building custom endpoints
- Storage buckets and NoSQL database for storing data
- Comprehensive Authentication system

Reasons for choosing Firebase

When looking for a backend solution, the Firebase was a natural choice as it provides many tools for faster development and a free usage quotas covering most of this project's expenses.During the development of the prototype, building a custom backend infrastructure was out of the question as it would be very time-consuming, which left cloud providers as an optimal option.

The GCP was selected over Azure and AWS for its relative ease of use and fast setup. Other cloud providers might be unnecessarily too complex for the scope of this project. Choosing Firebase specifically from other GCP offerings was a natural choice as it is designed precisely for web and mobile application development, which makes it a perfect solution for hybrid mobile applications.

Furthermore, using Firebase made it possible to develop backend code with Node.js. Employing Node on Firebase allowed the use of JavaScript both on the frontend and backend, which contributed to a more unified codebase.

Regarding financial considerations, Firebase currently provides free usage quotas for their services as part of their No-cost Spark plan. These free quotas currently cover most of the project expenses. However, some Firebase offerings are available only for the Pay-as-you-go Blaze plan, making it a typical example of the OpEx business model cloud providers typically use [27].

This project uses the Blaze plan to get access to several features that are not otherwise available. Even with this paid plan, the free quotas still cover most of the usage, resulting in an average monthly bill of 25 CZK.

4.2.2 PostgreSQL database on Supabase

Supabase is an open-sourced BaaS that provides solutions like edge functions, storage buckets, SQL databases, or an authentication system. Supabase presents itself as an open-sourced alternative to Firebase since it provides a similar subset of functions. It was created in 2020 and recently became popular, especially for its free PostgreSQL database and modern web UI.

Reasons for choosing PostgreSQL database on Supabase

Firebase currently provides only one production-ready[2] database solution called Firestore. Firestore is, however, a NoSQL database, and since this project operates with highly structured data, it is more suitable to use a traditional relational database [28]. Since Supabase currently provides a free PostgreSQL database with 500 MB of space, it was a suitable option for this project. Furthermore, Supabase provides a web-based table editor, which makes work with the database considerably easier.

4.2.3 Prisma

Prisma is an open-source Object Relational Mapper (ORM) that allows Node.js to interact with a database. It creates an abstraction over the database by mapping its tables to JavaScript objects, which allows querying the database without directly writing any SQL queries. Specifically, Prisma uses a so-called schema file that defines the structure of the database, including the data types of individual columns. This file allows quick deployment of the database and tracking of its structural changes. Another significant benefit of using Prisma is

2. Google I/O 2024 conference presented a new Firebase service called Data connect for PostgreSQL database. Currently, it is in a Gated Preview.

the reduced complexity of queries and its automatic parametrization, which significantly reduces the likelihood of SQL[3] injections.

Reasons for choosing Prisma

The main reason for choosing Prisma was its type safety, and ability to track database changes with the schema file. Another considered option was the Supabase JavaScript client which takes advantage of other Supabase-specific features, such as a combination of Supabase authentication with Postgres row level security [29]. However, since this project uses a more established Firebase for the authentication and rest of the backend infrastructure, employing a Supabase JavaScript client would not take full advantage of its capabilities. Therefore, after conducting research, Prisma came out as the best option for the needs of this project.

4.3 Native runtime

Native runtimes, or native wrappers, allow web applications to be packaged as native applications that can be distributed to different platforms, such as Windows, macOS, iOS, or Android. They provide an interface for communication between the web layer and the native layer through native bridge bindings [9], making it possible to access native smartphone functions via a web application. The following list mentions several popular native runtime frameworks and some of the famous applications created with them.

- Electron - Visual Studio Code, Discord, Slack
- Capacitor.js - Burger King, BBC Children's & Education
- Cordova/PhoneGap - JustWatch, Untappd, Sworkit

3. Structured Query Language

4.3.1 Capacitor.js

Capacitor.js is an open-source native runtime for building hybrid iOS and Android applications. It was first released in 2018 and brands itself as a spiritual successor of Apache Cordova and Adobe PhoneGap. These two older projects originally pioneered the hybrid mobile application development as we know it today. However, after Adobe acquired PhoneGap in 2011, it subsequently announced the end of its development and support for its open-sourced counterpart Cordova in 2020 [30]. This discontinuation effectively marked the fall of these frameworks and allowed the rise of younger ones like Capacitor.js. Capacitor.js backed by Ionic builds on top of Cordova, further extending its capabilities while keeping backward compatibility with its predecessor[31].

Since Capacitor.js displays the aplication in a webview, it allows to use virtually any frontend technology, like Svelt, React, Vue or even plain HTML. Furthermore, since this framework is backed by Ionic, which is the creator of the Ionic UI framework, it greatly benefits from tight integration with Ionic components.

Other notable benefits of this framework are the native plugins that allow developers to access native functionalities like a camera or a file system by further extending the native bridge bindings. Moreover, since this framework is open-source, there is an active community publishing many new unofficial plugins, to extend the capabilities of this technology [32], indicating a healthy ecosystem.

Reasons for choosing Capacitor.js

Choosing Capacitor.js as a native runtime for this project was very straightforward as it is practically the only currently available option. Its famous predecessors, PhoneGap and Cordova, are not very active anymore, making Capacitor.js the only popular native runtime for hybrid mobile applications at the moment.

There are many native runtimes for desktop applications like Electron or Tauri. However, in the case of mobile applications, the market is currently dominated by cross-platform frameworks like Flutter and React Native, making runtimes for hybrid mobile applications rela-

tively rare [33]. Currently, the only attractive potential alternative for Capacitor.js is Tauri Mobile.

Tauri is an established framework for building desktop applications similar to Electron. However, at the end of 2022, Tauri announced the release of Tauri Mobile, which will target iOS and Android devices. This extension of traditional Tauri is currently in the beta version [34], making it unsuitable for production and leaving Capacitor.js as the only currently available contender.

5 Implementation

The following chapter will describe the implementation of several notable changes and new features added to the project as well as highlight some intriguing problems faced during the development. Several originally proposed changes and new features are not described in this chapter either due to low complexity or low overall significance in the system, however all these skipped features are documented in the attached project files.

5.1 System design

Before delving deeper into the specific implementation details, it is necessary to first get a high-level overview of the developed system and its architecture. The following diagrams will depict different notable parts of the system to get a better understanding of this application.

The first diagram in the figure 5.1 shows all different system components and how they interact with each other. This includes all third-party services that provide data for the application.

The second diagram in the figure 5.2 depicts the database design in form of Entity Relationship Diagram (ERD), to better understand what data are collected and how they influence each other.

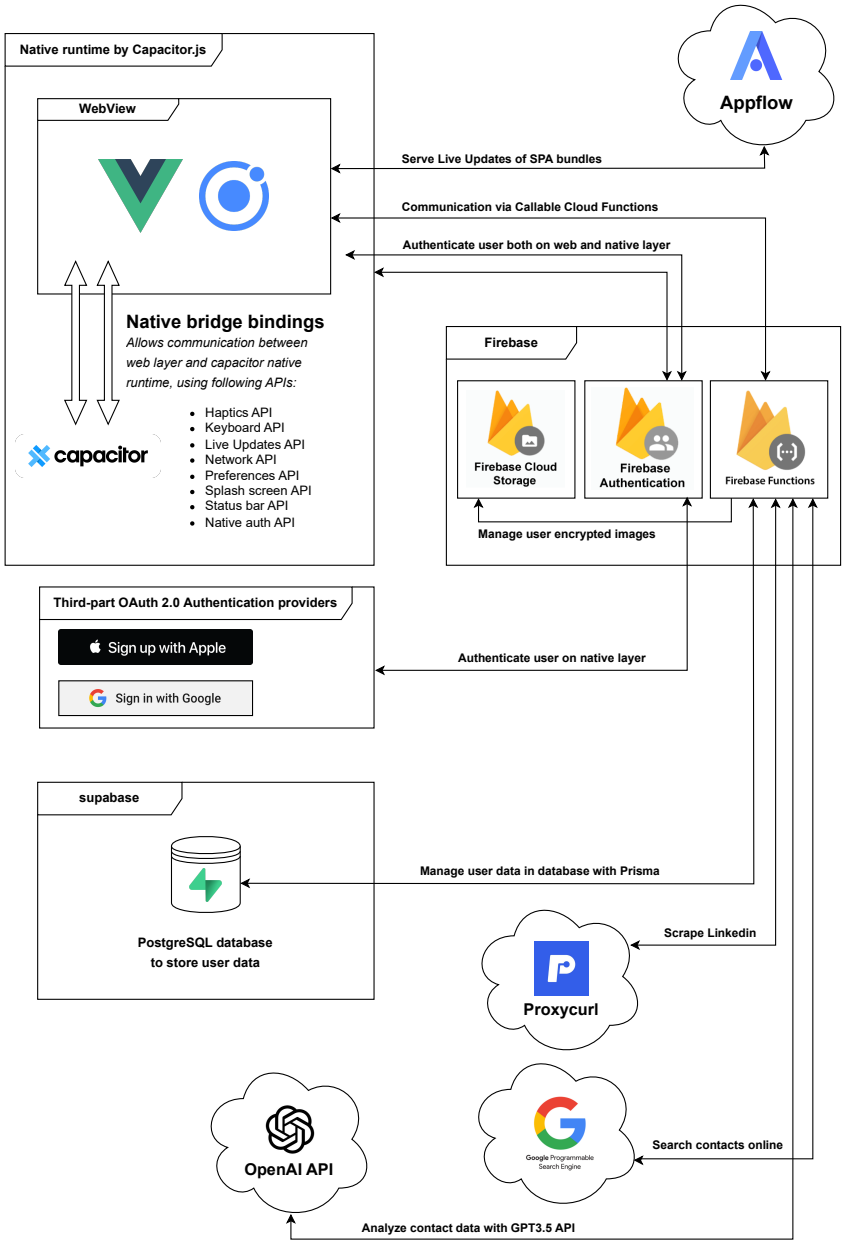

Figure 5.1: High level overview of the system architecture.

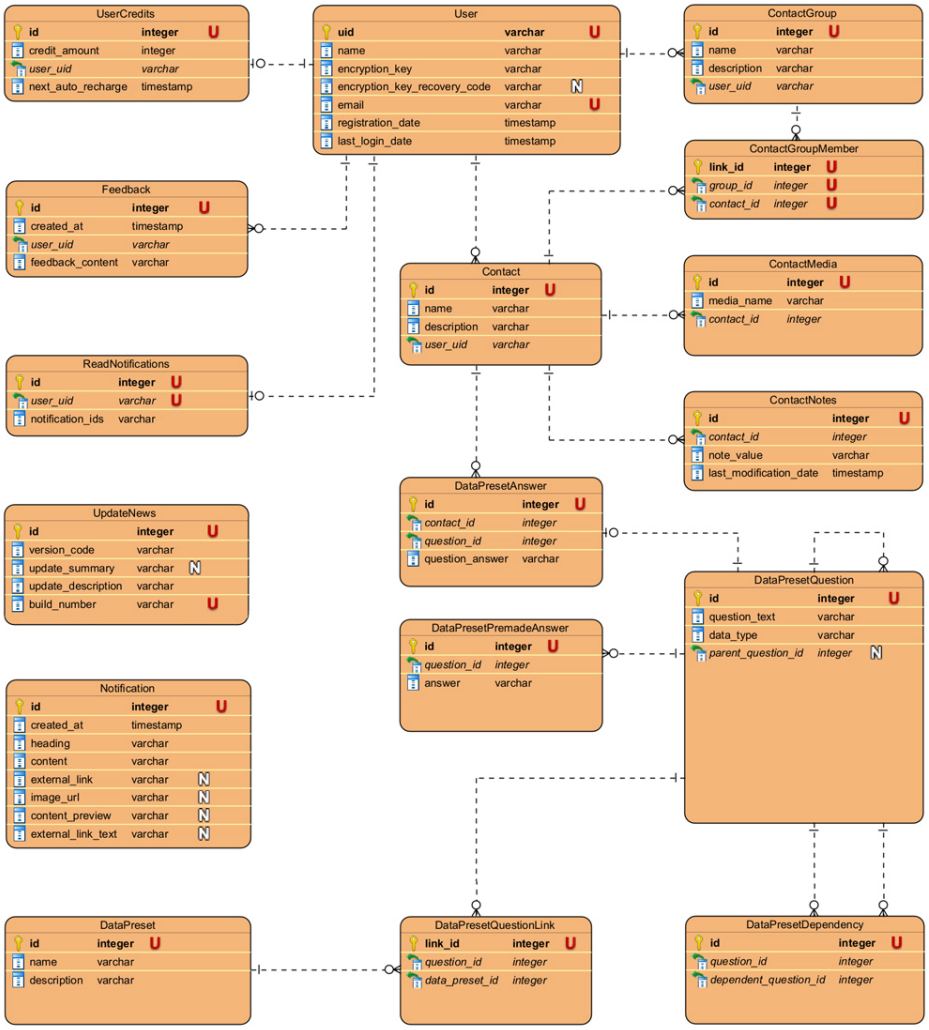

Figure 5.2: Entity relationship diagram of the project database.

5.2 Project structure

The directory structure of the project follows the default conventions set by Nuxt.js framework [23]. Following this standardized structure allows Nuxt capabilities like auto-import of Vue components or file-based route generation while maintaining a good separation of concerns for better project maintainability. The following bullet points describe the purpose of the core directories and files.

- **/android** folder contains Android Studio project files generated by Capacitor.js. The application can create a native Android build with the content of this directory.
- **/assets** folder contains all fonts, icons, images, and SCSS files used in the project.
- **/componencts** folder contains all Vue Single-File Components (SFC). These components allow the reuse of common UI elements across the application, and thanks to the Nuxt.js, they are also auto-imported.
- **/composables** folder is a new directory added in Nuxt3, to better manage Vue3 composition API. Nuxt.js uses this folder to auto-import all composable functions that serve to encapsulate and reuse stateful application logic across all Vue components.
- **/functions** folder contains code of all Firebase cloud functions. This means that this directory serves to stores the backend logic.
- **/icons** folder contains all generated icons of the application that will be used as favicons for the web version and as home screen icons for the mobile version.
- **/ios** folder contains Xcode project files generated by Capacitor.js. The application can create a native iOS build with the content of this directory.
- **/middleware** folder contains core routing logic for redirects.

- **/pages** folder contains all individual pages of this application. Nuxt.js auto-generates routes for all pages inside this folder.
- **/plugins** folder contains functionalities that are made available across the whole solution. In this project it is used to configure the connection to the Firebase project and to initialize the Vuex store.
- **/public** folder is used to store static assets that will not be modified with the build process.
- **/store** folder is used to manage Vuex which is utilized for state management of the the application.
- **/utils** folder serves to store stateless utility functions used in the application.
- **app.vue** is the root of the Vue application. This file is used to render all pages and to manage application-wide hooks like live update checks.
- **capacitor.config.ts** is the main configuration file of the Capacitor.js. It is used mainly to manage and configure Capacitor native plugins.
- **error.vue** is a default error layout of the Nuxt.js applications. It is used to gracefully handle all frontend errors within the application.
- **nuxt.config.ts** is the main configuration file for Nuxt.js. It is used to activate Nuxt plugins like `@nuxtjs/ionic` or to configure targeted rendering options.
- **types.ts** is a file for storing all TypeScript types and interfaces used throughout the solution.

5.3 Transition from Vue2 to Vue3

As mentioned in chapter 3.1, the frontend of this project was initially written with Vue2 and Nuxt2. Going forward with these versions

would be unacceptable since Vue2 reached its End of Life (EOL) on December 31, 2023 [35], and Nuxt2 will also reach its EOL on June 30, 2024. Upgrading the project to Vue version 3 brought many challenges and breaking changes across the solution. This was mainly caused by many broken dependencies that needed to either be upgraded to a new major release compatible with Vue3 or be completely replaced.

5.3.1 Switch to the Composition API

A significant change in the release of Vue3 was the introduction of a new preferred API style called a Composition API, which presented many functional and syntactical changes. The original application was written with the predecessor of the Composition API, called Options API, which is significantly more simplistic but also more restrictive than the Composition API [36].

Since the scope of required changes in the original application was already fairly great, it was decided to fully rewrite all existing Vue components to the Composition API as it provided several benefits such as less boilerplate code, greatly improved reuse of the stateful logic with composable functions and more granular control over the reactivity.

This decision to switch API styles significantly helped to reduce repetitive and boilerplate code as most of the shared logic was moved from individual Vue SFC files to the new *composables* directory [23], allowing the sharing of common stateful logic between multiple Vue components from a single file. Separating logic from the UI also helped with the separation of concerns, making the code overall more readable.

5.3.2 Replaced dependencies

Transitioning to the new major version of Vue and switching from Options to Composition API led to breaking many core dependencies in the project. These broken dependencies were caused by several breaking changes in the Vue3 and incompatibility with the Composition API, as many dependencies were made to be used only with the older Options API. This led to an even more profound rewrite and changes to some of the core used technologies. The following

sections will describe some of the affected dependencies that need to be replaced or upgraded.

Transition from Nuxt2 to Nuxt3

Since Nuxt2, initially used in the project, is incompatible with Vue3, it was necessary to update this framework to its latest release, Nuxt3. This new upgrade also brought many braking changes as Nuxt version 3 was a complete rewrite of version 2, and since this application is fundamentally built around Nuxt, this transition required many changes across the whole solution.

This upgrade was achieved using a comprehensive migration guide from the Nuxt team [37] and so-called *Nuxt Bridge*, which allowed incremental transition between the two versions. After completing this transition, the new version notably improved DX with new features like auto import of Vue components or faster bundling thanks to the Vite, which replaced webpack in the Nuxt3 [38].

Removal of Nuxt2 Firebase Module

The architecture of Nuxt.js allows developers to use so-called modules that extend Nuxt's default capabilities. The original application used a module called `@nuxtjs/firebase` that simplified and partially automated the setup of Firebase JavaScript SDK[1]. However, this module was supported only for Nuxt2 and had to be removed after the upgrade to Nuxt3. This module could have been replaced with its defacto successor called `VueFire`; however, after some research, it was concluded that this additional abstraction did not provide any significant added value. For this reason, VueFire was not integrated into this rewrite. Instead, the JavaScript Firebase SDK was used directly without additional wrapping, allowing more granular control over Firebase services.

Removal of BootstrapVue

The original frontend of the application was mostly created with a custom SCSS; however, for some more complex components, the UI

1. Software development kit

library called BootstrapVue was used. Specifically, this library was used for auto-expanding text areas, all progress bars, dropdowns, modals, and some button groups. This UI library greatly helped with the development of the prototype. However, its components were not designed for mobile but primarily for the web, negatively contributing to the native UX of the application. After the upgrade to Vue3, this library had to be removed as it did not support the Composition API. All components using this library were made from scratch with a custom implementation or replaced with an equivalent solution from the Ionic framework.

5.4 Integration of Ionic framework

One of the greatest challenges in the development of hybrid mobile applications is achieving native user experience with web technologies. Improvement of the native UX was also one of the key goals of this thesis because the original application lacked any mobile UI framework. To solve this issue, a UI framework called Ionic was integrated into this project, aiming to replicate the behavior of native applications with web technologies.

Before integrating this framework, it was essential to decide which provided UI components would be used. One of the significant benefits of hybrid applications is the ability to profoundly customize the UI without any native restrictions. By replacing all existing custom UI components with Ionic, this customizability would be lost. Furthermore, the original application already had an established and tested visual identity, so replacing it completely would not be optimal. For this reason, it was decided to use only Ionic components that would not require complete rework of common UI elements like buttons, dropdowns, or contact and group cards. This decision allowed to keep a fairly customized visual identity while still improving the native user experience with Ionic features like native modals, alerts, and non-linear routing. The following sections will demonstrate the usage of some notable Ionic components used in the project.

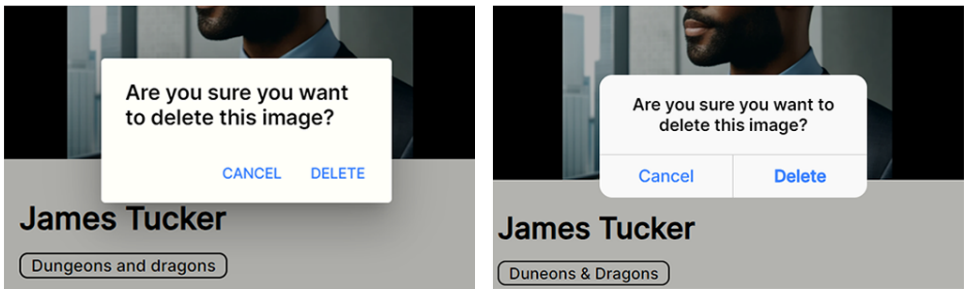

Figure 5.3: Ionic alert dialog with native Android UI on the left and native Apple UI on the right.

5.4.1 Alert dialog

This application uses native-like alert dialogs for several critical actions like image deletion, change of main contact picture, or merging of conflicting information. An example of alert dialog can be seen in the figure 5.3, showing different applied dialog styling based on the current platform.

5.4.2 Confirmation and sheet modals

Fullscreen confirmation and sheet modals are other used Ionic components in this project. Confirmation modals prohibit users from performing certain actions before explicitly confirming some statement, like an ownership of consent. On the other hand, sheet modals display extra information on an expandable panel. A good example of sheet modal usage is displaying contact details after clicking on the contact panel on a group page. Confirmation and sheet modal can be seen in the figure 5.4.

5.4.3 Tabbed navigation and Ionic router

The most advanced feature the Ionic framework provides is tabbed navigation, which significantly distinguishes web applications from mobile ones. While web applications typically have a single navigation stack with linear routing, mobile applications commonly employ multiple navigation stacks and, therefore, non-linear routing [39]. This

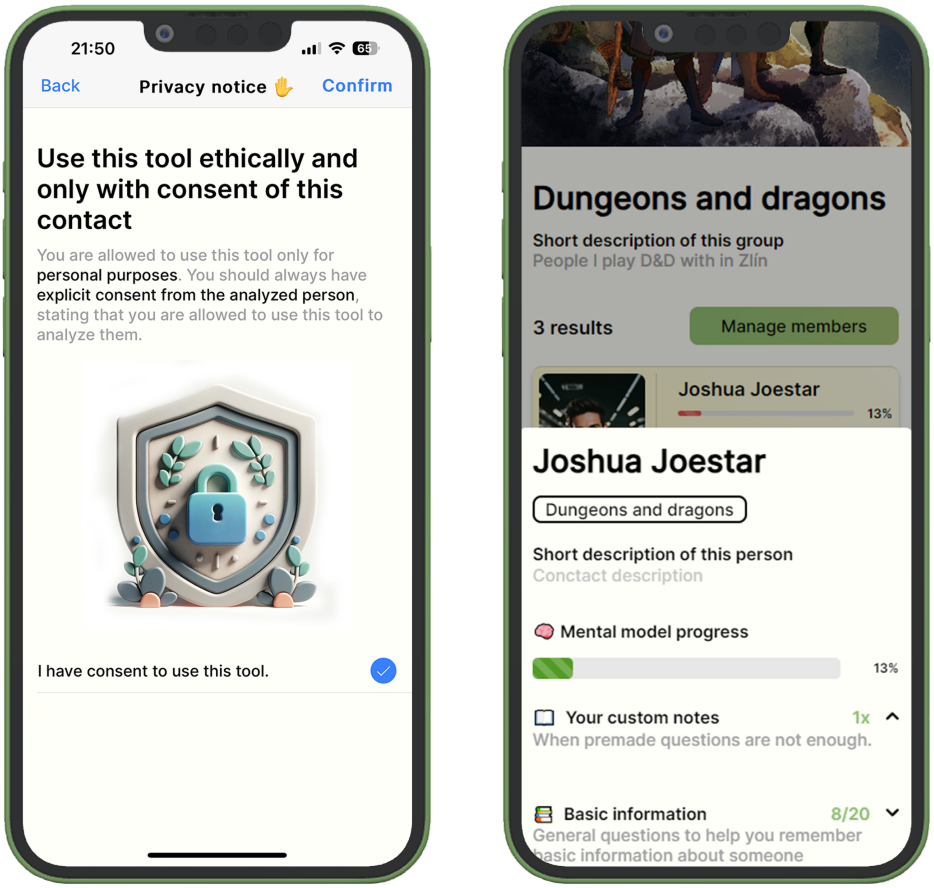

Figure 5.4: Ionic confirmation modal on the left and sheet modal on the right.

application uses Ionic tabbed navigation to achieve different navigation stacks for contacts, groups, and account pages.

Furthermore, by employing an Ionic router, which extends the functionality of the Vue router, this application now supports swipe back gestures and hardware back buttons for navigation. This is made possible by stacking individual pages on top of each other, leaving them in DOM instead of replacing them. Keeping previously visited pages loaded in DOM provides benefits like saved scroll position and faster page transitions, which is typical for mobile applications. The original application did not support any gestures or hardware buttons, and all of its page transitions were performed with custom Nuxt2 transition components. These custom transitions, however, did not mimic the native transition animations of the targeted device [40], which is why they were replaced with the Ionic router.

Challenges of tabbed navigation with Ionic

Implementing tabbed navigation with Ionic proved to be fairly complicated when using Vue.js. Ionic currently supports three frameworks: Angular, React, and, most recently also, Vue. Tabbed navigation currently works as expected when using React or Angular; however, in the case of Vue, things get complicated. Unfortunately, Ionic documentation for Vue.js contains several features that work only when using React or Angular. For example, tabbed navigation for Vue.js is documented as fully functioning, even though it is currently not usable due to several bugs in the Ionic core. Unfortunately, this was not mentioned in the documentation and was only later discovered in GitHub issues as an open bug [41, 42]. For this reason, it was necessary to develop a custom navigation stack management solution that does not rely on the one provided by Ionic.

5.4.4 Combining Ionic framework and Nuxt.js

It is also necessary to mention that Ionic and Nuxt are not designed to be compatible, and under normal circumstances, they cannot be used together. However, since this project was already deeply rooted in Nuxt.js, it would not be reasonable to remove it only to integrate a new UI framework. This complication was solved by using a recently re-

leased Nuxt module called `@nuxtjs/ionic`, which makes these frameworks compatible. This compatibility allowed structured development supported by Nuxt while also benefiting from native-like features provided by Ionic. Furthermore, this module significantly reduces the complexity of Ionic tabbed navigation by using Nuxt's file-based routing system to auto-generate routes for the ionic tabs.

5.5 Automatic grouping of contacts

In the original application, the user was able to create custom contact groups to better organize saved contacts. However, part of this process could have been automated by generating groups based on the contact similarities. This automation was achieved by checking for the same data across all contacts and automatically grouping contacts with the matching values. The result of this grouping is the formation of so-called smart groups. Smart groups are not saved in the database but rather regenerated on the user device every time the user modifies any contact data to check for potential new similarities between contacts.

Smart groups
All named generations since 1883
Contacts who are single
Sexual orientations other than heterosexual
Contacts who are still students
Unemployed contacts
Only child
Left-handed or Ambidextrous contacts
MBTI personality types
Elements according to the zodiac sign
Zodiac signs and Chinese zodiac signs
Enneagram types
Contacts with tattoos
Contacts with some form of PTSD or trauma
Married or divorced

Table 5.1: Currently available smart groups

Figure 5.5: Page for displaying auto-generated smart groups.

5.6 Implementation of passcodes

The ability to create a passcode that users would be required to enter in order to access their data was a request received in feedback from several users. During the analysis of this feature, it was decided that the passcodes would not be saved in the database but rather as a preference on the phone itself. Saving them as a preference allows the optional use of different passcodes on multiple devices. The passcode itself is saved via the Capacitor Preferences API provided by the native plugin called `@capacitor/preferences`. This plugin uses UserDefaults on iOS and SharedPreferences on Android.

The passcode is currently required at the fresh start of the application and after 2 minutes of inactivity. Furthermore, if the users forget their passcode, it can be reset with the user account password. The screenshots showing the passcode pages can be seen in the figure 5.6.

5.7 Native Google and Apple ID sign-in

Firebase authentication supports several login providers, including the Google and Apple ID. The original application provided only registration with email and password because all user data are encrypted with the user's password. However, when a user registers with a third party like Apple or Google, the application cannot access the user's password and, therefore, can't encrypt the user's data. This issue was solved with the Firebase ability to link one email address to multiple login providers. Since both Apple and Google provide the email addresses of registered users, it is possible to force users to enter additional passwords to create a new account with the email/password provider. This new account is then linked to the Apple or Google account, meaning that when users log in via these third parties, they will be asked to enter their password for the linked email account. This password can then be used to encrypt their data.

To implement native Google and Apple sign-in, a Capacitor plugin called `@capacitor-firebase/authentication` was used. This plugin renders the login form on the native layer on top of the webview, and after a successful login operation, it sends an authentication token to

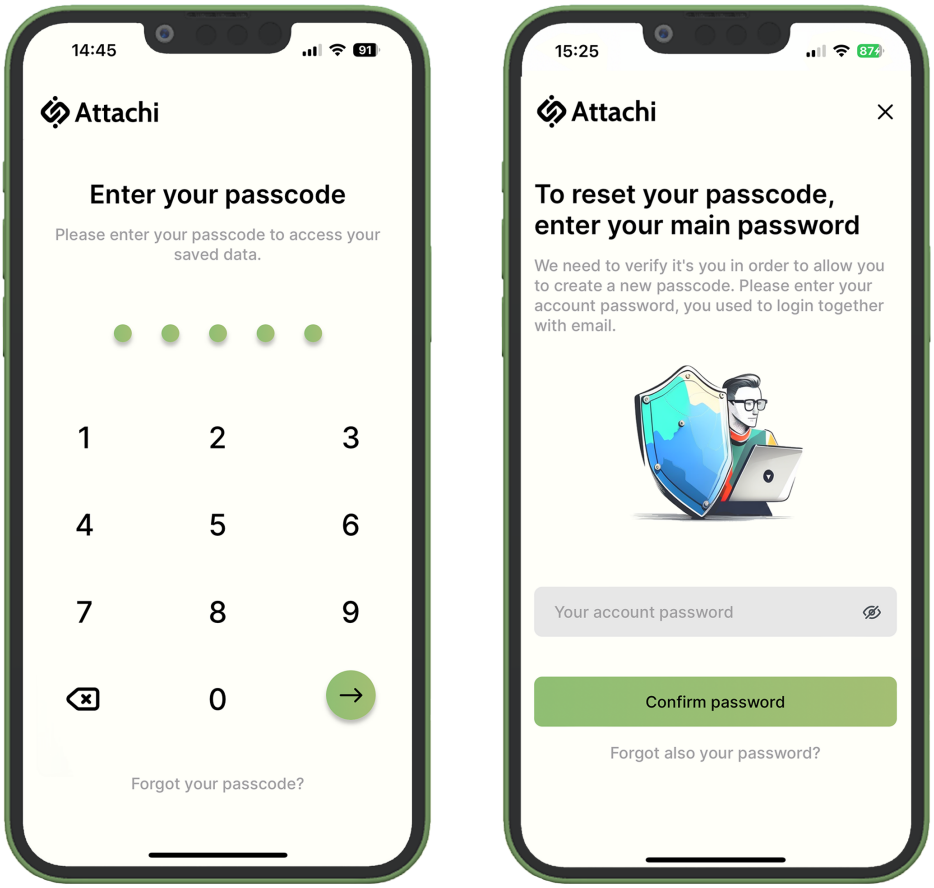

Figure 5.6: Passcode page blocking the user from entering the application.

the web layer. An example of such a native login overlay can be seen in the figure 5.7.

5.8 Support for live updates

Another notable advantage of hybrid applications is that they usually do not require directly updating the application binaries since the frontend is rendered in the webview. Unless some significant change that influences the native layer is introduced, all updates and changes can be handled without publishing a new version to app stores.

This application uses a service called Appflow that hosts the latest SPA bundles of the application. The upgrade is handled via a native Capacitor plugin called `@capacitor/live-updates`. On every start and resume event, this plugin sends a request to the Appflow asking for the latest SPA bundle. If a new version is available, it will be automatically downloaded in the background and applied on the application's next start or resume event.

Live updates allow a complete bypass of app store submission reviews, allowing quick deployment of bug fixes, UI changes, and minor updates in compliance with app store policies.

5.9 Cost optimization of backend services

The proof-of-concept version of the application used Firebase services for the backend infrastructure in a very unoptimized manner. The Firebase backend infrastructure in the context of this project refers specifically to Cloud functions and Firebase storage. The operational expenses of these services are determined by their usage, specifically by the size of stored files in the case of Firebase storage and by used computational power in the case of the Cloud functions. This rewrite addresses these pricing factors and proposes solutions described in the following sections.

5.9.1 Firebase storage optimization

The main factor determining the price of Firebase storage is the amount of stored data. Therefore, to decrease the OpEx of this service, the

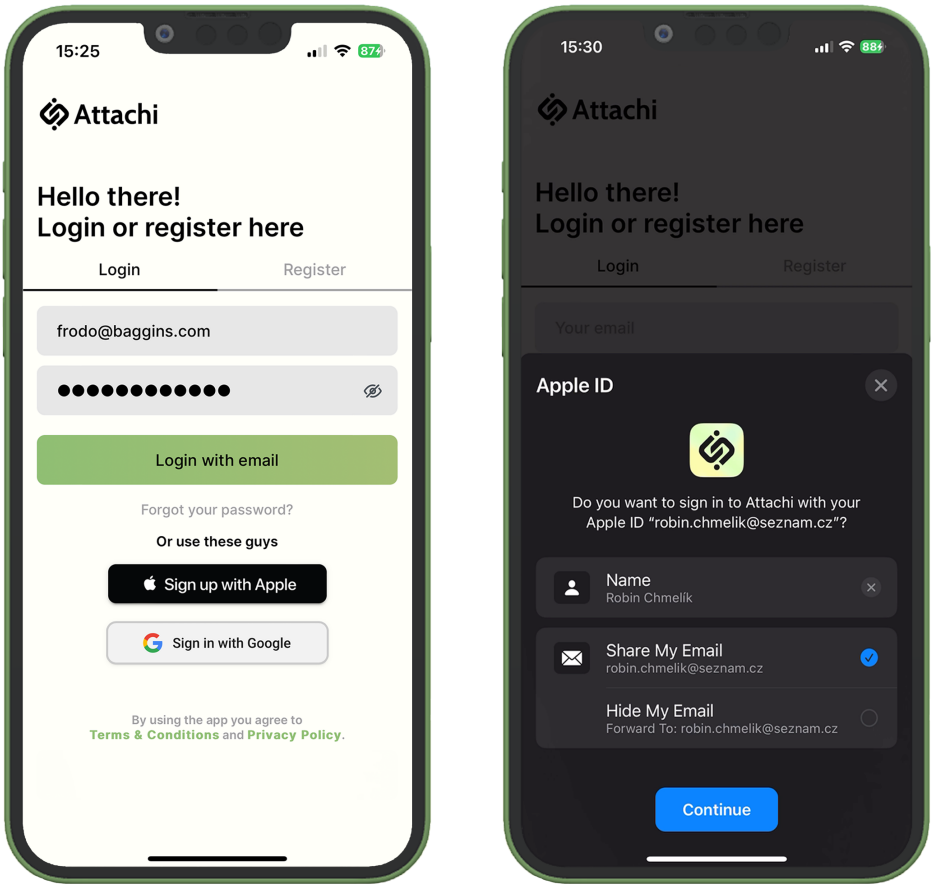

Figure 5.7: Third-party login providers on the left, and the native overlay for Sign in with Apple ID on the right.

size of uploaded files must have been reduced. This project uses Firebase storage to upload encrypted contact pictures and user profile pictures. In the original application, the images were immediately encrypted and uploaded to the storage without any additional modification. However, this approach was ineffective as the files were often several megabytes. This size problem was solved by compressing the image on the client with the library called Compressor.js, which on average reduces the file size by 67.99% for the price of 40% drop in the image quality. This greatly reduced the storage cost and significantly increased the speed and success rate of the upload.

5.9.2 Cloud function optimization

In the case of Cloud function, the main factor determining the price is the used computational power. The original application used individual Cloud function instances for each backend endpoint, wasting a lot of resources. The easiest solution for this problem was to unify all Cloud functions under a single endpoint that contained the logic of all existing functions. The core of this central Cloud function is a switch statement checking for a desired action based on the request parameter "*action*" and invoking corresponding logic accordingly. This allows to achieve the same results as with multiple cloud functions while decreasing the cost.

5.10 Encryption system

The original application had several problems regarding the design of the encryption system. Specifically, three notable problems needed to be addressed: reusing the same salt for all users, applying a low number of iterations for PBKDF2[2], and using a single static initialization vector for all cryptographic operations. In order to explain implemented solutions for these problems, it is necessary to first introduce the logic of this encryption system.

2. Password-Based Key Derivation Function 2

5.10.1 Design of the encryption system

Due to the sensitive nature of the user data, it is necessary to ensure that no one, including the developers with direct access to the database, can read the data except for the users themselves. This is achieved by encrypting all user data with an encryption key derived from the user's master secret, which is randomly generated upon registration. This master secret is further encrypted with the key derived from the user's password, which allows the decryption of user data only with the knowledge of that password. The revised workings of the encryption system are described in more detail below.

Creation of the master secret on user registration

1. Generate a new **master secret** for the new user account, in form of random 256 bytes.
2. Generate new unique **salt** for this user in the form of 32 random bytes.
3. Derive an **encryption key from the user password** using PBKDF2 and the user's generated salt. The PBKDF2 is configured to use HMAC-SHA512 hashing algorithm, with 21000 iterations and a hash output length of 32 bytes to match the required key length for AES-256-CBC.
4. Encrypt the master secret with the encryption key derived from the user password using **AES-256-CBC** with a random initialization vector (IV). This random IV has a length of 16 bytes to match the block size of AES-256-CBC.
5. The encrypted master secret is saved in the database to the corresponding user record in the following hexadecimal form: `(user salt + random IV + encrypted master secret)`

Process of user data encryption

1. User provides the encryption key derived from his password alongside the user data to be encrypted.
2. Provided encryption key is used to decrypt the encrypted master secret of the current user as it was encrypted with this key.
3. A new encryption key is derived from the decrypted master secret using the PBKDF2 and the user salt.

4. This new encryption key is used to encrypt the provided user data using AES-256-CBC with a new random IV.
5. The encrypted user data are saved to the database in the following hexadecimal form:
   `(random IV + data encrypted with the key derived from the master secret)`

5.10.2 Implemented improvements

The original application used a single salt and initialization vector, both saved as environment variables, for all cryptographic operations, which is a fundamentally flawed approach. The following changes were made to prevent potential exploitation of this flawed design.

Unique salt for all users

Regarding salt usage, the best approach was to generate a new random salt for each user during registration. Instead of saving it as a single environment variable, it is now converted to a hexadecimal format and added to the start of the encrypted master secret record of the corresponding user in the database

Random Initialization vectors for all cryptographic operations

Reusing the same Initialization vector (IV) with the same encryption key can lead to deterministic patterns in the ciphertext, making it vulnerable to cryptanalysis [43]. A new random IV is now being generated for all cryptographic operations to prevent these potential issues. This new IV is then prepended to the start of the resulting ciphertext, allowing a later description of the user data.

Updated PBKDF2 configuration

The initial prototype used an insufficient number of iterations to derive the encryption keys. Specifically, the original configuration of this function was eight iterations with a hash output length set to 16 bytes. According to the OWASP[3], the recommended number of iter-

3. Open Worldwide Application Security Project

ations for PBKDF2-HMAC-SHA512 is currently 210000 [44], which is well above the original value. Furthermore, the hash output length should match the key length of AES-256-CBC, which is 32 bytes, not 16. Therefore, the PBKDF2 configuration was updated according to these recommendations.

5.10.3 Backward compatibility

Since the original application was already in production before the rewrite of the encryption system, it contains a significant amount of user data encrypted with the old system. This situation required to design the new encryption system with backward compatibility to prevent data loss of existing users. This compatibility was achieved by checking the user's registration date and choosing the correct encryption system accordingly. If the user registered before the release of this rewrite logs in to the application, the old encryption system decrypts the master secret of the account and re-encrypts it with the new encryption system, allowing a seamless transition between these two systems.

5.11 Recovery codes and password change

The original application did not offer users any way to change or reset their password due to the design of the encryption system. As previously described, all user data are encrypted with an encryption key derived from a master secret unique to all users. This master secret is then encrypted with an encryption key derived from the user password. This approach greatly improves the security and privacy of the user data but at the cost of usability and convenience, as this removes the ability to reset forgotten passwords via confirmation emails.

Resetting the password this way would mean that the master secret, used to encrypt all user data, could not be decrypted anymore as this requires the old password. This means that the user password would be changed in the Firebase authentication system, but the user would not be able to access their data.

This problem was solved with the implementation of 16-character-long alphanumeric recovery codes. These recovery codes are used to derive a new encryption key, which then encrypts the user account's master secret. In case users now forget their password, they can use the recovery code to decrypt their master secret and subsequently re-encrypt it with a new password in the same way as described in the section "*Creation of the master secret on user registration*." After using the recovery code, it is automatically invalidated, and a new recovery code is generated for the user. This method allows users to change their passwords, balancing security and usability.

If the user is already logged in to the system, it is also possible to request a password change or a generation of a new recovery code. This is possible because logged-in users already have access to the master secret, which was decrypted during login. These features can be helpful if users lose their recovery code and need a new one. A screenshot of the recovery code page can be seen in the figure 5.8.

5.12 OSINT features

The most prominent new feature in this rewrite is the implementation of open-source intelligence (OSINT) capabilities. OSINT is the process of collecting publicly available information to gain more knowledge about someone or something. It is typically used by investigators, attorneys, or hackers to better understand their client or target [45].

OSINT is a feature that is not typically seen in personal relationship managers. However, its usage is reasonable as it allows to quickly gather publicly available information about the given contact in an automated manner. This automation helps to speed up the process of getting to know someone by searching for them on the Internet.

This application currently uses three tools for OSINT. Proxycurl People API is used for scraping data from LinkedIn, Google's Programmable Search Engine is utilized for automated Google search, and OpenAI GPT3.5 API is used to summarize the scraped data. The following sections will describe the tools used for the OSINT analysis and the logic of the scraping process.

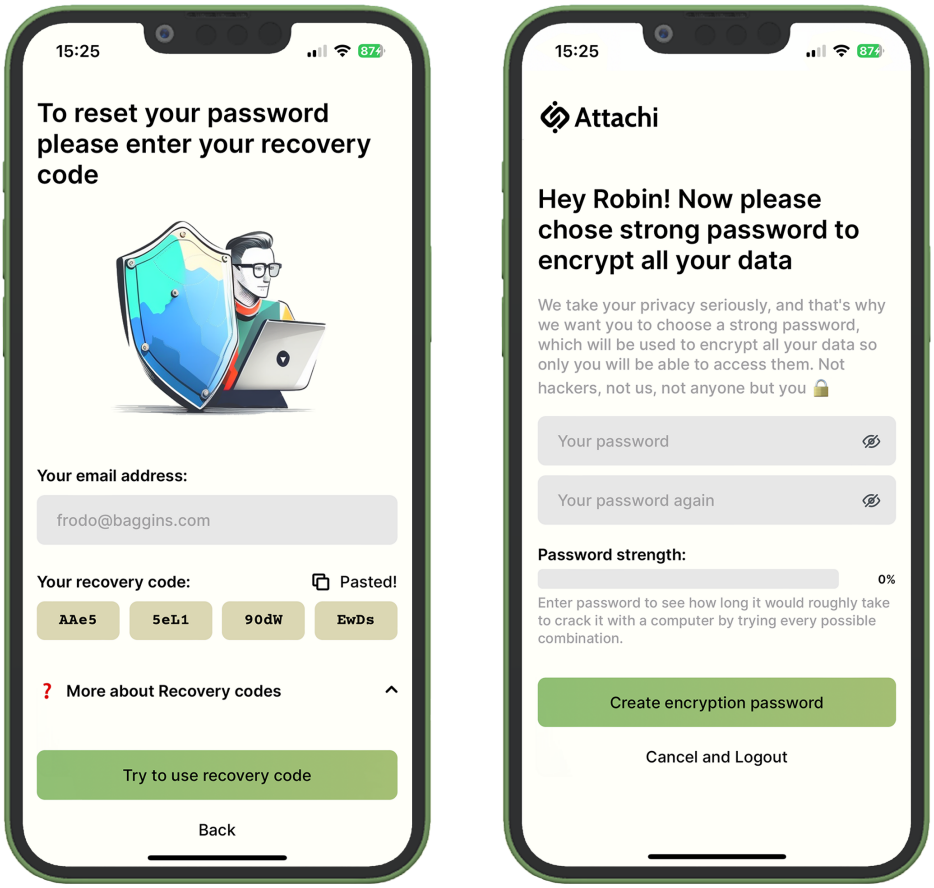

Figure 5.8: A recovery code page on the left, and the page for master secret creation on the right.

5.12.1 Proxycurl People API

When searching for the optimal source to scrape specific information about given contacts, LinkedIn was the best option as it usually contains the most publicly available information about a given person. There are several tools for scraping LinkedIn, Proxycurl being arguably the most popular. Proxycurl was also strongly recommended by the South Moravian Innovation Center, which funded and mentored this project.

Another considered option was the implementation of a custom web scrapper with tools for headless browsing like Puppeteer or Playwright. However, this would significantly increase the size of this feature as it would require renting an infrastructure for rotating proxy servers and dealing with compliance difficulties. For this reason, Proxycurl was selected for LinkedIn scraping as it is CCPA, GDPR, and SOC2 compliant and provides ready-to-use infrastructure.

Proxycurl provides several APIs for scraping different data. This project uses the People API, which returns structured JSON data from a given LinkedIn profile based on the URL.

5.12.2 Google Programmable Search Engine

Google Programmable Search Engine is not a typical web scraping tool as all the data are received directly from Google in JSON[4]. Programmable Search Engines allow developers to integrate custom-tailored search solutions directly into their applications. In the context of Google search, customizing the search engine means applying so-called Google hacking techniques that effectively filter the search results [45]. An example of Google hacking could be searching only for websites that contain an exact match of a provided string or limiting the scope of the search only to specified domains.

This application uses Google Programmable Search Engine to identify profiles of the user contacts across well-known social networks, to find news and articles about the contacts, to find PDFs and other files that contain the name of the searched contact, and to detect personal websites and Wikipedia pages if they exist. Specifically, the application takes advantage of several custom search engines configured to

4. JavaScript Object Notation

Search engine name	Target
YoutubeSearch	Channels and user profiles of the contact
TwitterSearch	Twitter/X profile page of the contact
FacebookSearch	Facebook profile page of the contact
WikipediaSearch	Dedicated page or mentions on Wikipedia
InstagramSearch	Instagram profile page of the contact
LinkedInSearch	LinkedIn profile page of the contact
FileSearch	Online files containing the contact name
GeneralSearch	News and articles mentioning the contact
OtherSocials	Tiktok, Snapchat, Quora, Pinterest, Reddit
NicknameSearch	Websites mentioning contact nickname

Table 5.2: Table of custom programmable search engines

exclude all URLs that do not match the profile page of the search platform. The table 5.2 describes these custom search engines.

5.12.3 Scraping process

The OSINT feature uses different custom search engines based on the information already known about the contact. The scraping process might take some time, so a visualization of scraping steps was implemented to prevent users from leaving before the completion of the search. This visualizer can be seen in the figure 5.9. along with the result page for scraped data. This console-like visualizer informs the user about the current stage of the scraping and shows already-identified information. The following sections will discuss the individual scraping stages in more detail.

Stage 1: LinkedIn profile lookup

Compared with other social networks, LinkedIn typically provides the most specific and useful information about the people registered on this platform. This availability makes it a perfect source for scraping data about contacts from PRM applications. This scraping can be performed with the People API from Proxycurl; however, this API requires the URL of the searched profile. Therefore, if the user has

already manually provided the contact's profile URL, this requirement is not a problem. However, if the contact is not linked to any LinkedIn profile, it must first be identified to use this API.

This identification is accomplished using Google custom search engine configured to return only LinkedIn profile pages with names similar to the searched contact. The search results contain several useful metadata, such as the profile picture, name, and job position of the found profile. These obtained metadata are then shown to the user, who is asked to select the correct profile. This search calibration also eliminates profiles with the same name, improving the final results. After confirming the correct LinkedIn profile, the found URL is sent to Proxycurl People API, triggering subsequent scraping stages.

Stage 2: Remaining custom search engines

After identifying the LinkedIn profile, the application calls the rest of the custom search engines to gather more information from other sources. This stage also includes the general search for files, news, and articles mentioning the contact's name and nickname. Furthermore, the application also tries to detect a personal website based on the Levenshtein distance between the found domain names and the contact name. For example, a domain name such as tucker-james.com would be marked as a personal website if the name of the searched contact is James Tucker.

Stage 3: LLM analysis of search results

Since the amount of returned results might be fairly extensive, all returned information is sent to the OpenAI GPT3.5 endpoint for summarization and analysis. Alongside the search results, the AI prompt is also appended with already-known information about the contact, allowing the AI to better distinguish between relevant and irrelevant search results. This AI summary helps users to quickly see the most relevant information without scrolling through all search results.

Stage 4: Saving the scraped data

After finishing all scraping operations and the AI summary, the user is allowed to modify and save all gathered data. These found data are furthermore automatically mapped to the premade contact questions, allowing them to be saved in a structured manner. Information that could not be mapped to any premade question can be saved as custom notes or as a contact description.

Furthermore, if the information that should be saved to the mapped question already has an answer, the user is shown an alert prompting him to select a merging strategy. There are currently two merging strategies: one replaces the original value, and the second merges both values with an easily recognizable separator.

Figure 5.9: The scraping process visualizer on the left, and the scraping result page on the right.

6 Testing and deployment

Testing the application before the deployment is an essential part of the SDLC[1]. This application was continuously tested during the development to discover issues as soon as possible to avoid accumulating and fixing them all at once just before release. Using this waterfall model might be initially faster but would bring a higher risk of not matching the user expectations due to the evolving requirements of the application. With the incorporated agile approach, it was possible to collect user feedback on many features before they were released and to discover bugs and security flaws in the early stages of the development, allowing faster fixes. The following sections will discuss different types of employed testing and the stages of the deployment process.

6.1 Penetration testing

Due to the sensitive nature of the information stored by this application, security testing played a fundamental role in its development. Performed security tests can be generally divided based on the following tested systems.

6.1.1 Cloud function testing

This project used the **BurpSuite** for intercepting and modifying all requests via a proxy to verify defenses against some of the most common attacks. By intercepting requests to Cloud functions, the attacker can modify its parameters and potentially bypass some restrictions if appropriate defenses are not implemented correctly. All Cloud functions were therefore tested with BurpSuite to ensure the attacker cannot use request tampering to access the data of other users or perform other malicious actions. An example of request tampering with BurpSuite can be seen in the figure 6.1.

1. Software Development Life Cycle

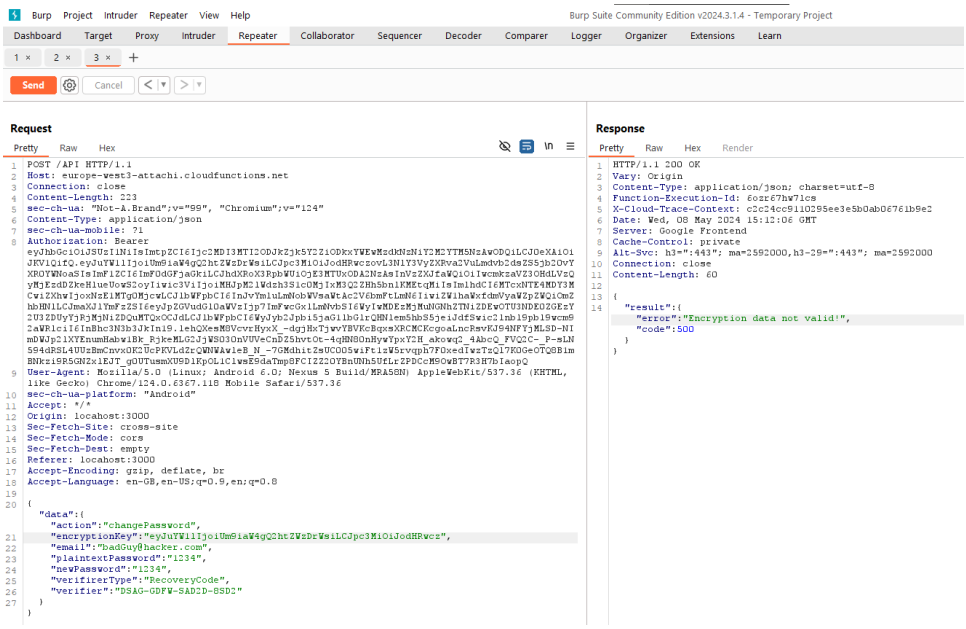

Figure 6.1: Security testing of an unauthenticated Cloud function for a password reset using the BurpSuite proxy.

6.1.2 Database access testing

It is important to remember that since the project does not use Firestore for the database, it cannot take advantage of its security features like automatic access verification to the requested data. The same thing can be said about Supabase Row Level Security rules for the used PostgreSQL database. These RLS rules cannot be used because the authentication is performed with the Firebase, not the Supabase.

Therefore, all database queries first manually compare the UID[2] of the authenticated user with the UID corresponding to the queried user data. Manually comparing these UIDs achieves the same result as the automatic access verification offered by Firestore. This comparison is possible since the UID of the authenticated user is provided by the context of the Callable Cloud functions, which cannot be forged. Even if an attacker would somehow query other users' data, it would not be possible to read them because they are encrypted with the other user's

2. User Identifier

password. This manual implementation of data access verification was also thoroughly tested to ensure database query parameters cannot be abused.

Furthermore, it is also important to mention that since the used Supabase database does not employ RLS rules, all its data could be quired directly with the Supabase client API, which would access the database as an anonymous user. Since this project does not use Supabase authentication, the only option to disable this API was to move all database tables to the custom private schema not exposed to the API. Direct querying via Supabase API was tested with the Supabase client library to verify that the only way to access the database is via Firebase Cloud functions.

6.1.3 Firebase storage testing

Unlike the database of this project, Firebase Storage, which is used for storing user-encrypted images, utilizes Firebase Security Rules to configure access permissions. The default configuration of these rules allows any authenticated user to access all files, regardless of the file ownership. Moreover, since the Firebase project configuration file is available within the SPA bundle, anyone can access it and query the Firestore and the Firebase Storage directly via the REST API or via pentesting tools like firepwn [46]. In January 2024, this insecure default configuration caused database breaches of over 900 websites, leaking 125 million account credentials [47], making it an ideal candidate for security testing. In the case of this application, all images in the Firebase Storage are encrypted, but users should still be able to access only the files they own, regardless of whether they are encrypted or not. For this reason, the security rules were configured to allow access only to files owned by the current user. Furthermore, this configuration was tested with the Firebase REST API to verify that users can access only files they own. The current Firebase Security Rules can be seen in the figure 6.2.

Figure 6.2: Firebase Security Rules for the Firebase Storage.

6.2 User testing

The second type of testing was not focused on security but rather on the functionality of the application itself. The application currently provides users with two in-app forms for submitting feedback and reporting bugs. This approach proved effective as it revealed many bugs in production and provided different perspectives on how the application should behave differently to better communicate its functionality. The application currently has over 200 unique registered users and has received 16 bug reports and feedback submissions that significantly helped to improve the user experience.

6.3 Continuous delivery

The deployment of this application currently contains several stages to thoroughly verify new features and minimize the amount of bugs in the production. The Git repository of this project contains two branches, specifically the master branch for the production builds and the staging branch for the development builds. Furthermore, the code bases in these branches use two copies of the database to separate production data and the testing environment with the mockup data. Finally, both branches also use two different Firebase projects, one for production and the other for staging. This separation ensures that new untested features that might pose a risk to the system do not interact with the production.

6.3.1 Static hosting for the web build

The staging branch is further connected to a hosting provider, render.com, which provides free static hosting for the application's SPA bundle. On each push to this branch, a new build is automatically triggered on render.com, allowing to easily share and test development builds in the form of a web application. These web builds are ideal to quickly share and test new features that do not interact with the native layer.

6.3.2 TestFlight and Google Play Internal testing

After testing the web build, the application is wrapped with the Capacitor native runtime and compiled as a native iOS and Android application. These native builds are then distributed to the beta testing platforms, specifically the TestFlight for iOS and Google Play Internal testing for Android. These release candidates are shared with a few selected people to ensure there are no severe bugs and to verify that the build is ready for the Google Play and App Store submission reviews.

6.3.3 Release on App Store and Google Play

Once the internal testing of the release candidate builds is finished, it is sent to Google Play and App Store for a review. This review usually takes around two days, and if no severe bugs or violations of guidelines are found, the application is released to the public. Currently, the application can be downloaded from Google Play and App Store under the name Attachi, version 1.3.2.

6.3.4 Live updates after release

In case a bug is found within the production SPA bundle and it does not interact with a native layer in any way, it is possible to fix it with the live update feature provided by Appflow to remotely update the SPA bundle on all devices that downloaded the application. These live updates allow to completely bypass the Google Play and App Store submission procedure, which might take several days, allowing to push patches for critical bugs as soon as they are discovered.

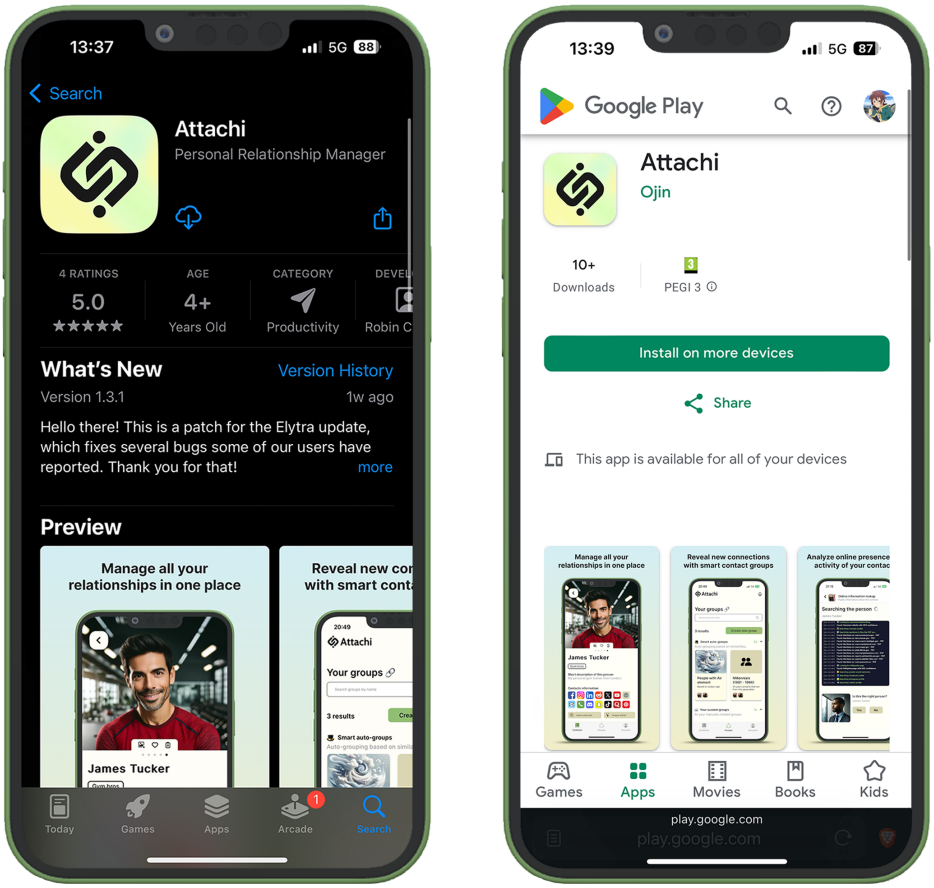

Figure 6.3: The product page of this application on the App Store and Google Play.

7 Possible future improvements

The application successfully implemented all proposed features and transitioned to the latest versions of used frameworks, however, there is still space for possible future enhancements. These improvements are not limited only to technical aspects but also to the addition of new features that are still missing in the application. A good example of further possible feature extension is the addition of the consent management system, as discussed in the chapter 2.5.3.

Furthermore, the application currently does not collect any data about user activity, making it difficult to determine how users use the provided tools. Integration of services like Google Analytics would significantly help to understand user behavior and detect existing bottlenecks in the UX.

Another possible improvement would be the implementation of a custom web scraper instead of utilizing Proxycurl and Google Programmable Search Engine. A custom web scraper would allow more granular control over collected data and also decrease operational expenses. The same thing can be said about the custom implementation of the LLM model instead of using OpenAI API. Even though this API is HIPAA and SOC2 compliant and does not use provided data to train new LLMs [48], it would still be better to host any public LLM model on a private server within the EU. Such migration would notably increase the privacy and trust of the users towards the application.

Apart from adding new features, the application would greatly benefit from properly implementing tabbed navigation. The current solution is not very user-intuitive due to the limited functionality caused by the internal Ionic bug [41] as described in chapter 5.4.3.

Last but not least, the application would also benefit from more strictly defined transactions between the frontend and the backend. The current solution uses the TypeScript interfaces called `requestData` and `responseData` for all transactions, which is not optimal as these interfaces do not enforce the required request parameters. Implementing unique interfaces for all endpoints would improve data consistency across the environments.

8 Conclusion

This thesis successfully achieved its proposed goals by implementing all required new features and rewriting the application to use the latest releases of its technologies. The application was thoroughly tested and later publically released on Google Play and App Store as a personal relationship manager with OSINT and AI capabilities.

Specifically, the new OSINT feature now supports scraping LinkedIn profiles using a Proxycurl People API. Furthermore, the application is also capable of locating contact profiles across the most significant social networks and of searching for news, articles, and various online files, mentioning the search contact name or nickname, using the Google Programmable Search Engine.

Since the application uses web technologies for its frontend, the Ionic framework was added to this project to enhance its native user experience by introducing features like tabbed navigation, native alerts and modals, or support for hardware back buttons and swipe gestures. Moreover, several native capabilities, like haptic feedback or a native overlay for registration with Apple or Google, were introduced using a native runtime called Capacitor.js.

Due to the uncommonly used hybrid technologies for the development of this mobile application, this thesis discussed different approaches to the development of modern mobile applications with a focus on hybrid and web-native applications. The thesis compared these different types of mobile applications and analyzed their advantages and disadvantages.

Finally, since personal relationship managers operate with highly sensitive personal information, this thesis also explored regulatory implications in the context of GDPR. Several mechanisms, like encryption and simple consent confirmation systems, were introduced to avoid legal repercussions and achieve compliance. Finally, this thesis analyzed the GDPR Household exemption and its limitations in the context of personal relationship managers.

Bibliography

1. CECI, Laura. *App stores - Statistics Facts* [online]. Statista, 2024 [visited on 2024-04-17]. Available from: https://www.statista.com/topics/1729/app-stores.
2. LATIF, M.; LAKHRISSI, Y.; NFAOUI, H.; ES-SBAI, N. Cross platform approach for mobile application development: A survey. In: *2016 International Conference on Information Technology for Organizations Development (IT4OD)*. 2016, pp. 1–5. Available from DOI: 10.1109/IT4OD.2016.7479278.
3. OLIVEIRA, W.; MORAES, B.; CASTOR, F.; FERNANDES, P. Analyzing the Resource Usage Overhead of Mobile App Development Frameworks. In: *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*. Oulu, Finland: Association for Computing Machinery, 2023, pp. 152–161. EASE '23. ISBN 9798400700446. Available from DOI: 10.1145/3593434.3593487.
4. *Who is using React Native?* [online]. React Native, 2024 [visited on 2024-04-17]. Available from: https://reactnative.dev/showcase.
5. AJAYI, O.; OMOTAYO, A.; OROGUN, A.; OMOMULE, T.; ORIMOLOYE, S. Performance Evaluation of Native and Hybrid Android Applications. *Communications on Applied Electronics*. 2018, vol. 7, no. 16, pp. 5–6. ISSN 2394-4714. Available from DOI: 10.5120/cae2018652701.
6. NETKOW, Matt. *Burger King's Mobile App Design System and Brand Consistency* [online]. Ionic, 2024 [visited on 2024-05-03]. Available from: https://ionic.io/resources/articles/burger-king-design-system.
7. ANDERSSON, Lina. *Usability and User Experience in Mobile App Frameworks* [online]. Uppsala, 2018 [visited on 2024-05-03]. Available from: https://www.diva-portal.org/smash/get/diva2:1275254/FULLTEXT01.pdf. MA thesis. Faculty of Science and Technology, Uppsala University.

8. *App Review Guidelines* [online]. Apple Developer Program, 2024 [visited on 2024-04-17]. Available from: https://developer.apple.com/app-store/review/guidelines/#minimum-functionality.
9. PAKSULA, Matti. *Measuring a HTML5 Hybrid Application's Native Bridge on iOS* [online]. Helsinki, 2016 [visited on 2024-05-03]. Available from: https://helda.helsinki.fi/items/1e7fd097-879f-4277-b640-e5d9b46460fc. MA thesis. Faculty of Science, University of Helsinki.
10. HAIRE, Andrew. *What is a Web Native App?* [online]. Ionic, 2020 [visited on 2024-05-04]. Available from: https://ionicframework.com/docs/theming/platform-styles.
11. *PWA Capabilities*. web.dev, 2024. Available also from: https://web.dev/learn/pwa/capabilities.
12. *Web APIs*. mdn, 2024. Available also from: https://developer.mozilla.org/en-US/docs/Web/API.
13. *Browser support for Web APIs*. Can I use..., 2024. Available also from: https://caniuse.com/?search=api.
14. *App Updates for HTML5 Apps* [online]. Apple Developer Program, 2020 [visited on 2024-04-17]. Available from: https://developer.apple.com/news/?id=01212020a.
15. RYBCZONEK, Mateusz. *Make Your PWA Available on Google Play Store* [online]. Netguru, 2024 [visited on 2024-04-17]. Available from: https://www.netguru.com/blog/make-your-pwa-available-on-google-play-store.
16. JOVIĆEVIĆ, Mladen; JANJETOVIĆ, Željko. Position of the European Union in the Global Trade System. *ECONOMICS*. 2017, vol. 5. Available from DOI: 10.1515/eoik-2017-0021.
17. *EU position in world trade* [online]. European Commission, 2024 [visited on 2024-05-04]. Available from: https://policy.trade.ec.europa.eu/eu-trade-relationships-country-and-region/eu-position-world-trade_en.
18. BARTH, Marlene. A Case Study on Data Portability. *Datenschutz und Datensicherheit - DuD*. 2021, vol. 45, pp. 190–197. Available from DOI: 10.1007/s11623-021-1416-3.

19. ADRIENN, Lukács; SZILVIA, Váradi. GDPR-compliant AI-based automated decision-making in the world of work. *Computer Law Security Review*. 2023, vol. 50, p. 105848. ISSN 0267-3649. Available from DOI: https://doi.org/10.1016/j.clsr.2023.105848.

20. *Recital 18 - Not Applicable to Personal or Household Activities* [online]. intersoft consulting, 2024 [visited on 2024-05-04]. Available from: https://gdpr-info.eu/recitals/no-18/.

21. *State of JavaScript 2022: Front-end Frameworks* [online]. State of JavaScript, 2022 [visited on 2024-04-17]. Available from: https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/.

22. *Vue and Web Components* [online]. Vue.js, 2024 [visited on 2024-05-04]. Available from: https://vuejs.org/guide/extras/web-components.html.

23. *Nuxt.js Directory Structure* [online]. Nuxt.js, 2024 [visited on 2024-05-04]. Available from: https://nuxt.com/docs/guide/directory-structure.

24. *Platform Styles* [online]. Ionic, 2024 [visited on 2024-05-04]. Available from: https://ionicframework.com/docs/theming/platform-styles.

25. *Building Your UI - Roll your own* [online]. Capacitor.js, 2024 [visited on 2024-05-04]. Available from: https://capacitorjs.com/docs/getting-started/ui#roll-your-own.

26. *Stack Overflow Developer Survey 2023* [online]. Ionic, 2023 [visited on 2024-04-17]. Available from: https://survey.stackoverflow.co/2023/.

27. ANDREO, S.; CALÀ, A.; BOSCH, J. OpEx Driven Software Architecture a case study. In: Sweden: CEUR Workshop Proceedings, Vol. 2978, 2021, pp. 1–3. 15th European Conference on Software Architecture - Companion, ECSA-C 2021. Available also from: https://ceur-ws.org/Vol-2978/industry-paper95.pdf.

28. MEIER, Andreas; KAUFMANN, Michael. *SQL NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management*. 1st ed. Germany: Springer Vieweg Wiesbaden, 2019. ISBN 978-3-658-24549-8. Available from DOI: 10.1007/978-3-658-24549-8.

29. MEYERS, Jon. *Using Row Level Security with Supabase Auth* [online]. supabase, 2024 [visited on 2024-05-04]. Available from: https://supabase.com/docs/guides/auth/row-level-security.

30. *Goodbye PhoneGap* [online]. Apache Cordova, 2020 [visited on 2024-05-04]. Available from: https://cordova.apache.org/announcements/2020/08/14/goodbye-phonegap.html.

31. *Migrating a Web App Using Cordova to Capacitor* [online]. Ionic, 2024 [visited on 2024-05-04]. Available from: https://capacitorjs.com/docs/cordova/migration-strategy.

32. *Capacitor Community Plugins* [online]. Ionic, 2024 [visited on 2024-05-04]. Available from: https://capacitorjs.com/docs/plugins/community.

33. VAILSHERY, Lionel Sujay. *Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2022* [online]. Statista, 2022 [visited on 2024-05-04]. Available from: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/.

34. NOGUEIRA, Lucas. *Announcing the Tauri v2 Beta Release* [online]. Tauri, 2024 [visited on 2024-05-04]. Available from: https://beta.tauri.app/blog/tauri-2-0-0-beta/.

35. *Vue 2 Has Reached End of Life* [online]. Vue.js, 2024 [visited on 2024-05-04]. Available from: https://v2.vuejs.org/eol/.

36. *Composition API and its relationship with Options API* [online]. Vue.js, 2024 [visited on 2024-05-04]. Available from: https://vuejs.org/guide/extras/composition-api-faq.html.

37. *Guide to migrating from Nuxt 2 to Nuxt 3* [online]. Nuxt.js, 2024 [visited on 2024-05-04]. Available from: https://nuxt.com/docs/migration/overview.

38. SAID, Mostafa. *Vite vs. Webpack: A Head-to-Head Comparison* [online]. Kinsta, 2024 [visited on 2024-05-04]. Available from: https://kinsta.com/blog/vite-vs-webpack.

39. *Linear Routing versus Non-Linear Routing* [online]. Ionic, 2024 [visited on 2024-05-04]. Available from: https://ionicframework.com/docs/vue/navigation#linear-routing-versus-non-linear-routing.

40. *Nuxt 2 Transition component* [online]. Nuxt.js, 2024 [visited on 2024-05-04]. Available from: https://v2.nuxt.com/docs/features/transitions/.

41. MOORE, Philip. *bug: switching between shared tab urls does not preserve individual tab stack history* [online]. GitHub, 2022 [visited on 2024-05-04]. Available from: https://github.com/ionic-team/ionic-framework/issues/25213.

42. *bug: certain tab methods missing on framework integrations* [online]. GitHub, 2019 [visited on 2024-05-04]. Available from: https://github.com/ionic-team/ionic-framework/issues/19918.

43. PICCA, Florian. *Initialization vector mishandling* [online]. Stackered, 2023 [visited on 2024-05-04]. Available from: https://stackered.com/blog/iv-mishandling/.

44. *Password Storage Cheat Sheet* [online]. OWASP, 2024 [visited on 2024-05-04]. Available from: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#pbkdf2.

45. BAZZELL, Michael. *OSINT techniques resources for uncovering online information*. 10th ed. USA: Independently published, 2023. ISBN 979-8366360401. Available also from: https://inteltechniques.com/book1.html.

46. 0XBIGSHAQ. *firepwn-tool* [online]. GitHub, 2024 [visited on 2024-05-04]. Available from: https://github.com/0xbigshaq/firepwn-tool.

47. LOGYKK; XYZEVA; MRBRUH. *900 Sites, 125 million accounts, 1 vulnerability* [online]. env.fail, 2024 [visited on 2024-05-04]. Available from: https://env.fail/posts/firewreck-1/.

48. *Enterprise privacy at OpenAI* [online]. OpenAI, 2024 [visited on 2024-05-04]. Available from: https://openai.com/enterprise-privacy/.

A Attachments

`Project.zip` - This file contains the source code of the application, including the Android Studio and Xcode project files.