MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Multi-platform mobile application for managing rent agreements

Master's Thesis

MICHAELA DRAŽKOVCOVÁ

Brno, Spring 2025

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Multi-platform mobile application for managing rent agreements

Master's Thesis

MICHAELA DRAŽKOVCOVÁ

Advisor: RNDr. Samuel Pastva, Ph.D.

Department of Computer Systems and Communications

Brno, Spring 2025


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 elaboration of this work are properly cited and listed in complete reference to the due source. During the preparation of this thesis, I used the following AI tools:

- Grammarly for grammar check,
- Github Copilot for faster code writing,
- ChatGPT to improve my writing style.

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.

Michaela Dražkovcová

Advisor: RNDr. Samuel Pastva, Ph.D.

Acknowledgements

I would like to thank On Point Services s.r.o. for allowing me to develop the mobile app as part of my thesis and thank all of my colleagues who worked with me on this project. Additionally, I would like to thank my advisor, RNDr. Samuel Pastva, Ph.D., for his valuable guidance, insightful feedback, and continuous support throughout the entire thesis process.

Abstract

This master's thesis presents the design, development, and deployment of a multi-platform mobile application for tenants, developed as part of a broader real estate management platform. The first part of the thesis outlines the problem definition and introduces the technologies used, including React Native and Expo. The main section provides an overview of the core functionality of the application, accompanied by relevant screenshots. The final chapters focus on implementation, testing, and the deployment process to app stores.

Keywords

React Native, Expo, Multi-platform, mobile app

Contents

List of Tables

List of Figures

Introduction

Mobile applications play an increasingly important role in simplifying everyday tasks and improving access to services across industries ranging from banking and communication to healthcare and housing. In the real estate sector, digital platforms are often designed to help property managers handle administrative workflows but they often lack features that provide tenants with a clear, centralized space to manage their rental needs. The goal of this thesis is to develop a multi-platform mobile application designed to support tenants in managing their rental experience. The application was built as part of a broader solution developed in collaboration with the company named On Point Services. By enabling users to access contracts, update personal information, submit requests, and sign documents through a mobile device, the application streamlines communication and reduces administrative overhead.

The thesis follows the app's development, from initial planning, technology selection, requirements gathering, implementation, to testing, and deployment. Chapter 1 introduces the problem context, presents market research, and describes the overall project management approach. The technologies used, with a focus on React Native and Expo, are discussed in Chapter 2. Functional and non-functional requirements are detailed in Chapter 3, while the overall system architecture is described in Chapter 4. Chapter 5 presents the user interface design and functionality of the application. Finally, Chapter 6 covers implementation details, deployment processes, and a summary of app stores statictics.

1 Project Overview

1.1 Introduction to the Problem

The mobile application was developed as a part of a broader product by On Point services s.r.o., a company based in Brno, Czech republic. The company specializes in providing Microsoft Dynamics 365 Business Central, an Enterprise Resource Planning (ERP) system and enhances this ERP solution by developing customized modules, add-ons, and integrations tailored to specific business needs.

In 2019 a real-estate company came with a request for a platform to manage all aspects of their real estate business, including property management, leases, and tenant interactions. In response, a customized Business Central module was developed to address their unique requirements. As part of this solution, a web portal was initially implemented for interactions between tenants and the company. In 2024, a business agreement was made between the real estate company (which mainly financed the development) and On Point services to commercialize the solution for the broader market. As part of this transition, a strategic decision was made to replace the existing web portal with a mobile application.

Unlike traditional single-company solutions, this application was designed with scalability in mind, allowing future adoption by multiple firms. This influenced architectural decisions, particularly regarding the backend, which was built to support multi-tenancy and flexible data segregation. Additionally, the application supports branding customization, enabling each company to tailor the mobile application by selecting primary colors and displaying their own logo on the home page. This maintainins a consistent brand identity within the shared platform.

As part of the customized Business Central solution developed for this platform, residential properties are grouped into entities referred to as complexes. A complex typically consists of one or more rental units (such as flats) and serves as a central grouping mechanism for configuring various property-specific settings. These complex-level configurations influence the behavior of the mobile application. For example, a complex may restrict the ability to add roommates, disable

the maintenance ticket portal, or define which contact persons are displayed to tenants.

1.2 Market Research

As part of the project initiation phase, an analysis of existing software solutions for real estate management was carried out, with a focus on the Czech market where the company operates. The market research has shown that there are multiple Czech software products for property management companies and landlords, such as Domsys [1], Sworp [2] or Starlit [3].

However, a key limitation of these platforms is that their mobile applications are typically designed exclusively for property owners or administrators, not for tenants. This gap reflects a broader pattern: many current systems optimize internal operations but fail to address the interaction between the realtors and their customers, the renters.

The mobile application developed in this project wants to address this exact gap. Unlike existing platforms, it enables real estate agents to fully onboard tenants digitally, without the need for repeated in-person meetings which can be time consuming. After a successful apartment viewing, the agent only needs to enter the tenant's email address into Microsoft Dynamics 365 Business Central, together with their first and last name (these are not mandatory for the onboarding process, but serve as an additional safety measure). From there, the mobile app takes over, guiding the tenant through all necessary steps: activating the account, providing personal data, verifying identity, signing contracts digitally, and having access to all necessary rental information. This significantly reduces administrative overhead and minimizes errors associated with manual data collection.

A further competitive advantage is the automated integration of the backend with the UbyPort system, a legal requirement in the Czech Republic for reporting foreign tenants to the authorities. Many real estate companies still handle this process manually or through loosely integrated systems. Meanwhile, the solution developed by On Point reports foreign renters automatically, reducing the administrative burden and helping to ensure the task is not forgotten.

1.3 Project Management

The development of the mobile app started around July in 2024. At first, there was no set deadline for the release but after a few weeks, around September, it was agreed that the first version of the app should go into production at the beginning of the year 2025. Therefore, we internally set the deadline to just before Christmas, to make sure everything could be tested and live by January. A second release was planned, but only after the first version was finished and deployed. The app was delivered to Google Play Store and Apple App Store on schedule by the end of December.

The team used Jira software for task management and communication throughout the whole project. Jira is an agile project management tool developed by Atlassian and is widely used in software development by teams to plan, track, release and support software [4]. A Software project was created in Jira to manage both the mobile app and backend development work. This unified setup ensured that all aspects of the application—from front-end features and API integration to backend logic—were tracked and coordinated in one centralized location. All the requirements were written up as Jira tickets where each represented a single feature or functionality and included a detailed description. Early on, a feasible timeline was put together in collaboration with the project leader. We tried to incorporate buffers for each phase, just in case something took longer than expected or plans had to change along the way.

To organize the workflow, a visual project management tool was used—a Kanban board with tickets in Jira. Originally developed in lean manufacturing and later adapted for software, Kanban encourages incremental improvement and helps teams avoid overload by focusing on completing tasks before starting new ones. It emphasizes limiting work in progress (WIP), visualizing workflow, and optimizing task flow to improve efficiency [5].

The board used in this project contained five basic columns:

- To Do: Features and tasks approved for development but not started.
- In Progress: Items that are actively being developed.

- Testing: Tasks that are undergoing quality control or awaiting feedback.
- Ready for Deployment: Completed and tested tasks that are in the queue for release.
- Done: Fully implemented and deployed items.

Bitbucket, another tool from the Atlassian ecosystem served as the version control system for the project. Jira's integration with Bitbucket enabled branches to be linked to tickets via naming conventions, ensuring easy traceability.

The development process itself was relatively straightforward: once a feature was completed, it was tested by a dedicated person. If something did not work as expected, or if there were any missing parts, notes were added directly in the Jira ticket. Features were revised and adjusted based on test results, followed by additional testing in an iterative cycle that continued until each ticket was completed and ready to be closed. The project leaned towards a lightweight Agile approach—iterative, flexible, and responsive to feedback.

The team was small and consisted of following people:

1. A backend developer who handled API development, adjusted backend logic, and made sure everything integrated well with our ERP system (Microsoft Dynamics 365 Business Central).
2. A sales representative who helped define the business side of the requirements and supported the testing from a customer's point of view.
3. The project leader, who was responsible for planning the features, setting priorities, and making sure everything was on track.
4. Marketing representative providing input on the UI choices and designing app icons.
5. Product Owner who took care of setting up App Store accounts and legal issues.
6. Myself, developing the mobile application and implementing features.

We had weekly check-in meetings where we went over progress, talked through any open issues, and shared feedback. It was also a

time when we revised the logic of the features or made small design decisions. On occasion, the CEO would join to see how things were progressing and offer input, especially from a product and business angle.

2 Technologies

This chapter provides an overview of the technologies used throughout the development of the project. It describes the main frameworks and tools that were selected, along with the reasoning behind their usage.

2.1 Multiplatform Development

Multiplatform mobile development [6] refers to building applications that run on multiple operating systems, most commonly Android and iOS, using a single shared codebase. This approach has become increasingly popular in recent years due to its efficiency and cost-effectiveness. In contrast, native development means building separate apps for each platform, using Swift or Objective-C for native iOS app and Kotlin or Java for native Android app. It offers the best performance and access to the full feature set of a particular device but at the cost of doubled effort and maintenance.

A multi-platform development approach was chosen mainly due to time and budget constraints. Since the product needed to be developed and delivered quickly, maintaining separate native codebases for iOS and Android would have been in this case inefficient and costly. Additionally, the application did not require high-performance graphics or complex native functionality, which made a multi-platform development a practical choice. By sharing the codebase across platforms, development time and effort were significantly lowered.

2.2 React Native

React Native (for general overview see [7] or [8]) is an open-source framework created by Meta (previously Facebook) that allows developers to build mobile apps using JavaScript and React. It supports multi-platform development, enabling a single codebase to run on both iOS and Android. React Native accomplishes this by rendering React primitives to native platform UI, meaning that the app uses the same native platform APIs other apps do [9]. It offers features

such as hot reloading—allowing to see changes without recompiling the entire application, and access to a broad ecosystem of third-party libraries.

The framework is backed by Meta, which actively maintains it since 2015 [9]. In addition to this support, React Native benefits from a large and active developer community that contributes to its ecosystem of libraries and tools [10]. Moreover, React Native is used by a wide range of well-known companies and high-traffic applications, including Facebook, Instagram, Amazon Appstore and Discord [11].

2.3 Other Multi-Platform Frameworks

React Native competes with several other multi-platform frameworks, most popular being Flutter, .NET MAUI (formerly known as Xamarin), Kotlin Multiplatform and Ionic [12].

- Flutter: developed by Google, is a popular multiplatform framework that uses the Dart programming language and its own high-performance rendering engine to create highly customizable user interfaces. Its code compiles to ARM or Intel machine code as well as JavaScript, enabling native-like performance [13].

- .NET MAUI: A Microsoft-owned framework used to develop apps across iOS, Android, macOS, and Windows abstracting them into one common architecture built on .NET with C# codebase [14]. It is the evolution of Xamarin.Forms (or just Xamarin) whose support ended in May 2024 [15].

- Kotlin Multi-platform: An open-source framework by JetBrains that enables sharing business logic and (using Compose Multiplatform) UI code across Android, iOS, web, and desktop [16].

- Ionic: A web-based framework that allows developers to build mobile apps using standard web technologies such as HTML, CSS, and JavaScript [17].

The decision regarding which framework to use for the mobile application was left open, providing the opportunity to select the

most appropriate technology based on project needs and prior experience. The deciding factor was previous experience with React, which made the transition to mobile app development much smoother. Another factor was a large and active developer community, extensive documentation, and a mature ecosystem of third-party libraries and tools.

2.4 Expo

Expo [18] is an open source framework built around React Native that simplifies the development and deployment of mobile applications. It provides a full ecosystem of tools and services that make it easier to develop, build, and submit React Native applications.

Expo was chosen for this project because it does not require extensive native development experience and offers a good support to develop fast through its managed workflow and high-quality built-in libraries. In addition, React Native recommends using this framework as they have found that many developers benefit from its features [19]. However, at the time of deciding which framework to use, there were still many articles and discussions online warning that certain features, particularly those requiring native modules not included in the Expo SDK might require "ejecting". Ejecting meant leaving Expo's managed workflow and generating full native Android and iOS projects, requiring manual setup of native code, dependencies, and build tools. Knowing from the beginning that the application would require advanced native features, such as rendering PDF documents, starting with the bare React Native workflow might have been a better option to avoid the need for this transition. Since 2022, however, this concern has been resolved, as the `expo eject` command was deprecated and replaced by prebuilds [20].

2.4.1 Expo Go

Expo Go is a mobile application that enables testing of apps on a real device without requiring a build process. All that is needed is to download the Expo Go App from the App Store or Play Store and scan

a QR code, moreover there is no need for Apple Developer account nor Xcode or Android Studio [21].

However it also comes with limitations: it cannot execute any native code outside Expo's managed workflow (already built into Expo Go), including any native modules and libraries requiring direct native dependencies (e.g., `react-native-pdf`). For these cases, developers must create a development build (a precompiled binary with native code support). This is because Expo Go uses a generic prebuilt runtime, while development builds embed app's specific native dependencies. Even though it has these drawbacks, it presents a simple entry point to familiarize with the environment and later switch to development builds.

2.4.2 Expo SDK

The Expo Software Development Kit (SDK) is part of the Expo ecosystem providing a collection of pre-built libraries and modules that cover many functionalities—simplifying the access to device features like the camera, gallery, push notifications, and GPS location without the need to write native code [22].

In our project, the following Expo SDK modules were used:

1. `expo-localization` to detect the user's locale and language
2. `expo-image-picker` to allow users to select images from their device's gallery or camera
3. `expo-secure-store` for securely storing sensitive data on the device
4. `expo-sharing` for sharing content with other apps installed on the device
5. `expo-linking` for interactions with other installed apps using deep links, namely email client and phone

2.4.3 Expo Application Services (EAS)

Expo Application Services (EAS) is a suite of cloud-based services designed to enhance the development, building, and deployment of React Native applications. It provides tools for building and managing apps more efficiently, without requiring developers to maintain complex native build environments locally [23]. Traditionally, iOS app

deployment requires Xcode and macOS hardware for builds, code signing, and App Store submissions. EAS theoretically eliminates this dependency by handling these processes remotely through its managed infrastructure. Developers can build their project by using EAS Build, which then generates production-ready iOS binaries without requiring direct access to Apple's native toolchain. However, for debugging tasks associated with hardware-specific features, macOS is still needed. To automatically submit apps to the Apple App Store and Google Play Store, `eas submit` command can be used.

2.5 TypeScript

TypeScript, a strongly typed superset of JavaScript, maintains full compatibility with existing JavaScript code while extending the language with additional features including modules, classes, interfaces, and a static type system [24].[1] It enhances code reliability by introducing static typing, enabling developers to catch errors during compilation rather than at runtime. This results in more maintainable and robust code, particularly beneficial for large-scale applications. New React Native projects target TypeScript by default. As such, when choosing the language for this project, TypeScript was an easy choice [25].

2.6 TanStack Query

Because the application relies heavily on numerous API calls to fetch and update data, TanStack Query [26] was used to simplify and optimize server-state management. TanStack Query (formerly React Query) is a data-fetching library that enhances state management by efficiently handling server-state synchronization, caching, and background updates. It simplifies complex tasks such as data fetching, pagination, and optimistic updates while reducing boilerplate code. TanStack Query is fully compatible with React Native by default; however, the devtools feature is currently limited to React DOM and not available for React Native.

1. Some of these (such as classes and modules) were later also introduced directly into JavaScript.

2.7 Internationalization

One of the requirements of the app was to support multiple languages. During planning, Czech and English were established as target languages for the first release, but the option to easily extend this list in the future must be provided (more on this in Section 3.2). To meet this need, the project utilized `react-i18next`, a robust internationalization framework tailored for React and React Native applications which is based on `i18next` [27]. It offers a flexible, hook-based API and supports features such as dynamic language switching, namespace separation, and resource loading from external JSON files. To automatically detect and apply the user's device language, `react-i18next` was used alongside with Expo Localization, a module that retrieves the system's locale settings at runtime.

3 Specification

This chapter presents the key specifications of the application, including both functional and non-functional requirements. All requirements listed below were initially defined by the project leader in collaboration with a sales representative who had prior experience in the rental housing industry. Throughout development, these requirements were continuously re-evaluated based on changing business priorities. For example, the ability to use a payment gateway to initiate payments right through the app was initially added to the backlog as a "nice-to-have" feature. However, payment gateways typically charge a percentage-based transaction fee, which can be substantial for higher payments like rent. Given this, and after considering the cost-benefit trade-off, it was ultimately decided to not implement this feature. From the described requirements, *Requests and Faults* and *FAQ* sections were planned for the second major release.

3.1 Functional Requirements

The mobile application is designed to support tenants in managing their rental experience easily. The app must enable the users to perform the following activities:

1. **Authorization**
   (a) The tenants are able to activate their account with the email they provided their real estate agent with. After entering their email, they receive a one-time verification code lasting 15 minutes that allows them to set their password.
   (b) An alternative login option using Google OAuth 2.0 is available.
2. **User Type**
   (a) The app must differentiate between two types of users: individual persons and companies in the following way:
      i. The personal information form displays different fields based on whether the user is a person or a company, and the title of this section is Company Information

ii. The prompt for uploading document scans varies accordingly, requesting company documents for companies and personal identity documents for individuals.

3. **Contract Management**
   (a) View a list of all contracts, including active, expired, and future-dated ones.
   (b) View a list of all amendments for each contract along with their key information.
   (c) Contracts and amendments must be displayed directly as PDFs within the app using an embedded PDF viewer, ensuring users can read the documents without needing to leave the application.
   (d) Sign contracts and amendments digitally, either via direct in-app approval or through an integration with a document signing DocuSign service.
   (e) Manage roommates:
      i. Add or remove roommates.
      ii. Edit roommate personal information and define the period of their occupancy.
      iii. Prefill data for previously added roommates to make the process easier.
   (f) Terminate the contract (if permitted), choosing a termination date within the contract limits.
   (g) Submit a deposit refund request.
   (h) View the house rules.

4. **Personal Information**
   (a) Users must be able to fill in and edit their personal details, such as full name, date of birth, nationality, address, and bank account information. The nationality and country picker offers options taken from the backend.
   (b) Validation rules should enforce format, required fields, and data length where needed.
   (c) Users must be able to upload identity documents (ID/passport) or company documents via:
      i. device gallery selection
      ii. direct camera capture

(d) The form must remain accessible after initial onboarding from the user page in case the user needs to make updates later.

5. **Payments**
   (a) Users can view their complete payment history, including both posted (paid) and pending (unpaid) items.
   (b) Payment records can be filtered by:
      i. associated contract
      ii. payment status (paid/unpaid)
   (c) Each payment item must display relevant details such as amount and due date and show the rest of relevant information on an expandable card.
   (d) Payment statuses must be visually indicated using color-coded labels.

6. **Requests and Faults**
   (a) Users must be able to submit maintenance requests directly through the app, describing the issue and optionally attaching one or more photos.
   (b) Tenants can view the current status of submitted maintenance requests (e.g., open, in progress, resolved).
   (c) The app must support direct communication between the user and the assigned maintenance contact.

7. **Localization**
   (a) The app must be fully localized in both English and Czech with possibility to add another language in the future.
   (b) On first launch, the app should default to the device's system language (if supported).
   (c) Users must be able to manually switch languages from a clearly visible setting, accessible from the login page and user menu.
   (d) All date, time, and number formats should follow the selected locale's conventions.
   (e) Any data fetched from the backend that contains language-specific content (such as contact role captions, document labels, or status descriptions) must be requested or displayed in the currently selected app language.

3.2 Non-functional Requirements

1. Passwords must adhere to a minimum complexity requirement (8+ characters, including uppercase, lowercase, and a number).
2. Administrators must be able to disable tenant access to the app (maintenance mode).
3. The app must support Android devices with version 8 or higher.
4. The app must support iOS devices with version 15 or higher.
5. The app should be intuitive and easy to use, with clear navigation. It should be simple for users to complete common tasks (such as viewing contracts or editing information) without requiring help.
6. Updates to the application must be manageable by administrators through backend, allowing them to enforce a minimum required version (forced updates).
7. All communication between the app and the backend must be secured using HTTPS. Sensitive data, such as authentication tokens should be stored securely locally using encrypted storage mechanisms.

4 Architecture

This chapter presents the software architecture underlying the mobile application's connection to its backend services. The mobile app uses Microsoft Dynamics 365 Business Central as the core backend system and accesses it via REST API through an intermediary C# proxy layer which acts as a bridge between the mobile app and the ERP system.

4.1 Microsoft Dynamics 365 Business Central

Microsoft Dynamics 365 Business Central is a cloud-based and on-premise ERP solution that is mostly suitable for small and medium-sized businesses (see Figure 4.1). Enterprise resource planning systems are enterprise-wide information systems that integrate and automate key business processes across various functional areas of an organization, such finance, HR, manufacturing, supply chain, sales, and procurement [28]. By consolidating data from various departments, ERP systems ensure data integrity with a single source of truth, eliminate data duplication, and enable seamless information flow across the organization [29].

The decision to use Business Central as the backend system for this project comes from the company's domain alignment. As the company's domain is centered on the functionalities provided by Business Central, the backend logic was developed within this environment to maintain consistency and take advantage of domain-specific expertise. This ERP system can be customized to meet customer's needs and extend its functionality by creating new tables and pages, modifying existing objects or integrating with external systems through web services and APIs. This is done using AL (Application Language)—a programming language that is used for manipulating data such as retrieving, inserting, and modifying records in a Business Central database [30]. To enable the mobile application to communicate with Business Central, API pages are created and exposed. API pages are specialized pages in Business Central that are designed to be consumed by external applications via RESTful APIs [31].

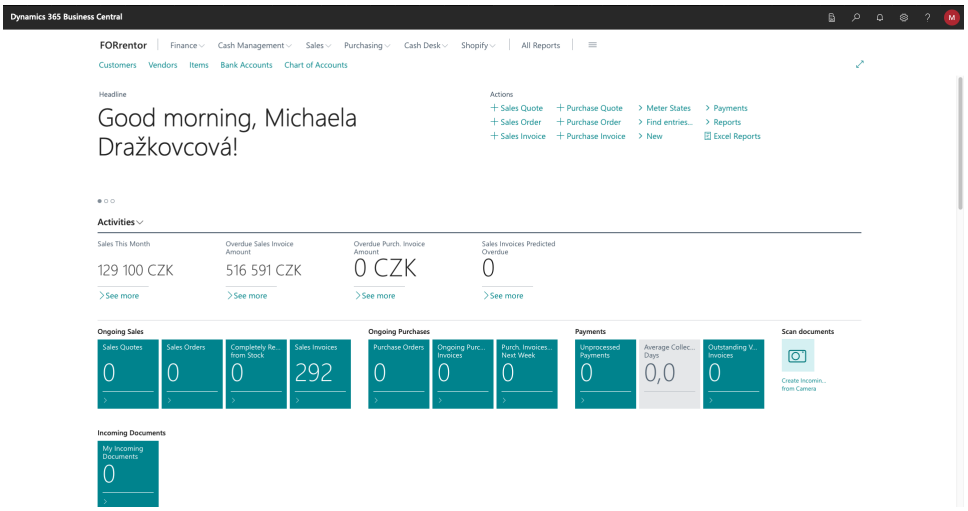


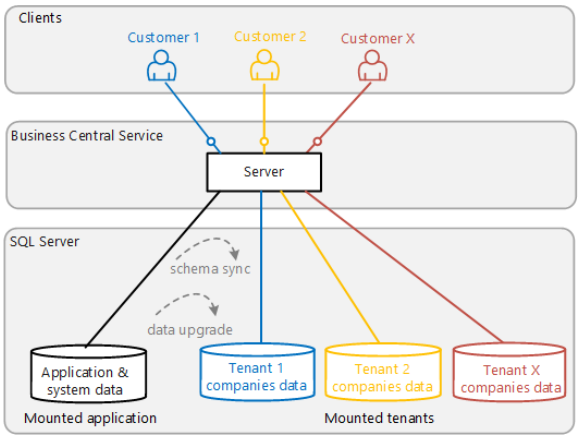
Figure 4.1: Microsoft Dynamics 365 Business Central

4.2 Multi-tenant Deployment Architecture

Microsoft Dynamics 365 Business Central supports a multitenant deployment architecture, allowing multiple customers (tenants) to share a centrally maintained application while keeping their business data isolated in separate databases. In this model, the application code is stored in a dedicated application database, and each tenant's data resides in its own business database, enabling centralized maintenance without affecting individual tenants (see Figure 4.2). This setup simplifies onboarding of new customers and allows updates to be rolled out quickly with minimal downtime [32].

4.3 The C# Proxy Layer

Even though Business Central serves as a robust backend system, exposing it directly to the mobile application could raise concerns regarding security, complexity, and maintenance. To address these issues, a C# proxy layer has been implemented by a colleague as an intermediary between the mobile app and Business Central. This proxy layer provides abstraction and enforces security by managing authentication and authorization. The layer also implements rate limiting.



[32]

Figure 4.2: Multi-tenant architecture

C# was chosen as the implementation language for the proxy layer due to its good integration with Microsoft technologies, including Business Central and Azure. Furthermore, the .NET ecosystem offers a set of libraries for building APIs, handling authentication and managing data.

The mobile application interacts with the C# proxy layer using RESTful API calls, each secured by an authorization header. After the user is successfully authenticated, the mobile app receives an access token for next API requests. Every API request from the mobile app then includes an authorization header containing the access token:

```
Authorization: Bearer <access_token>
```

The proxy validates the access token to ensure the request is authorized. If the token is invalid or expired, an error response (e.g., 401 Unauthorized) is returned. For valid tokens, the proxy forwards the request to Business Central. Once Business Central processes the request, the proxy layer retrieves the data, transforms it if necessary (for example, removing redundant or sensitive fields), and returns the response to the mobile application.

5 Design and Functionality

This chapter provides an overview of the key design decisions and the core functionality of the mobile application. The app's features are explained section by section along with relevant screenshots.

5.1 Design Choices

At the beginning of the development process, the main priority was to create a functional and reliable application. Because of the relatively short deadline for the app's release, most of the focus was placed on delivering working app, and visual design planning took a secondary role in the early stages of the project.

The planning of the app design started at the initial meetings where we outlined the basic structure and functionality of the individual screens—mainly the tabs and their content. This consisted of sketching the basic layouts of the main screens on a whiteboard and focusing only on putting the main elements together on each screen. These informal low-fidelity sketches helped us to agree on the general structure of the application, but they were not detailed. Because of this, many design decisions were made incrementally during development, as the particular screens were being developed. This flexible approach allowed us to move quickly, but also led to occasional inconsistencies and future rework. Expo supported this approach well as it is a capable prototyping tool, with hot reloading making it relatively easy to experiment with different layouts and UI components. In retrospect, the effort and planning required for good UI/UX design might have been underestimated at first. A more detailed design phase early on could have saved time in the long run and helped avoid design-related issues during later stages.

Another area that was not prioritized during the initial development phase was tablet-specific optimization. This decision was based on early assumptions that most users would access the app via mobile phones, which was later confirmed by usage statistics available in app store analytics (see Chapter 6.5). Given this, tablet-specific UI was not considered essential for the release. While the current layout remains functional on tablets, it is not fully optimized for larger screens. This

may be revisited in the future if user behavior or business requirements shift in that direction. However, right now, the percentage of users who use tablets is negligible.

To ensure a user-friendly experience, the design of the app focused on several key principles. Because the app does not have a narrowly defined target demographic, we wanted to make the app accessible to users of all ages. With this in mind, the focus was on consistency, simplicity, clear hierarchy and intuitive navigation to ensure that the app can be used by everyone.

One of the key goals was to maintain visual consistency throughout the whole app. Close attention was paid to keeping things unified by using consistent padding, matching the rounding of buttons, cards, and input fields, and following shared layout patterns. These small details made a more polished and cohesive user experience. Moreover, to keep things visually coherent, a single font was used throughout the app, consistent with other communication materials of the company. Headings, subheadings, and regular text were distinguished using different font sizes and weight.

Another important aspect of the design process was simplicity. The goal was to make the user interface as straightforward as possible, with clean screens to avoid overwhelming the user. After logging into the app, the UI follows a single color scheme. Wherever possible, icons were used to help clarify functionality and make the interface more intuitive.

In terms of user flow, the focus was on creating an intuitive navigation structure that allowed users to quickly find the actions or information they needed. We implemented multiple access points for common actions, meaning that users could reach the same feature through different routes. For example, to navigate to the screen for signing a contract, user can click on the action from the main page where it is highlighted as *To Do* or get there by clicking on the specific contract from the *Contracts* page. This redundancy tries to ensure that users will not get lost or frustrated if they cannot find an action.

5.2 App Screens

5.2.1 Login

Users are able to activate accounts in the mobile app using the email address they provided to their real estate agent. When signing into the app for the first time, they are prompted to enter their email address and then wait for a verification email containing a six-digit code, which then needs to be entered into the app. After a successful verification, a screen for setting a password is displayed, see Figure 5.1.

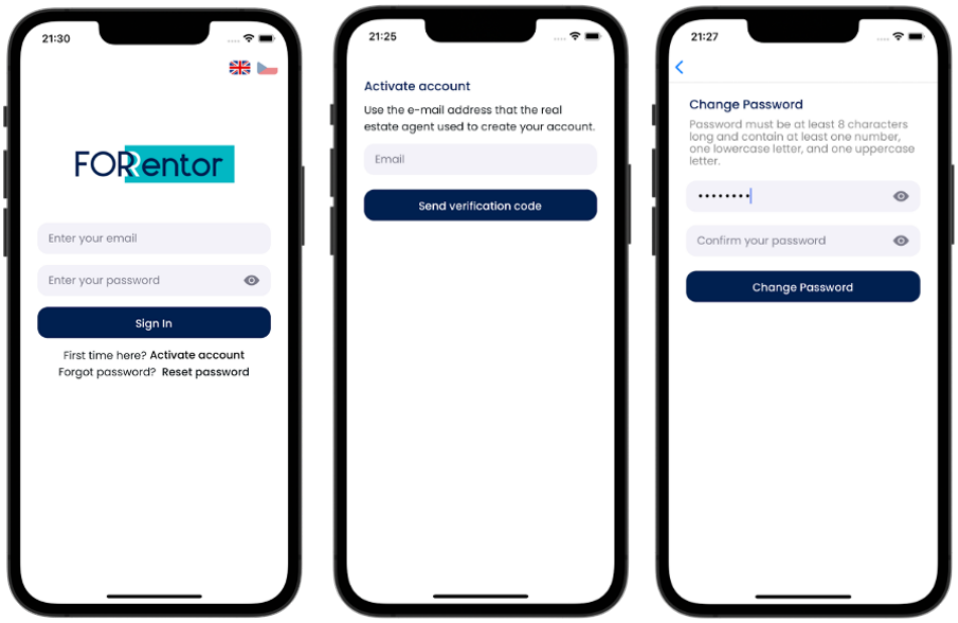


Figure 5.1: Login pages

The other option to activate the account is to sign in using Google OAuth. However, this option is only available on Android as we encountered problems with app review for iOS. Here, the app got rejected for not following App Store's guidelines—more on this in Section 6.3. For security reasons, the app does not indicate whether an entered email address exists in the system. This is to prevent the possibility of someone using the app to discover which email addresses are associated with tenant accounts. To reduce user confusion caused by this limitation, the error message again mentions to users that they

need to use the same email address they provided to their real estate agent.

5.2.2 App Tabs

Logged-in users can navigate between five main tabs:

1. Home (Section 5.2.3)
2. Payments (Section 5.2.5)
3. Contracts (Section 5.2.4)
4. Utilities (Section 5.2.6)
5. User (Section 5.2.7)

5.2.3 Home Page

After a successful login, users are directed to the home page, which serves as the main dashboard for navigating important actions within the app. At the top of the screen, the company logo is displayed for branding. Below that is the *To Do* section where users can find actions that are supposed to get their attention, see Figure 5.2, left. This section is a dynamic task list and helps with onboarding new users to clearly navigate them through the necessary steps.

For new users, especially those who have not yet signed any rental contract, the first required action is to fill in personal information. As shown in Figure 5.2, the form must be filled out before user can proceed to the next step. Until then, all subsequent actions (including signing the contract) remain disabled.

The personal information form allows users to fill in and update their details, such as date of birth, nationality, address and payment details. Some fields use custom input types, for example, a calendar picker for birth dates that supports different date formats based on localization, and a dropdown menu for selecting nationality and country based on options taken from backend. Most of the fields are mandatory and validated to check the correct format and length limits. Also, the form differs based on the type of user. For example, company users are shown fields like registration number and company name, while individuals are asked for birth certificate number and their first and last name. This form is also accessible from the *User* section of the app (Section 5.2.7), so users can make changes to these data if needed.

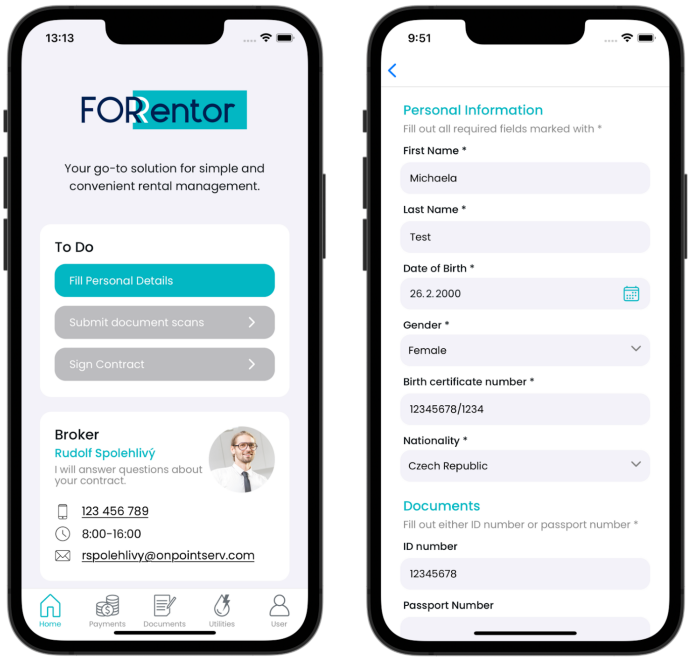


Figure 5.2: Home Page and Personal Info form

If the complex setting in the ERP system requires an ID/passport scan, the next step for the user is to upload these documents before signing the contract. The descriptions of what needs to be uploaded differ depending on whether the tenant is a person or a company. Users can either take a photo directly or select multiple photos from their library. The images are then sent to the backend in base64 format. The form for uploading document scans is shown in the Figure 5.3.

Below the *To Do* section, the user can find contact information for important people. Each complex can have four contacts in Business Central: *Estate Agent*, *Administration*, *Maintenance* and *Property Support*. Each of these contacts has a card with a phone number, email address, office hours, job description and an optional photo. The cards include a clickable contact feature, allowing users to tap on phone numbers to initiate a call, or on email addresses to open the default email client. This functionality is implemented using React Native's Linking API. In cases where a user has contracts in multiple distinct housing complexes, a complex filter is shown to allow toggling between them.

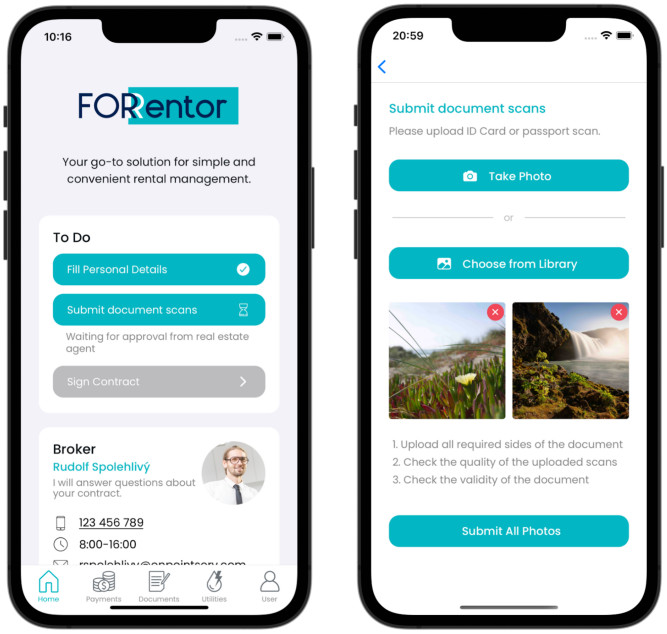


Figure 5.3: Submitting document scans

5.2.4 Contracts

Another section in the app is Contracts where users can view their current and past contracts, along with any amendments and additional documents associated with it, see Figure 5.4.

Each contract is displayed as a card containing key information and a number of relevant actions available to the user—terminating the contract, requesting a refund of the deposit, managing roommates, viewing house rules and submitting a fault report or other service request. On each contract card, a colored label indicating the status of the contract can be found—from *Valid* to *Waiting for signature* and other options. After clicking the whole card or the underlined contract number, the PDF of the contract is displayed with the option to share it.

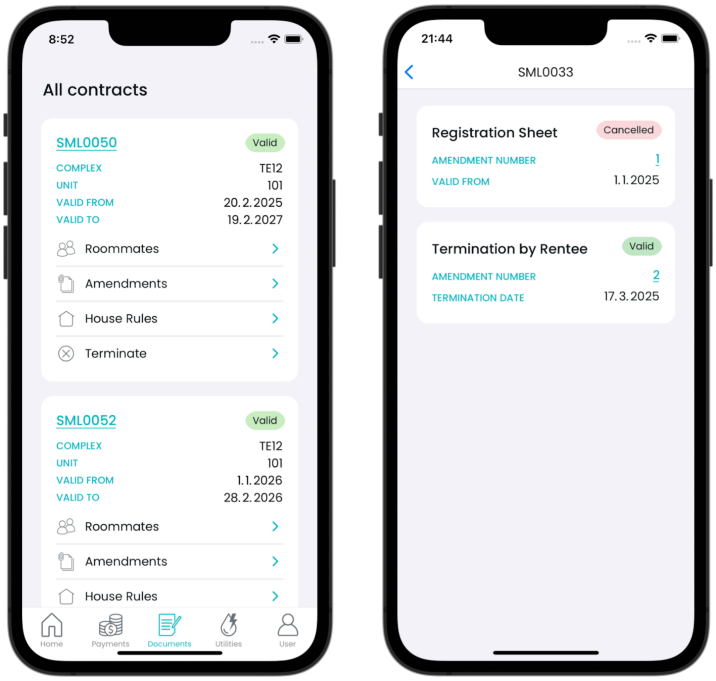


Figure 5.4: Contracts and their amendments

Document Signing

When a contract or an amendment is ready to be signed, the user can see this on the home page within the *To Do* section and also when viewing the PDF of the document. However, signing is only allowed if the tenant has already completed their personal details in the app. Additionally, if the complex has a setting that requires an ID/passport photo in Business Central, the user will be prompted to upload scans of these documents after clicking on the action from the home page or from the contract itself, as shown in the Figure 5.5 on the left screen. This helps users navigate between pages easily.

Once the tenant has viewed the contract awaiting signature, there are few check-marks with click-throughs to check their personal details and their roommates, and edit the details if necessary before signing the contract (see Figure 5.5, right screen). The signing process is integrated with Business Central and can happen in two ways: either through DocuSign, a legally recognized e-signature service,

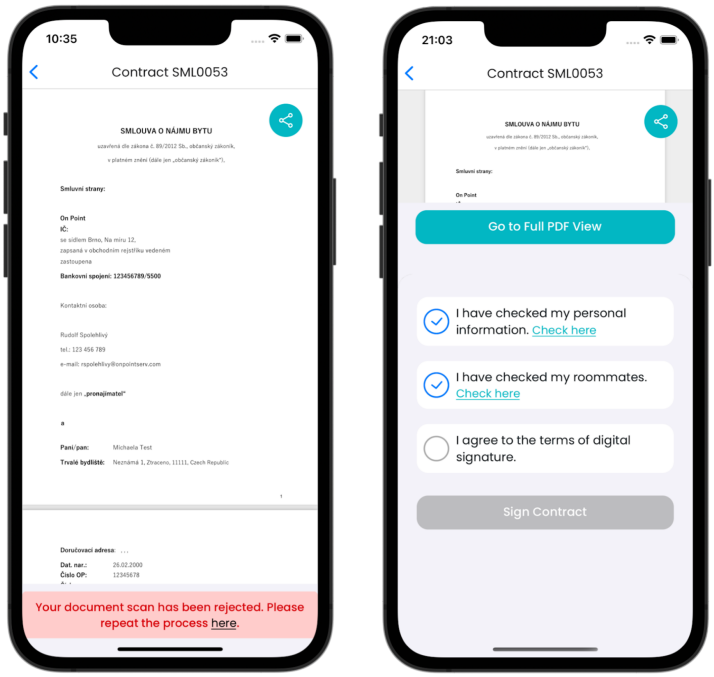


Figure 5.5: Contract Signing

or by simply clicking a *Sign Contract* button directly within the app, depending on the configuration for the specific contract. DocuSign e-signature is a document signing software that is legally enforceable for most businesses and lets you sign documents online [33].

Termination of Contract

The user is able to terminate the contract in the application if the conditions of the contract allow it. On the contracts page, the user can click on the action *Terminate* (see Figure 5.6) which redirects them to the termination form. The date can be chosen starting from the current day up to any date until the end of the contract. Whenever the date of termination is changed, the end date is calculated in the backend according to the set contract notice period and immediately shown below the selected date in the calendar. The resulting date must fall within the contract duration and leave enough time for the notice period. If not, a warning is displayed and termination is not possible.

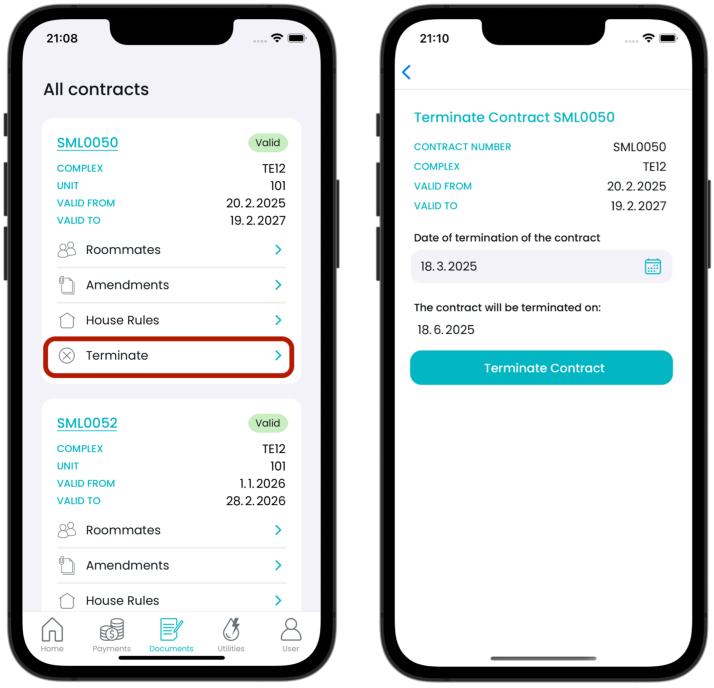


Figure 5.6: Contract Termination

Refund of Deposit

The user is able to request a refund of the security deposit if they meet the requirements. After clicking the action on the contracts page, they are redirected to a form with pre-filled bank details taken from their personal information (see Figure 5.7). They are able to change the bank details and after successfully requesting the deposit back, an amendment is created.

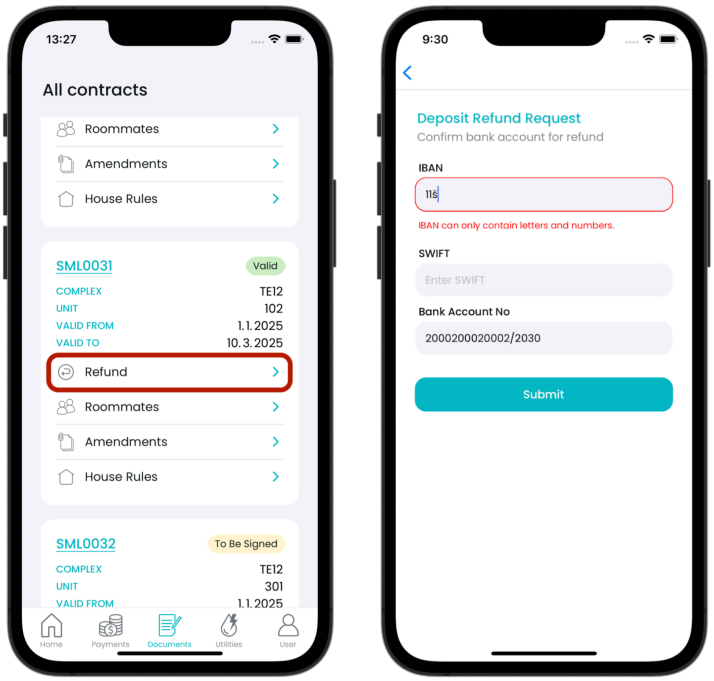


Figure 5.7: Refund of Deposit

Roommates

Another contract-related action available in the app is managing roommates. Users can view the list of current roommates associated with their contract and add new ones as needed, see Figure 5.8. If a user has previously added roommates in the past (it does not matter in which contract), there is an option to pre-fill the personal information of the previous roommate, making it quicker to add the same person again. The user only needs to specify the new dates for the roommate's stay. Roommate stay period can be modified under the following conditions: if the stay has not started yet, the user can update the full date range or remove the roommate entirely. If the stay has already begun, it is still possible to adjust the end date, but not the starting date or remove the roommate.

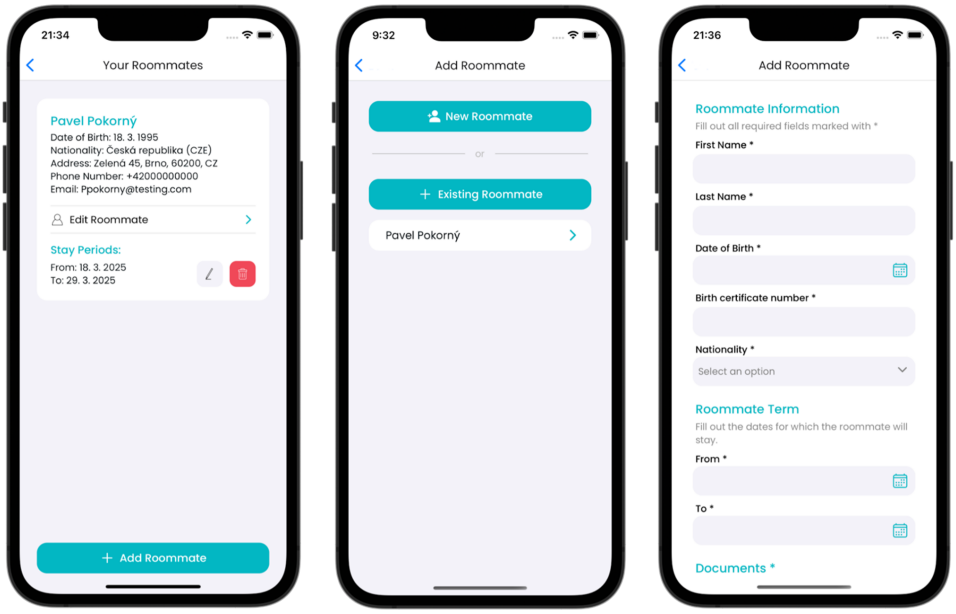


Figure 5.8: Roommates

Faults and Requests

One of the key features introduced in the second release of the mobile application was *Fault and Request Reporting*. This functionality was developed to improve communication between tenants and property management by offering a simple and structured way to report issues within their apartments, such as water leaks, non-functional Wi-Fi, or to submit various types of service requests, directly through the mobile application. Previously, tenants reported problems by phone or email, which was difficult to track and often disrupted staff.

The request form allows tenants to select a category (e.g., plumbing, electricity, heating), write a description of the problem, choose a priority (low, normal, high), and optionally attach photos from the gallery or by directly taking one. These categories are editable in the ERP system and can be customized by the company to fit their needs. Once a request is submitted, it is displayed in the ERP system to the appropriate staff or service provider. Tenants can view the current status of each request in real time, which is shown as a label on the request card (see Figure 5.9). Tenants can also view the history of their

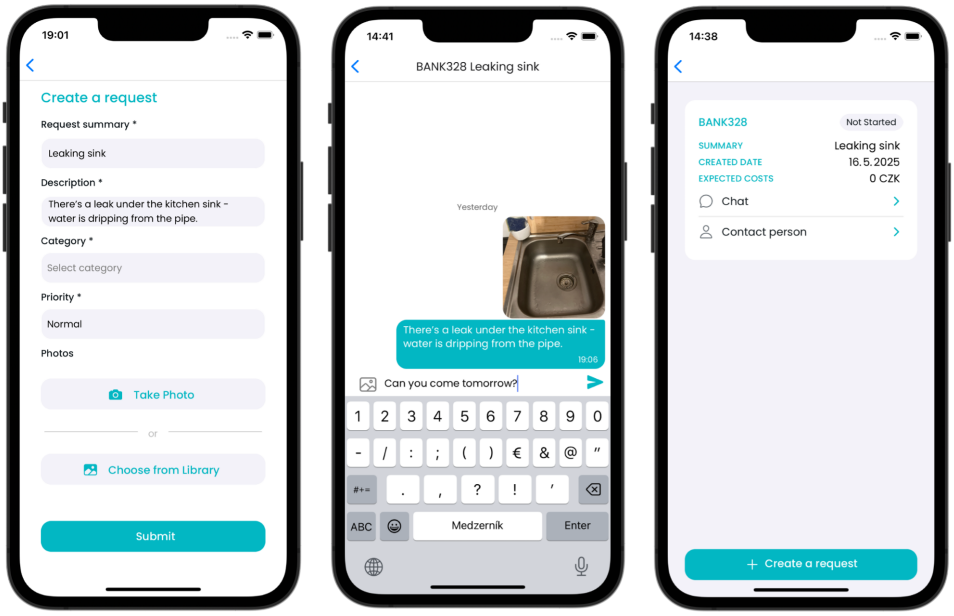


Figure 5.9: Faults and Requests

requests for each contract. However, not all contracts include access to the fault reporting feature; if it is disabled for a given contract, the option is not shown in the app.

After submitting a request, tenants see a request card with two quick-access buttons, one for opening a chat and another for viewing the contact details of the assigned person in case they need to call or email them. The chat feature allows tenants to chat with the assigned person (e.g., a plumber or technician) directly and supports sending images as well as receiving service documents like invoices in PDF, XLS, DOC, or TXT formats.

5.2.5 Payments

The tenant can view all their payments on one of the main tabs, see Figure 5.10. Payments can be filtered by contract number or their payment status (*unpaid* and *paid*). Each card shows the title of the transaction, the due date and the balance due, with more information available by clicking the card. Overdue receivables are marked in red and those due in the next 7 days are marked in orange.

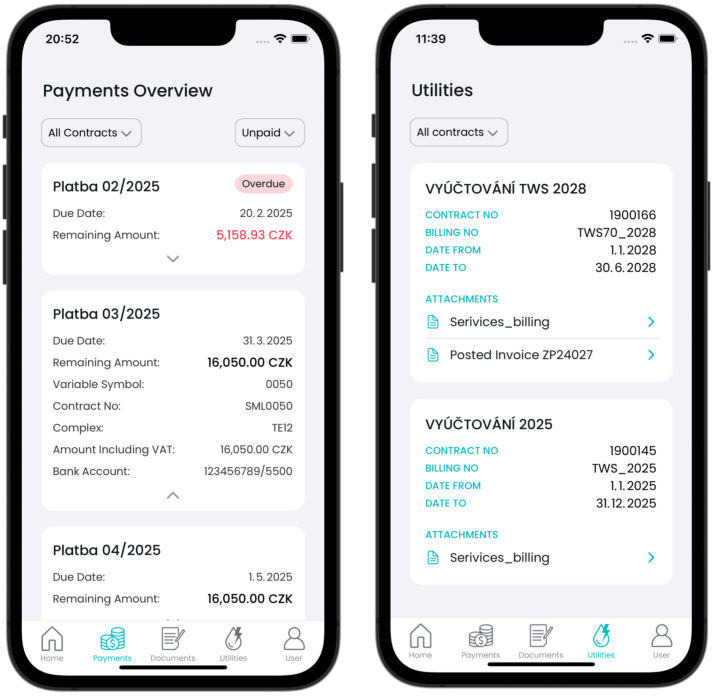


Figure 5.10: Payments and Utilities

5.2.6 Utilities

The *Utilities* section shows available utility charges tied to the user's rental contracts—things like electricity, water, and other services. The bills are listed by billing period, so users can see what was charged and when. If the user has more than one contract, they can also filter bills based on a specific contract. Each billing entry can have one or more PDF attachments, usually with detailed breakdowns of the charges. The *Utilities* page is shown in Figure 5.10.

5.2.7 User

The User section of the app is where users can manage their personal settings and account-related options.

It includes access to personal information, language preferences, and support resources. The screen is simple and organized, with the following options:

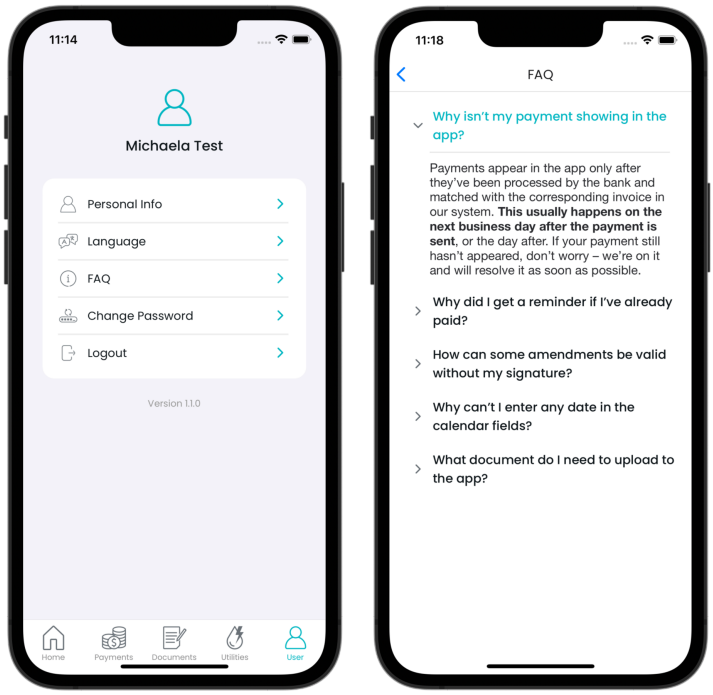


Figure 5.11: User Page

1. *Personal Information* - This redirects to the same form available during onboarding (mentioned in Section 5.2.3), allowing users to update or correct their personal details at any time.
2. *Language* - Users can select the app's language, currently from two options: Czech and English. This selection is stored on the device and used across the app to show content in the preferred language.
3. *FAQ* - A shortcut to a section with frequently asked questions that helps users better understand how to use the app and find answers to common issues without needing to contact support, see Figure 5.11. The FAQ content is managed directly in the back-end system (Business Central), where administrators can update the content at any time. They can apply bold text, adjust font sizes, or use color styling as needed, since the content supports basic HTML formatting. This HTML is then rendered in the app using a third-party library, which allows the text to appear as

intended on the screen without the need for hardcoded content in the mobile application.
4. *Change Password* - Allows users to set a new password.
5. *Logout* - This option logs the user out of the application and clears their session data from the device.

Finally, this menu also displays the current version of the app.

6 Implementation and Deployment

This chapter details the implementation, deployment, and post-launch monitoring of the mobile application. It covers the project structure, testing approach, app store submission challenges, user feedback, and future plans. Additionally, app store statistics are discussed.

6.1 Project Structure

Navigation in the application is handled using Expo Router, a routing library for React Native and web applications. It is built on top of React Navigation and enables developers to manage navigation through a file-based routing system and provides native navigation components [18]. The routing system uses route groups, represented by parentheses in folder names such as `(auth)` and `(tabs)`. Each group can contain a `_layout.tsx` file, which defines a shared layout and navigation container for the screens within that group. For instance:

- `(auth)` contains authentication-related screens like login and password recovery.
- `(tabs)` includes the main application views accessible via a tab navigator.

These groups help organize screens logically without affecting the actual route paths shown to the user. Expo Router also supports dynamic routes, which allow navigation to parameterized paths by using square bracket notation in the file names (e.g., `[contractNo].tsx` or `[amendmentNo].tsx`). For example, navigating to a specific contract is done with:

```
router.push(`/contracts/${contractNo}`)
```

All pages have a URL path that corresponds to the file's location in the `app` directory, supporting "universal deep-linking" [34]. This eliminates the need for manual route definitions and helps to keep the routing structure scalable and consistent with the file system.

The project is organized in a domain-driven manner. Below is the application's file structure including the screens:

```
app/
    (auth)/
        _layout.tsx
        sign-in.tsx
        sign-up.tsx
        ...
    (tabs)/
        _layout.tsx
        documents.tsx
        home.tsx
        payments.tsx
        user.tsx
        utilities.tsx
    contracts/
        [contractNo]/
            amendments/
                [amendmentNo].tsx
                index.tsx
            house-rules/
            refund/
            roommates/
                [roommateNo]/
                    edit-interval.tsx
                    edit-roommate.tsx
                add-roommate.tsx
                index.tsx
            ...
    _layout.tsx
    index.tsx
    language.tsx
    personalInfo.tsx
    ...
```

The following is the rest of the project directory structure, organized into modules to separate concerns and maintain scalability:

```
assets/
    images/
    fonts/
components/
    cards/
    forms/
    ...
context/
    GlobalProvider.js
    NetworkErrorContext.js
i18n/
hooks/
    useContracts.ts
    useRoommates.ts
    ...
service/
    authAPI.ts
    businessCentralAPI.ts
types/
app.json
eas.json
```

The assets folder contains images and fonts used throughout the app, mainly icons. The `components` folder holds various React components, with subfolders for cards, forms, and reusable components like `ThemedText`. The context folder contains context providers for global state management and network error handling. The `hooks` folder includes custom hooks like `useContracts` and `useRoommates` using Tanstack Query library. The `service` folder contains APIs for authentication and interactions with Business Central, however, this folder is removed from the attached codebase. Finally, `app.json` is the Expo configuration file that contains the configuration settings for the app and the `eas.json` file is used to configure build and submission profiles for an Expo app.

6.2 Testing and Quality Assurance

Due to limited budget and time, no automated tests were implemented. Instead, quality assurance relied on manual testing, done by two testers. The testers worked through a range of scenarios covering all core functionality, such as authentication, contract signing, editing personal information, roommate creation, payments, and document viewing. In total, approximately 20 test scenarios were covered, including typical user flows as well as edge cases such as invalid input validation or expired verification codes. Testers worked directly with the ERP system (Business Central) to verify that the correct data was displayed in the app and that changes made in the backend were properly reflected in the app.

The app was developed mainly on iOS simulator and the thorough testing of different scenarios was done using an Android `.apk` file. This file could be installed on a physical device after granting the appropriate permissions. The `.apk` file was built with the command:

```
eas build --profile preview --platform android --local
```

Expo's `eas.json` configuration allows for multiple build profiles (e.g., development, preview, production) which makes it easy to switch environments without manual reconfiguration. After this testing was complete, the iOS version was uploaded to TestFlight. Here, key features which could present potential issues, such as photo uploads and keyboard support, were tested before sending it for review.

Before the initial release, the app also went through a language review conducted by the marketing team. Their main focus was on making sure the Czech texts had a consistent tone and used appropriate language when addressing users.

6.3 App Stores Submission

In October 2025, accounts were created on both App Store Connect and Google Play Console, since it is known that getting a company account approved can take time, especially with Apple. This proved to be the right move, as it took over a month to complete the approval process on the App Store Connect due to issues with the verification process.

The iOS version of the app was built using:

```
eas build --platform ios
```

Expo's build service was used to generate a release binary, which was then delivered to App Store Connect. The free Expo plan includes a limited number of builds per month. As of now, this is 30 (up to 15 for iOS) which was enough for our purposes. Furthermore, there was always an option to build the app locally if needed.

The Android version was prepared for release using a local build command:

```
eas build --platform android --local
```

This produced an `.aab` (Android App Bundle) file, which was uploaded manually to the Google Play Console. Because Google OAuth authentication is tied to the app's signing key configured in the Google Cloud Console, a production build was briefly tested to ensure that Google Sign-In functioned correctly before submitting the app for review.

Before submitting the application, both app stores require essential metadata such as app name, description, privacy policy, and contact details to be filled in. Additionally, each new release includes version-specific release notes and updated screenshots if necessary. Initially, we used simple in-app screenshots taken directly. However, these were later replaced with marketing-enhanced images that featured the screenshots inside device frames with captions and background styling. These made the app presentation look more professional and visually appealing.

At the end of November, a first version of the app (not yet final) was uploaded to both platforms to test the publishing process. On the Google Play Store, the app was accepted without any issues, even though their review process usually takes longer. In our case, reviews took between 4 and 6 days in comparison to iOS which was usually completed within 24 hours. On iOS, however, the app was initially rejected for two reasons:

1. Missing account deletion option: According to Apple's guidelines, if an app allows users to create accounts, it must also allow them to delete those accounts [35]. Our app does not support account creation directly—user accounts are created by real estate

agents in Business Central, not by the users themselves. However, this was not clearly communicated at first. To fix this, we added a message during the activation process to explain that users need to log in using the credentials provided by their real estate agency. This clarified that the app complies with Apple's policy.

2. Google Sign-In on Apple devices: Apple requires apps using third-party login services (like Google Sign-In, Facebook etc) to also offer a privacy-focused alternative that limits data collection and allows users to hide their email address [35]. In practice, that alternative is Sign in with Apple. Our app originally allowed Google login on iOS, therefore got rejected. After evaluating our options, we decided to remove Google Sign-In from the iOS version of the app entirely. Apple's requirement to keep user's email address private did not align with our use case, where tenants are expected to log in using the same email address they provided to their real estate agent.

Once both issues were resolved, the app was successfully approved, and the first public release was published on December 16, 2025, as planned.

6.4 Post-Launch Feedback and Monitoring

After the first release, the application's performance was monitored using a combination of user feedback and server-side logging of API requests. This allowed us to detect failed API requests and analyze technical issues in real-time, even without integrated analytics or crash reporting tools.

One of the first issues reported by real users was related to the account activation process. For security reasons, a verification code was sent to the user's email address and set to expire after 15 minutes. However, it was soon discovered that some users, particularly those with @seznam.cz email addresses, received the code too late, often after the 15-minute window had already expired. As a result, they were unable to complete the login process. To address this, the validity period of the activation code was extended to 30 minutes, balancing security with practicality.

Shortly after launch, an issue was reported with IBAN input validation in the personal information form. A misconfigured length constraint prevented users from entering valid IBANs. While this primarily affected non-Czech users which only accounted for a small percentage of all users at that time (as most Czech users entered bank account numbers instead of IBANs), the bug required urgent resolution. The fix was deployed quickly, but users needed to update to the latest app version.

Around the same time, another problem occurred where users were having trouble logging in because an invisible trailing space was being added after their email address. This usually occurred when users selected their email through a keyboard autocomplete that included a space character at the end. Since the email matching in the login process was strict, this caused authentication to fail. The issue was resolved by trimming the whitespace from the client-side email input before sending the login request.

6.4.1 Forced Updates

To handle these types of critical fixes and to ensure users are using a stable version of the app, a forced update mechanism was implemented before the first release, managed via Business Central. Administrators can set a minimum required app version; if a user runs an outdated version, they are prompted to update with a direct link to the app store and are not allowed to use the app until they update. This was necessary to ensure stability and avoid keeping track of different outdated versions of the app and importantly, to take care of possible critical bugs. Also, from the beginning there was a plan for a second release with added features, so there was a need to make the users update the app to access them.

6.4.2 Downtime Mode

In addition to the update control, the app includes a downtime mode—a safeguard that allows the app to be temporarily disabled in case of critical issues. This feature can be used during backend deployments or when there is a risk of data inconsistency. When downtime mode is activated, all requests from the app return a specific HTTP error code

(423 Locked), which is intercepted globally on the client. When this happens, users are redirected to a dedicated screen informing them that the app is temporarily unavailable. However, this feature has not yet been used in production.

6.5 App Store Statistics

The mobile application was published on both iOS and Android platforms and has been live for about five months. This section summarizes the usage metrics collected through App Store Connect (iOS) and Google Play Console (Android). The data sheds light on user behavior, platform performance, and the effectiveness of some development decisions such as localization and device targeting.

6.5.1 iOS (App Store Connect)

The app was downloaded over 250 times on iOS, with more than 1,500 impressions and 550 product page views, indicating a relatively high conversion rate. Notably, the crash report shows zero crashes during this period. While this seems ideal, React Native crash reporting might not be fully reliable.

Table 6.1: iOS App Statistics

Metric	Value
Downloads	250+
Product Page Views	550+
Impressions	1,500+
Reported Crashes	0
Daily Sessions per Device	3.36

The distribution of download sources reveals clear user behavior pattern. A large portion of installs came from app referrals (45%) and App Store search (41%), while only 3.5% each came from web and browse sources; the rest of the data is unavailable. This supports the idea that the app is used by targeted users who receive QR codes through emails or find them on the company's website, or directly search for the app.

Geographically, 66% of users are from the Czech Republic, 12.5% from Slovakia, and the remaining 21.5% from other countries, confirming that localization into Czech and English was an appropriate choice. Device usage shows that iPhones account for 96% of users, with iPads representing only 4%.

6.5.2 Android (Google Play Console)

On Android, the app currently has over 150 downloads. Crash monitoring revealed one issue related to a JavaScript Exception, and the user loss rate was low at 0.17%, which is a positive indicator of app stability and user retention.

Table 6.2: Android App Statistics

Metric	Value
Downloads	150+
Reported Crashes	1
User Loss Rate	0.17%

The most common OS version was Android 14 (48%), with the rest spread across versions 13, 15, 11, and 12. As with iOS, the user base is primarily from the Czech Republic (73%) and Slovakia (13%). Nearly all devices used are phones, with only one user accessing the app on a tablet, justifying the current lack of tablet-specific optimization.

6.6 Future Plans

Although the application's essential features have been finished, a number of areas have been noted for future development to improve usability and provide a better experience.

Currently, users receive email notifications of significant updates, like new contract creation or overdue payments. In future versions of the app, support for push notifications is planned. Users could be informed about necessary actions, documents that need to be addressed, or changes to their rental status more quickly and easily with these.

Next feature that will be implemented in the future is the ability for users to draw directly into the floor plan. When reporting a fault

in an apartment, there will be an option to draw the location for better identification. In addition, this feature will support a zoom feature that will allow users to zoom in on specific areas of the plan for more accurate marking.

Another planned feature are reports on utility use. Currently, users can view utility charges and download individual billing PDFs for each period. A future improvement would be to visualize the use of water, electricity, and other utilities over time. These reports could include visual breakdowns (for example bar graphs or monthly comparisons) to help users better understand their consumption habits. This would be especially useful for tenants staying long-term or sharing costs with others. However, right now there is not enough data to back this up.

Lastly, a dark mode option has also been considered. While it is not a current development priority, it remains a potential addition depending on user feedback. If used, dark mode would guarantee readability and a unified app design while offering a different color scheme better suited for low light conditions.

7 Conclusion

The goal of this thesis was to design and implement a multi-platform mobile application that allows tenants to manage various aspects of their rental agreements. The application was successfully released on both iOS and Android platforms within the planned timeline using the React Native framework and the Expo development platform. The required core functionality, viewing and signing contracts, tracking payments, and managing roommates, was completed and deployed, while some non-essential features, such as payment integration via a pay gate, were excluded after re-evaluation. Additionally, the application includes several other features such as uploading ID documents, submitting maintenance requests, and utility bills viewing.

The app is integrated into the larger platform through secure API endpoints. The design of the application evolved during development; although basic layouts were drafted at the beginning, many design decisions were made iteratively. This approach allowed for flexibility but also led to occasional rework. One lesson learned from this process was the importance of allocating more time to UI/UX planning in early phases. From a technical perspective, React Native and Expo proved to be an appropriate choice. Their use enabled rapid prototyping and supported the build and deployment process as the application was submitted to both the App Store and Google Play using Expo EAS Build. Certain issues had to be resolved after the app was launched, such as input validation problems and handling autofill-related errors. These were addressed quickly, and fixes were shipped through forced updates.

In conclusion, the development of the mobile application successfully met its objectives, with the application fully functional and live on both app stores.

Bibliography

1. DOMSYS. *Profesionální softvér pro správu nemovitostí* [online]. 2025. [visited on 2025-05-08]. Available from: https://www.domsys.cz.
2. SWORP. *Transform your property management* [online]. 2025. [visited on 2025-05-08]. Available from: https://www.sworp.com.
3. STARLIT. *Software pro snadnou a bezchybnou správu nemovitostí* [online]. 2025. [visited on 2025-05-08]. Available from: https://starlit.cz/index.html.
4. ATLASSIAN. *Welcome to Jira* [online]. 2025. [visited on 2025-05-11]. Available from: https://www.atlassian.com/software/jira/guides/getting-started/introduction#what-is-jira-software.
5. REHKOPF, Max. *What is a kanban board?* [online]. 2025. [visited on 2025-05-09]. Available from: https://www.atlassian.com/agile/kanban/boards#:~:text=A%20kanban%20board%20is%20an,order%20in%20their%20daily%20work..
6. JETBRAINS, s.r.o. *Cross-platform and native app development: How do you choose?* [online]. 2025. [visited on 2025-05-09]. Available from: https://www.jetbrains.com/help/kotlin-multiplatform-dev/native-and-cross-platform.html#what-is-native-mobile-app-development.
7. META PLATFORMS, Inc. *Introduction* [online]. 2025. [visited on 2025-05-11]. Available from: https://reactnative.dev/docs/getting-started.
8. SAKHNIUK M., Boduch A. *React and React Native: Build cross-platform JavaScript and TypeScript apps for the web, desktop, and mobile*. 5th ed. Birmingham, UK: Packt Publishing Ltd., 2024. ISBN 1805127306.
9. META PLATFORMS, Inc. *React Native. Learn once, write anywhere.* [online]. 2025. [visited on 2025-04-12]. Available from: https://reactnative.dev.

10. FACEBOOK. *react-native* [online]. 2025. [visited on 2025-05-11]. Available from: https://github.com/facebook/react-native.

11. META PLATFORMS, Inc. *Who is using React Native?* [online]. 2025. [visited on 2025-05-11]. Available from: https://reactnative.dev/showcase.

12. JETBRAINS, s.r.o. *The Six Most Popular Cross-Platform App Development Frameworks* [online]. 2025. [visited on 2025-05-03]. Available from: https://www.jetbrains.com/help/kotlin-multiplatform-dev/cross-platform-frameworks.html.

13. FLUTTER. *Build for any screen* [online]. 2025. [visited on 2025-05-09]. Available from: https://flutter.dev.

14. MICROSOFT. *.NET Multi-platform App UI* [online]. 2025. [visited on 2025-05-09]. Available from: https://dotnet.microsoft.com/en-us/apps/maui.

15. MICROSOFT. *Xamarin* [online]. 2025. [visited on 2025-05-09]. Available from: https://dotnet.microsoft.com/en-us/apps/xamarin.

16. JETBRAINS, s.r.o. *Kotlin Multiplatform* [online]. 2025. [visited on 2025-05-03]. Available from: https://www.jetbrains.com/kotlin-multiplatform/.

17. IONIC. *A new way to build and ship for mobile.* [online]. 2025. [visited on 2025-05-09]. Available from: https://ionic.io.

18. EXPO. *Expo*. 2025. Available also from: https://expo.dev. Accessed: 2025-05-07.

19. META PLATFORMS, Inc. *Get Started with React Native* [online]. 2025. [visited on 2025-05-07]. Available from: https://reactnative.dev/docs/environment-setup.

20. EXPO. *Is ejecting deprecated?* [online]. 2025. [visited on 2025-05-07]. Available from: https://docs.expo.dev/faq/.

21. MOEDANO, Beto. *Expo Go vs Development Builds: Which should you use?* [online]. 2025. [visited on 2025-03-14]. Available from: https://expo.dev/blog/expo-go-vs-development-builds.

22. EXPO. *Reference* [online]. 2025. [visited on 2025-05-11]. Available from: https://docs.expo.dev/versions/latest/.

23. EXPO. *Expo Application Services* [online]. 2025. [visited on 2025-05-09]. Available from: https://expo.dev/eas.

24. BIERMAN, Gavin; ABADI, Martın; TORGERSEN, Mads. Understanding typescript. In: *ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Sweden, July 28–August 1, 2014. Proceedings 28*. Springer, 2014, pp. 257–281.

25. DOCS, React Native. *Using TypeScript* [online]. 2025. [visited on 2025-05-09]. Available from: https://reactnative.dev/docs/typescript.

26. TANSTACK. *TanStack Query – React Native* [online]. 2025. [visited on 2025-05-09]. Available from: https://tanstack.com/query/latest/docs/framework/react/react-native.

27. DOCUMENTATION, react-i18next. *What is react-i18next?* [online]. 2025. [visited on 2025-05-08]. Available from: https://react.i18next.com.

28. SAP. *What is ERP?* [online]. 2025. [visited on 2025-05-09]. Available from: https://www.sap.com/products/erp/what-is-erp.html.

29. ORACLE. *Definition of enterprise resource planning (ERP)* [online]. 2025. [visited on 2025-05-09]. Available from: https://www.oracle.com/erp/what-is-erp/#:~:text=ERP%20stands%20for%20enterprise%20resource,finance%2C%20accounting%2C%20and%20more..

30. MICROSOFT. *Programming in AL* [online]. 2025. [visited on 2025-05-09]. Available from: https://learn.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/devenv-programming-in-al.

31. MICROSOFT. *API page type* [online]. 2024. [visited on 2025-05-09]. Available from: https://learn.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/devenv-api-pagetype.

32. MICROSOFT. *Multitenant Deployment Architecture in Business Central* [online]. 2022. [visited on 2025-05-09]. Available from: https://learn.microsoft.com/en-us/dynamics365/business-central/dev-itpro/deployment/multitenant-deployment-architecture.

33. DOCUSIGN, Inc. *Everything you need to agree* [online]. 2025. [visited on 2025-05-09]. Available from: https://www.docusign.com.

34. EXPO. *Core concepts of file-based routing* [online]. 2025. [visited on 2025-05-12]. Available from: https://docs.expo.dev/router/basics/core-concepts/.

35. APPLE, Inc. *App Review Guidelines* [online]. 2025. [visited on 2025-05-02]. Available from: https://developer.apple.com/app-store/review/guidelines/#login-services.

A Attachments

- source code of the application: `forrentor-app.zip` (Note: the `/service` folder containing API calls was intentionally removed for security and privacy reasons.)
- .apk build of the application: `build-forrentor.apk` (Note: this build is connected to the official backend system and therefore requires a real user account in the live environment to function properly. Without valid credentials, it is not possible to log in or access most of the app's functionality.)