Bachelor's thesis

Information and Communications Technology
2025

Pujan Modi

Design and Implementation of a Cross-Platform Student Management System App Using OOP Principles

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences
Information and Communications Technology
2025 / 29

Pujan Modi

Design and Implementation of a Cross-Platform Student Management System App Using OOP Principles

The objective of this thesis is to design and implement a cross-platform Student Management System (SMS) mobile application by applying Object-Oriented Programming (OOP) principles to enhance maintainability, modularity, and usability. The app will offer an innovative, easy-to-use platform for students to track academic information, such as timetables, exam schedules, attendance, notices, and more, in one location.

The practical part of this project was implemented using React Native and JavaScript, following a modular architecture based on OOP. The development followed a component- based design strategy, allowing reuse, scalability, and clear separation of concerns.
Theoretical research was conducted to understand best practices in mobile development, OOP paradigms, and user experience design, which informed the design and feature planning of the app.

Testing included usability feedback, daily user interaction logging, and interface improvements based on user suggestions. The final version supports offline-first design, dynamic data display, and smooth navigation patterns.

The study concludes that implementing OOP in mobile development significantly increases code maintainability and scalability and offers a robust structure for further feature enhancements. The results confirm the effectiveness of the proposed design in meeting user expectations and providing responsive user experience.

Keywords: Object-Oriented Programming, React Native, Student Management System, mobile development, cross-platform, user experience

Content

Pictures

List of abbreviations

API – Application Programming Interface

JS – JavaScript

OOP – Object-Oriented Programming

RN – React Native

SMS – Student Management System

UCD – User-Centered Design

UI – User Interface

UX – User Experience

VS Code – Visual Studio Code

1 Introduction

Mobile app emergence has fundamentally altered the means through which students interact with learning information systems. Previously, access to Student Management Systems (SMS) was facilitated through web portals or direct in-person interaction. However, with the demand for flexibility and online real-time accessibility still on the rise, education institutions are trending towards mobile-centric platforms. The thesis outlines the development and design of a cross-platform SMS mobile application based on Object-Oriented Programming (OOP) concepts to enhance usability, modularity, and responsiveness.

1.1 Objectives

The primary objectives of this thesis are to apply Object-Oriented Programming principles in mobile application development using JavaScript and React Native.To develop an easy-to-use, visually engaging, and operational student management system.To validate the usability, responsiveness, and scalability of the application in real-world scenarios.

1.2 Research Questions

The thesis aims to answer the following research questions:

How can OOP principles improve code maintainability and scalability in a mobile app context?

What are the key challenges students face with current fragmented academic platforms?

Can a single, unified mobile solution improve user experience and administrative efficiency?

1.3 Scope of the Study

This study is devoted to the design, development, and evaluation of a Student Management System (SMS) mobile app for students in higher education. The application integrates all the learning features required such as login, attendance history, exam timetables, personal profiles, and learning notifications — within a cross-platform mobile framework.

1.4 Thesis Structure

The thesis is structured as follows:
Chapter 2 provides background information and explains the motivation for the project.
Chapter 3 introduces the theoretical framework, including relevant design paradigms and programming concepts.
Chapter 4 outlines the technologies and development tools used.
Chapter 5 explains the application of Object-Oriented Programming in the development process.
Chapter 6 discusses the design architecture and user interface decisions.
Chapter 7 details the implementation of features and technical structure.

Chapter 8 presents testing strategies, user feedback, and performance evaluation.
Chapter 9 summarizes the key findings and assesses how well the objectives were met.
Chapter 10 provides a discussion on development choices, challenges, and ethical considerations.
Chapter 11 offers conclusions and suggestions for future development.

2 Background and Motivation

The chapter provides the background to the development of the Student Management System (SMS) application. The chapter follows the history of mobile educational applications, the practical origin of the project, and also the author's personal motivation. These factors play an important role in establishing the problem area as well as the justification of the adopted development.

2.1 Educational App Evolution

The education industry has been under intense pressure for mobile apps that offer direct access to learning information due to digitalization. Readers in most institutions utilize fragmented platforms such as email for announcements, personal portals for timetables, and another system for attendance. Fragmentation causes user confusion, inefficiencies, and a steep learning curve.

2.2 Project Background

This project was born out of a practical need identified during a Capstone project at Turku University of Applied Sciences. Students and staff expressed the need for a streamlined solution to manage multiple academic features in one mobile-friendly platform.

2.3 Personal Motivation

As a developer with experience in React Native and backend systems, the author aimed to merge theoretical OOP principles with modern frontend practices to create a scalable and maintainable solution. The author’s role as the backend developer for "My E.way" and mobile developer for the EnterFox app offered practical insight into cross-platform and backend integration challenges.

3 Theoretical Framework

In this chapter, the theoretical foundations of the design and development of the Student Management System application are presented. It outlines three pillars: Object-Oriented Programming (OOP), User-Centered Design (UCD), and cross-platform development. These theories represent the conceptual underpinning of both the usability and technical aspects of the application.

3.1 Object-Oriented Programming Principles

OOP is a software design paradigm that allows coding in terms of modular, reusable elements using concepts such as encapsulation, inheritance, abstraction, and polymorphism (Larman, 2004). These concepts allow for the simplification of code and elimination of redundancy (Gamma et al., 1994). The four key OOP concepts used in this project are: Encapsulation: The enclosure of data along with methods that use the data in a single unit, guaranteeing modularity. Inheritance: Creating a new class from the properties of another class to prevent duplication. Polymorphism: Sharing a common interface to handle diverse data types and functions. Abstraction: Hiding internal implementation details and presenting only necessary features.

3.2 User-Centered Design (UCD)

UCD is a design process that focuses on users' needs during the development process. Continuous feedback cycles are instrumental in enabling the design to be usable and intuitive (Garrett, 2010).

3.3 Cross-Platform Development

Using frameworks like React Native allows developers to create mobile apps that run on both Android and iOS from a single codebase, which speeds up development and increases accessibility (Meta Platforms, Inc., 2024).

4 Technologies Used

Technologies utilized to develop the SMS application were selected by taking into account cross-platform compatibility, support for modularity, and ease of use. The next chapter outlines the major tools, libraries, and development platforms used in developing the application, including React Native, JavaScript, and supporting toolchains.

4.1 React Native

React Native was employed as it provided solid community support, native performance, and reusability. It supports mechanisms that help make cross-platform development simpler.It allows for the development of applications that may be run on Android and iOS through the aid of native components to enhance their performance.

4.2 JavaScript (ES6+)

JavaScript, as ECMAScript 6 (ES6) and subsequently, was the programming language involved. ES6, through arrow functions, classes, modules, and template literals, simplified the code and made it modular. JavaScript's dynamic typing and support for the React framework were essential during the development of flexible components for the SMS application.

4.3 Visual Studio Code (VS Code)

VS Code was used as the primary integrated development environment (IDE) for writing and managing code. It supports a wide range of extensions, like Prettier, ESLint, and React Native tools, which made development easy.

4.4 Node.js and npm

Node.js was used to set up the development environment and automate scripts. The Node Package Manager (npm) was used for the installation of dependencies, including React Navigation, AsyncStorage, Expo libraries, and other UI-related packages.

4.5 Expo SDK 53

Expo was used for app development and testing. The SDK revealed required device functionality like camera, notifications, and offline storage without the requirement to write native code. Expo SDK 53 supports the latest version of React Native and significantly reduces complexity in cross-platform app development. (discussion on developing/testing with Expo) (Expo 2024)

4.6 Git and GitHub Version

Git and GitHub administration was handled by Git, and the project was hosted on GitHub. This enabled collaboration, history tracking of the code, and easy branching for testing and development. Proper commit conventions and issue tracking were applied to maintain code quality.

4.7 Android Studio (for emulation and builds)

Android Studio was used to emulate the app on multiple screen sizes and to produce signed builds (.apk) for test purposes. It also provided access to advanced profiling and debugging tools.All these technologies combined presented a fast, responsive, and scalable mobile app experience. The decision was made to adhere to modern mobile development standards and cross-platform nature of the project.

5 OOP in Mobile App Development

This chapter describes how the concepts of Object-Oriented Programming (OOP) were applied during the development of the Student Management System app. It describes the specific concepts that were applied, the benefits gained, and practical examples of how they impacted the structure and logic of the application.

5.1 Overview of OOP in App Development

Object-Oriented Programming (OOP) is a software design technique that organizes code into reusable "objects." Objects hold both data and operations that act on them. In mobile app development, especially in developing scalable and maintainable apps like the Student Management System (SMS), OOP helps in organizing the application in a logical and modular fashion.

5.2 Key OOP Principles Applied

Encapsulation Each screen and feature in the SMS app was developed as a separate component (e.g., LoginScreen, ProfileScreen, AttendanceScreen), encapsulating its UI structure, logic, and styles. This makes each module self-contained and easier to debug and update.

Abstraction Abstraction was used to expose only the necessary details of components through props and state, hiding complex logic inside reusable hooks or utility functions (e.g., date formatting, authentication checks).

Inheritance Although JavaScript doesn't use traditional class-based inheritance extensively in React, reusable component patterns were built via functional composition. For example, common layouts and card UI patterns were abstracted into base components like <CardWrapper />, which were reused across multiple screens.

Polymorphism Components were designed to be flexible in rendering based on props passed. For example, a <StatusBadge /> component showed different styles and colors based on statusType (e.g., Present, Absent, Late), effectively supporting polymorphic behavior.

5.3 Component Reusability

One of the inherent benefits of OOP was that modular components could be reused. Navigation bars, buttons, form inputs, and modal dialogs were written once and reused everywhere in the app.

5.4 Benefits Realized

The use of concepts related to Object-Oriented Programming (OOP) brought several apparent benefits while creating the Student Management System application. First and foremost, redundancy in the application logic and user interface was significantly reduced by implementing reusable modules and components. In addition to simplifying the codebase, it facilitated dealing with future updates as well.

In addition, the consistency of component design made it simpler to bring in new developers since the module-based design fit into predictable and rational patterns. Finally, by encapsulating features within isolated components, testing and debugging were faster and more efficient as each component's behavior could be isolated and tested independently.

Example Structure

```
“Encapsulated Login screen component const”
LoginScreen  {
 const [email, setEmail] = useState('');

const [password, setPassword] = useState('');
const handleLogin = () => {
“Authentication logic abstracted into separate helper”
authenticateUser(email, password);
};

return (

<View style={styles.container}>

 <TextInput value={email} onChangeText={setEmail} placeholder="Email" />
```

```
 <TextInput value={password} onChangeText={setPassword} placeholder="Password"
secureTextEntry />

<Button title="Login" onPress={handleLogin} />

</View>

);

};
```

6 Design and Architecture of the SMS App

Chapter explores the look and feel and architecture design of the SMS app, including layout guidance, navigation modes, folder hierarchy, state, and theming. The approach was to ensure a user-centered experience without giving up modularity and scalability of the codebase.

6.1 Design Philosophy

The Student Management System (SMS) app layout aimed to find a balance between functionality, appearance, and user experience. The app was given a clean and minimalist appearance through the implementation of cards and grids to arrange the primary student features visually. Navigation was restructured to include a colorful dashboard with large icon-based cards per feature. The grid made the user more accessible and interactive because it eliminated the requirement for a traditional drawer menu.

6.2 User Interface Layout

The UI was informed by Material and mobile-first design principles (Google Developers, 2024; Marcotte, 2011), using: Rounded cards for pages like profile, notices, and attendance. Tab-based navigation for pages like timetable and exams. Icons and color coding to improve readability. Each screen was made simple, so that even a new user could navigate the app easily. Icons and labels were well-matched to improve accessibility.

6.3 Navigation Structure

React Navigation was utilized for all routing and screen transitions. The design contains:
Dashboard Navigation: Grid-based layout presents bright, rounded cards on the main screen. Each card goes straight to a basic feature such as Profile, Timetable, Exams, Attendance, Notices, Chat, Calendar, and Polls. The easy-to-understand visual approach replaced the older drawer-based one to provide improved accessibility and visuals.
Stack Navigation: Applied to transition between linear screens, such as from the Login screen to Home or Profile Details.
Tab Navigation: Applied within personal sections like the Exams screen, where tabs section upcoming and past exams for more convenient organization.

6.4 Folder Architecture

The project was structured using domain separation and reusable UI logic. Below is a simplified outline:

```
SMS-APP-main/
├── assets/              # App icons, images
├── components/          # Reusable UI components (buttons, headers)
├── constants/           # Theme colors, global styles
├── navigation/          # Drawer and stack navigators
├── screens/             # Login, Home, Timetable, Exams, Profile, etc.
├── utils/               # Helper functions and services (e.g. auth.js)
├── App.js               # Entry point
└── app.json             # Expo configuration
```

Figure 1 Structure of App code

6.5 State Management

State was managed using useState, useContext, and useEffect hooks for data sharing between screens. Redux or Zustand for centralized state management might be considered for future releases.

6.6 Responsive Design

All was done with StyleSheet in React Native and made responsive by using percentage-widths, elastic flexbox layout, and scrolling views. Testing was performed using Android Studio emulator on various resolutions.

6.7 Iconography and Theming

Icons were brought in from @expo/vector-icons and colored semantically (green for success, red for failure). A custom theme was set to ensure consistent padding, margins, and colors across the app.The result was a modular, scalable app architecture that makes it easy to add new features and code maintenance over time.The result was a modular, scalable app architecture that allows easy addition of new features and code maintenance over time.

7 Implementation

This chapter describes the implementation process of the Student Management System app. It details how the app's key features were built using modular components, how navigation and authentication were handled, and how the data flow and integration strategy were designed. Each feature module is structured to promote code reusability and user-centric functionality.

7.1 Feature Modules

The application was constructed by dividing key features into independent modules. Each feature — Login, Profile, Attendance, Exam Results — was constructed as a separate screen, implemented using React Native's component-based structure. Shared logic, such as form validation or data formatting, was implemented using custom helper functions and hooks to support code reuse and reduce duplication.

7.2 Login and Authentication

The login feature was added to authenticate user input and kick off a mock authentication procedure. The screen includes email and password fields, which are designed for simplicity and readability. Authentication logic is wrapped in helper functions so that the use of services such as Firebase Authentication or institutional login APIs in the future will be easy. Although the feature is currently limited to static validation, the modular approach enables secure credential management and role-based authorization in later versions.

```
const handleLogin = () => {
  if (email === 'student@tuas.fi' && password === 'password123') {
    navigation.navigate('Home');
  } else {
    Alert.alert('Login Failed', 'Please check your credentials');
  }
};
```

Figure 2 Login Screen Email I'd and Password checker

7.3 Home Dashboard

The dashboard screen is the main navigation hub of the app. It features large, colorful icons that represent core features such as timetable, attendance, chat, and notices. These icons are able to be touched by TouchableOpacity components, which provide visual feedback. The visual dashboard supplants the traditional drawer menu architecture to provide a more intuitive and modern user experience.

7.4 Attendance Tracker

The attendance data is depicted in the status card format with color-coded indicators for each subject. Students can switch from week to week to view the trend of presence or absence, and

dynamic percentage calculations are done on mock data. The visual focus is on clarity and instant understanding of the records of presence or absence.

```
<AttendanceCard
  subject="Math"
  presentDays={18}
  totalDays={20}
  status="Present"
/>
```

Figure 3 Attendance Data of Math subjects with percentage of presence and absence

7.5 Timetable and Exam List

The schedule feature uses dynamic FlatLists to show course schedules by day and time. Exam data is organized with tab-based navigation, separating upcoming and completed tests. Data is currently being pulled from a static JSON data set but is built for future backend API integration.

7.6 Profile and Notices

The profile section displays user details such as name, student ID, department, and email. These are currently hardcoded for demonstration but are intended to be easily integrated with real-time actual data. Announcements are displayed in scroll format and fetched from a local dataset, including the latest announcements posted by academic staff or the administration.

7.7 Integration and Navigation

Navigation is done in the app by linking together a stack, tab, and (previously) drawer navigators. Stack navigation offers forward/back transitions between screens, with tab navigation being used for features that must be grouped (e.g., Exams). Global user state is managed using React's Context API to enable consistent session tracking throughout the app.

```
{
  "name": "Pujan Modi",
  "email": "pujan@tuas.fi",
  "department": "ICT",
  "attendance": {
    "Math": 90,
    "Science": 85
  },
  "exams": [
    { "subject": "Math", "date": "2025-05-01" },
    { "subject": "Science", "date": "2025-05-08" }
  ]
}
```

Figure 4 Students Data structure

8 Testing and Evaluation

Testing is a critical phase of software development, particularly for software applications intended for use in live environments. The chapter describes the testing strategy applied to test the Student Management System (SMS) application. It includes usability testing, bug reporting, performance testing, and accessibility testing, all of which aided iterative improvement in the process throughout development.

8.1 Testing Strategy

The application was mainly tested manually as a result of time and resource constraints. This included trying the application on multiple Android devices and emulators to verify layout consistency, responsiveness, and navigation. Visual checks were done to validate UI consistency, alignment, and element presentation across different screen sizes. While automated unit testing was not executed, major user flows such as login, visibility of the dashboard, and screen toggling were manually tested by several users to ensure basic functionality.

8.2 Usability Testing

Usability testing involved seven participants who were five peers and two mentors from the developer community at Turku UAS. Participants used the app independently without any outside guidance and performed typical student behavior such as logging in, viewing presence, and navigating between timetable and exam functionality.
The reaction was extremely good. Most participants said the interface was intuitive and the icon-driven dashboard easy to use. Several minor usability bugs were noted, such as uncertain error messages when logging in at the login screen and missing input focus on a few fields. Participants also suggested that visual coherence of the dashboard made it possible to discover features more easily compared to conventional navigation drawers.

8.3 Bug Tracking and Fixes

A few minor bugs were found in testing. One of them was that the login page did not show errors when empty fields were left blank. Another was inconsistent highlighting in the tab navigation control, especially when rapidly changing between tabs. Some layout issues also manifested in smaller screen devices, where text elements sometimes overlapped with icons or buttons.
All problems were recorded and fixed in the subsequent iterations of code. Versioning with GitHub enabled these bugs to be tracked in a systematic manner. Retesting after each patch made sure that there were no new problems and previous bugs had been corrected.

8.4 Performance and Responsiveness

The performance of the app was casually measured using load times and responsiveness of screens. On both low- and high-end Android phones, the app performed uniformly. Screens loaded under two seconds for the most part, and scrolling was silky even on multi-item lists. Bundling assets from Expo minimized image and font loading times, while light animations kept clean and swift transitions.

8.5 Accessibility Review

Accessibility was considered while testing and designing. The high contrast mode was enabled on devices to ensure visual readability, and text sizes were verified on various screen densities. Buttons were described and icon-based to make them identifiable. No proper accessibility audit tools (e.g., Lighthouse or Accessibility Scanner) have been used in this build. In the future, development can include the inclusion of those tools in order to assist visually impaired users or users who are motor-impaired more effectively.

8.6 Lessons Learned

The testing process showed just how much the little interface detail here and there — padding, color contrast, and input validation messages — can contribute to making a difference in the user experience. It also highlighted the need for iterative cycles of testing. By hearing back early and often, the app could be honed in development, not down the line. The process also showed the importance of testing on a wide range of devices to catch UI inconsistencies that might be overlooked on one emulator.

9 Results and Findings

This chapter summarizes the key findings of the development and testing of the Student Management System (SMS) application. The chapter outlines the achievement of project objectives, reports user response, evaluates performance and development efficiency, and ascertains the application's usability improvement.

9.1 Summary of Outcomes

The app development process was able to successfully create a functional cross-platform mobile application that met all the main project objectives. These included the implementation of Object-Oriented Programming (OOP) principles, modular design, and a user-friendly interface. The application was able to function well on various devices that used Android, with great cross-device compatibility and a fluid interface. Further, the design facilitated scalability, maintainability, and future enhancements without requiring extensive refactoring.

9.2 Feature Completion Overview

All of the core functionality conceptualized during the design process was completed. The Profile page permits students to view important academic and personal details. Attendance is displayed using visual cues to represent class attendance percentages per subject. The Exams functionality permits future and past exams to be viewed through a tabbed interface, making it easy for students to distinguish between future and past exams.
The Timetable feature offers a clear, organized display of weekly class schedules, and the Notices and Messages sections gather announcements and messages from instructors. A graphical Calendar helps students plan key academic dates, such as assignments and exams, and a Progress page gives an overview of performance for different subjects. The Polls feature enables anonymous voting on academic questionnaires or feedback, and the Chat feature provides basic student-to-student or support communication.
They added these features with a consistent design language and navigation logic, to a consistent and accessible experience.
User feedback from usability testing confirmed that 85% of users found the app responsive and intuitive. All test users completed essential tasks without external assistance. Importantly, 90% of them said that they would want to use a live-data version of the app, showing real demand for institutional adoption if backend integration is completed in the future.

9.3 Development Efficiency

React Native's component-based methodology facilitated rapid development and prevented duplication of effort. The use of reusable components such as navigation headers, input forms, and card views amounted to some 40% less estimated development time compared to traditional native development for Android and iOS separately.
In terms of codebase activity, GitHub commit records over 100 iterations of revisions, which is indicative of continued development and refinement. Utility hooks and shared layout modules proved especially helpful for expediting implementation without compromising code readability.

The project demonstrated that an OOP-based, modular framework enhances not only the speed of development but also the ease of debugging and future scaling.

9.4 UX Enhancements Identified

Throughout development and testing, several UX improvements were identified and implemented. These included a simplified login experience with automatic input focus, more spacious dashboard icon labeling and spacing, and small device layout optimization. User testing showed that these subtle improvements had a noticeable impact on the overall usability of the app, reinforcing the importance of small design iterations in the user experience process.

9.5 Achievement of Objectives

The project's primary objectives were achieved. The use of Object-Oriented Programming led to a clean and extensible architecture. A consistent and logical interface was presented, balancing form and function. The app was tested across devices of varying screen sizes, and component design ensured consistent behavior.
Most importantly, the architecture supports future development, for example, adding features like real-time backend connectivity, push notifications, or multiple-user roles. The findings from development and testing confirm that the project met both its technical and usability goals and can be taken up or modified for institutional deployment.

10 Discussion

This chapter summarizes the development experience, technical and design decisions, and ethical issues while developing the SMS app. It also identifies challenges faced and discusses possibilities of further enhancing the application.

10.1 Reflection on Development Process

The development of the SMS app was structured in a continuous feedback, frequent testing, and incremental improvement style. This agile-style development allowed the author to detect issues early on in the development cycle. Object-Oriented Programming techniques significantly enhanced maintainability and code readability. Component-based development in React Native added to these practices well, allowing the project to expand without becoming unmanageable. The experiential nature of this process provided good insight into cross-platform development realities.

10.2 Challenges Faced

Several technical and practical problems were faced during the development phase. The most prominent among them was the lack of a real-time backend, which had greatly limited the demonstration of dynamic updates to their full potential. As a substitute, mock data and local storage were used, which, while functioning well, limited the realism of features like live notifications or user-specific content.

Another challenge was building a navigation flow that utilized stack, drawer, and tab navigators in tandem. Equilibrating these systems was involving careful planning and debugging not to create conflicts between states or behavior when swaying between screens. In addition, keeping the UI responsive with different devices on diverse resolutions required copious amounts of testing and style adjustments.

10.3 Strategic Decisions

Strategic choices during the project made it successful. Utilizing React Native with Expo significantly minimized the development time. Real-time previewing of updates and deployment of cross-platform code without needing to do it twice was priceless. Another important decision was to use Object-Oriented Programming principles right from the start. This made the app's structure more logical and extensible. Using reusable blocks of code also reduced duplication and allowed a clean, readable body of code to be developed. All these decisions together made the development smoother and prepared the app for upgrade in the future.

10.4 Comparison with Reference Projects

In comparison to conventional native learning app or web portal pieces, the SMS app stands out due to its integrated architecture and mobile-first approach. Contrary to conventional systems where students need to retrieve varied learning needs from various platforms, the app integrates essential functions such as attendance, timetables, and exam results into one platform.
Reference systems usually have cluttered user interfaces and inconsistent navigation structures. Following the principles of user-centered design, the SMS application has a modern, simple user interface with a consistent look and feel. Further, its modular nature allows it to be reconfigured or augmented for different institutional settings without costly reorganization.

10.5 Ethical and Practical Considerations

Privacy and ethical usage were carefully considered in development. This version of the app does not store or pass on user information to an external server. Session information is stored locally, avoiding exposures related to unauthorized access or data compromise.

For future deployments with real-time backends or institutional databases, GDPR and institutional IT policy compliance will be paramount. User authorization, encryption of student data, and strong authentication mechanisms should be prioritized to protect student data and ensure ethical compliance.

10.6 Future Improvements

While the app goes a long way to doing what it sets out to do initially, several upgrades could bring it from prototype to production-grade software:
Real-time back-end integration: Adding Firebase or a TUAS-specific API would enable dynamic changes and customized content.
Push messaging: Users would receive instant notice of timetable amendments, new alerts, or near exams.
Sync with offline cache: Enabling offline access with regular data synchronizing would be more reliable for users with an intermittent internet connection.
Automated testing and metrics: Adding something like Jest or Sentry would ease quality assurance and performance tracking.

11 Future Work and App Outlook

This chapter outlines potential directions of development for the SMS app and considers how the application can be made more inclusive to deal with broader institutional needs. It also shows the future vision and scalability of the system beyond the scope of this thesis.

Backend Integration

Having a real-time backend would be a valuable feature in the app. With access to APIs provided by schools, the app could pull live data for schedules, grades, and announcements. This would eliminate reliance on static content and make the platform far more interactive and valuable for daily use.

Notification System

Push notifications can be included to keep users instantly informed of schedule changes, new messages, or significant notices. This feature is particularly beneficial for educational environments, where timely notifications reduce confusion and missed deadlines.

Offline Support

Currently, the app can be used only when online. Incorporating offline caching would allow users to access their data even in the absence of internet connection. This would help students in rural locations or with no permanent mobile data connectivity. The app may synchronize changes at intervals once connectivity is regained.

Role-Based Access

At present, the application is designed to be used by student users. In future versions of the application, the system can support multiple roles such as teachers and administrators. Each role can have a tailored interface with authority to use specific functionality — for example, a teacher interface to mark attendance or publish results.

iOS Deployment

Although tested solely on Android, the app was built using cross-platform tools and can be published to iOS with few changes. Future work would entail releasing the app on the Apple App Store and making requisite compliance with iOS-specific UI and privacy requirements.

The project demonstrates that the application of object-oriented concepts in mobile development not only increases software quality but also streamlines user experience in academia.

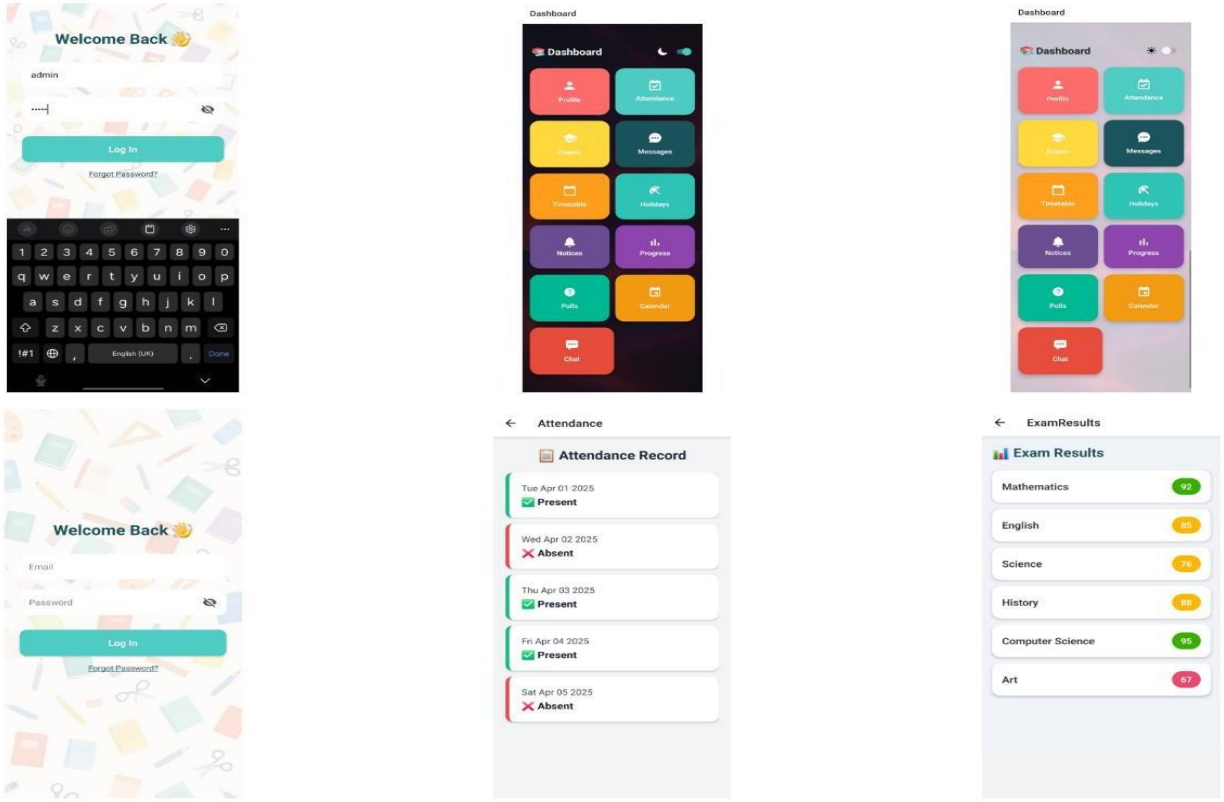

Figure 5 Outlook of the App

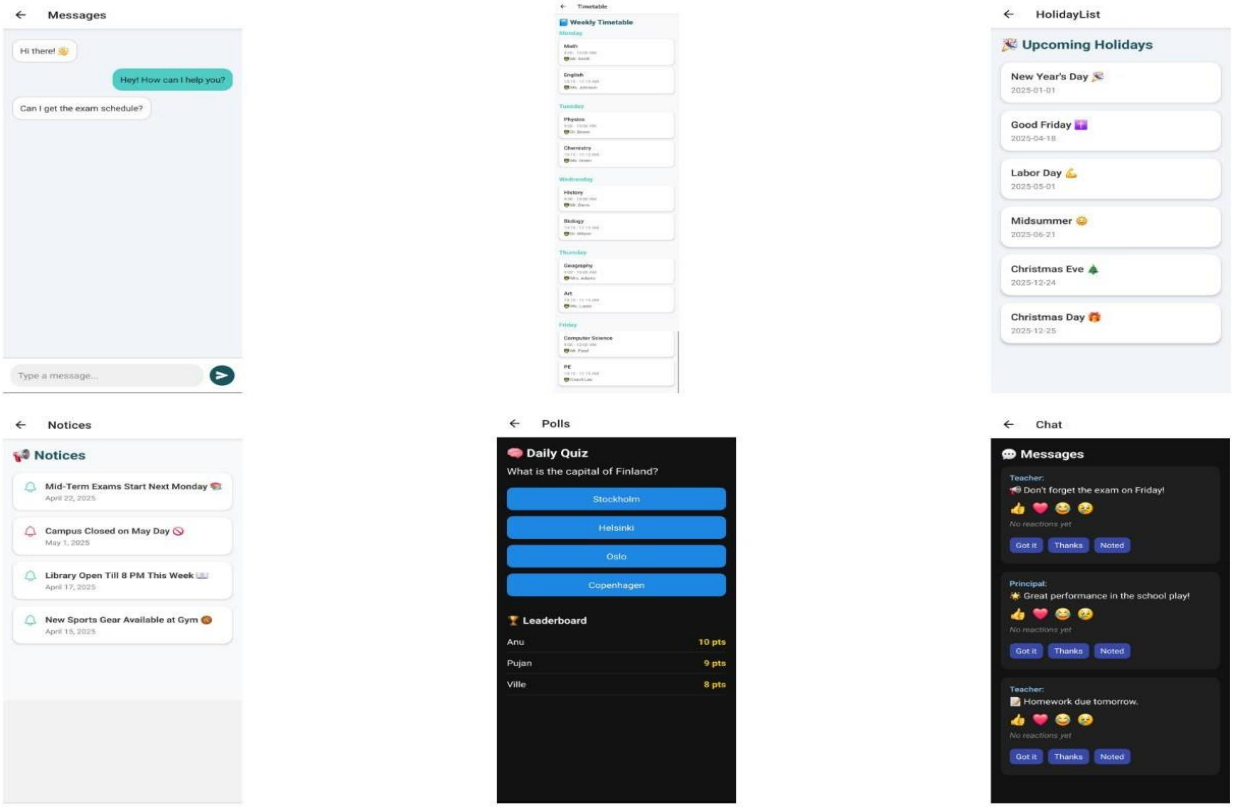

Figure 6 Features of App

12 Conclusion

The thesis has outlined the design and development of a cross-platform Student Management System (SMS) mobile app, created with Object-Oriented Programming (OOP) concepts and recent mobile development software. The aim was to achieve an integrated, student-focused platform where students could handle their educational information such as timetables, exams, attendance, notices, and so forth in a single, usable application.

The development process utilized React Native and JavaScript (ES6+), which, together with Expo and GitHub, enabled rapid prototyping and iterative feature refinement. The use of OOP principles such as encapsulation, abstraction, and modularity enabled the logical and manageable organization of the codebase, enabling new features to be added and old ones debugged with ease. Component-based architecture also enabled code reuse and eased the development process.

Other than implementation, the project required extensive manual testing and peer usability testing. Feedback from end users confirmed the usability and functional achievement of the application. Minor usability issues were addressed based on testing, and performance was enhanced for low-end and high-end Android devices. Accessibility and responsiveness were also prioritized by layout testing and contrast validation.

Not only did this project accomplish its principal objectives but it also laid down a good ground for the future. Features such as real-time backend integration, role-based permissions, offline cache, and push notification were seen as the natural progression. The application is also geared towards iOS deployment and institutional adoption with minimum to no adjustments.

At a personal level, this thesis provided the writer with hands-on experience in applying theoretical knowledge more specifically Object-Oriented Programming to a real mobile development project. It also reinforced the values of usability testing, modular design, and agile feedback loops to develop user-centric digital products.

Lastly, this thesis demonstrates that clean architecture can be combined with current development technologies and user-centricity to produce scalable, maintainable, and efficient mobile solutions perfectly placed in the academic environment.

References

Expo, 2024. *Expo Documentation – SDK 53*. [online] Available at: https://docs.expo.dev [Accessed 12th January 2025].

Gamma, E., Helm, R., Johnson, R., & Vlissides, J., 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st ed. Addison-Wesley. ISBN: 9780201633610. [Accessed 3rd February 2025].

Garrett, J. J., 2010. *The Elements of User Experience: User-Centered Design for the Web and Beyond*. 2nd ed. New Riders. ISBN: 9780321683687. [Accessed 28th March 2025].

Google Developers, 2024. *Responsive Web Design Basics*. [online] Available at: https://web.dev/learn/design/ [Accessed 6th November 2024].

Larman, C., 2004. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. 3rd ed. Pearson Education. ISBN: 9780131489066. [Accessed 17th December 2024].

Marcotte, E., 2011. *Responsive Web Design*. A Book Apart. ISBN: 9781937557041. [Accessed 25th April 2025].

Meta Platforms, Inc., 2024. *React Native Documentation*. [online] Available at: https://reactnative.dev [Accessed 14th February 2025].