جامعة الأخوين

ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵓⵅⴰⵡⴰⵢⵏ

AL AKHAWAYN UNIVERSITY

SCHOOL OF SCIENCE AND ENGINEERING

AUI CAMPUS: MOBILE CONSOLIDATION OF UNIVERSITY RESOURCES

FINAL CAPSTONE DESIGN REPORT

Submitted on April 28th, 2023

H. Rehioui

Supervised by: Dr. H. Harroud

AUI CAMPUS: MOBILE CONSOLIDATION OF UNIVERSITY RESOURCES

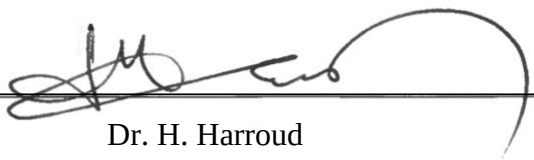
Capstone Report

Student Statement:

I, Hamza Rehioui, affirm that I have applied ethics to the design process and in the selection of the final proposed design. I have held the safety of the public to be paramount and have addressed this in the presented design wherever may be applicable.

Hamza Rehioui

Approved by the Supervisor(s)



Dr. H. Harroud

ACKNOWLEDGEMENTS

I would like to extend my sincere gratitude to the university administration, faculty, and staff for their unwavering support and invaluable guidance throughout the development of the AUI Campus application. Their insights and expertise have been instrumental in shaping the platform to best serve the needs of the campus community. I also express my heartfelt appreciation to my fellow students for their constructive feedback and active participation, which greatly informed the design and development process.

Additionally, I would like to express my deepest gratitude to my parents, who have provided me with unwavering support and encouragement throughout my capstone project. Their faith in my abilities and constant motivation have been invaluable in my journey, as they have always reminded me of my potential even during the most challenging times.

I am also immensely grateful to my supervisor, Dr. H. Harroud, whose expert guidance, dedication, and constructive feedback have been instrumental in shaping this project. His extensive knowledge, patience, and genuine interest in my work have not only enhanced the quality of my research but have also inspired me to become a better scholar. I sincerely thank him for his time and commitment, and for helping me transform my ideas into reality.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

AI: Artificial Intelligence

API: Application Programming Interface

AKS: Azure Kubernetes Service

AUI: Al Akhawayn University in Ifrane

CRUD: Create, Read, Update, and Delete

DDD: Domain-Driven Design

GraphQL: Graph Query Language

GUI: Graphical User Interface

IaaS: Infrastructure as a Service

ML: Machine Learning

NPM: Node Package Manager

ORM: Object-Relational Mapping

PaaS: Platform as a Service

RESTful: Representational State Transfer

UI: User Interface

UUID: Universally Unique Identifier

UX: User Experience

ABSTRACT

AUI Campus is a mobile application designed to consolidate all university resources in one easy-to-use platform. This project aims to improve the lives of students, faculty, and staff by providing access to essential campus services and information.

The application uses big data and data mining techniques to collect and analyze data on users and employs artificial intelligence for both its chatbot and search engine feature. This report discusses the various aspects of the project, including its implementation details and the technologies used.

RÉSUMÉ

AUI Campus est une application mobile conçue pour consolider toutes les ressources universitaires sur une plateforme facile à utiliser. Ce projet vise à améliorer la vie des étudiants, du corps professoral et du personnel en fournissant un accès aux services et informations essentiels du campus.

L'application utilise des techniques de big data et de data mining pour collecter et analyser des données sur les utilisateurs, et utilise l'intelligence artificielle pour ses fonctionnalités de chatbot et de moteur de recherche. Ce rapport examine les différents aspects du projet, y compris les détails de sa mise en œuvre et les technologies utilisées.

1 INTRODUCTION

Higher education institutions face numerous challenges in providing efficient access to resources and services for their constituents. AUI Campus is a mobile application designed to address these challenges by consolidating all campus resources into one place.

This application aims to improve the lives of students, faculty, staff, and even guest visitors by providing them a central platform for accessing essential campus information and services. The project leverages big data, data mining techniques, artificial intelligence, and geolocation to provide a seamless user experience and facilitate access to data.

The following sections of this report cover the feasibility study which includes a STEEPLE analysis, requirement elicitation, requirement specifications, software architecture, design, implementation details, technological enablers, and conclusions, as well as future of the AUI Campus application.

2 STEEPLE ANALYSIS

A STEEPLE (Social, Technological, Economic, Environmental, Political, Legal, and Ethical) analysis was conducted to evaluate the various factors influencing the development and implementation of AUI Campus. The analysis revealed the need for a comprehensive platform catering to the diverse needs of the campus community, while also considering the legal and ethical implications of data collection and usage.

The following analysis outlines the steeple aspects in relation to the application:

- Social: The AUI Campus application addresses the social needs of the campus community by providing a centralized platform for accessing resources and services. Students, faculty, and staff can easily find information and interact with various campus services, fostering a sense of community and engagement.

- Technological: Advances in mobile technology, big data, data mining, artificial intelligence, and neural networks enable the development of a robust and feature-rich application like AUI Campus. These technologies allow for efficient data collection and analysis, personalized user experiences, and automation of various campus processes.

- Economic: The AUI Campus application has the potential to generate revenue through targeted advertising and partnerships with local businesses. Additionally, the efficient management of campus resources and services can lead to cost savings for the university.

- Environmental: By providing a digital platform for accessing campus resources and services, AUI Campus can contribute to reducing paper waste and energy consumption associated with traditional means of information dissemination.

3 FEASIBILITY STUDY

A comprehensive feasibility study was conducted to assess the viability of the AUI Campus application from various perspectives, including technical, economic, operational, and legal aspects.

- Technical Feasibility: The technical feasibility of the AUI Campus application was analyzed by evaluating the availability and maturity of the required technologies, tools, and platforms. The study confirmed that existing technologies such as big data and data mining tools, artificial intelligence platforms. Moreover, the availability of suitable programming languages, libraries, and frameworks allows for the efficient and reliable development of the AUI Campus platform.

- Economic Feasibility: The economic feasibility of the AUI Campus application was assessed by estimating the potential revenue generation and cost savings for the university. By integrating targeted advertising and forming strategic partnerships with businesses and service providers, the application could generate significant revenue. Additionally, the automation of various campus processes could result in cost savings for the university. These potential financial benefits support the justification for investing in the development and maintenance of the AUI Campus platform.

- Operational Feasibility: The operational feasibility of the AUI Campus application was evaluated by considering the ease of integration with existing campus systems and the potential impact on daily campus operations. The application is designed to be compatible with various campus systems, ensuring smooth data exchange and minimal disruption to existing processes. Moreover, the user-friendly interface and comprehensive consolidation of campus resources are expected to enhance the efficiency of daily operations for students, faculty, and staff.

- Legal and Ethical Feasibility: The legal and ethical feasibility of the AUI Campus application was examined by analyzing the potential implications related to data privacy and security. The application is designed to comply with applicable privacy laws and

regulations, such as Moroccan Privacy Regulations as well as internal university policies. Additionally, the implementation of robust security measures, including encryption and secure authentication, ensures the protection of user data and helps mitigate potential risks associated with data breaches.

Based on the results of the feasibility study, it can be concluded that the AUI Campus application is a viable solution that addresses the need for a centralized platform for accessing university resources and services. The combination of technical, economic, operational, and legal feasibility supports the decision to proceed with the development and implementation of the AUI Campus platform.

4 REQUIREMENTS SPECIFICATION

In this section, I will outline the requirements specification for the AUI Campus application, detailing both the functional and non-functional aspects. Functional requirements focus on the core features and capabilities of the application, while non-functional requirements address characteristics such as performance, usability, and security. These specifications provide a comprehensive understanding of the expectations and constraints associated with the AUI Campus platform.

4.1 FUNCTIONAL REQUIREMENTS

- User authentication: The mobile application must allow users to create and log in to their accounts using their email addresses, passwords, and other authentication measures, such as two-factor authentication.

- User profile management: The mobile application must allow users to view and update their personal information, such as their first name, last name, email address, and contact information.

- User group management: The mobile application must allow users to create, view, and join different user groups. Users should also be able to leave user groups if they no longer want to be part of them.

- Event management: The mobile application should allow users to create, view, and RSVP to events. Users should be able to filter events based on different criteria such as location, date, and type.

- Location management: The mobile application should allow users to view and search for different locations. Users should also be able to filter locations based on different criteria such as location type and category.

- Contact management: The mobile application should allow users to view and search for different contacts. Users should also be able to filter contacts based on different criteria such as location, role, and organization.

- Content management: The mobile application should allow users to create and view different types of content such as posts and images. Users should be able to filter content based on different criteria such as organization and category.

- Chatbot for assistance and inquiries: The system should have a chatbot feature that can interact with users, answer frequently asked questions, and help with various tasks.

4.2 NON-FUNCTIONAL REQUIREMENTS

- Performance: The mobile application must be able to handle many users and data without experiencing any performance issues.

- Security: The mobile application must provide a secure environment for users to store and access their personal and sensitive information.

- Reliability: The mobile application must be always reliable and available to users.

- Usability: The mobile application must be easy to use and navigate, with intuitive design and user interface elements.

- Compatibility: The mobile application must be compatible with a wide range of mobile devices, operating systems, and web browsers.

- Scalability: The mobile application must be scalable and able to accommodate a growing user base and data volume.

- Integration: The mobile application should be able to integrate with other services such as social media platforms, cloud storage, and payment gateways.

5 ENGINEERING STANDARDS

To ensure the quality, reliability, and interoperability of the AUI Campus application, adherence to various engineering standards is crucial. These standards provide guidelines and best practices for the development and maintenance of software systems. The following are key engineering standards that were followed during the development of the AUI Campus application:

- IEEE Standards: The IEEE develops and publishes standards for various industries, including software engineering. The application adheres to relevant IEEE standards, such as:
    - *IEEE 830-1998:* Practice for Software Requirements Specifications
    - *IEEE 1016-2009:* Standard for Information Technology, Systems Design
    - IEEE 1028-2008: Standard for Software Reviews and Audits

  By following these IEEE standards, the AUI Campus application ensures a high level of quality and consistency in its requirements specification, design, and development processes.

- ISO/IEC Standards: The ISO and the IEC jointly develop and publish standards for information technology and software engineering. The application complies with relevant ISO/IEC standards, such as:
    - *ISO/IEC 25010:2011:* Systems and software engineering, SQuaRE, and System and software quality models
    - *ISO/IEC 27001:2013:* Information technology, and Security techniques

  These standards provide guidance on software quality attributes and information security management, ensuring that the AUI Campus application meets the highest levels of quality, reliability, and security.

In conclusion, compliance with these engineering standards (IEEE, and ISO/IEC) is essential for the AUI Campus application to guarantee the highest levels of quality, reliability, and security. By adhering to these standards, the AUI Campus platform ensures a consistent and effective user experience while maintaining compatibility with modern technologies and best practices in software engineering.

6 SOFTWARE ARCHITECTURE

The main building blocks of the AUI Campus solution include:

1. User Interactions through mobile application or web application interface
2. Business Logic Operations
3. Database for storage and analysis
4. AI-based chatbot system
5. AI-powered search engine
6. Many integrations with external APIs

The interface is expected to be user-friendly and compatible with various mobile devices and operating systems. The application should manage data processing as well as business logic related functions, as well as store and analyze large amounts of user data. The search engine indexes the entire database and allows searching through the entire database for relevant information. The AI-based chatbot system should assist users with inquiries and provides guidance to its users.

6.1 N-TIER VS. MICROSERVICES

To realize all of this, I must decide on a system architecture. Now, before choosing the architecture with which I will be developing the application, I must conduct a comparative study between the two most popular software architectures as of late, N-tier Architecture and Microservices. Below is a comparison table outlining pros and cons of each of these architectures:

Table 1 Pros and Cons of N-tier vs. Microservices Architectures

	Pros	**Cons**
N-tier Architecture	**Simplicity:** More straightforward approach, which might be easier to	**Limited scalability:** Does not scale as well as microservices, which could be a problem as

	implement and manage for a project like AUI Campus. **Familiarity:** Many developers have experience with this architecture, making it easier to find the necessary skillset for implementing and maintaining the project. **Centralized control:** There is centralized control over the application, which can make it easier to manage updates, enforce security policies, and maintain consistency.	AUI Campus grows and requires more resources. **Tight coupling:** Components are often more tightly coupled, which can make it harder to change or update individual components without affecting the entire system. **Monolithic nature:** Its monolithic nature can make it harder to leverage the latest technology or adapt to changing requirements.
Microservices Architecture	**Scalability:** Allows you to scale individual components independently, which is beneficial for a project like AUI Campus that may experience varying load on different services. **Flexibility:** Allows you to develop and deploy each service independently, reducing the dependencies among different components. **Technology agnostic:** Each microservice can use a different technology stack tailored to its specific requirements, providing more options for leveraging the best tools for each task. **Fault isolation:** Since each microservice is isolated, a failure in one service doesn't	**Complexity:** It introduces additional complexity in managing and orchestrating multiple services, which could be challenging for a project like AUI Campus that already has numerous functionalities. **Latency:** Communication among microservices may introduce latency, especially if there are many inter-service calls or if data must be retrieved from multiple services. **Operational overhead:** Monitoring, logging, and managing the deployment of numerous microservices can lead to increased operational overhead.

	necessarily lead to the entire application's collapse.	

Given the size of the team and the relatively small scale of the AUI Campus project, it is reasonable to conclude that an n-tier architecture might be a better choice initially. Implementing an n-tier architecture simplifies the development and management process, allowing the focus to be on delivering a high-quality application without getting overwhelmed by the complexities associated with microservices. However, as the project grows and evolves, the benefits of microservices, such as scalability, flexibility, and fault isolation, will become increasingly important. To accommodate these future needs, it is inevitable that the project will need to transition to a microservices architecture. By planning and architecting the n-tier solution with future modularity in mind, the team can ease the transition to microservices when the time comes, ensuring the AUI Campus application remains adaptable and resilient in the long term.

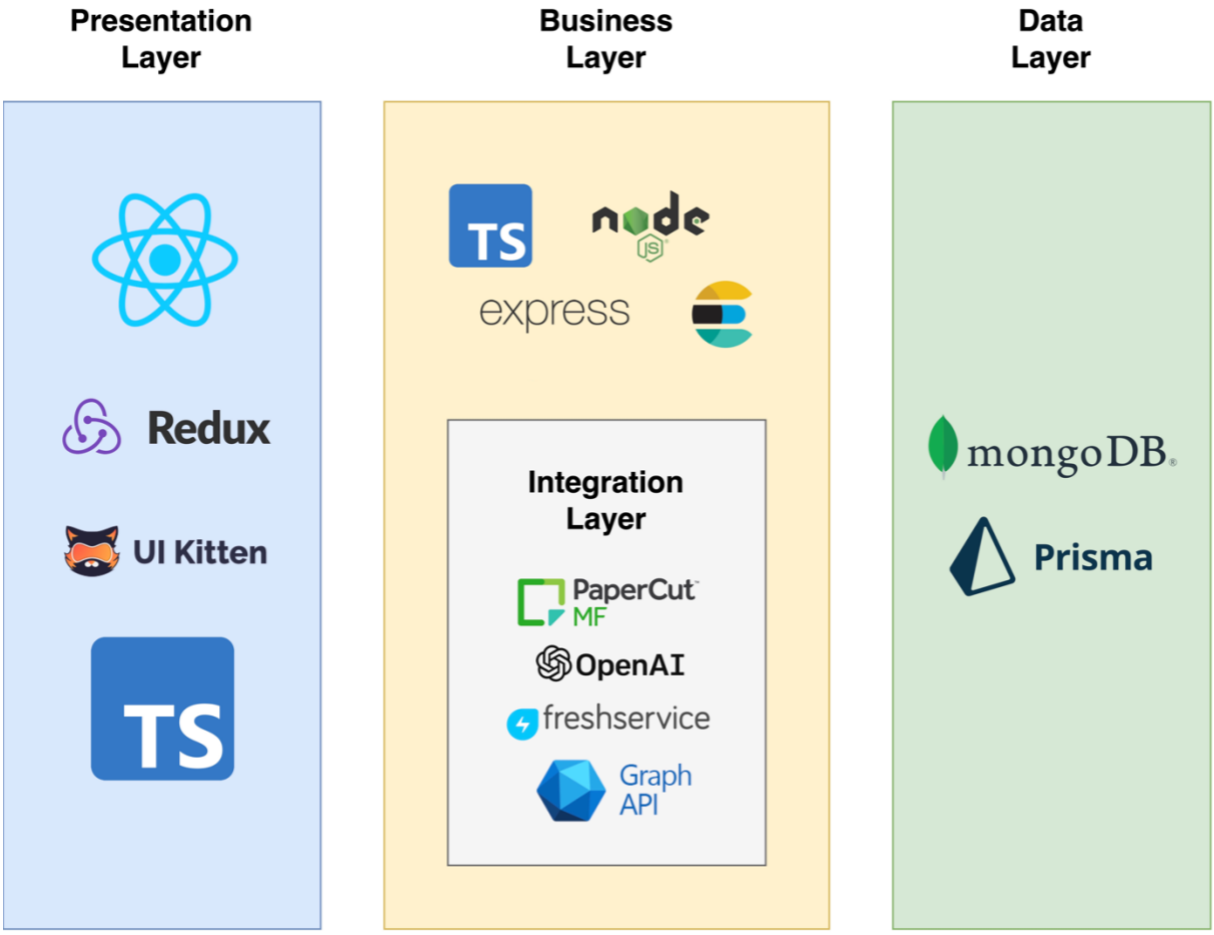


Figure 1 Software Architecture Diagram

6.2 TECHNOLOGY ENABLERS

In an N-tiered architecture for the described project, you can divide the application into the following layers:

1. Presentation Layer (Frontend):
   - React Native: Used as the primary framework for building the mobile application user interface.
   - UI Kitten: Integrated with React Native to provide a comprehensive set of UI components and styles.
   - Redux: Utilized as a state management library to manage the application's global state efficiently.
   - TypeScript: Applied as the primary programming language for both frontend and backend, ensuring type safety and better maintainability.

2. Business Logic Layer (Backend):
   - Node.js: Acts as the runtime environment for executing server-side JavaScript code.
   - Express: A minimal and flexible Node.js web application framework, used for building the backend APIs.
   - Elasticsearch: A powerful, scalable, and real-time search and analytics engine that enables rapid data indexing, retrieval, and analysis for backend applications.
   - TypeScript: As mentioned earlier, TypeScript ensures type safety and better maintainability on the backend as well.

3. Integration Layer:
   - External API Integrations: The backend connects to various external APIs such as OpenAI, Microsoft Graph API, FreshService API, Papercut XML Web Services API, and Payment system API to provide extended functionality and access to third-party services.
   - Middleware: Custom middleware can be developed to handle authentication, caching, error handling, and rate-limiting, among other tasks.

4. Data Access Layer:

- MongoDB: Serves as the primary database for storing user-related data.
- Prisma: is an open-source next-generation ORM that simplifies and streamlines data access through a type-safe, auto-generated, and intuitive API for modern application data layers.

In this architecture, the frontend (Presentation Layer) communicates with the backend (Business Logic Layer) through RESTful APIs or GraphQL, depending on the better alternative. The Business Logic Layer processes the incoming requests, performs the required operations, and interacts with the Integration Layer to utilize external APIs when needed. Finally, the Data Access Layer is responsible for managing interactions with the MongoDB database to store, update, or retrieve user-related data.

7 SYSTEM DESIGN

In the Design section, I will present the architectural blueprint of the AUI Campus application, including its data model, and any algorithms utilized in it. This section serves as a visual representation and detailed guide to the structure and relationships between various components of the system. The aim is to provide a clear understanding of the application's design, enabling efficient development and implementation.

7.1 DATA MODEL

The data model describes the structure of the database and the relationships between its entities.

The AUI Campus mobile application schema consolidates university resources and contains the following entities:

- User: Represents users with roles, personal information, and relations to user groups, content posts, and administered contacts.
- UserGroup: Represents groups of users with associated events.
- Event: Represents events with descriptions, time frames, and relations to categories and user groups.
- EventCategory: Represents event categories with associated events.
- Location: Represents locations with descriptions, coordinates, and categories.
- LocationEventCategory: Represents location categories with associated locations.
- Contact: Represents contacts with personal information, roles, office hours, and related organizations.
- Organization: Represents organizations with descriptions, main contacts, content posts, images, categories, and roles.
- Role: Represents roles with titles, descriptions, and relations to contacts and organizations.

Additionally, several join tables are present for many-to-many relationships, and an Enum UserRole defines user roles.

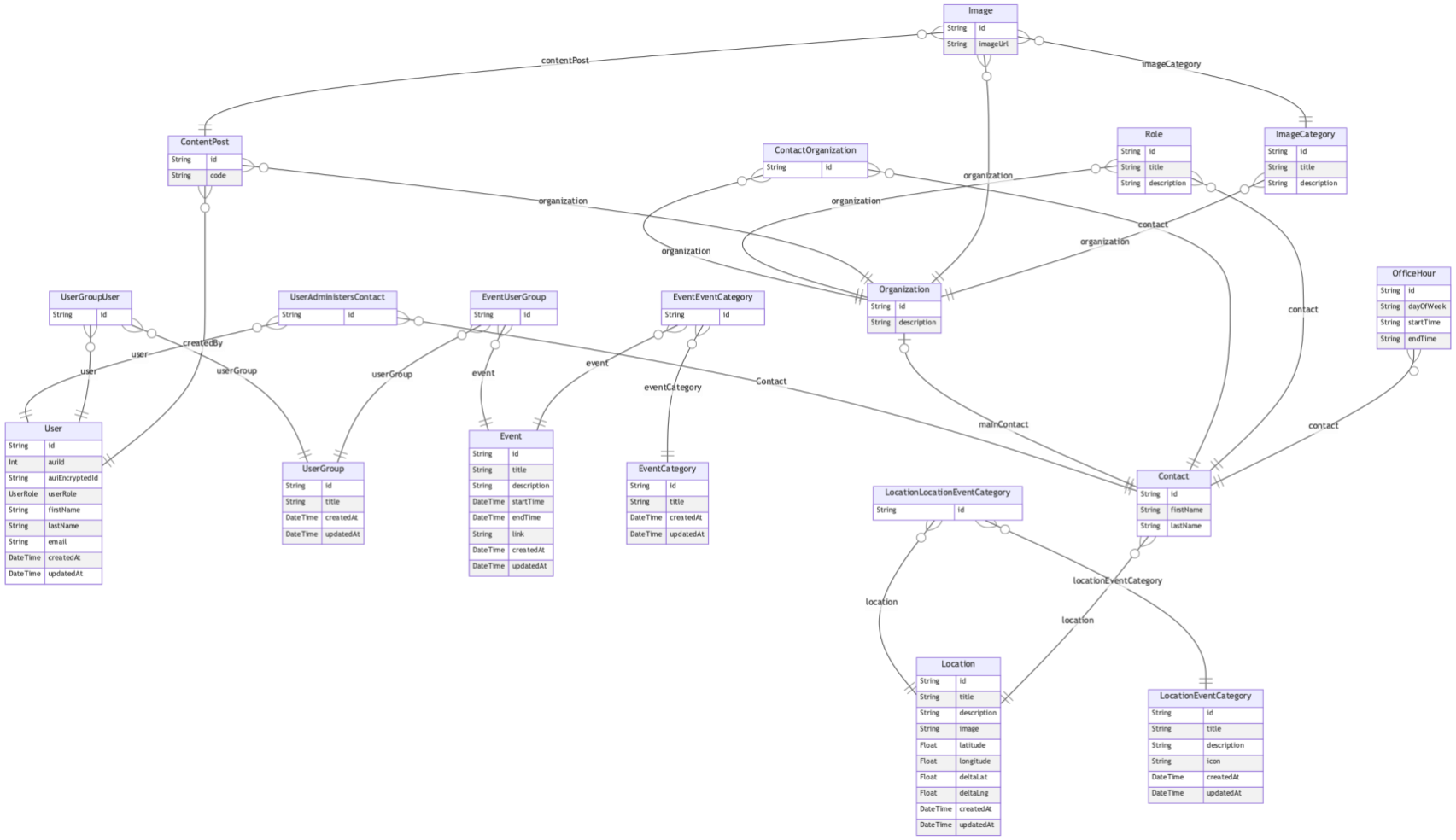


Figure 2 Entity Relationship Diagram of the MongoDB Database

7.2 ALGORITHMS

The AUI Campus application utilizes various algorithms for data analysis, chatbot functionality, and search engine functionality. These algorithms include natural language processing models for the chatbot and the search engine, as well as :

1. OpenAI's ChatGPT 3.5: ChatGPT 3.5 is based on the GPT-3 architecture, which uses the Transformer model. The core algorithm of the Transformer is based on the self-attention mechanism, allowing the model to weigh the importance of different words in a sequence. The training process involves unsupervised learning using a large corpus of text data. Fine-tuning is done on specific tasks or domains to improve performance. Transformers employ the techniques of masking, positional encoding, and multi-head attention to effectively understand context, dependencies, and relationships in the input text.

2. Elasticsearch: Elasticsearch utilizes an information retrieval technique called the Inverted Index for efficient search and retrieval. It tokenizes the text data, processes it through an analyzer (which can include lowercasing, stemming, and stopword removal), and then builds an index mapping terms to their occurrences in the documents. Elasticsearch uses a similarity algorithm, like the TF-IDF (Term Frequency-Inverse Document Frequency) and BM25 (Best Matching 25), to rank documents based on their relevance to a given query. The more similar a document is to the query, the higher its ranking.

3. R for Big Data Mining: R is a popular programming language and environment for statistical computing, data analysis, and visualization. While R is not an algorithm itself, it provides an extensive ecosystem of libraries and packages for big data mining and machine learning, such as:
    - *dplyr:* A library for data manipulation and transformation.
    - *ggplot2:* A library for creating complex and customizable visualizations.
    - *caret:* A library for training and tuning machine learning models.
    - *randomForest:* A library for creating decision tree-based ensemble models using the Random Forest algorithm.
    - *xgboost:* A library implementing the eXtreme Gradient Boosting algorithm, a powerful and efficient technique for supervised learning tasks.

    R also offers connectivity to various big data storage systems, such as Hadoop and Spark, enabling the analysis of large-scale datasets.

8 IMPLEMENTATION DETAILS

8.1 PRESENTATION LAYER

The presentation layer of a mobile application is responsible for displaying data and facilitating user interaction with the app's interface. In this paragraph, I will provide an overview of the different components of the presentation layer and the role they play in the app's overall functionality.

8.1.1 PROJECT CODE STRUCTURE

When it comes to developing a mobile application, having a well-organized code structure is crucial for ensuring that the project is maintainable and scalable over time. In this paragraph, I will explore the different components of the AUI Campus mobile application codebase and how they are organized. First, let us look at the folder structure of the React Native Expo Project:

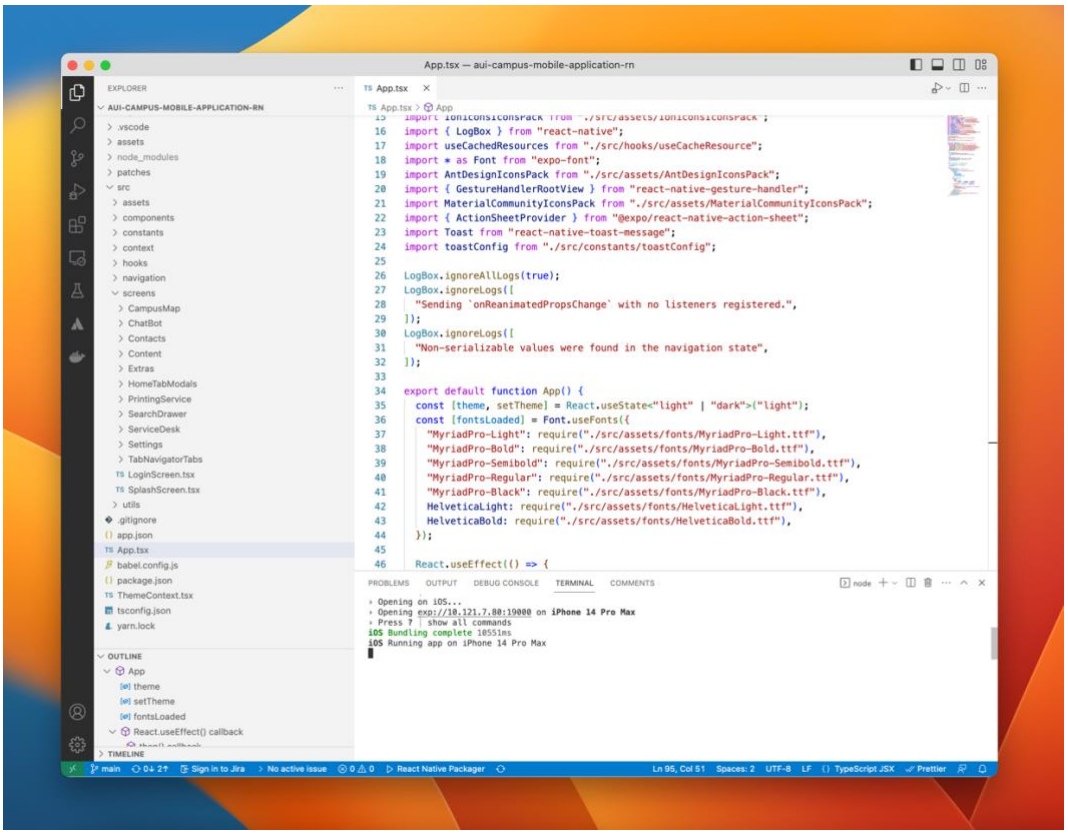


Figure 3 Visual Studio Code IDE for Mobile Application

The folder structure of an Expo project is specified to match the project's specific configuration, in our case, the project structure is relatively straightforward and easy to understand.

At the root level of the project, you will find several files and directories, including the "package.json" file, which lists the project's dependencies and configuration settings. You will also find files like ".gitignore" and ".env", which contain information about ignored files and environment variables, respectively. The "node_modules" directory contains all the dependencies that are installed via npm or Yarn. This folder is generated automatically and should not be modified manually. The "assets" folder contains static files such as images, videos, and fonts that are used in the app.

The "patches" folder contains patch files that are written in the "diff" format, which is a textual representation of the differences between two versions of a file. The patch files are applied using the "patch-package" tool, which automatically modifies the third-party package's source code before it is used in the project.

The "src" folder is where most of the application code is located. Inside the "src" folder, you will find several subfolders, including "components," "screens," "navigation," and "utils," each with its own specific purpose:

- The "components" folder contains reusable UI components used throughout the app.
- The "screens" folder contains the individual screens of the app, each with its own file.
- The "navigation" folder contains the navigation configuration for the app.
- The "utils" folder contains utility functions and other helper code.

Overall, the folder structure of an Expo project is designed to be easy to navigate and maintain, with clear organization and separation of concerns between different parts of the application.

Next, let us explore, and understand the "package.json" file:

```
{
  "name": "aui-campus-rn",
  "version": "1.0.0",
  "main": "node_modules/expo/AppEntry.js",
  "scripts": {
    "start": "expo start",
    "android": "expo start --android",
    "ios": "expo start --ios",
    "web": "expo start --web",
```

```
    "postinstall": "patch-package"
  },
  "dependencies": {
    "@eva-design/eva": "^2.1.1",
    "@expo/react-native-action-sheet": "^4.0.1",
    "@expo/vector-icons": "^13.0.0",
    "@flyerhq/react-native-chat-ui": "^1.4.3",
    "@gorhom/bottom-sheet": "^4.4.5",
    "@kichiyaki/react-native-barcode-generator": "^0.6.7",
    "@react-native-async-storage/async-storage": "~1.17.3",
    "@react-native-community/datetimepicker": "6.7.3",
    "@react-navigation/bottom-tabs": "^6.5.3",
    "@react-navigation/drawer": "^6.5.0",
    "@react-navigation/native": "^6.1.2",
    "@react-navigation/native-stack": "^6.9.0",
    "@react-navigation/stack": "^6.3.1",
    "@ui-kitten/components": "^5.1.2",
    "dayjs": "^1.11.7",
    "expo": "48.0.9",
    "expo-auth-session": "~4.0.3",
    "expo-document-picker": "^11.2.2",
    "expo-font": "11.1.1",
    "expo-image-picker": "^14.1.1",
    "expo-local-authentication": "^13.2.1",
    "expo-splash-screen": "^0.16.2",
    "expo-web-browser": "~12.1.1",
    "f-react-native-schedule": "^0.1.5",
    "patch-package": "^6.5.1",
    "react": "18.1.0",
    "react-native": "0.71.3",
    "react-native-bouncy-checkbox": "^3.0.7",
    "react-native-dropdown-picker": "^5.4.6",
    "react-native-gesture-handler": "^2.9.0",
    "react-native-maps": "^1.4.0",
    "react-native-qrcode-svg": "^6.2.0",
    "react-native-reanimated": "~2.9.1",
    "react-native-reanimated-carousel": "^3.0.6",
    "react-native-safe-area-context": "4.3.1",
    "react-native-screens": "S~3.15.0",
    "react-native-section-alphabet-list": "^3.0.0",
    "react-native-svg": "13.4.0",
    "react-native-toast-message": "^2.1.6",
    "react-native-web": "~0.18.7",
    "victory-native": "^36.6.8"
  },
  "devDependencies": {
```

```
    "@babel/core": "^7.12.9",
    "@types/react": "~18.0.14",
    "@types/react-native": "~0.69.1",
    "typescript": "^4.9.4"
  },
  "private": true
}
```

Figure 4 "package.json" in the React Native Project

This is the package.json file for a React Native mobile application called "aui-campus-rn". The file contains important information about the app's name, version number, and dependencies, as well as configuration settings for running the app locally and building the app for production.

Under the "scripts" section, there are commands for starting the app in various environments, including "start" for running the app in development mode, "android" for running the app on an Android device or emulator, "ios" for running the app on an iOS device or simulator, and "web" for running the app in a web browser. There is also a "postinstall" script that runs the "patch-package" tool, which is used to apply patches to dependencies in the "node_modules" folder.

The "dependencies" section lists all the external libraries and packages that the app depends on, including UI frameworks like UI Kitten and Eva Design, navigation packages like React Navigation, and third-party libraries for features like barcode generation, chat interfaces, and maps. The "devDependencies" section lists packages that are only needed during development, such as TypeScript for type checking and Babel for transpiling code. The "private" field is set to true, indicating that the app is not intended to be published as an npm package.

Overall, the package.json file is a critical part of a React Native project as it contains essential configuration details and dependencies that the application relies on. It helps to ensure that the application can run smoothly, and the dependencies are correctly managed throughout the app's development and deployment process.

8.1.2 USER EXPERIENCE

In addition to the technical aspects of a mobile application, the user experience (UX) is a critical factor in determining the success of the product. A great UX can be the difference between a user

abandoning the app after a few minutes and becoming a loyal customer. In this paragraph, I will discuss the different elements that contribute to a good user experience, such as intuitive navigation, clear and concise messaging, and engaging visuals. Here are all the user stories, subdivided into the following themes, implemented (so far) in the application:

8.1.2.1 SIGN IN

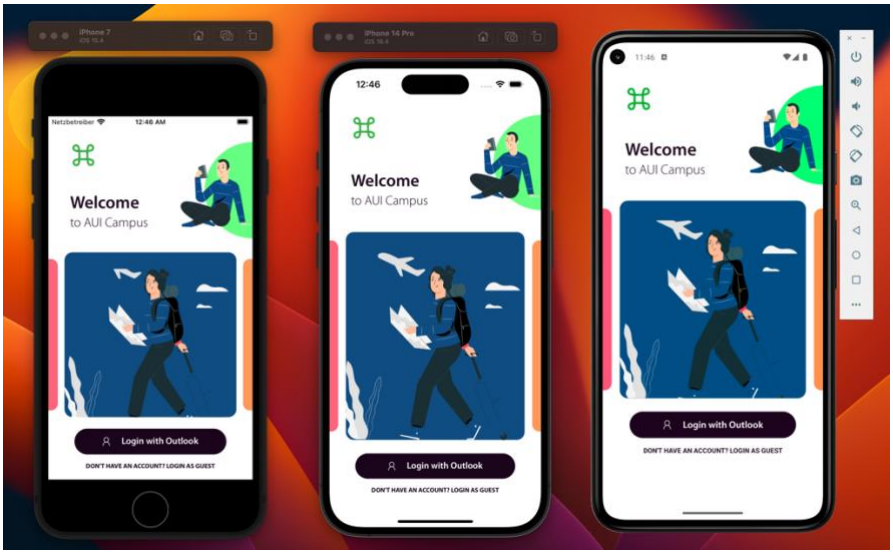


Figure 5 An Individual Can Sign In Using Oultook or as a Guest

8.1.2.2 MAIN OVERVIEW

Figure 6 A User has an Overview of all Upcoming Events, both Personal and Public

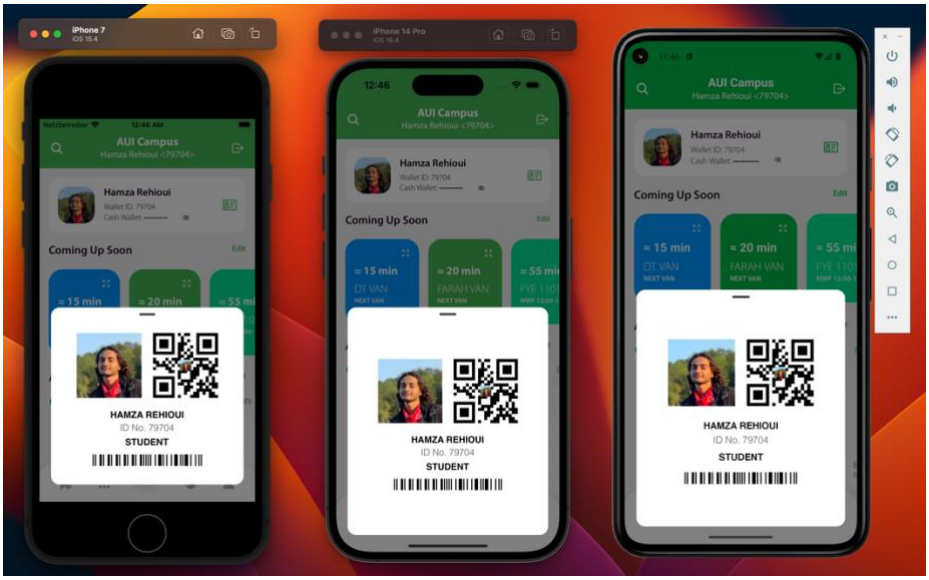


Figure 7 A User Can Pull Up Their Cash Wallet Credentials

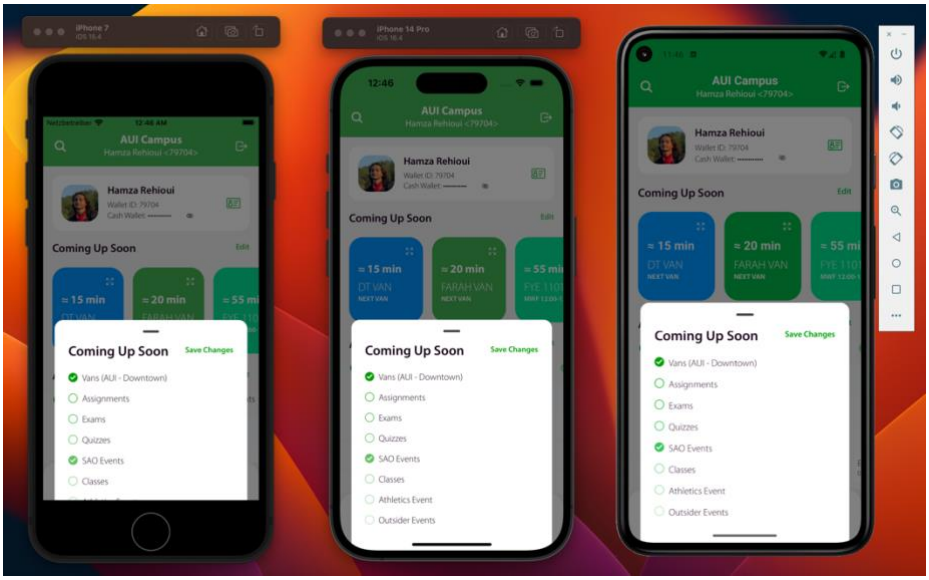


Figure 8 A User Can Edit their Coming Up Soon Cards

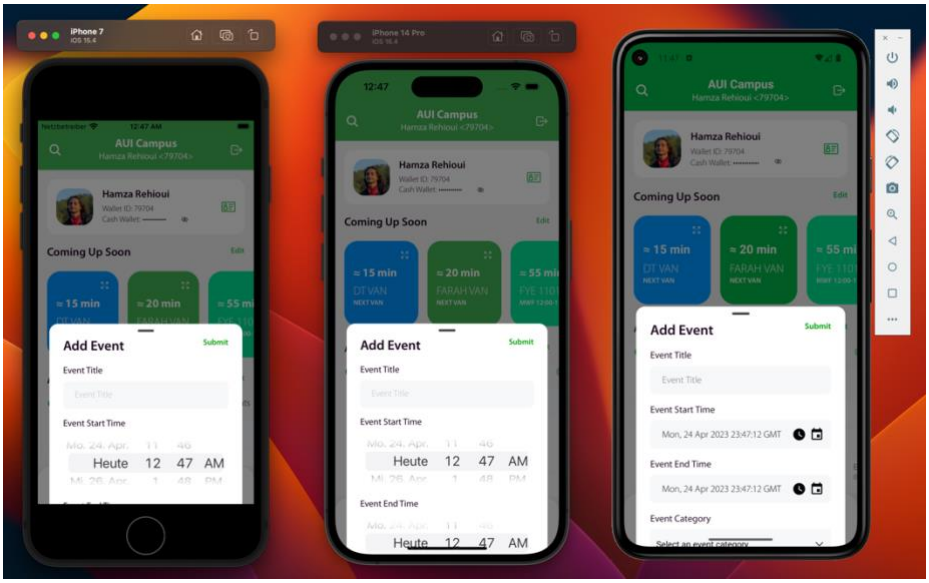


Figure 9 A User Can Add an Event to the Events Calendar

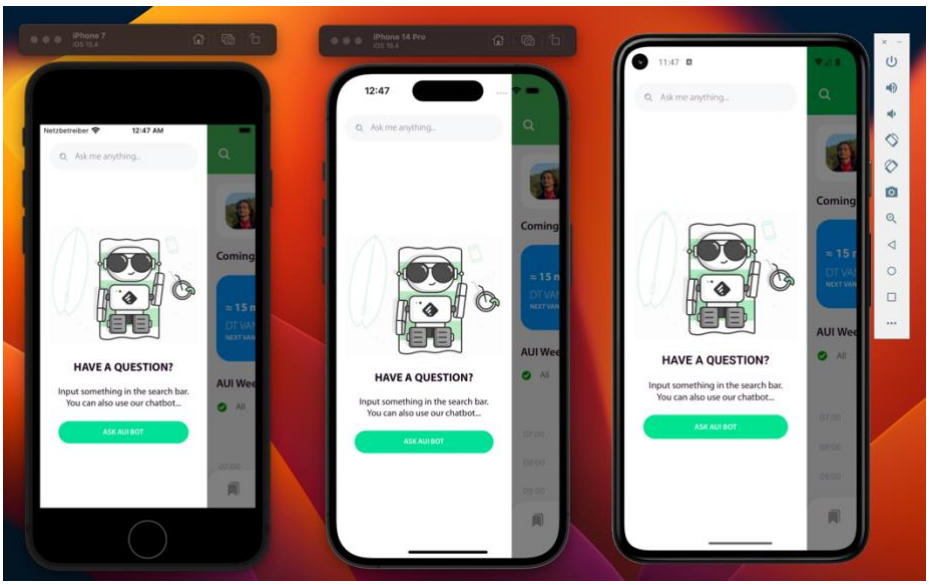


Figure 10 A User Can Use the Search Engine from Anywhere in the Application

8.1.2.3 CAMPUS ORGANIZATIONS

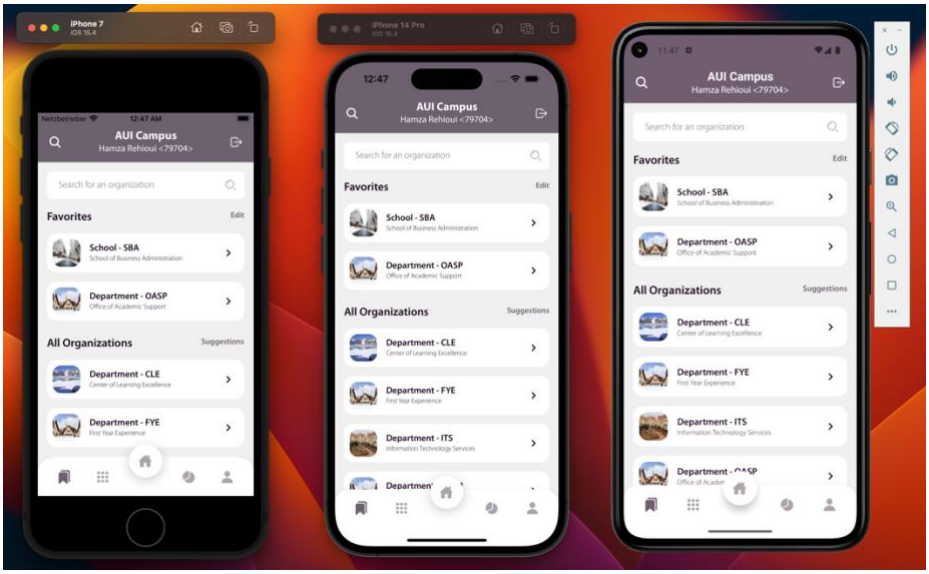


Figure 11 A User Has Access to all Content Dashboards of Campus Organizations

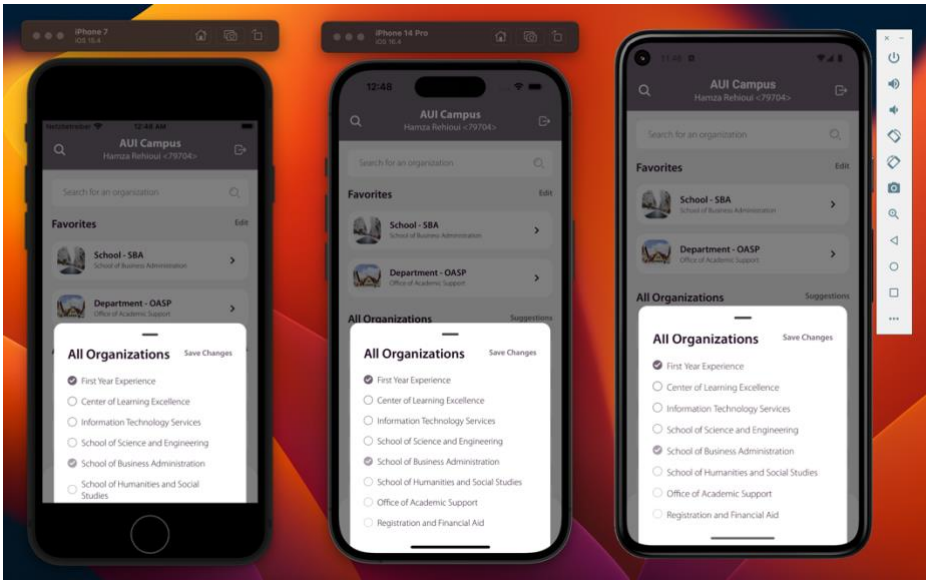


Figure 12 A User Can Add and Remove Favorite Campus Organizations

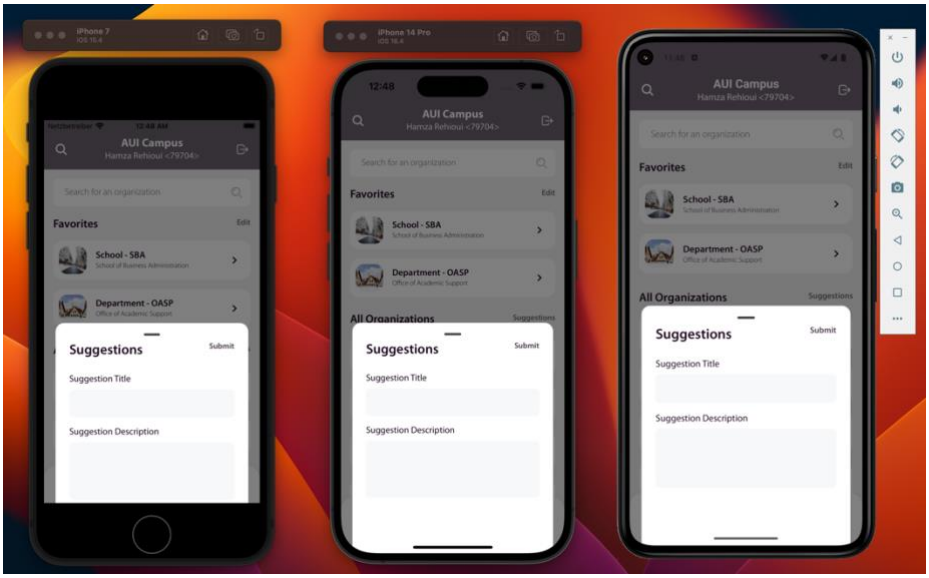


Figure 13 A User Can Send Suggestions Concerning Campus Organizations Dashboards

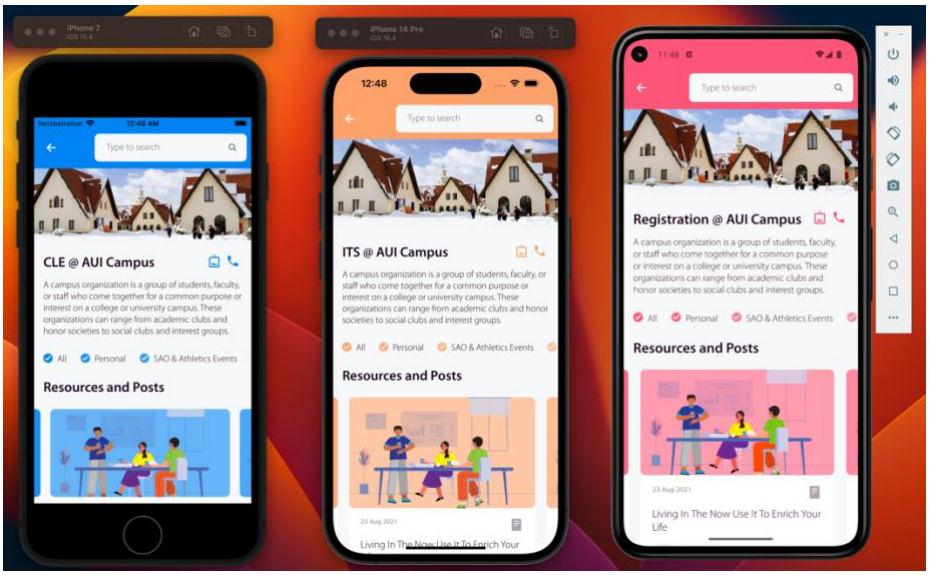


Figure 14 A User Has Access to any Campus Organization's Dashboard

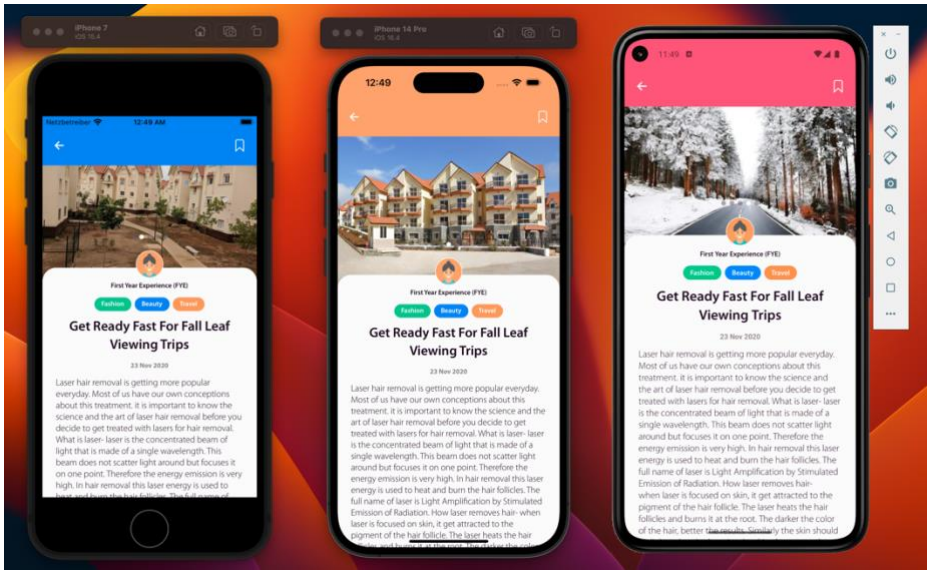


Figure 15 A User Has Access to any Campus Organization's Content Post

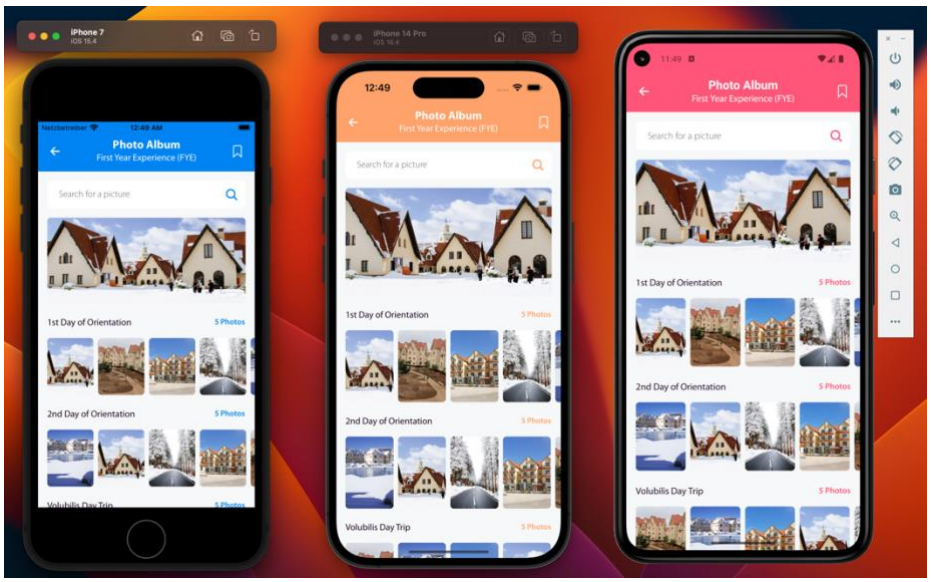


Figure 16 A User Has Access to any Campus Organization's Photo Album

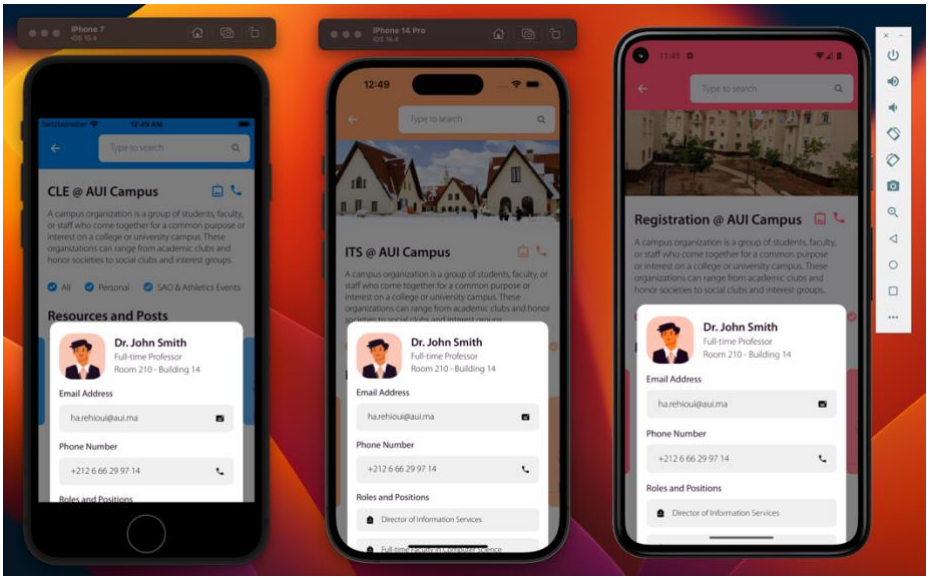


Figure 17 A User Has Direct Access to any Campus Organization's Main Point of Contact

8.1.2.4 APPLICATIONS THEME

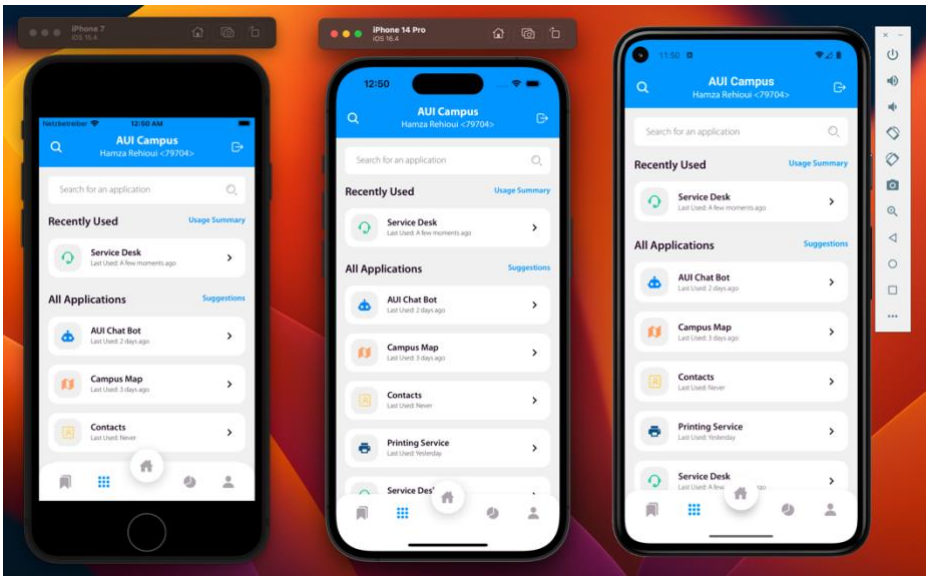


Figure 18 A User Can See All Applications Available to Them

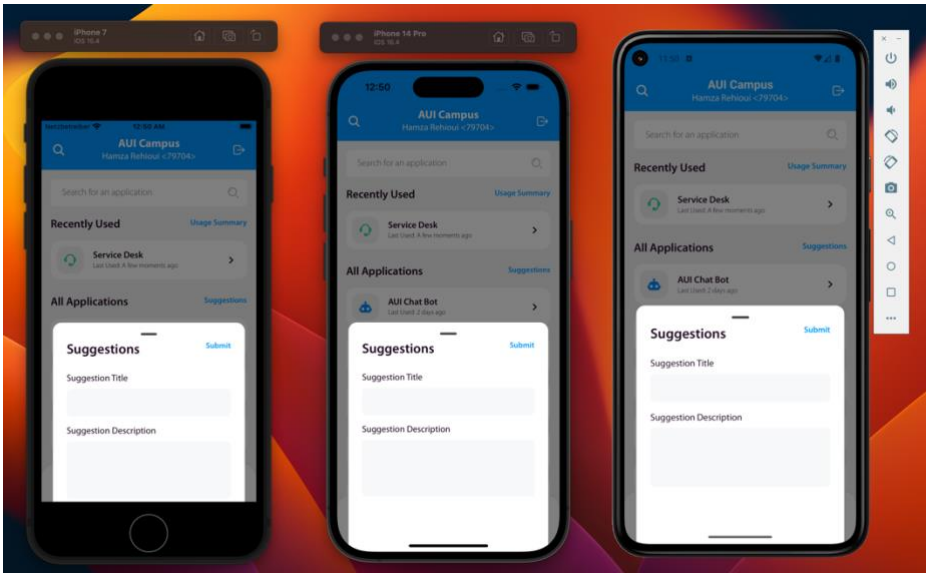


Figure 19 A User Can Send Suggestions Concerning Applications

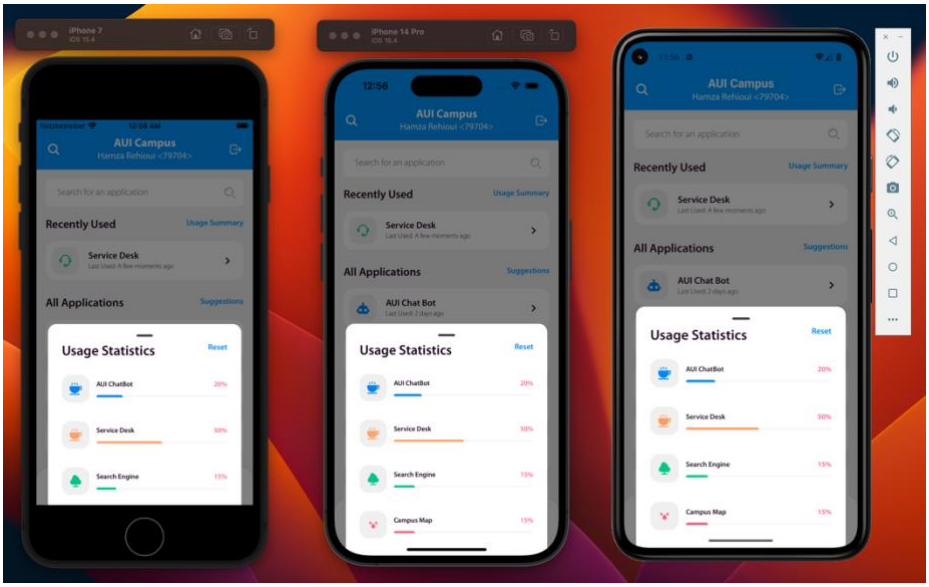


Figure 20 A User Can Have an Overview of their Usage Summary of Applications

8.1.2.5 TRANSACTIONS THEME

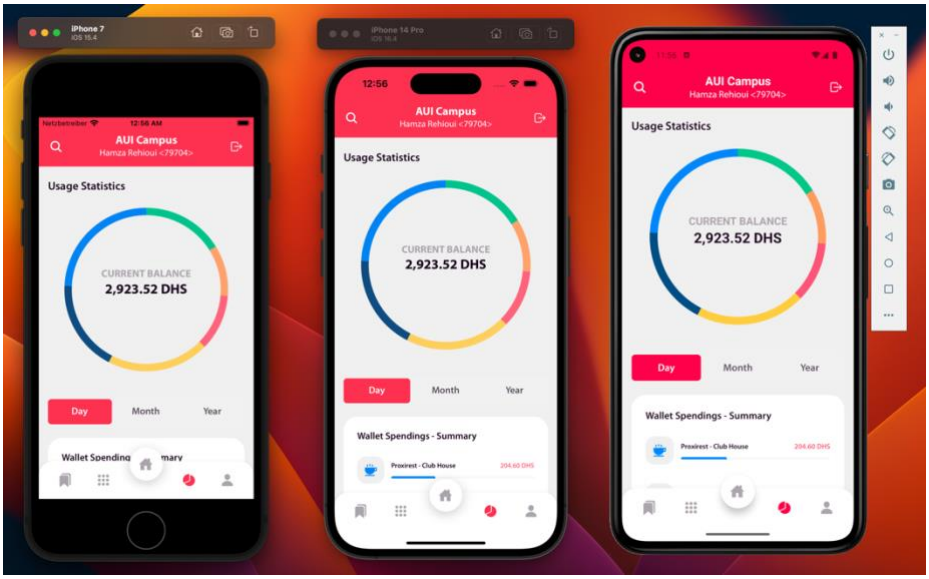


Figure 21 A User Can Have an Overview of their Wallet's Transactions

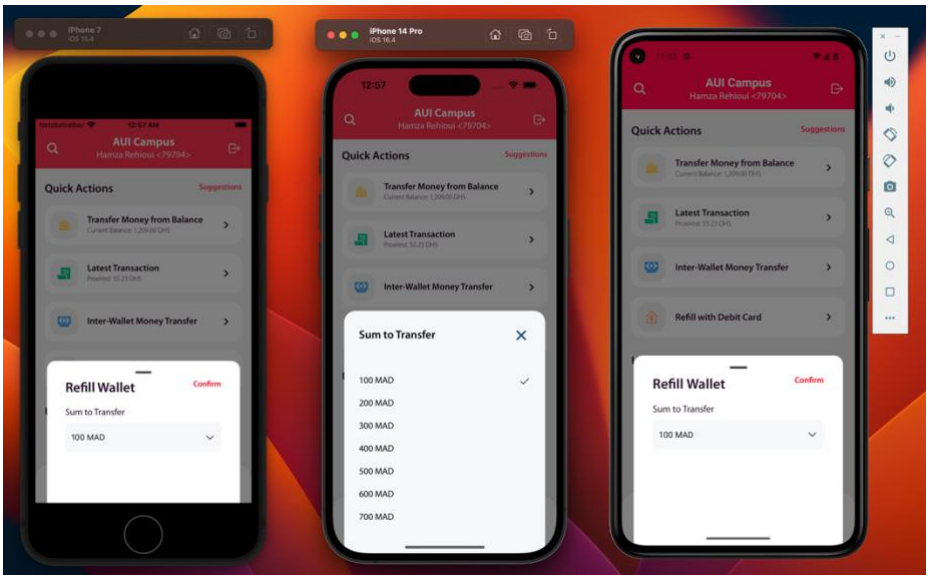


Figure 22 A User Can Transfer Money from their Balance

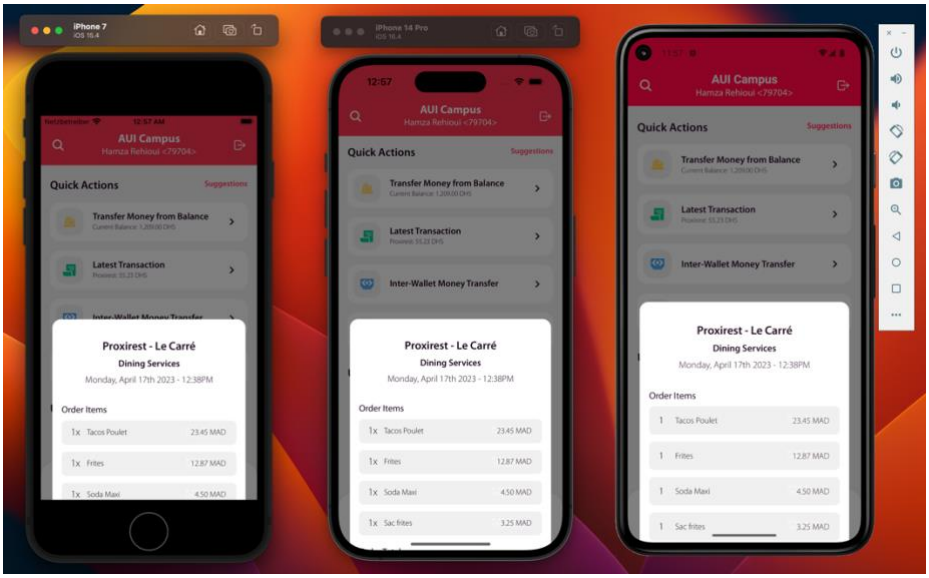


Figure 23 A User Can Check Out Their Latest Transaction

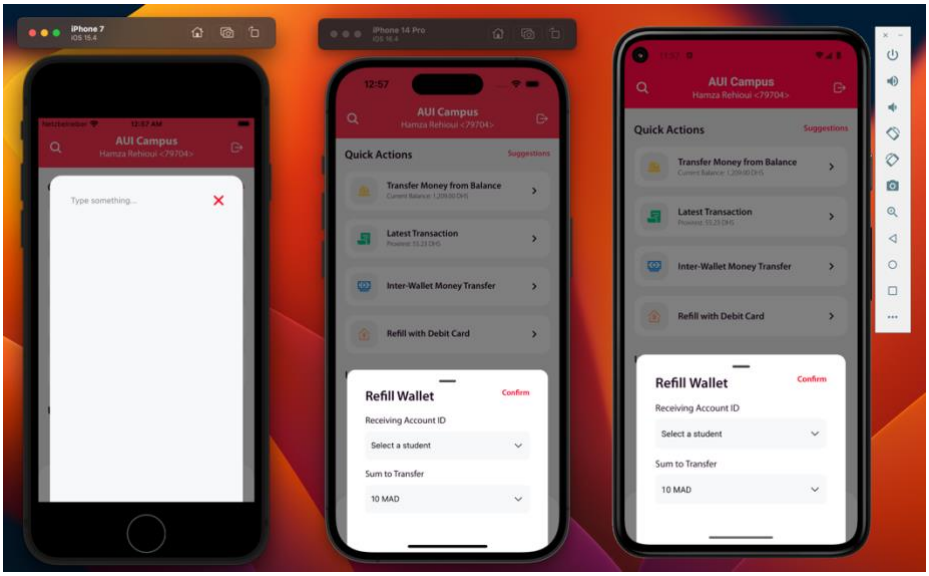


Figure 24 A User Can Transfer Money from their Wallet to Another User's

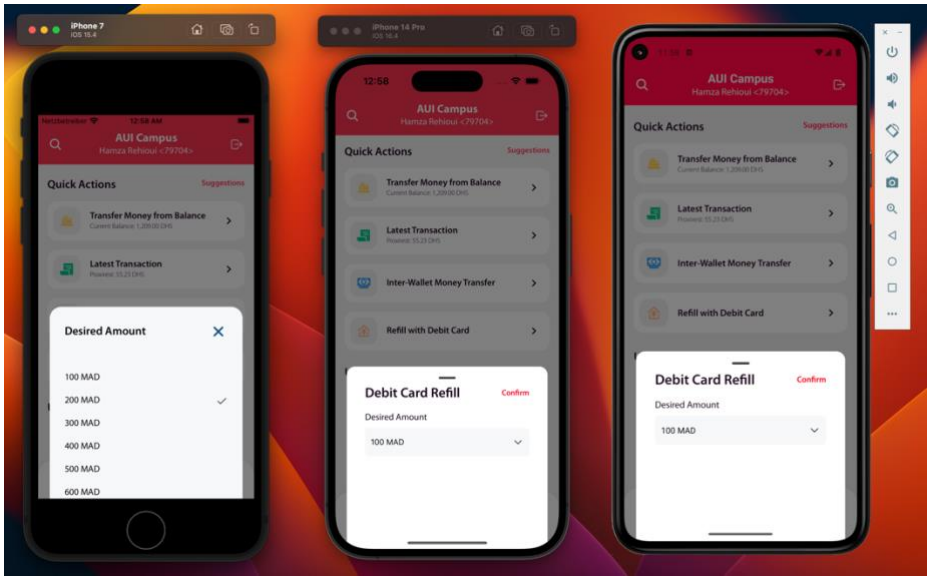


Figure 25 A User Can Refill their Wallet Using their Debit Card

8.1.2.6 PROFILE SETTINGS THEME

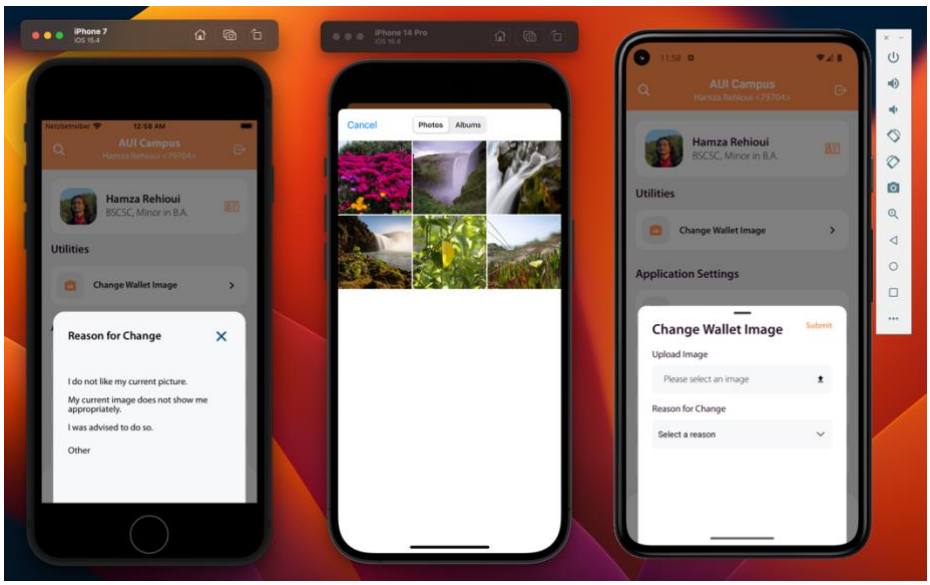


Figure 26 A User Can Change the Image on their Wallet

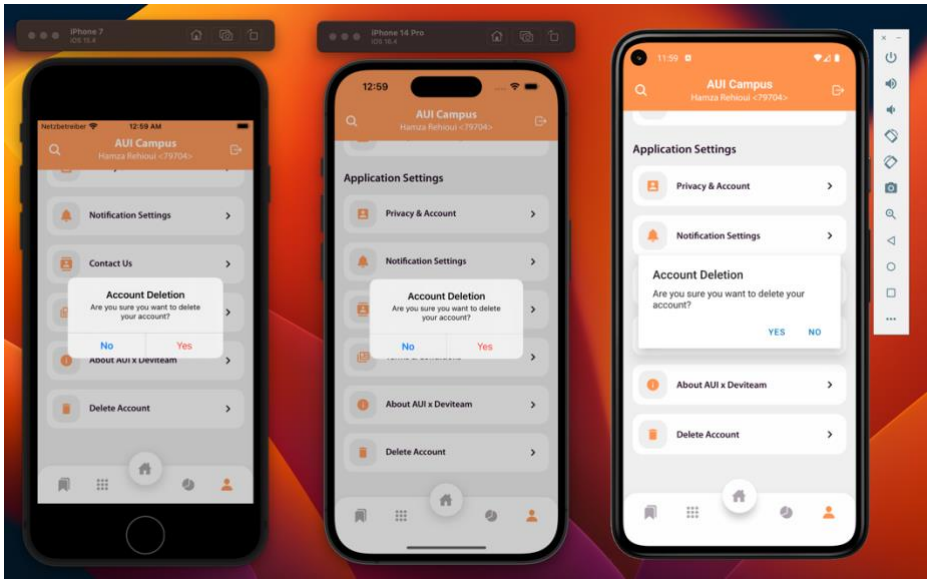


Figure 27 A User Can Delete their Account

8.1.2.7 CHATBOT THEME

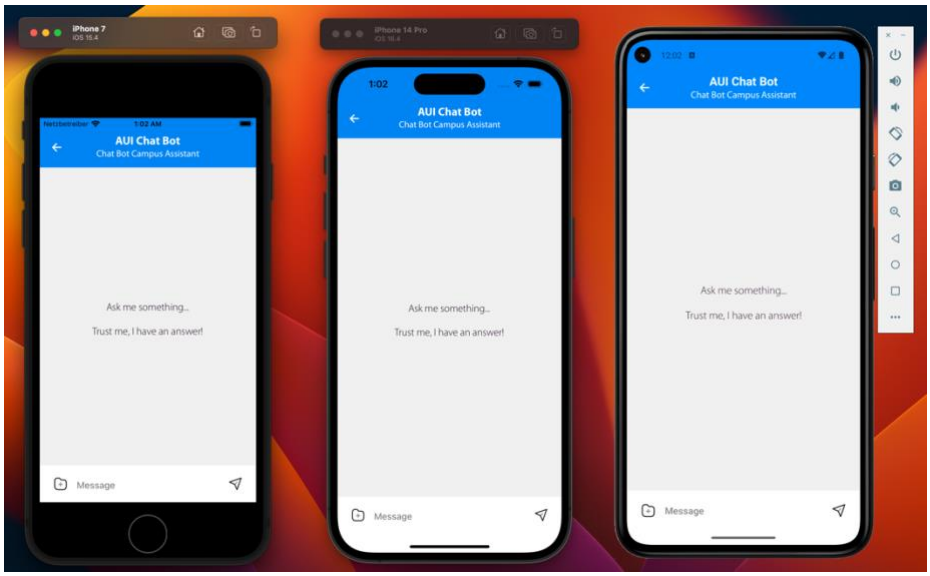


Figure 28 A User Can Converse with a Chatbot

8.1.2.8 CAMPUS MAP THEME

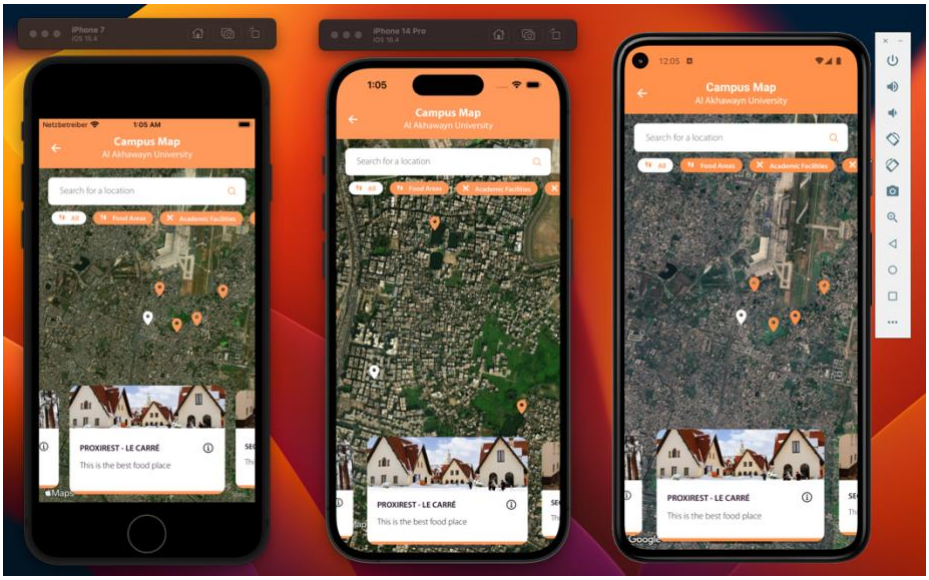


Figure 29 A User Can Browse Through Different Locations and Learn About Them

8.1.2.9 CONTACT THEME

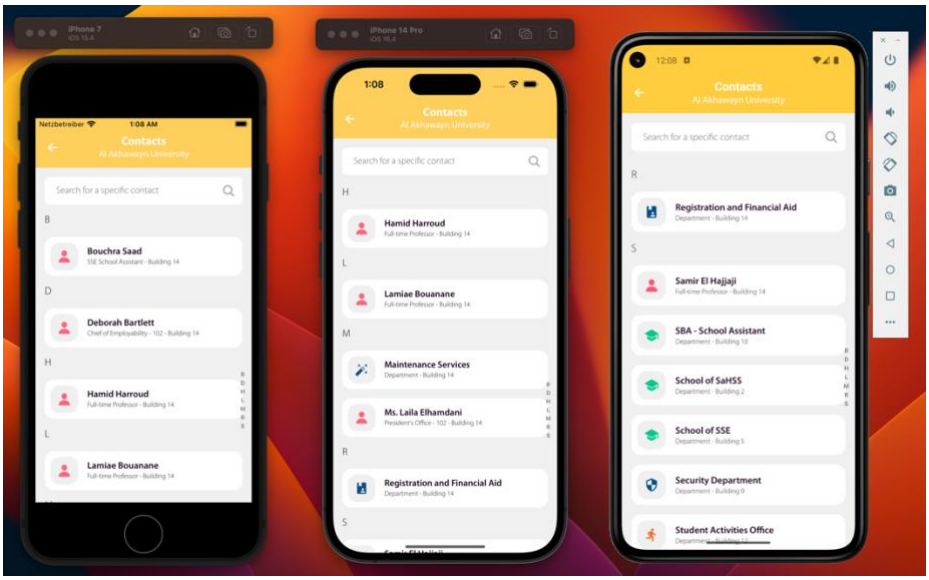


Figure 30 A User Can Browse Through All Contacts

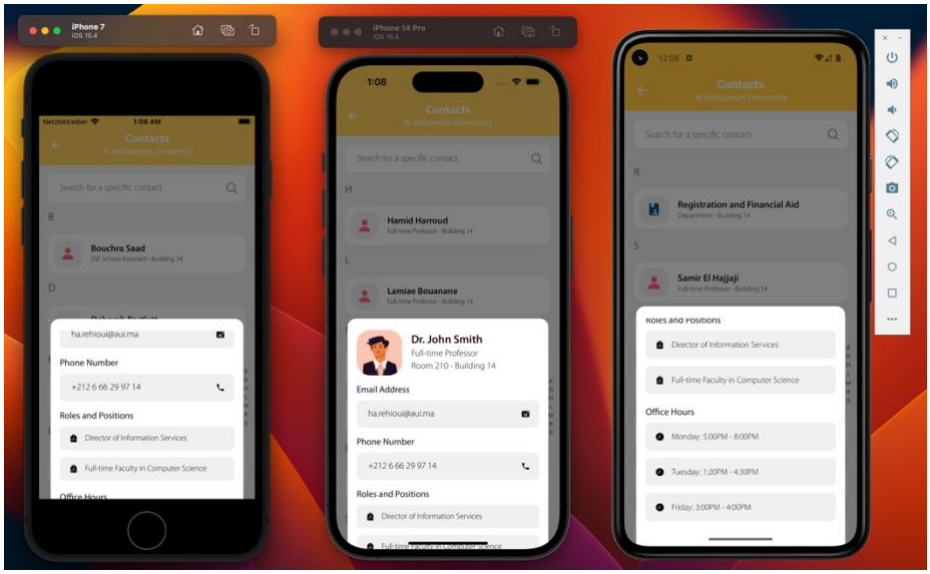


Figure 31 A User Can Check Out Information of a Specific Contact

8.1.2.10 PRINTING THEME

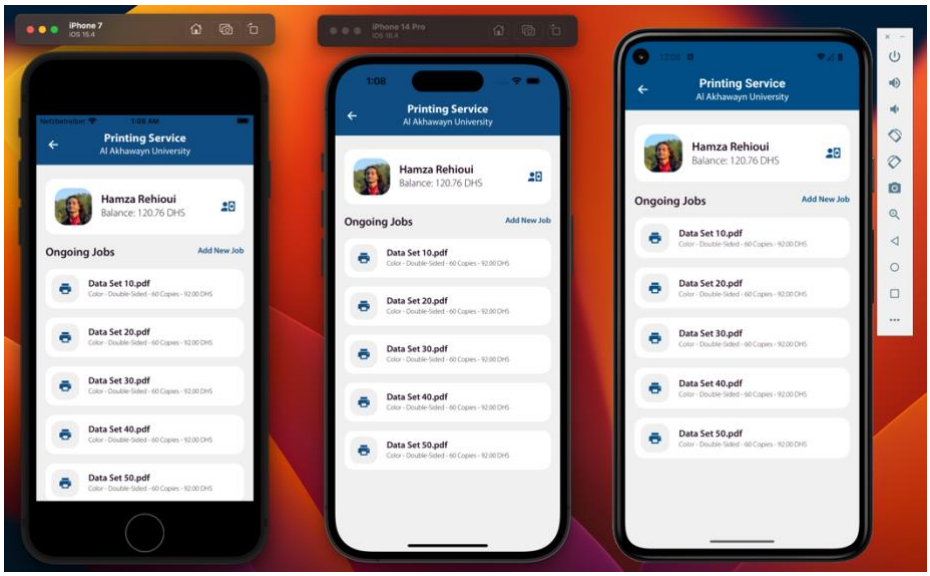


Figure 32 A User Can See All Their Ongoing Jobs

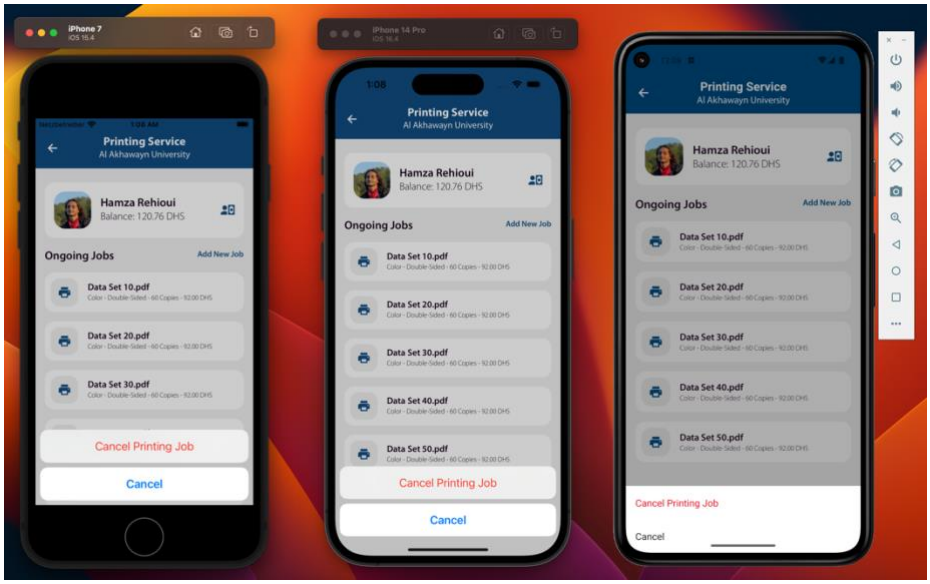


Figure 33 A User Can Cancel an Ongoing Job

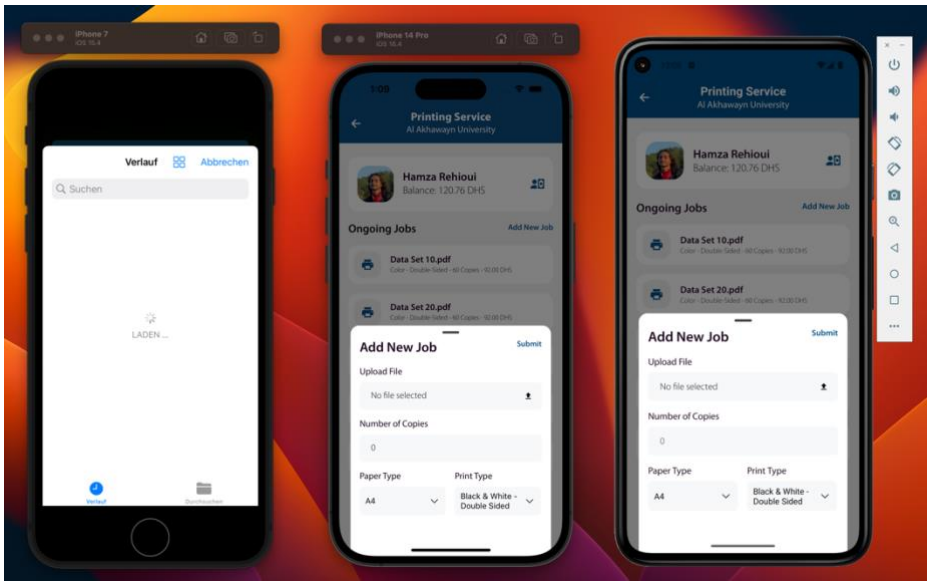


Figure 34 A User Can Add a New Job

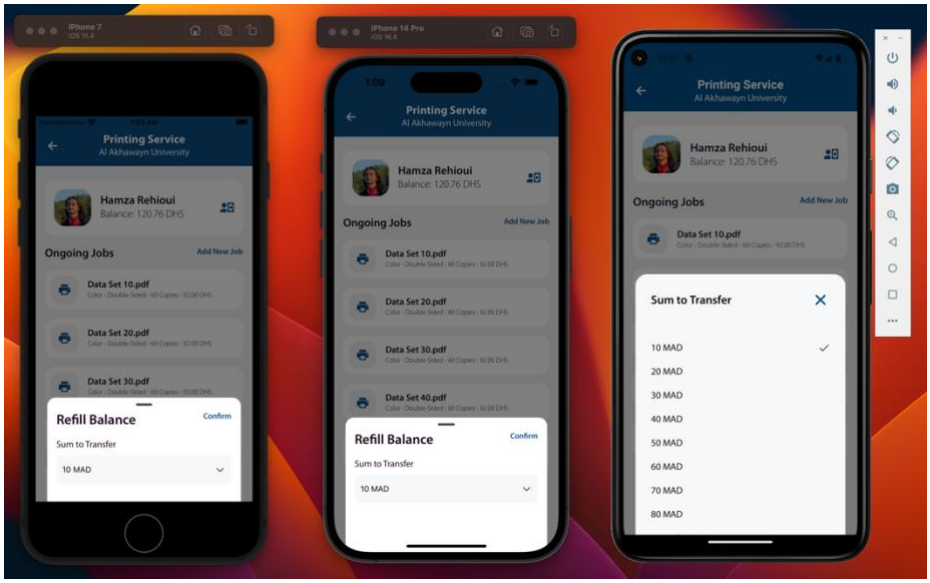


Figure 35 A User Can Refill Their Printing Balance

8.1.2.11 SERVICE DESK THEME

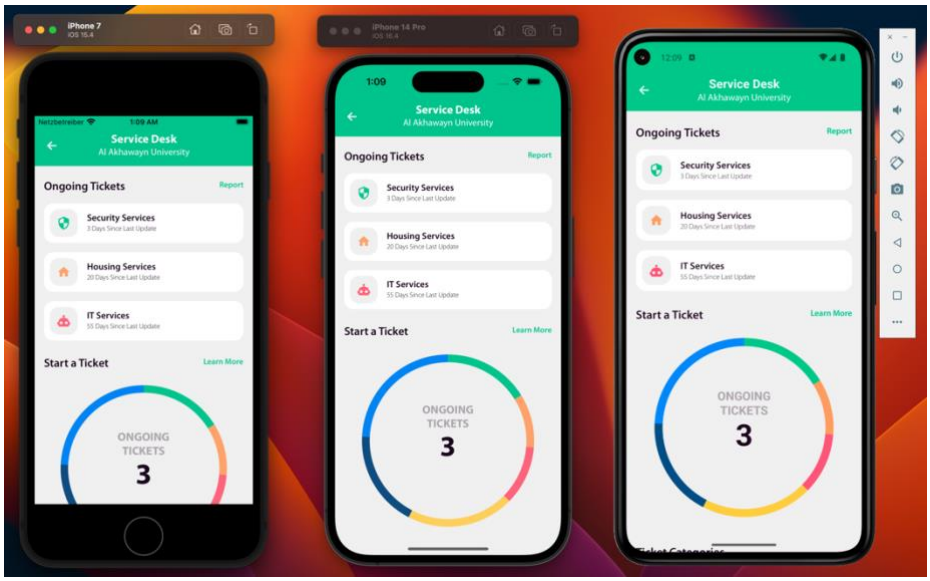


Figure 36 A User Can See their Ongoing Tickets

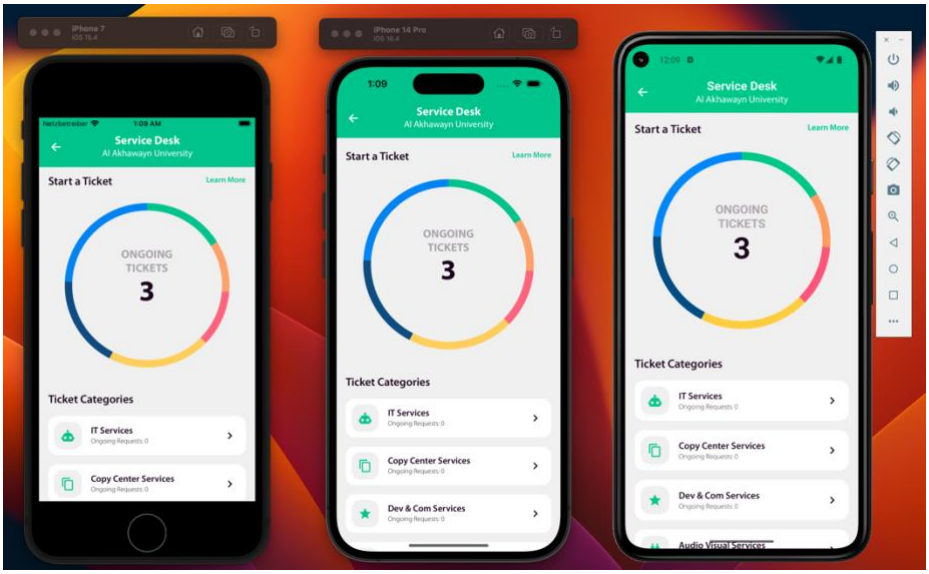


Figure 37 A User Can Start a Ticket

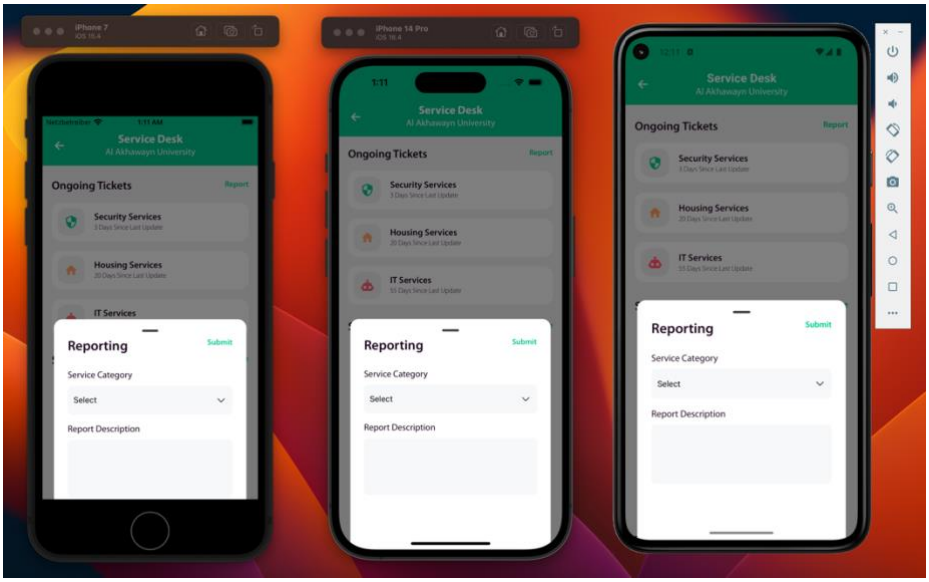


Figure 38 A User Can Report the Service Desk

8.2 BUSINESS LAYER

Next, when it comes to developing a server, having a well-organized code structure is as crucial, if not more, for ensuring that the project is maintainable and scalable over time. In this paragraph, I will explore the different components of the AUI Campus Server codebase and how it is organized. First, let us look at the folder structure of the Node Express Project:

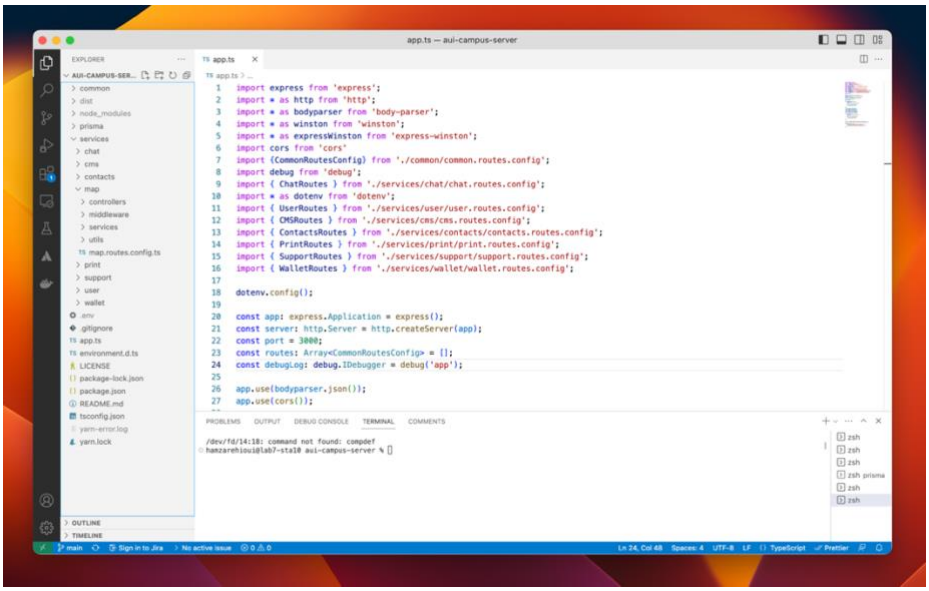


Figure 39 Visual Studio Code IDE for Mobile Application

A typical folder structure for a Node Express Server consists of several subfolders that contain various parts of the application's code. These subfolders are organized according to the separation of concerns principle, which aims to keep related code grouped together and minimize the overlap between different parts of the application.

Here are the implementation details for each of the subfolders in a service folder:

- Controllers: The "controllers" subfolder contains modules that define the logic for handling HTTP requests and responses. Each controller module corresponds to a specific resource or endpoint in the API and is responsible for validating incoming requests, interacting with the relevant service or model layer to retrieve or manipulate data, and formatting the response to be returned to the client.

- Middleware: The "middleware" subfolder contains modules that define middleware functions that are used to modify or intercept incoming requests and outgoing responses. Middleware functions are used to add functionality such as authentication, error handling, request logging, or parsing request bodies before they are passed on to the relevant controller or service module.

- Services: The "services" subfolder contains modules that define the business logic and data access layer functionality for a specific feature or resource in the API. Services are used by the controller modules to interact with the database or external services and encapsulate the logic for handling data manipulation or aggregation.

- Utils: The "utils" subfolder contains modules that define utility functions or helper classes that are used across the application. These functions or classes are not specific to a particular feature or resource but are instead general-purpose tools that can be used across different parts of the application.

Overall, this folder structure provides a well-organized and modular approach to building Node Express Server applications. It promotes the separation of concerns and makes it easier to develop and maintain code as the application grows and evolves over time.

This is a package.json file for a Node.js server application called "AUI Campus REST API". Let us explore and understand it:

```
{
 "name": "aui-campus-server",
 "version": "0.0.1",
 "description": "AUI Campus REST API",
 "main": "index.js",
 "scripts": {
  "start": "tsc && node ./dist/app.js",
  "debug": "export DEBUG=* && npm start",
  "test": "echo \"Error: no test specified\" && exit 1",
  "ingest": "tsx -r dotenv/config chat/utils/ingestData.ts"
 },
 "prisma": {
  "seed": "ts-node prisma/seed.ts"
 },
 "keywords": [
  "REST",
  "API",
  "ExpressJS",
  "NodeJS"
 ],
 "author": "Al Akhawayn University",
 "license": "MIT",
 "dependencies": {
  "@elastic/elasticsearch": "^8.7.0",
  "@huggingface/inference": "^1.8.0",
```

```
    "@pinecone-database/pinecone": "^0.0.12",
    "@prisma/client": "^4.13.0",
    "@supabase/supabase-js": "^2.20.0",
    "chromadb": "^1.3.1",
    "cohere-ai": "^6.2.0",
    "cors": "^2.8.5",
    "debug": "^4.2.0",
    "express": "^4.18.2",
    "express-winston": "^4.0.5",
    "hnswlib-node": "^1.4.2",
    "langchain": "^0.0.55",
    "openai": "^3.2.1",
    "puppeteer": "^19.9.0",
    "redis": "^4.6.5",
    "replicate": "^0.10.0",
    "tsx": "^3.12.6",
    "typeorm": "^0.3.14",
    "winston": "^3.3.3"
  },
  "devDependencies": {
    "@types/cors": "^2.8.7",
    "@types/debug": "^4.1.5",
    "@types/express": "^4.17.17",
    "@types/express-serve-static-core": "^4.17.30",
    "@types/node": "^18.15.11",
    "tslint": "^6.0.0",
    "typescript": "^5.0.4"
  }
}
```

Figure 40 "package.json" in the Node Express Project

The "main" field specifies the entry point for the application, which in this case is "index.js". The "scripts" field defines several scripts that can be run using the npm command, including a "start" script that compiles the TypeScript code and runs the resulting JavaScript code, and a "debug" script that sets a debugging flag and runs the "start" script.

The "dependencies" field lists the external dependencies of the application, including several popular Node.js libraries such as Express, Winston, and Redis. These libraries provide functionality such as web server handling, logging, and data caching. The "devDependencies" field

lists the development dependencies of the application, including TypeScript, which is used to compile the TypeScript code into JavaScript.

The "prisma" field specifies the seed file to use for the Prisma ORM data access layer. The Prisma library provides a type-safe and expressive way to interact with databases in Node.js applications. Overall, the package.json file provides important metadata about the application, as well as a list of its dependencies and scripts that can be used to build, test, and run the application.

8.3 DATA LAYER

The data layer described in the schema is implemented using Prisma, a popular open-source Object-Relational Mapping (ORM) library for Node.js and TypeScript. Prisma is used for working with databases in a type-safe and developer-friendly way. In this case, the database used is MongoDB, a NoSQL database.

```
datasource db {
  provider = "mongodb"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}

// Define the User model
model User {
  id                String              @id @default(auto()) @map("_id") @db.ObjectId
  auiId             Int?
  auiEncryptedId    String?
  userRole          UserRole
  firstName         String
  lastName          String
  email             String              @unique
  createdAt         DateTime            @default(now())
  updatedAt         DateTime            @updatedAt
  userGroups        UserGroupUser[]
  contentPosts      ContentPost[]       @relation("userPostsContent")
  userAdministersContact UserAdministersContact[]
}

// Define the UserGroup model
model UserGroup {
  id       String         @id @default(auto()) @map("_id") @db.ObjectId
  title    String
  createdAt DateTime      @default(now())
  updatedAt DateTime      @updatedAt
```

```
  events    EventUserGroup[]
  users     UserGroupUser[]
}

// Define the Event model
model Event {
  id            String            @id @default(auto()) @map("_id") @db.ObjectId
  title         String
  description   String?
  startTime     DateTime
  endTime       DateTime
  link          String?
  createdAt     DateTime          @default(now())
  updatedAt     DateTime          @updatedAt
  eventCategories EventEventCategory[]
  userGroups    EventUserGroup[]
}

// Define the EventCategory model
model EventCategory {
  id       String             @id @default(auto()) @map("_id") @db.ObjectId
  title    String
  createdAt DateTime          @default(now())
  updatedAt DateTime          @updatedAt
  events    EventEventCategory[]
}

// Define the Location model
model Location {
  id              String                  @id @default(auto()) @map("_id") @db.ObjectId
  title           String
  description     String?
  image           String?
  latitude        Float
  longitude       Float
  deltaLat        Float
  deltaLng        Float
  createdAt       DateTime                @default(now())
  updatedAt       DateTime                @updatedAt
  locationCategories LocationLocationEventCategory[]
  Contact         Contact[]               @relation("contactLocation")
}

// Define the LocationEventCategory model
model LocationEventCategory {
  id       String                  @id @default(auto()) @map("_id") @db.ObjectId
  title    String
  description String?
  icon     String?
  createdAt  DateTime              @default(now())
  updatedAt  DateTime              @updatedAt
  locations  LocationLocationEventCategory[]
}
```

```
// Join tables for many-to-many relationships

model UserAdministersContact {
  id        String  @id @default(auto()) @map("_id") @db.ObjectId
  userId    String  @db.ObjectId
  user      User    @relation(fields: [userId], references: [id])
  contactId String  @db.ObjectId
  Contact   Contact @relation(fields: [contactId], references: [id])

  @@unique([userId, contactId])
}

model UserGroupUser {
  id          String    @id @default(auto()) @map("_id") @db.ObjectId
  userId      String    @db.ObjectId
  user        User      @relation(fields: [userId], references: [id])
  userGroupId String    @db.ObjectId
  userGroup   UserGroup @relation(fields: [userGroupId], references: [id])

  @@unique([userId, userGroupId])
}

model EventUserGroup {
  id          String    @id @default(auto()) @map("_id") @db.ObjectId
  eventId     String    @db.ObjectId
  event       Event     @relation(fields: [eventId], references: [id])
  userGroupId String    @db.ObjectId
  userGroup   UserGroup @relation(fields: [userGroupId], references: [id])

  @@unique([eventId, userGroupId])
}

model EventEventCategory {
  id              String        @id @default(auto()) @map("_id") @db.ObjectId
  eventId         String        @db.ObjectId
  event           Event         @relation(fields: [eventId], references: [id])
  eventCategoryId String        @db.ObjectId
  eventCategory   EventCategory @relation(fields: [eventCategoryId], references: [id])

  @@unique([eventId, eventCategoryId])
}

model LocationLocationEventCategory {
  id                      String                @id @default(auto()) @map("_id") @db.ObjectId
  locationId              String                @db.ObjectId
  location                Location              @relation(fields: [locationId], references: [id])
  locationEventCategoryId String                @db.ObjectId
  locationEventCategory   LocationEventCategory @relation(fields: [locationEventCategoryId], references: [id])

  @@unique([locationId, locationEventCategoryId])
}
```

```
model Contact {
  id                  String              @id @default(auto()) @map("_id") @db.ObjectId
  firstName           String
  lastName            String
  location            Location?           @relation("contactLocation", fields: [locationId], references: [id])
  locationId          String?             @db.ObjectId
  roles               Role[]              @relation("contactRoles")
  officeHours         OfficeHour[]
  organization        Organization?       @relation("mainContact")
  organizations       ContactOrganization[]
  userAdministersContact UserAdministersContact[]
}

model Organization {
  id             String              @id @default(auto()) @map("_id") @db.ObjectId
  description    String?
  mainContact    Contact             @relation("mainContact", fields: [mainContactId], references: [id])
  mainContactId  String              @unique @db.ObjectId
  contentPosts   ContentPost[]       @relation("organizationPosts")
  images         Image[]
  imageCategories ImageCategory[]    @relation("organizationImageCategories")
  contacts       ContactOrganization[]
  roles          Role[]              @relation("organizationRoles")
}

model ContactOrganization {
  id             String      @id @default(auto()) @map("_id") @db.ObjectId
  contactId      String      @db.ObjectId
  contact        Contact     @relation(fields: [contactId], references: [id])
  organizationId String      @db.ObjectId
  organization   Organization @relation(fields: [organizationId], references: [id])

  @@unique([contactId, organizationId])
}

model Role {
  id             String      @id @default(auto()) @map("_id") @db.ObjectId
  title          String
  description    String?
  organization   Organization @relation("organizationRoles", fields: [organizationId], references: [id])
  organizationId String      @db.ObjectId
  contact        Contact?    @relation("contactRoles", fields: [contactId], references: [id])
  contactId      String?     @db.ObjectId
}

model OfficeHour {
  id       String  @id @default(auto()) @map("_id") @db.ObjectId
  dayOfWeek String
  startTime String
  endTime   String
  contact   Contact @relation(fields: [contactId], references: [id])
  contactId String  @db.ObjectId
}
```

```
model ContentPost {
  id             String       @id @default(auto()) @map("_id") @db.ObjectId
  code           String
  images         Image[]      @relation("postImages")
  organization   Organization? @relation("organizationPosts", fields: [organizationId], references: [id])
  organizationId String?      @db.ObjectId
  createdBy      User         @relation("userPostsContent", fields: [userId], references: [id])
  userId         String       @db.ObjectId
}

model Image {
  id             String       @id @default(auto()) @map("_id") @db.ObjectId
  imageUrl       String
  contentPost    ContentPost?  @relation("postImages", fields: [contentPostId], references: [id])
  contentPostId  String?      @db.ObjectId
  organization   Organization? @relation(fields: [organizationId], references: [id])
  organizationId String?      @db.ObjectId
  imageCategory  ImageCategory @relation("imageCategories", fields: [imageCategoryId], references: [id])
  imageCategoryId String       @db.ObjectId
}

model ImageCategory {
  id             String      @id @default(auto()) @map("_id") @db.ObjectId
  title          String
  description    String?
  images         Image[]     @relation("imageCategories")
  organizationId String      @db.ObjectId
  organization   Organization @relation("organizationImageCategories", fields: [organizationId], references: [id])
}

enum UserRole {
  ADMIN
  STUDENT
  GUEST
  FACULTY
  STAFF
  BETA_STUDENT
  BETA_GUEST
  BETA_FACULTY
  BETA_STAFF
  BETA_ADMIN
}
```

Figure 41 "schema.prisma" Data Layer Prisma Schema

The schema defines several models that represent the structure of the data in the application, including User, UserGroup, Event, EventCategory, Location, LocationEventCategory, Contact, Organization, Role, OfficeHour, ContentPost, Image, and ImageCategory. It also defines an enumeration called UserRole to represent different user roles.

There are also several join models for many-to-many relationships, such as UserAdministersContact, UserGroupUser, EventUserGroup, EventEventCategory, and LocationLocationEventCategory. These models are necessary to represent the relationships between the main models, as many-to-many relationships are not directly supported by MongoDB.

The schema is configured to use MongoDB as the datasource with the connection URL being provided through an environment variable. It uses the "prisma-client-js" generator to create a type-safe Prisma Client for working with the data in the application.

Each model is defined with its fields, data types, and any applicable attributes such as @id, @default, @unique, @relation, etc. These attributes provide metadata about the fields and help Prisma generate the appropriate database schema and client code. For example, the User model has fields like id, auiId, auiEncryptedId, userRole, firstName, lastName, email, createdAt, updatedAt, etc. Each field has a specified data type and may have additional attributes.

The @relation attribute is used to define relationships between models. For instance, the User model has a one-to-many relationship with the ContentPost model via the contentPosts field.

In addition, the schema also uses the @map and @db attributes to provide more specific information about how the fields should be mapped to the database. The @map attribute is used to specify the database field name if it differs from the field name in the model, and the @db attribute is used to provide information about the underlying database type.

In conclusion, this schema provides a clear and comprehensive representation of the data layer for the application. It uses Prisma to define models and relationships, making it easier to work with the data in a type-safe and developer-friendly way.

While implementing this application’s data layer, I found MongoDB Compass to be an invaluable tool for working with the database. It is a powerful and intuitive GUI that allows me to easily explore, visualize, and manipulate the data in MongoDB. By using MongoDB Compass, I could efficiently manage and interact with the data, making it easier to understand the structure of the database and ensure its integrity throughout the development process.

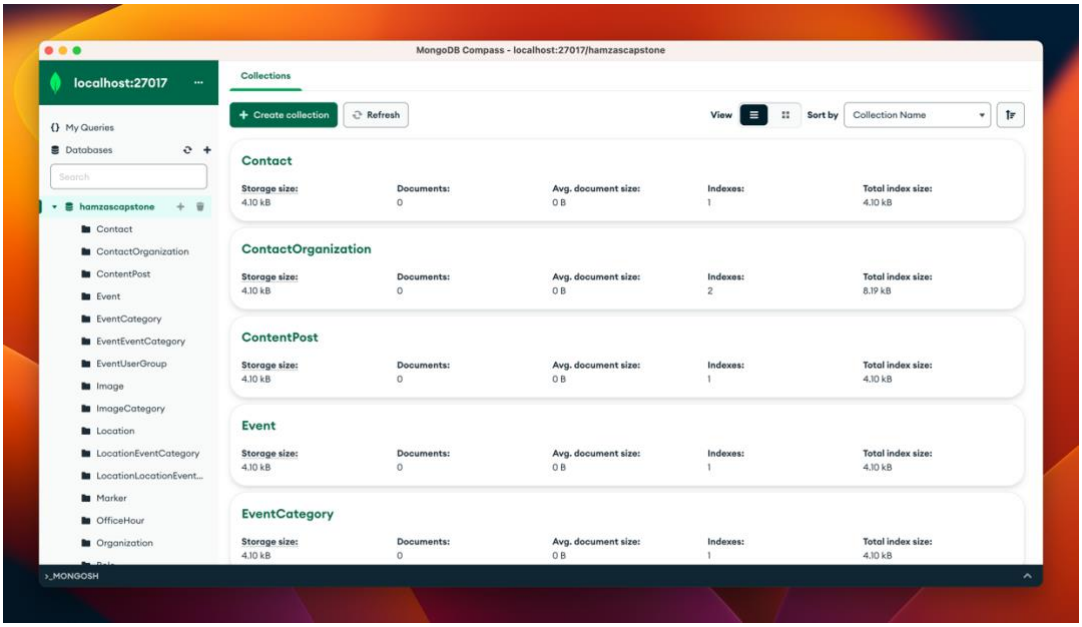


Figure 42 Using MongoDB Compass to Explore the Database

10 CONCLUSIONS

The AUI Campus application successfully consolidates university resources in a single, easy-to-use platform, providing students, faculty, and staff with efficient access to essential campus services and information. The application leverages big data, data mining techniques, artificial intelligence, and neural networks to enhance user experiences and automate various campus processes. The potential for monetization through targeted advertising and partnerships offers economic benefits for the university and further supports the project's feasibility.

Future perspectives for AUI Campus include the continuous improvement and expansion of features and services offered on the platform. This may involve integrating additional campus services, such as health and wellness resources, and incorporating more advanced AI and machine learning techniques to further personalize user experiences. Moreover, exploring partnerships with other universities could enable the expansion of AUI Campus to serve a wider audience, perhaps become X Campus, and further enhance the platform's impact on higher education in general.

11 FUTURE PLANS

To elevate the AUI Campus application to the next level and support its continuous growth, a strategic upgrade plan has been devised. This plan encompasses migrating to an all-Microsoft Azure tech stack and implementing a microservices software architecture to increase scalability, flexibility, and resilience. The plan also includes enhancing user personalization through advanced AI and machine learning techniques, expanding features and services offered on the platform, exploring monetization opportunities, and forging strategic partnerships with other universities. Furthermore, the upgrade plan emphasizes the importance of bolstering platform security, compliance, and monitoring to ensure a seamless and secure experience for all users.

1. Migrate existing infrastructure to Microsoft Azure:
    - Utilize Azure Kubernetes Service (AKS) for container orchestration
    - Employ Azure CosmosDB as opposed to MongoDB for database management
    - Leverage Azure Machine Learning for AI and ML capabilities
    - Implement Azure Cognitive Search for an enhanced search engine experience

2. Transition to a microservices architecture:
    - Break down existing monolithic application into smaller, independent services
    - Implement API Gateway for streamlined communication between microservices
    - Adopt a Domain-Driven Design (DDD) approach for better organization and scalability

3. Enhance user personalization:
    - Integrate Azure Cognitive Services for AI-powered recommendations
    - Utilize Azure Data Factory for data ingestion and processing
    - Leverage Azure Databricks for big data processing and analysis

4. Expand features and services offered on the platform:
    - Integrate health and wellness resources, such as telemedicine and mental health support
    - Collaborate with campus bookstore for textbook purchases and rentals
    - Incorporate event management tools for organizing and promoting campus events

5. Monetization and partnerships:
    - Implement targeted advertising using Azure Advertising Analytics
    - Develop a marketplace for local businesses to offer promotions to the campus community
    - Explore partnerships with other universities to expand the AUI Campus platform

6. Improve platform security and compliance:
    - Utilize Azure Security Center for security management and threat protection
    - Implement Azure Private Link for secure, private communication between services
    - Ensure compliance with relevant data protection regulations

7. Strengthen system monitoring and analytics:
    - Employ Azure Application Insights for application performance monitoring
    - Utilize Azure Log Analytics for centralized logging and diagnostics
    - Leverage Azure Monitor for comprehensive infrastructure monitoring

8. Full cloud migration to Microsoft Azure:
    - Leverage Azure's Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) offerings for seamless cloud migration
    - Take advantage of Azure's scalability, flexibility, and cost-efficiency benefits
    - Utilize Azure Migrate to assess, plan, and execute the migration process

12 REFERENCES

[1] Microsoft, "Microsoft Azure: Cloud Computing Services," [Online]. Available: https://azure.microsoft.com/. [Accessed: 12-03-2023].

[2] React Native, "React Native: A framework for building native apps using React," [Online]. Available: https://reactnative.dev/. [Accessed: 12-03-2023].

[3] Prisma, "Prisma: Next-generation ORM for Node.js and TypeScript," [Online]. Available: https://www.prisma.io/. [Accessed: 12-03-2023].

[4] MongoDB, "MongoDB Compass: The GUI for MongoDB," [Online]. Available: https://www.mongodb.com/products/compass. [Accessed: 12-03-2023].

[5] Azure Cognitive Services, "Microsoft Azure Cognitive Services," [Online]. Available: https://azure.microsoft.com/en-us/services/cognitive-services/. [Accessed: 12-03-2023].

[6] Elastic, "Elasticsearch: Search & Analyze Data in Real Time," [Online]. Available: https://www.elastic.co/elasticsearch/. [Accessed: 12-03-2023].

[7] Express, "Express: Fast, unopinionated, minimalist web framework for Node.js," [Online]. Available: https://expressjs.com/. [Accessed: 12-03-2023].

[8] UI Kitten, "UI Kitten: React Native UI Library," [Online]. Available: https://akveo.github.io/react-native-ui-kitten/. [Accessed: 12-03-2023].

[9] Redux, "Redux: A Predictable State Container for JS Apps," [Online]. Available: https://redux.js.org/. [Accessed: 12-03-2023].

[10] TypeScript, "TypeScript: Typed JavaScript at Any Scale," [Online]. Available: https://www.typescriptlang.org/. [Accessed: 12-03-2023].

[11] Node.js, "Node.js: A JavaScript runtime built on Chrome's V8 JavaScript engine," [Online]. Available: https://nodejs.org/en/. [Accessed: 12-03-2023].

[12] IEEE Standards Association. "IEEE 830-1998: Practice for Software Requirements Specifications." IEEE, 1998. [Online]. Available: https://standards.ieee.org/standard/830-1998.html. [Accessed: 12-03-2023].

[13] IEEE Standards Association. "IEEE 1016-2009: Standard for Information Technology, Systems Design." IEEE, 2009. [Online]. Available: https://standards.ieee.org/standard/1016-2009.html. [Accessed: 12-03-2023].

[14] IEEE Standards Association. "IEEE 1028-2008: Standard for Software Reviews and Audits." IEEE, 2008. [Online]. Available: https://standards.ieee.org/standard/1028-2008.html. [Accessed: 12-03-2023].
[15] International Organization for Standardization (ISO), International Electrotechnical Commission (IEC). "ISO/IEC 25010:2011: Systems and software engineering, SQuaRE, and System and software quality models." ISO, IEC, 2011. [Online]. Available: https://www.iso.org/standard/35733.html. [Accessed: 12-03-2023].
[16] International Organization for Standardization (ISO), International Electrotechnical Commission (IEC). [Online]. Available: "ISO/IEC 27001:2013: Information technology, and Security techniques." ISO, IEC, 2013. https://www.iso.org/standard/54534.html. [Accessed: 12-03-2023].

APPENDIX A

REHIOUI Hamza
CSC
AUI CAMPUS: MOBILE CONSOLIDATION OF UNIVERSITY RESOURCES
HARROUD
Spring 2023

The AUI Campus application is an innovative platform designed to streamline university resources and improve accessibility for students, faculty, and staff. By leveraging cutting-edge technologies such as big data, data mining techniques, artificial intelligence, and neural networks, this application aims to enhance user experiences and automate a wide range of campus processes.

The functional requirements of the AUI Campus application include the following:

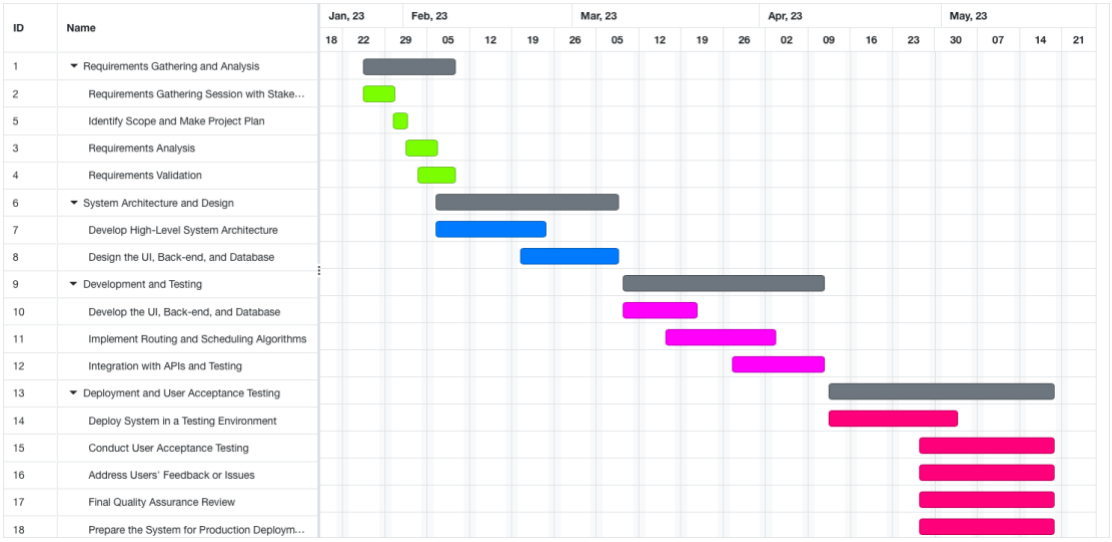
1. The system must provide a comprehensive, easy-to-use interface that consolidates essential campus services and information, such as course registration, academic records, and event calendars.
2. The system should utilize big data and data mining techniques to analyze user behavior and preferences, allowing for personalized content delivery and user-specific recommendations.
3. The system must incorporate artificial intelligence and neural networks to automate various processes, such as chatbot support, academic advising, and resource allocation.
4. The system should be compatible with various devices and platforms, including smartphones, tablets, and computers, ensuring maximum accessibility for users.
5. The system must include a robust user management tool and role-based access control to ensure data privacy and security.

The non-functional requirements of the AUI Campus application can be summarized as follows:

1. Scalability: The system must be designed to accommodate a growing user base, increased data volume, and expanded service offerings.
2. Reliability: The system should offer uninterrupted operation and minimal downtime, ensuring consistent access to essential resources and services.
3. Performance: The application must provide fast response times and real-time updates to ensure a seamless user experience.
4. Confidentiality: The system must employ advanced security measures to protect sensitive user data and maintain privacy.
5. Usability: The application should prioritize user-friendliness and intuitive navigation, facilitating ease of use for all members of the university community.

To further enhance the project's feasibility, the AUI Campus application offers opportunities for monetization through targeted advertising and strategic partnerships. By presenting relevant ads and promotional content to users based on their interests and behavior, the application can generate revenue for the university while also providing valuable offers and discounts to its users.

Below is a detailed tentative action plan for the 4 months of the project's timeline in a Gantt chart format:



In conclusion, the AUI Campus application presents a feasible and valuable solution for consolidating university resources and enhancing the campus experience for students, faculty, and staff. By leveraging advanced technologies and offering monetization opportunities, the application promises to bring both functional and economic benefits to the university.

Approved by the Supervisor(s)

Dr. H. Harroud