Campus Cart

Prosjekt Gruppe 30

av

Haben, Naibe, Habteab Teame Kiros and Teklit

i

IKT205-G 25V

Applikasjonsutvikling

Fakultet for teknologi og realfag

Universitetet i Agder

Grimstad, Mai 2025

Sammendrag

Managing student transactions for second-hand goods can be challenging, as existing marketplace platforms like Facebook Marketplace and Finn.no lack the necessary filters and security measures to cater specifically to students. To address this gap, we developed Campus Cart—a university-based marketplace designed to facilitate the buying, selling, and exchanging of textbooks, electronics, furniture, gym equipment, and other essential student-related items.

Our platform aims to create a secure, efficient, and student-friendly environment by implementing robust spam protection, user verification, and content moderation to prevent fraud and ensure a trusted marketplace. Unlike broader platforms, Campus Cart includes advanced search filters that allow students to find relevant course materials and essential items with ease. Additionally, the app supports both English and Norwegian, ensuring accessibility for both local and international students.

The development process followed the Scrum methodology, allowing for iterative improvements and efficient collaboration. Tools such as Git for version control, Tiaga for task management, Toogle for tracking time and Figma for UI/UX design were utilized to streamline development. Throughout the project, we conducted weekly sprints and retrospectives to refine our approach, ensuring that the application aligns with student needs.

At the conclusion of the project, we delivered a fully functional platform that provides a seamless experience for students to buy, sell, and exchange items within their university community. The app fosters a fair pricing culture, recognizing the financial challenges students face while promoting sustainability through the reuse of goods. By integrating modern security protocols, multilingual support, and a well-structured user interface, Campus Cart stands out as a unique solution in the student marketplace ecosystem.

This project not only enabled us to develop a scalable and user-centric application but also enhanced our expertise in full-stack development, agile project management, and secure marketplace design. The insights gained throughout this process have strengthened our understanding of how to build tailored solutions that address real-world challenges faced by students.

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Nei

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Innhold

Figurer

Tabeller

Listinger

Preface

This project, titled *Campus Cart*, was developed as part of the Ikt205 Applikasjonsutvikling course during our 4th semister of the software development engineering at UIA Campus Grimstad. The objective was to design and implement a second-hand marketplace application tailored specifically for students, leveraging modern web and mobile development technologies. The project followed the Scrum framework to ensure structured and efficient team collaboration.

The development of this project was carried out by Haben, Naibe, Habteab and Teklit at UIA Grimstad Campus and remotely from our respective locations. Throughout the process, we utilized various tools, including Git for version control, Tiaga for task management, and Figma for UI/UX design, to streamline our workflow and ensure effective collaboration.

We would like to express our sincere gratitude to Christian Auby, whose guidance and feedback were invaluable throughout the project. Their support helped us navigate challenges and refine our approach. Additionally, we appreciate the insights provided by our peers and stakeholders, which contributed to making this project a valuable learning experience.

Grimstad, 20. April 2025
Haben, Naibe , Habteab and Teklit

1 Introduction

The following report details the development of *Campus Cart*, a second-hand marketplace application specifically designed for students. Unlike existing platforms such as Facebook Marketplace or Finn.no, our application focuses exclusively on university and campus-based transactions, allowing students to buy, sell, or exchange textbooks, electronics, furniture, and other essential items with ease.

One of the major challenges students face when using general marketplace platforms is the difficulty in filtering relevant listings, especially for academic-related materials such as books for specific courses and semesters. Additionally, existing platforms do not provide a secure environment tailored to student needs, often leading to issues such as spam, fraud, and irrelevant listings. Our platform aims to address these gaps by implementing strict security measures, user verification processes, and content moderation to ensure a safe and reliable marketplace.

To enhance accessibility, *Campus Cart* currently supports English, with plans to add Norwegian support in the future to cater to the diverse student population, including international students studying in Norway. Furthermore, the marketplace will uphold fair pricing principles, recognizing the financial challenges students face and fostering a supportive community where affordability is prioritized.

The development of this application follows agile methodologies, ensuring iterative improvements and user-centered design. Through this project, we aim to create a secure, convenient, and well-structured platform that facilitates smooth transactions while promoting sustainability through the reuse of goods.

1.1 Bakground

E-commerce and online marketplaces have revolutionized the way people buy and sell goods. However, most mainstream platforms are not optimized for student needs. Platforms like Facebook Marketplace and Finn.no cater to a broad audience, making it challenging for students to find items relevant to their academic and daily lives. Additionally, these platforms often lack the necessary security measures to prevent spam, scams, or inappropriate content.

A dedicated student marketplace is essential in addressing these issues. By creating a platform exclusively for students, we can introduce features tailored to academic life, such as:

- **Advanced filtering options** to allow users to search for books based on course and semester.
- **A wide range of product categories**, including household essentials, electronics, gym equipment, and study materials.
- **Strict security and moderation policies**, ensuring that spam, unrelated posts, and prohibited items (such as alcohol, tobacco, and drugs) are not allowed.
- **A fair pricing system**, encouraging reasonable pricing that aligns with student budgets.

By integrating these features, *Campus Cart* will provide a seamless and student-friendly experience that is currently missing from existing marketplace solutions.

1.2 Product Vision

1.2.1 Who is the Target Customer?

- Primary users: University and college students looking for an affordable and convenient way to buy and sell used items.
- International students who need an easy way to find essential goods upon arrival in Norway.
- Students who value sustainability and wish to reduce waste through second-hand trading.

1.2.2 Which Customer Needs Will the Product Address?

- **Cost Efficiency**: Students can access affordable second-hand items, saving money on books, electronics, and household essentials.
- **Convenience**: A dedicated platform eliminates the need to browse through irrelevant listings on broader marketplace apps.
- **Security**: Built-in fraud prevention measures ensure a safer transaction environment.
- **Community & Sustainability**: The platform fosters a culture of sharing and reusing items within university communities, reducing waste and promoting environmental responsibility.

1.2.3 Which Product Attributes Are Critical for Success?

- **User-Friendly Interface**: A seamless and intuitive experience for listing, searching, and purchasing items.
- **Strict Moderation Policies**: Preventing spam, fraud, and the sale of prohibited items.
- **Advanced Search Filters**: Allowing students to find books and materials relevant to their specific courses and semester.

1.3 Similar Products

1.3.1 Comparison to Existing Solutions

Several general marketplace platforms exist, such as:

- **Facebook Marketplace**: While widely used, it lacks dedicated filters for student needs, making it difficult to find relevant academic materials.
- **Finn.no**: Popular in Norway, but does not cater specifically to student transactions and lacks features for university-based trading.
- **Book-specific resale platforms**: Some websites focus on second-hand textbooks, but they do not support other essential student-related items like furniture, electronics, and gym equipment.

1.3.2 Unique Selling Points of *Campus Cart*

Our platform differentiates itself by offering:

- **Campus-Centric Approach**: Listings are specifically for students, ensuring relevant and useful transactions.
- **Enhanced Search Functionality**: Users can filter items based on academic criteria (e.g., course, semester, subject).
- **Expanded Product Categories**: Beyond textbooks, students can buy/sell electronics, household items, gym equipment, and other essentials.
- **High Security and Moderation Standards**: Strict policies against spam, fraud, and prohibited items, ensuring a safe marketplace.
- **Fair Pricing Culture**: Encouraging reasonable pricing that reflects the financial realities of student life.

By addressing the limitations of existing platforms and introducing student-friendly features, *Campus Cart* aims to become the go-to marketplace for university students in Norway.

2 Methodology

This section describes the method used to develop the Campus Cart project. The development was structured around agile principles, collaboration, and teamwork to ensure the project remained aligned with overall goals.

2.1 Scientific Approach

The development of *Campus Cart* follows a structured and iterative approach, integrating software engineering methodologies and agile project management principles. Our goal is to create an efficient and secure student marketplace where university students can buy and sell second-hand items with ease.

Throughout the development, we have followed a systematic process that includes research, planning, design, development, testing, and deployment. By using industry-standard tools and methodologies, we aim to ensure that the application is user-friendly, scalable, and reliable.

2.2 Project Execution

We began the project by conducting thorough research on existing marketplace platforms, such as Facebook Marketplace and Finn.no, to identify their strengths and shortcomings. A major limitation we found was the absence of student specific filtering options, making it difficult for university students to locate listings relevant to their academic and daily needs. Based on these insights, we defined the core features of Campus Cart, ensuring that they cater specifically to the university student community.

In the design phase, we used Figma to create wireframes and interactive prototypes, helping us visualize the application's layout and user workflows[1]. To model the system architecture and interactions, we employed Enterprise Architect, producing use case diagrams, class diagrams, and sequence diagrams. These artifacts provided a clear blueprint for structuring the application's logic and behavior.

For the backend, we utilized Firebase, which offered real-time database functionality, user authentication, and cloud storage[2]. On the frontend, we built a cross-platform mobile application using React Native and Expo, allowing seamless deployment on both Android and iOS devices.

We followed the Scrum methodology for agile development. Weekly sprint meetings were organized using Taiga, where we handled task assignments, sprint planning, and progress tracking. However, one limitation we encountered was the inability to effectively track working hours within Taiga. To address this, we integrated Toggle into our workflow, linking it with our sprint tasks to monitor time spent on development activities more accurately.

Each sprint involved defining goals, distributing tasks among team members, and conducting reviews to assess progress. This iterative approach ensured steady delivery of features and allowed us to adapt flexibly based on continuous feedback.

2.3 Development Process

2.3.1 Idea Phase and Initial Planning

The project started with an idea phase where we brainstormed and discussed different possible marketplace solutions. After evaluating several ideas, we agreed on developing Campus Cart as there was no dedicated platform tailored specifically to students' needs.

During this phase, we held discussions on project scope, user requirements, and technical feasibility. We also assigned roles among team members and established a development timeline.

2.3.2 Planning and System Design

Once the project idea was finalized, we moved on to the planning phase. We defined the application's functional and technical requirements, identified potential challenges, and chose the most efficient development approach.

To ensure a clear structure and organization, we developed the following system models:

- **Use Case Diagrams** to outline key user interactions.
- **Class Diagrams** to show the main entities and their relationships.
- **Sequence Diagrams** to map out the flow of messages between objects.
- **Database Schema** to define our data tables and relationships.
- **Wireframes and UI Prototypes** to design and validate the user interface.

Additionally, we established a sprint backlog in Taiga to break down tasks and track progress throughout development.

2.3.3 Core Development

With the planning phase complete, we began developing the core functionalities of the application. We followed an agile approach, implementing essential features in increments and continuously refining them based on feedback.

We prioritized the development tasks using the MoSCoW method, ensuring that the "must-havefeatures were built first before moving on to should-have"and "could-haveé-nhancements. Git was used for version control, enabling seamless collaboration among team members.

2.3.4 Testing and Bug Fixing

As the development progressed, we conducted extensive testing to ensure the stability and reliability of Campus Cart. Testing was performed in multiple stages:

- **Unit Testing**: Individual components were tested to ensure they function correctly.
- **Integration Testing**: We verified that different parts of the application worked together seamlessly.
- **User Acceptance Testing (UAT)**: A selected group of students tested the platform and provided feedback.

Bugs and performance issues were tracked using Taiga, allowing us to categorize and prioritize issues effectively. Our focus was to address critical functionality issues before moving on to visual or minor usability enhancements.

2.3.5 Report Writing and Finalization

The documentation process was ongoing throughout the project. However, the final two weeks were dedicated to compiling all findings, documenting the challenges we faced, and summarizing the outcomes of the project.

2.4 Agile Methodology and Collaboration

The agile Scrum methodology was adopted throughout the development of the Campus Cart application to ensure continuous progress, clear communication, and flexibility. Scrum helped divide the work into small, manageable parts and deliver features gradually through weekly sprints.

2.4.1 Scrum Framework and Sprint Planning

The team followed the Scrum framework to organise the project timeline and coordinate tasks. Each sprint was accompanied by a planning session where the team discussed what needed to be done and how to approach it. Given scheduling constraints, the team daily stays with weekly sprint meetings, typically held at the start of each week.

During these meetings, the group reviewed progress from the provided sprints, discussed any blockers or difficulties, and adjusted our plans as needed. This approach kept the team aligned and responsive to change and feedback. Each sprint concluded with a short review, helping the group reflect on what went well and what needed improvement.

2.4.2 Roles and Responsibilities

Roles were assigned based on each team's performance and strengths to support a smooth collaboration. While everyone contributed to development, design, and documentation, clear responsibilities were defined to keep things organised and efficient. One member of the group took on the role of Scrum Master, responsible for leading sprint meetings, managing task assignments, and resolving any blockers. This role also serves as a link between task coordination and planning, making sure the team remains focused and progresses with the goals of the project.

2.4.3 Task Division and Workload Management

The project workload was distributed among team members, ensuring equal participation in different phases of development. Each team member was required to contribute at least 12 hours per week, and time was tracked via Toggle time sheets.

This method helped maintain transparency and accountability, making it easier to monitor individual contributions and identify potential delays early. When certain tasks took longer than expected, the team adjusted workloads to stay on track. Clear expectations and time tracking helped maintain a balance and efficient workflow.

2.4.4 Version Control and File Management

To maintain an organized and efficient workflow:

- **Git** was used for version control and code collaboration.
- A **shared Google Drive** was set up for storing reports, diagrams, and design files.

The development of *Campus Cart* has been a structured and iterative process, ensuring that the application is well-planned, technically sound, and aligned with the needs of university students. By integrating Agile methodologies, modern development tools, and a strong focus on usability, we have created a marketplace platform that simplifies student transactions while maintaining high security and efficiency.

Our approach to project execution starting with research, moving through design, implementing core functionalities, testing, and iterating allowed us to deliver a high-quality product within the given time frame. The lessons learned from this project, including team collaboration, agile development, and problem solving, have significantly enhanced our technical and project management skills.

2.5 Use of Artificial Intelligence Tools

During the writing of this project for Campus Cart, we used digital tools such as ChatGPT 4o [3] and Grammarly exclusively for language improvement [4]. These tools assisted in correcting grammar, improving sentence structure, and suggesting synonyms to reduce repetition. They were not used to generate content, chapters, or paragraphs. All academic writing and structure were produced entirely by the team.

3 Requirements

Requirements are the foundation of the Campus Cart development process. They are categorized into two categories: Functional requirements and nonfunctional requirements. Clear requirements ensure that the final product meets user expectations and project goals.

To deploy the project, the MoSCoW method is used to organise the requirements. This method helps categorise requirements based on their importance to the final product.

Figur 1: MoSCow method used to prioritize project requirements

As shown in the above figure 1, the MoSCoW method divides requirements into four categories[5]:

- Must Have: critical for release
- Should Have: Importnat, but not critical at launch
- Could Have: If time permits, it is nice to have
- Won't Have: not included in the project

3.1 Functional Requirements

Functional requirements describe the essential operations, functions, and characteristics that an application provides to a user. They define the project's expected functionality.

3.1.1 Must-have

1. A new user must be able to register with an email and password to create an account and gain access to the application.
2. Registered users must be able to log in to access the app's core functionality, including browsing and listing items.
3. Users must be able to recover their password through email in case they forget it, enabling continued access to their account.
4. Logged-in users must be able to create new item listings to sell the goods.
5. Including a title, description, category, price and condition to ensure complete and clear item information for buyers.
6. Users must be able to edit or delete the content they uploaded.
7. The app must have a search bar and filter options to help users find items by course, semester, category, and price, supporting the search and filter items functionality.
8. The system must have content moderation features to prevent spam and inappropriate items, supporting the platform's security goals.
9. Administrators must be able to view user profiles and take actions like viewing and deleting accounts.
10. administrators must be able to view and remove inappropriate listings from the database.

3.1.2 Should-have

1. The user should be able to upload and update images for their listings.
2. Each user should have access to a profile screen showing their info and listings.
3. The item listings should be displayed in a grid layout for a better browsing experience.
4. The app should offer basic listing statistics, such as how many times an item has been viewed.
5. The app should include a clear reporting mechanism for users to flag suspicious or inappropriate content.

3.1.3 Could-have

1. The app could include a chatbot to help users with FAQs or navigation.
2. Users could be able to share item listings to social media platforms like Instagram and Facebook.
3. The app could use AI-powered item recommendations based on user behavior.
4. The system could support rating and reviews for users after a transaction.

3.1.4 Will-not

1. The app will not include payment functionality (e.g., PayPal or card integration).
2. Guest users will not be allowed to access the app -all users must register for an account.
3. The app will not support integration with external marketplaces like finn.no or Facebook Marketplace.
4. In the current release, the app will not offer premium features, such as paid promotions of featured listings.

3.2 Non-functional Requirements

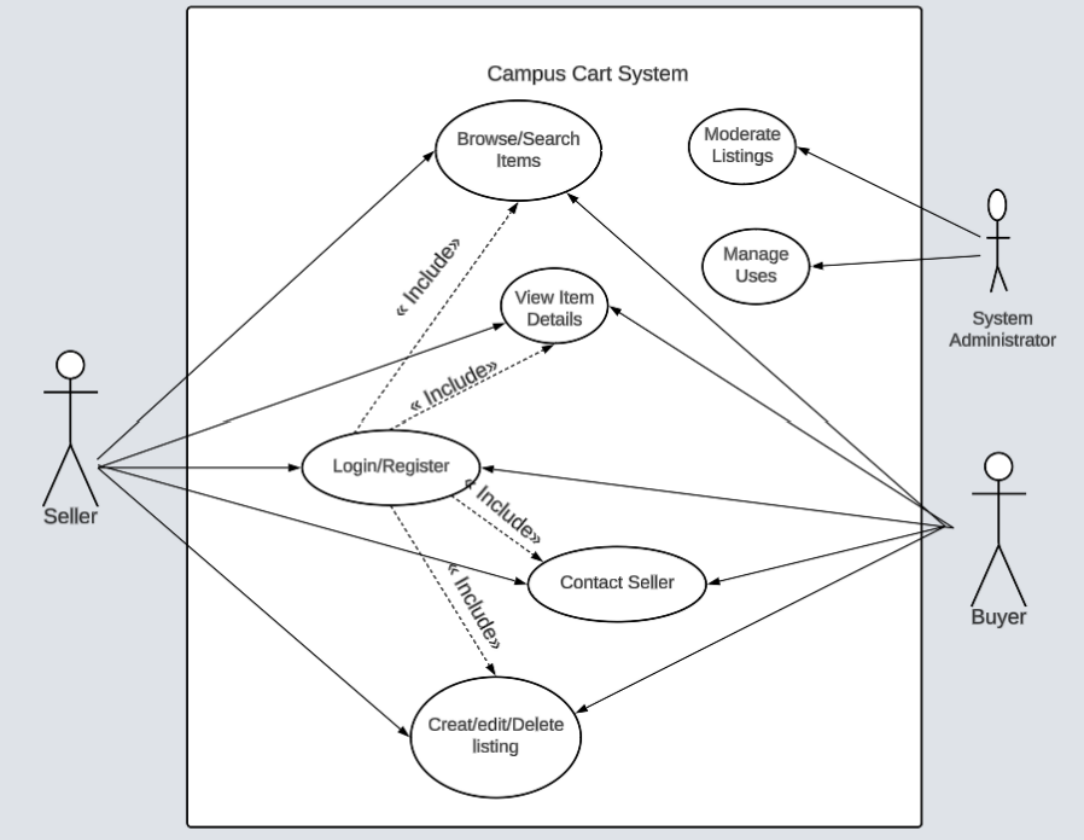
Non-functional requirements define the quality attributes, system constraints, and performance expectations that ensure the system operates effectively[6]. They focus on how the system should perform rather than what it should do.

1. The user interface must be intuitive and easy to navigate.
2. The Campus Cart must load quickly and handle multiple users simultaneously.
3. The system must be reliable and accessible 24/7.
4. The design should follow modern UI/UX principles, using tools like Figma to create early sample layouts.
5. All user information must be stored safely and securely, with measures in place to prevent unauthorized access.
6. The development process must follow agile practices, specifically the Scrum framework.

3.3 Use Case

The Campus Cart platform is designed to facilitate the buying and selling of used goods between individuals in a university environment. The system ensures secure, accessible, and user-friendly interactions tailored to the needs of students and administrators.

We have three main actors in the system: Seller, Buyer, and System Administrator. It is worth noting that Seller and Buyer represent the same user group – students – since any user can both buy and sell goods. Therefore, their functional access in the system is identical.



Figur 2: Use case diagram for the Campus Cart System with three main Actors

As shown in Figure 2, student users must first authenticate themselves through the Login/Register use case before they can access any functionality 1, 2. Once authenticated, they can browse or search for goods, view item details, and contact other users through the built-in messaging feature 7. In addition, users can create, edit, or delete their own listings 4, 6, with each listing containing necessary details such as description, category prices and condition 5. These core interactions are connected through "include" relationships to ensure consistency and security across the system, as defined in the functional requirements.

The System Administrator has a different role, with extended privileges that allow them to moderate posts and manage users, as described in requirements 9, 10. These tasks are important for ensuring security, preventing abuse, and upholding the policies outlined in the non-functional and administrative requirements. Administrator access is also structured around specific use cases, such as Moderate Posts and Manage Users, that are not available to regular users.

This use case model shows a modular and role-based design, where each use case clearly supports the system requirements and keeps clarity between user responsibilities. Each function is developed to meet specific needs defined in the requirements, making sure the system behaves as expected. These connections between the use case and requirements are consistently reflected throughout the report.

This requirement and the use of a case structure helped shape the design boxes in the following chapters.

4 Solution

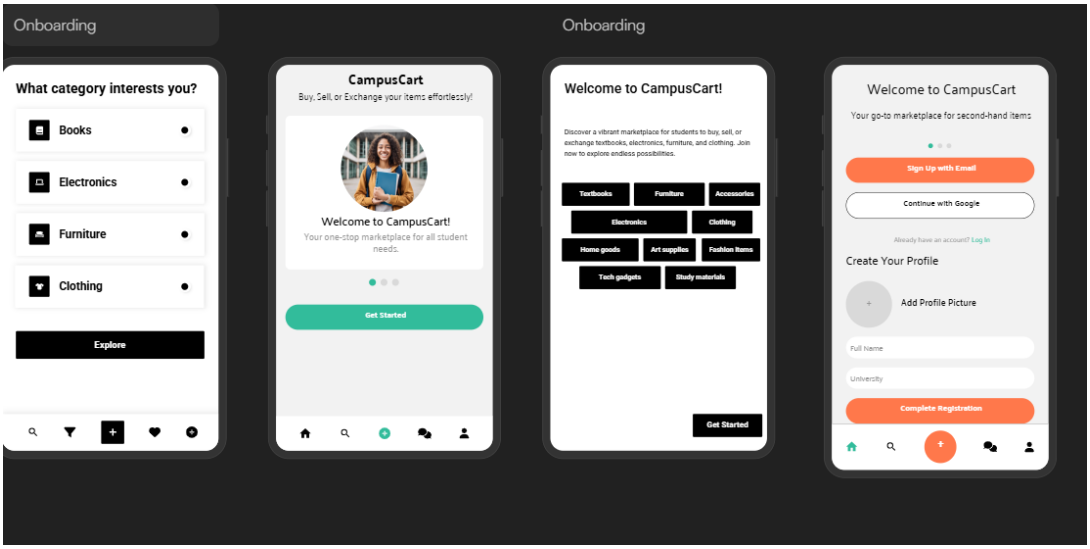
Campus Cart is a mobile app for students who want to buy and sell second-hand items easily and safely. It provides a student-friendly marketplace where users can post their items, browse what others have listed, and chat directly within the app. Everything is designed to be fast, easy to use, and meet the everyday needs of university students.

To make the app work across Android and iOS, we used React Native with Expo. For the backend, we chose Firebase, which handles users, real-time database, and image storage. We also added ComeChat to support direct messaging between users and the Google Cloud Vision to detect and block inappropriate images automatically. From the start, our solution was based on the requirements we defined in section 3. Important features like login 1, posting and managing listings (Requirements 4 5 and 6), messaging 8, and admin moderation 10 were planned early and prioritised using the MoSCoW method. These were then built and tested in sprints using agile development.

The application layout and features were kept clean and minimal to make it easy for any user. Throughout the development, we tested regularly, gathered feedback, and kept improving it to ensure a smooth experience for student users

4.1 Design and planning

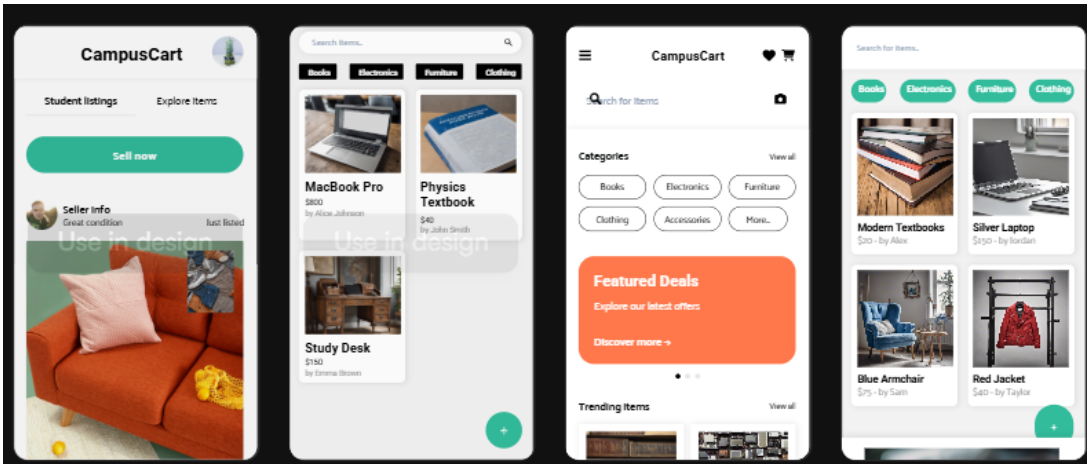
Before the development phase began, the main parts of the Campus Cart application were designed in Figma to create a clear visual overview of the expected layout and functionality. These early sketches included all core screens, such as the home page, login and registration, item listing, and chat. The purpose of this makeup was to define how users would navigate through the application and ensure that all functional requirements were addressed from the start.



Figur 3: Initial wireframe design - onboarding

The design focused on being simple, intuitive, and clean, making it easy for students to interacts with the application without instruction. Light backgrounds and soft colors were chosen to create a calm and welcoming look. Imortant buttons and input fields were placed prominenent to support natural user navigation.

Colour played a key role in the interface. Green was selected as the primary colour to convey safety and trust, which aligns well with the values of a student marketplace. This



Figur 4: Initial wireframe design - Screens

supported the non-functional requirements in terms of usability and visual comfort 3. Each wireframe was designed to directly address the functional requirements from section 3. For example, the create listing, 4, search and browse, 7, and the login and registration screen meet 1 AND 2.

The design phase provided a clear plan for the development team. It helped guide the frontend structure and ensured that each feature was visually and functionally consistent with the project's goals.

4.2 Core features

This section presents the key features that form Campus Cart's core functionality. Each feature is directly connected to the functional requirements defined in section 5 and supported by user interface elements developed during the implementation phase.

4.2.1 Login and Registretion

Campus Cart provides a secure login and registration system to ensure that only verified users can access the platform. Users can create an account, email, and password, or sign in with a Googlee account. Once registered, users are granted access to all core functionality of the application , for example browsing items, posting listings, and messaging other users. These features meet registration 1 and login with email and password 2.

These two requisite processes were implemented using Firebase Authentication, which offers secure and scalable identity handling across mobile platforms. This setup also helps to manage the session and ensure the maintainability of a safe user environment.

4.2.2 Posting and Managing Listings

Registered users can post an item by filling out a form that includes the item's title, description, price, category, and image. Listings are stored in the Firebase database and directly displayed on the home screen.

User can also edit or delete their listings at any time they want. These options are only available for the original author of the listing, and all changes are reflected in real time. This feature supported the requirements of creating listing 4, editing listing 5 and deleting listing 6.

Posting items was designed to be quick and easy, particularly on mobile devices, to support the goal of the application of providing a fast and userfriendly experience.

4.2.3 Searching and Filering Items

The Campus Cart Application has a built-in search bar and filter system to improve visibility and user experience. Users can search for specific items using keywords and filter the results by category, course, or semester.

Filtering allows users to see only the most relevant entries, making the process of finding the correct item more efficient. The filters were implemented using Firestore query logic to return dynamically matching results from the database. This feature addresses requirement 7, which is search and filter entries.

4.2.4 Messaging System

A messaging feature is integrated into Campus Cart to enable direct communication between buyers and sellers or vice versa. Users who are interested in an item can open a chat window to ask questions, negotiate, or arrange a meeting. All messages are delivered in real time using **CometChat**, ensuring fast and reliable communication. The messaging interface is very familiar, with a chat window linked to specific item listings and supporting text based conversations between the users. This feature fulfils requirement 8, that is, messaging between users.

This system helps maintain communication within the platform and improves trust and convenience during transactions.

4.2.5 Admin moderation

To ensure a safe environment, Campus Cart includes an admin moderation feature. Admins can review and manage posts, delete inappropriate content, and suspend users if necessary. Moderation is supported by an automated process that uses Google Cloud Vision to detect inappropriate images during uploading. If a violation is detected, the post is flagged or deleted before publication. This feature meets the requirement 10 Administrator Post Moderation.

Admin access is limited to authorised roles and is not visible to regular users. This structure supports content security and builds user trust in the platform.

4.3 System Architecture and Technology Stack

Campus Cart is built using a modular architecture, with the frontend interface, backend services, and external tools each handled in separate layers. This design ensures that the application remains scalable, easy to maintain, and suitable for cross-platform deployment. The system supports all the functionality explained in the requirements, such as user authentication, item listing, messaging, and content moderation.

4.3.1 System Overview

The application consists of three main components:

- **Frontend**: The frontend of the applications is built with React Native with Expo, the mobile interface allows users to browse listings, send messeges , post items, and

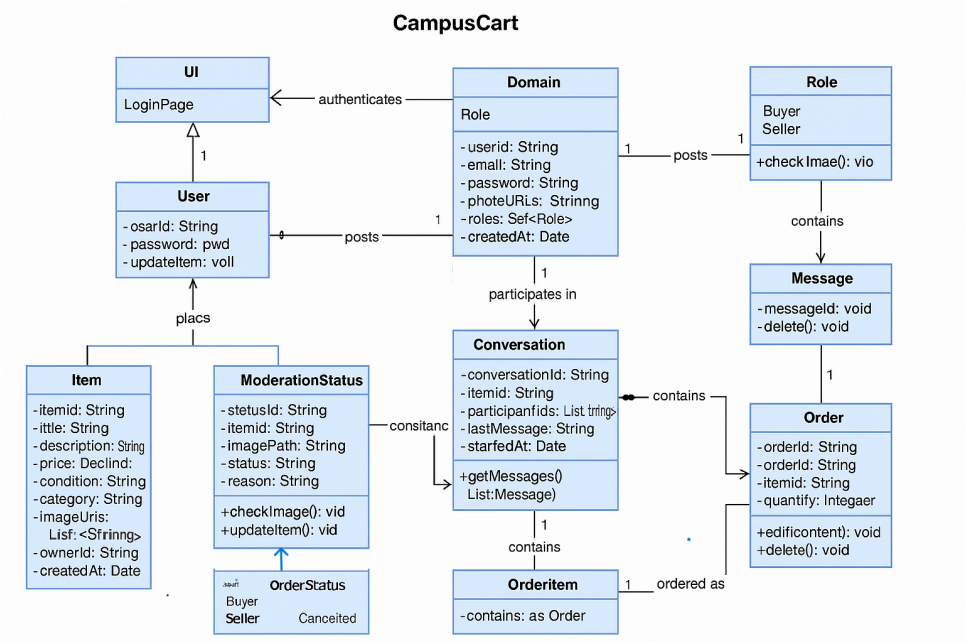
manage their profiles. Expo was chosen for its rapid development tools and support for both Android and iOS platforms.

- **Backend and Database**: The backend is powered by Firebase, which offers key services like user authentication, real time database, cloud storage, and serverless functionality. Firebase authentication handles login and session control, while Firestore stores all listings, user data, and messages in real-time.
- External Services also includes:
  - CometChat is used to provide in-app messaging between users (buyers and sellers). This integration helped with real time communication and user-to-user chats.
  - Google Cloud Vision is used during listing creation to analyse uploaded images for content violations. If an image is identified as inappropriate, the post is blocked before it is published.

This system is designed to ensure that Campus Cart stays dependable and fast across devices while effectively enforcing content moderation and protecting users.

4.3.2 Class diagram

The core data relationships in Campus Cart were represented using a class diagram, which helped shape the database model and supported the development of backend logic. The model includes entities such as users, items, messages, conversations and moderation status.



Figur 5: Class Diagram

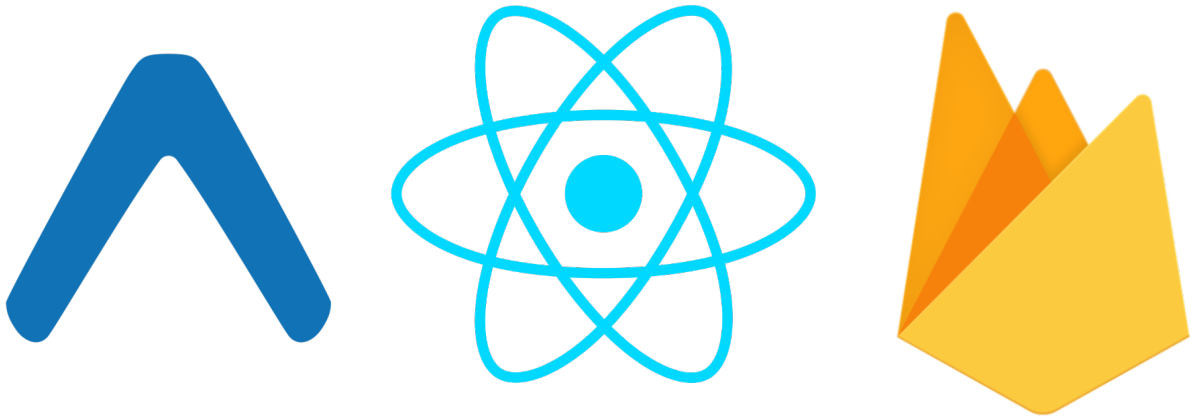
Figure 5 shows the main classes and how they connect to each other through relationships. These connections reflect the Campus Cart's main functionality, which is described in section 3

Key entities in the campus cart date model include the user, which stores information about each student using the applications. Each user can have multiple listings and par-

ticipate in conversations. The item entity holds details such as title, description, image, category, price, and status, and is linked to the user who posted it. Converseation connects users around specific items and enables messaging related to those listings, while each Message represents an individual communication, including sender ID, timestamp, and chat content. Finally, ModerationStatuses stores the image analysis results and helps block inappropriate listings when necessary. This model supports the Campus Cart application's main functionality and ensures that data management stays structured, scalable and secure.

4.3.3 Technology Stack

The tools and technologies used in Campus Cart were selected for their ease of integration, performance , and support for mobile applications. Together, they create a strong base for all application features.



Figur 6: Technology Stack

The frontend of the Campus Cart is built with React Native, using Expo to make development smoother. This setup lets the applications run on both Android and iOS devices using a single codebase, saving time during development and testing. On the backend services, Firebase manages user login, stores data in realtime, and hosts media files . This combination ensures secure login, fast listing updates and reliable data handling.

To support user communication, Cometchat was integrated to provide real-time chat functionality among users (seller and buyer). For constant safety, Google Cloud Vision automatically scans uploaded images and blocks any inappropriate content. These services add important flexibility while reducing the need to build complex features from scratch.

table 1 summarized the key components of the technology stack:

Component	**Technology**	**Purpose**
Mobile Frontend	React Native and Expo	CrossPlatform mobile interface
Backend Services	Firebase	Authentication, real-time database, storage
Messaging	CometChat	In-app bteewn users
Moderation	Google Cloud Vision	Detecting and blocking inappropriate images automatically
UI Design	Figma	Wireframe and interface design
Project Management	Taiga and Toggle	Sprint planning and time tracking
Version Control	Gitand GitHub	Code collaboration and version control

Tabell 1: technology stack which is used in the Campus Cart project

Together, these tools created a stable and efficient system that met all the requirements we had in the section 3. The combinations enabled smooth development, clean UI design, and real time user interaction, essential features for a student-focused marketplace.

4.4 Iterative Development and Feedback

Campus Cart was developed using an iterative process, where features were gradually implemented and improved across multiple sprints. Early designs in Figma included key screens such as login, listing, home, and chat. These mockups guided the initial implementation.

As the app took shape, minor adjustments were made based on internal testing and team discussions. For example, the layout of the home screen was improved to display listings more clearly on smaller screens, making filter options easier to access. In the chat feature, the message layout was simplified for better readability.

Feedback also led to visual design updates, such as improving colour contrast and spacing to improve usability. For content moderation, threshold settings in Google Cloud Vision were adjusted to reduce false positives, and admin override options were added.

Changes were prioritized in sprints reviews using Taiga, and several minor fixes were handled directly during development. This incremental approach allowed Campus Cart to continuously improve without the need for major redesign, while remaining in line with the original goals and requirements.

4.5 Summary of the Solution

Campus Cart was developed as a mobile marketplace platform tailored to the needs of university students. This solution includes key functions such as user registration, item listing, search and filtering, messaging, and admin moderation. The system architecture was designed to be modular and scalable, using modern tools like React Native, Firebase, and ComChat.

Each part of the application was planned and implemented based on the functional and non-functional requirements defined earlier in section 3. The result is a stable, user friendly, and secure mobile application that supports second hand trading in a student community.

5 Thechnical background

This section describes the technologies and tools used to design and develop the Campus Cart application. Each technology was chosen to support efficient development, cross-platform compatibility, secure data handling, and a responsive user experience. The stack includes frameworks for front-end development, backend services, real-time messaging, project management, and integrated development environments.

5.1 Technology stack and Development Tools

5.1.1 React Native

React is a JavaScript library that uses JavaScript XML (JSX) in functions to display HTML on a web page. React organizes JavaScript and HTML into components, which can also include other components. This means a React app only shows the parts the user needs, with features built into each element[7]. The main component receives the video list, goes through each video, and sends it to a separate 'video' component. Each video is then shown using its own "video"component.

5.1.2 TypeScript

TypeScript is an extension of JavaScript that is converted back into JavaScript, providing additional features including interfaces, typing and enums [8]. One of the reasons to use TypeScript is to avoid developers from making type errors during the compilation phase, which enables the reduction of issues during execution. By adding types, the code becomes easier for programmers to understand, benefits from better code suggestions and type predictions in modern development tools (IDE), supports type inference and IntelliSense, and is more maintainable and scalable [9].

5.1.3 Expo

Expo is an open-source platform built around React Native, making it easier to develop and deploy cross-platform applications [10]. It comes with a set of tools, libraries and services that streamline the development workflow, especially for small teams. One of its key advantages is that it enables developers to preview changes immediately on various devices without needing to build the app for each platform manually [10].

5.1.4 Firebase

Firebase is a Google-owned platform that provides a suite of back-end services for mobile and web apps[11]. In Campus Cart, we rely on Firebase to handle two critical needs: authenticating users and storing application data. By using a single, integrated platform, we cut down on infrastructure work and benefit from built-in security, real-time updates, and effortless scaling.

Firebase Authentication Firebase Authentication manages user sign-up, sign-in, and password recovery with just a few lines of configuration. In Campus Cart, students log in with their university email, receive secure password-reset links by email, and stay signed in across sessions. This service plugs directly into our React Native front end and works hand-in-hand with Firestore security rules, so we never have to write or maintain our own credential system.

Firebase Database Firebase Database is Firebase's NoSQL document database that syncs data in real time across connected devices[11]. We use it to store user profiles, item listings, and moderation flags. When a student posts or edits an item, changes appear instantly for everyone else no manual refresh needed. Firestore's offline support, automatic scaling, and tight integration with Firebase Authentication give us a fast, reliable data layer without extra operational overhead.

5.1.5 Google Cloud Vision API

The Google Cloud Vision API is a machine learning service provided by Google that enables applications to understand the content of images. It offers powerful features such as object detection, text recognition, and content moderation through SafeSearch analysis[12]. In the Campus Cart project, this API was used to automatically detect and filter inappropriate content from user-uploaded images. By integrating it with Firebase Cloud Functions, we ensured that any image flagged for adult, violent, or offensive content is automatically deleted from storage. This allowed us to enforce content policies effectively, maintain a safe user environment, and reduce the need for manual moderation.

5.1.6 Real-Time Messaging

Campus Cart needs fast, reliable chat that ties into our existing Firebase login. After weighing options, we chose the CometChat SDK because it plugs neatly into our Firebase authentication, scales easily for storing and retrieving messages, and gives us built-in features like presence, typing indicators, and delivery receipts. By using CometChat's core SDK, we got a proven messaging backbone without spending weeks building and testing real-time logic ourselves.

5.1.7 CometChat

CometChat was used to implement real-time messaging in the Campus Cart application. It provided features such as one-on-one chat, typing indicators, message history, and user presence[13]. The CometChat SDK was integrated alongside Firebase Authentication to ensure secure and seamless communication between users. While some parts of the UI were custom-built due to integration limitations with the Expo framework, the SDK handled the core messaging functionality reliably[14].

5.1.8 Visual Studio Code (VS Code)

VS Code is our primary Integrated Development Environment (IDE). It offers features such as IntelliSense, debugging, and Git integration, which streamline the development process and enhance productivity[14].

5.1.9 Taiga

Taiga is a flexible project management tool used to manage and monitor the progress of the Campus Cart development process. It works with both Scrum and Kanban methodologies [15]. In this project, Taiga was mainly used to organized tasks each week, give team members their duties, and show the status of work using its task board. It also includes helpful tools such as sprint planning, backlog management, and tracking problems or issues [15].

The team used Taiga to maintain a clear view of the most essential tasks and completed tasks. Each user story was documented and followed through each sprint, enabling trans-

parent collaboration and structured feedback during sprint reviews. Even though Tiaga did not have its own time tracking feature, it is connected to the toggle platform to address this issue. This integration enabled the tracking of time spent on every task, helping to improve accountability and balance within the team's work [15].

5.1.10 Hooks

Hooks are functions introduced in React 16.8 that allow developers to use state, lifecycle methods, and other React features inside functional components, without needing to write class components[14]. They make code more concise, reusable, and easier to manage by organizing logic based on behavior rather than lifecycle methods. Common Hooks include 'useState' for managing local state and 'useEffect' for handling side effects like data fetching. Hooks simplify the development process, reduce boilerplate, and promote cleaner component architecture while remaining fully backward compatible.[16]

5.2 Privacy, Security and Legal Compliance

Protecting user data and following rules are key in a Campus Cart project because the system can manage sensitive personal data such as user accounts, storage, and resource access. Strong security measures keep personal data secure by blocking unauthorized access, preventing the misuse of personal data, and preventing data leaks. The General Data Protection Regulation (GDPR) offers a clear guideline for protecting, storing, and managing user data, ensuring it is secure and accessible only to the right person [17].

GDPR Principles

- Data minimization: The GDPR requires that the system collect and store only important user data, reducing unnecessary risk exposure [17].
- Encryption and security: protecting stored data is a key requirement under GDPR. Encryption keeps user data safe, when it's stored [17].
- Access Control (RBAC): The regulation requires that users can only access the data needed for their role, reducing the risk of unauthorized use or modification [17].

5.3 Privacy Policy

To ensure compliance with data protection standards and Google Play policies, a formal privacy policy has been created for *Campus Cart*. The policy outlines how user data is collected, stored, and used, particularly with regard to camera access and image uploads. The privacy policy is publicly available at:

`https://www.freeprivacypolicy.com/live/0a03d1c3-60ed-4049-8d8f-455016d94315`

Compliance measures

GDPR requires organizations to log authentication attempts, resource usage, and date access through audit logs, which helps detect potential breaches and ensure effective security monitoring [17]. To remain compliant, regular security updates should be applied to authentication and storage systems to identify vulnerabilities and keep up with changing security standards [17]. It also secures individuals ' right to view, update, or delete their personal data, ensuring transparency and giving users greater control over their information [17]. These measures ensure data privacy, security, and compliance while protecting sensitive information.

6 Implementation

This chapter describes how Campus-cart implemented based on the planned solution. Each section corresponds to a key functionality of the application, including finished and partially implemented features. Screenshots and code snippets illustrate the technical implementation.

6.1 User Registration and Authentication

Authentication ensures that only verified users can access core features such as adding updating and removing items and contacting sellers.

6.1.1 Registration Page

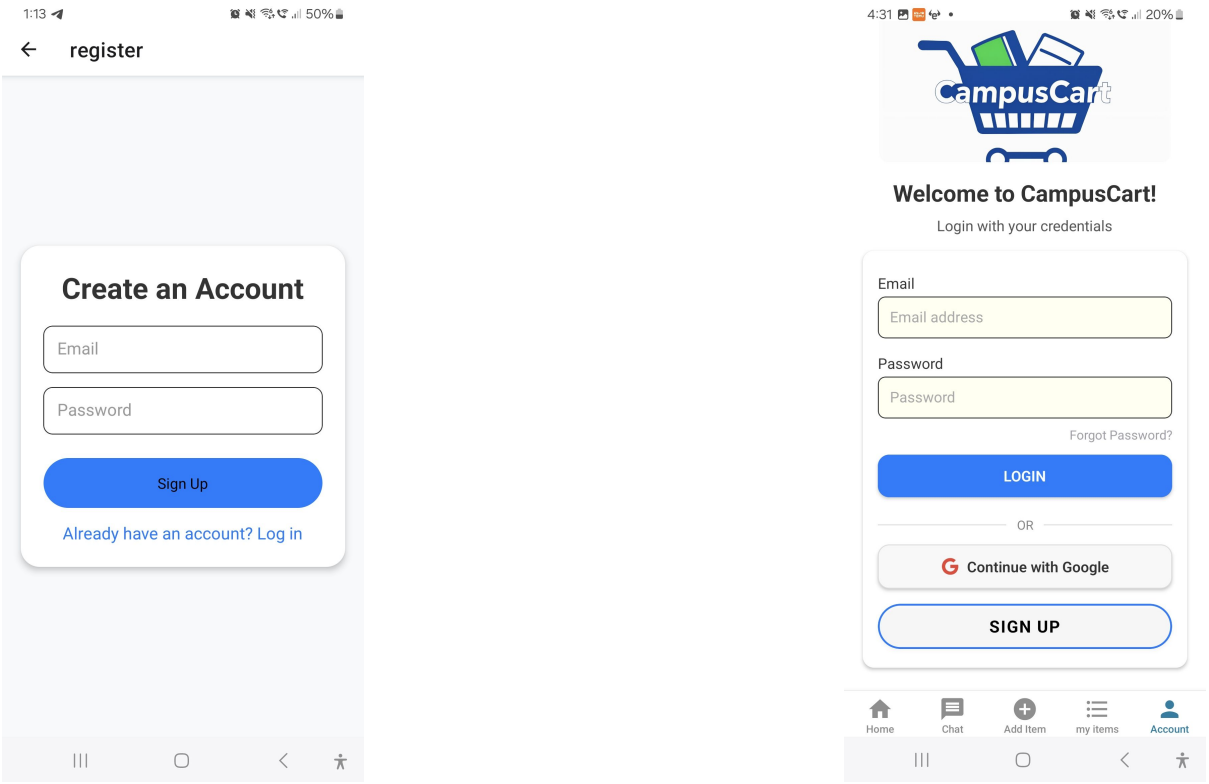
The registration page allows new users to sign up using their email and password. Input fields are validated for correct formatting and completeness before submission.

This feature uses Firebase Authentication, specifically the `createUserWithEmailAndPassword()` method, to register users securely.

```
1 await createUserWithEmailAndPassword(auth, email, password);
```

Listing 1: Firebase Registration

After successful registration, users are redirected to the home screen. The Firebase backend securely stores the user credentials and manages authentication sessions.



(a) Registration UI **(b)** Login UI

Figur 7: Authentication Interfaces for Registration and Login

6.1.2 Email/Password Login

Registered users can securely log in using their email and password credentials.

The method below verifies user credentials against Firebase and signs in the user if authentication succeeds.

```
await signInWithEmailAndPassword(auth, email, password);
```

Listing 2: Firebase Email/Password Login

6.1.3 Google Sign-In (OAuth)

Users can also log in with their Google account, making it quicker and easier to get started. The method below initializes a Google authentication provider and triggers a popup where the user selects their Google account. Once authenticated, Firebase handles the session.

```
const provider = new GoogleAuthProvider();
await signInWithPopup(auth, provider);
```

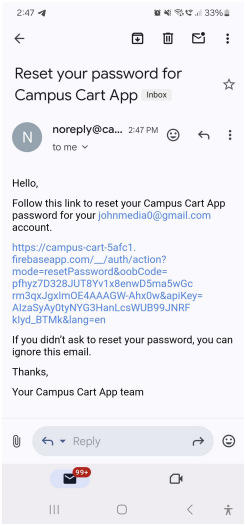
Listing 3: Google Sign-In with Firebase

6.1.4 Forgot Password

A password reset feature allows users to recover their account securely through email-based verification. The function below sends a reset link to the user's email. When the user clicks the link, they are directed to a secure Firebase-hosted page to create a new password.

```
await sendPasswordResetEmail(auth, email);
```

Listing 4: Firebase Password Reset



Figur 8: Password Reset Email

6.2 Database and Storage

6.2.1 Firestore Database for Item Management

Campus Cart uses Firestore to store all item listings submitted by users. Each item is saved as a document in the `items` collection and includes dynamic values such as the

item's name, description, price, condition, category, and image URLs.

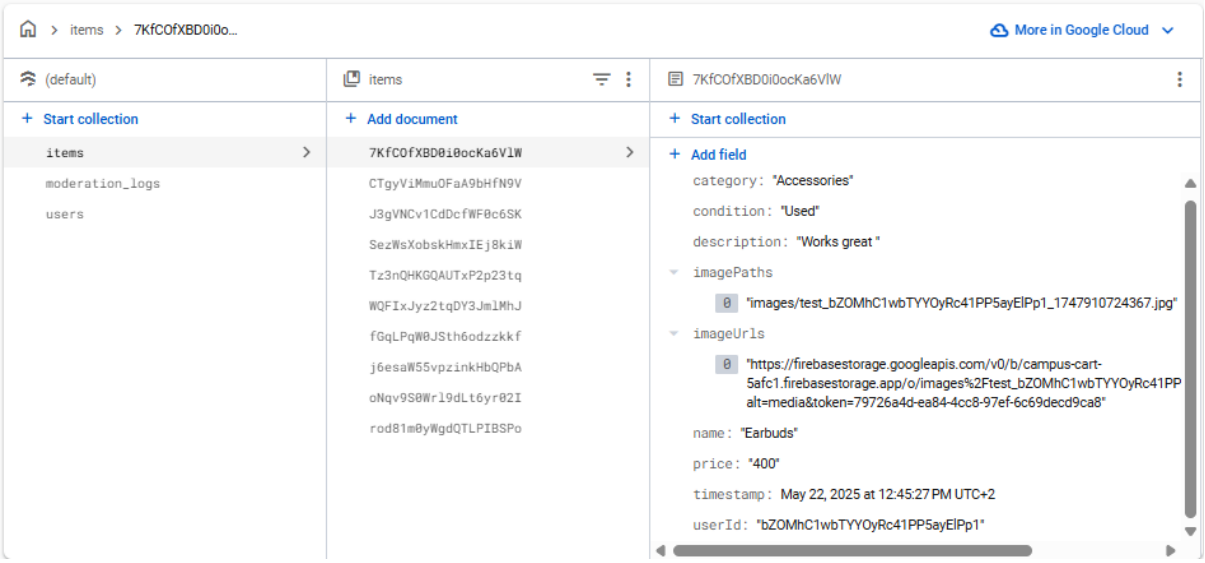
When the user submits the form, input values are captured from state variables and uploaded to Firestore using the `addDoc()` method. A server-generated timestamp is added to record the listing time.

```
const itemData = {
  name: itemName,
  description,
  userId: user.uid,
  price: condition === 'Free Giveaway' ? 'Free' : price,
  condition,
  category,
  imageUrls: uploadedImages,
  timestamp: new Date(),
};

await addDoc(collection(db, "items"), itemData);
```

Listing 5: Dynamic Item Upload to Firestore

The structure enables real-time updates, allowing all users to instantly see new listings without refreshing.



Figur 9: Firestore item document with metadata and image URL

6.2.2 Firebase Storage for Image Uploads

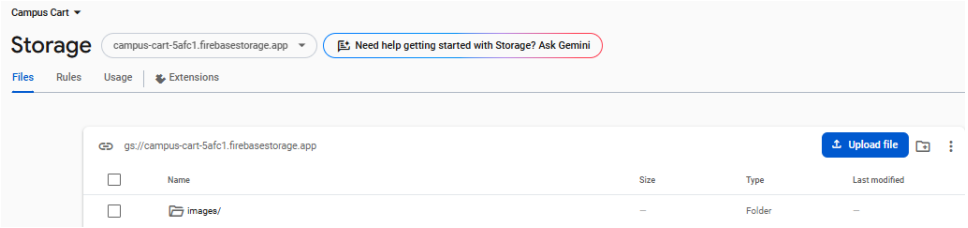
When users select or capture images while adding an item, the app uploads those files to Firebase Cloud Storage. Each local image URI is converted to a blob and uploaded using a unique filename based on the user ID and timestamp.

Once uploaded, a public download URL is retrieved and stored in Firestore along with the rest of the item data. These image URLs are later used to render the listing with visuals.

```
const response = await fetch(imageUri);
const blob = await response.blob();
const filename = `images/test_${user?.uid}_${Date.now()}.jpg`;
const storageRef = ref(storage, filename);
await uploadBytes(storageRef, blob);
const downloadUrl = await getDownloadURL(storageRef);
```

Listing 6: Uploading and Saving Image URLs

This approach ensures images are uploaded securely, uniquely named, and persistently linked to each item.



Figur 10: Firebase Storage for Uploaded Images

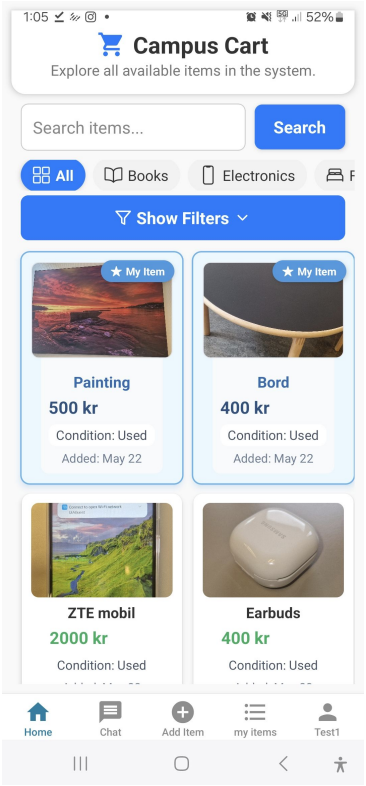
6.2.3 Firebase Firestore Integration on Home Screen

The Home Screen fetches the latest item listings from Firestore. Items are ordered by timestamp in descending order to ensure that newly posted items appear at the top. This data is retrieved using a one-time query.

The query is constructed with sorting, executed using `getDocs()`, and the resulting documents are mapped into a usable list. This list is then stored in a state variable and rendered in a scrollable UI.

```
1 const q = query(collection(db, "items"), orderBy("timestamp", "desc"));
2 const querySnapshot = await getDocs(q);
```

Listing 7: Firestore Query for Latest Items



(a) Home Screen

Figur 11: Home Screen displaying items fetched from Firestore

6.2.4 Add, Update, and Remove Item Listings

Campus Cart allows users to manage their listings through a streamlined interface that supports adding, editing, and deleting items. This process involves three core views: Category Selection, Add Item screen, and My Items screen.

When creating a new listing, the user is first presented with a category selection screen. Categories are rendered as buttons in a scrollable layout. Once a category is selected, it is passed to the Add Item screen via navigation parameters.

Item management is handled through state-bound input fields and Firebase services. When the user picks an image using Expo ImagePicker, the URI is stored and previewed. Each image is uploaded by converting it to a blob and saving it in Firebase Storage with a unique filename.

```
const blob = await (await fetch(uri)).blob();
const fileRef = ref(storage, `images/${user.uid}_${Date.now()}.jpg`);
await uploadBytes(fileRef, blob);
const url = await getDownloadURL(fileRef);
```

Listing 8: Upload Image and Get URL

To add or update an item, the app checks for the presence of an item ID. If editing, `updateDoc()` is called; otherwise, `addDoc()` creates a new Firestore document.

```
const ref = isEditing ? doc(db, "items", id) : collection(db, "items");
await (isEditing ? updateDoc(ref, data) : addDoc(ref, data));
```

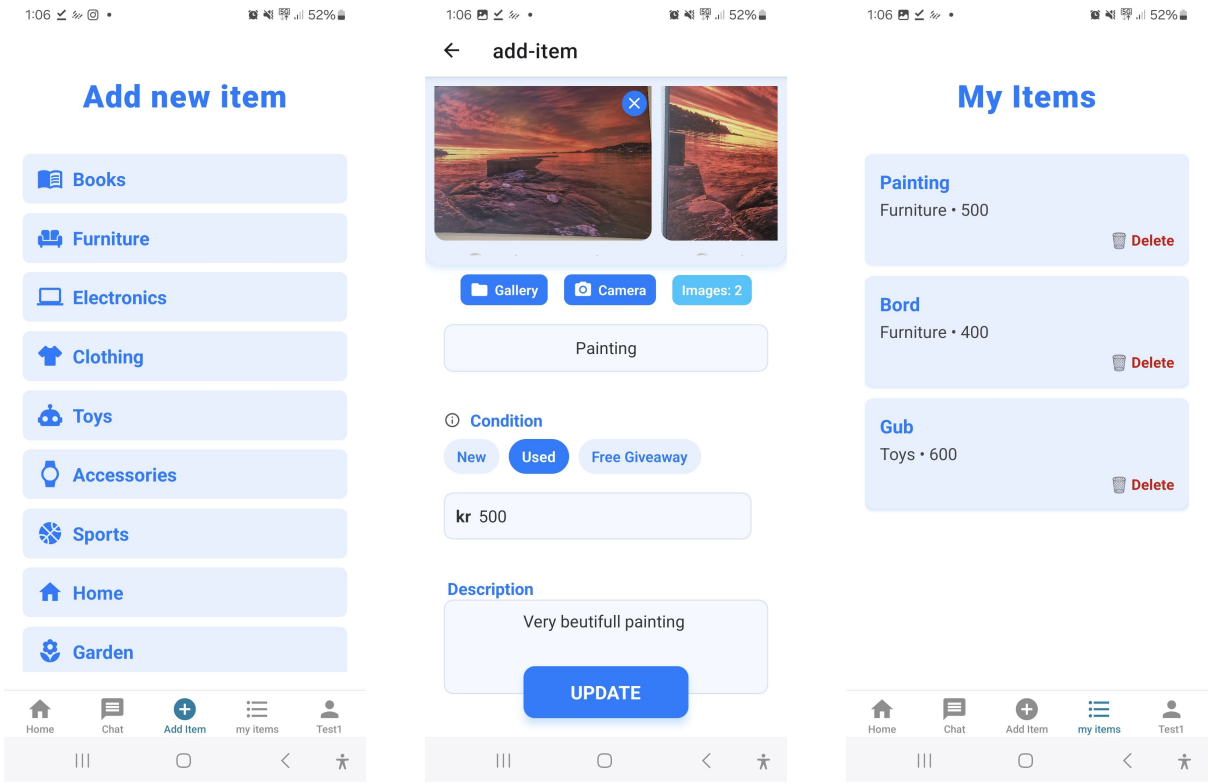
Listing 9: Add or Update Item

For deletion, the app removes both the Firestore item and its associated image from Storage:

```
await deleteObject(ref(storage, imagePath));
await deleteDoc(doc(db, "items", itemId));
```

Listing 10: Delete Item and Image

This setup ensures all listing operations are secure, efficient, and scoped to the authenticated user.

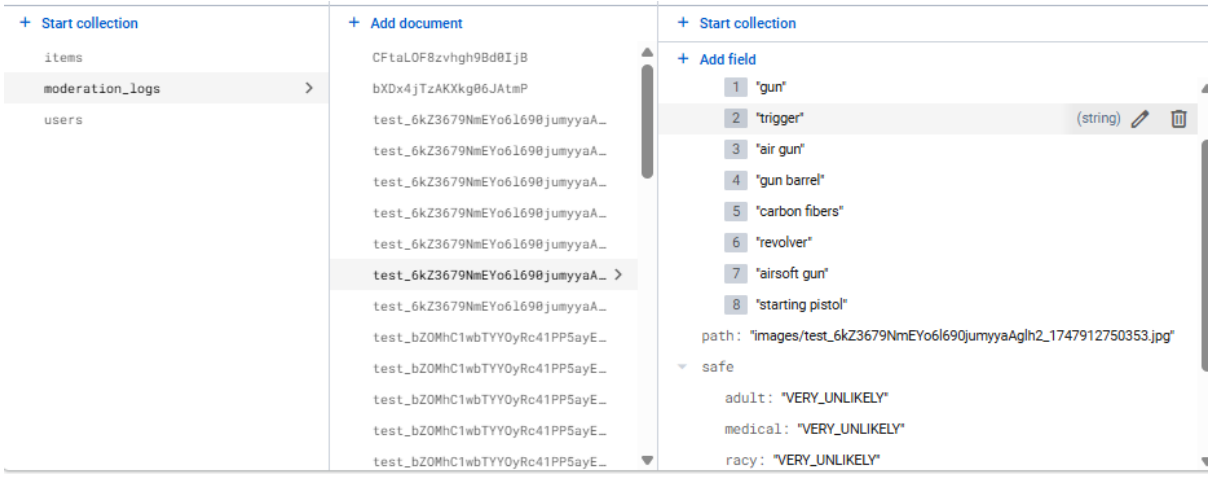


(a) Category Before Add **(b)** Add Item UI **(c)** My Items

Figur 12: Item management screens for adding, selecting categories, and viewing personal listings

6.2.5 Automated Image Moderation with Cloud Functions

To ensure that uploaded images follow content policies, an automated moderation system was implemented using Firebase Cloud Functions and the Google Cloud Vision API. The function is triggered each time a new image is uploaded to Firebase Storage. It analyzes the image using Google's safe search and label detection services.



Figur 13: Firestore moderation log showing detected labels and safety analysis

If the image is flagged as inappropriate—for example, if it is likely to contain adult, violent, or racy content, or if specific labels like "gun" or "weapon" are detected—the

image is automatically deleted from Storage. The corresponding item is also removed from Firestore, and a moderation notification is created for the user who uploaded it.

The function relies on the `@google-cloud/vision` package for image analysis, and `firebase-admin` and `firebase-functions` to access Firestore and define the trigger. The relevant dependencies were installed using:

```
npm install firebase-admin firebase-functions @google-cloud/vision
```

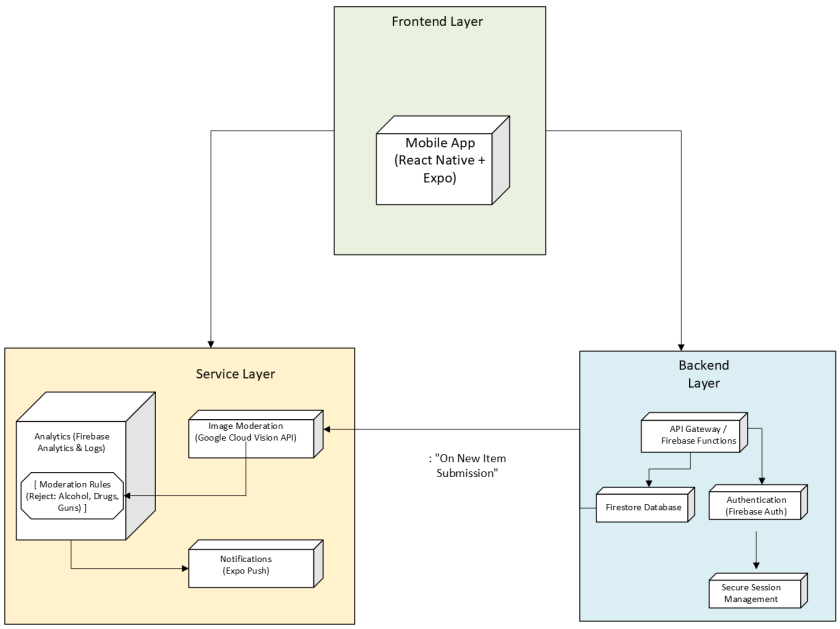
Listing 11: Installing Moderation Dependencies

Deployment is done through the Firebase CLI using:

```
firebase deploy --only functions
```

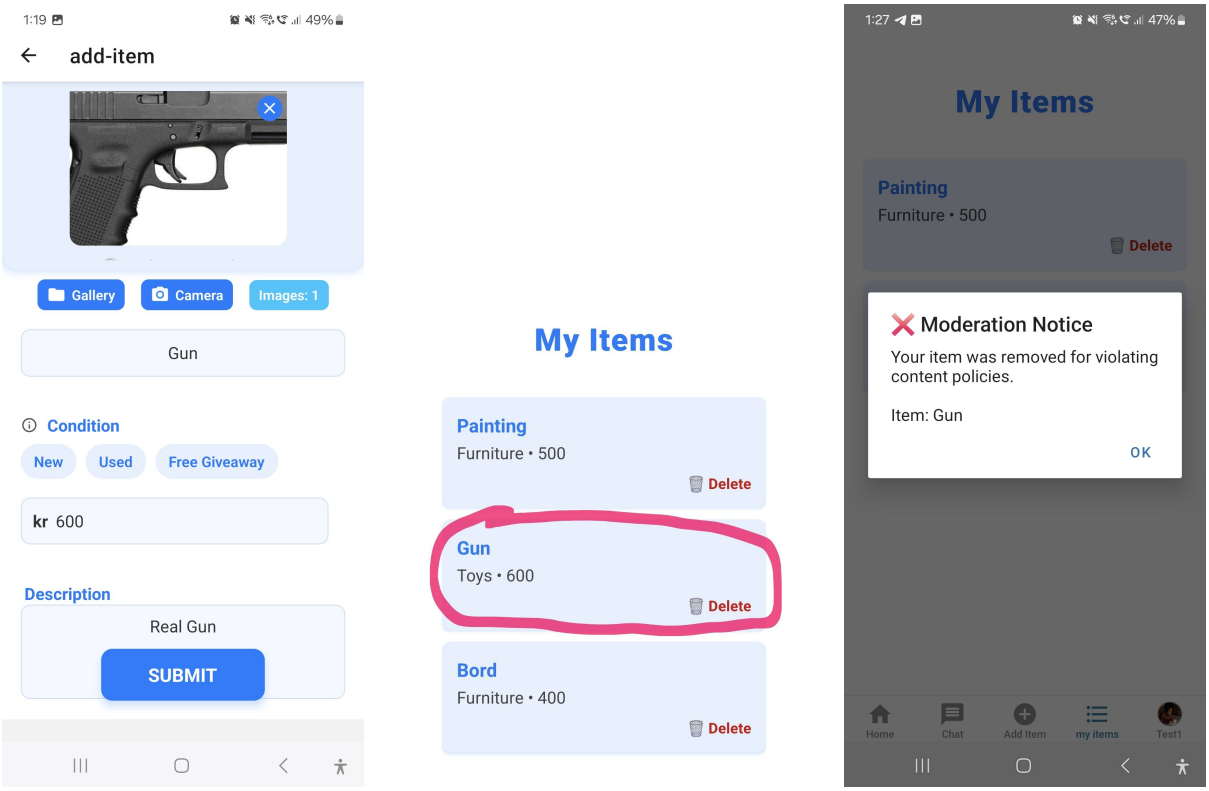
Listing 12: Deploy Moderation Function

This setup ensures that all uploaded images are screened in real-time and helps maintain a safe and trustworthy platform without requiring manual moderation.



Figur 14: Image moderation architecture for Campus Cart

Figure 14 shows the moderation workflow triggered when a user submits a new item with an image. The uploaded image is analyzed using the Google Cloud Vision API and notify the decision.

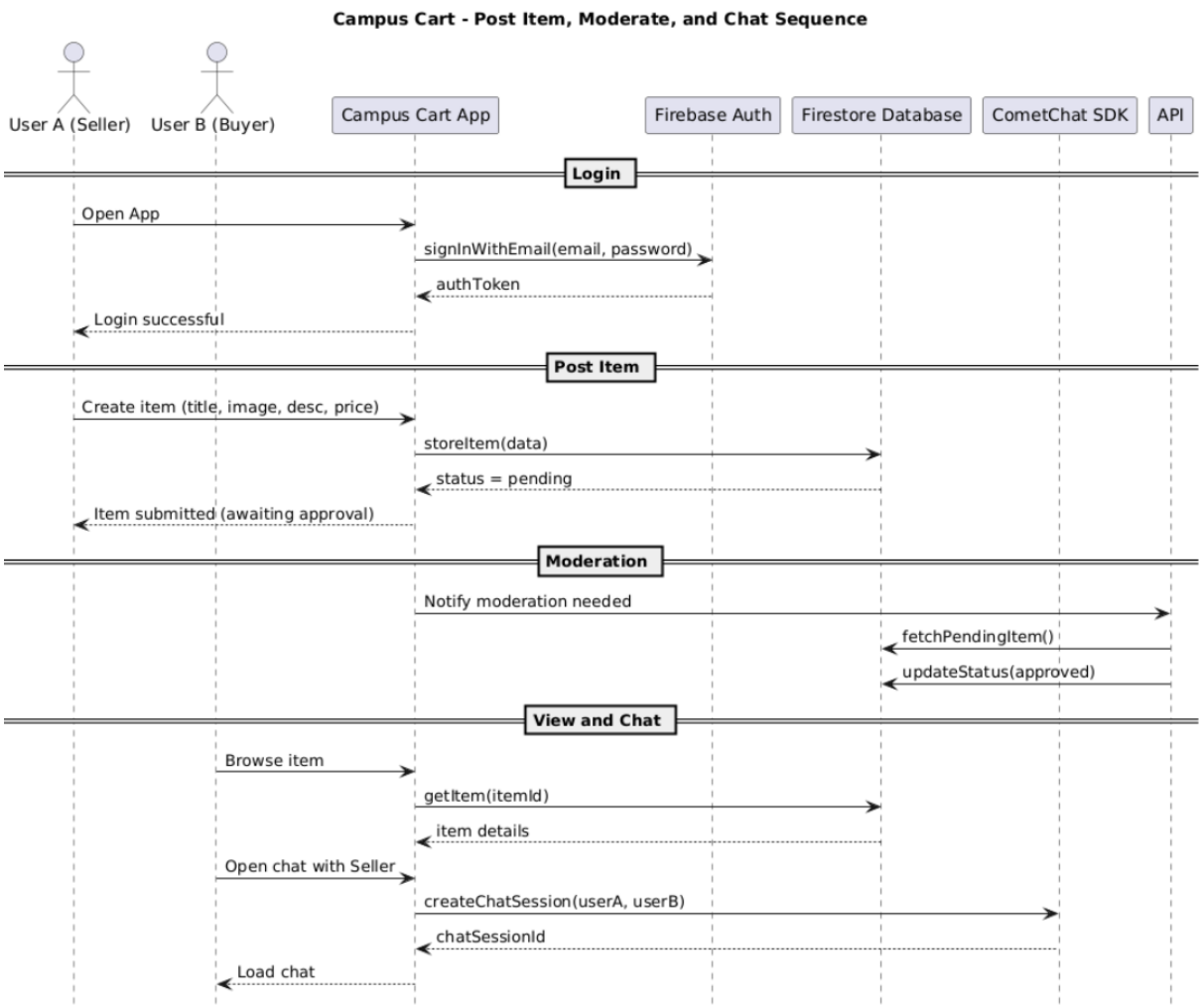


(a) Illegal Add Attempt **(b)** Illegal Item in List **(c)** Rejected Item Moderation

Figur 15: Item Listing, Management, and Moderation

6.2.6 System Flow: Authentication, Item Listing, Moderation, and Chat

The overall user interaction in Campus Cart follows a sequence that spans across multiple services including Firebase Authentication, Firestore, and CometChat SDK. This interaction is visualized in the sequence diagram below.



Figur 16: Sequence diagram of the Campus Cart flow

The user logs in with email and password, which is verified using Firebase Authentication. When a seller adds an item, it is saved to Firestore with a pending status. A Cloud Function checks the image using Vision API.

If approved, the item becomes visible. If not, the item and image are deleted. Buyers can view approved items and start a chat with the seller using CometChat.

6.3 Hooks and Components

6.3.1 Hooks

We created a reusable Hook called `useUser` to manage the authenticated user state across the app. It listens for authentication changes using Firebase's `onAuthStateChanged` and fetches profile data from the Firestore `users` collection.

This approach ensures that user information such as name and profile picture is always available to any component that needs it, without duplicating logic. It returns a single object that merges Firebase Auth data with Firestore profile fields.

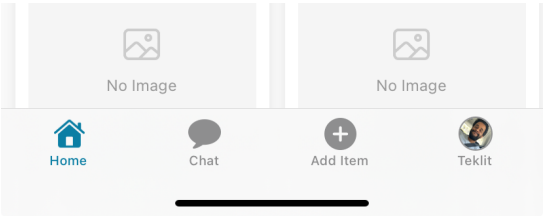
```
onAuthStateChanged(auth, async (firebaseUser) => {
  const snap = await getDoc(doc(db, 'users', firebaseUser.uid));
  setUser({ name: snap.data()?.name ?? '' });
});
```

Listing 13: Core logic of useUser Hook

Usage in Navigation The `useUser` hook is used in the app's tab layout to show the user's name and profile photo when available. This makes the interface feel personalized and responsive to login status. If the user is not logged in, the default icon and label are used instead.

```
<Tabs.Screen
  name="Account"
  options={{
    title: user?.name || 'Account',
    tabBarIcon: ({ color }) =>
      user?.photoURL ? (
        <Image source={{ uri: user.photoURL }} style={...} />
      ) : (
        <IconSymbol name="person.fill" color={color} />
      ),
  }}
/>
```

Listing 14: Dynamic Account Tab Icon with useUser



Figur 17: Dynamic tab icon and label based on authenticated user

6.3.2 Components

To protect certain screens from unauthenticated access, such as `AddItemScreen`, `MyItemsScreen`, and `ChatScreen`, we created a reusable `LoginPrompt` component. This component is conditionally rendered using a simple check: `if (!auth.currentUser)`.

It shows a message informing the user that login is required, along with a button that uses the `useRouter` hook to navigate to the login screen.

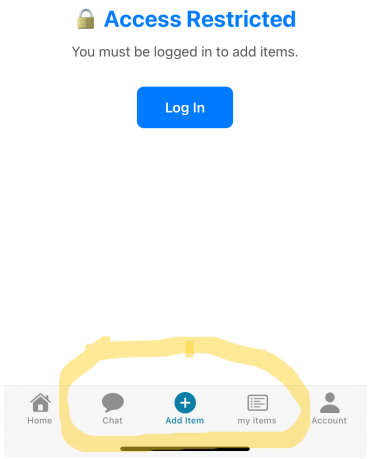
```
// Used in protected screens:
// if (!auth.currentUser) return <LoginPrompt />;

const router = useRouter();

<TouchableOpacity onPress={() => router.push('/login')}>
  <Text>Log In</Text>
</TouchableOpacity>
```

Listing 15: LoginPrompt usage and navigation logic

This approach improves user experience by guiding unauthenticated users to log in, while keeping the access logic clean and consistent across multiple parts of the app.



Figur 18: LoginPrompt component shown to unauthenticated users

6.4 Search and Filtering

CampusCart supports live search and dynamic filtering to help users find items quickly and efficiently. These features are implemented using React Native state management, local filtering logic, and Firestore real-time queries. the following implementations fulfills requirement FR8 and FR7.

6.4.1 How It Works

The search input updates results in real-time by filtering items already stored in memory. When a user types a keyword, the app filters the current list based on the item name, without triggering additional queries. This ensures fast local responsiveness.

Filtering by category, condition, and price uses real-time Firestore queries. The app constructs a query dynamically based on the user's selected criteria. Only filters that are set are applied. This selective logic minimizes performance cost and avoids unnecessary constraints.

Firestore's `onSnapshot` method is used to listen for live updates. Whenever a matching document is added, updated, or removed in the database, the filtered list on the client updates immediately. All filter values are stored in state, so changes in the UI automatically update the query.

6.4.2 Search Handler

The code below handles user keyword input. It performs a simple match on the `name` field and updates the local filtered list.

```
const handleSearch = (query) => {
  setFilteredItems(
    query
      ? items.filter(i => i.name.toLowerCase().includes(query.toLowerCase()))
      : items
  );
};
```

Listing 16: Search Input Filter

6.4.3 Firestore Filtering

This logic applies only active filters and updates the query accordingly. The result is streamed using `onSnapshot`.

```
let q = query(collection(db, "items"));
if (filters.category)
  q = query(q, where("category", "==", filters.category));
onSnapshot(q, snap => {
  const list = snap.docs.map(doc => ({ id: doc.id, ...doc.data() }));
  setFilteredItems(list);
});
```

Listing 17: Dynamic Firestore Query

6.4.4 Indexing and Interface Behavior

To support compound filtering efficiently, composite indexes were created in the Firebase Console. Firestore prompts the required index configuration when a multi-field query is attempted. The app uses a collapsible panel for filters, with styled inputs and icons for usability. All input fields are tied to local state, and any change automatically updates the query.

The overall experience is immediate, reactive, and aligned with Firebase's strengths in real-time data delivery. This approach ensures that users always view up-to-date results based on their current preferences.

6.5 Real-time Chat Implementation

To enable private messaging between users, CampusCart integrates real-time chat using CometChat, a third-party SDK that supports messaging, presence detection, and cross-platform compatibility. The integration is tied directly to Firebase Authentication to ensure secure and consistent user identity management.

6.5.1 Technology Stack and Setup

CometChat was selected for its support of real-time message delivery, Firebase authentication compatibility, and scalable infrastructure. It provides built-in presence features, typing indicators, and message history retrieval. React Native SDKs and Firebase tokens were used to bridge user authentication across both services.

6.5.2 Authentication Integration

Each time a user logs into CampusCart, the app attempts to log them into CometChat using their Firebase UID. If the user doesn't exist in CometChat, the app registers them using their Firebase profile data. This ensures the same credentials are used across both platforms without additional user setup.

6.5.3 Chat Flow and Context Management

Users can initiate a conversation directly from the item detail view. When they press "Contact Seller," a chat session is created using a custom conversation ID. This ID is constructed by sorting the buyer and seller UIDs along with the item ID. This ensures that each conversation is unique and consistent across sessions, even if a user returns to the item later.

6.5.4 Core Message Flow

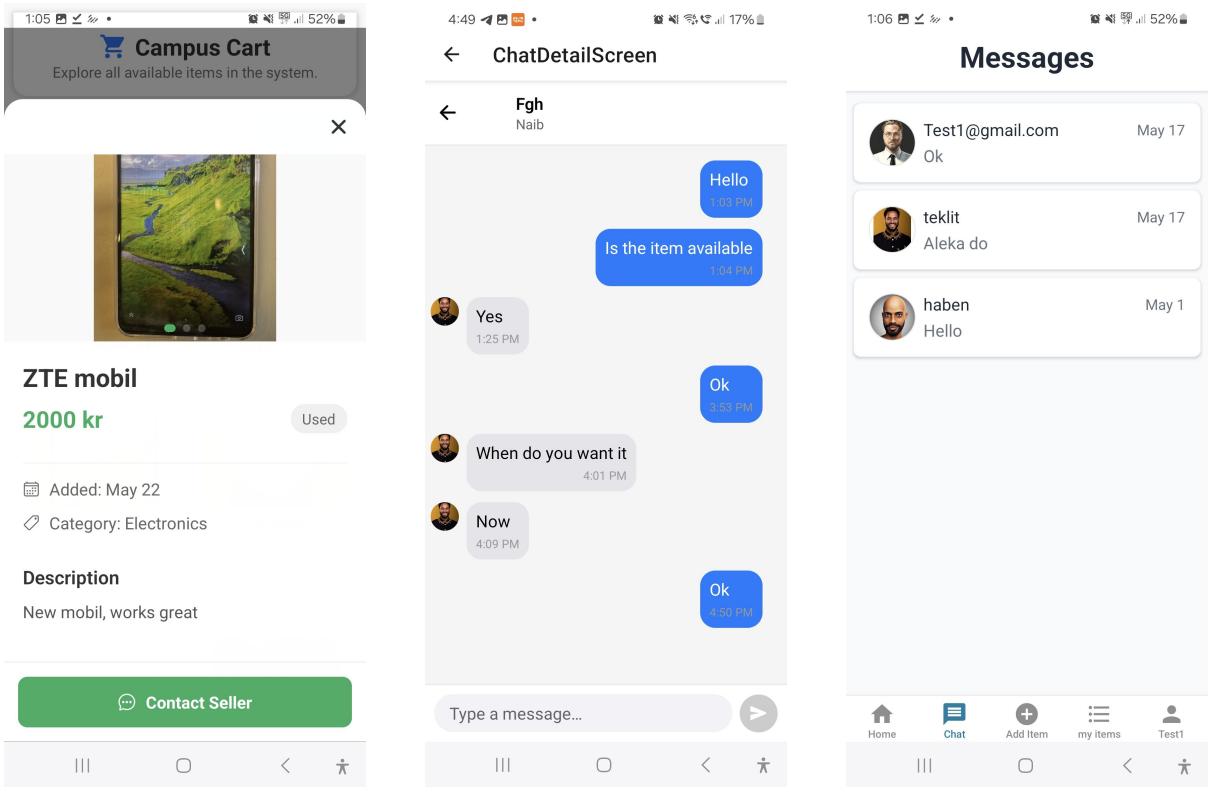
The main chat interface consists of two screens. The chat list shows all existing conversations. When a conversation is selected, the message screen opens, where users can exchange messages in real-time. Conversations are updated using listeners that receive new messages and typing indicators without requiring manual refresh.

6.5.5 Key Code Snippet: Starting a Chat

```
const conversationId = `${itemId}_${[user.uid, sellerId].sort().join("_")}`;
const conversation = await CometChat.getConversation(conversationId);
if (!conversation) {
  await CometChat.createConversation(conversationId, sellerId, 'user');
}
navigateToChat(conversationId);
```

Listing 18: Open or Create a Chat Room

6.5.6 User Interface



(a) Contact Seller **(b)** Chat Details Screen **(c)** Messaging Interface

Figur 19: In-App Messaging Screens

6.5.7 Performance and Security Considerations

To keep chats efficient, pagination is used to load older messages gradually. User data is cached to reduce redundant calls. Authentication checks ensure users can't message themselves or access others' conversations. Only logged-in users can initiate or receive messages.

The interface is styled to match the rest of the app, with readable chat bubbles, user names, and timestamps. The input field supports typing indicators and real-time delivery

acknowledgment.

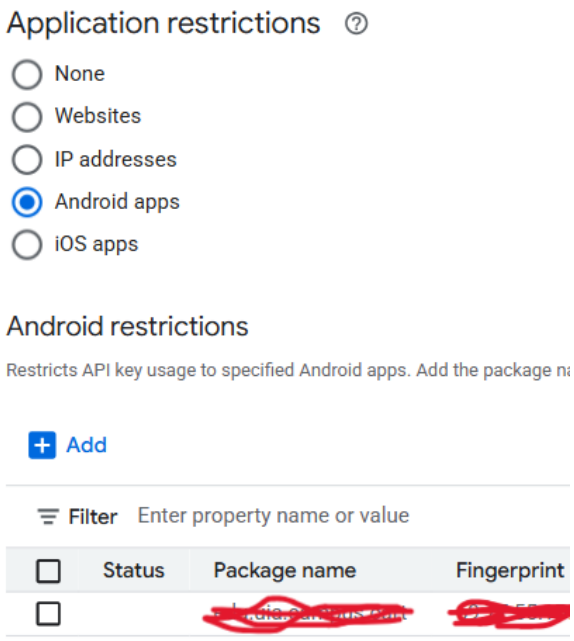
This implementation ensures reliable, secure, and user-friendly messaging between buyers and sellers, improving trust and communication within the CampusCart platform.

6.6 Application Security Practices

To protect user data and secure access to third-party services, several security measures were applied during the development of the Campus Cart mobile application. These include authentication management, controlled access to cloud resources, and the secure handling of sensitive credentials such as API keys.

6.6.1 Restricting Google API Access

Our application uses Google Cloud services, including the Cloud Vision API, which requires an API key for access. To prevent misuse, we applied strict restrictions to this key. Specifically, we configured it to allow requests only from our Android application by specifying the exact package name and the SHA-1 certificate fingerprint obtained from the EAS build system.



Figur 20: API key restricted to Android package and SHA-1 fingerprint

6.6.2 Why It Matters

Without proper restrictions, a publicly accessible API key can be used by unauthorized parties, leading to potential misuse, unexpected charges, or service abuse. By binding the key to our app's identity and signing certificate, we ensured that only requests from our verified app would be accepted by Google Cloud services.

7 Testing and Validation

This section outlines the testing strategy employed to validate the requirements specified in. We focus on two key testing methods: *Unit Testing* (to verify individual software components) and *User Acceptance Testing (UAT)* (to ensure the application meets real-world user expectations).

7.1 Overview of Testing Approach

Many testing approaches are available, such as integration testing, security testing, and usability testing[18]. For our project Campus Cart, we primarily use:

- **Unit Tests:** Ensure each functional unit (e.g., registration, item listing) works correctly in isolation.
- **User Acceptance Tests (UAT):** Validate that the system's core features satisfy end-user needs and adhere to the Must-have, Should-have, and Could-have requirements.

7.2 Unit Testing

7.2.1 Purpose and Scope

Unit tests in Campus Cart focus on the smallest, independent parts of our app to ensure they behave correctly. We verify the authentication and registration logic by confirming that valid student sign-ups succeed and that invalid credentials produce clear error messages. We also test the listing management flows to make sure users can create new listings, update existing ones, and delete them without issues. Search and filtering are covered by tests that simulate user queries, checking that items are correctly narrowed by category, course, and price. Finally, our moderation logic is exercised by flagging or reporting sample items and confirming that the appropriate internal processes are triggered.

7.2.2 Implementation

We chose Jest as our primary testing framework for its seamless integration with React Native and familiar syntax for JavaScript developers. Each core component or service has a corresponding test file where we write assertions against expected inputs and outputs. These tests run automatically through our continuous integration pipeline whenever code is pushed, helping us catch regressions early. To keep our codebase robust, we aim for at least eighty percent coverage on critical modules, ensuring that most branches and edge cases are exercised by the test suite.

7.2.3 Key Findings and Resolutions

Early in development, tests revealed that our registration flow lacked a rule to enforce minimum password length. We addressed this by adding a validation check and writing new tests to confirm that short passwords are rejected with a clear message. Another issue surfaced when users attempted to delete listings offline: the delete request would queue but then conflict upon reconnection. We fixed this by checking network availability before allowing a delete operation and added tests to simulate offline and online states, ensuring synchronization errors no longer occur. Overall, unit testing ensured that each *Must-have* requirement was functionally stable before proceeding to broader user tests.

7.3 User Acceptance Testing (UAT)

7.3.1 Rationale

User Acceptance Testing (UAT) provides insights into the application's usability and real-world performance. By engaging end users particularly students and university staff we verify that *Campus Cart* meets their expectations.

7.3.2 Test Scenario Overview

We asked each participant to work through the key "must-have," "should-have," and "could-have" features defined in functional requirements. Testers began by registering a new account, recovering a forgotten password, and completing the login flow. They then posted a listing entering a title, price, condition, and category before updating and deleting that same listing. Next, they used the search bar to find items by price range, category, or course code, and they tried flagging any suspicious listings to ensure our reporting mechanism worked. Finally, they uploaded one or more photos to their listings and visited their profile screen to confirm that all of their own items appeared correctly.

7.3.3 Results and Observations

Overall, participants were able to register and log in without difficulty, and every tester successfully created, edited, and removed a listing. Most found the search filters accurate, though one person noted that matching an exact course code could be clearer. When reporting a listing, everyone triggered the flagging process correctly, but a couple of testers felt the confirmation message could be more descriptive. Image uploads succeeded in all cases, with only minor delays when multiple photos were attached. On the profile screen, every user saw their active listings displayed as expected.

7.3.4 Action Items

Based on these insights, we made several refinements. We changed the "Save" button label to "Post" to eliminate any confusion around editing versus publishing. We enhanced the search filter logic so that exact course codes return reliably. We rewrote the reporting confirmation text to make it clearer when a listing has been flagged. Finally, we optimized the image uploader to reduce delays when multiple files are submitted at once.

7.4 Usability Testing and Feedback:

Over two weeks in May 2025, five students from UiA Agder Grimstad tested Campus Cart's main features, including our restricted-item moderation. They completed sign-up and login, created, edited, and deleted listings, searched and filtered by category, price, and course, uploaded photos, accessed their profiles, and logged out. They also tried posting restricted items (for example, alcohol) to confirm that our moderation flags them correctly.

Each tester completed tasks in a think-aloud session while we recorded pass/fail results against our predefined checklist. They spoke their thoughts as they navigated the app, and then tried to post restricted items to confirm our flagging system, like it is shown in the figure all the testes were passed.

All five successfully signed up with their student email, logged in with valid credentials, and saw incorrect logins rejected. They posted, edited and removed listings without errors. Every listing included title, description, price and condition as required. Search and filter

returned accurate results, though one tester suggested renaming "Course" to "Course Code" for clarity. Photo uploads always worked, with inline previews. Every restricted-item post was flagged instantly and routed to moderators, with no false positives. Profile access and secure logout also passed with no issues. The app prevented duplicate accounts using the same email in every case.

Based on this feedback, we renamed the "Course" filter to "Course Code" and added a tooltip. We added drag-and-drop support to the photo unloader. We built a bulk-review action into the moderator panel for faster handling of flagged items.

All tests passed successfully. We will release our application with these updates and plan a broader survey after launch to gather more feedback.

Requirement Tested	Alex	Nora	David	Leila	Markus
Users can sign up with their student email.	✓	✓	✓	✓	✓
Existing users can log in with valid credentials.	✓	✓	✓	✓	✓
Invalid login attempts are correctly rejected.	✓	✓	✓	✓	✓
Logged-in users can post a new item for sale.	✓	✓	✓	✓	✓
Users can edit or remove their listings.	✓	✓	✓	✓	✓
Listings must include title, description, price, and condition.	✓	✓	✓	✓	✓
Search and filter work by category, price, and course.	✓	✓	✓	✓	✓
Users can upload photos for their listings.	✓	✓	✓	✓	✓
Reported items are flagged and sent to moderators.	✓	✓	✓	✓	✓
Users can access their profile and view their items.	✓	✓	✓	✓	✓
The app logs out users securely.	✓	✓	✓	✓	✓
The app prevents duplicate accounts using the same email.	✓	✓	✓	✓	✓

Tabell 2: User Acceptance Testing Results for Campus Cart

7.5 Conclusion and Next Steps

Combining **Unit Tests** and **User Acceptance Testing** allowed us to verify both the technical robustness of *Campus Cart* and its alignment with user needs:

- **Unit Testing** uncovered bugs early, ensuring each core module met baseline quality.
- **UAT** confirmed that the final product addressed end-user expectations, particularly Must-have requirements essential for launch.

Further testing (e.g., load testing, security assessments) can be introduced as the app scales to a larger user base. For now, the completed test cycles suggest that *Campus Cart* is ready for deployment, meeting the project's functional and non-functional requirements.

8 Discussion

This section evaluates the project by comparing its goals, requirements, and intended functionality with the actual results. It includes a reflection on key decisions, the effectiveness of implemented solutions, challenges encountered, and how they were resolved. Alternatives that were considered but not implemented are also discussed, along with an assessment of what was learned and what could be improved in future iterations

8.1 Process, Meetings, and Sprint Planning

The project followed an Agile-inspired process with some flexibility in sprint length, depending on workload and availability. While most sprints lasted around 10 days to two weeks, the team consistently held weekly meetings to maintain steady communication and coordination.

Each weekly meeting served as a checkpoint to discuss completed tasks, address any blockers, and plan goals for the upcoming sprint. Team members shared updates on their progress and documented what was done during the week, which helped ensure transparency and accountability. The meetings also provided a forum for reassigning or re-scoping tasks as needed based on feedback or development challenges.

Taiga was used as the primary project management tool to organize the backlog, assign tasks, and visualize sprint progress. The team also integrated time tracking using Toggl, allowing better insight into how time was spent across different activities. This process allowed the group to adapt quickly, iterate on features based on technical or user feedback, and stay aligned with the project goals throughout the development cycle.

8.2 Problems Encountered and Solutions

8.2.1 Chat Feature Limitations

We faced challenges with chat features, especially offline messages and media support. These features are not included in this version but are planned for future updates.

8.2.2 Form Field Reversion

In the Add/Edit form, user input would sometimes reset. This happened because the code that updates the screen was running too often. We fixed it by only updating the state when the data actually changed.

8.2.3 Image Upload Issues

At first, deleting or updating images didn't work properly. We solved this by fixing how image URLs were managed in the app and database, and by updating the app's local state right after image changes.

8.2.4 Parameter Conversion and Deprecated API

Some data from the router came in the wrong format. We added helper code to convert it properly. We also replaced old code from the image picker API that no longer worked.

8.2.5 Navigation Issue After Submission

After a user submitted an item, they were not redirected to the "My Items" screen. This was due to a wrong route name. We fixed it by updating the navigation path to match the app's structure.

8.2.6 CometChat Integration

We tried using CometChat's UI Kit but had problems because of Expo's native module limits. Instead, we used only the core SDK and built our own simple chat UI. This way, we kept the core messaging working.

8.3 Evaluation of the Results

We completed key features like item listing, editing, deletion, and image moderation. Google Cloud Vision successfully flagged unsafe content. Real-time messaging works using CometChat's core SDK, although some features like media messages and typing indicators were not added. Google Sign-In was partly implemented but will be finished in future versions. Overall, the app is stable and meets most of the main goals.

8.4 Justification for Chosen Solution

We picked CometChat because it provided ready-to-use chat features, saving us time. Firebase was used for user login, storing item data, and running functions like image moderation. These tools worked well with React Native and helped us move quickly.

8.5 Alternatives and Their Evaluation

We considered building the chat system ourselves using Firebase. This would have given us more control, but it also meant writing everything from scratch like message status and typing indicators. CometChat gave us those features with less work, so we chose it and built our own UI to go with it.

8.6 Meeting the Project Requirements

The app includes the main features: login, item management, messaging, and image moderation. Users can post items, and the system removes inappropriate images. While some advanced chat features and full Google Sign-In are still missing, the app works well and meets most of the requirements.

8.7 Lessons Learned

We learned that it's important to test third-party tools early — especially when using Expo. Some tools like CometChat's UI Kit didn't work well with our setup. We also saw how useful Git branches and code reviews can be. Clear task planning and writing things down every week helped us stay on track.

8.8 Potential Improvements

In the future, we could support media messages in chat, add performance improvements for image uploads, and make the app easier to use for more people by supporting multiple languages and better accessibility.

8.9 Financial and Scalability Considerations

Campus Cart uses some Google services that may cost money if used a lot. Firebase Storage was used to store images, and Google Cloud Vision API was used to check images for safety. These services are free up to a limit but can cost more as usage grows.

If more students use the app, storage and image checking could become expensive. To handle this, we plan to add banner ads and optional paid features. This way, the app stays free for most users but can support itself as it grows.

8.10 Future Development

While Campus Cart includes most of the core features, several planned functionalities were not fully implemented and are considered for future development. These include full support for Google Sign-In, advanced chat features such as media messaging, delivery receipts, and typing indicators. Offline messaging support in the chat system is also a missing feature that would improve user experience.

In addition, future versions could include smarter search and filtering, item recommendation systems, and better moderation tools for administrators. Expanding the app to support iOS and integrating with university portals are also potential improvements. These additions would make the app more complete, scalable, and suitable for real-world deployment.

9 Conclusion

The Campus Cart project successfully delivered a mobile application tailored for university students to buy and sell second-hand items within their local academic community. The primary objective was to develop a secure, user-friendly, and feature-rich platform that includes login functionality, item management, real-time messaging, and automated content moderation.

Through iterative development and the use of modern technologies such as React Native, Expo, and Firebase, the team implemented the core functionalities efficiently. Key features include user authentication (with plans for full Google Sign-In), item listing with image support, and automated image moderation using the Google Cloud Vision API. These elements contribute to a safer, more trustworthy marketplace experience.

Although some features, such as advanced chat capabilities and full Google Sign-In remain as future improvements, the current version provides a stable foundation that addresses the core needs of its users. Feedback from user testing validated the platform's usability and its relevance for students seeking a simple and effective way to exchange items.

This project demonstrates how agile development, modern frameworks, and a usercentered approach can produce a scalable and extensible solution. Future versions may expand functionality to include integration with university portals, and personalized recommendations, further enhancing the value of Campus Cart for its target audience.

Referanser

[1] *Designing User Interfaces*. URL: https://books.google.no/books?hl=en&lr=&id=gMA5EAAAQBAJ&oi=fnd&pg=PT19&dq=Figma+to+create+wireframes+and+interactive+prototypes&ots=WOtci9NAXA&sig=aKZ18TyWA70vYHiXR-HL-zU4Z98&redir_esc=y#v=onepage&q&f=false (sjekket 22.05.2025).

[2] *Advanced Flutter: Databases and Layered Architecture - Google Books*. [Online; accessed 22. May 2025]. Mai 2025. URL: https://www.google.no/books/edition/Advanced_Flutter_Databases_and_Layered_A/bfTYEAAAQBAJ?hl=en&gbpv=1&dq=firebase+uses&pg=PA186&printsec=frontcover (sjekket 22.05.2025).

[3] OpenAI. *ChatGPT*. Accessed: 2025-05-22. OpenAI. 2025. URL: https://chatgpt.com/.

[4] Grammarly, Inc. *Grammarly: Free AI Writing Assistance*. Accessed: 2025-05-22. Grammarly. 2025. URL: https://app.grammarly.com/ddocs/2831511458.

[5] Tatiana Kravchenko, Tatiana Bogdanova og Timofey Shevgunov. «Ranking Requirements Using MoSCoW Methodology in Practice». I: *Cybernetics Perspectives in Systems*. Cham, Switzerland: Springer, jul. 2022, s. 188–199. ISBN: 978-3-031-09073-8. DOI: 10.1007/978-3-031-09073-8_18. (Sjekket 22.05.2025).

[6] [Online; accessed 22. May 2025]. Sep. 2022. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1f18a39d970af73b3edf1d8fd34fea914bb7b71b (sjekket 22.05.2025).

[7] React Team. *React*. URL: https://react.dev/.

[8] TypeScript Team. *TypeScript: JavaScript With Syntax For Types*. https://www.typescriptlang.org/. Accessed: 2025-05-20. 2025.

[9] TypeScript Team. *TypeScript Handbook: Introduction*. https://www.typescriptlang.org/docs/handbook/intro.html. Accessed: 2025-05-20. 2025.

[10] Expo. *Expo: Build universal native apps with React*. https://expo.dev/. Accessed: 2025-05-20. 2025.

[11] [Online; accessed 22. May 2025]. Mai 2025. URL: https://www.researchgate.net/profile/Anil-Gaikwad-12/publication/362539877_FIREBASE_-OVERVIEW_AND_USAGE/links/62efc738505511283e9a5318/FIREBASE-OVERVIEW-AND-USAGE.pdf (sjekket 22.05.2025).

[12] *Google Cloud AI Services Quick Start Guide*. URL: https://books.google.no/books?hl=en&lr=&id=aHteDwAAQBAJ&oi=fnd&pg=PP1&dq=he+Google+Cloud+Vision+API+is+a+machine+learning+service+provided+by+Google+that+enables+applications+to+understand+the+content+of+images.+&ots=JCN-vEexlt&sig=RiYUsLEmjGZGgG3gW5SB0pHyvjo&redir_esc=y#v=onepage&q=he%20Google%20Cloud%20Vision%20API%20is%20a%20machine%20learning%20service%20provided%20by%20Google%20that%20enables%20applications%20to%20understand%20the%20content%20of%20images.&f=false (sjekket 22.05.2025).

[13] Tomaso Erseghe. *Analysis of Robust Internet Instant Messaging Protocols for Chat Applications*. [Online; accessed 22. May 2025]. 2021. URL: https://thesis.unipd.it/handle/20.500.12608/30731 (sjekket 22.05.2025).

[14] [Online; accessed 22. May 2025]. Jun. 2021. URL: https://www.ljsd.org/documents/Board/Board-Policies/Series-1000---Community-Relations/BP-1311-Civility.pdf (sjekket 22.05.2025).

[15] Kaleidos Open Source. *Taiga: The free and open-source project management tool*. https://www.taiga.io/. Accessed: 2025-05-20. 2025.

[16] React Team. *Introducing Hooks*. https://legacy.reactjs.org/docs/hooks-intro.html. Accessed: 2025-05-10. n.d.

[17] Vijay Gadepally et al., red. *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Springer, 2020. ISBN: 978-3-030-33752-0. URL: https://doi.org/10.1007/978-3-030-33752-0.

[18] *Usability Testing Essentials: Ready, Set ...Test!* URL: https://books.google.no/books?hl=en&lr=&id=L6_SDwAAQBAJ&oi=fnd&pg=PP1&dq=Many+testing+approaches+are+available,+such+as+integration+testing,+security+testing,+and+usability+testing&ots=kBeMU6D9x9&sig=p2xQrA5MVzVCEqGANQDBjKh8E4Y&redir_esc=y#v=onepage&q&f=false (sjekket 22.05.2025).

Dictionary

API: Application Programming Interface. Defines methods and data formats allowing different software components to interact with each other.

App Store: A digital platform provided by Apple to distribute applications for iOS devices.

Authentication: The process of verifying the identity of a user or system to secure access to the application.

Backend: The server-side part of an application responsible for data handling, logic, and security management.

CometChat: A third-party platform providing ready-made APIs and SDKs to integrate real-time chat functionalities into applications.

CSS: Cascading Style Sheets. A language used to define visual styles and layout of application interfaces.

Expo: An open-source framework and toolset used to build, test, and deploy React Native applications efficiently.

Firebase: Google's platform providing backend services including user authentication, real-time database management, cloud storage, and hosting for mobile and web apps.

Frontend: The user-facing side of an application, involving visual elements like screens, buttons, and user interactions.

Git: A version-control system used to manage source code changes and collaboration among developers.

GitHub: A platform based on Git for hosting, managing, and collaborating on software projects.

iOS: The mobile operating system developed by Apple, used for devices such as iPhone and iPad.

JavaScript: A programming language used for creating dynamic and interactive application functionalities.

Node.js: A JavaScript runtime environment that allows running JavaScript on the server-side, commonly used with Expo and React Native.

NPM: Node Package Manager. A package manager for the JavaScript ecosystem, managing external libraries and dependencies.

React Native: An open-source framework used for developing native mobile applications for Android and iOS using JavaScript and React.

Responsive Design: An approach to design ensuring interfaces work effectively across different devices and screen sizes.

Tech Stack: The combination of programming languages, frameworks, and tools used to build an application.

UI (User Interface): The graphical layout of an application, including visual elements users interact with.

UX (User Experience): The overall experience of a user when interacting with an application, encompassing usability and accessibility.

Visual Studio Code (VS Code): A popular, open-source source-code editor developed by Microsoft used for software development, debugging, and version control.

Android Studio: The official integrated development environment (IDE) for Google's Android operating system, often used for testing React Native applications.

Google Play Store: Digital distribution platform operated by Google, used for releasing Android applications.

Debugging: The process of identifying and resolving errors and issues within the application's code.

State Management: Techniques used to manage and maintain application data states effectively across the application lifecycle.

Hooks: A feature in React allowing the use of state and other React features in functional components without writing class components.

SDK: Software Development Kit

UML: Unified Modeling Language

A Time Sheet

The project time sheets are available in the appendices folder, named `time sheet.pdf`.

B Sprint Reports

The reports for each sprint are also found in the appendices folder, named `sprint.pdf`.

C Meeting Minutes

The meeting minutes for all participants of the project are also found in the appendices folder, named `Meeting Summary.pdf`.

D Product Vision

The product vision of this project is also found in the appendices folder, named `Product vision.pdf`.

E Group Contract

The group contract of the participants in this project is also found in the main folder of the assignment.

F Individual Report

The individual report of every person in the group is written in one PDF file and is found in the main folder of the project.

G Demo Video

The demo video is found in the appendices folder, named `projectDemo.mp4`.

H The GIT Short-Log

The GIT short-log is found in the appendices folder.