OBJECT ORIENTED DEVELOPMENT GROUP ASSIGNMENT

chong yi te

[COMPANY NAME] [Company address]

DIPLOMA IN INFORMATION TECHNOLOGY

OBJECT-ORIENTED DEVELOPMENT (DIT1334)

GROUP ASSIGNMENT

ASSIGNMENT TITLE:

Campus Canteen Feedback Application

We hereby certify that this assignment is our work and where materials have been used from other resources, they have been properly acknowledged. We also understand we will face the possibility of failing the module if the content of this assignment is plagiarized.

No.	Name	Student ID	Signature
1	Te Chong Yi	SCSJ2400992	
2	Chew Zhi Xiang	SCSJ2500184	
3	Sam Lai Jia Xian	SCSJ2500239	

Date:___________________

LECTURER NAME	:	**NUR DIANA MADINAH BINTI AB HADI**
RELEASE DATE	:	**25th SEPTEMBER 2025**
SUBMISSION DATE	:	
PRESENTATION DATE		

MARK OBTAINED:

Table of Contents

PART A: Case Study Analysis

Introduction:

CCF (Campus Canteen Feedback App) is a mobile application specifically designed for schools to systematically collect student evaluations and improvement suggestions regarding campus cafeteria dishes. The app provides students with a secure, anonymous feedback platform where they can record dish names, ratings (1-5 stars), relevant tags (e.g., spicy, healthy, vegan), and comments.

Beyond feedback collection, CCF features data analytics capabilities that generate comprehensive reports. These include average dish ratings, the top three most popular dishes, and the frequency of each tag. This empowers cafeteria management to understand student preferences, enhance food quality, and promote healthier, more diverse dining options.

Reason for development:

Lack of a Systematic Feedback Mechanism:

- Existing approaches such as verbal comments or paper forms are fragmented, inconsistent, and often misplaced. The app provides a structured and centralized digital repository.

Inability to Effectively Analyze Student Preferences and Needs:

- Management currently has no insight into popular dishes, quality issues, or dietary preferences (healthy, vegan, spicy, etc.). The app generates analytics to fill this gap.

Insufficient Communication Channels Between Students and Cafeterias

- Students have no secure channel to provide detailed suggestions. The app encourages participation through anonymity and ease of use

Promoting Healthy and Diverse Food Culture

- Tag-based analysis enables cafeterias to identify trends and promote healthier options

Enhancing Management Efficiency and Decision Quality

- Automated reporting reduces manual work and supports data-driven menu adjustments.

Objectives of the CCF Application:

Table 1: Objectives of Establish a Systematic and Centralized Feedback Mechanism

SMART Criteria	**Details**
Specific	Provide a structured mobile platform for recording dish name, rating, tags, and comments.
Measurable	Achieve a minimum of 10 valid feedback submissions within the first month of deployment.
Achievable	The app features intuitive UI and real-time database syncing.
Relevant	Solves the current issue of unstructured, low-volume feedback.
Time-bound	To be achieved within one mounth after deployment.

Table 2 Objectives of Implement data-driven student preference analysis functionality

SMART Criteria	**Details**
Specific	Automatically generate average dish ratings, ranking lists, and tag-frequency charts.
Measurable	Reduce manual analysis time by 50%.
Achievable	Built-in analytics module performs calculations instantly.
Relevant	Supports management's menu improvement decisions.
Time-bound	Fully functional within 1 month of implementation.

Table 3 Objectives of Improve Communication Channels Between Students and the Cafeteria

SMART Criteria	**Details**
Specific	Provide feedback submission and a dedicated Contact Us channel with optional photos.
Measurable	Increase student participation rate in feedback submission by at least 10% within the first semester.
Achievable	Simple UI increase user motivation.
Relevant	Addresses weak communication in current system.
Time-bound	Target to be met within one month

Table 4 Objectives of Promote a Healthy and Diverse Food Culture on Campus

SMART Criteria	**Details**
Specific	Identify and highlight "healthy" or "green" meals based on feedback and tag analytics, displaying them on menus.
Measurable	Increase the proportion of healthy or green dishes by at least 15% in cafeteria menus.
Achievable	Achieved by utilizing analytics data to guide menu updates and labeling strategies.
Relevant	Supports the goal of promoting a healthy and diverse food culture aligned with school wellness initiatives.
Time-bound	Achieve the increase within one months of implementation.

Table 5 Objectives of Enhance Management Efficiency and Decision-Making Quality

SMART Criteria	**Details**
Specific	Implement an automated reporting and visual analytics module that summarizes student feedback data for administrators.
Measurable	Achieve a 50% reduction in manual data analysis time and enable reports to be generated on demand within 2 minutes.
Achievable	Feasible using existing data and dashboard visualization tools integrated into the system.
Relevant	Aligns with the rationale of enhancing management efficiency and decision-making quality through data-driven insights.
Time-bound	Fully operational within one months after deployment.

PACT target user:

P – people:

- **Primary Users:** Students who dine frequently on campus.
- **Secondary Users:** Cafeteria managers, administrative staff.

A – Activities:

- Students submit feedback and browse dish rankings.
- Admins review reports, charts, and statistical summaries**.**

C – Contexts:

- Campus environments (cafeterias, dorms, academic buildings).
- Usage anytime with mobile internet or WiFi.

T – Technologies:

- Android mobile app.
- Firebase realtime database.
- Data analytics module.

Conclusion:

The target user group for the CCF application was identified through the PACT framework.

Primary users are students who frequently dine at campus cafeterias, while secondary users are cafeteria management staff.

I-P-O analysing:

Table 6 IPO analysing of Student

Input	Login/register information
	Dish name, rating, tags, comment
	Photo (optional)
Process	Validate identity
	Store feedback in database
	Categorize and analyze data
Output	Successful login
	Feedback stored and displayed

Explanation:

- **Input Stage:** Students log in using their Student ID and submit feedback related to cafeteria dishes or services.
- **Process Stage:** The system validates the student’s identity, stores the feedback in a structured database, organizes it by category, and performs data analysis.
- **Output Stage:** The system will show that the login is successful, if the account exists and the student submits feedback, the feedback submitted by the student will be displayed

Table 7 IPO analysing of Admin

Input	Admin login credentials
	Filter and analytics commands
Process	Authenticate admin
	Retrieve feedback records
	Generate average ratings, rankings, tag counts
	Convert analytics into charts
Output	Dashboard with insights
	Analytical charts

Explanation:

- **Input Stage:** Administrators log in using their Admin ID and password, then request access to feedback reports or apply filters such as dish name, or tag type
- **Process Stage:** The system verifies the administrator's credentials, retrieves stored feedback data, performs data analysis to calculate averages, identify top dishes, and summarize, and then converts the results into visual reports.
- **Output Stage:** The system confirms successful login, displays an interactive dashboard with analytical charts

Input	**Access request** **Navigation command (e.g., Clicking on the "Feedback", "Rank", "Setting", or "Contact us" links)**
Process	**Handle Access Request (Page Load Logic)** • The system identifies the user's status as "Guest" (unauthenticated). • The system logic determines to fetch **only 3 feedback items**. • The system logic sets the "category filter" status to **disabled** **Handle Navigation Command (Permission Logic)** • The system receives the guest's Navigation command. • The system verifies the user's "Guest" status. • The system **blocks** navigation to the requested page. • The system prepares the restriction prompt message.
Output	• **Restricted Home Page screen** Displays a list of **only 3 feedback items**. Displays a **disabled category filter**. • **Login/Registration prompt message** Displayed when the user clicks on a navigation link: "Want to see more? Login or register to continue"

Explanation:

Input Stage: The Guest provides two types of input:

- An **Access request** （by opening the application） .
- A **Navigation command** (by clicking a link like "Feedback").

Process Stage: The system handles these inputs separately.

- **For the Access Request:** The system logic identifies the user as "Guest," **determines** to fetch only 3 feedback items, and **sets** the category filter status to disabled.
- **For the Navigation Command:** The system verifies the "Guest" status, **blocks** the navigation attempt, and **prepares** the restriction prompt message.

Output Stage: The system produces two distinct outputs based on the processing:

- In response to the Access Request, it **displays** the Restricted Home Page （with 3 items and a disabled filter） .
- In response to the Navigation Command, it **displays** the "Login/Registration prompt" when navigation is blocked.

Flowchart

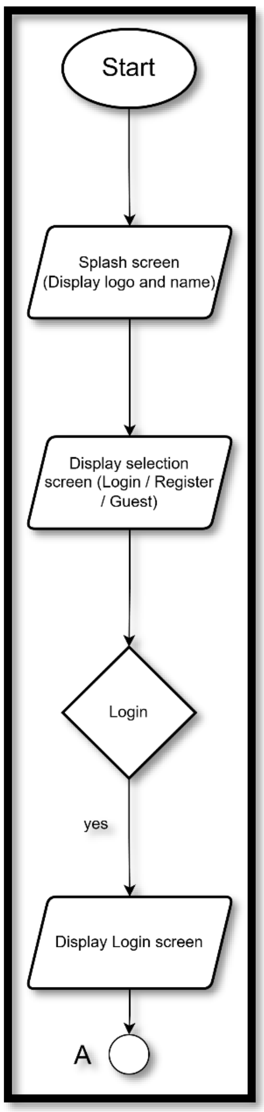

Start
Splash screen (Display logo and name)
Display selection screen (Login / Register / Guest)
Login
yes
Display Login screen
A

Figure 1 Login option

Start

- **Function:** Marks the **starting point** of the application process.

Splash screen (Display logo and name)

- **Function:** The first screen displayed when the application launches, used to showcase the **brand logo and name** and often serves as a buffer while core content loads.

Display selection screen (Login / Register / Guest)

- **Function:** Presents the user with the **main entry options**:
    - **Login:** To access an existing account.

  - **Register:** To create a new account.
  - **Guest:** To continue using the app anonymously.

Login

- **Function:** This is a **decision point** that checks if the user has selected the "Login" option. The flow path branches based on this selection.

Display Login screen

- **Function:** If Login was selected, the system **displays the login form** screen, awaiting user input of credentials (username and password).

Bottom Connector/End Point

- **Function:** Represents a **connector or temporary termination point** in the flow. The process would typically continue from here based on the result of the user's action (e.g., successful or failed login).

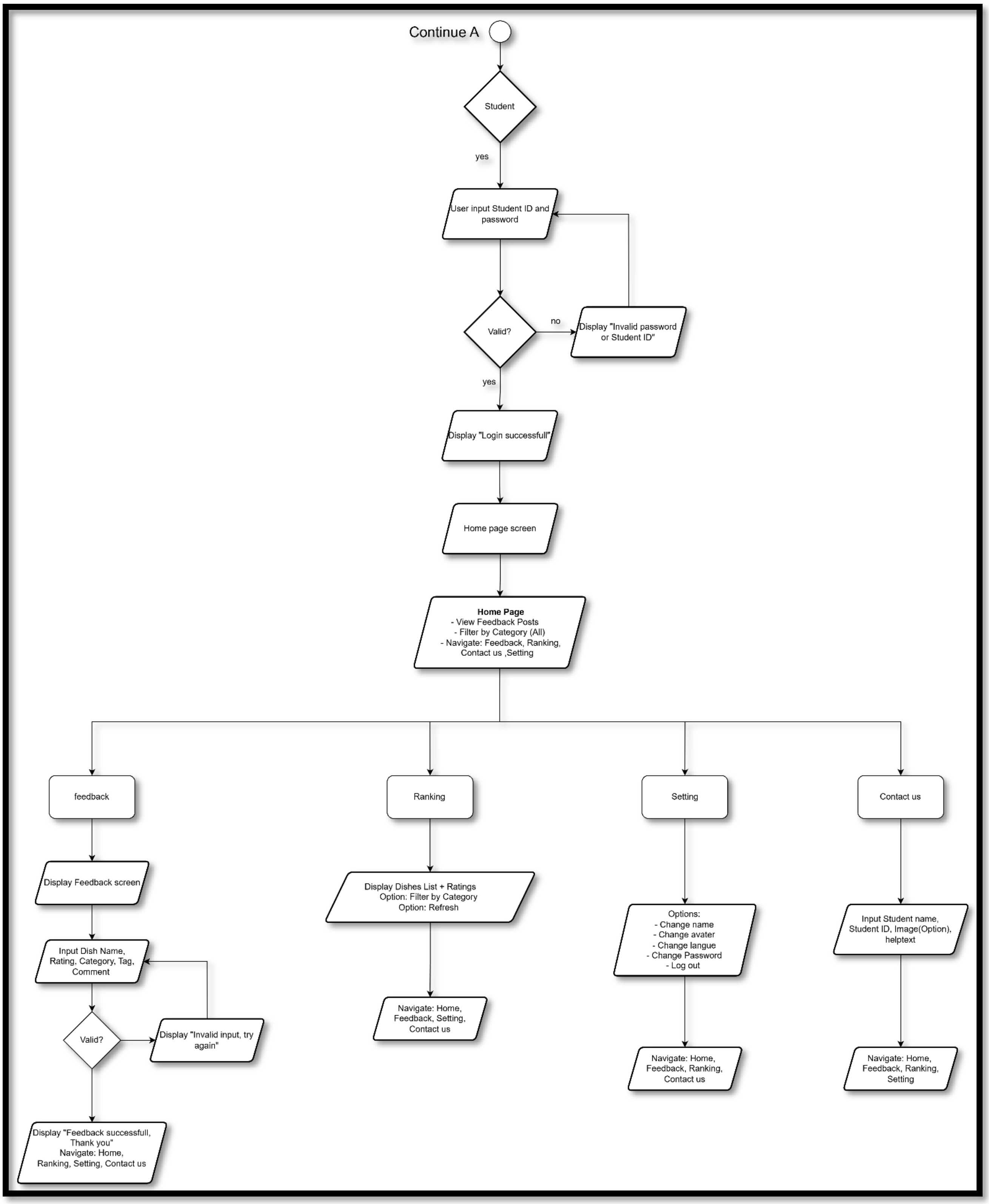

Continue A
Student
yes
User input Student ID and password
Valid?
no
Display "Invalid password or Student ID"
yes
Display "Login successfull"
Home page screen
Home Page
- View Feedback Posts
- Filter by Category (All)
- Navigate: Feedback, Ranking, Contact us ,Setting
feedback
Ranking
Setting
Contact us
Display Feedback screen
Input Dish Name, Rating, Category, Tag, Comment
Valid?
Display "Invalid input, try again"
Display "Feedback successfull, Thank you" Navigate: Home, Ranking, Setting, Contact us
Display Dishes List + Ratings Option: Filter by Category Option: Refresh
Navigate: Home, Feedback, Setting, Contact us
Options: - Change name - Change avater - Change langue - Change Password - Log out
Navigate: Home, Feedback, Ranking, Contact us
Input Student name, Student ID, Image(Option), helptext
Navigate: Home, Feedback, Ranking, Setting

Figure 2 User Type of Student

Login and Validation

- **Student:** Confirmation that the user identity is a **Student**.
- **User input Student ID and password:** The student enters their login credentials.
- **Valid? (Validation Check):** The system verifies if the credentials are a match.
    - **No:** Display **"Invalid password or Student ID"**.
    - **Yes:** Display **"Login successful"**.
- **Home page screen:** Navigate to the application's main home screen.

Home

- **Home Page:** Provides core functionality and navigation options:
    - View **Feedback details**.
    - **Filter** the food list by **Category**.
- Provides **Navigation:** Feedback, Ranking, Setting, and Contact us.

Feedback

- **Feedback:** User selects to submit feedback.
- **Display Feedback screen:** The feedback input interface is presented.
- **Input Dish Name, Feedback and Comment:** The user enters the dish name, feedback type, and comments.
- **Valid? (Validation Check):** Checks if the input content got bad word or not
    - **Yes:** Display **"Invalid Input, try again"**.
    - **No:** Display **"Feedback successful"**.
- Provides **Navigation:** Home, Ranking, Setting, and Contact us.

Ranking

- **Ranking:** User selects to view rankings.
- **Display Dishes List + Ratings... (Ranking Screen):**
    - Displays the **Dish List and their Ratings**.
    - Provides options to **Filter by Category** and **Refresh** the list.
- Provides **Navigation:** Home, Feedback, Setting, and Contact us.

Setting

- **Setting:** User selects to enter settings.

- **Options (Settings Options):** Provides the following functions:
    - Change name.
    - Change avatar.
    - Change password.
    - Change language
    - Log out.
- Provides **Navigation:** Home, Feedback, Ranking, and Contact us.

Contact us

- **Contact us:** User selects to request technical assistance
- **Display Contact us screen:** The screen will display:
    - **Input Student name**
    - **Input Student ID**
    - **Upload the image (optionor)**
    - **Input the issue**
- Provides **Navigation:** Home, Feedback, Ranking, and Contact us.

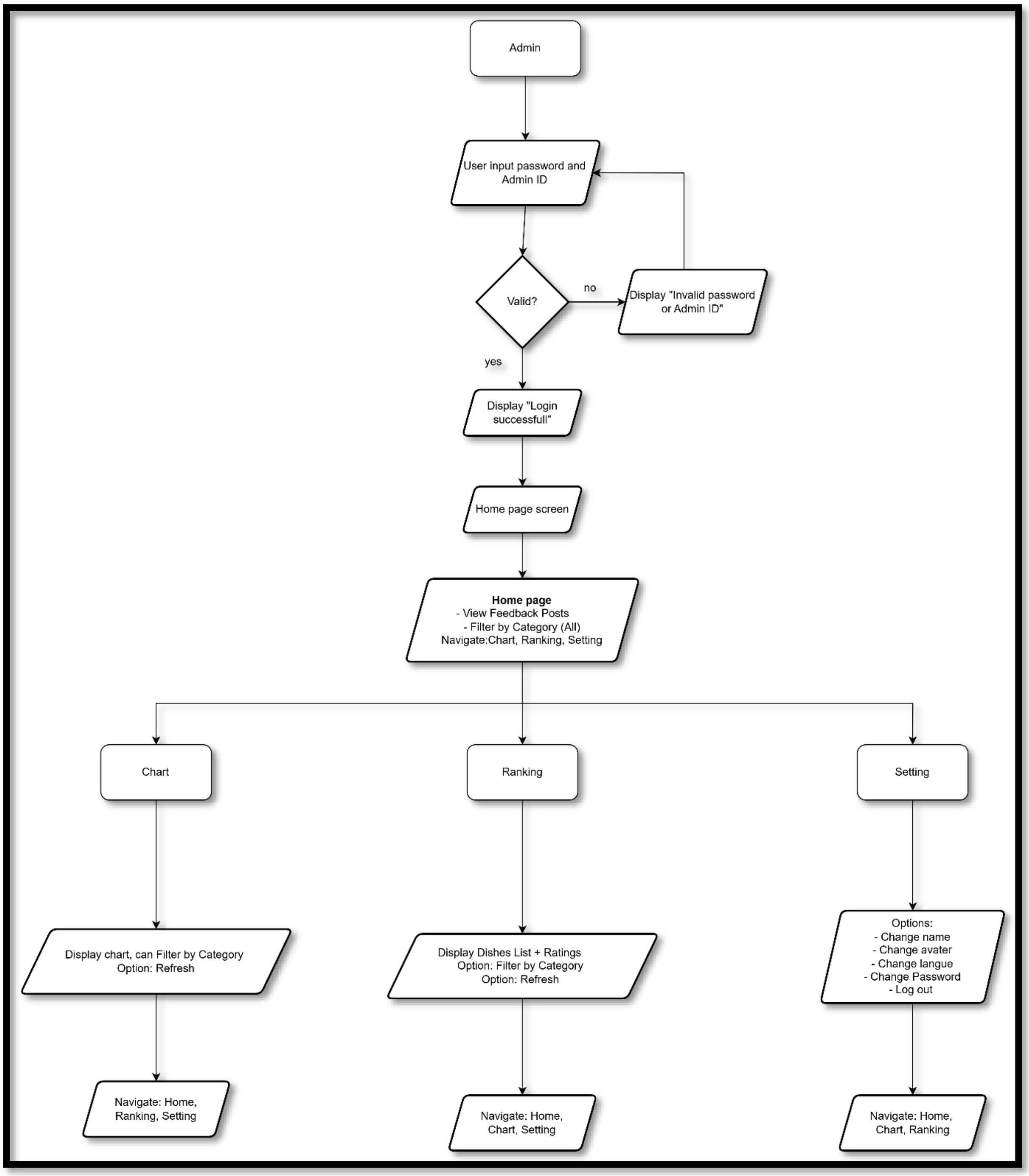

Admin
User input password and Admin ID
Valid?
no
Display "Invalid password or Admin ID"
yes
Display "Login successfull"
Home page screen
Home page
- View Feedback Posts
- Filter by Category (All)
Navigate:Chart, Ranking, Setting
Chart
Ranking
Setting
Display chart, can Filter by Category
Option: Refresh
Display Dishes List + Ratings
Option: Filter by Category
Option: Refresh
Options:
- Change name
- Change avater
- Change langue
- Change Password
- Log out
Navigate: Home, Ranking, Setting
Navigate: Home, Chart, Setting
Navigate: Home, Chart, Ranking

Figure 3 User Type of Admin

Login and Validation

- **Admin:** Confirmation that the user identity is an **Administrator**.
- **User input password and Admin ID:** The administrator enters their login credentials.
- **Valid? (Validation Check):** The system verifies if the credentials are a match.
    - **No:** Display **"Invalid password or Admin ID"**.
    - **Yes:** Display **"Login successful"**.

- **Home page screen:** Navigate to the application's main home screen.

Home Page

- **View all Feedback**
- **Can Filter by Category**
- **Provide Navigation**: Chart, Ranking, and Setting.

Chart

- **Display chart, can Filter by Category... (Display Chart Screen):**
    - Displays the **Chart** (for data analysis or overview).
    - Provides an option to **Filter** the data by **Category**.
    - Provides a **Refresh** option.
- **Provide Navigation**: Home, Ranking, and Setting.

Ranking

- **Display Dishes List + Ratings... (Ranking Screen):**
    - Displays the **Dish List and their Ratings**.
    - Provides an option to **Filter by Category**.
    - Provides a **Refresh** option.
- **Provide Navigation**: Home, Chart and Setting.

Setting

- **Setting:** The administrator selects to enter settings.
- **Options (Settings Options):** Provides the following functions:
    - Change name.
    - Change avatar.
    - Change language.
    - Change password.
    - Log out.
- **Provide Navigation**: Home, Chart, and Ranking.

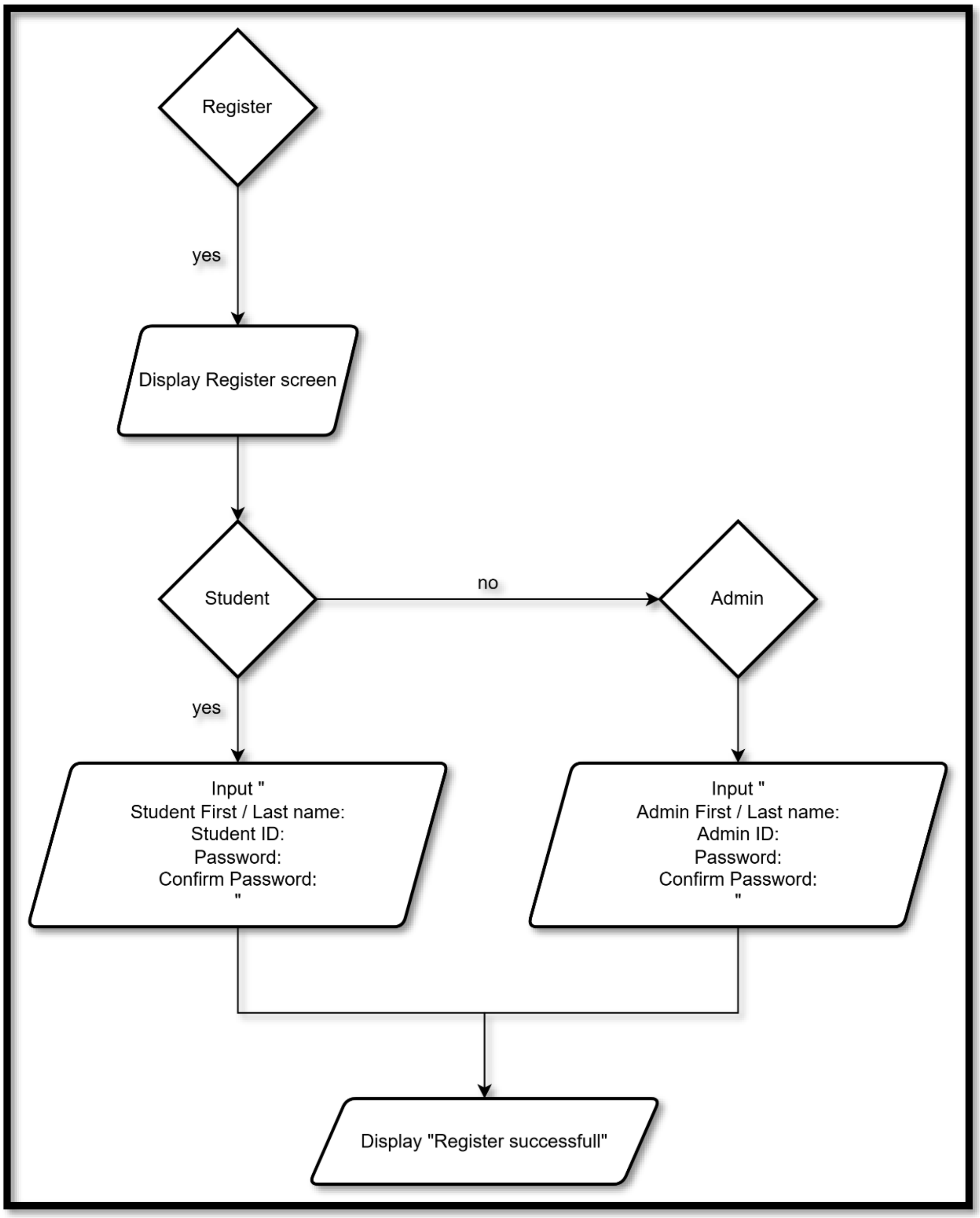


Figure 4 Register option

Display Register screen:

- **Function: Displays the registration interface**, prompting the user to begin entering their details.

User Type Selection and Input

- **Student / Admin:**
  - **Function:** These are **decision points** (diamonds) representing the user's **selection** or the system's **determination** of the user type (Student or Admin). The flow branches into two parallel paths here.

Student Registration

- **Input "Student First / Last name..." (Student Information Input):**
  - **Function:** Prompts the Student user to input the required registration details:
    - **First / Last name**
    - **Student ID**
    - **Password**
    - **Confirm Password**

Admin Registration

- **Input "Admin First / Last name..." (Admin Information Input):**
  - **Function:** Prompts the Admin user to input the required registration details:
    - **First / Last name**
    - **Admin ID**
    - **Password**
    - **Confirm Password**

Process Outcome

- **Display "Register successful"**
  - **Function:** After successfully entering the required information for either user type (assuming validation, which is not explicitly shown, passes), the flow converges and **displays a successful registration message**.

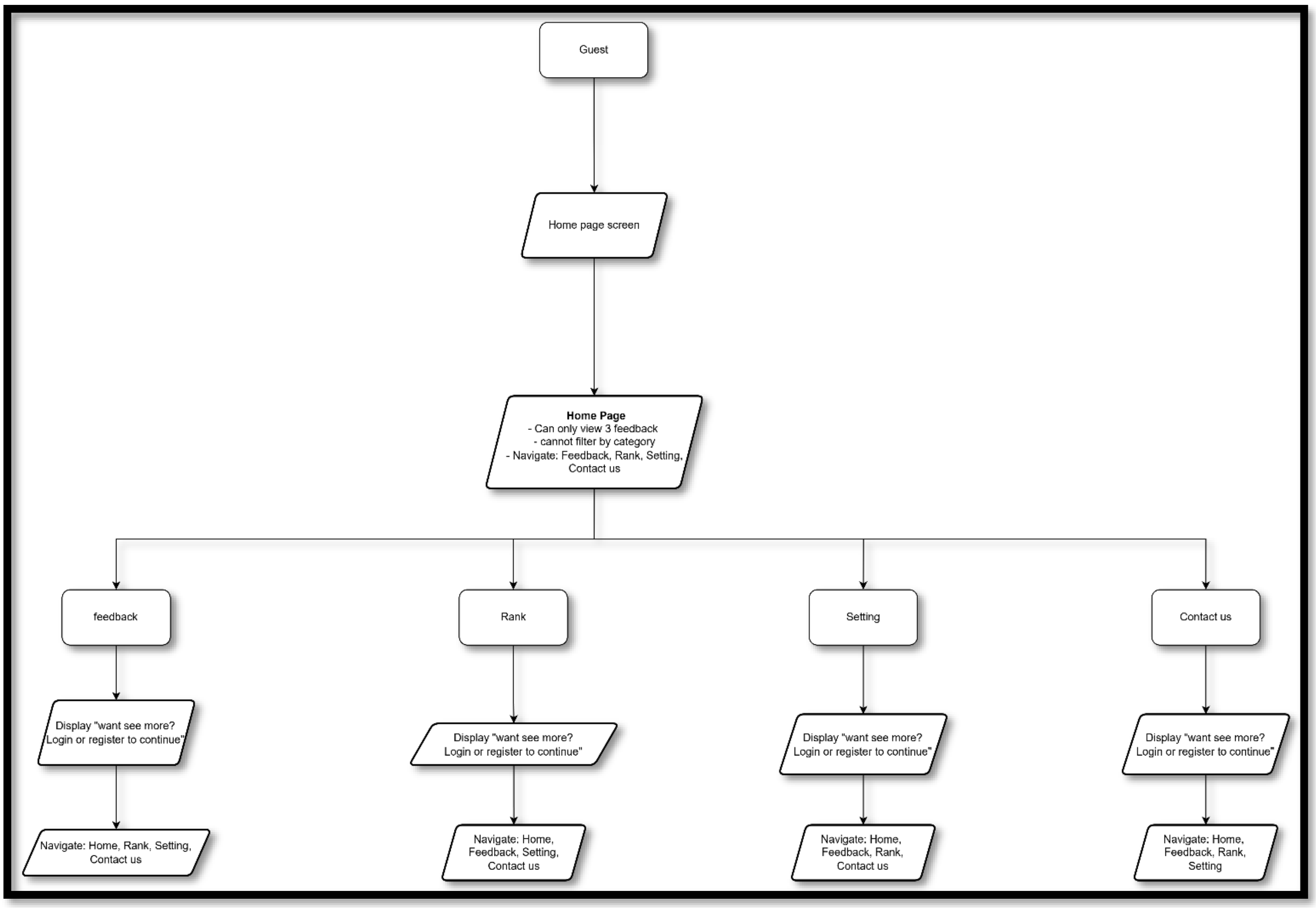

Guest
Home page screen
Home Page
- Can only view 3 feedback
- cannot filter by category
- Navigate: Feedback, Rank, Setting, Contact us
feedback
Rank
Setting
Contact us
Display "want see more? Login or register to continue"
Display "want see more? Login or register to continue"
Display "want see more? Login or register to continue"
Display "want see more? Login or register to continue"
Navigate: Home, Rank, Setting, Contact us
Navigate: Home, Feedback, Setting, Contact us
Navigate: Home, Feedback, Rank, Contact us
Navigate: Home, Feedback, Rank, Setting

Figure 5 Guest option

Home Page Features and Limitations

- Core Features and Limitations:
- Only 3 feedback entries can be viewed.
- Cannot filter by category.
- **Navigation Options:** Guests can navigate to the following four main sections: Feedback, Rank, Settings, Contact Us.

Restrictions for the Four Main Navigation Modules (Feedback, Rank, Settings, Contact Us)

Upon entering these four modules, guests encounter the same prompt and face functional limitations:

Feedback:

- Prompt: Displays “Want to see more?”
- Restriction: Prompts “Login or register to continue.”
- Navigation: Allows return to main navigation menu: Home, Feedback, Rank, Settings, Contact Us.

Rank (Rankings):

- Prompt: Display “Want to see more?”
- Restriction: Prompt “Login or register to continue.”
- Navigation: Allows return to main navigation menu: Home, Feedback, Rank, Settings, Contact Us.

Settings:

- Prompt: Display “Want to see more?”
- Restriction: Prompt “Login or register to continue.”
- Navigation: Allows return to main navigation menu: Home, Feedback, Rank, Settings, Contact Us.

Contact Us:

- Prompt: Display “Want to see more?”
- Restriction: Prompt “Login or register to continue.”
- Navigation: Allows return to main navigation menu: Home, Feedback, Rank, Settings, Contact Us.

PART B: UML Modelling and Analysis

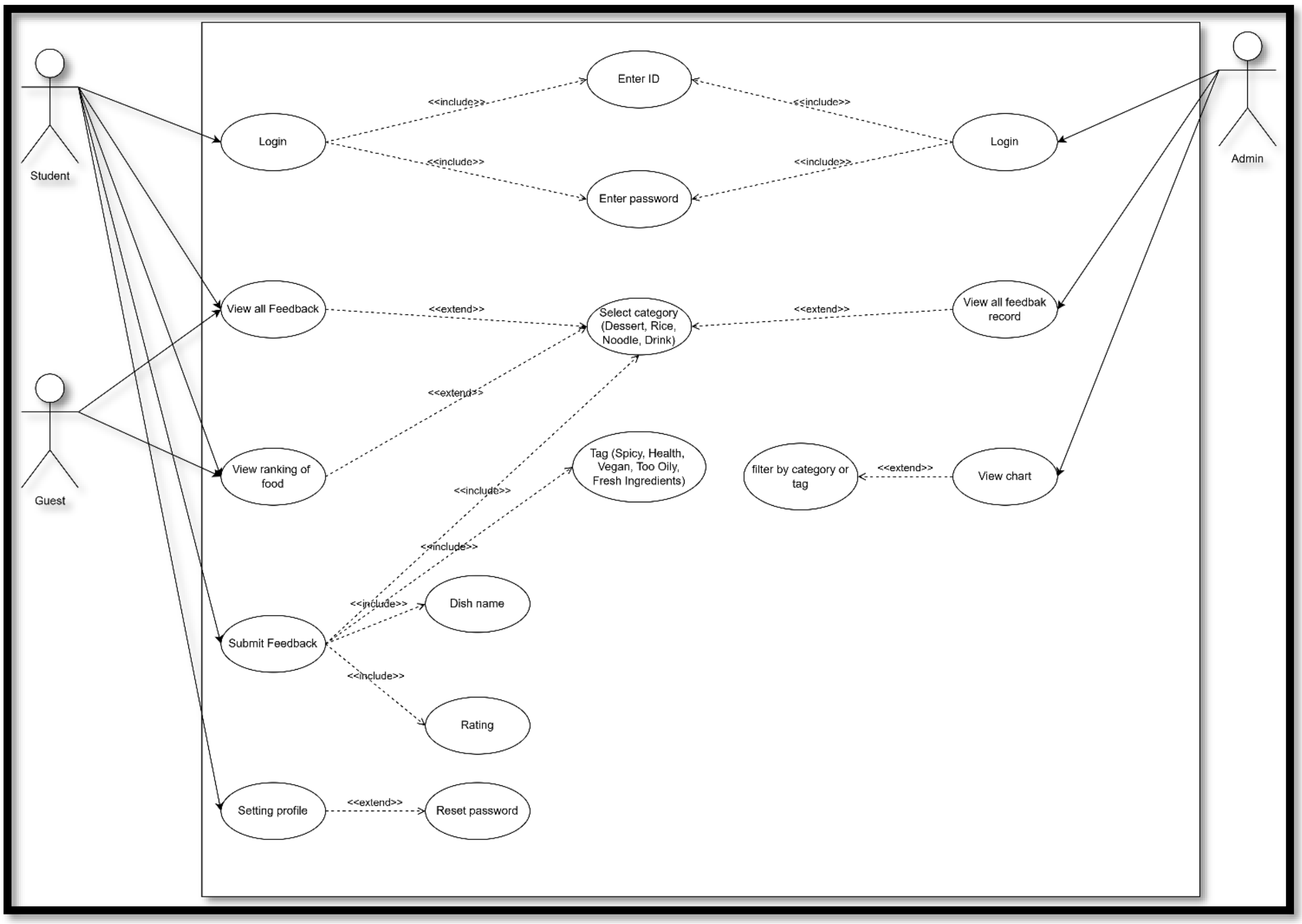


Figure 6 Use Case Diagram of CCF

Justification

The Use Case Diagram identifies all external interactions between users (Student, Guest, Admin) and the system. It helps establish system boundaries and ensures every feature required by the assignment is formally captured before moving into design.

- **Student Actor:**
  Students perform the majority of system actions, including registering an account, logging in, submitting feedback, selecting tags, viewing insights, and accessing announcements. These use cases form the core functionality and validate that the system supports all essential interactions.

- **Guest Actor:**
  Guests are allowed to explore the system but are restricted from submitting feedback

or accessing admin-level analytics. This actor ensures the system supports open exploration without compromising data integrity.

- **Admin Actor:**
  Admins manage announcements, review feedback, and access analytical dashboards. Their use cases justify the need for additional system control and backend data management.

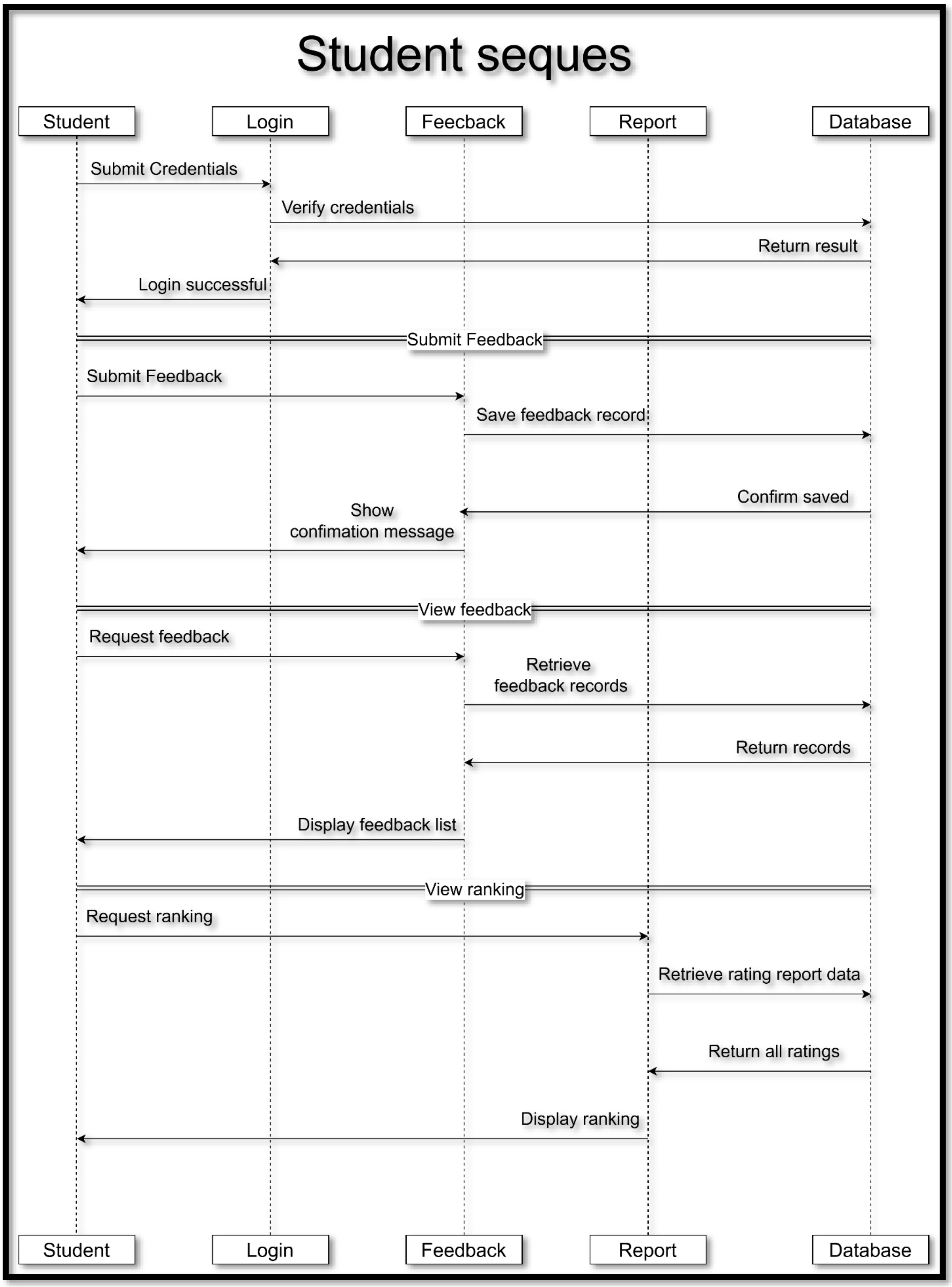


Figure 7 Student Sequence diagram

Login Process

- **Student** sends **Submit Credentials** (ID and Password) to **Login**.
- **Login** sends **Verify credentials** to **Database**.
- **Database** returns **Return result** to **Login**.
- **Login** notifies **Student** with **Login successful**.

Submit Feedback Process

- **Student** sends **Submit Feedback** (including content details) to **Feedback**.
- **Feedback** sends **Save feedback record** to **Database**.
- **Database** returns **Confirm saved** to **Feedback**.
- **Feedback** shows **Show confirmation message** to **Student**.

View Feedback Process

- **Student** sends **Request feedback** to **Feedback**.
- **Feedback** sends **Retrieve feedback records** to **Database**.
- **Database** returns **Return records** (the list of feedback).
- **Feedback** displays **Display feedback list** to **Student**.

View Ranking Process

- **Student** sends **Request ranking** to **Report**.
- **Report** sends **Retrieve rating report data** to **Database**.
- **Database** returns **Return all ratings** (raw data for calculation).
- **Report** calculates the ranking and displays **Display ranking** to **Student**.

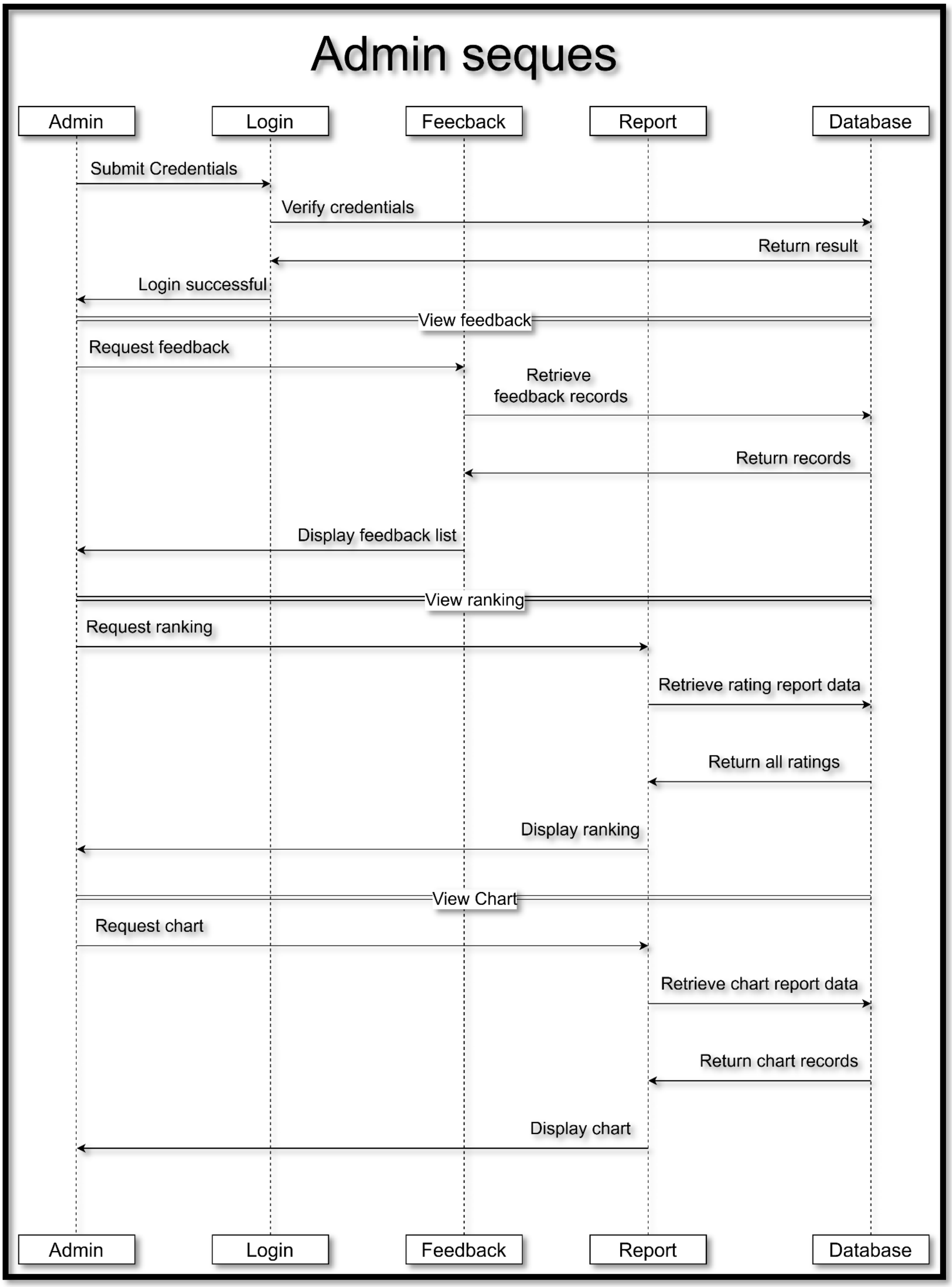


Figure 8 Admin Sequence diagram

Participants (Lifelines)

- **Admin:** The high-privilege user role initiating all management operations.
- **Login:** The module responsible for user authentication.
- **Feedback:** The module responsible for handling feedback data requests.
- **Report:** The module responsible for handling ranking and advanced chart data requests.
- **Database:** Stores all system records and data.

Scenario 1: Login Process

- Admin Sends **Submit Credentials** (ID and Password) to **Login**.
- Login Sends **Verify credentials** to **Database**.
- Database Returns **Return result** to Login.
- Login Notifies Admin with **Login successful**.

Scenario 2: View Feedback Process

- Admin Sends **Request feedback** to **Feedback**.
- Feedback Sends **Retrieve feedback records** to **Database**.
- Database Returns **Return records** (the complete list of feedback).
- Feedback Displays **Display feedback list** to Admin.
- **Permission Note:** Admin retrieves **all** records, contrasting with the Guest's "Return 3 records."

Scenario 3: View Ranking Process

- Admin Sends **Request ranking** to **Report**.
- Report Sends **Retrieve rating report data** to **Database**.
- Database Returns **Return all ratings** (raw data for ranking calculation).
- Report Displays **Display ranking** to Admin.
- **Permission Note:** Admin gets the **full** dataset for complete ranking reports without any registration prompt.

Scenario 4: View Chart Process

- Admin Sends **Request chart** to **Report**.
- Report Sends **Retrieve chart report data** to **Database**.
- Database Returns **Return chart records** (specific data needed for visualization).
- Report Displays **Display chart** to Admin.
- **Exclusive Feature:** "Request chart" is an advanced analytical feature unique to the Admin role, enabling data visualization and decision support.

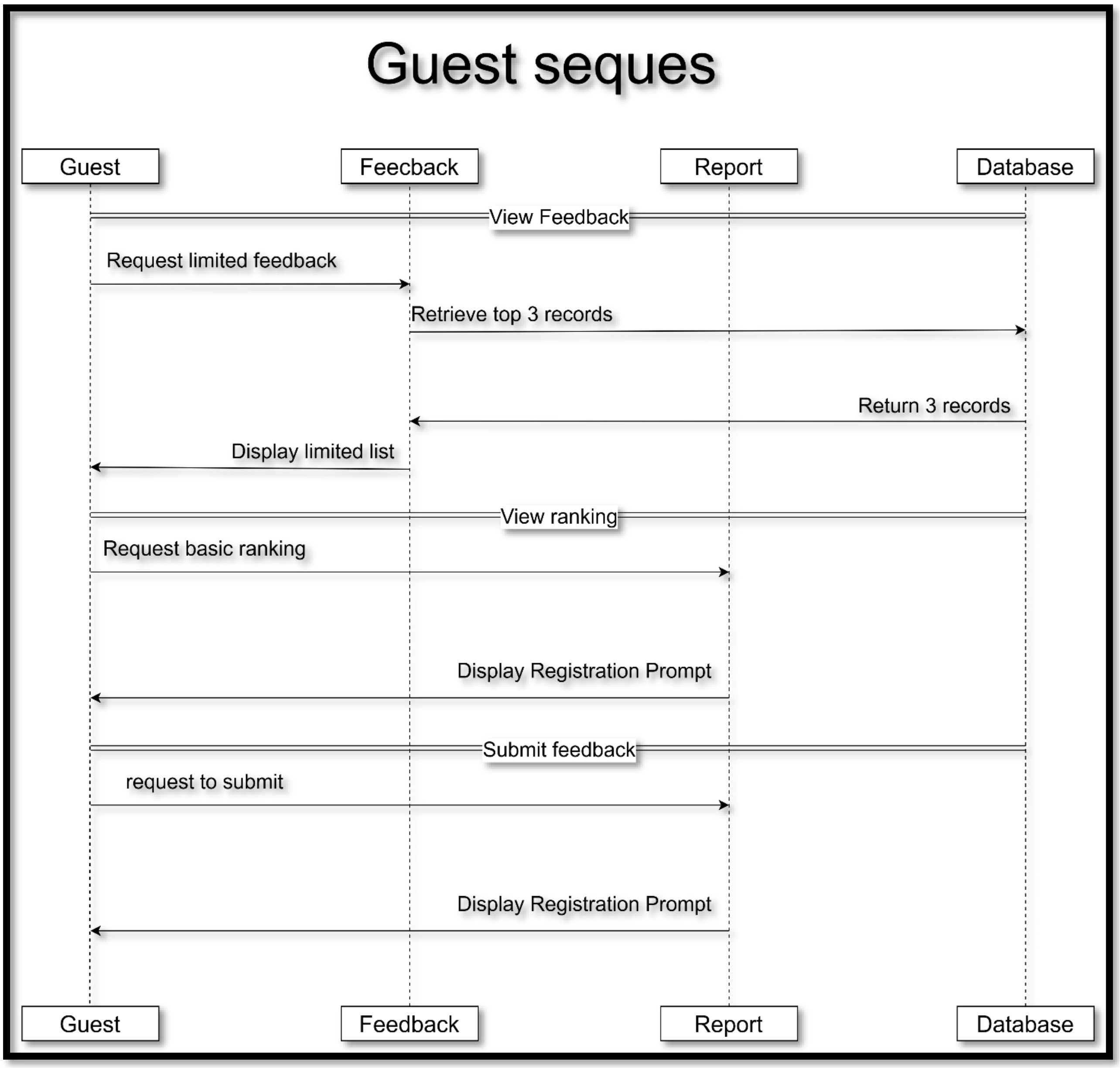


Figure 9 Guest Sequence diagram

Participants (Lifelines)

- **Guest:** The unregistered user initiating all operation requests.
- **Feedback:** The module responsible for handling feedback requests and responses.
- **Report:** The module handling the business logic for ranking and feedback submission.
- **Database:** Stores and returns data records.

Scenario 1: Requesting Limited Feedback

- **Goal:** To allow Guest users to view a small sample of feedback data.
- **Process:**
  - Guest sends a Request limited feedback.
  - The Feedback module queries the Database, explicitly requesting to **Retrieve top 3 records**.
  - The Database returns these **3 records**.
  - The Feedback module responds to the Guest with a **Display limited list**.
- **Conclusion:** Guests **are permitted** limited read-only operations.

Scenario 2: Requesting Ranking and Submission

- **Goal:** To restrict Guest users from performing sensitive view and write operations.
- **Requesting Ranking (Request basic ranking):**
  - The Guest sends a request for ranking.
  - The Report module processes the request, then responds by sending **Display Registration Prompt**.
- **Requesting Submission (request to submit):**
  - The Guest attempts to **request to submit** (feedback).
  - The Report module processes the submission attempt,and again returns **Display Registration Prompt**.
- **Conclusion:**
  - Guests **cannot** view full rankings or submit data.
  - The system **enforces user registration/authentication** by returning the **Registration Prompt**.

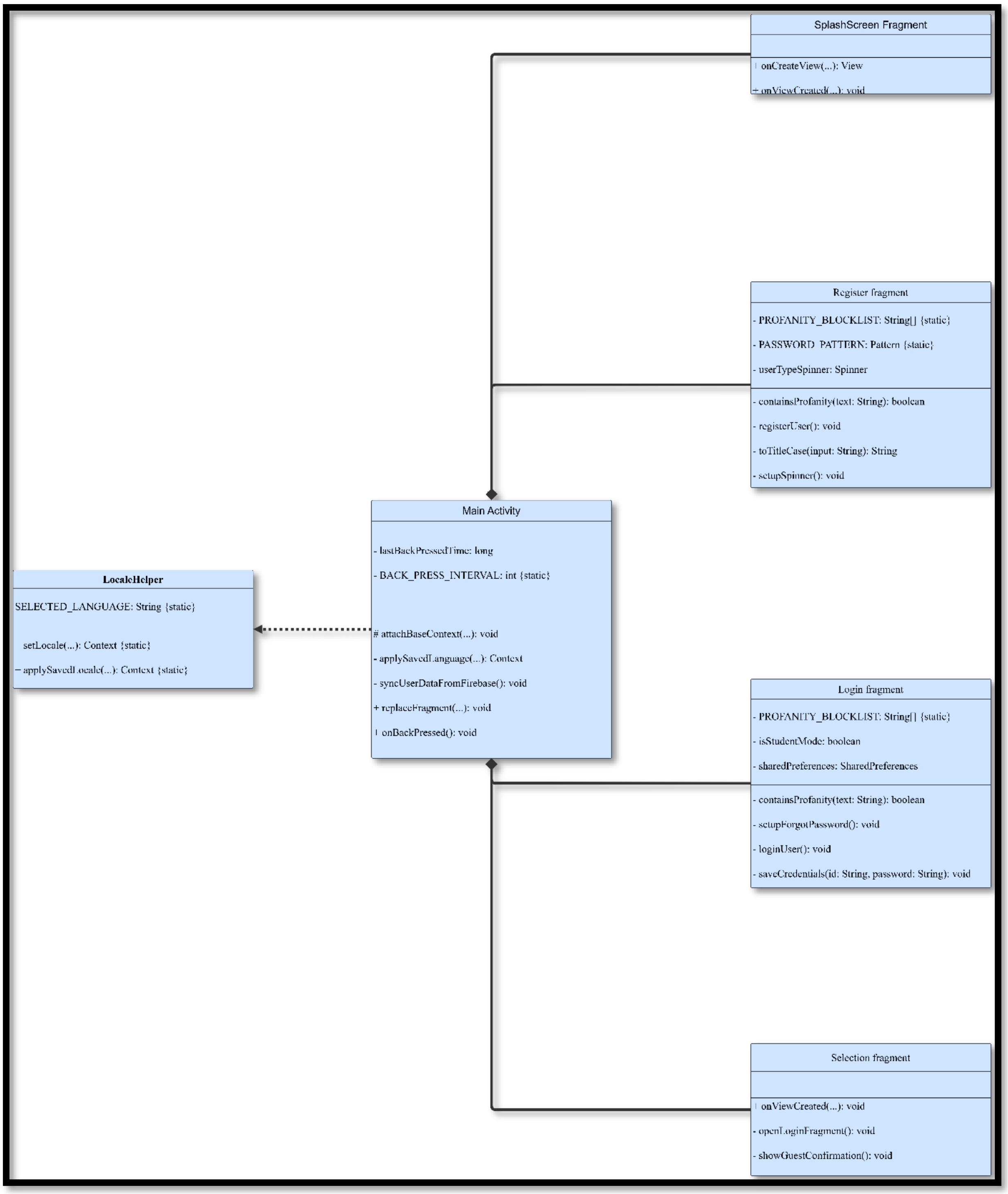

SplashScreen Fragment
+ onCreateView(...): View
+ onViewCreated(...): void
Register fragment
- PROFANITY_BLOCKLIST: String[] {static}
- PASSWORD_PATTERN: Pattern {static}
- userTypeSpinner: Spinner
- containsProfanity(text: String): boolean
- registerUser(): void
- toTitleCase(input: String): String
- setupSpinner(): void
Main Activity
- lastBackPressedTime: long
- BACK_PRESS_INTERVAL: int {static}
# attachBaseContext(...): void
- applySavedLanguage(...): Context
- syncUserDataFromFirebase(): void
+ replaceFragment(...): void
+ onBackPressed(): void
LocaleHelper
SELECTED_LANGUAGE: String {static}
setLocale(...): Context {static}
- applySavedLocale(...): Context {static}
Login fragment
- PROFANITY_BLOCKLIST: String[] {static}
- isStudentMode: boolean
- sharedPreferences: SharedPreferences
- containsProfanity(text: String): boolean
- setupForgotPassword(): void
- loginUser(): void
- saveCredentials(id: String, password: String): void
Selection fragment
+ onViewCreated(...): void
- openLoginFragment(): void
- showGuestConfirmation(): void

Figure 10 UML class diagram of Main Activity

Central Component: Main Activity (The Coordinator)

- **Role:** Main Activity is the application's **entry point**. It coordinates the entire startup flow, handles global configuration (language and data sync), and acts as the host for all initial screens.

Fragment Startup Flow

- **SplashScreenFragment (Splash Screen):**
    - Once loaded by MainActivity, it is the first Fragment displayed.
    - Its - onViewCreated(...) method contains the timing logic responsible for **transferring control** to the Selection Fragment after a brief delay.

Authentication Fragment Responsibilities

- **Selection Fragment (Selection Screen):**
    - **Role:** Provides the interface for the user to choose between logging in, registering, or continuing as a guest.
    - It contains methods like - openLoginFragment() and - openRegisterFragment() which trigger Fragment replacement operations handled by MainActivity.
- **Login Fragment (Login):**
    - **Core Role:** Handles user authentication.
    - **Internal Logic:** It holds security elements like - PROFANITY_BLOCKLIST and contains the - loginUser() method to validate credentials (ID and password) submitted by the user.
    - It holds a SharedPreferences reference, used to remember the user's login state via - saveCredentials().
- **Register Fragment (Registration):**
    - **Core Role:** Accepts user input, performs strict validation, and creates new accounts.
    - **Internal Logic:** It contains security elements like - PROFANITY_BLOCKLIST and - PASSWORD_PATTERN, ensuring the correctness and security of registration information.
    - The - registerUser() method is responsible for uploading the new user data to the Repository (an external dependency) after validation passes.

Auxiliary Tool

- **LocaleHelper (Language Utility):**
    - This is a pure static utility class, providing core methods like - setLocale(...) and - applySavedLocale(...).
    - Its sole responsibility is to manage, store, and apply the application's language configuration, and it is depended upon and called by Activities like MainActivity during startup.

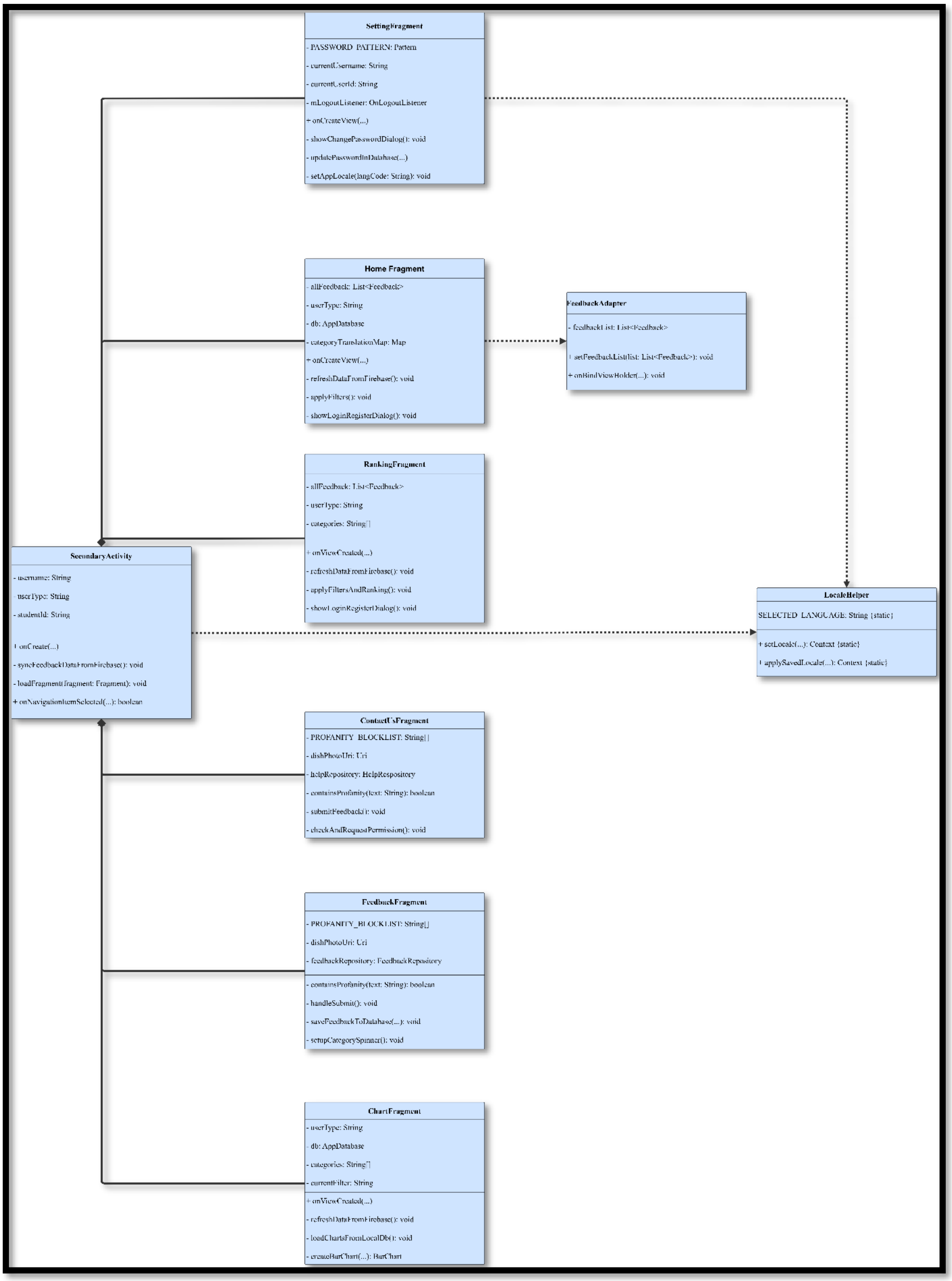

SettingFragment
- PASSWORD_PATTERN: Pattern
- currentUsername: String
- currentUserId: String
- mLogoutListener: OnLogoutListener
+ onCreateView(...)
- showChangePasswordDialog(): void
- updatePasswordInDatabase(...)
- setAppLocale(langCode: String): void
Home Fragment
- allFeedback: List<Feedback>
- userType: String
- db: AppDatabase
- categoryTranslationMap: Map
+ onCreateView(...)
- refreshDataFromFirebase(): void
- applyFilters(): void
- showLoginRegisterDialog(): void
FeedbackAdapter
- feedbackList: List<Feedback>
+ setFeedbackList(list: List<Feedback>): void
+ onBindViewHolder(...): void
RankingFragment
- allFeedback: List<Feedback>
- userType: String
- categories: String[]
+ onViewCreated(...)
- refreshDataFromFirebase(): void
- applyFiltersAndRanking(): void
- showLoginRegisterDialog(): void
SecondaryActivity
- username: String
- userType: String
- studentId: String
+ onCreate(...)
- syncFeedbackDataFromFirebase(): void
- loadFragment(fragment: Fragment): void
+ onNavigationItemSelected(...): boolean
LocaleHelper
SELECTED_LANGUAGE: String {static}
+ setLocale(...): Context {static}
+ applySavedLocale(...): Context {static}
ContactUsFragment
- PROFANITY_BLOCKLIST: String[]
- dishPhotoUri: Uri
- helpRepository: HelpRespository
- containsProfanity(text: String): boolean
- submitFeedback(): void
- checkAndRequestPermission(): void
FeedbackFragment
- PROFANITY_BLOCKLIST: String[]
- dishPhotoUri: Uri
- feedbackRepository: FeedbackRepository
- containsProfanity(text: String): boolean
- handleSubmit(): void
- saveFeedbackToDatabase(...): void
- setupCategorySpinner(): void
ChartFragment
- userType: String
- db: AppDatabase
- categories: String[]
- currentFilter: String
+ onViewCreated(...)
- refreshDataFromFirebase(): void
- loadChartsFromLocalDb(): void
- createBarChart(...): BarChart

Figure 11 UML class diagram of Secondary Activity

Central Component: Secondary Activity (The Core Host)

- **Core Role:** SecondaryActivity serves as the central controller for the main application interface. It is responsible for maintaining the user session state (username, userType) and providing a container for all functional modules to operate within.

UI Functionality and Data Presentation Collaboration

- **HomeFragment (Home Screen) and FeedbackAdapter (Adapter):**
    - **Collaboration Intent:** HomeFragment depends on FeedbackAdapter to fulfill its core responsibility—taking the feedback data retrieved from the database and mapping it to the list views displayed on the screen

Functional Module Independence

The diagram shows that all other functional Fragments are independent modules, connected directly to SecondaryActivity but generally not directly connected to each other

- **SettingFragment:** Focuses on settings and account management, including language change (setAppLocale(...)) and password logic.
- **RankingFragment:** Focuses on data filtering and ranking calculation (applyFiltersAndRanking).
- **FeedbackFragment / ContactUsFragment:** Focuses on data submission and security checks (containsProfanity, handleSubmit).

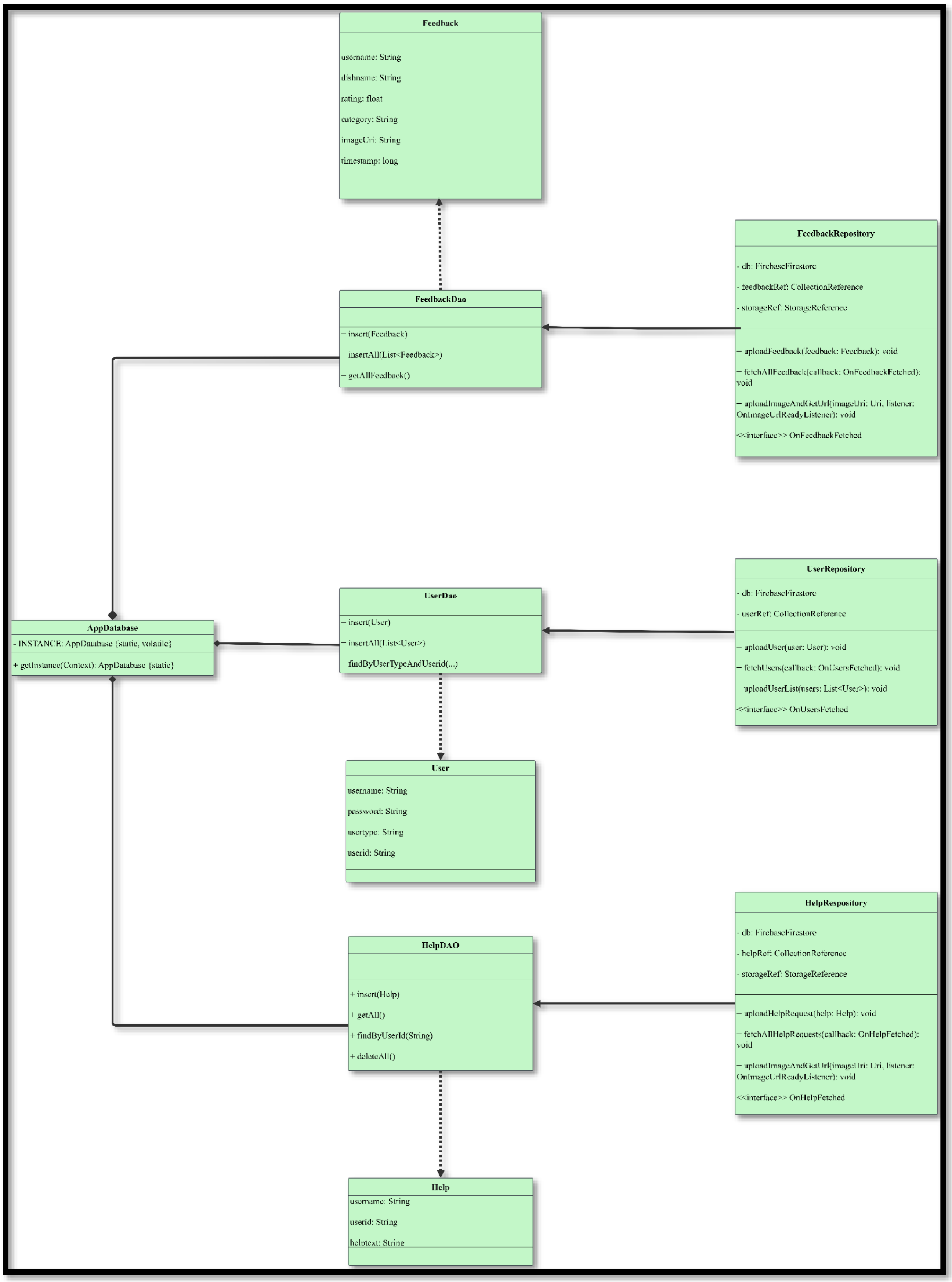

Feedback
username: String
dishname: String
rating: float
category: String
imageUri: String
timestamp: long
FeedbackRepository
- db: FirebaseFirestore
- feedbackRef: CollectionReference
- storageRef: StorageReference
– uploadFeedback(feedback: Feedback): void
– fetchAllFeedback(callback: OnFeedbackFetched): void
– uploadImageAndGetUrl(imageUri: Uri, listener: OnImageUrlReadyListener): void
<<interface>> OnFeedbackFetched
FeedbackDao
– insert(Feedback)
insertAll(List<Feedback>)
– getAllFeedback()
UserRepository
- db: FirebaseFirestore
- userRef: CollectionReference
– uploadUser(user: User): void
– fetchUsers(callback: OnUsersFetched): void
uploadUserList(users: List<User>): void
<<interface>> OnUsersFetched
UserDao
– insert(User)
– insertAll(List<User>)
findByUserTypeAndUserid(...)
AppDatabase
- INSTANCE: AppDatabase {static, volatile}
+ getInstance(Context): AppDatabase {static}
User
username: String
password: String
usertype: String
userid: String
HelpRespository
- db: FirebaseFirestore
- helpRef: CollectionReference
- storageRef: StorageReference
– uploadHelpRequest(help: Help): void
– fetchAllHelpRequests(callback: OnHelpFetched): void
– uploadImageAndGetUrl(imageUri: Uri, listener: OnImageUrlReadyListener): void
<<interface>> OnHelpFetched
HelpDAO
+ insert(Help)
+ getAll()
+ findByUserId(String)
+ deleteAll()
Help
username: String
userid: String
helptext: String

Figure 12 UML class diagram of database

Entity Classes (User, Feedback, Help)

- **Core Role:** Data Structure Carriers. They define the fields and types for all persistent data within the application (e.g., username, rating, helptext).

Local Persistence Layer (DAO / Room Database)

This vertical chain is responsible for local data storage and access.

- **AppDatabase (Database Instance):**
  - Serves as the local database's singleton factory. It provides abstract access to the three DAO interfaces (UserDao, FeedbackDao, HelpDAO).
- **DAO Interfaces (UserDao, FeedbackDao, HelpDAO):**
  - **Core Role:** Defines the CRUD (Create, Read, Update, Delete) operations for the local database tables.

PART C: Object-Oriented Development Class Design and Implementation

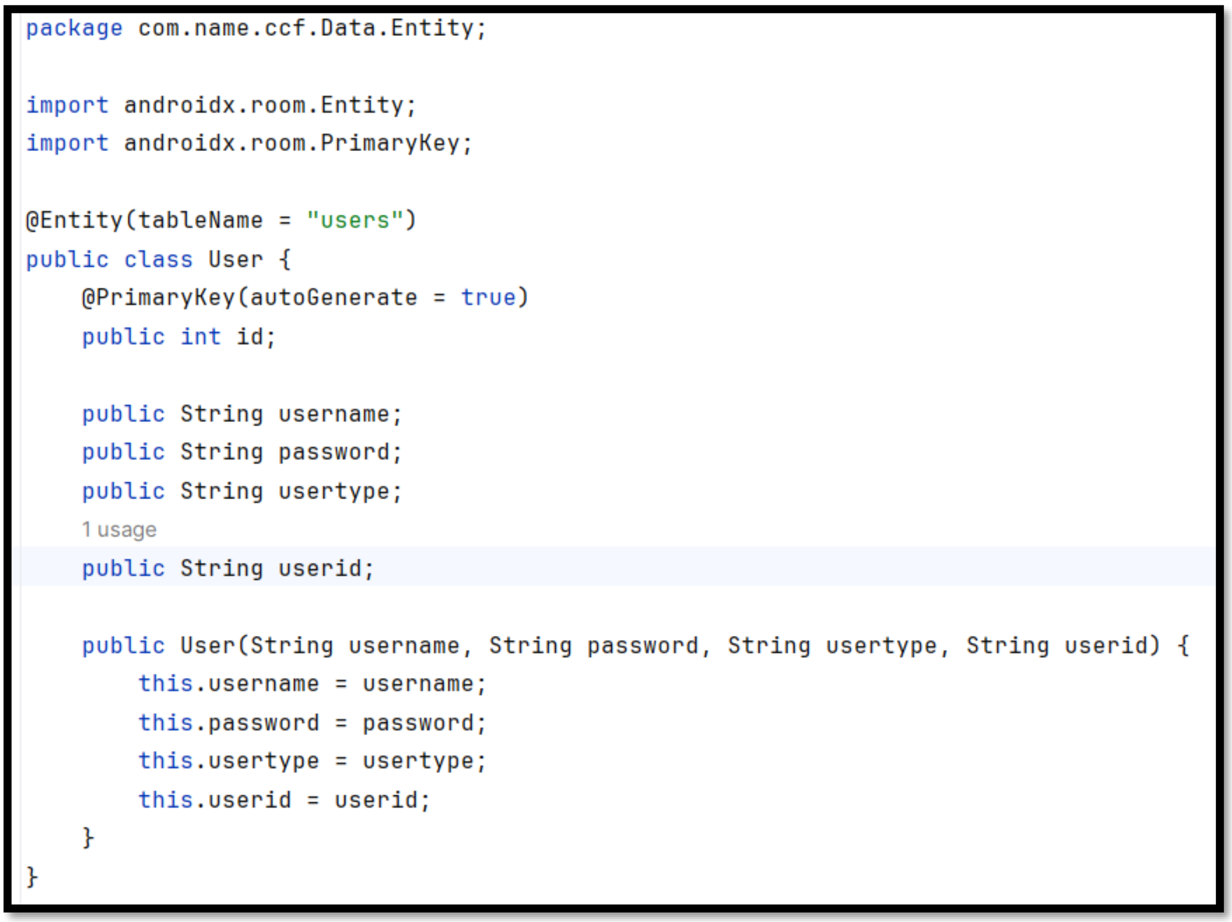
In the object-oriented design of this application, the Feedback and User classes are established as the core entities of the entire system. The Feedback class is mainly responsible for managing and storing all structured user feedback data (such as ratings, tags, and comments), while the User class is responsible for user authentication and permission management to ensure system security.

1. Applying OOP Concepts: Encapsulation

To protect core data (such as user passwords) from arbitrary modification by external modules, we strictly applied the principle of encapsulation.

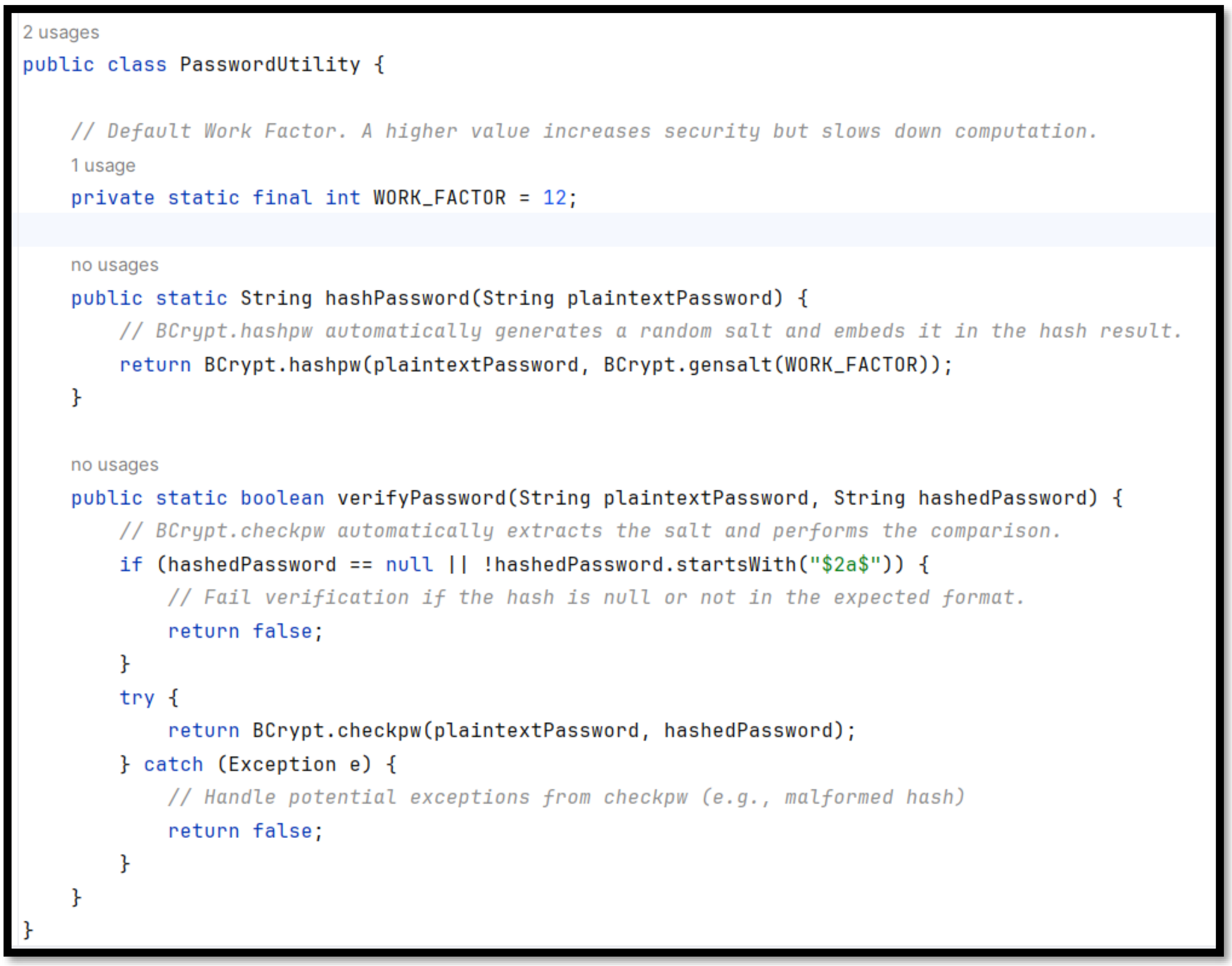
Implement:

The design of the User entity class demonstrates the definition of the data structure, and its properties are initialized through the constructor



```java
package com.name.ccf.Data.Entity;

import androidx.room.Entity;
import androidx.room.PrimaryKey;

@Entity(tableName = "users")
public class User {
    @PrimaryKey(autoGenerate = true)
    public int id;

    public String username;
    public String password;
    public String usertype;
    1 usage
    public String userid;

    public User(String username, String password, String usertype, String userid) {
        this.username = username;
        this.password = password;
        this.usertype = usertype;
        this.userid = userid;
    }
}
```

Figure 13 Entity User



```
2 usages
public class PasswordUtility {

    // Default Work Factor. A higher value increases security but slows down computation.
    1 usage
    private static final int WORK_FACTOR = 12;

    no usages
    public static String hashPassword(String plaintextPassword) {
        // BCrypt.hashpw automatically generates a random salt and embeds it in the hash result.
        return BCrypt.hashpw(plaintextPassword, BCrypt.gensalt(WORK_FACTOR));
    }

    no usages
    public static boolean verifyPassword(String plaintextPassword, String hashedPassword) {
        // BCrypt.checkpw automatically extracts the salt and performs the comparison.
        if (hashedPassword == null || !hashedPassword.startsWith("$2a$")) {
            // Fail verification if the hash is null or not in the expected format.
            return false;
        }
        try {
            return BCrypt.checkpw(plaintextPassword, hashedPassword);
        } catch (Exception e) {
            // Handle potential exceptions from checkpw (e.g., malformed hash)
            return false;
        }
    }
}
```

Figure 14 Password Utility

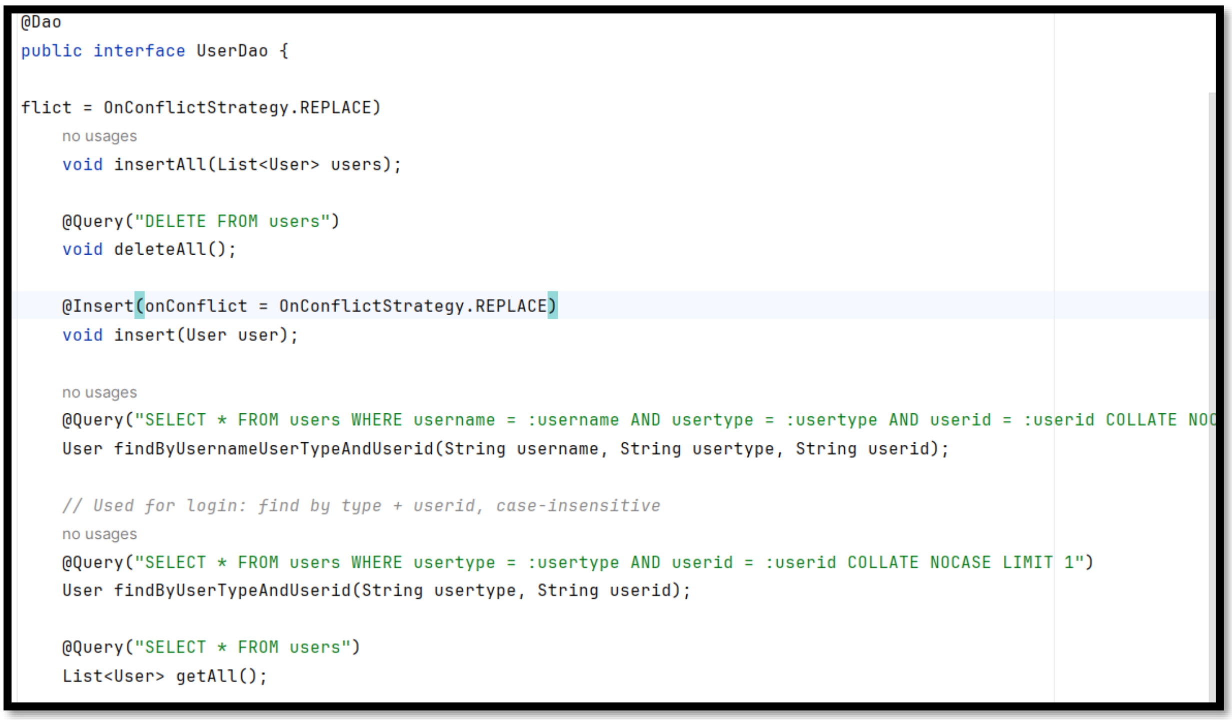
Data Structure: The User entity class defines key user attributes such as username and password.

Security Encapsulation: The password field stores the BCrypt hash value generated by the PasswordUtility.hashPassword() method, instead of the plaintext password. During login verification, the PasswordUtility.verifyPassword() method is called, encapsulating the password verification logic and ensuring the security of sensitive user information.

2. Applying the OOP concept: Abstraction

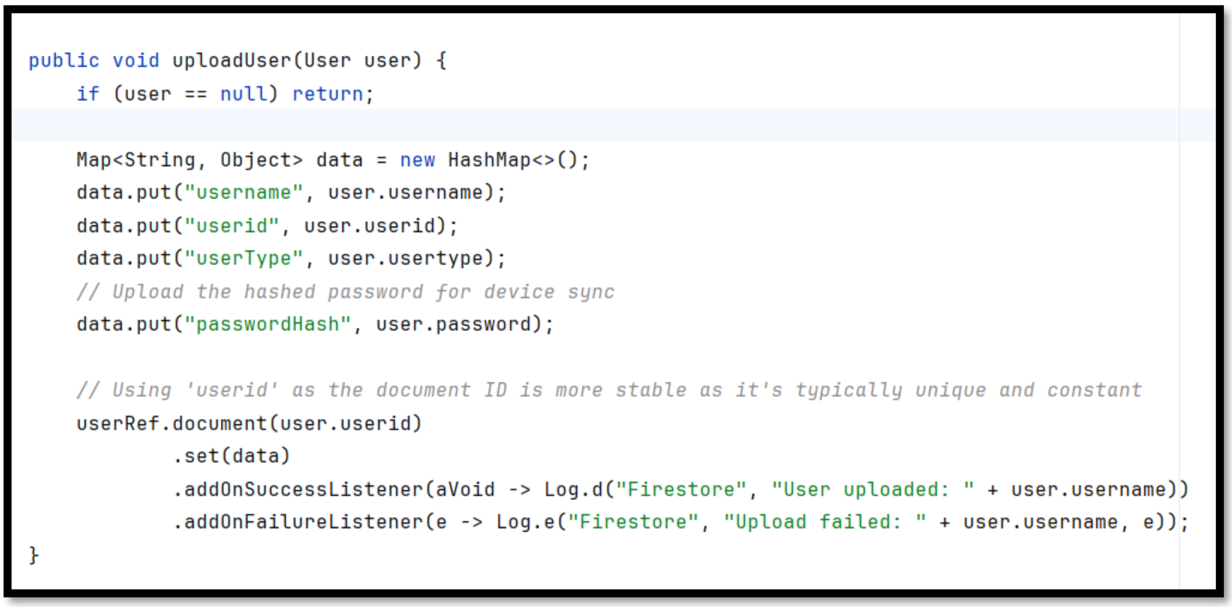
Abstraction is applied in the data persistence layer, hiding complex implementation details through the DAO (Data Access Object) interface and the Repository class, exposing only the necessary operation interfaces to the outside.

Implement:



```
@Dao
public interface UserDao {

flict = OnConflictStrategy.REPLACE)
    no usages
    void insertAll(List<User> users);

    @Query("DELETE FROM users")
    void deleteAll();

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void insert(User user);

    no usages
    @Query("SELECT * FROM users WHERE username = :username AND usertype = :usertype AND userid = :userid COLLATE NOC
    User findByUsernameUserTypeAndUserid(String username, String usertype, String userid);

    // Used for login: find by type + userid, case-insensitive
    no usages
    @Query("SELECT * FROM users WHERE usertype = :usertype AND userid = :userid COLLATE NOCASE LIMIT 1")
    User findByUserTypeAndUserid(String usertype, String userid);

    @Query("SELECT * FROM users")
    List<User> getAll();
```

Figure 15 User DAO



```
public void uploadUser(User user) {
    if (user == null) return;

    Map<String, Object> data = new HashMap<>();
    data.put("username", user.username);
    data.put("userid", user.userid);
    data.put("userType", user.usertype);
    // Upload the hashed password for device sync
    data.put("passwordHash", user.password);

    // Using 'userid' as the document ID is more stable as it's typically unique and constant
    userRef.document(user.userid)
            .set(data)
            .addOnSuccessListener(aVoid -> Log.d("Firestore", "User uploaded: " + user.username))
            .addOnFailureListener(e -> Log.e("Firestore", "Upload failed: " + user.username, e));
}
```

Figure 16User Repository

Database Abstraction: The UserDao interface defines abstract operations such as `insert(User user)`. Callers (such as RegisterFragment) only need to call the methods and do not need to know how Firebase performs data storage.

Data Source Abstraction: UserRepository provides a unified interface `uploadUser(User user)`. In RegisterFragment, after successful user registration, `db.userDao().insert(user)` and `userRepo.uploadUser(user)` are called, separating local database operations from Firebase upload operations and abstracting the data persistence logic.

PART D: Application Features and Testing

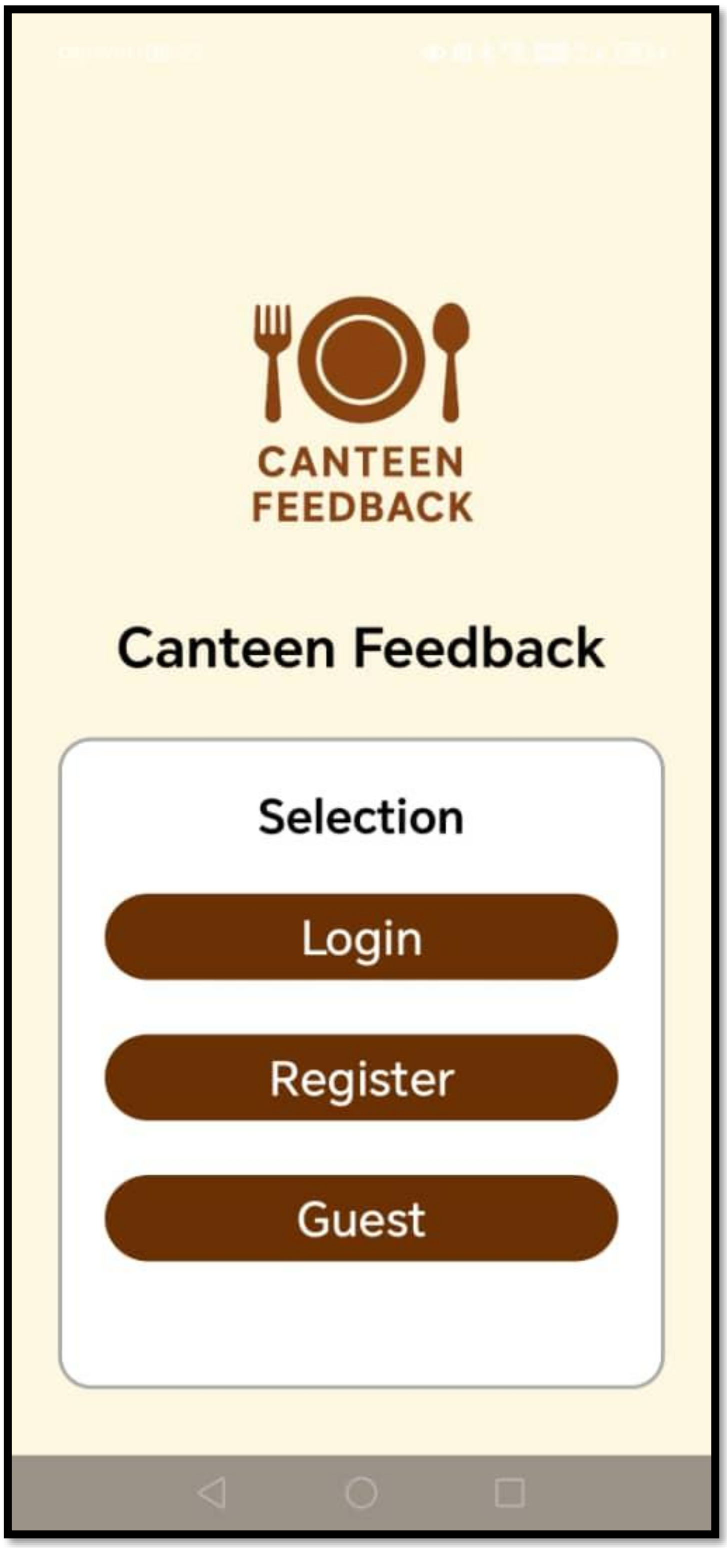


Figure 17 Selection page

Selection Page: System Portal and Identity Verification

This serves as the entry point for the “Canteen Feedback” application, playing a crucial role in user identity verification and process navigation. Users must first make a selection on this page:

- **Login:** For users with existing accounts, directly proceeds to the login process.
- **Register:** Guides new users to create a new system account.
- **Guest:** Allows users to access limited functionality (e.g., browsing feedback) without creating an account.

Based on the user's selection, the system clearly directs them to the corresponding operational page.

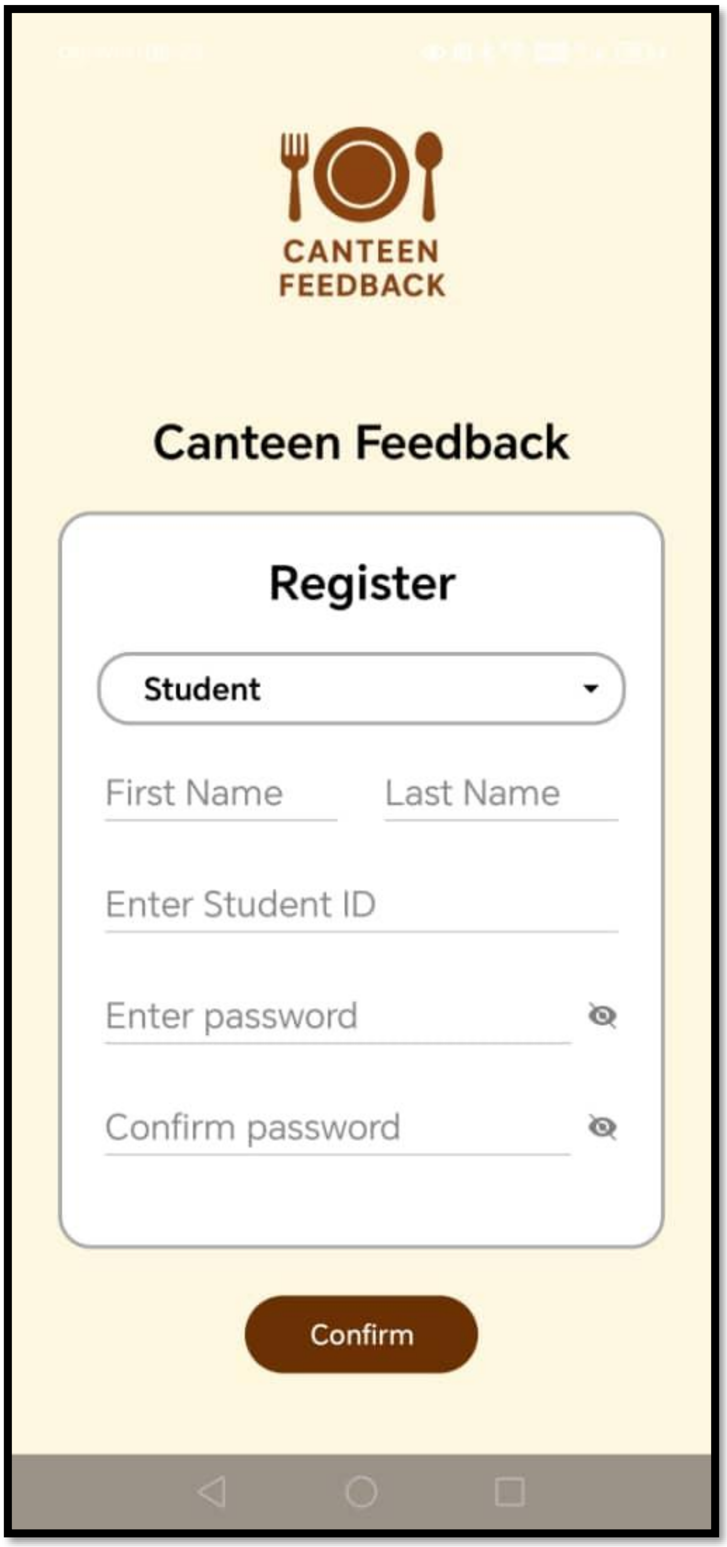


Figure 18 Register page

Registration Page: Account Creation Process for Collecting Key Information

When users select registration, the system redirects them to this account creation page. The core purpose of this page is to collect essential user identity information.

- **Role Selection:** The top dropdown menu allows users to confirm their identity within the system.
- **Basic Information:** Requires input of First Name and Last Name.
- **Unique Identifier:** Uses an ID as the account's unique credential to ensure accurate user identity.
- **Security Settings:** Set and re-enter the password (Confirm password) to ensure accuracy.

After clicking Confirm, the system validates the information. Upon successful verification, users are directed to the login page for their first login.

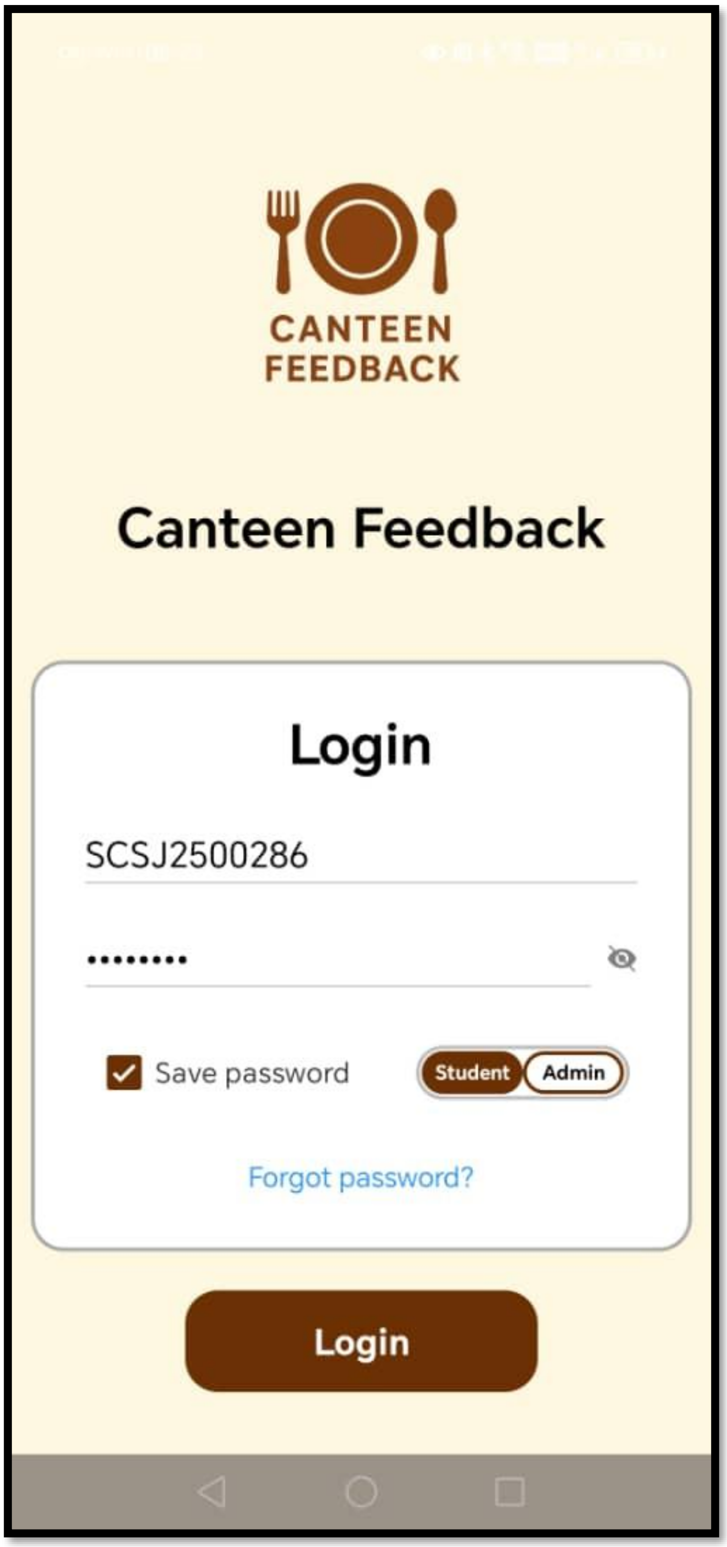


Figure 19 Login page

Login Page: Role-Based Permissions Portal

This serves as the secure authentication gateway for users accessing core system functions. Users may be automatically redirected here after successful registration or choose to enter directly from the initial page.

- **Credential Input:** Requires users to enter their registered ID and password.
- **Critical Role Selection:** The bottom of the page provides toggle options for Student and Admin roles. This is a critical step in the login process, as the system assigns corresponding access permissions and displays the appropriate interface based on the selected role.
- **Support Features:** Offers a "Save password" option for enhanced convenience in subsequent logins, along with a "Forgot password?" link to address password recovery scenarios.

Upon successful click of "Login," the system verifies credentials and directs the user to their exclusive main interface based on the selected role.

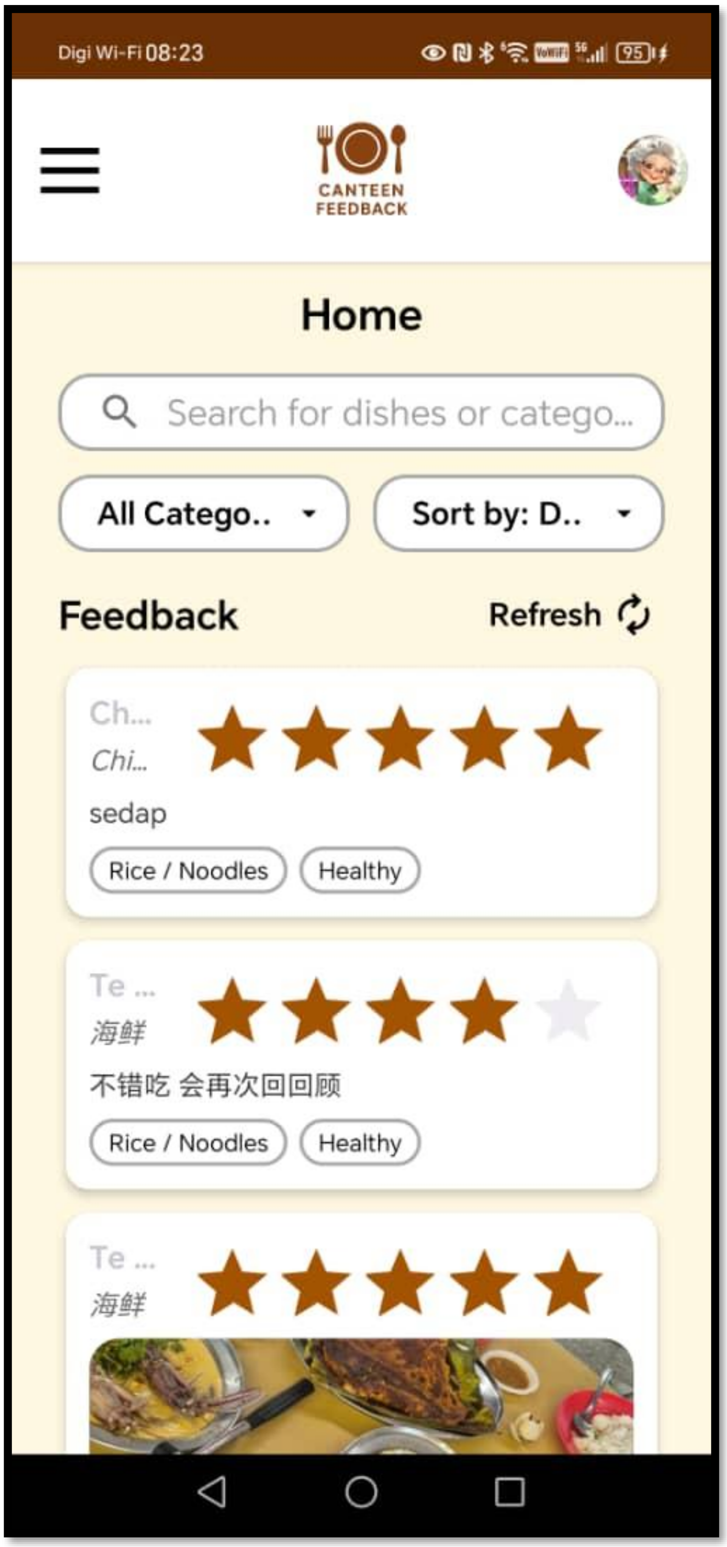


Figure 20 Home page

Homepage: Feedback Browsing Centre

This is the main system interface accessed after users successfully log in (or enter as guests), with its core function being to display feedback information about cafeteria food. This page provides a suite of tools enabling users to efficiently browse and filter feedback content.

- **Global Navigation and Settings:**
  The three horizontal lines (Hamburger Menu) in the top-left corner serve as the primary navigation menu for accessing other functional modules (such as submitting feedback). Clicking the user avatar in the top-right corner takes you directly to the settings page.

- **Search and Filter Tools:**
  - **Search Bar:** Users can enter keywords to search for specific dishes or categories.
  - **Filter (All Categories):** Allows users to filter feedback by dish category (e.g., “Rice / Noodles”).
  - **Sorting (Sort by: Date, Rating, etc.):** Users can sort the feedback list by date, rating, and other criteria.
  - **Feedback List:** The main page displays feedback for cafeteria dishes. Each entry includes key information:
  - **Dish Name/Content:** Shows the dish name (e.g., Ch... or Te...) and user comments (e.g., sedap, tasty).
  - **Rating:** Visually displays the user's rating for the dish using a five-star scale.
  - **Tags:** Such as Rice / Noodles and Healthy, used for quick identification of the dish's category and attributes.
  - **Refresh Function:** The Refresh button in the top-right corner allows users to manually refresh the feedback list to obtain the latest information.

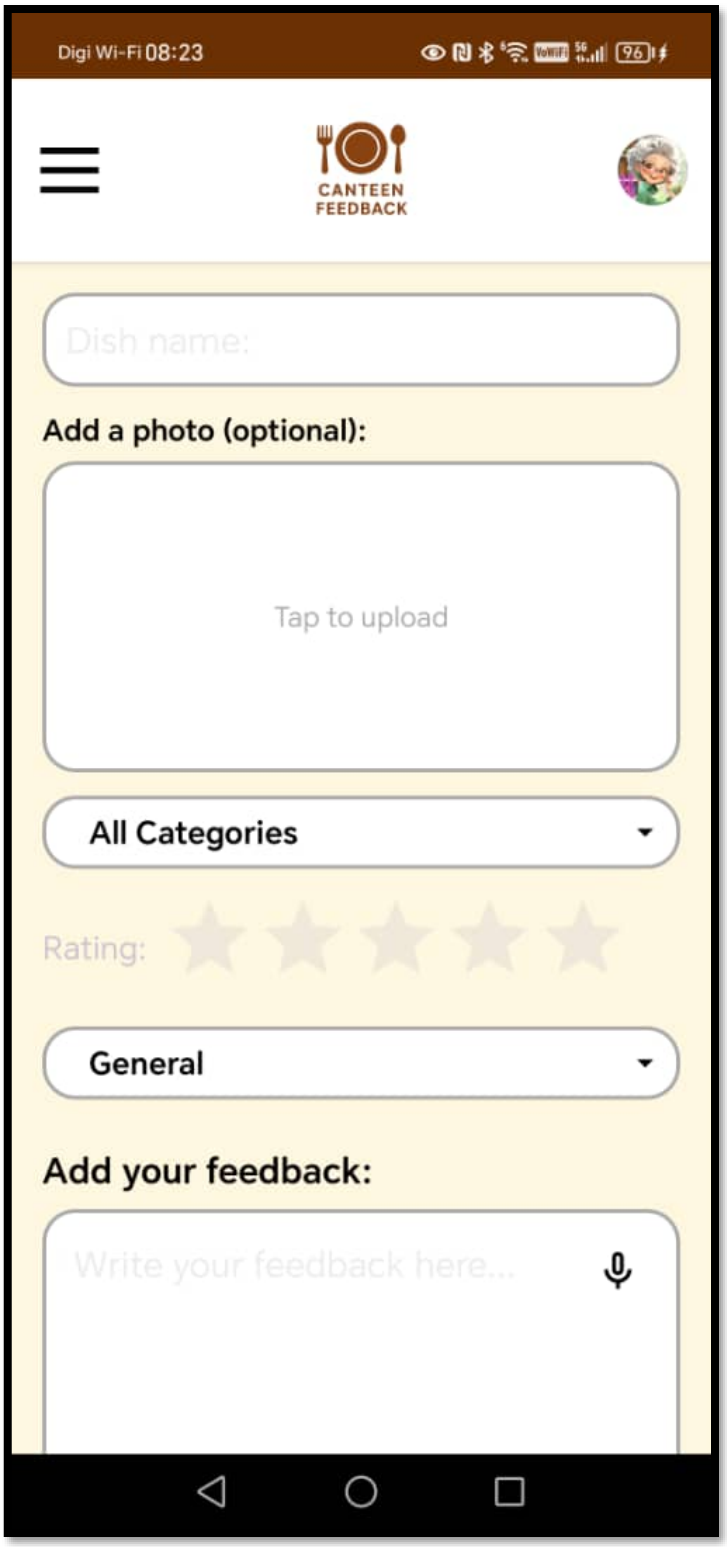
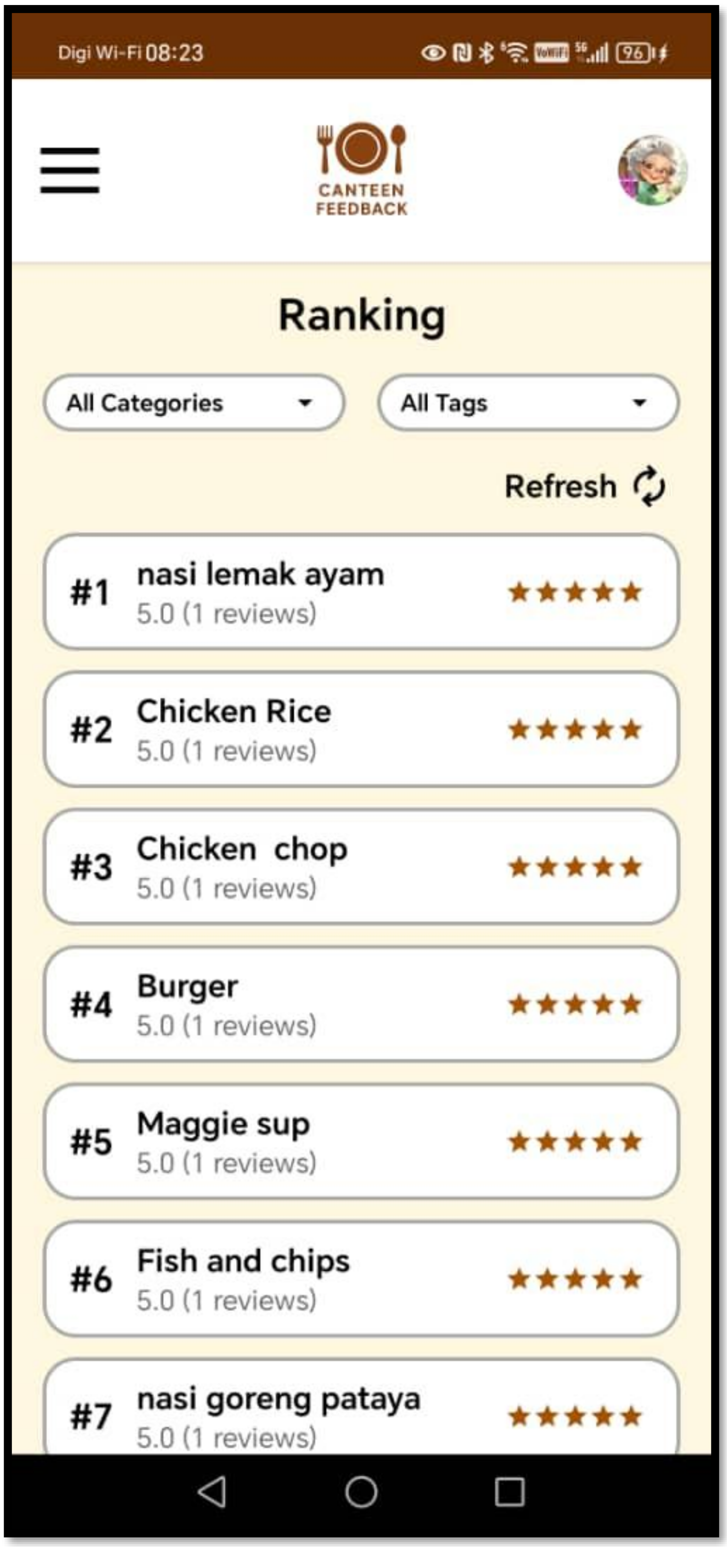


Figure 21 Feedback page

Feedback Page: Structured Opinion Collection Form

This is the dedicated form page for users to submit feedback on cafeteria dishes.

- **Dish Identification:** Users first enter the Dish name in the top input field.
- **Photo Attachment (Optional):** A large area is provided for users to Add a photo, which significantly enhances the feedback's visual appeal and credibility.
- **Dish Category:** The All Categories dropdown allows users to associate feedback with specific dish categories, facilitating subsequent filtering and analysis.
- **Rating:** Users intuitively quantify their experience by clicking on five stars, serving as the core evaluation metric for feedback.
- **Food Type:** Another dropdown menu (defaulted to General) enables users to further classify the nature of their feedback.
- **Text Comment:** The text box at the bottom labeled “Add your feedback” is where users freely express opinions and provide details.


Digi Wi-Fi 08:23

CANTEEN FEEDBACK

Ranking

All Categories

All Tags

Refresh

#1 nasi lemak ayam
5.0 (1 reviews) ★★★★★

#2 Chicken Rice
5.0 (1 reviews) ★★★★★

#3 Chicken chop
5.0 (1 reviews) ★★★★★

#4 Burger
5.0 (1 reviews) ★★★★★

#5 Maggie sup
5.0 (1 reviews) ★★★★★

#6 Fish and chips
5.0 (1 reviews) ★★★★★

#7 nasi goreng pataya
5.0 (1 reviews) ★★★★★

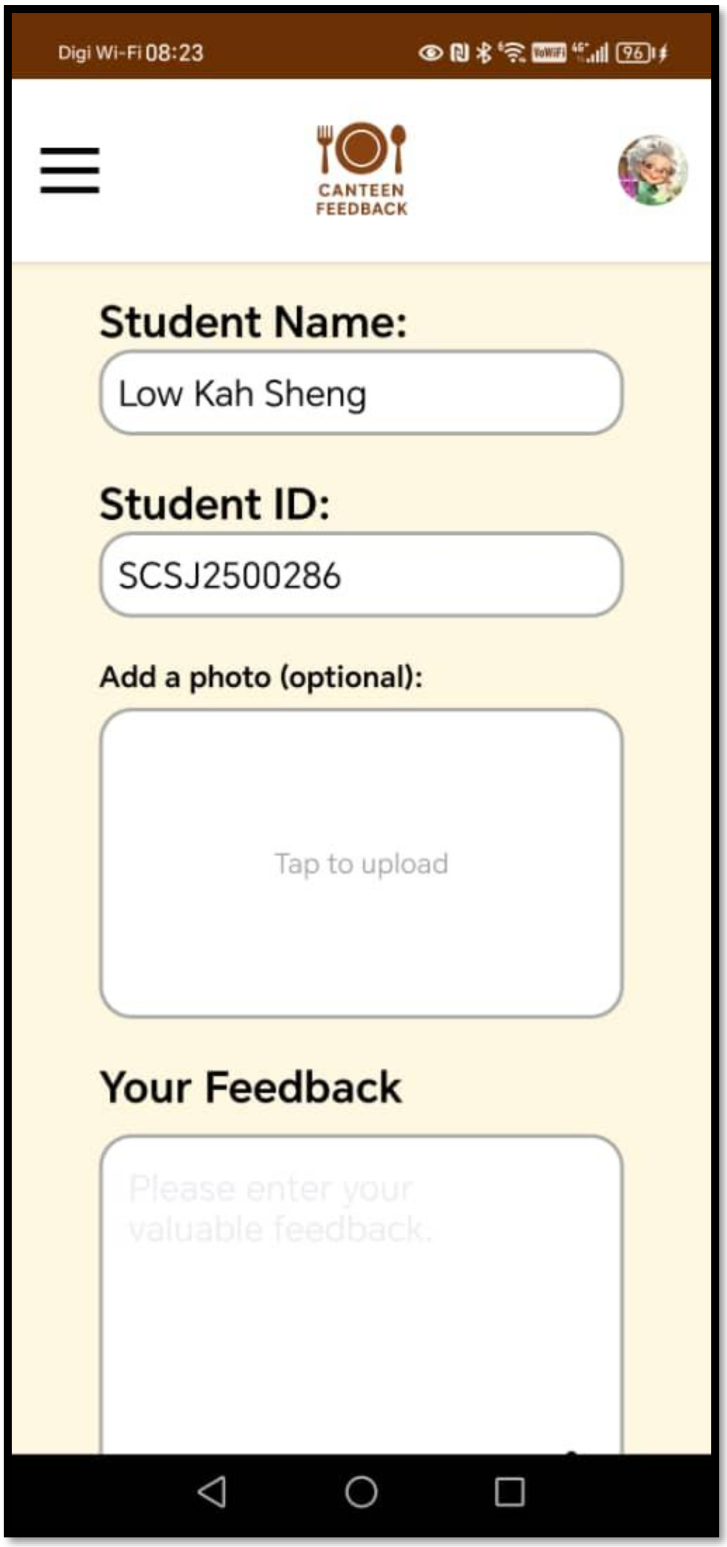
Figure 22 Ranking page

Ranking Page: A hub for dish rankings based on user ratings

This is the ranking page within the "Canteen Feedback" app, whose core function is to visually display the popularity of dishes based on users' overall ratings.

Filter Tools:

- **All Categories:** Allows users to filter the rankings by dish type (e.g., noodles, rice dishes) to view popular items within specific categories.
- **All Tags:** Enables users to further refine rankings by specific tags (e.g., "spicy") for more detailed comparisons.
- **Refresh Function:** The Refresh button ensures users can always access the latest ranking data.
- **Ranking List Structure:** The list presents ranking information in clear card formats:
- **Ranking Position (#1, #2, ...):** Clearly displays the dish's position under the current filter conditions.
- **Dish Name:** Clearly identifies the ranked dish (e.g., nasi lemak ayam, Chicken Rice).
- **Average Rating & Review Count:** Displays the dish's average star rating (e.g., 5.0) and total reviews contributing to the rating (e.g., 1 review), providing data-backed credibility for the ranking.
- **Star Icon:** Visually represents the dish's average rating using a five-star scale.



Student Name:

Low Kah Sheng

Student ID:

SCSJ2500286

Add a photo (optional):

Your Feedback

Figure 23 Contact us page

Contact Page: Feedback Submission Channel

This is a dedicated “Contact Us” page designed for users. Unlike the previous menu feedback page, this page focuses on collecting feedback or inquiries regarding cafeteria services, environment, or the application itself.

- **Image Attachment :**Features an “Add a photo” section. Users may upload images as supporting materials to visually illustrate their feedback.
- **Your Feedback:**A large text box for users to provide detailed, valuable feedback. With identity verification in place, users can confidently submit more formal or constructive suggestions.

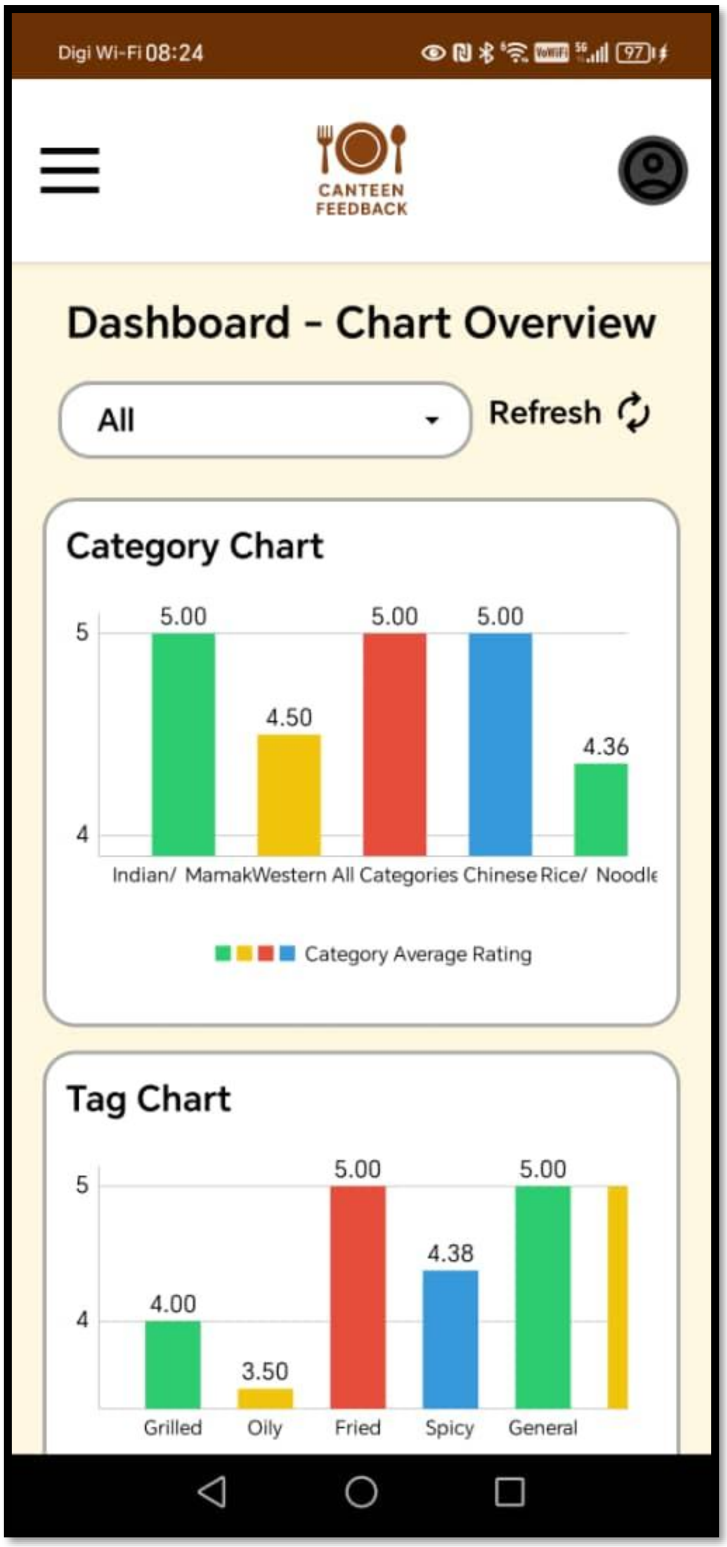


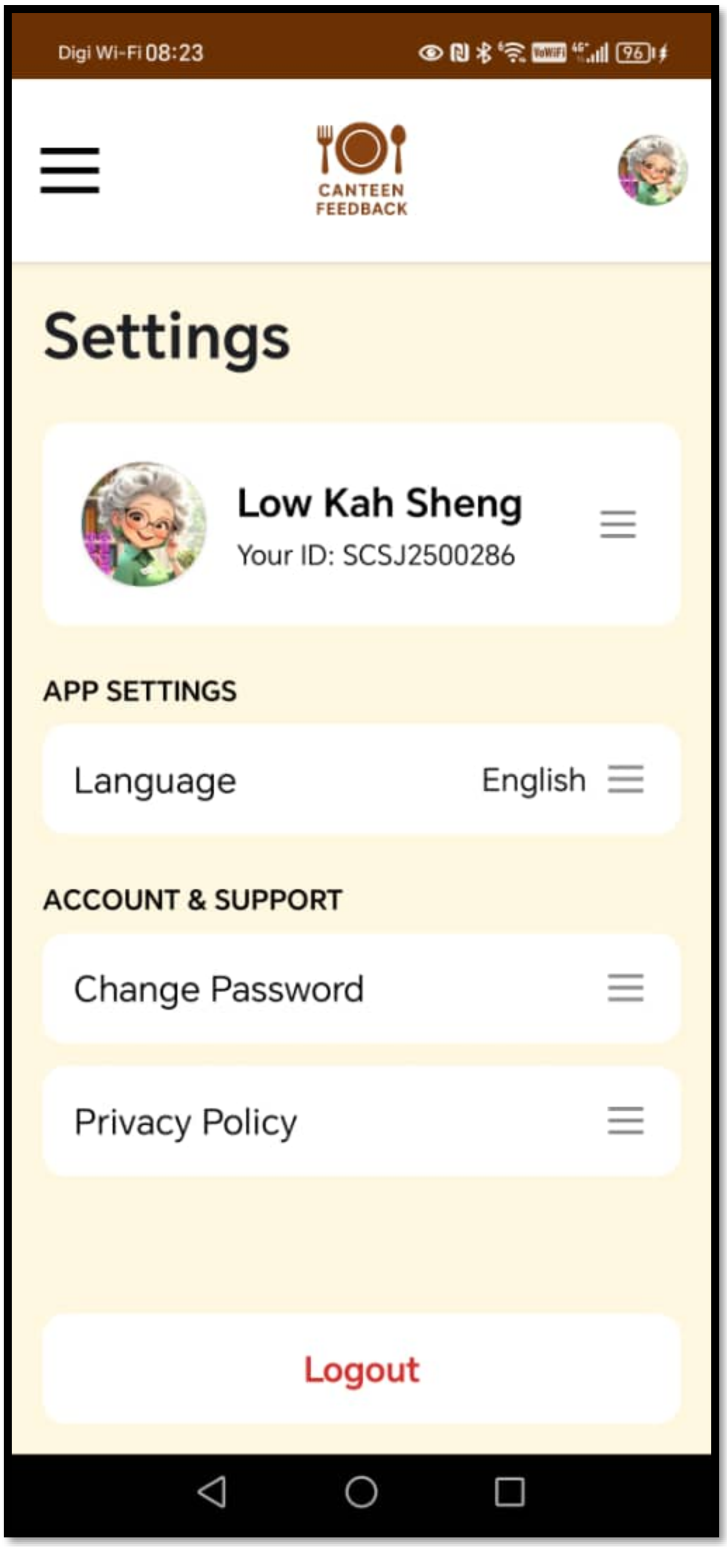
Figure 24 Chart page

Chart page: Data Overview Hub

This interface is designed for administrators to analyze user insights, with its core functionality being the visualization of all feedback data collected by the system. This enables administrators to quickly grasp user satisfaction levels.

Filtering and Refreshing:

- **Filtering:** The top dropdown menu (default: All) allows users to select specific categories to filter chart data for targeted analysis.
- **Refresh Function:** The Refresh button ensures charts display the latest, real-time feedback data.
- **Category Chart:** This chart displays the average ratings across different food categories.
- **Tag Chart:** This chart displays the average ratings for different food attribute tags.



Settings

Low Kah Sheng
Your ID: SCSJ2500286

APP SETTINGS

Language English

ACCOUNT & SUPPORT

Change Password

Privacy Policy

Logout

Figure 25 Setting page

Settings Page: Account Management and Application Configuration Centre

This is the personal settings hub where users manage their account information and configure application behaviour.

User Overview: The top of the page clearly displays the logged-in user's basic information, including username (Low Kah Sheng), user avatar, and user ID (SCSJ2500286). The menu icon (three horizontal lines) on the right allows users to access the profile editing page.

Application Settings (APP SETTINGS):

- **Language:** Allows users to switch the application's display language (currently set to English).

Account & Support:

- **Change Password:** A critical account security feature enabling users to modify their login credentials at any time.
- **Privacy Policy:** Provides access to the app's legal documentation, informing users about how their data is collected, used, and protected, thereby enhancing the app's transparency and credibility.

Logout: The red button at the bottom serves as the standard operation for users to securely exit their current session and return to the login page, protecting their account from unauthorized access.

Testing

Negative Feedback & Areas for Improvement

- **Password Limitations:** Users found the password options during registration to be too limited. This was inconvenient as it forced them to change their frequently used passwords.
- **Missing On-Screen 'Back' Button:** The application lacked a consistent on-screen back button (e.g., a < arrow). This was "very inconvenient" as users had to rely on their phone's "physical buttons" (or system gestures), which vary between devices and caused confusion.
- **Homepage Comment Display:** The space for displaying user comments on the homepage is too small, making it impossible to read the full text of the comments.
- **Feedback Page Layout:** The placement of the "Add your feedback" text on the feedback submission page was described as "out of place."
- **Dialog Box Design:** Users felt the default dialog box didn't match the application's theme.

Positive Feedback

- **Ease of Use:** Users stated that the application was "very easy to use."
- **Page Design:** The overall page design was praised as "user-friendly."

Part E: Consistency & UX Enhancement

1) **Compare your UML diagrams with the final code. Explain how they align.**
My final code implementation maintains a high level of alignment with the provided UML Use Case diagram. The diagram defined the system's functional requirements, and the final application successfully implemented them.

Table 8 comparison:

ID	Use Case	UML Description	Code Implementation (Screen)	Code Details/Match
1.	Submit Feedback	The use case <<includes>> Dish name, Rating, and Tag.	Feedback page (Figure 117)	Directly implemented with input fields for dish name, star rating, and tags.
2.	View all Feedback	The use case can <<extend>> to Select category.	Home page (Figure 16)	Displays all feedback and correctly includes the optional "All Categories" filter, matching the UML design.
3.	View ranking of food	This is a key feature for Students and Admin	Ranking page (Figure 18)	Successfully processes data to show a ranked list, fulfilling the use case.
4.	Setting profile	The use case can <<extend>> to Reset password.	"Settings" page (Figure 21)	Shows the logged-in user profile and includes the (Change Password) option, matching the UML design.

2) **Identify at least two design improvements for UX. You may illustrate using draw.io / Figma**

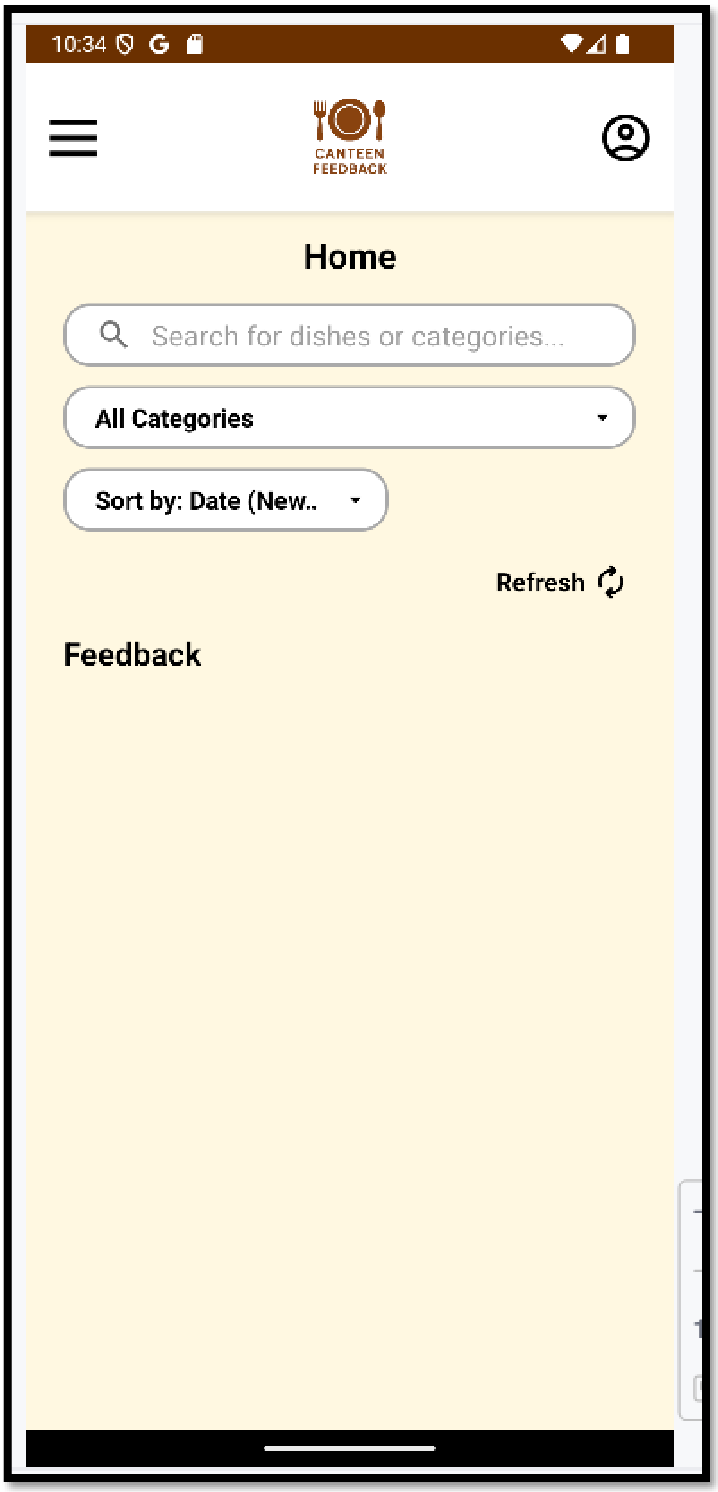


Figure 26 Home page Before

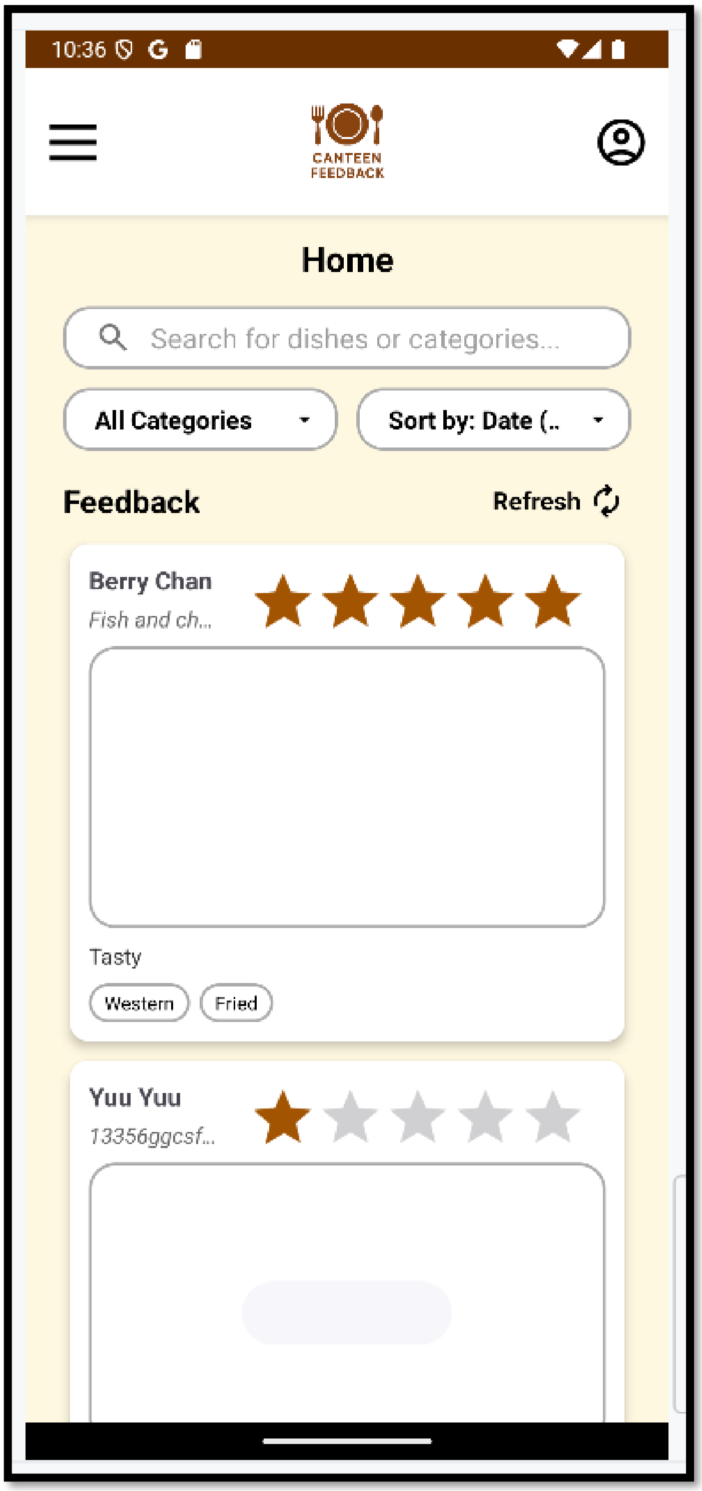
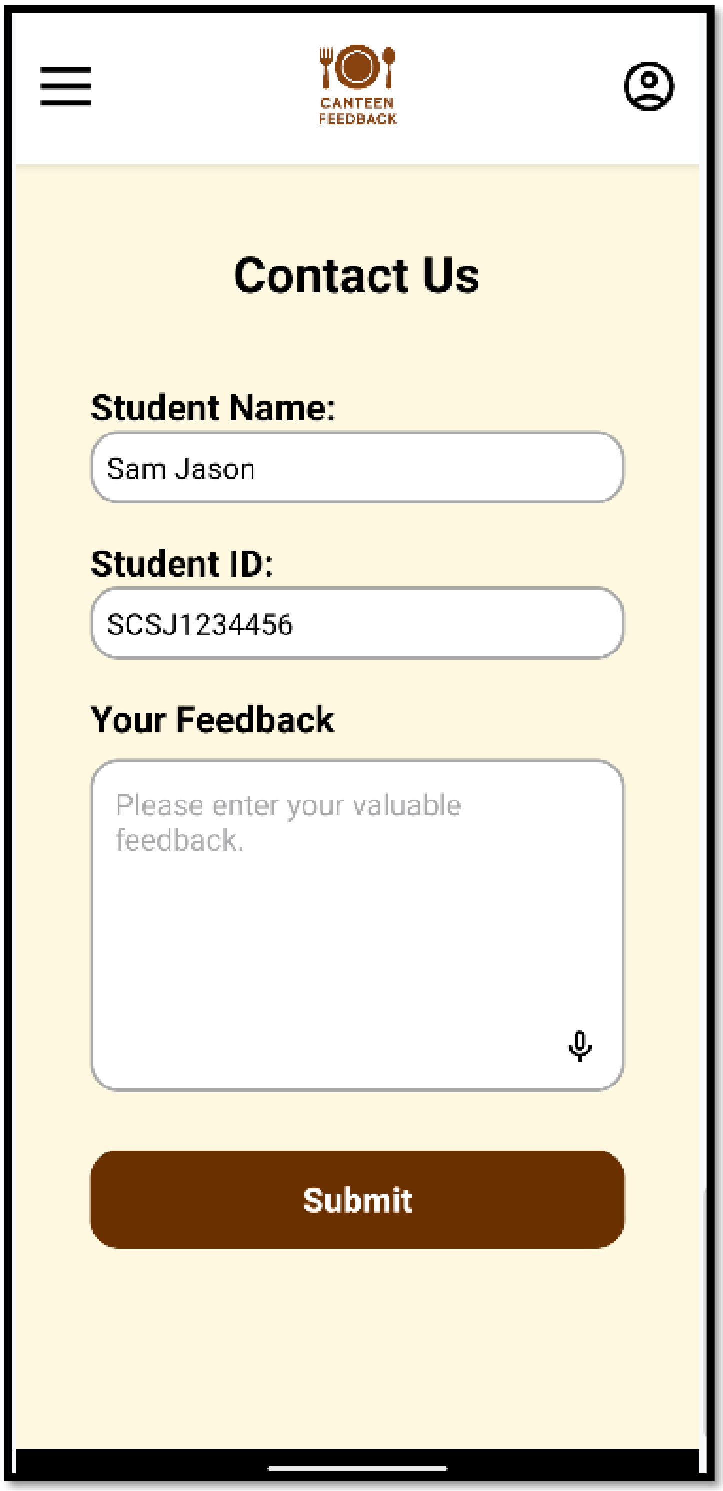


Figure 27 Home page after

Table 9 Comparison of home page

Improvement	**"Before" (Figure 22) - The Problem**	**After" (Figure 23) - The Solution (The Improvement)**
Space Optimization & Layout	**Friction from Pre-filled Text:** The "Student Name" and "Student ID" fields contained sample text (e.g., "Sam Jason"). The user was forced to manually delete this text.	The layout was **optimized by adjusting the filter area's position**. This change successfully **enlarged the feedback browsing area**, making the list easier to read.
Layout & Stainability	**No Layout:** It was just an empty space with no information structure. This is not scannable or usable.	**Card-Based Layout:** Each piece of feedback is encapsulated in its own "card." This design is **highly scannable** and allows users to easily distinguish between different entries.
Information Hierarchy	**No Information:** Lacked any data, and therefore, had no information hierarchy.	**Clear Visual Hierarchy:** Within each card, information is well-organized: **1. Stars (Rating)**, **2. User (Sam Chan)**, **3. Comment 4. Tags (Tasty)**. The user can see the most critical info (the rating) briefly.



Contact Us

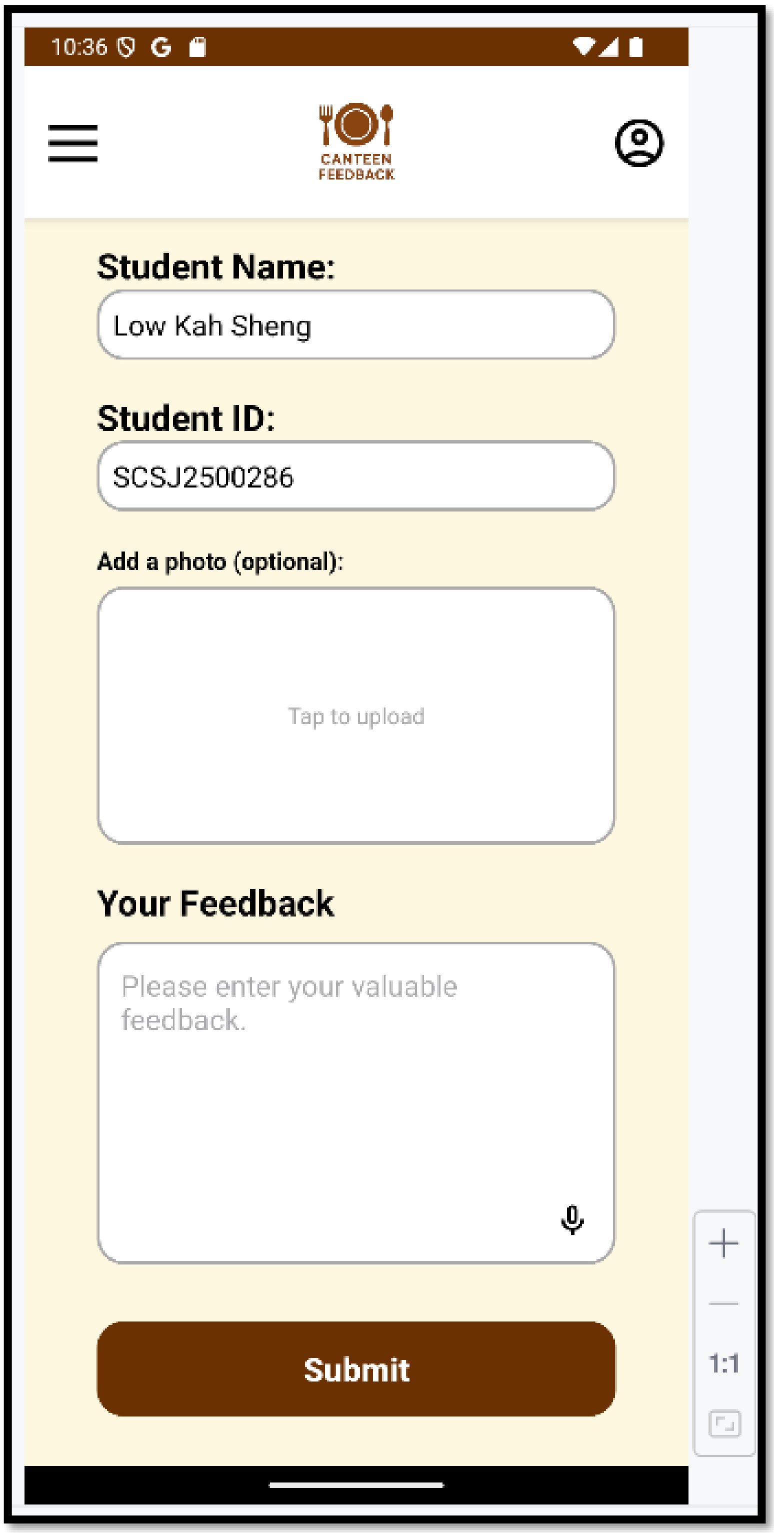
Student Name: Sam Jason

Student ID: SCSJ1234456

Your Feedback

Please enter your valuable feedback.

Figure 28 Contact us page before



Student Name:
Low Kah Sheng

Student ID:
SCSJ2500286

Add a photo (optional):

Your Feedback

Figure 29 Contact us page after

Table 10 Comparison of contact us page

Improvement	**"Before" Figure 24）- The Problem**	**"After" (Figure 25) - The Solution (The Improvement)**
Input Field Usability	**The inconvenience caused by pre-populated** text includes the "Student Name" and "Student ID" fields. Users must manually delete these sample texts before entering their own data.	**Empty, Ready-to-Use Fields:** The fields are now empty by default. This removes the unnecessary step of deleting text and makes the form immediately ready for user input, reducing friction.
Enhanced Functionality	**Input options** are limited: This form only accepts text feedback. If users encounter complex issues, they can only describe them in text.	**Media Input (Photo Upload)**: The new "Add Photo (Optional)" feature is a major upgrade. It allows users to provide visual background information, making their feedback more valuable.
Clear & Guiding Labels	**Ambiguous Form:** The form is very basic and provides no extra guidance.	**The clear "Optional"** label: The text design of "Add photo (optional)" is excellent. It indicates that this field is not required, thus alleviating user anxiety.

F） Reflection

This project served as a valuable learning experience, demonstrating how planning, coding, and user feedback work together. From the outset, planning tools were crucial. The IPO analysis clarified the functions’ requirements—including their inputs, processes, and outputs. Flowcharts provided a step-by-step overview of the process logic, while UML diagrams illustrated the overall system architecture, showing how users interact with each function. These tools acted as blueprints, facilitating a more efficient coding process.

The final code strictly adhered to this blueprint. As explained in Part E, the "Submit Feedback" use case in the UML diagram directly corresponds to the "Add Feedback" screen. Similarly, the "View All Feedback" use case translates into the main "List Feedback" screen. This demonstrates a clear consistency between the design and the final product.

Applying Object-Oriented Programming (OOP) principles in Java presented several challenges, particularly in terms of Abstraction and Encapsulation. Determining which data should be included in the Feedback or User classes, and deciding which variables should remain private to protect data while allowing safe access from other parts of the application, required careful consideration.

User feedback had a significant impact on the final design. It highlighted the importance of usability in addition to functionality. For example, users experienced confusion due to the absence of an on-screen "Back" button, prompting its addition to every page. Feedback also indicated that the password limit was inconvenient, leading to an adjustment from 8–16 to 8–24 characters.

Finally, Java and the Android SDK enabled the development of a real-world interactive application. Java’s OOP features allowed the creation of reusable objects, such as Feedback objects, to manage data efficiently. The Android SDK provided UI components, including RecyclerView for lists and interactive Buttons, allowing the design to be translated into a fully functional, interactive application.

References:

Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development* (3rd ed.). Addison-Wesley.

Daniels, R., & Kazman, R. (2000). *Objects, components, and frameworks with UML: The Catalysis approach*. Addison-Wesley.

Kendall, K. E., & Kendall, J. E. (2010). *Systems analysis and design* (8th ed.). Prentice Hall.

Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson.

Horstmann, C. S. (2020). *Java: The complete reference* (11th ed.). McGraw-Hill Education.

Bloch, J. (2018). *Effective Java* (3rd ed.). Addison-Wesley.

Larman, C. (2004). *Applying UML and patterns: An introduction to object-oriented analysis and design* (3rd ed.). Prentice Hall.

Nielsen, J. (1994). *Usability engineering*. Morgan Kaufmann.

Rubin, J., & Chisnell, D. (2008). *Handbook of usability testing: How to plan, design, and conduct effective tests* (2nd ed.). Wiley.

Farooq, U., & Yamin, F. (2021). *Usability testing on Android-based mobile applications: Case study of Smart Assistant Diabetes*. Journal of Mobile Technology in Medicine, 10(1), 25–36.

Deitel, P., & Deitel, H. (2017). *Android 8 for programmers: An app-driven approach*. Pearson.

Meier, R., & Kuriakose, S. (2012). *Professional Android 4 Application Development*. Wrox.

Li, L., Li, J., & Zhang, J. (2018). *Model-based testing of mobile systems: An empirical study on QuizUp Android app*. Journal of Systems and Software, 141, 1–15.

Nguyen, T., & Kim, D. (2020). *Humanoid: A deep learning-based approach to automated black-box Android app testing*. IEEE Access, 8, 215432–215443.