Project on Mobile Programming

The requirement of your report:

- Submit a video to show that your application works well;
- Include a `.pdf` file to explain your design and code structure
- Final Presentation: **7 May**; please prepare a 5-10 minutes demo
- he first two tasks can be done in groups of 1–2 people the last two (presentation and report) can be done in groups of 1–4 people

Criteria	**Points**
Functional completeness (core features, requirements met)	5
UI design and usability (navigation, layout, validation)	5
Database integration (SQLite/SharedPreferences usage)	4
Design complexity	3
Presentation and report (clarity, demo, code explanation)	3
Total	**20**

Table 1: Evaluation Table

1 Expense Tracker App

1.1 Description

The Expense Tracker App is designed to help users manage their daily financial activities. The app allows users to record income and expenses, view transaction histories, and track their financial status over time.

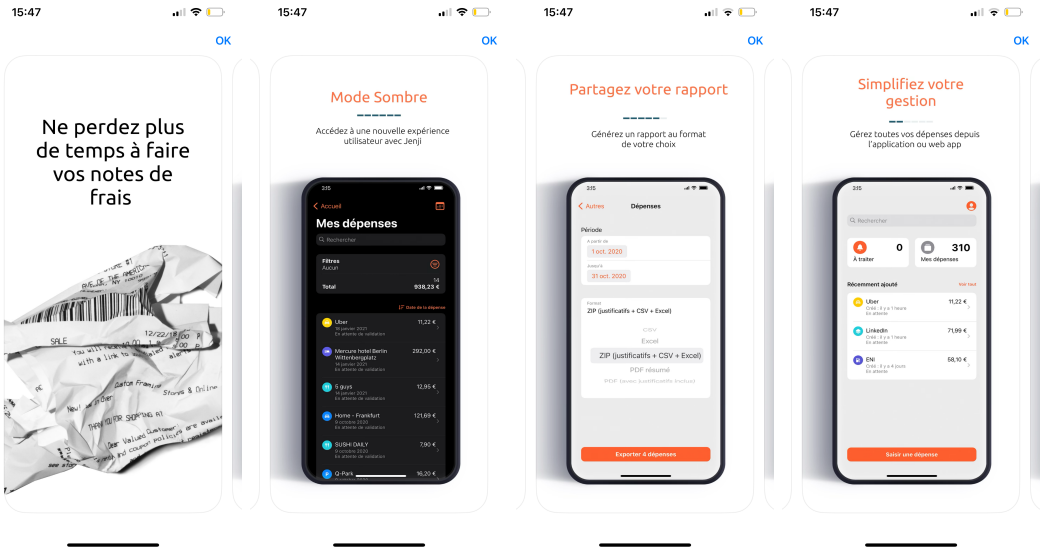

Figure 1: Just an example

1.2 Features

- **Add Income and Expense Records:**
  - Users can add transactions by specifying:
    - Amount
    - Category (e.g., Food, Transportation, Salary)
    - Date of the transaction

  - Use a simple input form with `EditText` fields for amount and category, and a `DatePickerDialog` for selecting the date.

- **Display Financial Summary:**
  - Show total income, total expenses, and net balance.
  - → Use `TextView` components to dynamically update and display these values on the home screen.
- **View Transaction History:**
  - Users can view a list of all recorded transactions, sorted by date.
  - Provide filtering options by :
    * Date range
    * Category
  - → Use `RecyclerView` to display the transaction list, with filters implemented as dropdown menus or dialogs.
- **Data Visualization:**
  - Show a visual breakdown of expenses by category using a pie chart.
  - → Use libraries like `MPAndroidChart` or `;`AnyChart for easy chart rendering.

1.3 Implementation Suggestions

- **Database:**
  - Use `SQLiteDatabase` or `Room` to store:
    * Transaction records (fields: ID, amount, category, date, type [income/expense]).
  - Use SQL queries to filter and sort data as needed.
- **Activity Flow:**
  - **Main Activity:**
    * Displays financial summary (income, expenses, balance) and a list of recent transactions.
  - **Add Transaction Activity:**
    * Allows users to input transaction details and save them to the database.
  - **History Activity:**
    * Displays a complete transaction list with options to filter by date or category.
  - **Statistics Activity:**
    * Visualizes expense breakdown by category using a pie chart.
  - ...
- **UI Design:**
  - Use `Material Design` components for a modern interface.
  - Use `CardView` for transaction items in the history list.
  - Include a `FloatingActionButton` for quick access to the Add Transaction Activity.
- **Additional Features (Optional):**
  - Export transaction history as a CSV file.
  - Add notifications for periodic expense tracking reminders.

2 Post Office Package Manager App

2.1 Description

This app simulates a post office package and letter registration system. The goal is to help postal staff track customer package details, validate input, and calculate prices in real time. Students are expected to build this app using Java and Android Studio, applying concepts learned throughout the course.

The application should allow:

- Each customer can send one or more packages in a visit.
- For each package, the following information must be recorded:
  - Destination: Domestic or International
  - Dimensions: Height, Width, Length (in cm)
  - Weight (in kg)
- System validation:
  - Max dimensions: 100 cm (each side), max weight: 100 kg
  - Max packages per day: 50
  - If these limits are not respected, display an error message
- Pricing model:
  - Domestic package: €2 per kg
  - International package: €3 per kg
  - Show price dynamically in the UI
- At the end of the day, a summary of all registered packages must be viewable for processing.

Recommended Implementation:

- Use `SQLiteDatabase` to store package info (ID, destination, size, weight, price).
- Use `RecyclerView` to display all packages in a list format.
- Use `Fragments` or `Activities` for separating input screen and summary.
- Input validation should be implemented via Java code and visual feedback (e.g., Toast, Snackbar).

2.2 Bonus: Letter Functionality

Add a separate menu for sending letters:

- Allow selection between Package and Letter modes via `TabLayout`, `BottomNavigationView`, or a `Spinner`.
- Letter constraints:
  - Max weight: 100 grams
  - Domestic letter: €0.50, International: €1.00
  - Max letters per day: 200

2.3 Implementation Suggestions

- Use `BottomNavigationView` or `TabLayout` to navigate between screens (Home, Package, Letters, Summary).
- Use `CardView` or `RecyclerView` to show package or letter info.
- Use `SQLiteOpenHelper` to manage the database and CRUD operations.
- Use `SharedPreferences` for quick settings or profile info (optional).
- Design should follow Material Design principles with proper validation and feedback.
- Notifications for summary or capacity limit can be done via `AlarmManager` or `WorkManager`.

3 Pet's Life App

3.1 Description

The pet is a human's friend; they bring companionship, protection, and entertainment. It is important to keep tracking a pet's health as well as capturing the remarkable moments with them. In this project, you design an app to record the information for pets, including but not limited to:

- Choose the type of pet, like cat, dog, rabbit, with the name and breed.

  → Use custom-designed buttons for pet type selection. Intents can be used here to direct to other activities, or use Fragments for better navigation.

- Pet's information can be saved in the SQL database, including age, weights, medical info.

  → Use `SQLiteDatabase` to create tables and manage data. Include fields such as pet ID, name, type, breed, age, weight, and medical history.

- Optional information can be recorded, including playtime, training sessions, and health checkups.

  → Log activity details in a separate table and display them in a timeline format using `RecyclerView`.

- One tab for reminders, such as vaccination dates, preferences for the pet's food, and bath dates.

  → Use Fragments to organize the reminder section. Notifications can be implemented using `AlarmManager` or `WorkManager` to alert users.

- One tab for the pet profile.

  → `SharedPreferences` can be used to save a small amount of persistent data, such as the selected profile photo or last-viewed pet. Use `ImageView` to display profile pictures.

- Nice UI design is a strong plus.

3.2 Bonus

Other functions you can add to the app can be:

- Real-time Weather Forecasting to provide indications for caring for pets, like avoiding walks in heavy rain or extreme heat.

  → Use weather APIs (e.g., OpenWeatherMap or WeatherStack) to fetch and display current weather conditions. Parse the JSON response and show the weather information on the home screen.

- Choose a photo from the album or take a photo to update the profile.

  → Use `Intent.ACTION_PICK` to allow photo selection from the gallery. Use the camera API to capture and save images directly. Store the image URI in the database.

- Activity Suggestions based on the pet type, weather, or time of the day.

  → For example, suggest outdoor activities like walks for dogs in the evening, or remind about feeding schedules. Use predefined rules or user preferences stored in `SharedPreferences`.

- A timeline feature for tracking the pet's milestones and key moments.

  → Log and display pet milestones such as first bath, first vaccination, or training achievements using a `RecyclerView`.

3.3 Implementation Suggestions

- Use `BottomNavigationView` for main app navigation, with tabs for Home, Reminders, Activities, and Profile.
- Use `Material Design` components for a user-friendly interface. Consider using `CardView` for displaying pet information in the Home tab.
- Structure the app using `Activity` and `Fragment` for better modularity and maintainability.
- For persistent storage, use `SQLiteDatabase` with helper classes for CRUD operations.
- For small and fast data (like preferences), use `SharedPreferences`.
- Implement notifications with `AlarmManager` or `WorkManager` for reminders.
- Use `Glide` or `Picasso` libraries to handle image loading efficiently.

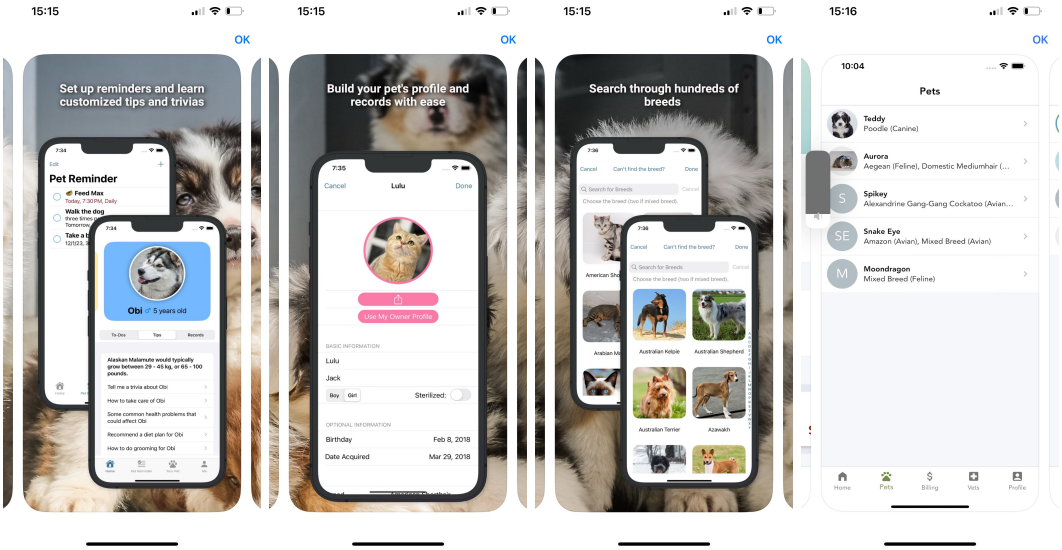

Figure 2: Just an example

4 Cinema Ticket Selling System

4.1 Description

This section focuses on designing a ticket-selling system for a cinema. The system includes functionalities to browse movies, select showtimes, pick seats, and confirm ticket purchases. The implementation is simplified for beginners to understand.

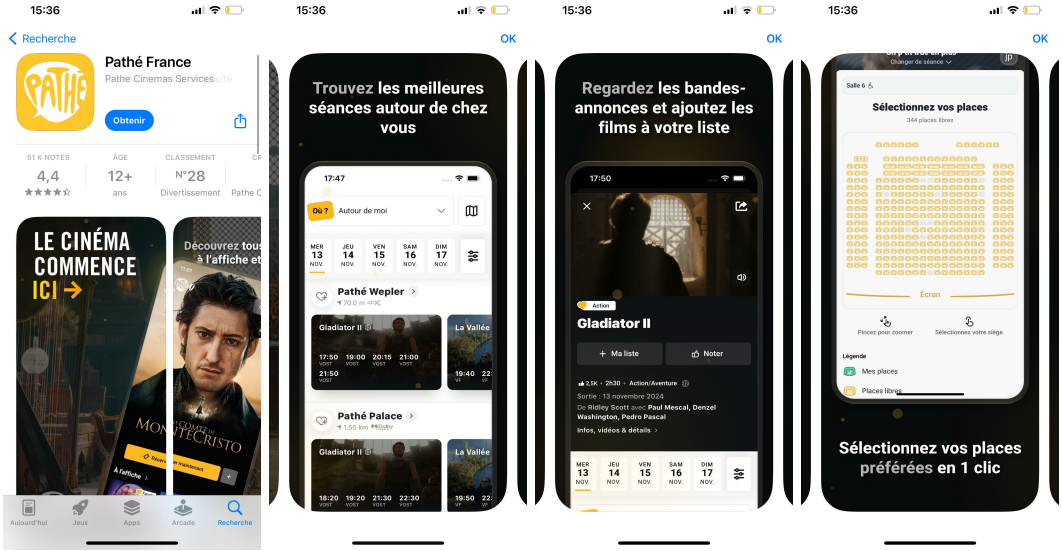

Figure 3: Just an example

4.2 Features

- **Movie Listing:**
  - Display a list of movies available for booking.
  - Each movie card shows the title, poster, showtimes, and ticket price.
  - → Use `RecyclerView` to display the list, with a card layout for each movie.
- **Showtime Selection:**
  - After selecting a movie, users choose a specific showtime (date and time).

  - → Use a `Spinner` or a vertical list to display available showtimes.

- **Seat Selection:**
  - Display a seat map for the selected showtime, where users can choose their desired seats.
  - Differentiate between available seats, selected seats, and sold-out seats using colors (e.g., green for available, red for sold).
  - → Use a `GridView` or a custom `View` to implement the seat layout.

- **Ticket Confirmation:**
  - Show a summary of the booking, including:
    - Movie name, selected showtime, seat numbers, and total cost.
  - Users confirm their booking by simulating a payment process.
  - → Use a `Dialog` to display the confirmation and show a "Success" message after payment.

4.3 Implementation Suggestions

- Use `SQLiteDatabase` to store:
  - Movie data (ID, title, poster URL, price, available showtimes).
  - Seat availability (mapped by showtime and movie ID).

- **Activity Flow:**
  - **Main Activity:** Display a list of movies.
  - **Details Activity:** Show details of a selected movie, including available showtimes.
  - **Seat Selection Activity:** Allow users to choose seats for a specific showtime.
  - **Confirmation Activity:** Display booking summary and process confirmation.
  - ...

- Use `Material Design` components like `CardView`, `BottomNavigationView`, and `FloatingActionButton` for a modern and user-friendly UI.
- For images (movie posters), use the `Glide` or `Picasso` library to load them efficiently.
- For mock payment, simply show a success dialog using `AlertDialog`.

4.4 Bonus Features

If time permits, consider adding:

- A history tab showing previously booked tickets.
- QR code generation for digital tickets using a library like ZXing.
- Sorting or filtering movies by genre or language.