Quick Mart - Project Documentation

TEAM: 280

1. Project Proposal

1.1. Overview

"Quick Mart" is a modern, user-friendly mobile e-commerce application designed for Android devices. It provides customers with a seamless and efficient platform to browse, search, and purchase a wide variety of products. The application features a clean, intuitive user interface built with modern Android technologies like Jetpack Compose and follows a robust MVVM architecture. The system integrates with a remote backend for real-time product data and leverages a local Room database for persisting user-specific data like the shopping cart, ensuring high performance and offline accessibility for the cart.

1.2. Objectives

- Develop an Intuitive Mobile Shopping Experience: To provide users with an easy-to-navigate interface for browsing products, viewing details, and managing their shopping cart.
- Ensure Robust Performance: To build a responsive and fast application by leveraging Jetpack Compose, asynchronous data loading, and local caching strategies.
- Implement a Complete E-commerce Workflow: To cover the entire customer journey, including product browsing, category filtering, search, adding items to a cart, and a simulated checkout process.
- Create a Scalable and Maintainable Architecture: To design the software using clean architecture principles (MVVM with a Repository pattern), making it easy to test, debug, and extend in the future.
- Provide Essential User Features: To implement core e-commerce functionalities such as a persistent shopping cart with an item count badge, product search, and browsing by categories.

1.3. Scope

The project scope covers the development of the "Quick Mart" Android application with the following key features, as identified in MainActivity.kt:

- Main Navigation: A bottom navigation bar for Home, Categories, and Cart screens.
- Home Screen: Displaying featured products and categories, with navigation to "All Products" and "All Categories".
- Product Listings: Screens for viewing all products (AllProductsScreen) and products filtered by a specific category (ProductsForCategoryScreen).
- Product Details: A dedicated screen (ProductDetailsScreen) with detailed product information and an "Add to Cart" option
- Category Exploration: A screen (CategoriesScreen) for users to browse all available product categories.
- Shopping Cart: A persistent cart (CartScreen) where users can view items and proceed to checkout. It includes a badged icon in the navigation bar to show the item count.
- Checkout Process: A simulated payment screen (PaymentScreen) and a success confirmation (SuccessPaymentActivity).

Out of Scope for this version:

- User authentication (login/registration).
- Real payment gateway integration (the current flow is a simulation).
- Order history and user profiles.
- Backend (API) development (assumed to be pre-existing).

2. Project Plan

2.1. Project Timeline

This project will be executed in three distinct phases to ensure organized development and timely delivery.

1. **Phase 1: Planning & Design (1 Week)**
    - Activities: Finalize requirements, create UI/UX mockups for all screens, and define the software architecture (MVVM) and database schema (Room).
    - Deliverable: A complete project documentation and approved UI designs.

2. **Phase 2: Development (5 Weeks)**
    - Sprint 1 (1 Week): Set up the project, including dependencies (Compose, Navigation, Room), and implement the core data layer (Models, Networking with Retrofit/Ktor, Room entities).
    - Sprint 2 (1 Week): Develop the Home, All Products, and Product Details screens with their corresponding ViewModel logic.
    - Sprint 3 (1 Week): Implement the Categories and Products-by-Category screens.
    - Sprint 4 (1 Week): Build the local database integration for the shopping cart (CartViewModel, LocalDataSourceImp) and the Cart screen UI.
    - Sprint 5 (1 Week): Finalize the checkout flow (PaymentScreen, SuccessPaymentActivity) and search functionality (SearchActivity).
    - Phase 3: Testing & Deployment (1 Week)
    - Activities: Conduct comprehensive testing (unit and integration), perform User Acceptance Testing (UAT) to gather feedback, and fix any reported bugs.
    - Deliverable: Final, polished .apk file and complete source code repository.

2.2. Milestones & Deliverables

- M1: Project Plan Approval: Submission of this complete documentation.
- M2: UI Prototype: High-fidelity mockups of all application screens.
- M3: Core Functionality Alpha: A build with home, product, and details screens functional.
- M4: Feature-Complete Beta: A build including cart, search, and checkout functionality for testing.
- M5: Final Application: The final, and polished app.

2.3. Resource Allocation

- Development: Android Studio, Kotlin, Jetpack Libraries (Compose, Navigation, ViewModel, Room), Coil for image loading.
- Design: A UI design tool like Figma or Adobe XD.
- Project Management: A Kanban board (Trello, Jira) for task tracking.
- Hardware: Android devices and emulators (API 24+).

3. Task Assignment & Roles

Name	Role	Description
Mohamed Ahmed Elsayed	Team leader & details feature	As the Team Leader, Mohamed was responsible for overseeing the project's progress and ensuring tasks were completed on time. He also developed the Product Details feature, allowing users to view detailed information about a specific product before adding it to their cart
Abdelrahman Elsayed Mahmoud	UI & App structure & home feature and Database	Abdelrahman designed the core application structure and the main user interface. He developed the Home Screen where products and categories are displayed and set up the local Room database to handle data persistence for features like the shopping cart.

Yasmine Mohamed Hassan	Categories feature And theme	Yasmine implemented the Categories feature, creating the screens where users can browse all product categories and view products belonging to a specific category. She also defined the app's visual Theme, ensuring a consistent and modern look and feel across the application.
Nour Ahmed Gamal	Checkout & payment feature	Nour developed the entire Checkout and Payment flow. This includes the simulated payment screen where users enter their details and the final success screen that confirms their order has been placed
Hana Abdelkader Attia	Cart feature	Hana and Jana worked together to build the Shopping Cart feature. They implemented the screen where users can view, manage, and remove items from their cart. They also integrated it with the local database to ensure the cart's contents are saved, and developed the logic for the cart icon badge that shows the number of items
Jana Haitham Mohamed	Cart feature	Hana and Jana worked together to build the Shopping Cart feature. They implemented the screen where users can view, manage, and remove items from their cart. They also integrated it with the local database to ensure the cart's contents are saved, and developed the logic for the cart icon badge that shows the number of items

4. KPIs (Key Performance Indicators)

- Response Time: API calls should ideally complete within 500ms; UI transitions should be fluid (60+ fps).
- System Uptime: Application should have a crash-free user rate of > 99.5%
- User Adoption Rate: Track the number of downloads and active users post-launch.
- Task Completion Rate: > 95% of users who add an item to the cart should be able to successfully reach the payment confirmation screen.
- Code Quality: Maintain a low rate of bugs found during QA cycles; ensure high unit test coverage (>70%).

5. Software Requirements Specification (SRS

5.1. Problem Statement & ObjectivesProblem:

Consumers often face cluttered and slow mobile shopping applications, leading to frustration and cart abandonment. There is a need for a streamlined, performant, and intuitive e-commerce app that simplifies the process of finding and buying products.Objectives:

- To solve this by creating "Quick Mart," an Android application focused on speed and simplicity.
- The system will provide a clean user interface, fast data loading through effective caching, and a straightforward navigation flow.
- The primary goal is to reduce the time and effort required for a user to go from launching the app to completing a purchase.

6.2. Use Case Diagram & Descriptions

1. Actors:

   Customer: The end-user of the application who browses and purchases products.

   Use Cases:

   - View Home Page: The Customer sees featured products and categories
   - Browse Products: The Customer views lists of products (all or by category).
   - View Product Details: The Customer selects a product to see its details.
   - Add to Cart: The Customer adds a product to their shopping cart from the details page.
   - Manage Cart: The Customer views their cart, can change quantities, or remove items.
   - Checkout: The Customer initiates the checkout process from the cart.

6.4. Software Architecture

The "Quick Mart" application is designed using the MVVM (Model-View-ViewModel) architecture pattern, layered for a clean separation of concerns. This is clearly visible in MainActivity.kt with the instantiation of HomeViewModel and CartViewModel.

- View (UI Layer): Composed of Jetpack Compose functions (HomeScreen, CartScreen, etc.) and Activities (MainActivity). This layer is responsible for displaying the data from the ViewModel and capturing user input (e.g., clicks). It is stateless and reacts to state changes from the ViewModel. The NavHost in MainActivity.kt manages the navigation between these composable screens.
- ViewModel (Presentation Layer): (HomeViewModel, CartViewModel). These classes act as a bridge between the UI and the Data layer. They fetch data from the Repository, hold the UI-related state (e.g., product list, cart items), and expose it to the UI, typically via StateFlow or LiveData. They are lifecycle-aware and survive configuration changes. HomeViewModelFactory and CartViewModelFactory are used for dependency injection.
- Model (Data Layer): This layer is the single source of truth for all application data. It is composed of:
- Repository (HomeRepositoryImp): Mediates between different data sources. It implements the logic for deciding whether to fetch fresh data from the network (RemoteDataSourceImp) or to serve cached data from the local database (LocalDataSourceImp).

Data Sources:

- Remote (RemoteDataSourceImp): Handles communication with the backend REST API using a library like Retrofit or Ktor.
- Local (LocalDataSourceImp): Manages the local Room database. It is responsible for caching product data and persisting the shopping cart.This architecture makes the app modular, scalable, and highly testable.

7. Database Design & Data Modeling

7.1. ER Diagram (Entity-Relationship Diagram

)Based on the application's functionality, the following entities and relationships can be inferred for the Room database.(A visual ER Diagram would be inserted here, showing the following entities and relationships)

•**Product:**

- id (Primary Key)
- name
- Description
- price
- imageUrl
- categoryId (Foreign Key to Category)

•**Category:**

- id (Primary Key)
- name
- imageUrl

CartItem: (Represents an item in the shopping cart)

- productId (Primary Key, Foreign Key to Product)
- quantity

Relationships:

- A Category has a one-to-many relationship with Product.
- A Product has a one-to-one relationship with a CartItem.

8. Data Flow & System Behavior

8.1. DFD

Level 0 (Context Diagram):

A single process, "Quick Mart App," interacts with one external entity, the "Customer," and has two main data interfaces: a "Product API" (remote) and "Device Storage" (local database).(A visual Context DFD would be inserted here)

Level 1 DFD (Example Flow: Adding an Item to Cart):

1. The Customer is on the ProductDetailsScreen and clicks the "Add to Cart" button.

2. The UI (ProductDetailsScreen) calls the addToCart() function on the CartViewModel.

3. The CartViewModel processes the request and calls a function on the LocalDataSourceImp (via a repository).

4. The LocalDataSourceImp performs a transaction on the Cart Data store (the cart_items Room table), either inserting a new row or updating the quantity of an existing one.

5. The CartViewModel updates its state, which is observed by the UI. The BadgedBox around the cart icon automatically recomposes to show the new count.

6. The Customer sees the updated badge count and a confirmation (e.g., a toast message or navigation to the cart).