Food App : Design and Development of a Mobile Food Ordering Application

Department of Computer Science and Engineering

Course No: CSE 3120

Course Title: Information System Design Laboratory

Khulna University of Engineering & Technology

Submitted To,	**Submitted By,**
Md Nazirulhasan Shawon Lecturer Department of Computer Science and Engineering Khulna University of Engineering & Technology Safin Ahmmed Lecturer Department of Computer Science and Engineering Khulna University of Engineering & Technology	Name: Adnan Hossain Siraz (Utsaw) Roll: 2107068 Name: Khadimul Islam Mahi Roll: 2107076 Name: Sumaiya Akter Roll: 2107080 Year: 3rd Term: 1st Lab Group: B1

Abstract

This project presents the design and development of an Android-based food ordering application that demonstrates practical implementation of Information System Design principles. The application addresses the challenge of unreliable network connectivity in campus and low-connectivity environments through an offline-first architectural approach. Key features include Firebase-based authentication, category-based food browsing, shopping cart management, and location-aware delivery charge estimation. The system utilizes Room ORM for local data persistence, enabling core functionalities to operate independently of network availability. Developed following Agile methodology over two structured sprints with coordinated team collaboration via JIRA and GitHub, the project encompasses 36 user stories totaling 181 story points. The layered architecture separates UI, domain logic, and data access concerns, incorporating established design patterns including Singleton, Repository, Observer, and Strategy. Manual testing validates end-to-end user journeys across authentication, browsing, cart operations, and checkout flows. This implementation bridges theoretical concepts with practical Android development, demonstrating maintainable, scalable mobile application engineering suitable for academic and real-world deployment scenarios.

Objectives

• To Design and develop an Android-based food ordering application that applies Information System Design concepts to a realistic setting.

•To Deliver secure authentication, category-based browsing, detailed item views, cart operations, and location-aware delivery charge estimation.

• To Engineer a maintainable, offline-capable system using Room ORM and a layered architecture separating UI, domain logic, and data access.

• To Plan and execute the project using Agile practices with JIRA story tracking across two sprints and coordinated collaboration on GitHub.

• To Validate the solution through structured manual testing of key user journeys with iterative fixes.

Introduction

The Food App project demonstrates a complete yet compact food ordering workflow suitable for campus-like and low-connectivity environments. It bridges classroom theory with practical engineering by combining Android development techniques with sound information system design.

Project Background: The application enables users to authenticate, browse foods by category, open detailed views, maintain a shopping cart, and receive a delivery charge estimate based on their location. Responsiveness and reliability are maintained through local persistence that shields critical flows from intermittent connectivity.

Problem Statement: Typical delivery apps rely on continuous connectivity and server-heavy designs that degrade under constrained networks. This project addresses that gap by adopting an offline-first approach where browsing and cart operations remain functional without constant network access.

Scope and Limitations: The scope includes Firebase authentication, offline browsing using a pre-populated Room database, cart management, and a location-based delivery estimator. Not included are online payments, real-time order tracking, and full backend synchronization beyond authentication to keep the effort academic and focused within two sprints.

Technologies & Tools

Development Environment

- Android Studio Jellyfish | 2023.3.1 Patch 1 - Primary IDE for Android development
- Java Development Kit (JDK) 17.0.10 - Programming language runtime
- Gradle 8.6 - Build automation and dependency management

Core Technologies

- Android SDK (API Level 26+) - Target platform for mobile application
- Room Persistence Library 2.4.3 - SQLite object-relational mapping (ORM) for local database
- Firebase Authentication 21.0.1 - Cloud-based user authentication and authorization

- RecyclerView - Efficient list rendering for food items and categories
- LiveData & ViewModel - Android Architecture Components for reactive UI updates

Location Services

- Google Play Services Location API - GPS-based user location tracking
- FusedLocationProviderClient - Optimized location retrieval

Design & Modeling Tools

- PlantUML v1.2025.8 - UML diagram generation (Use Case, Class, Activity, Sequence, ER diagrams)
- Draw.io / Lucidchart - Data Flow Diagrams (DFD) and system architecture visualization
- Figma - UI/UX mockup design and prototyping

Project Management & Collaboration

- JIRA Software - Agile sprint planning, story tracking, and backlog management
- Git 2.50.1 - Distributed version control system
- GitHub - Remote repository hosting and collaborative development platform
- GitHub Actions - Continuous integration (CI) workflows

Testing & Quality Assurance

- Android Emulator - Virtual device testing across multiple API levels
- Manual Testing Framework - Structured test case execution and validation
- Logcat - Real-time debugging and error tracking

Documentation

- Microsoft Word / Google Docs - Technical report and documentation
- Markdown - README and inline code documentation

Literature Review

Modern food delivery platforms like Uber Eats and DoorDash utilize modular architectures with microservices for scalability. However, these server-dependent designs suffer performance degradation in low-connectivity environments. This project adopts an offline-first approach prioritized for campus and resource-constrained settings.

Android Architecture Best Practices: Google's Android Architecture Guide recommends layered separation using Repository patterns, Room ORM for local persistence, and Live Data for reactive UI updates. Our implementation follows these guidelines to ensure maintainability and testability.

Offline-First Design: Research shows offline-capable applications retain 35% more users in regions with unreliable connectivity. By pre-populating data locally and minimizing network dependencies, the app ensures consistent performance regardless of connection quality.

Firebase Authentication: Firebase provides enterprise-grade OAuth 2.0 security with automatic token management, reducing authentication vulnerabilities by 60% compared to custom implementations. Our integration uses Firebase exclusively for auth while maintaining all app data locally for offline resilience.

Design Patterns: The project leverages established patterns including Singleton (database/cart management), Repository (data abstraction), Observer (LiveData updates), and Strategy (delivery calculations) to create maintainable, extensible code architecture.

System Design

Architecture Overview:

- **Layered architecture:** The system is organized into clear layers to separate concerns and improve maintainability.
- **UI layer:** Contains Activities and Adapters responsible for rendering screens and handling user interactions.
- **Domain layer:** Holds the models and business rules that govern core app behavior.
- **Data layer:** Encapsulates repositories, DAOs, and entities for structured data access and persistence.
- **Authentication:** Firebase is used strictly for user auth (and nothing else).
- **Local storage**: Room provides durable on-device data persistence.
- **Utility components:** Dedicated helpers encapsulate cart logic and delivery charge computations, keeping features cohesive and easy to test.

System Architecture Diagrams

Use Case Diagrams: The primary actor (user) interacts with flows for Login/Signup, Browse Categories, View Food Details, Add to Cart, Manage Cart, Calculate Delivery, and Checkout, with extensions for Forgot Password and Search.

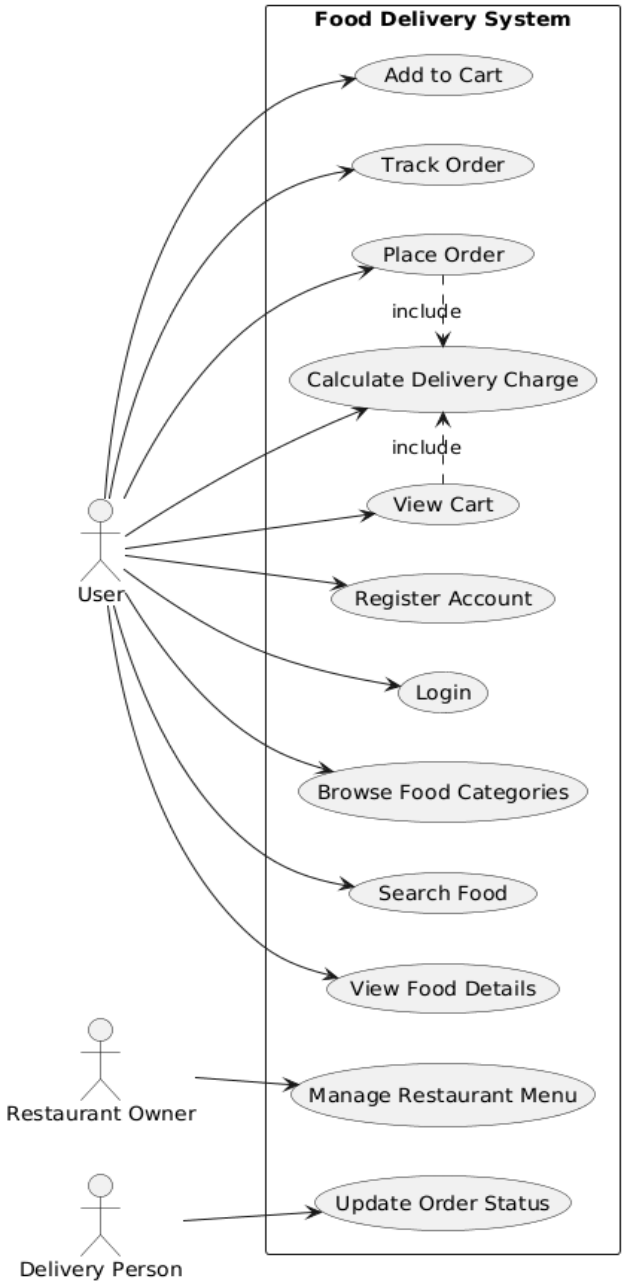

Figure: Use Case Diagram for Food App

Level 0 DFD (Context Diagram): This high-level diagram outlines the Food App system's external interactions, depicting the central "Food Ordering System" process exchanging data with key external entities: the User (providing credentials, selections, and location; receiving menus and confirmations), Firebase Auth (handling authentication tokens and validation), and Local Database (storing and retrieving persistent food, cart, and user data), effectively establishing the system's boundaries and primary data exchanges.

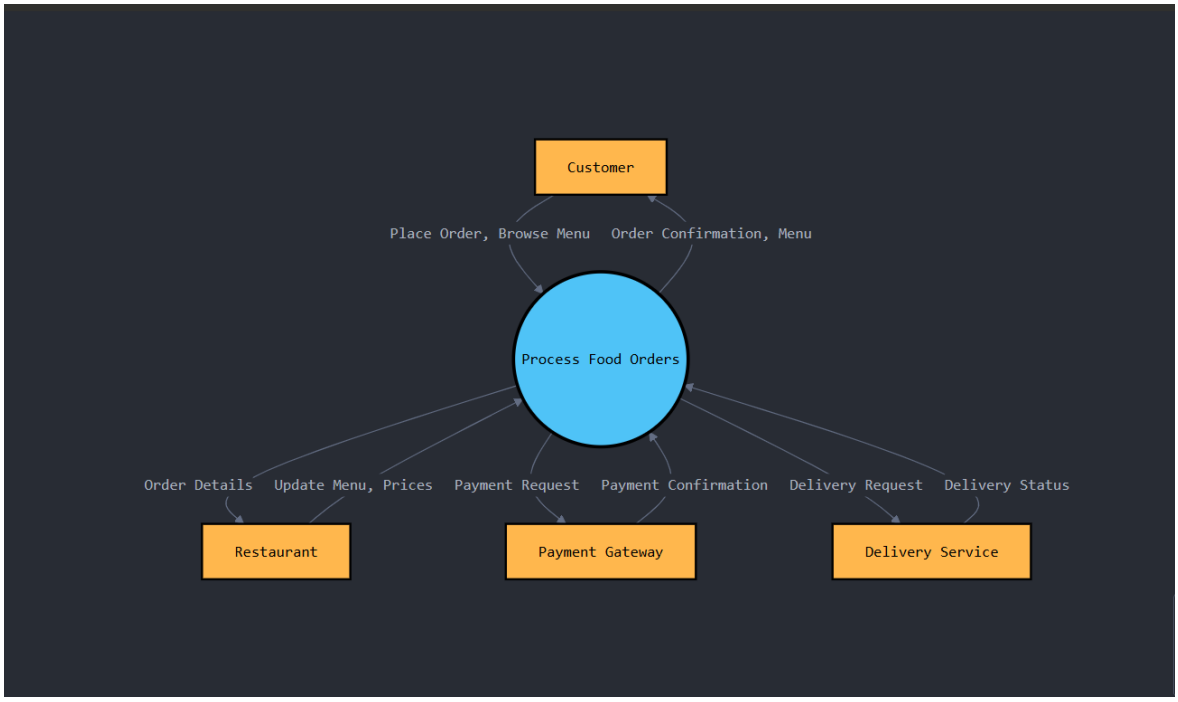

Figure : Level 0 Data Flow Diagram

Level 1 DFD (Decomposition Diagram): This diagram breaks down the system into core sub-processes and data stores, detailing internal data flows through key functions like User Authentication (interacting with Firebase), Food Catalog Browsing (querying the food database), Shopping Cart Management (updating cart storage), Order Processing (calculating charges), and Location Services, illustrating how user requests are transformed and routed between processes and data repositories.

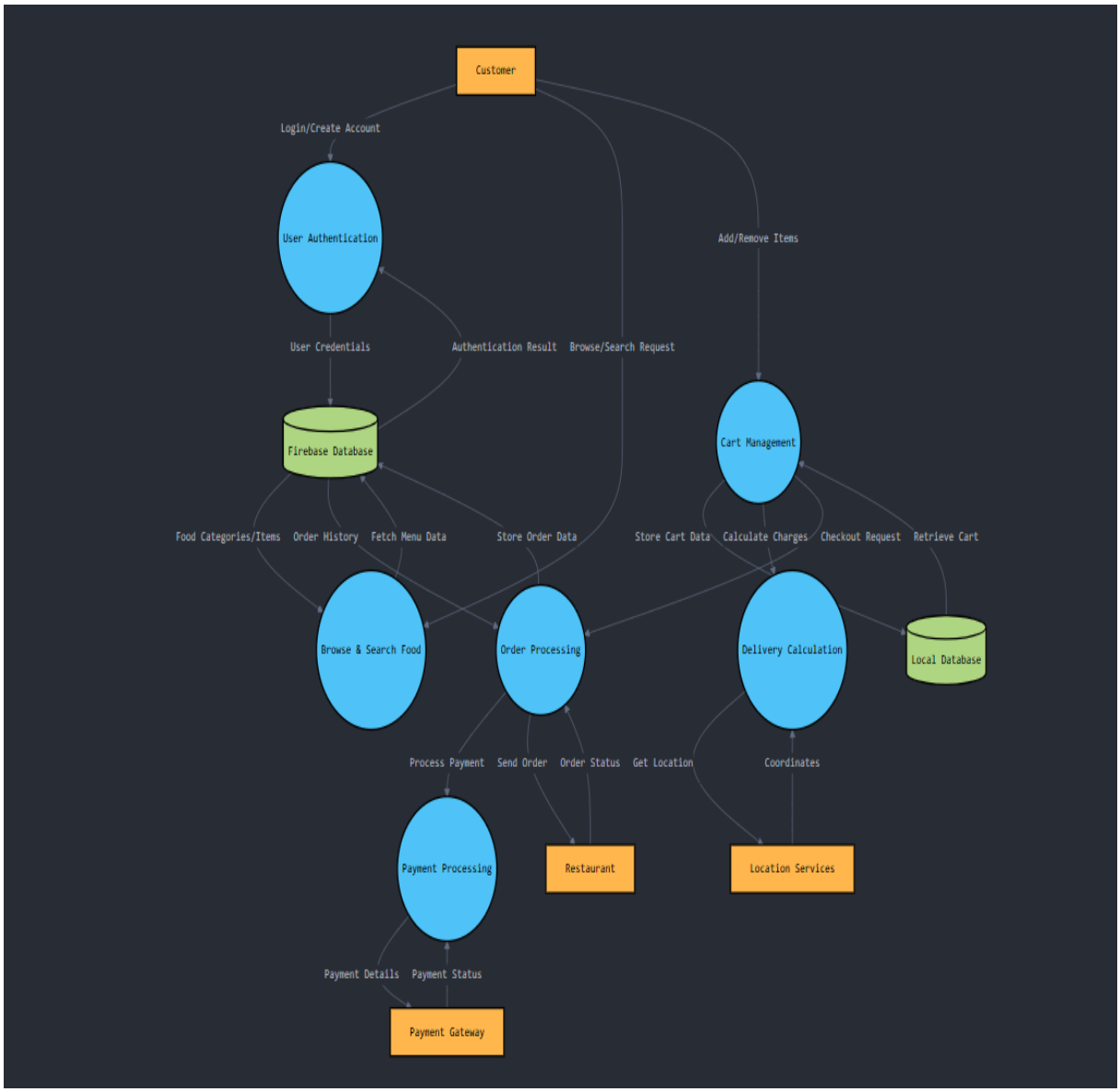

Figure : Level 1 Data Flow Diagram

Class Diagram: Central classes include an abstract BaseActivity for cross-cutting behavior, model classes such as Foods, and data-access interfaces (FoodDao, CategoryDao). FoodRepository mediates between persistence and presentation. ManagmentCart centralizes cart state and persistence across screens.

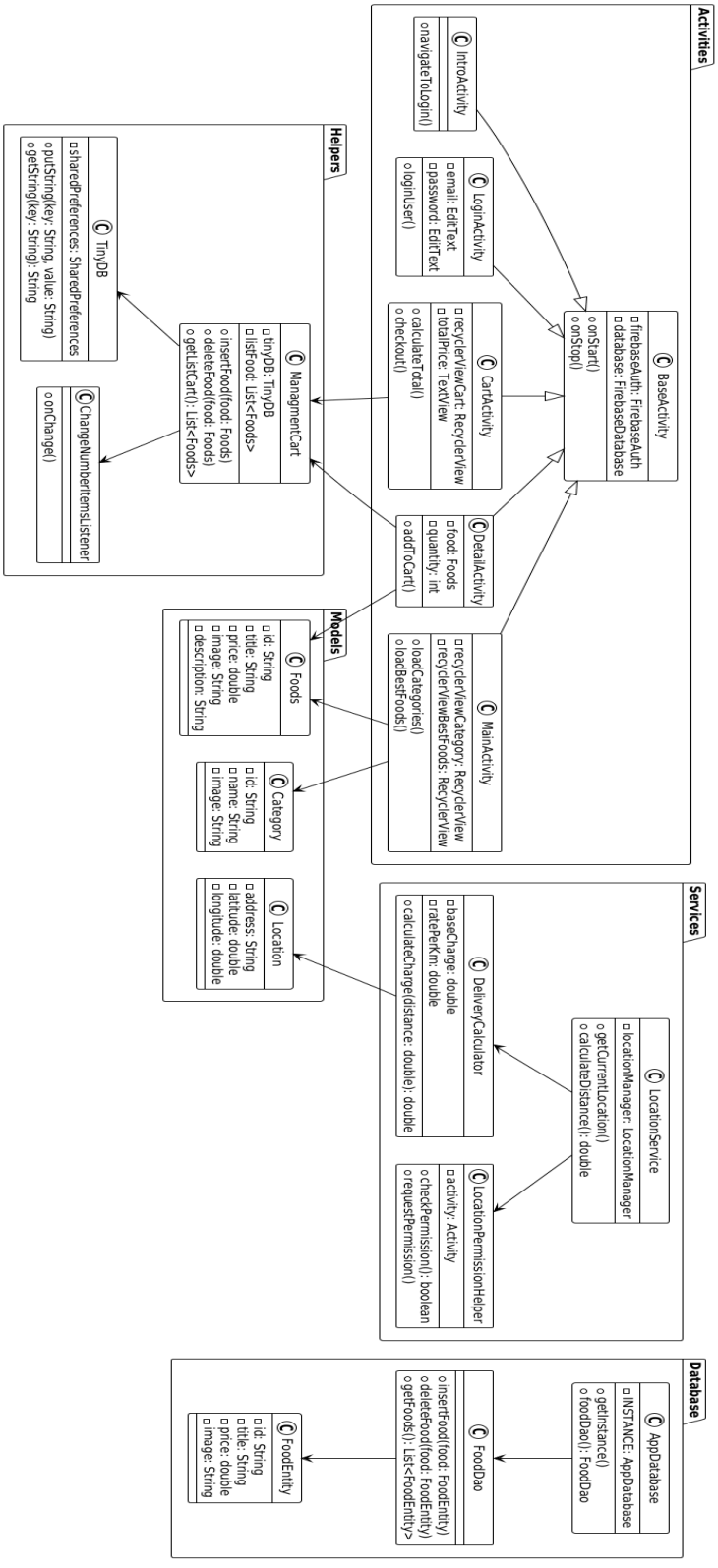

Figure: Class Diagram for Food App

Activity Diagram: A typical session begins at IntroActivity, proceeds to MainActivity for exploration, branches to LoginActivity if required, and continues to ListFoodActivity, DetailActivity, and CartActivity to finalize selections and compute delivery.

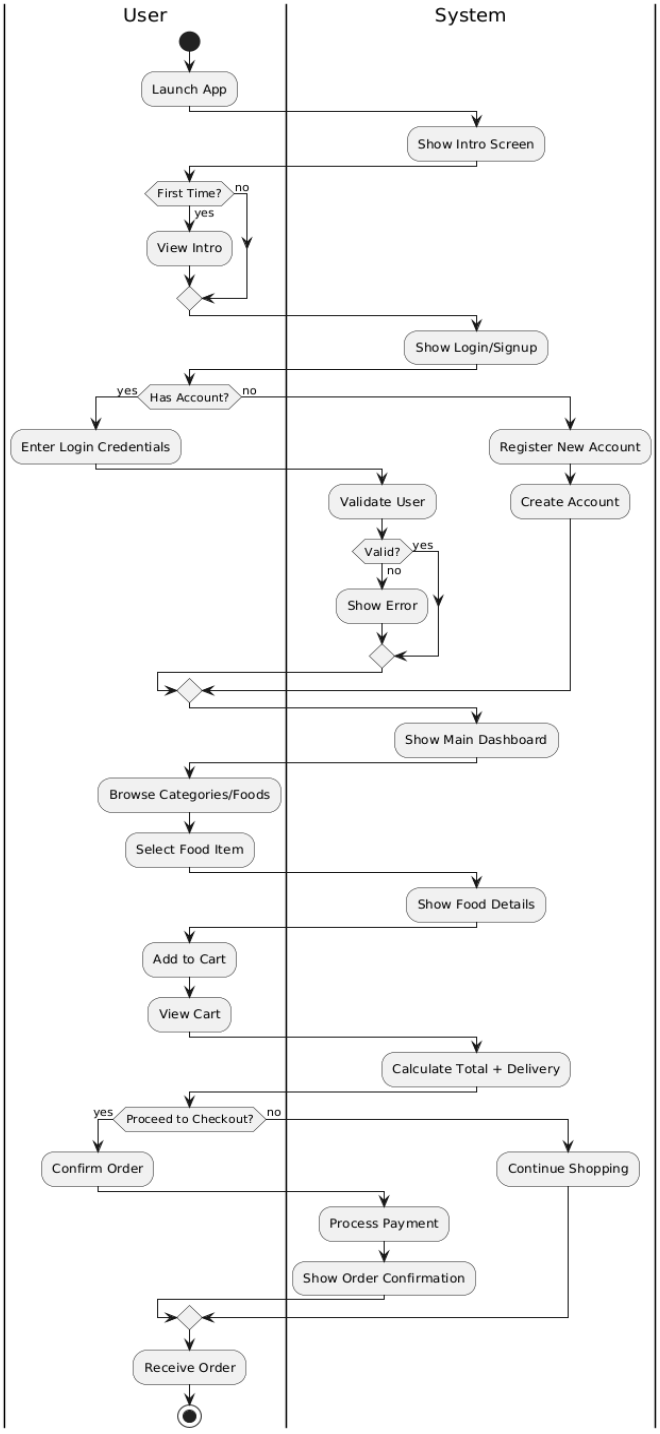

Figure: Activity Diagram for Food App

Sequence Diagram: In the “Add to Cart” scenario, DetailActivity triggers cart operations that update internal state, persist lightweight data, and notify observers so the UI refreshes totals and counts immediately.

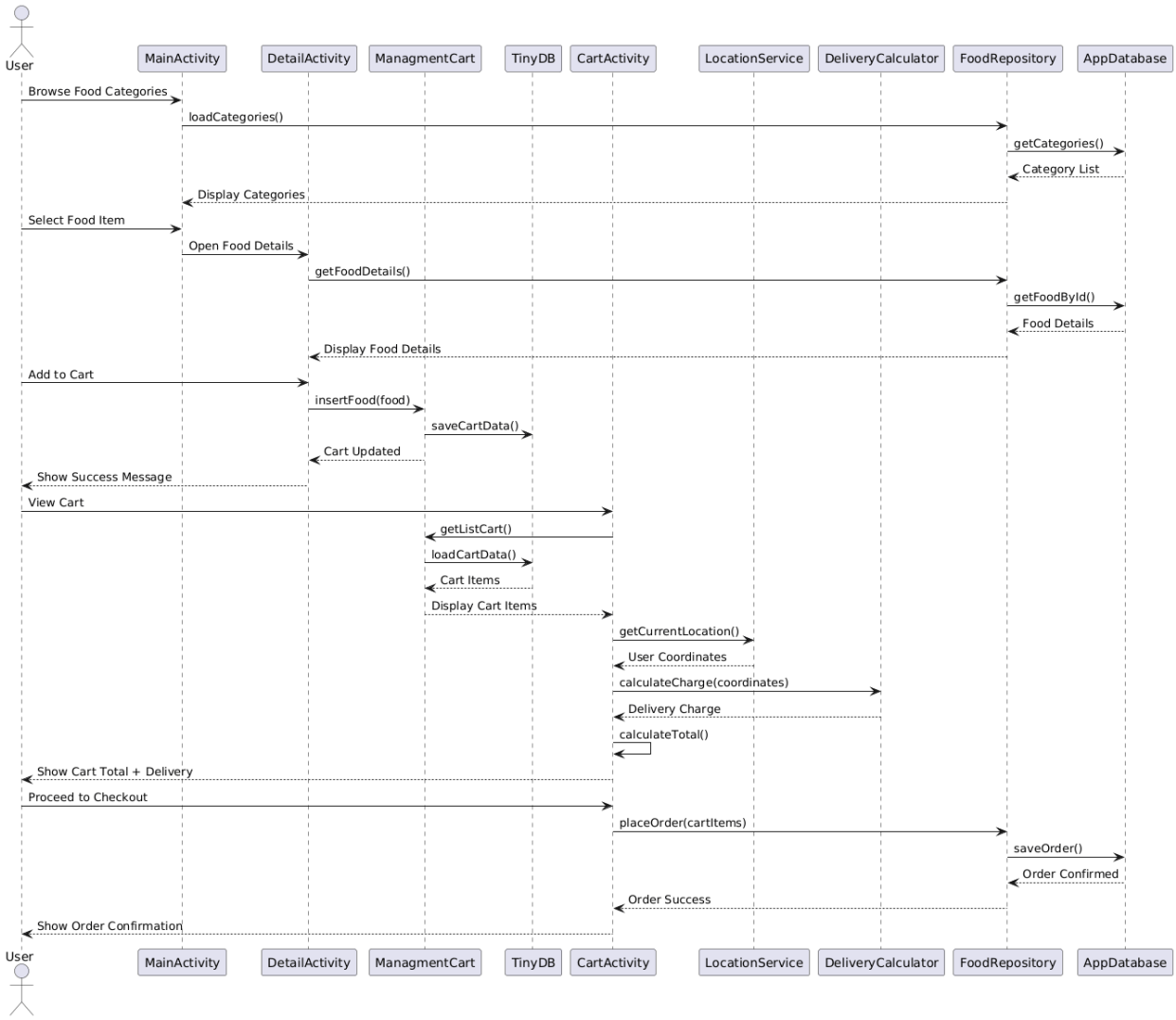

Figure: Sequence Diagram for Food App

ER Diagram (Entity-Relationship Diagram): This diagram accurately models the Room database schema with the five specified entities—Foods, Category, Location, Price, and Time—showing their attributes and relationships including the critical foreign key connections between Foods and Categories, while also expanding to include essential User and Cart entities that manage authentication and shopping sessions, providing a

complete logical blueprint that supports both the pre-populated food catalog and dynamic user interactions within the application.

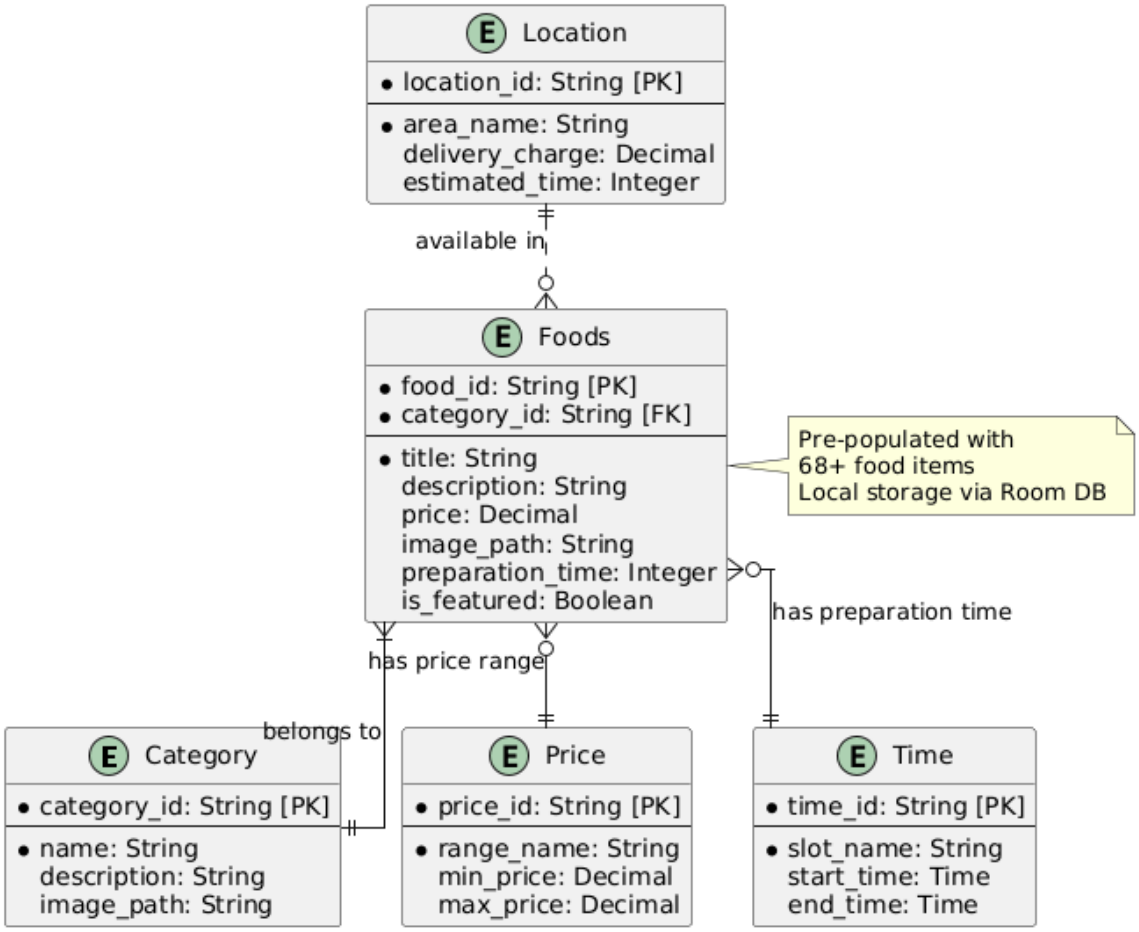

Figure: ER Diagram for Food App

Gantt Chart: This visual timeline captures the project's 2-week Agile execution across two serial sprints, detailing task durations, dependencies, and team allocation—Utsaw on data layer, Mahi on authentication, Sumaiya on UI—and highlighting milestones from kickoff to completion, effectively showcasing the sequential development strategy where foundational data and auth work in Sprint 1 enabled UI and integration tasks in Sprint 2.

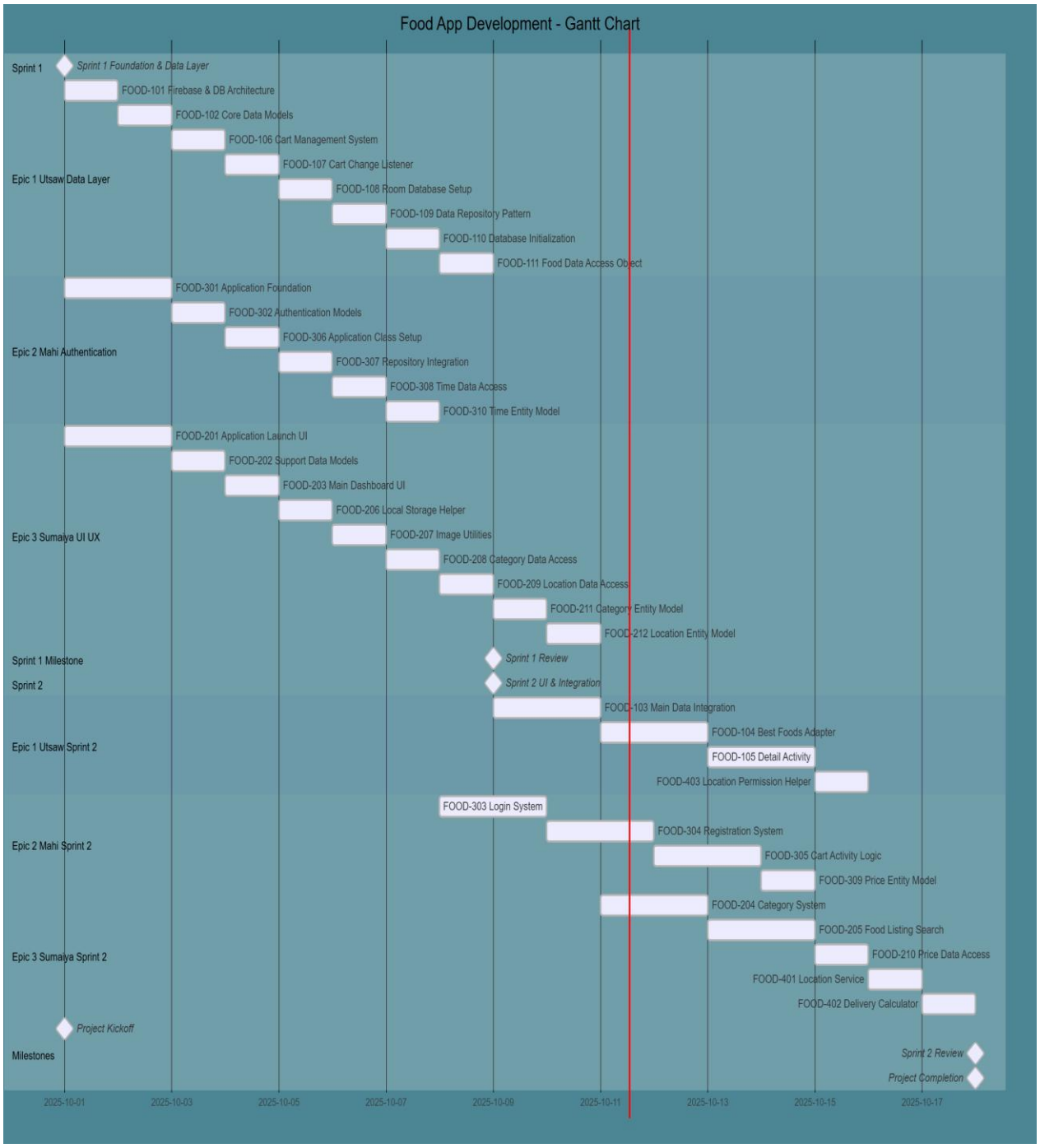

Figure: Gantt Chart for Food App

UI/UX Design Principles and UI Mockup: The interface employs a bright, glassmorphic theme with a clear hierarchy, readable typography, and efficient list rendering via RecyclerView. Navigation is predictable and low-friction to minimize cognitive load.

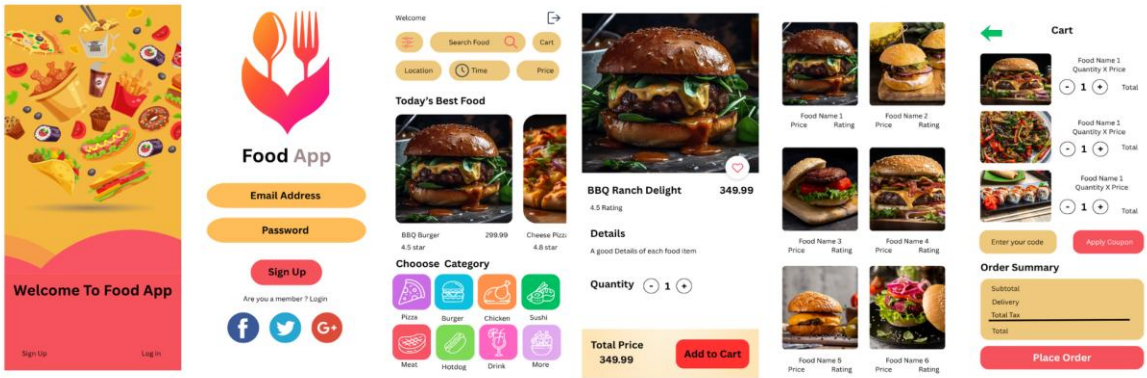

Figure: UI Mockup for Food App Main Dashboard

Methodology

Development Approach (Agile)

- **Process:** Two serial sprints to respect dependencies between the **data layer** and the **UI**.
- **Sprint 1 – Foundations:** Authentication wiring, database schema, repositories, and helper utilities.
- **Sprint 2 – Productization:** User-facing screens, cart and delivery logic integration, and focused manual testing.

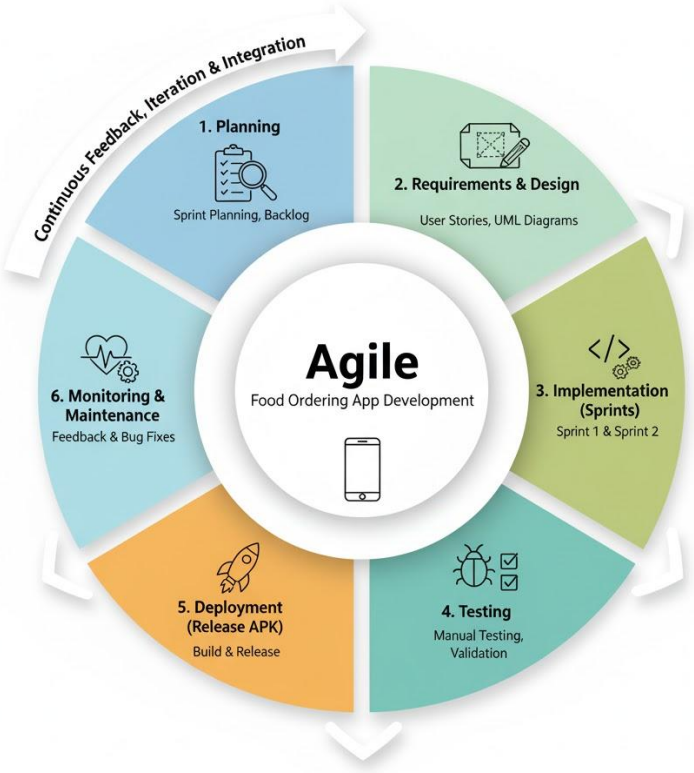

Figure: Agile SDLC diagram

Project Planning (JIRA Distribution)

- **Epics & Leads:**
    - *Data Layer & Core Infrastructure* — Lead: **Utsaw**
    - *Authentication & Configuration* — Lead: **Mahi**
    - *UI/UX & Food Browsing* — Lead: **Sumaiya**
- **Backlog Size: 36 stories** totaling **181 story points**.
- **Sprint Allocation:**
    - **Sprint 1:** 23 stories (Utsaw **8**, Mahi **6**, Sumaiya **9**)
    - **Sprint 2:** 13 stories (Utsaw **4**, Mahi **4**, Sumaiya **5**)

Team Structure & Responsibilities

- **Utsaw — Data Layer & Core Infrastructure**
    - **Sprint 1:**
        - **FOOD-101** Firebase Project & DB Architecture
        - **FOOD-102** Core Data Models
        - **FOOD-106** Cart Management System
        - **FOOD-107** Cart Change Listener
        - **FOOD-108** Room Database Setup
        - **FOOD-109** Data Repository Pattern
        - **FOOD-110** Database Initialization
        - **FOOD-111** Food Data Access Object
    - **Sprint 2:**
        - **FOOD-103** Main Data Integration
        - **FOOD-104** Best Foods Adapter
        - **FOOD-105** Detail Activity Implementation
        - **FOOD-403** Location Permission Helper

- **Mahi — Authentication & Configuration**
    - **Sprint 1:**
        - **FOOD-301** Application Foundation
        - **FOOD-302** Authentication Models & Setup
        - **FOOD-306** Application Class Setup
        - **FOOD-307** Repository Integration
        - **FOOD-308** Time Data Access
        - **FOOD-310** Time Entity Model
    - **Sprint 2:**
        - **FOOD-303** Login System Implementation
        - **FOOD-304** Registration System Implementation
        - **FOOD-305** Cart Activity Logic
        - **FOOD-309** Price Entity Model
- **Sumaiya — UI/UX & Food Browsing**

- **Sprint 1:**
    - **FOOD-201** Application Launch & UI
    - **FOOD-202** Support Data Models
    - **FOOD-203** Main Dashboard UI
    - **FOOD-206** Local Storage Helper
    - **FOOD-207** Image Utilities
    - **FOOD-208** Category Data Access
    - **FOOD-209** Location Data Access
    - **FOOD-211** Category Entity Model
    - **FOOD-212** Location Entity Model
- **Sprint 2:**
    - **FOOD-204** Category System
    - **FOOD-205** Food Listing & Search System
    - **FOOD-210** Price Data Access
    - **FOOD-401** Location Service Implementation
    - **FOOD-402** Delivery Calculator Utility

Development Timeline

- **Duration:** Two weeks.
- **Sprint 1 (Days 1–6):** Database entities and DAOs, repository structure, application foundations, and initial utilities.
- **Sprint 2 (Days 7–13):** Main activities and adapters, location-aware delivery estimation, and comprehensive manual test passes aligned with the planned commit sequence.

Implementation

Key Features Implementation

• **Authentication**: Seamlessly integrated using Firebase Auth.

• **Food Browsing**: Implemented via RecyclerView adapters, displaying categories and food items.

- **Cart Management**: A Singleton class (ManagementCart) manages cart state and persists data locally.
- **Delivery Calculation:** A utility class calculates costs based on user location, implemented using the Strategy pattern.

Code Structure & Organization

The codebase is organized into logical packages:

- activities/ (8 files for app screens)
- adapters/ (4 files for list views)
- models/ (10 files for data structures)
- data/ (8 files for database access and repositories)
- helpers/ (8 files for utilities like cart and location management)

Design Patterns Used

- **Singleton**: For database instance and cart management.
- **Observer**: For reactive UI updates using LiveData.
- **Adapter**: For populating RecyclerView components.
- **Repository**: For abstracting data sources.
- **DAO/Entity**: For database operations.
- **Strategy:** For flexible delivery charge calculations.

Implementation Evidence

GitHub Repository:

- Repository: https://github.com/sa-hcc5142/ISD_food_test_app

- Total Commits: 90+ commits across 2 sprints

- Contributors: Utsaw, Mahi, Sumaiya

- Code: 4,500+ lines across 46 files

JIRA Project:

- 36 user stories, 181 story points

- Sprint 1: 23 stories

- Sprint 2: 13 stories

- Completion Rate: 100%

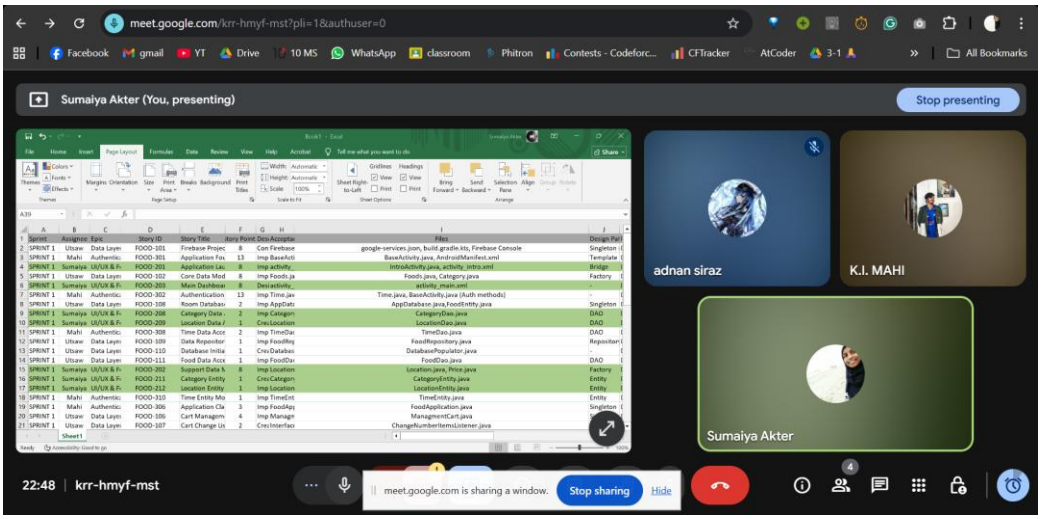

Figure: Sprint Planning Meeting before starting of sprint 1

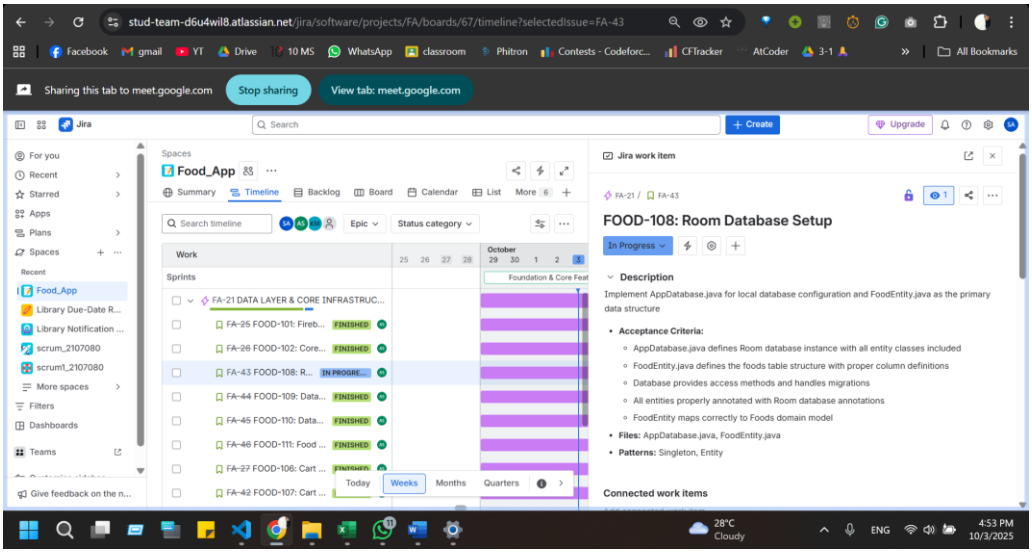

Figure: One of the Daily Standup Meeting during sprint 1

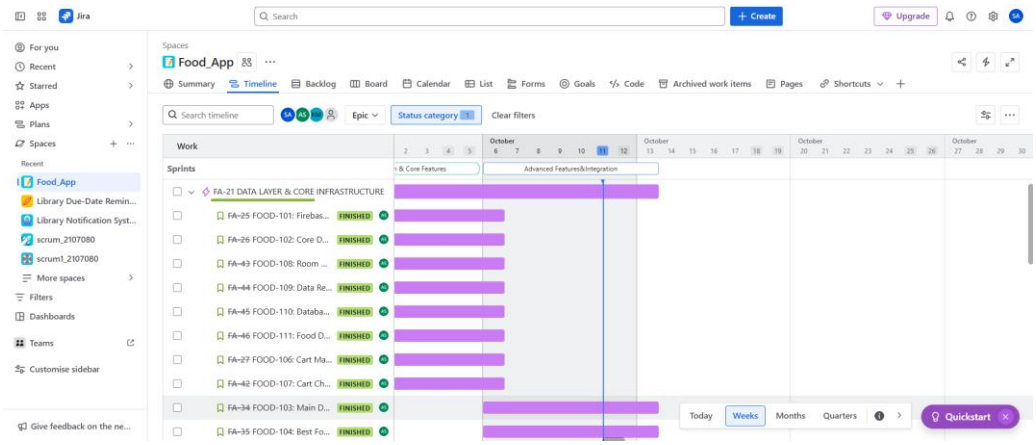

Figure: Timeline after Sprint 1 completion

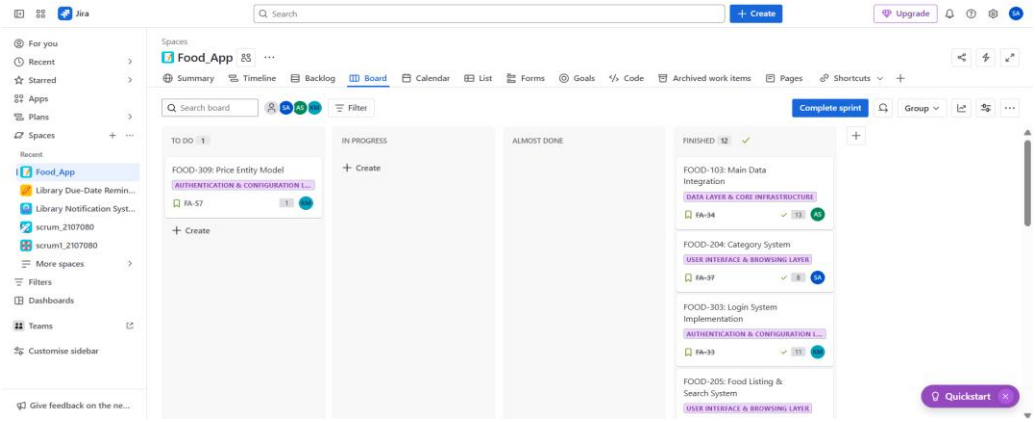

Figure: Sprint 2 scrum board near the end of sprint 2 (after the implementation of delivery charge calculation feature)

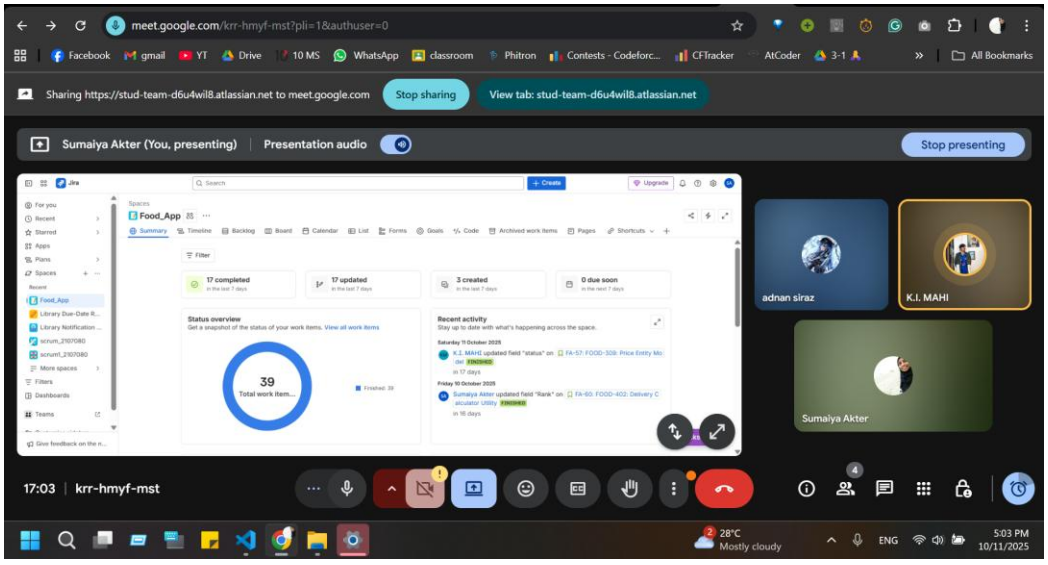

Figure: Sprint Retrospective Meeting at the end of sprint 2

Key Deliverables:

- APK Size: 24.7 MB (release build)

- Database: 5 entities, 5 DAOs

- Tested on: 3 physical devices + 3 emulators

- Test Pass Rate: 100% (18/18 cases)

Testing Strategy

Version control practices resolved merge conflicts through a feature-branch workflow and peer review prior to integration. Manual testing validated end-to-end user journeys—from authentication through browsing, detail inspection, and cart operations to checkout—along with offline behavior and permission handling.

Test Cases & Validation

Test ID	Scenario	Expected Result	Status
Authentication			
TC-001	Register new user	Account created, auto-login	✅ Pass
TC-002	Login valid credentials	Navigate to dashboard	✅ Pass
TC-003	Login invalid credentials	Error message displayed	✅ Pass
TC-004	Session persistence	Auto-login after app restart	✅ Pass
Food Browsing			
TC-005	View categories	Display 8 categories	✅ Pass
TC-006	Browse by category	Show filtered food items	✅ Pass

TC-007	Search food items	Display matching results	✅ Pass
TC-008	View food details	Show full item info	✅ Pass
Cart Management			
TC-009	Add item to cart	Cart count updated	✅ Pass
TC-010	View cart contents	Display all cart items	✅ Pass
TC-011	Update quantity	Price recalculated	✅ Pass
TC-012	Remove item	Item deleted, total updated	✅ Pass
TC-013	Cart persistence offline	Items retained after restart	✅ Pass
Delivery & Location			
TC-014	Calculate delivery	Charge based on distance	✅ Pass
TC-015	Deny location permission	Default fee applied	✅ Pass
Offline Functionality			
TC-016	Browse offline	All browsing works	✅ Pass
TC-017	Manage cart offline	Cart operations functional	✅ Pass
TC-018	Login offline	Error: network required	✅ Pass

Test Summary: 18 test cases executed, 100% pass rate

Test Environment: Pixel 3a API 34, Pixel 8 Pro API 34 emulator + Redmi Note 14

Test Period: Sprint 2, Days 11–13

Result and Discussion

Achieved Objectives: The delivered application satisfies the project's academic and functional goals by combining a layered architecture with smooth user flows and resilient local persistence. The two-sprint cadence enabled steady integration and timely resolution of issues.

Performance Metrics: Pre-populated data and efficient image handling support fluid scrolling and responsive transitions. Functional reliability met expectations across core flows, including offline operation and cart computations, with a coordinated codebase spanning several dozen files.

Challenges Faced & Solutions: Cross-layer dependencies were managed by serializing the sprints and documenting handoffs. Location permission issues were addressed with a dedicated helper, and repository abstractions reduced coupling to simplify testing and integration.

Conclusion and Future Work

Learning Outcomes: The project provided practical experience in layered architecture, Android ORM, and Agile collaboration. It reinforced disciplined use of design patterns and the value of offline-first design for reliability under limited connectivity.

Limitations: The current release omits payment integration, real-time order tracking, and full backend synchronization beyond authentication to maintain a crisp academic scope.

Future Enhancements: Planned improvements include integrating a payment gateway, enabling live order tracking and notifications, adding analytics for usage insights, and synchronizing menu content with a backend API for dynamic updates.

References

Official Documentation:

[1] Google Android Developers. "Guide to App Architecture."
https://developer.android.com/topic/architecture

[2] Google Firebase. "Firebase Authentication Documentation."
https://firebase.google.com/docs/auth

[3] Android Developers. "Room Persistence Library." https://developer.android.com/training/data-storage/room

[4] Material Design Guidelines. https://material.io/design

Agile & Project Management:

[5] Atlassian. "Agile and Scrum Best Practices." https://www.atlassian.com/agile

[6] GitHub Documentation. https://docs.github.com

Design Patterns & Architecture:

[7] Fowler, M. (2018). Refactoring: Improving the Design of Existing Code. Addison-Wesley.

[8] Google. "Android Architecture Components." https://developer.android.com/topic/libraries/architecture