CEBU INSTITUTE OF TECHNOLOGY
UNIVERSITY

IT342-G4
System Integration and Architecture

System Design Document (SDD)

Project Title: UniGear Tracker

Prepared By: Ybañez, Liezel, A.

Version: 0.1

Date: February 02, 2026
Status: Draft

REVISION HISTORY TABLE

Version	Date	Author	Changes Made	Status
0.1	02/02/26	Ybañez, Liezel, A.	Initial draft	Draft
0.2	[Date]	[Your Name]	Added API specifications	Review
0.3	[Date]	[Your Name]	Updated database design	Review
0.4	[Date]	[Your Name]	Added UI/UX designs	Review
0.5	[Date]	[Your Name]	Incorporated feedback	Revised
0.6	[Date]	[Your Name]	Final review and corrections	Final
1	[Date]	[Your Name]	Baseline version for development	Approved

TABLE OF CONTENTS

Contents

EXECUTIVE SUMMARY

1.1 Project Overview

UniGear Tracker is a generalized inventory and borrowing system in the university that lets students borrow and reserve items and equipment from the university offices and laboratory that they need for events and academic purposes. The system includes a Spring Boot backend API, a React web application, and an Android mobile app, all integrated to provide a seamless transaction experience across platforms.

1.2 Objectives

1. To develop a fully functional borrowing system with secure user authentication, role-based access control (Student, Admin), and a core borrowing module.
2. To implement a three-tier architecture using Spring Boot (backend), React (web), and an Android Kotlin application (mobile) that connects to the same backend.
3. To create RESTful APIs for communication between all system components, secured with JWT.
4. To integrate mandatory system features, including Google OAuth login, an external public API, file uploads, a sandbox payment gateway, and email notifications.
5. To design a responsive user interface for the web and a native Android mobile application using XML-based layouts that adhere to specified UI/UX standards.
6. To deploy all system components to a production-ready environment for final testing and submission.

1.3 Scope

Included Features:

- User registration and email/password login.
- Google OAuth 2.0 login, with user data saved in the database and generation of a internal JWT.
- JWT-based authentication and authorization for all protected endpoints and UI routes.
- Role-Based Access Control (RBAC) with at least two roles (e.g., Student, Admin), enforced at both the API and UI level.
- A core business module for managing borrow requests, including full CRUD operations and validation.
- Integration of a real, public external API, with data displayed in a meaningful feature within the application.
- File upload functionality (e.g., for project proposals or receipts), with files stored on the server and linked to a database record.

- Sandbox payment gateway integration for processing related fees (e.g., late fees, reservation deposits), with results recorded in the database.
- SMTP email sending for account-related (e.g., welcome email) and system notification (e.g., borrow request approval) purposes.
- A native Android mobile application built with Kotlin and XML layouts (Android API Level 34) that consumes the same backend API.
- A web application built with React that consumes the same backend API.
- PostgreSQL database with a minimum of five tables, proper relationships, and normalization.

Excluded Features:

- Real-time features using WebSockets or polling (optional bonus).
- Advanced analytics dashboards.
- Social media login options other than Google OAuth.
- Multi-language support.
- A forgot password flow (if not required by project scope).

1.0 INTRODUCTION

1.1 Purpose

This document serves as the comprehensive design specification for the UniGear Tracker system. It provides a detailed blueprint, including functional and non-functional requirements, architectural decisions, API contracts, database design, UI/UX wireframes, and an implementation roadmap. The purpose of this document is to guide the development team, ensuring all project requirements are met and all system components are seamlessly integrated.

2.0 FUNCTIONAL REQUIREMENTS SPECIFICATION

2.1 Project Overview

Project Name: UniGear Tracker
Domain: University Inventory & Borrowing Management
Primary Users:

1. Students: Borrowers who can browse, request, and track items.
2. Administrators: Office/Lab staff who manage items and process borrow requests.

Problem Statement: University students and staff lack a centralized, efficient system to locate, request, and manage the borrowing of equipment from various departments, leading to manual processes, miscommunication, and underutilized resources.

Solution: A unified platform that digitizes the borrowing process, providing transparency on item availability, streamlined request workflows, and a clear audit trail for all transactions.

2.2 Core User Journeys

Journey 1: Student Borrower - First-time Request

1. Student registers for an account using email/password or Google OAuth.
2. Student logs in and is redirected to the homepage.
3. Browses available items in the catalog, using filters (by category, office) or search.
4. Clicks on an item to view details, specifications, and availability calendar.
5. Submits a borrow request for the desired item, date range, and purpose.
6. Receives an email notification confirming the request submission.
7. Later, receives an email notification that the request has been approved.
8. Logs in to view the updated status and prepares to pick up the item.

Journey 2: Student Borrower - Returning User Checking Status

1. Student logs in using Google OAuth.
2. Navigates to their dashboard to view the status of their active and past borrows.
3. Views the details of an ongoing borrow, including the due date.
4. If the item is returned, the status updates to "Completed" (updated by admin).
5. The student can view their borrowing history at any time.

Journey 3: Administrator - Processing a Request

1. Admin logs in with credentials (role-based UI shows admin dashboard).
2. Views a list of pending borrow requests.
3. Clicks on a request to see full details, including any uploaded files from the student.
4. Approves the request, confirming item availability.
5. The system automatically updates the item's status and sends an approval email notification to the student.
6. When the item is returned, the admin marks the borrow as "Completed" in the system.

2.3 Feature List (MoSCoW)

MUST HAVE

1. Authentication & Security: User registration, login, logout, JWT, password hashing (BCrypt), protected routes, /me endpoint.

2. Role-Based Access Control: Minimum 2 roles (Student, Admin) with API and UI restrictions.
3. Core Business Module (Borrow Requests): Full CRUD operations with proper validation.
4. External API Integration: Consume a real public API in a meaningful feature.
5. Google OAuth Login: Implement social login, link/create user in DB, generate internal JWT.
6. File Upload: Allow users to upload files (e.g., requirements, project proposals), store on server, link to DB record, and provide view/download.
7. Email Sending (SMTP): Send at least one account-related email (e.g., welcome) and one system notification email (e.g., request status update).
8. Mobile Application: Android app (Kotlin, XML, API Level 34) connecting to the same backend, with role-aware UI.

SHOULD HAVE

1. Item availability calendar view.
2. Search and filter functionality for items.
3. Basic user dashboard showing active and past borrows.
4. Input validation and user-friendly error messages.

COULD HAVE

1. Item categorization (e.g., by lab, by type).
2. Email receipt for request confirmation.
3. In-app notification center.

WON'T HAVE

1. Payment gateway integration (the service is free).
2. Real-time features (WebSockets/Polling) - Optional Bonus.
3. Advanced analytics dashboard.
4. Social login other than Google.
5. Multi-language support.

2.4 Detailed Feature Specifications

Feature: User Authentication

- **Screens**: Registration (Email), Login (Email + Google Button), Post-Login Redirect.
- **Fields:** Email, Password, Name, Google Account.
- **Validation:** Email format, password strength (min 8 chars), uniqueness.

- **API Endpoints:** POST /auth/register, POST /auth/login, POST /auth/oauth2/google, POST /auth/logout, GET /users/me.
- **Security**: JWT tokens, password hashing with BCrypt. Google OAuth2 flow.

Feature: Core Module - Borrow Requests

- **Screens:** New Request Form, Request List (Student Dashboard), Request Detail, Request Management (Admin).
- **Entities:** Item, BorrowRequest, User**.**
- **Functions:** Student can create, view, and cancel their own requests. Admin can view all requests, update their status (Approve/Reject/Complete), and delete (or archive) requests.
- **Validation:** Request end date must be after start date; item must be available for selected dates.
- **API Endpoints:** GET /requests, POST /requests, GET /requests/{id}, PUT /requests/{id}/status (Admin), DELETE /requests/{id} (Admin/Cancel).

Feature: External API Integration

- **Feature: Enhance item details**. When viewing an item, the system could fetch additional information from a public API. *Example: For a "Digital Camera," the system could call a public API (e.g., The Movie Database API or a generic product info API) to display related tutorials or popular uses.*
- **Data Display:** Fetched data (e.g., links to tutorials, related items) is displayed in a dedicated section on the item detail page.
- **API Endpoints:** Backend service calls external API on behalf of the frontend.

Feature: File Upload

- **Screens:** Borrow Request Form (for students), Borrow Request Detail (for admin).
- **Function:** Students can upload supporting documents (e.g., faculty approval letter, project proposal) when submitting a borrow request.
- **Storage:** File saved to a server directory with a unique name; file path stored in the borrow_request table.
- **API Endpoint**: POST /requests/{id}/documents (multipart/form-data), GET /requests/{id}/documents/{docId}.

2.5 Acceptance Criteria

AC-1: Successful User Registration

➢ Given I am a new user

- When I am on the login page and click "Login with Google"
- And I select my Google account and grant permissions
- Then I should be redirected back to the application
- And a new user record should be created in the database (or linked if existing)
- And I should be issued a JWT token and logged in successfully.

AC-2: Student Submits Borrow Request with File Upload

- Given I am logged in as a student
- When I navigate to an item's detail page and click "Request to Borrow"
- And I fill in the required dates, purpose, and attach a valid file
- And I submit the form
- Then the request should be saved with a "Pending" status
- And the file should be uploaded to the server and linked to the request
- And I should see a success message and the request in "My Requests"
- And the admin should receive an email notification.

AC-3: Admin Processes Request and System Sends Email

- Given I am logged in as an admin
- When I view the list of pending requests and open the new request
- And I can view the uploaded file
- And I click "Approve"
- Then the request status should update to "Approved"
- And the associated item's availability should be updated
- And an email notification should be sent to the student.

AC-4: Android App Displays Role-Based UI

- Given I am logged into the Android app as a student
- When I view the main navigation
- Then I should see options for "Browse Items," "My Requests," and "Profile."
- But I should NOT see an "Admin Dashboard" option.
- When I log in as an admin, the "Admin Dashboard" option should appear.

3.0 NON-FUNCTIONAL REQUIREMENTS

3.1 Performance Requirements

- API response time: ≤ 2 seconds for 95% of authenticated requests.
- Web page load time: ≤ 3 seconds on a broadband connection.
- Mobile app cold start: ≤ 3 seconds on a reference device (API Level 34).

- Support at least 50 concurrent users for the borrowing and browsing features.
- Database queries, including those with joins across multiple tables, should complete within 500ms.

3.2 Security Requirements

- HTTPS for all communications between client and server.
- JWT token authentication for all protected endpoints.
- Passwords hashed with BCrypt (strength factor ≥ 10).
- Protection against SQL injection through JPA/Hibernate.
- Implement XSS protection measures.
- Rate limiting: 100 requests/minute per IP for public endpoints.
- Role-based access control must be enforced at the API level using Spring Security.
- JWT must be stored securely on the Android client.
- File uploads must be validated (type, size) and stored outside the application's webroot.

3.3 Compatibility Requirements

- Web Browsers: Latest two versions of Chrome, Firefox, Safari, and Edge.
- Android: Strictly API Level 34 (Android 14) with XML-based UI layouts.
- Screen Sizes: Responsive web design for common desktop (1024px+), tablet (768px), and mobile (360px+) viewports. Android app optimized for phone form factors.
- Operating Systems: Windows 10+, macOS 11+, Linux (for web clients).

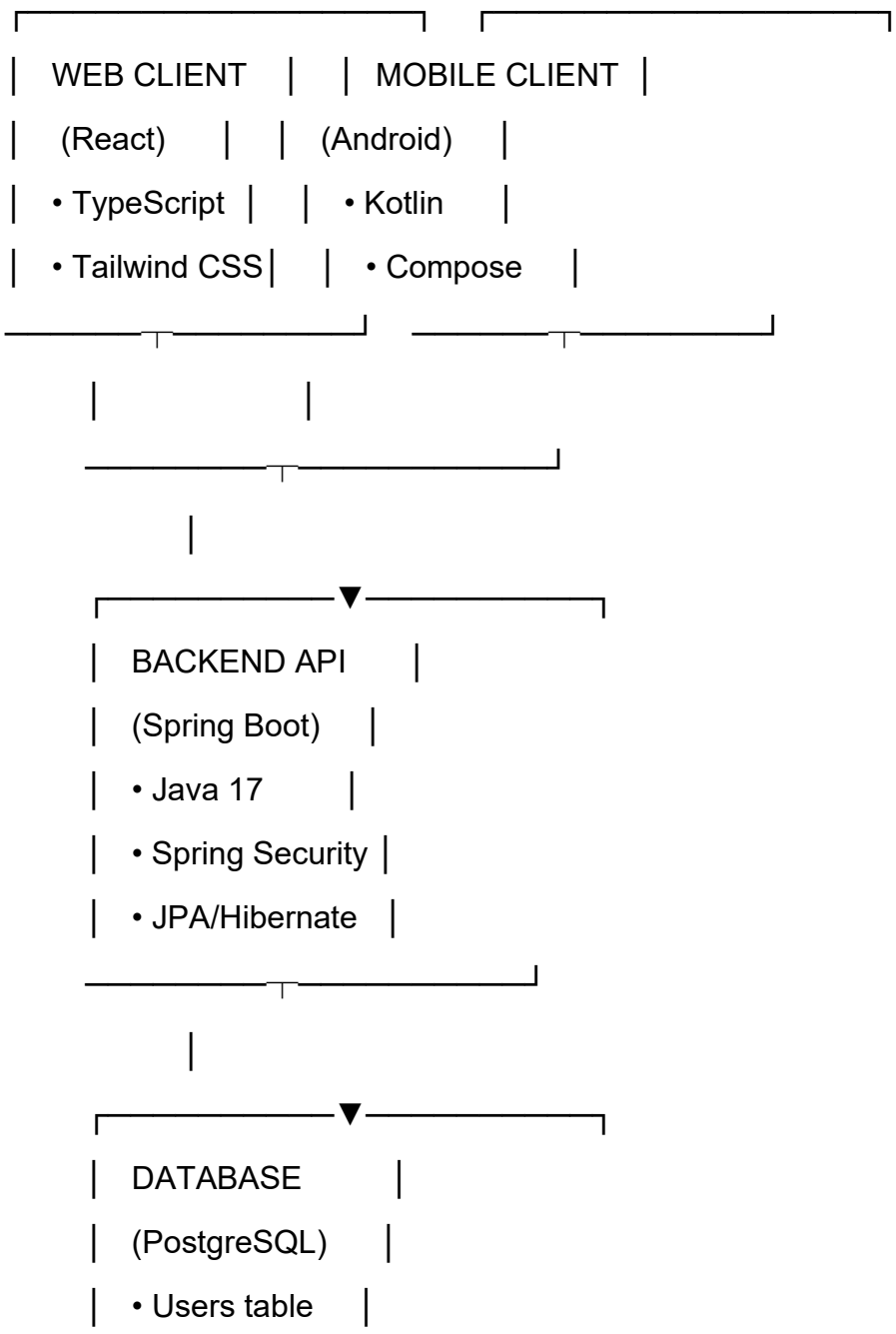
3.4 Usability Requirements

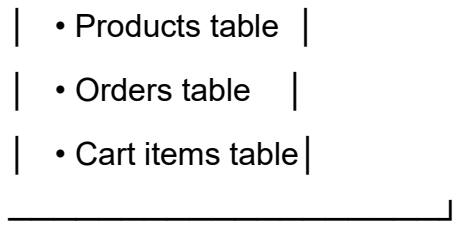
- A new user should be able to complete their first borrow request within 5 minutes.
- The Android app must follow Android design guidelines for touch targets (minimum 44x44dp).
- Clear, user-friendly error messages must be displayed for validation errors or failed API calls (e.g., "Invalid credentials," "Item not available for selected dates").
- Consistent navigation patterns across both web and mobile applications.
- Role-based UI changes must be intuitive, hiding features that are not permitted for the user's role.

4.0 SYSTEM ARCHITECTURE

4.1 Component Diagram

Note: This should be a component diagram





Technology Stack:

- **Backend:** Java 17, Spring Boot 3.x, Spring Security, Spring Data JPA
- **Database:** PostgreSQL 14+
- **Web Frontend:** React 18, TypeScript, Tailwind CSS, Axios
- **Mobile:** Kotlin, Jetpack Compose, Retrofit, Room
- **Build Tools:** Maven (Backend), npm/yarn (Web), Gradle (Android)
- **Deployment:** Railway/Heroku (Backend), Vercel/Netlify (Web), APK (Mobile)

5.0 API CONTRACT & COMMUNICATION

5.1 API Standards

- **Base URL:** https://api.marketplace.com/api/v1
- **Format:** JSON for all requests/responses
- **Authentication:** Bearer token (JWT) in Authorization header
- **Response Structure:**

```
json
{
  "success": boolean,
  "data": object|null,
  "error": {
    "code": string,
    "message": string,
    "details": object|null
  },
```

"timestamp": string

}

5.2 Endpoint Specifications

Authentication Endpoints

POST /auth/register

Body: {email, password, confirmPassword, fullName?}

Response: {user: {id, email, name}, token, refreshToken}

POST /auth/login

Body: {email, password}

Response: {user: {id, email, name, role}, token, refreshToken}

POST /auth/logout

Headers: Authorization: Bearer {token}

Response: {message: "Logged out successfully"}

Product Endpoints

GET /products

Query: ?page=1&limit=20&search=keyword&category=electronics

Response: {products: [...], pagination: {page, limit, total, pages}}

GET /products/{id}

Response: {product: {id, name, description, price, stock, imageUrl, category}}

POST /products (Admin only)

Body: {name, description, price, stock, imageUrl, category}

Response: {product: {...}}

PUT /products/{id} (Admin only)

Body: {name?, description?, price?, stock?, imageUrl?, category?}

Response: {product: {...}}

Cart Endpoints

GET /cart (Authenticated)

Response: {cart: {id, items: [...], total, itemCount}}

POST /cart/items

Body: {productId, quantity}

Response: {message: "Added to cart", cartItem: {...}}

PUT /cart/items/{itemId}

Body: {quantity}

Response: {message: "Cart updated", cartItem: {...}}

DELETE /cart/items/{itemId}

Response: {message: "Removed from cart"}

Order Endpoints

POST /orders

Body: {shippingAddress: {fullName, address, city, zipCode, country}}

Response: {order: {id, orderNumber, total, status, items: [...], createdAt}}

GET /orders

Response: {orders: [...]}

GET /orders/{id}

Response: {order: {...}}

5.3 Error Handling

HTTP Status Codes

- 200 OK - Successful request
- 201 Created - Resource created
- 400 Bad Request - Invalid input
- 401 Unauthorized - Authentication required/failed
- 403 Forbidden - Insufficient permissions
- 404 Not Found - Resource doesn't exist
- 409 Conflict - Duplicate resource
- 500 Internal Server Error - Server error

Error Code Examples

json

```
{
  "success": false,
  "data": null,
  "error": {
    "code": "AUTH-001",
    "message": "Invalid credentials",
    "details": "Email or password is incorrect"
  },
  "timestamp": "2024-01-28T10:30:00Z"
}
```

```
{
  "success": false,
  "data": null,
  "error": {
    "code": "VALID-001",
    "message": "Validation failed",
    "details": {
      "email": "Email is required",
      "password": "Must be at least 8 characters"
    }
  },
  "timestamp": "2024-01-28T10:30:00Z"
}
```


Common Error Codes

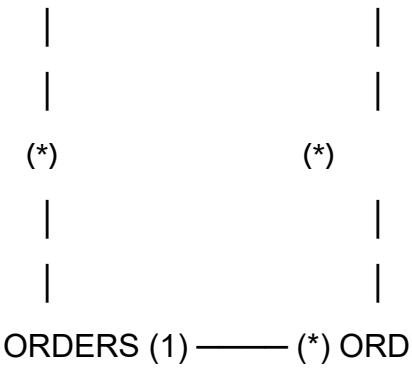
- AUTH-001: Invalid credentials
- AUTH-002: Token expired
- AUTH-003: Insufficient permissions
- VALID-001: Validation failed
- DB-001: Resource not found
- DB-002: Duplicate entry
- BUSINESS-001: Insufficient stock
- SYSTEM-001: Internal server error

6.0 DATABASE DESIGN

6.1 Entity Relationship Diagram

Note: This should be an ERD

USERS (1) ——— (1) CARTS (1) ——— (*) CART_ITEMS (*) ——— (1) PRODUCTS



Detailed Relationships:

- **One-to-One:** User ↔ Cart (Each user has exactly one cart)
- **One-to-Many:** User → Orders (User can have multiple orders)
- **One-to-Many:** Cart → CartItems (Cart contains multiple items)
- **One-to-Many:** Order → OrderItems (Order contains multiple items)
- **Many-to-One:** CartItems → Product (Items reference products)
- **Many-to-One:** OrderItems → Product (Items reference products)

Key Tables:

1. **users** - User accounts and authentication
2. **products** - Product catalog information
3. **carts** - Shopping cart per user
4. **cart_items** - Items in shopping cart
5. **orders** - Customer orders
6. **order_items** - Items in each order
7. **refresh_tokens** - JWT refresh tokens

Table Structure Summary:

- **users:** id, email, password_hash, full_name, role, created_at
- **products:** id, name, description, price, stock, image_url, category
- **carts:** id, user_id, created_at
- **cart_items:** id, cart_id, product_id, quantity
- **orders:** id, order_number, user_id, total, status, shipping_address

- **order_items:** id, order_id, product_id, product_name, quantity, price

7.0 UI/UX DESIGN

7.1 Web Application Wireframes

Note: This should be wireframes from Figma

Homepage (Product Listing)

Header: [Logo] [Search Bar] [Cart Icon] [User Menu]

Content: Product Grid (3 columns desktop)

Each Product Card: Image, Name, Price, "Add to Cart" button

Footer: Links, Copyright

Product Detail Page

Back Button

Product Image (large)

Product Name and Price

Description

Quantity Selector (1-10)

"Add to Cart" and "Buy Now" buttons

Product Specifications

Shopping Cart Page

Cart Title

List of Cart Items (Image, Name, Quantity, Price, Remove)

Order Summary: Subtotal, Shipping, Tax, Total

"Continue Shopping" and "Proceed to Checkout" buttons

Checkout Page

Shipping Address Form

Order Review (Items, Prices, Totals)

"Place Order" button

Terms and Conditions note

Admin Dashboard

Sidebar Navigation: Dashboard, Products, Orders, Users

Product Management: Add New button, Product list with Edit/Delete

Order Management: Order list with status filters

7.2 Mobile Application Wireframes

Note: This should be wireframes from Figma

Bottom Navigation

[🏠 Home] [🔍 Search] [🛒 Cart] [👤 Profile]

Home Screen

Search Bar

Product Grid (2 columns)

Swipe gestures for quick actions

Pull to refresh

Product Detail Screen

Back arrow

Product image (swipeable gallery)

Product info

Quantity selector

"Add to Cart" fixed bottom button

Cart Screen

Edit mode for quantity updates

Swipe to remove items

Order summary sticky bottom

Checkout button

Checkout Flow

Step indicator: Cart → Shipping → Payment → Confirm

Address form (auto-complete)

Order summary

Place order button

Mobile-Specific Features:

- Touch-optimized buttons (min 44x44px)
- Gesture support (swipe, pull-to-refresh)
- Offline caching for product images
- Bottom navigation for main actions
- Simplified forms for mobile input

Design System:

- **Colors:** Primary (#2563EB), Secondary (#7C3AED), Success (#10B981), Error (#EF4444)
- **Typography:** Inter font family, responsive sizing
- **Spacing:** 8px grid system
- **Components:** Consistent buttons, inputs, cards, modals
- **Responsive:** Mobile-first approach, breakpoints at 640px, 768px, 1024px

8.0 PLAN

8.1 Project Timeline

Phase 1: Planning & Design (Week 1-2)

Week 1: Requirements & Architecture

Day 1-2: Project setup and documentation

Day 3-4: Complete FRS and NFR

Day 5-7: System architecture design

Week 2: Detailed Design

Day 1-2: API specification

Day 3-4: Database design

Day 5-6: UI/UX wireframes

Day 7: Implementation plan finalization

Phase 2: Backend Development (Week 3-4)

Week 3: Foundation

Day 1: Spring Boot setup with dependencies

Day 2: Database configuration and entities

Day 3: JWT authentication implementation

Day 4: User management endpoints

Day 5: Product CRUD operations

Week 4: Core Features

Day 1: Cart functionality

Day 2: Order management

Day 3: Search and filtering

Day 4: Error handling and validation

Day 5: API documentation and testing

Phase 3: Web Application (Week 5-6)

Week 5: Frontend Foundation

Day 1: React setup with TypeScript

Day 2: Authentication pages (login, register)

Day 3: Product listing page

Day 4: Product detail page

Day 5: Shopping cart implementation

Week 6: Complete Web Features

Day 1: Checkout flow

Day 2: Order history and confirmation

Day 3: Admin dashboard

Day 4: Responsive design polish

Day 5: API integration and testing

Phase 4: Mobile Application (Week 7-8)

Week 7: Android Foundation

Day 1: Android Studio setup and project structure

Day 2: Authentication screens

Day 3: Product browsing

Day 4: Shopping cart

Day 5: API service layer

Week 8: Complete Mobile App

Day 1: Checkout flow

Day 2: Order management

Day 3: UI polish and animations

Day 4: Testing on emulator/device

Day 5: APK generation and documentation

Phase 5: Integration & Deployment (Week 9-10)

Week 9: Integration Testing

Day 1: End-to-end testing across platforms

Day 2: Bug fixes and optimization

Day 3: Security review

Day 4: Performance testing

Day 5: Documentation updates

Week 10: Deployment

Day 1: Backend deployment (Railway/Heroku)

Day 2: Web app deployment (Vercel/Netlify)

Day 3: Mobile APK distribution

Day 4: Final testing

Day 5: Project submission

Milestones:

- **M1 (End Week 2):** All design documents complete
- **M2 (End Week 4):** Backend API fully functional
- **M3 (End Week 6):** Web application complete
- **M4 (End Week 8):** Mobile application complete
- **M5 (End Week 10):** Full system deployed and integrated

Critical Path:

1. Authentication system (Week 3)
2. Product catalog API (Week 3-4)
3. Shopping cart functionality (Week 4)

4. Checkout process (Week 6)
5. Cross-platform testing (Week 9)

Risk Mitigation:

- Start with simplest working version of each feature
- Test integration points early and often
- Keep backup of working versions
- Focus on core functionality before enhancements