Project 1: Hospital Medical Information System

Project Description

Hospital Medical Information System (HMIS): Introduction

The HMIS is an integrated software system designed for hospitals to manage patient records, administrative tasks, and clinical workflows. The system will provide functionalities such as:

1. **Design a User-Friendly Interface**

- Develop dedicated modules for different functionalities (e.g., Patient Management, Billing, Diagnostics).
- Ensure intuitive navigation with a well-structured UI.
- Implement user roles (Doctor, Admin, Receptionist, Pharmacist) with controlled access.

2. **Integrate a Secure Database**

- Design relational database schema for patient records, appointments, billing, and staff details.
- Implement CRUD (Create, Read, Update, Delete) operations with appropriate indexing for efficiency.

3. **Develop Core Functionalities**

- **Patient Management:** Registration, unique ID generation, appointment scheduling.
- **Medical & Clinical Workflows:** Consultation records, lab test orders, pharmacy integration.
- **Administrative & Financial Modules:** Billing system, payroll management, inventory tracking.
- **Emergency & Critical Care:** Tracking ICU beds, emergency response team coordination.

4. **Implement Decision Support System**

- **Data-Driven Reports** – Generate reports for patient admission trends, department workload, and resource utilization.
- Provide real-time analytics for hospital resource optimization.

5. **Optimize Security & Compliance**

- Implement role-based access control with encrypted authentication.
- Enable logging for tracking user activity.
- Develop Notification & Alert System

- Set threshold-based alerts (e.g., bed occupancy limit, medicine stock levels).
- Implement SMS/email notifications for appointments and emergency cases.

6. **Testing & Quality Assurance**

- Unit testing and integration testing of different modules.
- Stress testing to ensure the system handles large hospital data volumes efficiently.

7. **Deployment & Maintenance**

- Deploy on a local server or cloud-based environment.
- Maintain version control using GitHub/GitLab.
- Provide post-deployment support, including bug fixes and feature updates.

8. **Integration with External Systems**

- Enable API-based interaction with insurance providers for claim processing.
- Support telemedicine features for remote consultations.

9. **Innovative Additions (X-Factor)**
   - **'Be creative'** and add some features after analyzing the situation which could be the x- factor.
   - For eg , voice-to-text conversion for doctor's notes.
   - **You are encouraged to add features to any of the above verticals based on your creativity.**

Background/Literature/Resources

Module	Tools / Tech Used
Frontend	HTML, CSS, JavaScript, React / Angular, Bootstrap
Backend	Node.js / Express OR Django / Flask OR Spring

DBMS	MySQL / PostgreSQL, ER Diagrams
Authentication & Security	JWT, bcrypt, OAuth
Reports & Analytics	Chart.js, Pandas, SQL
Notifications	Twilio, SendGrid, Nodemailer
Testing	Postman, Selenium, Jest, PyTest
Deployment	Git, GitHub, Heroku / AWS
Diagrams	Lucidchart, draw.io, Figma

Methodology & Milestones

Week 1: Planning, Requirements & System Design Goals:

- Understand the full scope and prepare a strong foundation for development.

Tasks:

- Finalize tech stack (e.g., React + Node.js + MySQL).
- Prepare **Software Requirements Specification (SRS)**.
- Identify functional & non-functional requirements.
- Design system architecture (modular overview).
- Create **UML diagrams**: Use Case, Class, Sequence.
- Design **ERD** (Entity-Relationship Diagram) for database.
- Start creating wireframes/prototypes.

Deliverables:

- Completed SRS document.
- UML diagrams + ERD.
- Wireframes (Figma or paper-based).
- GitHub repo initialized with module structure.

Week 2: Database Setup + Backend

Development Goals:

- Set up the database and begin backend API development.

Tasks:

- Create and normalize **relational database schema** in MySQL/PostgreSQL.
- Insert initial test data (e.g., patients, staff, appointments).
- Set up backend framework (Node.js/Django/Spring Boot).
- Build APIs for:
    - User login/authentication (role-based)
    - Patient registration and appointment scheduling
    - Staff & billing management
- Secure APIs using JWT/Auth middleware.

Deliverables:

Functional backend with at least 3–4 core APIs.

- Working database connected to backend.
- API documentation (Postman or Swagger).

Week 3: Frontend Integration + Core Module

Completion Goals:

- Build and connect frontend to backend. Complete key modules.

Tasks:

- Develop UI pages:
    - Login and Dashboard (role-specific views)
    - Patient registration, appointment booking
    - Billing & diagnostics view
- Integrate APIs with frontend using Axios/Fetch.
- Implement basic alert system (e.g., appointment reminders).
- Begin testing key flows (login, appointment, billing).

Deliverables:

- Working web app for core modules (UI + backend).
- Role-based login functioning.
- Partial alert/notification system.
- Initial testing reports.

Week 4: Testing, Enhancements, Deployment &

Demo Goals:

- Finalize, test, document, and present the complete system.

Tasks:

- Add additional features (analytics dashboard, stock alerts).
- Perform:
    - Unit testing
    - Integration testing
    - Stress testing (simulate large data)
- Prepare:
    - User manual
    - Developer documentation
    - Deployment guide
- Deploy app (Heroku, Render, or AWS).
- Final presentation and **live demo** preparation.

Implementation plan

Week	Focus Area	Key Deliverables
1	SRS + System & DB Design	SRS, UML, ERD, Wireframes
2	DB + Backend API Development	DB setup, core APIs, API docs
3	Frontend + Integration + Core Features	UI + backend integration, alerts, testing
4	Testing, Deployment & Presentation	Deployed app, docs, final demo

Evaluation Criteria

Dimension	Points	What We Look For

Technical Correctness	10	Functional code, all tests pass
Problem–Solution Fit	10	Clear link to a real pain point or requirement
Innovation / Creativity	20	Simulated delivery logic, interactive UI/UX, unique features.
Code Quality & Architecture	5	Modular code, clean architecture, scalable backend.
Documentation & Report	5	README, wireframes, schema, SRS, API docs.
Demo & Viva	50	Smooth walkthrough, real-time interaction, clear explanation of each feature.

Project 2: Automation of the Academic Section/Hostel Services at your University

Project Description

1. Introduction

The academic section at your university handles various administrative tasks, including course registrations, grade submissions, transcript generation, student record management, and faculty-student interactions.
The current processes involve a significant amount of manual work, leading to inefficiencies and delays. This project aims to build a **centralized web-based solution** that automates these workflows, minimizes human error, reduces delays, and ensures a seamless digital experience for students, faculty, and administrators.

2. Objectives

1. **Academic Record Automation**
   Develop a digital system to store, retrieve, and update student academic data such as course enrollments, grades, and attendance reducing dependency on paper-based records.
2. **Centralized Course Management**
   Allow students to register/add/drop courses within defined periods, and enable faculty to manage course rosters, monitor enrollments, and upload grades in real time.
3. **Transcript & Document Management**
   Automate transcript and certificate generation, with online request forms, real-time status tracking, and administrative approval workflows.
4. **Assignment Submission & Grading**
   Build a structured portal where faculty can upload assignments and deadlines, and students can submit their work. Enable digital grading, comments, and feedback uploads.
5. **Attendance Tracking System**
   Implement a feature for faculty to record class attendance, and for students to view their attendance summaries. Include visual attendance reports and alerts for shortages.

6. **Hostel Service Automation**
   Digitize hostel-related services:
   - Leave applications
   - Monthly mess subscription updates
   - Hostel room transfer requests
   - Complaint registration (infrastructure/gym/sports)
7. **Secure Role-Based Access**
   Integrate LDAP-based authentication and role-based access control (RBAC) to ensure that students, faculty, and admin staff access only the data relevant to their roles.
8. **Scalable Deployment**
   Deploy the system on a server with Docker support (optional Kubernetes), ensuring security, performance, and future scalability.

3. Scope of the Project:

Key Features Student Portal

- Course add/drop
- View grades, attendance, academic records
- Request & track transcripts/certificates
- Submit assignments with deadlines
- View attendance logs
- Apply for hostel leave, update mess subscription
- Request hostel transfers, register complaints

Faculty Portal

- Manage courses and student rosters
- Upload and edit grades
- Monitor and update attendance
- Upload assignments and give feedback
- Communicate with students

Admin Portal

- Manage student/faculty profiles

- Process document and transcript requests
- Approve leave, mess changes, hostel transfers
- Manage complaints and issue resolutions
- Integrate with ERP systems

Security & Authentication

- **Authentication**: ILDAP-based login
- **Access Control**: Role-Based Access Control (RBAC)
- **Security**: Encrypted data storage and secure API communication

Background/Literature/Resources

Component	Tools / Technologies
Frontend	React.js / Vue.js / Angular
Backend	Node.js + Express / Django / Flask
Database	PostgreSQL / MySQL / MongoDB
Authentication	LDAP, JWT
Deployment	Server, Docker, Kubernetes (if applicable)
CI/CD & Version Control	GitHub / GitLab, Jenkins, GitHub Actions

Methodologies & Implementation Plan

Week 1: Requirement Gathering & System
Design Goals:

- Understand academic & hostel workflows.
- Finalize system requirements and design structure.

Tasks:

- Analyze existing manual processes.
- Prepare **Software Requirements Specification (SRS)**.
- Design system architecture (micro-modular).

- Define user roles & permissions.
- Create:
    - UML Diagrams: Use Case, Class, Sequence
    - ERD for academic & hostel data
    - UI Wireframes for all portals

Deliverables:

- SRS Document
- UML & ERD diagrams
- UI wireframes/prototypes
- GitHub repo initialized with project structure

Week 2: Backend & Database

Development Goals:

- Build backend logic and connect database.

Tasks:

- Set up PostgreSQL/MySQL DB
- Define and normalize database schema
- Develop core APIs for:
    - Course registration, document requests
    - Assignment submission, attendance logging
    - Hostel services (leave, mess, complaints)
- Implement JWT/LDAP authentication
- Test API endpoints using Postman

Deliverables:

- Working backend APIs (6–8 key modules)
- Auth module with JWT or LDAP integration
- Connected database with sample data
- API documentation (Postman or Swagger)

Week 3: Frontend Development &

Integration Goals:

- Build and integrate UI with backend APIs.

Tasks:

- Develop frontend components for:

        - Student, faculty, and admin dashboards
        - Assignment portal, attendance views
        - Leave, mess, transfer, and complaint forms
    - Integrate frontend with backend APIs (Axios/Fetch)
    - Begin role-based rendering and state management

Deliverables:

- Functional portals with basic workflows
- UI connected with backend logic
- Role-based access navigation
- Initial screenshots/demos

Week 4: Testing, Deployment & Final

Demo Goals:

- Final testing, deployment, and documentation.

Tasks:

- Perform:
    - Unit testing
    - Integration testing
    - UAT (User Acceptance Testing) with dummy data
- Deploy application on a server (via Docker)
- Prepare:
    - Technical & user documentation
    - Deployment guide
- Create presentation slides and rehearse demo

Deliverables:

- Live deployed application
- Full documentation set (User + Technical)
- Final presentation slides
- Test report and feedback summary

Implementation Plan

Week	Focus Area	Key Deliverables
1	SRS + System & DB Design	SRS, UML, ERD, Wireframes
2	DB + Backend API Development	DB setup, core APIs, API docs
3	Frontend + Integration + Core Features	UI + backend integration, alerts, testing
4	Testing, Deployment & Presentation	Deployed app, docs, final demo

Evaluation Criteria

Dimension	Points	What We Look For
Technical Correctness	10	Functional code, all tests pass
Problem–Solution Fit	10	Clear link to a real pain point or requirement
Innovation / Creativity	20	Novel features, optimizations, unique approaches
Code Quality & Architecture	5	Clean structure, design patterns, and documentation
Documentation & Report	5	README, diagrams, API specification, explanations
Demo & Viva	50	Smooth walkthrough of the system, confident viva

Project 3: Community-Based Local Services Platform

Project Description

Introduction:

This project aims to develop a web-based platform that connects users with local service providers (plumbers, electricians, tutors, cleaners, etc.). The platform will streamline the process of discovering, booking, and paying for services while offering tools for communication, feedback, and support.

Objectives

1. **Service Provider Onboarding:**
    - Allow providers to register, verify, and set up detailed profiles with skills, pricing, location, and availability.
2. **User- Friendly Search & Booking System:**
    - Users can search based on service category, rating, distance, availability, and pricing.
    - Provide calendar-based appointment scheduling and instant booking functionality.
3. **Secure, Real-Time Messaging:**
    - In-app messaging between users and providers for coordination and negotiation.
    - Real-time notifications for new messages, booking confirmations, and reminders.
4. **Review and Rating System:**
    - After service completion, users can submit ratings and detailed reviews.
    - Ratings influence provider visibility and credibility.
5. **Integrated Payment Gateway:**
    - Seamless payments via Razorpay, Stripe, or PayPal.

- Options for wallets, UPI, cards, and net banking.
- Automatic invoice generation and transaction logs.
- Feedback and Support Module:
- Forms for suggestions, bug reports, and inquiries.
- Support chat/email integration for dispute resolution.

6. **Admin Dashboard:**
    - Manage providers and user accounts, view analytics, moderate reviews, resolve disputes.

Key Features

1. **User Features:**
    - Search and filter service providers by location, category, rating, and cost.
    - Schedule appointments via integrated calendar.
    - In-app chat with providers.
    - Secure online payments.
    - Submit reviews and rate services.
    - View booking history and invoices.
2. **Service Provider Features:**
    - Create and manage service profiles.
    - Set availability, pricing, and service types.
    - View and manage bookings.
    - Chat with users and send updates.
    - Track payments and earnings.
3. **Admin Features:**
    - Approve or reject provider accounts.
    - Monitor platform activity and resolve disputes.
    - Moderate reviews and complaints.
    - Generate reports on transactions, usage, and ratings.
4. **Notifications:**
    - SMS/email/push notifications for bookings, payments, and updates.
5. **Security Features:**
    - JWT-based role-specific access (Admin/User/Provider).
    - Encrypted storage of sensitive data.

Background/Literature/Resources

Module	Tools/Tech Used
Frontend	React.js / Vue.js / Angular, Tailwind CSS
Backend	Node.js + Express / Django / Flask
Database	MongoDB / PostgreSQL / MySQL
Authentication	JWT, OAuth 2.0
Notifications	OneSignal, Firebase, Email (SendGrid)
Payments	Razorpay / Stripe
Testing	Jest / Postman / Selenium
Deployment	GitHub + Render / Vercel / AWS
Diagrams/Wireframes	Figma, draw.io, Lucidchart

Methodologies & Implementation Plan

Week 1: Planning, Research & Design

Goals:

- Define complete system requirements and user flows.
- Choose the final tech stack.
- Design architecture and database schema.
- Build basic wireframes and layout.

Tasks:

- Gather requirements (user & service provider journeys).
- Create Software Requirement Specification (SRS).
- Draw UML diagrams: Use Case, Class, Sequence.
- Design ERD (Database schema).
- UI wireframes for:
    - Home, Search, Profile, Booking, Chat, Admin Panel.

Deliverables:

- SRS document
- UML + ER diagrams
- Wireframes (Figma)
- Initialized GitHub repo with folder structure

Week 2: Backend Development + Database

Integration Goals:

- Set up the database and develop core backend APIs.

Tasks:

- Set up PostgreSQL/MySQL or MongoDB.
- Define models: User, Provider, Booking, Chat, Payment.
- Build APIs:
    - User/provider registration/login
    - Profile CRUD
    - Booking system (create/update/cancel)
    - Review and rating logic
    - Payment integration
- JWT-based role-authentication setup

Deliverables:

- Backend codebase with routes, models, and middleware
- Database schema with dummy data
- API documentation (Postman/Swagger)

Week 3: Frontend Development + Integration

Goals:

- Build responsive UI and integrate frontend with backend.

Tasks:

- Pages:
    - Homepage, Search, Login/Signup
    - Profile (User/Provider)
    - Booking page & calendar
    - Review and Ratings
    - Chat & notification bar
- API integration using Axios/Fetch
- Implement:Real-time messaging (WebSockets or polling)
    - Notification system (email/push)

Deliverables:

- UI integrated with backend
- Working booking and chat system
- Frontend validation + feedback

Week 4: Testing, Final Touches, Deployment &

Demo Goals:

- Conduct comprehensive testing, deployment, and prepare for presentation.

Tasks:

- Unit & Integration testing
- Bug fixing and optimizations
- Admin dashboard creation
- Add customer support/contact module
- Deploy app (Render/Heroku/AWS)
- Write:
    - User Manual
    - Dev Documentation
    - Deployment Guide
- Prepare final slides and live demo

Deliverables:

- Live deployed site with demo credentials
- Test reports & issue log
- Documentation package
- Final presentation

Week	Focus Area	Key Deliverables
1	SRS + System Design	SRS, UML/ER diagrams, Wireframes
2	Backend & DB Setup	API routes, DB schema, Auth, API docs
3	Frontend & Integration	UI pages, Chat & Booking integration

Evaluation Criteria

Dimension	Points	What We Look For
Technical Correctness	15	Fully functional system with no critical bugs

Problem–Solution Fit	10	Solves the stated real-world problem effectively
Innovation/Creativity	15	Unique features, user experience enhancements
Code Quality & Design	5	Modular, clean, maintainable code with comments
Documentation & Reports	5	Well-maintained docs, diagrams, manuals
Demo & Viva	50	Smooth walkthrough, team clarity, depth of understanding

Project 4: Smart City Management Platform

Project Description

Introduction:
The Smart City Management Platform is a centralized digital solution aimed at helping city administrators monitor, analyze, and improve urban infrastructure and public services. It integrates real-time and historical data from various city systems to provide actionable insights for traffic management, energy optimization, waste management, and emergency response. The system features dashboards, analytics tools, and alert mechanisms to drive efficient, data-driven urban governance.

Objectives

1. **Centralized Monitoring Dashboard:**
   Create an intuitive, real-time dashboard to oversee various city operations such as traffic, air quality, waste collection, and utilities.
2. **Predictive Analytics and Insights:**
   Analyze operational data using algorithms to detect patterns, optimize city resources, and generate recommendations.

3. **Real-Time Alerts and Notifications:**
   Notify relevant departments or officials instantly when a threshold is crossed (e.g., air quality deteriorates, road congestion).
4. **Resource Allocation and Optimization:**
   Suggest optimized resource use (e.g., waste truck routing, energy usage balancing) using AI/ML models or rule-based logic.
5. **Role-Based Access and Security:**
   Ensure that only authorized personnel can access or modify specific modules using secure login and permission control systems.

Key Features

1. **Admin Dashboard:**
   - View real-time data from multiple city systems.
   - Graphs, charts, and maps for data visualization.
   - Control panels for emergency response.
2. **Alerts & Notifications:**
   - Auto-alert system triggered by thresholds or anomalies.
   - SMS, email, or in-app alerts for emergencies or maintenance needs.
3. **Analytics & Reports:**
   - Daily/weekly/monthly reports on air quality, energy use, traffic, etc.
   - Predictive analytics for future planning (e.g., event-based traffic forecasting).
4. **Optimization Modules:**
   - Traffic signal timing suggestions.
   - Garbage collection route planning.
   - Dynamic electricity grid balancing.
5. **Security & Access Control:**
   - Role-based dashboards (Admin, Environment Officer, Utility Officer, Traffic Control).
   - Authentication (JWT/OAuth/LDAP) and secure data storage.
6. **Integration with City Systems:**

   APIs to interface with utility grids, emergency services, transport networks.

Background/Literature/Resources

Component	Suggested Tools/Technologies
Frontend	React.js / Vue.js / Angular

Backend	Node.js / Django / Flask / Spring Boot
Database	PostgreSQL / MongoDB / MySQL
Visualization	D3.js / Chart.js / Power BI / Grafana
Authentication	JWT / OAuth 2.0 / LDAP
Notifications	Twilio (SMS), SendGrid (Email), Push APIs
Hosting/Deployment	Docker, Kubernetes, AWS / GCP / Azure
Version Control	GitHub / GitLab

Methodologies & Implementation Plan Week 1:

Planning & System Design

Focus Area: Requirement Analysis, Design Documentation, and Initial Prototyping

Objectives:

- Understand the scope of smart city features and services
- Define user roles (admin, maintenance staff, monitoring authority)
- Plan architecture and data flow
- Set up tools and initial repo

Key Activities:

- Conduct requirement gathering sessions (e.g., for traffic management, waste collection, etc.)
- Draft the **SRS Document** detailing:
    - Functional requirements (alerts, dashboards)
    - Non-functional requirements (scalability, security, availability)
- Design the **Database Schema** to store:
    - Service data logs (traffic, utilities, environment, etc.)
    - Alert logs, maintenance records, user accounts

- Draw UML diagrams:
    - Use case diagrams for each user type
    - Sequence diagrams for alert flow or data processing
- Create UI mockups using Figma/Adobe XD or simple wireframes for:
    - Admin dashboard
    - Data visualization panel
    - Alert console
- Select the **Tech Stack**:
    - Frontend: React.js / Vue.js
    - Backend: Node.js / Django / Flask
    - Database: MongoDB / PostgreSQL
    - IoT Communication: MQTT / HTTP APIs
    - Deployment: Docker + Cloud / local server

Deliverables:

- Complete SRS document
- Database schema diagram
- UI wireframes/mockups
- Version control (GitHub) setup

Week 2: Core Development – Backend + Frontend

Focus Area: Implement Functional Modules & Data Flow

Objectives:

- Build foundational infrastructure for data handling
- Create frontend interfaces

Key Activities:

- Backend Development:
    - Create RESTful APIs for:
        - User login (role-based)

        - Alerts and resource requests
    - Connect backend to the database
- Frontend Development:
    - Build UI for:
        - Dashboard with summary statistics
        - Basic alert interface
- Integrate API endpoints with frontend using Axios/Fetch

Deliverables:

- Basic working dashboard
- API integrations completed
- Live or simulated sensor data showing on dashboard
- Frontend + backend connected and tested for core flows

Week 3: Analytics & Optimization Engine

Focus Area: Real-time analytics, resource optimization

Objectives:

- Create insights and real-time analytics
- Enhance UI with charts, graphs, and maps

Key Activities:

- Simulate time-series operational data
- Store and update live metrics
- Implement analytics:
- Detect anomalies, compute trends
- Recommend resource allocation (e.g., dispatch trucks based on waste levels)

Visualizations:

Heatmaps, pie charts, bar graphs, trend lines

Deliverables:

- Working analytics dashboard
- Trend-based visualization
- Rule-based recommendation engine
-

Week 4: Testing, Alerts & Deployment

Focus Area: Finalize System, Secure It, Test It, and Deploy

Objectives:

- Polish the system
- Ensure robust alerting, authentication, and security
- Prepare final deployment and demo

Key Activities:

- Implement Alert System:
    - Define alert conditions (e.g., high traffic, low water level)
    - Use email/SMS/web pop-up notifications
- Add Role-Based Login:
    - Admin: Full control
    - Operator: View & act on alerts
    - Viewer: Read-only access
    - Use JWT or session tokens with password encryption
- Perform Testing:
    - Unit and integration testing
    - Stress testing using large data sets
- Deployment:
    - Use Docker for containerization
    - Deploy on cloud platform (Render, Heroku)
    - Final Documentation:
    - Technical documentation (API docs, DB structure)
    - User guide/manual
- Demo Prep:
    - Record a short walkthrough
    - Prepare slides and Q&A script

Deliverables:

- Fully functional deployed system
- Bug-free, tested modules
- Documentation pack
- Demo-ready application with alert triggers and visualizations

Implementation Plan

Week	Focus Area	Key Activities
Week 1	Planning & System Design	- Requirement gathering - SRS Document - Database schema - Wireframes/UI mockups
Week 2	Core Development: Backend +	- Backend API development - Frontend dashboard screens

	Frontend	
Week 3	IoT Integration & Analytics	- Connect with live/mock sensors - Build analytics engine - Visualize insights with charts and maps
Week 4	Testing, Alerts & Deployment	- Add alert system - Role-based login & security - Testing & bug fixes - Final deployment & demo preparation

Evaluation Criteria

Dimension	Points	What We Look For
Technical Correctness	15	Working features, reliable backend
Problem–Solution Fit	10	How well the platform addresses smart city use cases
Innovation/Creativity	15	Unique visualizations, smart resource suggestions, predictive alerts
Code Quality & Design	5	Clean code structure, modularity, readability, and inline documentation
Documentation & Reports	5	Comprehensive SRS, architectural diagrams, and user/admin manuals
Demo & Viva	50	Clear project walkthrough, real-time dashboard insights, confident presentation

Project 5: Personal Budget Tracker

Project Description

Introduction:

The Personal Budget Tracker is a comprehensive financial management tool designed to help users track income, expenses, budgets, and spending behavior. The platform

supports both individual and group-based expense tracking, allowing users to manage shared financial activities across different contexts like households, trips, or social events. It offers clean interfaces, strong CRUD operations, budget setting, and detailed data visualization to encourage better financial decisions.

Objectives

1. **Daily Financial Entry Logging:**
    - Allow users to add, edit, and delete income and expense entries.
    - Each entry includes fields such as amount, category, date, and optional notes.
2. **Category-Based Budget Setting:**
    - Let users define monthly budgets for specific categories like food, transport, entertainment, etc.
    - Provide real-time tracking of category spending vs. budget.
3. **Data Visualization:**
    - Generate intuitive bar graphs, pie charts, and trend lines to represent income vs. expense, category-wise spending, and budget usage.
4. **Financial Summaries & Insights:**
    - Show monthly summaries including savings, overspending alerts, and percentage breakdowns.
    - Help users develop smarter money habits through spending analysis.
5. **Data Storage and Management:**
    - Use a structured and optimized database to store user data.
    - Ensure efficient CRUD operations and data retrieval.
6. **Optional: User Authentication & Syncing:**
    - Implement basic login/logout functionality.
    - Optionally support data sync via cloud storage or export/import features (CSV, JSON).
7. **Group Expense Management:**

- Users can create and manage groups for shared financial activities.
- Within groups, expenses can be recorded with custom split configurations (equal or unequal).
- The system automatically calculates outstanding balances between members.
- Users can record settlements to update balances.

Key Features

1. **Transaction Management:**
    - Add/edit/delete transactions.

- Filter by date range, type (income/expense), and category.

2. **Category-Based Budgeting:**
    - Set monthly budgets for each category.
    - View budget progress bars and alerts for overspending.
3. **Dashboard & Visual Reports:**
    - Income vs. Expense graph.
    - Monthly trend charts.
    - Category-wise breakdown in pie charts.
4. **Search & Filters:**
    - Search transactions by keyword, category, or date.
    - Custom filters for reports and summaries.
5. **Secure Storage:**
    - Use local or cloud-based storage with encryption if needed.
    - Backup/export options.
6. **(Optional) User Authentication:**
    - Sign-up/Login system for saving personal budgets across sessions.
7. **Group Expense Handling:**

- Create groups and add participants.
- Automatically compute how much each member owes or is owed based on their share.
- Allow users to record payments or settlements to keep balances up to date.

Background / Literature / Resources

Component	Recommended Tools/Technologies
Frontend	React.js / Vue.js / HTML-CSS-JS
Backend	Node.js / Django / Flask
Database	SQLite / PostgreSQL / MongoDB
Data Visualization	Chart.js / D3.js / Google Charts
Authentication (Opt.)	Firebase Auth / JWT / Session-based

Hosting (Opt.)	Netlify / Render / Vercel
Version Control	Git + GitHub

Implementation Plan

Week 1: Requirement Gathering, Planning & Design

Objectives:

- Understand user needs
- Design user flows and UI
- Plan project architecture and technology stack

Tasks:

- Identify key user personas: e.g., a student, salaried professional
- List functional requirements: transaction logging, budget setting, graphs, etc.
- Define non-functional requirements: data privacy, responsiveness, ease of use
- Choose tech stack (e.g., React + Node.js + MongoDB or SQLite)
- Create wireframes for:
    - Dashboard
    - Add Transaction
    - Budget Management
    - Reports/Graphs page
- Design database schema:
    - Users (if auth is used)
    - Transactions (amount, type, category, date, notes)
    - Budgets (category, limit, month)
- Create ER Diagram and initial class/flow diagrams
- Setup version control (GitHub repo)

Deliverables:

- Software Requirements Specification (SRS)
- UI wireframes
- Database schema & ER diagram
- Technology stack finalized

Week 2: Core Development – CRUD & Budget Management

Objectives:

- Implement backend APIs and database connectivity
- Create frontend forms and pages for transactions and budgets
- Enable group management and shared expense calculation

Tasks:

- Backend:
    - Set up project structure
    - APIs for users, transactions, groups, budgets

        - Shared expense computation logic (based on ratio or equal split)
        - Implement CRUD APIs for transactions and budgets
        - Connect to chosen DB (MongoDB, SQLite, etc.)
    - Frontend:
        - Create Add/Edit/Delete Transaction page
        - Display transactions in a table with filters (category, date)
        - Build form for budget creation/updating
        - Show real-time budget consumption (progress bar)
        - Group creation and member management
        - Display of group balances and dues
    - Connect frontend to backend using API calls (Axios or Fetch)
    - Basic error handling and form validation

Deliverables:

- Working backend APIs
- Frontend pages for transactions and budget management
- Integration with the database
- Mid-week code review to ensure module completion

Week 3: Data Visualization & Insights

Objectives:

- Enable users to view analytics and financial summaries
- Visualize expenses with interactive charts

Tasks:

Use Chart.js or D3.js to display:

        - Pie chart: Category-wise spending
        - Bar chart: Monthly income vs expenses
        - Line chart: Spending trend over time
    - Summary widgets:
        - Total income & expenses
        - % of budget used per category
        - Current savings
    - Implement filters: monthly, category, type (income/expense)
    - Add transaction search and sort functionality
    - Start implementing optional features (auth, export, reminders)

Deliverables:

- Functional and dynamic graphs
- Summary dashboard
- Filterable and searchable transaction reports

Week 4: Testing, Deployment & Documentation

Objectives:

- Ensure stability, polish the app, and prepare for final presentation

Tasks:

- Perform unit testing for backend APIs
- Conduct integration testing between frontend and backend
- Fix bugs and resolve UI issues (styling, responsiveness)
- Deploy on GitHub Pages, Render, Netlify, or local server
- Write documentation:
    - Technical guide for developers
    - User manual
    - Deployment instructions
- Prepare final presentation/demo
- Collect user feedback (optional beta test)

Deliverables:

Fully functional deployed app

- All documentation (README, user guide, diagrams)
- Final recorded/live demo
- Presentation slides and Q&A readiness

Week	**Focus Area**	**Key Tasks**
Week 1	Requirement Analysis & Design	SRS, ERD, wireframes, Git setup
Week 2	CRUD System + Group Logic	Personal + shared transactions, group flows
Week 3	Budgeting & Visual Reports	Charts, summaries, group balances
Week 4	Testing, Polishing & Deployment	Final testing, deployment, documentation

Evaluation Criteria

Dimension	Points	What We Look For
Technical Correctness	10	Reliable CRUD, accurate balance computation
Problem–Solution Fit	10	Solves both personal and shared expense needs
Innovation / Creativity	20	Clean UI, intelligent summaries, group settlement workflows
Code Quality & Design	5	Modular structure, comments, reusable components
Documentation & Report	5	SRS, diagrams, technical + user guide
Demo & Viva	50	Clear walkthrough, realistic use case demonstration, smooth handling of group flows

Project 6: FOOD ORDER–DELIVERY SYSTEM

Project Description

This system enables users to order food from registered vendors, track their orders, make secure payments, and review services. It also allows vendors to manage menus, view orders, and coordinate deliveries efficiently.

Objectives

1. **Vendor Onboarding & Management**
    - Allow vendors to register, manage their profile, and update menus, prices, and availability.
2. **User-Centric Ordering System**
    - Let users browse menus, place food orders, and customize items within a seamless UI.
3. **Order Delivery & Tracking**

    - Simulate real-time order progress and delivery tracking with updates on each stage.

4. **Secure Payments & Billing**
    - Integrate popular digital payment methods like UPI, Paytm, and Cards with receipt generation.

5. **User Profiles & Order History**
    - Enable users to manage personal data, saved addresses, and view past orders for reordering.

6. **Review & Feedback System**
    - Facilitate post-delivery reviews and ratings for food and vendor service.

7. **Admin Panel (Optional/Advanced)**
    - Manage users, vendors, delivery reports, and access analytics.

KeyFeatures

1. Vendor Profile Module

- Registration and login for food vendors.
- Profile creation with vendor details: name, address, contact, cuisine type, certifications (optional).
- Menu management:
    - Add/edit/delete food items.
    - Set item prices, images, and availability status.
- View current orders and update status (e.g., Preparing, Ready, Delivered).

- View analytics: order volume, earnings, reviews.

2. Food Ordering System (Customer Side)

- Browse food items by vendor, cuisine, rating, or price.
- Add items to cart, modify quantities.
- View estimated delivery time and item availability.
- Place order with selected vendor and confirm address.
- Real-time status updates during order process.

3. Delivery Module

- Assign delivery tasks to delivery personnel.
- Track current delivery assignments and route optimization (basic simulation).
- Update order status as "Out for delivery", "Delivered".
- Allow customers to confirm receipt.
- View delivery history for performance tracking (admin/delivery person).

4. Real-time Order Tracking

- Display order progress: Ordered → Preparing → Out for Delivery → Delivered.
- Track delivery personnel location (mock/simulated).
- Push notifications for status updates.
- Allow users to cancel/reorder (with conditions).

5. Payments Integration

- Secure payment gateway with options:
    - UPI
    - Paytm/Wallets
    - Debit/Credit cards
    - Net Banking
- Invoice generation after successful payment.
- Handle refunds.

6. Customer Profile

- Registration and login for customers.
- Profile details: Name, Email, Contact Number, Addresses.
- Save favorite vendors or food items.
- Manage default payment methods and delivery addresses.

7. Order History Module

- List of past orders with filters (date, vendor, rating).
- Reorder functionality for past meals.
- View receipts, ratings given, delivery info.

8. Review & Rating Module

- Rate food items and vendors post-delivery.
- Write reviews (text-based) with optional images.

- Filter vendors based on average rating.
- Vendors can respond to feedback (optional).

Background/ Literature/ Technologies

Layer	Tools/Technologies
Frontend	HTML/CSS/JS
Backend	Node.js with Express / Django / Flask
Database	MongoDB / PostgreSQL / MySQL
Authentication	JWT (JSON Web Tokens), Session-based login
Payments	Razorpay API (mandatory) / Paytm API / Stripe (optional)
Hosting	Firebase / Render / Railway.app / Heroku (for demos)
Version Control	Git + GitHub

Methodologies & Implementation Plan

Week 1: Planning & System Design

- Conduct requirement analysis and define use cases for users, vendors, and delivery agents.

- Prepare **SRS (Software Requirements Specification)** document.
- Design database schema for user, vendor, order, and delivery entities.
- Create wireframes/mockups for:
    - User food ordering interface
    - Vendor dashboard
    - Order tracking screen
- Set up GitHub repo and decide tech stack.
- Begin basic UI scaffolding and routing.

Week 2: Core Development – Vendor & Customer Modules

- Implement user registration/login and role-based access control (user, vendor).
- Develop vendor dashboard with:
    - Menu/item management
    - Order view/update functionality
- Build user-side:
    - Menu browsing
    - Cart functionality
    - Checkout and address selection
- Connect frontend with backend using RESTful APIs.

Week 3: Delivery System, Tracking & Payments

- Simulate delivery flow with mock delivery agent status updates.
- Add order status transitions (Ordered → Preparing → Out for Delivery → Delivered).
- Implement live order tracking module (with interval-based updates or dummy maps).
- Integrate **payment gateway** (Razorpay/Paytm API).
- Build secure order placement and transaction validation system.

Week 4: Finalization – Reviews, History & Testing

- Add:
    - Order history and receipt module for users.
    - Review & rating feature for completed orders.
    - Admin dashboard (if applicable).
- Implement notification system (in-app or simulated push).
- Conduct:
    - Integration testing
    - UI/UX fixes
    - Bug resolutions
- Prepare final **documentation**, demo video, and deploy if needed.

Week-wise plan

Week	Focus Area	Key Activities
Week 1	Planning & System Design	- Define system requirements and use cases for user, vendor, and delivery roles. - Prepare SRS document. - Design database schema and ER diagrams. - Create wireframes for customer, vendor, and tracking interfaces. - Set up project repository and decide tech stack.
Week 2	Core Modules: Vendor & Customer	- Implement role-based registration and login. - Develop vendor dashboard (menu CRUD, order view). - Build customer UI for browsing food, adding to cart, and placing orders. - Create backend APIs for menus, users, and orders.
Week 3	Delivery, Payments & Tracking	- Simulate delivery system and status tracking. - Connect frontend to backend for real-time order updates. - Integrate Razorpay or Paytm payment gateway. - Build secure checkout and transaction confirmation features.
Week 4	Testing, Reviews & Finalization	- Implement user order history and downloadable receipts. - Add review & rating system for vendors. - Perform full system testing and resolve bugs. - Finalize UI design polish. - Prepare documentation, demo walkthrough, and deployment.

Evaluation Criteria

Dimension	Points	What We Look For

Technical Correctness	10	Functional order-placement, delivery, and vendor modules.
Problem–Solution Fit	10	Clearly addresses modern food ordering needs.
Innovation / Creativity	20	Simulated delivery logic, interactive UI/UX, unique features.
Code Quality & Architecture	5	Modular code, clean architecture, scalable backend.
Documentation & Report	5	README, wireframes, schema, SRS, API docs.
Demo & Viva	50	Smooth walkthrough, real-time interaction, clear explanation of each feature.

Project 7: Common Faculty-Student Interaction Portal

Project Description

Introduction

The Student Interaction Portal is a centralized academic management system designed to streamline communication and coordination between students, faculty, and administrators. It consolidates multiple academic utilities such as profile management, discussion forums, assignment submissions, seminar scheduling, study materials, and academic calendars into one unified interface.

To elevate its complexity and real-world applicability, the platform now includes real-time chat, intelligent scheduling conflict detection, course-specific content repositories, analytics dashboards, document versioning, and teaching assistant delegation. These features ensure dynamic interaction, better resource planning, and a scalable architecture suited for modern academic institutions.

Objectives

1. **Streamline Academic Communication**
   Structured forums and course-specific discussion threads for student-faculty interaction.
2. **Enable Role-Based Profile Management**
   Tailored dashboards and permissions for Students, Faculty, Admins, and Teaching Assistants.
3. **Centralize Academic Timelines & Deadlines**
   Unified calendar showing personalized exam, quiz, seminar, and assignment dates.
4. **Manage Course Projects, Assignments & Seminars**
   Faculty can assign, evaluate, and give feedback; students can submit and track deliverables.
5. **Provide Course Content Repository (Video + Docs)**
   Faculty can upload lectures, slides, notes, and reference links; access is controlled per course.
6. **Integrate Real-Time Communication**
   In-course private chat between students and faculty, group chats for enrolled users, powered by WebSockets.
7. **Enable Smart Deadline Conflict Detection**
   Automatically flag overlapping assessments for enrolled students and suggest alternative schedules.
8. **Analytics Dashboard for Faculty/Admins**
   Engagement metrics, resource access logs, submission trends, and usage statistics visualized through graphs.
9. **Document Version Control for Materials**
   Maintain version history for course materials and allow students to access prior versions.
10. **Enable Secure Access & Delegation**
    Support role-based access control with authentication and faculty-assisted delegation to TAs.
11. **Ensure Responsive and Accessible Design**
    Fully optimized for mobile and desktop with accessibility compliance and privacy considerations.

Key Features

1. **Home & FAQs Section**
   Introductory content, platform usage guide, FAQs for new users.
2. **Profile Management**
   Role-based dashboards for Students, Faculty, Admins, and TAs with permission control.
3. **Discussion Forums**
   Course-wise threads with moderation, comment threading, and content search.
4. **Calendar and Timeline Management**
   Visual timeline with filtering (by course or event type), notifications, and conflict detection.
5. **Projects & Seminar Scheduling**
   Faculty can create, assign, and track submissions; students submit and view feedback.
6. **Document & Video Repository**
   Course-wise upload of video lectures, notes, readings, and reference links; restricted access.
7. **Real-Time Chat Module**
   One-on-one and course-wide chat system with moderation features.
8. **Deadline Conflict Detector**
   Checks for overlaps across all enrolled courses when scheduling new events.
9. **Analytics Dashboard**
   Charts for faculty engagement stats, student participation, and admin overviews.
10. **Document Versioning**
    Maintain version history for uploaded materials with download and timestamp logs.
11. **Notification Engine**
    Real-time alerts for updates, replies, new uploads, and upcoming deadlines.
12. **Secure Authentication & Delegation**
    Role-based login (JWT/OAuth), institutional email support, co-instructor/TA delegation.

Background / Tools / Technologies

Component	Recommended Stack

Frontend	React.js / Vue.js / Flutter (mobile optional)
Backend	Node.js (Express) / Django / Flask
Database	MongoDB / PostgreSQL
Real-time Chat	Socket.io / WebSockets
File Storage	Firebase Storage / AWS S3 / Cloudinary
Video Playback	Custom player or video embed (with access checks)
Notifications	Firebase Cloud Messaging / SMTP / In-app alerts
Authentication	JWT / OAuth / Institutional Email Login
Hosting & Deployment	Render / Vercel / Heroku
Version Control	Git + GitHub / GitLab
Planning & Design	Figma / Draw.io / Notion / Trello

Methodologies & Implementation Plan

Week 1: Planning, Requirement Analysis & UI/UX Design

- Conduct meetings with faculty and students to define all modules.
- Identify user roles: Student, Faculty, Admin, Teaching Assistant.
- Create SRS document covering core and advanced modules.
- Design wireframes for:
    - Profile dashboards
    - Project and calendar views
    - Content repository
    - Real-time chat interface
- Design database schema:
    - Users, Courses, Projects, Events, Files, Messages, Versions, Roles, Groups.
- Choose tech stack and initialize GitHub repo with branching strategy.

Deliverables:

- SRS + UML diagrams
- UI wireframes
- ERD + DB schema
- Repo and tool setup

Week 2: Core Backend & Frontend Development

- Set up authentication (JWT/OAuth/email login) and role assignment.
- Develop backend APIs for:
    - User management, projects, deadlines, forum posts.
- Build frontend components:
    - Login, dashboard, timeline view, document upload.
- Begin implementing:
    - File storage system
    - Document upload (with versioning logic)
- Create initial views for:
    - Course dashboard
    - Student & faculty-specific pages

Deliverables:

- Working login system
- Core API endpoints
- Frontend integration of profile/timeline views
- File upload system draft

Week 3: Advanced Modules – Real-Time & Smart Features

- **Real-time Chat Module:**
    - One-on-one and group chat per course
    - WebSocket integration using Socket.io
- **Conflict Detection Engine:**
    - When adding new deadlines, scan enrolled student timelines
    - Suggest rescheduling if conflicts detected
- **Analytics Dashboards:**
    - Faculty: student submission status, material access logs
    - Admin: global platform usage
- **Content Repository:**
    - Video + doc upload with access control
    - Version tracking per document

Deliverables:

- Chat system integrated
- Conflict engine functional
- Dashboards with dummy data
- Document versioning tested

Week 4: Final Touches, Testing & Deployment

- Implement:
    - Role-based access control on all modules
    - Notification system for key events
    - Delegation logic for TA/Co-instructor assignments
- Perform:
    - Unit + integration testing
    - Security testing (auth, file access)
    - Mock run with sample data
- Prepare:
    - Developer docs
    - User manual
    - Demo walkthrough slides
- Deploy:
    - On cloud (Render/Vercel)

Deliverables:

- Fully functional system deployed
- Tested modules + documentation
- Final presentation and recorded demo

Week	Focus Area	Key Activities
Week 1	Requirement & Design	SRS, wireframes, DB schema, role identification
Week 2	Core Development	Login, profile dashboard, forum, file handling
Week 3	Real-Time & Analytics	Chat, scheduling logic, dashboards, versioning

Week 4	Testing & Deployment	Notifications, RBAC, polish, deployment, docs

Dimension	Points	What We Look For
Technical Correctness	10	All modules functional, secure, robust
Problem–Solution Fit	15	Solves real-world academic communication gaps
Innovation / Creativity	15	Smart deadline conflict checks, real-time chat, dashboards
Code Quality & Design	5	Modular, clean code with documentation
Documentation & Report	5	SRS, ER diagrams, usage guide, clean Git history
Demo & Viva	50	Confident walkthrough, all roles demonstrated, Q&A clarity

Evaluation Criteria

Dimension	Points	What We Look For
Technical Correctness	10	All modules functional, data flows correctly
Problem–Solution Fit	15	Solves real-world problem in academic interactions
Innovation / Creativity	15	Smart UX, unique features, integration depth
Code Quality & Architecture	5	Organized, readable code, reusable components
Documentation & Report	5	SRS, wireframes, user manual, clean Git history

Demo & Viva	50	Confident presentation, working system, question handling

Project 8: Merchandise Portal Platform

Project Description

This project involves the development of a **Merchandise Portal Platform** that streamlines the process of ordering, reviewing, and distributing merchandise within a campus, company, or organization. The portal will support individual and group-based ordering, secure payments, product reviews, distribution tracking, and profile management, while ensuring privacy and intuitive UI design. It aims to automate bulk orders, department-wise coordination, and ensure timely communication via reminders.

Objectives:

1. **Simplify Merchandise Orders**
    - Enable users to browse, select, and place orders for official or themed merchandise efficiently through the portal.
2. **Facilitate Group/Department-based Orders**
    - Support bulk ordering by departments, clubs, or societies with options to manage group orders collaboratively.
3. **Integrate Secure Payment Gateways**
    - Enable seamless payments through Razorpay.
4. **Enable Review and Feedback**
    - Let users leave product reviews in both open (visible to all) or closed (admin-only) formats to improve future product selection.

5. **Manage Distribution and Logistics**
    - Track delivery schedules, distribution points, and dates with notifications/reminders.
6. **Provide Account and Profile Management**
    - Let users manage their order history, addresses, communication preferences, and payment methods securely.
7. **Privacy and Design**
    - Implement a clean, responsive, and privacy-conscious UI/UX suitable for all user roles.

Key Features:

- Home and FAQ Section for general user assistance
- Product catalog with filtering, categories, and images
- Department/Club-based order grouping
- Integrations for payments
- Product review system (open/closed modes)
- Automated email/SMS reminders for order deadlines or delivery
- Distribution detail tracker and admin dashboard
- Role-based login (User, Department Head, Admin)
- Privacy and security protocols (SSL, encrypted data, secure authentication)

Background/ Literature/ Technologies

Layer	Technology
Frontend	React.js / Vue.js / HTML + CSS + JS
Backend	Node.js (Express) / Django / Flask
Database	MongoDB / MySQL / PostgreSQL
Authentication	JWT / Firebase Auth / Session-based login
Payments	Razorpay(mandatory) / Paytm API / UPI Payment Gateway(optional)
Notifications	Email (SMTP, Nodemailer), SMS (Twilio / Msg91)
Hosting	Render / Vercel / Heroku / DigitalOcean
Version Control	Git + GitHub / GitLab

Methodologies/ Implementation Plan

Week 1: Planning & System Design

Objective: Lay the foundation by understanding requirements, designing system architecture, and preparing UI/UX layouts.

Key Activities:

- **Requirement Analysis:**
    - Understand portal needs: product catalog, payment integration, group orders, user profiles, reviews, and distribution management.
    - Stakeholder/user interviews (if applicable) to gather expectations.
- **SRS Document:**
    - Clearly define functional & non-functional requirements.

    - Define user roles (User, Group Admin, Super Admin).

- **Database Design:**
    - Design schemas for:
        - Users
        - Products
        - Orders (individual/group)
        - Payments
        - Reviews
        - Distribution & delivery schedule
    - ERD diagrams created.
- **Wireframes/UI Mockups:**
    - Design portal layout for:
        - Home
        - FAQ
        - Order Page
        - Group Order Dashboard
        - Payment Page
        - Profile Section
        - Admin Panel
    - Use tools like Figma/Draw.io.

Week 2: Core Functional Modules Development

Objective: Develop the main user functionalities like browsing, ordering, and reviewing products.

Key Activities:

Frontend Development:

- Implement product listing page with filters (type, department, price).
- Product detail page with review and rating options.
- Group order interface with department tagging.

- **Backend API Setup:**
  - User registration/login (JWT/Firebase Auth).
  - Product APIs: Add/View/Update/Delete.
  - Order APIs: Place/View orders (individual & group).
  - Review system: open (public) and closed (private to admin).
- **Group Order Management:**
  - Enable creation of department-level orders.
  - Add members to a group order and track contributions.
- **Role-Based Access Control:**
  - Different permissions for Admin, Group Lead, and User.
  - Basic dashboard structure for Admin panel.

Week 3: Payments, Notifications & Distribution Management

Objective: Integrate real-time payments, reminders, and build logic for merchandise delivery & tracking.

Key Activities:

- **Payment Gateway Integration:**
    - Integrate Razorpay for transactions.
    - Payment confirmation, failure, and retry flow.
- **Review & Rating System Completion:**
    - Fully implement review UI & backend.
    - Include admin moderation options.
- **Distribution Module:**
    - Allow admins to assign delivery locations, dates.
    - Users can track distribution status (packed, shipped, delivered).
- **Reminder System:**
    - Email/SMS notifications for:
        - Order deadlines
        - Payment confirmations
        - Pickup dates
- **Security Enhancements:**
    - Form validation
    - Input sanitization

    - Secure payment flow

Week 4: Testing, Security & Deployment

Objective: Ensure the system is reliable, secure, and ready for real-world use.

Key Activities:

- **Comprehensive Testing:**
    - Unit testing of individual components.
    - Integration testing for full order/payment/review flow.
    - Test edge cases like order failure, delayed payment, duplicate reviews.
- **Admin Dashboard Completion:**
    - Add analytics: number of orders, sales, reviews.
    - Product and order management interface.
- **Final Security Review:**
    - Apply HTTPS (if hosted externally).
    - Secure login, logout, session handling.
- **Documentation:**
    - Developer docs (API references, DB schema).
    - User manual for each role.
- **Deployment:**
    - Deploy using Heroku/Render/DigitalOcean.
    - GitHub repo cleanup and public release.

- **Demo Preparation:**
    - Create a short walkthrough presentation.
    - Prepare answers to potential viva/demo questions.

Week- Wise Plan

Week	Focus Area	Key Activities
Week 1	Planning & System Design	- Requirement analysis (group orders, payments, review flow) - UI mockups and wireframe design - Database schema (users, orders, products, reviews) - SRS Documentation
Week 2	Core Functional Modules	- Implement product catalog UI and order placement module - Develop backend APIs for user registration, profile, and orders - Create group order logic & product review module
Week 3	Payment, Review & Distribution Logic	- Integrate payment gateway (Razorpay) - Enable open/closed review system - Build module for distribution updates, delivery scheduling - Add reminders via email/SMS

Week 4	Testing, Security & Deployment	- Implement role-based login (user/admin) - Apply security (form validations, encrypted credentials) - Perform testing (unit + integration) - Final deployment & documentation

Evaluation Criteria Table

Dimension	Points	What We Look For
Technical Correctness	10	Functional code; major features like ordering, payments, and reviews work correctly.
Problem–Solution Fit	15	Clear alignment of the portal with the real-world need for merchandise distribution.
Innovation / Creativity	15	Unique features like group orders, reminders, review types, or efficient UI flows.
Code Quality & Architecture	5	Clean folder structure, reusable components, good code documentation.
Documentation & Report	5	Proper README, SRS, database schema, API documentation, and user manual.
Demo & Viva	50	Smooth, live walkthrough; clarity in explaining roles, tech stack, challenges, etc.

Project 9: Fitness Tracker Platform

Project Description

The Fitness Tracker Platform is a platform designed to help users improve their physical well-being by tracking workouts, nutrition, goals, and progress. The platform integrates essential fitness tracking functionalities with user-centric design and offline-first capabilities, encouraging healthy lifestyle habits.

Objectives

- Build an Android fitness tracking app with a clean and intuitive UI.
- Allow users to **register/login securely** and manage personal profiles.
- Enable **goal setting** for fitness (e.g., steps/day, weight goals, calories intake).
- Log **daily workouts** with types (cardio, strength), time, and calories burned.
- Track **nutrition** with a food log (macros like carbs, proteins, fats).
- Offer **personalized health tips/recommendations** based on user data.
- Include **social features** such as friend connections, group challenges, and leaderboards.
- Support **offline functionality** for logging and syncing when online.
- Prioritize **user privacy and data protection** with encrypted storage and authentication.
- Enable community challenges where users can participate in events by paying a nominal fee or registering interest. Admins can create such events or fitness challenges (e.g., weight loss race, 10K run).

Key Features

User Management

- Sign up/login via email/password or Google
- Profile setup (age, height, weight, fitness goals)
- Role-based access (admin panel optional)

Workout Logging

- Daily workout entries (type, duration, calories)
- Predefined workouts and manual entry
- Workout history and statistics

Nutrition Tracker

- Add meals/snacks with calorie count
- Macronutrient breakdown (Carbs, Protein, Fats)
- Water intake tracker

Goal Setting & Progress Monitoring

- Set goals for weight loss, strength training, or general fitness
- Progress bars and analytics to track improvements

Community & Challenges

- Create/join group challenges (e.g., 10K steps for 7 days)
- Admins can create premium or scheduled community fitness challenges with a nominal entry fee or open participation. These challenges can be tracked through leaderboards.
- Leaderboards and achievements

Analytics & Recommendations

- Graphs/charts showing weekly/monthly trends
- AI-based tips: hydration, sleep reminders, workout suggestions

Privacy & Security

- User data encryption (at-rest & in-transit)
- GDPR-style data control (clear user data option)
- Secure local database using Room DB + SQLite

Offline Mode

- App functions offline with local cache
- Syncs with server when connected

<u>Background/ Literature/ Resources</u>

Layer	**Technology**
Frontend	HTML, CSS, JavaScript, React.js
Backend	Node.js (Express) / Django / Flask
Database	MongoDB / PostgreSQL / MySQL
Authentication	Firebase Auth / Auth0 / JWT-based authentication
Hosting	Vercel / Netlify / Render
Analytics	Chart.js / D3.js / Recharts

Notifications	Web Push via Firebase
Offline Support	LocalStorage / IndexedDB
Version Control	Git, GitHub / GitLab
Deployment Tools	Docker (optional for web backend)

Methodologies & Implementation Plan

Week 1: Planning & Design

Focus: Research, UI Mockups, Architecture

- Analyze user requirements
- Create wireframes and screens (Login, Dashboard, Workout Log, etc.)
- Design database schema (Room DB + Firestore)
- Define system architecture and project structure
- Choose tech stack and initialize repository

Deliverables:

- SRS Document
- Wireframes/UI Mockups
- Database & System Design
- UML Diagrams

Week 2: Core Development – User Auth + Workout & Nutrition Modules

Focus: Key Functional Modules

- Implement Firebase authentication (email, Google)
- Build UI and backend for:
    - Profile Setup
    - Workout Logging (manual and preset)
    - Nutrition Logging (meal entry, calories, macros)
- Set up local Room DB + sync with Firestore

Deliverables:

- Working login + registration
- Workout and food logging modules
- Local + cloud sync setup
- Data persistence and error handling

Week 3: Goal Setting, Analytics & Community

Focus: Goal tracking, charts, social modules

- Implement:
    - Goal-setting screen (weight, steps, etc.)
    - Progress visualization (charts: weekly/monthly)
    - Community features: friend requests, leaderboards
    - Challenge creation and participation

Deliverables:

- Goal and Progress Tracking

- Working friend/challenge system
- Develop community challenge participation flow with optional fee tracking

- Graphs and analytics integrated

- Leaderboards functional

Week 4: Testing, Privacy & Deployment

Focus: Polish, test, deploy

- Add:
    - Alerts (e.g., daily reminders)
    - Offline logging and background sync
    - Role-based data access & encrypted local storage
- Conduct:
    - Unit and integration testing
    - UI testing on Android devices/emulators
- Finalize:
    - User manual
    - Presentation slides
    - Final app demo

Deliverables:

- Fully working app build (APK)
- All modules tested

- Final presentation/demo
- Complete documentation

Week-wise Plan

Week	Focus Area	Key Activities
Week 1	Planning & System Design	- Requirement gathering (workout, goals, nutrition modules) - SRS document preparation - Define user roles (user, admin) - Database schema & wireframe design - Select technology stack (Web/App)
Week 2	Core Module Development	- Develop user registration, login, and profile management - Implement workout logging and goal-setting modules - Enable data storage using database or local storage - Build frontend dashboards (basic screens)
Week 3	Nutrition & Visualization Modules	- Build nutrition tracker (add/edit meals, calories, macros) - Create charts for calorie tracking, workout consistency, progress - Add daily/weekly summary views - Integrate reminders/notifications -Develop community challenge participation flow with optional fee tracking
Week 4	Testing, Personalization & Launch	- Implement personalized suggestions (based on past data) - Final testing: unit, integration, usability - Bug fixing, performance tuning - Prepare documentation and deploy final version

Evaluation Criteria

Dimension	Points	What We Look For
Technical Correctness	10	Functional features, proper data handling
Problem–Solution Fit	10	Aligns with fitness goals & user needs
Innovation / Creativity	20	Unique UX, features like challenges or AI tips
Code Quality & Architecture	5	Clean structure, modular design, comments
Documentation & Report	5	README, UI design, API use, user guide
Demo & Viva	50	Smooth walkthrough, technical Q&A, performance

Project 10: Car Rental System

Project Description

The **Car Rental System** is a platform designed to improve traditional ride-hailing services by integrating smart algorithms and file-based cab location tracking to connect users with nearby drivers efficiently. Inspired by popular services like Ola and Uber

The system allows users to book rides, track drivers within a specific radius, calculate fares, and securely complete trips. It also supports file-based and/or API-based cab location handling, making it ideal for learning advanced coding techniques and building scalable transportation solutions.

Objectives

1. **User & Driver Registration and Login**
    - Role-based access: user, driver, admin.

- Secure login using JWT or session tokens.

2. **Ride Booking and Nearby Cab Matching**
    - Users can request rides by entering pickup/drop locations.
    - System identifies nearby available drivers (using GPS data or formatted location text files).
3. **File-based Location Data Handling**
    - Read location coordinates from structured text files.
    - Match riders with drivers within a defined radius using algorithms like the Haversine formula.
4. **Real-time Trip Tracking (Simulated or API-based)**
    - Simulate movement on a map or provide trip status updates.
    - Track estimated time of arrival (ETA).
5. **Fare Estimation and Invoice Generation**
    - Calculate fares based on distance and time.
    - Show trip summary, invoice, and feedback option post-trip.
6. **Admin Dashboard**
    - Manage all users, drivers, and trips.
    - View system-wide reports, statistics, and logs.
7. **Security & Session Management**
    - Secure data storage.
    - Prevent unauthorized access through role-based permissions.

8. **Feedback & Rating System**
    - Users can rate rides and drivers.
    - Ratings impact driver visibility.
9. **Payment Integration**

- Integration with Razorpay (sandbox/demo)
- Optional integration with Stripe, PayPal, or any other gateway
- Allow users to pay after trip confirmation through a secure payment portal
- Store payment status and details in the trip record

Key Features

1. User Portal:

- Sign up/log in securely.
- Search and book available cabs.
- View ride history and invoices.
- Rate and review drivers.
- Secure payments handling via Razorpay or other optional methods

2. Driver Portal:

- Accept/decline ride requests.
- Update location (auto/manual).
- View upcoming and past rides.
- Edit profile and availability.

3. Admin Portal:

- Manage user and driver accounts.
- Monitor ongoing and completed trips.
- Handle complaints and reports.
- Oversee system analytics and logs.

Background/ Literature/ Resources

Layer	Technology
Frontend	React.js / Vue.js / Flutter (if mobile)
Backend	Node.js (Express) / Flask / Django
Database	MongoDB / PostgreSQL / MySQL
Authentication	Firebase Auth / JWT / OAuth2.0
Location Services	Google Maps API / OpenStreetMap / File-based GPS input
Payment (Mock)	Razorpay sandbox / Stripe (optional)
Testing Tool	Postman / Jest / PyTest
Hosting & Deployment	Vercel / Heroku / Render / Docker (optional)
Version Control	Git + GitHub / GitLab

Methodologies & Implementation Plan

Week 1: Planning & System Design

Focus: Requirement finalization, architecture setup, and design
Key Activities:

- Conduct requirement gathering and finalize core features (user booking, driver matching, etc.)

- Prepare the **Software Requirements Specification (SRS)** document
- Design the **database schema** including tables for Users, Drivers, Rides, Locations, Feedback, etc.
- Define the structure of **text files** for storing mock GPS coordinates
- Design **wireframes and UI mockups** for all modules (User, Driver, Admin)
- Plan and finalize the system architecture including the backend framework, APIs, and data flow
- Plan Razorpay integration flow and other if any

Week 2: Backend Development & Core Logic

Focus: Develop APIs, file parsing logic, and cab matching algorithm
Key Activities:

- Set up backend framework using Node.js / Django / Flask
- Build **user and driver registration & login APIs** with role-based access
- Develop **cab location reader**: parse structured text files to read GPS data
- Implement **distance calculation logic** (e.g., using the Haversine formula)
- Implement Razorpay server-side payment API integration
- Create **API endpoints** for booking rides, searching for nearby drivers, calculating fare
- Add backend support for storing and updating trip and user data

Week 3: Frontend & Ride Management Workflows

Focus: Build frontend interface and integrate with backend
Key Activities:

- Create user interface for User Dashboard (book ride, view rides, fare)
- Create driver dashboard (accept/decline rides, trip status update)
- Integrate all APIs for real-time data fetch and trip updates
- Implement **ride tracking flow** using either simulated or static cab movement
- Add map or coordinate-based interface to show driver and rider locations
- Razorpay checkout integration on frontend
- Implement basic **fare summary and ride invoice generation**

Week 4: Testing, Security, Feedback System & Deployment

Focus: Final touches, quality assurance, and demo readiness
Key Activities:

- Implement **feedback and rating system** for users and drivers
- Add **admin dashboard** to manage users, drivers, and system logs
- Integrate **basic authentication and session security** (JWT/Firebase/Auth0)
- Conduct **unit, integration, and end-to-end testing**
- Fix bugs, refine UI/UX, improve performance
- Prepare **project documentation**, README, API guide, and deployment steps
- Final deployment on hosting platform (Heroku, Vercel, etc.)
- Prepare **final presentation**

Week-wise plan

Week	Focus Area	Detailed Key Activities
Week1	Planning & System Design	- Finalize use cases and requirements - Draft SRS document - Design database schema - Create wireframes/UI mockups - Decide cab location file format -payment flow planning
Week 2	Backend & Location Logic	- Setup backend framework and routes - Implement file parser for location data - Write function to calculate distance - Create APIs for booking, driver search, fare estimate
Week 3	Frontend & Booking Workflows	- Build UI for users and drivers (login, dashboard) - Integrate backend APIs - Develop booking and trip flow - Add trip simulation (manual refresh or animation) -razorpay integration
Week 4	Testing, Feedback & Deployment	- Implement ride history, feedback system - Admin dashboard setup - Conduct full flow testing - Fix bugs, polish UI - Final documentation and demo preparation

Evaluation Criteria

Dimension	Points	What We Look For
Technical Correctness	10	Functional features, proper data handling
Problem–Solution Fit	10	Aligns with fitness goals & user needs

Innovation / Creativity	20	Payment integration, Unique UX, features like challenges or AI tips
Code Quality & Architecture	5	Clean structure, modular design, comments
Documentation & Report	5	README, UI design, API use, user guide
Demo & Viva	50	Smooth walkthrough, technical Q&A, performance