Project Plan Document

Name1:Swetha R & SRN1:PES1UG23CS833

Name1:Thrisha M & SRN1:PES1UG23CS834

1. Project Title

Car Rental System

2. Lifecycle Model

Lifecycle Model: Agile Methodology

Justification: The project requires flexibility and iterative improvements due to the nature of customer

interactions, feedback, and system integration with third-party services (e.g., payment

gateways,GPS).

Agile allows us to adapt quickly and deliver working versions of the Car Rental System at

regularintervals,

ensuring stakeholder involvement and satisfaction.

3. Tools Used

- Planning Tools: Jira for task management, Gantt charts for scheduling.

- Design Tools: Lucidchart for system diagrams (Use Case, Data Flow).

- Version Control: Git/GitHub to manage code repositories and track changes.

- Development Tools: Java (Spring Boot for back-end), React.js for front-end development.

- Bug Tracking: Jira for issue tracking.

- Testing Tools: Selenium for automated testing, JUnit for unit testing.

4. Deliverables (Reusable/Build Components)

- Reusable Components:

- Customer registration module.

- Payment processing integration

- REST APIs for vehicle inventory and reservation.

- Build Components:

  - UI/UX for customer and admin portals.

  - Business logic for vehicle reservation and rental management.

  - Integration with GPS for vehicle tracking.

5. Work Breakdown Structure (WBS)

Type: Deliverable-Oriented WBS

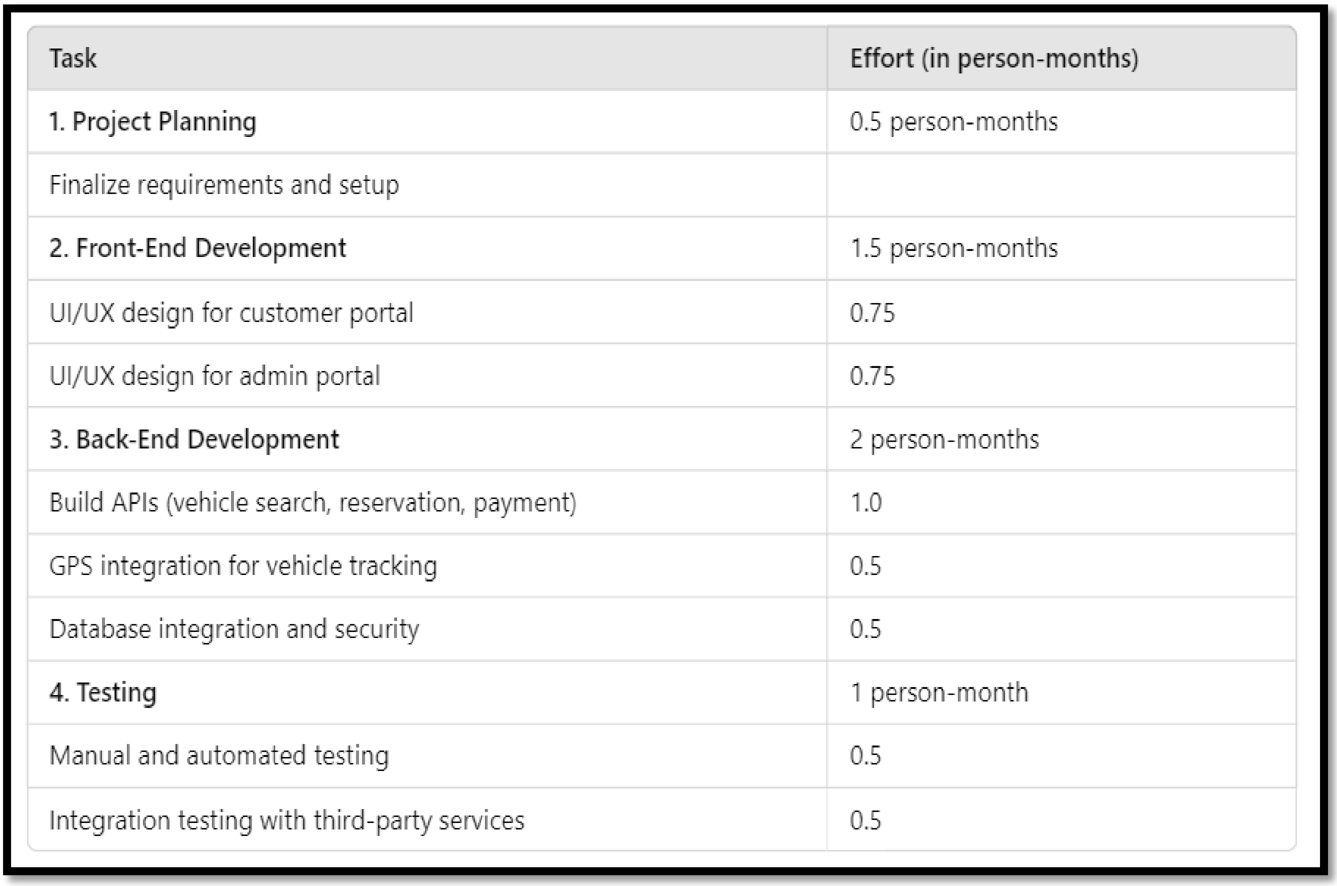
Justification: This type is most suitable since each phase (UI/UX, payment, reservation) is distinctand can be delivered independently, aligning with the agile sprints.

WBS Structure:

1. Project Planning and Initial Setup
2. Front-End Development
3. Back-End Development
4. Testing and Deployment

6. Effort Estimation (in person-months)

Effort estimation helps to calculate the amount of work needed for each phase of the project. Belowis a breakdown of the tasks and estimated effort using the COCOMO model for software developmenteffort estimation:



Task	Effort (in person-months)
1. Project Planning	0.5 person-months
Finalize requirements and setup	
2. Front-End Development	1.5 person-months
UI/UX design for customer portal	0.75
UI/UX design for admin portal	0.75
3. Back-End Development	2 person-months
Build APIs (vehicle search, reservation, payment)	1.0
GPS integration for vehicle tracking	0.5
Database integration and security	0.5
4. Testing	1 person-month
Manual and automated testing	0.5
Integration testing with third-party services	0.5

□ **Total Estimated Effort**: **5 person-months**

- Based on a team of 3-4 developers, you can adjust the timeline accordingly if more team members are involved.

□ **COCOMO Model Assumption:**

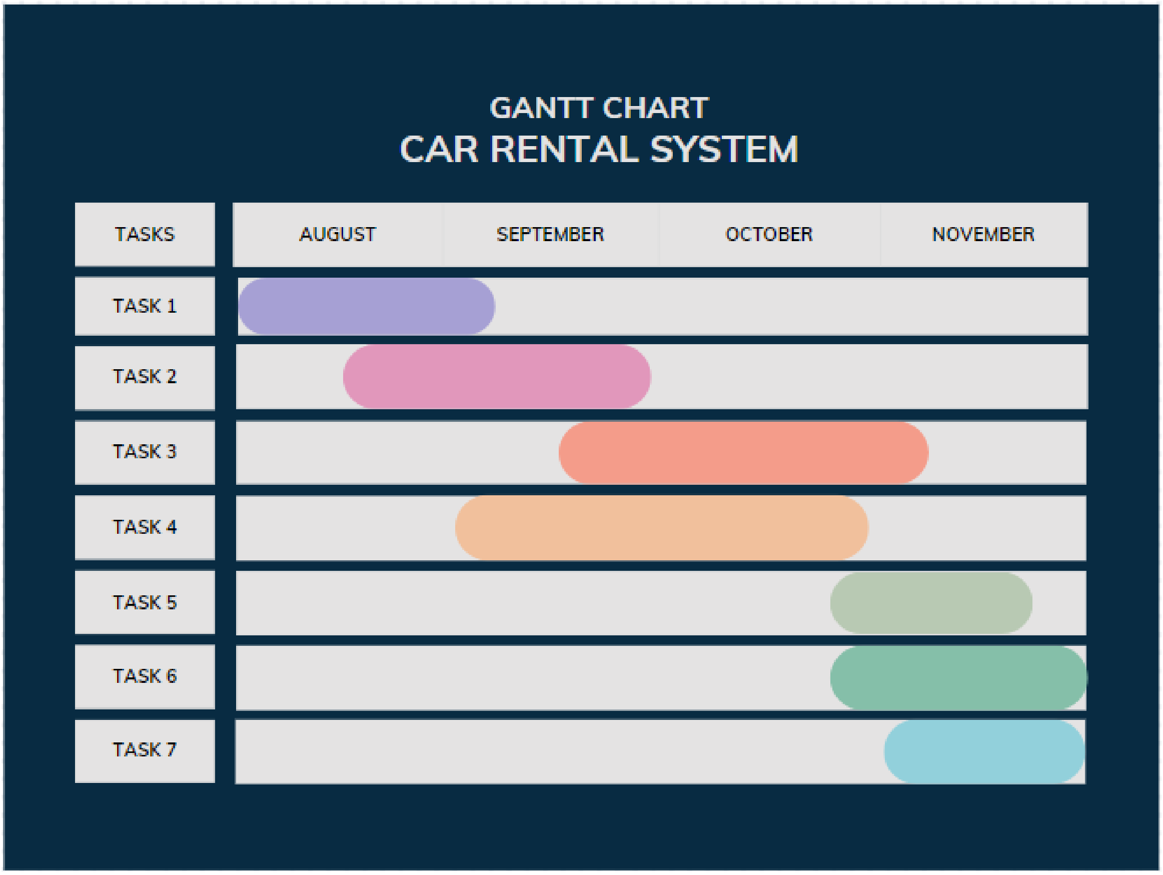
- **Size of the Project:** 15 KLOC (thousand lines of code).
- **Type of Project:** Intermediate-level complexity.

- **Effort Formula:** Effort = 2.94 * (Size)^1.099, where size is in KLOC. Effort multipliers for intermediate projects have been used based on the assumed size of your project.

7. Gantt Chart (September to November)

A Gantt chart provides a visual representation of the project timeline, indicating the start and end dates of each task and their dependencies.

1. Project planning and requirements gathering – August 1 to August 14 (2 weeks)
2. UI/UX design (customer and admin portals) – August 15 to August 30 (2 weeks)
3. Front-end development (customer and admin portals) – September 1 to September 21 (3 weeks)
4. Back-end development – September 22 to October 5 (2 weeks)
5. Database integration and security – October 6 to October 14 (1 week)
6. Testing (unit, integration, and system) – October 15 to October 31 (2 weeks)
7. Final deployment and review – November 1 to November 7 (1 week)



8. Risk Management Plan

- **Identify Risks:**
  - Integration issues with third-party services like payment gateways or GPS systems.
  - Potential delays in the development process due to unforeseen complexities.
  - Security vulnerabilities, especially with payment processing and customer data.
- **Mitigation Strategies:**
  - Set up regular check-ins with third-party service providers.
  - Ensure modular development to quickly identify and resolve issues.
  - Conduct thorough security testing and implement encryption for sensitive data.

9. Resource Allocation

- **Team Members:**
  - Assign specific tasks to each team member based on their skill set (e.g., front-end, back-end, testing).
- **Hardware/Software:**
  - Ensure all necessary tools (e.g., IDEs, servers for testing) are available to team members from the start.
  - Set up a CI/CD pipeline for continuous testing and integration.

10. Communication Plan

- **Internal Communication:**
  - Set up regular team meetings (e.g., weekly stand-ups) to review progress and address issues.
- **External Communication:**
  - Establish communication channels with stakeholders to provide regular project updates.

Architectural Document for Car Rental System

1. Introduction

- Purpose: This document outlines the system architecture for the Car Rental System (CRS), highlighting its key components, interactions, and deployment strategies. It serves as a guide for development and implementation teams.
- Scope: The Car Rental System facilitates vehicle rental services, managing vehicle reservations, customer interactions, and rental transactions. The architecture ensures scalability, security, and ease of use across multiple platforms.
- Definitions, Acronyms, and Abbreviations:
  - CRS: Car Rental System
  - API: Application Programming Interface
  - UI: User Interface
  - DB: Database
- References: IEEE Std 1471-2000 for architectural descriptions.

2. Architectural Representation

The system follows a layered architecture, including the presentation, business logic, and data access layers. It uses a RESTful service architecture for backend services and a web/mobile front end for customers and administrators.

3. Architectural Goals and Constraints

- Goals: The system is designed for high availability, data security, and ease of integration with third-party services (e.g., payment gateways, GPS).
- Constraints: It must comply with local vehicle rental regulations, support secure transactions (SSL/TLS), and handle real-time data for vehicle availability.

4. Use-Case View

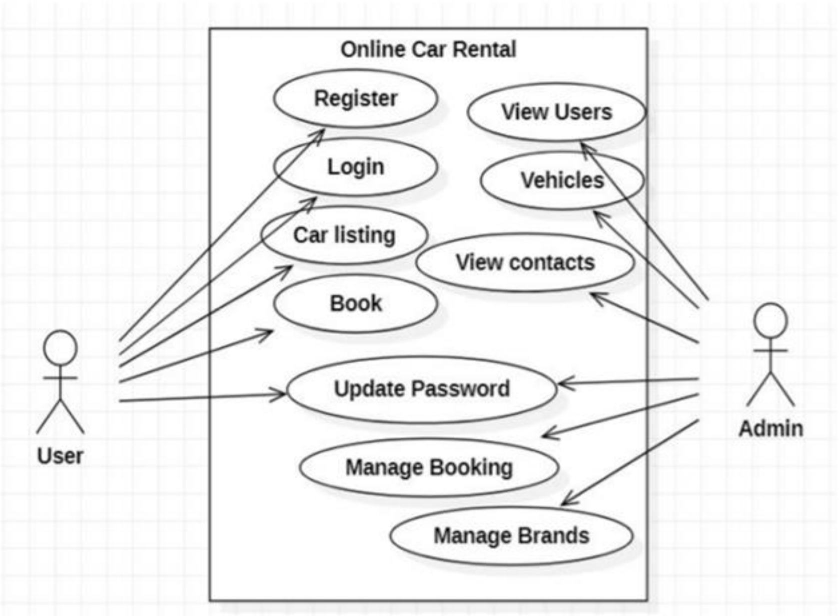
- Architecturally-Significant Use Cases:

  - Customer vehicle search and reservation.

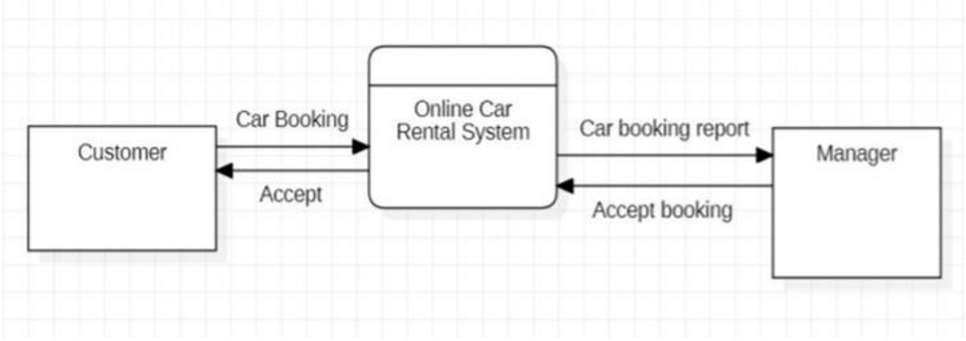
  - Administrator management of vehicle inventory.

  - Secure payment processing.

USE CASE DIAGRAM



DATA FLOW DIAGRAM



5. Logical View

- Architecture Overview: The system is divided into a presentation layer (web/mobile UI), business logic layer (vehicle reservation, rental management), and a data access layer (interfacing with the database for customer, vehicle, and transaction records).

- Process View:

  1. Processes: Vehicle reservation, payment, vehicle return.

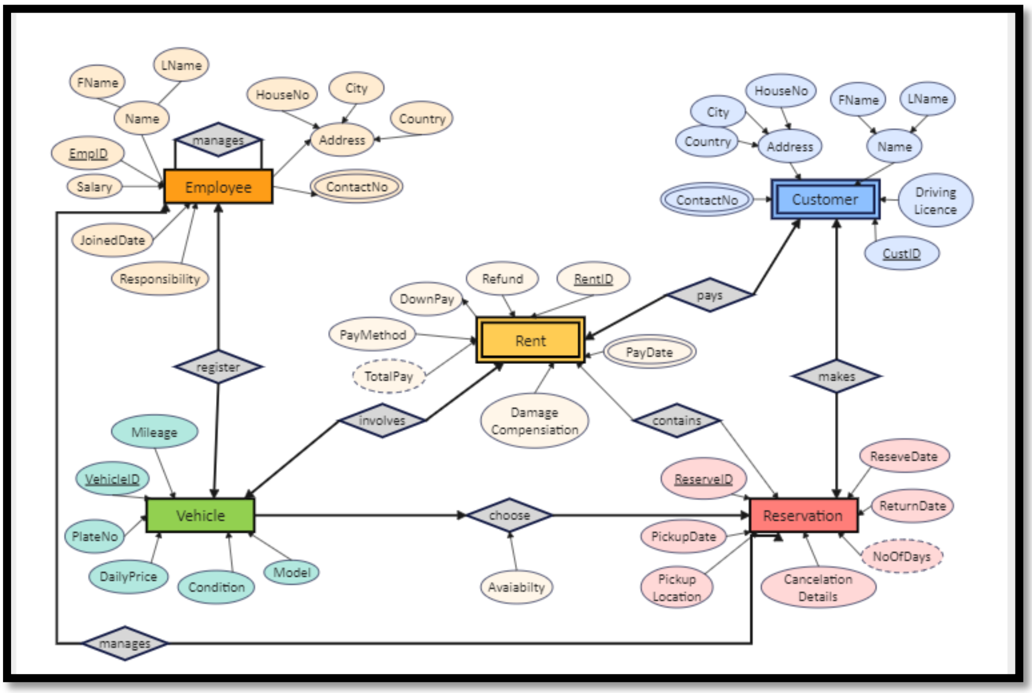
2. Process to Design Elements: Reservation handled via the business logic layer and interactions with the DB.

3. Process Model to Design: Customers use the UI to search, triggering APIs that query the database.

4. Model Dependencies: Inventory and customer data dependency for rental and return actions.

5. Processes to the Implementation: Implemented as RESTful services interacting with databases and third-party services (e.g., payment gateways).

ER-DIAGRAM



6. Deployment View

- External Desktop PC: Customer and admin access via web browsers.

- Desktop PC: Backend servers for the CRS application.

- Registration Server: Handles customer registration and authentication.

- Course Catalog (in CRS): Vehicle catalog for customer queries.

- Billing System: Integrated payment gateway for secure transactions.

7. Performance

The system must handle up to 1,000 concurrent users, with response times under 2 seconds for standard queries and transactions.

8. Quality

- Security: Encryption (SSL/TLS) for all data transactions, multi-factor authentication for admin users.
- Reliability: 99.9% uptime, with regular data backups and maintenance notifications.