

PROJECT REPORT

ON

# ISE TASK-2

**Comprehensive System Design and Agile Implementation for a Ride-Hailing Application**

Submitted to

**NMAM Institute of Technology, Nitte**

(Off-Campus Centre, Nitte Deemed to be University, Nitte – 576110, Karnataka, India)

On partial fulfilment of the requirements for the award of the

Degree of Bachelor of Technology

In

INFORMATION SCIENCE AND ENGINEERING

By

**Akash G**

**USN: NNM23IS008**

Under the guidance of

**Dr. JASON ELROY MARTIS, Associate Professor**

Department of Information Science and Technology,

NMAM Institute of Technology, Nitte, Karnataka,

India

Abstract

This project designs a ride-hailing app (like Uber) using system modelling to visualize its structure and workflows, paired with Agile practices (Scrum) to ensure iterative, user-focused development. The system models—context, interaction, and behavioural diagrams—map how the app interacts with external services, processes ride bookings, and manages ride lifecycles. The Agile plan breaks the app into sprints, prioritizes user stories for passengers and drivers, and defines clear quality checks. Together, these methods ensure a scalable, user-friendly app built collaboratively and adaptively.

Introduction

Ride-hailing apps are complex systems involving real-time coordination between passengers, drivers, and external services (like maps or payments). To build such an app efficiently, we need two things:

**System Models:** Visual blueprints to simplify complexity.

**Agile Practices: A** flexible development process that adapts to user needs.

This document explains:

How the app interacts with the outside world (context model).

The step-by-step flow of booking a ride (interaction model).

The journey of a ride from booking to completion (behavioural model).

A Scrum-based plan to build the app incrementally, with user stories and quality checks.

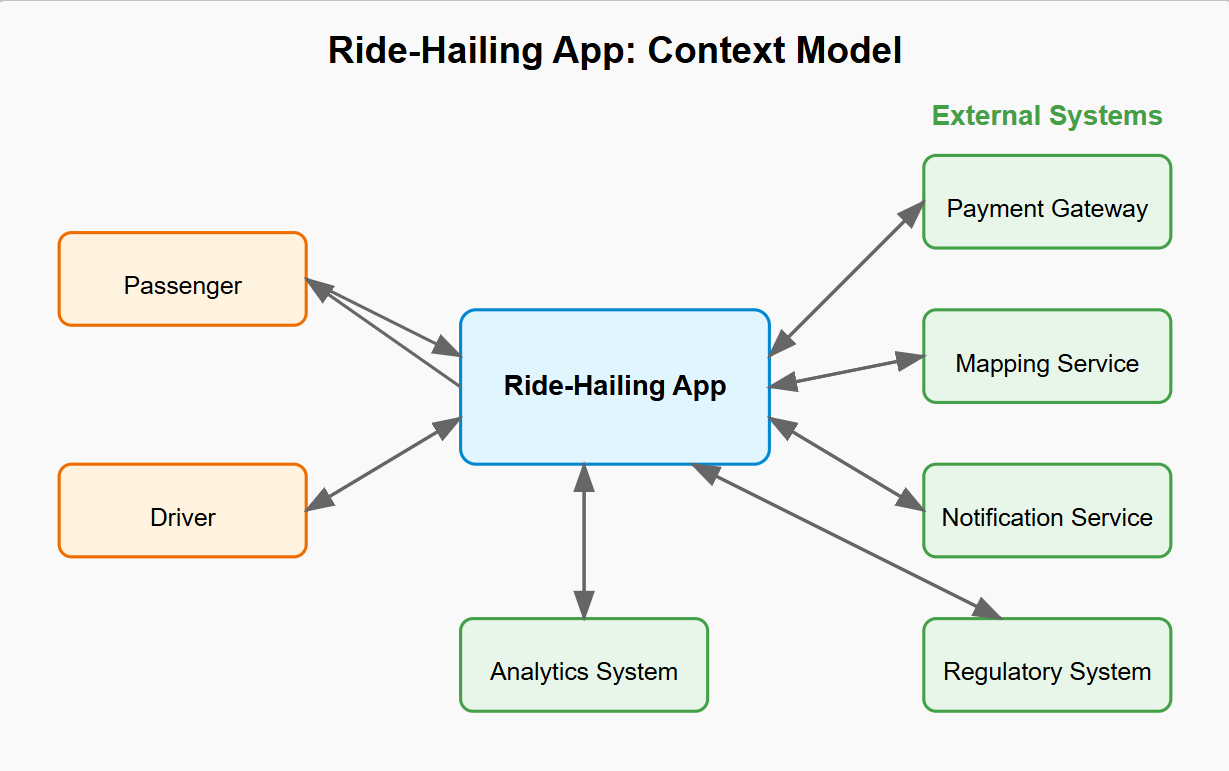
Think of this as building a house: the models are the architectural plans, and Agile is the construction crew working in phases to avoid mistakes and delays.

**Detailed Explanation of Models:**

**1. Context Model:**

What It Shows: The app’s "big picture" relationships with external systems.

Why It Matters: Just like Uber relies on Google Maps for navigation, our app depends on external tools to function.



**Key Components:**

Payment Gateway (e.g., Stripe): Handles transactions securely.

GPS/Map Service (e.g., Google Maps): Provides real-time navigation and driver tracking.

SMS/Email Service (e.g., Twilio): Sends ride confirmations and alerts.

User Devices: Passenger and driver apps (Android/iOS).

Vehicle Registry: Verifies driver licenses and car details.

**Example Workflow:**

When a passenger books a ride, the app uses Google Maps to find nearby drivers.

After the ride, Stripe charges the passenger’s card and splits the payment between the driver and app.

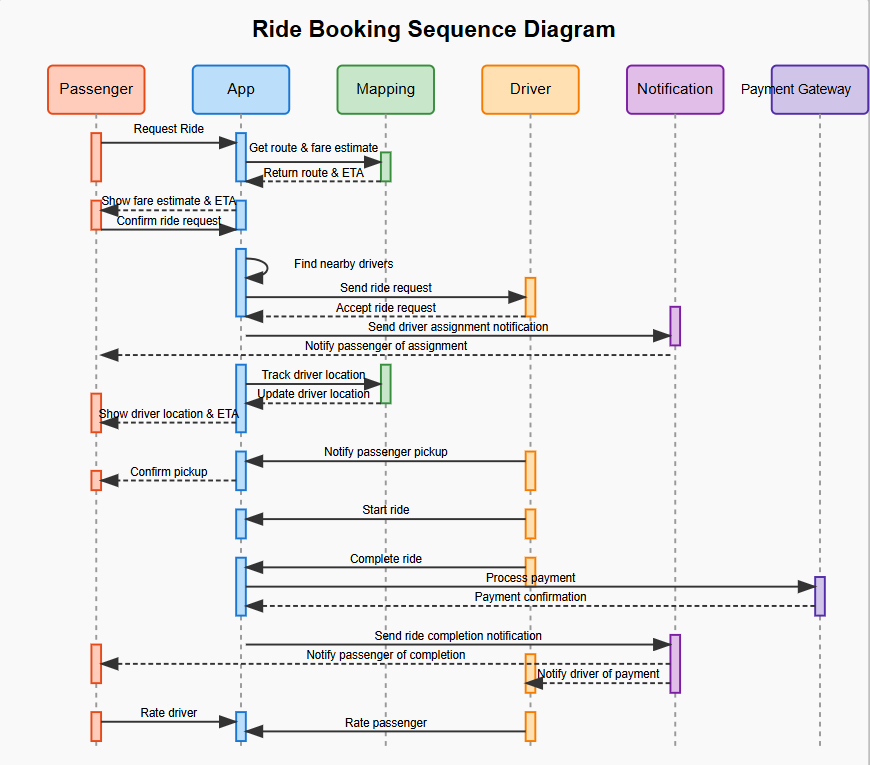
**2. Interaction Model:**

What It Shows: The step-by-step process of booking a ride, like a movie script.

Why It Matters: Ensures all parts of the system "talk" to each other smoothly.

Sequence Diagram Steps:

Passenger Action: Opens app → enters destination → clicks “Book Ride.”



**App’s Response:**

Fetches nearby drivers using GPS data.

Sends ride request to the closest available driver.

Driver Action: Accepts/declines the request via their app.

**Confirmation:**

App notifies passenger of driver details and ETA.

Driver navigates to pick up location using Google Maps.

**Post-Ride:**

App calculates fare → processes payment via Stripe.

Passenger and driver rate each other.

Why This Flow Works:

Clear handoffs between passenger, app, driver, and external services.

Mimics real-world interactions (e.g., hailing a taxi but digitally).

**3. Behavioral Model**

What It Shows: The ride’s lifecycle, like a timeline of states and transitions.

Why It Matters: Rides aren’t static—they change states (e.g., canceled, completed).

**State Diagram Breakdown:**

Requested: Passenger submits a ride request.

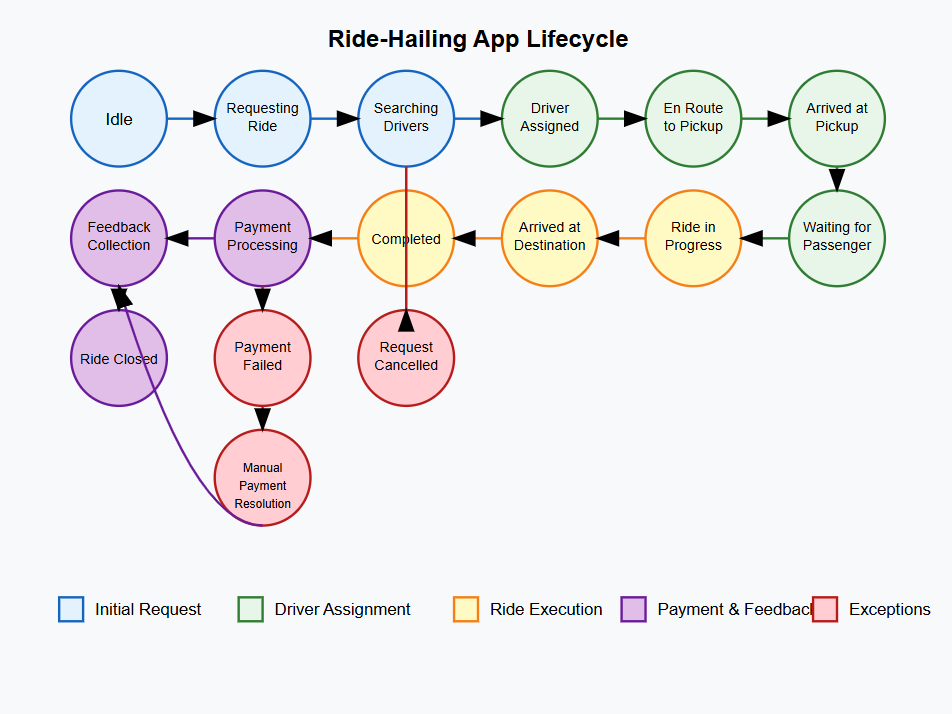
Driver Assigned: App matches a driver.

En Route to Pick up: Driver heads to passenger.

Passenger Picked Up: Ride starts.

En Route to Destination: Driver heads to the drop-off.

Completed: Ride ends → payment processed.



**Edge Cases:**

Cancelled: Driver/passenger cancels mid-request.

Payment Retry: Payment fails → app retries or alerts user.

Example:

If a driver cancels after accepting, the ride returns to the “Requested” state for a new match.

Agile Practices: Scrum Plan

Sprint Structure

Sprint Duration: 2 weeks (iterative cycles).

**Key Sprints & Deliverables:**

**Sprint 1:** User authentication (login/signup).

**Sprint 2:** Ride booking interface + driver matching algorithm.

**Sprint 3**: Real-time GPS tracking for drivers/passengers.

**Sprint 4:** Payment integration (Stripe/PayPal).

**Sprint 5:** Rating system + ride history dashboard.

Why Scrum Works:

Focuses on small, testable features (e.g., “Payment in Sprint 4” vs. building everything at once).

Allows feedback after each sprint (e.g., testing GPS tracking before moving to payments).

**Passenger Stories:**

**Sign Up/Login:**

“As a passenger, I want to create an account to book rides.”

DoD: Email validation, secure password storage, profile saved in the database.

**Fare Estimation:**

“As a passenger, I want to see a fare estimate before booking.”

DoD: Fare calculator using distance, time, and surge pricing.

**Ride Cancellation:**

“As a passenger, I want to cancel a ride if the driver is late.”

DoD: Cancel button with refund policy enforcement.

Driver Stories:

Availability Toggle:

“As a driver, I want to go online/offline to accept rides.”

DoD: Toggle button updates driver status in real-time.

Earnings Dashboard:

“As a driver, I want to see my daily/weekly earnings.”

DoD: Dashboard with filters for dates, ride types, and payout totals.

**Navigation Integration:**

“As a driver, I want in-app navigation to reach passengers quickly.”

DoD: Google Maps/Apple Maps embedded with turn-by-turn directions.

Agile: Sprint backlogs, user stories, DoD.

Team member roles (e.g., “Alice: Backend Developer”).

Commit history (e.g., “Fixed payment API bug – May 5”).

Conflict resolution notes (e.g., “Dispute over map service: Team voted for Google Maps after cost analysis”).

**Summary:**

System Models act as the app’s blueprint, ensuring all parts work together seamlessly:

**Context Model:** Who the app interacts with (payment, maps, etc.).

**Interaction Model:** How a ride is booked (step-by-step flow).

**Behavioural Model:** The ride’s journey from start to finish.

**Agile Practices keep the project on track:**

Sprints break the app into manageable pieces (e.g., authentication first, payments later).

User Stories prioritize passenger/driver needs (e.g., fare estimates, earnings tracking).

Definition of Done ensures quality (no half-built features).