PARKSY – Peer-to-Peer Parking Platform

**Version:** 1.0  
**Date:** 11 Aug 2025  
**Author(s):** Avinash Bagul  
**Contact:** bagulavinash74@gmail.com/ 9370941552

# Table of Contents

[1. Introduction 1](#_Toc205927758)

[2. Project Scope 1](#_Toc205927759)

[3. Problem Statement 2](#_Toc205927760)

[4. Objectives 2](#_Toc205927761)

[5. System Overview 2](#_Toc205927762)

[6. Requirements 3](#_Toc205927763)

[7. System Design 3](#_Toc205927764)

[8. Implementation Plan 4](#_Toc205927765)

[9. Testing Strategy 4](#_Toc205927766)

[10. Deployment Plan 4](#_Toc205927767)

[11. Maintenance & Future Scope 4](#_Toc205927768)

[12. Financial Overview 5](#_Toc205927769)

[13. References & Appendices 5](#_Toc205927770)

# 1. Introduction

**1.1 Project Background**

Parking problems in urban cities lead to traffic congestion, wasted time, and environmental impact. Meanwhile, large parking spaces in malls, office complexes, marriage halls, and private residences remain unused for hours.  
**ParkSy** solves this problem by connecting parking owners with drivers via a real-time web platform.

**1.2 Purpose of the Document**

This document describes ParkSy’s goals, system design, implementation plan, and deployment strategy to guide development, testing, and maintenance.

# 2. Project Scope

**2.1 In Scope**

* User registration/login
* Posting available parking spots
* Map integration for location tracking & navigation
* Booking & payment system
* Owner dashboard & booking history

# 3. Problem Statement

Finding parking in metro areas can take more than 20 minutes, while idle private spaces exist nearby. This mismatch results in wasted resources and increased urban congestion.

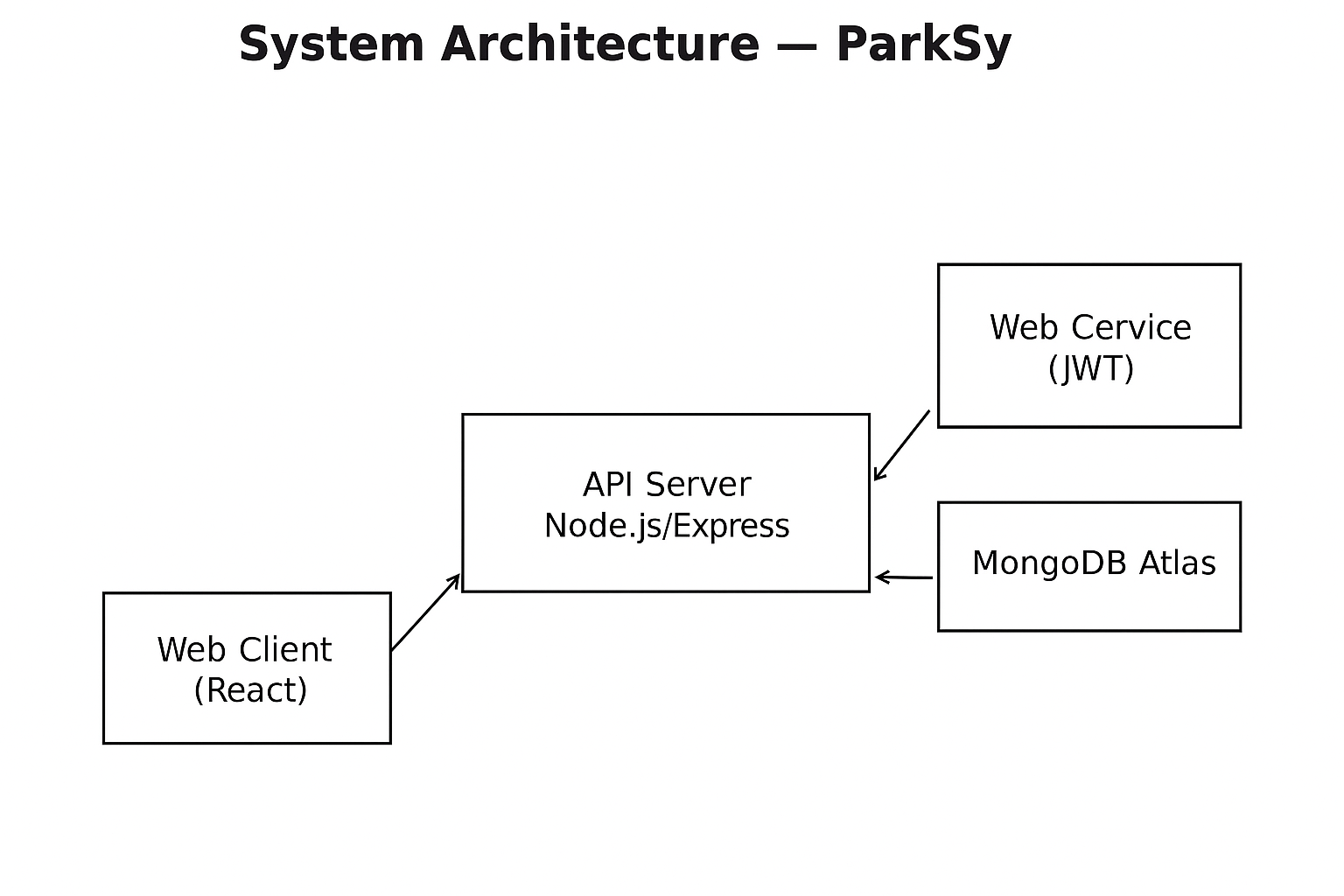
# 4. Objectives

* Enable space owners to earn from unused spots
* Provide drivers with quick, reliable parking booking
* Reduce traffic congestion
* Offer real-time navigation from current location to parking space

# 5. System Overview

**5.1 Technology Stack**

* **Frontend:** React.js (UI, Google Maps API integration)
* **Backend:** Node.js + Express (REST API, authentication)
* **Database:** MongoDB Atlas (cloud-hosted)
* **Hosting:**
  + Frontend:
  + Backend:
* **Payments:**
* **5.2 System Architecture Diagram**



# 6. Requirements

**6.1 Functional Requirements**

* FR1: User can register/login
* FR2: Owner can list parking spot with price/hour
* FR3: Driver can search and book available spots on map
* FR4: Integrated payment system
* FR5: Booking confirmation & route navigation

**6.2 Non-functional Requirements**

* NFR1: Response time < 2 seconds
* NFR2: Support 100+ concurrent users
* NFR3: Password encryption (bcrypt)
* NFR4: Mobile-friendly design

# 7. System Design

**7.1 Entity Relationship (ER) Diagram Placeholder**

**Entities & Attributes for ParkSy**

**1. User**

Stores information about both parking space owners and drivers.  
**Attributes:**

* **user\_id** (PK) – Unique identifier
* **full\_name** – Full legal name
* **email** – Unique email address (login credential)
* **password\_hash** – Encrypted password
* **phone\_number** – Contact number
* **role** – owner / driver / admin
* **profile\_picture** – Image URL (optional)
* **date\_joined** – Account creation date
* **last\_login** – Last active timestamp
* **status** – active / inactive / banned

**2. ParkingSpace**

Represents each available parking spot listed by owners.  
**Attributes:**

* **parking\_space\_id** (PK) – Unique identifier
* **owner\_id** (FK → User.user\_id) – Linked owner
* **title** – Short name/label for space
* **description** – Detailed description
* **location\_address** – Full address
* **latitude** – GPS coordinate
* **longitude** – GPS coordinate
* **price\_per\_hour** – Rate in local currency
* **price\_per\_day** – Optional daily rate
* **is\_available** – Boolean availability status
* **photo\_url** – Image(s) of the space
* **created\_at** – Listing creation date
* **updated\_at** – Last update date

**3. Booking**

Stores data about reservations made by drivers.  
**Attributes:**

* **booking\_id** (PK) – Unique identifier
* **driver\_id** (FK → User.user\_id) – Driver who booked
* **parking\_space\_id** (FK → ParkingSpace.parking\_space\_id) – Reserved space
* **start\_time** – Booking start datetime
* **end\_time** – Booking end datetime
* **total\_price** – Calculated total amount
* **status** – pending / confirmed / cancelled / completed
* **created\_at** – Booking creation date

**4. Payment**

Handles all payment transactions for bookings.  
**Attributes:**

* **payment\_id** (PK) – Unique identifier
* **booking\_id** (FK → Booking.booking\_id) – Linked booking
* **amount** – Transaction amount
* **payment\_method** – e.g., UPI, CreditCard, PayPal
* **payment\_status** – pending / paid / failed
* **transaction\_id** – Reference from payment gateway
* **paid\_at** – Timestamp of payment completion

**5. Review**

Stores user feedback on parking spaces.  
**Attributes:**

* **review\_id** (PK) – Unique identifier
* **parking\_space\_id** (FK → ParkingSpace.parking\_space\_id) – Related space
* **reviewer\_id** (FK → User.user\_id) – Reviewer
* **rating** – Integer (1–5)
* **comment** – Optional text feedback
* **created\_at** – Review date

**6. Notification**

Keeps track of system-generated alerts for users.  
**Attributes:**

* **notification\_id** (PK) – Unique identifier
* **user\_id** (FK → User.user\_id) – Recipient
* **message** – Notification text
* **type** – booking\_update, payment\_status, general
* **is\_read** – Boolean read/unread status
* **created\_at** – Timestamp

**7. AdminAction**

For tracking admin-level actions in case of disputes or maintenance.  
**Attributes:**

* **admin\_action\_id** (PK) – Unique identifier
* **admin\_id** (FK → User.user\_id) – Admin performing action
* **target\_user\_id** (FK → User.user\_id) – Affected user
* **action\_type** – ban\_user, approve\_listing, delete\_review
* **reason** – Explanation for action
* **created\_at** – Timestamp

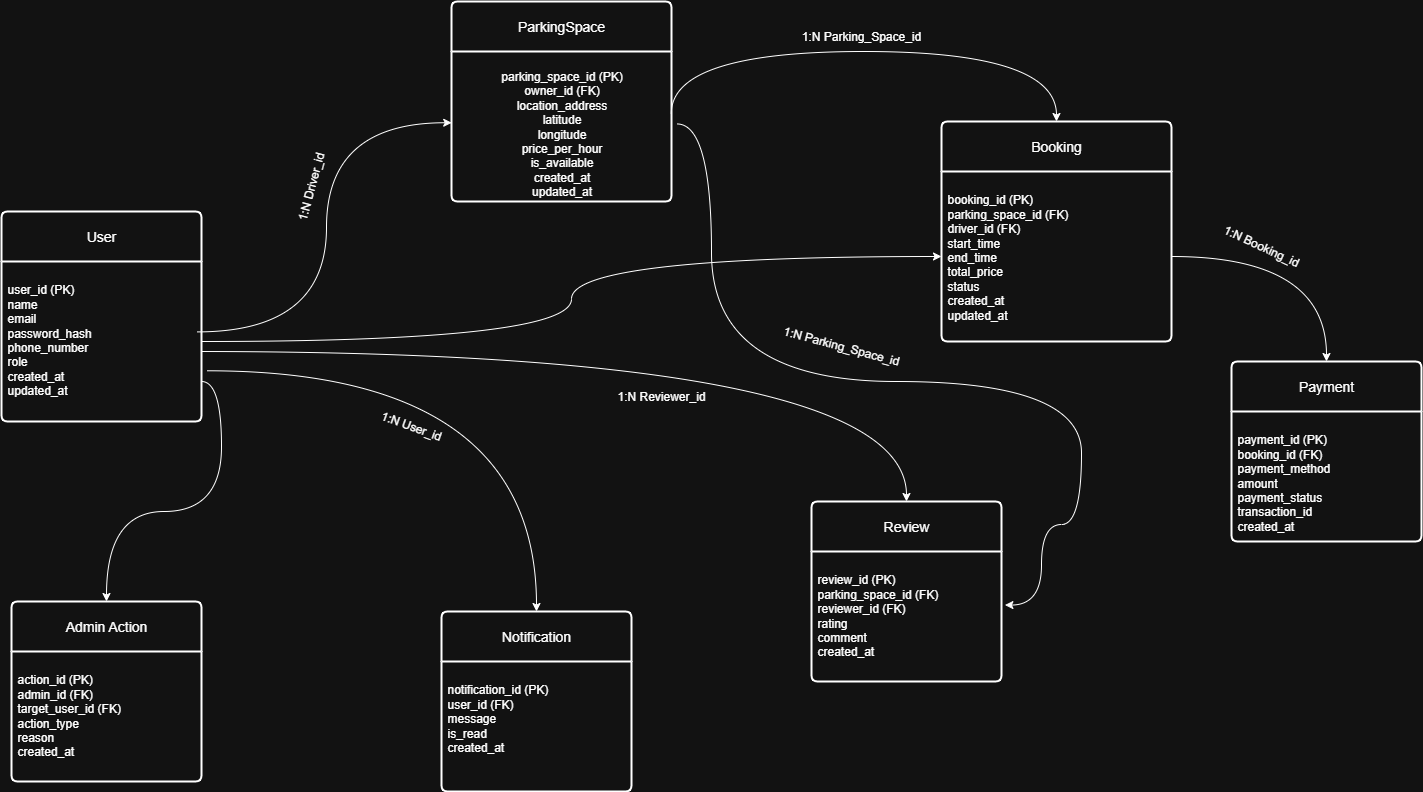
**8. Vehicle (Optional but Recommended)**

To store driver’s vehicle details for better booking and enforcement.  
**Attributes:**

* **vehicle\_id** (PK) – Unique identifier
* **user\_id** (FK → User.user\_id) – Owner of the vehicle
* **vehicle\_number** – Registration plate
* **vehicle\_type** – car, bike, truck, etc.
* **model** – Vehicle model
* **color** – Vehicle color

**Relationships Overview**

* **User ↔ ParkingSpace**: 1 owner can list multiple spaces (1:N)
* **User ↔ Booking**: 1 driver can make multiple bookings (1:N)
* **ParkingSpace ↔ Booking**: 1 parking space can have many bookings (1:N)
* **Booking ↔ Payment**: 1 booking has exactly 1 payment (1:1)
* **ParkingSpace ↔ Review**: 1 space can have multiple reviews (1:N)
* **User ↔ Review**: 1 user can write many reviews (1:N)
* **User ↔ Notification**: 1 user can receive many notifications (1:N)
* **User ↔ Vehicle**: 1 user can have multiple vehicles (1:N)
* **User ↔ AdminAction**: Admin can perform multiple actions (1:N)



**7.2 API Endpoints Table**

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /auth/register | POST | Register a new user |
| /auth/login | POST | Login user |
| /spots | GET | List all available spots |
| /spots/:id | GET | Get details of a spot |
| /booking | POST | Create a booking |
| /payment | POST | Process payment |

# 8. Implementation Plan

**8.1 Development Phases**

1. Environment setup & Git repo initialization
2. Backend development (APIs, authentication)
3. Database schema creation
4. Frontend UI implementation (React, Google Maps API)
5. Payment gateway integration
6. Testing (unit, integration, user)
7. Deployment

# 9. Testing Strategy

* **Unit Testing:** Jest/Mocha for backend, React Testing Library for frontend
* **Integration Testing:** Postman API tests
* **User Acceptance Testing:** Beta testers from target audience

# 10. Deployment Plan

**10.1 Hosting**

* **Frontend:** Netlify (connect GitHub repo, auto-deploy)
* **Backend:** Render (deploy via GitHub)
* **Database:** MongoDB Atlas (free shared cluster)

# 11. Maintenance & Future Scope

* Regular updates and bug fixes
* IoT gate control system
* AI-driven demand-based pricing
* Multi-city expansion