1. Introduction

The rapid urbanization of cities has led to a critical shortage of parking spaces, causing signifi- cant traffic congestion, fuel wastage, and environmental degradation. This project, titled ”Digi- tal Twin-Based Smart Parking System,” aims to bridge the gap between physical infrastructure and digital management. By creating a high-fidelity virtual replica (Digital Twin) of a physical parking area, we provide drivers with real-time occupancy data and administrators with pow- erful simulation and management tools. This Phase 2 report details the requirements identifi- cation, architectural design, and system modeling essential for the upcoming implementation phase.

2. Functional and Non-Functional Requirement Identification:

To ensure the project reflects industry-level practices and high practical usability, the following requirements have been identified through comprehensive analysis:

Functional Requirements (FR):

Matrix-Based Grid Mapping of Parking Slots:

The system must represent the entire parking lot using a 2D matrix-based coordinate grid structure.
Each parking slot shall be modeled as a coordinate point (x, y), enabling a structured digital representation.
Each slot must contain binary status values:
0 – Vacant
1 – Occupied
This grid-based mapping forms the foundation of the Digital Twin model and allows efficient visualization of parking availability

Real-Time Slot Occupancy Synchronization:

The system must ensure real-time synchronization between the physical parking infrastructure and its digital twin replica.
Any occupancy update made by the administrator or triggered through simulation must be stored instantly in the cloud database.
The updated slot status must reflect in the user interface within milliseconds.
This ensures that drivers receive accurate live information and avoids outdated parking availability data

Cloud Database Integration for Continuous Updates:

- The system must integrate with a cloud-based real-time database such as Firebase or MongoDB.
- Slot availability data must be stored in structured collections.
- The database must support real-time listeners to update the frontend grid instantly.
- This requirement enables continuous data flow between the backend system and the digital twin visualization.

Digital Twin Visualization Interface:

- The system must provide an interactive digital twin interface that visually displays parking slot status.

- Vacant slots should be shown as available.
- Occupied slots should be shown as unavailable.
- Users should be able to easily interpret the parking grid and locate free slots quickly.
- This reduces unnecessary vehicle movement and improves parking efficiency.

Dynamic Pricing Engine Implementation:

- The backend system must include a dynamic pricing engine that calculates parking fees based on demand prediction models.
- Parking charges should vary depending on:
- Time of day
- Slot location (premium vs normal slots)
- Peak demand situations
- This requirement supports congestion control and increases revenue generation during rush hours.

Simulation of Special Scenarios:

- The system must support a simulation interface that allows administrators to model special real-world conditions such as:
- Rush hour surges
- Festival overcrowding
- Emergency override situations
- During simulation, multiple slot statuses must be updated simultaneously, allowing predictive planning and operational preparedness.

Administrator Monitoring and Control Panel:

- The system must provide a dedicated web dashboard for administrators.
- Admin functionalities include:
- Updating occupancy status manually
- Monitoring live occupancy heatmaps
- Triggering demand-based pricing changes
- Activating scenario simulation modes (e.g., Festival Mode)

This ensures centralized management of the smart parking infrastructure.

Pricing and Surge Logs Maintenance:

The system must maintain pricing history through a pricing log mechanism.

- Timestamp-based records of price changes must be stored.
- Surge multipliers applied during high demand must be recorded for analytics.

This helps administrators evaluate pricing strategies and optimize parking operations.

Non-Functional Requirements (NFR):

Low Latency and Real-Time Accuracy:

- The synchronization delay between the cloud database and the digital twin interface must remain below 1 second.
- Low latency is essential to ensure real-time correctness and avoid driver confusion caused by outdated slot availability

Scalability and Load Handling:

- The system architecture must be scalable to support:
- Large parking areas with hundreds or thousands of slots
- Increasing numbers of simultaneous users
- The system should not experience performance degradation as usage grows.

User Interface Responsiveness:

- The application must be fully responsive across different devices, including:
- Mobile phones
- Tablets
- Desktop dashboards
- The digital grid interface must adjust smoothly to varying screen sizes to provide optimal interaction for both drivers and administrators**.**

Reliability and High Availability:

- The system must ensure at least 99.9% availability of real-time parking data.
- Reliable access is necessary to prevent traffic buildup, user frustration, and operational failures during high-demand situations.

Data Consistency and Integrity:

The system must guarantee consistency between:
- Physical parking slot status
- Cloud database records
- Digital twin visualization model

Any mismatch may lead to incorrect booking decisions and system inefficiency.

Security and Access Control:

- Only authorized administrators must be allowed to modify slot occupancy and pricing configurations.
- The system must implement secure authentication mechanisms such as Firebase Authentication to prevent unauthorized access.

3. Database Design

Introduction

- Administrator dashboard
- Database design is one of the most important components of the **Digital Twin–Based Smart Parking System**.
  Since the system continuously monitors parking slot occupancy in real time, the database must support:
- Digital twin grid visualization
- Dynamic pricing engine
- Efficient storage of pricing and simulation data
- Fast updates
- Real-time synchronization
- Scalability for large parking lots

The database acts as the central storage unit that connects:

The system uses a **NoSQL cloud database (Firebase/MongoDB)** because it provides real-time listeners and instant synchronization between the physical parking environment and the digital twin model.

- User application (drivers)

Need for Database in Smart Parking System:

In traditional parking systems, slot availability is not updated instantly, causing delays and confusion.
In this project, the database is required to:

- Store parking slot status (vacant/occupied)
- Maintain grid coordinates for each slot
- Support booking and reservation details
- Store dynamic pricing changes
- Provide simulation updates during festivals or emergencies

Thus, the database ensures that all system components remain synchronized.

Database Architecture Overview:

The database design is structured to support the Digital Twin concept, where the physical parking area is replicated virtually as a grid.

Key characteristics of the database:

- **Real-time data storage**
- **Fast slot occupancy updates**
- **Cloud synchronization**
- **Support for surge pricing logs**
- **Efficient slot coordinate mapping**

The system follows a collection-based schema design suitable for Firebase.

Entity Relationship and Schema Structure:

Although Firebase is a NoSQL database, the system entities can still be modeled conceptually using ER

design.
The main entities in the smart parking database include:

1. Parking Slots
2. Users
3. Bookings
4. Admin
5. Pricing Logs

Database Collections and Attributes:

1. **Collection: parking_slots:**

- This is the most important collection in the database.
  It stores information about every parking slot in the parking lot grid.
- Each slot is uniquely identified and mapped to a coordinate system.

Field Name	Type	Description
slot_id	String	Unique identifier of slot (ex: A1, B3)
coordinates	Object {x,y}	Grid position of slot
status	Integer	0 = Vacant, 1 = Occupied
premium_status	Boolean	Indicates premium slot near entrance/exit
base_price	Float	Standard parking fee

Example:

```
{
 "slot_id": "B4",
 "coordinates": { "x": 2, "y": 4 },
 "status": 0,
 "premium_status": true,
 "base_price": 50
}
```

This structure allows the digital twin interface to display the parking lot as a matrix grid.

2. Collection: users
This collection stores driver/user details for login and booking purposes.

Field Name	Type	Description
user_id	String	Unique user identifier
name	String	User name
email	String	Login email
phone	Number	Contact number
password	String	Encrypted password

Users interact with the system to check availability and reserve slots.

3. Collection: bookings
This collection stores booking and reservation details made by drivers.
Each booking links a user with a parking slot.

Field Name	Type	Description
booking_id	String	Unique booking identifier
user_id	Foreign Key	References users collection
slot_id	Foreign Key	References parking_slots
start_time	DateTime	Booking start time
end_time	DateTime	Booking end time
payment_status	Boolean	Paid / Unpaid

Example:

```
{
 "booking_id": "BK101",
 "user_id": "U22",
 "slot_id": "A3",
 "start_time": "10:00 AM",
 "end_time": "12:00 PM",
 "payment_status": true
}
```

This ensures organized reservation tracking.

4. Collection: admin
The admin collection provides secure authentication for administrators.
Admins manage occupancy updates, simulation controls, and pricing strategies.

Field Name	Type	Description
admin_id	String	Unique admin identifier
email	String	Admin login
password	String	Secured password

Only admins are authorized to update system data.

5. Collection: pricing_logs
This collection stores records of dynamic pricing changes applied during high-demand conditions.

Field Name	Type	Description
log_id	String	Unique log identifier
timestamp	DateTime	Time of pricing update
surge_multiplier	Float	Demand-based multiplier

Example:

```
{
  "log_id": "PL09",
  "timestamp": "2026-01-25 6:00 PM",
  "surge_multiplier":  1.5
}
```

Pricing logs help in analytics and revenue tracking

Relationships Between Entities

Even though NoSQL databases are non-relational, conceptual relationships exist:

- One user can make multiple bookings
- One parking slot can be booked many times, but only once at a given time
- Admin controls slot updates and simulation modes
- Pricing logs track dynamic pricing adjustments

Benefits of Firebase Database for This Project

Firebase is chosen because it provides:

- Real-time slot synchronization
- Cloud scalability
- Instant updates to digital twin UI
- Support for event-driven simulation updates
- Secure authentication integration

Thus, Firebase ensures the system remains responsive and reliable.

Conclusion

The database design forms the backbone of the **Digital Twin–Based Smart Parking System**. By storing parking slot coordinates, occupancy status, booking details, and pricing logs in a structured cloud database, the system achieves:

- Real-time availability updates
- Efficient parking management
- Dynamic pricing support
- Scenario-based simulation handling

This robust database architecture ensures seamless interaction between the physical parking infrastructure and its virtual digital twin environment.

4. SRS Document Preparation

The purpose of this Software Requirements Specification (SRS) document is to define the functional and non-functional requirements of the **Digital Twin–Based Smart Parking System**. This document provides a complete description of the system's features, performance expectations, external interfaces, and operational environment.

The SRS serves as a blueprint for developers, project guides, and stakeholders to understand the design and implementation requirements before development begins.

1. Scope of the Project

Urban areas face increasing parking challenges due to rapid growth in vehicle usage and limited parking infrastructure. Drivers spend significant time searching for parking spaces, resulting in:

- Traffic congestion
- Fuel wastage
- Increased pollution
- Driver frustration

The proposed **Digital Twin Smart Parking System** addresses these issues by creating a real-time virtual replica of the parking lot. The system provides:

- Live parking slot availability
- Digital grid-based visualization
- Simulation of special scenarios (rush hours, festivals)
- Dynamic surge-based pricing

This improves parking efficiency, reduces unnecessary movement, and enhances urban traffic management.

Definitions, Acronyms, and Abbreviations

Term	Meaning
SRS	Software Requirements Specification
Digital Twin	Virtual replica of a physical system
Slot Status	Indicates whether a slot is vacant or occupied
Firebase	Cloud-based real-time database
Surge Pricing	Dynamic pricing based on demand

Overview oof the docment:

This document includes:

- System description
- Functional requirements

Non-functional requirements

- Interface specifications
- Operational scenarios
- Future enhancements

2. Overall Description

2.1 Product Perspective

The Digital Twin–Based Smart Parking System is designed as a smart city solution that integrates:

- Physical parking infrastructure
- Cloud database storage
- Digital twin simulation environment
- Web/mobile application interface

The parking lot is modeled as a **2D matrix grid**, where each slot is represented digitally with binary occupancy values:

- 0 → Vacant
- 1 → Occupied

2.2 Product Functions

The major functions of the system include:

- Real-time slot monitoring
- Parking slot booking
- Digital twin visualization
- Administrator management dashboard
- Simulation of emergency/festival scenarios
- Dynamic pricing implementation

2.3 User Classes and Characteristics

User Type	**Description**
Driver/User	Searches for vacant parking slots and checks pricing
Administrator	Updates slot occupancy, controls simulations, manages pricing

2.4 Operating Environment

The system will operate using:

- Frontend: React / HTML, CSS, JavaScript
- Backend: Node.js with Express
- Database: Firebase / MongoDB

- Hosting: Firebase Hosting / AWS / Google Cloud

Accessible through:

- Mobile devices
- Tablets
- Desktop dashboards

2.5 Design and Implementation Constraints

- Requires stable internet connectivity
- Cloud synchronization must occur within milliseconds
- Database must support real-time listeners
- Secure admin authentication is mandatory

5. Use-Case Diagram Design

The system interaction is modeled to ensure seamless data flow between actors and the digital twin core:

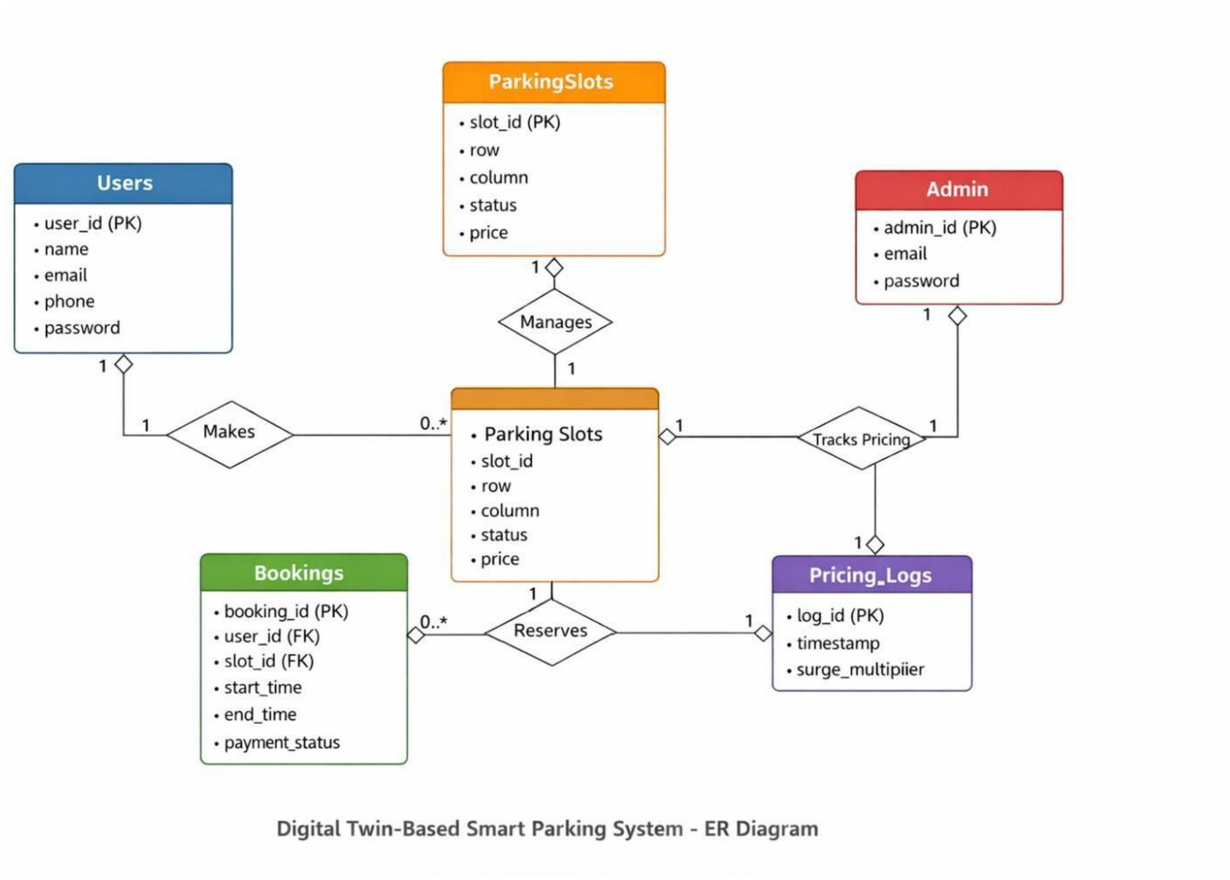

Figure 1: Interaction model between the User, Admin, and the Cloud Database Architecture.

Use-Case Diagram Design

(For Digital Twin–Based Smart Parking System)

This is written in a professional way, ready to paste into your report.

A Use-Case Diagram is an important part of system modeling in software engineering.
It visually represents the interactions between users (actors) and the system.

In the **Digital Twin–Based Smart Parking System**, the use-case diagram helps to understand:

- How drivers access parking information
- How administrators manage parking slots
- How the digital twin interface synchronizes real-time data
- How dynamic pricing and simulation features operate

Thus, the use-case model provides a high-level functional view of the system.

Purpose of Use-Case Diagram

The main objectives of designing a use-case diagram are:

- To identify system actors and their roles
- To define key system functionalities
- To show interactions between users and the digital twin system
- To provide a blueprint for implementation and testing

Use-case diagrams ensure that all required features are clearly captured before development begins.

Actors in the System

An actor is an external entity that interacts with the system.

In this project, there are two major actors:

1. Driver (User)

The driver is the primary end-user of the smart parking application.

Driver responsibilities include:

- Viewing available parking slots
- Checking pricing information
- Reserving or booking a slot
- Navigating to the selected slot

The driver benefits by saving time and avoiding unnecessary traffic movement.

2. Administrator (Admin)

The administrator is responsible for managing the parking infrastructure through the digital twin dashboard.

Admin responsibilities include:

- Updating slot occupancy status
- Monitoring real-time parking usage
- Controlling dynamic pricing
- Running simulations for special scenarios

Admins ensure smooth parking operations during rush hours or emergencies.

Major Use-Cases in the Smart Parking System

The use-cases represent the main services provided by the system.

Driver Use-Cases

UC1: View Parking Slot Availability

The driver can access the application to view real-time slot availability.

- Vacant slots are shown as available
- Occupied slots are shown as unavailable

This helps users quickly locate free parking spaces.

UC2: Check Parking Pricing

Before booking, the driver can check the current parking price.
Pricing may vary due to:

- Demand level
- Time of day
- Slot location

This supports the system’s dynamic pricing model.

UC3: Reserve or Book a Parking Slot

The driver can reserve a vacant parking slot in advance.
Booking includes:

- Slot selection
- Time duration
- Confirmation message

This prevents overcrowding and ensures convenience.

UC4: Navigate to Slot Location

After booking, the system provides navigation support to guide the driver to the selected slot.
This reduces unnecessary searching inside the parking area.

Administrator Use-Cases

UC5: Update Slot Occupancy Status

The admin can manually or automatically update slot status in the system:

- Mark slot as occupied when a car enters
- Mark slot as vacant when a car leaves

This ensures accurate digital twin synchronization.

UC6: Trigger Simulation Modes

The admin can activate simulation scenarios such as:

- Festival Mode
- Rush Hour Mode
- Emergency Override

During simulation, multiple slots may be updated simultaneously.
This helps in planning and managing high-demand situations.

UC7: Manage Dynamic Pricing

The admin can control the surge multiplier and pricing strategy.

Dynamic pricing ensures:

- Congestion control
- Efficient slot usage
- Increased revenue during peak demand

UC8: Review Analytics and Logs

The admin can review reports such as:

- Occupancy heatmaps
- Pricing history logs
- Peak demand statistics

This helps in future decision-making and optimization.

Use-Case Diagram Interaction Flow

The system interaction can be summarized as:

Driver Flow:

View Availability → Check Pricing → Book Slot → Navigate to Slot

Admin Flow:

Update Occupancy → Trigger Simulation → Manage Pricing → Review Analytics

This ensures smooth data flow between the physical parking infrastructure and the digital twin model.

Importance of Use-Case Diagram in This Project

The use-case diagram is essential because it:

- Defines all user and admin interactions clearly
- Helps developers implement modules systematically
- Ensures real-time synchronization requirements are met
- Supports smart city-level scalability and usability

It provides a complete functional overview of the Digital Twin Smart Parking System.

6. UI Wireframes (Figma / Sketches)

UI Wireframes are an essential part of the system design phase.

A wireframe is a **visual blueprint** or **basic layout structure** of the application interface before actual development begins.

In the **Digital Twin–Based Smart Parking System**, UI wireframes are designed to clearly represent:

- Parking slot availability
- Digital twin grid visualization
- Dynamic pricing controls
- Booking and navigation options
- Administrator management dashboard

Wireframes help in understanding how users will interact with the system and ensure a user-friendly experience.

6.2 Purpose of UI Wireframes

The main objectives of designing wireframes are:

- To provide a clear interface structure for developers
- To visualize system components before implementation
- To ensure easy navigation and usability for drivers
- To design an efficient admin control dashboard
- To reduce confusion and redesign during development

Thus, wireframes act as the foundation for the final UI design.

6.3 Tools Used for Wireframe Design

Wireframes for this project can be created using:

- **Figma** (Professional UI/UX tool)
- Sketches (Hand-drawn layouts for initial planning)
- Draw.io or Canva (Simple design alternatives)

Figma is preferred because it supports:

- Interactive prototyping
- Responsive layouts
- Collaboration and component reuse

6.4 UI Wireframe Design Concepts

The project includes two main interfaces:

1. **Mobile Application (Driver View)**
2. **Web Dashboard (Admin View)**

6.5 Mobile Application Wireframe (User View)

The mobile application is the primary interface for drivers searching for parking spaces.

Key Screens in Driver Application

1. Login / Registration Screen

Purpose:
Allows users to securely access the system.

Components:

- Email/Phone input
- Password field
- Login button
- New user registration option

2. Parking Slot Availability Grid Screen

This is the most important screen in the driver app.

Purpose:

- Displays the digital twin grid of the parking lot
- Helps drivers locate vacant slots quickly

Features:

- 10×10 grid representation of parking slots
- Color indication:
    - Vacant Slots → Green
    - Occupied Slots → Red
- Slot ID labels (A1, B2, etc.)

This reduces unnecessary vehicle movement and saves time.

3. Slot Details and Pricing Screen

When the driver taps on a vacant slot:

The system displays:

- Slot number
- Current parking fee
- Surge-adjusted price
- Slot location type (premium/normal)

This supports the dynamic pricing engine.

4. Booking Confirmation Screen

Purpose:

- Allows drivers to reserve a slot

Includes:

- Selected slot details
- Booking duration
- Confirm booking button
- Payment status

5. Navigation Assistance Screen

After booking, the system provides:

- Direction guidance
- Slot coordinates
- Entry/exit proximity

This improves driver convenience.

6.6 Web Dashboard Wireframe (Admin View)

The admin dashboard is designed for parking authorities to manage the parking infrastructure.
It provides high-level control over the digital twin environment.

Key Screens in Admin Dashboard

1. Live Parking Monitor Screen

Purpose:

- Displays real-time occupancy of the entire parking lot

Features:

- Digital twin grid interface
- Heatmap visualization of occupied zones
- Instant slot updates

Admins can monitor congestion levels easily.

2. Slot Management Panel

Purpose:

- Allows admin to update slot occupancy status

Functions:

- Mark slot as vacant or occupied
- Bulk update during emergencies
- Coordinate-based slot mapping

This ensures accurate synchronization with the cloud database.

3. Dynamic Pricing Control Section

Purpose:

- Controls surge pricing multiplier

Features:

- Surge pricing toggle (ON/OFF)
- Multiplier slider (1.0x – 2.0x)
- Premium slot pricing adjustment

This helps manage demand and revenue generation.

4. Simulation Control Interface

Purpose:

- Allows simulation of special scenarios such as:
    - Festival Mode
    - Rush Hour Mode
    - Emergency Override

Features:

- Scenario selection library
- Bulk slot status updates
- Real-time impact visualization

Simulation helps in predictive planning.

5. Logs and Analytics Screen

Purpose:

- Provides historical system insights

Displays:

- Pricing change logs
- Occupancy reports
- Peak demand statistics
- Booking history

This supports decision-making and optimization.

6.7 Importance of UI Wireframes in This Project

UI wireframes play a major role in ensuring:

- Clear digital twin visualization
- Smooth interaction for drivers
- Efficient admin control operations
- Responsive design across devices
- Better user experience and usability

Conclusion

The completion of Phase 2 (Requirements & Design) establishes a robust foundation for the Digital Twin-Based Smart Parking System. By defining the matrix-based grid mapping and the dynamic pricing logic, the project ensures both technical feasibility and industry-level usability. The next phase will focus on the module-wise implementation of the core features, starting with the real-time database integration.