### **Sharib Ahmad**

ID: 24f2001786 | Email:24f2001786@ds.study.iitm.ac.in Full-stack developer passionate about scalable backend systems

# 1. Project Overview

## 1.1 Description

Web-based application for efficient parking facility management serving regular users (vehicle registration, parking information) and administrators (comprehensive control over lots, spots, analytics).

Target Users: Vehicle owners, parking facility managers, technical maintenance staff.

### **Key Objectives:**

- Secure user authentication with RBAC
- Vehicle registration and management
- Administrative facility control
- Real-time parking availability
- Revenue insights and analytics
- Responsive cross-device design

## 2. Technical Architecture

### Backend:

Python Flask, Flask-Login, Flask-WTF, Flask-RESTful, Jinja2 templating, Bootstrap 5, Font Awesome, SQLAlchemy ORM, SQLite, Blueprint routing

### Frontend & Documentation:

### 2.1 Architecture & Database

#### **Architecture Pattern:**

Controller: HTTP requests via Blueprint modules Model: Data structures and database operations View: UI rendering and presentation logic

API: Programmatic system access

#### Database Schema:

User: Credentials, profiles, role assignments Vehicle: User-linked details with metadata ParkingLot: Facility info, pricing, hours ParkingSpot: Individual spaces with status

View ER Diagram →

## 3. System Features

### **Authentication & Security**

- Secure registration with validation
- Password hashing algorithms
- Session-based authentication
- Role-based access control
- **CSRF** protection

### **Administrative Features**

- User account management
- Parking lot configuration
- Spot status monitoring
- Analytics dashboard
- System monitoring

### **User Features**

- Vehicle registration
- Profile management
- Parking history
- Search functionality
- Real-time updates

# 4. Implementation & Challenges

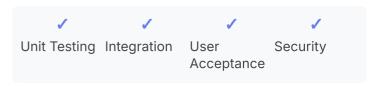
## 4.1 Development Methodology

Iterative development with continuous integration: modular component development, Git-based workflow with feature branches, unit testing, comprehensive documentation.

### 4.2 Technical Challenges Resolved

- Session Management: Flask-Login with secure session cookies and timeout handling
- Real-time Synchronization: Database-driven state management with optimistic locking
- Scalability: Efficient database queries and caching strategies

## 4.3 Testing Strategy



# 5. Results & Future Development

## **5.1 Project Outcomes**

- Fully functional dual-role web application
- Swagger-integrated API documentation
- · Responsive multi-device interface
- Robust data management with integrity
- Real-time parking availability tracking

## 5.2 Quality Metrics

- · Comprehensive test coverage
- · Optimized response times
- Robust error handling
- · Clean, documented code

#### 5.3 Lessons Learned

- · Early database design prevents changes
- · Modular organization enhances maintainability
- UX design drives system adoption
- Documentation ensures sustainability

### 5.4 Future Enhancements

- Native mobile applications
- Online payment processing
- IoT sensor integration
- Machine learning analytics
- Multi-tenant architecture

## 6. Conclusion

The Vehicle Parking Management System successfully demonstrates modern web development practices addressing real-world parking challenges. The project showcases proficiency in full-stack development, database design, and user experience considerations. Its modular architecture ensures maintainability and extensibility while addressing both user and administrative requirements, providing a solid foundation for smart city solutions.

## 7. Al Usage Declaration

Al Assistance: 15% of total development

- Swagger Documentation (8%): Generated swagger.yaml file
- Error Handling Documentation (4%): Error scenarios and responses
- Code Optimization (3%): Minor database query optimization

**Independent Development:** System architecture, database design, core business logic, UI design, testing strategy, security implementation.

### 8. Resources

Video: Project Demo

API Endpoints:

• /api/users - List all users

/api/user/<string:user\_id> - Get a specific user

• /api/vehicles — List all vehicles

•

/api/vehicle/<string:vehicle number>

- Get a specific vehicle

 /api/parking-lots - List all parking lots

/api/parking-lot/<int:lot\_id> - Get
a specific parking lot

• /api/parking-spots — List all parking spots

- [/api/parking-spot/<int:spot\_id>] -

Get a specific parking spot

MinRecBrowserPython 3.8+Python 3.9+Chrome 90+512MB RAM2GB RAMFirefox 88+1GB Storage5GB StorageSafari 14+