**Software Requirements Specification (SRS)**

# Version History

| Version | Date | Author | Reviewed By | Approved By | Change Description |
| --- | --- | --- | --- | --- | --- |
| 1.0 | 08-07-2025 | [Devkrishna] | [Reviewer] | [Approver] | Initial Release |

# 1. Introduction

## 1.1 Purpose

This document specifies detailed requirements for developing Rentavec, a robust vehicle rental system using Django and REST APIs.

## 1.2 Scope

The document covers functionalities, database schemas, RESTful API specs, UI/UX interactions, security, performance metrics, and reliability for Rentavec v1.0.

## 1.3 Intended Audience

- Developers: Implementation, debugging, integration.  
- Software Architects: System design, architectural compliance.  
- Quality Assurance Engineers: Test planning, execution.  
- Maintenance Teams: Support, troubleshooting, monitoring.  
- Project Managers: Progress oversight, timelines, deliverables.

## 1.4 Definitions, Acronyms, and Abbreviations

API: Application Programming Interface  
JWT: JSON Web Token  
REST: Representational State Transfer  
DRF: Django Rest Framework  
CRUD: Create, Read, Update, Delete

## 1.5 References

- Django Framework Documentation  
- Django REST Framework Documentation  
- JWT Standards (RFC 7519)

## 1.6 Overview

Detailed functional/non-functional requirements, complete API definitions, database schemas, UI/UX details, security protocols, UML diagrams, performance and scalability guidelines.

# 2. Overall Description

## 2.1 Product Perspective

Rentavec is a standalone platform with RESTful APIs, JWT authentication, cloud asset management, designed for future scalability and third-party integrations.

## 2.2 Product Functions

- Secure Authentication  
- Comprehensive Vehicle Management  
- Robust Rental Reservation System  
- Rental Operations  
- User Reviews  
- Advanced Admin Functionalities

## 2.3 User Classes and Characteristics

- Visitor  
- Authenticated User  
- Administrator

## 2.4 Operating Environment

- Cloud-hosted  
- Compatible with major browsers

## 2.5 Design and Implementation Constraints

- Django Framework  
- PostgreSQL Database

## 2.6 User Documentation

Detailed online user manuals accessible to all users.

## 2.7 Assumptions and Dependencies

- Reliable cloud infrastructure  
- Django and DRF maintained regularly  
- PostgreSQL available and reliable  
- No external payment integration initially

# 3. Specific Requirements

## 3.1 Functional Requirements

FR-1: User Authentication  
- FR-1.1: Users must register using email and phone number.  
- FR-1.2: Passwords must comply with complexity standards.  
- FR-1.3: JWT tokens issued upon successful login.  
  
FR-2: Vehicle Browsing  
- FR-2.1: Vehicles can be filtered by type, fuel, condition, and price.  
- FR-2.2: Real-time availability status.  
  
FR-3: Rental Management  
- FR-3.1: Reserve vehicles specifying dates and locations.  
- FR-3.2: Confirmation tickets provided post-rental.  
- FR-3.3: Vehicle returns with specific reasons.  
- FR-3.4: Rentals can be renewed subject to availability.  
  
FR-4: Review Management  
- FR-4.1: Users can leave reviews after rentals.  
- FR-4.2: Reviews displayed publicly with ratings.

## 3.2 External Interface Requirements

3.2.1 User Interface  
- Detailed Figma wireframes linked explicitly.  
  
3.2.2 API Interface  
- Endpoint definitions detailed in Appendix B.  
  
3.2.3 Software Interface  
- Django Framework  
- Django REST Framework  
- PostgreSQL Database  
- JSON Web Tokens (JWT)  
- Excel Import/Export capability.

## 3.3 Performance Requirements

- ≤ 2-second response under 500 concurrent users.  
- Vehicle search results load ≤ 1 second.  
- Supports 1000 requests per minute without significant latency.

## 3.4 Logical Database Requirements

- Detailed ERD(Entity Relationship Diagram) in Appendix C.  
- Defined primary and foreign keys with cascade rules.  
- Indexed fields: username, email, vehicle name, rental dates.  
- Rentals restricted to one active rental per vehicle per user.

## 3.5 Design Constraints

Strict adherence to Django and Django REST Framework.

## 3.6 Software System Attributes

- Maintainability: Modular, documented codebase.  
- Portability: Cloud-agnostic deployment.  
- Reliability: System uptime ≥ 99.9%.

# 4. Detailed Data Models

- Explicit tables, fields, validations, constraints, relationships.  
- Refer to detailed ERD(Entity Relationship Diagram) (Appendix C).

# 5. API Endpoint Specifications

- URLs, HTTP methods, payload and response structures.  
- Error codes and messages explicitly documented in Appendix B.

# 6. UI/UX Specifications

- Explicit wireframes, navigation flows, interaction mappings.  
- Refer to Appendix E.

# 7. Security and Privacy Specifications

- JWT authentication.  
- Data encryption standards.  
- GDPR(General Data Protection Regulation) compliance guidelines.

# 8. Reliability and Availability Specifications

- System uptime ≥ 99.9%.  
- Backup and disaster recovery protocols.

This document is structured into distinct sections detailing the purpose, overall system scope, specific functional and non-functional requirements, data models, API specifications, UI/UX descriptions, and security considerations. It also includes appendices for diagrams, endpoint details, and test cases.

# 9. Test Cases and Acceptance Criteria

Below are selected high-priority test cases mapped to functional requirements.

**Test Case Table Example: Rental Reservation**

| Test ID | Scenario | Steps | Expected Result | Priority |
| --- | --- | --- | --- | --- |
| TC-001 | Reserve available vehicle | Login > Select vehicle > Set dates > Confirm | Rental confirmation ticket generated | High |
| TC-002 | Reserve unavailable vehicle | Select dates with no stock | System shows availability error | High |

# 10. Change Management Policy

All document changes are tracked in the version control table. Changes are proposed by the author, reviewed by a peer or lead engineer, and approved by a project manager. A new version is released only after QA verification.

# 11. Appendices

Appendix A: Glossary

Appendix B: API Endpoint Reference

Appendix C: ER Diagrams and Data Models

Appendix D: UML Diagrams (Sequence, Use Case)

Appendix E: UI/UX Wireframes

Appendix F: Future Enhancements

*Fig 1.1.1: System Context Diagram showing boundaries and interactions between users, API, and external services*

*Fig 3.1.1: Wireframe - Landing Page layout for vehicle listing*

*Fig 3.1.2: Wireframe - Vehicle Detail Page with reserve option*

*Fig 4.1.1: ER Diagram showing User, Rental, Vehicle, Review relationships*

*Fig 5.1.1: API Flow Diagram showing endpoint interaction sequence*

*Fig 6.1.1: UI flow diagram from login to dashboard*

## Appendix B: API Endpoint Reference

**1. User Authentication APIs**

| Endpoint | Method | Request Payload | Success Response | Error Response |
| --- | --- | --- | --- | --- |
| /api/register/ | POST | { 'username': '', 'email': '', 'password': '', 'phone': '' } | { 'id': 1, 'username': 'dev' } | { 'error': 'User already exists' } |
| /api/token/ | POST | { 'username': '', 'password': '' } | { 'access': '', 'refresh': '' } | { 'detail': 'No active account found' } |
| /api/token/refresh/ | POST | { 'refresh': '' } | { 'access': '' } | { 'detail': 'Token is invalid or expired' } |

**2. Vehicle APIs**

| Endpoint | Method | Request Payload | Success Response | Error Response |
| --- | --- | --- | --- | --- |
| /api/vehicles/ | GET | None | [{ 'id': 1, 'name': 'Car A', 'type': 'car' }] | None |
| /api/vehicles/{id}/ | GET | None | { 'id': 1, 'name': 'Car A', 'details': {...} } | { 'error': 'Vehicle not found' } |
| /api/vehicles/ | POST | { 'name': '', 'type': '', 'price': 1000, 'condition': '', ... } | { 'id': 2, 'name': 'Car B' } | { 'error': 'Validation failed' } |

**3. Rental APIs**

| Endpoint | Method | Request Payload | Success Response | Error Response |
| --- | --- | --- | --- | --- |
| /api/rentals/ | POST | { 'vehicle': 1, 'pickup\_date': '', 'dropoff\_date': '', ... } | { 'rental\_id': 101, 'status': 'active' } | { 'error': 'Vehicle unavailable' } |
| /api/rentals/{id}/return/ | POST | { 'return\_reason': '', 'dropoff\_location': 2 } | { 'message': 'Return initiated' } | { 'error': 'Rental not found' } |

**4. Return APIs**

| Endpoint | Method | Request Payload | Success Response | Error Response |
| --- | --- | --- | --- | --- |
| /api/rentals/{id}/return/ | POST | { 'return\_reason': 'damaged seat', 'dropoff\_location': 3 } | { 'message': 'Return initiated', 'return\_id': 44 } | { 'error': 'Invalid rental ID' } |

**5. Renew Rental APIs**

| Endpoint | Method | Request Payload | Success Response | Error Response |
| --- | --- | --- | --- | --- |
| /api/rentals/{id}/renew/ | POST | { 'renewal\_type': 'week', 'new\_dropoff\_location': 2 } | { 'status': 'renewed', 'new\_end\_date': '2025-08-01' } | { 'error': 'Vehicle not available for extension' } |

**6. Change Drop-off Location APIs**

| Endpoint | Method | Request Payload | Success Response | Error Response |
| --- | --- | --- | --- | --- |
| /api/rentals/{id}/change-dropoff/ | POST | { 'new\_dropoff\_location': 4 } | { 'message': 'Drop-off location updated' } | { 'error': 'New and old drop-off locations are the same' } |

**7. Review APIs**

| Endpoint | Method | Request Payload | Success Response | Error Response |
| --- | --- | --- | --- | --- |
| /api/reviews/ | POST | { 'rental': 101, 'title': 'Great ride', 'subject': 'Smooth pickup', 'stars': 5 } | { 'review\_id': 1, 'status': 'created' } | { 'error': 'User has not rented this vehicle' } |
| /api/reviews/ | GET | None | [{ 'title': 'Great ride', 'stars': 5 }] | None |

**8. User Profile APIs**

| Endpoint | Method | Request Payload | Success Response | Error Response |
| --- | --- | --- | --- | --- |
| /api/user/profile/ | GET | None | { 'username': 'dev', 'email': 'dev@example.com' } | { 'error': 'Unauthorized' } |
| /api/user/change-password/ | POST | { 'old\_password': '', 'new\_password': '' } | { 'message': 'Password updated' } | { 'error': 'Incorrect current password' } |
| /api/user/delete/ | DELETE | None | { 'message': 'Account deleted' } | { 'error': 'Unauthorized' } |

**9. Rental History APIs**

| Endpoint | Method | Request Payload | Success Response | Error Response |
| --- | --- | --- | --- | --- |
| /api/rentals/history/ | GET | None | [{ 'rental\_id': 2, 'status': 'returned' }] | { 'error': 'User unauthorized' } |

**10. Admin Order View APIs**

| Endpoint | Method | Request Payload | Success Response | Error Response |
| --- | --- | --- | --- | --- |
| /api/admin/orders/ | GET | ?status=returned&type=car | [{ 'ticket\_id': 101, 'status': 'returned' }] | None |

**11. Dashboard Summary APIs**

| Endpoint | Method | Request Payload | Success Response | Error Response |
| --- | --- | --- | --- | --- |
| /api/dashboard/summary/ | GET | None | { 'total\_income': 15000, 'top\_vehicles': [...] } | None |

**12. Vehicle Bulk Operations**

| Endpoint | Method | Request Payload | Success Response | Error Response |
| --- | --- | --- | --- | --- |
| /api/vehicles/bulk-delete/ | POST | { 'vehicle\_ids': [1,2,3] } | { 'deleted': 3 } | { 'error': 'Invalid IDs' } |
| /api/vehicles/import-excel/ | POST | Excel file upload (multipart/form-data) | { 'imported': 10, 'skipped': 2 } | { 'error': 'File format error' } |

**13. Visualization API**

| Endpoint | Method | Request Payload | Success Response | Error Response |
| --- | --- | --- | --- | --- |
| /api/dashboard/visualizations/ | GET | None | { 'chart\_data': { 'daily\_orders': [...], 'monthly\_income': [...] } } | None |

## Appendix C: Data Models

**1. User**

Stores registered user information including phone and credentials.

Fields: id, username, email, phone\_number, password, is\_staff, is\_superuser

**2. Vehicle**

Represents a rental vehicle in the system.

Fields: id, name, image, type (scooter/car/bike), year, condition, price\_per\_day/week/month, total\_stock, available\_stock, fuel\_type, pickup\_locations[], dropoff\_locations[], next\_available\_date

**3. Rental**

Stores rental transactions made by users.

Fields: id, user\_id (FK), vehicle\_id (FK), rental\_type, pickup\_location\_id, dropoff\_location\_id, pickup\_date, dropoff\_date, status, ticket\_id, created\_at

**4. ReturnRequest**

Captures return requests linked to active rentals.

Fields: id, rental\_id (FK), return\_reason, new\_dropoff\_location\_id, issue\_reported, created\_at

**5. Review**

User feedback after using the vehicle.

Fields: id, rental\_id (FK), user\_id (FK), title, subject, stars (1-5), created\_at

**6. Location**

Represents pickup and drop-off points associated with vehicles and rentals.

Fields: id, name, address, city, pincode, is\_active

**7. DropoffChangeRequest**

Captures a user's mid-rental request to change the original drop-off location.

Fields: id, rental\_id (FK), old\_location\_id, new\_location\_id, created\_at

**8. RenewalRequest**

Represents renewal actions on existing rentals, storing requested duration and status.

Fields: id, rental\_id (FK), requested\_duration, approved (bool), created\_at

**9. RentalTicket**

Represents a finalized rental summary shared with user on booking.

Fields: id, rental\_id (FK), issued\_at, ticket\_number, pickup\_date, dropoff\_date, dropoff\_location, vehicle\_snapshot (JSON)

**10. ExcelImportLog**

Tracks admin Excel uploads for bulk vehicle management.

Fields: id, file\_name, upload\_time, uploaded\_by, status, records\_added, records\_failed, error\_log

**11. DashboardStatSnapshot**

Materialized view or model to store periodic snapshot of system metrics.

Fields: id, date, total\_income, top\_vehicle\_ids (array), rental\_counts, pie\_chart\_data (JSON)

**12. RentalStatusLog**

Tracks every status change in a rental order.

Fields: id, rental\_id (FK), timestamp, from\_status, to\_status, triggered\_by (user/admin/system)

## Appendix D: UML Diagrams

Fig D.1.1: Use Case Diagram showing actors (Admin, User) and interactions (rental, return, review, etc.)

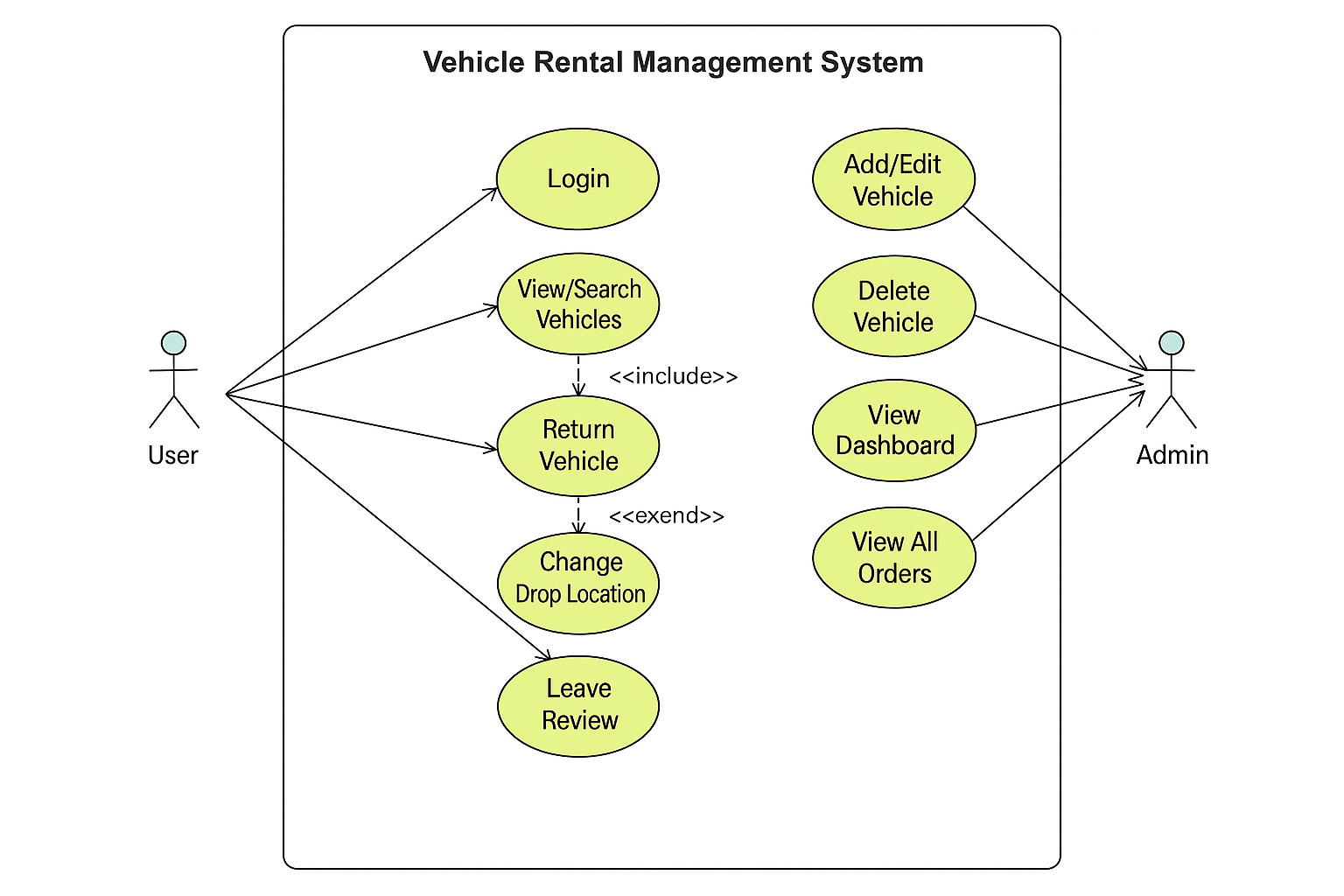


Fig D.1.2: Sequence Diagram for Vehicle Booking: user logs in, browses, selects, books a vehicle

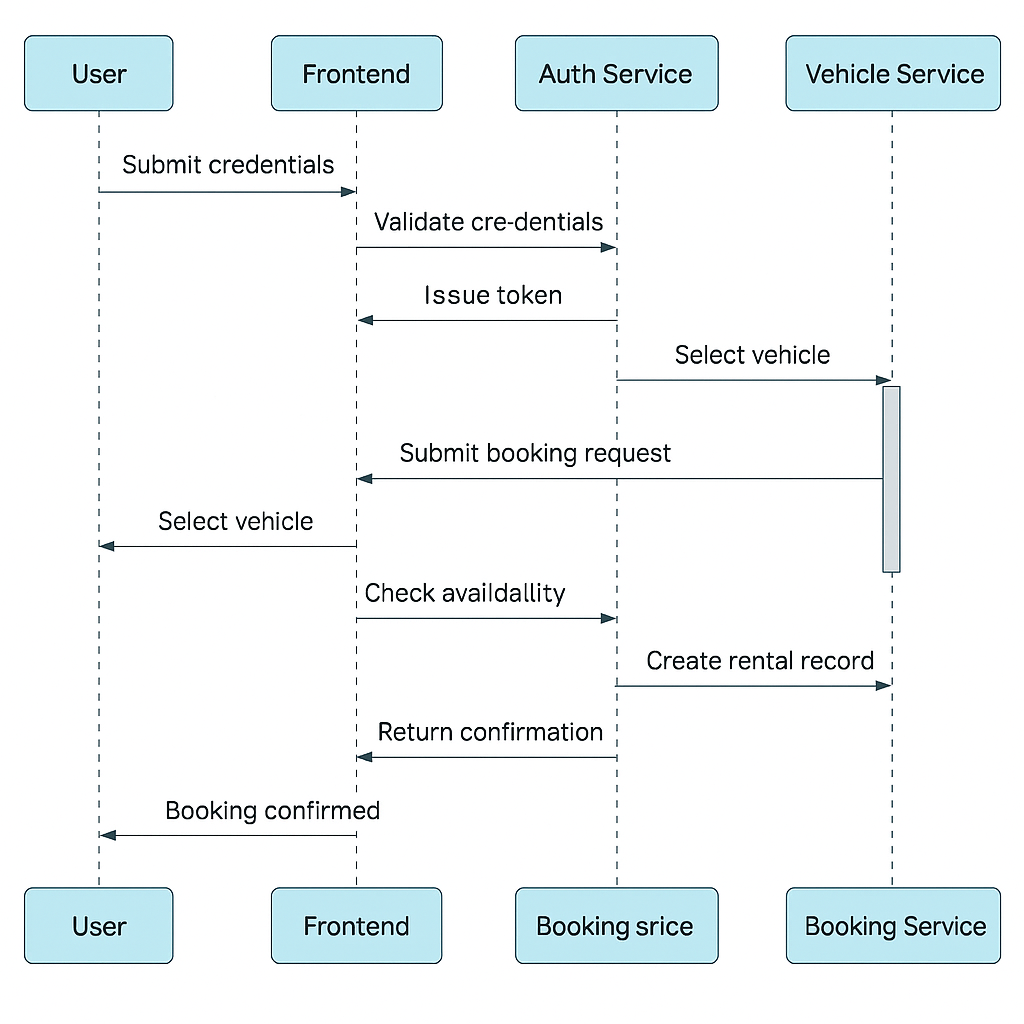


Fig D.1.3: Sequence Diagram for Vehicle Return: return request flow with drop-off selection

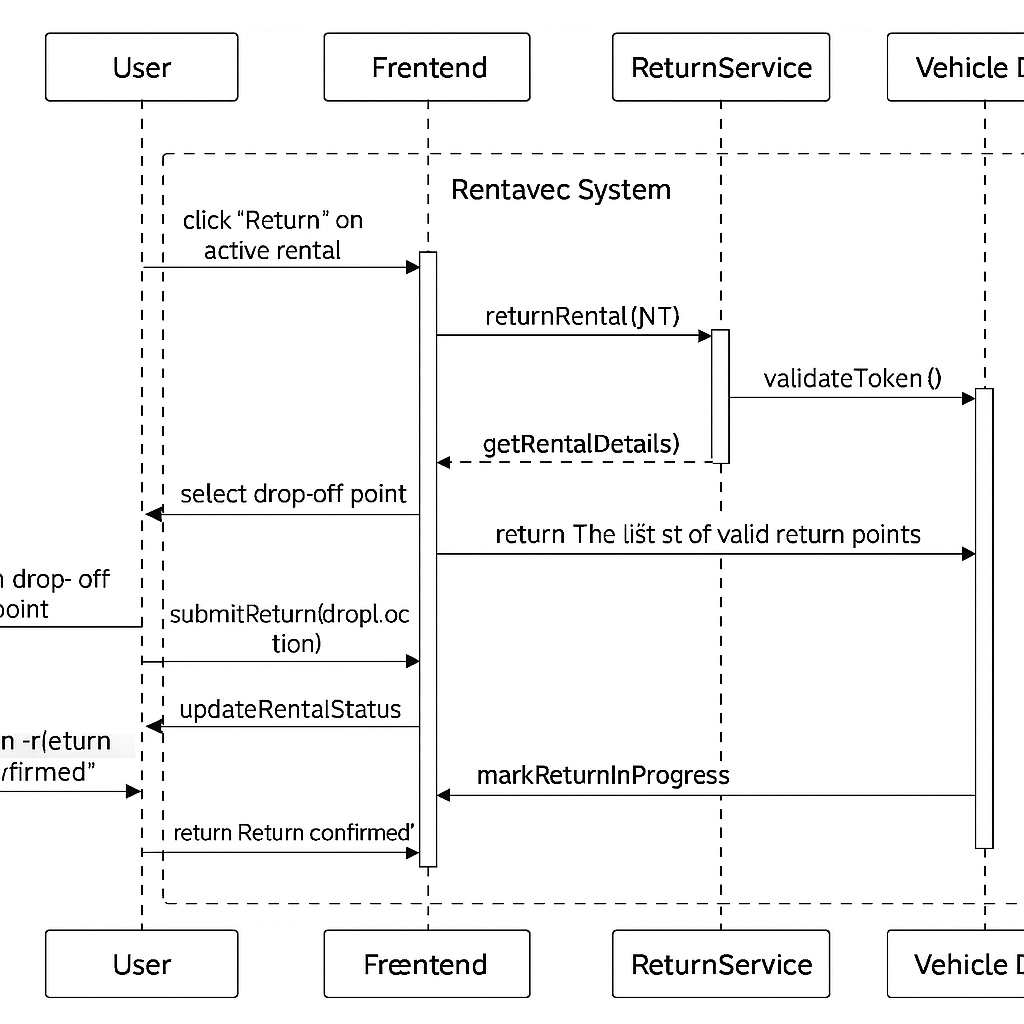


Fig D.1.4: Class Diagram showing entities: User, Vehicle, Rental, Review, etc., and their relationships

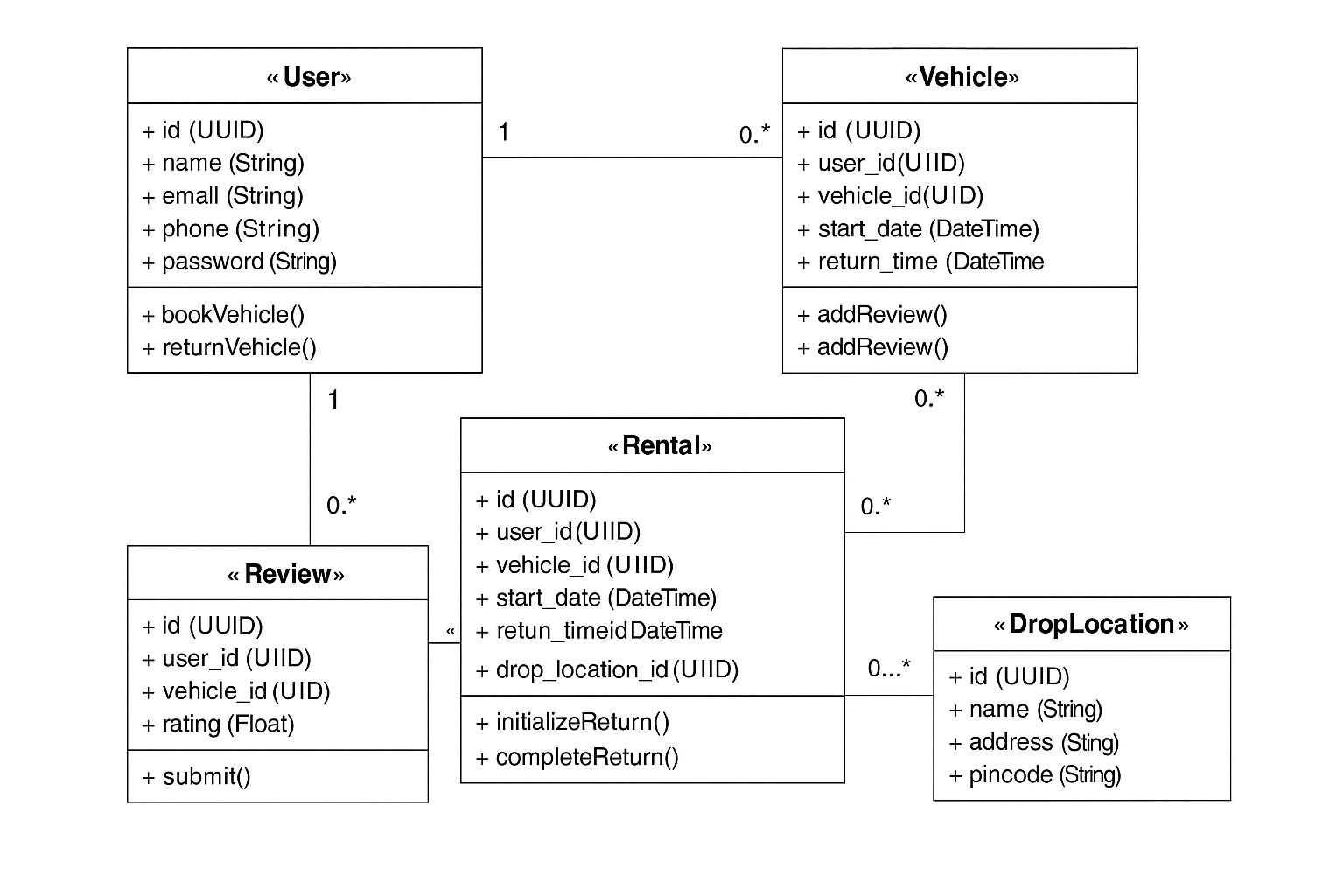


Fig D.1.5: Activity Diagram: admin uploading Excel to bulk-add vehicles

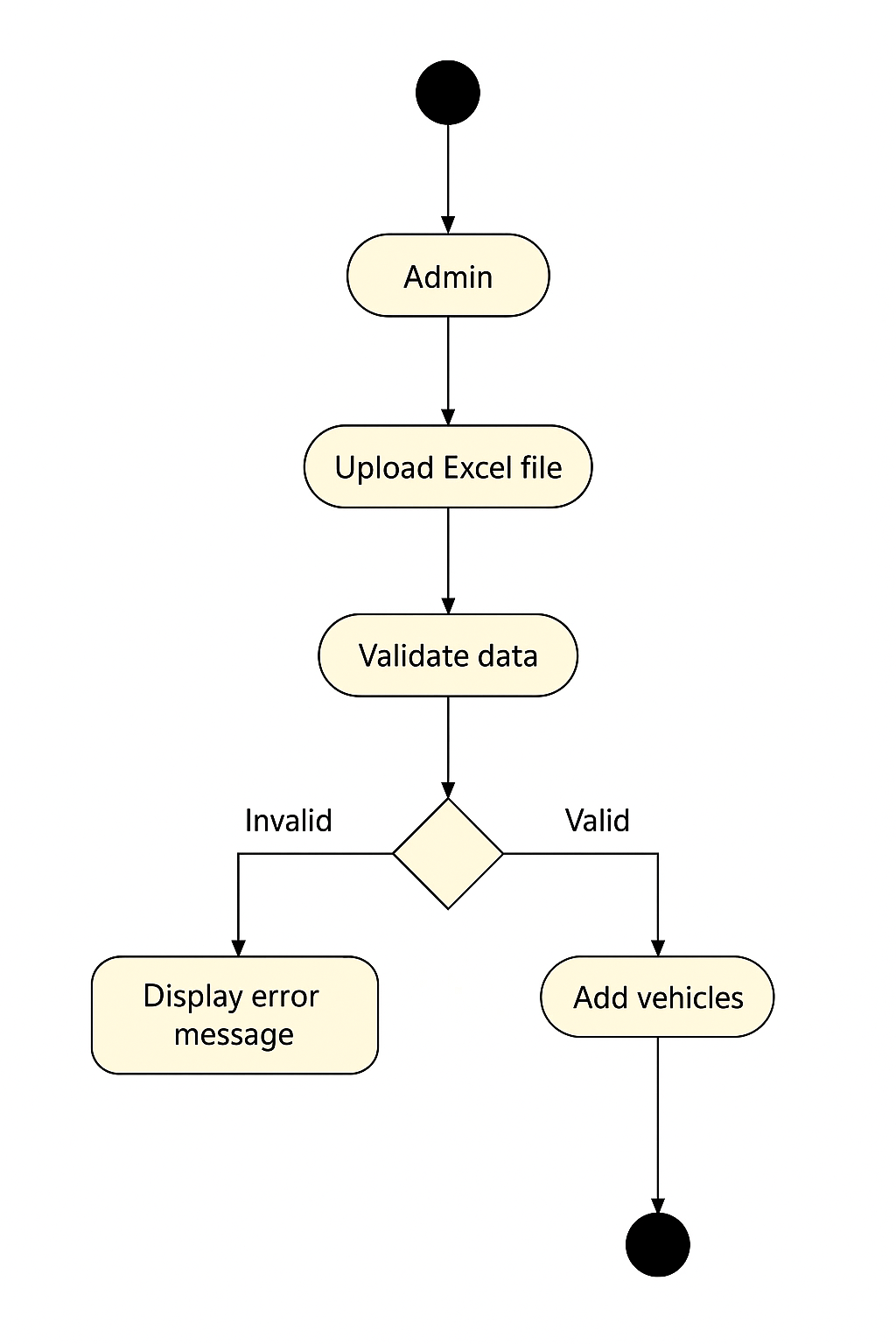
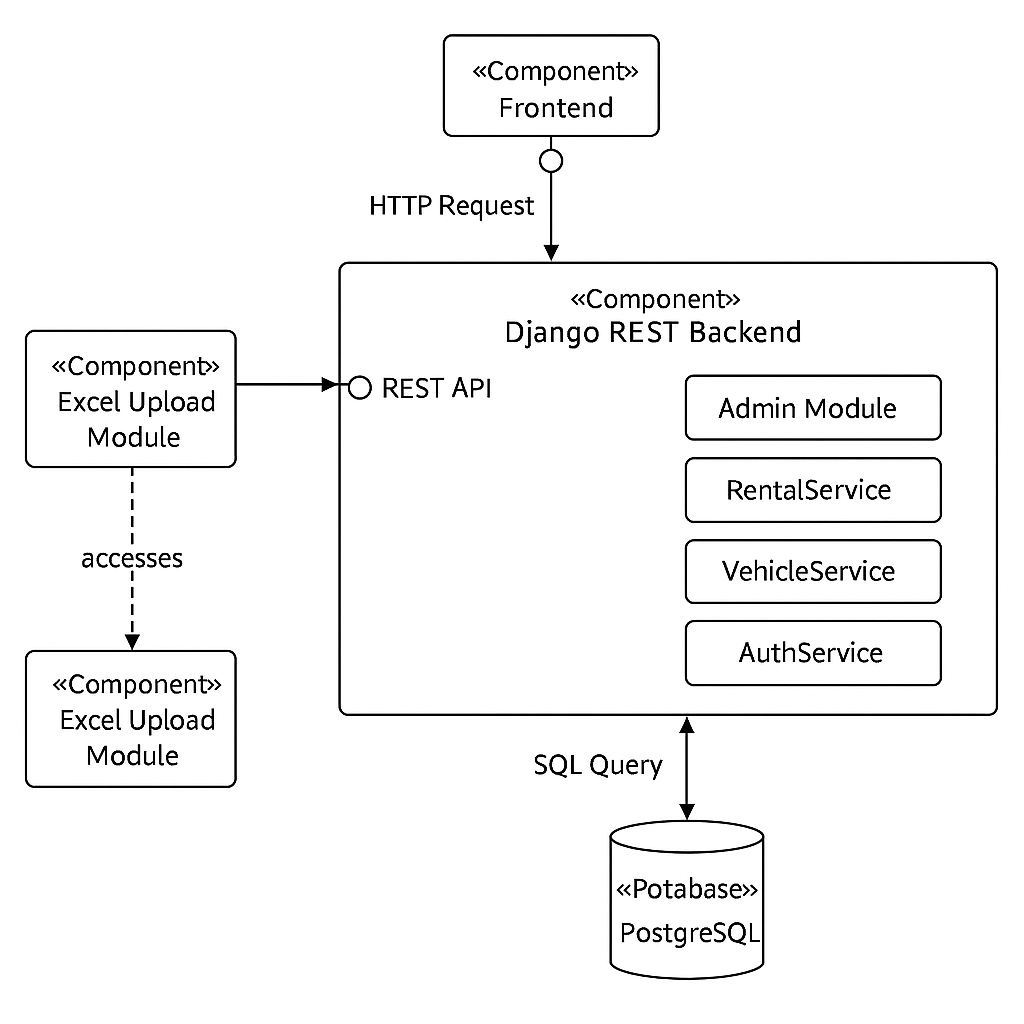


Fig D.1.6: Component Diagram: Django REST backend, PostgreSQL DB, frontend and API interactions



# Final Remarks & Sign-Off

## Revision History

| Version | Date | Author | Changes Made |
| --- | --- | --- | --- |
| 1.0 | 2025-07-08 | Dev Krishna | Finalized v1.SRS document with all appendices and diagrams. |

## Acknowledgements

This document was prepared by Dev Krishna for academic and professional project development purposes, and adheres to industry-grade software engineering documentation standards.

## End Note

This SRS marks the baseline scope and functional boundaries of the Rentavec application. Future versions may include extended features and refinements as outlined in the product roadmap.