<Rent a Car>

Requirements Specification and Analysis

<1.0>

<16.03.2025>

<Nehir Karameşe>

<Ayşe Korkut>

<Sıla Özcan>

<Ata Kılıç>

<Tuğçe Yeşilyurt>

<Hümeyra Bilgin>

<Eylül Buket Soğukbulak>

<Umut Enis Gögçe>

<Arda Aydoğdu>

<Arda Gül>

<Kaan Berk Kıral>

Prepared for

SOFT3102 Software Engineering Practice



Table of Contents

[1. Introduction 1](#_Toc433996772)

[1.1. Purpose of the System 1](#_Toc433996773)

[1.2. Design Goals 1](#_Toc433996774)

[1.3. Definitions, Acronyms, and Abbreviations 1](#_Toc433996775)

[1.4. References 1](#_Toc433996776)

[2. Current Software Architecture 1](#_Toc433996777)

[3. Proposed Software Architecture 1](#_Toc433996778)

[3.1. Overview 1](#_Toc433996779)

[3.2. System Decomposition 1](#_Toc433996780)

[3.3. Hardware Software Mapping 2](#_Toc433996781)

[3.4. Persistent Data Management 2](#_Toc433996782)

[3.5. Access Control and Security 2](#_Toc433996783)

[3.6. Global Software Control 2](#_Toc433996784)

[3.7. Boundary Conditions 2](#_Toc433996785)

[4. Subsystem Services 2](#_Toc433996786)

[5. References 2](#_Toc433996787)

SYSTEM DESIGN DOCUMENT[1]

# Introduction

## Purpose of the System

The purpose of the Rent a Car Management System is to provide users with a secure, fast, and convenient way to rent vehicles through their mobile devices. The system aims to eliminate the traditional hassles of car rental by offering a user-friendly platform where users can browse available vehicles, make reservations, and complete payments with ease. Our goal is to modernize the car rental industry by adapting it to today's digital world, making rentals more accessible anytime and anywhere.

## Design Goals

Our design approach focuses on simplicity, efficiency, and user experience. The system is built with the following principles in mind:

* **Intuitive UI/UX:** Ensuring users can navigate the app seamlessly without unnecessary complexity.
* **Aesthetic Consistency:** Colours, fonts, and design elements are selected to be easy on the eyes and enhance readability.
* **Efficient Workflow:** Users can rent a car in just a few steps, minimizing the time required for reservations.
* **Categorized Content:** Information is structured in an organized way to avoid confusion and improve accessibility.
* **Responsive Design:** The interface adapts to different screen sizes, providing an optimal experience across all devices.

## Definitions, Acronyms, and Abbreviations

**User**: any person who uses the system.

**Admin**: administrator of the system, can add and remove cars.

**RAD:** Requirement Analysis Document.

**GUI:** A GUI or graphical user interface is a form of user interface that allows users to interact with electronic devices through a graphical interface.

## References

References to existing systems, etc.

# Current Software Architecture

The Rent a Car Management System follows a modern web application architecture built using the Model-View-Controller (MVC) pattern. The system is designed to be scalable, maintainable, and efficient, ensuring a seamless rental experience for users. The core technologies used in the system include:

* **Backend:** **Django REST Framework (DRF)** – A powerful and flexible API development framework built on Django.
* **Frontend:** **React.js** – A dynamic and interactive user interface library for building web applications.
* **Database:** **PostgreSQL** – A reliable, high-performance relational database system.

The system is structured into three primary layers:

1. **Presentation Layer (Frontend – React.js)**
   * Implements the user interface (UI) and handles user interactions.
   * Communicates with the backend through RESTful API requests.
   * Uses React Router for smooth navigation and state management.
   * Ensures responsive design for an optimal experience across different devices.
2. **Business Logic Layer (Backend – Django REST Framework)**
   * Handles authentication, business rules, and data processing.
   * Exposes a REST API for communication between the frontend and backend.
   * Implements user roles (admin, customer, etc.) and permissions.
   * Manages vehicle inventory, reservations, and payments.
3. **Data Layer (SQLite Database)**
   * Stores user data, vehicle records, reservations, and transaction details in a lightweight file-based database.
   * Ensures data integrity and security using Django's ORM (Object-Relational Mapping).
   * Suitable for development and small-scale applications, with the flexibility to migrate to PostgreSQL for larger-scale deployments.

# Proposed Software Architecture

## Overview

The Rent a Car Management System is designed using a modular and layered architecture to ensure scalability, maintainability, and security. The system follows a three-tier architecture, where functionality is distributed across different subsystems:

1. **Frontend (React.js)** – Provides an interactive and responsive user interface for customers and administrators.
2. **Backend (Django REST Framework)** – Handles business logic, API management, authentication, and data processing.
3. **Database (SQLite)** – Stores and manages persistent data, including user details, reservations, and vehicle information in a lightweight, file-based database.

This architecture ensures separation of concerns, allowing independent development and maintenance of each layer while ensuring efficient communication through RESTful APIs. SQLite is ideal for development and small-scale applications, with the option to migrate to PostgreSQL for larger deployments.

## System Decomposition

The TECHSAN-RAC is designed with a React frontend, a Django REST API backend, and an SQLite database. The system is modular, ensuring scalability and maintainability.

**1. Frontend (React)**

The frontend provides a user-friendly interface and interacts with the backend via an API Client. It includes:

* **Authentication Pages** (Login, Logout, Forgot Password, Reset Password)
* **User Management** (Profile Page, Registration)
* **Main Application Pages** (Dashboard, Rental Services)

**2. Backend (Django REST API)**

The backend handles business logic and database operations through the following services:

* **Authentication Service:** Manages login/logout and session handling.
* **User Service:** Handles registration, profile management, and password resets.
* **Rental Service:** Manages car reservations and rental history.
* **Vehicle Service:** Stores and retrieves vehicle data.

**3. Database (SQLite)**

The database consists of:

* **Users Table:** Stores user accounts.
* **Rentals Table:** Manages reservations.
* **Vehicles Table:** Maintains available cars.

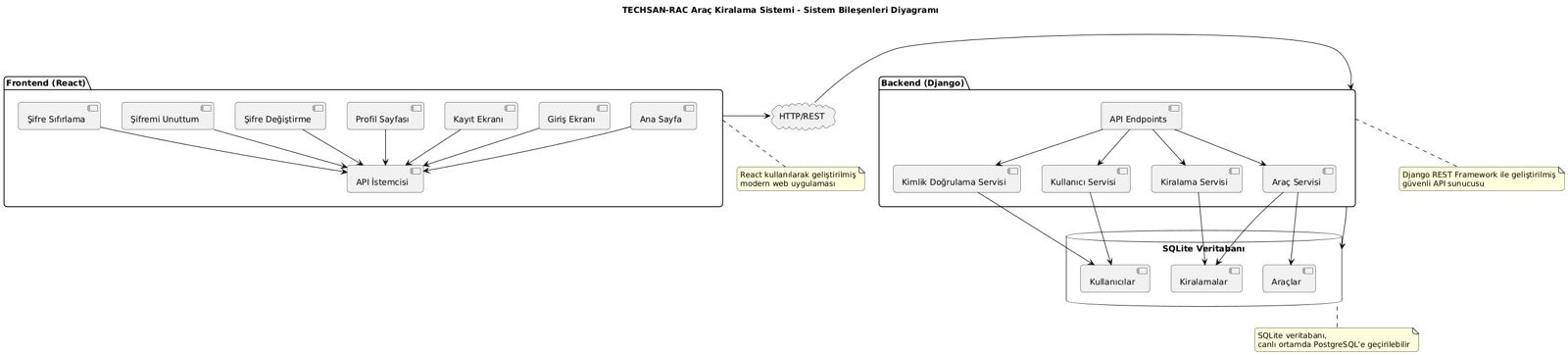
The frontend communicates with the backend via HTTP REST API, enabling a structured and efficient interaction between components. The system can migrate to PostgreSQL in production.

Figure 1:System Decomposition Diagram

## Hardware Software Mapping

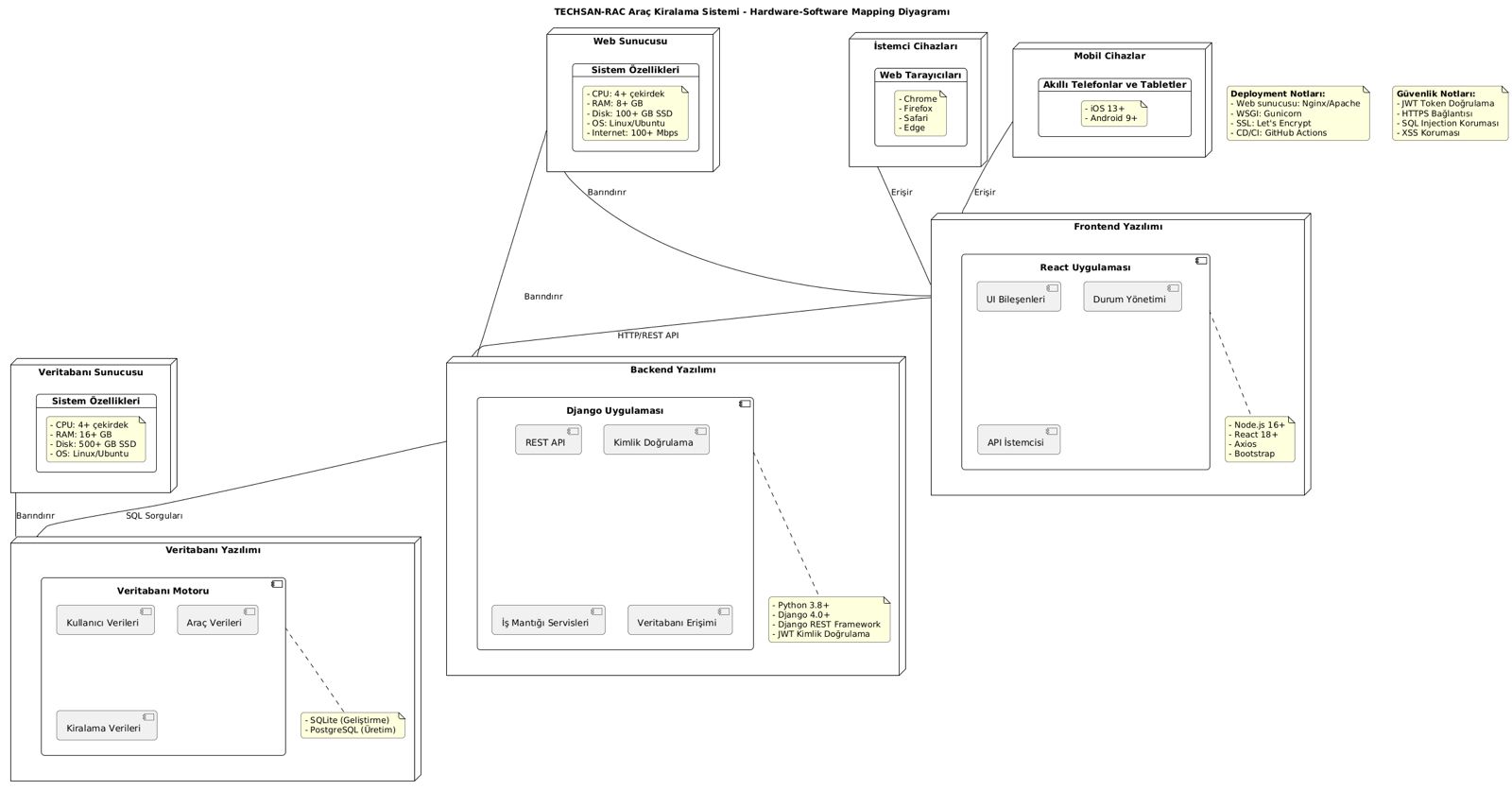
The TECHSAN-RAC Car Rental System establishes a structured relationship between hardware and software components. The system architecture includes a database server (SQLite), web server, backend (Django), and frontend (React), all communicating via HTTP/REST APIs. The backend processes requests and manages business logic, while the frontend provides an interactive user interface. Users can access the system through web browsers and mobile devices, ensuring flexibility and accessibility. Security measures such as JWT authentication, HTTPS encryption, SQL injection protection, and CI/CD deployment enhance system reliability and data integrity.

Figure 2:Hardware Software Mapping Diagram

## Persistent Data Management

The system securely stores persistent user data using an SQLite database, with the flexibility to migrate to PostgreSQL for scalability.

**Persistent Data:**

* **User Information:** Name, surname, phone number, birthdate, email, password (hashed).

çizgi, ekran görüntüsü, dikdörtgen, paralel içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

## Access Control and Security

## Global Software Control

## Boundary Conditions

The TECHSAN-RAC Car Rental System currently operates with a single entry point for user interaction, focusing on the user authentication process. The system's boundary conditions are defined as follows:

**1. System Startup**

* The system initializes with the user authentication module as the primary entry point.
* The frontend (React) loads the login and registration pages, while the backend (Django) starts the API services.
* The SQLite database is preconfigured to store user data.

**2. Normal Operation**

* Users can register, log in, and manage their accounts.
* The system handles session management to maintain secure access.
* User authentication is enforced using JWT tokens for security.

**3. Failure Scenarios & Handling**

* **Incorrect Login:** Displays an error message and prompts the user to retry.
* **Database Connection Failure:** The system logs errors and prevents access until the issue is resolved.
* **Expired Session:** Users are logged out and required to reauthenticate.

**4. Future Expansion Considerations**

* Currently, the system does not include rental or vehicle management.
* Additional features such as admin roles, transaction management, and reporting can be integrated in future updates.

# Subsystem Services

1. **Authentication Subsystem**

**Description:** Manages user authentication and session handling.

**Services:**

* **User Login Service:**
  + Accepts user credentials and validates them against the database.
  + Generates JWT tokens for session management.
  + Redirects users to the homepage upon successful authentication.
* **User Logout Service:**
  + Ends the user session and invalidates authentication tokens.
  + Redirects users to the login page after logout.
* **Password Reset Service:**
  + Handles forgotten passwords by sending a reset link via email.
  + Updates user passwords securely and enforces password policies.

1. **User Management Subsystem**

**Description:** Manages user **registration and profile updates**.

**Services:**

* **User Registration Service:**
  + Allows new users to sign up by providing personal details (name, surname, phone number, birthdate, email, password).
  + Validates and stores the user’s information securely.
  + Ensures passwords are hashed for security.
* **Profile Management Service:**
  + Enables users to view and update their profile details.
  + Allows users to change their password from the profile page.

1. **Database Management Subsystem**

**Description:** Handles data storage and retrieval for users.

**Services:**

* **User Data Service:**
  + Stores user details in the SQLite database.
  + Ensures data integrity and consistency using Django ORM.
* **Security & Backup Service:**
  + Encrypts sensitive information (e.g., passwords).
  + Supports automated database migrations and backups.

1. **API Gateway Subsystem**

**Description:** Manages communication between the frontend (React) and backend (Django REST API).

**Services:**

* Handles incoming API requests from the frontend.
* Routes authentication and user management requests to the appropriate subsystems.
* Ensures secure data transfer using HTTPS.

# References

The following is an example of listing a book in this section. Check the text to see how it is cross referenced (The whole document is based on [1]).

1. Bruegge B. & Dutoit A.H.. (2010). *Object-Oriented Software Engineering Using UML, Patterns, and Java*, Prentice Hall, 3rd ed.