**Table of Contents**

[1.0 Document Purpose 3](#_Toc94636300)

[2.0 Intended Audience 3](#_Toc94636301)

[3.0 Project Background, Objective(s) 4](#_Toc94636302)

[4.0 Design Pattern 5](#_Toc94636305)

[5.0 Solution Diagram 5](#_Toc94636306)

[6.0 Solution Steps 6](#_Toc94636307)

[7.0 Classes/function name 7](#_Toc94636308)

[8.0 Data model/Tables Diagram 8](#_Toc94636309)

[9.0 Use Case Diagram 8](#_Toc94636310)

[10.0 Data Flow Diagram 9](#_Toc94636314)

[11.0 API Canvas 10](#_Toc94636311)

**1.0 Document Purpose**

The documents contain a detailed description of the solution architecture of the on-demand Car Wash System.

**2.0 Intended Audience**

|  |  |
| --- | --- |
| Role | Nature of Engagement in the On Demand Car Wash System Architecture |
| Product Owner/SME | Key stakeholder to ensure that the architecture is aligned with business goals. |
| Business Analysts | Business analysts are one of the stakeholders who are informed with the key architectural decisions. |
| Enterprise Architects | To enforce Customer management Platform Architecture is aligned to business goals and architecture, architectural guidelines. |
| Developers | Use Technical Architecture Document as the guiding document for detail design and implantation approach to align with Customer management Microservice |
| End-User | An End- user can check the train timings, train fares and other trains information and book/cancel a ticket. |

**3.0 Project Background & Objectives**

**3.1 Project Background**

On Demand Car Wash System leads to perform Management of Car wash details where one can register themselves and perform various operations related to washing Cars.

**3.2 Project Objectives**

On Demand Car Wash System will perform various operations like instantly booking the car wash service or scheduling it for later.

The user of this system should first register with his/her email ID and their password for any interaction with the system. Once registered and after logging in the user should select the kind of activity, he would like to perform.

**3.3 System Requirements**

**3.3.0 Development Environment**

1. Database: Microsoft SQL Server Management Studio 18
2. Operating System: Windows 10 or higher
3. SDK and IDE: .NET 5, Visual Studio 2019 or Higher, Visual Studio Code
4. Web Browser: Latest Browser
5. Internet Connectivity
6. 8 GB RAM (Recommended)
7. Minimum 4 GB of free storage
8. Processor – 2 GHz or higher

**3.3.1 Testing Environment**

1. Database: Microsoft SQL Server Management Studio 18
2. Operating System: Windows 10
3. SDK and IDE: .NET 5, Visual Studio 2019 or Higher, Visual Studio Code
4. Web Browser: Latest Browser
5. Internet Connectivity
6. 8 GB RAM (Recommended)
7. Minimum 4 GB of free storage
8. Processor – 2 GHz or higher
   * 1. **Hosting Environment**
9. Web Browser: Latest Browser
10. 8 GB RAM (Recommended)
11. Minimum 4 GB of free storage
12. Processor – 2 GHz or higher

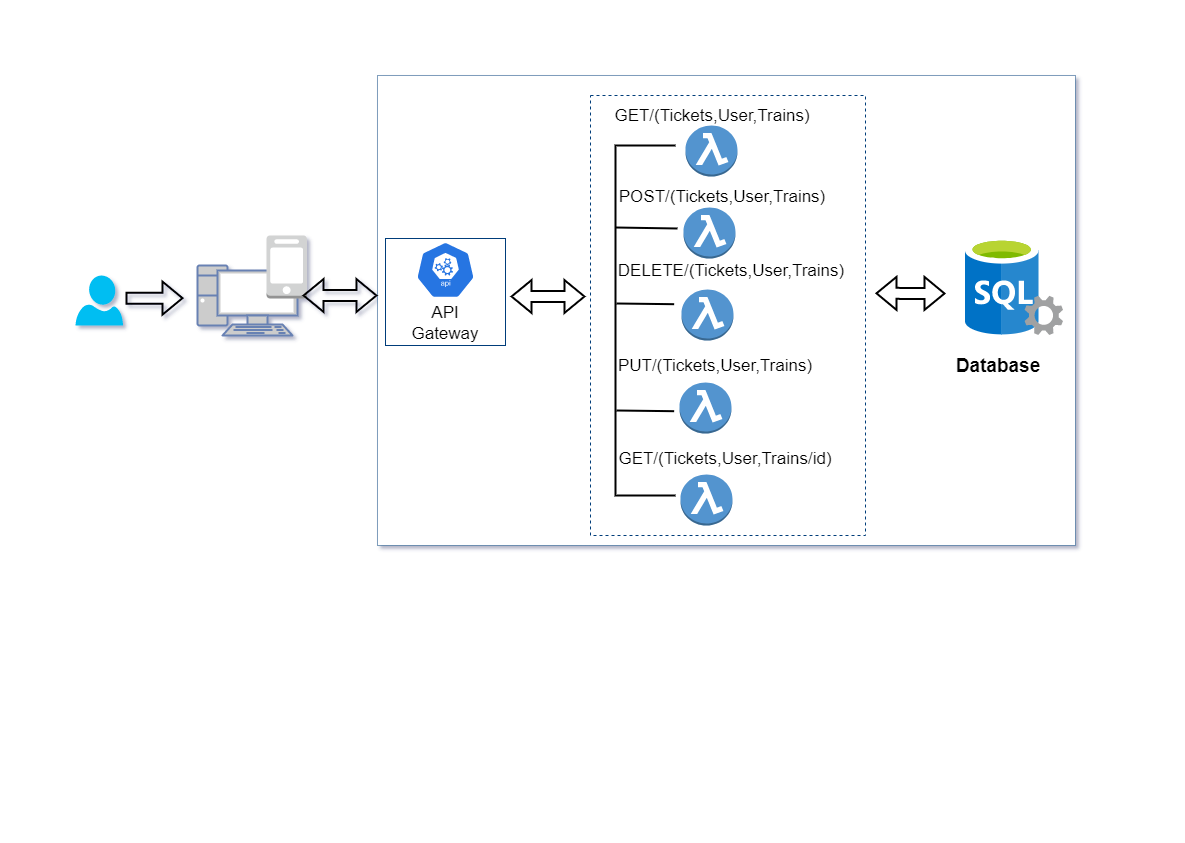
**3.4 Technologies Used**

* 1. Angular 10
  2. ASP .Net Web API (.NET Version 3.1)
  3. Microsoft SQL Server Management Studio 18

**4.0 Design Pattern**

|  |  |  |
| --- | --- | --- |
| Serial no. | Name | Description |
| 1 | Angular | Creating a user interface (Front-end), and consuming API services. |
| 2 | Database | For storing, maintaining and accessing user, train and booking details. |
| 3 | API | Using HTTP requests, we will use the respective action to trigger various operations |

**5.0 Solution Diagram**

****

**6.0 Solution Steps**

**6.1 User**

**Registering User**

1. User will be able to register himself by entering the details like Name, age, sex, address, Phone, email and Password.
2. After filling the user credentials the form is validated.
3. If the validation is successful, by clicking the submit button browser directs the request to customer registration API.
4. The call reaches the API gateway.
5. API gateway does the routing and saves the data in the database.
6. Once a user is successfully registered an alert is displayed and the user is redirected to the login page.

**Viewing Train and Fare Details**

1. User will be able to view trains without logging in.
2. User can view trains by providing Destination Station, Arrival Station and date of journey.
3. Call reaches the API gateway.
4. **GetTrains(parameter1, parameter2,parameter3)** is invoked.
5. If the trains are found with the given parameters it is displayed along with their fare, to the user otherwise an alert (‘No trains found’) is displayed.

**Booking of tickets**

1. After a user found their suitable train, the user will click on Book Now button.
2. In order to book the train, the user will have to log in first.
3. The user will be redirected to the Log in page where they can fill their details and if they do not have an account they can click on Sign-up.
4. Once the user is logged in, they can add passenger (max. 6 at a time).
5. Once the passengers are added fare details are updated and displayed to the user.
6. The user can then Pay and Book a ticket.
7. An alert is shown after successful booking.
8. A user can track their booking status by clicking on bookings.
9. The API will then getUserBookings() by user ID.

**Cancellation of tickets**

1. User can also cancel a booked ticket if they require.
2. When the user clicks on booking and sees their status there will be a cancel button.
3. After clicking on cancel button an alert is displayed (‘Are you sure you want to cancel?’)

**6.2 Admin**

1.Admin will be able to edit and add new train, add news Classes in trains and fare details and get report of all the Passengers for a particular train.

2.GetAllTrains() will let the admin view all the train details.

3.GetCustomerById() will allow Admin to view customer details by ID.

**7.0 Classes/Functions**

|  |  |  |
| --- | --- | --- |
| Serial no. | Class | Description |
| 1 | Model Class | Model for holding the booking schema details for user. |
| 2 | Repository | The Interface in Data Access Layer for the user. |
| 3 | Controller | Controller handles the incoming HTTP requests and send the response back to the caller. |
| 4 | Services | It’s the Business Access Layer holding the Business Logic and meditates the communication between the controller and repository (Data Access) Layer. |
| 5 | Exception Handlers | Exception Handlers handles all the exceptions that which are revealed during runtime. |

**8.0 Database Diagram**

**User Table**

|  |  |  |
| --- | --- | --- |
| Sl no. | Name | Type |
| 1 (PK) | **ID** | **Varchar(10)** |
| 2 | **Full Name** | **Varchar(50)** |
| 3 | **Email** | **Varchar(50)** |
| 4 | **Token** | **Varchar(100)** |
| 5 | **Enabled** | **Boolean** |
| 6 | **Password** | **Varchar(20)** |

**Wash Packs**

|  |  |  |
| --- | --- | --- |
| Sl no. | Name | Type |
| 1 (PK) | **ID** | **Varchar(20)** |
| 2 (FK) | **Package Name** | **Varchar(50)** |
| 3 | **Cost** | **Numeric** |
| 4 | **Description** | **Varchar(100)** |

**Ratings**

|  |  |  |
| --- | --- | --- |
| Sl no. | Name | Type |
| 1 (PK) | **ID** | **Numeric** |
| 2 | **Name** | **Varchar(50)** |
| 3 | **Comments** | **Varchar(20)** |
| 4 | **Rating** | **Numeric** |

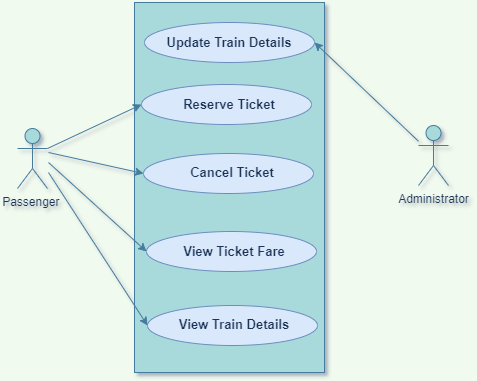
**Car**

|  |  |  |
| --- | --- | --- |
| Sl no. | Name | Type |
| 1 (PK) | **ID** | **Numeric** |
| 2 | **Name** | **Varchar(20)** |
| 3 | **Model** | **Varchar(20)** |

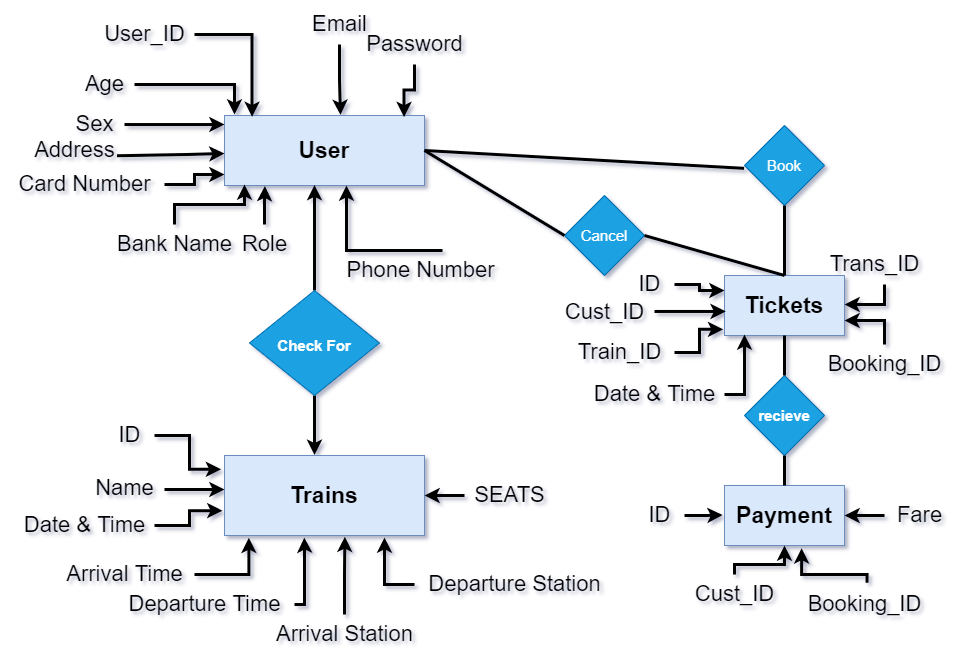
**Order details**

|  |  |  |
| --- | --- | --- |
| Sl no. | Name | Type |
| 1 (PK) | **OrderID** | **Varchar(20)** |
| 2 (FK) | **UserEmailID** | **Varchar(20)** |
| 3 | **WasherName** | **Varchar(20)** |
| 4 | **Washpack** | **Varchar(20)** |
| 5 | **PhoneNo** | **Numeric** |
| 6 | **Area Pincode** | **Numeric** |
| 7 | **Status** | **Varchar(10)** |

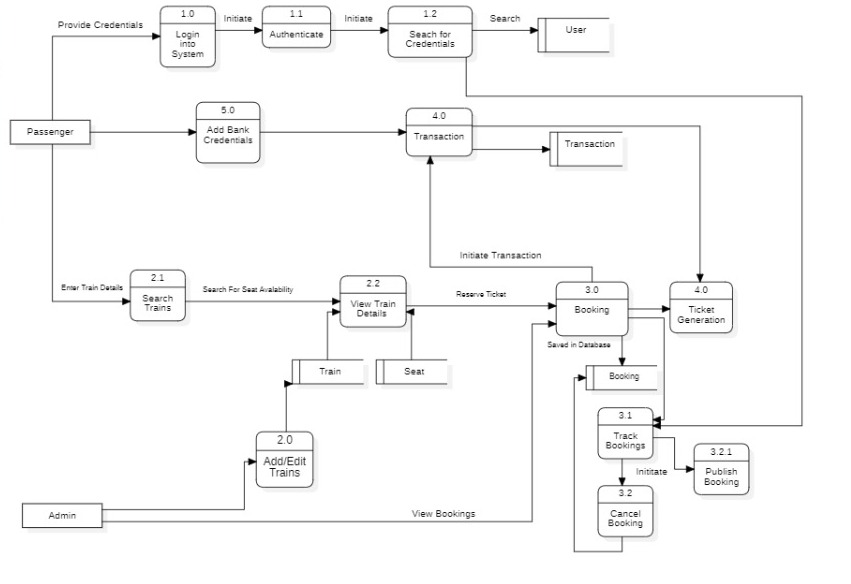
**9.0 Use Case Diagram**

****

**10.0 Entity Relationship Diagram**

****

**11.0 Data Flow Diagram**

****

**12.0 API Canvas**

**12.1 User**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Service | Path | Verb | API Description | Role | Auth |
| Customer-management | /Customer | POST | To register a customer | No | True |
| Customer-management | /Customer/Id | GET | To get a customer by Id | Admin | True |
| Customer- management | /Customer | GET | To get the list of customers | Admin | True |

**12.2 Booking**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Service | Path | Verb | API Description | Role | Auth |
| Booking-management | /booking | POST | To reserve a ticket | No | True |
| Booking-management | /Booking/Id | DELETE | To delete a Booking | No | True |
| Booking-management | /Booking | PUT | To update Booking details | No | True |
| Booking-management | /Booking/Id | GET | To get the Booking  Details by Id | No | True |

**12.3 Train**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Service | Path | Verb | API Description | Role | Auth |
| Train-management | /train | POST | To add Booking | Admin | True |
| Train-management | /Booking/ID | PUT | To update the details of a train | Admin | True |
| Train-management | /train | GET | To get the list of the trains | No | True |