Final Report - RentAPlace

An online platform for renting homes

Name : Balaga Sairam

Date : 08 – 09 - 2025

Batch Name : WIPRO NGA - .Net Full Stack Angular - FY26 – C2

Instructor : Jyoti S Patil (Trainer)

Table of Contents:

Content		page no.
1.Project Definition	---	3
2.Project Objective	---	4
3.Frontend Architecture	---	5 - 7
4.Backend Architecture	---	8 - 11
5.Component Breakdown & API Design	---	12 - 13
6. Database Design & Storage Optimization	---	14 - 15
7.Swagger Screenshots	---	16 - 17

Project Definition :

Rent A Place is an online application that bridges the gap between **property renters** and **owners** through a centralized platform.

- **For Renters:**

  Users can register, log in, and search for properties based on **location, availability, type, and features**. They can view detailed property information, including **photos, ratings, and categories**, and make reservations. Renters can also send messages to owners within the platform.

- **For Owners:**

  Owners can register, log in, and manage multiple properties by adding, updating, or deleting listings. They can receive **notifications** when a reservation is made, manage reservation status, and respond to user messages.

- **Technical Scope:**

  - **Frontend:** Angular (UI/UX, search, filtering, and reservation workflows).
  - **Backend:** ASP.NET Core MVC with RESTful APIs, JWT authentication, and business logic encapsulated in services.
  - **Database:** SQL Server (with Entity Framework Core ORM for schema management, CRUD operations, and relationships).
  - **Messaging:** A separate Web API for sending/receiving messages.
  - **Media Management:** Server folder for storing and retrieving property images.
  - **Documentation & Testing:** Swagger UI for API testing and validation.

The system will be developed in **three sprints** covering use case design, schema creation, user/property management, search/reservations, messaging, and notifications.

Project Objectives :

The main objective of the Rent A Place system is to develop a full-stack online rental platform that connects **renters (users)** with **property owners**. The system should enable renters to search, view, and reserve properties, while owners can manage their property listings and communicate with users.

The platform aims to:

- Provide a seamless **search and reservation experience** for renters.
- Enable **property management** (add, update, delete, view) for owners.
- Facilitate **secure messaging and notifications** between renters and owners.
- Ensure **scalability and maintainability** by using modern frameworks (Angular, ASP.NET Core, and Entity Framework).
- Deliver an **Airbnb-like user experience** with essential features for property rental.

Frontend Architecture (Angular)

The **Rent A Place** application frontend is built using **Angular** to provide a modular, scalable, and maintainable user interface. The architecture follows Angular 's best practices with **feature-based modules**, **shared reusable components**, and **state management**.

1. Modules and Responsibilities

- **Core Module**
  - Authentication services (AuthService)
  - API communication services
  - Interceptors for JWT token and error handling
  - Route guards (AuthGuard, OwnerGuard, RenterGuard)
- **Shared Module**
  - Common UI components (buttons, modals, property cards)
  - Reusable pipes (currency, date formatting)
  - Reusable directives (lazy loading, validation)
- **Feature Modules**
  - **Auth** → Login, Register, Logout for both renters and owners
  - **Renter** → Property search, filters, reservation, messages
  - **Owner** → Property CRUD, reservation confirmation, messages
  - **Property** → Property details, image gallery, reviews, ratings
  - **Notification** → Email and in-app notifications
- **Layouts**
  - MainLayout → Header, Footer, Navbar for public/renter views
  - OwnerDashboardLayout → Sidebar, Dashboard UI for property owners

2. Routing Strategy

- **Public Routes** → Login, Register, Search, Property Details
- **Renter Routes** → Reservations, Messaging (protected by AuthGuard)
- **Owner Routes** → Property Management, Messaging, Reservation Confirmation (protected by OwnerGuard)

3. State Management

The application uses a **global state management library** such as **NgRx** (Redux pattern) to maintain consistency across modules.

- **Auth State** → User info, JWT token
- **Property State** → Search results, property details
- **Reservation State** → Active reservations, history
- **Message State** → Conversations between users and owners
- **Notification State** → Alerts, updates

4. API Communication

- **HttpClient** is used for backend API communication.
- Services are placed in core/services/:
  - AuthService → Login, Register, Logout
  - PropertyService → Search, CRUD operations
  - ReservationService → Reserve, Cancel, Confirm
  - MessageService → Send and fetch messages
  - NotificationService → In-app and email notifications

  Interceptors handle:
- JWT token attachment in requests
- Unauthorized (401/403) error redirection

5. UI/UX Considerations

- **Angular Material** or **Bootstrap** for consistent, responsive design
- **Lazy loading** for feature modules to improve performance
- **Reusable components** for property cards, image galleries, forms
- **Angular Router** for smooth navigation
- **Accessibility & responsiveness** for mobile and desktop users

Backend Architecture (ASP.NET Core) :

The **Rent A Place** backend is designed using **ASP.NET Core MVC** with **RESTful APIs**, **Entity Framework Core (EF Core)** as the ORM, and **SQL Server** as the database. The architecture ensures **modularity, scalability, and maintainability** by separating concerns into **Controllers, Services, Repositories, and Models**.

1. Layered Architecture

- **Controllers (API Layer)**

  Handle incoming HTTP requests, validate inputs, and return responses.

  Example: PropertyController → /api/properties/search

- **Services (Business Logic Layer)**

  Contain the core logic of the application.

  Example: ReservationService checks property availability before booking.

- **Repositories (Data Access Layer)**

  Interact with the database using EF Core.

  Example: PropertyRepository → Fetch properties with filters.

- **Models (Entities)**

  Represent database tables as C# classes (EF Core entities).

- **DTOs (Data Transfer Objects)**

  Define request/response objects to prevent exposing raw entities.

3. Database Communication

- **Entity Framework Core (ORM)** is used for database interaction.
- **DbContext (RentAPlaceContext)** manages entity sets:
  - DbSet<User>
  - DbSet<Property>
  - DbSet<Reservation>
  - DbSet<Message>

4. API Endpoints

AuthController

- POST /api/auth/register → Register user/owner
- POST /api/auth/login → Login & get JWT token
- POST /api/auth/logout → Logout

PropertyController

- POST /api/properties → Add property (owner only)
- GET /api/properties → Search/filter properties
- GET /api/properties/{id} → Get property details
- PUT /api/properties/{id} → Update property
- DELETE /api/properties/{id} → Delete property

ReservationController

- POST /api/reservations → Create reservation
- GET /api/reservations/user/{id} → Get user reservations
- PUT /api/reservations/{id}/confirm → Confirm reservation (owner only)

MessageController

- POST /api/messages → Send message
- GET /api/messages/{conversationId} → Get conversation

NotificationController

- GET /api/notifications/user/{id} → Fetch notifications
- Email notifications triggered on reservation creation

5. Security

- **JWT Authentication** for protecting APIs (renter/owner roles).
- **Role-based Authorization** ([Authorize(Roles="Owner")]).
- **Input Validation** using FluentValidation/Data Annotations.
- **Exception Handling Middleware** for consistent API error responses.

6. Middleware

- **JWT Middleware** → Validates tokens for protected routes.
- **Exception Middleware** → Centralized error handling.
- **CORS Policy** → Allows frontend Angular app to call APIs.

8

7. Notifications

- **Email Notifications** → Sent via SMTP (owner notified of reservation).
- **In-app Notifications** → Stored in DB, fetched via NotificationController.

8. Documentation & Testing

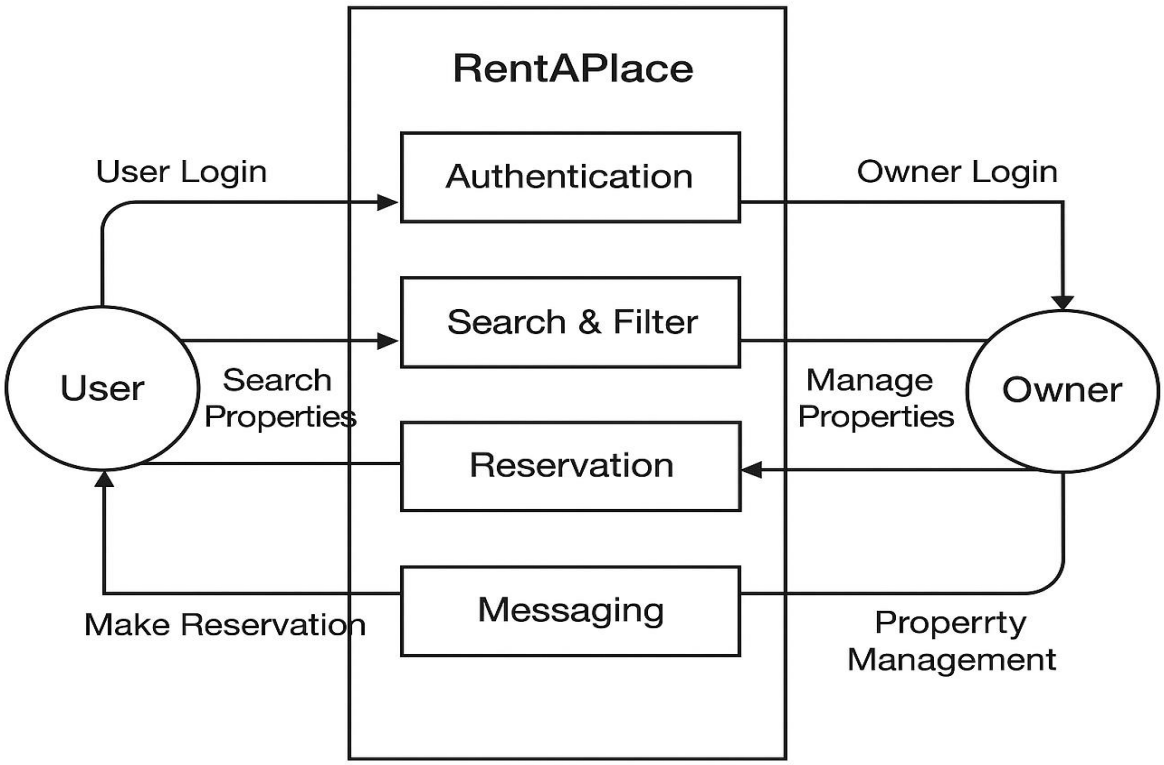
- **Swagger UI** integrated for API documentation.
- **Postman / Unit Tests (xUnit or NUnit)** for API testing.

This architecture ensures:

- **Modularity** (clear separation of controllers, services, repositories).
- **Maintainability** (easy to extend features like payments, reviews).
- **Security** (JWT, role-based access).
- **Scalability** (support for microservices if needed later).

10

SYSTEM DESIGN DIAGRAM:



Component Breakdown & API Design :

1. Frontend Component Breakdown (Angular) : The frontend is structured into feature-based modules with reusable shared components and global state management.

a. State Management

- NgRx (Redux pattern) or Akita is used to maintain consistent application state.
- Main state slices:
  - Auth State → Current user, JWT token, roles (user/owner)
  - Property State → Search results, property details, top-rated properties
  - Reservation State → Active reservations, reservation history
  - Message State → Conversations between renters and owners
  - Notification State → In-app notifications, email alerts

b. Routing

Angular Router is used to manage public, renter, and owner routes.

- Public Routes: Login, Register, Property Search, Property Details
- Renter Routes: Search results, Reservation history, Messaging
- Owner Routes: Property CRUD, Reservation confirmation, Messaging

c. UI Components

- **Core UI**

  Header, Footer, Navigation Bar, Sidebar (for Owner Dashboard)
- **Shared UI**
  - Property Card (used in search results and favorites)
  - Image Carousel (property images)
  - Renter → SearchComponent, ReservationListComponent, MessageBoxComponent
  - Owner → PropertyFormComponent, PropertyListComponent, ReservationApprovalComponent
  - Property → PropertyDetailsComponent, ReviewComponent

2. API Design (Backend)

The backend follows **RESTful API design principles** with a clear separation of resources.

a. Authentication Mechanism

- **JWT (JSON Web Token) Authentication**
  - On login, the server issues a JWT containing user role (User/Owner).
  - Token is attached to each request in the Authorization: Bearer <token> header.
  - Role-based access control ensures secure endpoints.

3. Example Flow (Reservation)

1. **Renter** logs in → receives JWT.
2. Renter searches properties (GET /api/properties).
3. Renter reserves a property (POST /api/reservations).
4. **Owner** receives email + in-app notification (NotificationService).
5. Owner confirms reservation (PUT /api/reservations/{id}/confirm).

Database Design & Storage Optimization:

1. Database Design

The **Rent A Place database** is built using **SQL Server** with **Entity Framework Core ORM** for schema management. The design ensures **data integrity, normalization, and efficient relationships** between entities.

Main Entities

1. **Users** – Stores renter and owner accounts.
2. **Properties** – Contains property details (location, type, features, images).
3. **Reservations** – Links renters with properties and stores reservation details.
4. **Messages** – Supports communication between renters and owners.
5. **Notifications** – Stores in-app and email notifications.
6. **PropertyImages** – Stores multiple images for each property.

ERD Relationships

- **Users – Properties:** One-to-Many (An owner can list multiple properties).
- **Users – Reservations:** One-to-Many (A renter can book many reservations).
- **Properties – Reservations:** One-to-Many (A property can have multiple reservations).
- **Properties – Images:** One-to-Many (Each property has multiple images).
- **Users – Messages:** One-to-Many (Users can send multiple messages).
- **Users – Notifications:** One-to-Many (Each user can have many notifications).

3. Storage Optimization Techniques

To ensure **fast queries, scalability, and efficient storage**, the following techniques are applied:

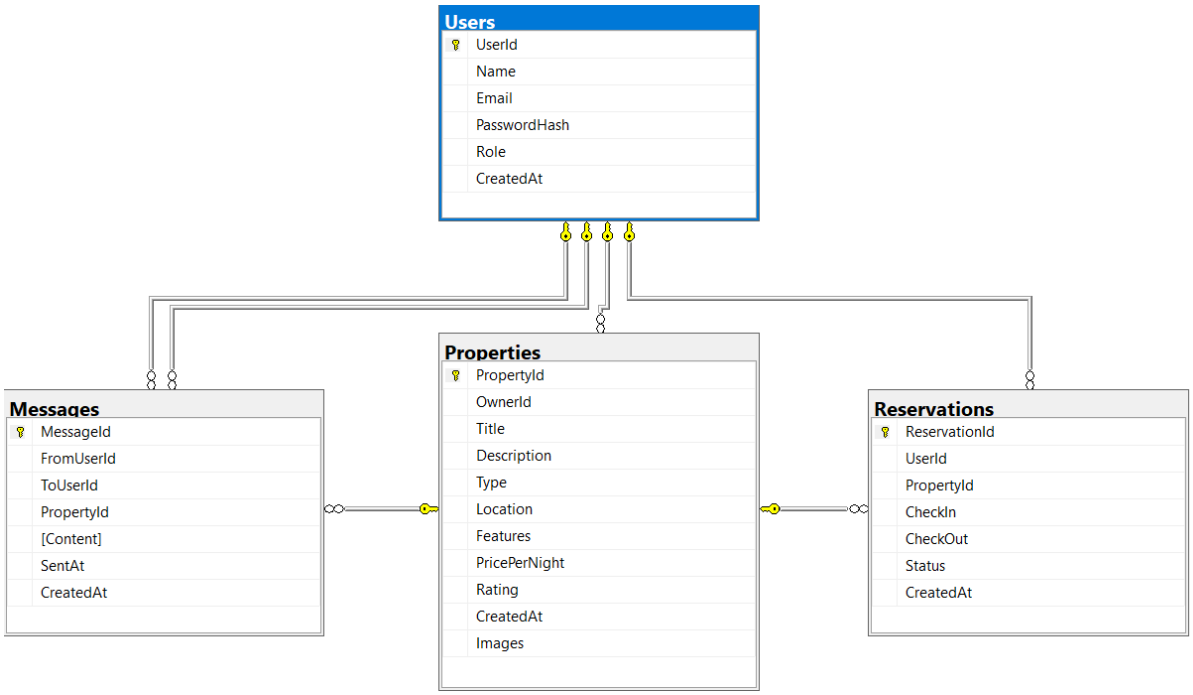
a. Indexing

- **Clustered Index** on primary keys (UserId, PropertyId, ReservationId).
- **Non-clustered Indexes** on:
  - Email (for login queries).
  - Location, PropertyType, PricePerNight (for property search).

b. Query Optimization

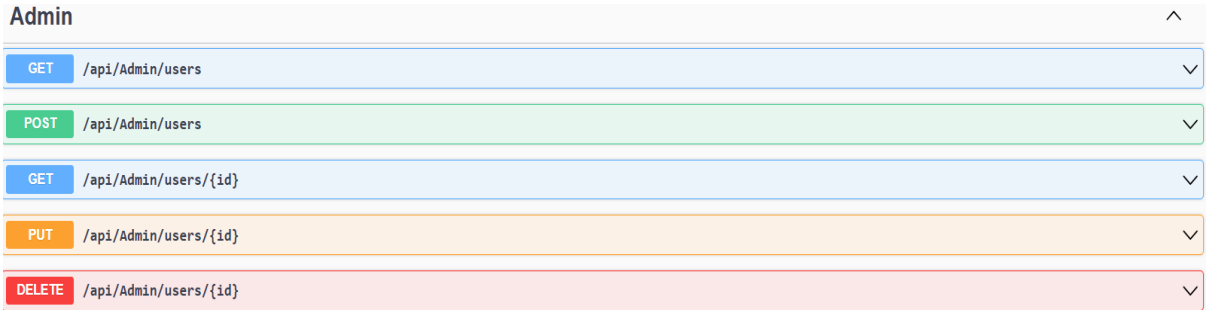
- Use **parameterized queries** to prevent SQL injection and improve caching.
- Apply **JOINs with proper indexing** to avoid full table scans.
- Use **pagination (OFFSET-FETCH / LIMIT)** for property listings.
- Use **eager loading (Include) in EF Core** to fetch related entities (like property + images) efficiently.

ER diagram:

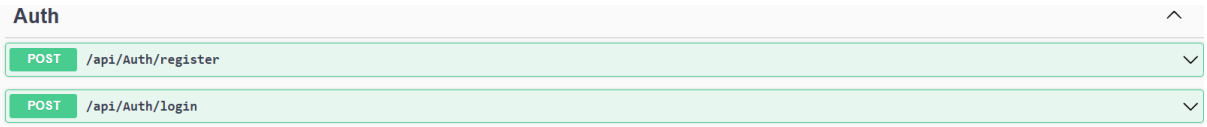


Swagger Screenshots :

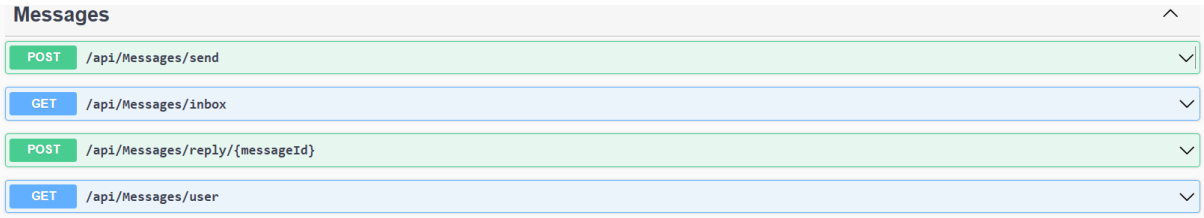
Admin:



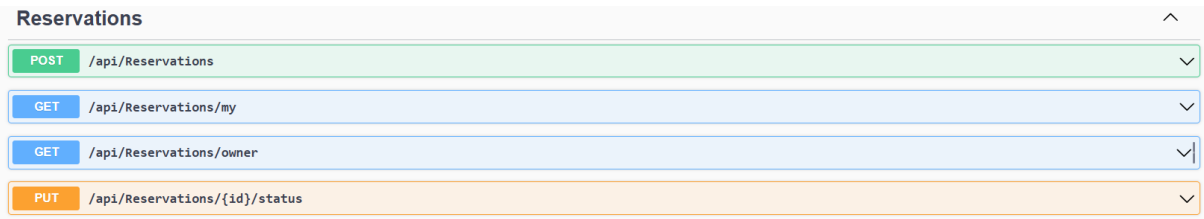
Auth:



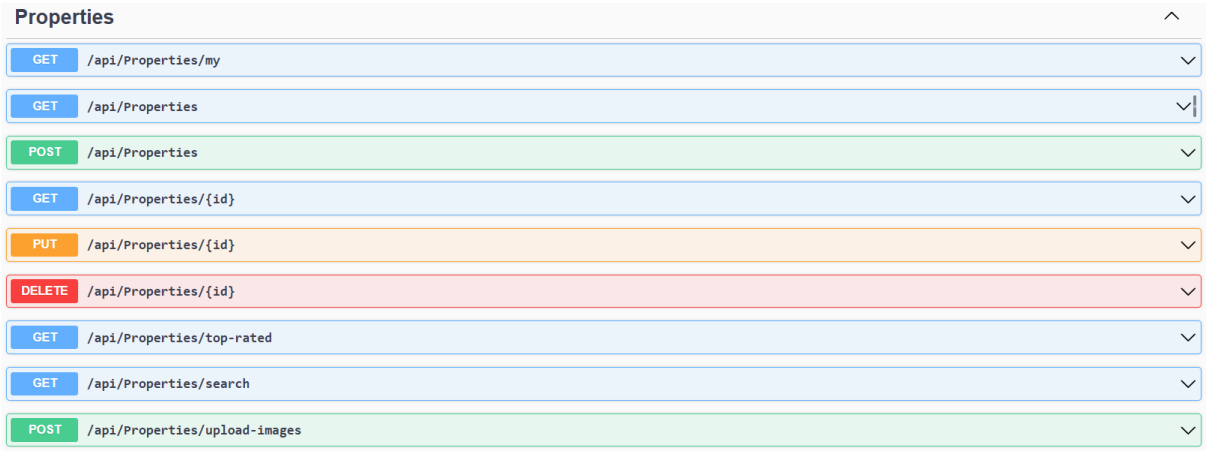
Messages:



Reservations:



Properties:



Email: