**Technologies Used:**

* HTML ,CSS ,JAVASCRIPT’S framework JQUERY
* ASP.NET CORE for backend

**Project Description:**

1. The project consists of a Property Rental management website where there will be 3 different type of users using the application with different roles. property owners and administrators have full control over the website, with the ability to manage property manager accounts and tenant accounts. Property managers can perform CRUD operations related to buildings and apartments, keep track of apartment statuses, schedule appointments with potential tenants, and report to property owners. Tenants can create online accounts, search for suitable apartments, schedule appointments with property managers, and communicate with them. Each user (Tennant, manager, admin) can only do the tasks they are designated to do and are able to Log in, Log out and register.

Database:

**1. \_\_EFMigrationsHistory Table:**

Columns:

MigrationId (Primary Key): A unique identifier for each migration.

ProductVersion: The version of Entity Framework used for migrations.

**2. TblApartment Table:**

Columns:

ApartmentId (Primary Key): Auto-incremented identifier for each apartment.

BuildingId (Foreign Key): Relates to the BuildingId in the TblBuilding table.

Address: The address of the apartment.

Price: The price of the apartment.

Bedrooms: Number of bedrooms in the apartment.

Bathrooms: Number of bathrooms in the apartment.

Status: Current status of the apartment.

Createdby: User ID who created the record.

CreatedDateTime: Date and time when the record was created.

Updatedby: User ID who last updated the record.

UpdatedDateTime: Date and time of the last update.

Primary Key: ApartmentId

Foreign Key: BuildingId (references TblBuilding(BuildingId))

**3. TblAppointment Table:**

Columns:

AppId (Primary Key): Auto-incremented identifier for each appointment.

TenantId (Foreign Key): Relates to the UserID in the TblUser table for tenants.

ManagerId (Foreign Key): Relates to the UserID in the TblUser table for managers.

ApartmentId (Foreign Key): Relates to the ApartmentId in the TblApartment table.

SuggestedDateTime: Suggested date and time for the appointment.

FromDateTime: Start date and time of the appointment.

Status: Current status of the appointment.

Createdby: User ID who created the record.

CreatedDateTime: Date and time when the record was created.

Updatedby: User ID who last updated the record.

UpdatedDateTime: Date and time of the last update.

Primary Key: AppId

Foreign Keys:

TenantId (references TblUser(UserID))

ManagerId (references TblUser(UserID))

ApartmentId (references TblApartment(ApartmentId))

**4. TblBuilding Table:**

Columns:

BuildingId (Primary Key): Auto-incremented identifier for each building.

BuildingName: Name of the building.

PropertymangerId (Foreign Key): Relates to the UserID in the TblUser table for property managers.

Status: Current status of the building.

Createdby: User ID who created the record.

CreatedDateTime: Date and time when the record was created.

Updatedby: User ID who last updated the record.

UpdatedDateTime: Date and time of the last update.

Primary Key: BuildingId

Foreign Key: PropertymangerId (references TblUser(UserID))

**5. TblUser Table:**

Columns:

UserID (Primary Key): Auto-incremented identifier for each user.

FirstName: First name of the user.

LastName: Last name of the user.

Email: Email address of the user.

ContactNo: Contact number of the user.

Address: Address of the user.

Pasword: Password of the user.

Status: Current status of the user.

Role: Role of the user (e.g., Admin, PropertyManager).

Createdby: User ID who created the record.

CreatedDateTime: Date and time when the record was created.

Updatedby: User ID who last updated the record.

UpdatedDateTime: Date and time of the last update.

Primary Key: UserID

**2/3**

**Backend Language and Details:**

I used **ASP.NET** Core MVC For my web app.

The web pages(views)

-Account

-create

-delete

-edit

-Index

-LogIn

-Appointment

-Request Submit For Appointment

-Home

-Index

-Reports

-Completed Request

-Incomming reports

-Reports Index

-Tennants Report

-Setup

-Admin index

-Appartment index

-Building index

-CRUD admin

-CRUD Apartments

-CRUD Building

-CRUD property manager

-Property Manager Index

The **controllers** I have made are

-Account controller

-(Retrieves user info based on provided credentials, creates user claims and signs in using Cookie Authentication.)

-Displays view for creating a new tenant.

-Creates a new tenant with hashed password.

-Records creation details, including creator's user ID.

-Retrieves existing user data for editing.

-Retrieves paginated tenant data for DataTables.

-Displays view for deleting a user.

-Deletes specified user after confirmation.

-Checks if a user with a given ID exists in the database.

- Redirects to Home/Index for authenticated users.

-Appointment controller

- Retrieves apartment details for appointment request.

- Populates dropdowns for property managers and apartments.

- Records creation details, including creator's user ID.

- retrieves a list of property managers and appartments from the database.

-Home controller

- Retrieves data from the TblApartment table in the database.

-Reports controller

- Retrieves data from the TblAppointment table in the database.

- Handles a POST request to look up incoming appointment reports with the status "Unread."

- Updates the status of an appointment to "Unread."

- Redirects to the ReportsIndex action.

- Handles a POST request to look up completed appointment reports with the status "Approved.

- Performs filtering based on the search value.

- retrieves data from the TblAppointment, TblApartment, and TblBuilding tables in the database, joining them appropriately.

- Handles a POST request to look up appointment reports with the status "Request."

- Returns JSON data for DataTables with information about appointments

-SetUp controller

- handles building-related operations, including viewing, adding, editing, and deleting buildings. It incorporates AJAX-enabled methods for dynamic data retrieval and interaction.

- handles appartment-related operations, including actions for viewing, adding, editing, and deleting. It includes dropdown lists for associating apartments with specific buildings.

- The use of Data Transfer Objects (DTO\_ClsBuilding and Dto\_ClsApartment) streamlines data handling and assist in filtering and pagination of building and apartment data.

- Methods such as AdminIndex, LookUpAdmin, and InsertUpdateDeleteAdmin handle the viewing, filtering, and manipulation of user data with the "Admin" role, incorporating password hashing for security.

- DTO\_ClsUser ensures structured handling of user-related data

- The controller employs MD5 hashing to secure user passwords

I used jQuery, a simple and efficient JavaScript framework. It boosted client-side scripting by swiftly interacting with HTML elements, making DOM manipulation and server communication through AJAX requests easier.

1. **Dynamic DOM Manipulation:**
   * jQuery played a key role in changing webpage elements as users interacted, instantly modifying the interface without refreshing the entire page for a smooth user experience.
2. **Data Loading with AJAX:**
   * Using jQuery's AJAX capabilities, the system could fetch data from the server smoothly, benefiting tasks like retrieving property details, tenant info, and appointment schedules without interrupting the user.
3. **Event Handling and User Interactions:**
   * jQuery streamlined event handling for efficient capture and response to user interactions. This facilitated features like appointment scheduling, property selection, and interactive elements for a more engaging user experience.
4. **Cross-browser Compatibility:**
   * jQuery's cross-browser compatibility ensured consistent behavior across different web browsers, reducing compatibility issues and providing a uniform experience for all users.
5. **Simplified DOM Traversal and Manipulation:**
   * jQuery's simplified syntax for DOM traversal and manipulation significantly reduced development time and complexity. Selecting elements, modifying content, and managing styles became more straightforward and efficient tasks.

**4**.

|  |  |  |
| --- | --- | --- |
| **Tenant** | **Create account** | **works** |
| **tenant** | **View appartment** | **works** |
| **tenant** | **Make appointment** | **works** |
| **Property Manager** | **Crud Building** | **works** |
| **tenant** | **Send message** | **Dosent work** |
| **Admin** | **Log In/Log out** | **works** |
| **Tenant** | **Log In/Log out** | **works** |
| **Property Manager** | **Log In/Log out** | **works** |
| **Property Manager** | **Crud Appartment** | **works** |
| **Admin** | **Create manager** | **works** |
| **Admin** | **Full functionality** | **works (kinda)** |
| **Manager** | **Send report to admin** | **works** |