PropertyManagement.API

Auth	
POST	/api/Auth/Register
POST	/api/Auth/Login
CloudStorage	
POST	/api/CloudStorage/upload
POST	/api/CloudStorage/upload-image
GET	/api/CloudStorage/download/{publicId}
DELETE	/api/CloudStorage/delete/{publicId}
Property	
GET	/api/Property
POST	/api/Property
GET	/api/Property/{id}
PUT	/api/Property/{id}
DELETE	/api/Property/{id}
User	
POST	/api/User
GET	/api/User
GET	/api/User/{id}
DELETE	/api/User/{id}
Report	
POST	/api/Report/{userId}/{format}

Libraries

Cloudinary	Cloud storage
EPPlus	Excel library
Syncfusion	Excel to PDF library
AutoMapper	Object to object mapping
Identity	Auth database
SkiaSharp	Render graphics

Program.cs

- Added CORS policy with origins
  - http://localhost:4200
  - https://property-management-application.netlify.app
- Added Render.com deployment port
- Added Swagger authorization header
- Added EPPlus 8.3 license (free)
- Added Syncfusion license
- Added DbContexts
- Added HttpClient
- Added Services, Repositories
- Added Automapper
- Added Cloudinary
- Added Identity with options
- Added JWT authentication

1. Auth

AuthController -> CreateToken()

- Create token takes identity user and roles
- Create claims list using email and roles
- Get secret JWT key
- Create credentials
- Create token using (issuer, audience, claims, expiry, credentials)

Register

- Create new identity user using username and email using user manager class
- If succeeded, add roles to identity user

Login

- Find identity user by email
- Check password
- If password matches, get user role
- Create token by passing this user and roles
- Return token as response

2. User

UserController -> SQLUserRepository

- CreateUser()
- GetAllUsers()
- GetUserById()
- DeleteUser()

3. Cloud storage

CloudStorageController -> CloudStorageService

- UploadFileAsync()
    - Get file, open read stream
    - Get file extension
    - Check if file is an image (.jpg, .jpeg, .png, .gif, .webp)
    - If image file, upload image file
    - If raw file, upload raw file
- UploadImageAsync()
    - Need publicId and URL in response, hence this endpoint is separate
    - Accept file, upload it
    - Return publicId and URL in response
- DownloadFileAsync()
    - Download file using publicId
    - It downloads as memory stream
    - Returns file bytes
- DeleteFileAsync()
    - If image file, delete image file
    - If raw file, delete raw file

4. Property

PropertyController -> PropertyRepository

- GetAllPropertiesAsync(string? filterOn = null, string? filterQuery = null, string? sortBy = null, bool isAscending = true, int pageNumber = 1, int pageSize = 10)
    - Make the result as AsQueryable()
    - Filter properties based on filter query
    - Sort properties based on sortBy query
    - Add pagination
- GetPropertyByIdAsync()
- CreatePropertyAsync()
- UpdatePropertyAsync()
- DeletePropertyAsync()

5. Report

ReportController -> ReportService

- Report endpoint takes userId and format
- HttpClientFactory is used to download images from web using URL

GeneratePropertyPortfolioReport()

- Download the template from Cloudinary
- Use the template stream to create new Excel package

CreateReport(package, userId)

- Declare SPs, sheet variables
- Hide raw sheets
- Call report repository
- Open SQL connection using connection string
- GetDataSet() -> get data with multiple tables
- GetDataTable() -> get data with single table
- SPs return data for sheets
- There are two methods to populate data
- PopulatePortfolioData()
    - Populate portfolio KPIs and chart data
- PopulatePropertyData()
    - Build new sheet for each new property
    - Load property data with images
- Once Excel report is prepared, get the new package bytes
- If PDF, convert the file to PDF, set filename and content type
- If Excel, set the same file bytes, set filename and content type
- Convert the file bytes into memory stream
- Convert it into file and upload this file to Cloudinary

PropertyManagement - Stored Procedures

SP syntax

```
CREATE PROCEDURE sp (@parameter)
AS
BEGIN
    …
END
```

dbo.sp_GetPortfolioData

- This SP generates portfolio level summary for a user including
- KPIs (Key Performance Indicators)
- Data for chart (top 3 properties + others)

dbo.sp_GetPropertyData

- This SP generates property level data in table

What is CTE (Common Table Expression)?

- CTE is like a temporary view created from a query which we can reuse immediately to get data in different ways
- First, we run a query and name its result, then use that result like a table for further queries

Syntax:

```
;WITH MyCTE AS
(
    SELECT …
)
SELECT * FROM CTE;
```

- Defined using WITH
- Used immediately after
- Improves readability, not storage
- CTE is used to simplify complex queries, to avoid repeating queries and to make queries easier to understand

How have you used CTE in your SP?

- SortedProperties
    - Calculate each property's contribution (in %) of total portfolio
    - Centralize this calculation once
- Top 3
    - Get top 3 highest value properties
- Others
    - Group remaining properties into 'Others'
- Without CTE:
    - Same logic will be repeated, difficult to read and nested subqueries
- With CTE:
    - Step by step logic, easy to understand

PropertyManagement.UI

- This is an Angular SPA used to manage users, properties and generate detailed report
- It consumes REST APIs from ASP.NET Core backend and uses JWT-based authentication

1. app.module.ts

- Root Angular module to register all feature modules, shared modules, HTTP client, global providers

2. app-routing.module.ts

- Defines main routing of the application

3. app.component.ts / app.component.html

- Root component that contains the main layout and router outlet

4. Feature modules

4.1. Auth

- Auth module handles login, logout, token storage and route protection
- Login
    - Login form
    - Form validation
    - Call AuthService
    - Save token
    - Navigate to dashboard
- Register
    - Register form
    - Form validation
    - Call AuthService
    - Register users
    - Navigate to login

4.2. Dashboard

- Dashboard – Users
- Show loading spinner
- Show empty state -> Create user
- Display user -> Create another user (both desktop and mobile views)
- For each user, actions
    - View properties
    - Delete user
    - Download report
- Backend sends a file, not a normal JSON
- Browser does not automatically download it
- So, we must manually download the file using JavaScript
- **Blob**: is a raw binary file that is returned from the backend
- **Content-Disposition**: When backend sends a file, it also sends headers which tells browser that this is a downloadable file
    - Content-Disposition: attachment
    - Filename = ‘Report.pdf’
    - Body -> file data (Blob)
    - Header -> filename + metadata

4.3. Property

- Property details
    - Show single property details
- Property form
    - If edit mode = true, update form, patchvalue
    - If edit mode = false, create form
- Property list
    - Show property list for a user

4.4. User

- User form
    - Create new user

4.5. Shared

- Header
    - Contains header, logout button
- Footer
    - Contains footer info

4.6. Core

Services

- auth.service.ts
- cloud.service.ts
- property.service.ts
- user.service.ts
- report.service.ts

Models

- auth.model.ts
- cloud-upload.model.ts
- property.model.ts
- user.model.ts

Guards

- auth.guard.ts
    - Auth guard protects private pages by checking if the user is logged in
    - This guard runs before a route opens
    - AuthService -> to check login
    - Router -> to redirect user
    - If user is logged in, allow page access
    - If not logged in, redirect to login page
    - Protect private routes
- public.guard.ts
    - Public guard prevents logged-in users from accessing public pages like login
    - If user is already logged in, redirect to dashboard, block login page
    - Protect public routes

Interceptor

- Interceptor runs automatically for every HTTP request and response
- A security gate through which all API calls pass
- Why do we need an interceptor?
    - Backend expects JWT token
    - We don’t want to manually add token in every API call
    - Interceptor attaches JWT token to every API request by getting token from local storage
    - If token exists, add it to request header
    - Otherwise send request as is
    - Without this backend will reject requests
    - Handle 401 unauthorized errors globally, if unauthorized, logs out user and redirects to login
- Without interceptor:
    - We had to manually add token in every service
    - Handle 401 errors in every component
- With interceptor:
    - Centralize auth handling, clean code, less bugs