

„Users” Applications

Created by: Kornidesz Máté

<https://github.com/Kornimate>

<https://kornimate.github.io/introduction/>

List of content

„Users” Applications	1
1. Task	3
1.1 System Requirements	3
2. User’s Documentation	3
2.1. Aspire Web App Host.....	3
2.2. API Swagger UI.....	4
2.3. WPF Application	4
2.3.1. Home Page	4
2.3.2. Create User Page	5
2.3.3. List Users Page.....	5
2.3.4. Delete User Page	6
2.3.5. Update User Page.....	7
2.3.6. Requests Page	7
3. Developer’s Documentation.....	9
3.1 Solution Structure.....	9
3.2. Use Cases.....	10
3.3. Components	10
3.3.1. The Web API	11
3.3.2. The WPF Client	13
3.4 Dependencies	14
4. Testing.....	15

1. Task

TodoApi.7z contains a small .NET web API

Your tasks are:

1. Rename TodoItem to User and expand the User model to include fields like Address (street, house number, zip, city, country).
2. Write a console application that will call this API, allowing you to retrieve a User by a given ID and perform all other CRUD operations.
3. Create two requests that, for example, call GetUser(Id) 1000 times: one that executes sequentially and another that runs in parallel.

1.1 System Requirements

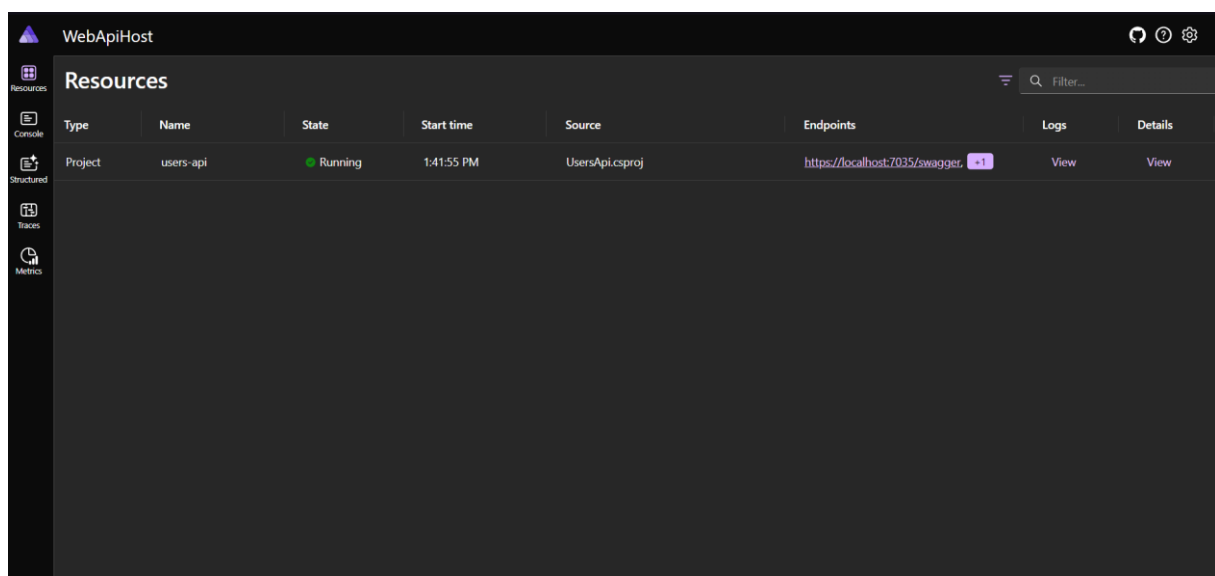
- Windows machine (Win 10 or later)
- 1GB of RAM
- .NET 8 runtime

2. User's Documentation

The application consists of a web API a WPF desktop application and an Aspire project for local monitoring of the web API. To start the application, it is recommended to have multiple start up projects configured in Visual Studio which are the following: The Aspire project (which starts the web API and the WPF application).

2.1. Aspire Web App Host

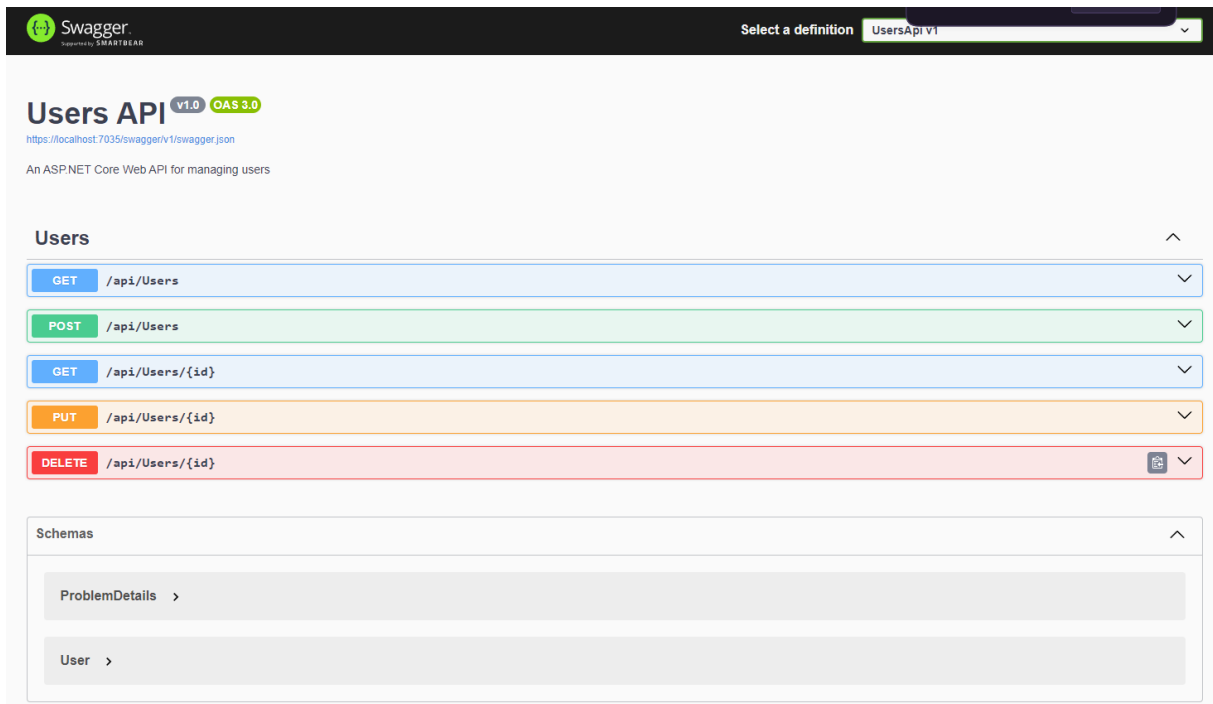
This application provides a UI for monitoring the registered services. The web API is registered so it will be monitored through Aspire's dashboard. It also provides navigation to the web API's UI.



1. picture: Aspire Dashboard

2.2. API Swagger UI

The API provides a swagger UI for interactive interaction with the API itself and it is the documentation for the API usage too as well as for the DTO present in it.



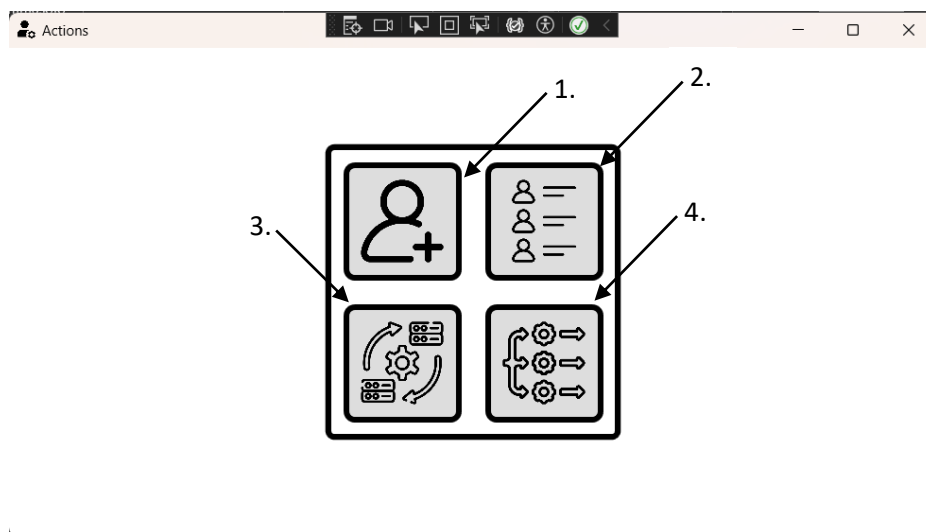
2. picture: Web API swagger UI

2.3. WPF Application

The WPF application is simulating the CRUD operations to the web API through a desktop UI.

2.3.1. Home Page

The home page contains 4 buttons. One for creating new user, one for listing existing users, one for simulating API calls synchronously, one for simulating API calls in a parallel way.



3. picture: WPF app home page

1. Button to open create user window
2. Button to open list users window
3. Button to start synchronous API calls simulation
4. Button to start parallel API calls simulation

2.3.2. Create User Page

This page enables the user to create new user and save it to the database through the API. This page has some restrictions as empty data is not allowed, house number and zip number fields only allow numbers. By clicking the add user page the app will post data to the endpoint and will notify the user about the success of the action.

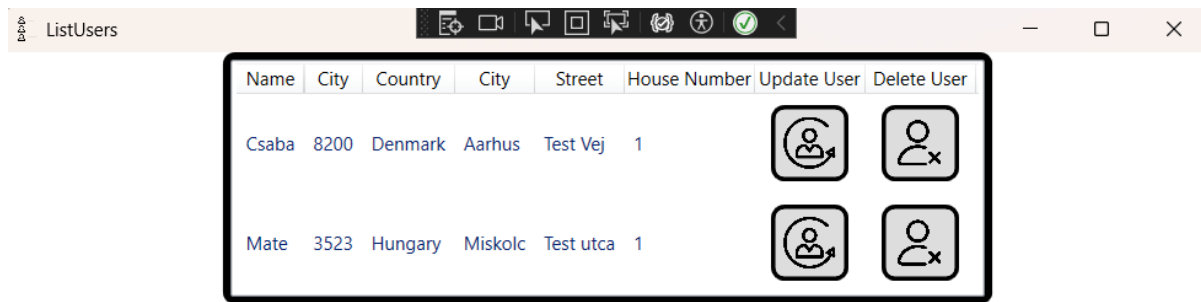


Name	<input type="text"/>
Zip Code	<input type="text"/>
Country	<input type="text"/>
City	<input type="text"/>
Street	<input type="text"/>
House Number	<input type="text"/>
<input type="button" value="Add New User"/>	

4. picture: Create user page

2.3.3. List Users Page

This page allows the user to see the state of the database behind the API. The users are listed and can be modified or deleted.



5. picture: List users page

2.3.4. Delete User Page

This page allows the user to delete an existing user from the database. If an error occurs the user is notified about it, otherwise about the success of the action. By pressing the „Delete User” button the user can start the action.

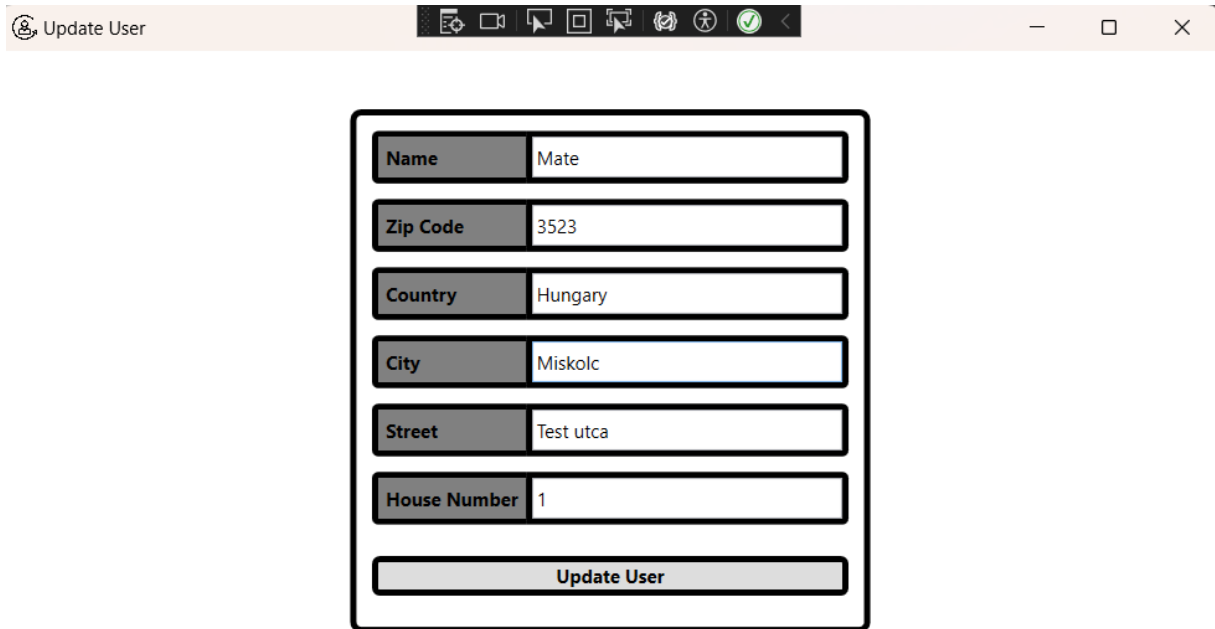
The screenshot shows a web browser window titled "Delete User". Inside, there is a form with the following fields and a button:

Name	Mate
Zip Code	3523
Country	Hungary
City	Miskolc
Street	Test utca
House Number	1
Delete User	

6. picture: Delete user page

2.3.5. Update User Page

This page allows the user to update the details of an existing user and then save it to the database through the API. By pressing the „Update User” button the user can start the action. After the action a message box notifies the user about the success of the action.



Name	Mate
Zip Code	3523
Country	Hungary
City	Miskolc
Street	Test utca
House Number	1
Update User	

7. picture: Update user page

2.3.6. Requests Page

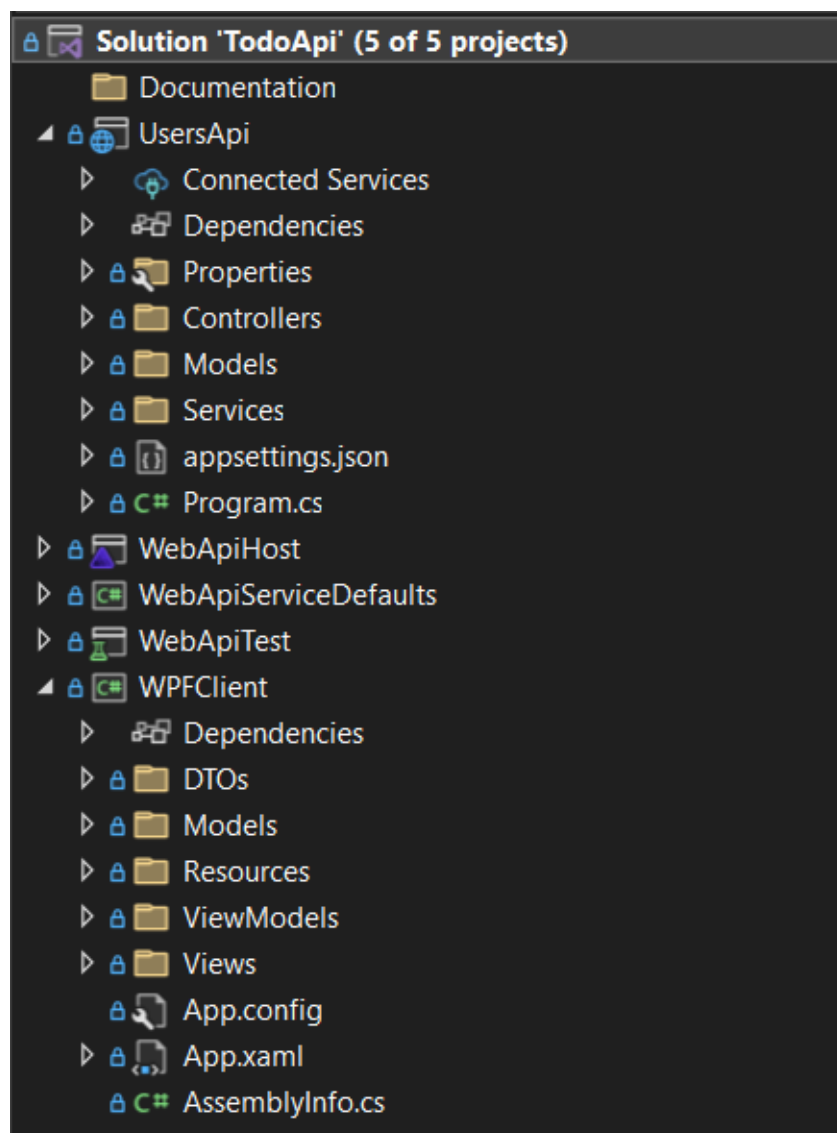
This page is to simulate 1000 API calls to the `/api/Users/1` endpoint. The requests can be done synchronously and parallel too. The page displays the logs from the actions made by the application and a little benchmark about the whole action of 1000 API calls. The parallel way creates as many windows as parallel the action was and display the same thing for each parallel component.

3. Developer's Documentation

3.1 Solution Structure

The solution consists of 5 projects:

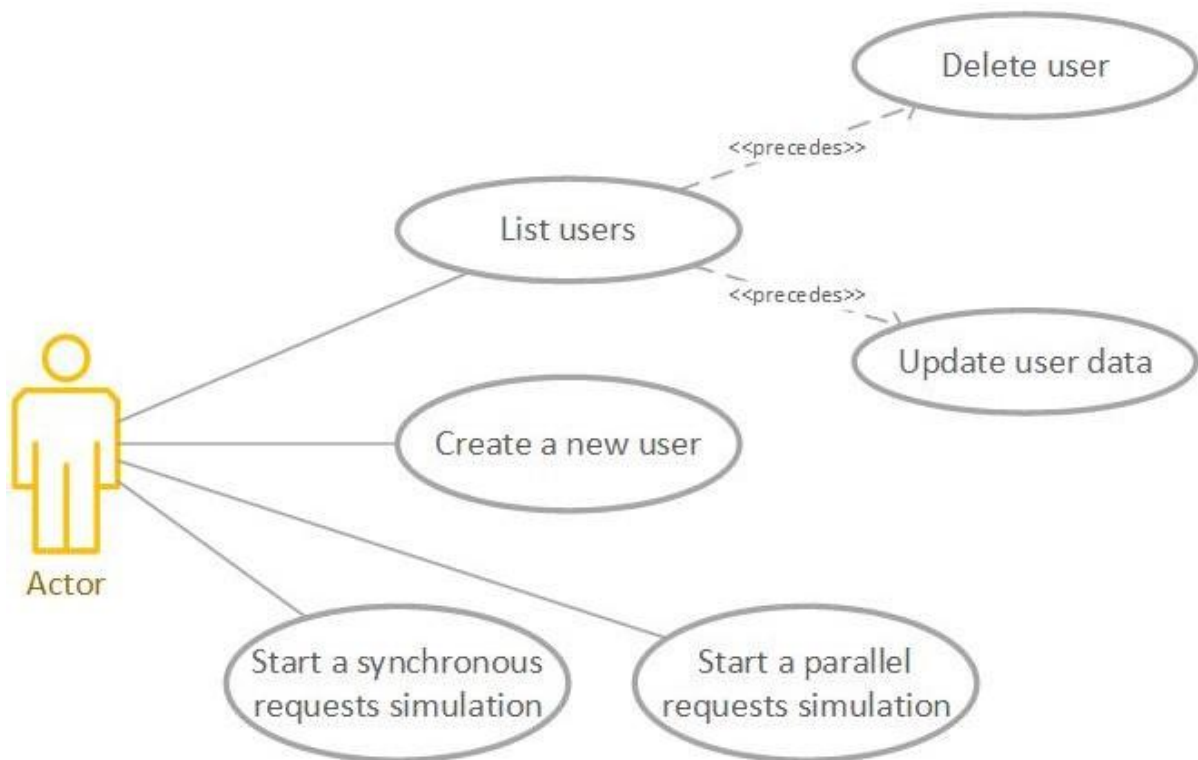
- A web API for handling HTTP requests
- An Aspire App Host project for API monitoring
- A Aspire Service Defaults projects for monitoring web API (extensions for OpenTelemetry and health checks)
- A WPF application to simulate the client
- An MSTest project for automatic testing of the API



10. picture: Solution structure

3.2. Use Cases

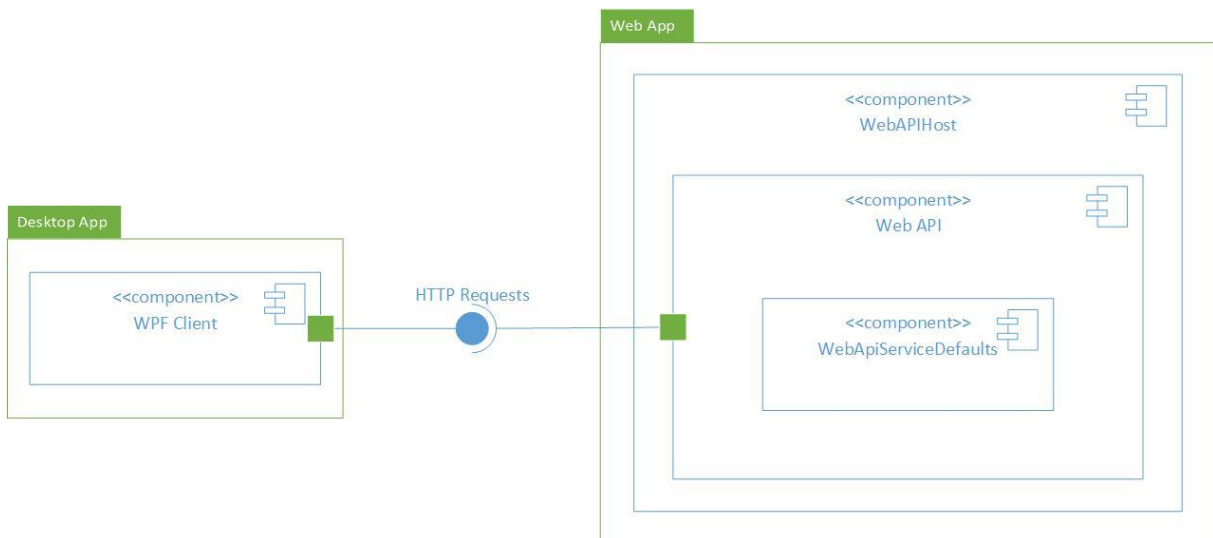
The following use case diagram show the basic interactions with the application.



11. picture: Use case diagram of client

3.3. Components

The following diagram shows the components of the application regarding the projects inside the solution.



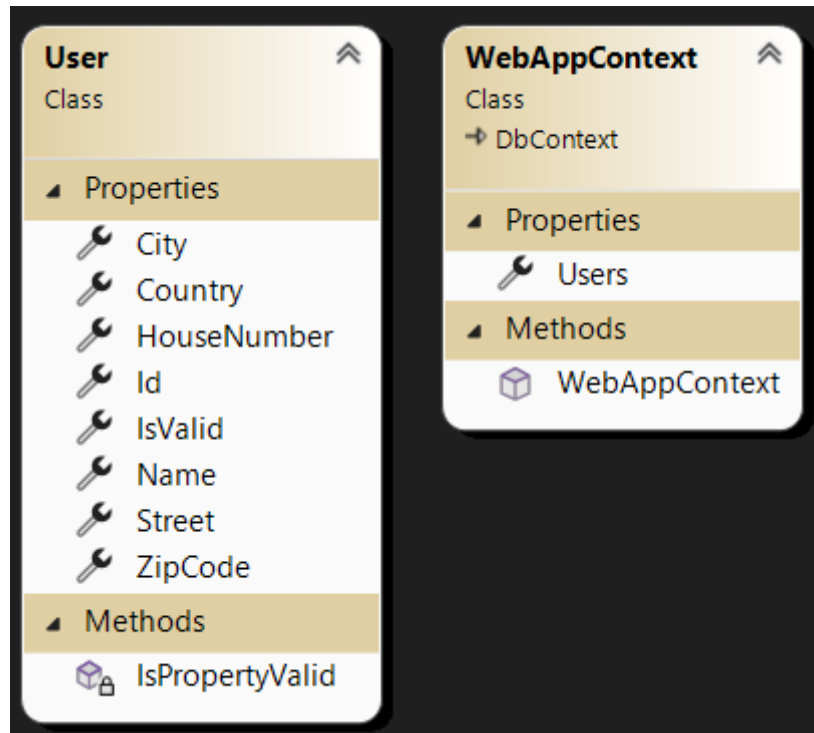
12. picture: Component diagram

3.3.1. The Web API

The web API is responsible to handle requests coming from the user. This is a REST API. The API's structure follows the MVC architecture except the fact that there are no views.

3.3.1.1. The Model

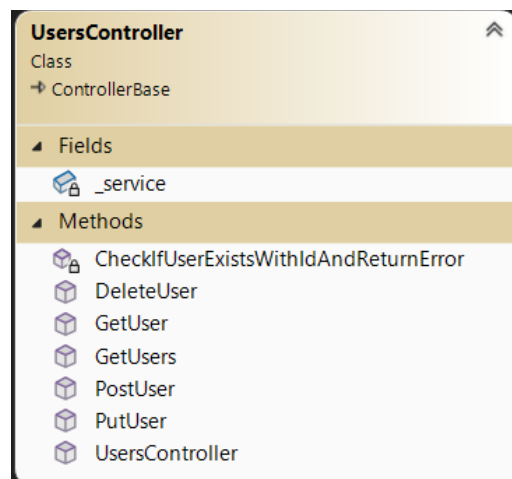
The models consist of a User class as the base of EF's code-first database generation approach. There is also a WebApplicationContext class for configuring EF and setting the classes as base of tables in the database.



13. picture: Class diagrams of models

3.3.1.2. The Controllers

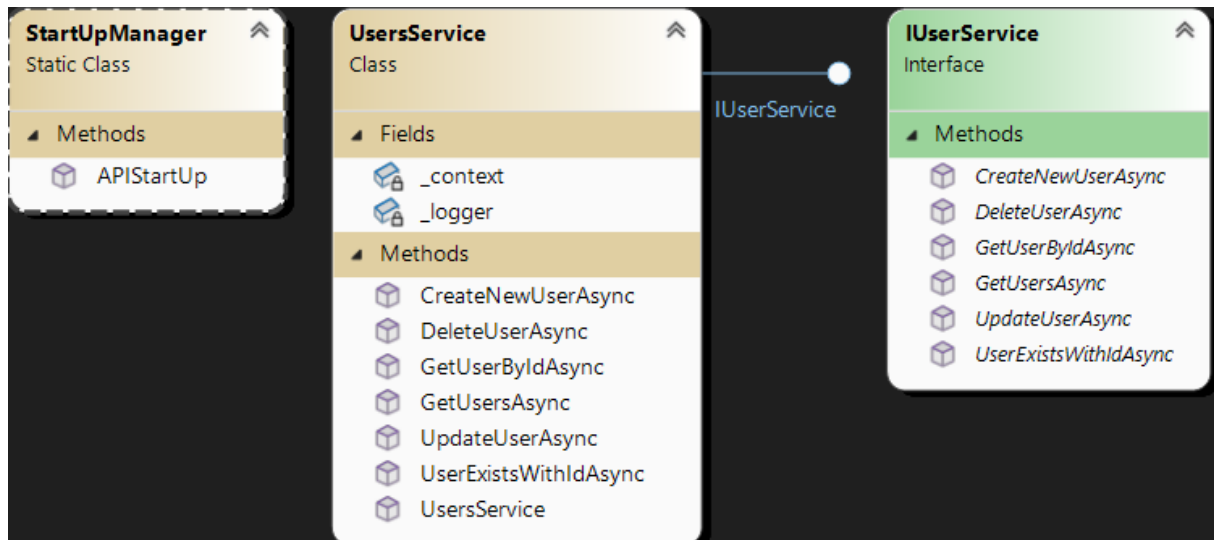
The controllers handle the requests. The app has one controller responsible for users handling.



14. picture: UsersController class diagram

3.3.1.3. The Services

An extra service layer is responsible for interaction with the database, this class is injected into the controller to be able to make database actions. The IUserService and UserService classes are responsible for these actions and StartUpManager class is responsible for creating the database and seed with some example data.

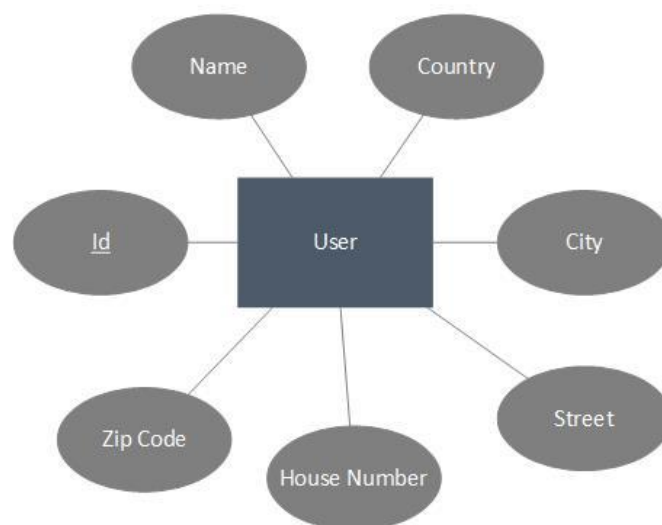


15. picture: Class diagrams of services

3.3.1.4. The Database

The database scheme consists of a single table named „Users“. It was created by Entity Framework on a code-first approach.

Web API In Memory Database scheme



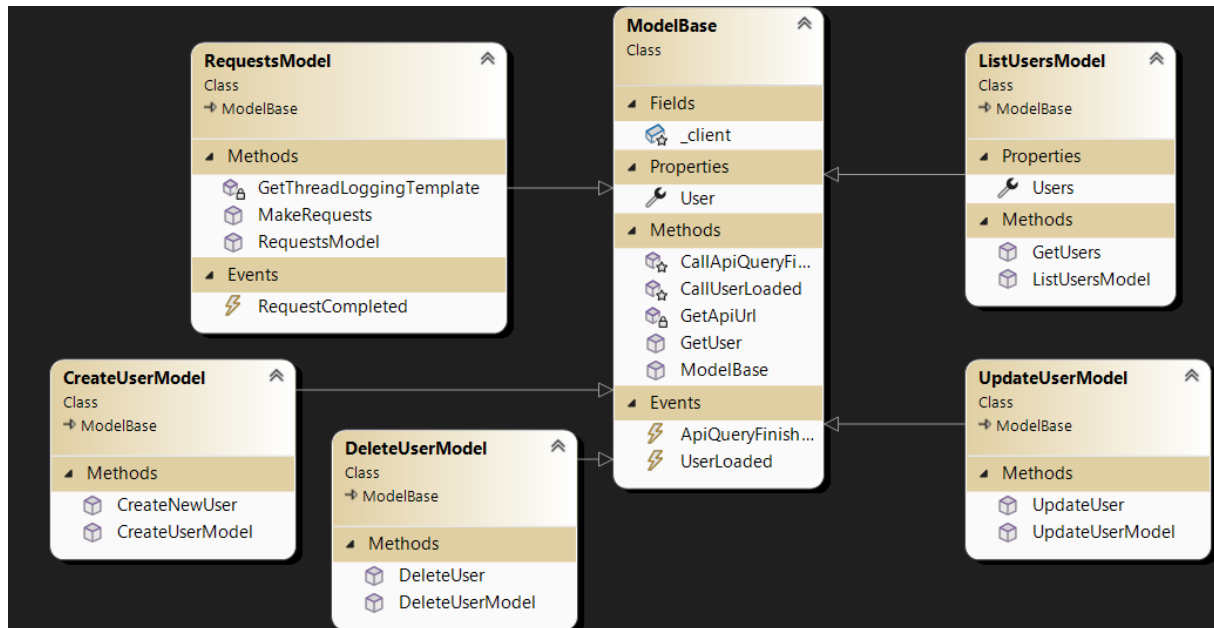
16. picture: Database scheme

3.3.2. The WPF Client

The WPF client is a desktop application for interacting with the web API. It is made in an MVVM architecture. The application uses an event driven approach.

3.3.2.1. The Models

Every page has its own model respectfully to the page's name. The models usually interact with the web API through a HttpClient object. Every model inherits from a ModelBase class for common methods, properties and fields.



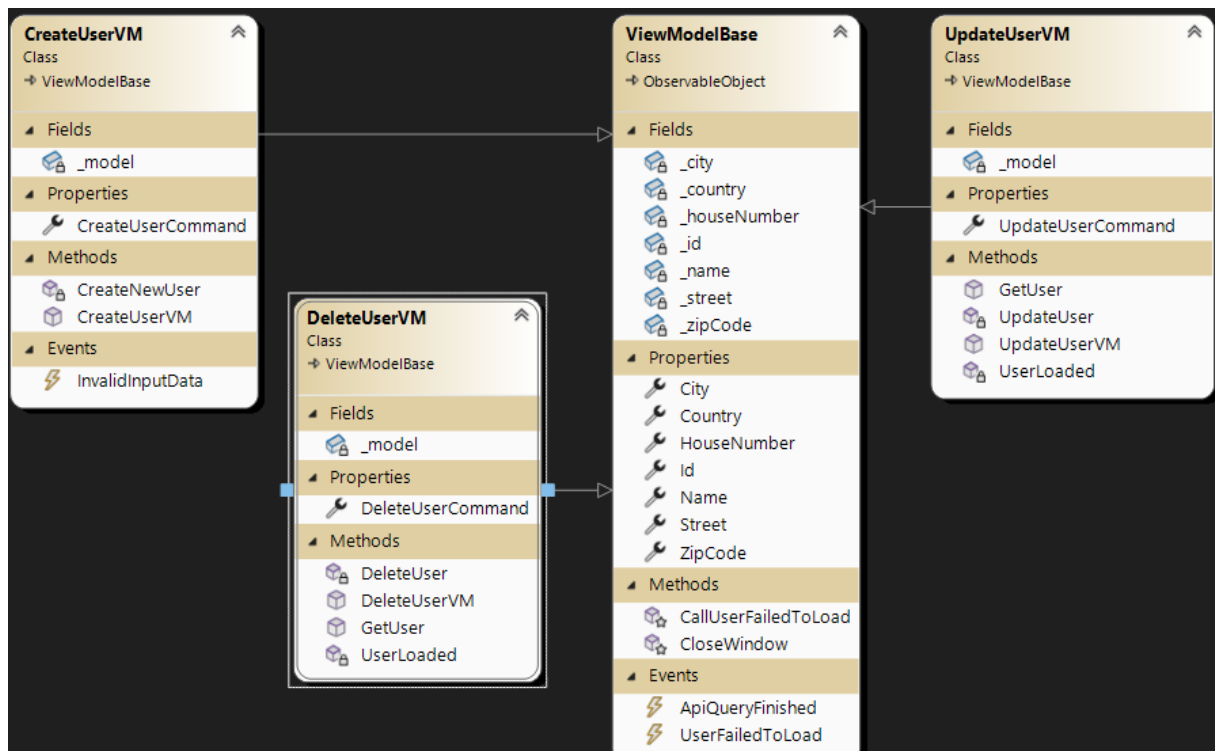
17. picture: Class diagrams of models

3.3.2.2. The Views

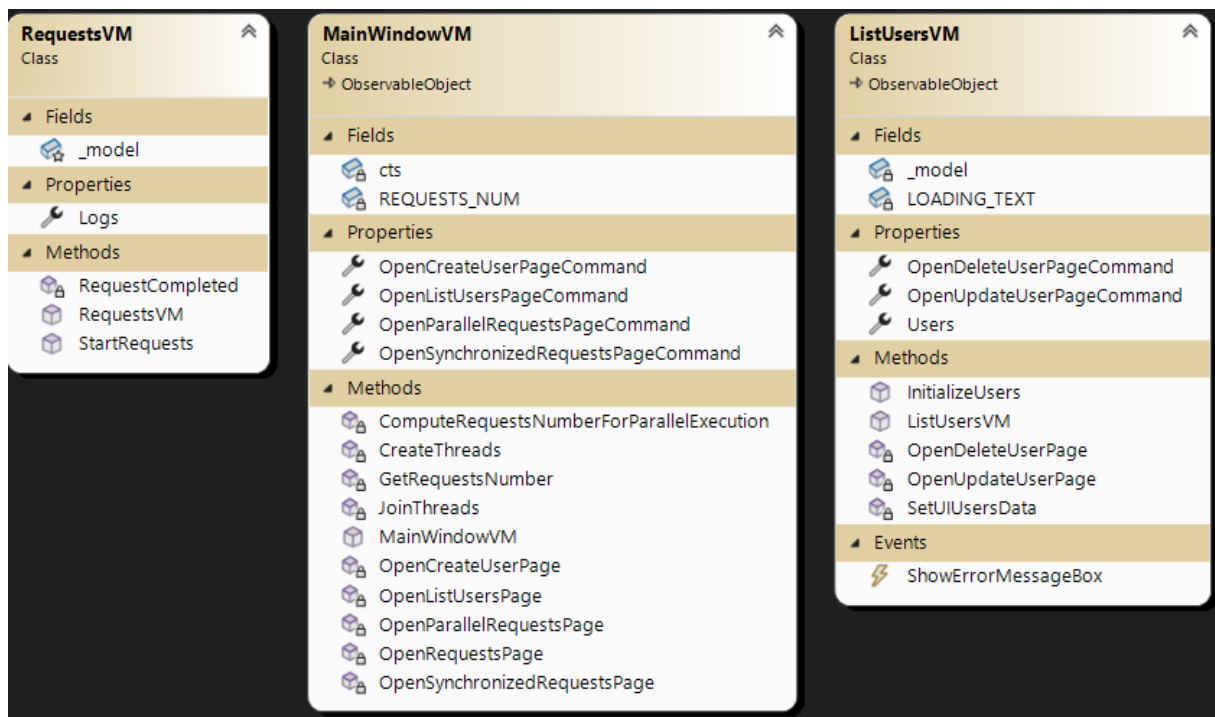
Most of the API interactions have their own window to carry actions out. These contains the XAML code to show elements and controls to the screen and the code behind part sets the viewmodels as data context for further actions.

3.3.2.3. The ViewModels

The viewmodels are responsible for showing data on the UI and managing interactions from user. This is done through data binding. There is one viewmodel for every page and some viewmodels share common features, there are from the ViewModelBase class.



18. picture: Class diagrams of viewmodels I.



19. picture: Class diagrams of viewmodels II.

3.4 Dependencies

The following dependencies are present in the project.

- CommunityToolkit.Mvvm@8.3.2

- Microsoft.AspNetCore.OpenApi@8.0.10
- Microsoft.EntityFrameworkCore.InMemory@8.0.10
- Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore@8.0.10
- Microsoft.EntityFrameworkCore@8.0.10
- Microsoft.EntityFrameworkCore.Design@8.0.10
- Microsoft.EntityFrameworkCore.Proxies@8.0.10
- Microsoft.EntityFrameworkCore.SqlServer@8.0.10
- Microsoft.EntityFrameworkCore.Tools@8.0.10
- Microsoft.VisualStudio.Web.CodeGeneration.Design@8.0.6
- Swashbuckle.AspNetCore@6.9.0

4. Testing

The solution contains an automatic test project (WebApiTest, MSTest) for the web API. The project contains 30 unit tests for service methods and controller actions.

The WPF client is manually tested with the following user stories.

Given	When	Then
The user is on the home page	Clicks the create user button	Create user page opens
The user is on the home page	Clicks the list users button	List users page opens
The user is on the home page	Clicks the synchronous requests button	Synchronous requests simulation starts
The user is on the home page	Clicks the parallel requests button	Parallel requests simulation starts
The user is on the create user page	Clicks the create user button, fields contain valid data	The app sends http request and displays result
The user is on the create user page	Clicks the create user button, fields contain not valid data	The app displays error message
The user is on the list users page	The page opens	Users are listed
The user is on the list users page	Clicks the update user button	The update user page opens with the specific user
The user is on the list users page	Clicks the delete user button	The delete user page opens with the specific user
The user is on the update user page	Clicks the update user button, fields contain valid data	The app sends http request and displays result
The user is on the update user page	Clicks the update user button, fields contain not valid data	The app displays error message
The user is on the delete user page	Clicks the delete user button	The app sends http request and displays result