Developer Guide
Computing For Good Spring 2022
Homelessness Team 1: Financial Achievement Club
Team Members: Gerald Meixiong, Roberto Ochoa

Frontend

Application Overview

The frontend is a React application that relies heavily on Material UI components and templates. API requests to the server are made via the standard fetch API. For local development, the frontend/README.me file can be referenced for the recommended way to install dependencies and start the server. A .env file is also included in the repository and can be used to reference either a local API server or the deployed API server for the frontend application.

Code Structure

All relevant source code for the frontend lives within the frontend/src/ directory. The directory has a structure according to the following diagram.

- frontend/src
    - components
        - Houses the various components and tables used throughout the application. The building blocks for individual pages and views.
    - layouts
        - Layouts determine the overall view of a page. For example, the Dashboard Layout dictates that have a sidebar on the left and a navigation bar at the top. Layouts are combined with pages to form a page view in routes.js.
    - pages
        - The main content for any one page that is not dictated by the layout. For example for the Users page, the main content is the Users table which is defined within pages/Users.js.
    - providers
        - Context providers for the application to be used in App.js. Currently, there is only one context provider, the AuthenticatedUserProvider. This provides all users of the context access to the currently logged in user of the application; this is used throughout the application as the logged in user's access token is typically sent as an authorization token for API requests to the server.
    - routes.js

        - Enumerates the various routes for the application. Pages and layouts are mapped to specific routes which can be static or dynamic.
    - App.js
        - The main component for the application. Context providers should be added in this file, but this file can largely remain untouched.

Example Walkthrough

Imagine the backend API server exposes a new endpoint that returns analytics and reporting data such as how many unique users have used the application in the last month, the cumulative amount of money saved, and the total number of courses completed by all users.

To support this, we would first need to decide whether we want to display this information on a new route. Perhaps this information is relevant on the home page, in which case a new route is not needed; if we decide we want a new route, then routes.js would need to be updated with the new path and element. To start, we can create a new path and use an existing page element as we have not created the new page or other components to be used.

Next, we need to add a page in the pages/ directory which houses the main content we want to show. This page will likely reference several components, one for each distinct piece of information that we want to display. Many areas of the application use tables, but for this example, we probably want to build off an existing page that does not use tables such as the “Course” page. This page demonstrates conditional rendering of components based on a user’s role which may come in handy.

Pages directly reference components. Standard React components will suffice. We will likely create a UniqueUsers component, a TotalSaved component, and TotalCourses component. Whether to create separate components or a single component that houses all this data is up to your discretion. It may be easier to expand and configure new functionality if the components are separate.

Once the components have been written and are referenced by the new page we added, the route for the page can be tested by running the server locally and visiting the route that you defined. On Google Chrome, the developer console is extremely useful for debugging and compilation errors, warnings, and even network requests.

Backend

Programming language: C#

Framework: .Net 6

Solution type: RESTful API

Project structure

The project in general follows a hexagonal or clean architecture and the backend arrangement is compliant with it.

This is the current backend structure:

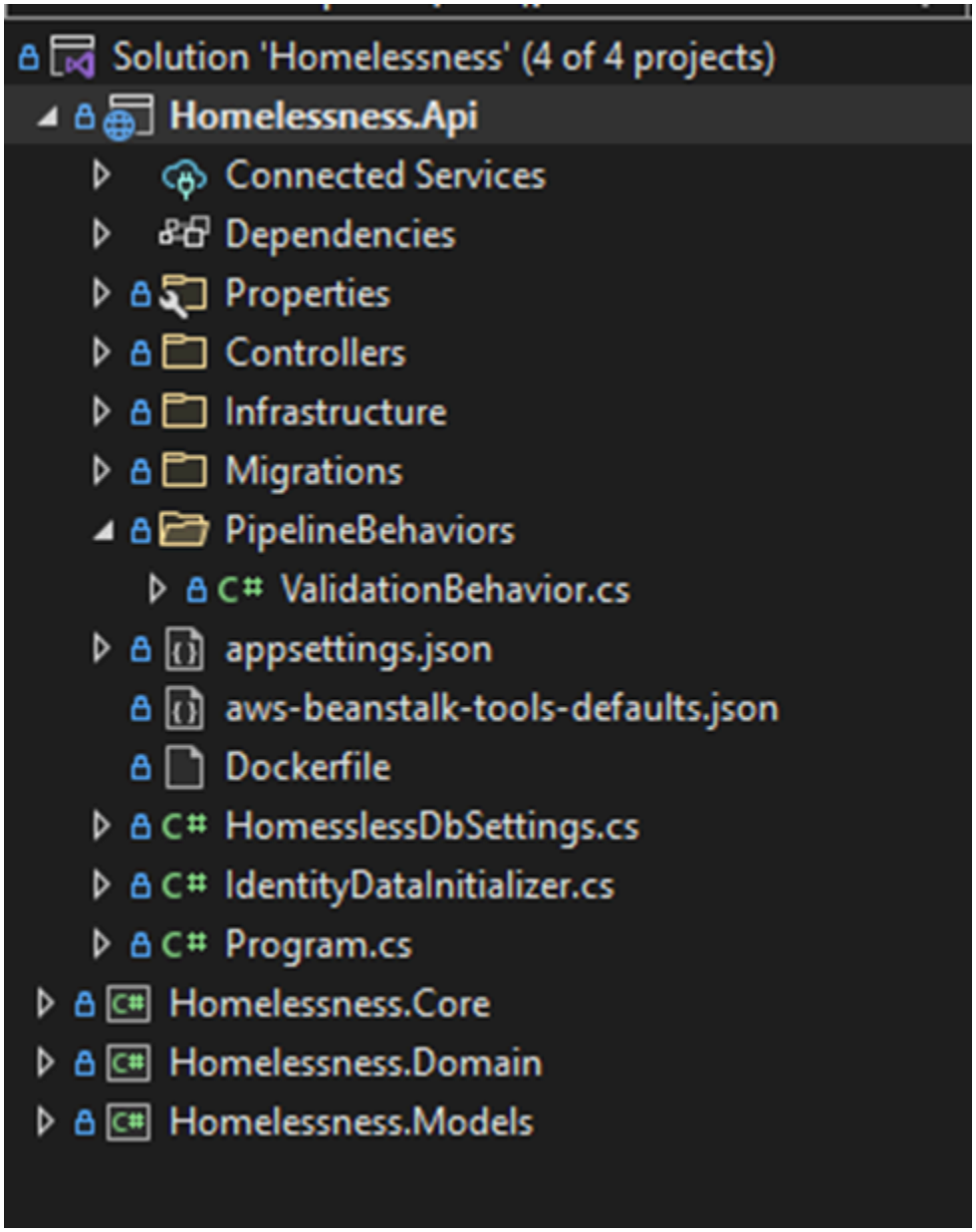

Figure 1 – Backend solution structure.

Homelessness.Api is the startup project and is the one consumer applications communicate with as it is the one that exposes the client facing API Url. The ***Controllers*** directory contains all the controllers with the different http endpoints.

Since the implementation uses Entity Framework as the ORM, EF Repository pattern is used to differentiate the different entity repositories to make it easier for developers to code, maintain and expand functionalities. The ***Infrastructure*** folder contains all the repositories and the DB context class.

Migrations holds all the EF code-first migrations and the context model snapshot and the ***PipelineBehaviors*** directory contains the validation behavior class that is executed generically with Fluent Validation for every validation class.

Homelessness.Core is where the business rules are applied. This project contains the handlers, services, custom exception implementations as well as validation rules. The ***Services*** folder has the services use for authentication and token related actions. ***Exceptions*** is where the custom exceptions that are used in all the request handlers are declared. ***Helpers*** contain only a validation helper class, whose methods are used everywhere. ***Interfaces*** contains all the interfaces relates to the repositories or the business logic ones.

The most important directories in this project are ***Queries***, ***Commands***, ***Validators*** and ***Handlers***. The first two contain the queries and the commands sent from the frontend via http requests. The validation classes are inside ***Validators*** and all the query and command handlers are inside ***Handlers***.

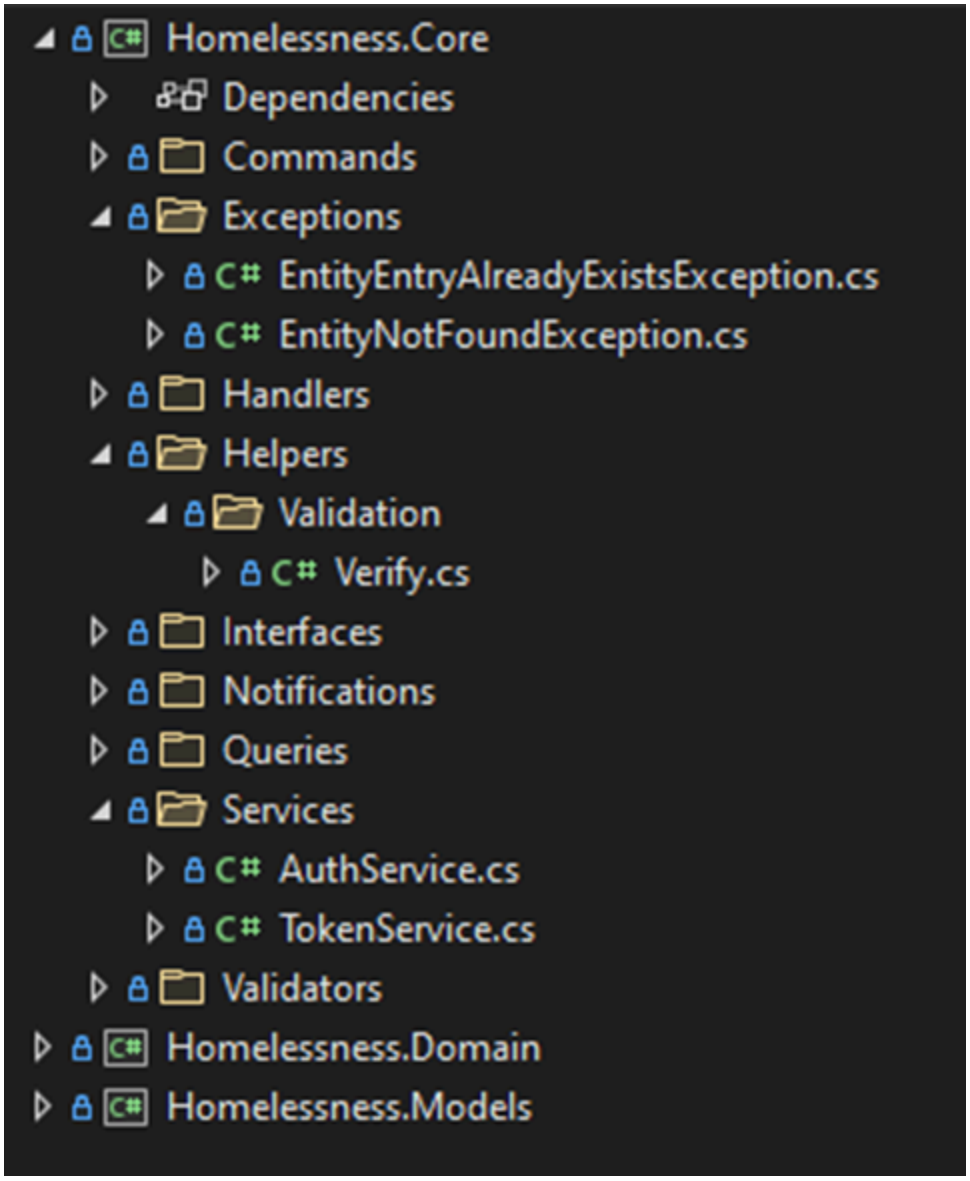

Figure 2 – Homelessness.Core project.

Homelessness.Domain and **Homelessness.Models** are projects that don't contain any business logic. They just have the entity classes that are needed for the ORM to map to tables on the DB and models that the backend maps to when returning data to the frontend application. **Homelessness.Models** also contains enums and custom requests and responses.

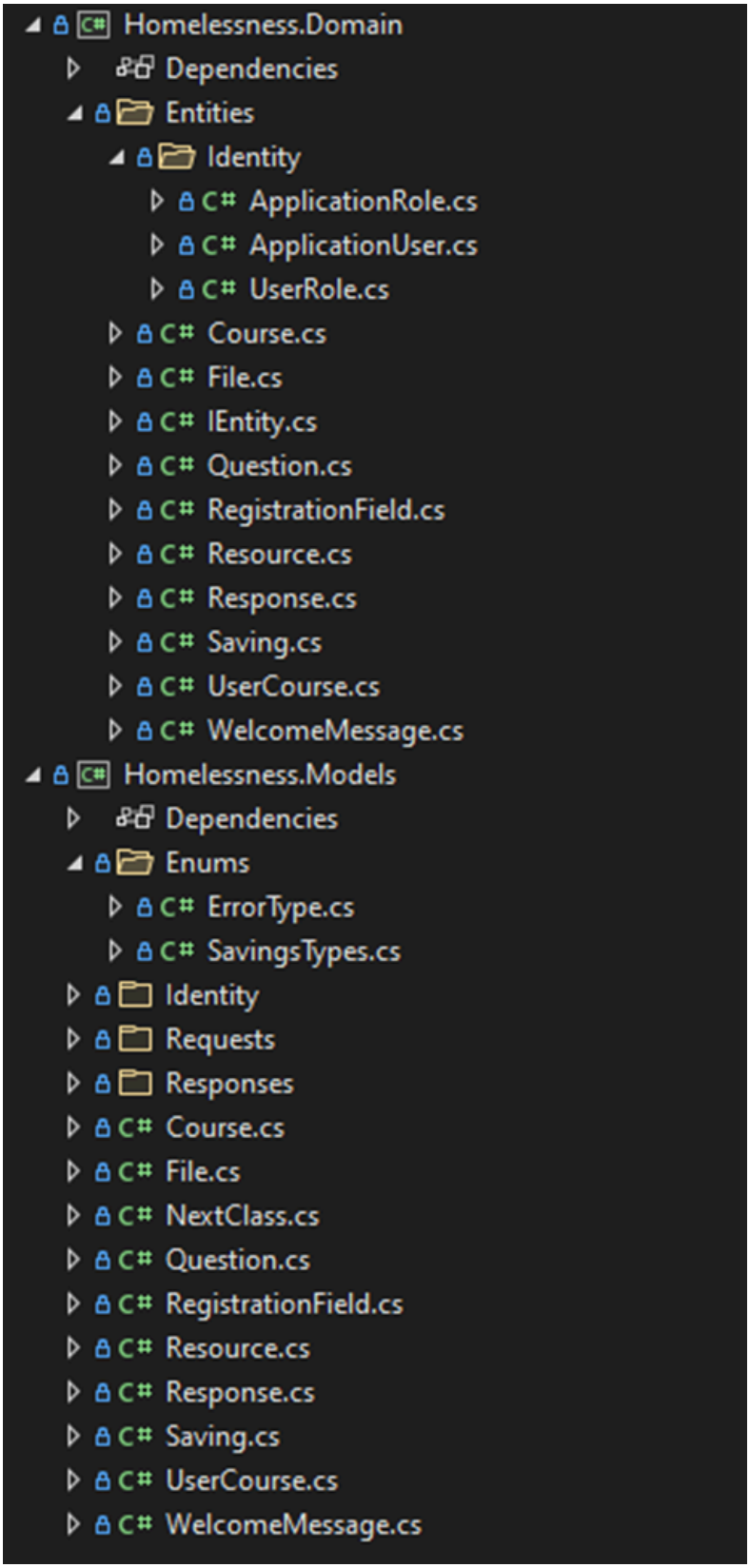

Figure 3 – Homelessness.Domain and Homelessness.Models projects.

Steps to run locally

1. Install PostgreSQL locally. Make sure the username and password locally is *"postgres"*
2. Open the backend .Net solution
3. Restore nuget packages
4. Make sure the startup project is Homelessness.Api
5. Run (this will automatically create the database, the schema, and seed the predefined roles and user)

Resources to learn the language and technology used

C#: https://dotnet.microsoft.com/en-us/learn/csharp

.Net API: https://docs.microsoft.com/en-us/learn/modules/build-web-api-aspnet-core/

Entity Framework (EF): https://docs.microsoft.com/en-us/ef/

MediatR:

https://github.com/jbogard/MediatR

https://code-maze.com/cqrs-mediatr-in-aspnet-core/

References

https://github.com/minimal-ui-kit/material-kit-react
https://docs.microsoft.com/en-us/learn/dotnet/