AngNetUser Documentation

By: Hyuk Jin Kwon

Email: [hyukjkwon@gmail.com](mailto:hyukjkwon@gmail.com)

GitHub: <https://github.com/Hkwon18/AngNetUser>

# Notes

1. DO NOT run the script to generate the database and table! The script was only included for review purposes. Creating a database manually will conflict with Entity Core’s database mapping, and cause fatal errors. Please run the application without a pre-existing database to allow the Entity Core ORM mapping. [[1] [2]](#_References)
2. ASP.NET Core no longer stores database connection strings in “*web.config”*. Although I have included the default local connection string in “*web.config*”, this application will fetch the connection string from “*appsettings.json*”. [[3]](#_References)

# Introduction

The purpose of this document is to briefly explain the techniques, tools, and design of AngNetUser.

With the use of Microsoft ASP.Net Core and Angular 2, AngNetUser is a simple web project which stores basic user information in a local database backend. The project’s data layer and models take advantage of the Entity Framework Core ORM tool, and the UI layer will utilize basic Twitter bootstrap css.

The application roughly follows a three-tiered architecture, and separates the presentation layer, web API layer, and data layer. However, the traditional roles of the business layer are handled by both the data layer and web API layer. As such, the data and application layers can be considered sub-layers of the business layer.

User input/output is communicated via the web application, and then communicated via the web API layer to the server by HTTP requests. Responses are resolved from the database as JSON objects and communicated back the client-side service and presented back to the user in the web application.

A simple outline of the various layers of AngNetUser is shown in the following diagram.

## Architecture Diagram

**Service/Web API Layer**

* Http requests for client and server

**Dbo.Users**

[Id] INT PRIMARY KEY NOT NULL,

[Email] NVARCHAR(50),

[FirstName] NVARCHAR(50) NOT NULL,

[Joined] date NOT NULL,

[LastName] NVARCHAR(50) NOT NULL,

[UserName] NVARCHAR(50) NOT NULL

**Database**

* SQL Tables generated by Entity Core

**Server Side**

**Client Side**

**Business & Data Layer**

* Models to generate tables via Entity Core Framework
* ASP.NET Core Controllers
  + CRUD Operations & Server-side Validations
* DBContext (Data layer)
* Web API Layer

**Entity Core Framework**

**UI Layer**

* Angular 2 Javascript UI
* Client-side models
* Client-side validation

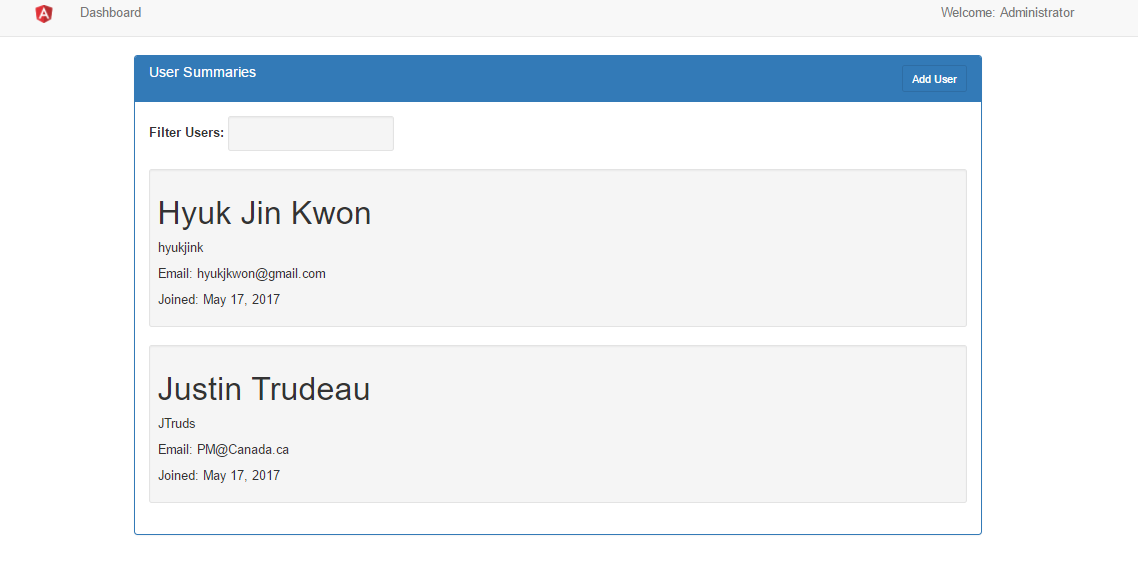
**Browser**

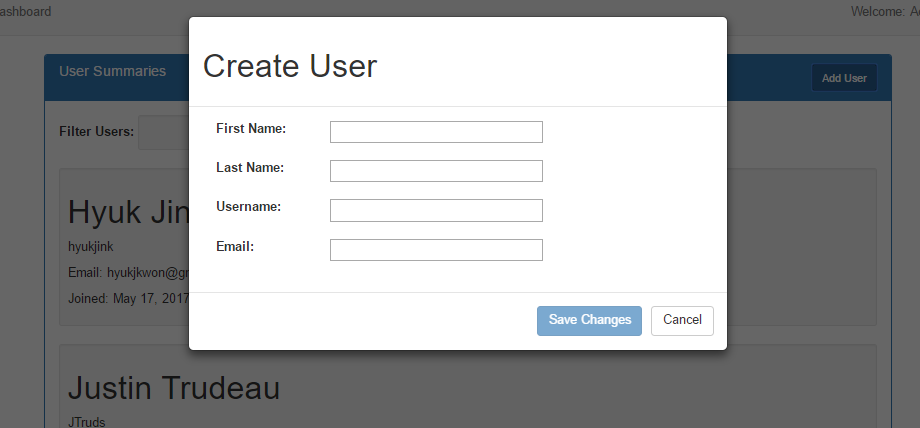
# UI Layer

The UI layer is written using Javascript, Angular 2, and twitter bootstrap and generates a single-page application which provides a table showing users currently in the database. Rows in the database are dynamically shown in the table. Client-side models are created in Angular 2 to reflect data coming from the server.

Users can be searched and filtered by first name, last name, email, and username at the top of the table. In the right corner of the table header, new users can be created. Clicking the “Add User” button will show a modal view which allows user information to be input and added to the database. Only valid user information is allowed to be saved.

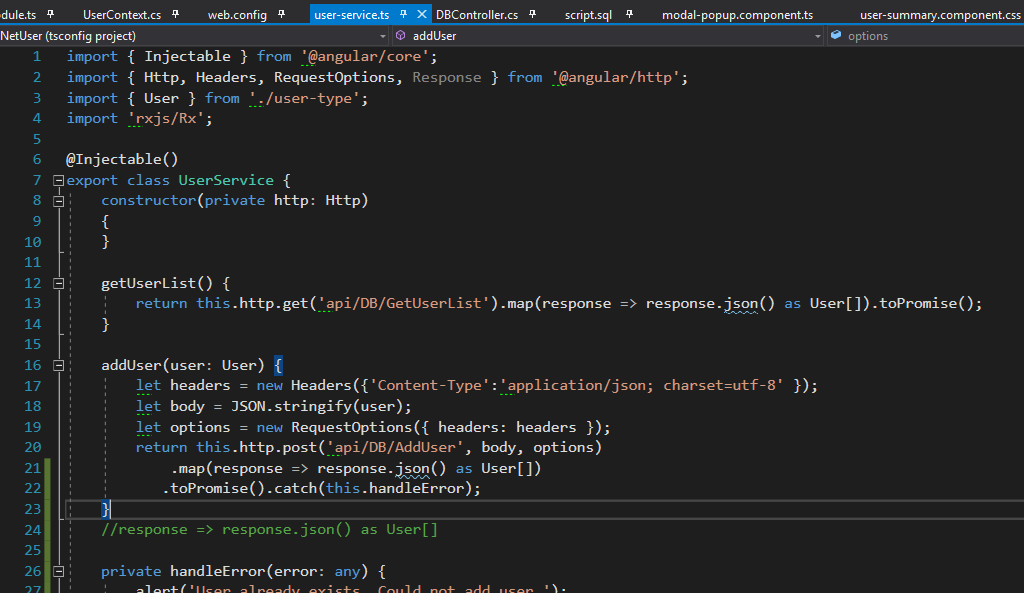
Client-side validation is implemented using aspects of Angular 2. For example, email validation is done using Angular 2 patterns. Attempts to add a duplicate user will fail, and an alert message box will be prompted.





# Service/Web API Layer

The Web API layer is a simple service including HTTP requests located in the “*user-service.ts*” file in the solution under *“../app/components/shared*”. This file is an Angular 2 file and resides on the client-side to communicate with the server. Methods in this service include getting and posting to the server, and handling responses that come back to the UI layer. Data is passed by JSON serialization.



# Business & Data Layer

After an HTTP request is received from the client, CRUD calls can be made to the database. Model classes designed in C# are mapped to the SQL database automatically by Entity Core.

Inside the “*UserContext.cs*” file located under “../Context/UserContext.cs” the settings which dictate the ORM mapping to the database are found. Important database settings include a pointer to the database connection string, and some specific definitions on the SQL database to be made with the Entity Core Framework.

Inside the “*DBController.cs”* file located under “../*Controllers/DBController.cs”*, HTTP requests are handled and communicated to the database. The CRUD methods which handle the HTTP requests also implement server-side validation before information is passed to or from the database.

# Database

As mentioned, the database is automatically generated using the Entity Core Framework. As such, SQL scripts are not necessary or advisable to generate the database or the tables within the database.

**In ASP.NET Core and Entity Core, migrations with existing databases are not feature complete** [**[1] [2]**](#_References)

As such, creating a database for this application manually without the Entity Core Framework will cause a fatal error. An SQL script creating the database and tables named “*script.sql”* can be found in the main directory of the solution, but it is highly advised that it is not used, as it conflicts with Entity Core.

As mentioned, the specifics of the generated database are designated in “*UserContext.cs”*. In ASP.NET Core, connection strings are not contained in the “*web.config”* file, and are instead stored in the “*appsettings.json”* file [[3].](#_References) Both files can be found under the main directory of the solution. However, the connection string has been included in webconfig for reference.

Table indexes are also specified in “*UserContext.cs”*. Non-clustered indexes for columns Id, FirstName, LastName, and Email are created by Entity Core.

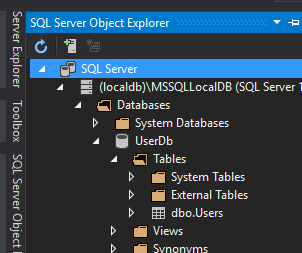


Table created by Entity Core

# References

[1]: <https://cmatskas.com/ef-core-migrations-with-existing-database-schema-and-data/>

[2]: <https://forums.asp.net/t/2106054.aspx?Using+EntityFrameworkCore+with+an+existing+database>

[3]: <http://benjii.me/2016/04/new-configuration-appsettings-mvc6-web-config-gone/>