**ASP.NET Core 2.0 + Angular 2/4 - User Registration and Login Tutorial & Example**

 07 June 2017

Oct 14, 2017 - Updated to **ASP.NET Core 2.0**and **Angular 4.4.5**

Late last year I posted a tutorial on my personal web development blog showing how to build [user registration and login functionlity in Angular 2/4](http://jasonwatmore.com/post/2016/09/29/angular-2-user-registration-and-login-example-tutorial) using a mock backend, it includes the boilerplate front end code for a secure web application that I developed for a law firm in Sydney.

In this post I've expanded on that tutorial with the addition of a real backend Web API developed with ASP.NET Core and C#, it's based on the code from the web api in the Sydney law firm application, making it a full stack ASP.NET Core + Angular 2/4 application (ANCA Stack??).

The project was originally developed as part of a larger business application that allows Sydney lawyers to access and manage sensitive data relating to their law firm, the example is a pared down boilerplate version that shows how to integrate Angular 2/4 with a .NET Web API end-to-end in a real world application, including JWT authentication and data access with Entity Framework Core.

**In-memory database with Entity Framework Core**

For easy testing the example application is configured to use the [Entity Framework Core InMemory](https://docs.microsoft.com/en-us/ef/core/providers/in-memory/) provider that allows EF Core to be used with an in-memory database, this can be easily switched out to a real db provider when working with a real database.

**Download the project code**

The project code is available on GitHub at <https://github.com/cornflourblue/aspnet-core-angular2-registration-login-example>.

Update History:

* Oct 23, 2017 - For the same example built with a React + Redux front-end go to [React + Redux with ASP.NET Core 2.0 - Login & Registration Tutorial & Example](https://www.pointblankdevelopment.com.au/blog/124/react-redux-with-aspnet-core-20-login-registration-tutorial-example)
* Oct 14, 2017 - Updated to **ASP.NET Core 2.0**and **Angular 4.4.5**
* June 07, 2017 - Built with **ASP.NET Core 1.1.2**and **Angular 4.1.3 -**to access the code for this version go to <https://github.com/cornflourblue/aspnet-core-angular2-registration-login-example/releases/tag/v1.0>

**Running the .NET Core with Angular 2/4 User Registration & Login Example Locally**

To develop and run ASP.NET Core applications locally, download and install the following:

* [.NET Core SDK](https://www.microsoft.com/net/download/core) - includes the .NET Core runtime and command line tools
* [Visual Studio Code](https://code.visualstudio.com/) - code editor that runs on Windows, Mac and Linux
* [C# extension](https://marketplace.visualstudio.com/items?itemName=ms-vscode.csharp) for Visual Studio Code - adds support to VS Code for developing .NET Core applications

To develop and run Angular 2/4 applications locally, download and install the following:

* [NodeJS](https://nodejs.org/) - javascript runtime environment that includes the node package manager (npm)

The sample project contains two applications, one for the Angular client and another for the .NET Core server api, below are the steps to get each of them setup and running:

**Running the ASP.NET Core Web API**

1. Download the project code from the GitHub link above
2. Open the project root folder in Visual Studio Code
3. Start the application by pressing F5 or by selecting Debug -> Start Debugging from the top menu in VS Code, this runs the web api at http://localhost:5000

The web api can also be started directly from the command line by running dotnet run from the /server folder of the project.

**Running the Angular 2/4 Client**

1. Open a command line / terminal window and navigate to the "/client" folder below the project root folder
2. Run npm install to install all required npm packages that are defined in the package.json file
3. Run npm start to start the client, a browser window should automatically open to the application at http://localhost:3000

**ASP.NET Core with Angular 2/4 Project Structure**

Click any of the below links to jump down to a description of each file along with its code:

* [server](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#server-folder)
  + Controllers
    - [UsersController.cs](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#users-controller-cs)
  + Dtos
    - [UserDto.cs](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#user-dto-cs)
  + Entities
    - [User.cs](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#user-cs)
  + Helpers
    - [AppException.cs](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#app-exception-cs)
    - [AppSettings.cs](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#app-settings-cs)
    - [AutoMapperProfile.cs](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#automapper-profile-cs)
    - [DataContext.cs](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#data-context-cs)
  + Services
    - [UserService.cs](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#user-service-cs)
  + [appsettings.Development.json](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#app-settings-development-json)
  + [appsettings.json](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#app-settings-json)
  + [Program.cs](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#program-cs)
  + [Startup.cs](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#startup-cs)
  + [WebApi.csproj](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#web-api-csproj)
* [client](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#client-folder)
  + app
    - \_directives
      * [alert.component.html](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#alert-component-html)
      * [alert.component.ts](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#alert-component-ts)
      * index.ts
    - \_guards
      * [auth.guard.ts](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#auth-guard-ts)
      * index.ts
    - \_models
      * [user.ts](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#user-ts)
      * index.ts
    - \_services
      * [alert.service.ts](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#alert-service-ts)
      * [authentication.service.ts](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#authentication-service-ts)
      * index.ts
      * [user.service.ts](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#user-service-ts)
    - home
      * [home.component.html](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#home-component-html)
      * [home.component.ts](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#home-component-ts)
      * index.ts
    - login
      * index.ts
      * [login.component.html](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#login-component-html)
      * [login.component.ts](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#login-component-ts)
    - register
      * index.ts
      * [register.component.html](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#register-component-html)
      * [register.component.ts](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#register-component-ts)
    - [app.component.html](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#app-component-html)
    - [app.component.ts](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#app-component-ts)
    - [app.config.ts](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#app-config-ts)
    - [app.module.ts](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#app-module-ts)
    - [app.routing.ts](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#app-routing-ts)
    - [main.ts](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#main-ts)
  + app.css
  + index.html
  + package.json
  + system.config.js
  + tsconfig.json

**ASP.NET Core with Angular 2/4 Server**

**Path: /server**

The server directory contains all the files for the ASP.NET Core Web API. The Web API project is organised into the following folders:

* Controllers - define the end points / routes for the web api, controllers are the entry point into the web api from client applications via http requests.
* Services - contain business logic, validation and database access code.
* Entities - represent the application data that is stored in the database.
* Dtos - data transfer objects used by controllers to expose a limited set of entity data via the api, and for model binding data from HTTP requests to controller action methods.
* Helpers - anything that doesn't fit into the above folders.

Separating and encapsulating concerns like this is a good way to keep your application code clean, organised and maintainable as the app grows.

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**ASP.NET Core Users Controller**

**Path: /server/Controllers/UsersController.cs**

The ASP.NET Core users controller defines and handles all routes / endpoints for the api that relate to users, this includes authentication, registration and standard CRUD operations. Within each route the controller calls the user service to perform the action required, this enables the controller to stay 'lean' and completely separated from the database / persistence code.

The controller actions are secured with JWT using the [Authorize] attribute, with the exception of the Authenticate and Register methods which allow public access by overriding the [Authorize] attribute on the controller with [AllowAnonymous] attributes on each action method. I chose this approach so any new action methods added to the controller will be secure by default unless explicitly made public.

On successful authentication the Authenticate method generates a JWT (JSON Web Token) using the JwtSecurityTokenHandler class that generates a token that is digitally signed using a secret key stored in appsettings.json. The JWT token is returned to the Angular client which then includes it in the HTTP Authorization header of subsequent web api requests for authentication.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132 | using System;  using System.Collections.Generic;  using Microsoft.AspNetCore.Mvc;  using WebApi.Services;  using WebApi.Dtos;  using AutoMapper;  using System.IdentityModel.Tokens.Jwt;  using WebApi.Helpers;  using Microsoft.Extensions.Options;  using System.Text;  using Microsoft.IdentityModel.Tokens;  using System.Security.Claims;  using WebApi.Entities;  using Microsoft.AspNetCore.Authorization;    namespace WebApi.Controllers  {      [Authorize]      [Route("[controller]")]      public class UsersController : Controller      {          private IUserService \_userService;          private IMapper \_mapper;          private readonly AppSettings \_appSettings;            public UsersController(              IUserService userService,              IMapper mapper,              IOptions<AppSettings> appSettings)          {              \_userService = userService;              \_mapper = mapper;              \_appSettings = appSettings.Value;          }            [AllowAnonymous]          [HttpPost("authenticate")]          public IActionResult Authenticate([FromBody]UserDto userDto)          {              var user = \_userService.Authenticate(userDto.Username, userDto.Password);                if (user == null)                  return Unauthorized();                var tokenHandler = new JwtSecurityTokenHandler();              var key = Encoding.ASCII.GetBytes(\_appSettings.Secret);              var tokenDescriptor = new SecurityTokenDescriptor              {                  Subject = new ClaimsIdentity(new Claim[]                  {                      new Claim(ClaimTypes.Name, user.Id.ToString())                  }),                  Expires = DateTime.UtcNow.AddDays(7),                  SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)              };              var token = tokenHandler.CreateToken(tokenDescriptor);              var tokenString = tokenHandler.WriteToken(token);                // return basic user info (without password) and token to store client side              return Ok(new {                  Id = user.Id,                  Username = user.Username,                  FirstName = user.FirstName,                  LastName = user.LastName,                  Token = tokenString              });          }            [AllowAnonymous]          [HttpPost]          public IActionResult Register([FromBody]UserDto userDto)          {              // map dto to entity              var user = \_mapper.Map<User>(userDto);                try              {                  // save                  \_userService.Create(user, userDto.Password);                  return Ok();              }              catch(AppException ex)              {                  // return error message if there was an exception                  return BadRequest(ex.Message);              }          }            [HttpGet]          public IActionResult GetAll()          {              var users =  \_userService.GetAll();              var userDtos = \_mapper.Map<IList<UserDto>>(users);              return Ok(userDtos);          }            [HttpGet("{id}")]          public IActionResult GetById(int id)          {              var user =  \_userService.GetById(id);              var userDto = \_mapper.Map<UserDto>(user);              return Ok(userDto);          }            [HttpPut("{id}")]          public IActionResult Update(int id, [FromBody]UserDto userDto)          {              // map dto to entity and set id              var user = \_mapper.Map<User>(userDto);              user.Id = id;                try              {                  // save                  \_userService.Update(user, userDto.Password);                  return Ok();              }              catch(AppException ex)              {                  // return error message if there was an exception                  return BadRequest(ex.Message);              }          }            [HttpDelete("{id}")]          public IActionResult Delete(int id)          {              \_userService.Delete(id);              return Ok();          }      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**ASP.NET Core User DTO (Data Transfer Object)**

**Path: /server/Dtos/UserDto.cs**

The user DTO is a data transfer object used send selected user data to and from the users api end points.

It doesn't contain the PasswordHash and PasswordSalt fields of the user entity class so these fields aren't included in responses from the web api when the controller maps data from user entities to user dtos.

The Password property in the DTO is only used for model binding data coming into the controller from http requests (e.g. authenticate, register etc), passwords are never included in responses from the web api. Some developers might prefer to have two DTOs in this case, one for incoming requests that includes a password and another for responses without a password, but I prefer to have less code where possible for maintainability.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | namespace WebApi.Dtos  {      public class UserDto      {          public int Id { get; set; }          public string FirstName { get; set; }          public string LastName { get; set; }          public string Username { get; set; }          public string Password { get; set; }      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**ASP.NET Core User Entity**

**Path: /server/Entities/User.cs**

The user entity class represents the data stored in the database for users. It's used by Entity Framework Core to map relational data from the database into .NET objects for data management and CRUD operations.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | namespace WebApi.Entities  {      public class User      {          public int Id { get; set; }          public string FirstName { get; set; }          public string LastName { get; set; }          public string Username { get; set; }          public byte[] PasswordHash { get; set; }          public byte[] PasswordSalt { get; set; }      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**ASP.NET Core App Exception**

**Path: /server/Helpers/AppException.cs**

The app exception is a custom exceptions class used to differentiate between handled and unhandled exceptions. Handled exceptions are ones generated by the application and used to display friendly error messages to the client, for example business logic or validation exceptions caused by incorrect input from the user. Unhandled exceptions are generated by the .NET framework and caused by bugs in the application code.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | using System;  using System.Globalization;    namespace WebApi.Helpers  {      // Custom exception class for throwing application specific exceptions (e.g. for validation)      // that can be caught and handled within the application      public class AppException : Exception      {          public AppException() : base() {}            public AppException(string message) : base(message) { }            public AppException(string message, params object[] args)              : base(String.Format(CultureInfo.CurrentCulture, message, args))          {          }      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**ASP.NET Core App Settings**

**Path: /server/Helpers/AppSettings.cs**

The app settings class contains properties defined in the appsettings.json file and is used for accessing application settings via objects that injected into classes using the ASP.NET Core built in dependency injection. For example the Users Controller accesses app settings via an "IOptions<AppSettings> appSettings" object that is injected into the constructor.

Mapping of configuration sections to classes is done in the ConfigureServices method of the Startup.cs file.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | namespace WebApi.Helpers  {      public class AppSettings      {          public string Secret { get; set; }      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**ASP.NET Core AutoMapper Profile**

**Path: /server/Helpers/AutoMapperProfile.cs**

The automapper profile contains the mapping configuration used by the application, it enables mapping of user entities to dtos and dtos to entities.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | using AutoMapper;  using WebApi.Dtos;  using WebApi.Entities;    namespace WebApi.Helpers  {      public class AutoMapperProfile : Profile      {          public AutoMapperProfile()          {              CreateMap<User, UserDto>();              CreateMap<UserDto, User>();          }      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**ASP.NET Core Data Context**

**Path: /server/Helpers/DataContext.cs**

The data context class is used for accessing application data through Entity Framework Core. It derives from the EF Core DbContext class and has a public Users property for accessing and managing user data. The data context is used by services for handling all low level data operations.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | using Microsoft.EntityFrameworkCore;  using WebApi.Entities;    namespace WebApi.Helpers  {      public class DataContext : DbContext      {          public DataContext(DbContextOptions<DataContext> options) : base(options) { }            public DbSet<User> Users { get; set; }      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**ASP.NET Core User Service**

**Path: /server/Services/UserService.cs**

The ASP.NET Core user service is responsible for all database interaction and core business logic related to user authentication, registration and management.

The top of the file contains an interface that defines the user service, below that is the concrete user service class that implements the interface. The bottom of the class contains a couple of private methods used for creating and verifying hashed passwords that are stored in the database.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154 | using System;  using System.Collections.Generic;  using System.Linq;  using WebApi.Entities;  using WebApi.Helpers;    namespace WebApi.Services  {      public interface IUserService      {          User Authenticate(string username, string password);          IEnumerable<User> GetAll();          User GetById(int id);          User Create(User user, string password);          void Update(User user, string password = null);          void Delete(int id);      }        public class UserService : IUserService      {          private DataContext \_context;            public UserService(DataContext context)          {              \_context = context;          }            public User Authenticate(string username, string password)          {              if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))                  return null;                var user = \_context.Users.SingleOrDefault(x => x.Username == username);                // check if username exists              if (user == null)                  return null;                // check if password is correct              if (!VerifyPasswordHash(password, user.PasswordHash, user.PasswordSalt))                  return null;                // authentication successful              return user;          }            public IEnumerable<User> GetAll()          {              return \_context.Users;          }            public User GetById(int id)          {              return \_context.Users.Find(id);          }            public User Create(User user, string password)          {              // validation              if (string.IsNullOrWhiteSpace(password))                  throw new AppException("Password is required");                if (\_context.Users.Any(x => x.Username == user.Username))                  throw new AppException("Username " + user.Username + " is already taken");                byte[] passwordHash, passwordSalt;              CreatePasswordHash(password, out passwordHash, out passwordSalt);                user.PasswordHash = passwordHash;              user.PasswordSalt = passwordSalt;                \_context.Users.Add(user);              \_context.SaveChanges();                return user;          }            public void Update(User userParam, string password = null)          {              var user = \_context.Users.Find(userParam.Id);                if (user == null)                  throw new AppException("User not found");                if (userParam.Username != user.Username)              {                  // username has changed so check if the new username is already taken                  if (\_context.Users.Any(x => x.Username == userParam.Username))                      throw new AppException("Username " + userParam.Username + " is already taken");              }                // update user properties              user.FirstName = userParam.FirstName;              user.LastName = userParam.LastName;              user.Username = userParam.Username;                // update password if it was entered              if (!string.IsNullOrWhiteSpace(password))              {                  byte[] passwordHash, passwordSalt;                  CreatePasswordHash(password, out passwordHash, out passwordSalt);                    user.PasswordHash = passwordHash;                  user.PasswordSalt = passwordSalt;              }                \_context.Users.Update(user);              \_context.SaveChanges();          }            public void Delete(int id)          {              var user = \_context.Users.Find(id);              if (user != null)              {                  \_context.Users.Remove(user);                  \_context.SaveChanges();              }          }            // private helper methods            private static void CreatePasswordHash(string password, out byte[] passwordHash, out byte[] passwordSalt)          {              if (password == null) throw new ArgumentNullException("password");              if (string.IsNullOrWhiteSpace(password)) throw new ArgumentException("Value cannot be empty or whitespace only string.", "password");                using (var hmac = new System.Security.Cryptography.HMACSHA512())              {                  passwordSalt = hmac.Key;                  passwordHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));              }          }            private static bool VerifyPasswordHash(string password, byte[] storedHash, byte[] storedSalt)          {              if (password == null) throw new ArgumentNullException("password");              if (string.IsNullOrWhiteSpace(password)) throw new ArgumentException("Value cannot be empty or whitespace only string.", "password");              if (storedHash.Length != 64) throw new ArgumentException("Invalid length of password hash (64 bytes expected).", "passwordHash");              if (storedSalt.Length != 128) throw new ArgumentException("Invalid length of password salt (128 bytes expected).", "passwordHash");                using (var hmac = new System.Security.Cryptography.HMACSHA512(storedSalt))              {                  var computedHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));                  for (int i = 0; i < computedHash.Length; i++)                  {                      if (computedHash[i] != storedHash[i]) return false;                  }              }                return true;          }      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**ASP.NET Core App Settings (Development)**

**Path: /server/appsettings.Development.json**

Configuration file with application settings that are specific to the development environment.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | {    "Logging": {      "IncludeScopes": false,      "LogLevel": {        "Default": "Debug",        "System": "Information",        "Microsoft": "Information"      }    }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**ASP.NET Core App Settings**

**Path: /server/appsettings.json**

Root configuration file containing application settings for all environments.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | {    "AppSettings": {      "Secret": "REPLACE THIS WITH YOUR OWN SECRET, IT CAN BE ANY STRING"    },    "Logging": {      "IncludeScopes": false,      "LogLevel": {        "Default": "Warning"      }    }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**ASP.NET Core Program**

**Path: /server/Program.cs**

The program class is a console app that is the main entry point to start the application, it configures and launches the web api host and web server using an instance of WebHostBuilder. ASP.NET Core applications require a host in which to execute.

Kestrel is the web server used in the example, it's a new cross-platform web server for ASP.NET Core that's included in new project templates by default. Kestrel is fine to use on it's own for internal applications and development, but for public facing websites and applications it should sit behind a more mature reverse proxy server (IIS, Apache, Nginx etc) that will receive HTTP requests from the internet and forward them to Kestrel after initial handling and security checks.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | using System.IO;  using Microsoft.AspNetCore;  using Microsoft.AspNetCore.Hosting;    namespace WebApi  {      public class Program      {          public static void Main(string[] args)          {              BuildWebHost(args).Run();          }            public static IWebHost BuildWebHost(string[] args) =>              WebHost.CreateDefaultBuilder(args)                  .UseStartup<Startup>()                  .Build();      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**ASP.NET Core Startup**

**Path: /server/Startup.cs**

The startup class configures the request pipeline of the application and how all requests are handled.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84 | using System.Collections.Generic;  using System.Linq;  using System.Threading.Tasks;  using Microsoft.AspNetCore.Builder;  using Microsoft.AspNetCore.Hosting;  using Microsoft.Extensions.Configuration;  using Microsoft.Extensions.DependencyInjection;  using Microsoft.Extensions.Logging;  using Microsoft.EntityFrameworkCore;  using WebApi.Helpers;  using WebApi.Services;  using AutoMapper;  using Microsoft.IdentityModel.Tokens;  using System.Text;  using Microsoft.AspNetCore.Authentication.JwtBearer;  using Microsoft.Extensions.Options;    namespace WebApi  {      public class Startup      {          public Startup(IConfiguration configuration)          {              Configuration = configuration;          }            public IConfiguration Configuration { get; }            // This method gets called by the runtime. Use this method to add services to the container.          public void ConfigureServices(IServiceCollection services)          {              services.AddCors();              services.AddDbContext<DataContext>(x => x.UseInMemoryDatabase("TestDb"));              services.AddMvc();              services.AddAutoMapper();                // configure strongly typed settings objects              var appSettingsSection = Configuration.GetSection("AppSettings");              services.Configure<AppSettings>(appSettingsSection);                // configure jwt authentication              var appSettings = appSettingsSection.Get<AppSettings>();              var key = Encoding.ASCII.GetBytes(appSettings.Secret);              services.AddAuthentication(x =>              {                  x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;                  x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;              })              .AddJwtBearer(x =>              {                  x.RequireHttpsMetadata = false;                  x.SaveToken = true;                  x.TokenValidationParameters = new TokenValidationParameters                  {                      ValidateIssuerSigningKey = true,                      IssuerSigningKey = new SymmetricSecurityKey(key),                      ValidateIssuer = false,                      ValidateAudience = false                  };              });                // configure DI for application services              services.AddScoped<IUserService, UserService>();          }            // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.          public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)          {              loggerFactory.AddConsole(Configuration.GetSection("Logging"));              loggerFactory.AddDebug();                // global cors policy              app.UseCors(x => x                  .AllowAnyOrigin()                  .AllowAnyMethod()                  .AllowAnyHeader()                  .AllowCredentials());                app.UseAuthentication();                app.UseMvc();          }      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**ASP.NET Core Web Api csproj**

**Path: /server/WebApi.csproj**

The csproj (C# project) is an MSBuild based file that contains target framework and NuGet package dependency information for the application.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <Project Sdk="Microsoft.NET.Sdk.Web">    <PropertyGroup>      <TargetFramework>netcoreapp2.0</TargetFramework>    </PropertyGroup>    <ItemGroup>      <PackageReference Include="AutoMapper" Version="6.0.2" />      <PackageReference Include="AutoMapper.Extensions.Microsoft.DependencyInjection" Version="2.0.1" />      <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.0" />    </ItemGroup>  </Project> |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**ASP.NET Core with Angular 2/4 Client**

**Path: /client**

I used the [Angular 2 quickstart](https://angular.io/docs/ts/latest/quickstart.html) project as a base for the client application, it's written in TypeScript and uses systemjs for loading modules. If you're new to angular 2 I'd recommend checking out the quickstart as it provides details on the project tooling and configuration files which aren't covered in this post.

The project and code structure mostly follows the recommendations in the official [Angular 2 style guide](https://angular.io/styleguide), with my own tweaks here and there.

Each feature has it's own folder (home, login & register), other code such as services, models, guards etc are placed in folders prefixed with an underscore to easily differentiate them and group them together at the top of the folder structure.

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 Alert Component Template**

**Path: /client/app/\_directives/alert.component.html**

The alert component template contains the html for displaying alert messages at the top of the page.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1 | <div \*ngIf="message" [ngClass]="{ 'alert': message, 'alert-success': message.type === 'success', 'alert-danger': message.type === 'error' }">{{message.text}}</div> |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 Alert Component**

**Path: /client/app/\_directives/alert.component.ts**

The alert component passes alert messages to the template whenever a message is received from the alert service. It does this by subscribing to the alert service's getMessage() method which returns an Observable.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import { Component, OnInit } from '@angular/core';    import { AlertService } from '../\_services/index';    @Component({      moduleId: module.id,      selector: 'alert',      templateUrl: 'alert.component.html'  })    export class AlertComponent {      message: any;        constructor(private alertService: AlertService) { }        ngOnInit() {          this.alertService.getMessage().subscribe(message => { this.message = message; });      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 Auth Guard**

**Path: /client/app/\_guards/auth.guard.ts**

The auth guard is used to prevent unauthenticated users from accessing restricted routes, in this example it's used in app.routing.ts to protect the home page route. For more information about angular 2 guards you can check out [this post](http://blog.thoughtram.io/angular/2016/07/18/guards-in-angular-2.html) on the thoughtram blog.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import { Injectable } from '@angular/core';  import { Router, CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';    @Injectable()  export class AuthGuard implements CanActivate {        constructor(private router: Router) { }        canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {          if (localStorage.getItem('currentUser')) {              // logged in so return true              return true;          }            // not logged in so redirect to login page with the return url          this.router.navigate(['/login'], { queryParams: { returnUrl: state.url }});          return false;      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 User Model**

**Path: /client/app/\_models/user.ts**

The user model is a small class that defines the properties of a user.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | export class User {      \_id: string;      username: string;      password: string;      firstName: string;      lastName: string;  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 Alert Service**

**Path: /client/app/\_services/alert.service.ts**

The alert service enables any component in the application to display alert messages at the top of the page via the alert component.

It has methods for displaying success and error messages, and a getMessage() method that returns an Observable that is used by the alert component to subscribe to notifications for whenever a message should be displayed.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | import { Injectable } from '@angular/core';  import { Router, NavigationStart } from '@angular/router';  import { Observable } from 'rxjs';  import { Subject } from 'rxjs/Subject';    @Injectable()  export class AlertService {      private subject = new Subject<any>();      private keepAfterNavigationChange = false;        constructor(private router: Router) {          // clear alert message on route change          router.events.subscribe(event => {              if (event instanceof NavigationStart) {                  if (this.keepAfterNavigationChange) {                      // only keep for a single location change                      this.keepAfterNavigationChange = false;                  } else {                      // clear alert                      this.subject.next();                  }              }          });      }        success(message: string, keepAfterNavigationChange = false) {          this.keepAfterNavigationChange = keepAfterNavigationChange;          this.subject.next({ type: 'success', text: message });      }        error(message: string, keepAfterNavigationChange = false) {          this.keepAfterNavigationChange = keepAfterNavigationChange;          this.subject.next({ type: 'error', text: message });      }        getMessage(): Observable<any> {          return this.subject.asObservable();      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 Authentication Service**

**Path: /client/app/\_services/authentication.service.ts**

The authentication service is used to login and logout of the application, to login it posts the users credentials to the api and checks the response for a JWT token, if there is one it means authentication was successful so the user details including the token are added to local storage.

The logged in user details are stored in local storage so the user will stay logged in if they refresh the browser and also between browser sessions until they logout. If you don't want the user to stay logged in between refreshes or sessions the behaviour could easily be changed by storing user details somewhere less persistent such as session storage or in a property of the authentication service.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | import { Injectable } from '@angular/core';  import { Http, Headers, Response } from '@angular/http';  import { Observable } from 'rxjs/Observable';  import 'rxjs/add/operator/map'    import { AppConfig } from '../app.config';    @Injectable()  export class AuthenticationService {      constructor(private http: Http, private config: AppConfig) { }        login(username: string, password: string) {          return this.http.post(this.config.apiUrl + '/users/authenticate', { username: username, password: password })              .map((response: Response) => {                  // login successful if there's a jwt token in the response                  let user = response.json();                  if (user && user.token) {                      // store user details and jwt token in local storage to keep user logged in between page refreshes                      localStorage.setItem('currentUser', JSON.stringify(user));                  }              });      }        logout() {          // remove user from local storage to log user out          localStorage.removeItem('currentUser');      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 User Service**

**Path: /client/app/\_services/user.service.ts**

The user service contains a standard set of CRUD methods for managing users, it contains a jwt() method that's used to add the JWT token from local storage to the Authorization header of each http request.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | import { Injectable } from '@angular/core';  import { Http, Headers, RequestOptions, Response } from '@angular/http';    import { AppConfig } from '../app.config';  import { User } from '../\_models/index';    @Injectable()  export class UserService {      constructor(private http: Http, private config: AppConfig) { }        getAll() {          return this.http.get(this.config.apiUrl + '/users', this.jwt()).map((response: Response) => response.json());      }        getById(\_id: string) {          return this.http.get(this.config.apiUrl + '/users/' + \_id, this.jwt()).map((response: Response) => response.json());      }        create(user: User) {          return this.http.post(this.config.apiUrl + '/users/register', user, this.jwt());      }        update(user: User) {          return this.http.put(this.config.apiUrl + '/users/' + user.\_id, user, this.jwt());      }        delete(\_id: string) {          return this.http.delete(this.config.apiUrl + '/users/' + \_id, this.jwt());      }        // private helper methods        private jwt() {          // create authorization header with jwt token          let currentUser = JSON.parse(localStorage.getItem('currentUser'));          if (currentUser && currentUser.token) {              let headers = new Headers({ 'Authorization': 'Bearer ' + currentUser.token });              return new RequestOptions({ headers: headers });          }      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 Home Component Template**

**Path: /client/app/home/home.component.html**

The home component template contains html and angular 2 template syntax for displaying a simple welcome message, a list of users and a logout link.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <div class="col-md-6 col-md-offset-3">      <h1>Hi {{currentUser.firstName}}!</h1>      <p>You're logged in with the MEAN Stack & Angular 2!!</p>      <h3>All registered users:</h3>      <ul>          <li \*ngFor="let user of users">              {{user.username}} ({{user.firstName}} {{user.lastName}})              - <a (click)="deleteUser(user.\_id)">Delete</a>          </li>      </ul>      <p><a [routerLink]="['/login']">Logout</a></p>  </div> |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 Home Component**

**Path: /client/app/home/home.component.ts**

The home component gets the current user from local storage and all users from the user service, and makes them available to the template.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | import { Component, OnInit } from '@angular/core';    import { User } from '../\_models/index';  import { UserService } from '../\_services/index';    @Component({      moduleId: module.id,      templateUrl: 'home.component.html'  })    export class HomeComponent implements OnInit {      currentUser: User;      users: User[] = [];        constructor(private userService: UserService) {          this.currentUser = JSON.parse(localStorage.getItem('currentUser'));      }        ngOnInit() {          this.loadAllUsers();      }        deleteUser(\_id: string) {          this.userService.delete(\_id).subscribe(() => { this.loadAllUsers() });      }        private loadAllUsers() {          this.userService.getAll().subscribe(users => { this.users = users; });      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 Login Component Template**

**Path: /client/app/login/login.component.html**

The login component template contains a login form with username and password fields. It displays validation messages for invalid fields when the submit button is clicked. On submit the login() method is called as long as the form is valid.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <div class="col-md-6 col-md-offset-3">      <h2>Login</h2>      <form name="form" (ngSubmit)="f.form.valid && login()" #f="ngForm" novalidate>          <div class="form-group" [ngClass]="{ 'has-error': f.submitted && !username.valid }">              <label for="username">Username</label>              <input type="text" class="form-control" name="username" [(ngModel)]="model.username" #username="ngModel" required />              <div \*ngIf="f.submitted && !username.valid" class="help-block">Username is required</div>          </div>          <div class="form-group" [ngClass]="{ 'has-error': f.submitted && !password.valid }">              <label for="password">Password</label>              <input type="password" class="form-control" name="password" [(ngModel)]="model.password" #password="ngModel" required />              <div \*ngIf="f.submitted && !password.valid" class="help-block">Password is required</div>          </div>          <div class="form-group">              <button [disabled]="loading" class="btn btn-primary">Login</button>              <img \*ngIf="loading" src="data:image/gif;base64," />              <a [routerLink]="['/register']" class="btn btn-link">Register</a>          </div>      </form>  </div> |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 Login Component**

**Path: /client/app/login/login.component.ts**

The login component uses the authentication service to login and logout of the application. It automatically logs the user out when it initializes (ngOnInit) so the login page can also be used to logout.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | import { Component, OnInit } from '@angular/core';  import { Router, ActivatedRoute } from '@angular/router';    import { AlertService, AuthenticationService } from '../\_services/index';    @Component({      moduleId: module.id,      templateUrl: 'login.component.html'  })    export class LoginComponent implements OnInit {      model: any = {};      loading = false;      returnUrl: string;        constructor(          private route: ActivatedRoute,          private router: Router,          private authenticationService: AuthenticationService,          private alertService: AlertService) { }        ngOnInit() {          // reset login status          this.authenticationService.logout();            // get return url from route parameters or default to '/'          this.returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/';      }        login() {          this.loading = true;          this.authenticationService.login(this.model.username, this.model.password)              .subscribe(                  data => {                      this.router.navigate([this.returnUrl]);                  },                  error => {                      this.alertService.error(error.\_body);                      this.loading = false;                  });      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 Register Component Template**

**Path: /client/app/register/register.component.html**

The register component template contains a simplae registration form with fields for first name, last name, username and password. It displays validation messages for invalid fields when the submit button is clicked. On submit the register() method is called if the form is valid.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | <div class="col-md-6 col-md-offset-3">      <h2>Register</h2>      <form name="form" (ngSubmit)="f.form.valid && register()" #f="ngForm" novalidate>          <div class="form-group" [ngClass]="{ 'has-error': f.submitted && !username.valid }">              <label for="firstName">First Name</label>              <input type="text" class="form-control" name="firstName" [(ngModel)]="model.firstName" #firstName="ngModel" required />              <div \*ngIf="f.submitted && !firstName.valid" class="help-block">First Name is required</div>          </div>          <div class="form-group" [ngClass]="{ 'has-error': f.submitted && !username.valid }">              <label for="lastName">Last Name</label>              <input type="text" class="form-control" name="lastName" [(ngModel)]="model.lastName" #lastName="ngModel" required />              <div \*ngIf="f.submitted && !lastName.valid" class="help-block">Last Name is required</div>          </div>          <div class="form-group" [ngClass]="{ 'has-error': f.submitted && !username.valid }">              <label for="username">Username</label>              <input type="text" class="form-control" name="username" [(ngModel)]="model.username" #username="ngModel" required />              <div \*ngIf="f.submitted && !username.valid" class="help-block">Username is required</div>          </div>          <div class="form-group" [ngClass]="{ 'has-error': f.submitted && !password.valid }">              <label for="password">Password</label>              <input type="password" class="form-control" name="password" [(ngModel)]="model.password" #password="ngModel" required />              <div \*ngIf="f.submitted && !password.valid" class="help-block">Password is required</div>          </div>          <div class="form-group">              <button [disabled]="loading" class="btn btn-primary">Register</button>              <img \*ngIf="loading" src="data:image/gif;base64," />              <a [routerLink]="['/login']" class="btn btn-link">Cancel</a>          </div>      </form>  </div> |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 Register Component**

**Path: /client/app/register/register.component.ts**

The register component has a single register() method that creates a new user with the user service when the register form is submitted.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | import { Component } from '@angular/core';  import { Router } from '@angular/router';    import { AlertService, UserService } from '../\_services/index';    @Component({      moduleId: module.id,      templateUrl: 'register.component.html'  })    export class RegisterComponent {      model: any = {};      loading = false;        constructor(          private router: Router,          private userService: UserService,          private alertService: AlertService) { }        register() {          this.loading = true;          this.userService.create(this.model)              .subscribe(                  data => {                      this.alertService.success('Registration successful', true);                      this.router.navigate(['/login']);                  },                  error => {                      this.alertService.error(error.\_body);                      this.loading = false;                  });      }  } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 App Component Template**

**Path: /client/app/app.component.html**

The app component template is the root component template of the application, it contains a router-outlet directive for displaying the contents of each view based on the current route, and an alert directive for displaying alert messages from anywhere in the system.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <!-- main app container -->  <div class="jumbotron">      <div class="container">          <div class="col-sm-8 col-sm-offset-2">              <alert></alert>              <router-outlet></router-outlet>          </div>      </div>  </div>    <!-- credits -->  <div class="text-center">      <p>          <a href="<http://jasonwatmore.com/post/2017/02/22/mean-with-angular-2-user-registration-and-login-example-tutorial>" target="\_top">MEAN Stack with Angular 2 - User Registration and Login Example & Tutorial</a>      </p>      <p>          <a href="[http://jasonwatmore.com](http://jasonwatmore.com/)" target="\_top">JasonWatmore.com</a>      </p>  </div> |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 App Component**

**Path: /client/app/app.component.ts**

The app component is the root component of the application, it defines the root tag of the app as <app></app> with the selector property.

The moduleId property is set to allow a relative path to be used for the templateUrl.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import { Component } from '@angular/core';    @Component({      moduleId: module.id,      selector: 'app',      templateUrl: 'app.component.html'  })    export class AppComponent { } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 App Config**

**Path: /client/app/app.config.ts**

The app config class is used to store application config variables (like the api endpoint url) in a single place that's easily injected into any component. In the example it's used by the Angular 2 User Service and Authentication Service.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3 | export class AppConfig {      public readonly apiUrl = '[http://localhost:4000](http://localhost:4000/)';  }; |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 App Module**

**Path: /client/app/app.module.ts**

The app module defines the root module of the application along with metadata about the module. For more info about angular 2 modules check out [this page](https://angular.io/docs/ts/latest/guide/ngmodule.html) on the official docs site.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | import { NgModule } from '@angular/core';  import { BrowserModule } from '@angular/platform-browser';  import { FormsModule } from '@angular/forms';  import { HttpModule } from '@angular/http';    import { AppComponent } from './app.component';  import { routing } from './app.routing';  import { AppConfig } from './app.config';    import { AlertComponent } from './\_directives/index';  import { AuthGuard } from './\_guards/index';  import { AlertService, AuthenticationService, UserService } from './\_services/index';  import { HomeComponent } from './home/index';  import { LoginComponent } from './login/index';  import { RegisterComponent } from './register/index';    @NgModule({      imports: [          BrowserModule,          FormsModule,          HttpModule,          routing      ],      declarations: [          AppComponent,          AlertComponent,          HomeComponent,          LoginComponent,          RegisterComponent      ],      providers: [          AppConfig,          AuthGuard,          AlertService,          AuthenticationService,          UserService      ],      bootstrap: [AppComponent]  })    export class AppModule { } |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 App Routing**

**Path: /client/app/app.routing.ts**

The app routing file defines the routes of the application, each route contains a path and associated component. The home route is secured by passing the AuthGuard to the canActivate property of the route.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | import { Routes, RouterModule } from '@angular/router';    import { HomeComponent } from './home/index';  import { LoginComponent } from './login/index';  import { RegisterComponent } from './register/index';  import { AuthGuard } from './\_guards/index';    const appRoutes: Routes = [      { path: '', component: HomeComponent, canActivate: [AuthGuard] },      { path: 'login', component: LoginComponent },      { path: 'register', component: RegisterComponent },        // otherwise redirect to home      { path: '\*\*', redirectTo: '' }  ];    export const routing = RouterModule.forRoot(appRoutes); |

[Back to top](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example#projectstructure)

**Angular 2/4 Main (Bootstrap) File**

**Path: /client/app/main.ts**

The main file is the entry point used by angular to launch and bootstrap the application.

[?](https://www.pointblankdevelopment.com.au/blog/113/aspnet-core-20-angular-24-user-registration-and-login-tutorial-example)

|  |  |
| --- | --- |
| 1  2  3  4  5 | import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';    import { AppModule } from './app.module';    platformBrowserDynamic().bootstrapModule(AppModule); |