Packages needed to build the application:

1. **ASP.Net Core** // dotnet.microsoft.com/download {version: .Net Core 2.2}  
   C:\dotnet --info //shows SDK and .Net Core version
2. **node.js and npm tool** //nodejs.org/en  
   c:\node –version //shows node.js version  
   c:\npm //shows npm version
3. **SQLite** //Lighsqlitebrowser.org
4. **Visual Studio Code** //Coding IDE. code.visualstudio.com
5. **Postman** //Test API. getpostman.com

Create API:

c:\mkdir DatingApp  
c:\DatingApp\dotnet new webapi -n DatingApp.API  
c:\code . //open project

Install Extensions:

1. C# for VS Code
2. C# Extesnions: Set of classes
3. NuGet Package Manager: To install additional packages into our application. It creates a new folder to the project “vscode”, which allows us to launch the debugger for our application. If files under .vscode are missing, open the command pallet and type > generate, it will generate assets for Build and Debug.

Hide folders “obj, bin” by editing settings.json (change setting format into Json format)

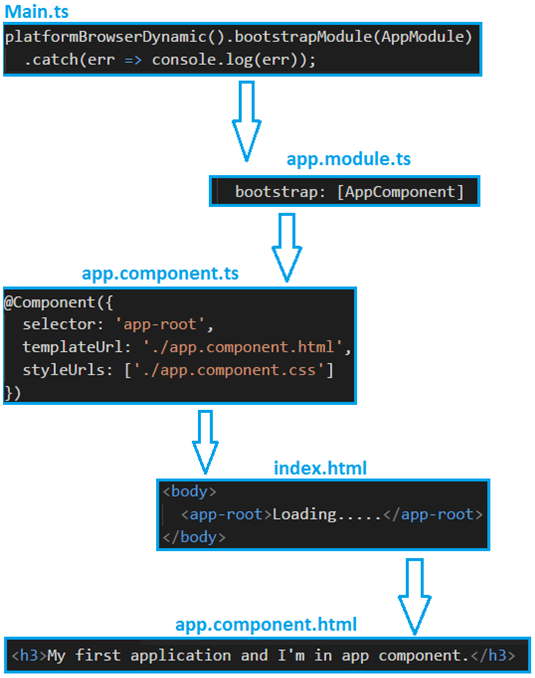
Startup.cs: Turn off app.UseHsts(), app.UseMvc() in development

launchSetting.json  
 launchBrowser: false,  
 launchURL: “api/values”  
 applicationUrl: <http://localhost:5000>  
 ASPNETCORE:ENVIRONMENT: Development

Show Terminal window: Ctrl + `

>dotnet run //run project  
>dotnet watch run //enable changes on runtime

**[Bootstrap sequence]**



**[Creating database with EntityFramework]**

Add DataContext.cs file under Data folder  
Edit appsettings.json, add Connectionstrings

Edit Startup.cs, add service for database connection

>dotnet ef -h //shows command available to use with entity framework

Tables are created based on DbSet<> defined in the DataContext class.

>dotnet ef migrations add InitialCreate //InitialCreates.cs is created with Up, Down methods

>dotnet ef database update //Create db applying this migration, DatingApp.db is created.

**[Retrieve value from db]**

ValueController.cs

Add constructor:

private readolnly DataContext \_context;

public ValuesController(DataContext context)  
{  
 \_context = context;  
}

Remove ActionResult and use IActionResul Get() instead

**[Making the code asynchronous]**

**[Angular]**

Install Angular CLI. Open Node.js command prompt:

>npm install -g @angular/cli

Create App:

>ng new DatingApp-SPA //JavaScript does not allow . in file name  
…routing: No…

>code . under parent folder //Open both project

>ng serve //run the code

The node\_modules contains hundreds of dependencies, which is created based on the information in package.json. If move to another computer we won’t have to save the module into source control, all of the dependencies can be recreated based on what’s contained inside package.json

app.modules.ts: it bootstraps app.component.ts, a class that provides data for our Views, such as: templateUrl and styleUrls

main.ts: it bootstraps our app, it has information about the module that we’re going to bootstrap when we run our app.

index.html: SPA

**[Retrieving data from database]**

Add extensions: Angular Snippets, Angular Files, Angular Language Service (provide intellisense, a rich editing experience), Auto Rename Tag, Bracket Pair Colorizer 2, Debugger for Chrome, Material Icon Theme, Prettier-Code formatter, TSLint, Angular2-switcher.

On app folder generate new component (Value), a folder named Value is added containing value.component.ts file and ValueComponent will be automatically added to app.module.ts

Add HttpClientMoudule to imports right afer BrowserModule and import angular client module:  
import {HttpsClientModule} from ‘@angular/common/http’;

Configure CORS policy, so Angular can access the service. Add lines to Startup.cs inside ConfigureServices and Configure methods)

\* represents a structural directive in angular, it modify the DOM in some way.

**[Styling – Adding Bootstrap and Font-Awesome]**

getbootstrap.com

>npm install bootstrap font-awesome

angular.json

“styles”:[  
 “src/styles.css”  
…add styles or import the style we need into it…  
],

Open styles.css:

@import ‘../node\_modules/bootstrap/dist/css/bootstrap.min.css’;  
@import ‘../node\_modules/font-awesome/css/font-awesome.min.css;

Change in app.component.html, remove style and change welcome.

**[Add Git for source control]**

git-scm.com

.gitignore 🡪under # dependencies, modules to source control

Create a repository in DatinApp: >git init

Create .gitignore file under DatingAPp.API and edit it to exclude .vscode, bin, obj, \*.db.

.gitignore under DatingApp-SPA is already set to exclude dependencies.

**Commit files to Source control:**

Set global username/email configuration:

>git config --global user.name “First\_name Last\_name”  
>git config –-global user.email “myname@example.com”

Stage changes, clicking on +.

Execute “Initial Commit”. Changes are stored locally.

Create an account in github.com

Create new repository: DatingApp, Public

><https://github.com/marlonfdez/DatingApp.git>”

Source Control Menu 🡪 Select origin

Wait… while files are being transferred to Git

**SECURITY**

Hashing not so secure

Hash + Salt: increases difficulty. Same password does not result in the same hash.

**[Creating the user model]**

Create User class, properties: Id, Username, PasswordHash, PasswordSalt

Add DbSet<User> to DataContext.cs

Update db:

1. >dotnet ef migrations add AddedUserEntity
2. Check AddedUserEntity (Up method) under Migration folder
3. dotnet ef database update
4. Check db in qlite db browser if table was created

**[The repository pattern]**

Kestrel 🡪 Controller 🡪 DbContext 🡪 EF (level of abstraction) 🡪 DB  
Create Interface IAuthRepository.cs under Data  
Create class AuthRepository.cs inheriting Interface to implement methods

**[Login method]**

Implement login and register methods in AuthRepository.cs

**[Registering services in the Startup class]**

Add services.AddScoped<IAuthRepository, AuthRepository>(); It is created once per request within the scope. It is equivalent to a singleton but in the current scope itself.

**[Creating the Controller of Authentication]**

Create controller to inject repository and logic (user login)

class classname : ControllerBase

ControllerBase: base class for an MVC controller without view support  
Controller: base class for an MVC controller with view support

Create AuthController.cs

Inject repository into constructor

Create Task<IActionResult>Register(UserForRegisterDto userForRegisterDto) method

No need to use [FromBody] in the argument since attribute [ApiController] on top of class definition

**[Data transfer object]**

Create Dtos Folder and UserForRegisterDto.cs

It is a class with Username and Password, which is passed as a parameter to the Register method of AuthController in json format.

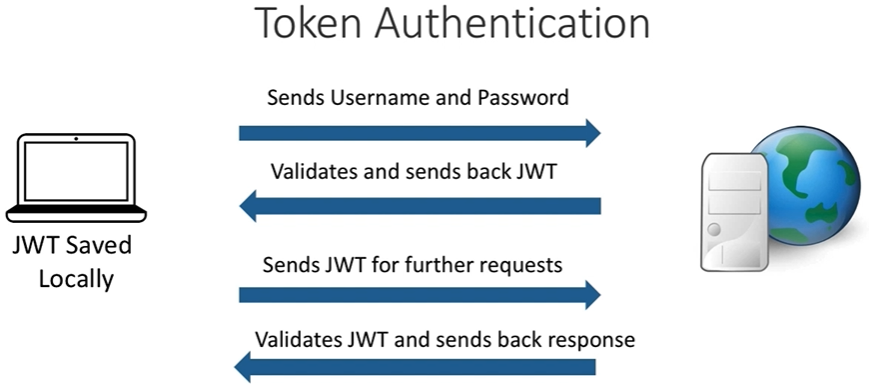
Validate User through the use of [Required] and [StringLength()] attributes. AuthController must have [ApiController] on top of class in order to handle the error: 500 Internal Server Error and inform the user that username and password are required.

If [ApiController] is not used then we must use [FromBody] inside the register method and add ModelState condition to the method, otherwise dto would return empty string and not a null value for Username causing the line with “.ToLower()” to be executed and the empty user be created.

[ApiController] better approach since it gives more details about the error.

**[Token authentication]**

Json Web Tokens: JWT



If using .Net core 3.0 install adicional packages:

Press F1 to open Command palette:

>NuGet Package Manager: Add Package  
 Micorsoft.IdentityModel + CR 🡪 Microsoft.IdentityModel.Tokens (last version)  
 System.IdentityModel.Tokens.Jwt (last version)  
 Microsoft.AspNetCore.Authentication + CR 🡪 .JwtBeare, version = runtime version (3.1.0)

When popup message appears regarding unresolved dependency, click on Restore.

Create class, UserForLoinDto in Dtos and write correspondent codes

**[Authentication Middleware]**

>NuGet Package Manager: Add Package  
 Microsoft.AspNetCore.Authentication + CR 🡪 .JwtBeare, version = runtime version (2.2.0)

Add Authentication as a service in ConfigurreService method of Startup class

Add app.UserAuthentication() before app.UseMvc().

Add [Authorize] attribute on controller. [AllowAnonymous] attribute before a method: access without token