Updating to .Net 7 and Angular 14

Installing the updated SDKs/CLIs

Install .Net 7 SDK from https://dotnet.microsoft.com/en-us/download

Install Angular 14 Cli by running:

```
npm install -g @angular/cli@14
```

Make sure both the .Net application AND the angular application is running without errors or warnings.

Updating to .Net 7

Need to update project to use .Net 7.0 and update any EF packages to the same

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net7.0</TargetFramework>
    <Nullable>disable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="AutoMapper.Extensions.Microsoft.DependencyInjection" Version="12.0" />
    <PackageReference Include="CloudinaryDotNet" Version="1.20.0" />
    <PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="7.0.0" />
    <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="7.0.0" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="7.0.0">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="7.0.0" />
    <PackageReference Include="System.IdentityModel.Tokens.Jwt" Version="6.15.1" />
  </ItemGroup>

</Project>
```

Execute the following commands:

```
dotnet clean
dotnet restore
dotnet build

// If you have any kind of SSL related error:
dotnet dev-certs https --clean
dotnet dev-certs https --trust

// run the app if no errors
dotnet watch —no-hot-reload
```

Updating to the “Minimal hosting model”

If you have a Startup.cs in your project then we will remove this and update the Program.cs to use the minimal hosting model.

Directly underneath the using statements create the following line:

```
using System;
using System.Threading.Tasks;
using API.Data;
using API.Entities;
using API.Extensions;
using API.Middleware;
using API.SignalR;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

var builder = WebApplication.CreateBuilder(args);
```

Below this copy your services from the ConfigureServices() method in the Startup.cs class (whatever you currently have) into the Program.cs class and adjust services.Whatever to builder.Services.Whatever.   This example is from the end of section 18 so if you are earlier on in the course you will have less services:

```
var builder = WebApplication.CreateBuilder(args);

// services container

builder.Services.AddApplicationServices(builder.Configuration);
builder.Services.AddControllers();
builder.Services.AddCors();
builder.Services.AddIdentityServices(builder.Configuration);
builder.Services.AddSignalR();
```

Below the services above we add the following line for our “Middleware”

```
// middleware

var app = builder.Build();
```

Then copy everything inside the Startup.cs Configure() method and adjust the endpoints to work with the new hosting model. Again this is an example from the end of section 18:

```
// middleware

var app = builder.Build();

app.UseMiddleware<ExceptionMiddleware>();

app.UseHttpsRedirection();

app.UseRouting();

app.UseCors(x => x.AllowAnyHeader()
    .AllowAnyMethod()
    .AllowCredentials()
    .WithOrigins("https://localhost:4200"));

app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();
app.MapHub<PresenceHub>("hubs/presence");
app.MapHub<MessageHub>("hubs/message");
```

If you have the code inside your Program.cs to initialize the database then add this below the middleware which should look like the following:

```
using var scope = app.Services.CreateScope();
var services = scope.ServiceProvider;
try
{
    var context = services.GetRequiredService<DataContext>();
    var userManager = services.GetRequiredService<UserManager<AppUser>>();
    var roleManager = services.GetRequiredService<RoleManager<AppRole>>();
    await context.Database.MigrateAsync();
    await Seed.SeedUsers(userManager, roleManager);
}
catch (Exception ex)
{
    var logger = services.GetRequiredService<ILogger<Program>>();
    logger.LogError(ex, "An error occurred during migration");
}
```

Finally, add the method to run the app at the bottom:

```
app.Run();
```

The final code for the updated Program class should look as follows (as at the end of section 18):

```
using API.Data;
using API.Entities;
using API.Extensions;
using API.Middleware;
using API.SignalR;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// services container

builder.Services.AddApplicationServices(builder.Configuration);
builder.Services.AddControllers();
builder.Services.AddCors();
builder.Services.AddIdentityServices(builder.Configuration);
builder.Services.AddSignalR();

// middleware

var app = builder.Build();

app.UseMiddleware<ExceptionMiddleware>();

app.UseHttpsRedirection();

app.UseRouting();

app.UseCors(x => x.AllowAnyHeader()
    .AllowAnyMethod()
    .AllowCredentials()
    .WithOrigins("https://localhost:4200"));

app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();
app.MapHub<PresenceHub>("hubs/presence");
app.MapHub<MessageHub>("hubs/message");

using var scope = app.Services.CreateScope();
var services = scope.ServiceProvider;
try
{
    var context = services.GetRequiredService<DataContext>();
    var userManager = services.GetRequiredService<UserManager<AppUser>>();
    var roleManager = services.GetRequiredService<RoleManager<AppRole>>();
    await context.Database.MigrateAsync();
    await Seed.SeedUsers(userManager, roleManager);
}
catch (Exception ex)
{
    var logger = services.GetRequiredService<ILogger<Program>>();
    logger.LogError(ex, "An error occurred during migration");
}

app.Run();
```

Delete the Startup.cs file from your project.

Run the following commands and make sure the .Net app starts without errors:

```
dotnet build
dotnet run
```

Using the DateOnly for the DateOfBirth

We can now use a DateOnly for things that we only need a Date rather than a DateTime for.   This has been implemented in the updated course.

Update the DateOfBirth property in the AppUser class to use it:

```
public class AppUser
{
    // omitted other properties
    public DateOnly DateOfBirth { get; set; }
```

Update the DateTimeExtensions class:

```
using System;

namespace API.Extensions
{
    public static class DateTimeExtensions
    {
        public static int CalculateAge(this DateOnly dob)
        {
            var today = DateOnly.FromDateTime(DateTime.UtcNow);
            var age = today.Year - dob.Year;
            if (dob > today.AddYears(-age)) age--;
            return age;
        }
    }
}
```

Correct the errors in the UserRepository by updating the following lines:

```
var minDob = DateOnly.FromDateTime(DateTime.Today.AddYears(-userParams.MaxAge - 1));
var maxDob = DateOnly.FromDateTime(DateTime.Today.AddYears(-userParams.MinAge));
```

Run the following commands as you will need to drop and recreate the database so the seeded users have the correct type of date in the DB:

```
dotnet ef database drop
dotnet run
```

Make sure there are no errors and the app is running correctly.

End of .Net updates.

Updating Angular to Angular 14

We are basically following the guide here from here:

https://update.angular.io/?v=12.0-14.0

The steps we need to take are as follows:

1. Before we can run the update command we need to commit our changes into GitHub so run the following in the solution folder (we have to go to Angular 13 first then Angular 14)

```
git add .
git commit -m "updating to angular 13"
```

Then run the command to update to Angular 13.   We will need to use the force switch as our other packages will cause a dependancy error if we do not.

```
ng update @angular/core@13 @angular/cli@13 --force
```

Once complete, add another commit:

```
git add .
git commit -m "updating to angular 13"
```

Then we can update to Angular 14, again with the force switch:

```
ng update @angular/core@14 @angular/cli@14
```

Then we can update our other packages to the latest (working) versions using the following command:

```
npm install @angular/cdk@14 @kolkov/ngx-gallery@latest @microsoft/signalr@latest
ngx-bootstrap@latest ngx-spinner@latest ngx-timeago@latest ngx-toastr@15.2.2
rxjs@latest tslib@latest ng2-file-upload@next --force
```

Once that has run update the dev packages using this command:

```
npm install @types/jasminewd2@latest @types/node@latest codelyzer@latest karma-chrome-launcher@latest
karma-jasmine@latest karma-jasmine-html-reporter@latest --save-dev --force
```

We also need to add some extra css for the updated version of ngx-spinner so update the angular.json file styles array with the following:

```
"node_modules/ngx-spinner/animations/line-scale-party.css",
```

You should now be able to restart the app by running the following command and it should be working fine:

```
ng serve
```

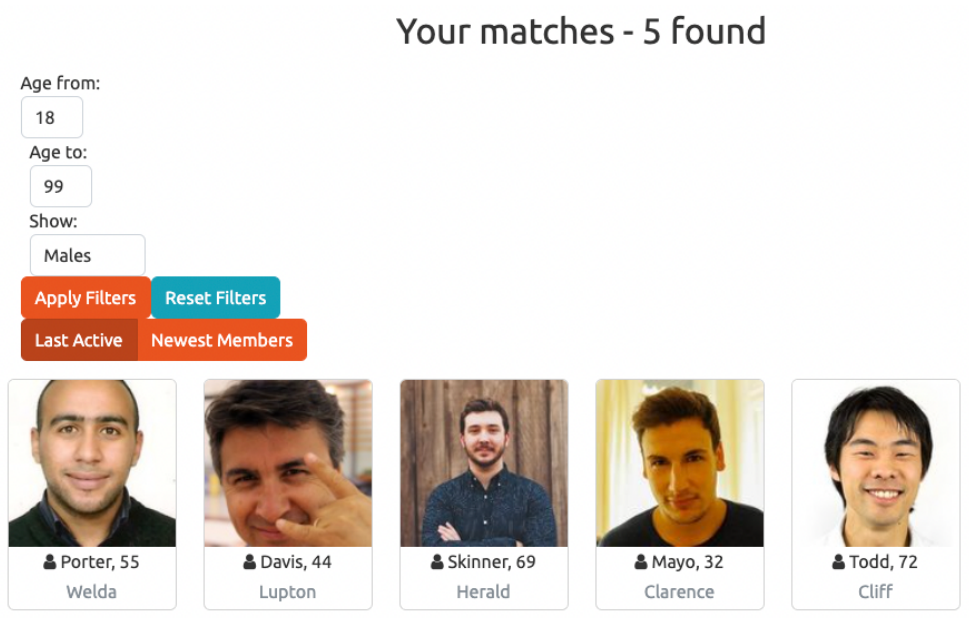
Updating to Bootstrap 5

This one is a little bit of work

Run the following command in the client folder:

```
npm install bootstrap@5 bootswatch@5 --force
```

Check the app and you will find its a bit of a mess.


Your matches - 5 found

Age from: 18

Age to: 99

Show: Males

A lot of the changes we need to make are quite minor though as they are just renaming existing classes. Right-click the app folder inside the client folder and use the “find in folder” function and use **find and replace all** for the following:

Rename anything that starts with ‘ml-’ to ‘ms-’

Rename anything that starts with ‘mr-’ to ‘me-’

Rename ‘form-inline’ to ‘d-flex’

Rename ‘form-group’ to ‘d-flex’

Rename ‘radio-inline’ to ‘form-check-label’

Rename ‘float-right’ to ‘float-end’

This gets us most of the way there and there are a few more specific changes we need to make.

In **register.component.html** give inputs a class of ‘form-check-input’

```
<div class="mb-3">
    <label class="control-label" style="margin-right: 10px;">I am a: </label>
    <label class="form-check-label">
        <input class="form-check-input" type="radio" value='male' formControlName='gender'> Male
    </label>
    <label class="form-check-label">
        <input class="form-check-input ms-3" type="radio" value='female' formControlName='gender'> Female
    </label>
</div>
```

Also, in the **register.component.html** for the buttons use ‘text-center’ and remove any other style.

```
<div class="text-center">
    <button [disabled]='!registerForm.valid' class="btn btn-success me-2" type="submit">Register</button>
    <button class="btn btn-default me-2" (click)="cancel()" type="button">Cancel</button>
</div>
```

In the **member-list.component.html** change the second div to use ‘d-flex’

```
<div class="text-center mt-3">
    <h2>{{predicate === 'liked' ? 'Members I like' : 'Members who like me'}}</h2>
</div>

<div class="container mt-3">
    <div class="d-flex">
```

In the **messages.component.html** do the same

```
<div class="d-flex mb-4">
    <div class="btn-group" name='container'>
        <button class="btn btn-primary" btnRadio='Unread' [(ngModel)]="container"
            (click)="loadMessages()">
            <i class="fa fa-envelope"></i> Unread
        </button>
        <button class="btn btn-primary" btnRadio='Inbox' [(ngModel)]="container"
            (click)="loadMessages()">
            <i class="fa fa-envelope-open"></i> Inbox
        </button>
        <button class="btn btn-primary" btnRadio='Outbox' [(ngModel)]="container"
            (click)="loadMessages()">
            <i class="fa fa-paper-plane"></i> Outbox
        </button>
    </div>
</div>
```

In the nav.component.html add the text-decoration-none class to the a tag where we have the welcome user:

```
<a class="dropdown-toggle text-light ms-2 text-decoration-none " dropdownToggle>Welcome {{user.knownAs || user.username | titlecase}}<
            <div class="dropdown-menu mt-2" *dropdownMenu>
```

Enabling strict mode (Optional)

The updated version of the course uses strict mode all the way through. When this course was first created (using Angular 2) one of the selling points of Angular was that we “could use as much or as little TypeScript as we want!”. At the time of Angular 2 release a lot of developers were not keen on TypeScript and the Angular team did not want to put them off using Angular (that was written in and uses TypeScript) so there was no such thing as “Strict mode” when I initially created the code. Since Angular 9 strict mode was “opt in” and from Angular 10 it was “opt out”. For this reason a lot of the code in the original version of the course was not written with strict mode in mind.

If you have been following the demos with Strict mode turned off (as per my recommendation) then turning on strict mode could cause your app to start reporting a LOT of errors so please take note of the following further guidance:

1. You do not need to turn on strict mode. As the old saying goes “if it is not broken, don’t fix it”.
2. You can keep following the course from where you are without any problems. The updated code uses strict mode but this works perfectly well if you do not have strict mode turned on.

If you are still here and you want to give it a go the first thing to do is run the following command:

```
git add .
git commit -m "About to break a working app"
```

This will give you a place to go back to when you see the number of errors you have to deal with and decide its not really necessary as per points 1 and 2 above.

Overwrite your tsconfig.json file with the following:

```
/* To learn more about this file see: https://angular.io/config/tsconfig. */
{
  "compileOnSave": false,
  "compilerOptions": {
    "baseUrl": "./",
    "outDir": "./dist/out-tsc",
    "forceConsistentCasingInFileNames": true,
    "strict": true,
    "noImplicitOverride": true,
    "noPropertyAccessFromIndexSignature": true,
    "noImplicitReturns": true,
    "noFallthroughCasesInSwitch": true,
    "sourceMap": true,
    "declaration": false,
    "downlevelIteration": true,
    "experimentalDecorators": true,
    "moduleResolution": "node",
    "importHelpers": true,
    "target": "es2020",
    "module": "es2020",
    "lib": [
      "es2020",
      "dom"
    ]
  },
  "angularCompilerOptions": {
    "enableI18nLegacyMessageIdFormat": false,
    "strictInjectionParameters": true,
    "strictInputAccessModifiers": true,
    "strictTemplates": true
  }
}
```

Your app is now broken and there are errors to fix 🙂 .

Restart the app and have a look at the terminal to see how many errors there are (brace yourself - there could be a lot).

VS Code by default does not report TypeScript errors in the problems area. We can enable this functionality though by updating our VS Code settings (you will need to get the JSON settings in VS Code to do this). Open the settings from preference as normal and at the top right there is a button to go to the JSON settings. Add the following line:

```
"typescript.tsserver.experimental.enableProjectDiagnostics": true,
```

This will then report all the errors in the problems area in VS Code (you may need to restart VS Code here AND open an angular component file and it's template) and then each problem can be fixed one by one.   There were 77 strict mode related errors when I tested this from the code at the end of Section 18.   There are 4 types of errors that you will encounter.   In the video I show how to resolve each type of error so you can do the same in your project.

Property 'thing' has no initializer and is not definitely assigned in the constructor.

This is probably the most common error when enabling strict mode as we do not need to initalise class properties when not using strict mode.   These are also very easy to fix.

- If the property is an array just set it to an empty array initially:

```
export class HasRoleDirective implements OnInit {
  @Input() appHasRole: string[] = [];
```

- If the property is a string just set it to an empty string:

```
export class TextInputComponent implements ControlValueAccessor {
  @Input() label = '';
```

- If the property is an object then make it optional:

```
export class HasRoleDirective implements OnInit {
  @Input() appHasRole: string[];
  user?: User;
```

- If the property is a boolean give it the initial value of false (the default)

```
export class ConfirmDialogComponent implements OnInit {
  title = '';
  message = '';
  btnOkText = '';
  btnCancelText = '';
  result = false;
```

Type 'AbstractControl<any, any> | null' is not assignable to type 'FormControl<any>'.\n Type 'null' is not assignable to type 'FormControl<any>'.

In our re-usable components we see this one. To get around this we need to cast the Abstract 'ngControl' we are using in our components to a FormControl

Create a method in the component:

```
get control(): FormControl {
    return this.ngControl.control as FormControl
  }
```

Then update the template to use 'control' instead of ngControl

```
<div class="mb-3">
    <input
        [class.is-invalid]="ngControl.touched && ngControl.invalid"
        type={{type}}
        class="form-control"
        [formControl]="control"
        placeholder={{label}}>
```

When we use this in the RegisterForm we need to also cast the control we are using e.g:

Property 'thing' comes from an index signature, so it must be accessed with ['thing']

We see these types of errors in our template and instead of accessing the property with dot notation, we need to wrap in square brackets and quotes.   So the following code:

```
<div *ngIf="control.errors?.required" class="invalid-feedback">Please enter a {{label}}</div>
```

Changes to:

```
<div *ngIf="control.errors?.['required']" class="invalid-feedback">Please enter a {{label}}</div>
```

Parameter 'pageNumber' implicitly has an 'any' type

If we do not specify a type in the parameters of a method we see this error.   We just need to specify the type so this code:

```
getLikes(predicate: string, pageNumber, pageSize) {
    let params = getPaginationHeaders(pageNumber, pageSize);
    params = params.append('predicate', predicate);
    return getPaginatedResult<Partial<Member[]>>(this.baseUrl + 'likes', params, this.http);
  }
```

Needs to be updated to:

```
getLikes(predicate: string, pageNumber: number, pageSize: number) {
    let params = getPaginationHeaders(pageNumber, pageSize);
    params = params.append('predicate', predicate);
    return getPaginatedResult<Partial<Member[]>>(this.baseUrl + 'likes', params, this.http);
  }
```

Type 'Observable<true | undefined>' is not assignable to type 'Observable<boolean>’

In our guards we are not returning false, if the if condition is not met.   So this code:

```
export class AdminGuard implements CanActivate {
  constructor(private accountService: AccountService, private toastr: ToastrService) { }

  canActivate(): Observable<boolean> {
    return this.accountService.currentUser$.pipe(
      map(user => {
        if (user.roles.includes('Admin') || user.roles.includes('Moderator')) {
          return true;
        }
        this.toastr.error('You cannot enter this area');
      })
    )
  }
}
```

Would become:

```
export class AdminGuard implements CanActivate {
  constructor(private accountService: AccountService, private toastr: ToastrService) { }

  canActivate(): Observable<boolean> {
    return this.accountService.currentUser$.pipe(
      map(user => {
        if (user.roles.includes('Admin') || user.roles.includes('Moderator')) {
          return true;
        } else {
          this.toastr.error('You cannot enter this area');
          return false;
        }
      })
    )
  }
}
```

Variable 'currentUser' is used before being assigned

This is a mistake in the jwt.interceptor.   Currently we have:

```
this.accountService.currentUser$.pipe(take(1)).subscribe(user => currentUser = user);
    if (currentUser) {
      request = request.clone({
        setHeaders: {
          Authorization: `Bearer ${currentUser.token}`
        }
      })
    }
```

We should get this in the subscribe method instead:

```
intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    this.accountService.currentUser$.pipe(take(1)).subscribe({
      next: user => {
        if (user) {
          request = request.clone({
            setHeaders: {
              Authorization: `Bearer ${user.token}`
            }
          })
        }
      }
    })
```

Argument of type 'string | null' is not assignable to parameter of type 'string’

Another common error when using strict mode. If we use a method that potentially returns undefined then we see this.   In the member-detailed.resolver.ts there is an example of this here:

```
resolve(route: ActivatedRouteSnapshot): Observable<Member> {
        return this.memberService.getMember(route.paramMap.get('username'));
    }
```

To get around this we need to check we have the ‘thing’ before we attempt to use it:

```
export class MemberDetailedResolver implements Resolve<Member | null> {

    constructor(private memberService: MembersService) {}

    resolve(route: ActivatedRouteSnapshot): Observable<Member | null> {
        const username = route.paramMap.get('username');
        if (username)
            return this.memberService.getMember(username);
        else return of(null);
    }
}
```

Argument of type 'OperatorFunction<User, void>' is not assignable to parameter of type 'OperatorFunction<Object, void>’

In our http methods we need to specify the type of thing we get back from the API.   So in the account.service.ts we have:

```
login(model: any) {
    return this.http.post(this.baseUrl + 'account/login', model).pipe(
      map((response: User) => {
        const user = response;
        if (user) {
          this.setCurrentUser(user);
          this.presence.createHubConnection(user);
        }
      })
    )
  }
```

This can be resolved by adding the type to the post method:

```
login(model: any) {
    return this.http.post<User>(this.baseUrl + 'account/login', model).pipe(
      map(response => {
        const user = response;
        if (user) {
          this.setCurrentUser(user);
          this.presence.createHubConnection(user);
        }
      })
    )
  }
```

Argument of type 'null' is not assignable to parameter of type 'User'.

In the account.service we are using ‘null’ if we do not have a user in this line:

```
logout() {
    localStorage.removeItem('user');
    this.currentUserSource.next(null);
    this.presence.stopHubConnection();
  }
```

To correct this we need to tell the Observable that it can either be a User or it can be ‘null’.

```
export class AccountService {
  baseUrl = environment.apiUrl;
  // note: in the course this has been updated to use BehaviorSubject rather than ReplaySubject
  private currentUserSource = new BehaviorSubject<User | null>(null);
  currentUser$ = this.currentUserSource.asObservable();
```

Alternatively, if you would prefer to go back to just having a working application then you can run the following command (as long as you committed your changes as per above) then you can run the following command:

```
git stash
```

This will remove the strict mode configuration and you can go back to the working application you had before enabling strict mode 😀