Update Guide to .Net 7 and React 18

Update to .Net 7

Note: The starting point for this example is from end of section 21. If you are at an earlier point then please only update what is relevant to your current project state.

1. Downloand and install the .Net 7 SDK from https://dotnet.microsoft.com
2. Update all .csproj files to target .Net 7.0 and update all packages to the latest current version.

API.csproj

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

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

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="7.0.0" NoWarn="NU1605" />
    <PackageReference Include="Microsoft.AspNetCore.Authentication.OpenIdConnect" Version="7.0.0" NoWarn="NU1605" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="7.0.0">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="Swashbuckle.AspNetCore" Version="5.6.3" />
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\Application\Application.csproj" />
    <ProjectReference Include="..\Infrastructure\Infrastructure.csproj" />
  </ItemGroup>

</Project>
```

Application.csproj

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

  <ItemGroup>
    <ProjectReference Include="..\Domain\Domain.csproj" />
    <ProjectReference Include="..\Persistence\Persistence.csproj" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="AutoMapper.Extensions.Microsoft.DependencyInjection" Version="12.0.0" />
    <PackageReference Include="FluentValidation.AspNetCore" Version="11.2.2" />
    <PackageReference Include="MediatR.Extensions.Microsoft.DependencyInjection" Version="11.0.0" />
  </ItemGroup>

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

</Project>
```

Domain.csproj

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

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

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="7.0.0" />
  </ItemGroup>

</Project>
```

Infrastructure.csproj

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

  <ItemGroup>
    <ProjectReference Include="..\Application\Application.csproj" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="CloudinaryDotNet" Version="1.20.0" />
  </ItemGroup>

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

</Project>
```

Persistence.csproj

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

  <ItemGroup>
    <ProjectReference Include="..\Domain\Domain.csproj" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="7.0.0" />
  </ItemGroup>

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

</Project>
```

3. Execute dotnet restore at the solution level to remove the errors.

```
dotnet restore
```

Fluent Validation changes:

The current method of using Fluent validation has been deprecated. Update the Startup project to the following:

```
public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllers(opt =>
            {
                var policy = new AuthorizationPolicyBuilder().RequireAuthenticatedUser().Build();
                opt.Filters.Add(new AuthorizeFilter(policy));
            });
            services.AddApplicationServices(_config);
            services.AddIdentityServices(_config);
        }
```

Then add the new config to the Application Services extensions:

```
services.AddFluentValidationAutoValidation();
services.AddValidatorsFromAssemblyContaining<Create>();

return services;
```

.Net 6.0 and onwards minimal API.

Move the services and middleware from the Startup.cs into the Program.cs. The updated Program.cs is:

```
using API.Extensions;
using API.Middleware;
using API.SignalR;
using Domain;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc.Authorization;
using Microsoft.EntityFrameworkCore;
using Persistence;

var builder = WebApplication.CreateBuilder(args);

// add services to the container

builder.Services.AddControllers(opt =>
{
    var policy = new AuthorizationPolicyBuilder().RequireAuthenticatedUser().Build();
    opt.Filters.Add(new AuthorizeFilter(policy));
});
builder.Services.AddApplicationServices(builder.Configuration);
builder.Services.AddIdentityServices(builder.Configuration);

// Configure the HTTP request pipeline

var app = builder.Build();

app.UseMiddleware<ExceptionMiddleware>();

if (builder.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "API v1"));
}

app.UseCors("CorsPolicy");

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

```
app.MapControllers();
app.MapHub<ChatHub>("/chat");

using var scope = app.Services.CreateScope();
var services = scope.ServiceProvider;

try
{
    var context = services.GetRequiredService<DataContext>();
    var userManager = services.GetRequiredService<UserManager<AppUser>>();
    await context.Database.MigrateAsync();
    await Seed.SeedData(context, userManager);
}
catch (Exception ex)
{
    var logger = services.GetRequiredService<ILogger<Program>>();
    logger.LogError(ex, "An error occured during migraiton");
}

app.Run();
```

We can now remove the Startup.cs from our project

Ensure the project runs correctly.

```
dotnet build
dotnet watch --no-hot-reload
```

If using Postgres as your DB Server

If you have completed the course and are using PostGres instead of Sqlite. The UTC dates are now being returned with the correct date format so if you have concatenated a ‘Z’ onto the date in the commentStore.ts this can now be removed from this code:

commentStore.ts

```
this.hubConnection.on('LoadComments', (comments: ChatComment[]) => {
    runInAction(() => {
        comments.forEach(comment => {
            comment.createdAt = new Date(comment.createdAt);
        });
        this.comments = comments;
    });
});
```

Update to React 18

Update the project dependancies to React 18

```
npm install react@latest react-dom@latest react-scripts@latest
```

We will also update our 3rd party packages as we will need to get the updated versions that work with React 18

Run npm outdated which gives us a list of outdated packages:

```
npm outdated
```

Will probably see something like this:

To update all the packages in one easy command we will run the following:

```
npm i -g npm-check-updates && ncu -u && npm i
```

The above command does the following:

1. Installs a package that checks updates for us
2. Then use this package to update all package versions in the package.json (-u is short for —updateAll)
3. Uses npm -i to then install all the updated versions of the packages

For your own projects *use this command with care* as it installs the major updates of the packages that could include breaking changes. We will have breaking changes in our app with React router and React 18 added a bit of strictness for us to deal with. We will go through the breaking changes and update our app. Let's see how we are getting on by running:

```
npm start
```

We will probably see lots of errors, especially related to React Router which introduced significant breaking changes we need to deal with.

React Router 6.4 Changes

1. Inside the app folder create a new folder called ‘router’
2. Create a new React component called Routes.tsx. We will add our routes inside this file. For every route you currently have. I’m attempting to update from the end of section 21 and the routes defined in my App.tsx are as follows:

```
return (
    <>
      <ToastContainer position='bottom-right' hideProgressBar />
      <ModalContainer />
      <Route exact path='/' component={HomePage} />
      <Route
        path={'/(.+)'}
        render={() => (
          <>
            <NavBar />
            <Container style={{ marginTop: '7em' }}>
              <Switch>
                <Route exact path='/activities' component={ActivityDashboard} />
                <Route path='/activities/:id' component={ActivityDetails} />
                <Route key={location.key} path={['/createActivity', '/manage/:id']} component={ActivityForm} />
                <Route path='/profiles/:username' component={ProfilePage} />
                <Route path='/errors' component={TestErrors} />
                <Route path='/server-error' component={ServerError} />
                <Route path='/login' component={LoginForm} />
                <Route component={NotFound} />
              </Switch>
            </Container>
          </>
        )}
      />
    </>
  );
```

We need to remove all these routes and replace it with an Outlet component we get from React Router, but first we will replicate these routes in the **Routes.tsx** file we have just created, and create a ‘router’ using the createBrowserRouter method we get from React-Router.

```
import { createBrowserRouter, Navigate, RouteObject } from "react-router-dom";
import ActivityDashboard from "../../features/activities/dashboard/ActivityDashboard";
import ActivityDetails from "../../features/activities/details/ActivityDetails";
import ActivityForm from "../../features/activities/form/ActivityForm";
import NotFound from "../../features/errors/NotFound";
import TestErrors from "../../features/errors/TestError";
import ProfilePage from "../../features/profiles/ProfilePage";
import App from "../layout/App";

export const routes: RouteObject[] = [
    {
        path: '/',
        element: <App />,
        children: [
            {path: 'activities', element: <ActivityDashboard />},
            {path: 'activities/:id', element: <ActivityDetails />},
            {path: 'createActivity', element: <ActivityForm key='create' />},
            {path: 'manage/:id', element: <ActivityForm key='manage' />},
            {path: 'profiles/:username', element: <ProfilePage />},
            {path: 'errors', element: <TestErrors />},
            {path: 'server-error', element: <TestErrors />},
            {path: 'not-found', element: <NotFound />},
            {path: '*', element: <Navigate replace to='/not-found' />},
        ]
    }
```

```
]

export const router = createBrowserRouter(routes);
```

Then in the App.tsx we use <Outlet /> which will basically replace this component with the component we route to detailed in the children of our Root element (App.tsx).

Update the App.tsx to use this. This is how my App.tsx looks after the change (as at end of section 21 so you may not have the exact same content here):

```
import { useEffect } from 'react';
import { Container } from 'semantic-ui-react';
import NavBar from './NavBar';
import { observer } from 'mobx-react-lite';
import { Outlet, useLocation } from 'react-router-dom';
import { ToastContainer } from 'react-toastify';
import { useStore } from '../stores/store';
import LoadingComponent from './LoadingComponent';
import ModalContainer from '../common/modals/ModalContainer';

function App() {
  const location = useLocation();
  const {commonStore, userStore} = useStore();

  useEffect(() => {
    if (commonStore.token) {
      userStore.getUser().finally(() => commonStore.setAppLoaded());
    } else {
      commonStore.setAppLoaded();
    }
  }, [commonStore, userStore])

  if (!commonStore.appLoaded) return <LoadingComponent content='Loading app...' />

  return (
    <>
      <ToastContainer position='bottom-right' hideProgressBar />
      <ModalContainer />
      <NavBar />
      <Container style={{ marginTop: '7em' }}>
        <Outlet />
      </Container>
    </>
  );
}

export default observer(App);
```

Now update the Index.tsx so we can use this router. We will also use the new Root API provided in React 18 as using ReactDOM.render is now deprecated in favour of ‘createRoot’.

To use this we need to change the import for ReactDOM to come from ‘react-dom/client’. The updated index.tsx file including the new RouterProvider we add should look like this:

```
import ReactDOM from 'react-dom/client';
import 'semantic-ui-css/semantic.min.css';
import 'react-calendar/dist/Calendar.css';
import 'react-toastify/dist/ReactToastify.min.css';
import 'react-datepicker/dist/react-datepicker.css';
import './app/layout/styles.css';
import reportWebVitals from './reportWebVitals';
import { store, StoreContext } from './app/stores/store';
import { RouterProvider } from 'react-router-dom';
import { router } from './app/router/Routes';

const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);
```

```
root.render(
  <StoreContext.Provider value={store}>
      <RouterProvider router={router} />
  </StoreContext.Provider>
)

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

Anywhere we used ‘history’ in our app to navigate outside the context of a React component will need to be updated to use our router object which has a navigate function. So the agent.ts will use this in the interceptor:

```
switch (status) {
    case 400:
        if (config.method === 'get' && data.errors.hasOwnProperty('id')) {
            router.navigate('/not-found');
        }
    // omitted
    case 404:
        router.navigate('/not-found');
        break;
    case 500:
        store.commonStore.setServerError(data);
        router.navigate('/server-error');
        break;
```

You will see that the info/documentation suggests not to use this but we are okay for what we are doing. The React Router devs have commented about this here if you want more info:

https://github.com/remix-run/react-router/issues/9422#issuecomment-1301182219

Also in our userStore we have used ‘history’ so remove that import and use our router to do this and replace any ‘history.push’ with ‘router.navigate’ in the following methods:

```
login = async (creds: UserFormValues) => {
        try {
            const user = await agent.Account.login(creds);
            store.commonStore.setToken(user.token);
            runInAction(() => this.user = user);
            router.navigate('/activities');
            store.modalStore.closeModal();
        } catch (error) {
            throw error;
        }
    }

    logout = () => {
        store.commonStore.setToken(null);
        window.localStorage.removeItem('jwt');
        this.user = null;
        router.navigate('/');
    }

    register = async (creds: UserFormValues) => {
        try {
            const user = await agent.Account.register(creds);
            store.commonStore.setToken(user.token);
            runInAction(() => this.user = user);
            router.navigate('/activities');
            store.modalStore.closeModal();
        } catch (error) {
            throw error;
        }
    }
```

This completes the router config.

Dealing with ‘strict’ type checking issues.

We still have a number of issues to resolve relating to TypeScript strictness. In the agent.ts file we see:

We can resolve this by checking we have config.headers before attempting to use it:

```
axios.interceptors.request.use(config => {
    const token = store.commonStore.token;
    if (token && config.headers) config.headers.Authorization = `Bearer ${token}`
    return config;
})
```

We also see a number of errors related to the use of ‘data’ in this same file with the error ‘Object type is unknown’.

We can resolve this by giving the response a type:

```
}, (error: AxiosError) => {
    const { data, status, config } = error.response as AxiosResponse;
    switch (status) {
        case 400:
            if (config.method === 'get' && data.errors.hasOwnProperty('id')) {
                router.navigate('/not-found');
            }
```

In my case I am now down to 4 errors in the following files:

1. PhotoWidgetDropzone
2. ActivityFilters
3. ActivityForm
4. ProfilePage.

PhotoWidgetDropzone:

```
const onDrop = useCallback(acceptedFiles => {
    setFiles(acceptedFiles.map((file: any) => Object.assign(file, {
        preview: URL.createObjectURL(file)
    })))
}, [setFiles])
const { getRootProps, getInputProps, isDragActive } = useDropzone({ onDrop })
```

This error is saying the accepted files has an implicit type of ‘any’. We will give it an explicit type of any to resolve this:

```
const onDrop = useCallback((acceptedFiles: any) => {
        setFiles(acceptedFiles.map((file: any) => Object.assign(file, {
            preview: URL.createObjectURL(file)
        })))
    }, [setFiles])
    const { getRootProps, getInputProps, isDragActive } = useDropzone({ onDrop })
```

ActivityFilters.tsx has the same issue so we will use the same resolution:

```
    <Header />
    <Calendar
        onChange={(date) => setPredicate('startDate', date as Date)}
        value={predicate.get('startDate') || new Date()}
    />
/>
```

```
<Calendar
                onChange={(date: any) => setPredicate('startDate', date as Date)}
                value={predicate.get('startDate') || new Date()}
            />
```

In the ActivityForm.tsx we have a ‘useHistory()’ hook that is no longer available. We can use the ‘useNavigate()’ hook instead.

```
export default observer(function ActivityForm() {
    const history = useHistory();
    const { activityStore } = useStore();
```

```
useEffect(() => {
        if (id) loadActivity(id).then(activity => setActivity(new ActivityFormValues(activity)))
    }, [id, loadActivity]);

    function handleFormSubmit(activity: ActivityFormValues) {
        if (!activity.id) {
            let newActivity = {
                ...activity,
                id: uuid()
            };
            createActivity(newActivity).then(() => navigate(`/activities/${newActivity.id}`))
        } else {
            updateActivity(activity).then(() => navigate(`/activities/${activity.id}`))
        }
    }
```

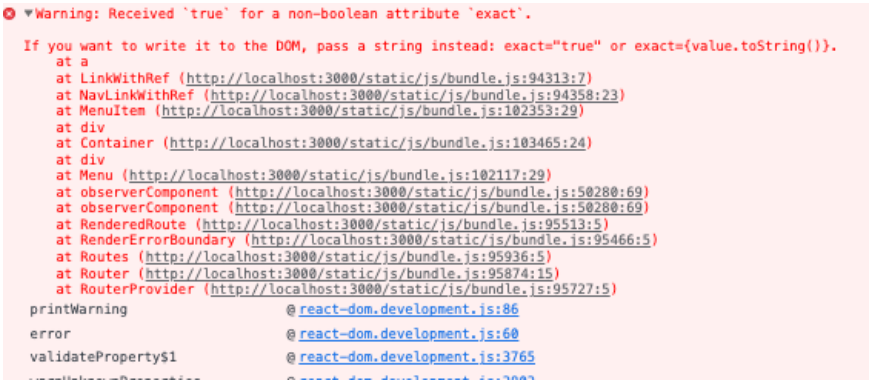
In the ProfilePage.tsx we have an error with the username stating it could possibly be undefined so we just need to check it exists first:

```
14
15      useEffect(() => {
16          loadProfile(username);
17          return () => {
18              setActiveTab(0);
19          }
20      }, [loadProfile, username, setActiveTab])
```

```
useEffect(() => {
        if (username) loadProfile(username);
        return () => {
            setActiveTab(0);
        }
    }, [loadProfile, username, setActiveTab])
```

Restart the React app and make sure the app is now working

The app starts but we still have some issues to solve:



In the NavBar we no longer need ‘exact’ as the router is a bit smarter now about working out which route is loaded so remove the exact attribute from this NavLink.

```
<Menu.Item as={NavLink} to='/' header>
    <img src='/assets/logo.png' alt='logo' style={{ marginRight: '10px' }} />
    Reactivities
</Menu.Item>
```

In order to move the HomePage outside of the NavBar we need to use a conditional in the App.tsx as we no longer have access to Regular expressions in our routes. This will do the job:

```
import { useEffect } from 'react';
import { Container } from 'semantic-ui-react';
import NavBar from './NavBar';
import { observer } from 'mobx-react-lite';
import { Outlet, useLocation } from 'react-router-dom';
import { ToastContainer } from 'react-toastify';
import { useStore } from '../stores/store';
import LoadingComponent from './LoadingComponent';
import ModalContainer from '../common/modals/ModalContainer';
```

```
import HomePage from '../../features/home/HomePage';

function App() {
  const location = useLocation();
  const { commonStore, userStore } = useStore();

  useEffect(() => {
    if (commonStore.token) {
      userStore.getUser().finally(() => commonStore.setAppLoaded());
    } else {
      commonStore.setAppLoaded();
    }
  }, [commonStore, userStore])

  if (!commonStore.appLoaded) return <LoadingComponent content='Loading app...' />

  return (
    <>
      <ToastContainer position='bottom-right' hideProgressBar />
      <ModalContainer />
      {location.pathname === '/' ? <HomePage /> : (
        <>
          <NavBar />
          <Container style={{ marginTop: '7em' }}>
            <Outlet />
          </Container>
        </>
      )}
    </>
  );
}

export default observer(App);
```

App should now be working and updated.