You pushed to **version3** 32 minutes ago

version 3
Bob Ghosh authored 33 minutes ago

d9125b12

Name	Last commit	Last update
docs	Create UserCreateTemplate.csv	2 hours ago
src	version 3	33 minutes ago
.gitignore	add gitignore	3 weeks ago
README.md	Update README.md	2 hours ago
package-lock.json	location auto	3 days ago

README.md

UBC-2020 Blueprint

This document is a project blueprint for the Associated Engineering Resource Management System project. The purpose of the doucment is to provide the teams with a starting point, and some guidance on the project structure.

Prerequisites

You'll need the following applications installed on your machine before getting started:

Application	Version	Usage
git	Any	for cloning repo
Visual Studio	Community Edition or Latest	for service
SQL Server Express	14.0.1000.169	for database
Node.js/NPM	Latest LTS Version	for client
Docker	2.2.0.5 (43884)	for publishing

Account	Usage
Microsoft Azure	for authentication

Set-up and Configuration Instructions

1. Clone the repo here
2. After the repo is cloned, navigate to the ../cpsc319-2020-winter-blueprint directory

```
cd cpsc319-2020-winter-blueprint
```

Authentication

Registering a new application on Azure

1. Log into Microsoft Azure
2. Navigate to 'portal'
3. Click on 'Azure Active Directory', click on 'App registrations', click 'New Registration' to create a new application
4. Give the application a name, under 'Supported account types' select 'Accounts in this organizational directory only (... only - Single tenant), under 'Platform configuration (Optional) select 'Client Application (Web, iOS, Android, Desktop+Devices)', then hit 'Register'

5. You should be redirected to the 'Authentication' page of the app your just registered, if not then navigate to the 'Authentication' page of your app
6. Under 'Platform configurations' click on 'Add a platform'
7. In the Configure Web screen, under 'Redirect URIs enter https://localhost:5001, leave the 'Logout URL' blank, under 'Implicit grant' check 'Access tokens' and 'ID tokens'; hit 'Configure' to save changes
8. After saving there will be a Web dropdown under 'Platform configurations', expand the dropdown and click 'Add URI'; and add http://localhost:3000 and hit the 'Save' button at the top

Configure Database and Service

Create local database

1. Open SQL Server Management Studio (SSMS)
2. Create a new local database instance (Right-click Databases>New Database...>give Database a name (eg. ResourceMGMT))

Prepare the database and populate

1. Navigate to the ../src/svc-dotnetcore3 directory

```
cd src/svc-dotnetcore3
```

2. Double click on svc-dotnetcore.sln to open up the Visual Studio solution, in solution explorer you will find three projects (Database, Tests, Web.API)
3. Right click on the Database project and select 'Publish...'
4. In the Publish Database window select 'Edit...'
5. Select the 'Browse' tab and enter the server name (same as the one in SSMS), using the dropdown menu, select the database instance in (SMSS), click on 'Test Connection' and ensure the connection is successful, click 'OK' once the test connection is successful
6. In the Publish Database window click on 'Advanced...'
7. In the 'General' tab under the 'Deployment Behavior' section, 'Uncheck Block incremental deployment if data loss might occur' and click 'OK' to confirm settings
8. In the Publish Database window, click on 'Save Profile As...', enter the File name as Database.DEV.publish.xml, ensure the file path pointing at the Database project in Visual Studios before hitting 'Save'
9. In the Publish Database window click 'Publish'
10. In the Database project open up the seed_corrected.sql file (scripts>seed_corrected.sql) and copy the SQL statements
11. Navigate to (SMSS) and create a 'New Query' in your database instance
12. Paste the SQL statments from the seed.sql and execute the statements to populate the database; some duplicated usernames may not copy over, this is fine

Configure the service

1. In the Web.API project, create a copy of appsettings.json and name it appsettings.Development.json
2. Edit appsettings.Development.json, in the ConnectionString replace the Data Source with the server name (same server name in SMSS), replace the Initial Catalog with the database name (same as database name in SMSS), and add http://localhost:3000 to AllowedOrigins, also populate the AzureAd Tenant and ClientId with the ID's from the app registration on azure, and save the changes to the file

```
"ConnectionString": "Data Source=server_name; Initial Catalog=database_name; Integrated Security=True;"
"AllowedOrigins": "http://localhost:3000",
"AzureAd": {
  "Instance": "https://login.microsoftonline.com/",
  "TenantId": "<Directory (tenant) Id>",
  "ClientId": "<Application (client) Id>"
}
```

3. Run the Web.API project (default might be IIS Express in the debugging options), click on the (green play button button dropdown and change to Web.API); to see more detail about the port settings go to the Web.API project and go to properties > launchSettings.json. Feel free to make any changes to the ports but make sure the changes are echoed in the Azure Active Directory ports as well.
4. Visit https://localhost:5001/users/, https://localhost:5001/projects/, https://localhost:5001/locations/ you should be getting a 401 unauthorized, this is expected
5. To ensure the database is connected, navigate to the LocationsController (Web.API>Controllers>LocationControllers) and comment out the [Authorize] attribute, now run the Web.API project again and visit https://localhost:5001/locations/, you should be getting the locations data back; the same can be done to test for the other controllers

Configure the Main Client

1. Navigate to ../cpsc319-2020-winter-blueprint/src/ui-react-client
2. Run npm install

```
npm install
```

3. Open up ../ui-react-client using a text editor or IDE of your choice
4. Add an .env file to the root ui-react-client and paste in the following (fill in the application and directory id from your app in azure, fill in the production service url before pushing to production)

```
REACT_APP_SVC_ROOT = <Your production service URL>

REACT_APP_CLIENT_ID = <Application (Client) Id>
REACT_APP_TENANT_ID = <Directory (Tenant) Id>
```

5. Make a copy of the .env file and rename it to .env.development, paste in the following (fill in the application and directory id from your app in azure)

```
REACT_APP_SVC_ROOT = https://localhost:5001/

REACT_APP_CLIENT_ID = <Application (Client) Id>
REACT_APP_TENANT_ID = <Directory (Tenant) Id>
```

6. Run npm start to run the client application

```
npm start
```

   You should be prompted with a login page, and after logging in you should have access to the application. The Users, Projects, and Locations navigation should be giving a failedError: Network Error if your service isn't running

Configure the Admin Client

1. Navigate to ../cpsc319-2020-winter-blueprint/src/admin-react-client
2. Run npm install

```
npm install
```

3. Open up ../ui-react-client using a text editor or IDE of your choice
4. Add an .env file to the root admin-react-client and paste in the following (fill in the application and directory id from your app in azure, fill in the production service url before pushing to production)

```
REACT_APP_SVC_ROOT = <Your production service URL>

REACT_APP_CLIENT_ID = <Application (Client) Id>
REACT_APP_TENANT_ID = <Directory (Tenant) Id>
```

5. Make a copy of the .env file and rename it to .env.development, paste in the following (fill in the application and directory id from your app in azure)

```
REACT_APP_SVC_ROOT = https://localhost:5001/

REACT_APP_CLIENT_ID = <Application (Client) Id>
REACT_APP_TENANT_ID = <Directory (Tenant) Id>
```

6. Run npm start to run the client application

```
npm start
```

   You should be prompted with a login page, and after logging in you should have access to the application. The Users, Projects, and Locations navigation should be giving a failedError: Network Error if your service isn't running

Run Full Stack (service and client)

1. Run the service Web.API project in Visual Studio
2. Run the main front end, navigate to localhost:3000 if browser does not open page up on its own

```
npm start
```

3. Will be prompted with a microsoft login / permissions page, enter the necessary login credentials (if you're already logged in then it will take you directly to the main page of the application)
4. Ctrl+C when done with the application.
5. Run the admin front end, navigate to localhost:3000 if browser does not open page up on its own

```
npm start
```

6. Login with Username: "turtle", and password: "319".

- Note: The projects and users endpoints may take longer to load. The admin and the main client both run at PORT = 3000 on local systems, so that we don't have to configure the backend system everytime.

Tests

Service

1. Navigate to the 'Test Explorer'
2. Select test(s) to run

Clients

1. Type npm test to run all tests

Dockerfile For Publishing

1. In the Solution Explorer, right click on Web.API and then Add->Docker Support...
2. Azure App Services only support DotNetCore 3.1 for Linux, so select Linux Container.
3. A Dockerfile should be created inside the svc-dotnetcore3 folder containing

```
FROM mcr.microsoft.com/dotnet/core/aspnet:3.1-buster-slim AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/core/sdk:3.1-buster AS build
WORKDIR /src
COPY ["svc-dotnetcore3/Web.API.csproj", "svc-dotnetcore3/"]
RUN dotnet restore "svc-dotnetcore3/Web.API.csproj"
COPY . .
WORKDIR "/src/svc-dotnetcore3"
RUN dotnet build "Web.API.csproj" -c Debug -o /app/build

FROM build AS publish
RUN dotnet publish "Web.API.csproj" -c Debug -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "Web.API.dll"]
```

4. Make sure to change .env.development and .env in the client folders accordingly.

Adding Users to the Active Directory for Authentication

There is a Template File for Adding Bulk Users to the AD in the Docs folder. The file is called "UserCreateTemplate.csv".