Exercises will refer to my week 7 GitHub repo which is at **[https://github.com/oit-gaden/Web-Development-2020-Winter/tree/master/Week7 (Links to an external site.)](https://github.com/oit-gaden/Web-Development-2020-Winter/tree/master/Week7)**

Exercise 1 - Set up Postgres

1. Pull the Docker image for Postgres:

   ```
   docker pull postgres
   ```

2. Pull the Docker image for the Postgres admin tool:

   ```
   docker pull dpage/pgadmin4
   ```

Exercise 2 - Create the database with data

1. Create a Docker volume for storing the database data across container restarts by running the command:

   ```
   docker volume create --name=db_data
   ```

2. Make a week 7 lab folder by copying your week 6 folder. This will take a bit of time as the node_module folder contents is quite large.

3. If you open your week 7 folder in VSCode and you see the following dialog at the bottom right of VSCode then answer "Yes":

4. Create a folder called database inside the lab folder.

5. Copy docker-compose.yml, student.sql and student-data.sql files from my week7/database folder to your database folder.

6. Start the Postgres container by opening a cmd/terminal window into the database folder and running the command:

   ```
   docker-compose up -d
   ```

7. Run the command:

   ```
   docker ps
   ```

   to see what port the container is listening on. It should be 5432.

8. Run the command:

   docker logs dbserver

   If the database starts successfully you should see a message a the end of the logs saying it is ready to receive connections.

9. Start up **pgAdmin** Docker container**:**

   docker run -p 8000:80 -e PGADMIN_DEFAULT_EMAIL=user@domain.com -e PGADMIN_DEFAULT_PASSWORD=SuperSecret -d --name pgadmin dpage/pgadmin4

10. Open **pgAdmin** by accessing it through a browser with the address:

    http://[docker-machine ip]:8000

    You should see a login screen where you will login with:

    username: user@domain.com
    password: SuperSecret

    This will then display the following interface:

11. Right click on "Servers" in the left pane and select "Create -> Server". Give the server any name you want. For hostname/address in the "Connection" tab enter your **Docker IP address** . Set the **Username** to "**postgres**".

Click on "Save". pgAdmin should now connect to the Postgres database in your Docker container.

12. Navigate down the tree from "Servers" to where you see "Databases" as shown here:

13. Right click on **"Databases"** and select **"Create -> Database"**. Name the database **"School"**. Click on "Save".

14. Click on the **"School"** database and **pgAdmin** should now "connect" to the "School" database.

15. Select **"Query Tool"** from the "Tools" menu.

16. Paste in the content from the **student.sql** script and click on the button with the forward symbol (hovering over it reveals "Execute/Refresh") in the toolbar just above the query window to execute the create table script.

17. Navigate down **School/schemas/public/Tables** to see the newly created student table.

18. Paste in the content from the student-data.sql script in the query window you have open to populate the student table with data. Feel free to modify the data if you want before executing the script. Query the data from the query window to make sure data has been inserted:

    select * from student;

Exercise 3 - Modify your REST service to now access the database

1. In a cmd/terminal window go to the webapi folder and run these commands:

    dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL

    This will install the Entity Framework for Postgres package.

2. Replace the your **Startup.cs** file with the one in my week7/webapi folder. Change **ECommerceDatabase** to **SchoolDatabase** in Startup.cs.

3. Replace your webapi/Database folder with the one from my week7/webapi folder and change the code that I have for product under Entities to reflect student. The Table and Column attributes refer to columns in the database you created. You can find those names from the script you used to create the student database table. Change the name of the **DBContext** class from ECommerceContext to SchoolContext. You'll need to change

it anywhere else it is referenced. You'll also need to change the contents of SchoolContext to reflect your entity, student.

4. Edit the code for your student controller. The only method you need to implement is the one to get all of the entities from the database. Follow the pattern from my ProductController for retrieving the student data from the database.

5. Add the ConnectionStrings setting to appSettings.json file like this:

   Change **ECommerceDatabase** to **SchoolDatabase, ecommerce** to **School** and **localhost** to the docker-machine IP address.

6. Make sure your Postgres container is still running. Start your webapi by going to the Debug menu in VSCode and selecting **"Start without debugging"**.

   Use Postman to attempt to retrieve your students by using the URL:

   http://[docker-machine ip]:5000/api/student

   Using **psgAdmin**, try inserting more data into your tables and retrieving the data via Postman.

Exercise 4 - Add a call to your API/REST service from your Web app

1.
    1. In a command window/shell go to the **webapp** folder and run the commands:

       npm install vue-axios

       npm install axios

       These are the libraries we are using for making HTTP calls from the Web app.
    2. Add logic to the Vue component for student to retrieve the data from your REST service rather than from a hardcoded JSON string. See my **Product.vue** component for how to do this. You'll need to add Axios and VueAxios to the

main.js file under the src folder as follows:

You'll also add the line: **Vue.use(VueAxios, axios).** See my main.js to see where to put it.

3. Copy the files .env.development and .env.docker from my webapp folder to your webapp folder.

4. If you stopped your webapi from exercise 3 then restart it.

5. Open a cmd/terminal window into your webapp folder and run the command:

   npm run serve

6. Navigate to the URL displayed at the end of the command in step 5. You should be able to go to your student page and see the data displayed from the database.

Exercise 5 - Build and run your Web app in a Docker container

1.
   1. Copy the Dockerfile from my webapi folder to your webapi folder.
   2. Copy the docker-compose.yml file from the root of my week 7 folder to your lab folder. Change the Docker Hub tag names from mine (oitgaden) to yours.
   3. In your webapp/package.json file, add the following script to the "scripts" section:

      "build-docker": "vue-cli-service build --mode docker"

   4. Build your webapp like in exercise 4 by running the command:

      npm run build-docker

This will configure your webapp using .env.docker to access your webapi from the correct host:port combination.

5. Run the command

   docker-compose up -d --build

   from your lab folder. You should see the images being built and the containers started. After the command completes run the command

   docker ps

   to see the running containers **webapp** and **webapi**.
6. Try accessing your webapp at http://localhost:8080/ You should be able to go to the student page and see the table populated with data coming from your API.
7. To stop the containers run the command

   docker-compose down

   in your lab folder. When it completes run

   docker ps

   and you should now not see your containers running.
8. Push your Docker images to Docker Hub and push your code to GitHub.

Extra Credit - Implement other CRUD operations

Implement any or all of the additional CRUD functions and associated webapp UI in your application like my week 7 example. You will receive **20 lab points** for each of the CRUD functions beyond GET that you implement.

Extra Credit - Implement instructor data from database (20 lab points)

Implement the GET operation using the database for instructors as you did for students.

Extra Credit - Incorporate an external service (50 lab points)

Incorporate an external service of your choice from http://www.programmableweb.com/