ACIT 3855 – Lab 10

Dashboard User Interface

Instructors

- Mike Mulder (mmulder10@bcit.ca)
- Tim Guicherd (tguicherd@bcit.ca)

Due date: demo and submission by end of next class.

Purpose

- Use Docker and Docker Compose to run our microservices
- Build a simple Single Page Application (SPA) user interface on top of your services

Part 1 – Dashboard User Interface

Using React or plain HTML/JS, build a simple user interface for your system that does the following:

- Displays the current statistics by calling the `/stats` API endpoint from your Processing Service.
- Displays analyzer information by calling API endpoints from your Analyzer Service.
    - Display the stats from the analyzer
    - Display (at least) one random event from one type
- Displays a graphic representing your system (i.e., your Logo). Find something from the web to use.
- Displays the last updated time (this could be obtained from the /stats API endpoint or from Javascript in the browser).
- Updates the statistics and analyzer information every 2 to 4 seconds.

Option 1: Use plain HTML and Javascript for the Dashboard UI

You can find an example of a dashboard application, using only Vanilla JS. Make sure you adjust the URLs in the Javascript file and add all required elements. You will have to deploy it using a static webserver (for example, Nginx). You can use the `nginx` Docker image - by default, it serves static content from `/usr/share/nginx/html`. Create a dedicated Docker image for it:

```
FROM nginx

COPY dashboard /usr/share/nginx.html
```

Option 2 (deprecated): React App

You may use the sample application (Sample_Code.zip) provided on D2L:

- Extract the zipfile into a new folder in your Git repo called dashboard-ui.
- Make sure you have npm installed on your machine
- In the folder run:
  - `npm install`
  - This will bring in the dependencies from package.json
- Make changes to the following file:
  - `src/App.js` – Set the endpoints to those for your three analyzer endpoints (i.e., stats, blood-pressure and heart-rate)
  - `src/AppStats.js` – Change the URL to that of your Cloud VM. Also, change the StatsTable to reflect your stats and the keys from the JSON response from your /stats endpoint
- You should add the last updated time to your stats endpoint so you can display that as well
  - `src/EndpointAnalyzer.js` – Change the URL to that of your Cloud VM.
- Replace the logos in the public and src folders, logo.png and logo_icon.png with those that are reflective of your project.
- In the folder, run `npm start` to start the web application. It will run on http://localhost:3000

Your data should be refreshed every 2 to 4 seconds using a timer as per the example. You will need to Dockerize the application in order to run it with your other services. An example Dockerfile is provided below - adjust it to your needs.

```
FROM node:13.12.0-alpine

# Set working directory
WORKDIR /app

# Add `/app/node_modules/.bin` to $PATH
ENV PATH /app/node_modules/.bin:$PATH

# Install app dependencies
COPY package.json ./
COPY package-lock.json ./
RUN npm install
RUN npm install react-scripts@3.4.1 -g

# add app
COPY . ./

# Start app
CMD ["npm", "start"]
```

> ⚠️ The version of NodeJS used is old and unsupported. Unless you have a very good reason, you should use the vanilla JS dashboard.

Setup CORS on your microservices

By default, your GET HTTP requests will fail due to CORS. You will have to update your backend microservices (Processing and Analyzer) to disable CORS checking as documented in the Connexion Cookbook.

Add the following code to your `app.py`:

```
from connexion.middleware import MiddlewarePosition
from starlette.middleware.cors import CORSMiddleware


app = FlaskApp(__name__)

app.add_middleware(
    CORSMiddleware,
    position=MiddlewarePosition.BEFORE_EXCEPTION,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

You will need to re-build your Docker images for the Processing and Analyzer services: `docker compose build` should be enough. Don't forget to restart your containers after updating the images.

> 💡 Please note that this is a very insecure CORS setup!

Deploy your changes

Run your provisioning tool to update the setup on your cloud VM, and check that you can access the dashboard UI. Bring up all your services, including the new dashboard. Test that the Dashboard UI works from your Cloud VM and updates when you run your jMeter script.

Grading and Submission

Your Dashboard UI has been updated to: (3 marks)

- Retrieve and display the stats from the processing service
- Retrieve and display the stats from the analyzer service
- Retrieve and display an analyzer event for each of the two event types
- You have updated the image to be reflective of your project

The Processing and Analyzer services have been updated to disable CORS. (1 mark)

Demo the Dashboard UI (6 marks):

- Show it updating with valid values as your run your jMeter test script. It should show your stats and some information from your analyzer
- It should run both locally (on your laptop) and from your Cloud VM

Submit a zipfile containing the following to get your marks:

- your updated `docker-compose.yml` file
- a screenshot of your dashboard UI with non-zero values

Total: 10 marks