ACIT 3855 – Lab 9 – Dashboard UI

Instructor	Mike Mulder (mmulder10@bcit.ca) – Sets A and C Tim Guicherd (tguicherd@bcit.ca) – Set B
Total Marks	10
Due Dates	Demo by end of next class: • March 18th for Set C • March 21st for Sets A and B

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 – Simple Dashboard User Interface

Using React, 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 audit information by calling both of the API endpoints from your Audit Log Service. For example, display a random event or the first event of each 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 you could get the current time in your React app).
- Updates the statistics and audit information every 2 seconds.

Option 1: Simple React App

You may use the sample application (Sample_Code.zip) provided in the Week 9 content 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 two audit endpoints (i.e., 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/EndpointAudit.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

Here is an example UI:

Latest Stats

Blood Pressure Heart Rate

BP: 59 # HR: 58

Max BP Systolic: 125

Max BR Diastolic: 160

Max HR: 125

Last Updated: 2021-03-11T15:58:16

Audit Endpoints

blood_pressure-0

{"blood_pressure":{"diastolic":80,"systolic":120},"device_id":"A12345","patient_id":"d290f1ee-6c54-4b01-90e6-d701748f0851","timestamp":"2016-08-29T09:12:33.001000+00:00"}

heart_rate-13

{"device_id":"A12345","heart_rate":89,"patient_id":"992716","timestamp":"2016-08-29T09:12:33.001000+00:00"}

Note: It would be good to return the last updated timestamp from the /stats endpoint of your Processing service so we can see the UI updating even when there are no events being processed.

Your data should be refreshed every 2 to 4 seconds using a timer as per the example.

By default, your GET HTTP requests will fail due to CORS. You will have to update your backend microservices (Processing and Audit Log) 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 Audit_Log services.
    - Stop the services (docker-compose down in the deployment folder)

    - Remove the existing Docker images for Processing and Audit_Log
        - docker images will list the images
        - docker image rm <image id> will remove the specified image
        - cd to the folder of the service you want to build
        - docker build -t processing:latest (or audit_log:latest) to rebuild your images
    - Start the services (docker-compose up -d in the deployment folder)
    - Make sure the code in your IDE (on your Laptop) is synched up with the code on Cloud VM (via your Git repo)
- Now your React application should be able to successful call the API endpoints.

Option 2: Use pure HTML and Javascript for the Dashboard UI

You can find another example of a dashboard application, using only Vanilla JS. Make sure you adjust the URLs in the Javascript file. You will have to deploy it using a static webserver (for example, Nginx).

Note: You'll still need to update the backend services disable CORS checking as in Option 1.

Part 2 – Containerize and Deploy Your Dashboard UI

Option 1: Deploy Your Simple React App

Add a dockerfile to your dashboard-ui folder with the following contents to run our React application:

```
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"]
```

Make sure you check this (and the other code) into your Git repository.

Then update your repository on your Cloud VM to get the new dashboard-ui folder.

Build the docker image:

```
docker build -t dashboard:latest .
```

Add the dashboard to your docker-compose.yml file:

```
dashboard:
    image: dashboard
    ports:
      - "3000:3000"
    depends_on:
      - "processing"
      - "audit_log"
```

Note that you'll need to open Port 3000 on your Cloud VM to access the Dasbboard 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. You will need to demo this for this lab.

Option 2: Deploy your vanilla JS Dashboard UI using Nginx

You can use the following Dockerfile as a starting point:

```
FROM nginx

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

Then, reference your build in the docker-compose.yml file:

```
dashboard:
    build: dashboard
    ports:
      - "8888:80"
    depends_on:
      - "processing"
      - "audit_log"
```

Grading and Submission

Submit the following to the Lab 9 Dropbox on D2L:

- Your updated docker-compose.yml file

Your Dashboard UI has been updated to: • Retrieve and display your stats from the processing service • Retrieve and display an audit event for each of the two event types • You have updated the image to be reflective of your project	4 marks

The Processing and Audit Log services have been updated to disable CORS.	
Demo the Dashboard UI: • Show it updating with valid values as your run your jMeter test script. It should show your stats and some information from your audit log • It should run both locally (on your laptop) and from your Cloud VM	6 marks
Total	**10 marks**