Ahmad Shalaby and Ulises Almonte Homework 3 with Extra Credit Readme File

<http://54.208.88.112:30424/> - deployed vue frontend

<http://54.208.88.112:30010/> - deployed springboot backend

Files in the Project Folder(VueJsSpringbootPipeline)  
Springboot Project including Dockerfile  
- ualmonte-survey-page/

Kubernetes deployment files for front and backend applications

* frontend-survey-page-deployment.yaml
* frontend-survey-page-service.yaml
* backend-survey-page-deployment.yaml
* backend-survey-page-service.yaml

Vue.js project

* Frontend/

Jenkins file for the pipeline

* Jenkinsfile

Contains the kubeconfig of the cluster that holds the application  
- kubeconfigcredentials

My partner Ahmad Shalaby and I worked on this Hw3 assignment I was responsible for the deployment and testing the backend, setting up the RDS instance and connection. Ahamad created the front-end application, deployed it and hosted our deployments (frontend and backend) on his Kubernetes cluster. We collaborated to fix bugs, test our application and write the pipeline.

**Set up a database:**

I set up the database using a RDS MySql instance. I used an instance with micro class, woth 20 Gibs of storage, open it on port 3306 and I made sure to make it publicly available, I also allowed access from any port to make it easier to connect from the application.  
A screenshot of a computer

AI-generated content may be incorrect.  
A screenshot of a computer screen

AI-generated content may be incorrect.  
A screenshot of a computer

AI-generated content may be incorrect.  
once the database is running make sure to copy the dns address and use it in the springboot application

**Springboot project:**  
I created the template for the project using the spring initializer <https://start.spring.io/index.html> and I installed the necessary dependencies. I followed these guidelines to name the project and get the correct dependencies A close-up of a document

AI-generated content may be incorrect.  
A screenshot of a computer

AI-generated content may be incorrect.  
once the project is downloaded, unzip to where you want your root directory.   
the next thing I did was set up the database connection. I opened the project and look for the application.properties file and pasted this in there   
A screenshot of a computer program

AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.  
spring.datasource.url=jdbc:mysql://database-1.cmpoo40mmop3.us-east-1.rds.amazonaws.com:3306/SurveyPage

spring.datasource.username=admin

spring.datasource.password=evo4PTdT5aEmmhwHBm5k

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update

spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect  
  
Writing the controller:  
for this I wrote a couple of endpoints to create, get, update, and delete

as I wrote the controller I had to create an entity to represent my database schema and a repository object to give me an interface to interact with the database.  
A screenshot of a computer program

AI-generated content may be incorrect.  
  
As an example ill explain the components of this endpoint used to create survey records  
A screenshot of a computer screen

AI-generated content may be incorrect.

the annotation @PostMapping can be accessed using “/submitSurveyRecord “ and consomes={“application/json”} defines what type of content it expects.  
ReponeseEntity<Long>is an object that encapsulates a status along with the response type. In this case the endpoints returns a Long for the id. In the parameters theres is SurveyRecord which is a class I defined in the entity section of the project   
A screen shot of a computer program

AI-generated content may be incorrect.  
inside the method I use a repository object that is connected to my database to save the surveyRecord and I return the Id that received from the SruveyRepository along the status code 200.

Adding CORS access>  
this part of the controller allows the controller to be accessed by other applications outside of the domain where the controller is running. Essentially this is to allow our front end to make requests to this backend.   
  
you want to create this package configuration under the java directory. It should be the same as the controller package.

A screen shot of a computer

AI-generated content may be incorrect.  
  
In it I created this class   
A screen shot of a computer program

AI-generated content may be incorrect.  
This is a mapping that allows requests to be filtered. the method addCorsMappings contains the origin of requests that are allowed and the methods they can use. The allowed origin here is the ipv4 address of the front end cluster.  
we have to also add this annotation to our controller   


**Testing with postman:**  
to test my implementation in post man i built the application using maven A screenshot of a computer

AI-generated content may be incorrect.

And ran it

A screenshot of a computer

AI-generated content may be incorrect.  
you should see something like this in the console   
A screen shot of a computer program

AI-generated content may be incorrect.  
to test using postman use the base url: http://localhost:8080/surveyapi/ {endpoint value}  
this is an example of me running the getSurveyRecords endpoint which retrieves all the existing surveys  
A screenshot of a computer

AI-generated content may be incorrect.

To test the post request switch the Get method type to Post and change the api endpoint to /submitSurveyRecord, got to body, click raw and switch the content type to json because that’s why this endpoint expects  
A screenshot of a computer

AI-generated content may be incorrect.

**Docker containerization:**

Once the application is running fine we need to make sure it can build and run as a .jar file.   
in ecplise you can export the project as a .jar file by clicking on the project top folder then go to File in the top bar and selecting Export   
A screenshot of a computer

AI-generated content may be incorrect.  
A screenshot of a computer

AI-generated content may be incorrect.

Hit next.   
A screenshot of a computer

AI-generated content may be incorrect.  
make sure you are exporting with these settings. The export destination needs to be somewhere you can reference from the Dockerfile.   
  
To containerize the image we start off by creating a Docker file for the application under the top directory   
A screenshot of a computer program

AI-generated content may be incorrect.  
A screenshot of a computer

AI-generated content may be incorrect.  
FROM command is to use jdk version 22 which is what I have installed in my machine.  
COPY gets the built application from the projects target folder. This would be different if you exported to another folder. This command copies it into a file at the root directory.   
After this we save and go to a console to build this image and push it to dockerhub for later use in the pipeline. Make sure you are in the directory with your dockerfile and run this command “ulisesalmonte/survey-page:v1” is the name and tag of the image.  
A screen shot of a computer

AI-generated content may be incorrect.

We can run this image using   
docker run -p 8888:8080 <image name>

**Kubernetes deployment:**  
To deploy on Kubernetes you need to first start up a rancher docker container and write the configuration files. I used a similar process as last assignment to start the rancher container in a EC2 instance, and I wrote these files to deploy the springboot application.  
A screen shot of a computer

AI-generated content may be incorrect.  
A screenshot of a computer program

AI-generated content may be incorrect.  
after creating these files I used  
kubectl create -f <deployment-filename>.yaml

Kubectl create -f <service-filename>.yaml

After these steps our application backend is live on a kubernetes cluster and we can test this using postman. Make sure you add an inbound rule to your EC2 intance that allows traffic from anywhere on port 30010.   
A screenshot of a computer

AI-generated content may be incorrect.

**Front End:**

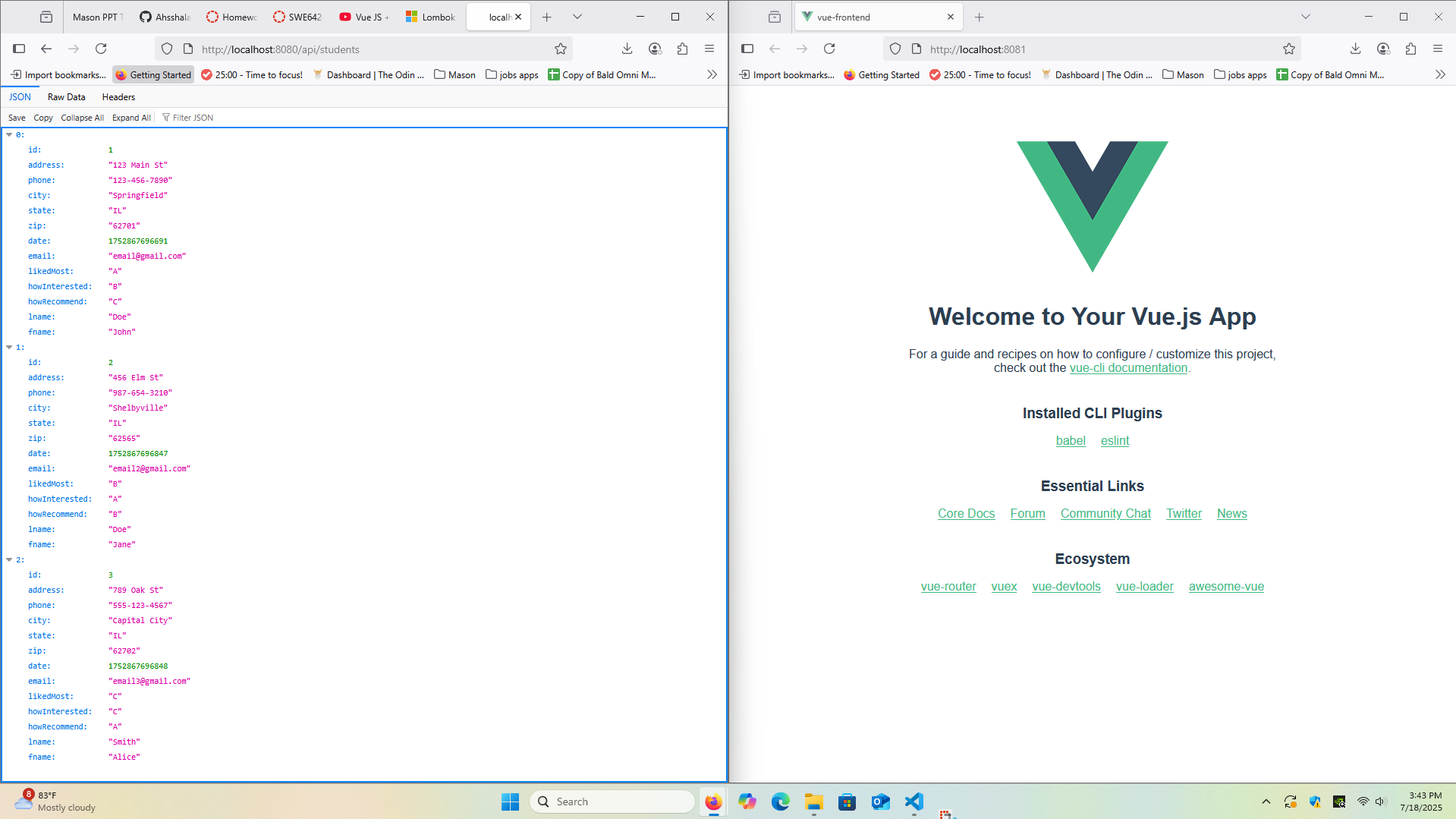
For learning’s sake, Ulisses and I both created our own separate Spring boot backends. Afterwards we consolidated our code together so the frontend and backend communicated properly. To prevent redundancy, this section will be written with the assumption that the springboot backend is already built and ready on your local machine.

First, open the spring boot backend onto your local machine. Run your application on some port (automatically 8080), as shown above.

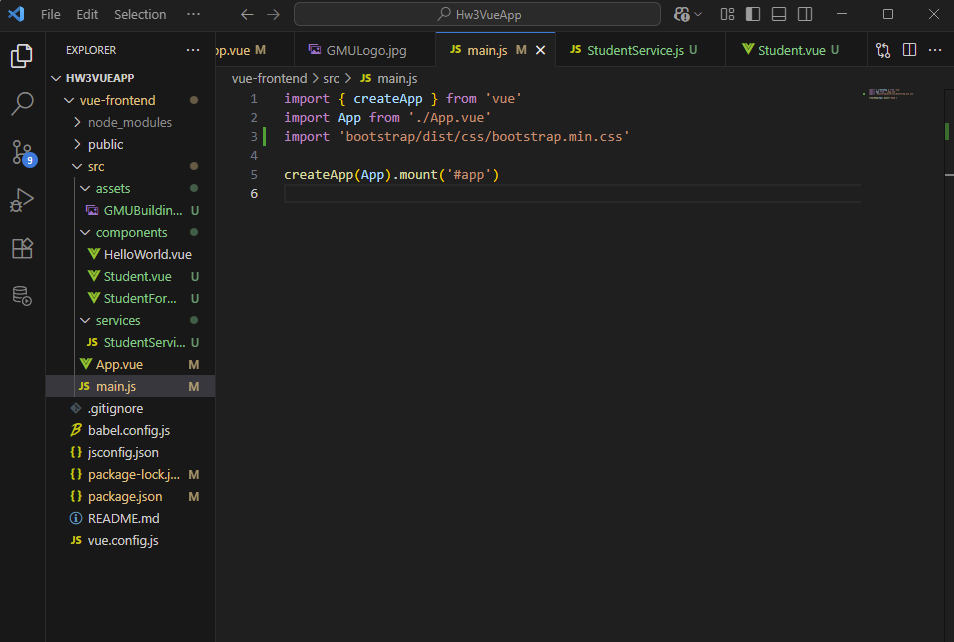


Now, create a new folder, preferably on the desktop for ease of use, install node and npm <https://nodejs.org/en/download> and check that they are installed through node –version and npm –version. The IDE should not be too important, however, for both the backend and frontend implementation I used visual studio code. In order to use this IDE, make sure to install the extension Vetur (for vue). In the terminal, type sudo npm install -g @vue/cli. This is similar to spring initializer, and will help us set up the vue application with all our autoconfigurations

At this point all you have to do is type vue create vue-frontend (or other name) into the terminal inside the folder you want and the application will be created. Cd into the folder we just created (vue-frontend) and type npm run serve to run the application. This will put the vue application running on port 8081 (8080 is already taken up by the springboot). Also run npm install bootstrap –save for the bootstrap library and npm add axios. Then import than library into the [main.js](http://main.js) file in your vue project



This is what your current setup should look like. There is a “hello world” frontend running on port 8081 and the springboot api running on 8080.



We need a javascript file for the frontend that we will be creating. This was done in a file called [StudentService.js](http://studentservice.js). This file contains the different get/set methods that will call the springboot backend.

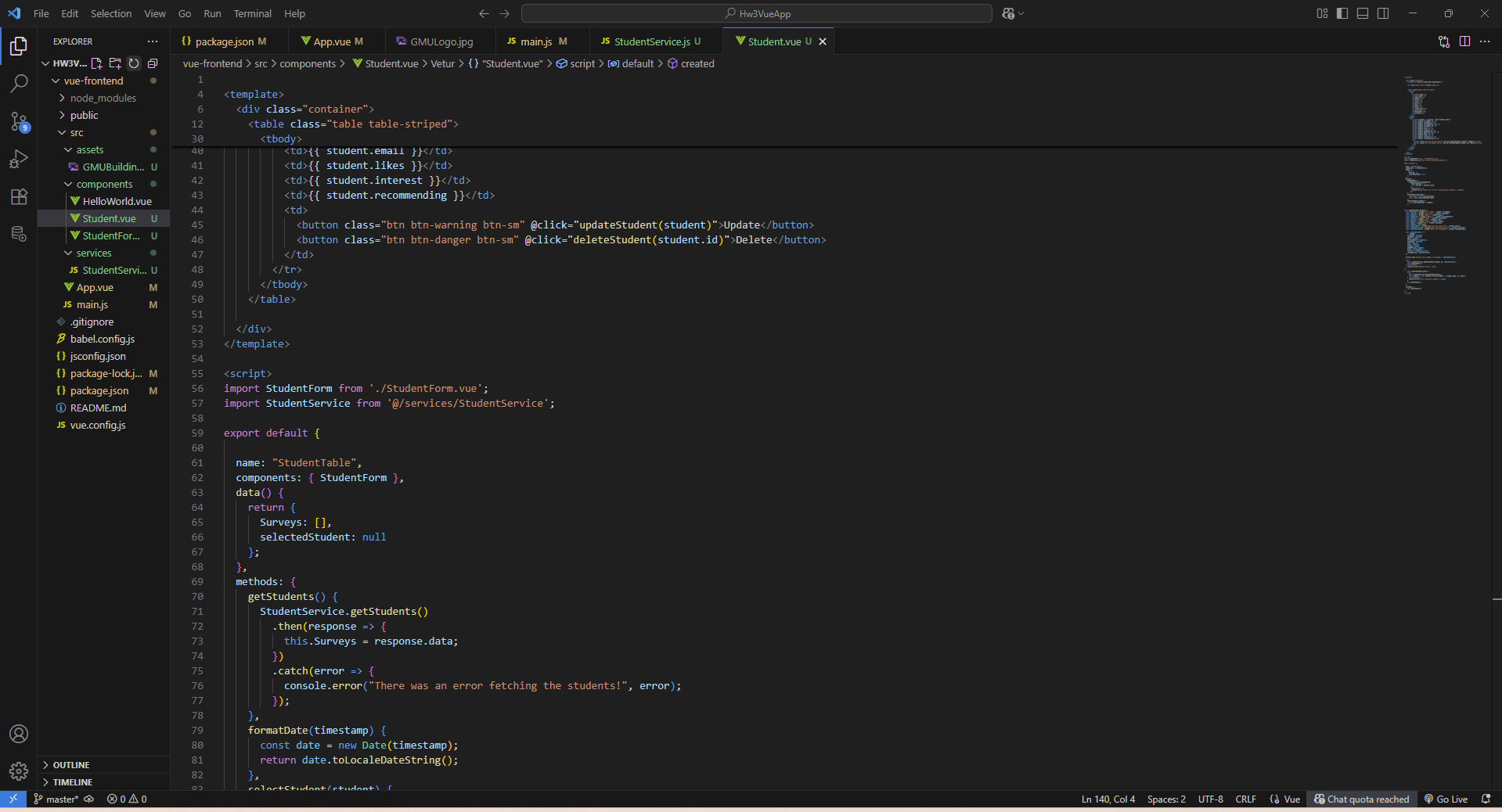


As shown, the base URL is our springboot local host, the different methods reference the postman methods shown in the springboot implementation. For example, the first getStudends() methods references the url with getSurveyRecords A screenshot of a computer

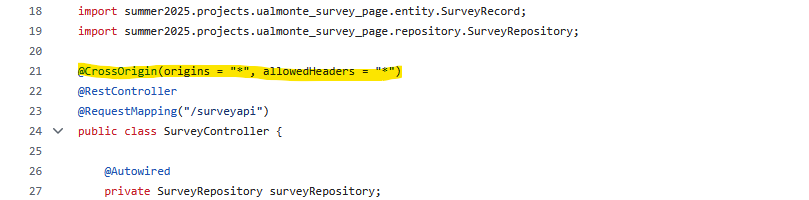
AI-generated content may be incorrect.

Please note: after deploying the application to kubernetes, make sure you change the base URL to the new one after implementation.

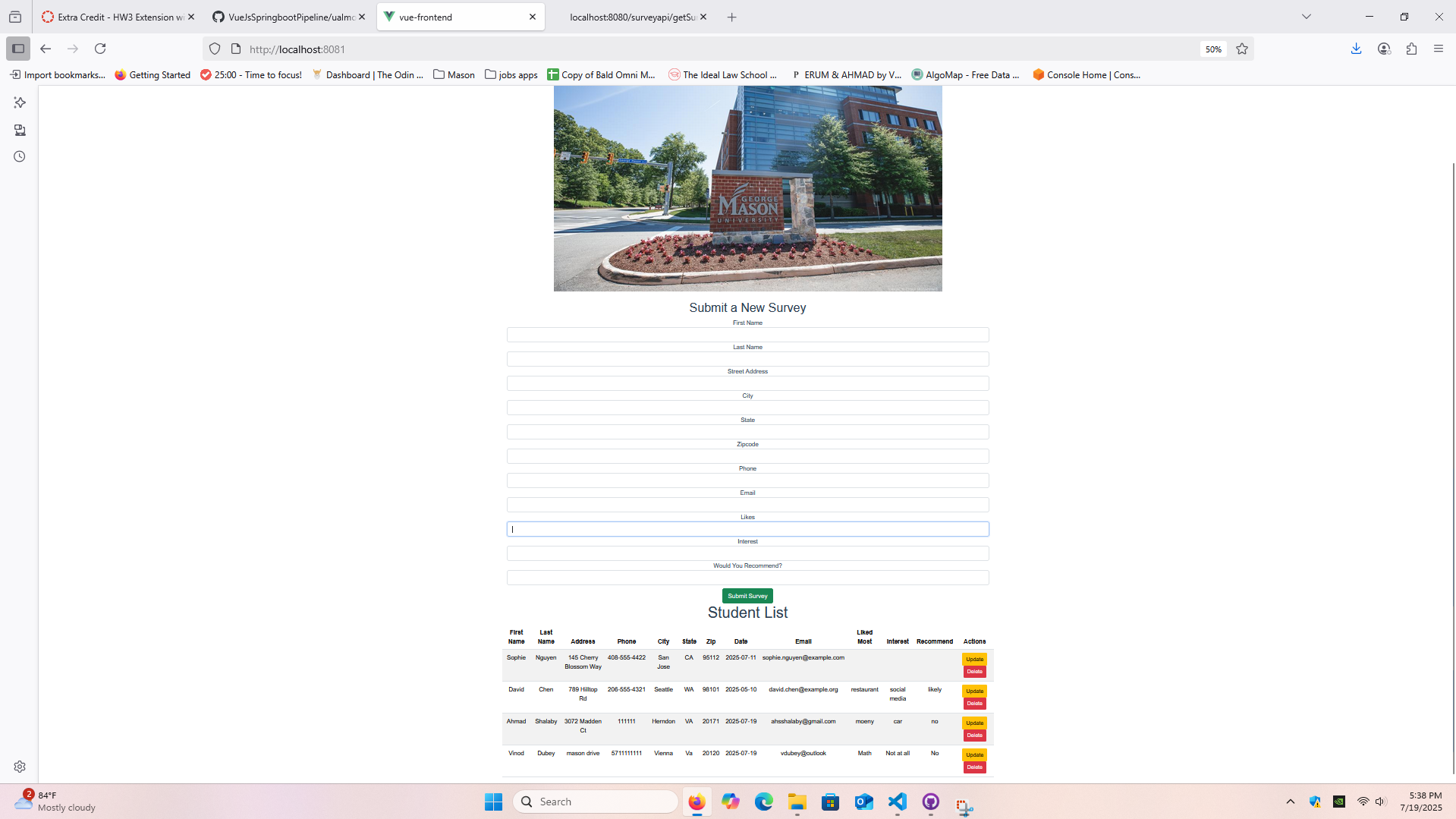
Now, we make a different component to replace the hello world generic one. This component, which I called student.vue contains essentially all the regular html front end code for the student table, as well as methods and buttons to call the backend functions. Please refer to the code on github for more detail on how that looks alike. Don't forget to import the javascript file which contains the methods which call on the springboot backend API. Also make sure the app.vue file references this new component instead of hello world as well.



Lastly, go back to the springboot backend and add code to allow it to communicate with the front end running on port 8081.



Note: we allowed origins from “\*” which means all addresses. This is not what you actually want to do when deploying a real project. This was just for convenience since we changed IP addresses a lot in deployment and we wanted to minimize errors when we were troubleshooting.

Your front end application should now be able to call on the backend API. This is the basic application with functionality to delete and edit, which shows the springboot API. This also includes an extra student survey form file to allow submitting a survey functionality. Now to finalize implementation, first confirm that the variable names, springboot URL, and functionality all work correctly.

**Deploying to kubernetes:**

Repeat the steps above for deploying an application for kubernetes, except this time use a different dockerfile. Make sure to install npm and other dependencies for the application to work.

FROM node:20 AS build-stage

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

RUN chmod +x node\_modules/.bin/vue-cli-service

RUN npx vue-cli-service build

RUN npm run build

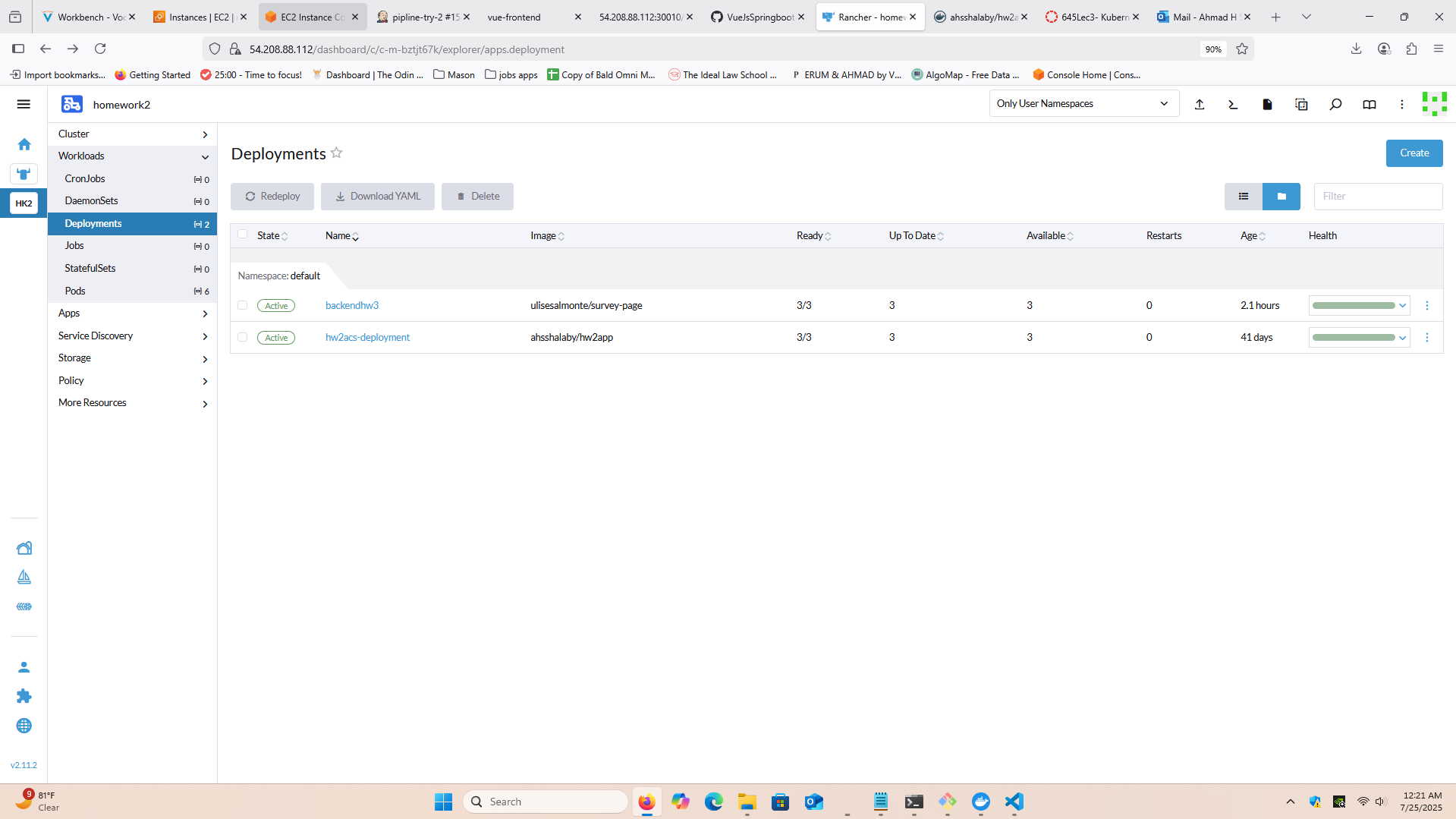
FROM nginx:latest AS production-stage

COPY --from=build-stage /app/dist /usr/share/nginx/html

EXPOSE 80

CMD ["nginx","-g","daemon off;"]

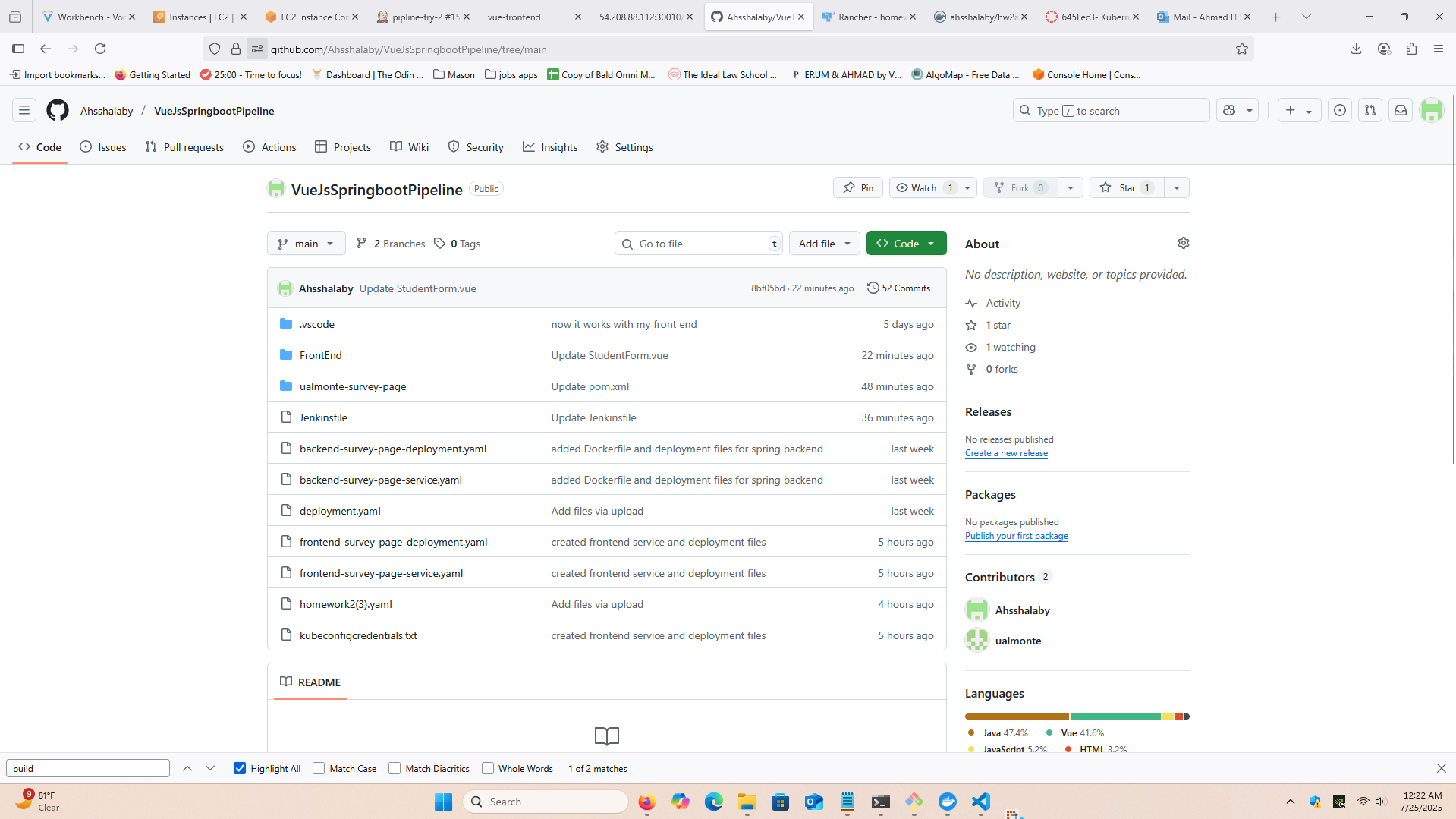
Our setup involved having both parts of the application deployed on the same virtual machine to simplify troubleshooting. But other setups would work as well.



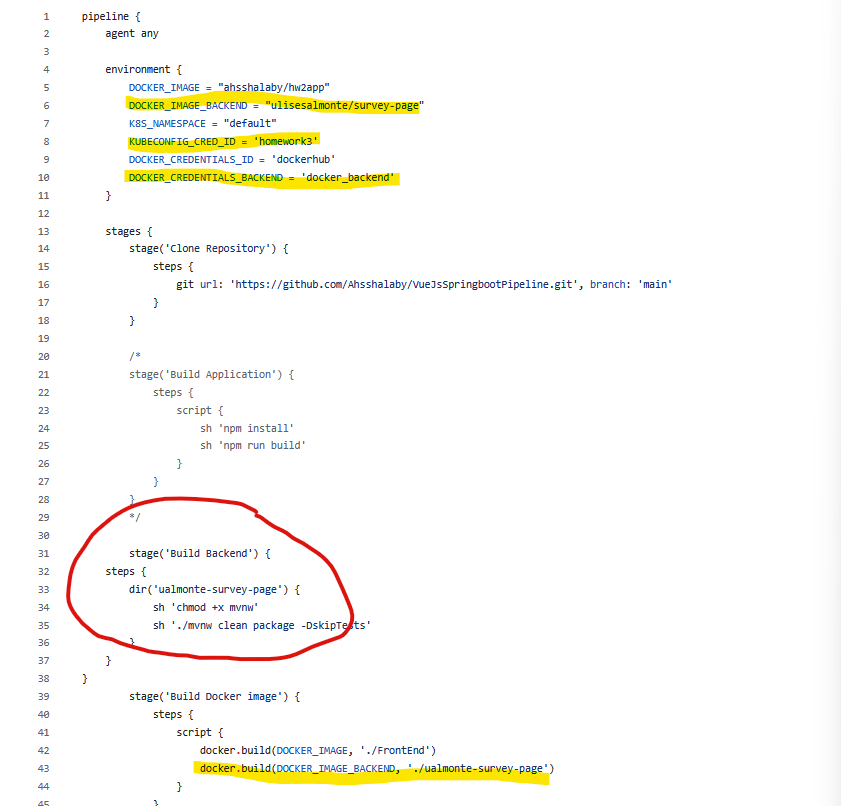
**Jenkins CI/CD Pipeline**

The jenkins CI/CD pipeline used is similar to homework 2 implementation, however, is more complicated due to the addition of key steps. The pipeline updates changes to both the front-end and backend of the application.

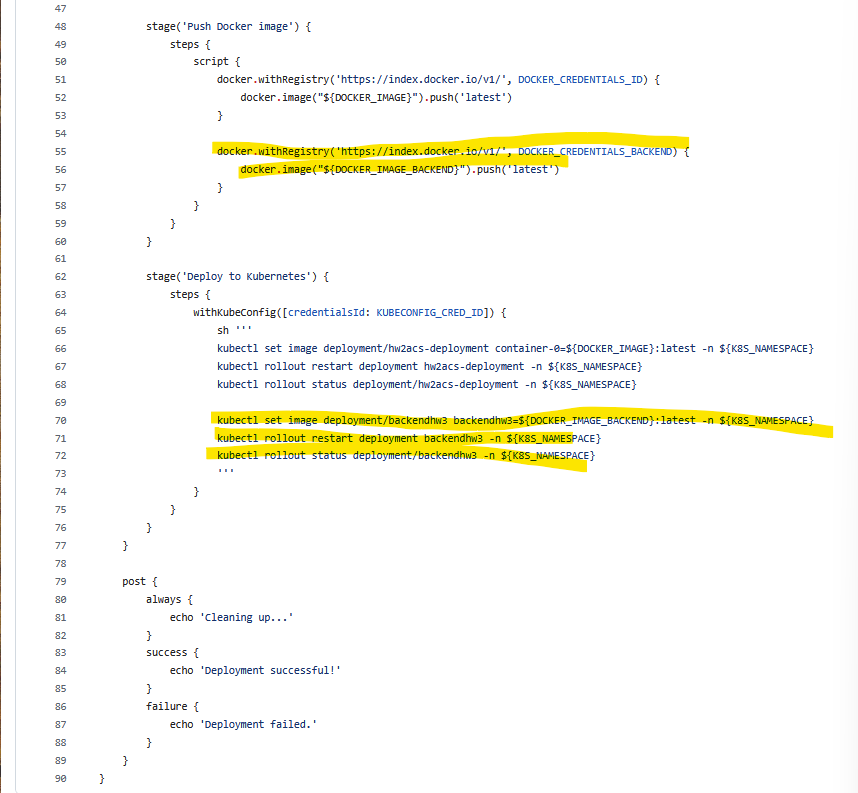
First, a github repository was created with all our files, the dockerfiles, yaml configuration files from kubernetes, and a newly created jenkins file.



The jenkins file was built by taking the generic homework 2 jenkins file (which updated an html document on one kubernetes server) and expanded for greater functionality. The jenkins file with changes annotated can be found below

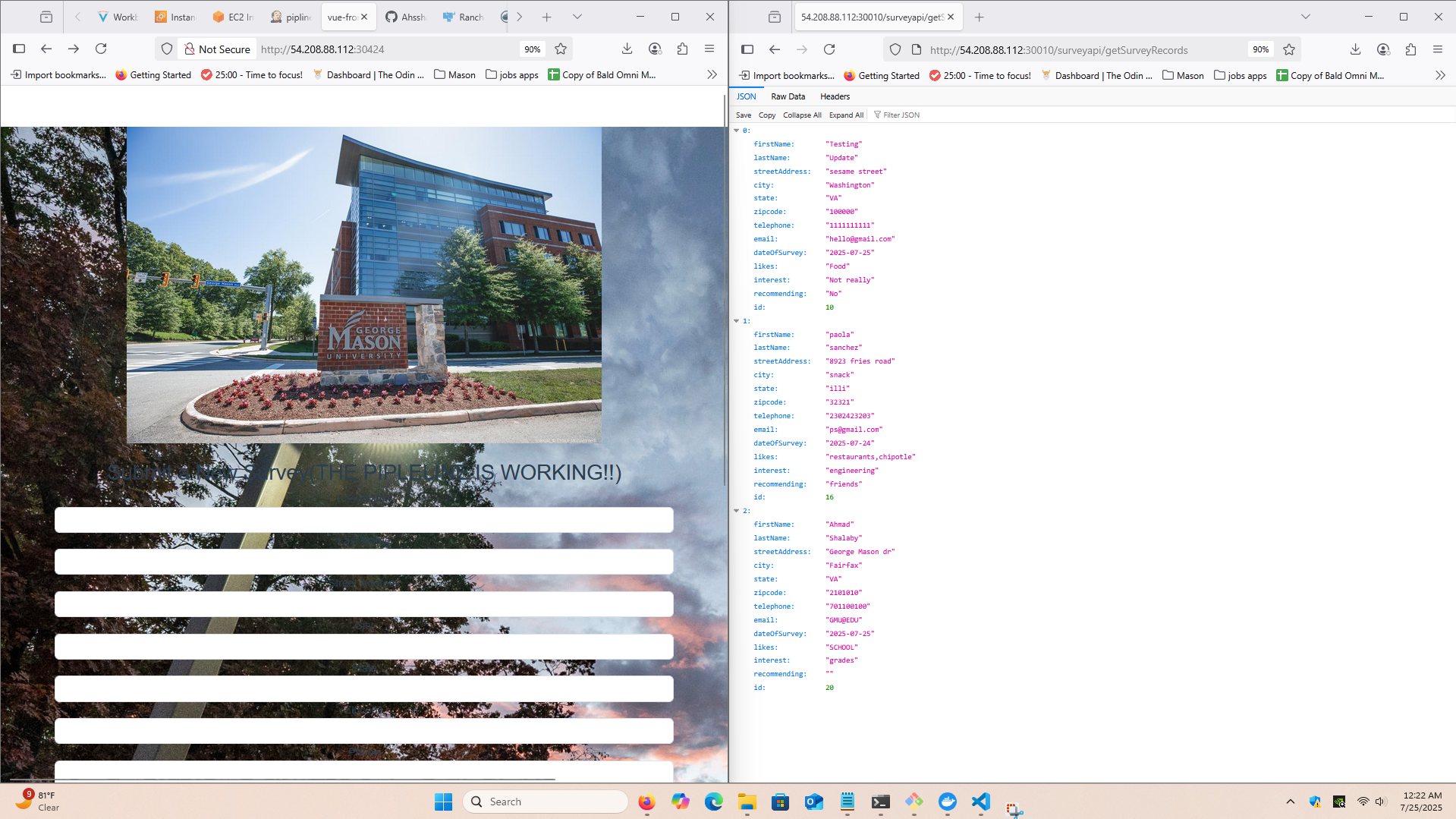


New environment variables for added, this was for the new kubernetes config files and the login information for the second dockerhub account used. The docker image is built from the dockerfile to my dockerhub account, and the backend is built from the corresponding dockerfile to Ulises dockerhub account. The build backend stage is necessary to run before pushing the image since it creates the .jar file for the docker image.



The code to push and deploy the images is the same, but copied for the other dockerhub account and for the second kubernetes pod deployment.

After lots of troubleshooting, the CI/CD pipeline runs properly and updates any changes made to the repository. Again, remember to ensure that the URL links are updated with the new IP addresses from Kubernetes. Also, keep in mind that the database backend takes longer to start up again then the frontend, so it causes a delay where the backend stops working for a couple minutes after restarting the deployments. Video of the CI/CD pipeline at work is attached to the submission.



Final product after cleaning up the front end to look nicer, and running the pipeline successfully.