Student Survey Page (Youtube Demo Link)

Team:

Name	Email	GMU ID
Janit Bidhan	jbidhan@gmu.edu	G01326011
Rosy Sultana	rsultan6@gmu.edu	G01187642
Brenda Henriquez	bhenriqu@gmu.edu	G01139846
Uday Kumar Kamalapuram	ukamalap@gmu.edu	G01340201

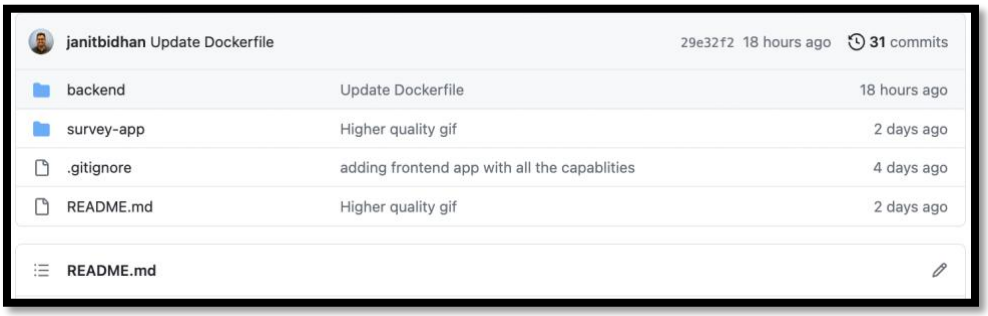
Prerequisites for the assignment:

1. **IDE for development.**
2. **Postman for API testing**
3. **DBeaver for Database testing/setup**
4. **Docker Desktop for building and testing local images.**
5. **Personal accounts on GitHub, Google Cloud Platform and Docker Hub.**

Technology Used:

1. **GitHub**
2. **IDE**
3. **Postman**
4. **Docker Desktop**
5. **Angular**
6. **DBeaver**
7. **Serverless Database (Cockroach DB Serverless)**

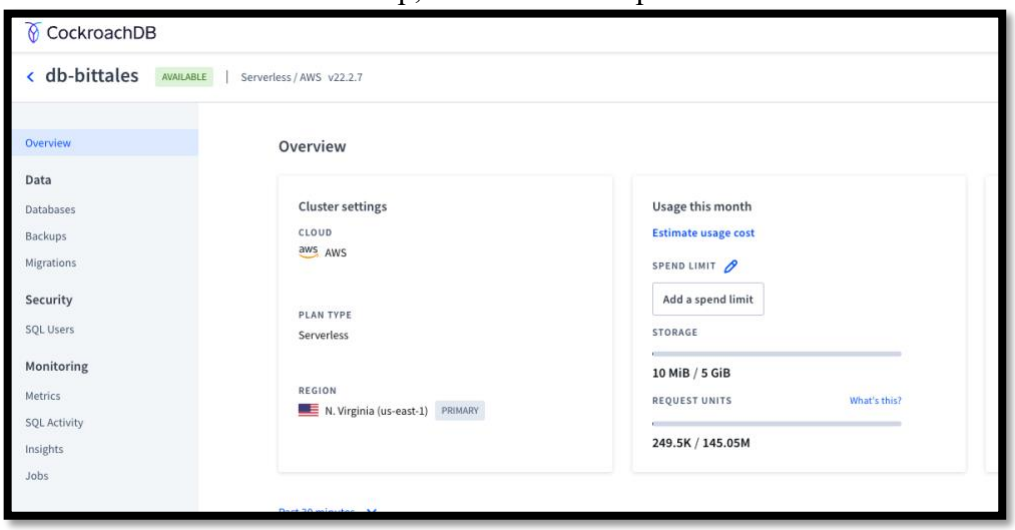
1. Prerequisites:
   a. Setup Project on GitHub



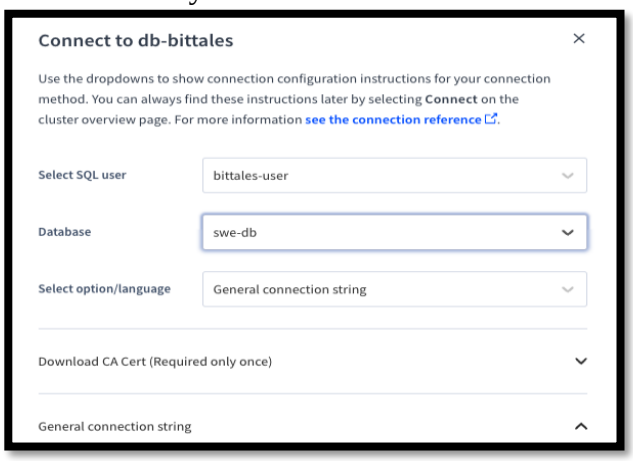
   b. Setup Database on CockroachDB Serverless

Open the cockroach cloud and create a new cluster and make sure to note down username, password that you create and map to database.

Once it is up, it would show up like this:



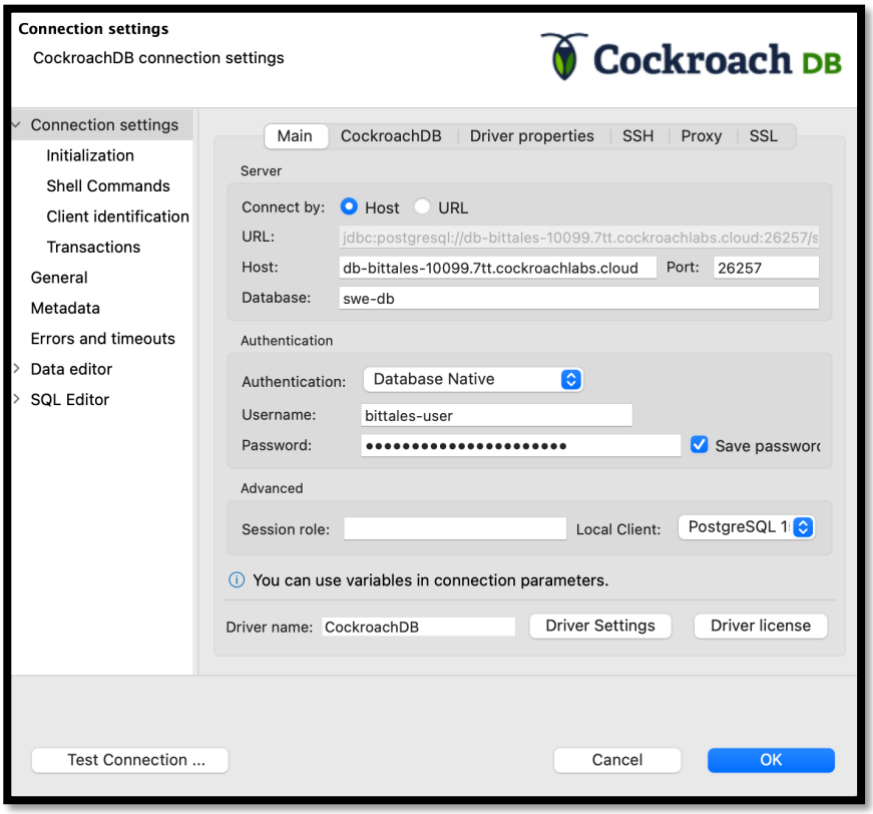
Press connect and find the necessary details for connection.

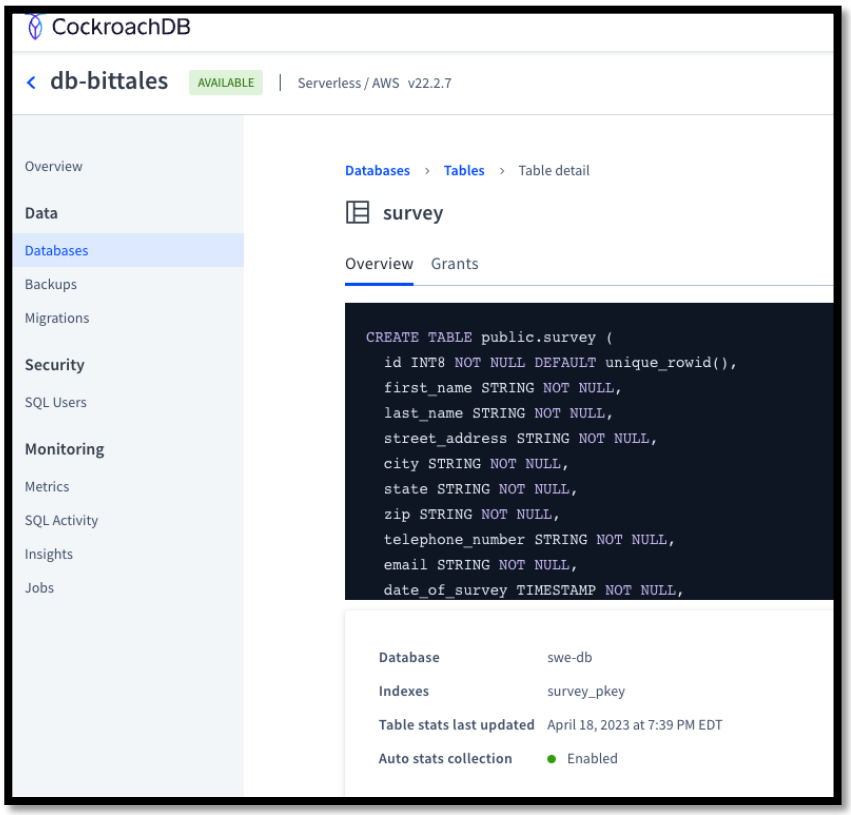


Now we have the database setup, we can create a table next,

c. Connect through local and create a New Table.

Connect through DBeaver and create new table and on checking database> tables. We should now see a new table.


Connection settings
CockroachDB connection settings
Cockroach DB
Connection settings
Initialization
Shell Commands
Client identification
Transactions
General
Metadata
Errors and timeouts
Data editor
SQL Editor
Main
CockroachDB
Driver properties
SSH
Proxy
SSL
Server
Connect by: Host URL
URL: jdbc:postgresql://db-bittales-10099.7tt.cockroachlabs.cloud:26257/s
Host: db-bittales-10099.7tt.cockroachlabs.cloud
Port: 26257
Database: swe-db
Authentication
Authentication: Database Native
Username: bittales-user
Password: ••••••••••••••••••••••
Save passwor
Advanced
Session role:
Local Client: PostgreSQL 1
You can use variables in connection parameters.
Driver name: CockroachDB
Driver Settings
Driver license
Test Connection ...
Cancel
OK


CockroachDB
db-bittales
AVAILABLE
Serverless / AWS v22.2.7
Overview
Data
Databases
Backups
Migrations
Security
SQL Users
Monitoring
Metrics
SQL Activity
Insights
Jobs
Databases › Tables › Table detail
survey
Overview
Grants
CREATE TABLE public.survey (
id INT8 NOT NULL DEFAULT unique_rowid(),
first_name STRING NOT NULL,
last_name STRING NOT NULL,
street_address STRING NOT NULL,
city STRING NOT NULL,
state STRING NOT NULL,
zip STRING NOT NULL,
telephone_number STRING NOT NULL,
email STRING NOT NULL,
date_of_survey TIMESTAMP NOT NULL,
Database swe-db
Indexes survey_pkey
Table stats last updated April 18, 2023 at 7:39 PM EDT
Auto stats collection Enabled

```sql
CREATE TABLE survey (
                        id SERIAL PRIMARY KEY,
                        first_name STRING NOT NULL,
                        last_name STRING NOT NULL,
                        street_address STRING NOT NULL,
                        city STRING NOT NULL,
                        state STRING NOT NULL,
                        zip STRING NOT NULL,
                        telephone_number STRING NOT NULL,
                        email STRING NOT NULL,
                        date_of_survey TIMESTAMP NOT NULL,
                        liked_options JSON,
                        interested_source JSON,
                        recommend String NOT NULL,
                        additional_comments STRING
);
```

2. Backend:

a. Creating Spring Boot REST Application and Writing business logic.

Created a spring boot application with Spring Initializer. https://start.spring.io/

Downloaded the Spring Initializer project with required dependencies like Spring Web, Spring Data JPA and others. In application.properties file given the datasource.url using the property as below.

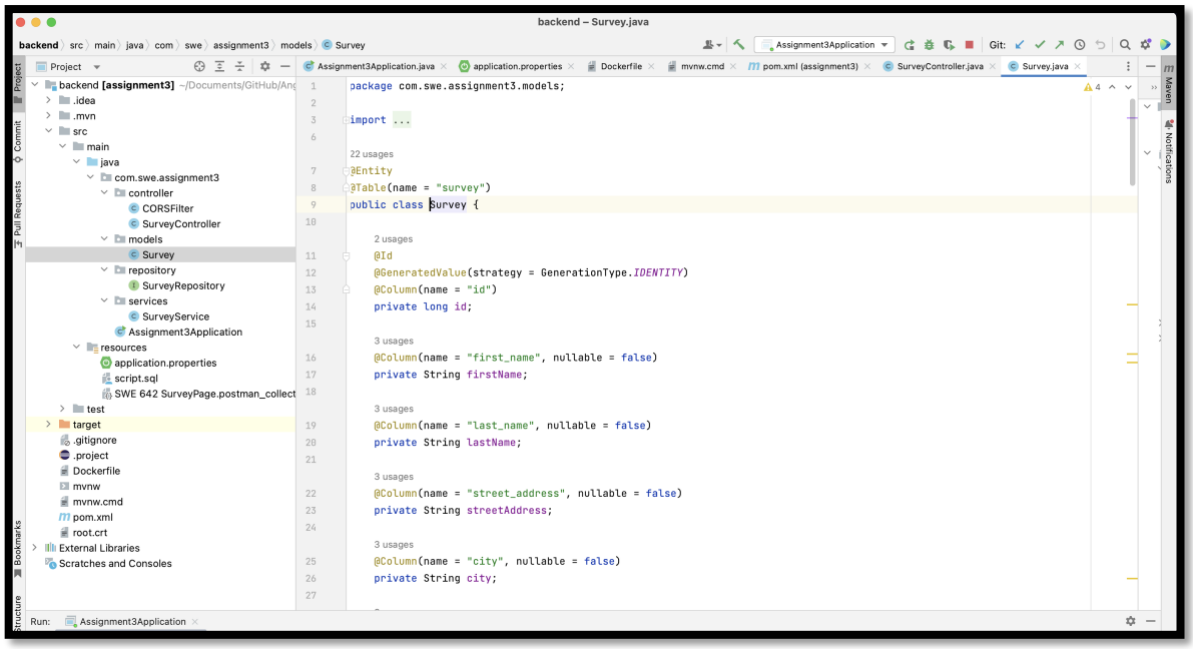
```
spring.datasource.url=jdbc:postgresql://db-bittales-10099.7tt.cockroachlabs.cloud:26257/swe-db
```

We have created the following Controller package and it consists of Survey Controller Java class which handles the Rest API requests from the frontend application.

We are using @GetMapping to get the data from the database and send response back to the frontend. We are using @PostMapping to handle the Save Survey to store the new Survey object coming from Frontend into the database.

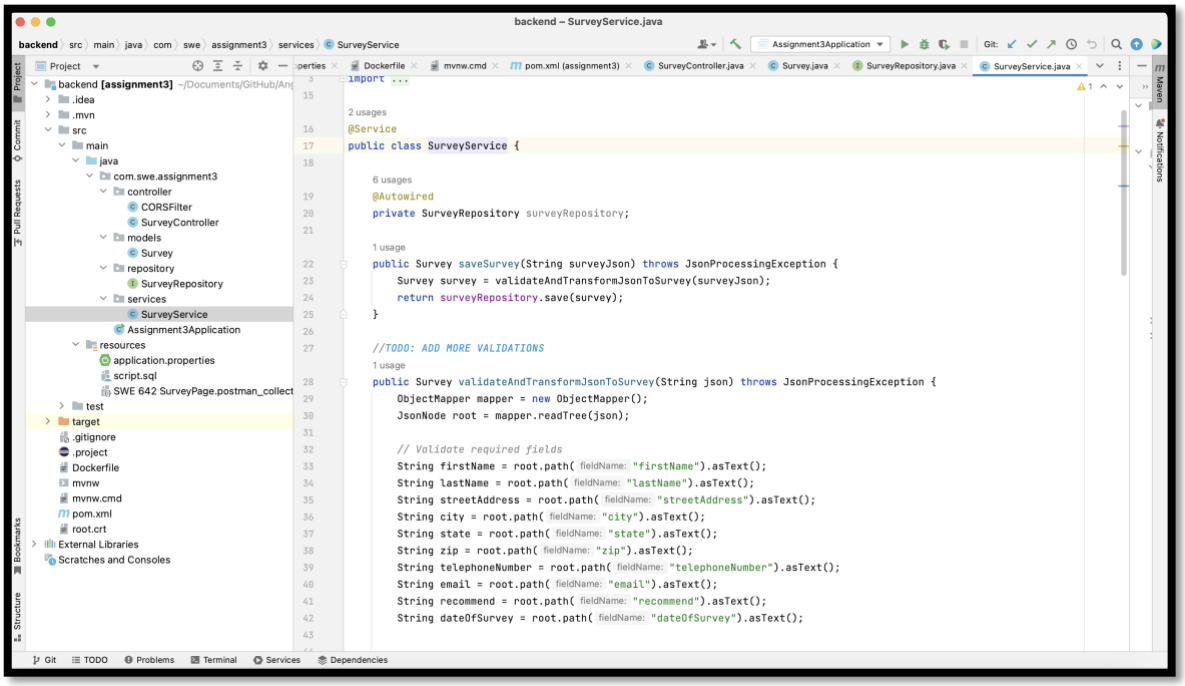
```java
package com.swe.assignment3.controller;

import ...

@RestController
@RequestMapping("/surveys")
public class SurveyController {

    @Autowired
    private SurveyService surveyService;

    @GetMapping
    public List<Survey> getAllSurveys() { return surveyService.getAllSurveys(); }

    @GetMapping("/{id}")
    public ResponseEntity<Survey> getSurveyById(@PathVariable Long id) {...}

    @PostMapping
    public ResponseEntity<?> saveSurvey(@RequestBody String surveyJson) {...}
```

We also created Survey Table Model class in the Model Package. It will map each entry of the table we created to store the survey information with Class attributes.

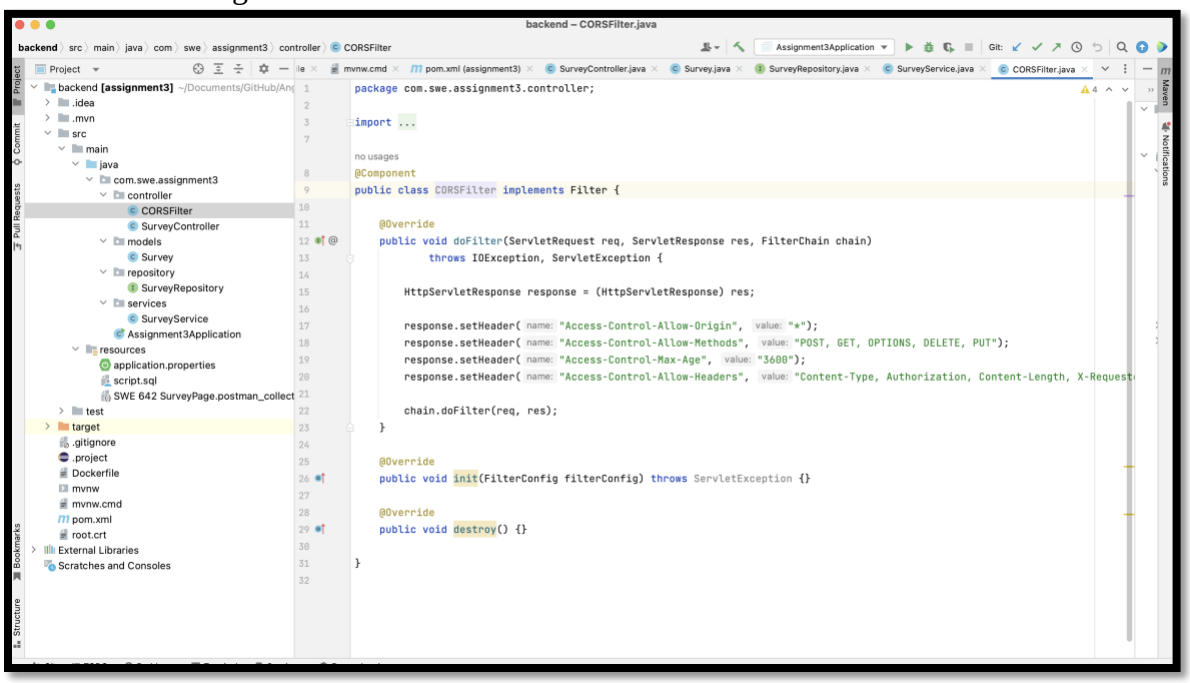


After that we created the SurveyRepository class for Survey Model Class extending the JpaRespository. Which extends the CRUD commands to communicate with the data base.

We created the Service class, which handles the Validation and JSON transform business logic before the Saving the Survey data into data base.



We also created a class to handle the CORS error. CROS is a Cross-Origin Resource Sharing. This error occurs when we the browser attempting to make a cross origin error. In our case we are attempting to make cross origin request between different ports. We are handling the CORS error with following code.



b. Dockerization of Spring Boot Application

```
FROM openjdk:latest

CMD ["mvn clean install"]

COPY target/assignment3-1.jar .

# Copy the SSL root certificate file to the container
COPY root.crt /root/.postgresql/

EXPOSE 8080

CMD ["java", "-jar", "assignment3-1.jar"]

# docker build -t survey_backend .
# docker run -p 8080:8080 --rm  --name survey_backend_container survey_backend
```

The file takes base image as openjdk, runs maven clean install to create artifacts and test project, copy the jar file and the certification for connecting with database server, expose 8080 port and then run the jar. This jar contains Springboot application.

c. Testing using Postman.

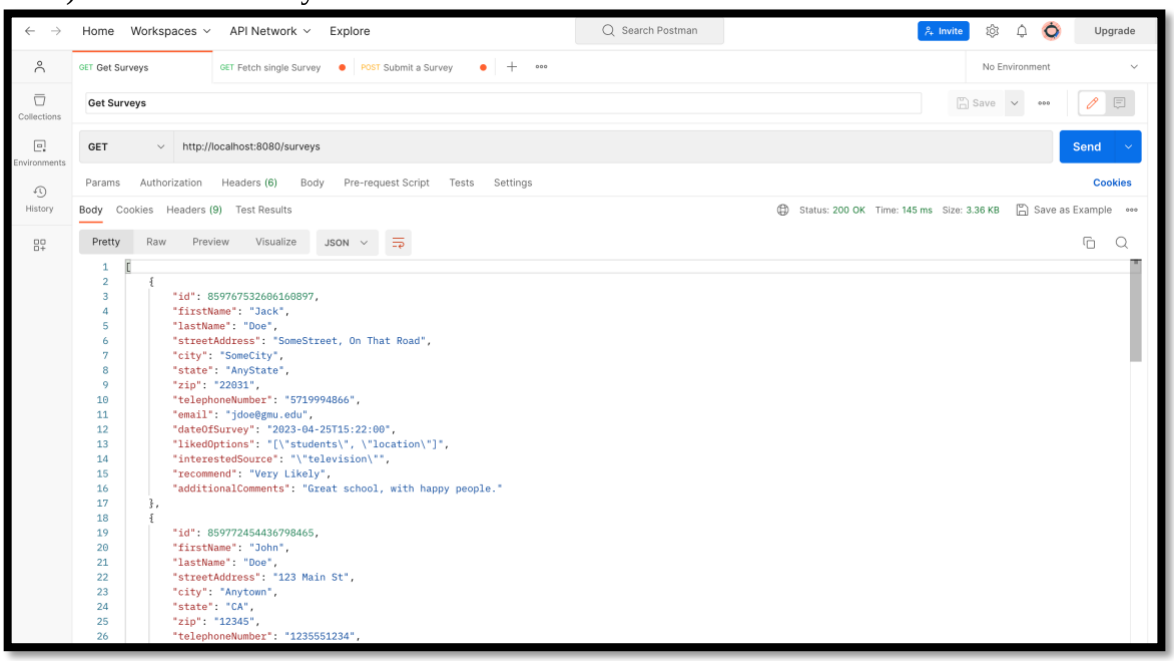
There are 4 different calls:

1. GET: Fetch content of surveys: Get all surveys and get one survey.
2. POST: Store a new survey to the database.
3. DELETE: Delete a survey from the database. (Not on UI)
4. PUT: Update/Edit an already created survey (Not on UI)

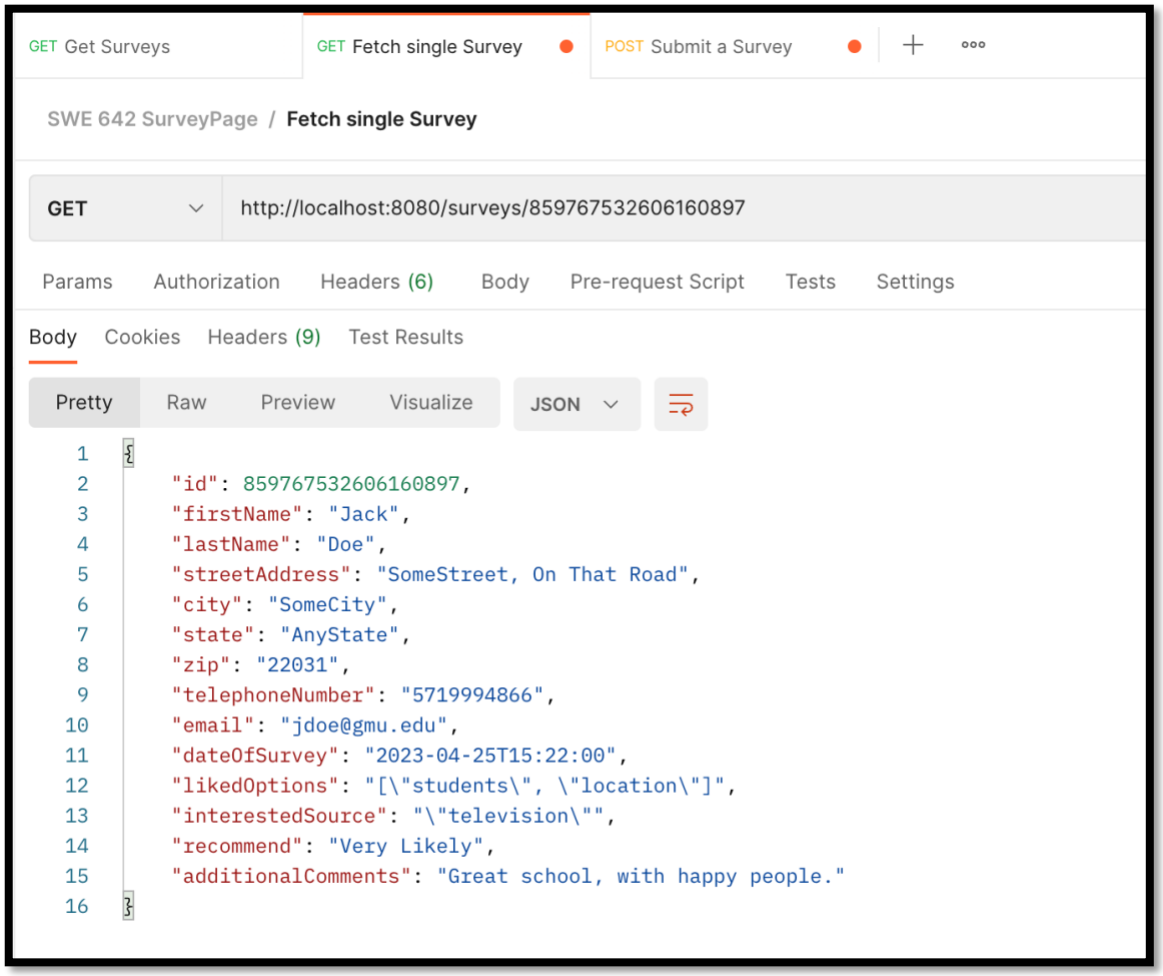
These are the testing calls using postman. (The postman ‘s collection is exported as a JSON file and uploaded to GitHub, and can be imported in postman to test the APIs,)

Sample Tests:

A) Get all the surveys.



B) Get a single survey.



C) Posting a new survey data

GET Get Surveys | GET Fetch single Survey | POST Submit a Survey

SWE 642 SurveyPage / **Submit a Survey**

Save

POST http://localhost:8080/surveys

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON

```
{
    "firstName": "Jane",
    "lastName": "Ryan",
    "streetAddress": "SomeStreet, On That Road",
    "city": "SomeCity",
    "state": "AnyState",
    "zip": "22031",
    "telephoneNumber": "5719994886",
    "email": "jryan@gmu.edu",
    "dateOfSurvey": "2023-04-27T15:22:00",
    "likedOptions": ["location"],
    "interestedSource": "television",
    "recommend": "Very Likely",
    "additionalComments": "Great school, with goood eating places."
}
```

Body | Cookies | Headers (9) | Test Results — Status: 201 Created | Time: 323 ms | Size: 792 B

Pretty | Raw | Preview | Visualize | JSON

```
{
    "id": 860365396324876289,
    "firstName": "Jane",
    "lastName": "Ryan",
    "streetAddress": "SomeStreet, On That Road",
    "city": "SomeCity",
    "state": "AnyState",
    "zip": "22031",
    "telephoneNumber": "5719994886",
    "email": "jryan@gmu.edu",
    "dateOfSurvey": "2023-04-27T15:22:00",
```

GET Get Surveys | GET Fetch single Survey | POST Submit a Survey

SWE 642 SurveyPage / **Submit a Survey**

POST http://localhost:8080/surveys

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON

Export Collection

SWE 642 SurveyPage
will be exported as a JSON file. Export as:

- Collection v2
- Collection v2.1 (recommended)

Learn more about collection formats

Sending your collection to a **teammate**? Save time by **sharing** it instead.

Share Collection | Learn more

Cancel | Export

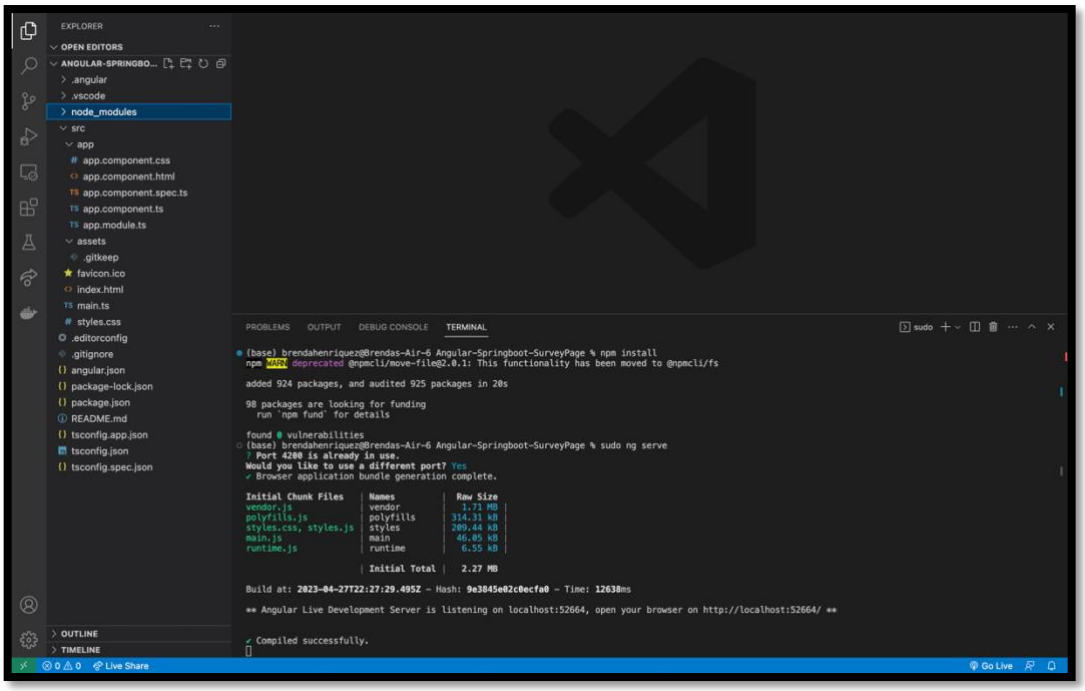
Pretty | Raw | Preview | Visualize | JSON

3. Frontend:
   a. Base Setup:
      i. Install Angular 2 and Angular 2 CLI (Command Line Interface)

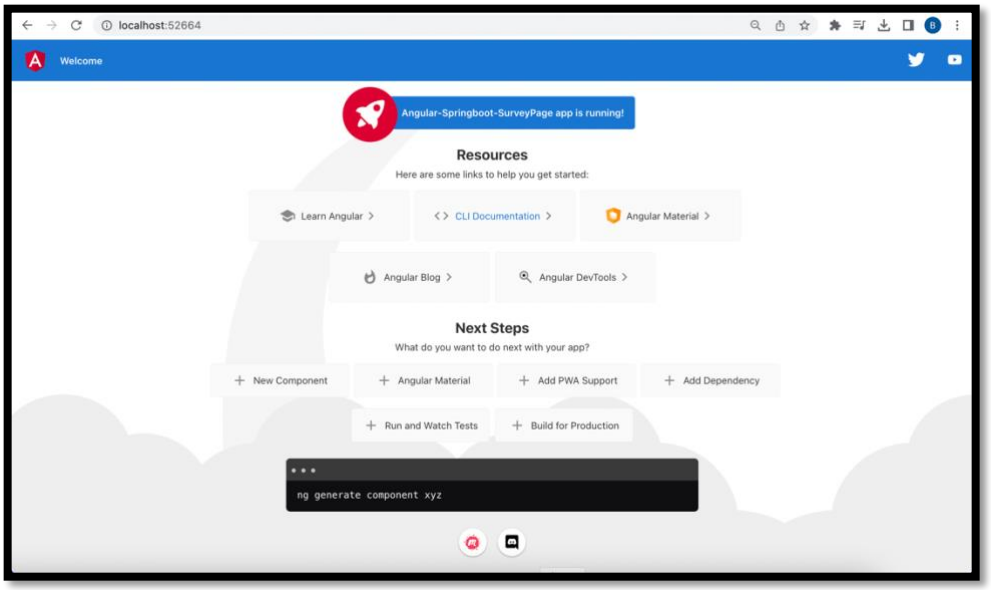
npm install -g angular-cli
ng new Angular-Springboot-SurveryPage

ng version
sudo npm install //to get all the dependencies.

npm install //adds node-modules
sudo ng serve //starts up a local development server



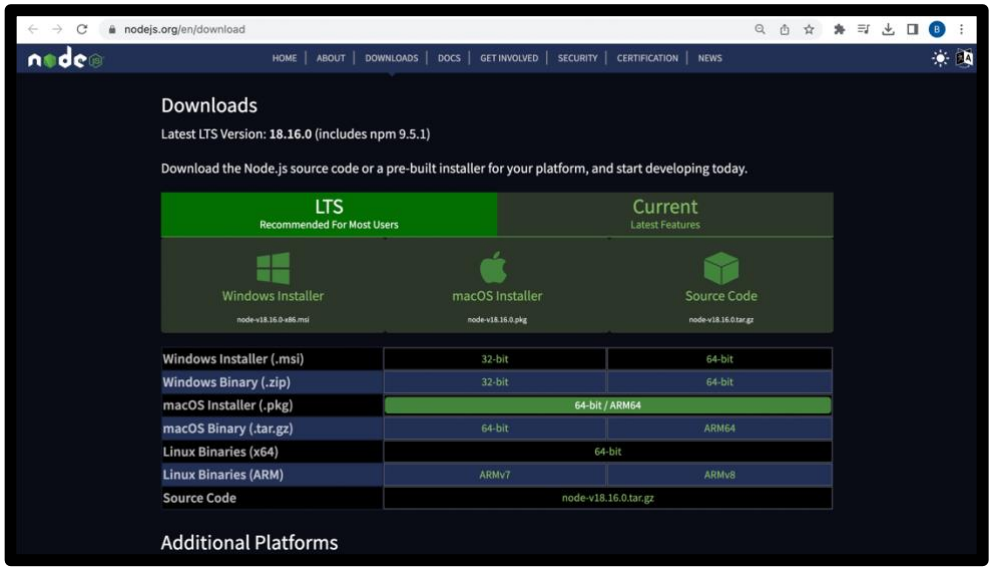
Angular app running on localhost:



ii. Install nodejs

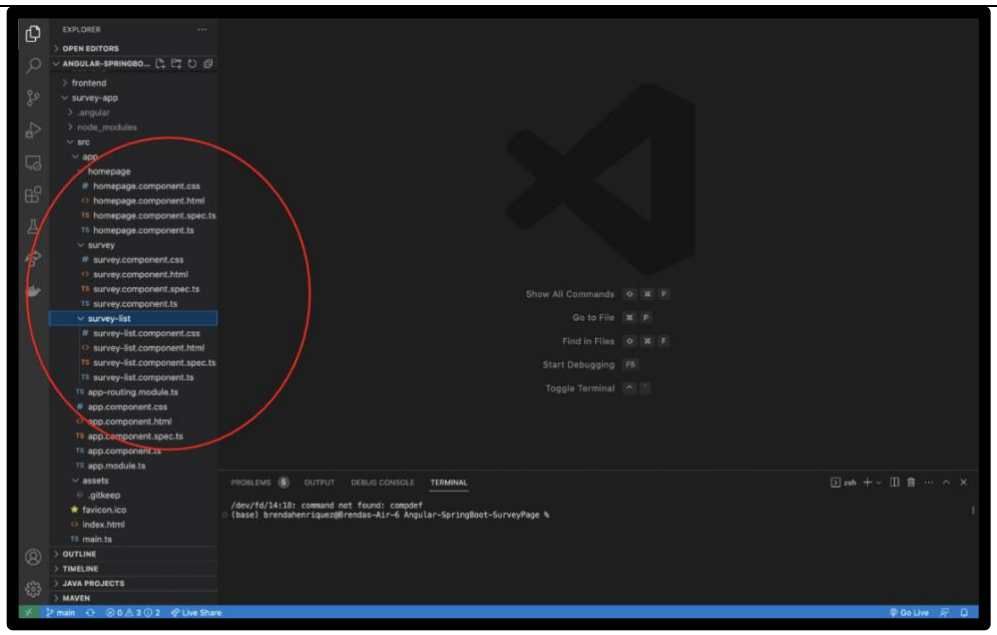
Download Node.js .pkg installer from https://nodejs.org/en/download

node -v to check version.

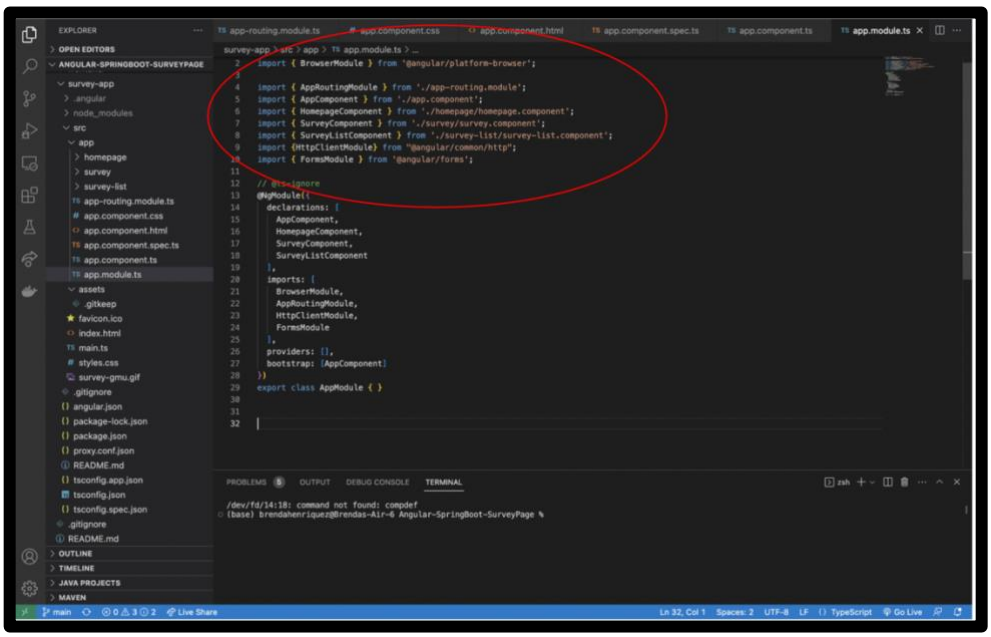


Frontend code structure:

We divided each page into a component (Homepage, Survey, and Survey-list) each with their own CSS, HTML, ts, and spec.ts file.



All components (Homepage, Survey, and Survey-list) are imported in app.module.ts from their respective folders (‘./homepage/hompage.component’, ‘./survey/survey.component’, ‘./survey-list/survey-list.component’).



Routing for the pages is done in app-routing.module.ts
homepageComponent has the path ''
survey has the path 'survey'
Survey-list has the path 'survey-list'

```
TS app-routing.module.ts ×
survey-app > src > app > TS app-routing.module.ts > ...
1   import { NgModule } from '@angular/core';
2   import { Routes, RouterModule } from '@angular/router';
3   import { HomepageComponent } from './homepage/homepage.component';
4   import { SurveyComponent } from './survey/survey.component';
5   import { SurveyListComponent } from './survey-list/survey-list.component';
6
7   const routes: Routes = [
8     { path: '', component: HomepageComponent },
9     { path: 'survey', component: SurveyComponent },
10    { path: 'survey-list', component: SurveyListComponent }
11  ];
12
13  @NgModule({
14    imports: [RouterModule.forRoot(routes)],
15    exports: [RouterModule]
16  })
17  export class AppRoutingModule { }
18
```

Component Structure:
We have three components (Homepage, Survey, and Survey-list) all with the same structure.
Homepage component contains 4 files:

- hompage.component.css
- homepage.component.html
- homepage.component.spec.ts
- homepage.component.ts

hompage.component.ts contains the component class called HomepageComponent.
The **@Component** decorator defines the metadata for the component.
The **selector** is a HTML tag that will represent the component in the DOM.
The **templateUrl** and **styleUrl** properties point to the HTML and CSS files that define the component UI.

```
TS homepage.component.ts ×
survey-app > src > app > homepage > TS homepage.component.ts > ...
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-homepage',
5     templateUrl: './homepage.component.html',
6     styleUrls: ['./homepage.component.css']
7   })
8   export class HomepageComponent {
9
10  }
11
```

homepage.component.spec.ts contains unit tests using the Jasmine testing framework.
The "should create" test case checks that the component is created successfully when the beforeEach hook is called. The beforeEach hook is used to set up the test environment before each test case.

```
TS homepage.component.spec.ts ●
survey-app > src > app > homepage > TS homepage.component.spec.ts > ...

import { HomepageComponent } from './homepage.component';

describe('HomepageComponent', () => {
  let component: HomepageComponent;
  let fixture: ComponentFixture<HomepageComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ HomepageComponent ]
    })
    .compileComponents();

    fixture = TestBed.createComponent(HomepageComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

4. Testing it together:

Step 1: Build docker-image for backend and run it:

```
janitbidhan@Janit-MBP assignment3 % cd backend
janitbidhan@Janit-MBP backend % mvn clean install
```

Once done, we would see a jar being generated under the target folder.

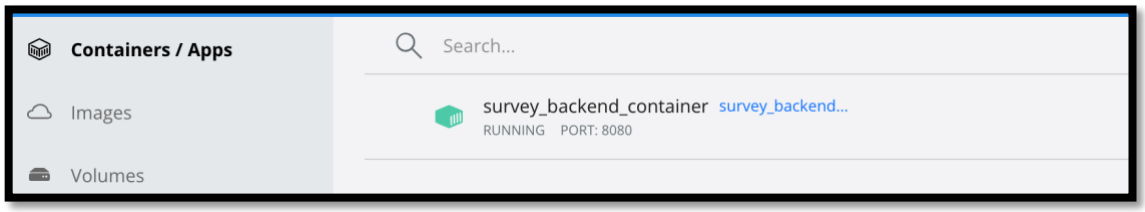
```
backend [assignment3] ~/SWE Projects/assig
  .idea
  .mvn
  src
  target
    classes
    generated-sources
    generated-test-sources
    maven-archiver
    maven-status
    surefire-reports
    test-classes
    assignment3-1.jar
    assignment3-1.jar.original
  .gitignore
  assignment3.iml
  Dockerfile
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
  root.crt
survey-app ~/SWE Projects/assignment3/surv
  .gitignore

FROM openjdk:latest

CMD ["mvn clean install"]

COPY target/assignment3-1.jar .

# Copy the SSL root certificate file to the container
COPY root.crt /root/.postgresql/

EXPOSE 8080

CMD ["java", "-jar", "assignment3-1.jar"]

# mvn clean install
# docker build -t survey_backend_642 .
# docker run -p 8080:8080 --rm  --name survey_backend_container survey_backend_642

Terminal:  Local
[INFO]
[INFO] --- install:3.0.1:install (default-install) @ assignment3 ---
[INFO] Installing /Users/janitbidhan/SWE Projects/assignment3/backend/pom.xml to /Users/janitbidhan/.m2/repository/com/swe/assignment3/1/assignment3-1.pom
[INFO] Installing /Users/janitbidhan/SWE Projects/assignment3/backend/target/assignment3-1.jar to /Users/janitbidhan/.m2/repository/com/swe/assignment3/1/assignment3-1.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  5.801 s
[INFO] Finished at: 2023-04-27T18:12:28-04:00
[INFO] ------------------------------------------------------------------------
```

Building docker file:

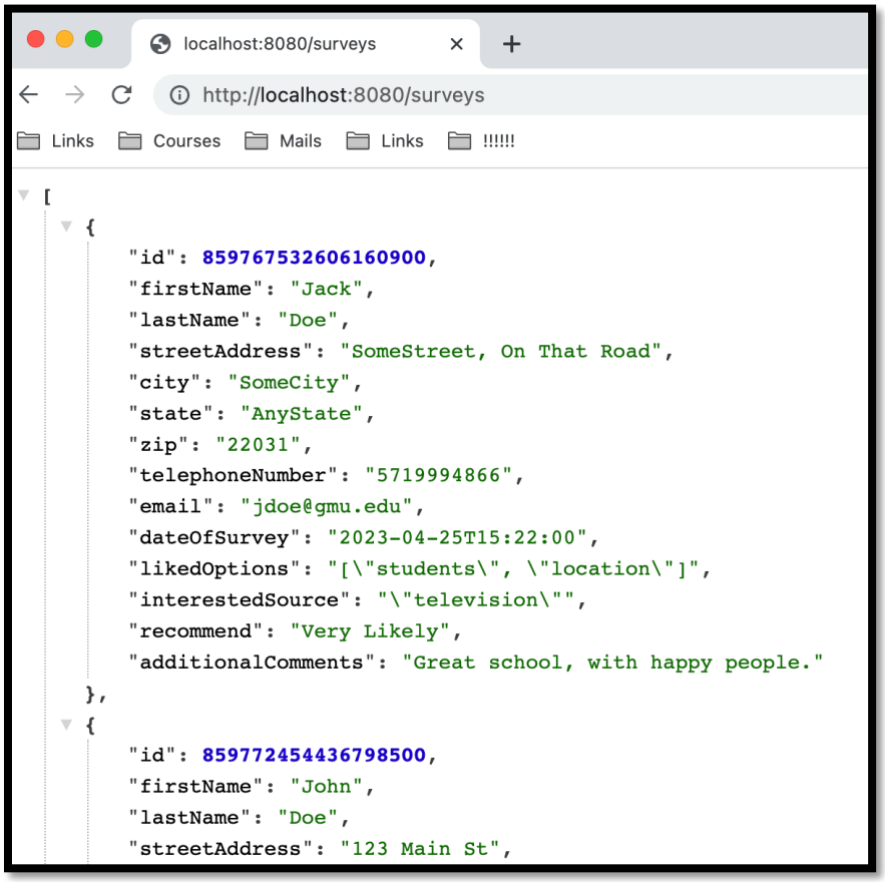
```
janitbidhan@Janit-MBP backend % docker build -t survey_backend_642 .
[+] Building 2.1s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 410B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/openjdk:latest
 => [auth] library/openjdk:pull token for registry-1.docker.io
 => [internal] load build context
 => => transferring context: 45.54MB
 => CACHED [1/3] FROM docker.io/library/openjdk:latest@sha256:9b448de897d211c9e0ec635a485650aed6e28d4eca1efbc34940560a480b3f1f
 => [2/3] COPY target/assignment3-1.jar .
 => [3/3] COPY root.crt /root/.postgresql/
 => exporting to image
 => => exporting layers
 => => writing image sha256:50760aed119f2dc9543afbb55f88c3923312bdd3a5e2cfb013d5daf064e3e0e7
 => => naming to docker.io/library/survey_backend_642
```

Running it on docker desktop:

```
janitbidhan@Janit-MBP backend % docker run -d -p 8080:8080 --rm  --name survey_backend_container survey_backend_642
e5ef09499086ce9cec8bb66e12e98bd563e3c68daa2f8c65bcca510dd6d48f0b
```

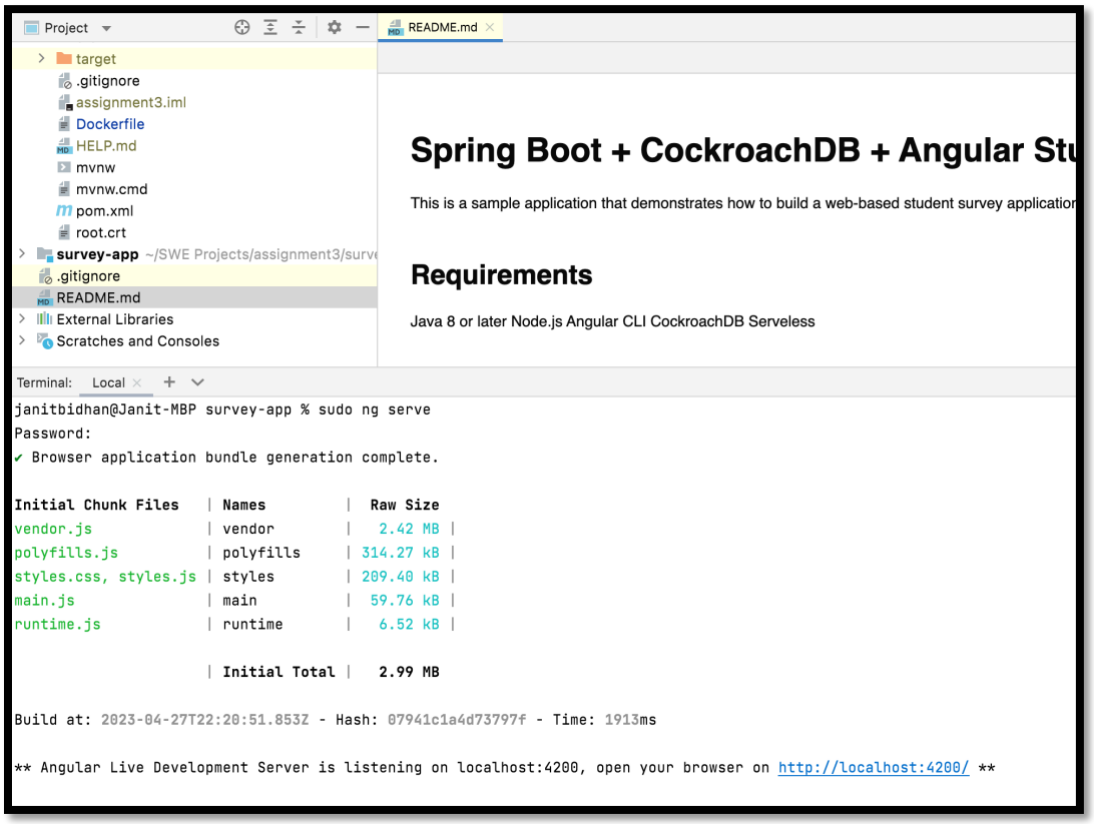


Testing if backend is up and running:



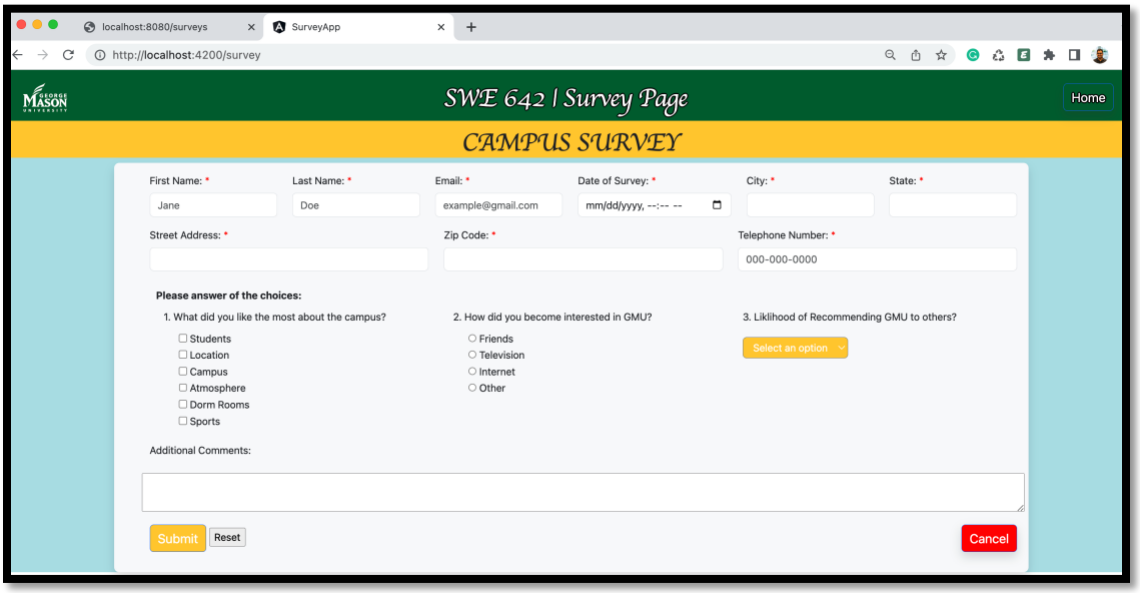
Now running frontend:

```
janitbidhan@Janit-MBP backend % cd ..
janitbidhan@Janit-MBP assignment3 % cd survey-app/
janitbidhan@Janit-MBP survey-app % sudo ng serve
```



Now running the application on hitting localhost:4200.

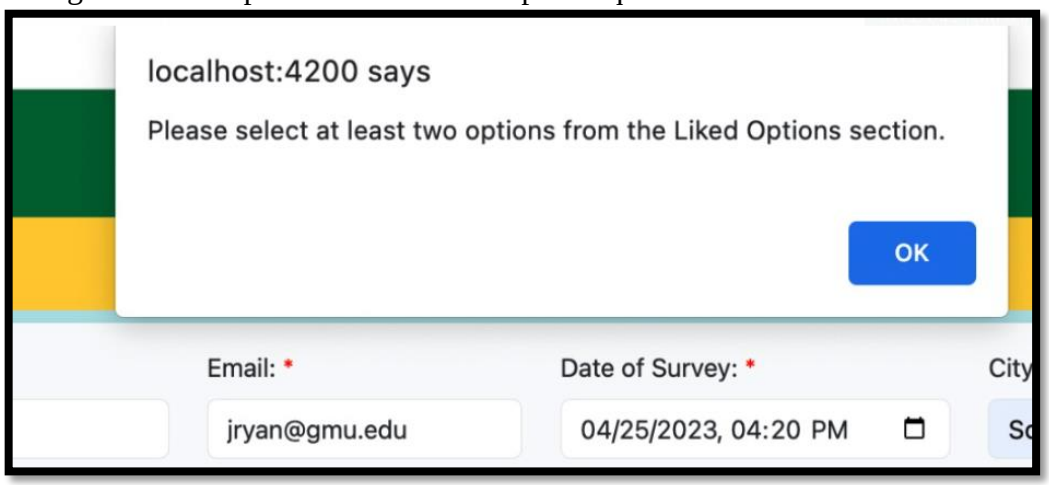
On clicking Student Survey



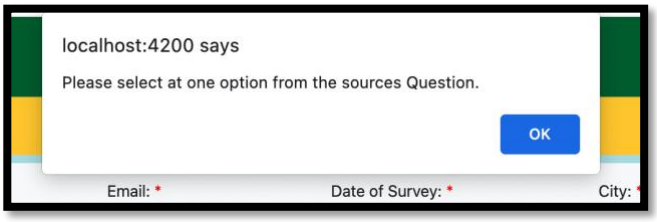
There are validations embedded in the UI:

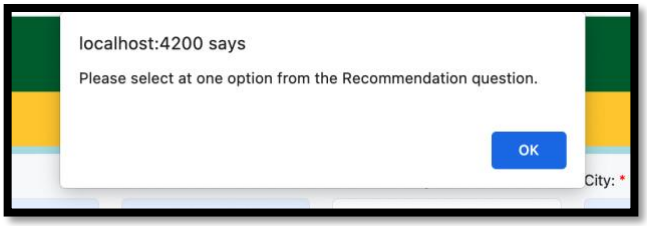
1. Missing required field like date of survey:

2. Missing to select 2 options in the linked Options question.

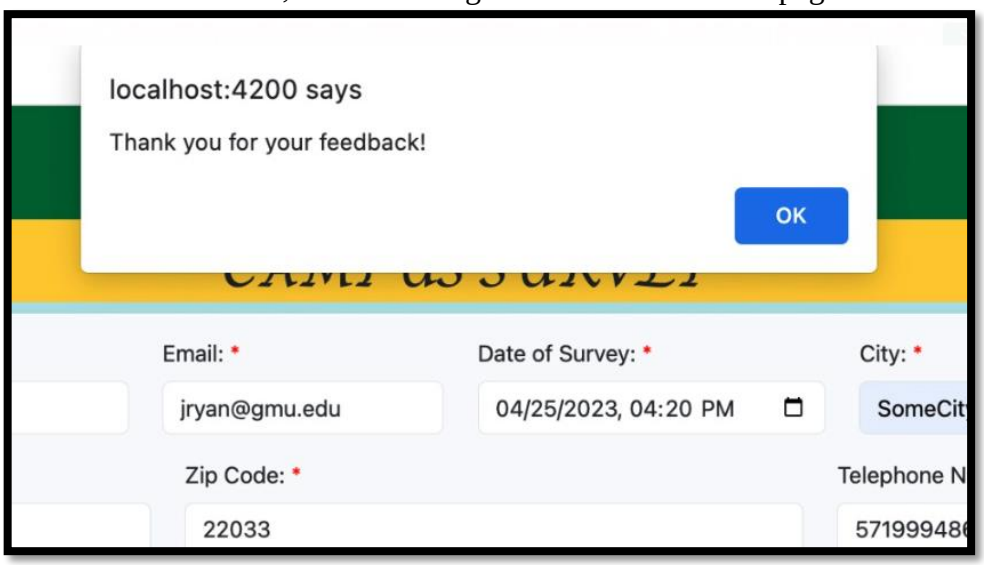


3. Missing to select the source or

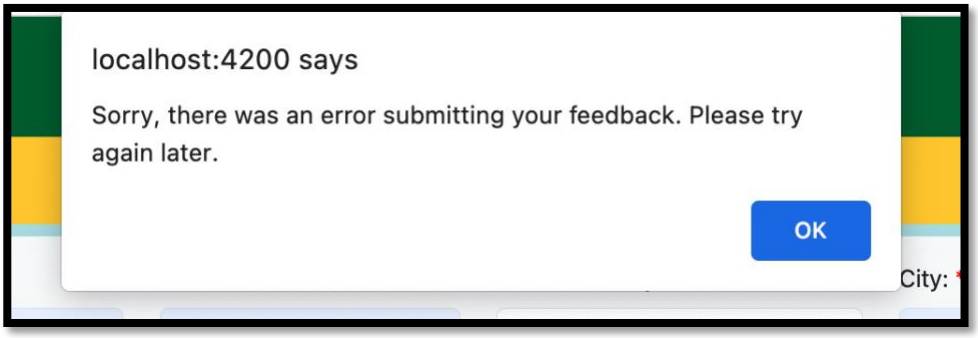




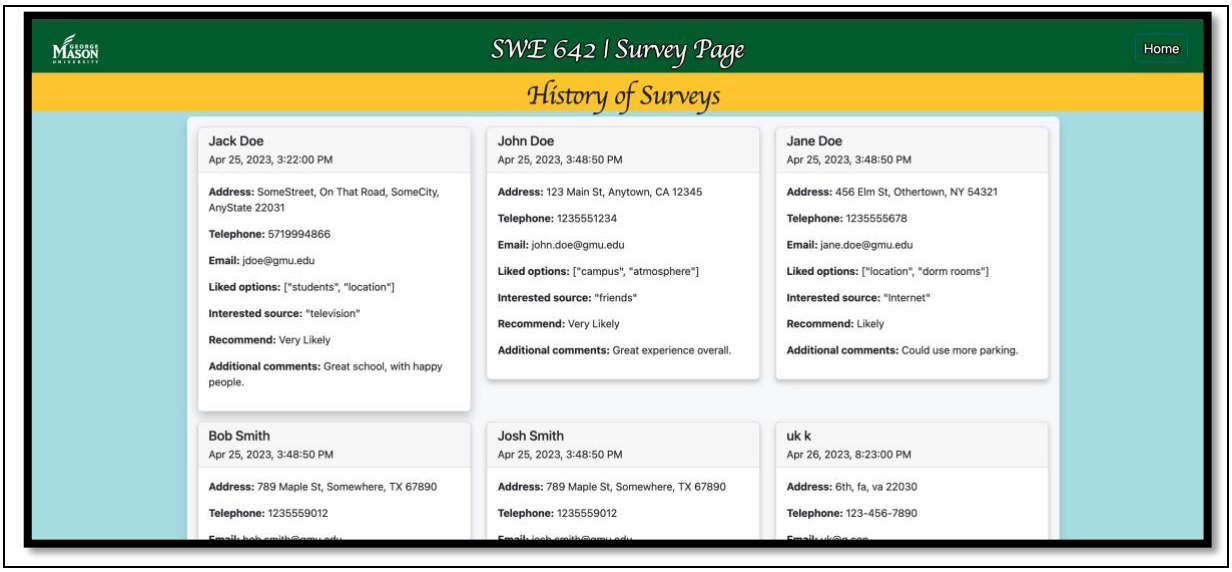
4. On Successful submission, shows message and redirects to homepage.



5. On backend failure or not able to reach out backend.



On clicking List All surveys:



Links and References:

1. https://spring.io/
2. https://start.spring.io/
3. https://www.cockroachlabs.com/docs/stable/build-a-spring-app-with-cockroachdb-jdbc.html
4. https://spring.io/guides/topicals/spring-boot-docker/
5. https://medium.com/bb-tutorials-and-thoughts/how-to-develop-and-build-angular-app-with-java-backend-87fb603c6e17
6. https://medium.com/@anooprvarrier/angular-reading-data-from-json-file-using-httpclient-3c46ba1aaf22
7. https://janitbidhan.medium.com/spring-boot-part-1-understanding-326a96abe2cb