

## Student Survey Page ([Youtube Demo Link](#))

Team:

Name	Email	GMU ID
Janit Bidhan	<a href="mailto:jbidhan@gmu.edu">jbidhan@gmu.edu</a>	G01326011
Rosy Sultana	<a href="mailto:rsultan6@gmu.edu">rsultan6@gmu.edu</a>	G01187642
Brenda Henriquez	<a href="mailto:bhenriqu@gmu.edu">bhenriqu@gmu.edu</a>	G01139846
Uday Kumar Kamalapuram	<a href="mailto:ukamalap@gmu.edu">ukamalap@gmu.edu</a>	G01340201

Prerequisites for the assignment:

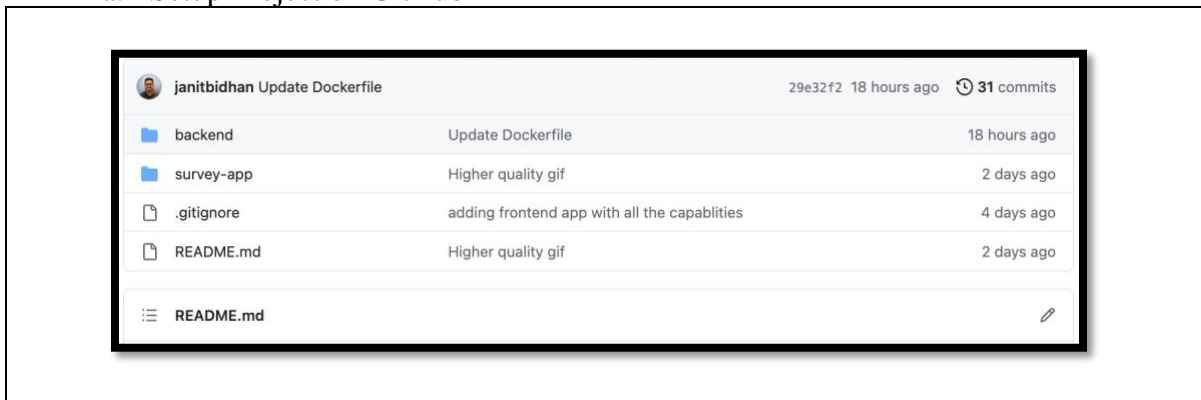
1. IDE for development.
2. Postman for API testing
3. DBeaver for Database testing/setup
4. Docker Desktop for building and testing local images.
5. Personal accounts on GitHub, Google Cloud Platform and Docker Hub.

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.



## Assignment 3 – SWE 642

Create your cluster

Choose a Plan

**Serverless**  
Highly available clusters that scale instantly. Only pay for what you use.  
Get started for free  
250M Request Units free  
5 GiB Storage free

**Dedicated**  
Dedicated single-tenant clusters starting at \$250/month.  
Free 30-Day trial  
Multi-region capabilities  
VPC Peering and IP allowlisting

Cloud provider: AWS

Regions: N. Virginia (us-east-1)

Spend limit: \$0.00

Cluster name: second-sage

**Summary**

Plan: Serverless

Cloud: AWS

Region: N. Virginia (us-east-1)

Performance: Run queries and use storage up to your spend limit.

Is this a production cluster? ☐ No

\$0.00 maximum/month

Create your free cluster

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

CockroachDB

< db-bittales AVAILABLE | Serverless / AWS v22.2.7

Overview

**Cluster settings**

CLOUD: AWS

PLAN TYPE: Serverless

REGION: N. Virginia (us-east-1) PRIMARY

**Usage this month**

Estimate usage cost

SPEND LIMIT: Add a spend limit

STORAGE: 10 MiB / 5 GiB

REQUEST UNITS: 249.5K / 145.05M

Press connect and find the necessary details for connection.

Connect to db-bittales

Use the dropdowns to show connection configuration instructions for your connection method. You can always find these instructions later by selecting **Connect** on the cluster overview page. For more information [see the connection reference](#).

Select SQL user: bittales-user

Database: swe-db

Select option/language: General connection string

Download CA Cert (Required only once)

General connection string


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.

## Assignment 3 – SWE 642

**Connection settings**  
CockroachDB connection settings



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 password ☒


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

## Assignment 3 – SWE 642

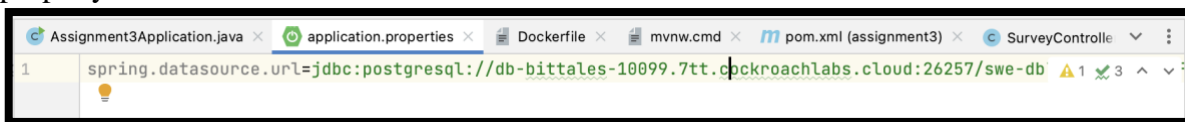
```
CREATE TABLE survey (  
    id SERIAL PRIMARY KEY,  
    first_name STRING NOT NULL,  
    last_name STRING NOT NULL,  
    street_address STRING NOT NULL,  
    city STRING NOT NULL,  
    state STRING NOT NULL,  
    zip STRING NOT NULL,  
    telephone_number STRING NOT NULL,  
    email STRING 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/>

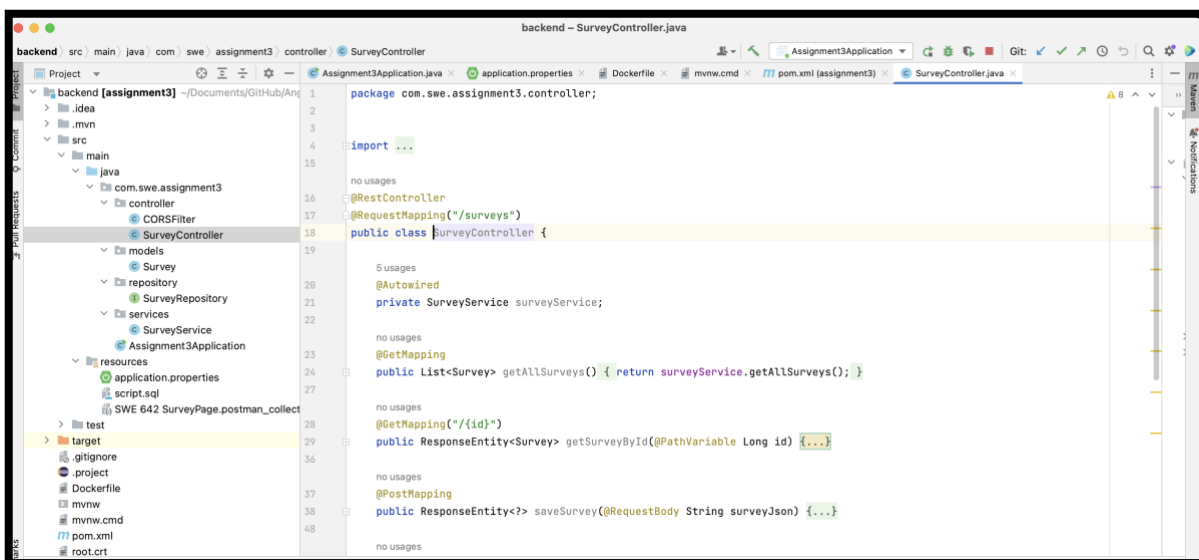
Downloaded the Spring Initializer project with required dependencies like Spring Web, Spring Data JPA and others. In application.properties file given the datasource.url using the property as below.



```
spring.datasource.url=jdbc:postgresql://db-bittales-10099.7tt.dockroachlabs.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.

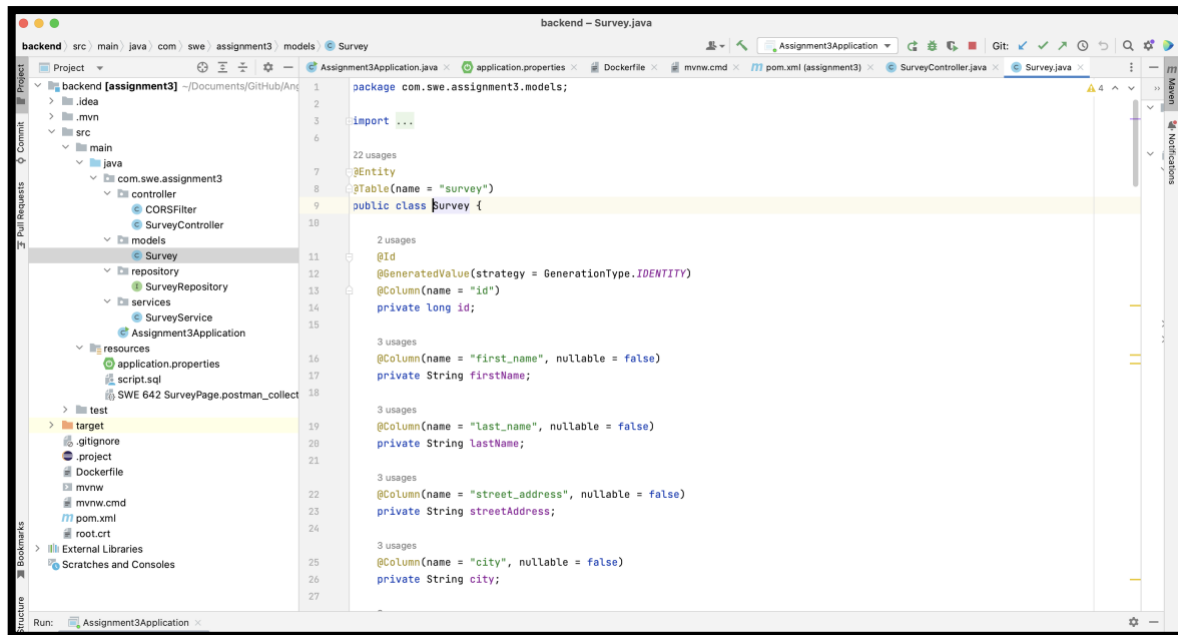
We are using @GetMapping to get the data from the database and send response back to the frontend. We are using @PostMapping to handle the Save Survey to store the new Survey object coming from Frontend into the database.



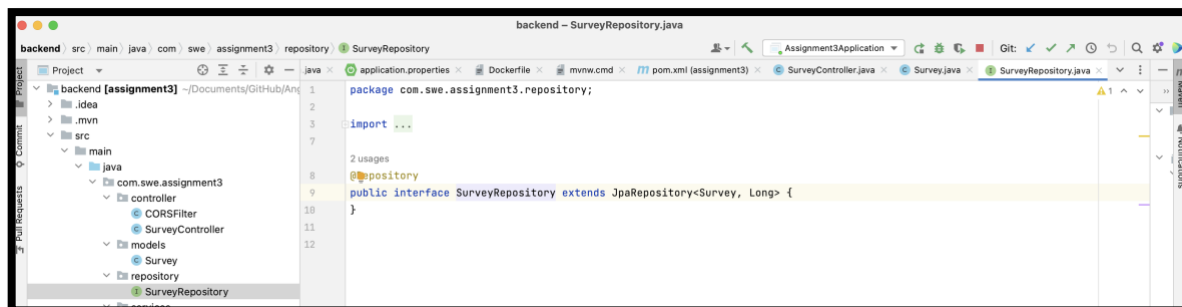
```
package com.swe.assignment3.controller;  
  
import ...  
  
@RestController  
@RequestMapping("/surveys")  
public class SurveyController {  
  
    @Autowired  
    private SurveyService surveyService;  
  
    @GetMapping  
    public List<Survey> getAllSurveys() { return surveyService.getAllSurveys(); }  
  
    @GetMapping("/{id}")  
    public ResponseEntity<Survey> getSurveyById(@PathVariable Long id) {...}  
  
    @PostMapping  
    public ResponseEntity<?> saveSurvey(@RequestBody String surveyJson) {...}
```

## Assignment 3 – SWE 642

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.

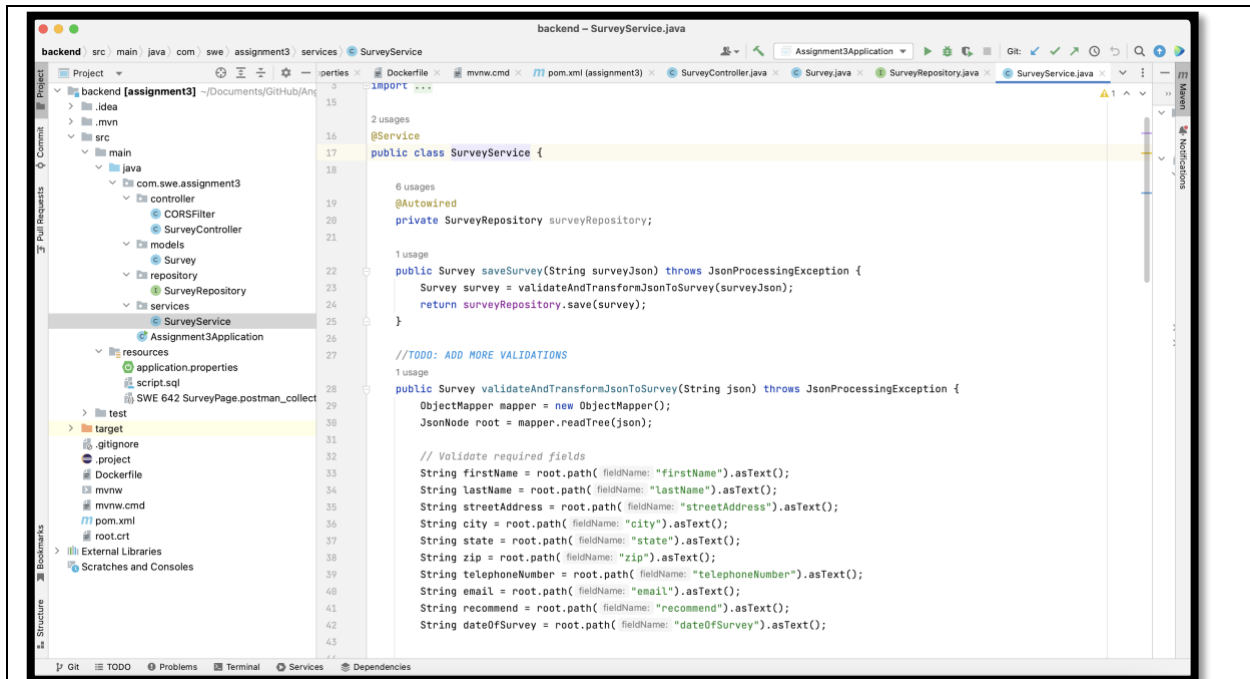


After that we created the SurveyRepository class for Survey Model Class extending the JpaRepository. Which extends the CRUD commands to communicate with the data base.

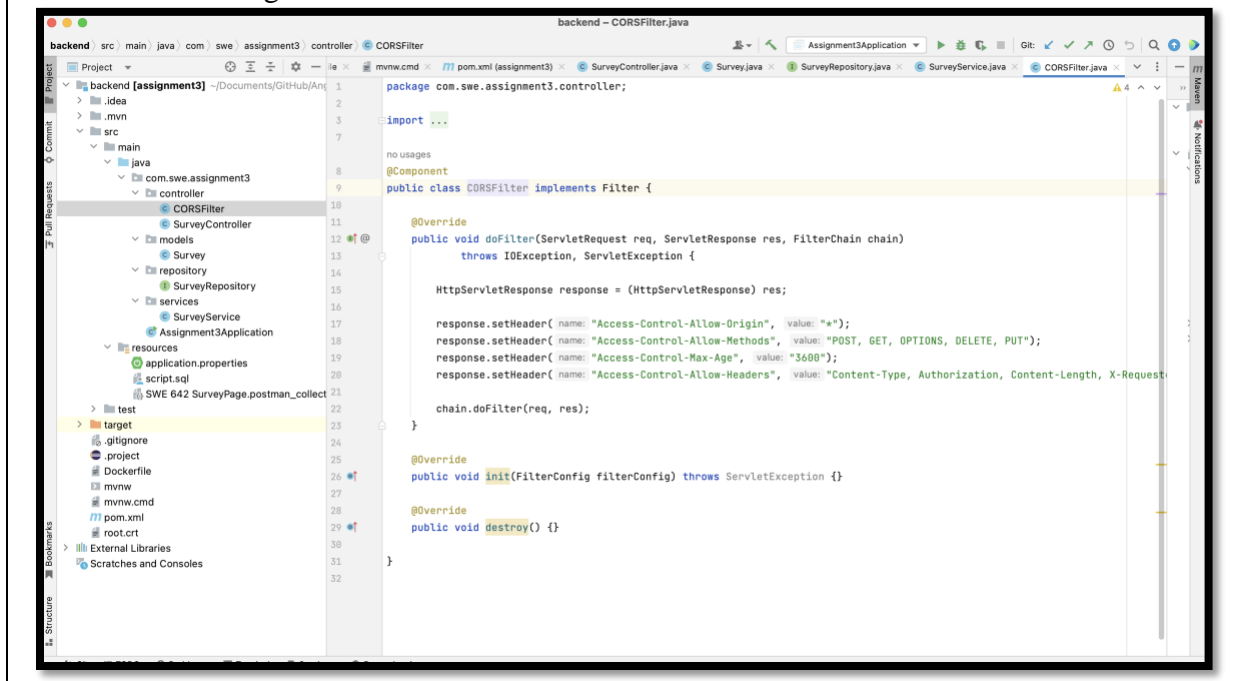


We created the Service class, which handles the Validation and JSON transform business logic before the Saving the Survey data into data base.

## Assignment 3 – SWE 642



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

## Assignment 3 – SWE 642

```
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.

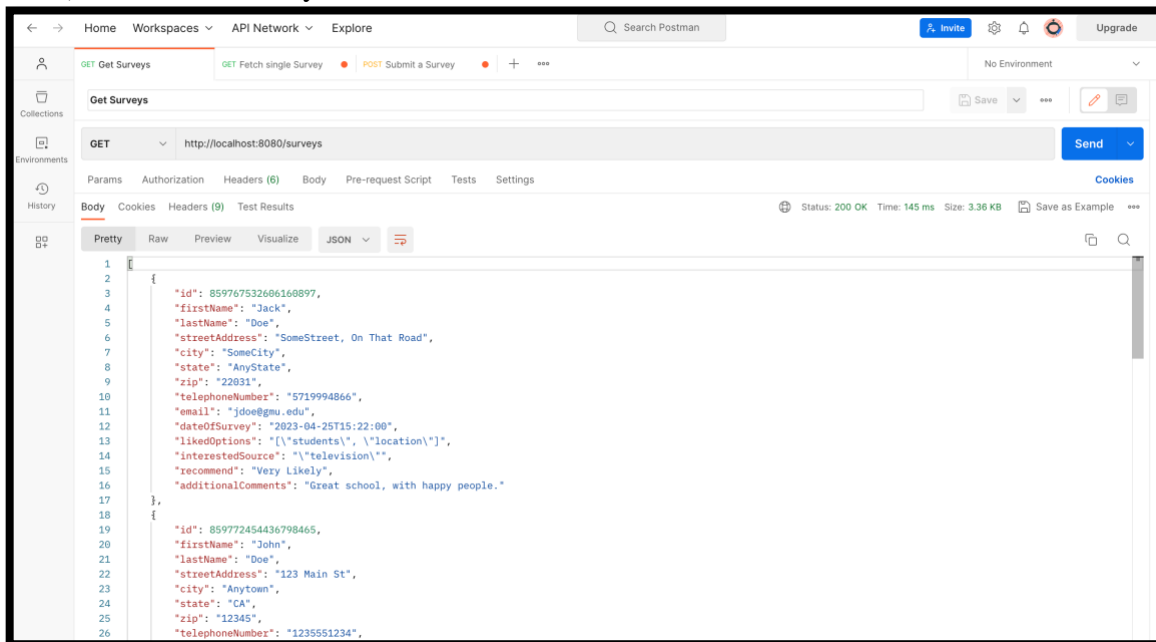
There are 4 different calls:

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.

## Assignment 3 – SWE 642

The screenshot shows a REST client interface with three tabs: "GET Get Surveys", "GET Fetch single Survey" (selected), and "POST Submit a Survey". The breadcrumb path is "SWE 642 SurveyPage / Fetch single Survey". The request method is "GET" and the URL is "http://localhost:8080/surveys/859767532606160897". The "Body" tab is selected, showing a JSON response in "Pretty" format. The response contains survey data for a user named Jack Doe.

GET ▼ http://localhost:8080/surveys/859767532606160897

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (9) Test Results

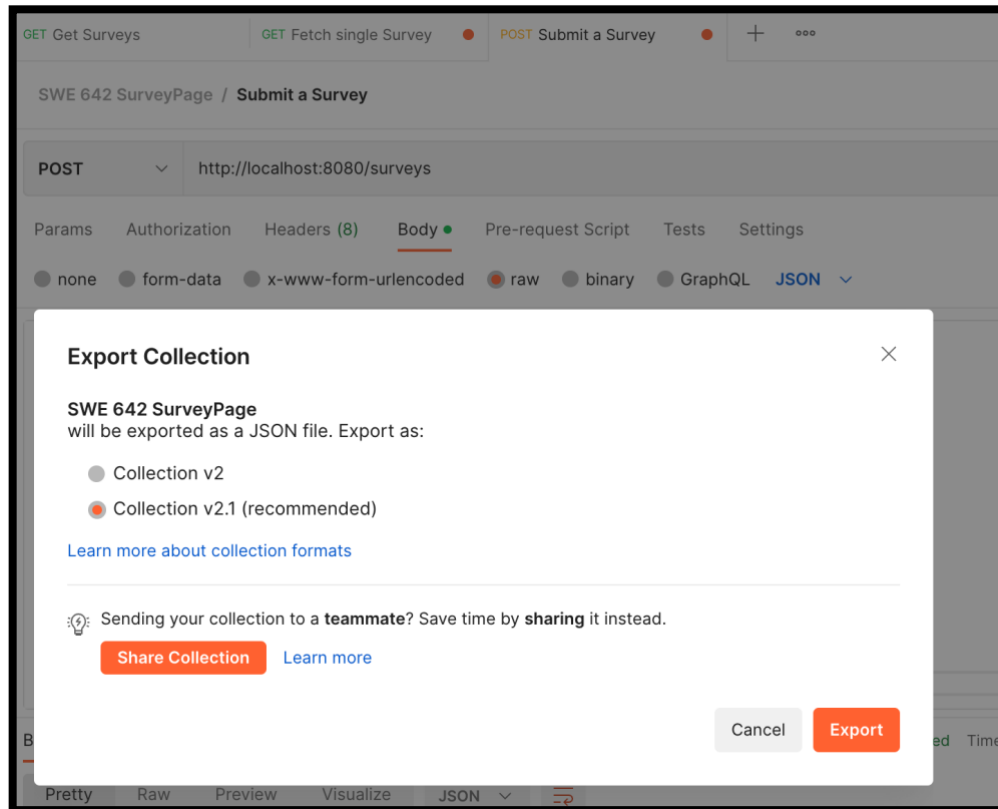
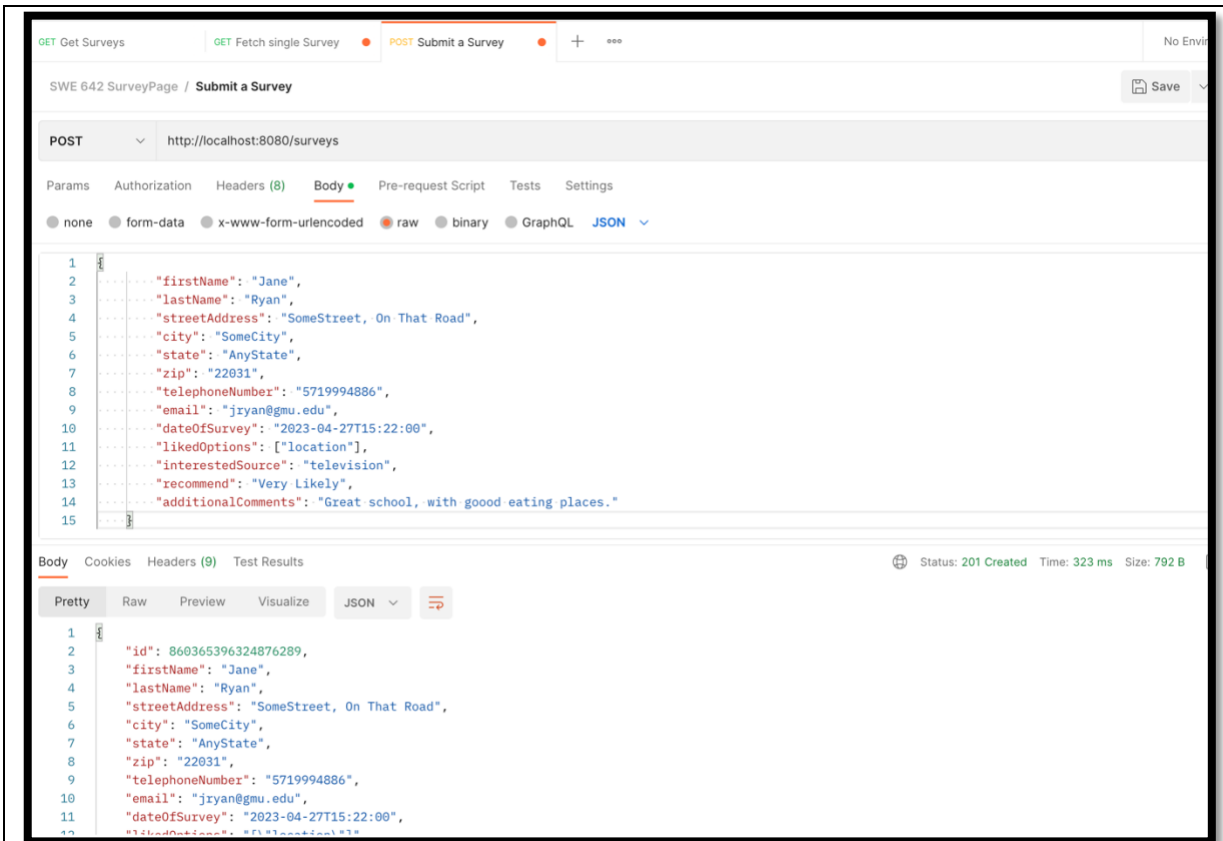
Pretty Raw Preview Visualize JSON ▼ ≡

```
1 {
2   "id": 859767532606160897,
3   "firstName": "Jack",
4   "lastName": "Doe",
5   "streetAddress": "SomeStreet, On That Road",
6   "city": "SomeCity",
7   "state": "AnyState",
8   "zip": "22031",
9   "telephoneNumber": "5719994866",
10  "email": "jdoe@gmu.edu",
11  "dateOfSurvey": "2023-04-25T15:22:00",
12  "likedOptions": ["students", "location"],
13  "interestedSource": "television",
14  "recommend": "Very Likely",
15  "additionalComments": "Great school, with happy people."
16 }
```

C) Posting a new survey data



## Assignment 3 – SWE 642



## 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-SurveyPage
```

```
ng version
```

```
sudo npm install //to get all the dependencies.
```

```
(base) brendahenriquez@Brendas-Air-6 Angular-SpringBoot-SurveyPage % ng version
```



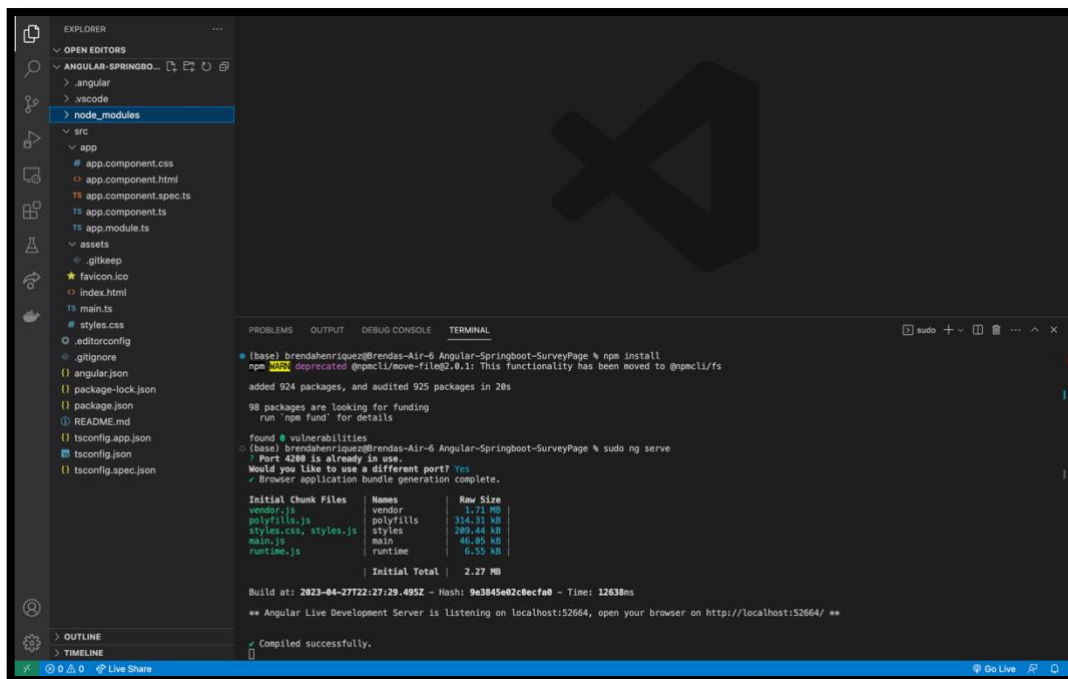
```
Angular CLI: 15.2.6
Node: 18.16.0
Package Manager: npm 9.5.1
OS: darwin x64

Angular:
...

Package           Version
-----
@angular-devkit/architect    0.1502.6 (cli-only)
@angular-devkit/core        15.2.6 (cli-only)
@angular-devkit/schematics   15.2.6 (cli-only)
@schematics/angular          15.2.6 (cli-only)
```

```
npm install //adds node-modules
```

```
sudo ng serve //starts up a local development server
```



```
(base) brendahenriquez@Brendas-Air-6 Angular-Springboot-SurveyPage % npm install
```

```
npm WARN deprecated @npmcli/move-file@2.0.1: This functionality has been moved to @npmcli/fs
```

```
added 924 packages, and audited 925 packages in 20s
```

```
98 packages are looking for funding
  run 'npm fund' for details
```

```
found 0 vulnerabilities
```

```
(base) brendahenriquez@Brendas-Air-6 Angular-Springboot-SurveyPage % sudo ng serve
```

```
Port 4200 is already in use.
Would you like to use a different port? Yes
```

```
Browser application bundle generation complete.
```

Initial Chunk Files	Names	Raw Size
vendor.js	vendor	1.71 MB
polyfills.js	polyfills	314.31 kB
styles.css, styles.js	styles	289.44 kB
main.js	main	46.05 kB
runtime.js	runtime	6.55 kB
Initial Total		2.27 MB

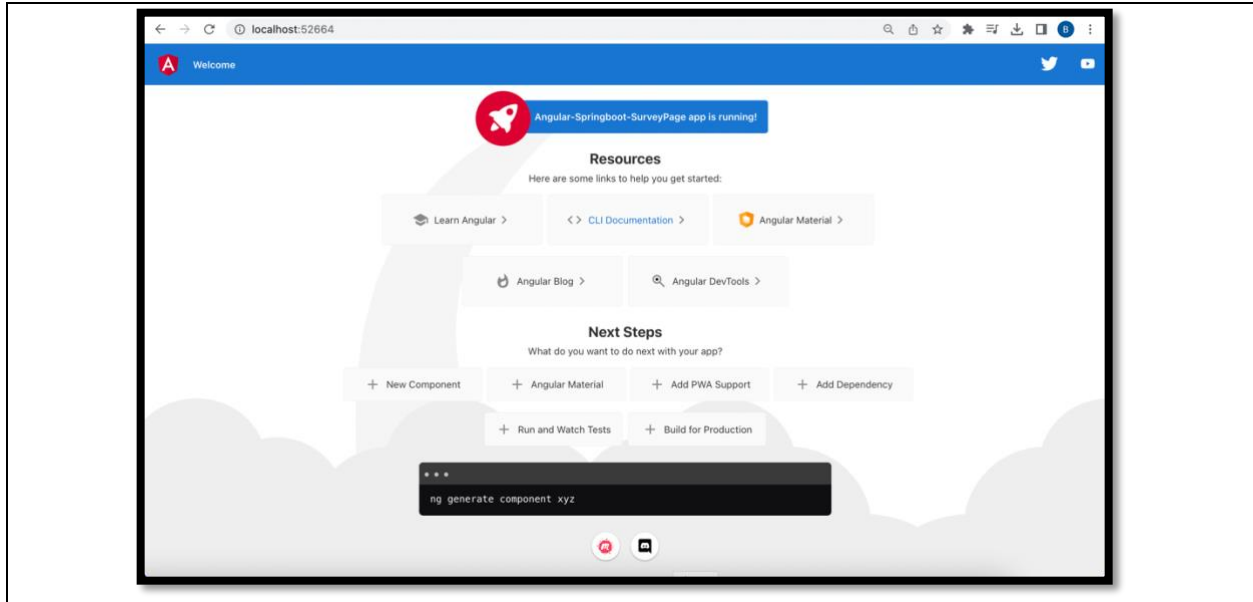
```
Build at: 2023-04-27T22:27:29.495Z - Hash: 9e3845e02c0ecfa0 - Time: 12638ms
```

```
** Angular Live Development Server is listening on localhost:52664, open your browser on http://localhost:52664/ **
```

```
Compiled successfully.
```

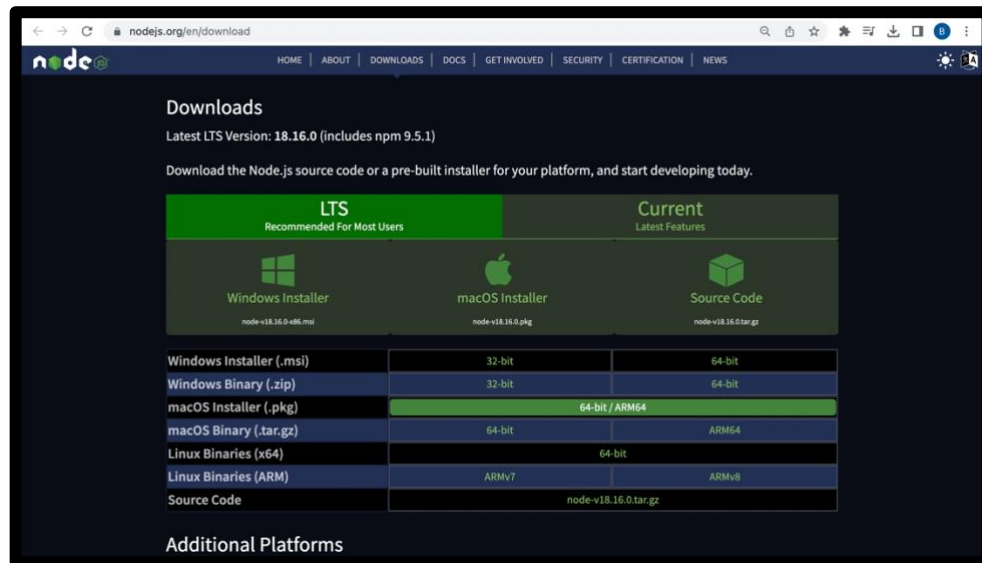
Angular app running on localhost:

## Assignment 3 – SWE 642



### ii. Install nodejs

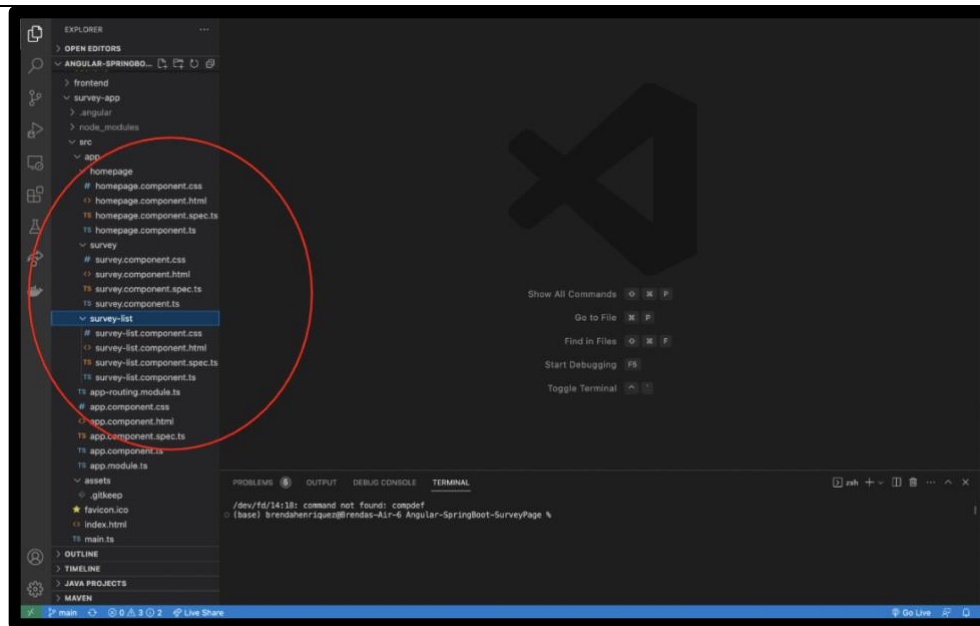
Download Node.js .pkg installer from <https://nodejs.org/en/download>  
node -v to check version.



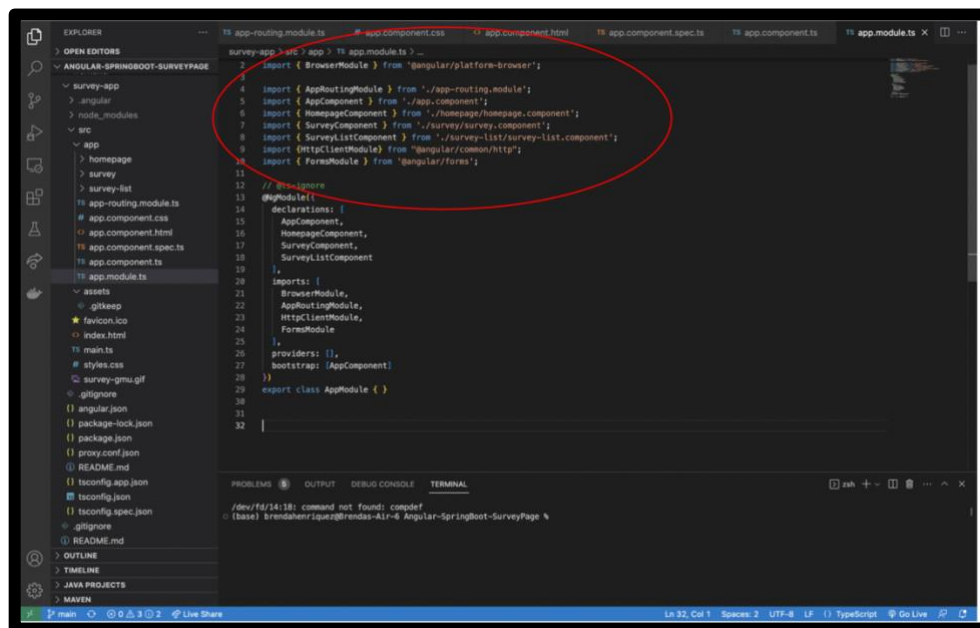
Frontend code structure:

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

## Assignment 3 – SWE 642



All components (Homepage, Survey, and Survey-list) are imported in app.module.ts from their respective folders ('./homepage/homepage.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 X
survey-app > src > app > TS app-routing.module.ts > ...
1 import { NgModule } from '@angular/core';
2 import { Routes, RouterModule } from '@angular/router';
3 import { HomeComponent } 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: HomeComponent },
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:

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

homepage.component.ts contains the component class called HomeComponent.

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 X
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 HomeComponent {
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.

## Assignment 3 – SWE 642

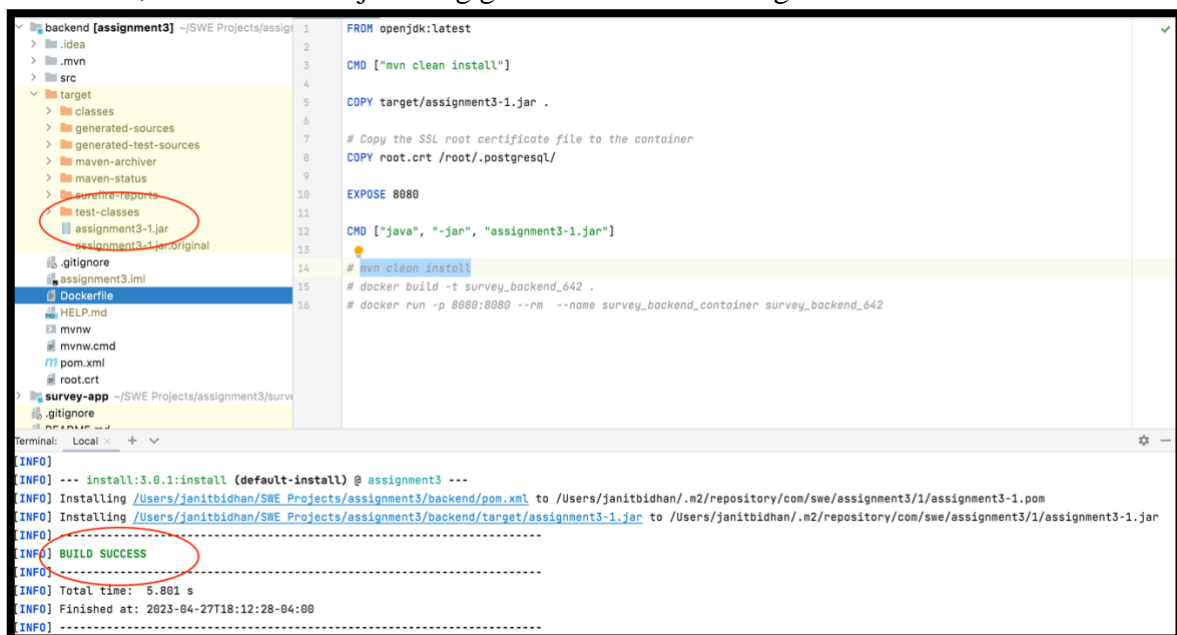
```
TS homepage.component.spec.ts •
survey-app > src > app > homepage > TS homepage.component.spec.ts > ...
2
3 import { HomepageComponent } from './homepage.component';
4
5 describe('HomepageComponent', () => {
6   let component: HomepageComponent;
7   let fixture: ComponentFixture<HomepageComponent>;
8
9   beforeEach(async () => {
10     await TestBed.configureTestingModule({
11       declarations: [ HomepageComponent ]
12     })
13     .compileComponents();
14
15     fixture = TestBed.createComponent(HomepageComponent);
16     component = fixture.componentInstance;
17     fixture.detectChanges();
18   });
19
20   it('should create', () => {
21     expect(component).toBeTruthy();
22   });
23 });
24
```

### 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.



```
FROM openjdk:latest
2
3 CMD ["mvn clean install"]
4
5 COPY target/assignment3-1.jar .
6
7 # Copy the SSL root certificate file to the container
8 COPY root.crt /root/.postgresql/
9
10 EXPOSE 8080
11
12 CMD ["java", "-jar", "assignment3-1.jar"]
13
14 # mvn clean install
15 # docker build -t survey_backend_642 .
16 # docker run -p 8080:8080 --rm --name survey_backend_container survey_backend_642
```

```
[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] BUILD SUCCESS
[INFO] Total time: 5.801 s
[INFO] Finished at: 2023-04-27T18:12:28-04:00
[INFO]
```

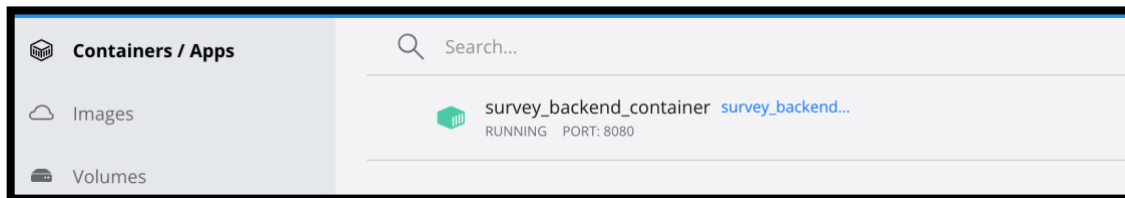
Building docker file:

## Assignment 3 – SWE 642

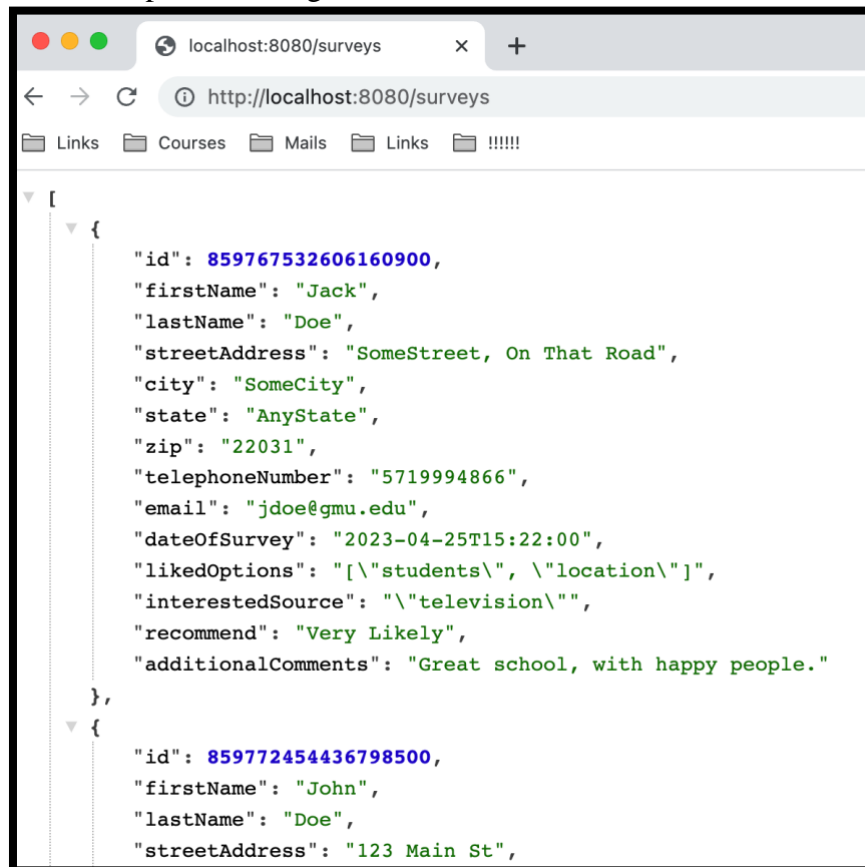
```
janitbidhan@Janit-MBP backend % docker build -t survey_backend_642 .
[+] Building 2.1s (9/9) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 410B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/openjdk:latest
=> [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
```

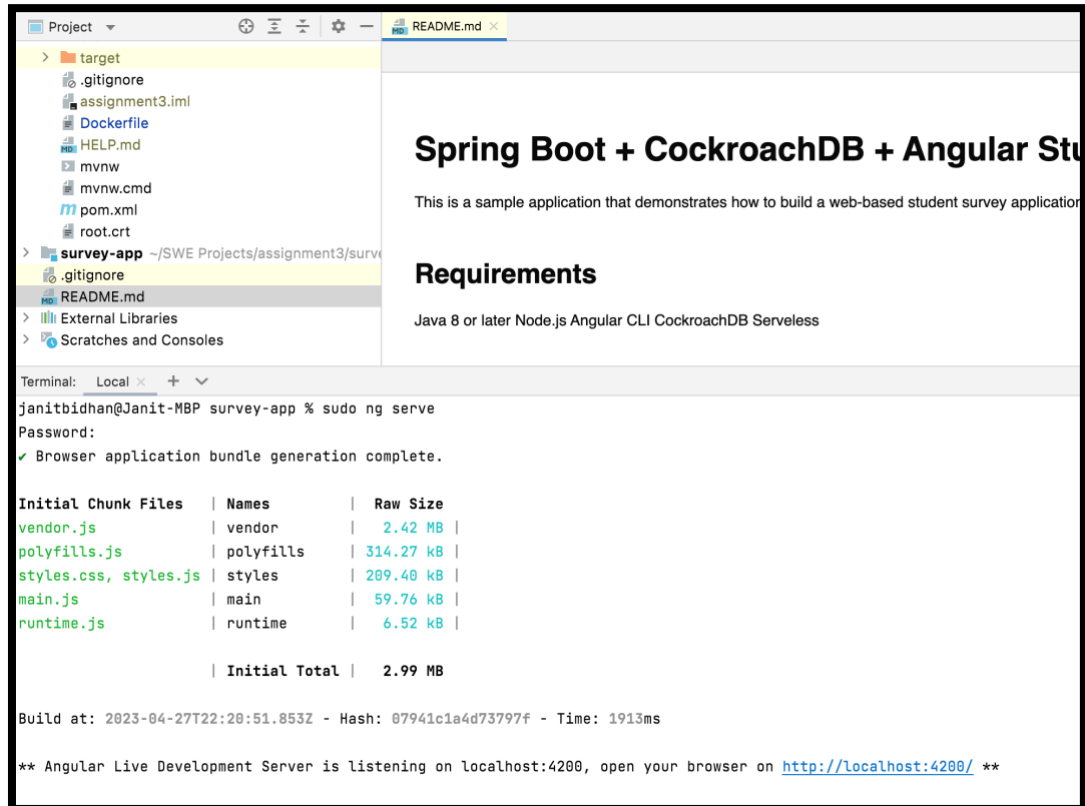


Testing if backend is up and running:

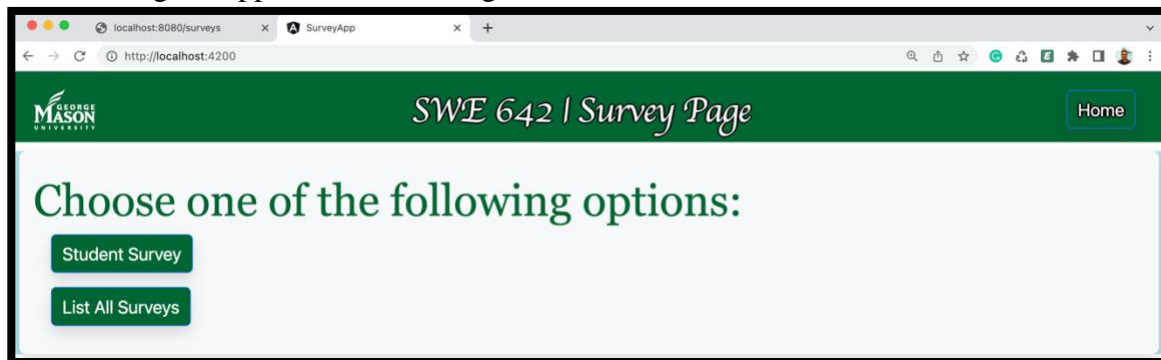


Now running frontend:

```
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



## Assignment 3 – SWE 642

The screenshot shows a web browser window with the URL `http://localhost:4200/survey`. The page has a green header with the text "SWE 642 | Survey Page" and a "Home" link. Below the header is a yellow banner with the text "CAMPUS SURVEY". The form contains several input fields: "First Name" (Jane), "Last Name" (Doe), "Email" (example@gmail.com), "Date of Survey" (mm/dd/yyyy, --:-- --), "City" (empty), "State" (empty), "Street Address" (empty), "Zip Code" (empty), and "Telephone Number" (000-000-0000). There are three sections of choices: "1. What did you like the most about the campus?" with checkboxes for Students, Location, Campus, Atmosphere, Dorm Rooms, and Sports; "2. How did you become interested in GMU?" with radio buttons for Friends, Television, Internet, and Other; and "3. Likelihood of Recommending GMU to others?" with a "Select an option" dropdown. At the bottom, there is an "Additional Comments" text area, "Submit" and "Reset" buttons, and a "Cancel" button.

There are validations embedded in the UI:

1. Missing required field like date of survey:

The screenshot shows a validation message box with the text "localhost:4200 says Please fill out required fields." and an "OK" button. Below the message box, the form fields are visible: "Name" (han), "Email" (janitbidhan@gmail.com), "Date of Survey" (mm/dd/yyyy, --:-- --), "City" (Fairfax), "Zip Code" (22030), and "Telephone Number" (15719994865).

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

The screenshot shows a validation message box with the text "localhost:4200 says Please select at least two options from the Liked Options section." and an "OK" button. Below the message box, the form fields are visible: "Email" (jryan@gmu.edu), "Date of Survey" (04/25/2023, 04:20 PM), and "City" (So).

3. Missing to select the source or

### Assignment 3 – SWE 642

localhost:4200 says

Please select at one option from the sources Question.

OK

Email: \* Date of Survey: \* City: \*

localhost:4200 says

Please select at one option from the Recommendation question.

OK

City: \*

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

localhost:4200 says

Thank you for your feedback!

OK

Email: \* Date of Survey: \* City: \*

jryan@gmu.edu 04/25/2023, 04:20 PM SomeCity

Zip Code: \* Telephone N

22033 571999486

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

localhost:4200 says

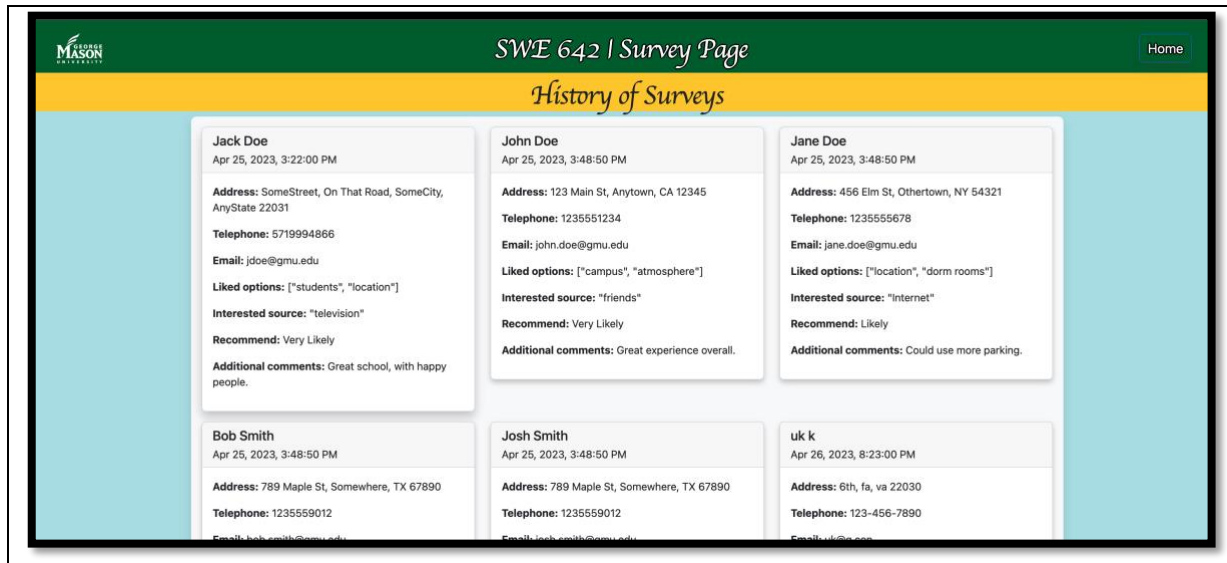
Sorry, there was an error submitting your feedback. Please try again later.

OK

City: \*

On clicking List All surveys:

## Assignment 3 – SWE 642



### 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>