SWE642-ASSIGNMENT 3

FULL STACK APPLICATION DEVELOPMENT

1. Team Members

- Meet Rajesh Popat - G01506756
- Tanuja Konda Reddy - G01507320
- Ritika Prashanth - G01487838
- Manasa Mummadi - G01515437

2. Introduction

This project implements a full-stack web application to collect and manage student survey data. The frontend is built with Angular, and the backend is a Spring Boot REST API using JPA/Hibernate with MySQL hosted on Amazon RDS. Users can submit a survey and view, update, or delete existing surveys.

3. System Architecture

- Frontend: Angular SPA using standalone components and Reactive Forms.
- Backend: Spring Boot REST controllers exposing CRUD endpoints.
- Database: MySQL on Amazon RDS accessed via JPA/Hibernate.
- Communication: JSON over HTTP between Angular and Spring Boot.

4. Technology Stack

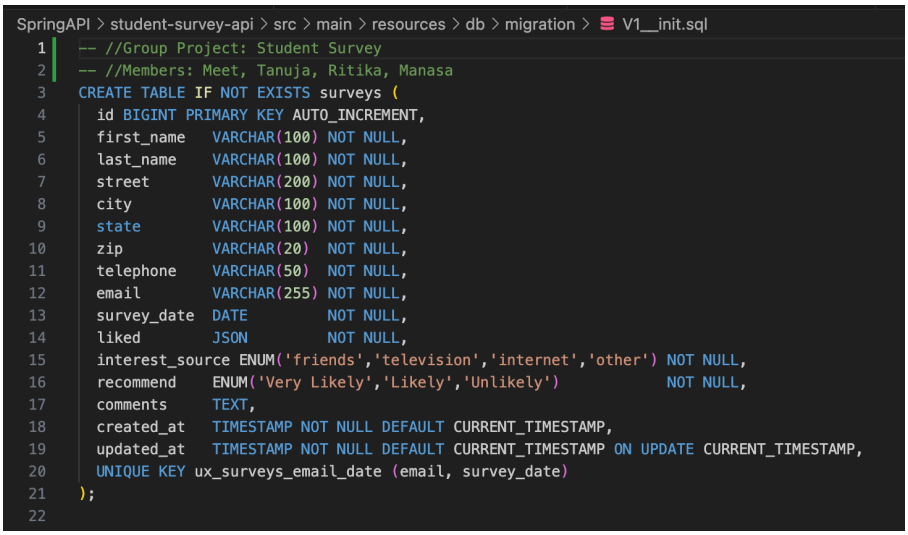
Layer	Technology	Notes
Frontend	Angular 19 (standalone APIs)	Standalone components, Angular Router, HttpClient, Reactive Forms
Backend	Spring Boot (REST)	Controllers, Services, Repositories
ORM	JPA / Hibernate	Entity mapping to MySQL
Database	MySQL (Amazon RDS)	Cloud-hosted, secured via SG
Tools	Node.js, npm, Maven, Spring, JDK, JRE	Build & test

5. Database Design

Primary Entity: StudentSurvey

Fields :

- id - number (optional)
- first_name - string (required)
- last_name - string (required)
- street - string (required)
- city - string (required)
- state - string (required)
- zip - string (required)
- telephone - string (required)
- email - string (required, email format)
- survey_date - string (required, date)
- liked - string[] (checkboxes: students, location, campus, atmosphere, dorm rooms, sports)
- interest_source - enum: friends | television | internet | other
- recommend - enum: Very Likely | Likely | Unlikely
- comments - string (optional)



6. Backend Implementation (Spring Boot)

This section will include controller endpoints, service layer, repository interfaces, entity annotations, application.properties (RDS connection), and CURL examples.

6.1 Tech & Requirements

• Java 17 or newer

• Maven 3.8+

• Spring Boot (Web, Data JPA, Validation)

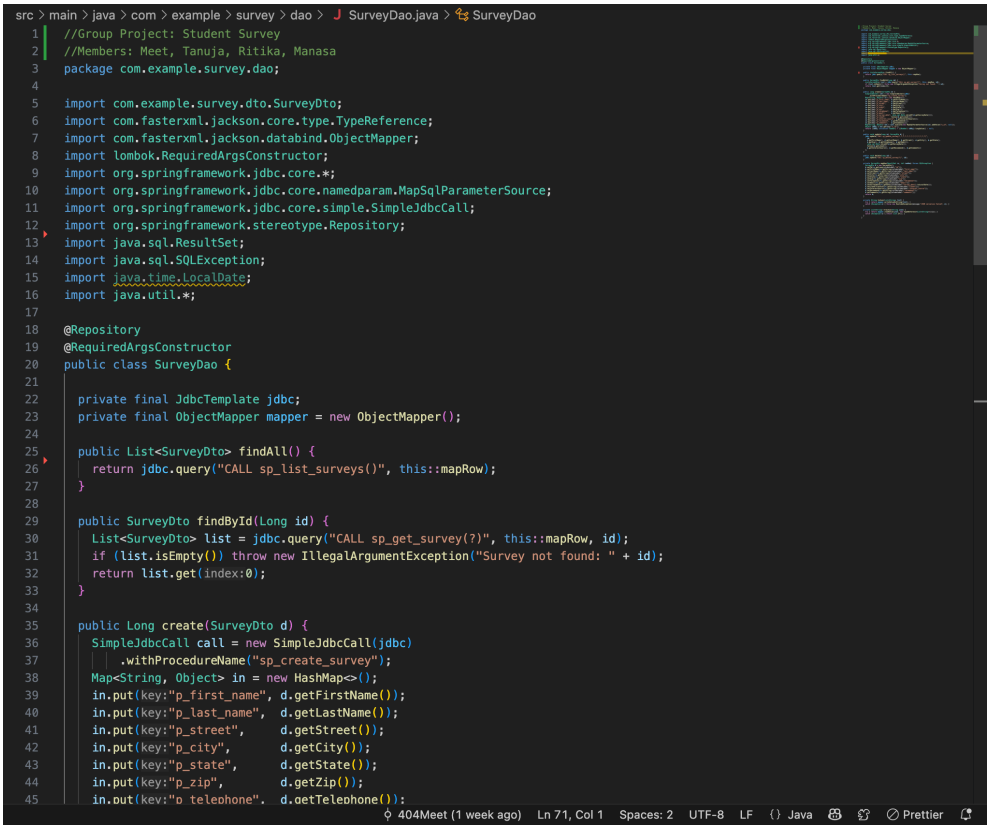
• Database: MySQL in production; H2 in-memory for local dev/testing

6.2 Project Layout

- The repository uses standard Maven layout.
- Data Access object – SurveyDao.java
- Data Transfer Object – SurveyDto.java
- Service – SurveyService.java
- Controller – SurveyController.java
- Start Point – StudentSurveyApiApplication.java
- DB Migrations- V1__Init.sql, V2__Procedures.sql
- Properties- application.properties
- Dependencies: pom.xml

6.3 Data Access Object

- This class directly interacts with the database layer.
- Typically wraps JPA repository calls.
- Handles CRUD operations like findAll(), create(), delete(), etc.
- In many modern Spring apps, DAOs are often replaced by JpaRepository interfaces, but having a DAO adds flexibility for custom SQL.



6.4 Data Transfer Object

- A DTO (Data Transfer Object) is used to send and receive data between layers (e.g. frontend - backend) without exposing the full entity model.
- Helps enforce validation and prevent exposing internal fields like IDs or timestamps.
- Useful for mapping user input to entity objects.

src > main > java > com > example > survey > dto > SurveyDto.java > ...

```java
//Group Project: Student Survey
//Members: Meet, Tanuja, Ritika, Manasa
package com.example.survey.dto;

import jakarta.validation.constraints.*;
import lombok.*;
import java.time.LocalDate;
import java.util.List;

@Getter @Setter @NoArgsConstructor @AllArgsConstructor @Builder
public class SurveyDto {
  private Long id;

  @NotBlank private String firstName;
  @NotBlank private String lastName;
  @NotBlank private String street;
  @NotBlank private String city;
  @NotBlank private String state;
  @NotBlank private String zip;
  @NotBlank private String telephone;
  @Email @NotBlank private String email;

  @NotNull private LocalDate surveyDate;

  @NotNull private List<String> liked; // sent/received as JSON array

  @NotBlank private String interestSource; // friends | television | internet | other
  @NotBlank private String recommend;      // Very Likely | Likely | Unlikely

  private String comments;
}

```

6.5 Service

- The business logic layer of your app.
- Acts as a bridge between Controller (web layer) and DAO/Repository (data layer).
- Handles validations, transformations, and transactional operations.

src > main > java > com > example > survey > service > SurveyService.java > ...

```java
//Group Project: Student Survey
//Members: Meet, Tanuja, Ritika, Manasa
package com.example.survey.service;

import com.example.survey.dao.SurveyDao;
import com.example.survey.dto.SurveyDto;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class SurveyService {
  private final SurveyDao dao;

  public List<SurveyDto> findAll() { return dao.findAll(); }
  public SurveyDto findById(Long id) { return dao.findById(id); }
  public SurveyDto create(SurveyDto dto) { Long id = dao.create(dto); return dao.findById(id); }
  public SurveyDto update(Long id, SurveyDto dto) { dao.update(id, dto); return dao.findById(id); }
  public void delete(Long id) { dao.delete(id); }
}

```

6.6 Controller

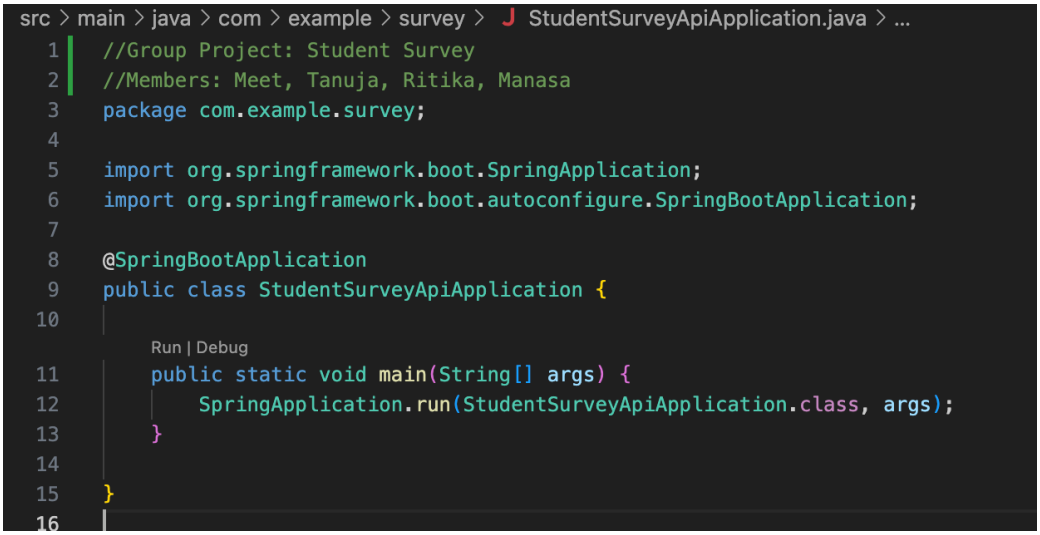
- The entry point for HTTP requests (REST endpoints).
- Handles client requests (GET, POST, PUT, DELETE) and returns JSON responses.
- Uses @RestController, @RequestMapping, and @RequiredArgsConstructor.

src > main > java > com > example > survey > web > J SurveyController.java > ...

```java
//Group Project: Student Survey
//Members: Meet, Tanuja, Ritika, Manasa
package com.example.survey.web;

import com.example.survey.dto.SurveyDto;
import com.example.survey.service.SurveyService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.net.URI;
import java.util.List;

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

  private final SurveyService service;

  @GetMapping
  public List<SurveyDto> all() { return service.findAll(); }

  @GetMapping("/{id}")
  public SurveyDto one(@PathVariable Long id) { return service.findById(id); }

  @PostMapping
  public ResponseEntity<SurveyDto> create(@Valid @RequestBody SurveyDto dto) {
    SurveyDto saved = service.create(dto);
    return ResponseEntity.created(URI.create("/api/surveys/" + saved.getId())).body(saved);
  }

  @PutMapping("/{id}")
  public SurveyDto update(@PathVariable Long id, @Valid @RequestBody SurveyDto dto) {
    return service.update(id, dto);
  }

  @DeleteMapping("/{id}")
  public ResponseEntity<Void> delete(@PathVariable Long id) {
    service.delete(id);
    return ResponseEntity.noContent().build();
  }
}
```

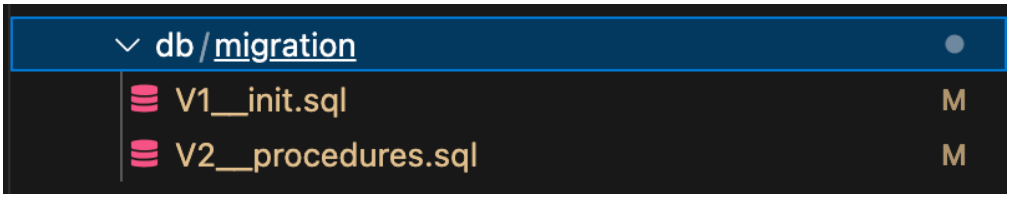
6.7 Start Point (StudentSurveyApiApplication.java)

- The main class that boots the entire Spring application.
- Contains the main() method with SpringApplication.run() which starts the embedded Tomcat server.



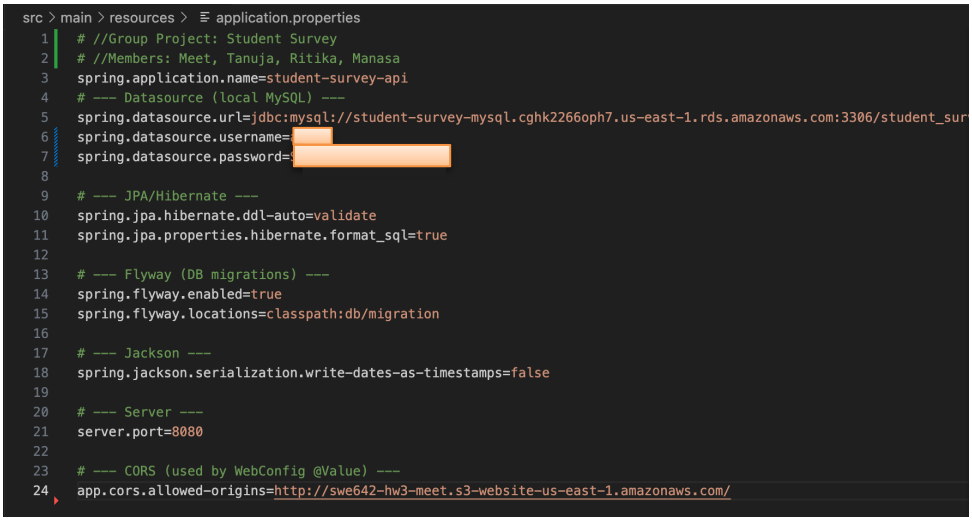
6.8 Database Migration

- These are Flyway migration scripts used to version-control your database schema.
- Automatically executed when the app starts (if Flyway is enabled).
- Example:
  - V1__Init.sql: creates tables and constraints
  - V2__Procedures.sql: adds stored procedures or triggers



6.9 Properties (application.properties)

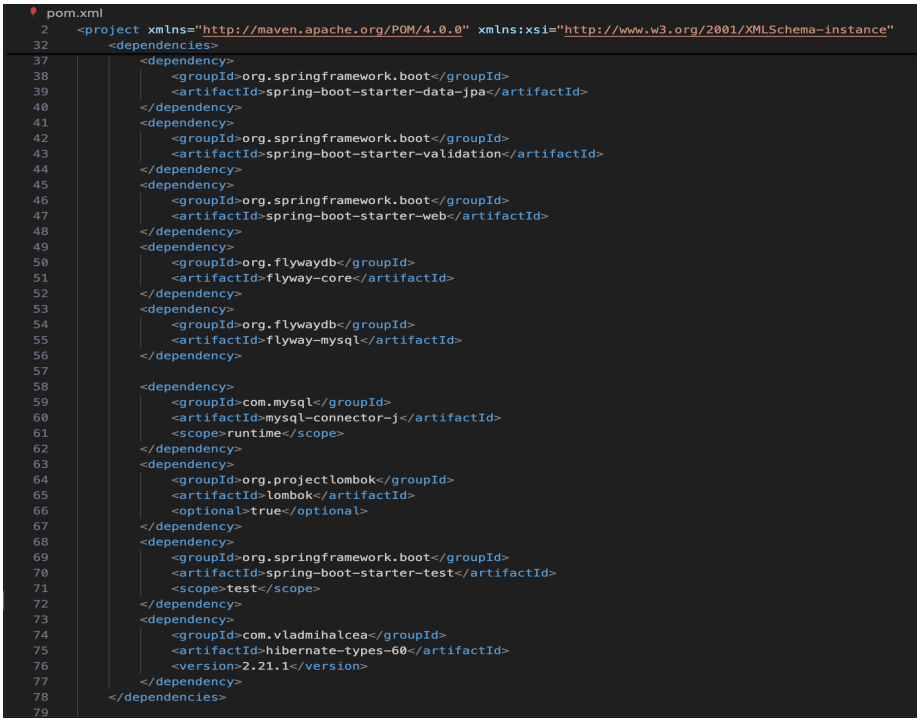
- Central configuration file for Spring Boot.
- Defines port, DB connection, logging, and JPA behavior.



```
src > main > resources > application.properties
# //Group Project: Student Survey
# //Members: Meet, Tanuja, Ritika, Manasa
spring.application.name=student-survey-api
# --- Datasource (local MySQL) ---
spring.datasource.url=jdbc:mysql://student-survey-mysql.cghk2266oph7.us-east-1.rds.amazonaws.com:3306/student_sur
spring.datasource.username=
spring.datasource.password=

# --- JPA/Hibernate ---
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.properties.hibernate.format_sql=true

# --- Flyway (DB migrations) ---
spring.flyway.enabled=true
spring.flyway.locations=classpath:db/migration

# --- Jackson ---
spring.jackson.serialization.write-dates-as-timestamps=false

# --- Server ---
server.port=8080

# --- CORS (used by WebConfig @Value) ---
app.cors.allowed-origins=http://swe642-hw3-meet.s3-website-us-east-1.amazonaws.com/
```

6.10 Dependencies (pom.xml)

- Maven's Project Object Model file.
- Lists all required libraries, build plugins, and metadata.



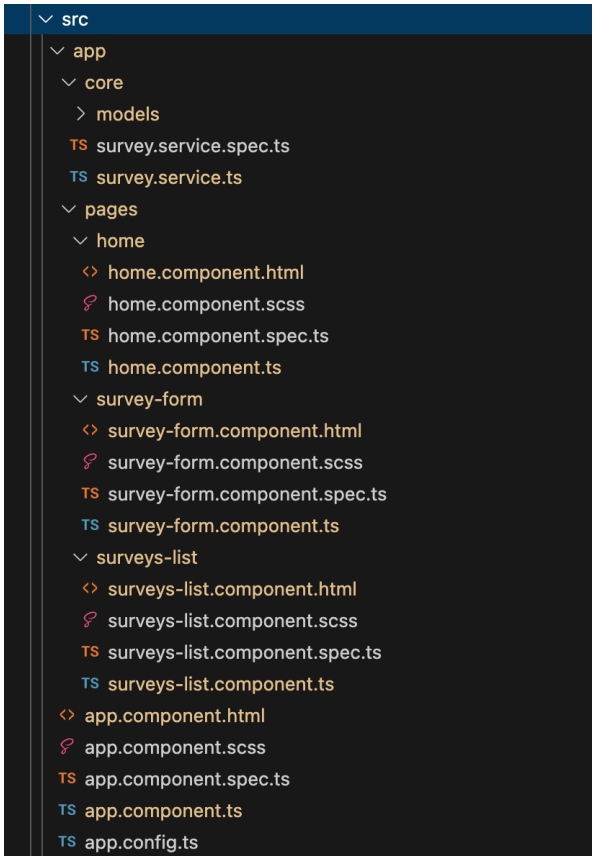
```
pom.xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.flywaydb</groupId>
            <artifactId>flyway-core</artifactId>
        </dependency>
        <dependency>
            <groupId>org.flywaydb</groupId>
            <artifactId>flyway-mysql</artifactId>
        </dependency>

        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.vladmihalcea</groupId>
            <artifactId>hibernate-types-60</artifactId>
            <version>2.21.1</version>
        </dependency>
    </dependencies>
```

7. Frontend Implementation (Angular)

7.1 Project Structure

Root: src/app/ with the following notable areas:

- core/survey.service.specs.ts — Testing file for service.
- core/survey.service.ts — HttpClient-based CRUD methods
- core/models/survey.ts — Interface (Properties for service)
- pages/home — HomeComponent for the welcome page
- pages/survey-form — SurveyFormComponent for create/update form
- pages/surveys-list — SurveysListComponent for listing and actions



7.2 Routing (Standalone)

Routes configured in app.routes.ts:

- path: '' → HomeComponent
- path: 'survey' → SurveyFormComponent
- path: 'surveys' → SurveysListComponent
- path: 'surveys/:id/edit' → SurveyFormComponent

Angular > student-survey > src > app > TS app.routes.ts > ...

```ts
import { Routes } from '@angular/router';
import { HomeComponent } from './pages/home/home.component';
import { SurveyFormComponent } from './pages/survey-form/survey-form.component';
import { SurveysListComponent } from './pages/surveys-list/surveys-list.component';

export const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'survey', component: SurveyFormComponent },
  { path: 'surveys', component: SurveysListComponent },
  { path: 'surveys/:id/edit', component: SurveyFormComponent },
  { path: '**', redirectTo: '' }
];

```

7.3 Components

- HomeComponent — Welcome page with links to Student Survey and List All Surveys.

Angular > student-survey > src > app > pages > home > TS home.component.ts > ...

```ts
//Group Project: Student Survey
//Members: Meet, Tanuja, Ritika, Manasa

import { Component } from '@angular/core';
import { RouterModule } from '@angular/router';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-home',
  (property) Component.styleUrls?: string[] | undefined

  Relative paths or absolute URLs for files containing CSS stylesheets to use in this component.
  styleUrls: ['./home.component.scss']
})
export class HomeComponent {}
```

- SurveyFormComponent — Reactive form to create/update a survey (required fields, checkboxes, radios, dropdown).

```ts
import { Component, OnInit } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormBuilder, FormGroup, FormArray, FormControl, ReactiveFormsModule, Validators } from '@angula
import { Router, RouterModule, ActivatedRoute } from '@angular/router';
import { SurveyService } from '../../core/survey.service';
import { Survey } from '../../core/models/survey';

@Component({
  selector: 'app-survey-form',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModule, RouterModule],
  templateUrl: './survey-form.component.html',
  styleUrls: ['./survey-form.component.scss']
})
export class SurveyFormComponent implements OnInit {
  form!: FormGroup;
  editId?: number;

  constructor(
    private fb: FormBuilder,
    private svc: SurveyService,
    private route: ActivatedRoute,
    private router: Router
  ) {}

  ngOnInit(): void {
    this.form = this.fb.group({
      firstName: ['', Validators.required],
      lastName: ['', Validators.required],
      street: ['', Validators.required],
      city: ['', Validators.required],
      state: ['', Validators.required],
      zip: ['', Validators.required],
      telephone: ['', Validators.required],
      email: ['', [Validators.required, Validators.email]],
      surveyDate: ['', Validators.required],
      recommend: ['', Validators.required],
```

```
    liked: new FormArray([
      new FormControl(false, { nonNullable: true }),
      new FormControl(false, { nonNullable: true }),
      new FormControl(false, { nonNullable: true }),
      new FormControl(false, { nonNullable: true }),
      new FormControl(false, { nonNullable: true }),
      new FormControl(false, { nonNullable: true }),
    ]),
    interestSource: ['', Validators.required],
    comments: ['']
  });

  const id = this.route.snapshot.paramMap.get('id');
  if (id) {
    this.editId = +id;
    this.svc.getById(this.editId).subscribe(s => this.patchFromModel(s));
  }
}

get likedArray(): FormArray<FormControl<boolean>> {
  return this.form.get('liked') as FormArray<FormControl<boolean>>;
}

private patchFromModel(s: Survey) {
  this.form.patchValue({
    firstName: s.firstName,
    lastName: s.lastName,
    street: s.street,
    city: s.city,
    state: s.state,
    zip: s.zip,
    telephone: s.telephone,
    email: s.email,
    surveyDate: s.surveyDate,
    recommend: s.recommend,
    interestSource: s.interestSource,
```

```
    comments: s.comments
  });

  const likedKeys = ['campus', 'dorms', 'students', 'location', 'atmosphere', 'sports'];
  likedKeys.forEach((key, i) => {
    this.likedArray.at(i).setValue(s.liked?.includes(key) ?? false);
  });
}

submit() {
  if (this.form.invalid) {
    alert('Please fill all required fields');
    return;
  }

  const likedKeys = ['campus', 'dorms', 'students', 'location', 'atmosphere', 'sports'];
  const likedPicked = this.likedArray.value
    .map((v, i) => (v ? likedKeys[i] : null))
    .filter(Boolean) as string[];

  const payload: Survey = { ...this.form.getRawValue(), liked: likedPicked };

  const req$ = this.editId
    ? this.svc.update(this.editId, payload)
    : this.svc.create(payload);

  req$.subscribe({
    next: () => {
      alert('Survey submitted successfully!');
      this.router.navigate(['/surveys']);
    },
    error: (err) => {
      console.error('Save failed:', err);
      alert(`Save failed: ${err?.status || ''} ${err?.statusText || err?.message || ''}`);
    }
  });
}

cancel() {
  this.router.navigate(['/']);
}
```

- SurveysListComponent — Tabular list of all surveys with Update/Delete actions.

```
Angular > student-survey > src > app > pages > surveys-list > TS surveys-list.component.ts > ...
1  //Group Project: Student Survey
2  //Members: Meet, Tanuja, Ritika, Manasa
3
4  import { Component, OnInit } from '@angular/core';
5  import { SurveyService } from '../../core/survey.service';
6  import { Survey } from '../../core/models/survey';
7  import { RouterModule, Router } from '@angular/router';
8  import { CommonModule } from '@angular/common';
9  import { finalize } from 'rxjs/operators';
10
11 @Component({
12   selector: 'app-surveys-list',
13   standalone: true,
14   imports: [RouterModule, CommonModule],
15   templateUrl: './surveys-list.component.html',
16   styleUrls: ['./surveys-list.component.scss']
17 })
18 export class SurveysListComponent implements OnInit {
19   rows: Survey[] = [];
20   loading = true;
21   errorMsg = '';
22
23   constructor(private svc: SurveyService, private router: Router) {}
24
25   ngOnInit(): void {
26     this.load();
27   }
29   load() {
30     this.loading = true;
31     this.errorMsg = '';
32     this.svc.getAll()
33       .pipe(finalize(() => this.loading = false))
34       .subscribe({
35         next: (data) => this.rows = data ?? [],
36         error: (err) => {
37           console.error('Survey fetch failed', err);
38           this.rows = [];
39           this.errorMsg = typeof err?.message === 'string'
40             ? err.message
41             : 'Could not load surveys.';
42         }
43       });
44   }
45
46   edit(id?: number) {
47     if (id) this.router.navigate(['/surveys', id, 'edit']);
48   }
49
50   remove(id?: number) {
51     if (!id) return;
52     if (!confirm('Delete this survey?')) return;
53     this.svc.delete(id).subscribe(() => this.load());
54   }
55 }
56
```

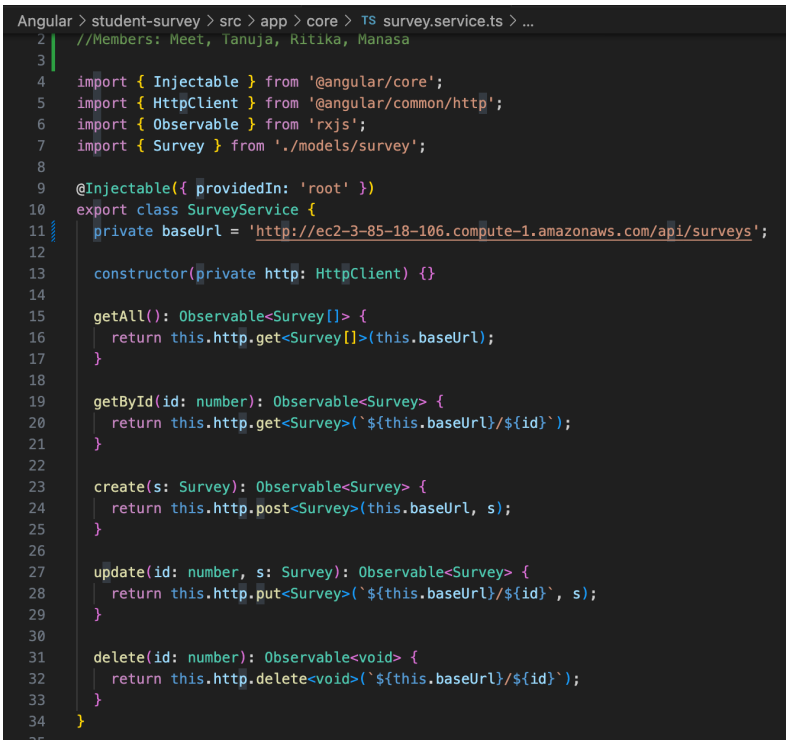
7.4 Services (HttpClient)

Base API URL: http://ec2-3-85-18-106.compute-1.amazonaws.com/api/surveys

Methods:

- getAll() - GET http://ec2-3-85-18-106.compute-1.amazonaws.com/api/surveys
- getById(id) -GET http://ec2-3-85-18-106.compute-1.amazonaws.com/api/surveys/{id} — Fetch a survey by ID (if implemented).

- create(survey) — POST http://ec2-3-85-18-106.compute-1.amazonaws.com/api/surveys — Create a new survey.
- update(id, survey) — PUT http://ec2-3-85-18-106.compute-1.amazonaws.com/api/surveys/{id} — Update an existing survey.
- delete(id) — DELETE http://ec2-3-85-18-106.compute-1.amazonaws.com/api/surveys/{id} — Delete a survey by ID.



7.5 Reactive Forms & Validation

SurveyFormComponent builds a FormGroup with required validators for firstName, lastName, street, city, state, zip, telephone, email, surveyDate, and validators.email for email. It binds checkboxes (liked), radio buttons (interestSource), and dropdown (recommend).

7.6 List Page & Actions

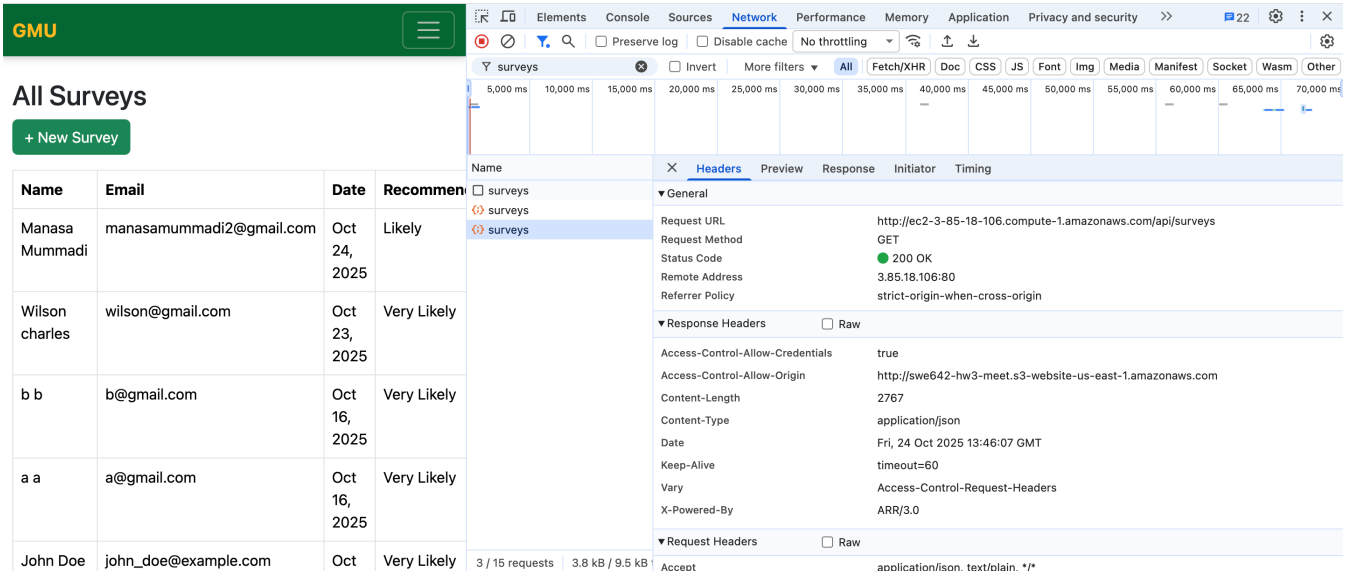
SurveysListComponent displays all surveys in a table, handles loading/error states, and triggers update/delete actions via SurveyService. Includes navigation to edit route /surveys/:id/edit.

```
<table class="table table-bordered"
       *ngIf="!loading && (rows?.length ?? 0) > 0">
  <thead>
    <tr>
      <th>Name</th><th>Email</th><th>Date</th><th>Recommend</th><th></th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let s of rows">
      <td>{{ s.firstName }} {{ s.lastName }}</td>
      <td>{{ s.email }}</td>
      <td>{{ s.surveyDate | date:'mediumDate' }}</td>
      <td>{{ s.recommend }}</td>
      <td>
        <button class="btn btn-sm btn-outline-secondary" (click)="edit(s.id)">Edit</button>
        <button class="btn btn-sm btn-outline-danger" (click)="remove(s.id)">Delete</button>
      </td>
    </tr>
  </tbody>
</table>
```

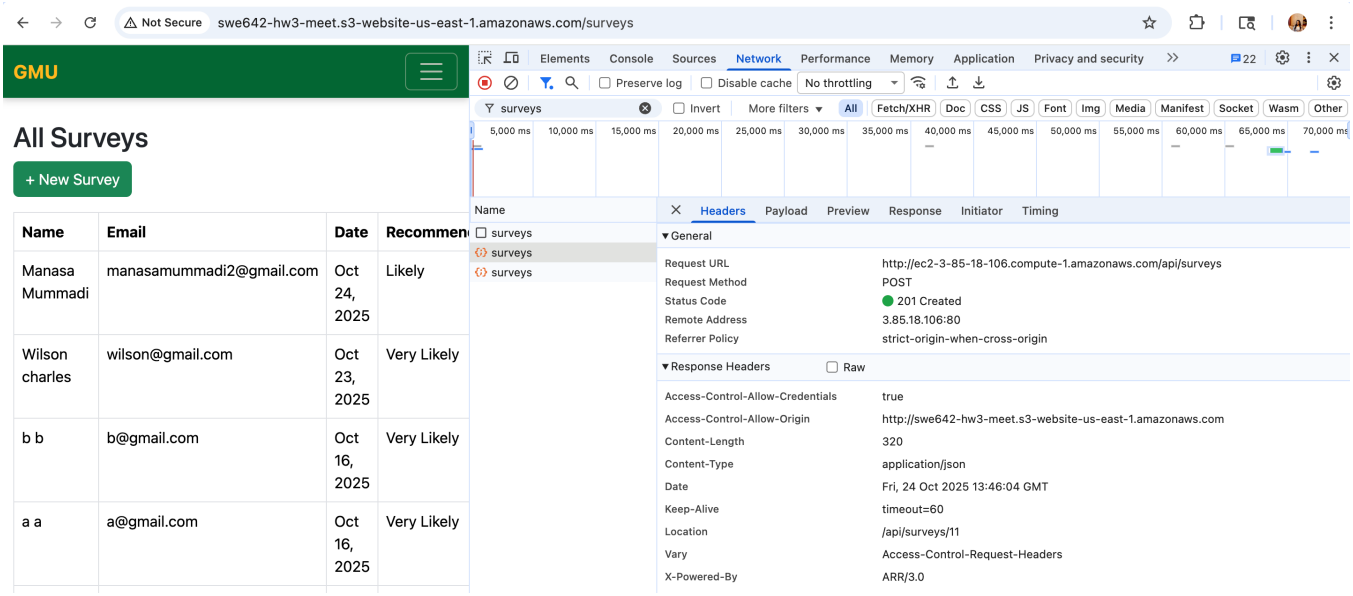
8. Frontend–Backend Integration

Angular consumes the Spring Boot REST API using HttpClient. CORS must be enabled on the backend. End-to-end flow: User submits survey → Angular POST /api/surveys → Spring Boot persists via JPA → Record appears on List All Surveys.

GET



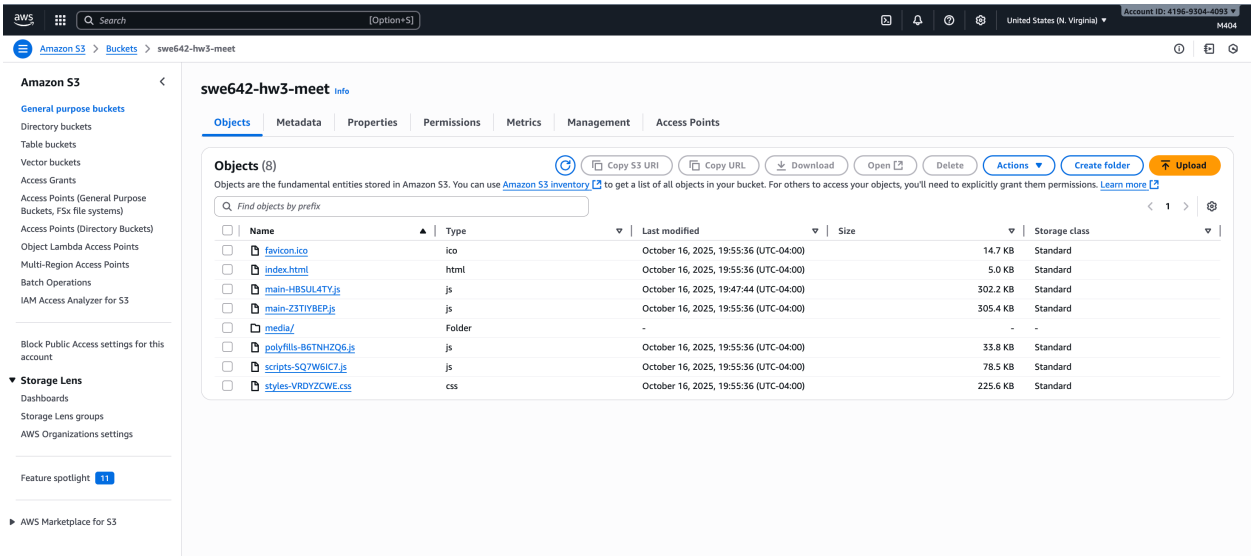
POST



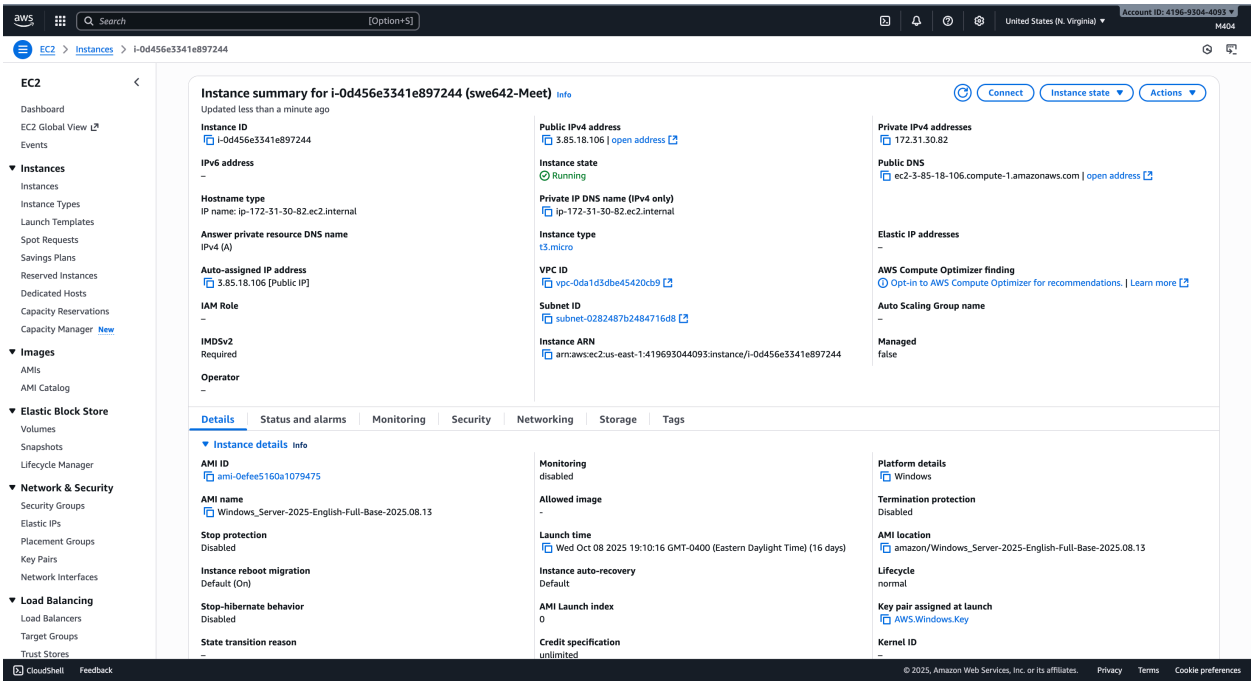
9. Deployment Setup

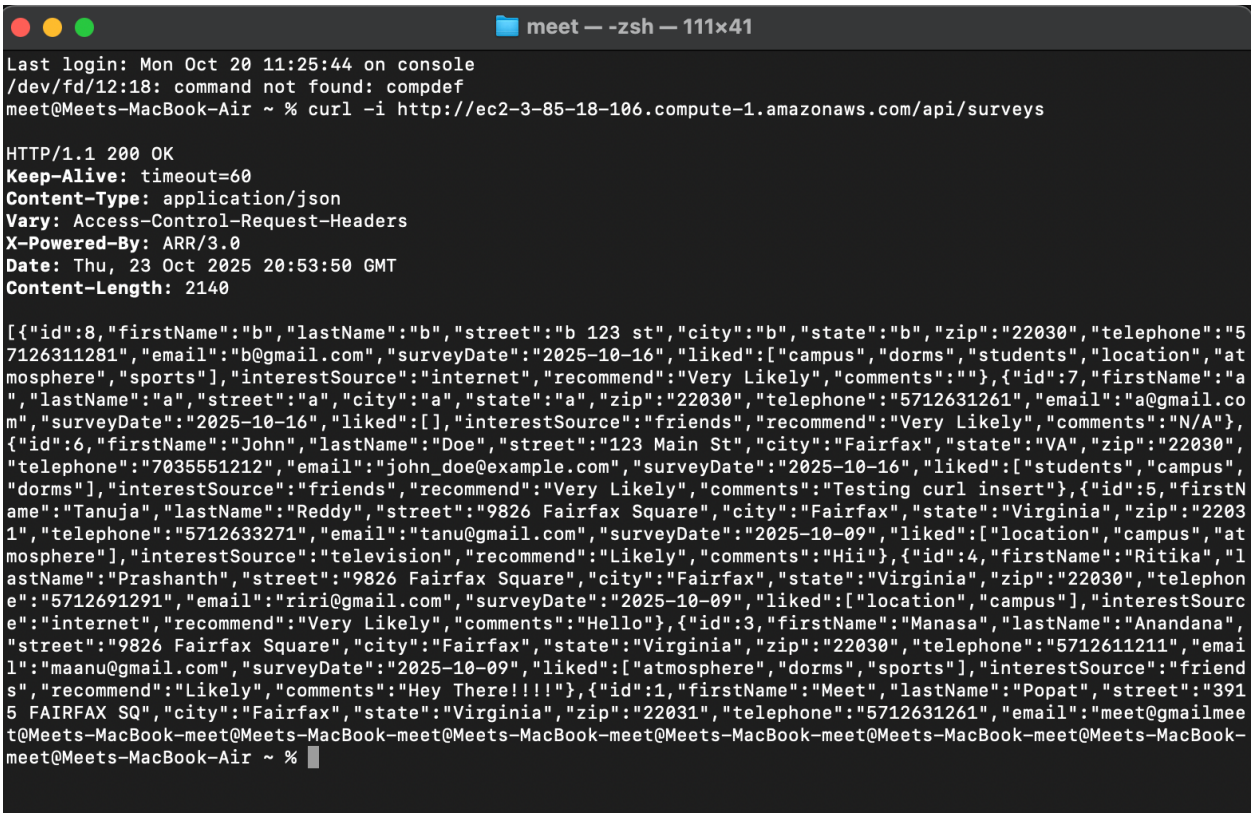
Amazon RDS: MySQL instance with security group allowing the backend host. Backend is packaged as a JAR and ran on an EC2 instance. Angular can served locally (ng serve) or deployed as static assets (ng build) as we did behind S3 bucket.

S3 Bucket: http://swe642-hw3-meet.s3-website-us-east-1.amazonaws.com/

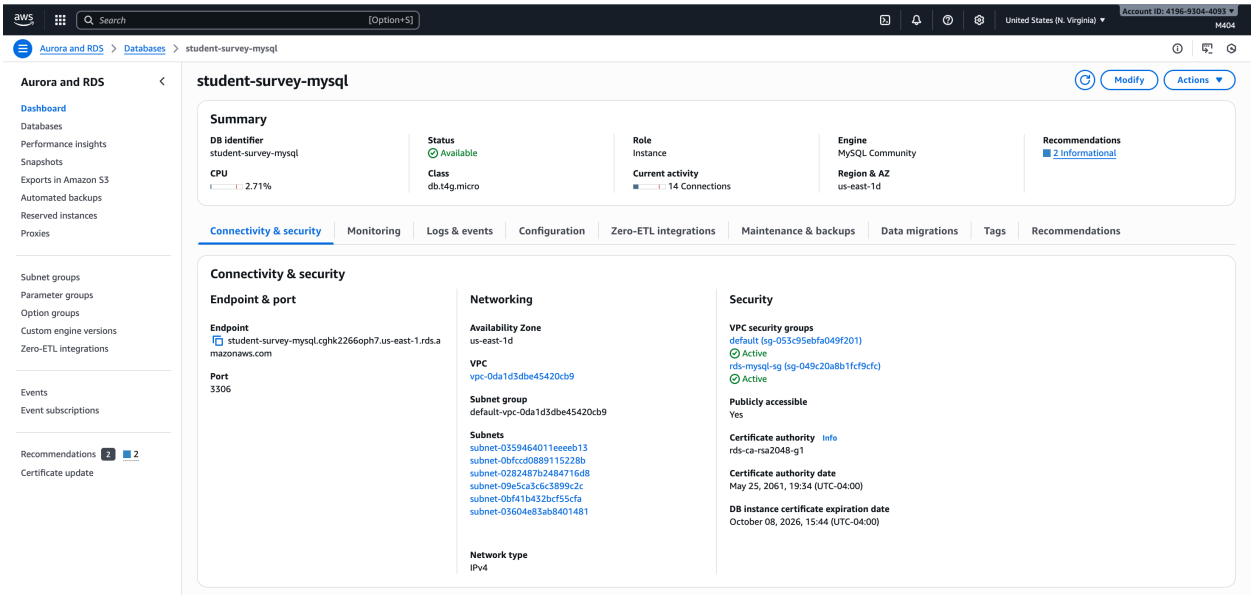


EC2 Instance: http://ec2-3-85-18-106.compute-1.amazonaws.com/api/surveys



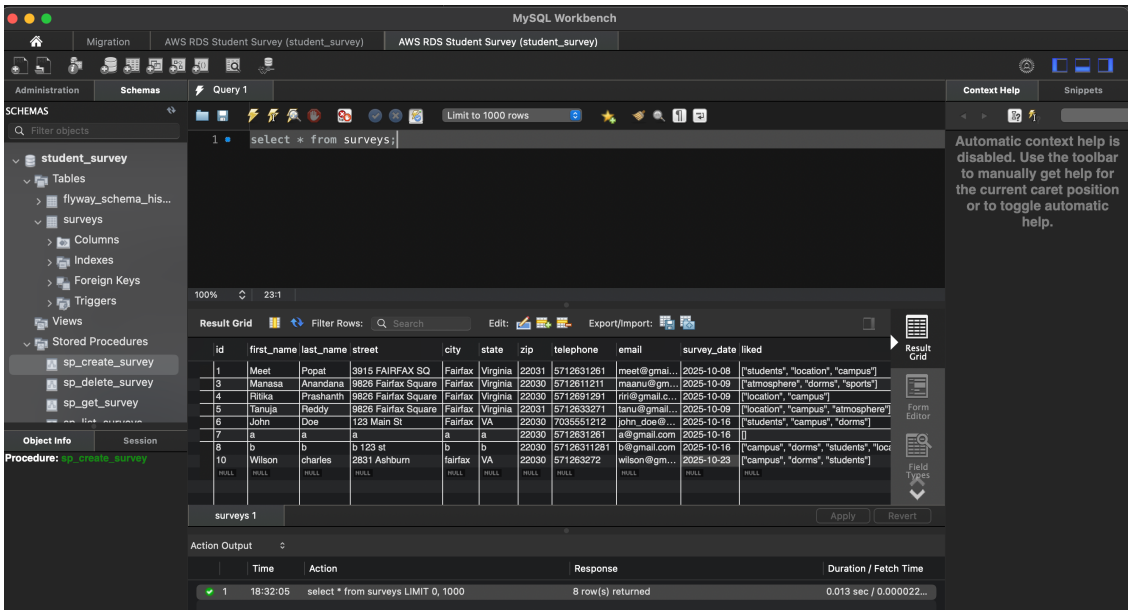


Amazon RDS (MySQL):



10. Testing & Validation

- Functional testing of CRUD endpoints via CURL and UI.
- Form validation: required fields, email format, date selection.
- Integration testing: Verify DB records after UI actions.

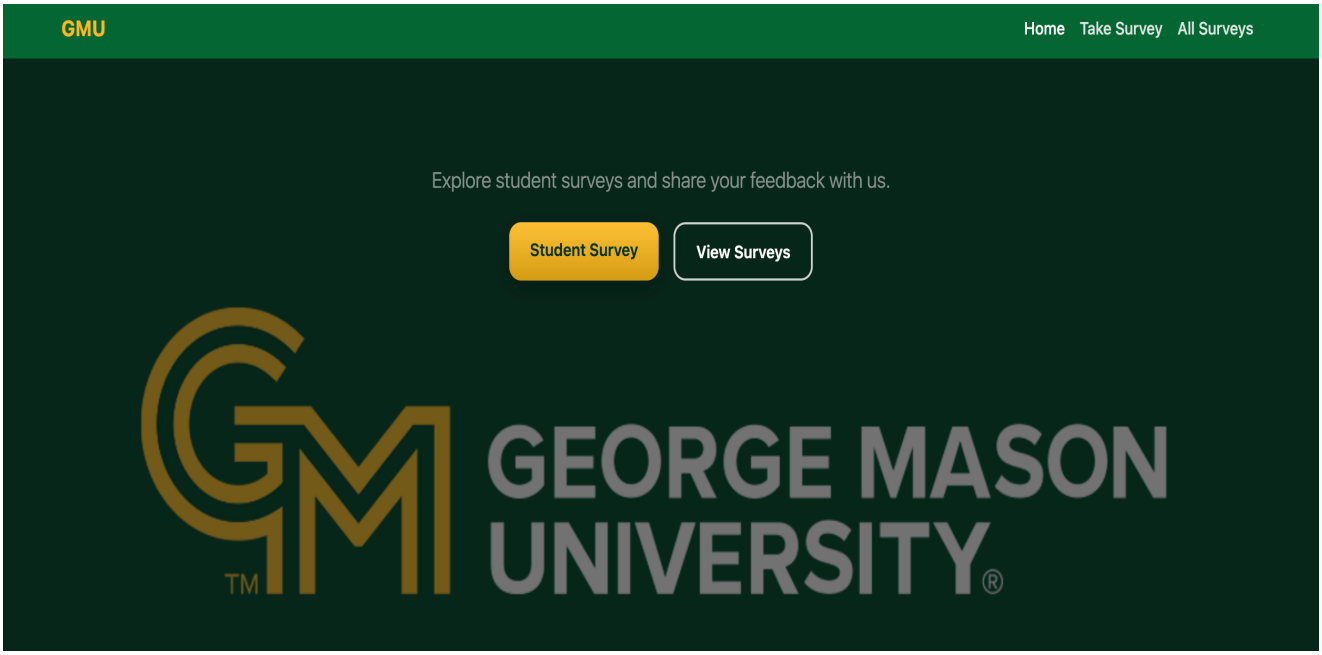


11. Application Overview

11.1 Home Page:

We start on the Home Page, which is a clean entry point. Displays a welcoming message and two navigation buttons:

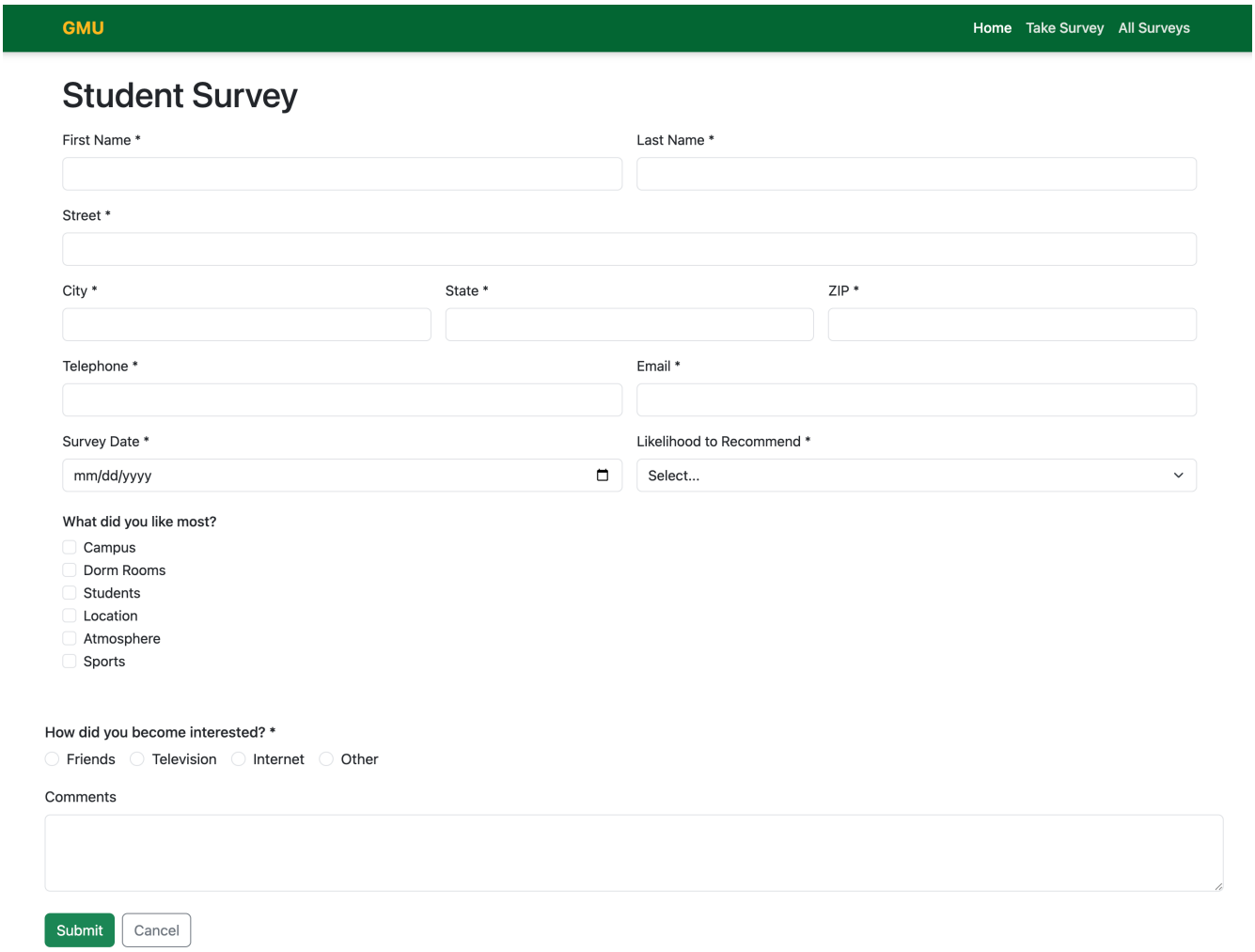
- **“Student Survey”** - opens the survey form page
- **“View Surveys”** - opens the table view of all existing survey entries



11.2 Student Survey Page:

Navigate to the Student Survey page by clicking on **Student Survey** Button

- **Required Fields:** All personal and contact details, like name, address, email, and phone number, are required.
- **Checkboxes:** We use checkboxes for what they liked most about the campus: students, location, campus, and more.
- **Radio Buttons:** They indicate how they heard about us using radio buttons for friends, television, or internet.
- **Dropdown:** And they select their likelihood of recommending the school from a simple dropdown list.

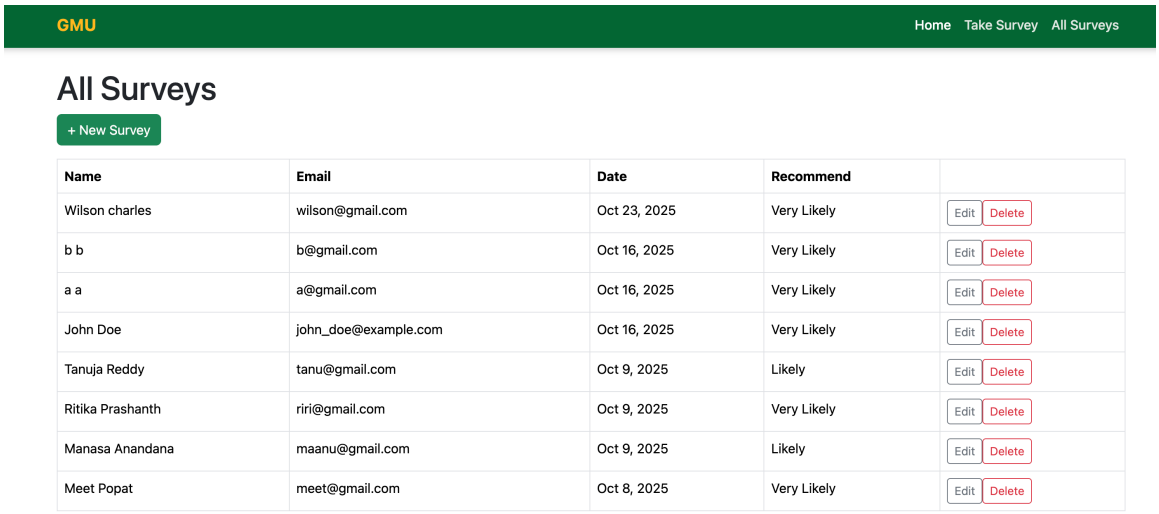


When the user hits Submit, the Angular app fires a RESTful call to our backend to store the data permanently in the database.

11.3 All Surveys Page:

This page shows all surveys recorded to date in a simple table format. This is our Read operation.

- **Update:** Every row has an **Edit** button. When clicked, it takes us back to the survey form, pre-filling all the data. We can make changes and resubmit. That handles our **Update** requirement.
- **Delete:** And of course, there's a **Delete** button right next to it. One click, and the record is gone.



12. Conclusion

This project demonstrates a complete Angular + Spring Boot + MySQL (RDS) stack implementing a student survey with full CRUD functionality: **C**reate, **R**ead, **U**pdate, and **D**elete, all driven by the Angular frontend.

Website URL: http://swe642-hw3-meet.s3-website-us-east-1.amazonaws.com/