SWE 642 EXTRA CREDIT REPORT

Team Members:

Jithendra Sai Pappuri – G01506453 - jpappuri@gmu.edu

Venkata Abhiram Karuturi – G01505660 – vkarutur@gmu.edu

Ganesh Jasti – G01505410 – gjasti@gmu.edu

Lakshmi Sankari Vissapragada - G01481312- lvissapr@gmu.edu

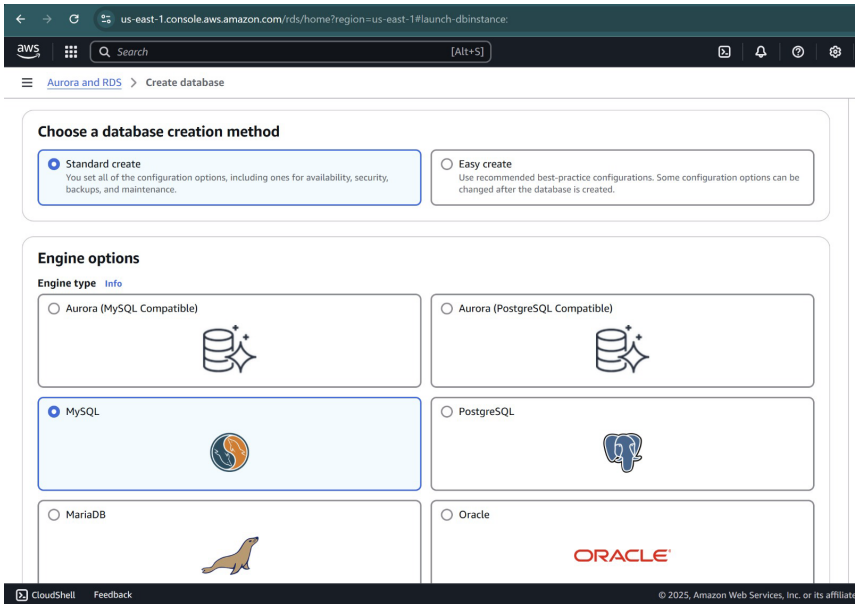
Technologies Used:

1. Django
2. Amazon RDS DB
3. MySQL

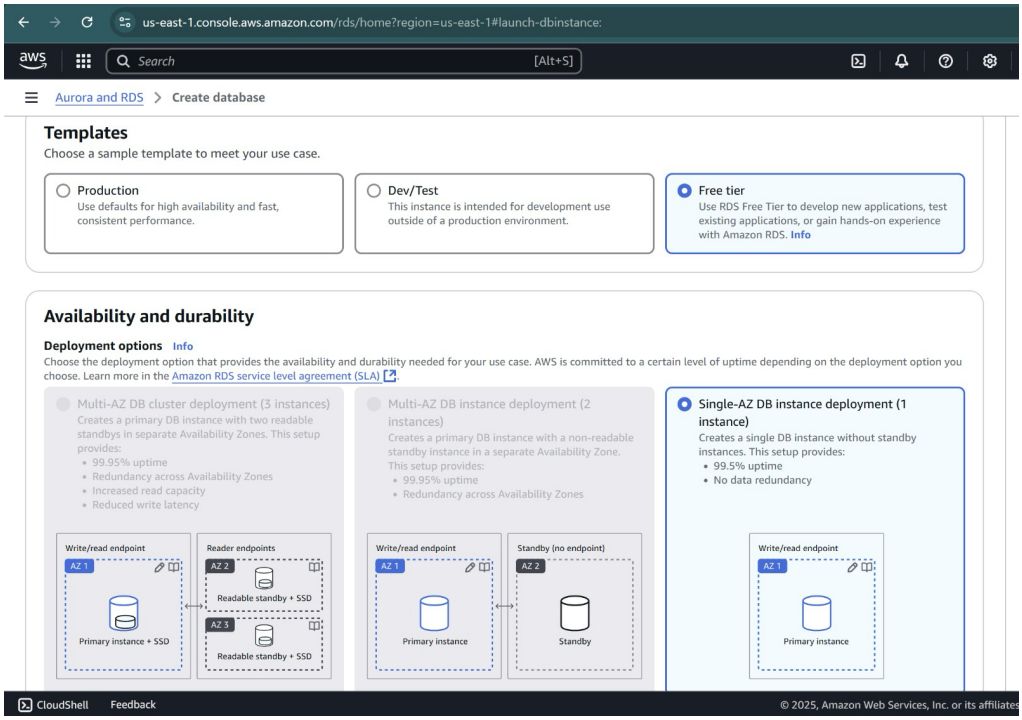
Project Setup:

Creating MySQL DB using Amazon RDS:

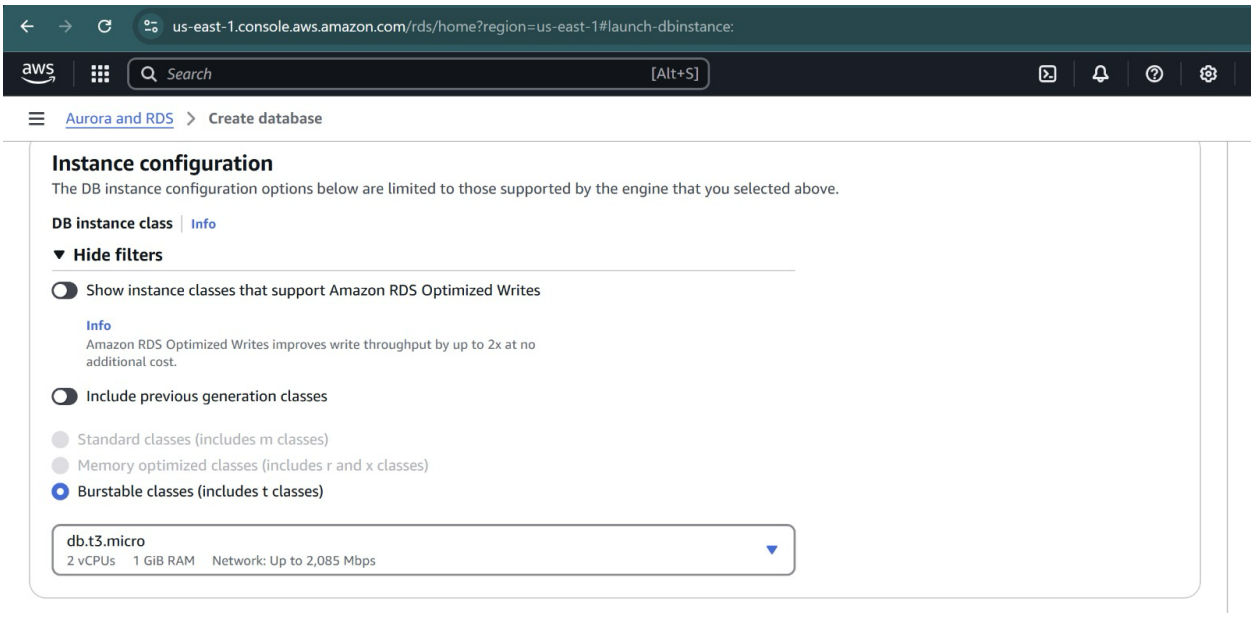
1. Login to AWS console and go to RDS section.
2. Create a Database
3. Select Standard Create from the given database create options.
4. Select MySQL from the given database options.



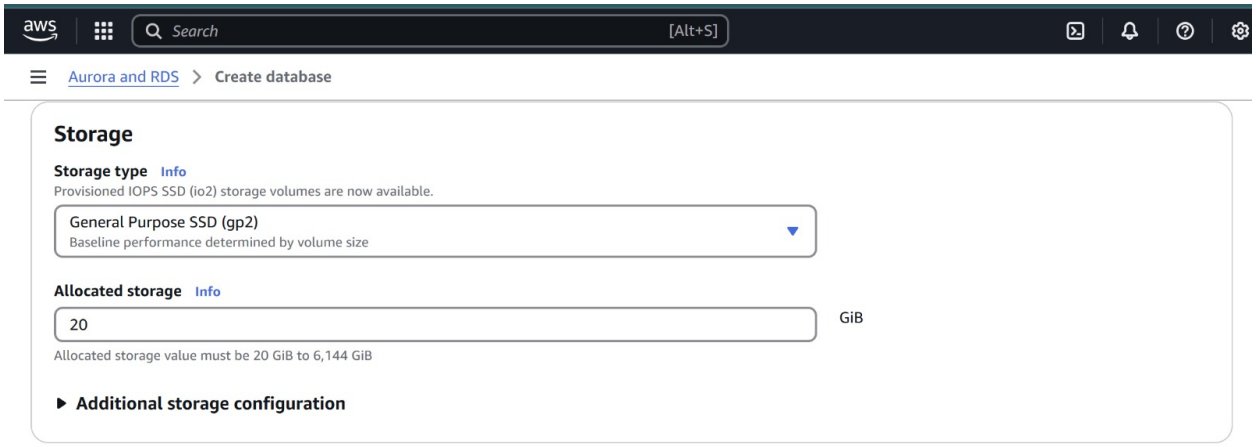
5. leave the engine options as it is.
6. Select **Free Tier** from the templates.



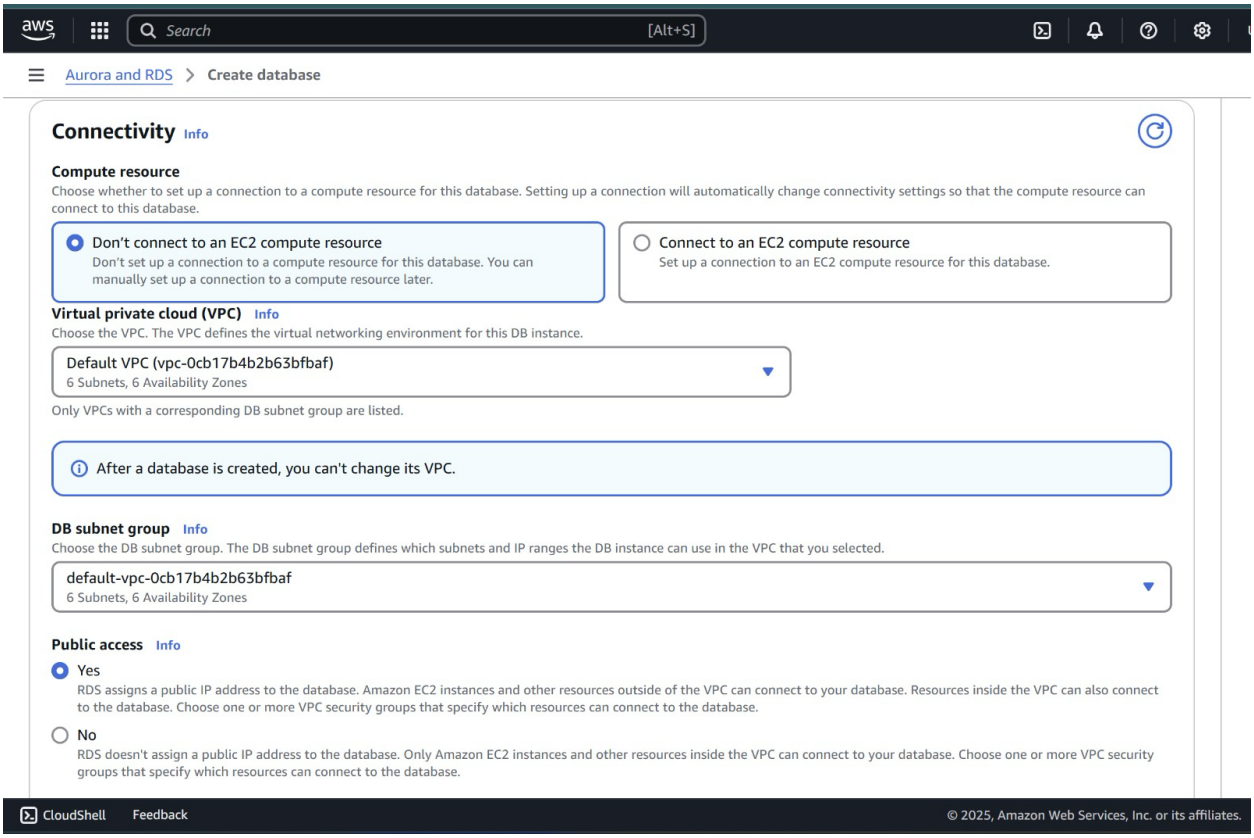
7. Select a name for database and credentials for username and password.
8. Select **db.t3.micro** in Instance Configuration.



9. Leave the storage option as it is.



10. Don't change anything in **Connectivity** and allow the **Public Access.**



11. Leave the rest of the database settings as it is and scroll to the end and select **Create Database.**

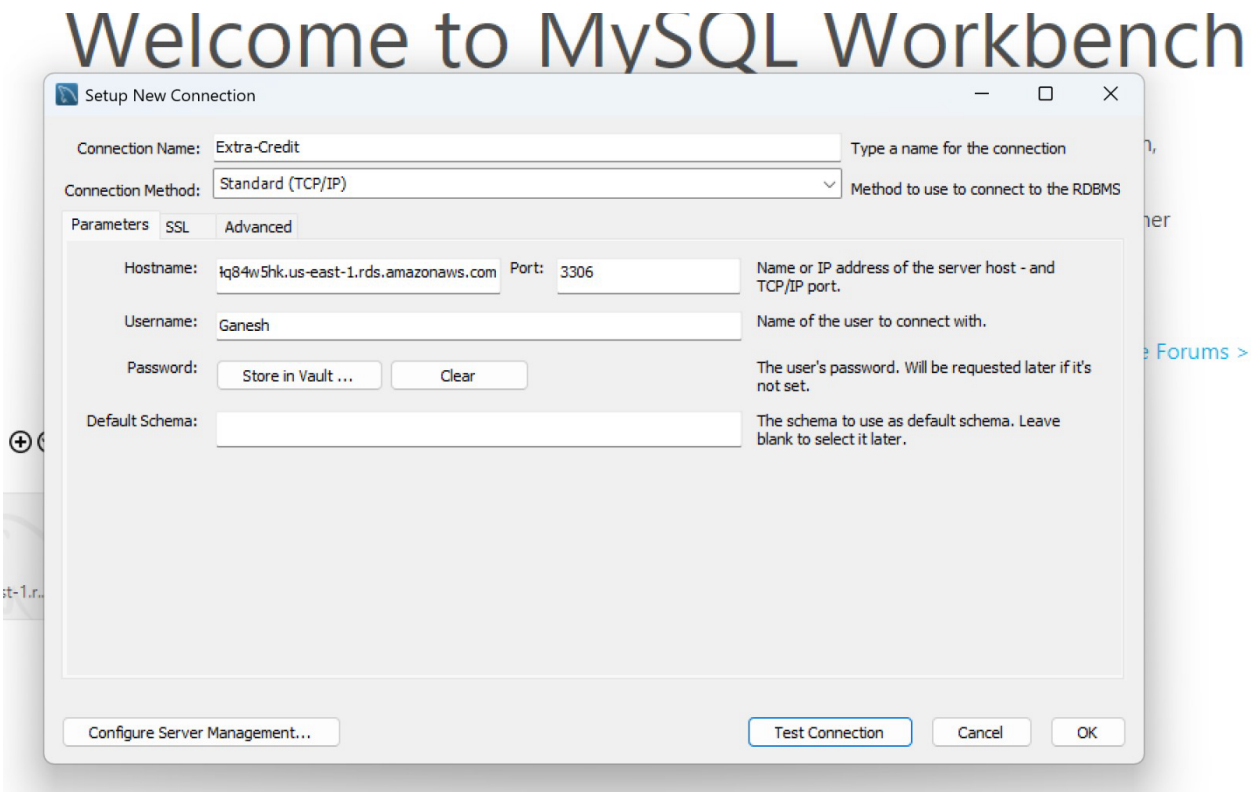
12. The database will be created.

13. After creating, we will get a Database endpoint.

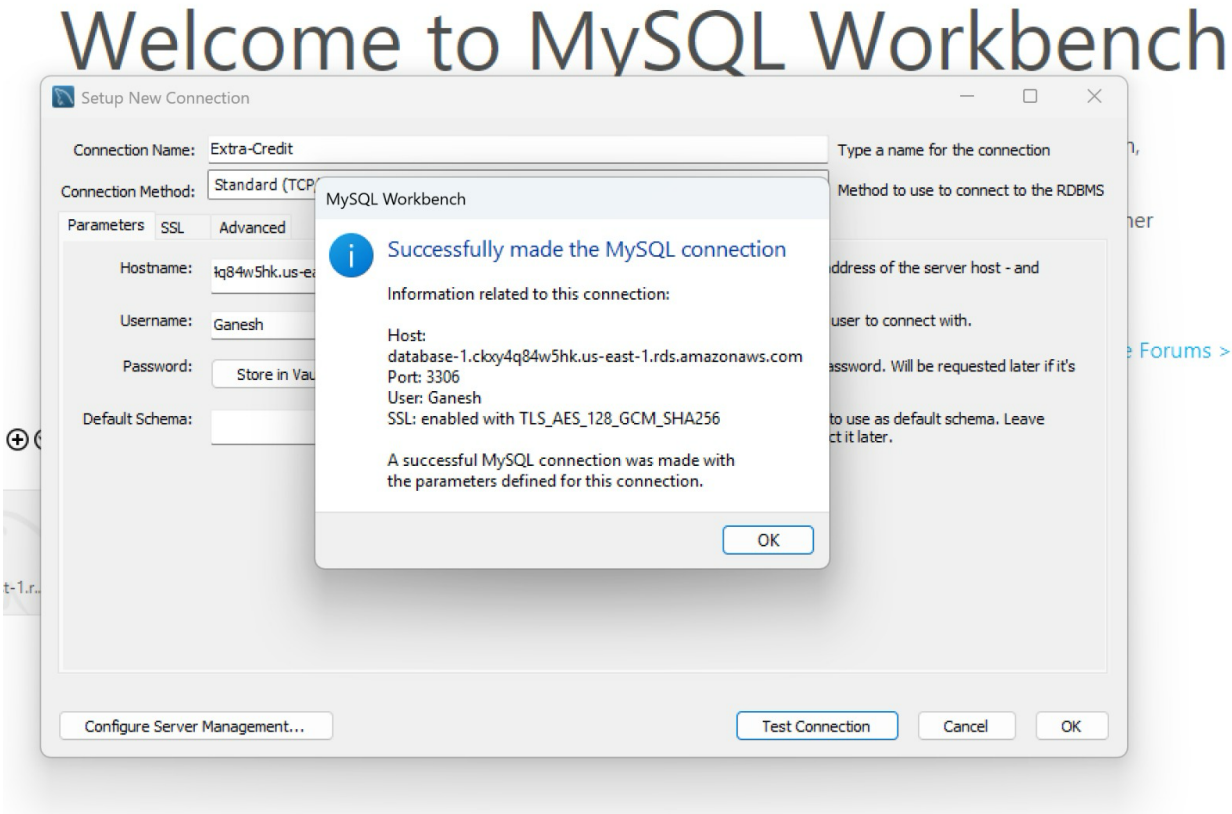
DB Endpoint: database-1.ckxy4q84w5hk.us-east-1.rds.amazonaws.com

Using MySQL Workbench tro connect to DB:

1. To create a new connection, select the + icon in the MySQL Workbench
2. Enter connection name to database.
3. enter the endpoint **database-1.ckxy4q84w5hk.us-east-1.rds.amazonaws.com** as host.
4. Enter username and password that was given while creating AWS database.



5. Click test connection and check if the connection is established.
6. If established successfully you will get below message.



7. Now the connection was established to MySQL.

8. Create a database with name surveys using the command in workbench

create database survey_details;

Creating Django Project(front end and backend):

For this assignment we are using python framework Django for both front end and back end.

1. First create a new folder in Visual studio and name it Django-copy.

2. Install Django in the above folder using the below command:

pip install django

3. After installing django we will see some default files and folders like models.py, views.py, forms.py, manage.py

4. Now create a subfolder named **survey_app** for frontend and **survey_project** for backend.

5. Now create a subfolder named templates in survey_app and create the below files

base.html

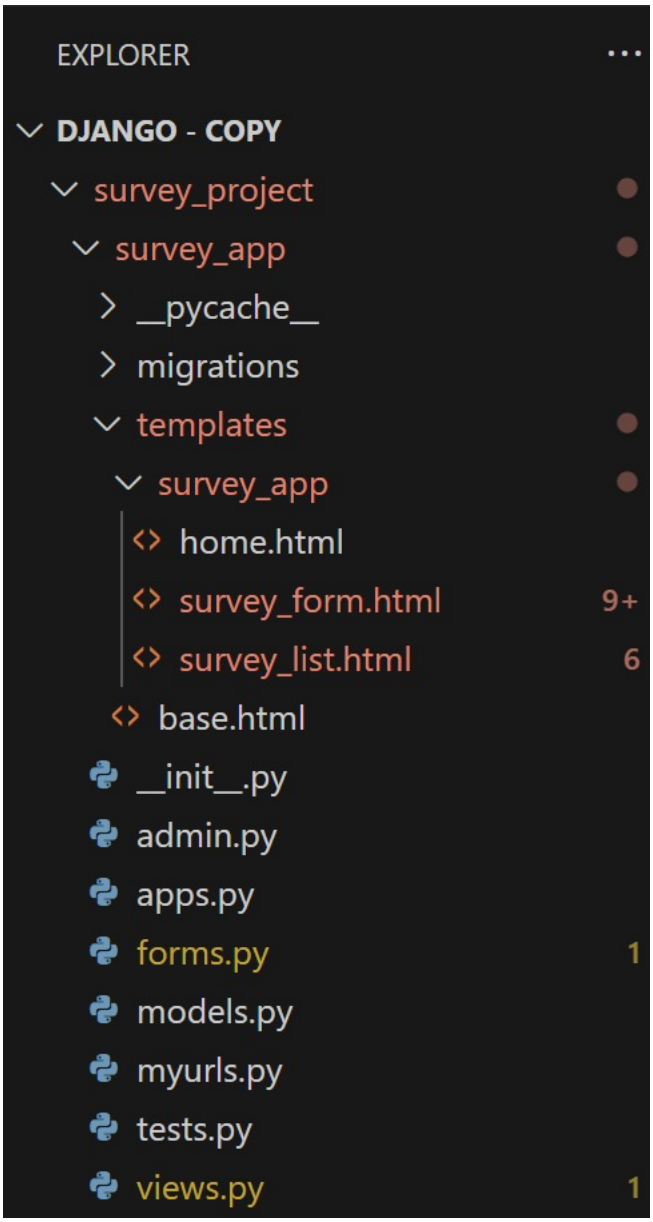
home.html

survey_form.html

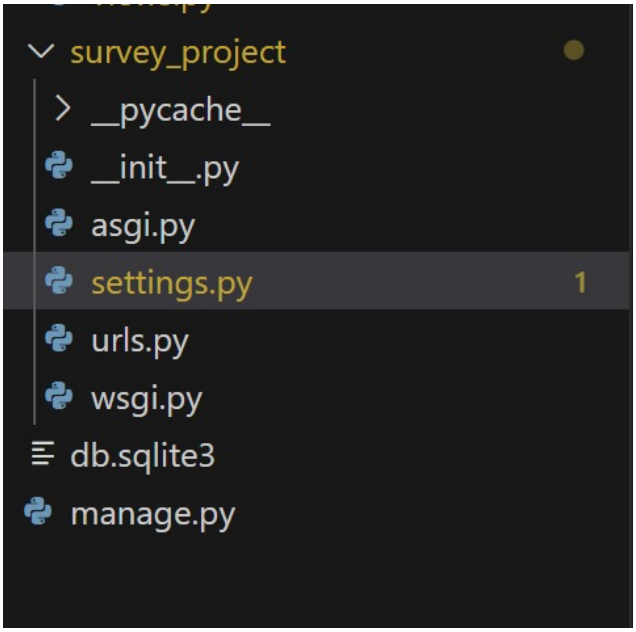
survey_list.html

6. Below is the structure of the project

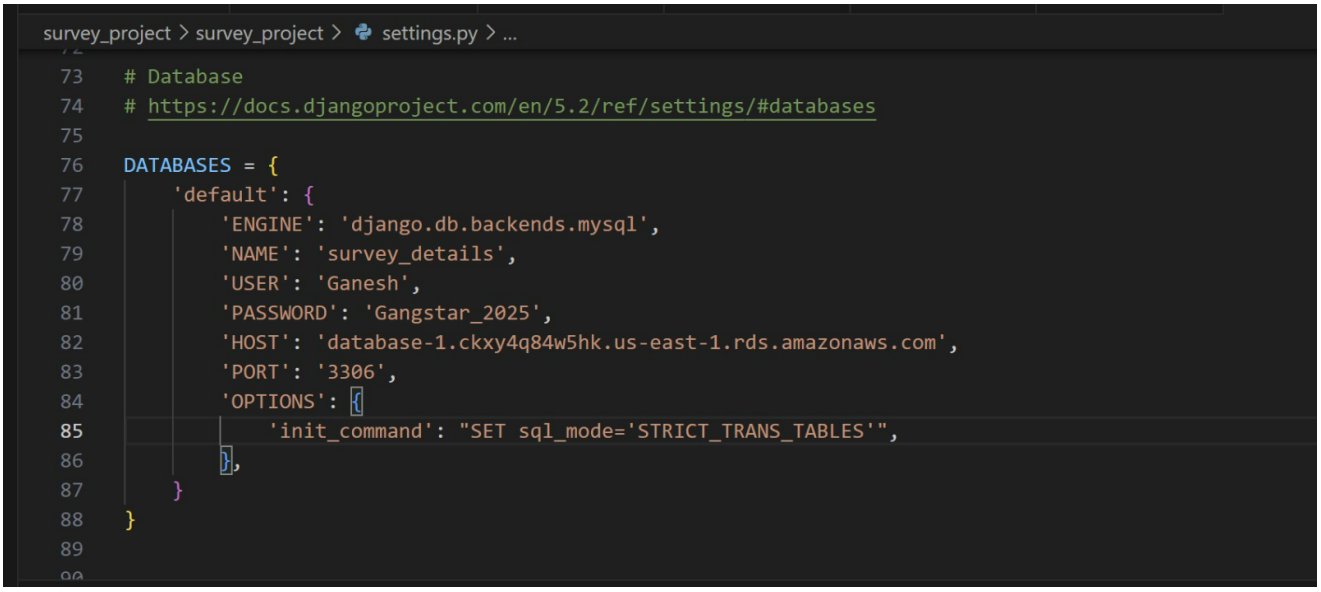
Front end:



Backend:



7. For backend database connectivity edit setting.py file and add the respective database details as shown in the below screenshot.



Implementation details:

1. Model Design (models.py)

The Survey model was created to store survey information. Each field in the model represents one form input, such as:

first_name, last_name, email, etc. – simple text fields

liked_most – a MultiSelectField (or similar logic) to allow multiple choices

source_of_interest – a single choice field using Django choices

date_of_survey – stores the selected date

Each field corresponds to a form input (e.g., first name, liked options, etc.).

2. Form Handling (forms.py)

A SurveyForm (a subclass of ModelForm) was implemented to automatically generate the form based on the Survey model fields. Custom widgets and validation rules are handled here. In Django, custom widgets are used to control how specific form fields are rendered in HTML.

Eg: The date_of_survey field uses a DateInput widget with type="date" so that it shows a proper calendar date picker in the browser.

3. View Logic (views.py)

survey_form_view()

Handles both creation and update based on the presence of an id.

On valid submission, saves the data and uses Django's messages to show feedback.

If updating, it redirects to the survey list page to give users a confirmation via alert.

survey_list_view()

Fetches all survey objects and displays them in a table.

delete_survey_view()

Deletes a survey after confirmation.

Shows a "Deleted successfully" alert using Django messages and JavaScript alert()

4. Templates (.html Files)

survey_form.html:

Renders the form with Bootstrap styling and JavaScript zip code autofill.

survey_list.html:

Displays all surveys and provides "**Update**" and "**Delete**" actions for each.

base.html:

Shared layout template.

5. Message alerts for form submissions and actions:

Django's messages.success() is used to pass messages like:

"Survey submitted successfully!"

"Survey updated successfully!"

"Deleted successfully!"

These messages are rendered in JavaScript alert boxes for immediate feedback

6. Database Integration

MySQL is configured in settings.py to store data remotely.

Migrations (makemigrations and migrate) are used to sync models to tables.

Data retrieval and manual queries are tested via MySQL Workbench.

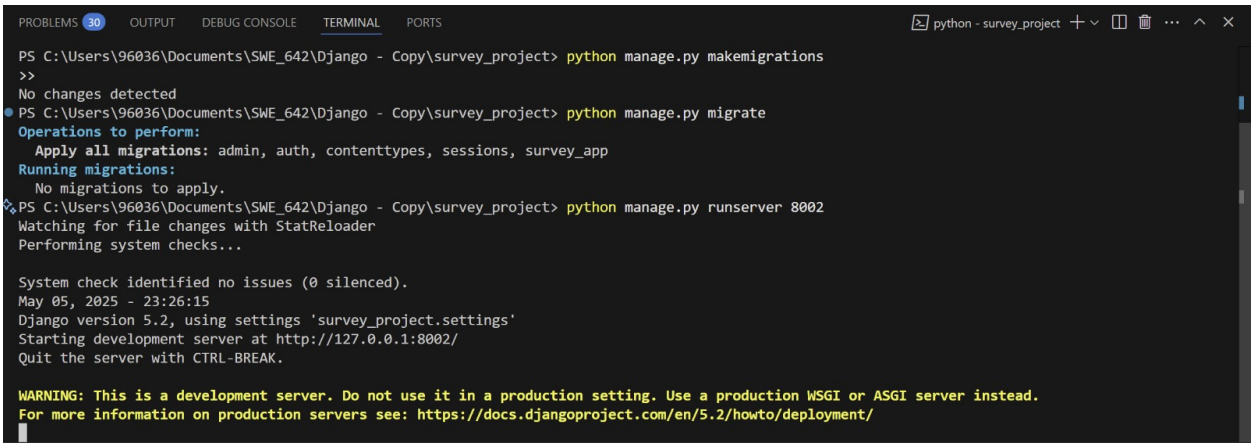
Running the project:

Save the project. Now enter the below commands.

python manage.py makemigrations

python manage.py migrate

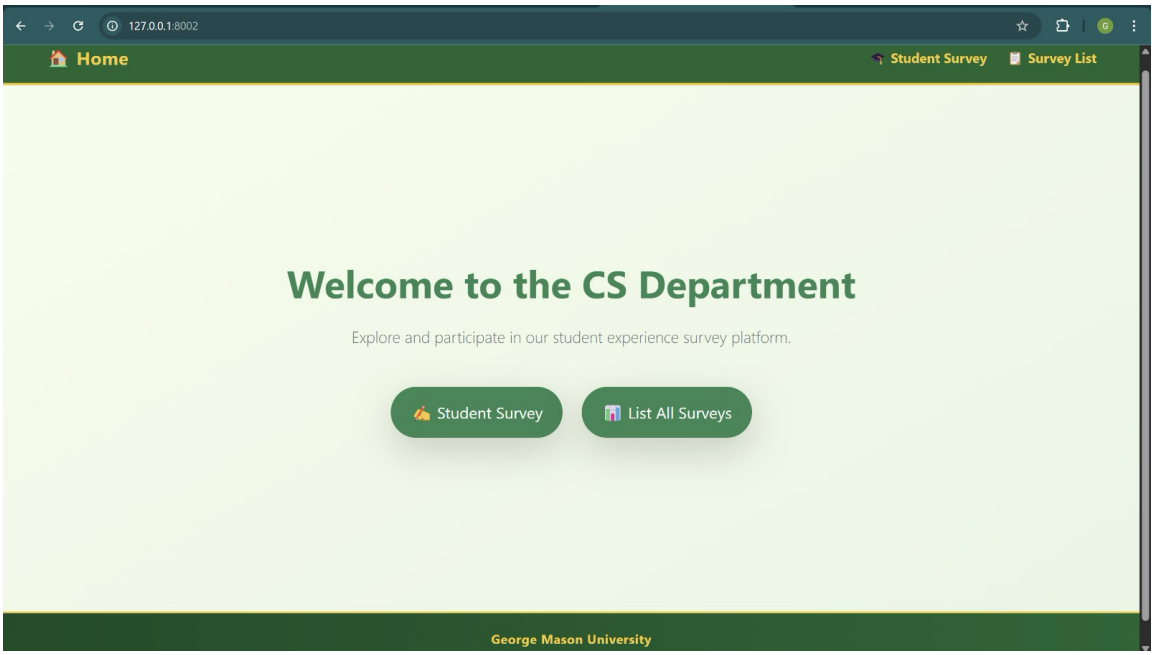
python manage.py runserver 8002



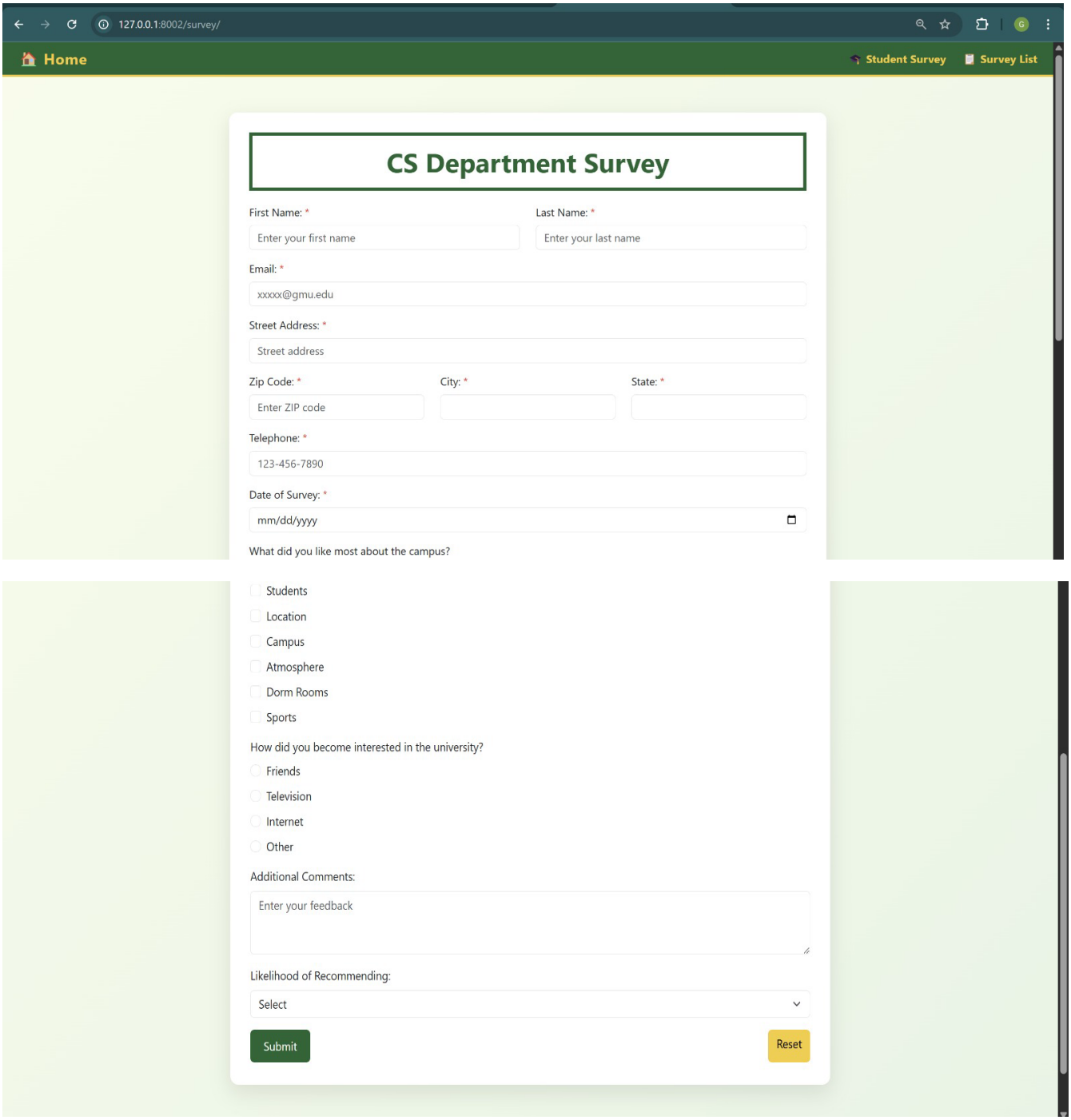
Now we have the application running in localhost on port 8002.

Screenshots of front end pages:

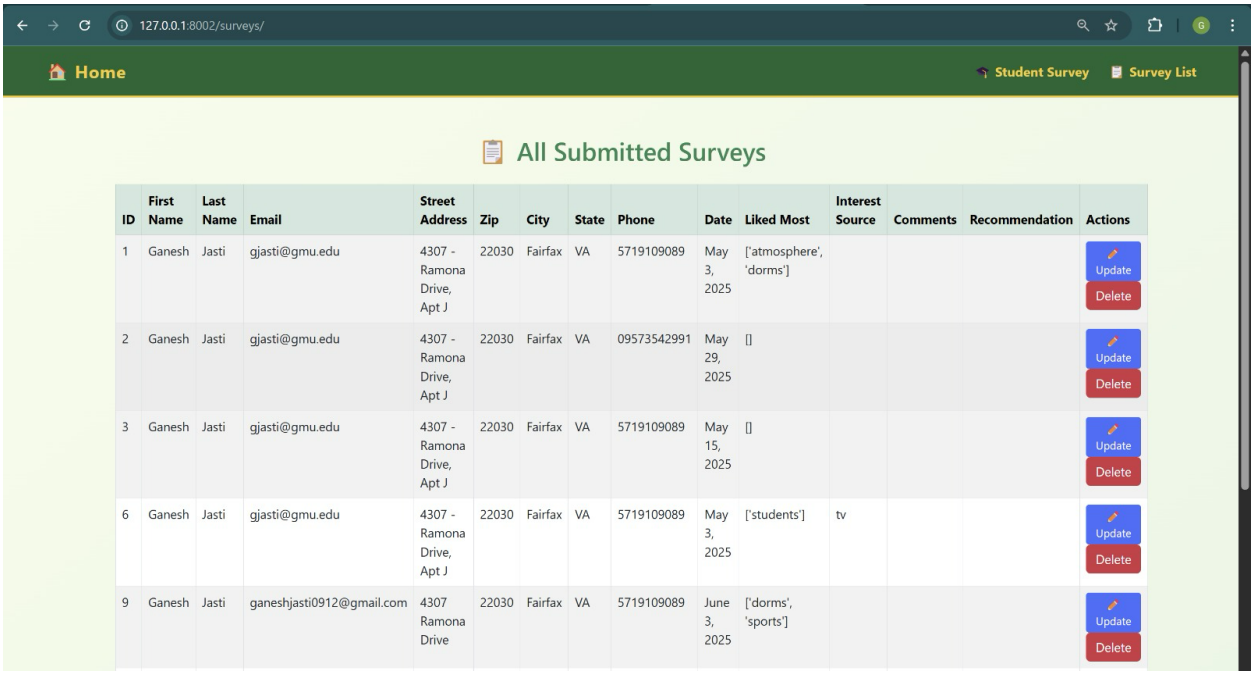
Home Page:



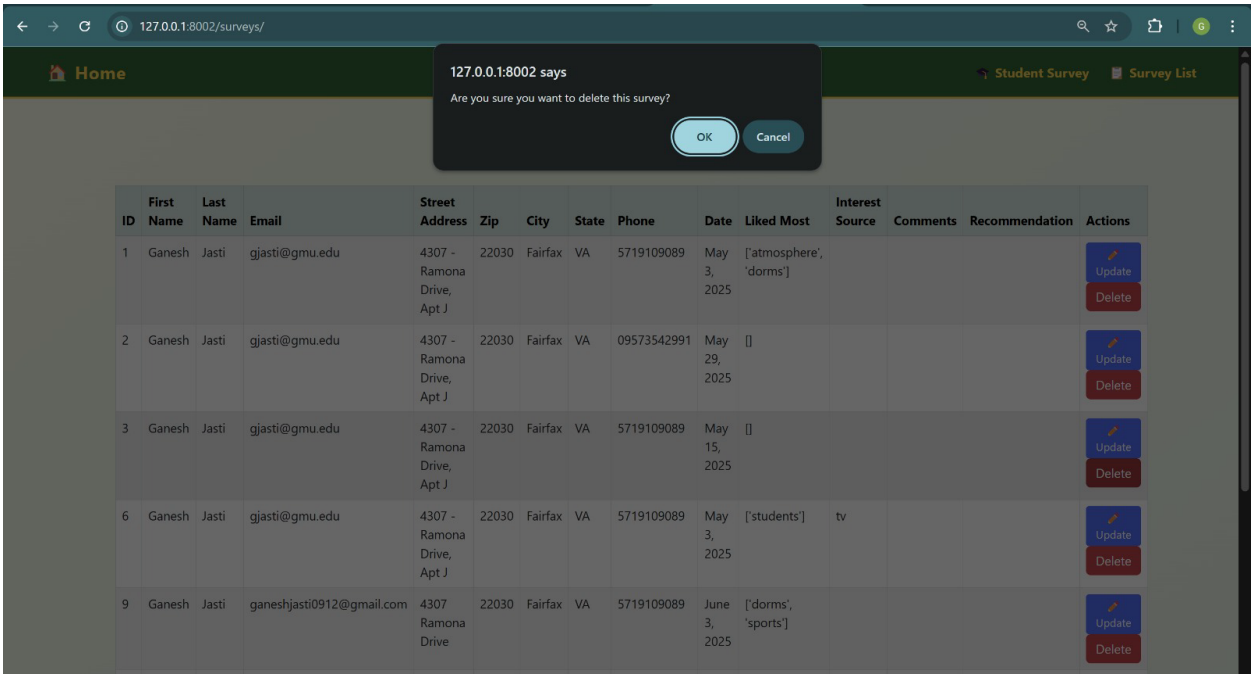
Student survey page:



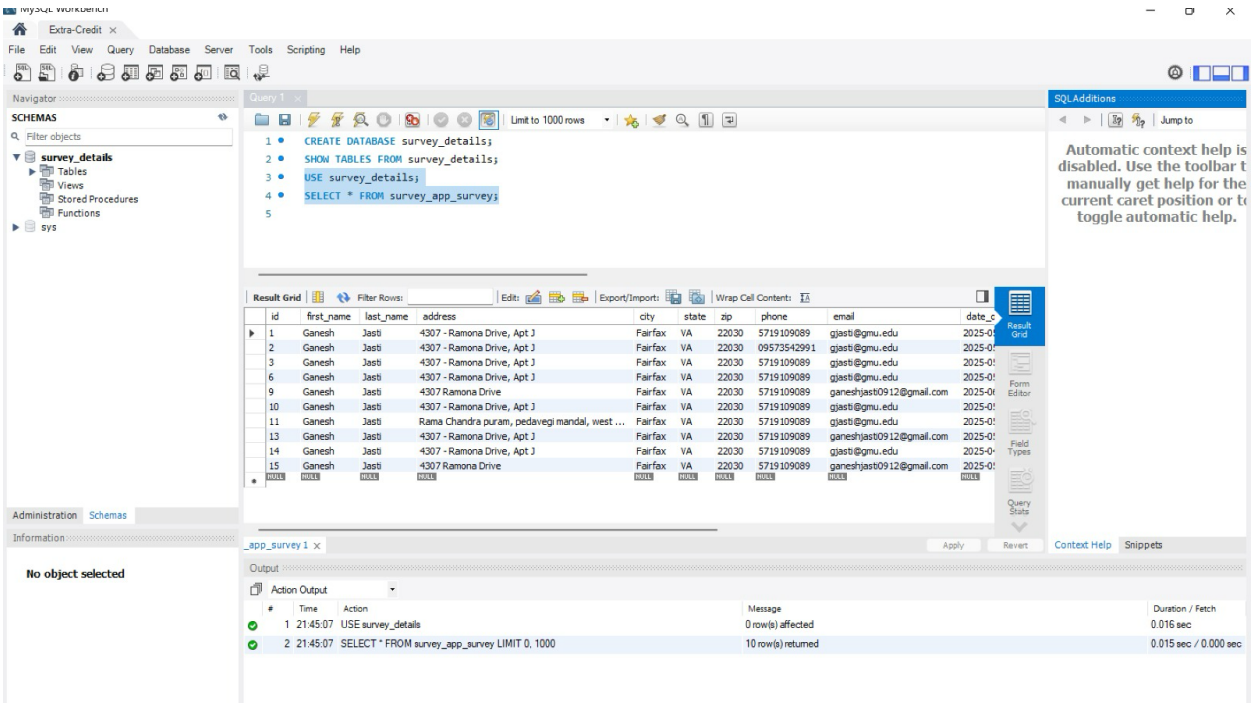
Survey list:



Deleting record :



Database result:



Conclusion:

We were able to implement CRUD operation to manage student survey data in the database such as MySQL successfully by implementing the both frontend application and back end application using Django.