

SWE 645 – Homework 3

Group Name: Alpha1

Name	GID	Contribution
Venkata Sathya Gopala Krishna Chada	G01477305	Micro Service based Application, Video, Documentation
Reeva Chandeshwar Prasad	G01478485	MYSQL & RDS, Video, documentation
Pavansrivatsa Meka	G01480617	Docker and Deployment, Video & documentation
S M A Sri Harsha Sunkavalli	G01463839	Deployment and CI/CD pipeline, documentation

1. Accessible URLs

- *Kubernetes URL:* <http://ec2-44-209-242-173.compute-1.amazonaws.com:30116/api/students>
- *GitHub source code URL:* <https://github.com/Reevacprasad/swe645hw3>

2. Amazon RDS-MySQL database

- Login to AWS Console account
- Search or click on the AWS RDS service.
- Once the RDS dashboard UI is displayed, click on the Create Database button
- Under the Create Database page, provide details of the database. We selected MySQL engine option with the latest version and select Free Tier option.
- Under Settings, provide DB instance ID or the default identifier, in our case default was database-1. Then provide username and credentials which will be later used to connect to the DB instance.
- Under Instance Configuration, you will have option to select db.t3.micro or t2.micro/t4.micro provided with the free tier option.
- For Connectivity, we selected the default VPC and subnet group, and selected Public Access as Yes.
- For VPC security group, click create New and provide security group name and availability zone as no preference.
- Select database password authentication and provide database name.
- Click on the Create Database button. It will take few minutes to become Available.
- Click on the Connectivity & Security, click on the VPC security group name.
- Edit Inbound Rules section, click on the add rule button to add all applicable and necessary network protocols and ports and select available from anywhere, and click save.

Inbound rules (5)

 [Manage tags](#) [Edit inbound rule](#)

<input type="checkbox"/>	Name	Security group rule ID	IP version	Type	Protocol	Port range
<input type="checkbox"/>	-	sgr-0f97aecd7cb085383	IPv4	Custom TCP	TCP	8080
<input type="checkbox"/>	-	sgr-0023bf5c7829b74f9	IPv4	HTTP	TCP	80
<input type="checkbox"/>	-	sgr-090740b5c3e070fcb	IPv4	HTTPS	TCP	443
<input type="checkbox"/>	-	sgr-0dd3eec2f03d8c604	IPv4	All TCP	TCP	0 - 65535
<input type="checkbox"/>	-	sgr-0ae0b46346ccaa2ca	IPv4	MYSQL/Aurora	TCP	3306

MySQL Workbench download and connect:

- Navigate to <https://dev.mysql.com/downloads/workbench/> and download the applicable version for your machine.
- Once the file is downloaded, run the file and open the MySQL Workbench User application.
- Click on the setup new connection, and provide the Hostname with database endpoint **swe645hw3.coqjjbgylstn.us-east-1.rds.amazonaws.com** and username credentials that was setup while doing RDS. Then click Ok and connect to the database.

3. SPRING BOOT To Create Microservices-based Student Survey Application which connects and performs CRUD operations with MYSQL:

Getting Started:

1. To begin, go to <https://start.spring.io/> and create a new Maven project with the required dependencies. Download the zip file and extract it.
 - Project: Maven
 - Language: Java
 - Spring Boot: Choose the Latest stable version
 - Group: Give a name for group
 - Artifact: Give a name for artifact.
 - Packaging: Choose jar
 - Dependencies: Spring Web, Spring Data JPA, MySQL Driver.
 - Click Generate to download the project.
 - Extract the project
2. Import the downloaded project into Eclipse EE IDE.
3. Setting up Database: Update application.properties file in src/main/resources folder in the project as below.

```
spring.application.name=hw3
spring.datasource.url=jdbc:mysql://swe645hw3.coqjjbgylstn.us-east-1.rds.amazonaws.com/hw3db
spring.datasource.username=root
spring.datasource.password=12345678
spring.jpa.hibernate.ddl.auto=update
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

4. Create a JPA entity in swe645.hw3.repository package with all the fields and annotations required.

5. In the same hw3 package create a Controller and Repository for writing the api's and for performing queries in the database.
6. In the Controller file, create few endpoints for performing the following operations on the table.
 - Create a new survey
 - View all surveys
 - View individual survey by surveyId
 - Update or edit an existing survey
 - Delete a survey

The following are the endpoints to test the api's in local using Postman (Install Postman to test the endpoints in local)

Create (POST) - <http://localhost:8080/api/students>

View all (GET) - <http://localhost:8080/api/students>

View by id (GET) - <http://localhost:8080/api/students/{id}>

Update by using id (PUT) - <http://localhost:8080/api/students/{id}>

Delete a survey (DELETE) - <http://localhost:8080/api/students/{id}>

- Sample Body → raw → JSON format: (POST and PUT)
- GET can be done directly

Sample PUT api:

```
{  
  "firstName": "John",  
  "lastName": "Doe",  
  "email": "johndoe@example.com",  
  "streetAddress": "123 Elm Street",  
  "city": "Fairfax",  
  "state": "VA",  
  "zip": "22030",  
  "telephoneNumber": "123-456-7890",  
  "dateOfSurvey": "2024-11-25",  
  "likedMost": "Friendly Staff",  
  "interestSource": "Social Media",  
  "recommendationLikelihood": "Highly Likely"  
}
```

SurveyRepository Interface:

The StudentRepository interface extends the JpaRepository to perform database operations on the Survey entity.

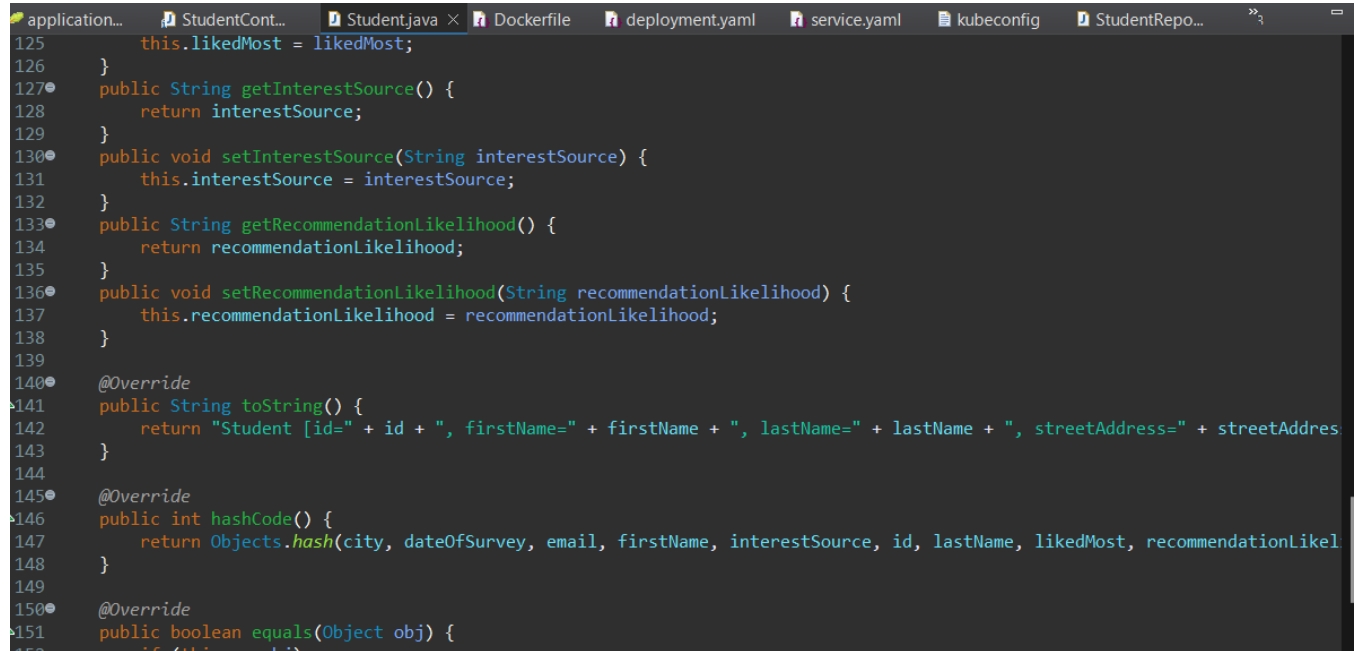
```
application... StudentCont... Student.java Dockerfile deployment.yaml service.yaml kubeconfig StudentRepo... ×
1 package swe645.hw3.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5
6
7 public interface StudentRepository extends JpaRepository<Student, Long> {
8
9 }
```

StudentController:

The StudentController class acts as the RESTful controller, providing endpoints to interact with the survey data. It handles HTTP requests for creating, retrieving, updating, and deleting survey records.

```
application... StudentCont... × Student.java Dockerfile deployment.yaml service.yaml kubeconfig StudentRepo... ×
1 package swe645.hw3.controller;
2
3 import java.util.List;
4
5
6
7 @RestController
8 @RequestMapping("/api/students")
9 public class StudentController {
10     private StudentService studentService;
11
12     public StudentController(StudentService studentService) {
13         super();
14         this.studentService = studentService;
15     }
16
17     @PostMapping
18     public ResponseEntity<Student> saveStudent(@RequestBody Student student) {
19         return new ResponseEntity<Student>(studentService.saveStudent(student), HttpStatus.CREATED);
20     }
21
22     @GetMapping
23     public List<Student> getAllStudents(){
24         return studentService.getAllStudents();
25     }
26
27     @GetMapping("/{id}")
28     public ResponseEntity<Student> getStudentById(@PathVariable("id") long studentId){
29         return new ResponseEntity<Student>(studentService.getStudentById(studentId), HttpStatus.OK);
30     }
31 }
```

Student Object:



```
125     this.likedMost = likedMost;
126 }
127 public String getInterestSource() {
128     return interestSource;
129 }
130 public void setInterestSource(String interestSource) {
131     this.interestSource = interestSource;
132 }
133 public String getRecommendationLikelihood() {
134     return recommendationLikelihood;
135 }
136 public void setRecommendationLikelihood(String recommendationLikelihood) {
137     this.recommendationLikelihood = recommendationLikelihood;
138 }
139
140 @Override
141 public String toString() {
142     return "Student [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", streetAddress=" + streetAddress;
143 }
144
145 @Override
146 public int hashCode() {
147     return Objects.hash(city, dateOfSurvey, email, firstName, interestSource, id, lastName, likedMost, recommendationLikelihood);
148 }
149
150 @Override
151 public boolean equals(Object obj) {
152     if (this == obj)
153         return true;
154     if (obj == null)
155         return false;
156     if (obj.getClass() != this.getClass())
157         return false;
158     Student that = (Student) obj;
159     return id == that.id && firstName == that.firstName && lastName == that.lastName && streetAddress == that.streetAddress && city == that.city && dateOfSurvey == that.dateOfSurvey && email == that.email && interestSource == that.interestSource && likedMost == that.likedMost && recommendationLikelihood == that.recommendationLikelihood;
160 }
```

4. Docker Containerization

1. Sign in to hub.docker.com and install Docker Desktop and login.
2. Create a Docker file

```
# Use an official Java runtime as a parent image
FROM openjdk:23-jdk-slim

# Set the working directory inside the container
WORKDIR /app

# Copy the JAR file to the container
COPY target/hw3-0.0.1-SNAPSHOT.jar hw3-0.0.1-SNAPSHOT.jar

# Expose the port the app runs on
EXPOSE 8080

# Run the application
ENTRYPOINT ["java", "-jar", "hw3-0.0.1-SNAPSHOT.jar"]
```

3. To sign with docker credentials on the command line, use `docker login -u <username>`.
Enter password when prompted.

4. Build the image using the docker file with the following command.
`docker build -t gopalchada10010/swe645:01 .`
5. Run the image to create a container with the command below and verifying if it is correct.
`docker run -p 8080:8080 gopalchada10010/swe645:01`
6. Push the image into the docker hub using the following commands
`docker push gopalchada10010/swe645:01`
7. Verify whether the image is pushed or not by logging into dockerhub and check under Repositories section as below.

5. Deployment of containerized application on Kuburnetes through rancher

1. Log into the AWS Learner Lab and go to the EC2 Instances page.
2. Create three EC2 instances with the following settings:
 - Use the Ubuntu Server 22.04 AMI LTS (HVM), SSD Volume Type.
 - Security group allowing inbound traffic on ports 8080, 80, 443, and 22 from anywhere.
 - Assign 30GB storage to each instance.
 - Configure outbound rules to allow all traffic.
3. Assign Elastic IPs to each instance.
4. Connect to instances 1 and 2 to install Docker.
5. Install Docker on both instances using the following commands:
 - `sudo su -`
 - `sudo apt-get update`
 - `sudo apt install docker.io`
6. Start the Rancher server on instance 1.
 - Use the below command to start the Rancherserver
`$ sudo docker run --privileged -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher`
 - Once that install is complete, run “sudo docker ps” to view the current docker instance information and note down the container-ID.
 - In the UI, copy the password command and paste in instance1 console to get the default password to enter in the RancherUI for the first time login. The noted container-ID must be replaced in the below command
`sudo docker logs container-ID 2>&1 | grep “Bootstrap Password:”`
 - Use the generated password to login and once login, select the option “Set a specific password to use” to set a custom password for future.
7. Set up Kubernetes on instance 2.
 - Once custom password is set navigate to home and create a new cluster by clicking the create button and selecting custom under “Use existing nodes and create and cluster using RKE”. Give the cluster a name and click Create. In the next page click the three checkboxes named etcd, Control Plane and Worker and apply the checkbox – Insecure: Select this to skip TLS

verification if your server has a self-signed certificate. This would give a registration command to be pasted on instance2 connect console.

- Once the command is completed on instance2 connect console, wait in the RancherUI till the cluster state is changed to 'active' from 'updating'. Once the state is active the cluster is ready to deploy
- Now install kubectl with the command `snap install kubectl --classic` in console
- Download the KubeConfig file by selecting the cluster and clicking on Download KubeConfig button
- Copy the local Kubeconfig file into the console using "nano config" and move it then to kube folder
 - `mkdir -p ~/.kube`
 - `mv config ~/.kube/config`
 - `chmod 600 ~/.kube/config`
- Now create deployment.yaml and service.yaml files referencing from hw2 and changing the docker images name accordingly
- Apply the deployment and service using the following commands
 - `kubectl apply -f deployment.yaml`
 - `kubectl apply -f service.yaml`
- Get the node port number by the following command "kubectl get service"
- Add the Node port number you get on running the above deployments command to the security group as custom TCP and access from anywhere.
- You can access the application using the following format:
- `<Public DNS IP of instance 2>:<assigned NodePort>/<api for CRUD op>`

<http://ec2-44-209-242-173.compute-1.amazonaws.com:30116/api/students>

Create (POST) - <http://ec2-44-209-242-173.compute-1.amazonaws.com:30116/api/students>

View all (GET) - <http://ec2-44-209-242-173.compute-1.amazonaws.com:30116/api/students>

View by id (GET) –

`http://ec2-44-209-242-173.compute-1.amazonaws.com:30116/api/students/{id}`

Update by using id (PUT) –

`http://ec2-44-209-242-173.compute-1.amazonaws.com:30116/api/students/{id}`

Delete a survey (DELETE) –

`http://ec2-44-209-242-173.compute-1.amazonaws.com:30116/api/students/{id}`

6.Setting up Jenkins to run CI/CD pipeline

1. SSH into instance 3.

a. Install Docker, JDK:

- Update the system packages: ``sudo apt-get update``

- Install Docker: ``sudo apt-get install -y docker.io``

- Install Java Development Kit (JDK): ``sudo apt install openjdk-11-jre-headless``

- b. Install and Add Jenkins repository key: `wget -O /usr/share/keyrings/jenkins-keyring.asc https://pkg.jenkins.io/debian-stable/jenkins.io.key`

- `echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/" | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null`
- `sudo apt-get update`
- `sudo apt-get install Jenkins`

If there is any error while installing Jenkins run below command

- `curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null`
- `echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null`
- `sudo apt-get update`
- `sudo apt-get install fontconfig openjdk-17-jre`
- `sudo apt-get install jenkins`

2. Start Jenkins service:

- ``sudo systemctl start jenkins``
- ``sudo systemctl enable jenkins``
- Verify Jenkins is running: ``sudo systemctl status jenkins``

3. Access the Jenkins UI at: ``<instance 3 public IPv4 address>:8080``

4. Retrieve the initial Jenkins admin password:

- ``sudo cat /var/lib/jenkins/secrets/initialAdminPassword``
- Use the password to log in to Jenkins for the first time.

5. Install the required Jenkins plugins:

- Navigate to **Manage Jenkins** -> **Manage Plugins**.
- Install plugins such as Git, Docker Pipeline, and Kubernetes CLI to ensure integration capabilities.

6. Add GitHub and Docker credentials to Jenkins:

- Go to **Manage Jenkins** -> **Manage Credentials**.
- Create credentials for GitHub and Docker Hub (use username and password).
- Create credentials for Kubeconfig(Secret file)

7. To install maven in the instance for jenkins usage

- `sudo apt-get update`
- `sudo apt-get install maven`
- `mvn --version`
- `sudo systemctl restart jenkins`
- `sudo systemctl status jenkins`

8. Configure Jenkins to use Docker:

- Add Jenkins user to the Docker group to allow Jenkins to run Docker commands: ``sudo usermod -`

aG docker jenkins`

- Restart Jenkins to apply changes: `sudo systemctl restart jenkins`

9. Create a Jenkins pipeline for CI/CD:

- From the Jenkins dashboard, click ****New Item****.

- Provide a name for the pipeline (e.g., `cicdPipeline`) and select ****Pipeline****.

- Under ****Build Triggers****, select ****Poll SCM**** and set the schedule to `* * * * *` for every minute.

- Under ****Pipeline****, select ****Pipeline script from SCM**** and choose ****Git**** as the SCM.

- Provide the GitHub repository URL and select the saved credentials.

- Specify the branch to use (`main`) and the script path (`Jenkinsfile`).

10. Test the Jenkins pipeline:

- Make a commit to the GitHub repository to trigger the build.

- Verify that Jenkins automatically builds and deploys the updated application to Kubernetes.

Git repository link: <https://github.com/Reevacprasad/swe645hw3>

Rancher URL: <http://ec2-34-237-54-212.compute-1.amazonaws.com>

Jenkins URL: <http://ec2-3-235-81-211.compute-1.amazonaws.com:8080/>

AWS URL to view all surveys: <http://ec2-44-209-242-173.compute-1.amazonaws.com:30116/api/students>