Salesken Technical Assignment for Intern

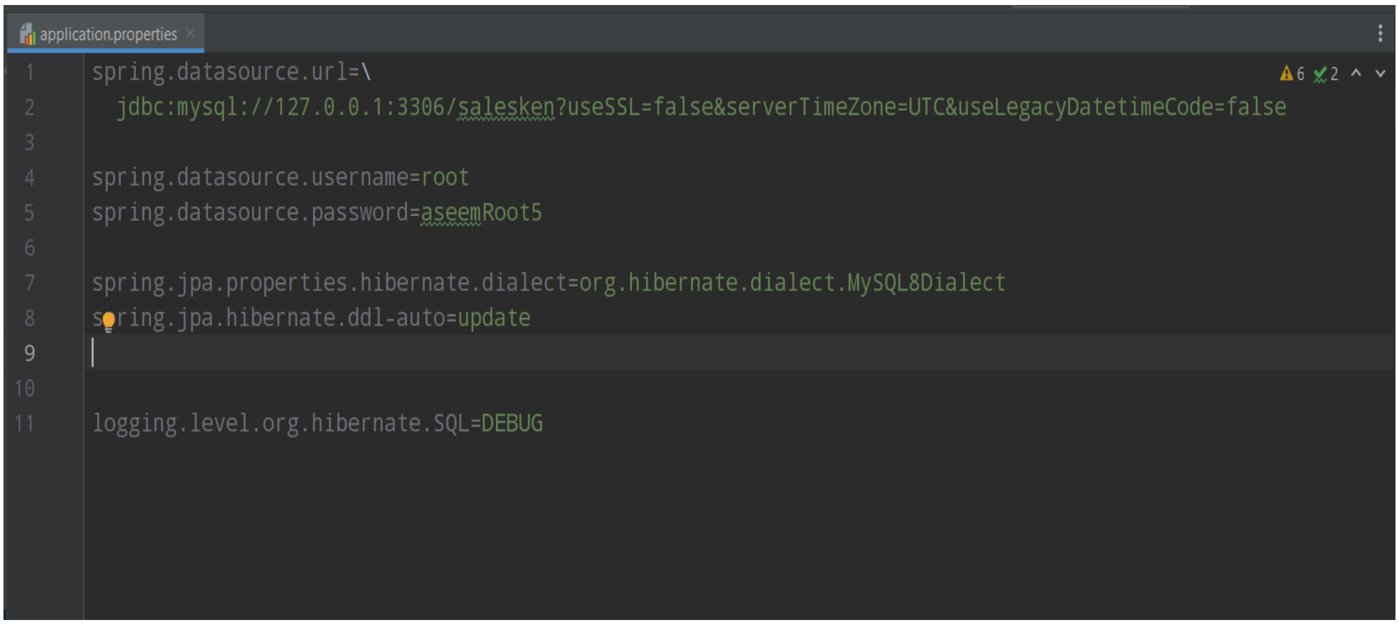
I used Spring Boot framework of JAVA, to complete the assignment. And I used MySQL local server as my database. And for the frontend, I've used Thymeleaf templates along with Bootstrap CSS 4.

Also, I used Docker to dockerize the application as a version control system and to make it possible torun the application on any device without worrying about the dependency issues.

I used Spring Boot because it is the frame work I'm most comfortable with. It allows us to makestandalone applications which can are easily executable and are production ready.
It also provides us with lots of functionalities which makes the development life cycle easy.

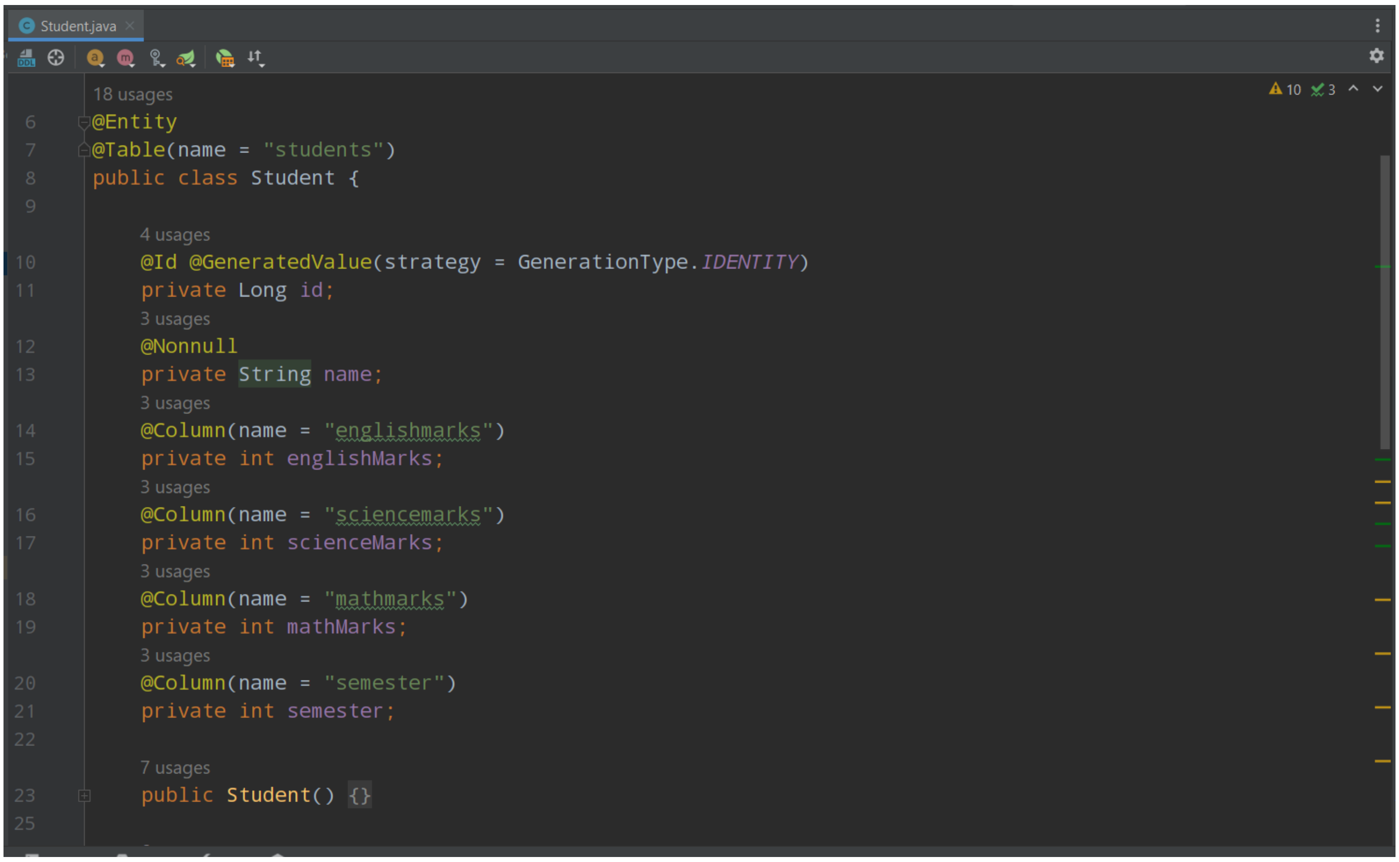
It wasn't possible for me to create a form with dynamic questions with my current skillset and I couldn't find any documentation for the same in my preferred language i.e., JAVA, so I skipped that and made a form with fixed questions.

Firstly, I made a database in MySQL in my local server named Atlan, which would be used as storage for our forms. Then I hard-coded the database connection into the application to bind the two.



```
spring.datasource.url=\
  jdbc:mysql://127.0.0.1:3306/salesken?useSSL=false&serverTimeZone=UTC&useLegacyDatetimeCode=false

spring.datasource.username=root
spring.datasource.password=aseemRoot5

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=update


logging.level.org.hibernate.SQL=DEBUG
```

Then, I defined a simple entity in the program as Student using @Entity annotation, which would serve as the table in our database and also our object of contact for the data in the form.



```java
18 usages
@Entity
@Table(name = "students")
public class Student {

    4 usages
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    3 usages
    @Nonnull
    private String name;
    3 usages
    @Column(name = "englishmarks")
    private int englishMarks;
    3 usages
    @Column(name = "sciencemarks")
    private int scienceMarks;
    3 usages
    @Column(name = "mathmarks")
    private int mathMarks;
    3 usages
    @Column(name = "semester")
    private int semester;

    7 usages
    public Student() {}
```

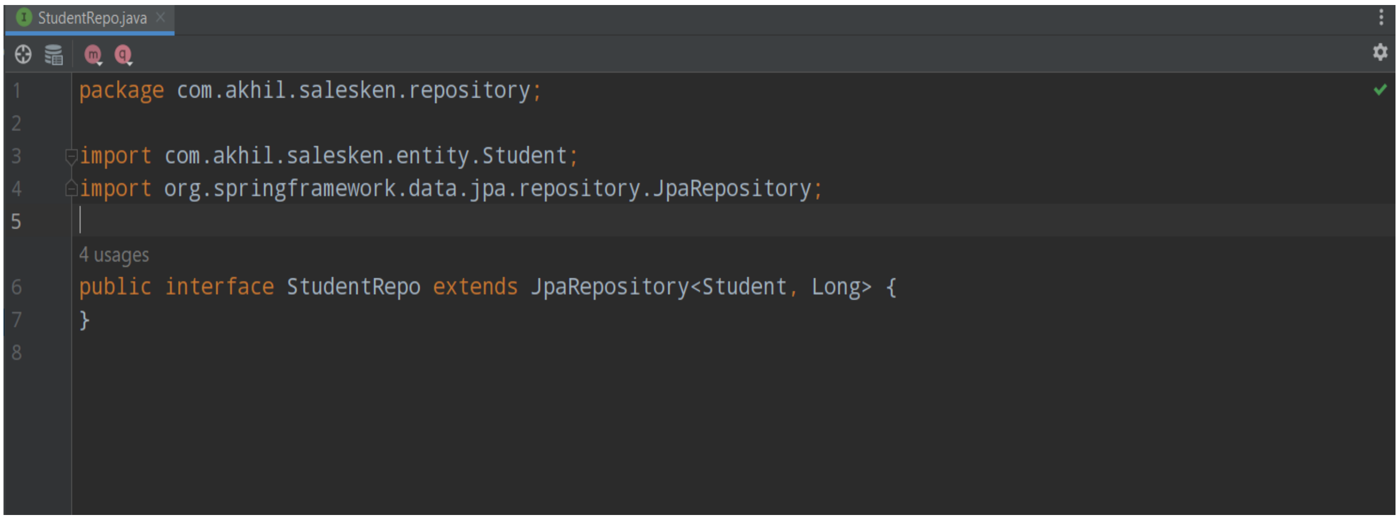
Pros of local MySQL Database:

1. Can easily be changed to a proper remote with just changing the JDBC connection point.
2. Easy to access.
3. Free to use.
4. Extensive documentation support available if needed.
5. Easily scalable.

Cons of local MySQL Database:

1. Not very efficient for large databases.
2. We would have to hard-code the database connection in case of any change.
3. Hard-coded entity structure, for any change in it we'd have to refactor the whole programme.
4. Not very flexible.
5. Upon changing the MySQL database server all previous data will be only.
6. And in case of intentional server change, the data will still be present in the Google Sheet which can pose a security threat.

Then, I defined an Interface StudentRepo which extended an imported interface JpaRepository.This would give us all the access to the user repository in the database.

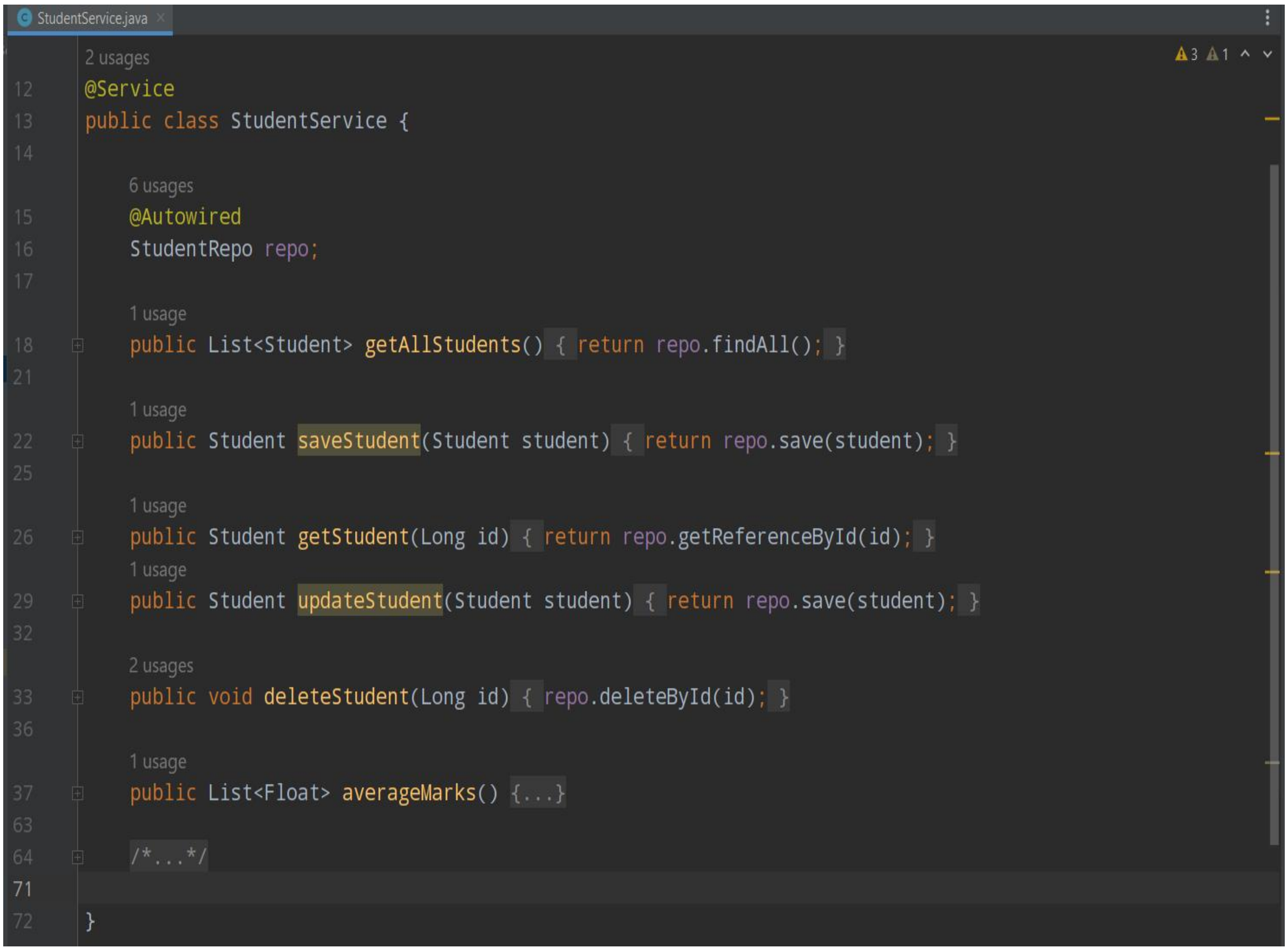


StudentRepo.java

```java
package com.akhil.salesken.repository;

import com.akhil.salesken.entity.Student;
import org.springframework.data.jpa.repository.JpaRepository;

4 usages
public interface StudentRepo extends JpaRepository<Student, Long> {
}
```

We don't need to define any methods into this as all that we need is already present in the JpaRepository we extended. We also don't need to use any annotation for this.

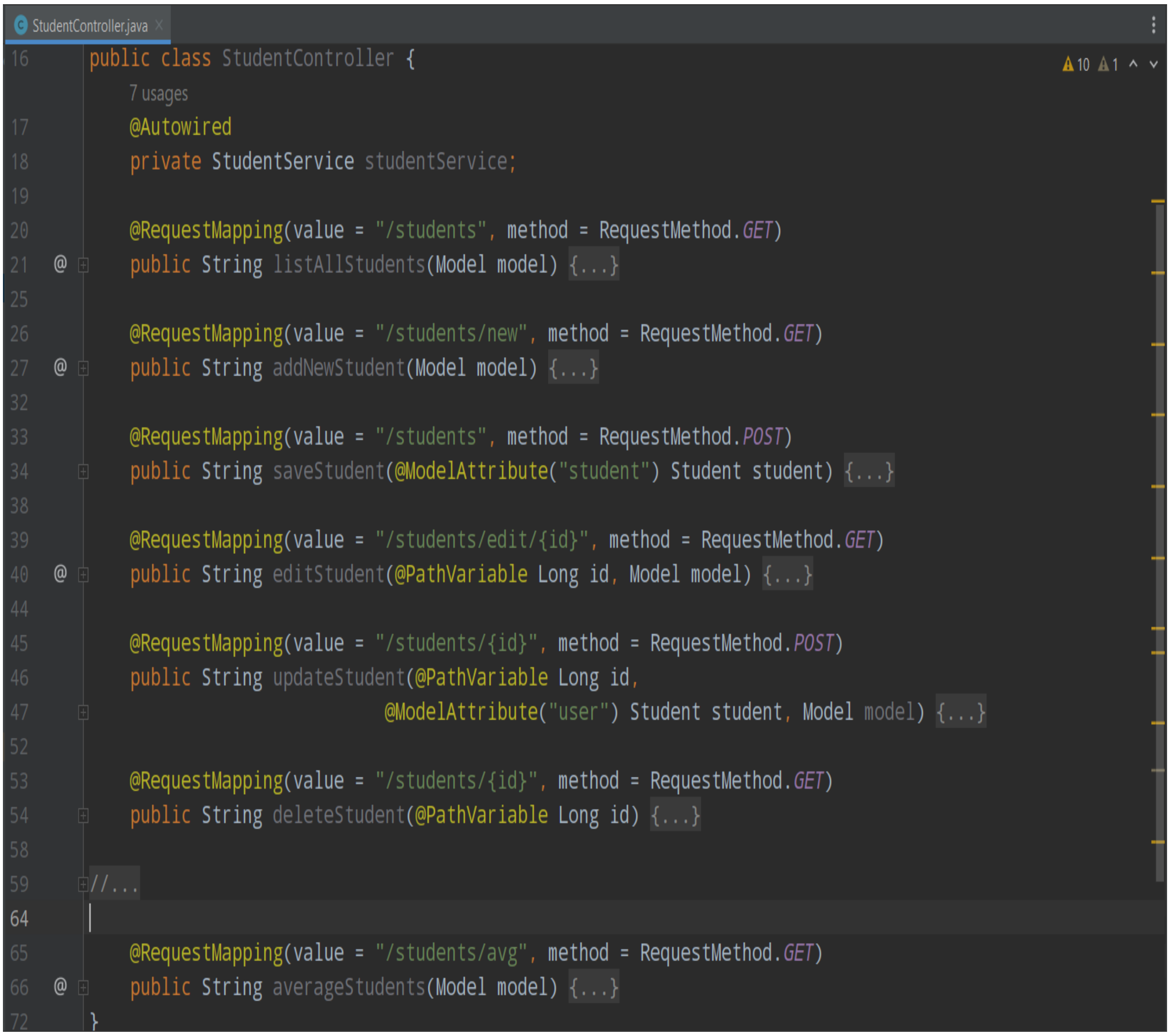
Then, I created a service class StudentService and used @Service annotation for the Spring Boot which allows for implementation classes to be autodetected through classpath scanning. This Service Class used the Autowired UserRepo dependency to implement methods to manipulate the database repository.



StudentService.java

```java
2 usages
@Service
public class StudentService {

    6 usages
    @Autowired
    StudentRepo repo;

    1 usage
    public List<Student> getAllStudents() { return repo.findAll(); }

    1 usage
    public Student saveStudent(Student student) { return repo.save(student); }

    1 usage
    public Student getStudent(Long id) { return repo.getReferenceById(id); }
    1 usage
    public Student updateStudent(Student student) { return repo.save(student); }

    2 usages
    public void deleteStudent(Long id) { repo.deleteById(id); }

    1 usage
    public List<Float> averageMarks() {...}

    /*...*/

}
```

Finally, I made a controller class using @Controller annotation to handle all the incoming web requests. This class as the name suggests is a Controller class as per the MVC model and it has everything we need to fully manipulate the Database of form answers.

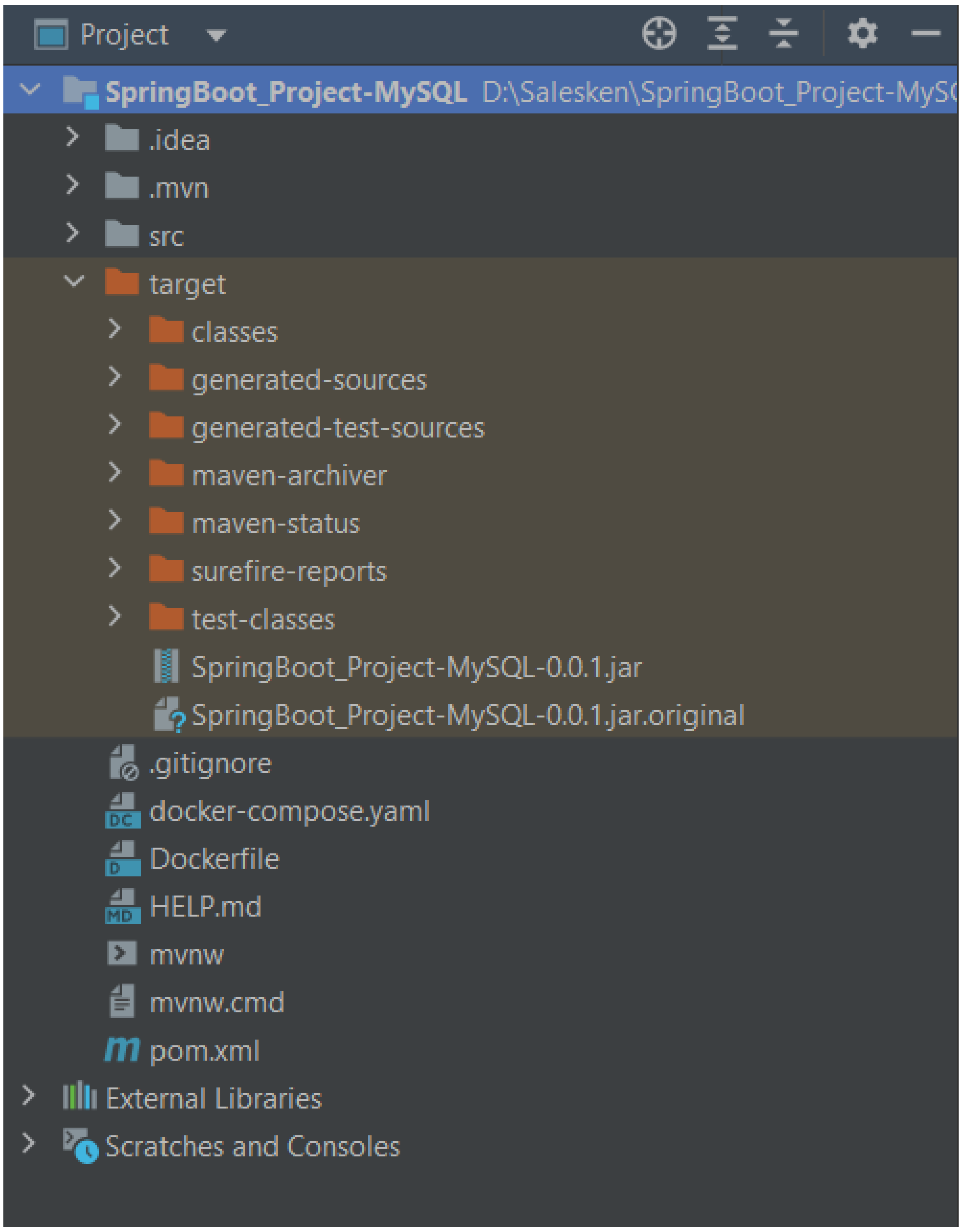


```java
public class StudentController {
    7 usages
    @Autowired
    private StudentService studentService;

    @RequestMapping(value = "/students", method = RequestMethod.GET)
    public String listAllStudents(Model model) {...}

    @RequestMapping(value = "/students/new", method = RequestMethod.GET)
    public String addNewStudent(Model model) {...}

    @RequestMapping(value = "/students", method = RequestMethod.POST)
    public String saveStudent(@ModelAttribute("student") Student student) {...}

    @RequestMapping(value = "/students/edit/{id}", method = RequestMethod.GET)
    public String editStudent(@PathVariable Long id, Model model) {...}

    @RequestMapping(value = "/students/{id}", method = RequestMethod.POST)
    public String updateStudent(@PathVariable Long id,
                                @ModelAttribute("user") Student student, Model model) {...}

    @RequestMapping(value = "/students/{id}", method = RequestMethod.GET)
    public String deleteStudent(@PathVariable Long id) {...}

//...

    @RequestMapping(value = "/students/avg", method = RequestMethod.GET)
    public String averageStudents(Model model) {...}
}
```

All the controller methods directly or indirectly point to a VIEW. The front-end part of the application.

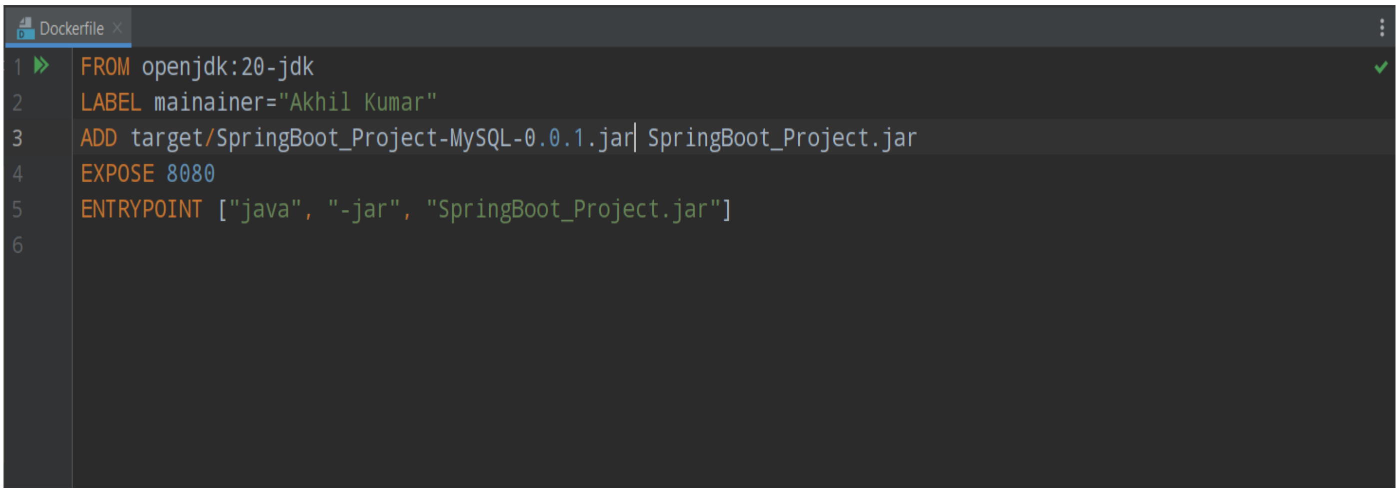
The four views, students.html, new_ student.html, edit_ student.html and average.html are used to view, add new and update students and give us average of students respectively and they are present in the Template package inside the resources, as per the convention.

Finally, after all the coding I made the whole application into a Maven JAR file which contains all the necessary internal dependencies needed to run the web application and can be used directly and easily transferable.



```
Project
SpringBoot_Project-MySQL  D:\Salesken\SpringBoot_Project-MyS
  .idea
  .mvn
  src
  target
    classes
    generated-sources
    generated-test-sources
    maven-archiver
    maven-status
    surefire-reports
    test-classes
    SpringBoot_Project-MySQL-0.0.1.jar
    SpringBoot_Project-MySQL-0.0.1.jar.original
  .gitignore
  docker-compose.yaml
  Dockerfile
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
External Libraries
Scratches and Consoles
```

At the end, I created a deploy by making a simple docker command, using the **Dockerfile** and docker-compose file present in the project.

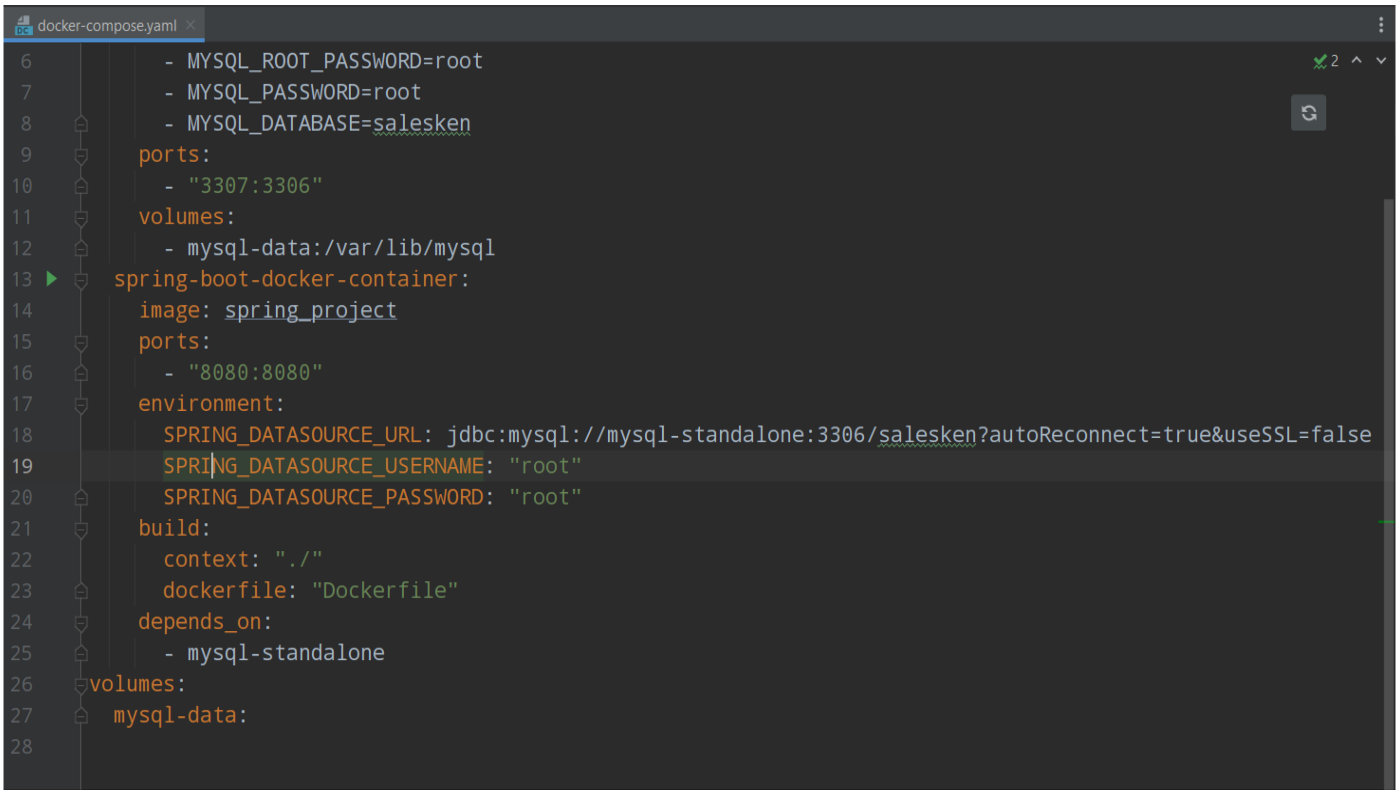


```dockerfile
FROM openjdk:20-jdk
LABEL mainainer="Akhil Kumar"
ADD target/SpringBoot_Project-MySQL-0.0.1.jar SpringBoot_Project.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "SpringBoot_Project.jar"]
```

This file contains the configuration to convert our Spring Boot Application JAR into a docker image which can then be used alongside the image of MySQL database present in the Docker Hub.

Now, we can separately build the Spring Application Container and then the MySQL container and then manually interlink them, but this happens to be a rather tedious task not appropriate to be performed again and again every time the application is executed.

So, I made a **docker-compole.yaml** file which would ensure that both the containers are properly linkedand can be executed at a moment's notice by using a single docker-compose command.
Also, I added "volumes" property to the MySQL container to ensure that the data won't get lost while own containers are down.

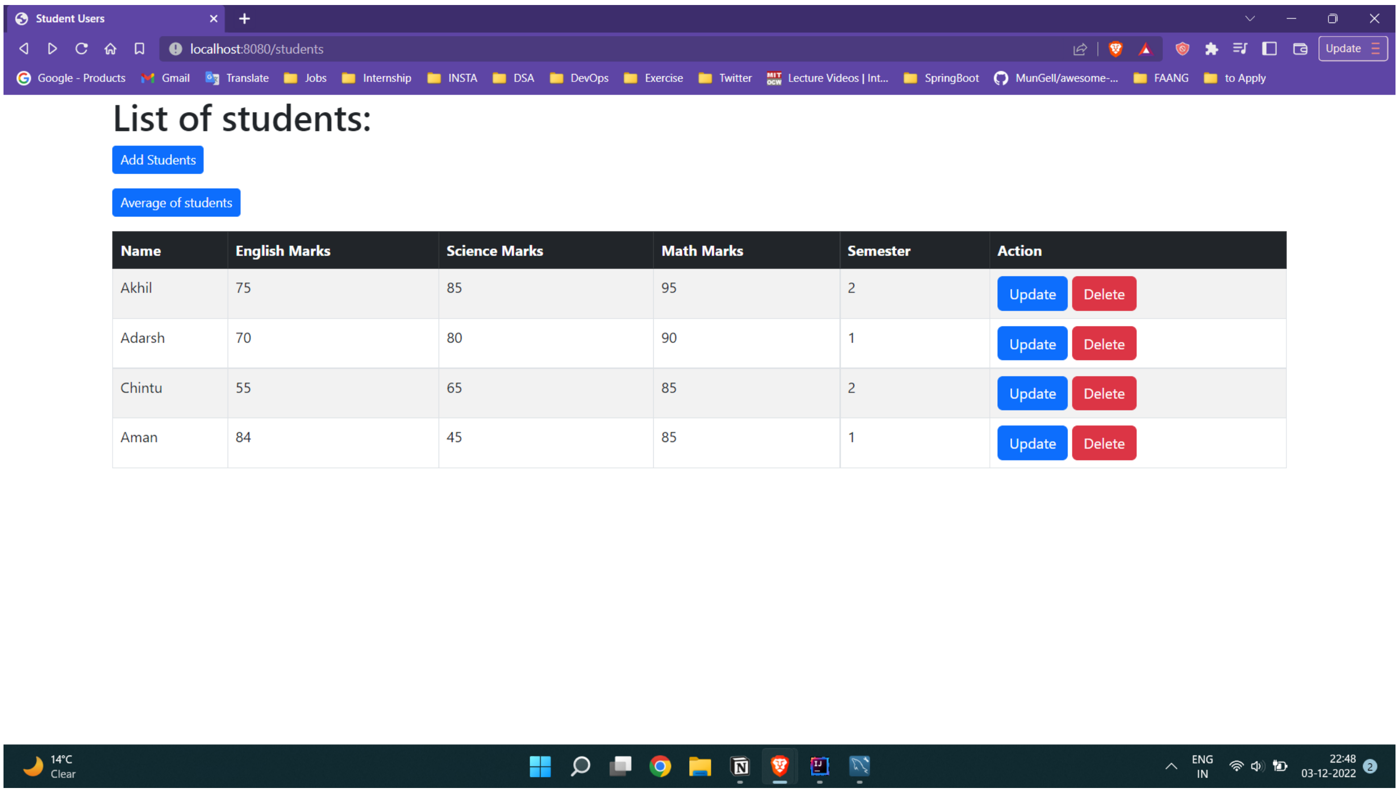


```yaml
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_PASSWORD=root
      - MYSQL_DATABASE=salesken
    ports:
      - "3307:3306"
    volumes:
      - mysql-data:/var/lib/mysql
  spring-boot-docker-container:
    image: spring_project
    ports:
      - "8080:8080"
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql-standalone:3306/salesken?autoReconnect=true&useSSL=false
      SPRING_DATASOURCE_USERNAME: "root"
      SPRING_DATASOURCE_PASSWORD: "root"
    build:
      context: "./"
      dockerfile: "Dockerfile"
    depends_on:
      - mysql-standalone
volumes:
  mysql-data:
```

This **docker-compose.yaml** file makes it so that we can upload it on **DockerHub** and anyone can clone the image and run it on their local system without any trouble or need to download any other external dependency.

Final Product:

The final web app, this is the user.html page which shows us all the users present.

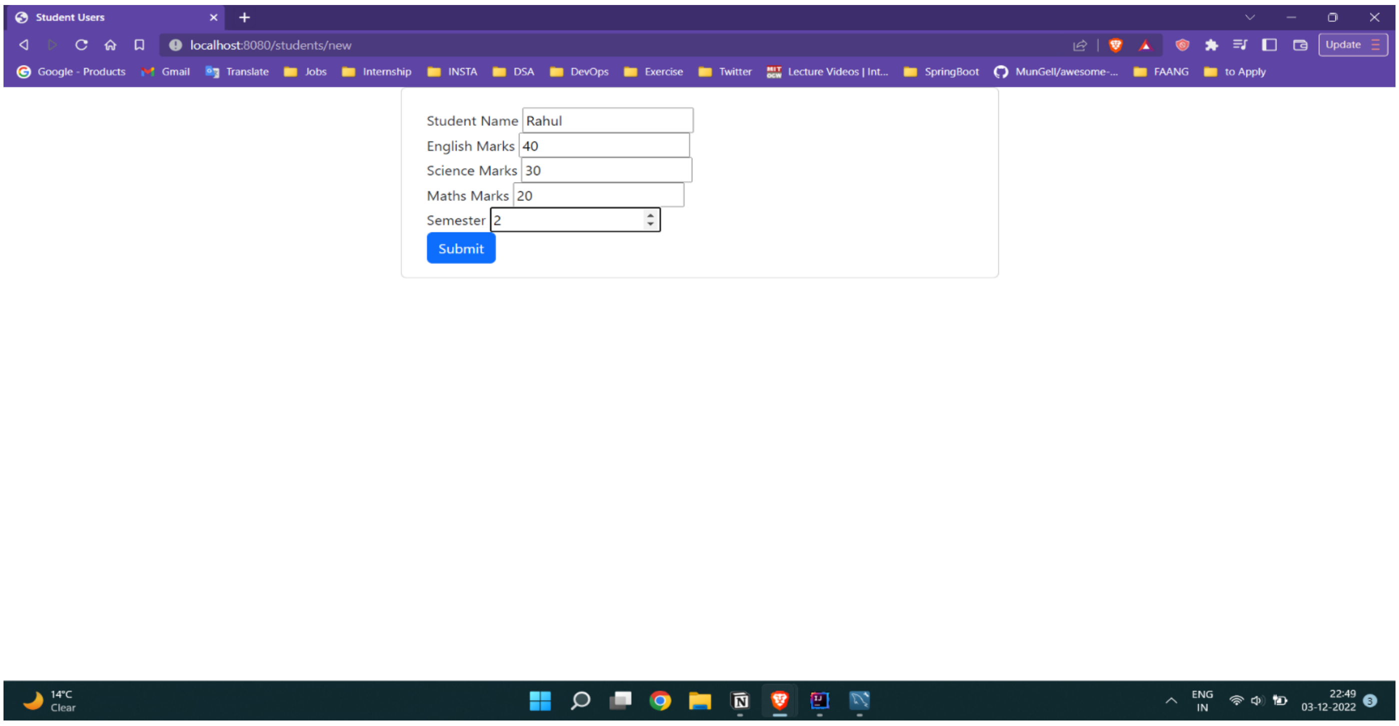

List of students:

Add Students

Average of students

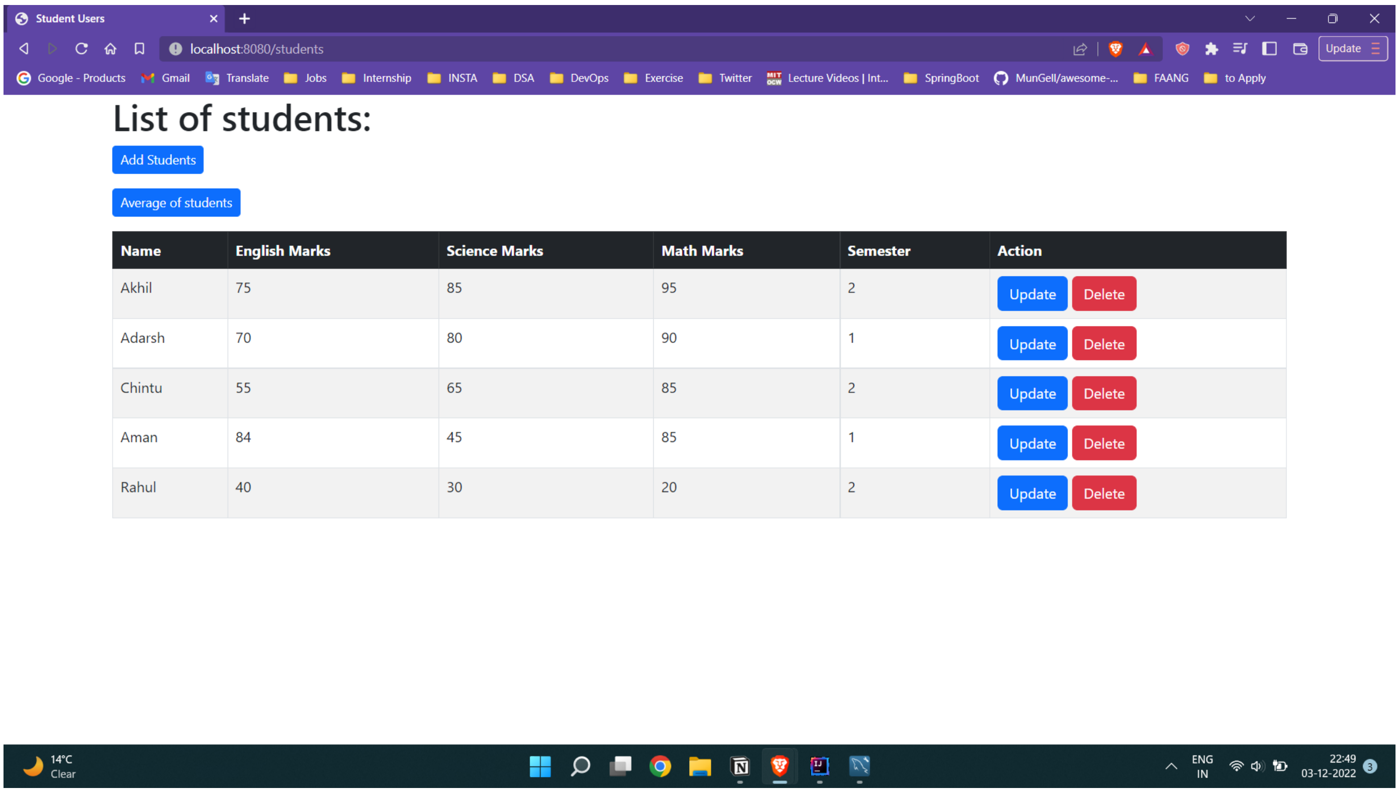
Name	English Marks	Science Marks	Math Marks	Semester	Action
Akhil	75	85	95	2	Update Delete
Adarsh	70	80	90	1	Update Delete
Chintu	55	65	85	2	Update Delete
Aman	84	45	85	1	Update Delete

Page dedicated to adding new students.


Student Name	Rahul
English Marks	40
Science Marks	30
Maths Marks	20
Semester	2

Submit

Then,



localhost:8080/students

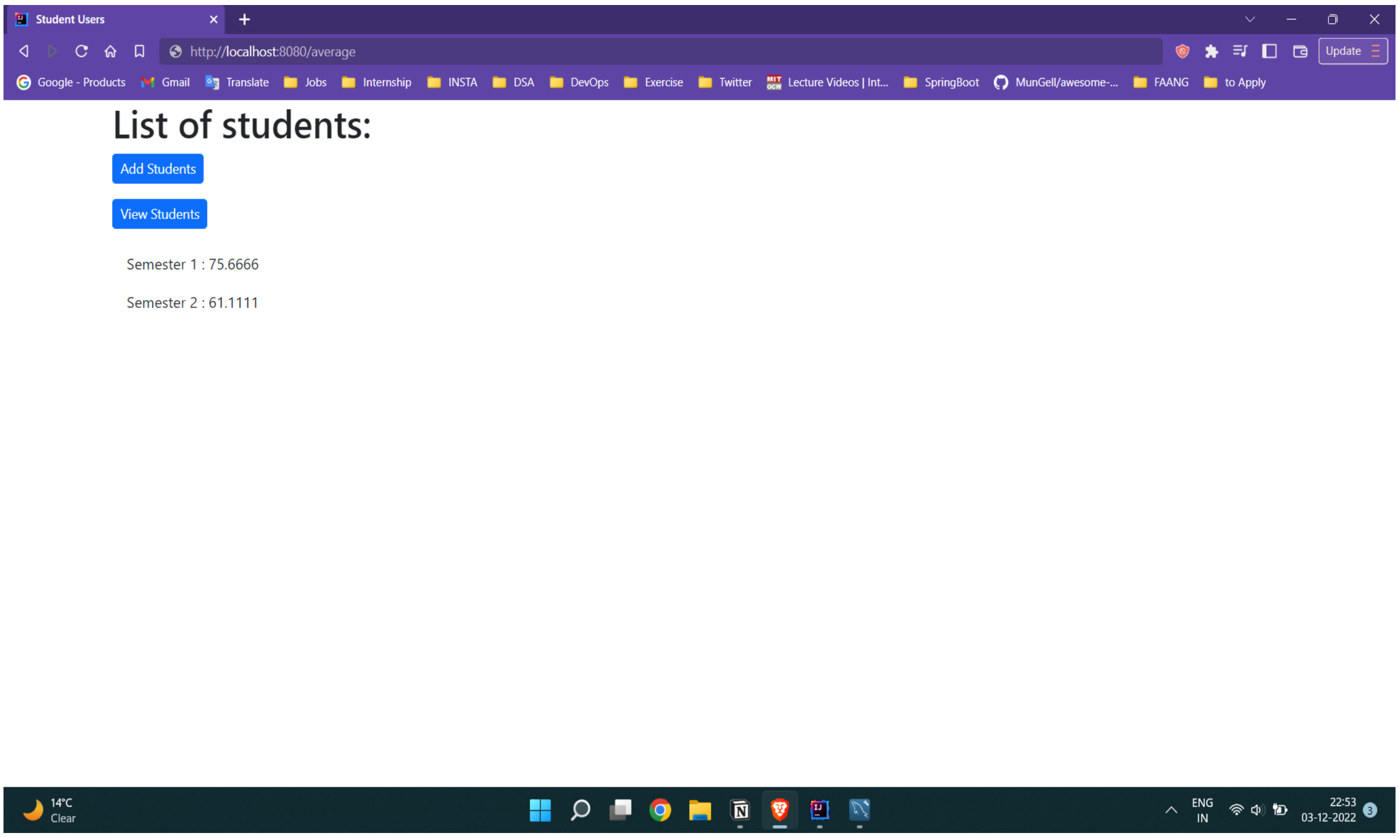
List of students:

Add Students

Average of students

Name	English Marks	Science Marks	Math Marks	Semester	Action
Akhil	75	85	95	2	Update Delete
Adarsh	70	80	90	1	Update Delete
Chintu	55	65	85	2	Update Delete
Aman	84	45	85	1	Update Delete
Rahul	40	30	20	2	Update Delete

Page dedicated to showing average of students.



http://localhost:8080/average

List of students:

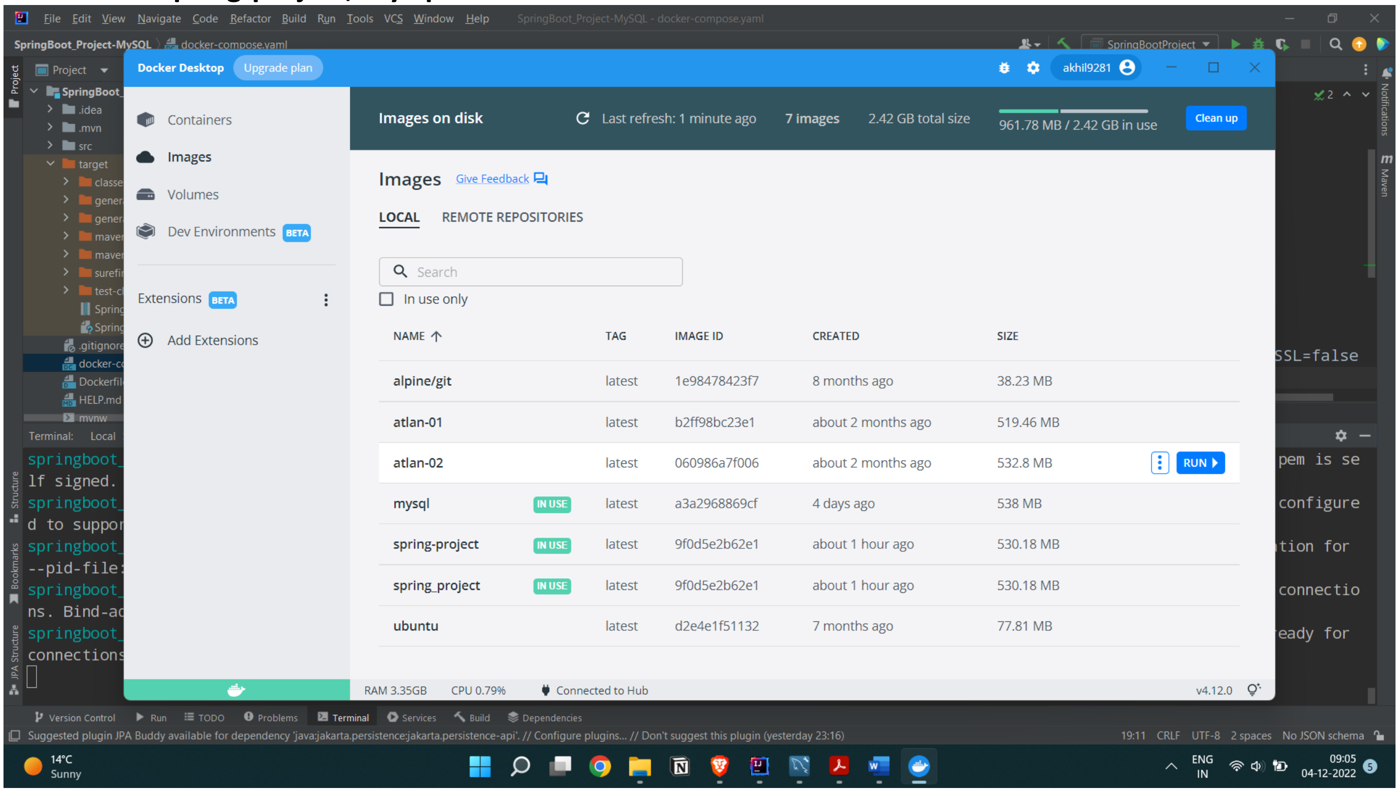
Add Students

View Students

Semester 1 : 75.6666

Semester 2 : 61.1111

Docker desktop running the application as a container:
Containers: **spring-project, mysql.**



Images on disk   Last refresh: 1 minute ago   7 images   2.42 GB total size   961.78 MB / 2.42 GB in use

Images

NAME		TAG	IMAGE ID	CREATED	SIZE
alpine/git		latest	1e98478423f7	8 months ago	38.23 MB
atlan-01		latest	b2ff98bc23e1	about 2 months ago	519.46 MB
atlan-02		latest	060986a7f006	about 2 months ago	532.8 MB
mysql	IN USE	latest	a3a2968869cf	4 days ago	538 MB
spring-project	IN USE	latest	9f0d5e2b62e1	about 1 hour ago	530.18 MB
spring_project	IN USE	latest	9f0d5e2b62e1	about 1 hour ago	530.18 MB
ubuntu		latest	d2e4e1f51132	7 months ago	77.81 MB

RAM 3.35GB   CPU 0.79%   Connected to Hub   v4.12.0

Suggested plugin JPA Buddy available for dependency 'java:jakarta.persistence:jakarta.persistence-api'. // Configure plugins... // Don't suggest this plugin (yesterday 23:16)