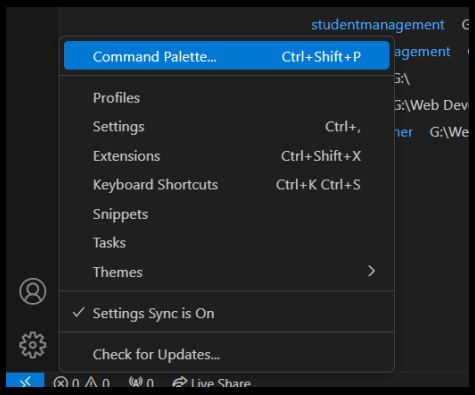
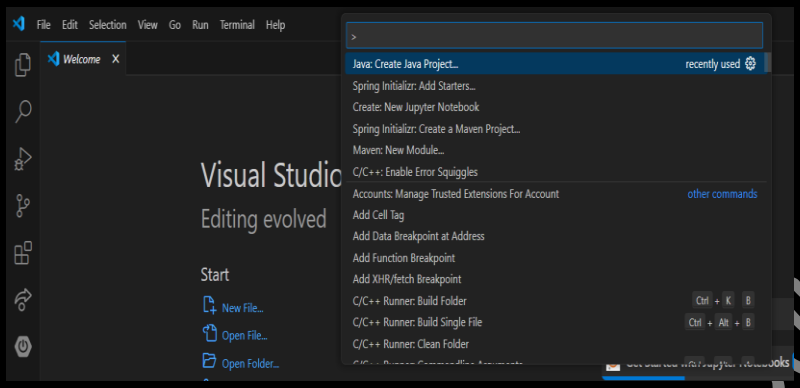
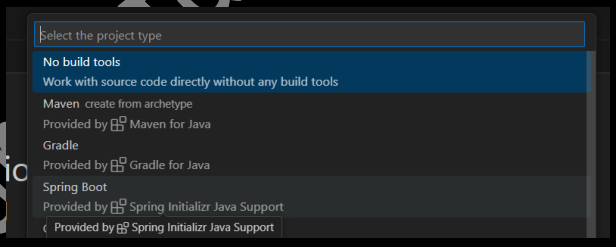
# *Project Setup*

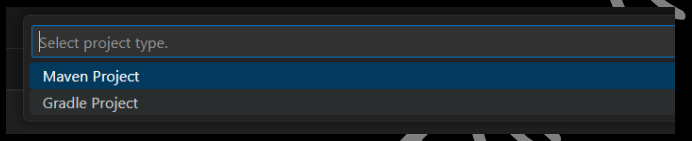
Step1: Open VS Code New Window and click Setting -> Command Paletter(Ctrl+Shift+P)

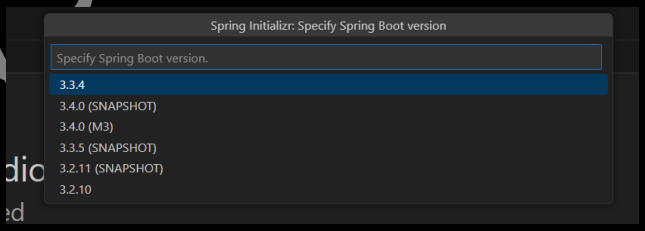


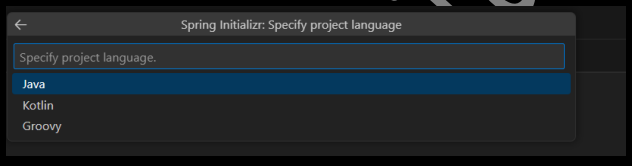

Step2: Click 'Create Java Project' -> Spring Boot -> Maven Project -> Select Version -> java



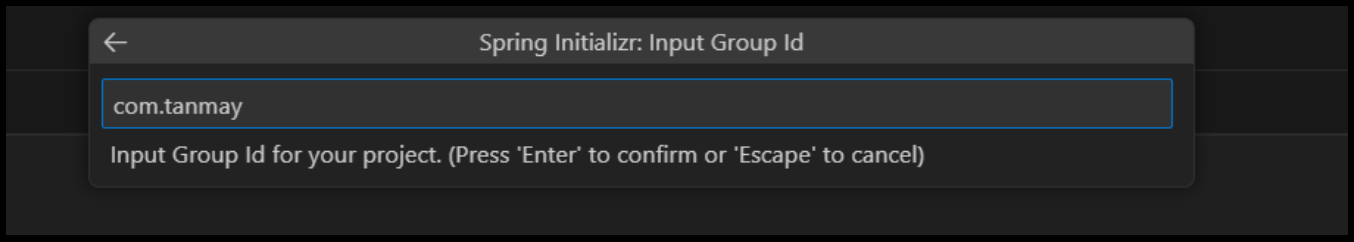



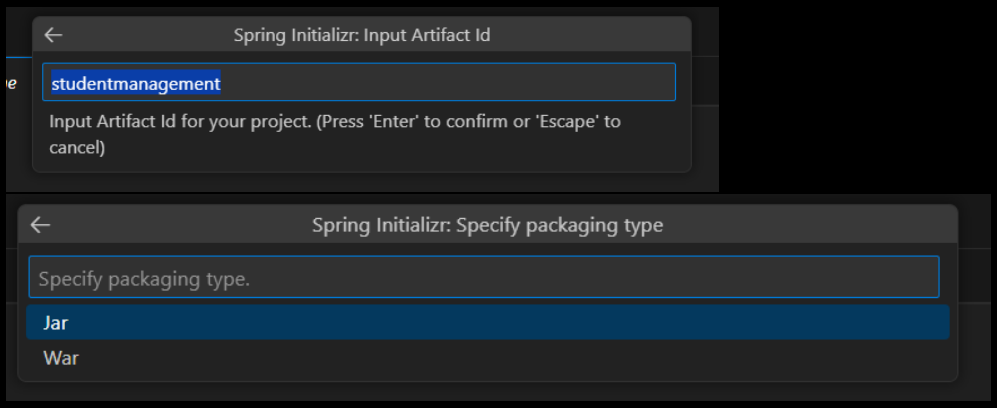



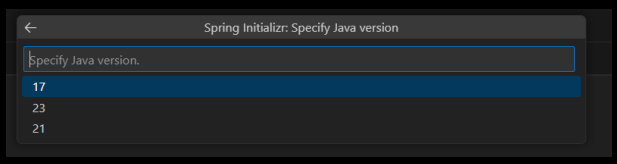

Step3: type 'Group ID' for your Project.



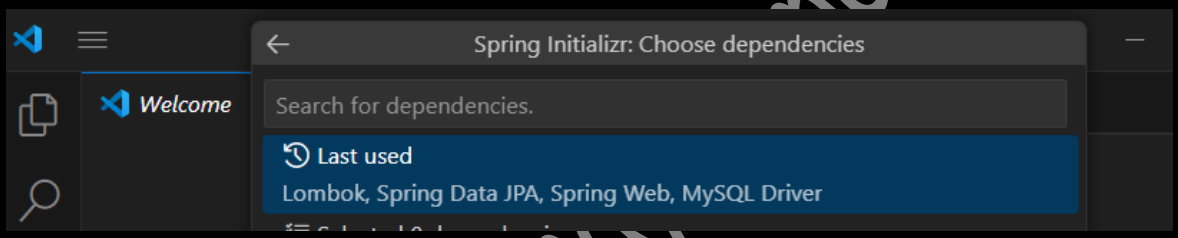
Step4: type ‘Artifact ID’ for your Project. And select ‘jar’ option.



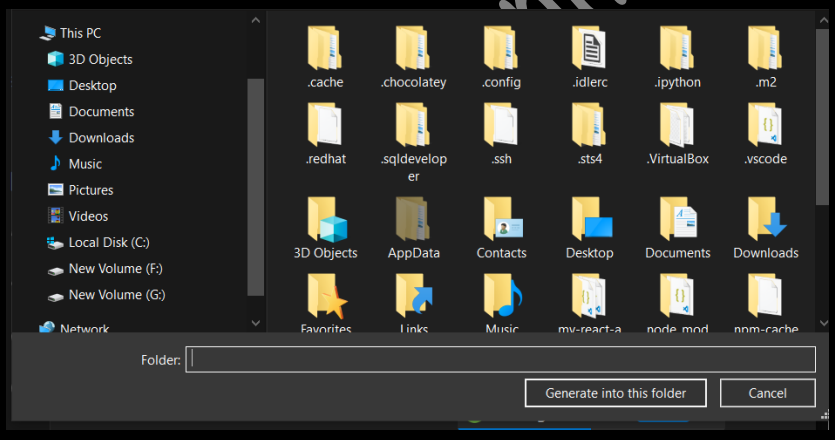

Step4: Select Java Version.



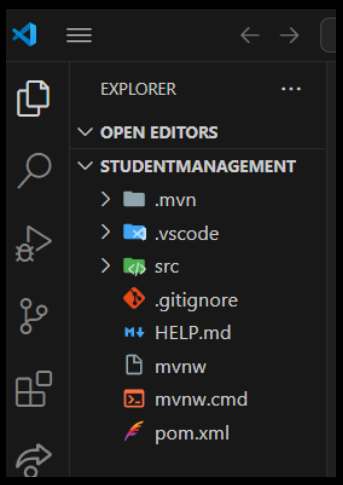

Step 5: Add Dependency. Dependencies -> Spring Web, Data JPA, Lombok, MySQL Driver



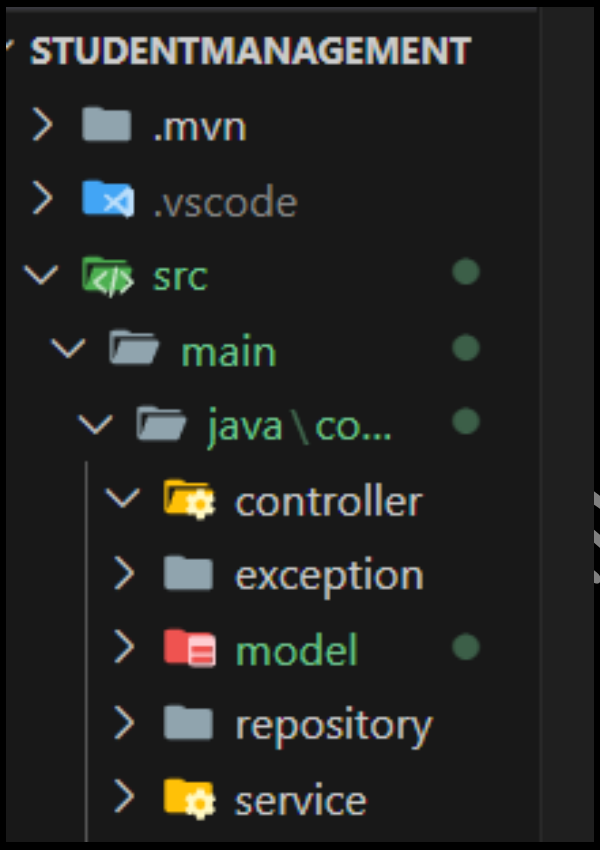

Step 6: Select Folder and Generate project into that folder.




Step 6: Open the created Spring Boot Project




**Create some Packages under 'src\main\java\com' this directory






**-> Cretae 'Student' class in model directory.**

src\main\java\com\tanmay\studentmanagement\model\Student.java

```
package com.tanmay.studentmanagement.model;


import org.hibernate.annotations.NaturalId;


import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
```

```
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;


@Entity   //This annotation indicates that the class is a JPA entity, representing a database table.
@Getter   // This annotation automatically generates getter methods for all the fields in the class.
@Setter   // This annotation automatically generates setter methods for all the fields in the class.
@AllArgsConstructor //This annotation generates a constructor that takes all the fields as arguments.
@NoArgsConstructor //This annotation generates a no-argument constructor.
public class Student {
    @Id   //used to mark a field or property in a Java class as the unique identifier of an entity. It's equivalent to a primary key in a database table.
    @GeneratedValue(strategy=GenerationType.IDENTITY)  //This nnotation specifies that the database will automatically
                                        //generate a unique value for this primary key when a new entity is created.
    private Long id;
    private String firstName;
    private String lastName;
    @NaturalId(mutable=true)   //This annotation in Spring Data JPA indicates that a field or property should be used as a natural identifier for an entity.
                          //This means that the field's value can be used to uniquely identify an entity, even if it's not the primary key.    Mutable: Indicates that the value of the field can be changed after the entity is persisted.
    private String email;
    private String department;
}
```

**-> Create 'IStudentService' interface under 'src\main\java\com\tanmay\studentmanagement\service' directory**

src\main\java\com\tanmay\studentmanagement\service\IStudentService.java

```
package com.tanmay.studentmanagement.service;
```

```
import java.util.List;


import com.tanmay.studentmanagement.model.Student;


public interface IStudentService {
   Student addStudent(Student student);
   List<Student> getStudents();
   Student updateStudent(Student student, Long id);
   Student getStudentById(Long id);
   void deleteStudent(Long id);
}
```

**-> Create Implement class 'StudentService' in service directory to implement '**IStudentService**'Interface.**

src\main\java\com\tanmay\studentmanagement\service\StudentService.java

```
package com.tanmay.studentmanagement.service;


import java.util.List;


import javax.swing.Spring;


import org.springframework.stereotype.Service;


import com.tanmay.studentmanagement.model.Student;


@Service
//The @Service annotation in Spring Boot is used to indicate that a class provides a service to the application.
//It's a component scan stereotype that tells Spring that the class is a bean and should be managed by the Spring IoC container.
//@Service annotation is a crucial tool in Spring Boot for organizing and managing application services. It promotes loose coupling,
//dependency injection, and simplifies the development process.
public class StudentService implements IStudentService{


   @Override
```

```
    public Student addStudent(Student student) {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'addStudent'");
    }

    @Override
    public List<Student> getStudents() {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'getStudents'");
    }

    @Override
    public Student updateStudent(Student student, Long id) {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'updateStudent'");
    }

    @Override
    public Student getStudentById(Long id) {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'getStudentById'");
    }

    @Override
    public void deleteStudent(Long id) {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'deleteStudent'");
    }

}
```

**-> Create Interface ‘JpaRepository’ in ‘repository’ directory and extends ‘JpaRepository’.**

src\main\java\com\tanmay\studentmanagement\repository\StudentRepository.java

```
package com.tanmay.studentmanagement.repository;


import org.springframework.data.jpa.repository.JpaRepository;


import com.tanmay.studentmanagement.model.Student;


public interface StudentRepository extends JpaRepository<Student, Long>{
}
```

**-> Let's continue to build Implement class 'StudentService'**

**src\main\java\com\tanmay\studentmanagement\service\StudentService.java**

```
package com.tanmay.studentmanagement.service;


import java.util.List;


import org.springframework.stereotype.Service;


import com.tanmay.studentmanagement.exception.StudentAlreadyExistsException;
import com.tanmay.studentmanagement.exception.StudentNotFoundException;
import com.tanmay.studentmanagement.model.Student;
import com.tanmay.studentmanagement.repository.StudentRepository;


import lombok.RequiredArgsConstructor;


@Service
//The @Service annotation in Spring Boot is used to indicate that a class provides a service to the application.
//It's a component scan stereotype that tells Spring that the class is a bean and should be managed by the Spring IoC container.
//@Service annotation is a crucial tool in Spring Boot for organizing and managing application services. It promotes loose coupling,
//dependency injection, and simplifies the development process.
@RequiredArgsConstructor   //it automatically generates a constructor for the StudentService class that takes a StudentRepository as an argument and initializes the studentRepository field with it.
public class StudentService implements IStudentService{
```

```
    private final StudentRepository studentRepository;//establishing a dependency between the StudentService class and the StudentRepository interface. It enables the StudentService
                                                      //to access and utilize the data access capabilities provided by the StudentRepository implementation, promoting loose coupling, testability, and maintainability of the code.


    @Override
    public List<Student> getStudents() {
        return studentRepository.findAll();
    }


    @Override
    public Student addStudent(Student student) {
        if (studentAlreadyExists(student.getEmail())){
            throw  new StudentAlreadyExistsException(student.getEmail()+ " already exists!");
        }
        return studentRepository.save(student);
    }


    @Override
    public Student updateStudent(Student student, Long id) {
            return studentRepository.findById(id).map(st -> {
            st.setFirstName(student.getFirstName());
            st.setLastName(student.getLastName());
            st.setEmail(student.getEmail());
            st.setDepartment(student.getDepartment());
            return studentRepository.save(st);
        }).orElseThrow(() -> new StudentNotFoundException("Sorry, this student could not be found"));
    }


    @Override
    public Student getStudentById(Long id) {
        return studentRepository.findById(id)
        .orElseThrow(() -> new StudentNotFoundException("Sorry, no student found with the Id :" +id));
    }
```

```
    @Override
    public void deleteStudent(Long id) {
        if (!studentRepository.existsById(id)){
            throw new StudentNotFoundException("Sorry, student not found");
        }
        studentRepository.deleteById(id);
    }

    private boolean studentAlreadyExists(String email) {
        return studentRepository.findByEmail(email).isPresent();
    }
}
```

**-> Create class 'StudentController' (as a RESTful controller) under controller directory & controller class responsible for handling HTTP requests:**



@RestController

**@RestController** is a Spring annotation used to define a class as a RESTful web service controller. It combines the functionality of @Controller and @ResponseBody annotations, making it easier to create RESTful APIs.

In simpler terms, it tells Spring that the class is responsible for handling requests and responses in a RESTful manner.

@RequestMapping("/students")

The annotation `@RequestMapping("/students")` in Spring MVC is used to map incoming HTTP requests to a specific Java method. In this case, it indicates that any HTTP request with the URL path "/students" will be handled by the method annotated with this annotation.

@RequiredArgsConstructor

The `@RequiredArgsConstructor` annotation in Java is used to automatically generate a constructor that initializes all required fields of a class. This means that you don't have to manually write the constructor yourself, saving you time and effort.

Here's a simple example:

Java

```java
@RequiredArgsConstructor
public class Person {
    private final String name;
    private final int age;
}
```

This will automatically generate a constructor like this:

Java

```java
public Person(String name, int age) {
    this.name = name;
    this.age = age;
}
```

Use code with caution.

As you can see, the constructor initializes both the `name` and `age` fields, ensuring that they are not null when an instance of the `Person` class is created.

```java
public class StudentController {
    private final IStudentService studentService;

    @GetMapping
    public ResponseEntity<List<Student>> getStudents(){
        return new ResponseEntity<>(studentService.getStudents(), HttpStatus.FOUND);
    }
}
```

The code snippet is a Java method that returns a list of students as a response in a Spring Boot application.

Here's a breakdown of what it does:

1. `ResponseEntity<List<Student>>` : This indicates that the method will return a response entity, which is a container for the actual response data and HTTP status code. The response data in this case is a list of `Student` objects.
2. `getStudents()` : This calls a method named `getStudents()` within the `studentService` class. This method is likely responsible for fetching the list of students from a database or other data source.
3. `new ResponseEntity<>(studentService.getStudents(), HttpStatus.FOUND)` : This creates a new `ResponseEntity` object. The first argument is the list of students obtained from the `studentService` , and the second argument is the HTTP status code, which is `HttpStatus.FOUND` in this case. This indicates that the students were found and are being returned in the response.

In summary, the method retrieves a list of students from a service, creates a response entity with the list and a "found" status code, and returns it to the caller. This could be used in a REST API endpoint to handle a GET request for a list of students.

```
@PostMapping
public Student addStudent(@RequestBody Student student){

    return studentService.addStudent(student);
}
```

**@RequestBody** is an annotation used in Spring Boot to indicate that the request body of an HTTP request should be mapped to a specific Java object. This means that the incoming data in the request body will be automatically converted into a Java object based on the structure defined in the annotated class.

In simpler terms, it helps you easily extract and process data from HTTP requests in your Spring Boot applications.

```
@PutMapping("/update/{id}")
public Student updateStudent(@RequestBody Student student, @PathVariable Long id){
    return studentService.updateStudent(student, id);
}
```

The annotation `@PutMapping("/update/{id}")` is used in Spring MVC to define a HTTP PUT request handler method. This method will be invoked when a client sends a PUT request to the specified URL, where `{id}` represents a placeholder for a variable path parameter. The method will typically update a resource with the given ID using the data provided in the request body.

The `@PathVariable Long id` annotation indicates that the `id` parameter in the controller method should be extracted from the URL path and converted to a `Long` value.

src\main\java\com\tanmay\studentmanagement\controller\StudentController.java

```
package com.tanmay.studentmanagement.controller;


import java.util.List;


import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;


import com.tanmay.studentmanagement.model.Student;
import com.tanmay.studentmanagement.service.IStudentService;


import lombok.RequiredArgsConstructor;


@RestController  //@RestController is a Spring annotation that tells Spring Boot
                 //to treat a class as a RESTful controller. This means that the
                 //methods within this class will be exposed as HTTP endpoints.
```

```
@RequestMapping("/students")  //The annotation @RequestMapping("/students") in Spring MVC is used to map incoming HTTP requests
                              //with the URL pattern "/students" to a specific method within a controller class.
                              //This means that whenever a client sends a request to the server with the URL "/students",
                              //the annotated method will be executed to handle the request.
@RequiredArgsConstructor
public class StudentController {
   private final IStudentService studentService;

   @GetMapping
   public ResponseEntity<List<Student>> getStudents(){ //method that returns a list of students as a response in a Spring Boot application.
      return new ResponseEntity<>(studentService.getStudents(), HttpStatus.FOUND);
      //This method retrieves a list of students from a service, creates a response entity with the list
      //and a "found" status code, and returns it to the caller. This could be used in a REST API endpoint
      //to handle a GET request for a list of students.
   }

   @PostMapping
   public Student addStudent(@RequestBody Student student){ //@RequestBody annotation used in Spring Boot to indicate that the request body of an HTTP request should be mapped to a specific Java object.
                                                            //This means that the incoming data in the request body will be automatically converted into a Java object based on the structure defined in the annotated class.
      return studentService.addStudent(student);
   }

   @PutMapping("/update/{id}") //The annotation @PutMapping("/update/{id}") is used in Spring MVC to define a HTTP PUT request handler method.
                               //This method will be invoked when a client sends a PUT request to the specified URL, where {id} represents
                               //a placeholder for a variable path parameter. The method will typically update a resource with the given ID
                               //using the data provided in the request body.
   public Student updateStudent(@RequestBody Student student, @PathVariable Long id){  //The @PathVariable Long id annotation indicates that the id parameter in the controller method should be extracted from the URL path and converted to a Long value.
```

```
                                                    //This makes it easier to create RESTful APIs that can handle different
resources based on their unique identifiers.
        return studentService.updateStudent(student, id);
    }

    @DeleteMapping("/delete/{id}")
    public void deleteStudent(@PathVariable Long id){
        studentService.deleteStudent(id);
    }

    @GetMapping("/student/{id}")
    public Student getStudentById(@PathVariable Long id){
        return studentService.getStudentById(id);
    }
}
```

**-> : Configure Database Connection and JPA Settings**

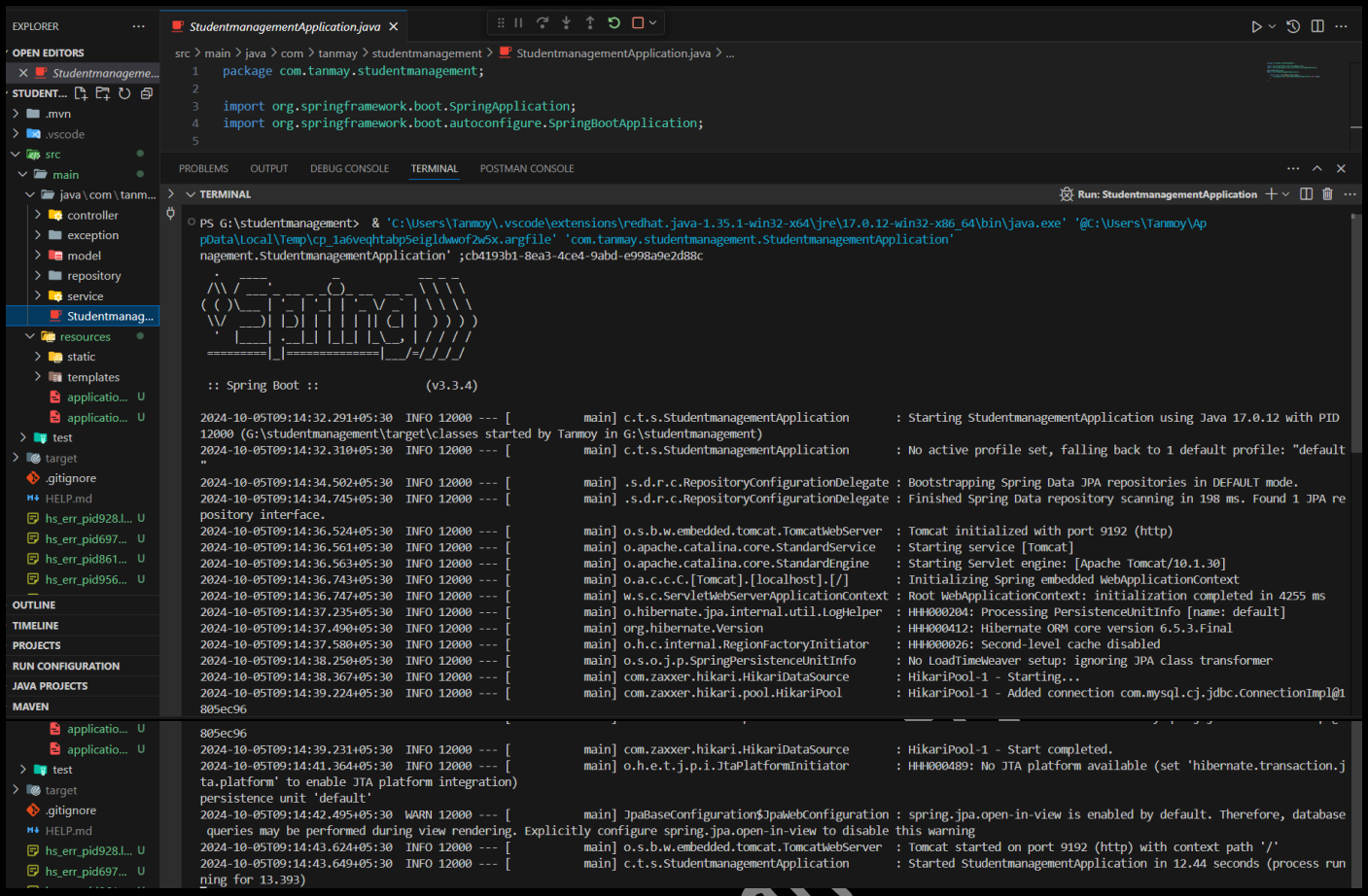
*Create ‘application.yml’ file in ‘src\main\resources\application.yml’ directory and remove ‘application.properties’ file from this directory

src\main\resources\application.yml

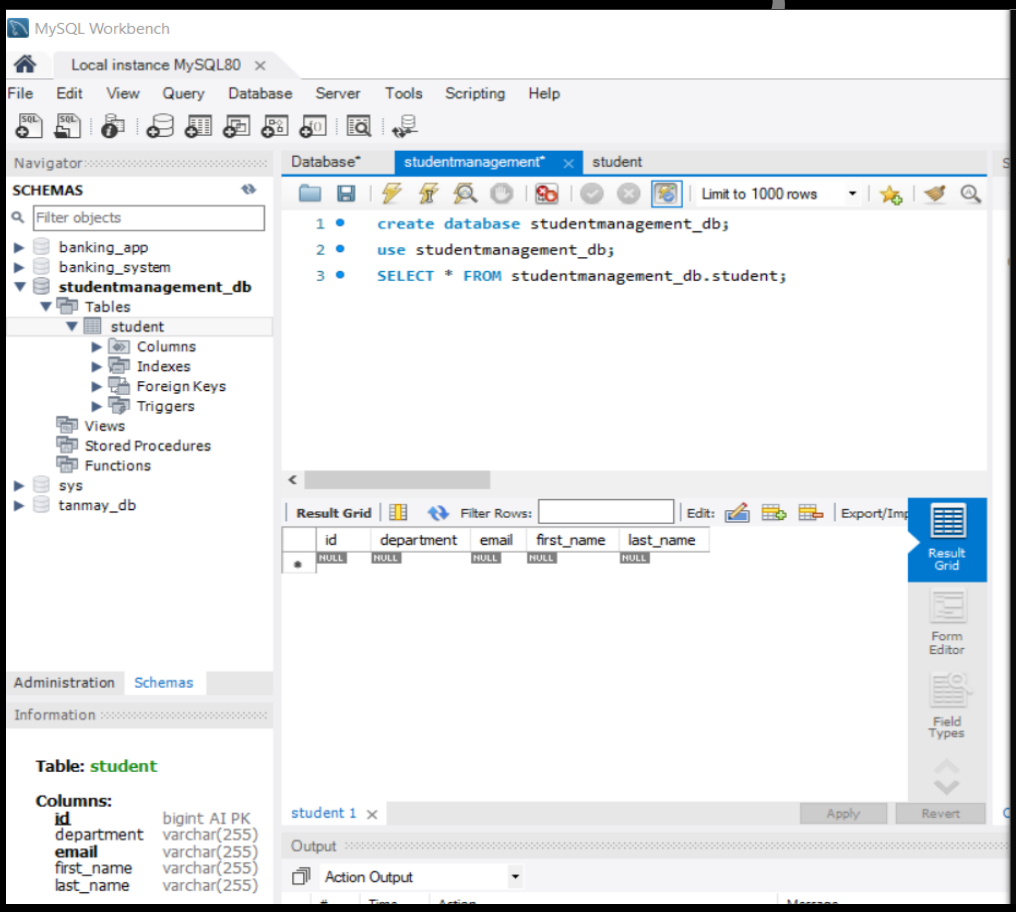
```
server:
  port: 9192


spring:
  datasource:
    username: root
    password: *********
    url: jdbc:mysql://localhost:3306/studentmanagement_db
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    show-sql: true
    hibernate:
      ddl-auto: create
      format_sql: true
```

*Now Run Project and Check whether ‘student’ table is created in ‘studentmanagement_db’ database or not.




Check whether ‘student’ table is created in ‘studentmanagement_db’ database or not.



**-> : Let's Go to 'Postman' and test our endpoints**



## Frontend Part (React JS)

**-> : Let's Start building Frontend Part (ReactJs):**

package.json:

```
{
  "name": "sbr-client",
  "version": "0.1.0",
  "private": true,
```

```
"dependencies": {
  "@testing-library/jest-dom": "^5.16.5",
  "@testing-library/react": "^13.4.0",
  "@testing-library/user-event": "^13.5.0",
  "@babel/plugin-proposal-private-property-in-object": "^7.16.0",
  "axios": "^1.4.0",
  "bootstrap": "^5.3.0",
  "react": "^18.2.0",
  "react-dom": "^18.2.0",
  "react-icons": "^4.10.1",
  "react-router-dom": "^6.14.1",
  "react-scripts": "^5.0.1",
  "web-vitals": "^2.1.4"
},
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
"eslintConfig": {
  "extends": [
    "react-app",
    "react-app/jest"
  ]
},
"browserslist": {
  "production": [
    ">0.2%",
    "not dead",
    "not op_mini all"
  ],
  "development": [
```

```
    "last 1 chrome version",
    "last 1 firefox version",
    "last 1 safari version"
  ]
 }
}
```