Spring framework

- Dependency Injection and Inversion of control
- Aspect-Oriented programming
- Data access and persistence
- WEB AND RESTful services
- Messaging and integration
- Testing and debugging

Sring Boot

- Embedded web server(Tomcat,jetty)
- Auto configuration of spring and third-party libraries
- Production-ready feature such:health checks,metrics..
- Easy Integration with popular tools:Maven,Gradle, STS
- Developer-friendly features such as hot reloading

why you should learn spring?

- Extensive support and community
- Modular and flexible architecture(promote good coding practice)
- Entreprise grade features and functionality(including data access,web and restful services, messaging and security and testing)
- Easy to learn and use

Spring framework architecture

1. Data access - JDBC, ORM, OXM, JMS, Transactions

2. WEB - Servlet, Web, Portlet, Websocket

3. AOP(modularize cross cutting concern), Aspect, Instrumentation, Messaging

4. Core container -Beans, core, context, SpEL

5. Test

ApplicationContext

ApplicationContext --> uses (@Qualifier and @Primary)

ApplicationContext --> @Autowired InMemoryStudentService and @Autowired DBStudentService both implements studentService

index.html

--. html statemtent

-- localhost:8080 (here it will print html statement)

create new repository(remote)

-- Spring_boot_With_Github

in intellij in terminal git bash or local powershell

git init

git status (here we will get which files are in staging and unstaging)

git add pom.xml or git add .

git status

git commit -m "first comment with Initializiing spring boot project"

git status

git log

git push // first time , no configured push destination, we dont connect remote yet

git remote add origin https://github.com/Subhashchandra-Birajdar/Spring_boot_With_Github.git

git push -u origin

// add password here

****

Bean ---> create using --> @Bean, @Component, @Service, @Repository

Bean --> to --> ApplicationContext -----dependency injection(StudentService)--->StudentController(this uses StudentService)

```
package com.SB_SpringBoot_with_sb.controller;

import com.SB_SpringBoot_with_sb.model.Student;

import com.SB_SpringBoot_with_sb.service.StudentService;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import java.time.LocalDate;

import java.util.List;

@RestController

@RequestMapping("/api/v1/students")

public class StudentController {

 @GetMapping      // http://localhost:8080/api/v1/students

 public List<String> findAllStudents(){ // add commit first rest web service

    return List.of(

      "Subhash",

      "Birajdar"

    );

 }

}
```

//Student model class code

```
package com.SB_SpringBoot_with_sb.model

import java.time.LocalDate;
```

```
public class Student {
   private String firstName;
   private String lastName;
   private LocalDate dateOfBirth;
   private String email;
   private int age;
//getter and setter, parameterize constructor, NoargsConstructor
}
```

```
// studentServiceClass
package com.SB_SpringBoot_with_sb.service;
import com.SB_SpringBoot_with_sb.model.Student;
import org.springframework.stereotype.Service;
import java.time.LocalDate;
import java.util.List;

@Service
public class StudentService {
   public List<Student> findAllStudentThroughModelClass(){
      return List.of(
            new Student(
                  "shubham",
                  "Patil",
                  LocalDate.now(),
                  "spatil@gmail.com",
                  21
```

```
            ) ,
            new Student(
                  "raj",
                  "Patil",
                  LocalDate.now(),
                  "rajpatil@gmail.com",
                  29
            ),
            new Student(
                  "vinayak",
                  "biradar",
                  LocalDate.now(),
                  "vinayak@gmail.com",
                  29
            )
      );
   }
}
```

--- commit -- implementing service layer ----------

==========

it is always recommended that application is close to the modification and open to the extension

recomemded to use interface and implement in the class

```
package com.SB_SpringBoot_with_sb.service;
import com.SB_SpringBoot_with_sb.model.Student;
import java.util.List;
```

```
public interface StudentServiceInterface {
   List<Student> findAllStudentThroughM           odelClass();
}


package com.SB_SpringBoot_with_sb;


import com.SB_SpringBoot_with_sb.model.Student;
import com.SB_SpringBoot_with_sb.service.StudentServiceInterface;
import org.springframework.stereotype.Repository;
import java.time.LocalDate;
import java.util.List;


@Repository
public class InMemoryStudentService implements StudentServiceInterface {


   public List<Student> findAllStudentThroughModelClass(){
      return List.of(
            new Student(
                  "shubham",
                  "Patil",
                  LocalDate.now(),
                  "spatil@gmail.com",
                  21
            ) ,
            new Student(
                  "raj",
                  "Patil",
```

```
                LocalDate.now(),
                "rajpatil@gmail.com",
                29
            ),
            new Student(
                "vinayak",
                "biradar",
                LocalDate.now(),
                "vinayak@gmail.com",
                29
            )
        );
    }
}
```

in Controller

```
private final StudentServiceInterface studentService;
```

---

interface update

```
package com.SB_SpringBoot_with_sb.service;
import com.SB_SpringBoot_with_sb.model.Student;
import java.util.List;
public interface StudentServiceInterface {
    Student save(Student s);
    List<Student> findAllStudentThroughModelClass();
```

```
    Student findByEmail(String email);
    void delete(String email);
}
```

studentrepository dao

```
package com.SB_SpringBoot_with_sb.repository;
import com.SB_SpringBoot_with_sb.model.Student;
import org.springframework.stereotype.Repository;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.IntStream;

@Repository
public class InMemoryDao {
     private List<Student> students = new ArrayList<>();
    public Student save(Student s){
        students.add(s);
        return s;
    }

    public Student findByEmail(String email){
        return students.stream()
                .filter(s->email.equals(s.getEmail()))
                .findFirst()
                .orElse(null);
```

```
    }

    public void delete(String email){
        var student = findByEmail(email);
        if(student!=null){
            students.remove(student);
        }
    }

    public Student update(Student s){
        var studentIndex = IntStream.range(0,students.size())
                .filter(index->students.get(index).getEmail().equals(s.getEmail()))
                .findFirst()
                .orElse(-1);
        if(studentIndex>-1){
            students.set(studentIndex,s);
            return s;
        }
        return null;
    }
}
```

inject bean in InmemoryStudentService

```
package com.SB_SpringBoot_with_sb;

import com.SB_SpringBoot_with_sb.model.Student;
```

```
import com.SB_SpringBoot_with_sb.repository.InMemoryDao;
import com.SB_SpringBoot_with_sb.service.StudentServiceInterface;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class InMemoryStudentService implements StudentServiceInterface {
   private final InMemoryDao  inMemoryDao;
   public InMemoryStudentService(InMemoryDao inMemoryDao) {
      this.inMemoryDao = inMemoryDao;
   }

   // save the student
   @Override
   public Student save(Student s) {
      return inMemoryDao.save(s);
   }

   // get all students
   public List<Student> findAllStudentThroughModelClass(){
      return  inMemoryDao.findAllStudent();
   }

   @Override
   public Student findByEmail(String email) {
      return inMemoryDao.findByEmail(email);
   }
```

```
    @Override
    public void delete(String email) {
        inMemoryDao.delete(email);
    }
}
```

---

changes in controller save,update,delete

========

```
package com.SB_SpringBoot_with_sb.controller;

import com.SB_SpringBoot_with_sb.model.Student;
import com.SB_SpringBoot_with_sb.service.StudentServiceInterface;
import org.springframework.web.bind.annotation.*;
import java.time.LocalDate;
import java.util.List;

@RestController
@RequestMapping("/api/v1/students")
public class StudentController {
    // injecting the other class in constroller class
    // this is loosely coupling, managed by spring boot
    // private final StudentService studentService;
    private final StudentServiceInterface studentService;

    public StudentController(StudentServiceInterface studentService) {
```

```
        this.studentService = studentService;
    }

    // this is tightly coupling,hard to manage and maintain the code
    //private final StudentService studentService1=new StudentService();

    // constructor injection
//    public StudentController(StudentService studentService) {
//        this.studentService = studentService;
//    }


    @GetMapping      // http://localhost:8080/api/v1/students
 public List<String> findAllStudents(){
     return List.of(
       "Subhash",
       "Birajdar"
     );
 }

 // now these logic we use in service layer
@GetMapping("/getAll") // http://localhost:8080/api/v1/students/getAll
 public List<Student> findAllStudent_ThroughModelClass(){
     return List.of(
       new Student(
             "shubham",
             "Patil",
```

```
            LocalDate.now(),
            "spatil@gmail.com",
            21
        ) ,
            new Student(
                "raj",
                "Patil",
                LocalDate.now(),
                "rajpatil@gmail.com",
                29
            ),
            new Student(
                "vinayak",
                "biradar",
                LocalDate.now(),
                "vinayak@gmail.com",
                29
            )
    );
}
    @GetMapping("/getAllStudents") // http://localhost:8080/api/v1/students/getAllStudents
    public List<Student> findAllStudentsUseImlementation() {
        return studentService.findAllStudentThroughModelClass();
    }

    // save the student
    @PostMapping // http://localhost:8080/api/v1/students
```

```
    public Student save(@RequestBody Student studentdata){
        return studentService.save(studentdata);
    }

    //get the student by email
    @GetMapping("/{email}") // http://localhost:8080/api/v1/students/email
    public Student findByEmail(@PathVariable("email") String email){
        return studentService.findByEmail(email);
    }

    //update the student
    @PutMapping // http:///localhost:8080/api/v1/students
    public Student updateStudent(@RequestBody Student student){ // @RequestBody serialise the json data
        return studentService.update(student);
    }
    @DeleteMapping("/{email}") // http://localhost:8080/api/v1/students/email
    public void delete(@PathVariable("email") String email){
        studentService.delete(email);
    }
}
```

Open postman

==========>

Post : http://localhost:8080/api/v1/students

```
{
    "firstName":"subhash",
    "lastName":"birajdar",
```

```
    "email":"subha@gmail.com",
    "age":100
}
```

output: response

```
{
    "firstName": "subhash",
    "lastName": "birajdar",
    "dateOfBirth": null,
    "email": "subha@gmail.com",
    "age": 100
}
```

//getall

Get : http://localhost:8080/api/v1/students/getAllStudents

```
[
    {
        "firstName": "subhash",
        "lastName": "birajdar",
        "dateOfBirth": null,
        "email": "subha@gmail.com",
        "age": 100
    }
]
```

//getone

Get : http://localhost:8080/api/v1/students/subha@gmail.com

```
{
   "firstName": "subhash",
   "lastName": "birajdar",
   "dateOfBirth": null,
   "email": "subha@gmail.com",
   "age": 100
}
```

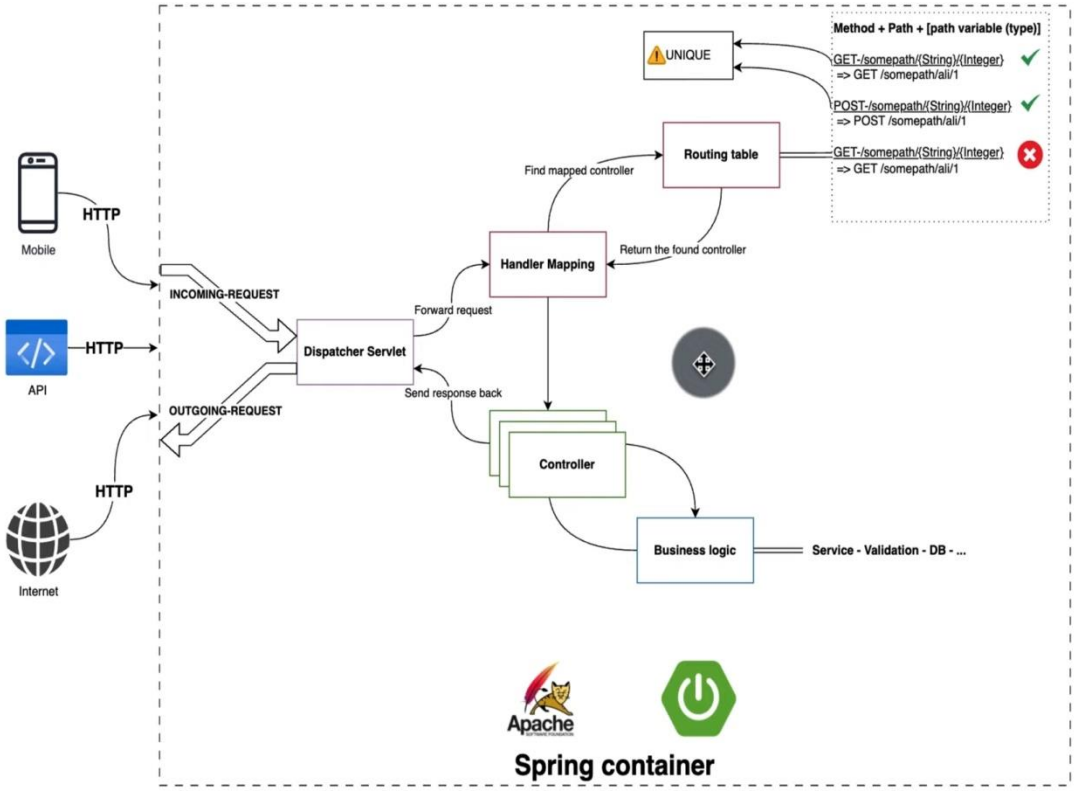
//update student

put : http:///localhost:8080/api/v1/students

//delete

//update not return updated record

Internally working of spring boot



Here Our API can be consumed by a mobile API or Web application so previous test we used postman

In Web browser so when you click on the send button **postman** this will create Incoming request to our system and then the incoming request will go to the **Apache Tomcat** which is where we have our spring boot container running and the request will be intercepted or like the first thing that will receive this request is the **dispatcher servlet** and the dispatcher servlet and then it will forward this request to an **interface** or to an **object** or spring it will forward it to a bean of type **Handler mapping** and this Handler mapping will use to find the mapped controller it will be stored in a **routing table** so here I just want to highlight something routing table I not a technical word or it's not technical implementation from spring but for me to make it easier for you I just called it routing table and this routing table .It contains list of the URLs and also the controller or the method to be invoked and the method is composed of controller name dot method name and now let me show you how this routing table is composed(ambiguity) and this already happening when the application starts. It means same http method urls.

We have the path so the path is that you have on the controller the plus path you have on method so this will have parameters and of course type here matters. If you have a path like this, if you have a

@GetMapping("/postsome/string/{integer}")

Exa : @Getmapping("subhash/birajdar/1")

But within same class different method on postsome path

@PostMapping("/postsome/String/{Integer}")

This is also correct but if you have the same thing a get some path and here

Exa : @Getmapping("subhash/birajdar/1") --- we will get error

Here for string we human we pas email but spring it understand the types so a string email or a string username or string firstname for string so there is no difference so if you have the same thing right here so this one is that composition should be unique and remind you finally composition means method + path , the path variable or the variable you have within your url

Then once the path or the fetched path is found so then it will return to this handler mapping and then return to the found controller and from handler spring will invoke the method as based in the controller name dot method name and then of course it will go through all the layers of our application , business logic, validation database and so and so forth. And we will send back the response and the response will return to the user or the final user as an outgoing request so generally and way of the code this is how spring is dispatching request from the client to backend to a specific controller or a method and so forth.

Now, add a database to our application and then communicate within this database.

Now, in order to connect to a database first of all we need to configure our project and add some dependencies. First if all you know that spring boot relies and uses spring starters so we have a starters for that which is the **spring boot starter data jpa** so this one will give us necessary classes and configurations in order to connect to a database and manipulate a database and then to connect a database you need to have a driver for that database type for example Mysql/postgres database then you need to add the dependency the mysql dependency

Springboot starter-data-jpa here you don't need to specify the version because it will inherit it from the spring boot parent

Add Mysql driver in pom.xml

Update the project and maven update or reload project.

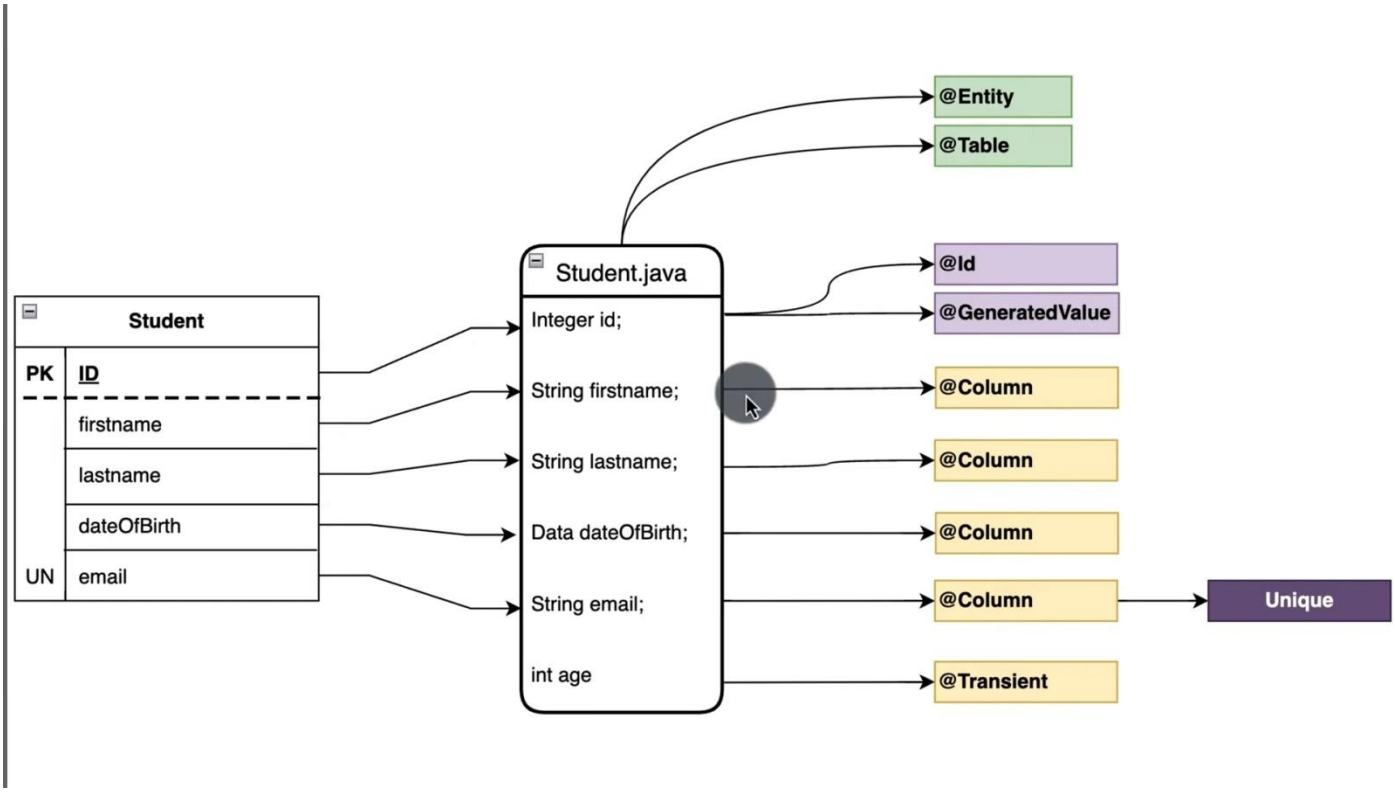
Changes in application.properties

```
spring.application.name=SpringBoot_with_sb

spring.datasource.url=jdbc:mysql://localhost:3306/student_db
spring.datasource.username=root
spring.datasource.password=sb@12345

#create a table(create,update,create-drop(means create and drop with session
end),validate,none)
spring.jpa.hibernate.ddl-auto=update
#console sql query, by default it is false
spring.jpa.show-sql=true
spring.jpa.hibernate.dialect =org.hibernate.dialect.MySQL5InnoDbDialect
spring.jpa.properties.hibernate.format_sql=true
```

in mysql database create like **student_db**

start the spring boot application

Now Transform student.java to student table in mysql database



Be able to manage and manipulate this object and process it to the database it from the database.

Here field name === row name

I want to ignore the age and I don't want to create in table for that I used @Transient

Now we will create entity

```
package com.SB_SpringBoot_with_sb.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.LocalDate;

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@Table(name="tb_student")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    private String firstname;

    private String lastname;

    private LocalDate dateOfBirth;

    private String email;

    @Transient // here age is not create column in database table, not persist
    private int age;

    //calcualte age
    /*  here i will get how old this student is
    public int  getAge(){
    return period.between(dateofBirth,LocalDate.now()).get();
    }
     */

}
```

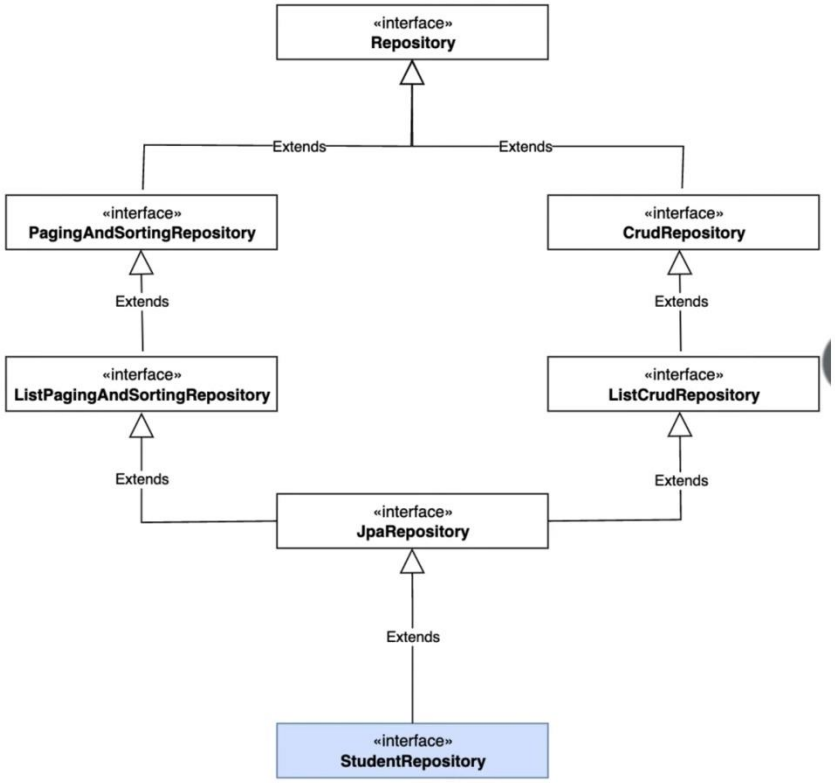
now table is created.

If we extend Jpa repository you will have all of the repository within this interface.

All of the repository(these) they extend from the mother interface called **repository.**

```
package com.SB_SpringBoot_with_sb.repository;

import com.SB_SpringBoot_with_sb.entity.Student;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
```

```java
@Repository
public interface StudentRepository extends JpaRepository<Student,Integer> {
    // Jparepository that this is a repository and it will automatically inject the
bean,
    Student findbyEmail(String email);
    void deleteByEmail(String email);

}
```



In repository, spring data jpa so with we can use the query DSL or like general the generate query so this means that spring data jpa will understand from the method name which query generate behind so I want to find by and then all I need is to specify the field name with a capital letter.

InMemoryStudent → persist into memory

DbStudent→persist in database.

```java
package com.SB_SpringBoot_with_sb.service;

import com.SB_SpringBoot_with_sb.entity.Student;
import com.SB_SpringBoot_with_sb.repository.StudentRepository;
import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@Primary
```

```java
public class DBStudentService implements StudentServiceInterface {

    private final StudentRepository studentRepository;

    public DBStudentService(StudentRepository studentRepository) {
        this.studentRepository = studentRepository;
    }

    @Override
    public Student save(Student s) {
        return studentRepository.save(s);
    }

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

    @Override
    public Student findByEmail(String email) {
        return studentRepository.findbyEmail(email);
    }

    @Override
    public Student update(Student s) {
        return studentRepository.save(s);
    }

    @Override
    public void delete(String email) {
        studentRepository.deleteByEmail(email);
    }
}
```

controller

```java
package com.SB_SpringBoot_with_sb.controller;

import com.SB_SpringBoot_with_sb.model.Student;
import com.SB_SpringBoot_with_sb.service.StudentServiceInterface;
import org.springframework.web.bind.annotation.*;

import java.time.LocalDate;
import java.util.List;

@RestController
@RequestMapping("/api/v1/students")
public class StudentController {

    // injecting the other class in constroller class
    // this is loosely coupling, managed by spring boot
    // private final StudentService studentService;
    private final StudentServiceInterface studentService;

    public StudentController(StudentServiceInterface studentService) {
        this.studentService = studentService;
    }
```

```java
    // this is tightly coupling,hard to manage and maintain the code
    //private final StudentService studentService1=new StudentService();

    // constructor injection
//    public StudentController(StudentService studentService) {
//        this.studentService = studentService;
//    }


    @GetMapping      // http://localhost:8080/api/v1/students
 public List<String> findAllStudents(){
     return List.of(
       "Subhash",
       "Birajdar"
     );
 }

 // now these logic we use in service layer
@GetMapping("/getAll") // http://localhost:8080/api/v1/students/getAll
 public List<Student> findAllStudent_ThroughModelClass(){
     return List.of(
       new Student(
               "shubham",
               "Patil",
               LocalDate.now(),
               "spatil@gmail.com",
               21
       ) ,
             new Student(
                     "raj",
                     "Patil",
                     LocalDate.now(),
                     "rajpatil@gmail.com",
                     29
             ),
             new Student(
                     "vinayak",
                     "biradar",
                     LocalDate.now(),
                     "vinayak@gmail.com",
                     29
             )
     );
 }
    @GetMapping("/getAllStudents") //
http://localhost:8080/api/v1/students/getAllStudents
    public List<Student> findAllStudentsUseImlementation() {
        return studentService.findAllStudentThroughModelClass();
    }

    // save the student
    @PostMapping // http://localhost:8080/api/v1/students
    public Student save(@RequestBody Student studentdata){
        return studentService.save(studentdata);
    }

    //get the student by email
    @GetMapping("/{email}") // http://localhost:8080/api/v1/students/email
    public Student findByEmail(@PathVariable("email") String email){
```

```
        return studentService.findByEmail(email);
    }

    //update the student
    @PutMapping // http:///localhost:8080/api/v1/students
    public Student updateStudent(@RequestBody Student student){ // @RequestBody
serialise the json data
        return studentService.update(student);
    }
    @DeleteMapping("/{email}") // http://localhost:8080/api/v1/students/email
    public void delete(@PathVariable("email") String email){
        studentService.delete(email);
    }
}
```

---

here the interface is implementing in two places so we will get error in controller

we have to resolve this using @Qualifier annotation through or @Primary annotation through

There is more than one bean of student type beans : **DBStudentServiceImpl,InMemoryStudent**

first way we will apply to @Qualifier on controller class, depends on how you are injecting the bean in controller like filed injection or constructor injection

field injection through

@Autowired

@Qualifier(“DBStudentServiceImpl”) //spring by default use class name as bean name

private StudentService studentService;

constructor injection through

public StudentController(**@Qualifier(“InMemoryStudent”)**StudentService studentService){

this.studentService = studentService;

}

Another option is using @Primary

@Primary

@Service

Public class DBStudentServiceImpl implement StudentServiceInterface{ …}

“ dateOfBirth” :”1998-04-04T00:00:00Z”