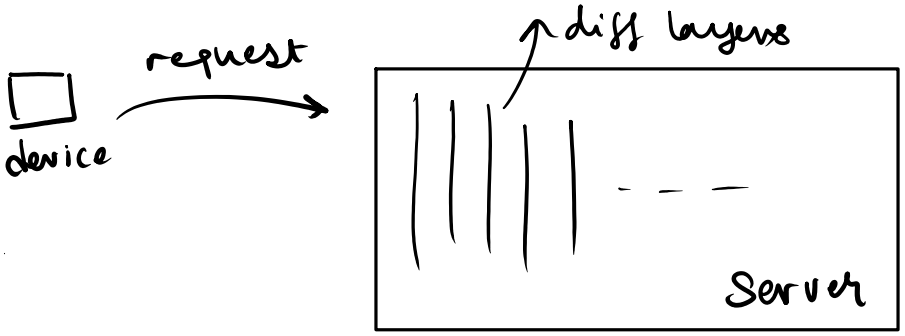
19 January 2023 12:38

. This request will go through various layers in server.



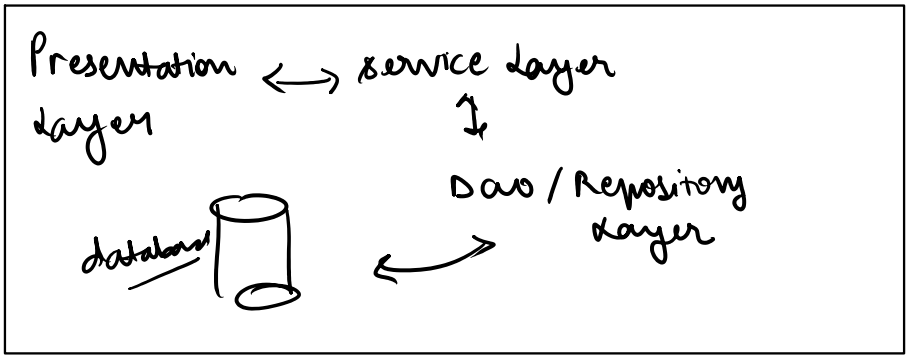
↳ ① Presentation Layers {Controllers}
↳ they accept the request.

② Service Layers.
(providing Service)
(Bussiness Logic execution happens here).

③ Dao / Repository layer

④ Database.

J2EE Backend architecture



Controller: waiter : accepts the food.
Service : waiter gives order to chef : ∴ chef
Dao : If chef wants some ingridient such as flour.
∴ Dao → provides it.

API URLs

example app's requirement.

These URLs start with base URL.

METHOD	API URL Example	Operation	
GET	/courses	Get all courses	✓
GET	/courses/{courseID}	Get Single course of given id in URL	✓

GET	/courses	Get all courses
GET	/courses/{courseID}	Get Single course of given id in URL
POST	/courses	Add new course
PUT	/courses	Update the course
DELETE	/courses/{courseid}	Delete the course Id

Things we require

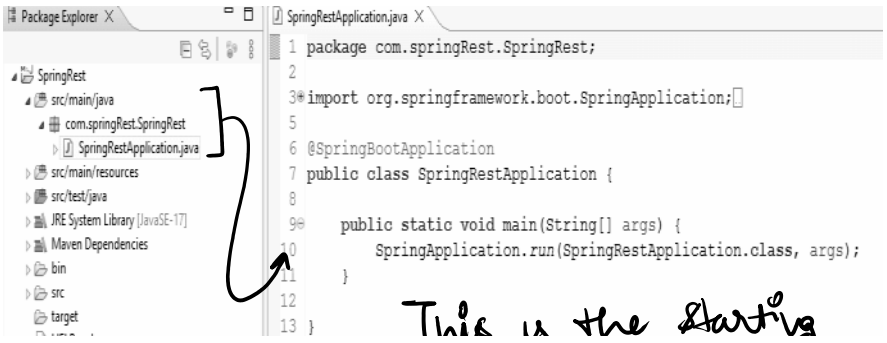
↳ IDE

↳ Postman. (Postman will generate requests)

we don't have client run to generate requests. ∴ we will use Postman to do so.

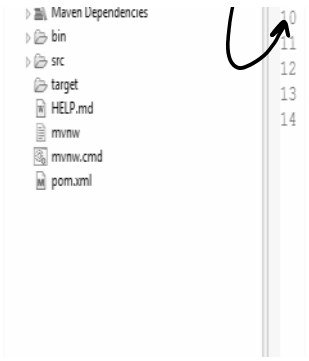
STEPS :

1) Download Postman
2) Via Spring Initializr (start.spring.io) create a template for your project
3) Group Id : Identifies our project
4) Dependencies : Web , Postgre , SQL Data JPA
5) You will get a zip file … extract it
6) Open it from Ecllipse / any other ide
7) File >> Import >> Maven >> Existing Maven Project >> choose the path
8) Download Postman



This is the starting

The configuration/ settings of the application is done in




```
10        SpringApplication.run(SpringRestApplication.class, args);
11    }
12
13 }
14
```

This is the starting point of our appⁿ.

done in application.properties.

src/main/resources

POM.xml : Heart of the appⁿ... contains all the dependancies of your project.

For time being, remove the {jpa} dependancy from the pom file -- else it will create error while running.
[Cut and Paste it somewhere]

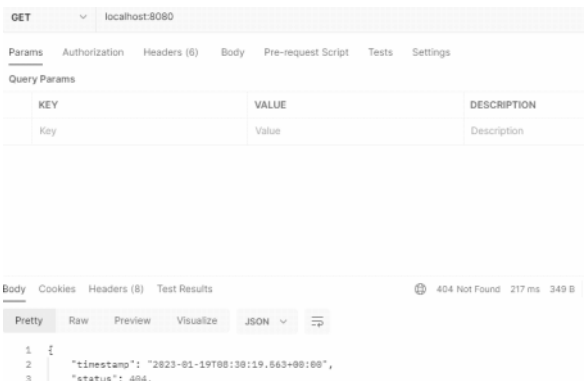
Spring Boot → Inbuilt → Tomcat Server.

When you run the project.... in your console you will see this.

```
main] c.s.SpringRest.SpringRestApplication     : Starting SpringRestApplication using Java 17.0.2 with PID 8208 (C:\User
main] c.s.SpringRest.SpringRestApplication     : No active profile set, falling back to 1 default profile: "default"
main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.4]
main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1635 ms
main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
main] c.s.SpringRest.SpringRestApplication     : Started SpringRestApplication in 3.093 seconds (process running for 3.5
```

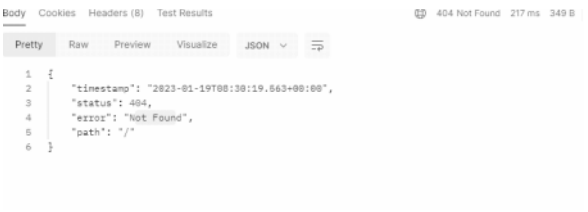
gets started on the tomcat server. Port → 8080

To check you can go to postman and check localhost:8080
Initially it will display NOT FOUND.



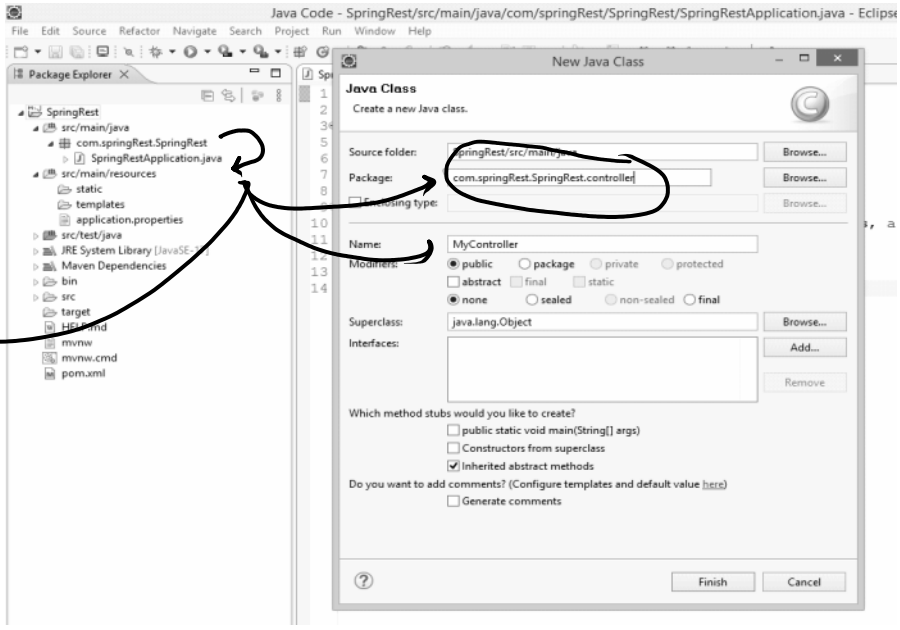
The response is given by Spring Appⁿ.
Since we haven't formed

Since we haven't formed any controller --- so there is no one to recieve the request. (ref - layers in J2EE Archi above)



To form controllers :

1) In the main package … form a new class and name it
2) It should go in main package >> Sub package.



But how we will tell spring boot that this is my controller How will we tell. Ki this will be the first layer to handle my requests?

↳ For this we have annonation @Controller. via this spring will understand that its duty is in the presentation layer.

Since we are creating REST API, therefore a REST Controller.

REST= Representational State Transfer.
Here data is sent in form of JSON (mostly). Javascript object Notation.

{Order yeh Kega}

Now we will create mapping. like for /home /profile etc.

For this we will use annonation

`@GetMapping("/name")`

- GET : Method — "/home"
- Mapping : Map — "/profile"

} where the request is fired.

As soon as you fire home--- this fun will catch it.

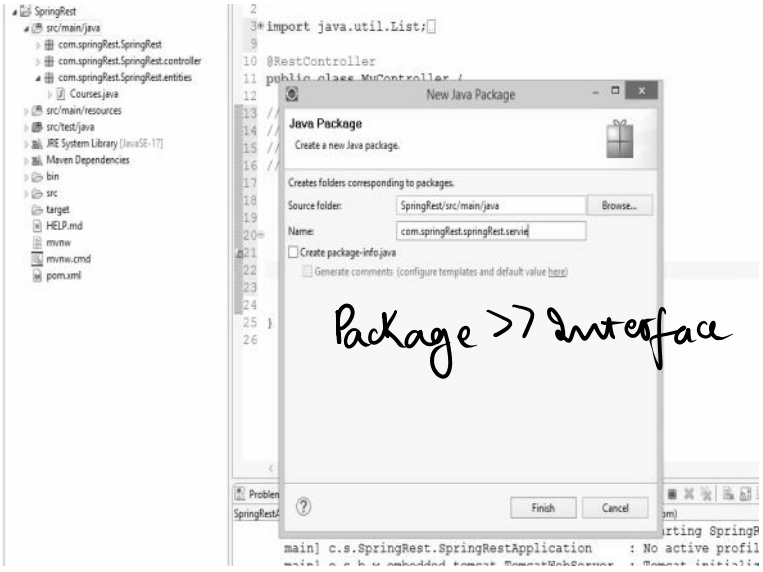
Creating entities and getting the data.

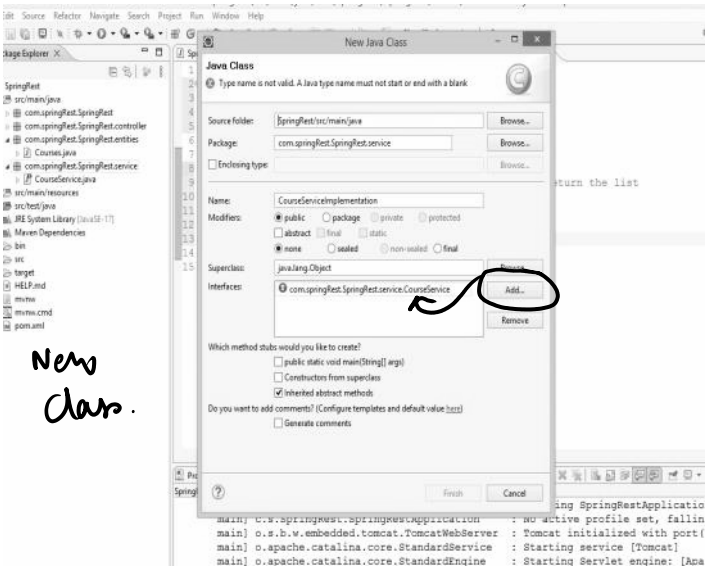
Now when the GET courses request is fired up … we would need the list of courses so …. We will return List<Courses> from our method……but we haven't defined this entity named as Courses ….. So firslty under the src/main/java….we will create a new package and inside that package we will create a class named as Courses….now in this we will define our things.
After defining those…..we will go to Source tab >> Generate Getters and Setters for all >> Generate constructors >> Generate to string……

Once it is defined…now we will create our things in the controller file

Once i have created the fun--- the presentation layer (controller) would ask the data from the service layer. [waiter → chef]

For that we would need another package in the src/main/ Java. And in that we will create a new class / Interface.
And then we will Implement that Interface.





Now once we create our service layer interface….for loose coupling we will do this :

1) In the Control Service Implementation …. We will initially create a hardcoded list and we will return that.

2)

```
package com.springRest.SpringRest.service;
import java.util.ArrayList;
import java.util.List;

import com.springRest.SpringRest.entities.Courses;

public class CourseServiceImplementation implements CourseService {

    //Storing the list temporary here
    //In future we will connect this from data-base
    List<Courses> list;

    public CourseServiceImplementation() {
        list = new ArrayList<>();
        list.add(new Courses(145, "Maths " , "Mathematic Course"));
        list.add(new Courses(146, "Science" , "Science Courses"));
        list.add(new Courses(147, "English" , "Englishh Course"));

    }


    @Override
    public List<Courses> getCourses() {
        // TODO Auto-generated method stub
        return list;
    }
```

2) After we do so we will go to our controller file and in that we will create
a variable of interface…why to call different functions for different API calls.

But we have to connect the variable to object of interface…..Auto wiring….so for that we will use auto wired annotation

```
@RestController
public class MyController {

//  @GetMapping("/home")
//  public String home() {
//      return "This is Home Page";
//  }

    //Get the courses

    //Interface serive ka variable
    @Autowired
    private CourseService csvariable;


    @GetMapping("/courses")
    public List<Courses> getCourses(){

        return this.csvariable.getCourses();

    }
}
```

.

And then we will return the list by calling the function in the interface

This is how our interface file looks

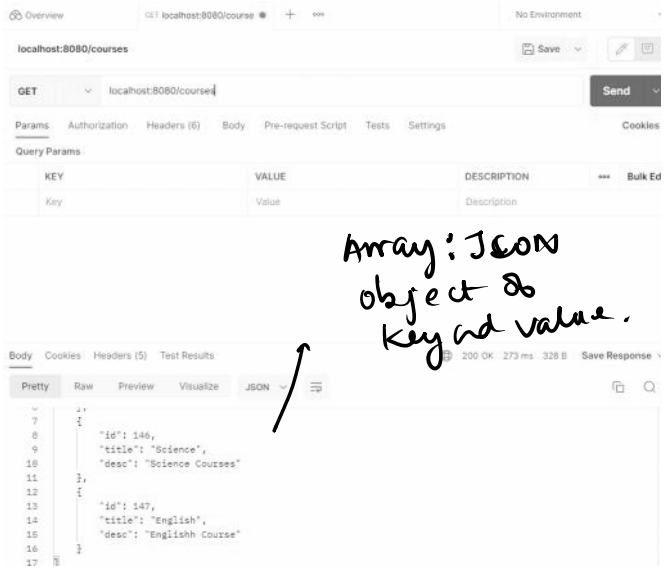
```
package com.springRest.SpringRest.service;
import java.util.List;
import com.springRest.SpringRest.entities.Courses;



public interface CourseService {
```

```java
import java.util.List;
import com.springRest.SpringRest.entities.Courses;


public interface CourseService {

    //Here we will create an abstract method that will return the list
    //of courses

    public List<Courses> getCourses();
    //We won't define it over here....loose coupling
    //Loose Coupling ...changes are easy
}
```

Autowire: Auto object creation and Injection.

To tell spring that Implementation file will be on service layer we will use @Service annonation.

```java
package com.springRest.SpringRest.service;
import java.util.ArrayList;

@Service
public class CourseServiceImplementation implements CourseService {

    //Storing the list temporary here
    //In future we will connect this from data-base
    List<Courses> list;

    public CourseServiceImplementation() {
        list = new ArrayList<>();
        list.add(new Courses(145, "Maths " , "Mathematic Course"));
        list.add(new Courses(146, "Science" , "Science Courses"));
        list.add(new Courses(147, "English" , "Englishh Course"));

    }


    @Override
    public List<Courses> getCourses() {
        // TODO Auto-generated method stub
        return list;
    }
```



Array: JSON object do key and value.

When you run your application now, these will be the results.

Now creating one more layer: DOA, we can get this data from database also.

data from database also.

Now we can create more fxns for the other METHODS.

GET → single id:

```
package com.springRest.SpringRest.service;
import java.util.List;

public interface CourseService {

    //Here we will create an abstract method that will return the list
    //of courses

    public List<Courses> getCourses();
    //We won't define it over here....loose coupling
    //Loose Coupling ...changes are easy
    //it will call its child body

    public Courses getSingleCourse(long courseId);
}
```

Creating abstract Method in the interface

MyController.java | CourseService.java | CourseServiceImplementation.java | SpringRestApplication.java

```
20
21      }
22
23
24      @Override
25      public List<Courses> getCourses() {
26          // TODO Auto-generated method stub
27          return list;
28      }
29
30
31      @Override
32      public Courses getSingleCourse(long courseId) {
33
34          Courses c = null;
35
36          for(Courses course : list) {
37              if(course.getId()==courseId) {
38                  c = course;
39                  break;
40              }
41          }
42
43          return c;
44
45      }
46
```

Implementing the abstract method in the Implementation file.

MyController.java | CourseService.java | CourseServiceImplementation.java | SpringRestApplication.java

```
28
29      @GetMapping("/courses")
30      public List<Courses> getCourses(){
31
32          return this.csvariable.getCourses();
33
34      }
35
36
37      //to get single course
38      @GetMapping("/courses/{courseId}")
39      //via {} : You can get the values....else whole will become the url
40      public Courses getCourse(@PathVariable String courseId) {
41
42          //@PathVariable : Annonation to get the {} : Dynamic Value
43          //Input will be in the form of string so we will convert it
44          //to long first
45
46          return this.csvariable.getSingleCourse(Long.parseLong(courseId));
47      }
48  }
49
50
51
52
53
54
```

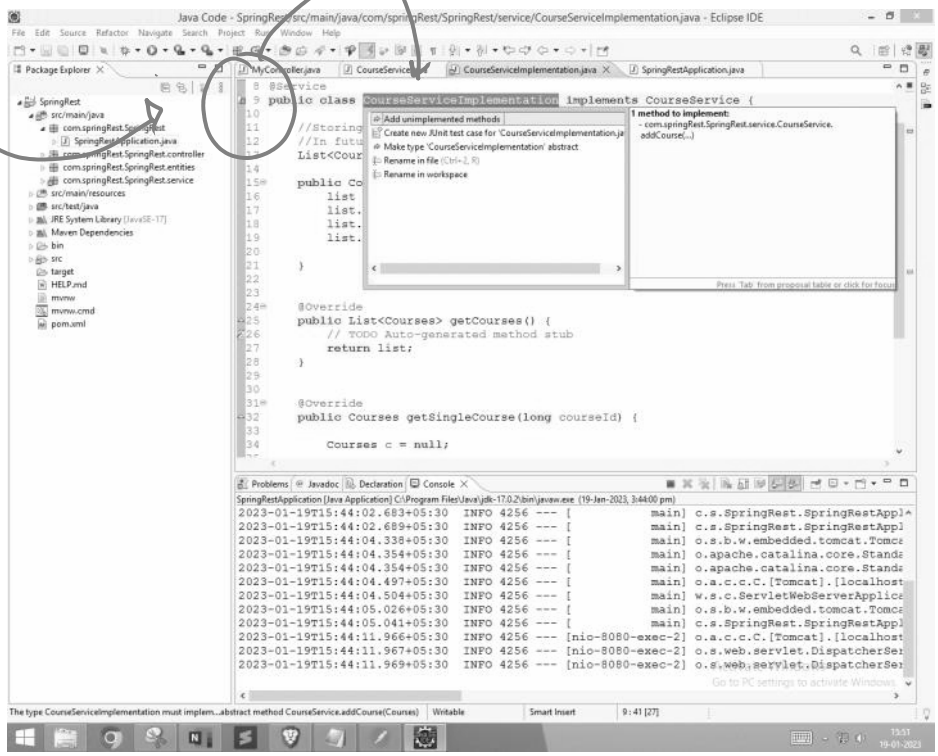
Writing the statements in controller.

{ } : Dynamic I/P

@PathVariable

↳ Convention

When i make some changes in the interface - i can use the "💡" Bulb icon to add the overridden fns in the implementation class.



FOR POST:

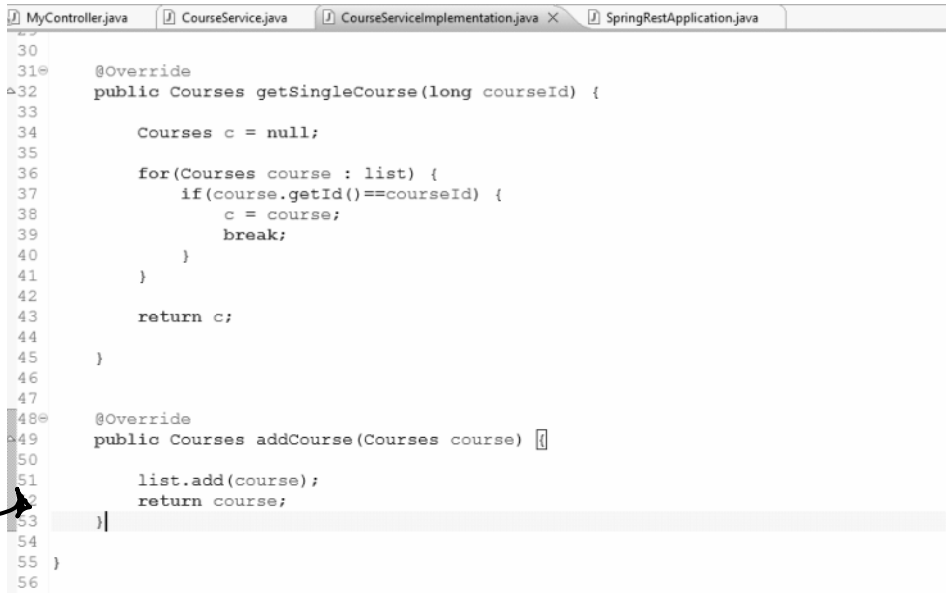
```
MyController.java × | CourseService.java | CourseServiceImplementation.java | SpringRestApplication.java

41      //via {} : You can get the values....else whole will become the url
42      public Courses getCourse(@PathVariable String courseId) {
43
44          //@PathVariable : Annotation to get the {} : Dynamic Value
45          //Input will be in the form of string so we will convert it
46          //to long first
47
48          return this.csvariable.getSingleCourse(Long.parseLong(courseId));
49      }
50
51
52
53      @PostMapping("/courses")
54      public Courses addCourse(@RequestBody Courses course) {
55
56          //Request Body : The object will come from the Request
57          return this.csvariable.addCourse(course);
58      }
59  }
60
61
62
```

Mapping changed to POST.

Annotation to get JSON object from Request.

Alternative: @PostMapping(path="/courses" , consumer = "application/json")

After creating an abstract method in the interface... we will implement it.



To generate request from postman we need to pass data in the form of JSON.

Alternative of @GET/Post Mapping:

@RequestMapping(path="/courses" , method =RequestMethod.Get)

DOA layer