🡪Create spring boot project and maven dependency

🡪Added some dependency like lombok, data-jpa, mysql-connector, web dependency

🡪configure the application.properties file

#Server port

server.port=8888

#Mysql jdbc Properties

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.username=root

spring.datasource.password=jhalla

spring.datasource.url=jdbc:mysql://localhost:3306/thoughti

#Logging JPA Properties

spring.jpa.show-sql=true

#DDL Logging

spring.jpa.hibernate.ddl-auto=update

#Hibernate Dialect

spring.jpa.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

🡪Create two Entity which name is Orders and Item

🡪We will add some annotation which will help to create bean class and tell to spring container to create the table, primary key and foreign key in both entity class. And also help to mapping in both tables.

@Entity

@Table(name="orders")

@Data

@AllArgsConstructor

@NoArgsConstructor

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@ManyToOne(optional = **false**)

@JoinColumn(name = "item\_id", nullable = **false**)

@JsonIgnore

@OneToMany(mappedBy = "items", cascade = CascadeType.***ALL***)

🡪Now we will create the Dao(Data access object) interface which will extends the JpaRepository<Orders, Long> class which will provide the lots of methods. This methods helps to perform CURD operations as well as other operations.

We will annotate that interface to @Repository annotation which will tell to spring container that decorated class/interface is a repository.

I created two interfaces OrderRepo and ItemRepo. We will also execute some custom query in this interface.

🡪Now We will create the service package and inside will create two service class and will annotate with @Service annotation which will tell to spring container that provide some business functionalities and perform some business tasks.

We will create the reference of repository class with @Autowired annotation. And that reference will help to call jpa methods means will help to perform CURD operations like insert data into database, get all details, get details using id etc.

🡪when we call findById(id) method so it will return optional class so we have to get the data and check data is present or not .

If present, it will call the get() method otherwise we created some custom Exception which will return custom output.

**public** Item findItemsById(**long** id) **throws** InvalidItemByException {

Optional<Item> items = itemRepo.findById(id);

**if**(items.isPresent()) {

**return** items.get();

}**else** {

**throw** **new** InvalidItemByException("There is no item existing with id : "+id);

}

}

🡪For handle the Exception, We will create the another package and create class which will extends Exception class and generate the constructor of the super class.

**package** com.nikhil.exception;

**public** **class** InvalidItemByException **extends** Exception{

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** InvalidItemByException() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** InvalidItemByException(String message, Throwable cause) {

**super**(message, cause);

// **TODO** Auto-generated constructor stub

}

**public** InvalidItemByException(String message) {

**super**(message);

// **TODO** Auto-generated constructor stub

}

**public** InvalidItemByException(Throwable cause) {

**super**(cause);

// **TODO** Auto-generated constructor stub

}

}

🡪 After setup basic things, now we will create the controller class of both entity and will annotate with @RestController(@Controller + @ResponseBody) which will tell to spring container that this class handles all the request and process or call some methos then return response body.

🡪Here firstly we will create the url using some annotation like @RequestMapping, @GetMapping, @PostMapping, @PutMapping, @Delete Mapping and many more.

🡪and also we will autowire the orderService reference which will help to call methods or business logic. And also control all the requests and return the response using restful api.

🡪We will handle all the requests and perform some operations and return the response.

🡪In our task, we want to get the data and insert the data into database so we will used only two mapping which is @GetMapping and @PostMapping.

🡪We will create the methods and using orderService reference, we will control the request and call the method and return the response.

List<Orders> orders = orderService.findAllOrders();

Orders orderSaved = orderService.saveOrder(order);

🡪but we are also manage the http status so we will create the object of ResponseEntity which will help to manage the http status code.

@GetMapping

**public** ResponseEntity<List<Orders>> getFindAllOrders() {

List<Orders> orders = orderService.findAllOrders();

ResponseEntity<List<Orders>> responseEntity = **new** ResponseEntity<List<Orders>>(orders,HttpStatus.***OK***);

**return** responseEntity;

}

@PostMapping

**public** ResponseEntity<Orders> saveBook(@RequestBody Orders order) {

Orders orderSaved = orderService.saveOrder(order);

ResponseEntity<Orders> responseEntity = **new** ResponseEntity<Orders>(orderSaved, HttpStatus.***CREATED***);

**return** responseEntity;

}

@GetMapping("/{id}")

**public** ResponseEntity<List<Orders>> findAllPages(@PathVariable("id") **long** id) **throws** InvalidItemByException{

List<Orders> orders = itemService.findItemsById(id).getOrders();

ResponseEntity<List<Orders>> responseEntity = **new** ResponseEntity<List<Orders>>(orders, HttpStatus.***OK***);

**return** responseEntity;

}

🡪After that, I created some junit test case for checking data. I created two methods which will check data is inserted successfully or not.

@Test

**public** **void** itemAdded() {

Item i = **new** Item("Mobile",1,2);

itemRepo.save(i);

*assertNotNull*(itemRepo.findById((**long**)i.getItemId()).get());

}

@Test

**public** **void** orderAdded() {

Orders o = **new** Orders("01/01/2021","In progress");

orderRepo.save(o);

*assertNotNull*(orderRepo.findById((**long**)o.getOrderId()).get());

}

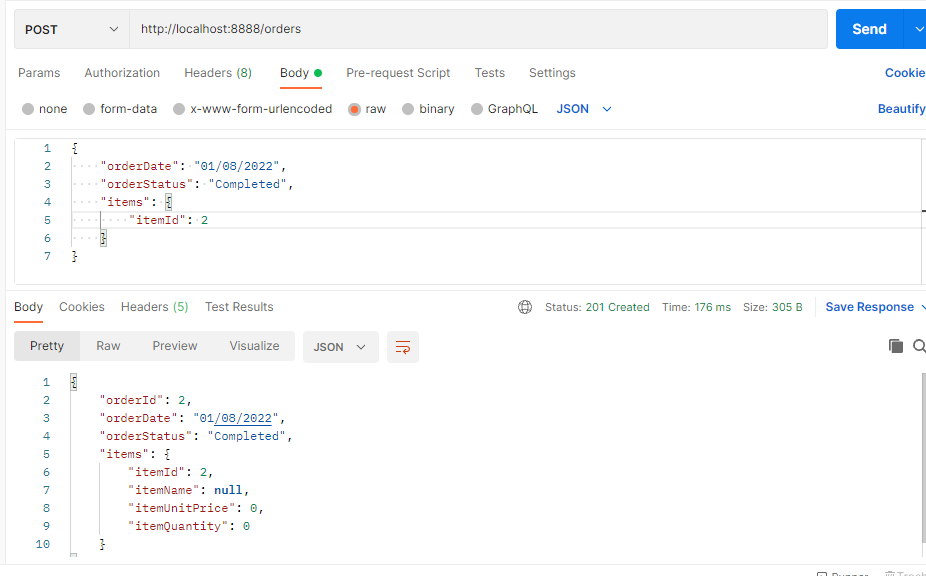
🡪Now, code is completed. We will change the code according the requirements.

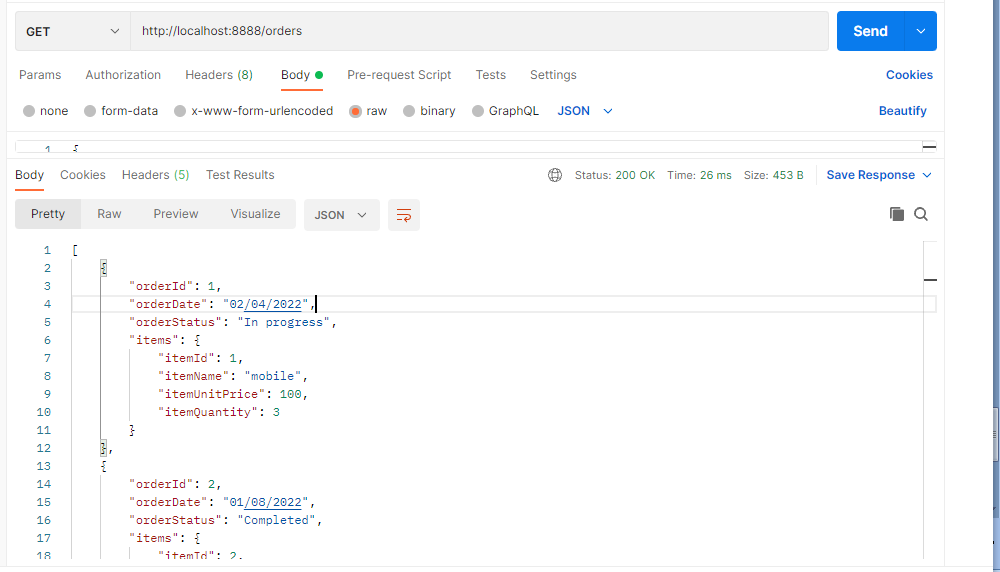
🡪when we will run the spring boot project, so spring container will create the query and perform some operation which will help to create the table and columns into mysql database.

🡪Now, we will hit the api into postman and give some payload if needed. and check the results.

**Note:** I can create any REST API according the requirements and also manage the mapping(one to one, one to many, many to one and many to many) using annotations.

**POSTMAN:** I put some screenshot which you want the output so please have a look once. I handled all the request and generate the response.

****

****

**[**

**{**

**"orderId": 1,**

**"orderDate": "02/04/2022",**

**"orderStatus": "In progress",**

**"items": {**

**"itemId": 1,**

**"itemName": "mobile",**

**"itemUnitPrice": 100,**

**"itemQuantity": 3**

**}**

**},**

**{**

**"orderId": 2,**

**"orderDate": "01/08/2022",**

**"orderStatus": "Completed",**

**"items": {**

**"itemId": 2,**

**"itemName": "lappy",**

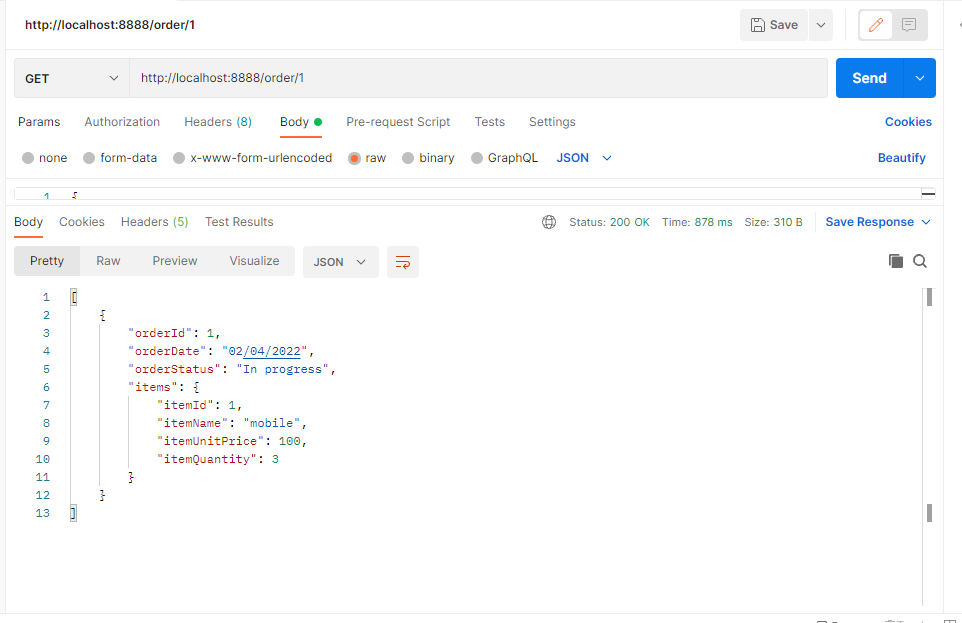
**"itemUnitPrice": 5555,**

**"itemQuantity": 1**

**}**

**}**

**]**

****