**Amigosmaster**

**============**

Front end calls >> API

Controller <> Service <> Repository

inside the controller instintiate the service

Controller to Service

controller is a class, you define the service class as a property, now you should instintiate the

service class property to start use it and link it, but this not needed when you use @component

and @Autwired

@Componenet at the service class will make it available for use in one copy and no need to create

copies at other classes , at the other classes for example the controller you can directly inject

the service class property in side the controller class constructor and it will be linked to the

service class

DAO and Interface (Repostiory)

Repo interface is a class which if other clases called it has to implement the method inside

At the service you need to refrence the repo which you will pull the data from

at service also after pull the data you manupilated it

Repo is a class where represent the source of data it could be data base or single or multiple new

records of data or it could be List\Array

Controller > Service > Interface ProductRepo >> ProductFakeRepo { new records } value = "fake"

>> ProductRepository {Collection.emptyList} value = "real"

Two impls and single repo!! , run the appp will cause error

you have to define @Primary or @Qualifier at the service to prior one over other

@Components is the father of @Service and @Repository

These annotations creats BEANS of the annotated class, bean is replacing the new keyword which we use to

instintiate classes to start use it and operate

In behind spring boot create the beans and keep it ready for use

spring boot @Autowired helps you to inject the classes to each others through constructors

@Bean annotation

it gives you the option to predefine configuration and instruction to spring boot before instintiate the class

usually the class named NameConfiguration

CommandLineRunner class used in Beans to inject any kind of commands or logic while starting

@Configuration is used on top of the class and @Bean on the methods used

How SB serialize and deserialze the json? using Jackson

@JsonProperty("sku\_num") to rename the property in the json

@JsonIgnore // to ignore password form showing in the json

Validations

===========

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

@NotBlank(message = "any message") at the properties class + @Valid at the controller method

@Email to make sure the format is email

javax.validation.constraints has many constraints

@Max the max value

@Min same

@Decimal max min

Exceptions

==========

=> GET 127.0.0.1:8080/api/v1/product/4 and lets say 4 not exist the server will reply 500 and terminal print (java.lang.IllegalStateException: Product not found)

and the client postman get

{

"timestamp": "2022-11-06T09:32:57.329+00:00",

"status": 500,

"error": "Internal Server Error",

"path": "/api/v1/product/4"

}

(java.lang.IllegalStateException: Product not found) the "Product not found" custom message from the getProduct method in the productService

server.error.include-binding-errors=always

server.error.include-message=always

server.error.include-stacktrace=always

These parameters at the application properties will pipe the IllegalStateException custom message to the Response

=> 127.0.0.1:8080/api/v1/product/4 this giving 500 where it should give 404

to do that we created new class NotFoundException extends RunTimeException and annotated by

@ResponseStatus(value = HttpStatus.NOT\_FOUND)

now the reposnse is

{

"timestamp": "2022-11-06T16:17:18.640+00:00",

"status": 404,

"error": "Not Found",

"message": "Product not found",

"path": "/api/v1/product/4"

}

Data and JPA

============

H2 is in memory database good for practicing

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.show-sql=true

spring.h2.console.enabled=true

spring.h2.console.path=/h2

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

@Entity and @Table annotations to tag the original class

@Id annotation for the id field of the persistence class since its the PK

create empty constructor for the persistence main class

public Product() {

}

ProductRepository class should extends JpaRepository to use its features and driver ...etc.

ProductRepository should tagged by @Repository

Reload the project and access H2 from http://127.0.0.1:8080/h2

URL jdbc:h2:mem:testdb

Open Feign

==========

--need to install spring cloud and Open Feign

2021.0.x since the spring version is 2.7

<properties>

<spring.cloud-version>2021.0.x</spring.cloud-version>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring.cloud-version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

So basically you need client class annotated by @Restcontroller and @EnableFeignClients

This class has the get methods which will

Logging

=======

First you will need to instintiate variable from Logger class

private final static Logger LOGGER = LoggerFactory.getLogger(ProductService.class);

then use LOGGER methods at any method to throw log or error

Actuator

========

Great tool for monitoring

Setup

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

#Info section

info.app.name=Learn spring projects chain

info.app.description=Learn Spring boot

info.app.version=1.0.0

info.company.name=Fahd Mannaa

#Actuator

management.endpoints.jmx.exposure.include=health,info,env,beans

management.endpoints.web.exposure.include=\*

management.info.env.enabled=true

management.endpoint.health.show-details=always

Reload

http://127.0.0.1:8080/actuator

now at the console when start the app

2022-11-10 08:17:26.018 INFO 2279 --- [ main] o.s.b.a.e.w.EndpointLinksResolver : Exposing 15 endpoint(s) beneath base path '/actuator'

Application properties general notes

====================================

-- How to get value from application property to your code

// At your class you can call value from application.properties

// a.p info.company.name=Fahd Mannaa

@Value("${info.company.name}")

private String companyName;

System.out.println(companyName);

Above will print Fahd Mannaa when the sout called

-- Getting value from application properties and map it to clsss

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

create a class has the exact properties you want from the a.p

annotate the class

@Configuration

@ConfigurationProperties(prefix = "info.app")

at any other method you can call the InfoApp class and use infoApp variable to access

the InfoApp properties and of course the application properties values

usefull for gettin the DB user name for example to the code

Spring Profiles

===============

Using multiple a.p to have two different or more environments

the a.p file copy and paster with name application-dev.properties

and at app configuration add program param

--spring.profiles.active=dev

Convert a.p to yaml

===================

to convert your s.p file to yaml use this plugin

Convert YAML and Properties File

easily click right the a.p file and convert to yml

Package

=======

remove the target folder first

from the menu click maven then install this will clean up, run test, validate ..etc.

and at the end will create the .jar file

same we can do from the command line

remove the target folder first

./mvnw clean install

the output is the target folder again but it has the .jar file

er/target/projectname-0.0.1-SNAPSHOT.jar

and to run the jar file

java -jar projectname-0.0.1-SNAPSHOT.jar

and to mention the exact profile

java -jar amigosmaster-0.0.1-SNAPSHOT.jar --spring.profiles.active=dev

to have two instances running at the same time

java -jar amigosmaster-0.0.1-SNAPSHOT.jar --spring.profiles.active=dev --server.port=3000

run with dockerr

First create the image but make sure first that docker installed

./mvnw spring-boot:build-image -Dspring-boot.build-image.imageName=springio/gs-spring-boot-docker

Second to run the docker image

docker run -p 8080:8080 -t springio/gs-spring-boot-docker

Lombok

======

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

**task-management-app-lesson**

**=========================**

Project and task management application

-- How Class A take access to Class B

(1) Class A create it in the constructor

public class A {

private B bClass;

public A() {

bClass = new B();

}

}

(2) someone out of A create it and pass it to A

and this is what spring take care of it

public class A {

private B b;

public A(B bDependency) {

this.b = bDependency;

}

}

-- @Component annotation needed to make spring scan the annotated class

-- @SpringBootApplication(basePackages = "package name ") this will limit the scan to specific package

-- @Component include @Repository and @Service nothing different names fit better

--