

Core Features

02 May 2024 12:52

IOC - Does the object lifecycle of java Objects, when they are called and destroyed

AOP - handles application spanning functions, Logging, caching and auth etc

DAF - Does database communication, JVC, JPA also does resource management and exception handling

MVC - Allows developers to use MVC, handles requests and building of controllers and responses via view.

Spring Bean - An object handled by the Spring framework

Spring beans get their lifecycle and dependencies managed by Spring.

Spring beans can be configured using an XML or Java code

Spring bean lifecycle:

Handles how it is created, how it behaves, when and how it's destroyed

Beans are all managed within IOC containers

Beans & Components

05 May 2024 16:20

Spring Bean - An object handled by the Spring framework

Spring beans get their lifecycle and dependencies managed by Spring.

Spring beans can be configured using an XML or Java code

Spring bean lifecycle:

Handles how it is created, how it behaves, when and how it's destroyed

Beans are all managed within IOC containers

The screenshot shows a YouTube video player with the title '@CONFIGURATION EXAMPLE'. The video content displays Java code for configuration:

```
@Configuration
public class AppConfig {
    @Bean
    public PaymentService paymentService(
        AccountRepository accountRepository
    ) {
        return new PaymentServiceImpl(accountRepository);
    }
}
```

To declare a BEAN CONFIG use the @Configuration annotation, must be a public class

Use the @Configuration annotation if you want to declare lots of beans

Use @Bean to declare a bean. Cannot be a private (singleton).

```

@Configuration
public class AppConfig {

    @Bean
    public PaymentService paymentService() {
        return new PaymentServiceImpl(accountRepository());
    }

    @Bean
    public AccountRepository accountRepository() {
        return new JdbcAccountRepository(dataSource());
    }

    @Bean
    public DataSource dataSource() {
        return (...)

    }
}

```

Components:

@Component creates a bean for the class and managed by container (cuz bean)

There is the @Repository, @Service and @Controller to further refine containers behaviour.

@Autowired creates instances of required services when needed in the IOC container.

```

@Component
public class PaymentServiceImpl {
    private final AccountRepository accountRepository;

    @Autowired
    public PaymentServiceImpl(
        AccountRepository accountRepository
    ) {
        this.accountRepository = accountRepository;
    }
}

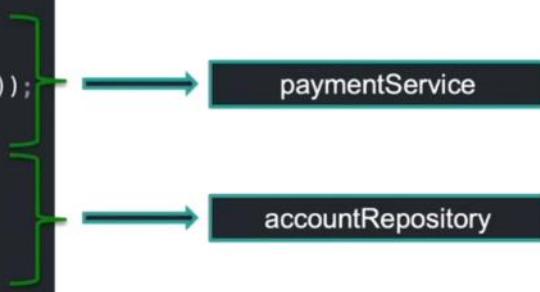
```

Naming:

By default beans are named same as method, but if name is given it will be used

BEAN NAMING

```
@Configuration  
public class AppConfig {  
  
    @Bean  
    public PaymentService paymentService() {  
        return new PaymentServiceImpl(accountRepository());  
    }  
  
    @Bean  
    public AccountRepository accountRepository() {  
        return new JdbcAccountRepository(dataSource());  
    }  
  
    @Bean("ds")  
    public DataSource dataSource() {  
        return (...)  
    }  
}
```



Dependency Injection

05 May 2024 16:48

Constructor Injection:

The slide has a dark background with a green header bar. The title 'CONSTRUCTOR INJECTION' is in bold black font at the top. Below it is a horizontal green bar. The main content area contains two side-by-side code snippets in white text on a dark background.

```
@Service
public class DefaultPaymentService {
    private final AccountRepository accountRepository;
    public DefaultPaymentService(
        AccountRepository accountRepository
    ) {
        this.accountRepository = accountRepository;
    }
}
```

```
@Repository
public class JdbcAccountRepository implements AccountRepository {
    private final DataSource dataSource;
    public JdbcAccountRepository(DataSource dataSource) {
        this.dataSource = dataSource;
    }
}
```

(A) REVIEW

Bean handles lifecycle with IOC container using constructor

@Autowired isn't included here, so dependences aren't instantiated by in practice, should be.

THIS IS THE INJECTION SHOULD YOU USE TO CREATE BEANS

The slide has a dark background with a light gray header bar. The text 'Suppose you have two implementations of a 'MessageService' interface:' is in white. Below it is a code editor window with Java code.

```
java
```

```
public interface MessageService {
    String getMessage();
}

@Component("emailService")
public class EmailService implements MessageService {
    public String getMessage() {
        return "Email message";
    }
}

@Component("smsService")
public class SmsService implements MessageService {
    public String getMessage() {
        return "SMS message";
    }
}
```

Copy code

Now, if you have a class that depends on 'MessageService', and you have both 'EmailService' and 'SmsService' beans defined in your Spring application context, you can use '@Qualifier' to specify which one to inject:

```
java
```

```
Copy code
```

which one to inject:

```
java Copy code
```

```
@Component
public class MyService {
    private final MessageService messageService;

    @Autowired
    public MyService(@Qualifier("emailService") MessageService messageService) {
        this.messageService = messageService;
    }

    // Other methods
}
```

Remember @Autowired handles dependency injection

Use @Qualifier if you have multiple beans in a service so that dependent classes know what bean is what

If you only have two, you can just use @primary

You should use constructor injection because you can test it much easier, not field injection

Method Injection:

METHOD INJECTION

Method injection allows setting one or many dependencies by one method

Allows for initialization work if needed while receiving dependencies.

```
@Service
public class DefaultPaymentService {
    @Autowired
    public void configureClass(
        AccountRepository accountRepository,
        FeeCalculator feeCalculator
    ) {
        // ...
    }
}
```

Using a method to create beans and do dependency injection allows us to do 'pre' injection work

Setter Injection:

SETTER INJECTION

Setter injection follows Java bean naming convention to inject dependency

```
@Service
public class DefaultPaymentService {
    @Autowired
    public void setAccountRepository(
        AccountRepository accountRepository
    ) {
        // ...
    }
}
```

Use setter injection for optional dependencies.

Bean Scoping

05 May 2024 19:09

BEAN SCOPING

- What is "Bean scope"?
- Spring provides multiple scopes
- "Singleton" is the default scope of a bean



All beans are by default singleton

There is a selection of scopes:

Singleton - default (doesn't carry state)

Prototype - creates new instance with each request useful for user specific state

Request - a new bean is created and destroyed with each HTTP request

Session - only kept alive and created during a HTTP session

Application - kept alive with application

Websocket - only kept alive during websocket

BEAN SCOPING

```
@Configuration
public class MyConfiguration {
    @Bean
    @Scope("prototype")
    public Bean1 bean1() { // ... }

    @Bean
    @SessionScope
    public Bean2 bean2() { // ... }
}
```



Bean profile

05 May 2024 19:42

You use bean profiles to trigger beans when conditions are met (development, testing, production)

You can also use this for configurations (that create many beans u know)

The screenshot shows a code editor with three sections of Java code:

```
// Spring component
@Service
@Profile("cloud")
public class DefaultPaymentService implements PaymentService {}

// Configuration class
@Configuration
@Profile("cloud")
public class ApplicationConfig { }

@Configuration
public class ApplicationConfig {
    // Bean configuration method
    @Bean
    @Profile("cloud")
    public PaymentService paymentService() {
        return new DefaultPaymentService();
    }
}
```

PROFILE ACTIVATION - PROGRAMMATICALLY

```
public static void main(String[] args) {
    AnnotationConfigApplicationContext applicationContext;
    applicationContext = new AnnotationConfigApplicationContext();
    applicationContext.getEnvironment().setActiveProfiles("cloud");
    applicationContext.scan("com.alibou.sample");
    applicationContext.refresh();

    PaymentService paymentService =
        applicationContext.getBean(PaymentService.class);
}
```

Because of we have set profile and we can now use the beans with this profile.

We can activate it using the prop file as well

PROFILE ACTIVATION - PROPERTIES FILE

application.yaml

```
spring:  
  profiles:  
    active: cloud
```

application.properties

```
spring.profiles.active= cloud
```



Value Annotation

06 May 2024 17:11

@Value Annotation allows us to inject values from a file

@PropertySource allows us to set the location of this file

The screenshot shows a Java code editor with the following code:

```
@Configuration
@PropertySource("classpath:database.properties")
public class ApplicationConfig {
    @Value("${jdbc.url}")
    private String url;
    @Value("${jdbc.username}")
    private String username;
    @Value("${jdbc.password}")
    private String password;
    @Bean
    public DataSource dataSource() {
        return ....;
    }
}
```

The code uses the `@Value` annotation to inject properties from a file named `database.properties`. The `@Configuration` and `@PropertySource` annotations are also present.

We can read values from other beans using this method

The screenshot shows a Java code editor with the following code:

```
@Component
public class FeeCalculator {
    private String defaultLocale;
    @Value("#{systemProperties['user.region']}")
    public void setDefaultLocale(String defaultLocale) {
        this.defaultLocale = defaultLocale;
    }
}
```

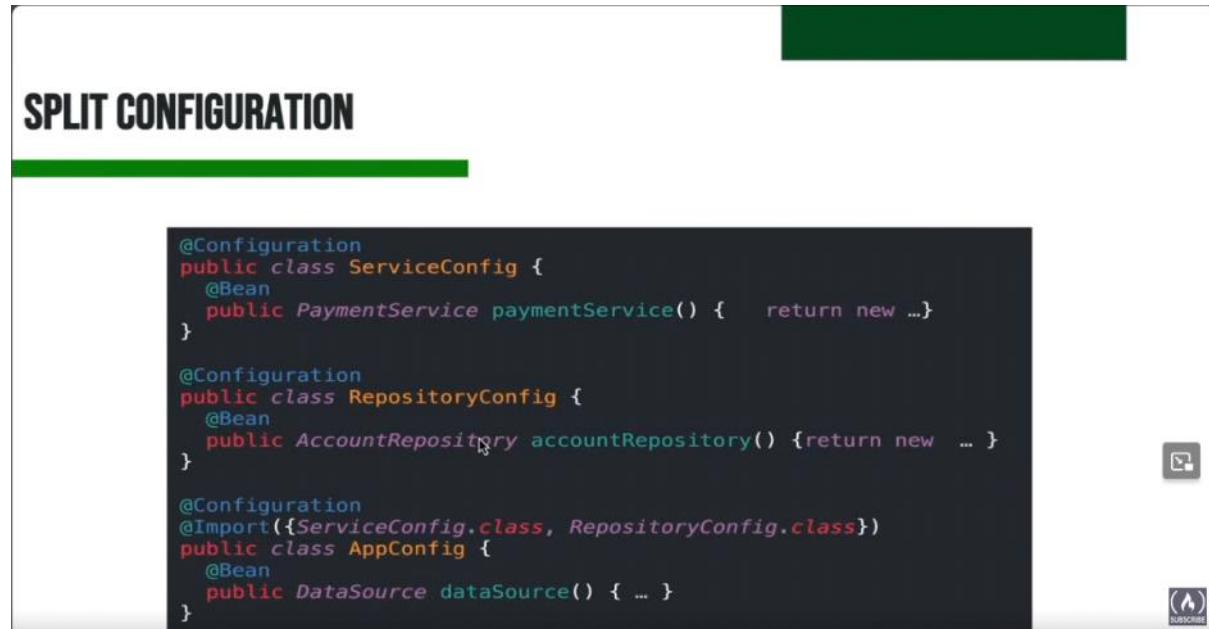
The code uses the `@Value` annotation to read the value of the `user.region` system property and set it as the default locale for the `FeeCalculator` bean.

Split Configuration

06 May 2024 17:16

Ensure that all your spring bean configs are split into different classes

Then, merge them using @Import



The screenshot shows a presentation slide with a dark background. At the top, the title "SPLIT CONFIGURATION" is displayed in large, bold, white capital letters. Below the title, there is a horizontal green bar. The main content area contains Java code for Spring configuration:

```
@Configuration
public class ServiceConfig {
    @Bean
    public PaymentService paymentService() { return new ... }

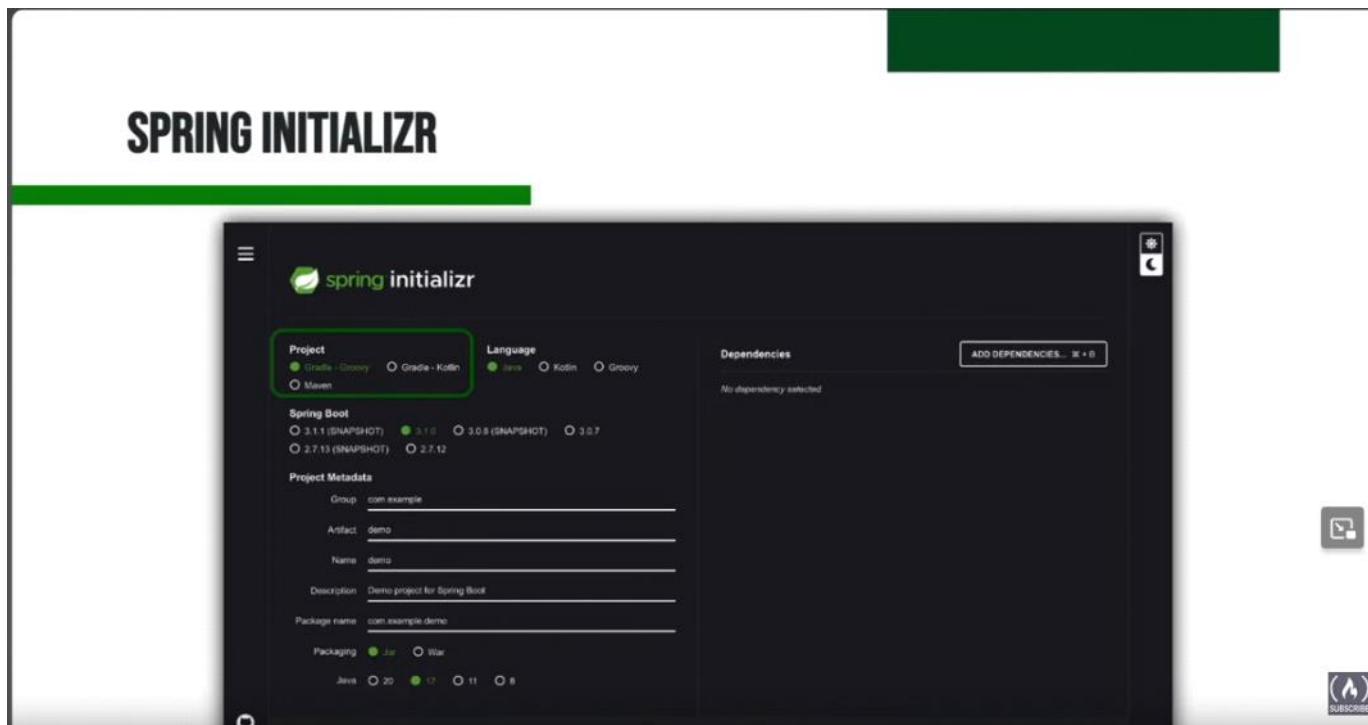
    @Configuration
    public class RepositoryConfig {
        @Bean
        public AccountRepository accountRepository() { return new ... }
    }

    @Configuration
    @Import({ServiceConfig.class, RepositoryConfig.class})
    public class AppConfig {
        @Bean
        public DataSource dataSource() { ... }
    }
}
```

On the right side of the slide, there are two small icons: a square with a play symbol and a square with a downward arrow.

Spring Initializer & setup

06 May 2024 17:21



You need Spring web dependency to use it with REST

Port: 8080

Group: com.Company

Artifact: Product Name

In intelj use Project Structure to change Java Version

Why Spring?

06 May 2024 17:24

Standalone apps

Embeded server (tomcat) etc

Starters - Porm or Gradle (Use groovy)

A lot of Auto Config

A lot of Production ready features

No XML Configs

File Structure

07 May 2024 05:26

Resources folder:

Static is for static HTML etc (don't use this, use REST)

Application.properties - stores custom spring configuration and/or our own custom configuration values

Test folder:

All tests go in here

Pom.xml:

Parent - means everything extends from this parent (in our case spring-boot-starter-parent)

Dependencies -> dependency - Is where you can view, remove and install new dependency.
You can also set there 'scope' i.e in what profile they are available

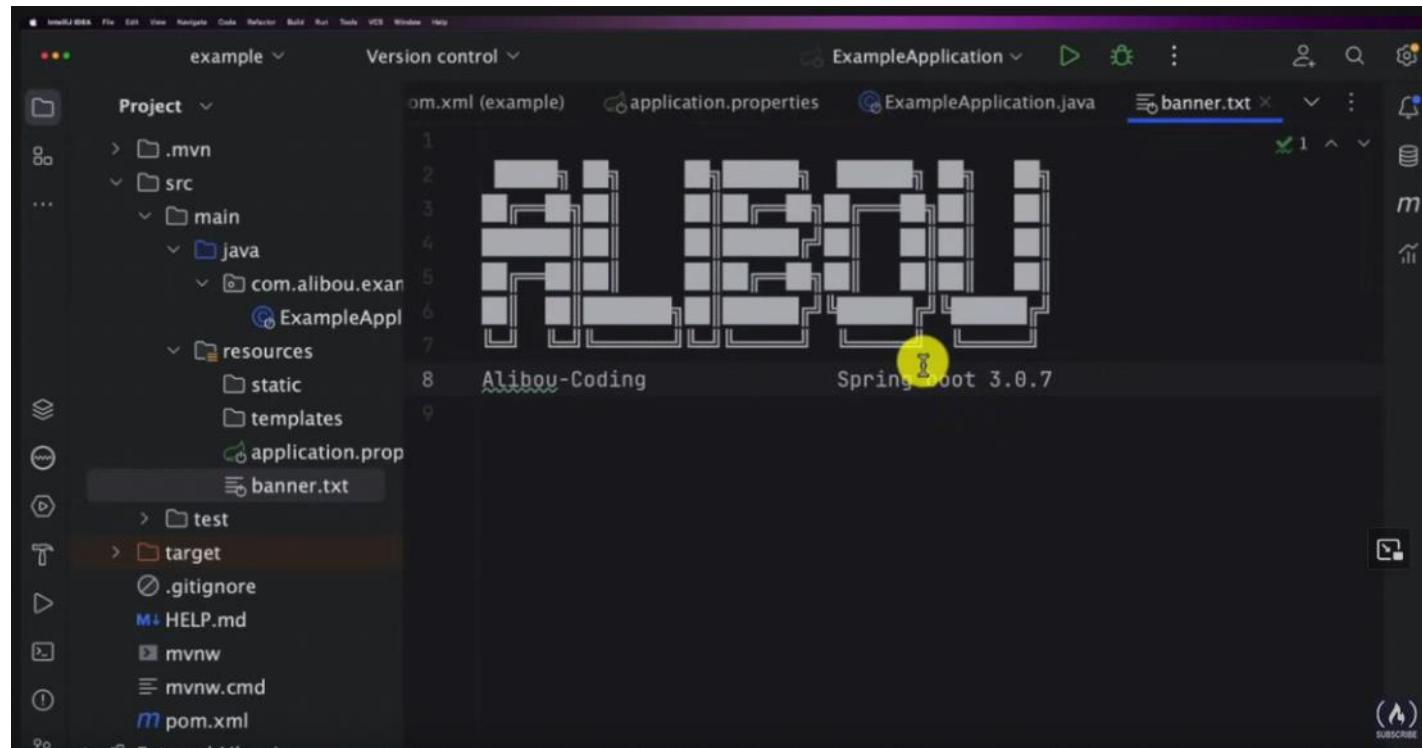
For all dependency - parent version is extended

Changing the banner

07 May 2024 05:38

You can change the banner from Spring to whatever

Create a file in resources called banner.txt



Beans in action + BP

07 May 2024 05:44

The screenshot shows the IntelliJ IDEA interface. The left sidebar displays the project structure under 'example' with files like pom.xml, myFirstService.java, and ExampleApplication.java. The right side shows the code editor for ExampleApplication.java, which contains the main method and a @Bean annotation for myFirstService. Below the code editor is the 'Run' tool window showing the application's log output. The log shows the application starting up, Tomcat initializing, and the service returning "Hello from my firstService".

```
package com.matt.example;
import ...;
@SpringBootApplication
public class ExampleApplication {
    public static void main(String[] args) {
        var ctx = SpringApplication.run(ExampleApplication.class, args);
        myFirstService myFirstClass = ctx.getBean(myFirstService.class); //Now this is managed by the spring framework, as a singleton
        System.out.println(myFirstClass.sayHello());
    }
    @Bean
    public myFirstService myFirstClass() {
        return new myFirstService();
    } //This bit creates a bean instance of myFirstService
}
```

```
2024-05-07T05:50:43.762+01:00 INFO 14636 --- [example] [main] com.matt.example.ExampleApplication : Starting ExampleApplication using Java 17.0.11 with PID 14636 (C:\Users\mbutl\Desktop\example\target\classes)
2024-05-07T05:50:43.766+01:00 INFO 14636 --- [example] [main] com.matt.example.ExampleApplication : No active profile set, falling back to 1 default profile: "default"
2024-05-07T05:50:44.341+01:00 INFO 14636 --- [example] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2024-05-07T05:50:44.347+01:00 INFO 14636 --- [example] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-05-07T05:50:44.348+01:00 INFO 14636 --- [example] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.20]
2024-05-07T05:50:44.382+01:00 INFO 14636 --- [example] [main] o.a.c.c.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-05-07T05:50:44.382+01:00 INFO 14636 --- [example] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 579 ms
2024-05-07T05:50:44.615+01:00 INFO 14636 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ""
2024-05-07T05:50:44.621+01:00 INFO 14636 --- [example] [main] com.matt.example.ExampleApplication : Started ExampleApplication in 1.35 seconds (process running for 1.546)
Hello from my firstService
```

A BETTER WAY BELOW:

The screenshot shows the code for myFirstService.java. It defines a class named myFirstService with a single method sayHello() that returns the string "Hello from my firstService".

```
package com.matt.example;

import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Component;
@Component
public class myFirstService {

    public String sayHello() {
        return "Hello from my firstService";
    }
}
```

```
1 package com.matt.example;
2
3 > import ...
4
5
6 @SpringBootApplication
7 public class ExampleApplication {
8
9
10 >     public static void main(String[] args) {
11         var ctx = SpringApplication.run(ExampleApplication.class, args);
12
13         myFirstService myFirstClass = ctx.getBean(myFirstService.class); //Now this is managed by the spring framework, as a singleton
14         System.out.println(myFirstClass.sayHello());
15     }
16
17
18 }
19
```

We don't need a bean instance creation if we use Component annotation, same with Service annotation .

@Service, @Repository extends @Component

Best practice:

Split your configuration into a ApplicationConfig to handle bean instance creation

```
pom.xml (example)  myFirstService.java  ApplicationConfig.java  ExampleApplication.java
1 package com.matt.example;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5
6 @Configuration
7 public class ApplicationConfig {
8
9     @Bean
10     public myFirstService myFirstClass() {
11         return new myFirstService();
12     } //This bit creates a bean instance of myFirstService
13 }
```

Now no need for @Service, @Component annotation for myFirstService class because its all handled here

Bean naming in action

07 May 2024 06:02

```
1 package com.matt.example;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5
6 @Configuration
7 public class ApplicationConfig {
8
9     @Bean("myBean")
10    public myFirstService myFirstClass() {
11        return new myFirstService();
12    } //This bit creates a bean instance of myFirstService
13 }
14 |
```

```
1 package com.matt.example;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class ExampleApplication {
8
9     public static void main(String[] args) {
10         var ctx = SpringApplication.run(ExampleApplication.class, args);
11
12         myFirstService myFirstClass = ctx.getBean("myBean", myFirstService.class); //Now this is managed by the spring framework, as a
13         System.out.println(myFirstClass.sayHello());
14     }
15
16
17
18 }
19 |
```

Beans & Constructor Dependency Injection

07 May 2024 06:22

Autowired handles Dependency injection

Autowired ensures that all bean dependencies are initialized when needed

Without autowired in below code there wouldn't be an instance of myFirstClass bean for constructor to use

KEEP IN MIND: MyFirstCLass is the BEAN here (as per our ApplicationConfig) not myFirstService

MyFirstService is where the MyFirstClass bean is injected

```
1 package com.matt.example;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5
6 2 usages
7 @Service
8 public class myFirstService {
9
10    2 usages
11    private final myFirstClass myFirstClass;
12
13    @Autowired
14    public myFirstService(myFirstClass myFirstClass) {
15        this.myFirstClass = myFirstClass;
16    } //Dependency beans are initialized using Autowired
17
18    1 usage
19    public String TellAStory() {
20        return "the dependency is saying: " + myFirstClass.sayHello();
21    }
22
```

In modern spring, we don't need autowired, spring will search for a bean called "myFirstClass" and inject it

Lots of beans...

07 May 2024 06:55

If you have serval beans of the same type use qualifiers

```
2 usages
@Service
public class myFirstService {

    2 usages
    private final myFirstClass myFirstClass;

    @Autowired
    public myFirstService(@Qualifier("bean1") myFirstClass myFirstClass) {
        this.myFirstClass = myFirstClass;
    } //Dependency beans are initialized using Autowired
```

```
6
7 @Configuration
8  public class ApplicationConfig {
9
10    @Bean("myBean")
11    @Qualifier("bean1")
12    public myFirstClass myFirstBean() {
13        return new myFirstClass(myVar: "firstBean");
14    } //This bit creates a bean instance of myFirstService
15
16    @Bean("myBean")
17    @Qualifier("bean2")
18    public myFirstClass mySecondBean() {
19        return new myFirstClass(myVar: "secondBean");
20    } //This bit creates a bean instance of myFirstService
21 }
22 |
```

This also works with different types (int strings)

You can tell Spring in what the 'primary' bean is instead of using qualifiers

```
5 import org.springframework.context.annotation.Configuration;
6 import org.springframework.context.annotation.Primary;
7
8 @Configuration
9 public class ApplicationConfig {
10
11     @Bean
12     public myFirstClass myFirstBean() {
13         return new myFirstClass( myVar: "firstBean");
14     } //This bit creates a bean instance of myFirstService
15
16     @Bean
17     public myFirstClass mySecondBean() {
18         return new myFirstClass( myVar: "secondBean");
19     } //This bit creates a bean instance of myFirstService
20
21     @Bean
22     @Primary
23     public myFirstClass myThirdBean() {
24         return new myFirstClass( myVar: "ThirdBean");
25     } //This bit creates a bean instance of myFirstService
26
27
```

Naming & Field injection

07 May 2024 07:07

```
1 package com.matt.example;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.beans.factory.annotation.Qualifier;
5 import org.springframework.stereotype.Service;
6
7 2 usages
8 @Service
9
10 public class myFirstService {
11
12     @Autowired
13     private myFirstClass myFirstClass;
14
15     1 usage
16     public String TellAStory() {
17         return "the dependency is saying: " + myFirstClass.sayHello();
18     };
19
20 }
```

DON'T USE THIS

The default name of beans are their method names so qualifiers can be used like this:

Project

- example C:\Users\mbutl\Desktop\example
 - idea
 - .mvn
 - wrapper
 - maven-wrapper.jar
 - maven-wrapper.properties
 - src
 - main
 - java
 - com.matt.example
 - ApplicationConfig
 - ExampleApplication
 - myFirstClass
 - myFirstService
 - resources
 - test
 - target
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
 - External Libraries
 - Scratches and Consoles

Run ExampleApplication

Console

```

2024-05-07T07:12:44.545+01:00 INFO 66984 --- [example] [main] com.matt.example.ExampleApplication : Starting ExampleApplication using Java 17.0.11 with PID 66984 (C:\User
2024-05-07T07:12:44.549+01:00 INFO 66984 --- [example] [main] com.matt.example.ExampleApplication : No active profile set, falling back to 1 default profile: 'default'
2024-05-07T07:12:45.179+01:00 INFO 66984 --- [example] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2024-05-07T07:12:45.178+01:00 INFO 66984 --- [example] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-05-07T07:12:45.178+01:00 INFO 66984 --- [example] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.20]
2024-05-07T07:12:45.213+01:00 INFO 66984 --- [example] [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-05-07T07:12:45.213+01:00 INFO 66984 --- [example] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 626 ms
2024-05-07T07:12:45.443+01:00 INFO 66984 --- [example] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-05-07T07:12:45.449+01:00 INFO 66984 --- [example] [main] com.matt.example.ExampleApplication : Started ExampleApplication in 1.213 seconds (process running for 1.591
the dependency is saying: Hello from my myFirstClass: secondBean
  
```

Method injection

07 May 2024 07:15

```
1 package com.matt.example;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.beans.factory.annotation.Qualifier;
5 import org.springframework.stereotype.Service;
6
7     2 usages
8 @Service
9 public class myFirstService {
10     2 usages
11     private myFirstClass myFirstClass;
12     @Autowired
13     public void injectDependencies(@Qualifier("bean1")myFirstClass myFirstClass) {
14         this.myFirstClass = myFirstClass;
15     }
16
17     1 usage
18     public String TellAStory() {
19         return "the dependency is saying: " + myFirstClass.sayHello();
20     }
21 }
```

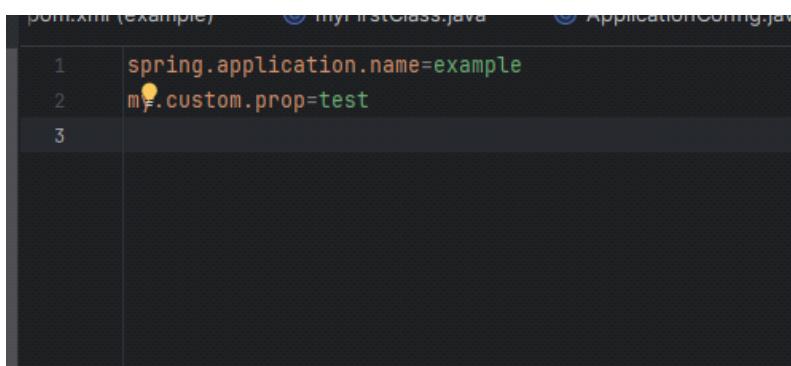
Environment bean

07 May 2024 07:29

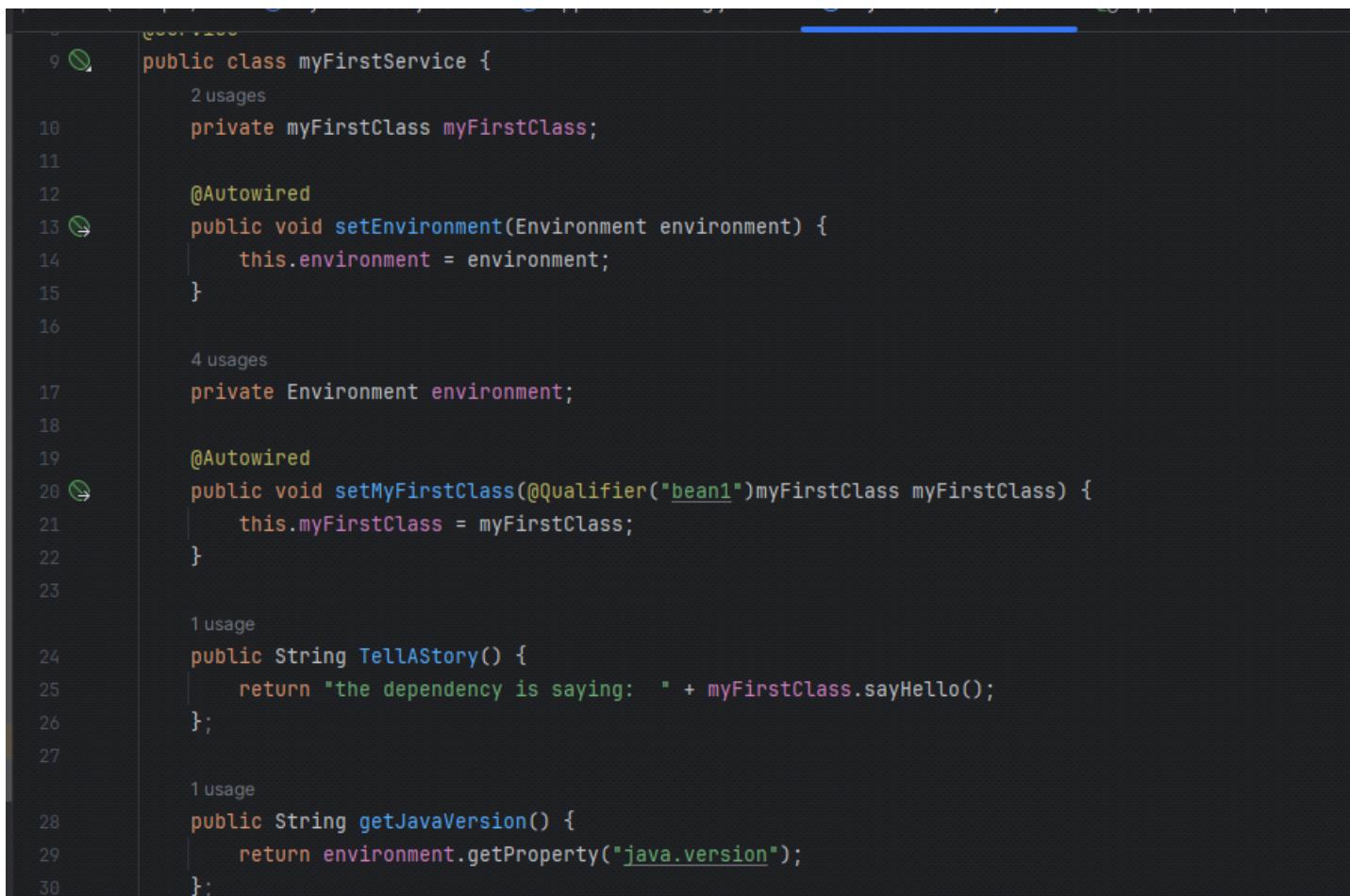
Beans can be injected in any way (ideally constructor) in below examples we use setter injection

The environment bean has lots of built in functionality

We can add custom props in application.properties file



```
spring.application.name=example
my.custom.prop=test
```



```
public class myFirstService {
    private myFirstClass myFirstClass;

    @Autowired
    public void setEnvironment(Environment environment) {
        this.environment = environment;
    }

    private Environment environment;

    @Autowired
    public void setMyFirstClass(@Qualifier("bean1")myFirstClass myFirstClass) {
        this.myFirstClass = myFirstClass;
    }

    public String TellAStory() {
        return "the dependency is saying: " + myFirstClass.sayHello();
    }

    public String getJavaVersion() {
        return environment.getProperty("java.version");
    }
}
```

Setter injection

07 May 2024 07:25

Create a setter for each bean to handle injection

```
1 package com.matt.example;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.beans.factory.annotation.Qualifier;
5 import org.springframework.stereotype.Service;
6
7 2 usages
8 @Service
9 public class myFirstService {
10
11     2 usages
12     private myFirstClass myFirstClass;
13
14     @Autowired
15     public void setMyFirstClass(@Qualifier("bean1")myFirstClass myFirstClass) {
16         this.myFirstClass = myFirstClass;
17     }
18
19 }
20
21 1 usage
22 public String TellAStory() {
23     return "the dependency is saying: " + myFirstClass.sayHello();
24 }
```

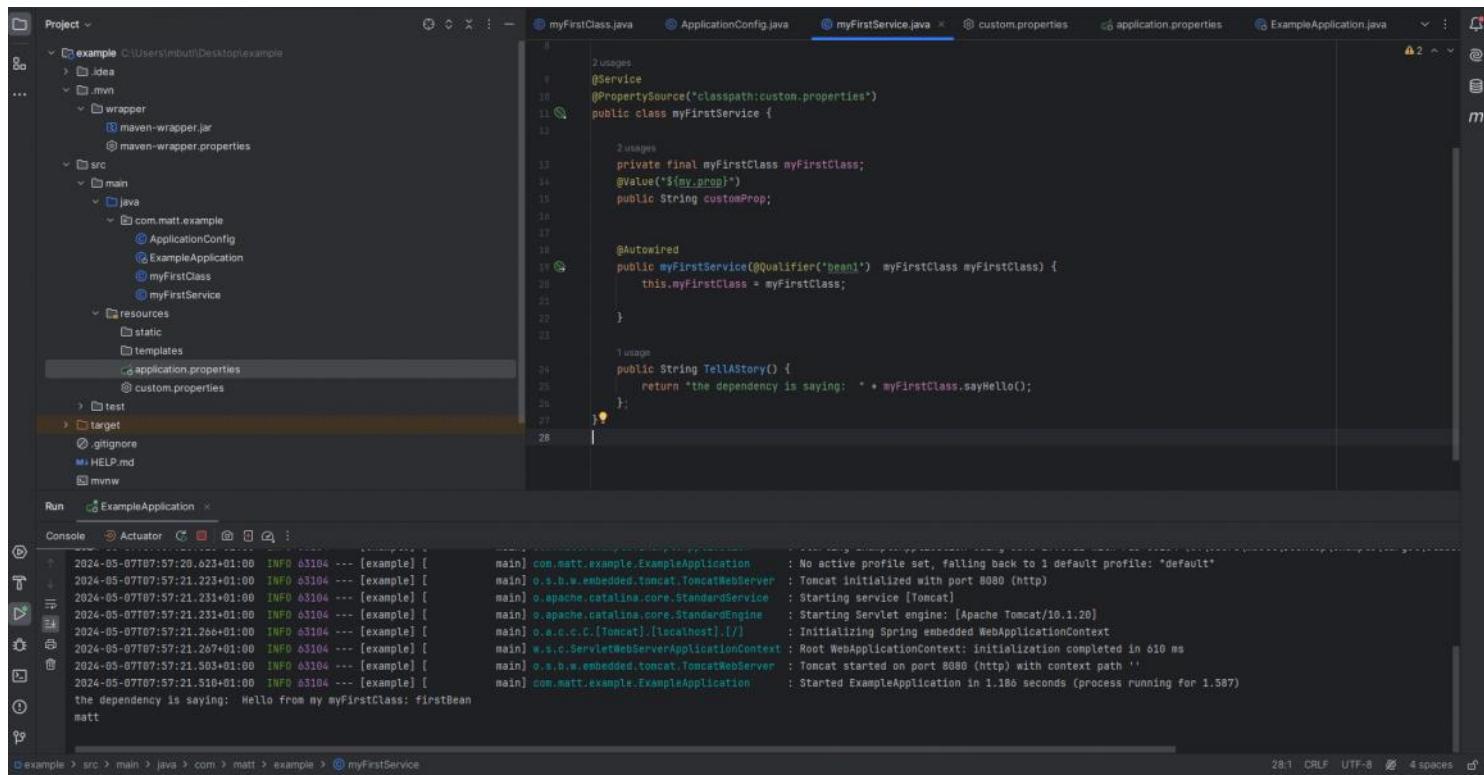
ALWAYS USE CONSTRUCTOR INJECTOR

\$Value

07 May 2024 07:51

\$Value Is used to read custom values stored in external files

Set a custom location with PropertySource



Project tree:

- example
- src
 - main
 - java
 - com.matt.example
 - resources
 - application.properties
 - custom.properties
 - test
 - target

Code Editor (myFirstService.java):

```
1 2 usages
2 @Service
3     (@PropertySource("classpath:custom.properties"))
4     public class myFirstService {
5
6         2 usages
7         private final myFirstClass myFirstClass;
8         @Value("${my.prop}")
9         public String customProp;
10
11     @Autowired
12     public myFirstService(@Qualifier("bean1") myFirstClass myFirstClass) {
13         this.myFirstClass = myFirstClass;
14     }
15
16     1 usage
17     public String TellAStory() {
18         return "the dependency is saying: " + myFirstClass.sayHello();
19     }
20 }
```

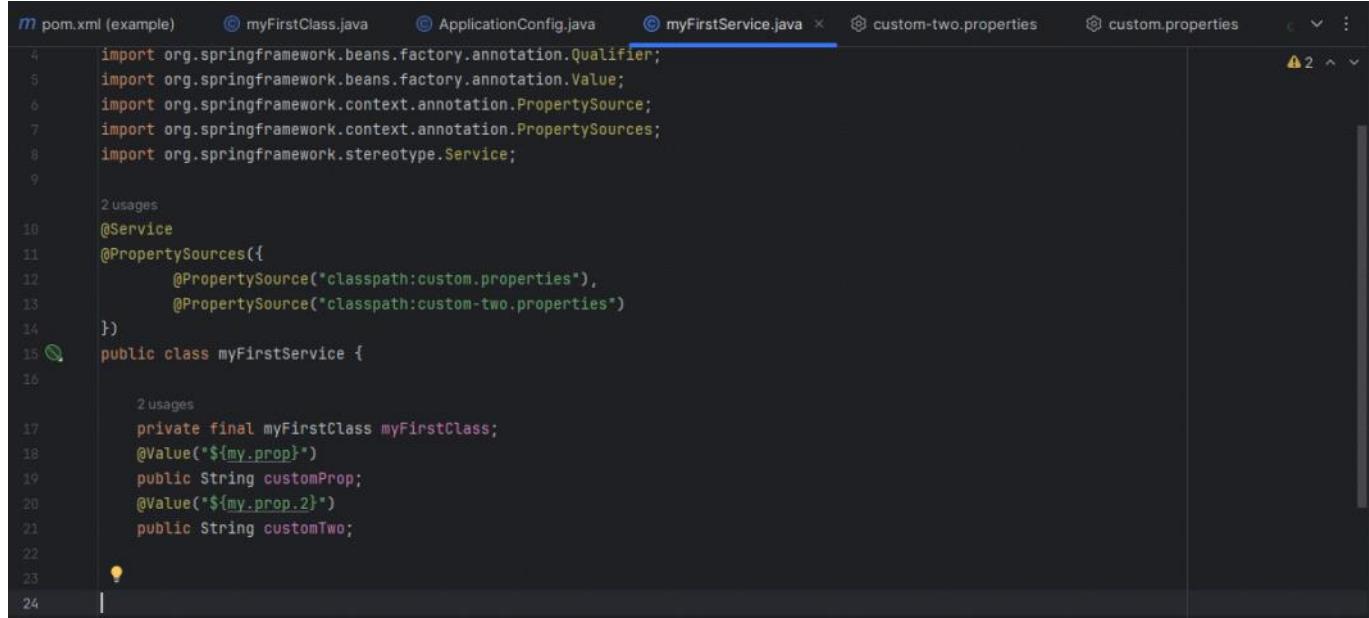
Run Tab (ExampleApplication):

```
2024-05-07T07:57:20,623+01:00 INFO 63104 --- [example] [main] com.matt.example.ExampleApplication : No active profile set, falling back to 1 default profile: 'default'
2024-05-07T07:57:21,233+01:00 INFO 63104 --- [example] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2024-05-07T07:57:21,251+01:00 INFO 63104 --- [example] [main] o.apache.catalina.core.StandardService : Starting service [tomcat]
2024-05-07T07:57:21,251+01:00 INFO 63104 --- [example] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.20]
2024-05-07T07:57:21,266+01:00 INFO 63104 --- [example] [main] o.e.c.c.C.[tomcat].[localhost].[] : Initializing Spring embedded WebApplicationContext
2024-05-07T07:57:21,267+01:00 INFO 63104 --- [example] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 610 ms
2024-05-07T07:57:21,503+01:00 INFO 63104 --- [example] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-05-07T07:57:21,510+01:00 INFO 63104 --- [example] [main] com.matt.example.ExampleApplication : Started ExampleApplication in 1.186 seconds (process running for 1.587)
matt
```

Console Tab:

```
2024-05-07T07:57:20,623+01:00 INFO 63104 --- [example] [main] com.matt.example.ExampleApplication : No active profile set, falling back to 1 default profile: 'default'
2024-05-07T07:57:21,233+01:00 INFO 63104 --- [example] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2024-05-07T07:57:21,251+01:00 INFO 63104 --- [example] [main] o.apache.catalina.core.StandardService : Starting service [tomcat]
2024-05-07T07:57:21,251+01:00 INFO 63104 --- [example] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.20]
2024-05-07T07:57:21,266+01:00 INFO 63104 --- [example] [main] o.e.c.c.C.[tomcat].[localhost].[] : Initializing Spring embedded WebApplicationContext
2024-05-07T07:57:21,267+01:00 INFO 63104 --- [example] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 610 ms
2024-05-07T07:57:21,503+01:00 INFO 63104 --- [example] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-05-07T07:57:21,510+01:00 INFO 63104 --- [example] [main] com.matt.example.ExampleApplication : Started ExampleApplication in 1.186 seconds (process running for 1.587)
matt
```

Using two files:



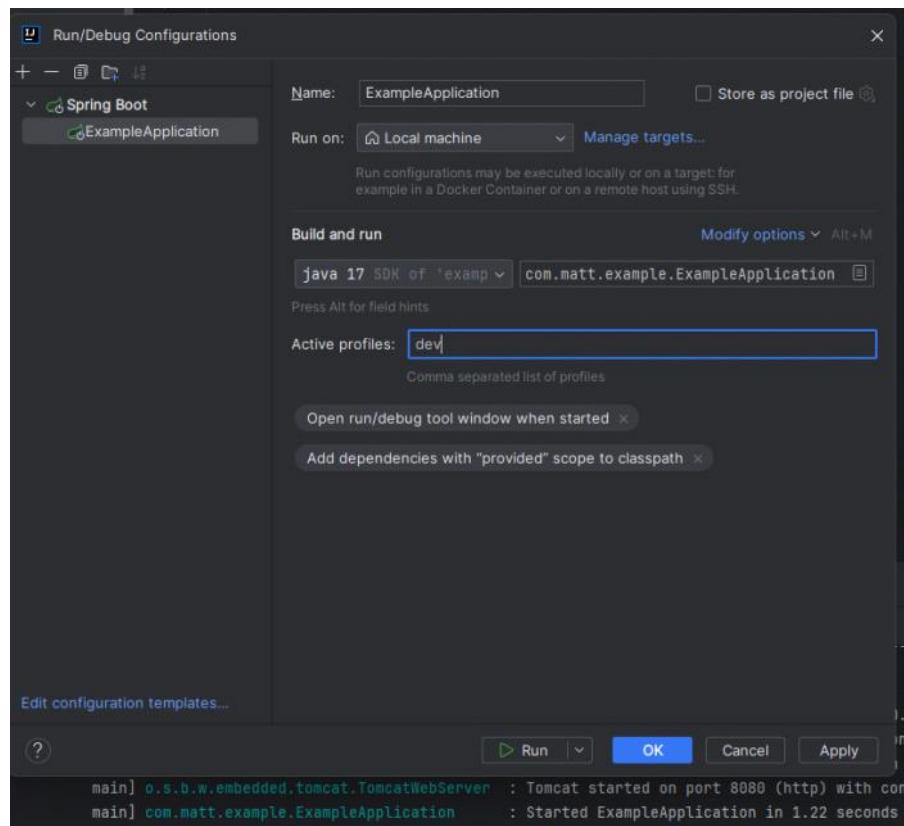
Code Editor (myFirstService.java):

```
4 import org.springframework.beans.factory.annotation.Qualifier;
5 import org.springframework.beans.factory.annotation.Value;
6 import org.springframework.context.annotation.PropertySource;
7 import org.springframework.context.annotation.PropertySources;
8 import org.springframework.stereotype.Service;
9
10 2 usages
11 @Service
12 @PropertySources({
13     @PropertySource("classpath:custom.properties"),
14     @PropertySource("classpath:custom-two.properties")
15 })
16 public class myFirstService {
17
18     2 usages
19     private final myFirstClass myFirstClass;
20     @Value("${my.prop}")
21     public String customProp;
22     @Value("${my.prop.2}")
23     public String customTwo;
24 }
```

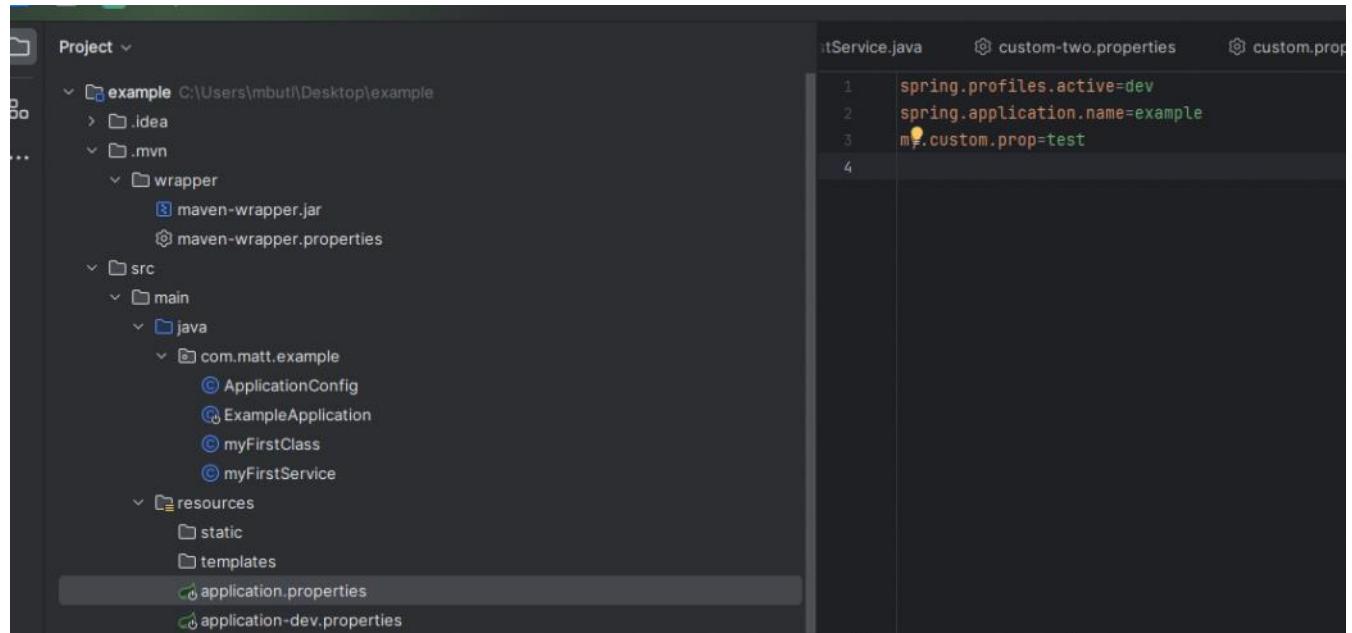
Spring Profiles

07 May 2024 08:16

Spring profiles allow you to activate and de-active beans depending on if it's a production environment or not



OR:



OR:

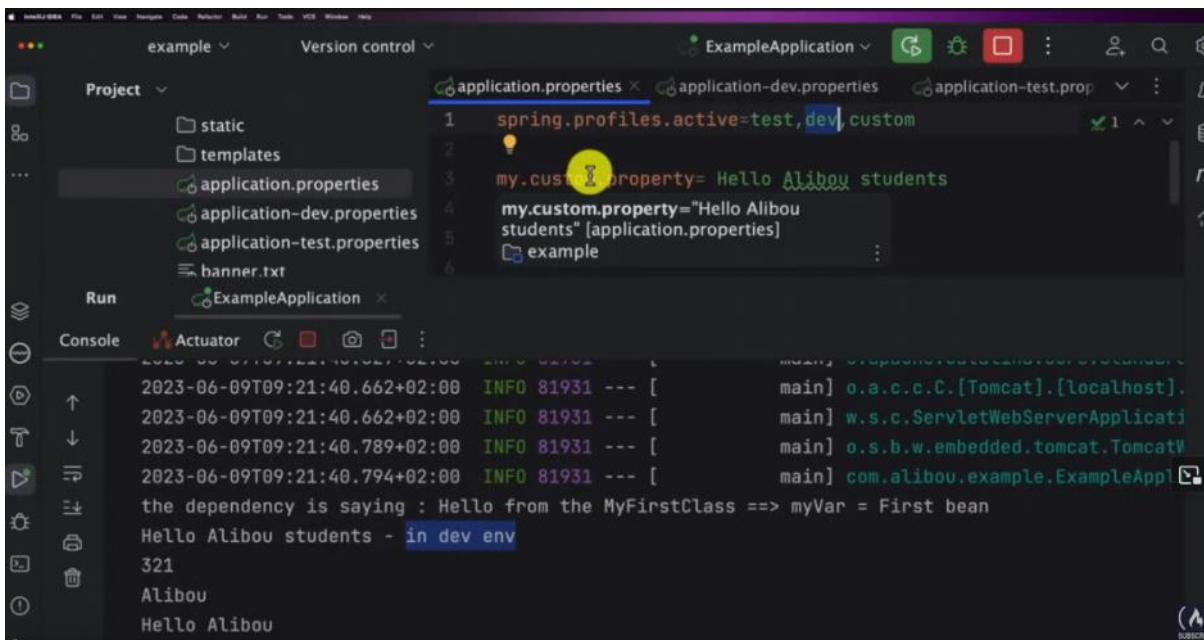
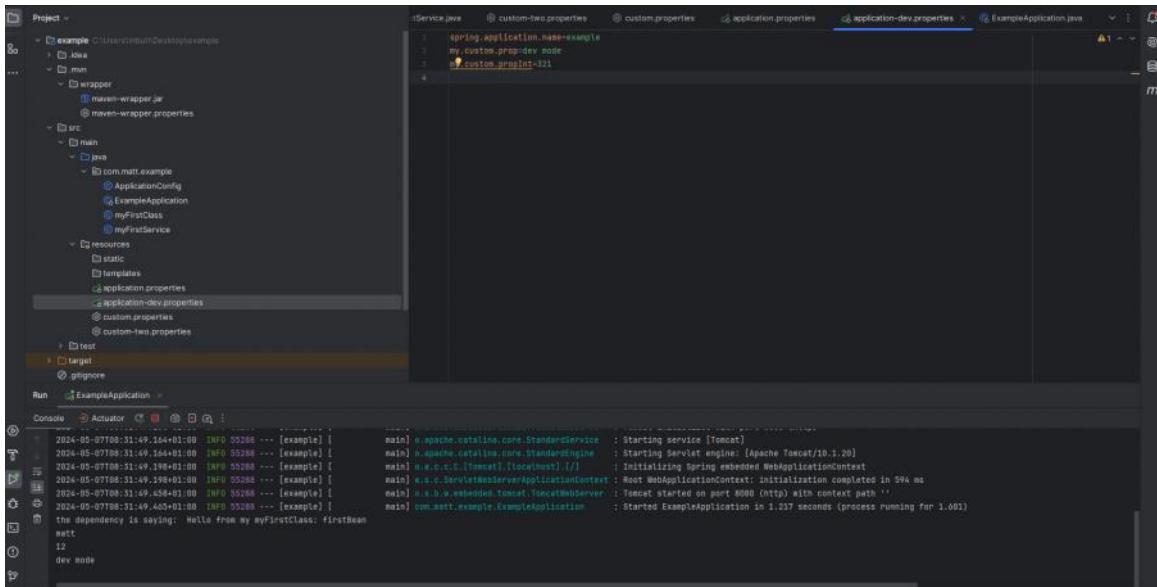
properties custom.properties application.properties application-test.properties application-dev.properties

```

2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 import java.util.Collections;
7
8  @SpringBootApplication
9  public class ExampleApplication {
10
11      public static void main(String[] args) {
12         var app = new SpringApplication(ExampleApplication.class);
13         app.setDefaultProperties(Collections.singletonMap("spring.profiles.active", "dev"));
14         var ctx = app.run(args);

```

Will use Dev named properties file (check screenshot)



The final profile in the list is the 'overriding' one

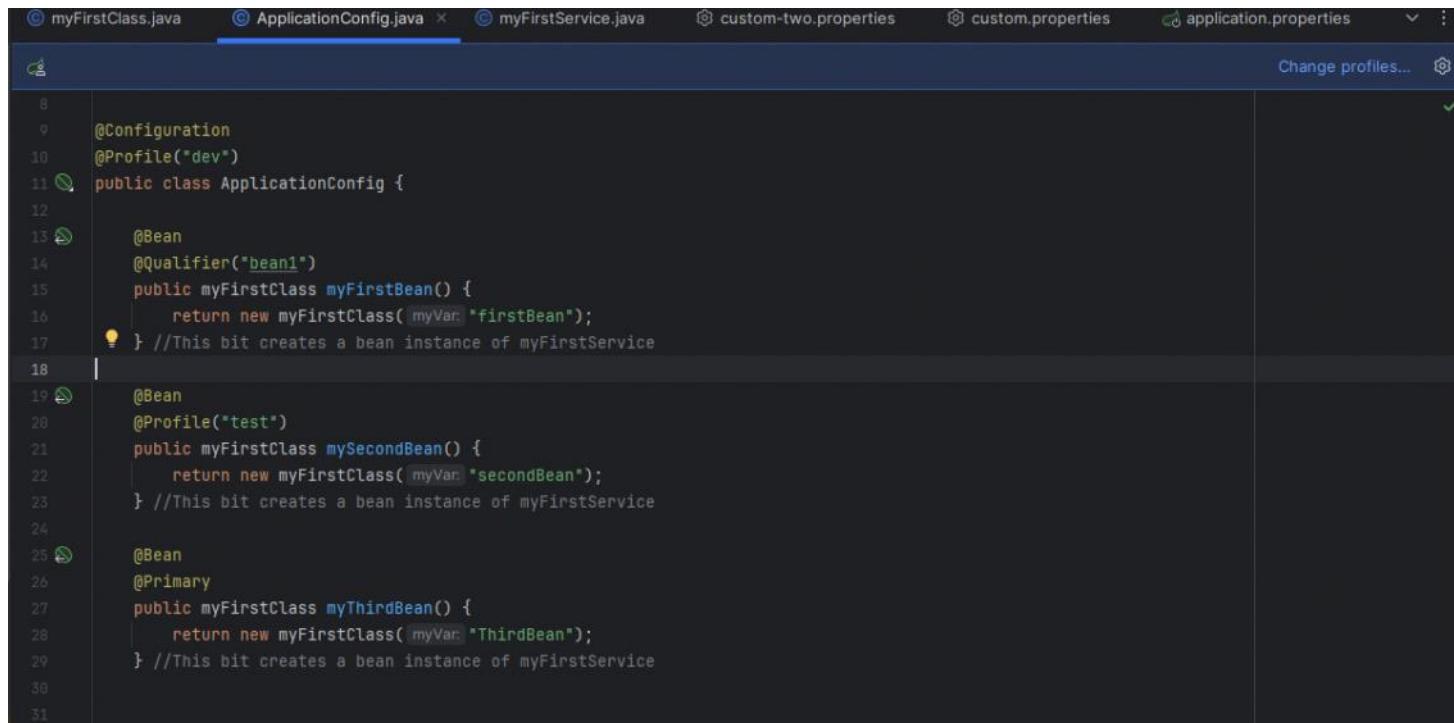
Profile available beans

07 May 2024 10:37

```
8
9
10 @Configuration
11 public class ApplicationConfig {
12     @Bean
13     @Qualifier("bean1")
14     @Profile("dev")
15     public myFirstClass myFirstBean() {
16         return new myFirstClass(myVar: "firstBean");
17     } //This bit creates a bean instance of myFirstService
18
19     @Bean
20     @Profile("test")
21     public myFirstClass mySecondBean() {
22         return new myFirstClass(myVar: "secondBean");
23     } //This bit creates a bean instance of myFirstService
24
25     @Bean
26     @Primary
27     public myFirstClass myThirdBean() {
28         return new myFirstClass(myVar: "ThirdBean");
29     } //This bit creates a bean instance of myFirstService
30
```

These beans are now only available on dev and test profiles respectively

Make a configuration profile dependent:



```
8
9
10 @Configuration
11 @Profile("dev")
12 public class ApplicationConfig {
13
14     @Bean
15     @Qualifier("bean1")
16     public myFirstClass myFirstBean() {
17         return new myFirstClass(myVar: "firstBean");
18     } //This bit creates a bean instance of myFirstService
19
20     @Bean
21     @Profile("test")
22     public myFirstClass mySecondBean() {
23         return new myFirstClass(myVar: "secondBean");
24     } //This bit creates a bean instance of myFirstService
25
26     @Bean
27     @Primary
28     public myFirstClass myThirdBean() {
29         return new myFirstClass(myVar: "ThirdBean");
30     } //This bit creates a bean instance of myFirstService
```

SPRING REST - OVERVIEW

- » REST = Representational State Transfer
- » REST was first defined by Roy Fielding in his 2000 Doctoral dissertation
- » Client-Server Architecture
- » Stateless
- » Cacheable
- » Layered System
- » Code on Demand (optional)
- » Uniform Interface
 - » Identification of resources
 - » Manipulation of resources through these representations
 - » Self-descriptive messages
 - » Hypermedia As The Engine Of Application State (HATEOS)



(🔍)
SUBSCRIBE

SPRING REST - OVERVIEW

- » The Web's architectural principles
- » Unique Identification of resources (URI)
- » Different resource representation
- » Hypermedia / Linking of resources
- » Stateless communication
- » Standard methods (GET, POST, PUT, DELETE) and responses (200 OK, 404 Not Found)

(🔍)
SUBSCRIBE

REST Resource Design

07 May 2024 11:20

You should always use Plural nouns for API endpoints

SPRING REST - RESOURCE DESIGN

- The resource should always be **plural nouns** in the API endpoint and if one instance resource should be retrieved, pass the id in the URL:
 - ➔ GET /accounts
 - ➔ GET /accounts/1
 - ➔ DELETE /accounts/2

You should use a standard URL endpoint with different functions integrated into it

- In case of nested resources (resources under a resource), the resources should be accessible as follows:
 - ➔ GET /accounts/1/payments/56
- Use the HTTP methods to specify what to do with this resource. With methods GET, POST, PUT, PATCH, DELETE you can provide CRUD functionality (Create, Read, Update, Delete)



To access a resource with nested context do as above

Always use correct HTTP methods for CRUD

Table example:

SPRING REST - RESOURCE DESIGN

- Use HTTP methods (aka "verbs") to specify what to do with this resource. With methods GET, POST, PUT, PATCH, DELETE you can provide CRUD functionality (Create, Read, Update, Delete)

Resource	GET	POST	PUT	DELETE
/accounts	Get all accounts	Create new account	Bulk update all accounts	Delete all accounts
/accounts/1	Get account with ID 1	Error	Update account with ID 1	Delete account with ID 1
/accounts/1/payments	Get all payments for account with ID 1	Create new Payment for account with ID 1	Bulk update all payments for account with ID 1	Delete all payments for account with ID 1



HTTP Methods

07 May 2024 11:28

- **GET:**
 - Retrieve data from the server.
 - Used for reading resources.
 - Idempotent (multiple identical requests will produce the same result).
- **POST:**
 - Submit data to the server.
 - Create a new resource or perform a custom action.
 - Not idempotent (multiple identical requests may result in different outcomes).
- **PUT:**
 - Update data on the server.
 - Typically used to update an existing resource with the provided data.
 - Idempotent (multiple identical requests will produce the same result).
- **PATCH:**
 - Partially update data on the server.
 - Used when you want to update only a part of the resource.
 - Typically used when a full resource update via PUT is not practical.
 - Not always guaranteed to be idempotent.
- **DELETE:**
 - Remove data from the server.
 - Used to delete a resource.
 - Idempotent (multiple identical requests will produce the same result).
- **OPTIONS:**
 - Get information about the communication options available for the target resource.
 - Used to describe the communication options for the target resource.
- **HEAD:**
 - Retrieve the headers for a resource.
 - Similar to GET but only retrieves the headers, not the actual content.
 - Useful for checking resource metadata without downloading the entire resource.

These methods are fundamental building blocks for interacting with RESTful APIs, each serving a specific purpose in the CRUD (Create, Read, Update, Delete) operations.

From <<https://chatgpt.com/c/c20759ce-54e7-403e-acd3-a7ed55823995>>

HTTP status codes

07 May 2024 11:30

Codes:

100 (Info):

200 (success):

200 - ok went as planned, data returned if needed

201 - Created, resource created

204 - ok went as planned, no data (delete, put, post)

300 (redirection):

304 - not modified, is cached version from client up to date, saves a lot bandwidth

400 (client error):

400 - bad request, bad client request form

401 - unauthorized, invalid creds or bad permissions etc

403 - bad permissions, can we used to hide details

500 (server error):

500 - internal server error (servers fault, no more info)

503 - Server unavailable, not available for a short amount of time, header includes how long to wait before retry

Rest example

07 May 2024 14:14

EXAMPLE

```
@RestController
public class PaymentRestController {

    @PostMapping(value = "/payments")
    public ResponseEntity<PaymentInformation> initiatePayment(
        @RequestBody PaymentRequest paymentRequest
    ) {

        // business code
        URI resultLocation = UriComponentsBuilder
            .fromPath("/payments/{id}")
            .buildAndExpand(confirmation.getId())
            .toUri();
        return ResponseEntity.created(resultLocation).body(confirmation);
    }
}
```

PostMapping - HTTP verb

ResponseEntity<PaymentInfomation> - Will return a ResponseEntity containing a PaymentInfomation instance

RequestBody PaymentRequest paymentRequest - A payment request body is expected with request

Ingore resultLocation

Return ResponseEntity.created() - is what sorts our response, created being HTTP code

ResponoseStatus

07 May 2024 14:21

Using `@ResponseStatus` you can set your response code

This is only used if no expectations are made

RestController

07 May 2024 14:36

Look at screenshot for details

```
1 package com.matt.example;
2
3 import org.springframework.http.HttpStatus;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.ResponseStatus;
6 import org.springframework.web.bind.annotation.RestController;
7
8 @RestController
9 public class FirstController {
10
11     @GetMapping("/hello")
12     public String sayHello() {
13         return "my first controller";
14     }
15     @GetMapping("/hello-2")
16     @ResponseStatus(HttpStatus.ACCEPTED)
17     public String sayHelloTwo() {
18         return "my second controller";
19     }
20 }
```

ResponseStatus only used with no exceptions

```
1 package com.matt.example;
2
3 import org.springframework.http.HttpStatus;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.PostMapping;
6 import org.springframework.web.bind.annotation.ResponseStatus;
7 import org.springframework.web.bind.annotation.RestController;
8
9 @RestController
10 public class FirstController {
11
12     @GetMapping("/hello")
13     public String sayHello() {
```

```
12     @GetMapping("/hello")
13     public String sayHello() {
14         return "my first controller";
15     }
16     @PostMapping("/post")
17     public String post(String message) {
18         return "request accepted and" + message;
19     }
20
21 }
```

Note request body used to specify input body requirements

Obviously we will normally use model but here's how to do it with variables

Because it's just a variable not a model we can use raw

'input' assigned as message

Rest Controller with JSON input

07 May 2024 14:54

```
1 package com.matt.example;
2
3     1 usage
4
5     public class Order {
6
7         3 usages
8         private String customerName;
9         3 usages
10        private String ProductName;
11        3 usages
12        private int quantity;
13
14        @Override
15        public String toString() {
16            return "Order{" +
17                "customerName='" + customerName + '\'' +
18                ", ProductName='" + ProductName + '\'' +
19                ", quantity=" + quantity +
20                '}';
21        }
22
23        no usages
24        public String getCustomerName() {
25            return customerName;
26        }
27    }
```

Custom To String method generated from intelj

@override is overriding the default Object parent class toString method

You must have accessors (getters and setters) to serialize and deserialize the data

```
15             return "request accepted and" + message;
16         }
17
18     @PostMapping("/post-order")
19     public String postOrder(@RequestBody Order order) {
20         return "request accepted and" + order.toString();
21     }
22 }
23 |
```

http://localhost:8080/post-order

POST http://localhost:8080/post-order

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Body

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 "customerName": "Alibou",
2 "productName": "iPhone",
3 "quantity": 1
```

Body Cookies Headers (5) Test Results

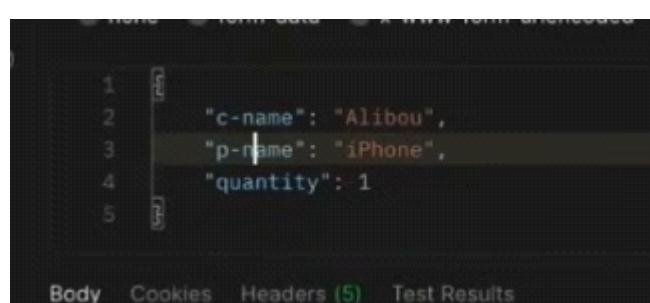
Pretty Raw Preview Visualize Text

1 Request accepted and order is : Order{customerName='Alibou', productName='iPhone', quantity=1}

Renaming request Json field compared to class

08 May 2024 05:34

```
3 import com.fasterxml.jackson.annotation.JsonProperty;
4
5     1 usage
6
7         public class Order {
8
9
10            3 usages
11                @JsonProperty("c-name")
12                private String customerName;
13
14            3 usages
15                @JsonProperty("p-name")
16                private String ProductName;
17
18            3 usages
19                @JsonProperty("quantity")
20                private int quantity;
21
22            @Override
23            public String toString() {
24                return "Order{" +
25                    "customerName='" + customerName + '\'' +
26                    ", ProductName='" + ProductName + '\'' +
27                    ", quantity=" + quantity +
28                    '}';
29            }
30        }
```



The screenshot shows a browser's developer tools Network tab with a captured POST request. The request body is a JSON object:

```
1 {
2     "c-name": "Alibou",
3     "p-name": "iPhone",
4     "quantity": 1
5 }
```

The tabs at the bottom of the developer tools are: Body, Cookies, Headers (5), Test Results.

JsonProperty allows you to rename fields

Java Records

08 May 2024 05:44

POJO classes are normally used for data transfer however, from version 14 records can be used

You can use java records as request body's in Spring

Records do not require you to make getters and setters to serialize and deserialize

The `toString` function formats the output correctly without needing an override

```
    }  
    }  
  
20 }  
21     @PostMapping(@GetMapping("/post-record"))  
22     @  
23     public String postRecord(@RequestBody OrderRecord record) {  
24         return "request accepted and" + record.toString();  
25     }  
26 }
```

Records vs POJO Classes

08 May 2024 05:55

Classes:

Requires `ToString` override

Requires setters and getters to serialize and deserialize

Can have logic within the class

Records:

Records are final so they can't be used with mutable requirement library

A better choice for simple data transfer with no logic

Path variables & request params

08 May 2024 05:58

Path variables allows us to do things like

/hello/Matt (username)

If URL path name is the same as variable name:

```
24
25
26     @GetMapping("/hello/{username}")
27     public String pathVar(@PathVariable String username) {
28
29         return "my value = " + username;
30     }
```

Use parameter if not

```
25
26     @GetMapping("/hello/{user-name}")
27     public String pathVar(@PathVariable("user-name") String username) {
28
29         return "my value = " + username;
30     }
```

This way is recommended for dash readability

Request Params:

Request param:

/hello/?paramName=param_value¶m_value_2=value_2

```
24
25
26     @GetMapping("/helloParams")
27     public String pathVar(@RequestParam("param_value") String ParamValue, @RequestParam("param_value_2") String ParamValue2) {
28
29         return "my value = " + ParamValue + ParamValue2;
30     }
31
32 }
```

POST http://localhost:8080/ ● POST http://localhost:8080/ ● POST http://localhost:8080/ ● GET http://localhost:8080/l ●

http://localhost:8080/hello?user-name=alibou&user-lastname=Bouali

GET http://localhost:8080/hello?user-name=alibou&user-lastname=Bouali

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/> user-name	alibou		
<input checked="" type="checkbox"/> user-lastname	Bouali		

Body Cookies Headers (S) Test Results

Pretty Raw Preview Visualize Text

my value = alibou Bouali

Status: 200 OK Time: 71 ms Size: 190 B Save Response

(A) SUBSCRIBE

Path Variables vs Request Params

08 May 2024 06:16

Path Variables:

Used to identify resources as part of the URL

Request Params:

Used to extract query params

```
IntelliJ IDEA 2023.1.3  
File Edit View Navigate Code Refactor Build Run Tools VCS Windows Help  
example < Version control < ExampleApplication <   
FirstController.java <   
39  
40  
...  
41     // http://localhost:8080/hello/alibou  
42     @GetMapping("/{user-name}")  
43     public String pathVar(  
44         @PathVariable("user-name") String userName  
45     ) {  
46         return "my value = " + userName;  
47     }  
48  
49     // http://localhost:8080/hello?param_name=paramvalues&param_name_2=value_2  
50     @GetMapping("/hello")  
51     public String paramVar(  
52         @RequestParam("user-name") String userName,  
53         @RequestParam("user-lastname") String userLastname  
54     ) {  
55         return "my value = " + userName + " " + userLastname;  
56     }  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2069  
2070  
2071  
2072  
2073  
2074  
2075  

```

@RequestController, @PostMapping, @GetMapping

08 May 2024 06:20

```
1 package com.matt.example;
2
3 import org.springframework.http.HttpStatus;
4 import org.springframework.web.bind.annotation.*;
5
6 @RestController
7 public class FirstController {
8
9     @GetMapping("/hello")
10    public String sayHello() {
11        return "my first controller";
12    }
13
14    @PostMapping("/post")
15    public String post(@RequestBody String message) {
16        return "request accepted and" + message;
17    }
18    @PostMapping("/post-order")
19    public String postOrder(@RequestBody Order order) {
20        return "request accepted and" + order.toString();
21    }
22    @PostMapping("/post-record")
23    public String postRecord(@RequestBody OrderRecord record) {
24        return "request accepted and" + record.toString();
25    }
26 }
```

Spring will scan for Rest Controllers components and create beans of them

```
8
9     @GetMapping("/hello")
10    public String sayHello() {
11        return "my first controller";
12    }
13
14    @PostMapping("/post")
15    public String post(@RequestBody String message) {
16        return "request accepted and" + message;
17    }
18    @PostMapping("/post-order")
19    public String postOrder(@RequestBody Order order) {
20        return "request accepted and" + order.toString();
21    }
22    @PostMapping("/post-record")
23    public String postRecord(@RequestBody OrderRecord record) {
24        return "request accepted and" + record.toString();
25    }
26 }
```

Various mappings are created by Spring because of Post, Get mapping annotations

done by RequestMappingHandlerMapping which builds a registry of all your URL's

Diagram explainer

08 May 2024 06:39

Dispatcher Servlet - dispatches requests and forwards them to Handler Mapping

Handler Mapping - Consults the Mapping Registry to find correct controller, returns found controller (all handles Methods, pathVariables, Params etc)

Controller - Handles request and contacts required business logic

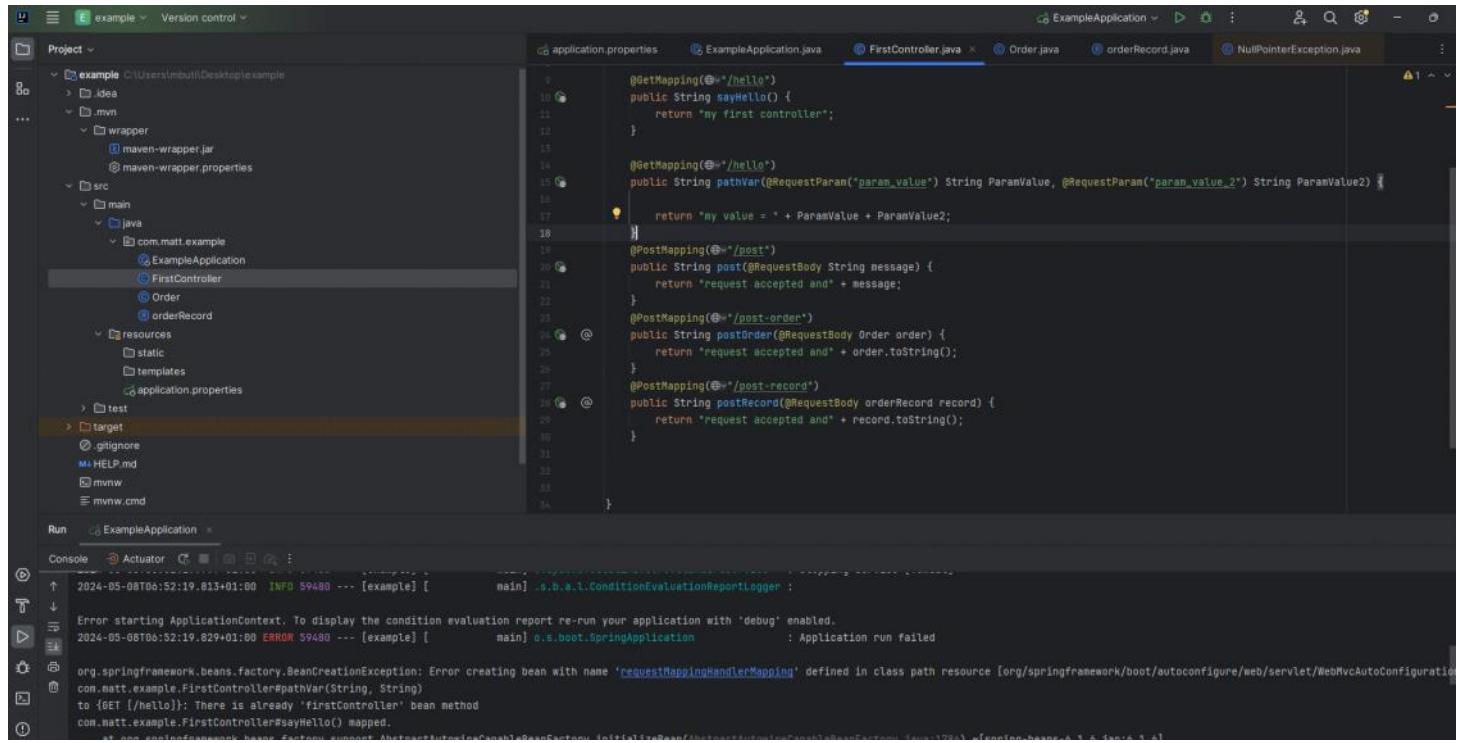
Business logic - Does "service" then returns response to controller

Controller sends responses to servlet dispatcher

Servlet dispatcher - sends response back to client

URL uniqueness

08 May 2024 06:52



The screenshot shows an IDE interface with several tabs and panes. The top tab bar includes 'application.properties', 'ExampleApplication.java', 'FirstController.java' (which is the active tab), 'Order.java', 'orderRecord.java', and 'NullPointerException.java'. The left pane displays the project structure under 'example' (C:\Users\mbutti\Desktop\example). The 'src' folder contains 'main' and 'java' subfolders. 'java' contains 'com.matt.example' which has 'FirstController', 'Order', and 'orderRecord' classes. 'resources' contains 'static' and 'templates' folders, and 'application.properties'. The right pane shows the code for 'FirstController.java'. The code defines four methods: a GET mapping for '/hello' returning 'my first controller'; a GET mapping for '/hello' with path variables (@RequestParam("param_value") String ParamValue, @RequestParam("param_value_2") String ParamValue2) returning 'my value = ' + ParamValue + ParamValue2; a POST mapping for '/post' returning 'request accepted and' + message; and a POST mapping for '/post-order' taking an Order object (@RequestBody Order order) and returning 'request accepted and' + order.toString(). The 'Console' tab at the bottom shows application logs, including an error message about a bean creation exception for 'requestMappingHandlerMapping' due to a duplicate mapping for '/hello'.

Only URL variables make a controller unique not parameters (check screenshot)

The path not the method name makes a controller unique

Spring data JPA dependencies

08 May 2024 06:55

Spring Data JPA

```
    </dependency>
@①    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
ect > dependencies > dependency
```

PostgresSQL

```
23
24 @①    </dependency>
25    <dependency>
26        <groupId>org.postgresql</groupId>
27        <artifactId>postgresql</artifactId>
28        <scope>runtime</scope>
29    </dependency>
```

Application.yaml (formally application.Properties):

```
1  spring:
2   datasource:
3     url: jdbc:postgresql://localhost:5432/demo_db
4     username: postgres
5     password: Buzz0001
6     driver-class-name: org.postgresql.Driver
7   jpa:
8     hibernate:
9       ddl-auto: create
10      show-sql: true
11      properties:
12        hibernate:
13          format_sql: true
14        database: postgresql
15
```

Demo_db being our database

Ignore jpa config, the rest is good though and is self explanatory

Data storage Diagram

08 May 2024 07:53

Entity - A java object that is designed to be saved in a database

All entities must have an empty constructor

Each entity is a table in a database

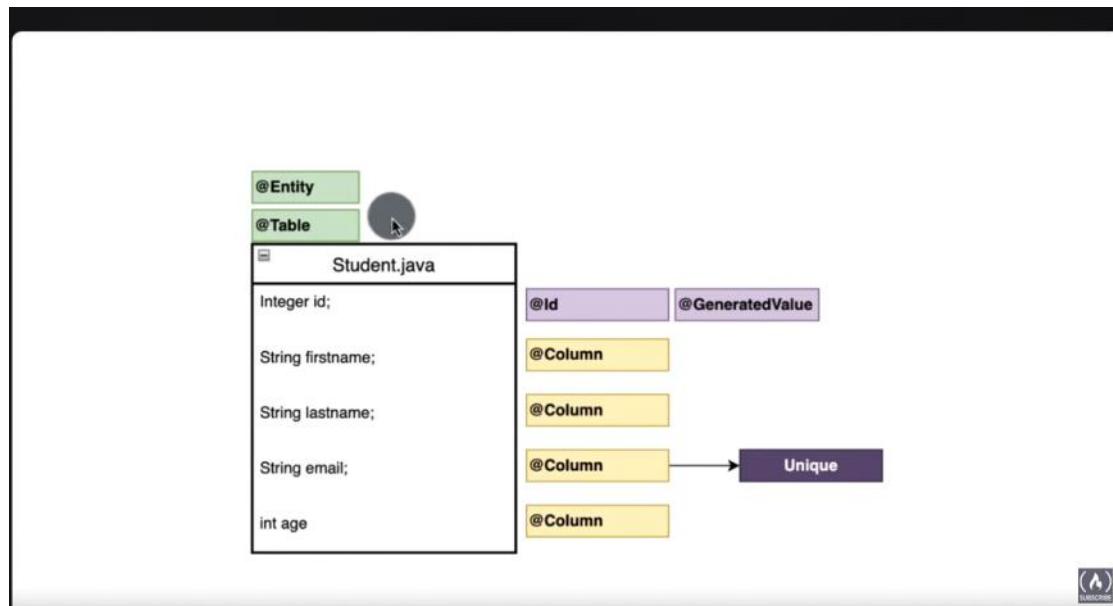
Columns - Each field in your model is a field

Row - the data

Every Entity must contain an ID that is a unique value

The table name is by default the class name but we set it with @table("NAME")

Spring Data then generates SQL queries on these entities



Spring Data Entity

08 May 2024 08:04

```
 5  S usages
 6  @Entity
 7  └─ @Table(name = "students")
 8
 9  └─ @Id
10  └─ @GeneratedValue
11  ┌─ private Integer id;
12
13  2 usages
14  └─ @Column(
15  └─   name = "c_fname",
16  └─   length = 20
17  ┘)
18  ┌─ private String firstname;
19  2 usages
20  ┌─ @Column(unique = true)
21  ┌─ private String email;
22  2 usages
23  ┌─ private int age;
24
25  └─ )
26  ┌─ private String some_column;
27  ┌─ public Student() {
28
29  ┘}
30
31
32
```

Must have an empty constructor

@Entity - A java object that is designed to be saved in a database

@Table - sets a table name, by default class name

@Column - lots of different features (use lookup) examples in screenshot. Remember each field is a different column

@Column(unique=true) - forces this field to be unique per user

@Column(updatable=false) - means field cannot be modified

@Column(insertable=false) - means field cannot be inserted

@Id ensure a unique primary key is made (required)

@Id

@GeneratedValue - will generate an id for our object that is completely unique

For all of these, use lookup for more info

Spring Data Repository

08 May 2024 10:46

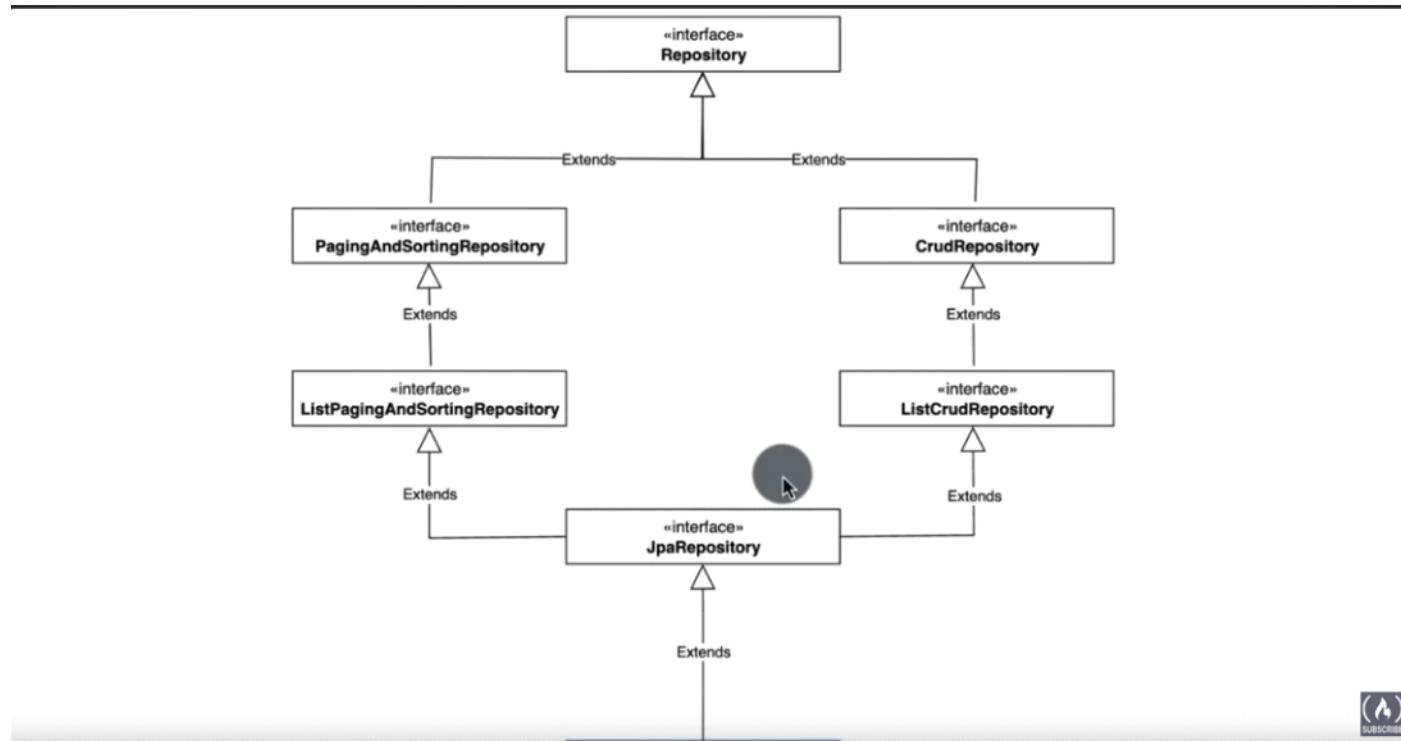
The Spring data JPA repository extends various different interfaces creating a lot of different functionality

These various interfaces, that you can choose, have functions like data save, data find, data delete. Allowing you to interact with your database without SQL

The Spring JPA Repositories interact contains a lot of different method signatures with SQL functionality (findOneByID)

Spring JPA repositories also allow you to join together Method signatures for queries

Spring JPA Repositories also allow you to create custom queries using @Query annotation in JPQL or SQL to a custom DTO



Student Repository

08 May 2024 10:56

```
2 usages
6 @Repository
7 public interface StudentRepository extends JpaRepository<Student, Integer> {
8
9 }
10
```

JPA Repositories takes in T (Object Type) and the type of the objects ID

```
5
6 @RestController
7 public class FirstController {
8     2 usages
9         private final StudentRepository repository;
10
11     @Autowired
12     public FirstController(StudentRepository repository) {
13         this.repository = repository;
14     }
15
16     @PostMapping("/students")
17     public Student post(@RequestBody Student student) {
18         return repository.save(student);
19     }
20 }
```

Insomnia

Application File Edit View Window Tools Help

Scratch Pad

POST http://localhost:8080/students

Send 200 OK 5.69 ms 90 B Just Now

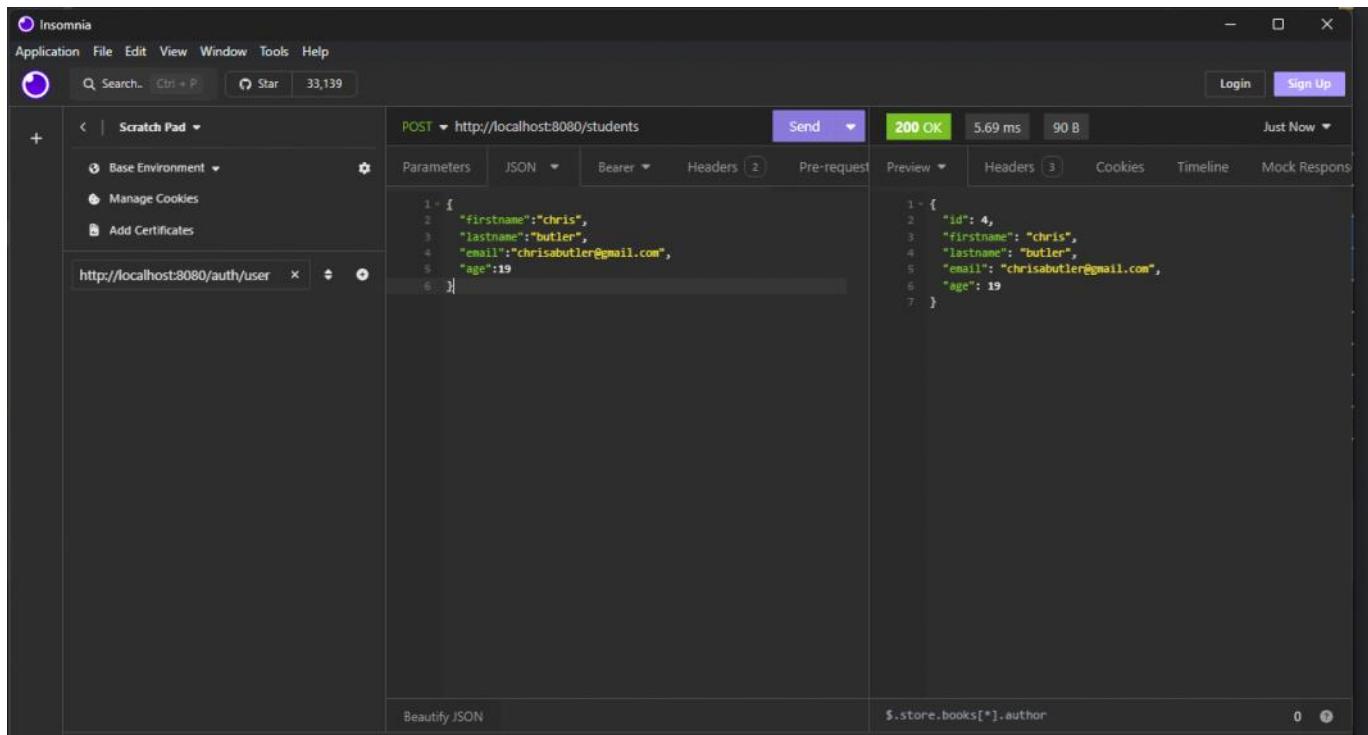
Parameters JSON Headers Pre-request Preview Headers Cookies Timeline Mock Response

Base Environment Manage Cookies Add Certificates

http://localhost:8080/auth/user

```
1: {
2:   "id": 4,
3:   "firstname": "chris",
4:   "lastname": "butler",
5:   "email": "chrisabutler@gmail.com",
6:   "age": 19
7: }
```

Beautify JSON \$.store.books[*].author 0 0



REST crud functions + OrElse

08 May 2024 11:28

```
1 package com.matt.example;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 import java.util.List;
7
8 2 usages
9 @Repository
10 public interface StudentRepository extends JpaRepository<Student, Integer> {
11     1 usage
12     List<Student> findAllByFirstname(String name);
13 }
14
```

Build queries with a method signatures like above

```
10
11 @RestController
12 public class FirstController {
13     6 usages
14     private final StudentRepository repository;
15
16     @Autowired
17     public FirstController(StudentRepository repository) {
18         this.repository = repository;
19     }
20
21     @PostMapping("/students")
22     public Student post(@RequestBody Student student) {
23         return repository.save(student);
24     }
25
26     @GetMapping("/students")
27     public List<Student> findAllStudent() {
28         return repository.findAll();
29     }
30     @GetMapping("/students/{student-id}")
31     public Student findStudentById(@PathVariable("student-id") int studentId) {
32         return repository.findById(studentId).orElse(new Student());
33     }
34     @GetMapping("/students/search/{student-name}")
35     public List<Student> findStudentByName(@PathVariable("student-name") String studentName) {
36         //Student[] students = {};
37         return repository.findAllByFirstname(studentName);
38     }
39
```

```

36         return repository.findAllByFirstname(studentName);
37     }
38
39     @DeleteMapping("/students/{student-id}")
40     @ResponseStatus(HttpStatus.OK) //Sets response code
41     public void delete(@PathVariable("student-id") Integer studentId) {
42         repository.deleteById(studentId);

```

The screenshot shows the Insomnia REST Client interface. A GET request is made to `http://localhost:8080/students/search/chris`. The response is `200 OK` with a response time of `101 ms` and a size of `92 B`, timestamped `1 Minute Ago`. The JSON response is displayed as:

```

1: [
2:   {
3:     "id": 1,
4:     "firstname": "chris",
5:     "lastname": "butler",
6:     "email": "chrisabutler@gmail.com",
7:     "age": 19
8:   }
9: ]

```

```
return repository.findById(studentId).orElse( other: null);
```

.OrElse function - can be used on any optional object

Repository.save() returns saved values

```

38
39     @DeleteMapping("/students/{student-id}")
40     @ResponseStatus(HttpStatus.OK) //Sets response code
41     public void delete(@PathVariable("student-id") String studentId) {
42         repository.deleteStudentById(studentId);
43     }
44
45
46

```

ResponseStatus set given no exceptions

Spring Relationships

08 May 2024 12:18

Two Relationship Example:

One to one Student to StudentProfile

```
1 usage
6   @Entity
7   public class StudentProfile {
8
9     @Id
10    @GeneratedValue
11    private String id;
12    3 usages
13    @
14    private String bio;
15
16    no usages
17    @OneToOne
18    @JoinColumn(
19      name = "student_id"
20    )
21    private Student student; //mapped by will look for this
22
23    no usages
24    @
25    public StudentProfile(String bio) {
26      this.bio = bio;
27    }
28
29    no usages
30    public String getId() {
31      return id;
32    }
```

```

9 usages
5   @Entity
6   @Table(name = "students")
7   public class Student {
8
9     @Id
10    @GeneratedValue
11    private Integer id;
12
13    2 usages
14    @Column(
15      name = "c_fname",
16      length = 20
17    )
18    private String firstname;
19    2 usages
20    @Column(unique = true)
21    private String email;
22    2 usages
23    private int age;
24
25    no usages
26    @OneToOne(mappedBy = "student",
27    cascade = CascadeType.ALL) //Cascade ='s what happens on delete
28    private StudentProfile studentProfile; //One to one relationship
29

```

@OneToOne - create a one to one relationship:

mappedBy = "student" - looks for the correct corresponding field in another object (in this case school) - MUST BE SPELT SAME AS FIELD NAME

Cascade = CascadeType.ALL

One to Many Student to School

```
3 usages
10 @Entity
11 public class School {
12
13     @Id
14     @GeneratedValue
15     private String id;
16
17     2 usages
18     private String name;
19
20     2 usages
21     @OneToOne(
22         mappedBy = "school"
23     )
24     private List<Student> students;
```

```
6     @Table(name = "students")
7     public class Student {
8
9         @Id
10        @GeneratedValue
11        private Integer id;
12
13         2 usages
14         @Column(
15             name = "c_fname",
16             length = 20
17         )
18         private String firstname;
19         2 usages
20         @ManyToOne
21         @JoinColumn(
22             name = "school_id"
23         )
24         private School school;
```

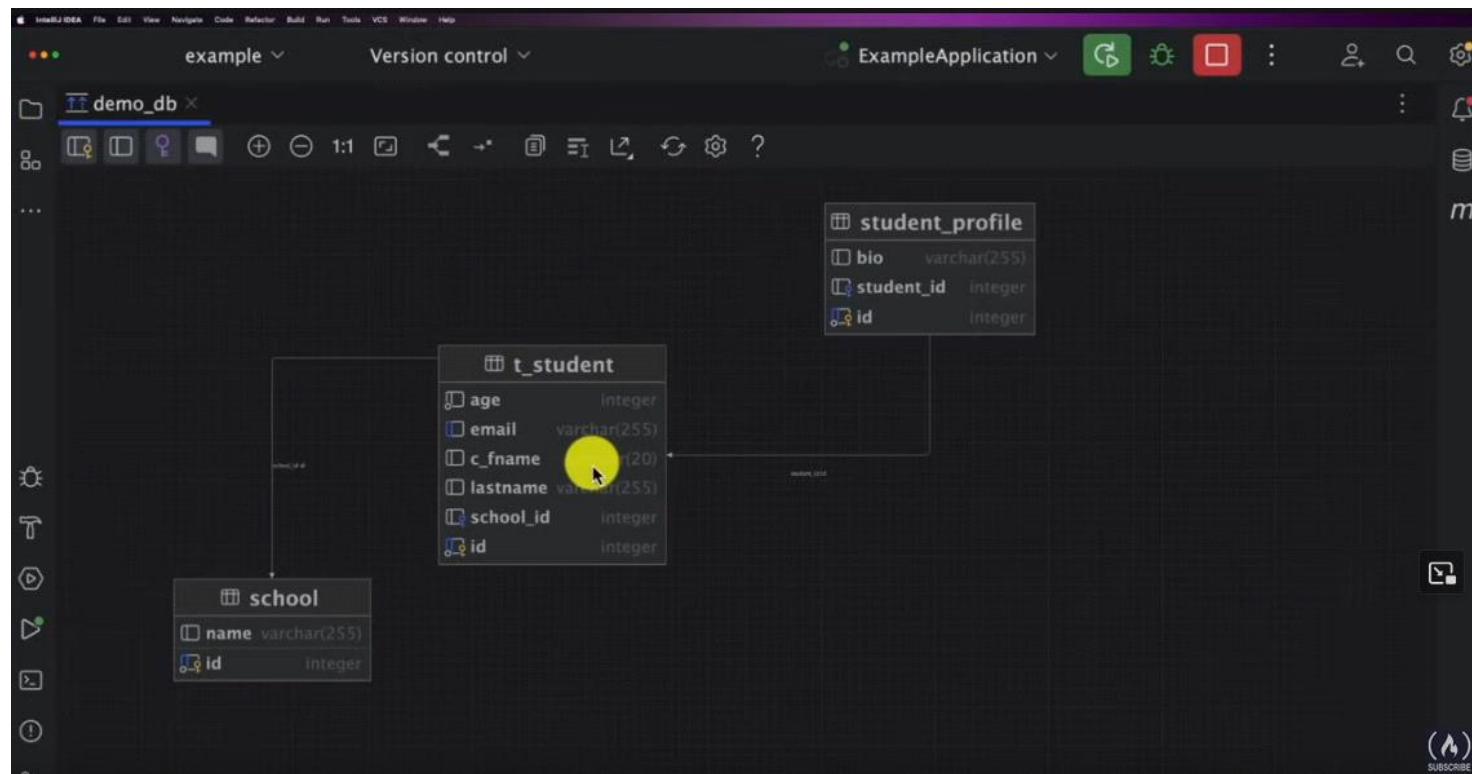
@JoinColumn - creates a column to store the ID of the related object
Join Column should be given a name and added to the secondary object (child)

So Student has it for Schools

And StudentProfile has it for Student

Checking relationships

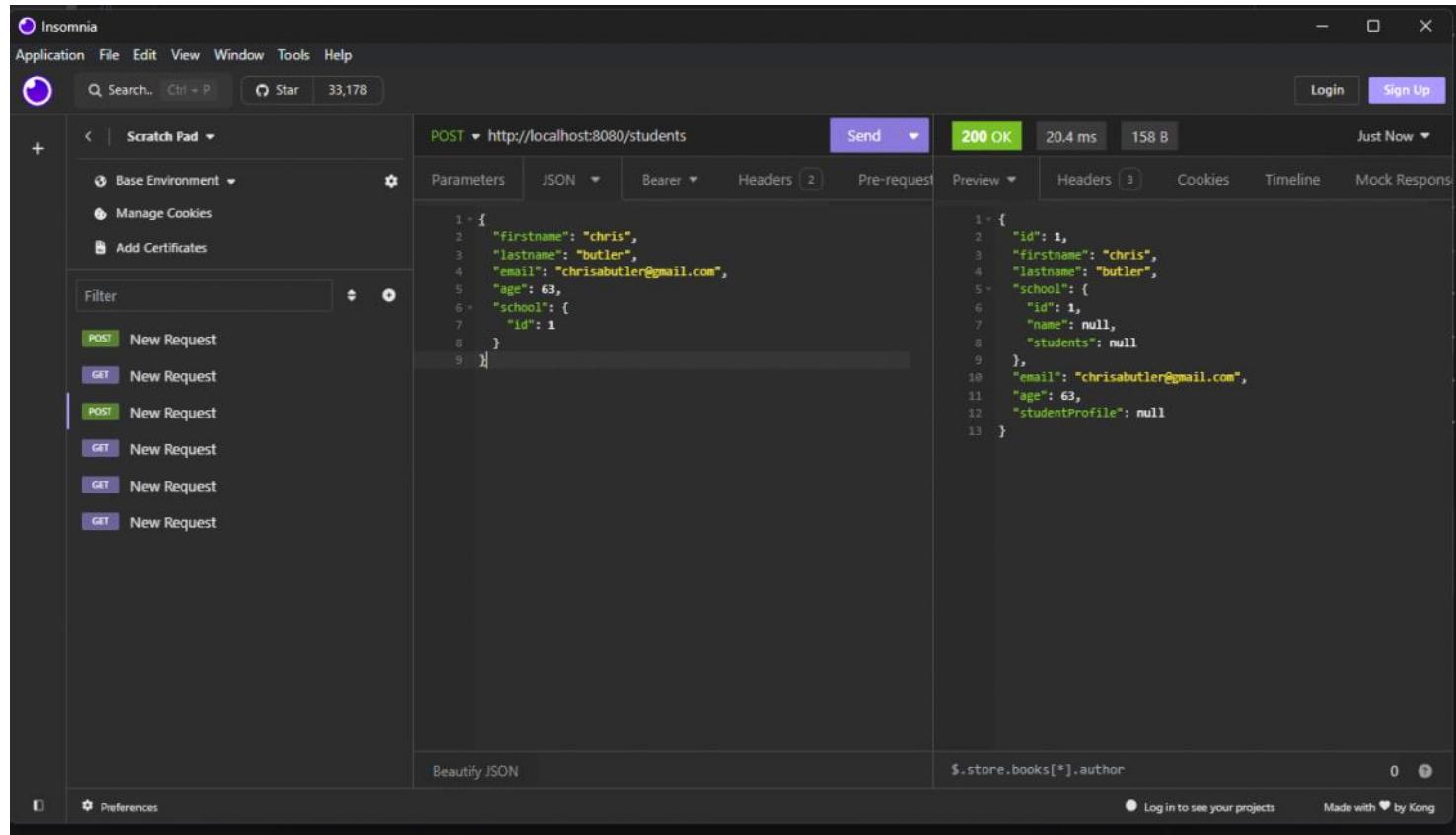
08 May 2024 13:39



Inteji Show Diagrams

Relationships working Screenshots

08 May 2024 14:00



POST <http://localhost:8080/students>

```
1: {
2:   "firstname": "chris",
3:   "lastname": "butler",
4:   "email": "chrisabutler@gmail.com",
5:   "age": 63,
6:   "school": {
7:     "id": 1
8:   }
9: }
```

```
1: {
2:   "id": 1,
3:   "firstname": "chris",
4:   "lastname": "butler",
5:   "school": {
6:     "id": 1,
7:     "name": null,
8:     "students": null
9:   },
10:  "email": "chrisabutler@gmail.com",
11:  "age": 63,
12:  "studentProfile": null
13: }
```

```
3 usages
6 @Entity
7 public class StudentProfile {
8
9     @Id
10    @GeneratedValue
11    private String id;
12    @
13    private String bio;
14
15    no usages
16    @OneToOne
17    @JoinColumn(
18        name = "student_id"
19    )
20    private Student student; //mapped by will look for this
```

```
9
10    8 usages
11    @Entity
12    public class School {
13
14        @Id
15        @GeneratedValue
16        private Integer id;
17
18        2 usages
19        private String name;
20
21        2 usages
22        @OneToMany(
23            mappedBy = "school"
24        )
25        private List<Student> students;
26
27
28        public School() {
```

```
12 usages
5  @Entity
6  @Table(name = "t_student")
7  public class Student {
8
9      @Id
10     @GeneratedValue
11     private Integer id;
12
13
14     2 usages
14     private String firstname;
15     2 usages
15     private String lastname;
16     2 usages
17     @ManyToOne
18     @JoinColumn(
19         name = "school_id"
20     )
20     private School school;
21
```

```
25
25     2 usages
26     @OneToOne(mappedBy = "student",
27     cascade = CascadeType.ALL) //Cascade ='s what happens on delete
28     private StudentProfile studentProfile; //One to one relationship
29
30     public Student() {
```

StudentProfile bond Student

Student bond School

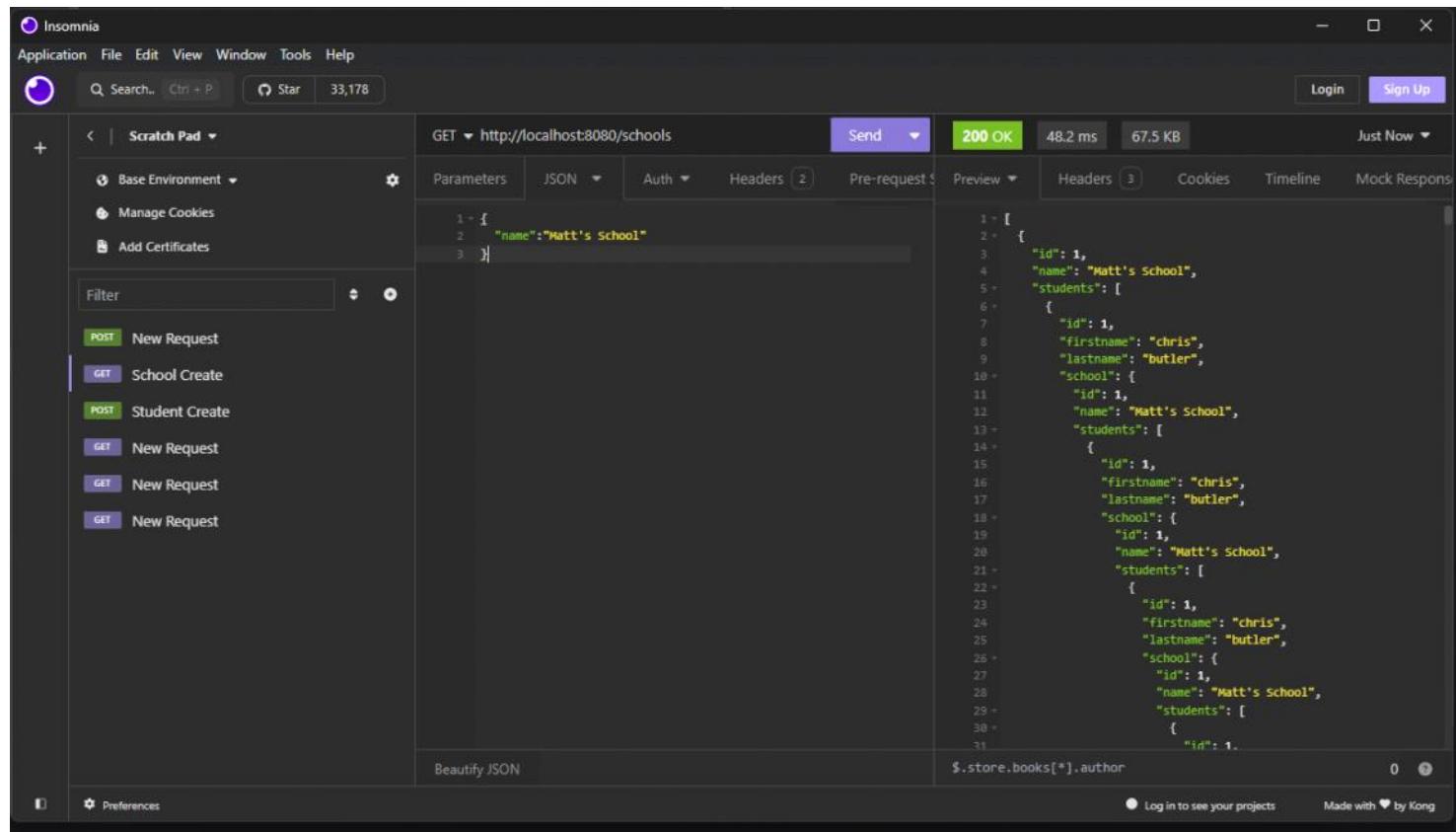
School (One to many)

Student to school (Many to one)

StudentProfile to student (One to One)

Relationship looping @JsonManagedReference

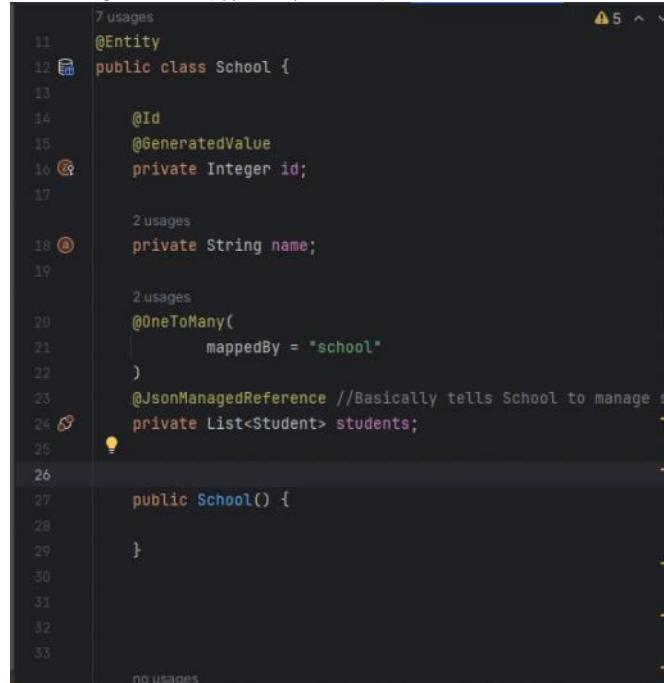
09 May 2024 05:38



The screenshot shows the Insomnia REST client interface. In the top navigation bar, 'Scratch Pad' is selected. Below it, a request is being made to 'http://localhost:8080/schools' via a GET method. The 'Parameters' tab contains a JSON object: { "name": "Matt's School" }. The 'Send' button is highlighted in purple. The response status is '200 OK' with a duration of '48.2 ms' and a size of '67.5 KB'. The response body is a deeply nested JSON structure representing a self-referencing relationship, with many lines of code numbered from 1 to 31. The response ends with '\$.store.books[*].author'.

When request objects that are in a relationship with each other you can get stuck in a loop

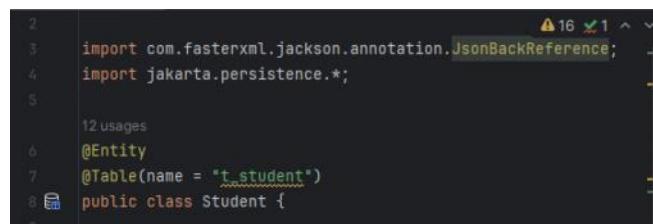
@JsonManagedReference (applied to parent field)



```
7 usages
11 @Entity
12 public class School {
13
14     @Id
15     @GeneratedValue
16     private Integer id;
17
18     2 usages
19     private String name;
20
21     2 usages
22     @OneToMany(
23         mappedBy = "school"
24     )
25     @JsonManagedReference //Basically tells School to manage s
26     private List<Student> students;
27
28     public School() {
29
30     }
31
32
33     no usages

```

Basically tells School to manage serializing the child and not to allow child to serialize parent



```
2
3     import com.fasterxml.jackson.annotation.JsonBackReference;
4     import jakarta.persistence.*;
5
6     12 usages
7     @Entity
8     @Table(name = "t_student")
9     public class Student {

```

```

2
3 import com.fasterxml.jackson.annotation.JsonBackReference;
4 import jakarta.persistence.*;
5
6
7 12 usages
8 @Entity
9 @Table(name = "t_student")
10 public class Student {
11
12     @Id
13     @GeneratedValue
14     private Integer id;
15
16     2 usages
17     @Column(name = "firstname")
18     private String firstname;
19
20     2 usages
21     @Column(name = "lastname")
22     private String lastname;
23
24     2 usages
25     @ManyToOne
26     @JoinColumn(
27         name = "school_id"
28     )
29     @JsonBackReference
30     private School school;
31
32     2 usages
33     @Column(unique = true)
34     private String email;
35
36     2 usages

```

@JsonBackReference (applied to child field)

Basically tells Student to not attempt to serialize School

The screenshot shows the Insomnia REST Client interface. A successful GET request to `http://localhost:8080/schools` has been made. The response is a JSON object containing an array of schools. Each school has an ID, a name, and a list of students. Each student has an ID, a first name, a last name, an email, an age, and a student profile.

```

1 [
2   {
3     "id": 1,
4     "name": "Matt's School",
5     "students": [
6       {
7         "id": 2,
8         "firstname": "chris",
9         "lastname": "butler",
10        "email": "chrisbutler@gmail.com",
11        "age": 63,
12        "studentProfile": null
13      },
14      {
15        "id": 3,
16        "firstname": "matt",
17        "lastname": "butler",
18        "email": "mattbutler001@gmail.com",
19        "age": 63,
20        "studentProfile": null
21      }
22    ]
23  ]

```

DTO Pattern, Representations

09 May 2024 05:53

The goal of a DTO is to encapsulate and transfer data between systems

DTO only includes simple data fields, doesn't have behaviours of model it represents

We use a mapper to prevent sensitive information from being dispatched when it shouldn't be.

Mappers have multiple representations, only expose X fields

We can also use different representations for read and write

This ensures:

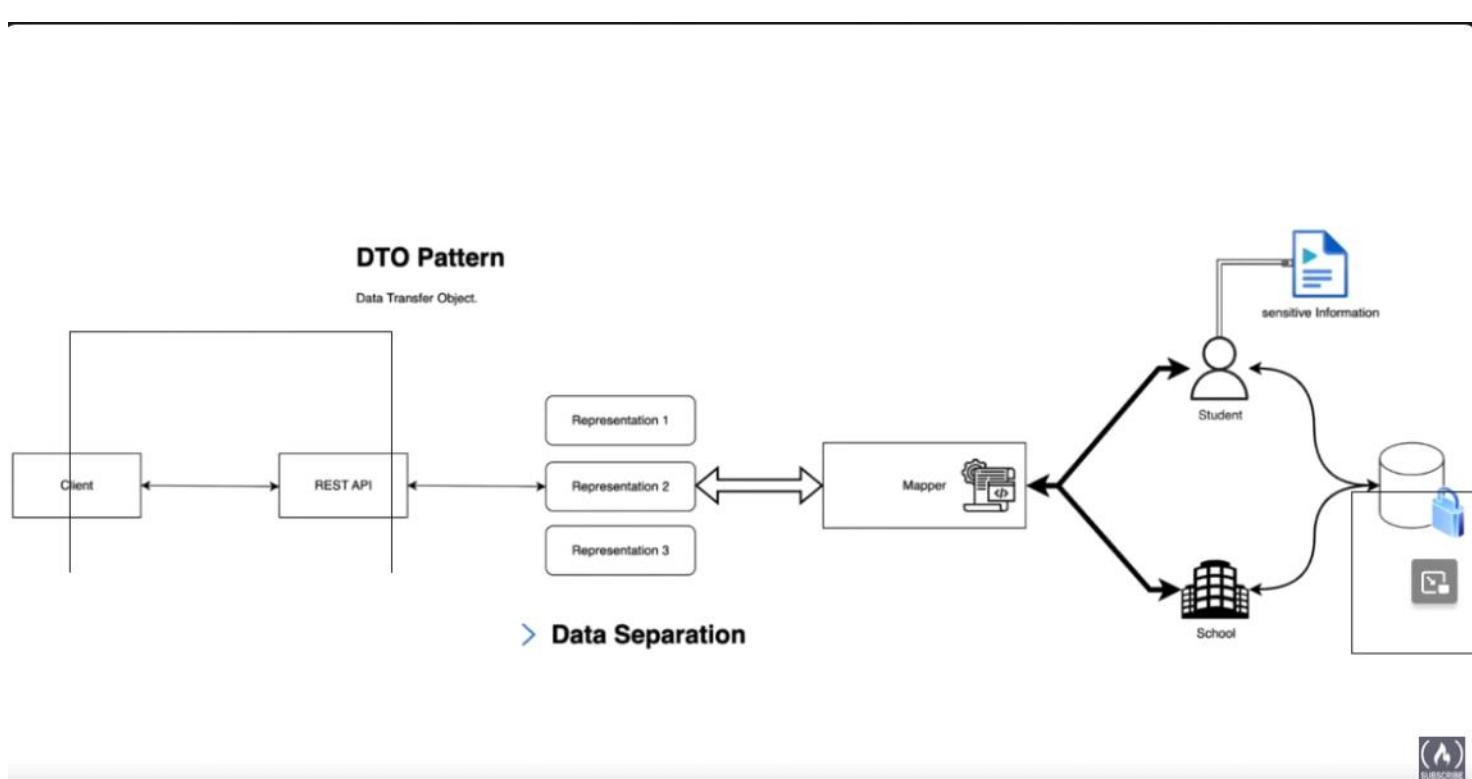
Data separation - only gives required data

Abstractions layer - gives you a clear representation of what will be sent

Performance - instead of sending lots of data, we only send what is required over network

Flexibility - allows you to only give the client what your client needs

Versioning - allows you to support multiple versions of the same API



DTO in action

09 May 2024 06:13

```
 ① @PostMapping("/students")
 ② public StudentResponseDTO post(@RequestBody StudentDTO student) { // 
    ③     Student dto = toStudent(student);
    ④     Student savedStudent = repository.save(dto); //Uses studentDTO (reduced input)
    ⑤     return ToStudentResponseDTO(savedStudent); //response with studentResponseDTO (reduced output)
 ⑥ }

 ⑦ 1 usage
 ⑧ private Student toStudent(StudentDTO dto) {
    ⑨     var student = new Student();
    ⑩     student.setFirstname(dto.getFirstname());
    ⑪     student.setLastname(dto.getLastname());
    ⑫     student.setEmail(dto.getEmail());
    ⑬     var school = new School();

    ⑭     school.setId(dto.getSchoolId());

    ⑮     student.setSchool(school);

    ⑯     return student;
 ⑰ }

 ⑷ 1 usage
 ⑸ private StudentResponseDTO ToStudentResponseDTO(Student student) {
    ⑹     return new StudentResponseDTO(student.getFirstname(), student.getLastname(), student.getEmail());
 ⑺ }
```

Mapper functions basically encode and decode to and from DTO objects to the 'real data' model

ToStudentResponseDTO is a mapping function

ToStudent is a mapping function

```
public class SchoolController {  
  
    3 usages  
    private final SchoolRepository schoolRepository;  
  
    @Autowired  
    public SchoolController(SchoolRepository schoolRepository) { this.schoolRepository = schoolRepository; }  
  
    @PostMapping(value="/schools")  
    public SchoolDTO create (@RequestBody SchoolDTO school) {  
        School schoolFromDTO = toSchool(school);  
        School savedSchool = schoolRepository.save(schoolFromDTO);  
        return school;  
    }  
  
    1 usage  
    public School toSchool(SchoolDTO dto) {  
        School school = new School();  
        school.setName(dto.name());  
        return school;  
    }  

```

ToSchool is a mapping function

```
public record studentResponseDTO(  
    no usages  
    String firstname,  
    no usages  
    String lastname,  
    no usages  
    String email  
) {  
}  

```

```
public record studentDTO(String firstname,  
                         1 usage  
                         String lastname,  
                         1 usage  
                         String email,  
  
                         1 usage  
                         Integer schoolId  
) {  
}
```

```
public record schooldTO(  
    1 usage  
    String name  
    |  
    ) {  
}
```

Above is various representations

Java Streams

09 May 2024 07:18

Java Streams:

```
private schoolDTO toSchoolDTO(School school) {
    return new schoolDTO(school.getName());
}

@GetMapping("/schools")
public List<schoolDTO> findAll () {
    return schoolRepository.findAll().stream()
        .map(this::toSchoolDTO).collect(Collectors.toList());
}
```

Stream - in java converts a list of items to be processed one at a time

.Map(this::toSchoolDTO) - applies schoolDTO to each of the items in the map

.Collect puts the results of this back into a list

The screenshot shows the Insomnia REST client interface. The top bar includes the application name, menu options (Application, File, Edit, View, Window, Tools, Help), and a search bar. On the right, there are buttons for Login and Sign Up. The main area has tabs for Scratch Pad, Requests, and Responses. The Requests tab is active, showing a list of recent requests. One request is highlighted: a GET request to 'http://localhost:8080/schools'. The response details show a status of '200 OK', a duration of '252 ms', and a size of '51 B'. The timestamp is 'Just Now'. The Headers section shows a Content-Type header. The Preview section displays the JSON response:

```
1 - [
2 -   {
3 -     "name": "Matt's School"
4 -   }
5 - ]
```

Below the preview, there are buttons for 'Beautify JSON' and '\$.store.books[*].author'. At the bottom, there are links for 'Preferences', 'Log in to see your projects', and 'Made with ❤ by Kong'.

Service Layer

09 May 2024 07:31

Does all complex logic and business functions

Putting our code in a service layer makes our code more reusable and maintainable

Controllers - handles incoming requests and responses, nothing more

Repositories - Handle data persistence and retrieval

Services - handle all in-between logic and promotes separation

Service layer allows us to unit test our business logic independently

Services in action

09 May 2024 07:36

Check codebase for overview

Inject service with controller constructor

```
● ● ●
@RestController
public class StudentController {
    6 usages
    private final StudentService studentService;

    @Autowired
    public StudentController(StudentService studentService) {

        this.studentService = studentService;
    }

    @PostMapping("students")
    public StudentResponseDTO saveStudent(@RequestBody StudentDTO student) { // 
        return this.studentService.saveStudent(student);
    }

    @GetMapping("students")
    public List<Student> findAllStudent() {
        return this.studentService.findAllStudent();
    }

    @GetMapping("students/{student-id}")
    public Student findStudentById(@PathVariable("student-id") int studentId) {
        return this.studentService.findStudentById(studentId);
    }

    @GetMapping("students/search/{student-name}")
    public List<Student> findStudentByName(@PathVariable("student-name") String studentName) {
        //Student[] students = {};
        return this.studentService.findStudentByName(studentName);
    }

    @DeleteMapping("students/{student-id}")
    @ResponseStatus(HttpStatus.OK) //Sets response code
    public void delete(@PathVariable("student-id") Integer studentId) {
        this.studentService.delete(studentId);
        return;
    }
}
```

As you can see the Student Service has been injected

```
● ● ●
@Service
public class StudentService {

    6 usages
    private final StudentRepository repository;

    6 usages
    private final StudentMapper studentMapper;

    @Autowired
    public StudentService(StudentRepository repository, StudentMapper studentMapper) {
        this.repository = repository;
        this.studentMapper = studentMapper;
    }
    1 usage
    public StudentResponseDTO saveStudent(StudentDTO student) {
        Student dto = studentMapper.toStudent(student);
        Student savedStudent = repository.save(dto); //Uses studentDTO (reduced input)
        return studentMapper.TostudentResponseDTO(savedStudent); //response with studentResponseDTO (reduced output)
    }

    1 usage
    public List<StudentResponseDTO> findAllStudent() {
        return this.repository.findAll().stream().map(studentMapper::TostudentResponseDTO).collect(Collectors.toList());
    }
    1 usage
    public StudentResponseDTO findStudentById(int id) {
        return repository.findById(id).map(studentMapper::TostudentResponseDTO).orElse( other: null);

        //Any Optional Type in java can be transformed into a map and sent to a method for transforming.
    }
    1 usage
    public List<StudentResponseDTO> findStudentByName(String name) {
        return this.repository.findAll().stream().map(studentMapper::TostudentResponseDTO).collect(Collectors.toList());
    }
    1 usage
    public void delete(int studentId) {
        repository.deleteById(studentId);
        return;
    }
}
```

The service injects the repository beans

Java Optional

09 May 2024 08:13

Any Optional Type in java can be transformed into a mapped to a method and transformed, :: being a method reference.

Spring repositories return Optional Objects

```
public StudentResponseDTO findStudentById(int id) {  
    return repository.findById(id).map(StudentMapper::toStudentResponseDTO).orElse(null);  
}
```

Organisation of codebase

09 May 2024 08:38

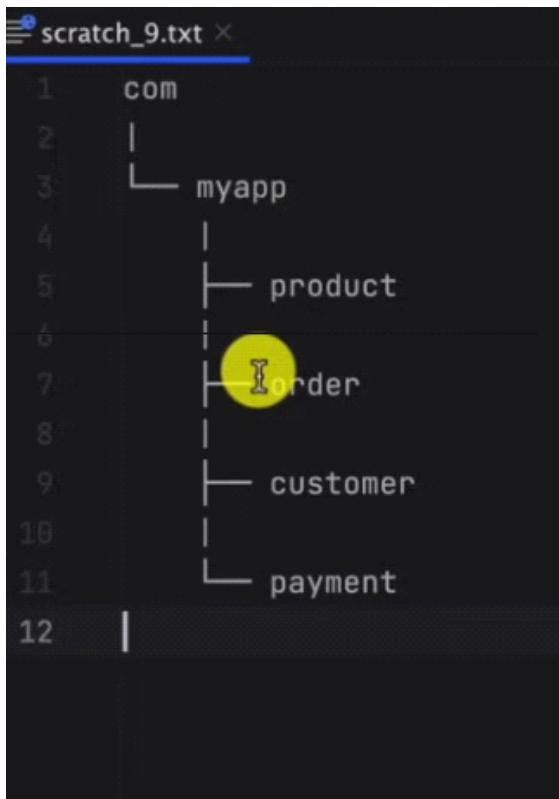
Look at codebase

By feature (preferred personally) :

Split your codebase by features

Each with controller, repositories, services

Good for big teams

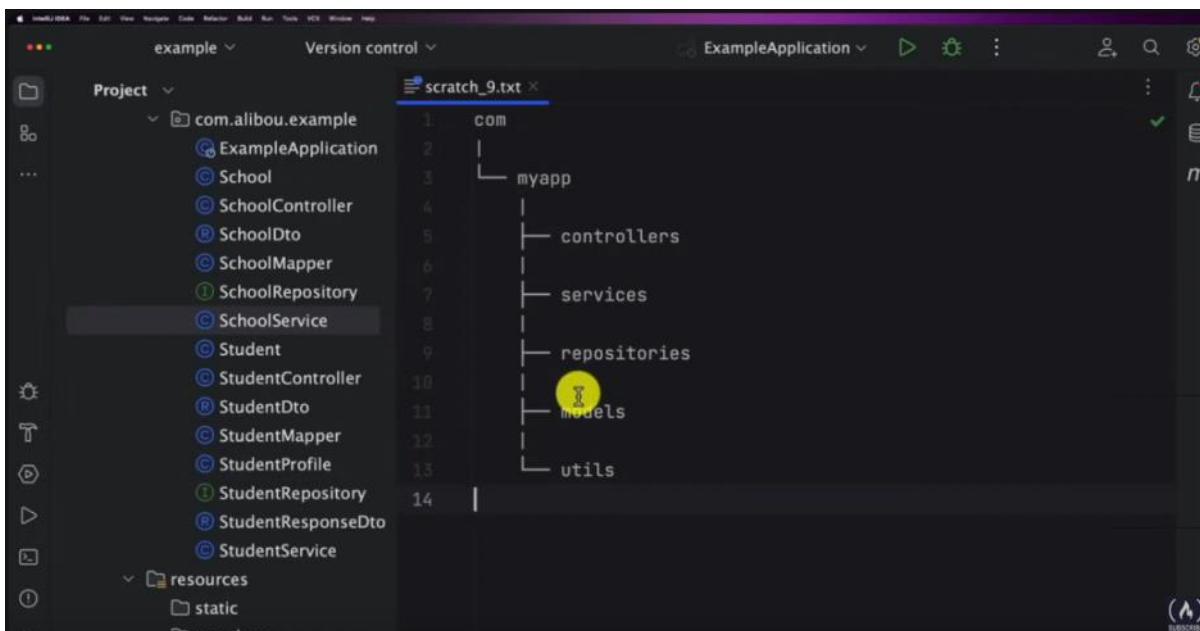


```
scratch_9.txt ×
1   com
2   |
3   └── myapp
4   |
5   ├── product
6   |
7   ├── order
8   |
9   ├── customer
10  |
11  └── payment
12  |
```

Layer approach:

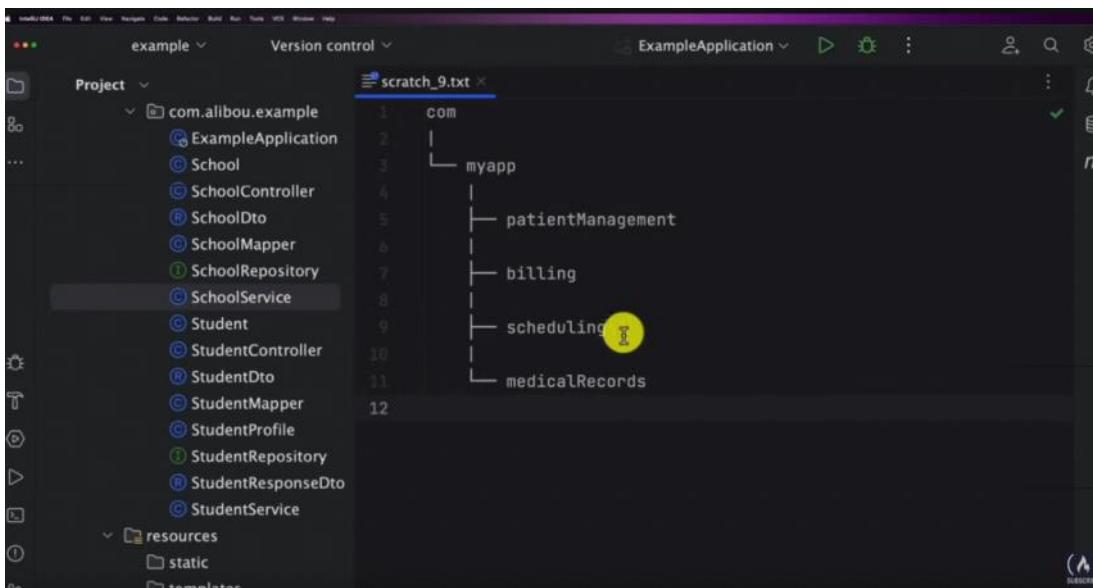
All tech layers are stored in corresponding class

However a change to one feature might break more features within the package



By domain:

Basically by department of the software



By Component:

Break it down into components of your choosing

then further into controllers, services etc

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** On the left, the project structure is displayed under "com.allibou.example". It includes files like ExampleApplication, School, SchoolController, SchoolDto, SchoolMapper, SchoolRepository, SchoolService, Student, StudentController, StudentDto, StudentMapper, StudentProfile, StudentRepository, StudentResponseDto, and StudentService.
- Scratch File:** In the center, there is an open file named "scratch_9.txt" containing the following code:

```
1 com
2 |
3   myapp
4   |
5     usercomponent
6     |
7       controllers
8       services
9
10    productcomponent
11    |
12      controllers
13      services
```
- Toolbars and Status Bar:** The top bar shows "example" and "Version control". The status bar at the bottom right shows "ExampleApplication" and some icons.

Data validation

09 May 2024 11:10

Data validation is important because:

Data integrity - make sure all your data is valid (like emails)

Prevent Attacks - prevent users from sending incorrect data inputs to attack your product

Error Prevention - insures your data is valid for the rest of your code

User experience - ensuring all data appears correctly and errors are well formatted

Performance - receiving large amounts of incorrect data can affect your applications performance

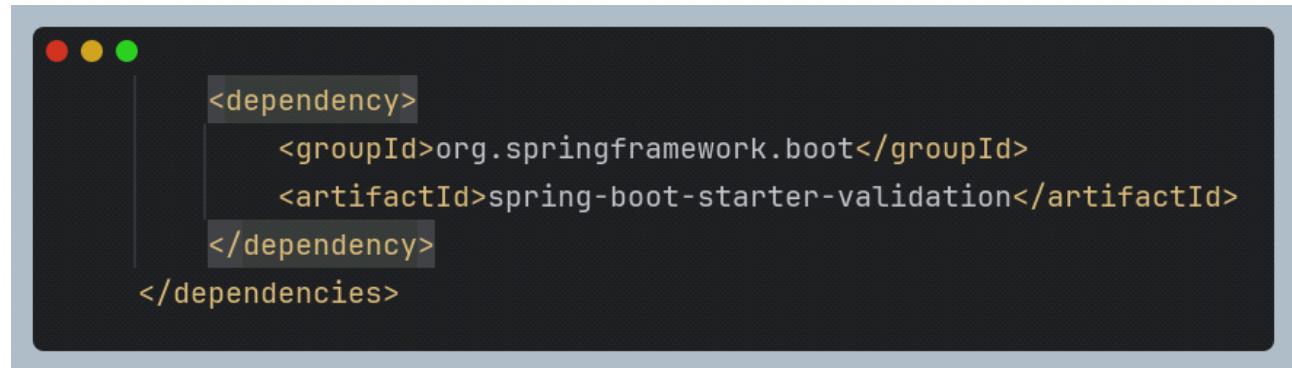
Business Logic compliance - business services will require certain types and formats of data

Spring validation

09 May 2024 11:18

We need to ensure the user is entering the correct data inputs to our services and DTO's

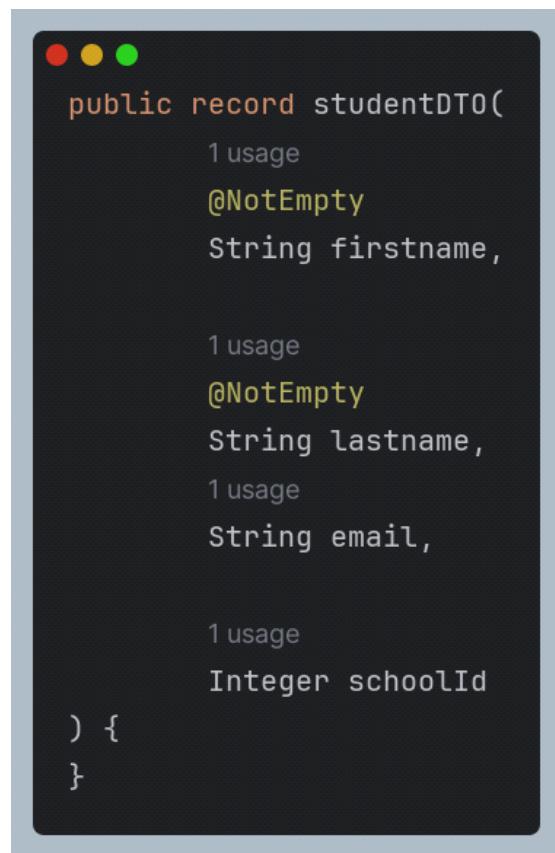
Spring boot validation dependency:



```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
</dependencies>
```

Validation should be added to the first point of user interaction (the DTO normally)

All the below annotations work with POJO's and Records:



```
public record studentDTO(
    1 usage
    @NotEmpty
    String firstname,
    1 usage
    @NotEmpty
    String lastname,
    1 usage
    String email,
    1 usage
    Integer schoolId
) { }
```

@NotEmpty - this value cannot be empty

```
@RestController
public class StudentController {
    6 usages
    private final StudentService studentService;

    @Autowired
    public StudentController(StudentService studentService) {

        this.studentService = studentService;
    }

    @PostMapping("/students")
    public StudentResponseDTO saveStudent(@Valid @RequestBody StudentDTO student) { // 
        return this.studentService.saveStudent(student);
    }
}
```

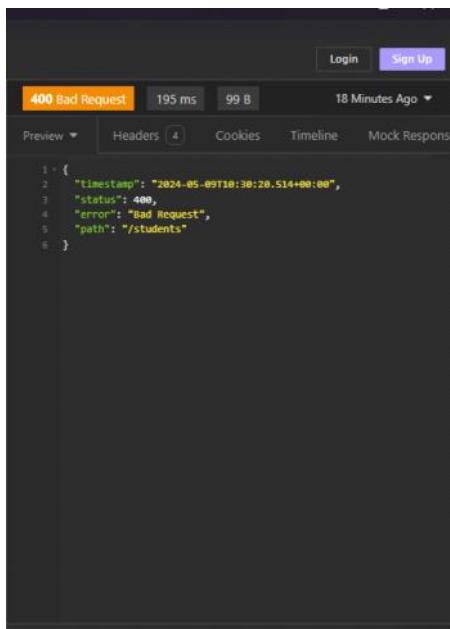
@Valid - enables validation for this controller

Raising validation Exceptions and General Exceptions + ForEach

09 May 2024 11:35

Creating a method to handle our exception.

By default if validation fails user gets:



Produced this exception:

```
00 WARN 72440 --- [nio-8080-exec-1] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved [org.springframework.web.bind.MethodArgumentNotValidException:
```

Exception handler:

```
@ExceptionHandler  
//Every time this exception is thrown, this code will be called  
public ResponseEntity<?> MethodArgumentNotValidException( //Question Mark means of return a ResponseEntity of any type  
                                         MethodArgumentNotValidException exp  
) {  
    HashMap<String, String> errors = new HashMap<?>();  
    exp.getBindingResult().getAllErrors().forEach(error -> {  
        var fieldName = ((FieldError) error).getField();  
        var errorMessage = error.getDefaultMessage();  
        errors.put(fieldName, errorMessage);  
    });  
    return new ResponseEntity<?>(errors, HttpStatus.BAD_REQUEST);  
}
```

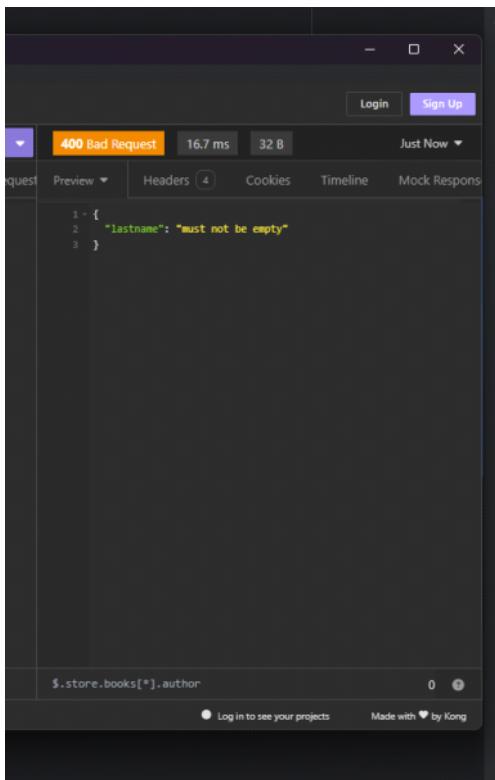
ResponseEntity<?> - Means its gonna return a ResponseEntity of some type

Exp.getBindingResult().getAllErrors() - get the exception results then get all the errors in a list

.forEach - for each one in the list do this lambda

Return HashMap of errors

Makes:



To set a custom message:

```
1 package com.matt.example.Student;
2
3 import jakarta.validation.constraints.NotEmpty;
4
5 public record studentDTO(
6     @NotEmpty(message = "Firstname should not be empty")
7     String firstname,
8
9     @NotEmpty (message = "last name shouldn't be empty")
10    String lastname,
11
12    String email,
13
14    Integer schoolId
15) {
16}
```

Validation Annotations List

09 May 2024 12:10

- **@NotNull:** Ensures a field is not null.
- **@NotBlank:** Enforces non-nullity and requires at least one non-whitespace character.
- **@NotEmpty:** Guarantees that collections or arrays are not empty.
- **@Min(value):** Checks if a numeric field is greater than or equal to the specified minimum value.
- **@Max(value):** Checks if a numeric field is less than or equal to the specified maximum value.
- **@Size(min, max):** Validates if a string or collection size is within a specific range.
- **@Pattern(regex):** Verifies if a field matches the provided regular expression.
- **@Email:** Ensures a field contains a valid email address format.
- **@Digits(integer, fraction):** Validates that a numeric field has a specified number of integer and fraction digits.
- **@Past and @Future :** Checks that a date or time field is in the past and future respectively.
- **@AssertTrue and @AssertFalse:** Ensures that a boolean field is true. and false respectively.
- **@CreditCardNumber:** Validates that a field contains a valid credit card number.

From <<https://medium.com/@himani.prasad016/validations-in-spring-boot-e9948aa6286b>>

To find a bigger list go to External libs and look at constraints

Why Test

09 May 2024 13:01

Quality assurance - Tests help ensure your application works as expected

Regression Testing - These tests makes sure existing functions work as excepted

Documentation - Helps show your code works as excepted

Code Maintainability - Encourage, more modular, better quality code

Refactoring Confidence - show's you if something you have done is working or not

Collaboration - Tests make sure the work of one doesn't impact another

CI/CD pipelines - automatic testing when you push

Reduced Debugging time - reduces the time spent debugging

Scalability - allows you to confidentially scale up

Security - Some test help find security problems



SPRING TEST - INTRODUCTION

- ▷ Spring Boot provides utilities and annotations for testing applications
- ▷ Test support is provided by two modules
 - ➔ spring-boot-test contains core items
 - ➔ spring-boot-test-autoconfigure supports auto-configuration for tests
 - ➔ Spring-boot-starter-test "Starter", imports:
 - ➔ Spring Boot test modules
 - ➔ JUnit, AssertJ, Hamcrest, and a number of other useful libraries



Spring test

09 May 2024 13:21

A spring boot application is a Spring ApplicationContext

To declare a test with `SpringBootTest` annotation when you need spring features like `ApplicationContext`

Junit 4 support requires `@RunWith(SpringRunner.class)` to be added to your test

JUnit5 doesn't require this

Spring has lots of autoConfigurations, its often recommended you only load 'slices' of the auto-configs though, as they are too much for tests often.

These auto configs give you a number of annotations and features you can use

Each slice restricts the amount of these you can access

What to test

09 May 2024 13:29

You only test things with logic within:

Repositories - don't require tests

Mappers - do require tests

Services - Do require tests

Controllers - do require tests

Test function explainer

09 May 2024 13:32

Always keep the same package naming between 'main' and tests

Tests use JUnit5 or JUnit4

Use CTRL+SHIFT+T to generate a TEST

Screenshot explains different functions

```
class StudentMapperTest {

    @BeforeAll
    static void beforeAll() {
        System.out.println("before class, before class initialize something");
    }

    @AfterAll
    static void afterAll() {
        System.out.println("after we stopped using class, clean up database example");
    }

    @BeforeEach
    void setUp() {
        //Initialize anything before tests here
        System.out.println("before each method, initialize stuff per test");
    }

    @AfterEach
    void tearDown() {
        //Runs after every test
        System.out.println("after each method");
    }

    @Test
    public void testMethod1() {
        System.out.println("my first test");
    }
    @Test
    public void testMethod2() {
        System.out.println(" my second test");
    }
}
```

Testing the mappers (coverage tests)

09 May 2024 14:00

Remember, we are using JUnit5

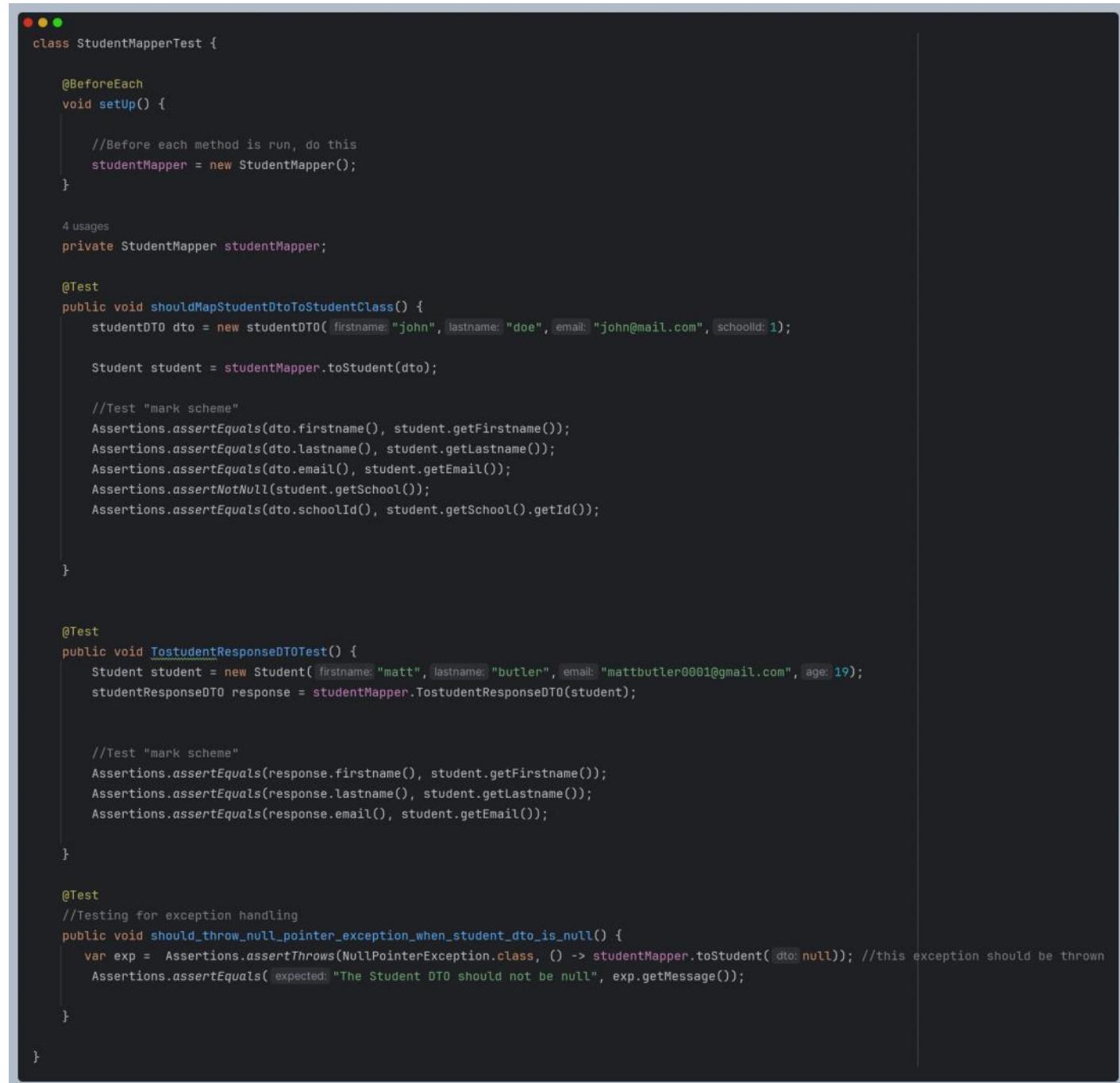
Assertions generate errors based on conditions, used during tests

assertEquals will error if two things are not the same

assertNotNull will error something is null

AssertThrows allows us to check what exception was thrown

Screenshot demo's everything



```
class StudentMapperTest {

    @BeforeEach
    void setUp() {
        //Before each method is run, do this
        studentMapper = new StudentMapper();
    }

    4 usages
    private StudentMapper studentMapper;

    @Test
    public void shouldMapStudentDtoToStudentClass() {
        studentDTO dto = new studentDTO( firstname: "john", lastname: "doe", email: "john@mail.com", schoolId: 1);

        Student student = studentMapper.toStudent(dto);

        //Test "mark scheme"
        Assertions.assertEquals(dto.firstname(), student.getFirstname());
        Assertions.assertEquals(dto.lastname(), student.getLastname());
        Assertions.assertEquals(dto.email(), student.getEmail());
        Assertions.assertNotNull(student.getSchool());
        Assertions.assertEquals(dto.schoolId(), student.getSchool().getId());
    }

    @Test
    public void TostudentResponseDTOTest() {
        Student student = new Student( firstname: "matt", lastname: "butler", email: "mattbutler0001@gmail.com", age: 19);
        studentResponseDTO response = studentMapper.TostudentResponseDTO(student);

        //Test "mark scheme"
        Assertions.assertEquals(response.firstname(), student.getFirstname());
        Assertions.assertEquals(response.lastname(), student.getLastname());
        Assertions.assertEquals(response.email(), student.getEmail());

    }

    @Test
    //Testing for exception handling
    public void should_throw_null_pointer_exception_when_student_dto_is_null() {
        var exp = Assertions.assertThrows(NullPointerException.class, () -> studentMapper.toStudent( dto: null)); //this exception should be thrown
        Assertions.assertEquals( expected: "The Student DTO should not be null", exp.getMessage());
    }
}
```

Testing the student service

11 May 2024 06:05

Check screenshots for this one (trust)

We need to use test isolation where possible

This often includes mocking a database

"Mock" means "faking an objects and/or methods"

In this case, we will be mocking our dependences using Mockito

injectMocks - will find any constructor compatible dependencies with @mock and inject them into annotated service

Mock - mock this method / object

Then just open the mocks / enable them for current class

Mocking our dependency calls:

To find what calls to mock go through method logic

We must mock our dependency calls used in our test

Use Mockito's logic to create them with the same logic as real function

argumentMatcher.any(Class) basically, if the input to the call is dynamic, could be any object of a certain type (like many different students). use any(CLASS OBJECT TYPE).

Test layout:

Given (mock calls as well),

When,

Then (mark scheme),

performance:

Verify - allows us to see how many times a method is called during the execution of the test (stops database dupes etc) This includes nested calls with our methods (the entire point)

```
class StudentServiceTest {  
    // 4 usages  
    @InjectMocks //Inject our mocked objects into student service  
    private StudentService studentService;  
  
    //Declare dependencies  
  
    // 8 usages  
    @Mock //Mock these objects  
    private StudentRepository repository; //these need to be mocked  
  
    // 7 usages  
    @Mock //Mock these objects  
    private StudentMapper studentMapper; //these need to be mocked  
  
    @BeforeEach  
    void setup() {  
        MockitoAnnotations.openMocks(this);  
    }  
  
    @Test  
    public void should_successfully_save_a_student() {  
        //Given  
        StudentDTO dto = new StudentDTO("john", "doe", "john@mail.com", 20);  
        Student student = new Student("john", "doe", "john@mail.com", 20);  
        Student savedStudent = new Student("john", "doe", "john@mail.com", 20);  
        savedStudent.setId(1);  
  
        //Mock the calls used by this test  
        Mockito.when(studentMapper.toStudent(dto))  
            .thenReturn(student);  
        Mockito.when(repository.save(student)).thenReturn(savedStudent);  
  
        Mockito.when(studentMapper.toStudentResponseDTO(savedStudent)).thenReturn(new StudentResponseDTO("john", "doe", "john@mail.com"));  
  
        //When  
        StudentResponseDTO response = studentService.saveStudent(dto);  
  
        //Then  
        assertEquals(dto.firstname(), response.firstname());  
        assertEquals(dto.lastname(), response.lastname());  
        assertEquals(dto.email(), response.email());  
  
        Mockito.verify(studentMapper, Mockito.times(wantedNumberofInvocations(1))).toStudent(dto); //Check toStudent has only been called once during the execution of the test  
        Mockito.verify(repository, Mockito.times(wantedNumberofInvocations(1))).save(student); //Check save has only been called once during the execution of the test  
        Mockito.verify(studentMapper, Mockito.times(wantedNumberofInvocations(1))).toStudentResponseDTO(savedStudent); //Check ToStudentResponseDTO has only been called once during the execution of the test  
    }  
  
    @Test  
    public void should_return_all_students() {  
        //Given  
        List<Student> students = new ArrayList<>();  
        students.add(new Student("john", "doe", "john@mail.com", 20));  
        //Mock Calls  
        Mockito.when(repository.findAll()).thenReturn(students);  
        Mockito.when(studentMapper.toStudentResponseDTO(ArgumentMatchers.any(Student.class))).thenReturn(new StudentResponseDTO("john", "doe", "john@mail.com")); //Because we could pass any student use any()  
    }
```

```

//Given
List<Student> students = new ArrayList<>();
students.add(new Student("john", "doe", "john@mail.com", 20));
//Mock Calls
Mockito.when(repository.findAll()).thenReturn(students);
Mockito.when(studentMapper.TostudentResponseDTO(ArgumentMatchers.any(Student.class))).thenReturn(new studentResponseDTO("john", "doe", "john@mail.com")); //Because we could pass any student use any()

//When
List<studentResponseDTO> findAllStudent = studentService.findAllStudent();

//Then
assertEquals(students.size(), findAllStudent.size());
Mockito.verify(repository, Mockito.times(wantedNumberOfInvocations 1)).findAll();

}

@Test
public void should_return_student_with_id() {
    //Given
    int studentID = 1;
    Student student = new Student("john", "doe", "john@mail.com", 20);
    studentResponseDTO dto = new studentResponseDTO("john", "doe", "john@mail.com");

    //Mock calls
    Mockito.when(repository.findById(studentID)).thenReturn(Optional.of(student));
    Mockito.when(studentMapper.TostudentResponseDTO(student)).thenReturn(dto);

    //When
    studentResponseDTO studentFound = studentService.findStudentById(1);

    //then
    assertEquals(dto.firstname(), studentFound.firstname());
    assertEquals(dto.lastname(), studentFound.lastname());
    assertEquals(dto.email(), studentFound.getEmail());
    Mockito.verify(repository, Mockito.times(wantedNumberOfInvocations 1)).findById(studentID);
}

@Test
public void findStudentByNameTest() {

    //Given
    String name = "john";
    List<Student> studentList = new ArrayList<>();
    studentList.add(new Student("john", "doe", "john@mail.com", 20));
    studentResponseDTO dto = new studentResponseDTO("john", "doe", "john@mail.com");

    //Mock calls
    Mockito.when(repository.findAllByFirstname(name)).thenReturn(studentList);
    Mockito.when(studentMapper.TostudentResponseDTO(Mockito.any(Student.class))).thenReturn(dto);

    //When
    List<studentResponseDTO> studentDTOS = studentService.findStudentByName("john");

    //Then
    System.out.println(studentDTOS.size());
    System.out.println(studentList.size());
    assertEquals(studentList.size(), studentDTOS.size());
    Mockito.verify(repository, Mockito.times(wantedNumberOfInvocations 1)).findAllByFirstname(name);
}
}

```

Java Optional

09 May 2024 08:13

Any Optional Type in java can be transformed into a mapped to a method and transformed, :: being a method reference.

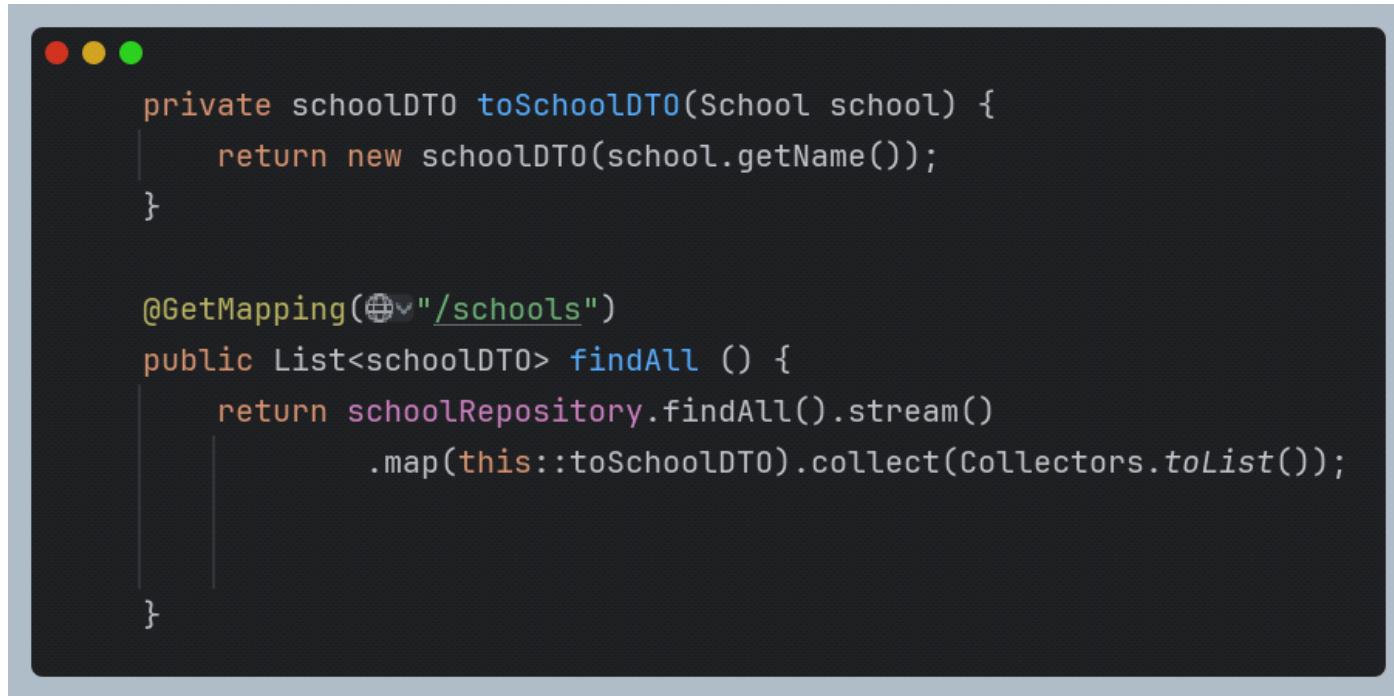
Spring repositories return Optional Objects

```
public StudentResponseDTO findStudentById(int id) {  
    return repository.findById(id).map(studentMapper::toStudentResponseDTO).orElse( other: null);  
}
```

Java Streams

09 May 2024 07:18

Java Streams:



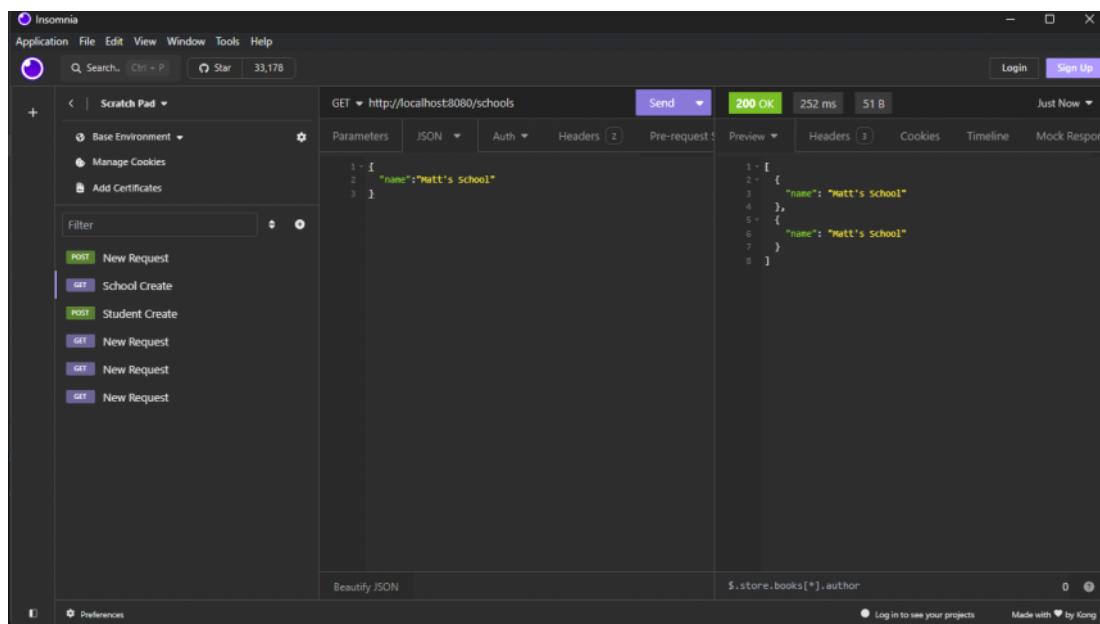
```
private schoolDTO toSchoolDTO(School school) {
    return new schoolDTO(school.getName());
}

@GetMapping("/schools")
public List<schoolDTO> findAll () {
    return schoolRepository.findAll().stream()
        .map(this::toSchoolDTO).collect(Collectors.toList());
}
```

Stream - in java converts a list of items to be processed one at a time

.Map(this::toSchoolDTO) - applies schoolDTO to each of the items in the map

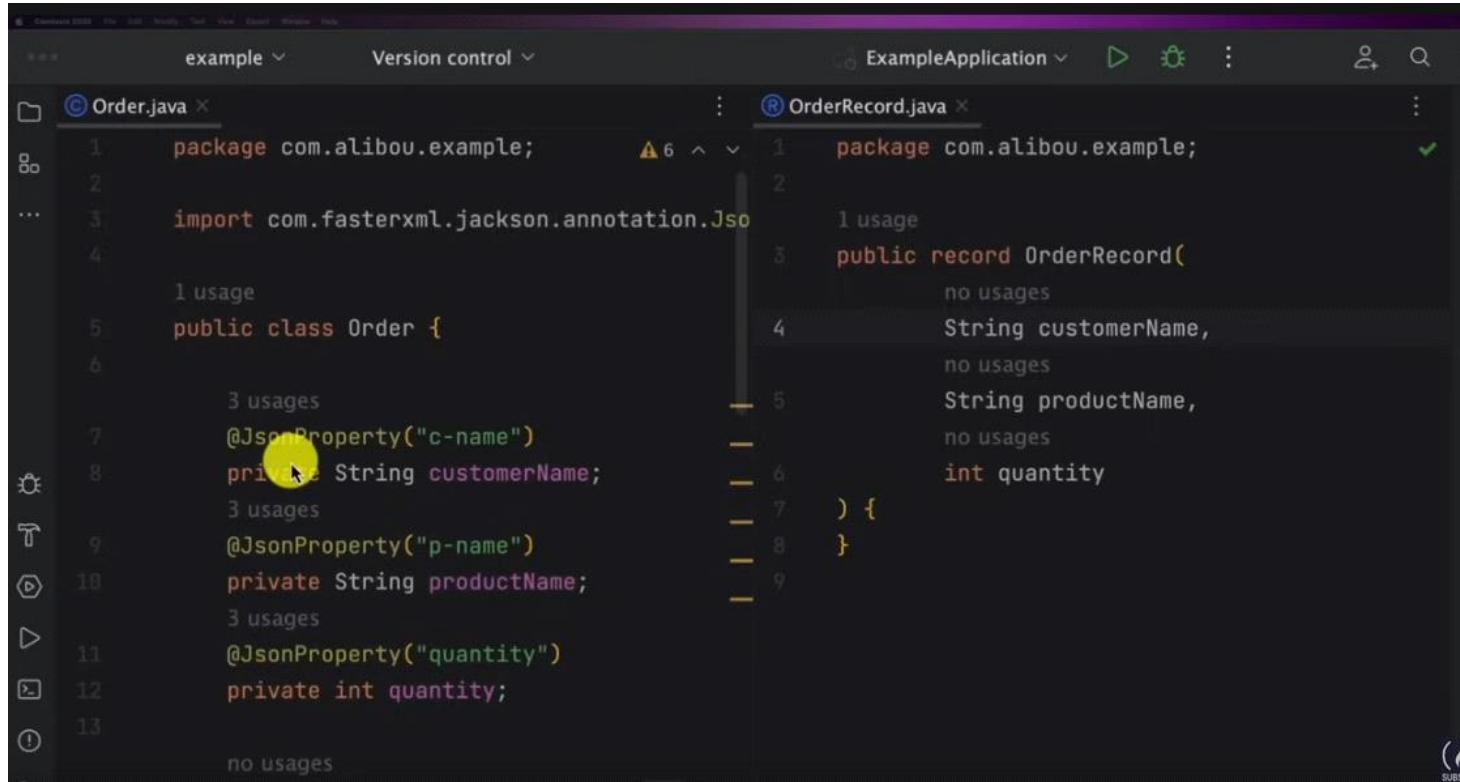
.Collect puts the results of this back into a list



The screenshot shows the Insomnia REST client interface. The top bar includes 'Insomnia' logo, 'Application', 'File', 'Edit', 'View', 'Window', 'Tools', 'Help', 'Login', and 'Sign Up'. The main area has a 'Scratch Pad' tab selected. The URL 'GET http://localhost:8080/schools' is entered. The response status is '200 OK' with '252 ms' and '51 B' response size. The 'Preview' tab shows a JSON array with one item: [{ "name": "Matt's School" }]. The 'Headers' tab shows 'Content-Type: application/json'. The bottom right corner shows 'Log in to see your projects' and 'Made with ❤ by Kong'.

Records vs POJO Classes

08 May 2024 05:55



```
example Version control ExampleApplication Order.java OrderRecord.java
1 package com.alibou.example; 1 package com.alibou.example;
2
3 import com.fasterxml.jackson.annotation.Json 2
4
5 public class Order { 3 usage
6
7     1 usage
8     private String customerName; 4
9
10    3 usages
11    @JsonProperty("c-name") 5
12    private String productName; 6
13    3 usages
14    @JsonProperty("p-name") 7
15    private int quantity; 8
16
17    3 usages
18    @JsonProperty("quantity") 9
19
20    private int quantity; 20
21
22    no usages 21
23
24    no usages 22
25
26    String customerName, 23
27
28    no usages 24
29    String productName, 25
30
31    int quantity 26
32
33    ) { 27
34        } 28
35    } 29
36
37    } 20
38
39    } 21
40
41    } 22
42
43    } 23
44
45    } 24
46
47    } 25
48
49    } 26
50
51    } 27
52
53    } 28
54
55    } 29
56
57    } 20
58
59    } 21
60
61    } 22
62
63    } 23
64
65    } 24
66
67    } 25
68
69    } 26
70
71    } 27
72
73    } 28
74
75    } 29
76
77    } 20
78
79    } 21
80
81    } 22
82
83    } 23
84
85    } 24
86
87    } 25
88
89    } 26
90
91    } 27
92
93    } 28
94
95    } 29
96
97    } 20
98
99    } 21
100
101    } 22
102
103    } 23
104
105    } 24
106
107    } 25
108
109    } 26
110
111    } 27
112
113    } 28
114
115    } 29
116
117    } 20
118
119    } 21
120
121    } 22
122
123    } 23
124
125    } 24
126
127    } 25
128
129    } 26
130
131    } 27
132
133    } 28
134
135    } 29
136
137    } 20
138
139    } 21
140
141    } 22
142
143    } 23
144
145    } 24
146
147    } 25
148
149    } 26
150
151    } 27
152
153    } 28
154
155    } 29
156
157    } 20
158
159    } 21
160
161    } 22
162
163    } 23
164
165    } 24
166
167    } 25
168
169    } 26
170
171    } 27
172
173    } 28
174
175    } 29
176
177    } 20
178
179    } 21
180
181    } 22
182
183    } 23
184
185    } 24
186
187    } 25
188
189    } 26
190
191    } 27
192
193    } 28
194
195    } 29
196
197    } 20
198
199    } 21
200
201    } 22
202
203    } 23
204
205    } 24
206
207    } 25
208
209    } 26
210
211    } 27
212
213    } 28
214
215    } 29
216
217    } 20
218
219    } 21
220
221    } 22
222
223    } 23
224
225    } 24
226
227    } 25
228
229    } 26
230
231    } 27
232
233    } 28
234
235    } 29
236
237    } 20
238
239    } 21
240
241    } 22
242
243    } 23
244
245    } 24
246
247    } 25
248
249    } 26
250
251    } 27
252
253    } 28
254
255    } 29
256
257    } 20
258
259    } 21
260
261    } 22
262
263    } 23
264
265    } 24
266
267    } 25
268
269    } 26
270
271    } 27
272
273    } 28
274
275    } 29
276
277    } 20
278
279    } 21
280
281    } 22
282
283    } 23
284
285    } 24
286
287    } 25
288
289    } 26
290
291    } 27
292
293    } 28
294
295    } 29
296
297    } 20
298
299    } 21
300
301    } 22
302
303    } 23
304
305    } 24
306
307    } 25
308
309    } 26
310
311    } 27
312
313    } 28
314
315    } 29
316
317    } 20
318
319    } 21
320
321    } 22
322
323    } 23
324
325    } 24
326
327    } 25
328
329    } 26
330
331    } 27
332
333    } 28
334
335    } 29
336
337    } 20
338
339    } 21
340
341    } 22
342
343    } 23
344
345    } 24
346
347    } 25
348
349    } 26
350
351    } 27
352
353    } 28
354
355    } 29
356
357    } 20
358
359    } 21
360
361    } 22
362
363    } 23
364
365    } 24
366
367    } 25
368
369    } 26
370
371    } 27
372
373    } 28
374
375    } 29
376
377    } 20
378
379    } 21
380
381    } 22
382
383    } 23
384
385    } 24
386
387    } 25
388
389    } 26
390
391    } 27
392
393    } 28
394
395    } 29
396
397    } 20
398
399    } 21
400
401    } 22
402
403    } 23
404
405    } 24
406
407    } 25
408
409    } 26
410
411    } 27
412
413    } 28
414
415    } 29
416
417    } 20
418
419    } 21
420
421    } 22
422
423    } 23
424
425    } 24
426
427    } 25
428
429    } 26
430
431    } 27
432
433    } 28
434
435    } 29
436
437    } 20
438
439    } 21
440
441    } 22
442
443    } 23
444
445    } 24
446
447    } 25
448
449    } 26
450
451    } 27
452
453    } 28
454
455    } 29
456
457    } 20
458
459    } 21
460
461    } 22
462
463    } 23
464
465    } 24
466
467    } 25
468
469    } 26
470
471    } 27
472
473    } 28
474
475    } 29
476
477    } 20
478
479    } 21
480
481    } 22
482
483    } 23
484
485    } 24
486
487    } 25
488
489    } 26
490
491    } 27
492
493    } 28
494
495    } 29
496
497    } 20
498
499    } 21
500
501    } 22
502
503    } 23
504
505    } 24
506
507    } 25
508
509    } 26
510
511    } 27
512
513    } 28
514
515    } 29
516
517    } 20
518
519    } 21
520
521    } 22
522
523    } 23
524
525    } 24
526
527    } 25
528
529    } 26
530
531    } 27
532
533    } 28
534
535    } 29
536
537    } 20
538
539    } 21
540
541    } 22
542
543    } 23
544
545    } 24
546
547    } 25
548
549    } 26
550
551    } 27
552
553    } 28
554
555    } 29
556
557    } 20
558
559    } 21
560
561    } 22
562
563    } 23
564
565    } 24
566
567    } 25
568
569    } 26
570
571    } 27
572
573    } 28
574
575    } 29
576
577    } 20
578
579    } 21
580
581    } 22
582
583    } 23
584
585    } 24
586
587    } 25
588
589    } 26
590
591    } 27
592
593    } 28
594
595    } 29
596
597    } 20
598
599    } 21
600
601    } 22
602
603    } 23
604
605    } 24
606
607    } 25
608
609    } 26
610
611    } 27
612
613    } 28
614
615    } 29
616
617    } 20
618
619    } 21
620
621    } 22
622
623    } 23
624
625    } 24
626
627    } 25
628
629    } 26
630
631    } 27
632
633    } 28
634
635    } 29
636
637    } 20
638
639    } 21
640
641    } 22
642
643    } 23
644
645    } 24
646
647    } 25
648
649    } 26
650
651    } 27
652
653    } 28
654
655    } 29
656
657    } 20
658
659    } 21
660
661    } 22
662
663    } 23
664
665    } 24
666
667    } 25
668
669    } 26
670
671    } 27
672
673    } 28
674
675    } 29
676
677    } 20
678
679    } 21
680
681    } 22
682
683    } 23
684
685    } 24
686
687    } 25
688
689    } 26
690
691    } 27
692
693    } 28
694
695    } 29
696
697    } 20
698
699    } 21
700
701    } 22
702
703    } 23
704
705    } 24
706
707    } 25
708
709    } 26
710
711    } 27
712
713    } 28
714
715    } 29
716
717    } 20
718
719    } 21
720
721    } 22
722
723    } 23
724
725    } 24
726
727    } 25
728
729    } 26
730
731    } 27
732
733    } 28
734
735    } 29
736
737    } 20
738
739    } 21
740
741    } 22
742
743    } 23
744
745    } 24
746
747    } 25
748
749    } 26
750
751    } 27
752
753    } 28
754
755    } 29
756
757    } 20
758
759    } 21
760
761    } 22
762
763    } 23
764
765    } 24
766
767    } 25
768
769    } 26
770
771    } 27
772
773    } 28
774
775    } 29
776
777    } 20
778
779    } 21
780
781    } 22
782
783    } 23
784
785    } 24
786
787    } 25
788
789    } 26
790
791    } 27
792
793    } 28
794
795    } 29
796
797    } 20
798
799    } 21
800
801    } 22
802
803    } 23
804
805    } 24
806
807    } 25
808
809    } 26
810
811    } 27
812
813    } 28
814
815    } 29
816
817    } 20
818
819    } 21
820
821    } 22
822
823    } 23
824
825    } 24
826
827    } 25
828
829    } 26
830
831    } 27
832
833    } 28
834
835    } 29
836
837    } 20
838
839    } 21
840
841    } 22
842
843    } 23
844
845    } 24
846
847    } 25
848
849    } 26
850
851    } 27
852
853    } 28
854
855    } 29
856
857    } 20
858
859    } 21
860
861    } 22
862
863    } 23
864
865    } 24
866
867    } 25
868
869    } 26
870
871    } 27
872
873    } 28
874
875    } 29
876
877    } 20
878
879    } 21
880
881    } 22
882
883    } 23
884
885    } 24
886
887    } 25
888
889    } 26
890
891    } 27
892
893    } 28
894
895    } 29
896
897    } 20
898
899    } 21
900
901    } 22
902
903    } 23
904
905    } 24
906
907    } 25
908
909    } 26
910
911    } 27
912
913    } 28
914
915    } 29
916
917    } 20
918
919    } 21
920
921    } 22
922
923    } 23
924
925    } 24
926
927    } 25
928
929    } 26
930
931    } 27
932
933    } 28
934
935    } 29
936
937    } 20
938
939    } 21
940
941    } 22
942
943    } 23
944
945    } 24
946
947    } 25
948
949    } 26
950
951    } 27
952
953    } 28
954
955    } 29
956
957    } 20
958
959    } 21
960
961    } 22
962
963    } 23
964
965    } 24
966
967    } 25
968
969    } 26
970
971    } 27
972
973    } 28
974
975    } 29
976
977    } 20
978
979    } 21
980
981    } 22
982
983    } 23
984
985    } 24
986
987    } 25
988
989    } 26
990
991    } 27
992
993    } 28
994
995    } 29
996
997    } 20
998
999    } 21
1000
1001    } 22
1002
1003    } 23
1004
1005    } 24
1006
1007    } 25
1008
1009    } 26
1010
1011    } 27
1012
1013    } 28
1014
1015    } 29
1016
1017    } 20
1018
1019    } 21
1020
1021    } 22
1022
1023    } 23
1024
1025    } 24
1026
1027    } 25
1028
1029    } 26
1030
1031    } 27
1032
1033    } 28
1034
1035    } 29
1036
1037    } 20
1038
1039    } 21
1040
1041    } 22
1042
1043    } 23
1044
1045    } 24
1046
1047    } 25
1048
1049    } 26
1050
1051    } 27
1052
1053    } 28
1054
1055    } 29
1056
1057    } 20
1058
1059    } 21
1060
1061    } 22
1062
1063    } 23
1064
1065    } 24
1066
1067    } 25
1068
1069    } 26
1070
1071    } 27
1072
1073    } 28
1074
1075    } 29
1076
1077    } 20
1078
1079    } 21
1080
1081    } 22
1082
1083    } 23
1084
1085    } 24
1086
1087    } 25
1088
1089    } 26
1090
1091    } 27
1092
1093    } 28
1094
1095    } 29
1096
1097    } 20
1098
1099    } 21
1100
1101    } 22
1102
1103    } 23
1104
1105    } 24
1106
1107    } 25
1108
1109    } 26
1110
1111    } 27
1112
1113    } 28
1114
1115    } 29
1116
1117    } 20
1118
1119    } 21
1120
1121    } 22
1122
1123    } 23
1124
1125    } 24
1126
1127    } 25
1128
1129    } 26
1130
1131    } 27
1132
1133    } 28
1134
1135    } 29
1136
1137    } 20
1138
1139    } 21
1140
1141    } 22
1142
1143    } 23
1144
1145    } 24
1146
1147    } 25
1148
1149    } 26
1150
1151    } 27
1152
1153    } 28
1154
1155    } 29
1156
1157    } 20
1158
1159    } 21
1160
1161    } 22
1162
1163    } 23
1164
1165    } 24
1166
1167    } 25
1168
1169    } 26
1170
1171    } 27
1172
1173    } 28
1174
1175    } 29
1176
1177    } 20
1178
1179    } 21
1180
1181    } 22
1182
1183    } 23
1184
1185    } 24
1186
1187    } 25
1188
1189    } 26
1190
1191    } 27
1192
1193    } 28
1194
1195    } 29
1196
1197    } 20
1198
1199    } 21
1200
1201    } 22
1202
1203    } 23
1204
1205    } 24
1206
1207    } 25
1208
1209    } 26
1210
1211    } 27
1212
1213    } 28
1214
1215    } 29
1216
1217    } 20
1218
1219    } 21
1220
1221    } 22
1222
1223    } 23
1224
1225    } 24
1226
1227    } 25
1228
1229    } 26
1230
1231    } 27
1232
1233    } 28
1234
1235    } 29
1236
1237    } 20
1238
1239    } 21
1240
1241    } 22
1242
1243    } 23
1244
1245    } 24
1246
1247    } 25
1248
1249    } 26
1250
1251    } 27
1252
1253    } 28
1254
1255    } 29
1256
1257    } 20
1258
1259    } 21
1260
1261    } 22
1262
1263    } 23
1264
1265    } 24
1266
1267    } 25
1268
1269    } 26
1270
1271    } 27
1272
1273    } 28
1274
1275    } 29
1276
1277    } 20
1278
1279    } 21
1280
1281    } 22
1282
1283    } 23
1284
1285    } 24
1286
1287    } 25
1288
1289    } 26
1290
1291    } 27
1292
1293    } 28
1294
1295    } 29
1296
1297    } 20
1298
1299    } 21
1300
1301    } 22
1302
1303    } 23
1304
1305    } 24
1306
1307    } 25
1308
1309    } 26
1310
1311    } 27
1312
1313    } 28
1314
1315    } 29
1316
1317    } 20
1318
1319    } 21
1320
1321    } 22
1322
1323    } 23
1324
1325    } 24
1326
1327    } 25
1328
1329    } 26
1330
1331    } 27
1332
1333    } 28
1334
1335    } 29
1336
1337    } 20
1338
1339    } 21
1340
1341    } 22
1342
1343    } 23
1344
1345    } 24
1346
1347    } 25
1348
1349    } 26
1350
1351    } 27
1352
1353    } 28
1354
1355    } 29
1356
1357    } 20
1358
1359    } 21
1360
1361    } 22
1362
1363    } 23
1364
1365    } 24
1366
1367    } 25
1368
1369    } 26
1370
1371    } 27
1372
1373    } 28
1374
1375    } 29
1376
1377    } 20
1378
1379    } 21
1380
1381    } 22
1382
1383    } 23
1384
1385    } 24
1386
1387    } 25
1388
1389    } 26
1390
1391    } 27
1392
1393    } 28
1394
1395    } 29
1396
1397    } 20
1398
1399    } 21
1400
1401    } 22
1402
1403    } 23
1404
1405    } 24
1406
1407    } 25
1408
1409    } 26
1410
1411    } 27
1412
1413    } 28
1414
1415    } 29
1416
1417    } 20
1418
1419    } 21
1420
1421    } 22
1422
1423    } 23
1424
1425    } 24
1426
1427    } 25
1428
1429    } 26
1430
1431    } 27
1432
1433    } 28
1434
1435    } 29
1436
1437    } 20
1438
1439    } 21
1440
1441    } 22
1442
1443    } 23
1444
1445    } 24
1446
1447    } 25
1448
1449    } 26
1450
1451    } 27
1452
1453    } 28
1454
1455    } 29
1456
1457    } 20
1458
1459    } 21
1460
1461    } 22
1462
1463    } 23
1464
1465    } 24
1466
1467    } 25
1468
1469    } 26
1470
1471    } 27
1472
1473    } 28
1474
1475    } 29
1476
1477    } 20
1478
1479    } 21
1480
1481    } 22
1482
1483    } 23
1484
1485    } 24
1486
1487    } 25
1488
1489    } 26
1490
1491    } 27
1492
1493    } 28
1494
1495    } 29
1496
1497    } 20
1498
1499    } 21
1500
1501    } 22
1502
1503    } 23
1504
1505    } 24
1506
1507    } 25
1508
1509    } 26
1510
1511    } 27
1512
1513    } 28
1514
1515    } 29
1516
1517    } 20
1518
1519    } 21
1520
1521    } 22
1522
1523    } 23
1524
1525    } 24
1526
1527    } 25
1528
1529    } 26
1530
1531    } 27
1532
1533    } 28
1534
1535    } 29
1536
1537    } 20
1538
1539    } 21
1540
1541    } 22
1542
1543    } 23
1544
1545    } 24
1546
1547    } 25
1548
1549    } 26
1550
1551    } 27
1552
1553    } 28
1554
1555    } 29
1556
1557    } 20
1558
1559    } 21
1560
1561    } 22
1562
1563    } 23
1564
1565    } 24
1566
1567    } 25
1568
1569    } 26
1570
1571    } 27
1572
1573    } 28
1574
1575    } 29
1576
1577    } 20
1578
1579    } 21
1580
1581    } 22
1582
1583    } 23
1584
1585    } 24
1586
1587    } 25
1588
1589    } 26
1590
1591    } 27
1592
1593    } 28
1594
1595    } 29
1596
1597    } 20
1598
1599    } 21
1600
16
```

REST crud functions + OrElse

08 May 2024 11:28

```
1 package com.matt.example;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 import java.util.List;
7
8 2 usages
9 @Repository
10 public interface StudentRepository extends JpaRepository<Student, Integer> {
11     1 usage
12     List<Student> findAllByFirstname(String name);
13 }
14
```

Build queries with a method signatures like above

```
10
11 @RestController
12 public class FirstController {
13     6 usages
14     private final StudentRepository repository;
15
16     @Autowired
17     public FirstController(StudentRepository repository) {
18         this.repository = repository;
19     }
20
21     @PostMapping("/students")
22     public Student post(@RequestBody Student student) {
23         return repository.save(student);
24     }
25
26     @GetMapping("/students")
27     public List<Student> findAllStudent() {
28         return repository.findAll();
29     }
30     @GetMapping("/students/{student-id}")
31     public Student findStudentById(@PathVariable("student-id") int studentId) {
32         return repository.findById(studentId).orElse(new Student());
33     }
34     @GetMapping("/students/search/{student-name}")
35     public List<Student> findStudentByName(@PathVariable("student-name") String studentName) {
36         //Student[] students = {};
37         return repository.findAllByFirstname(studentName);
38     }
39
```

```

36         return repository.findAllByFirstname(studentName);
37     }
38
39     @DeleteMapping("/students/{student-id}")
40     @ResponseStatus(HttpStatus.OK) //Sets response code
41     public void delete(@PathVariable("student-id") Integer studentId) {
42         repository.deleteById(studentId);

```

The screenshot shows the Insomnia REST client interface. A GET request is made to `http://localhost:8080/students/search/chris`. The response is `200 OK` with a response time of `101 ms` and a size of `92 B`, timestamped `1 Minute Ago`. The response body is a JSON array containing one element:

```

1: [
2:   {
3:     "id": 1,
4:     "firstname": "chris",
5:     "lastname": "butler",
6:     "email": "chrisabutler@gmail.com",
7:     "age": 19
8:   }
9: ]

```

```
return repository.findById(studentId).orElse( other: null);
```

.OrElse function - can be used on any optional object

Repository.save() returns saved values

```

38
39     @DeleteMapping("/students/{student-id}")
40     @ResponseStatus(HttpStatus.OK) //Sets response code
41     public void delete(@PathVariable("student-id") String studentId) {
42         repository.deleteStudentById(studentId);
43     }
44
45
46

```

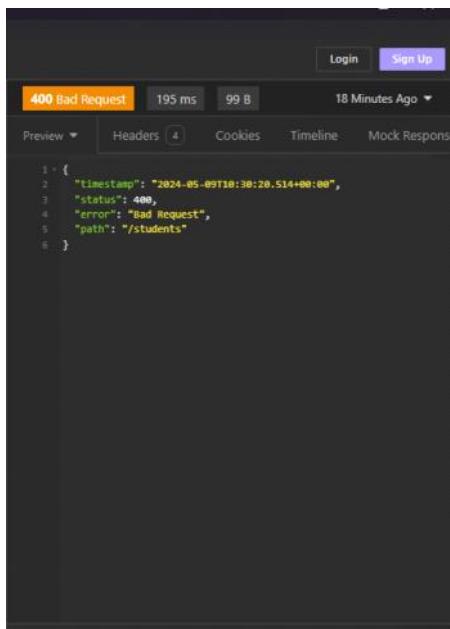
ResponseStatus set given no exceptions

Raising validation Exceptions and General Exceptions + ForEach

09 May 2024 11:35

Creating a method to handle our exception.

By default if validation fails user gets:



Produced this exception:

```
00 WARN 72440 --- [nio-8080-exec-1] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved [org.springframework.web.bind.MethodArgumentNotValidException:
```

Exception handler:

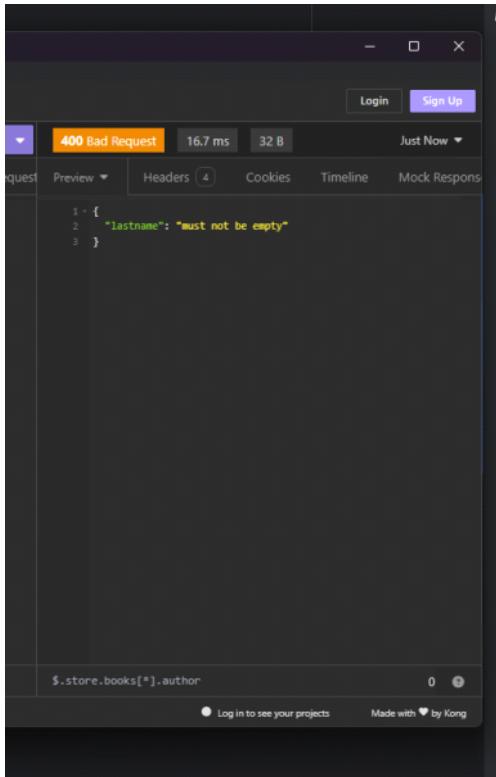
```
@ExceptionHandler  
//Every time this exception is thrown, this code will be called  
public ResponseEntity<?> MethodArgumentNotValidException( //Question Mark means of return a ResponseEntity of any type  
                                         MethodArgumentNotValidException exp  
) {  
    HashMap<String, String> errors = new HashMap<>();  
    exp.getBindingResult().getAllErrors().forEach(error -> {  
        var fieldName = ((FieldError) error).getField();  
        var errorMessage = error.getDefaultMessage();  
        errors.put(fieldName, errorMessage);  
    });  
    return new ResponseEntity<>(errors, HttpStatus.BAD_REQUEST);  
}
```

ResponseEntity<?> - Means its gonna return a ResponseEntity of some type

Exp.getBindingResult().getAllErrors() - get the exception results then get all the errors in a list

.forEach - for each one in the list do this lambda

Makes:



Evaluations in intelIJ

11 May 2024 06:32

The screenshot shows the IntelliJ IDEA interface with the 'Evaluate' tool window open. The 'Expression' field contains the code `studentService.saveStudent(dto)`. The 'Result' field displays the output `Optional[Student] result = null`, with the word 'Student' highlighted and a yellow circle drawn around it. The IntelliJ UI includes a file tree on the left, a navigation bar at the top, and various toolbars and status bars on the right.

Exceptions

11 May 2024 15:20

YOU ONLY NEED TO BUILD HANDLERS IN YOUR CONTROLLERS TO CATCH EXCEPTIONS NO NEED FOR TRY CATCHES

```
}

//Will catch all dataAccessExceptions (databasing) errors here (no need for try catch within code)
@ExceptionHandler
public ResponseEntity<?> DataAccessException(DataAccessException dataAccessException) {
    return new ResponseEntity<?>(dataAccessException.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
}

}

Vulnerable Dependencies
```

.forEach(DO THIS REFERENCE)

15 May 2024 11:56

.Foreach() can be run on collections (LISTS etc)

Or on streams

```
repository.findByNamedQuery( age: 30).forEach(System.out::println);  
};
```

shortcuts

08 May 2024 08:23

CTRL+B for lookup on basically anything, just download the sources