

Building a backend application

With Java 25 and Spring Boot 4.0

Blogger box

The image shows a blog interface with a light gray background and a white card for each post. At the top right of the card is a 'Write' button with a pencil icon. The first post is titled 'My toolkit for 2024' and discusses its growth from design and productivity to encompassing every aspect of life. It was posted 2 days ago under the category 'Productivity'. The second post is titled 'Unveiling the Power of `private static` in Java: A Deep Dive' and discusses Java's versatility and object-oriented principles, mentioning access modifiers. It was posted 3 days ago under the category 'Java'. The third post is titled 'Modern Git Commands and Features You Should Be Using' and discusses the evolution of git commands. A blue bar is visible at the bottom of the screen.

My toolkit for 2024

I didn't expect to write about my toolkit again so soon since I already made a big update in January 2023. Originally, my toolkit was only about design and productivity when I first talked about it in 2019. Now it's grown to encompass every aspect of my life....

Productivity Posted 2 days ago

Unveiling the Power of `private static` in Java: A Deep Dive

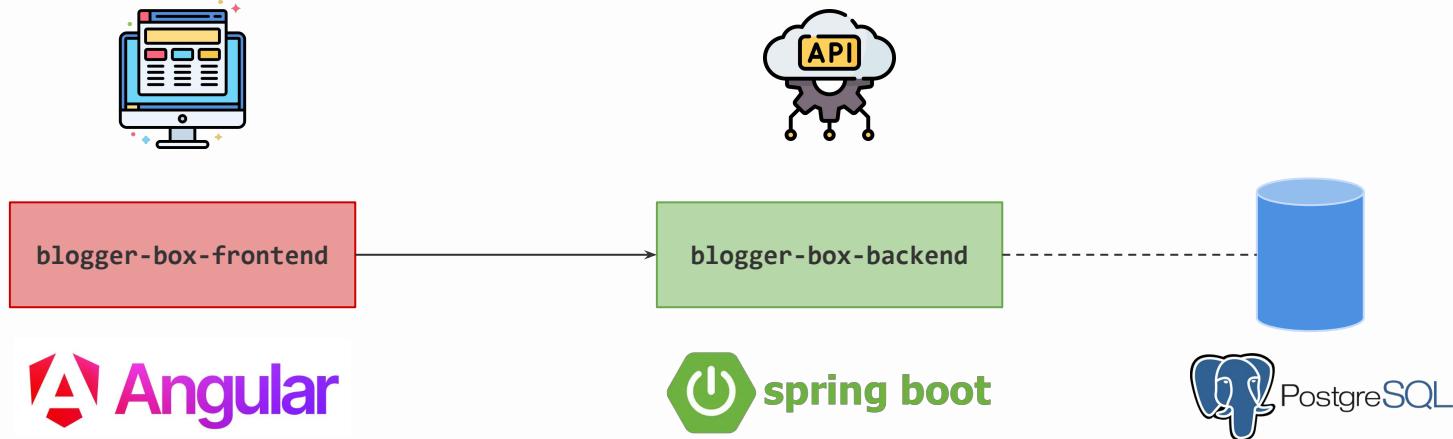
Introduction Java, a language celebrated for its versatility, robustness, and object-oriented principles, provides a rich set of tools for developers. One of its key features is the ability to define variables, methods, and classes with various access modifiers. Among these modifiers,...

Java Posted 3 days ago

Modern Git Commands and Features You Should Be Using

It's not 2005 anymore and git offers more than just add, commit, push and pull. Let's explore all the new, modern git commands, that you should know about — All of us — software engineers — use git every day, however most people only ever touch the mo...

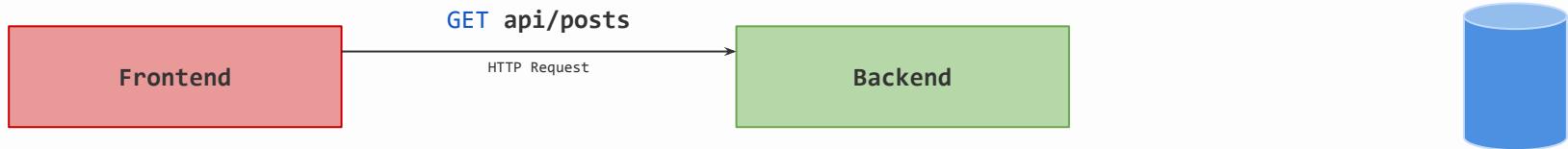
Blogger box architecture



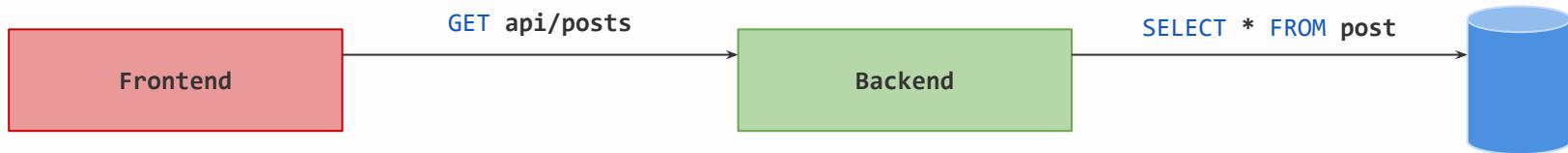
Flow frontend <-> backend



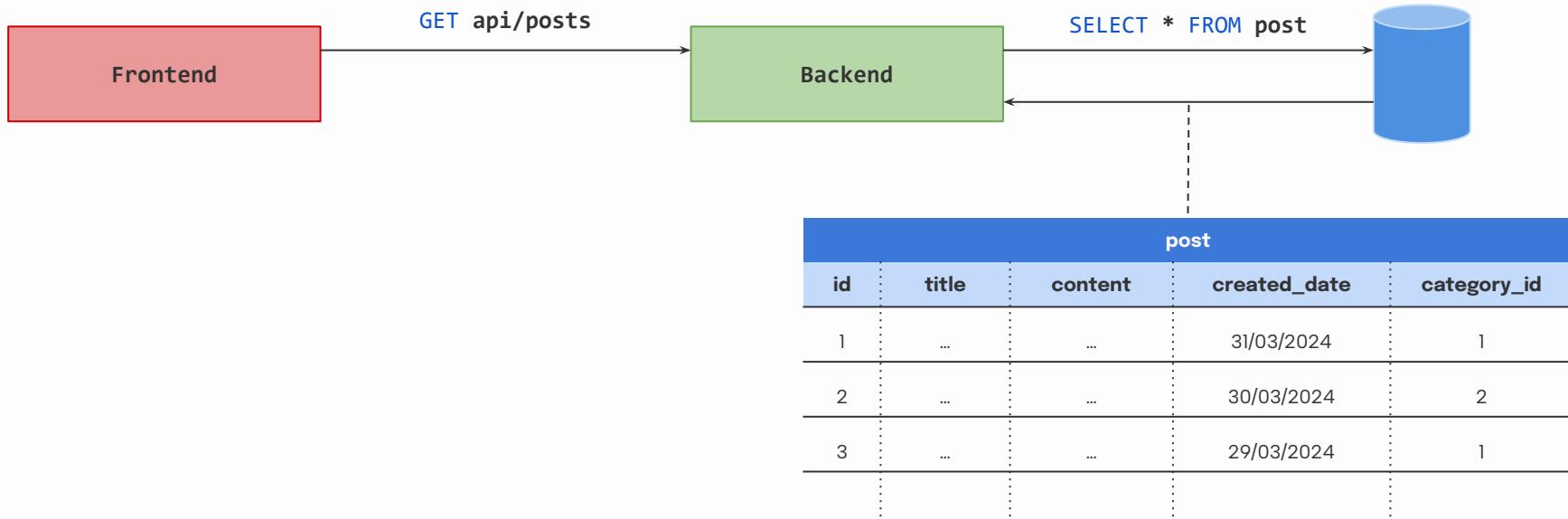
Flow frontend <-> backend



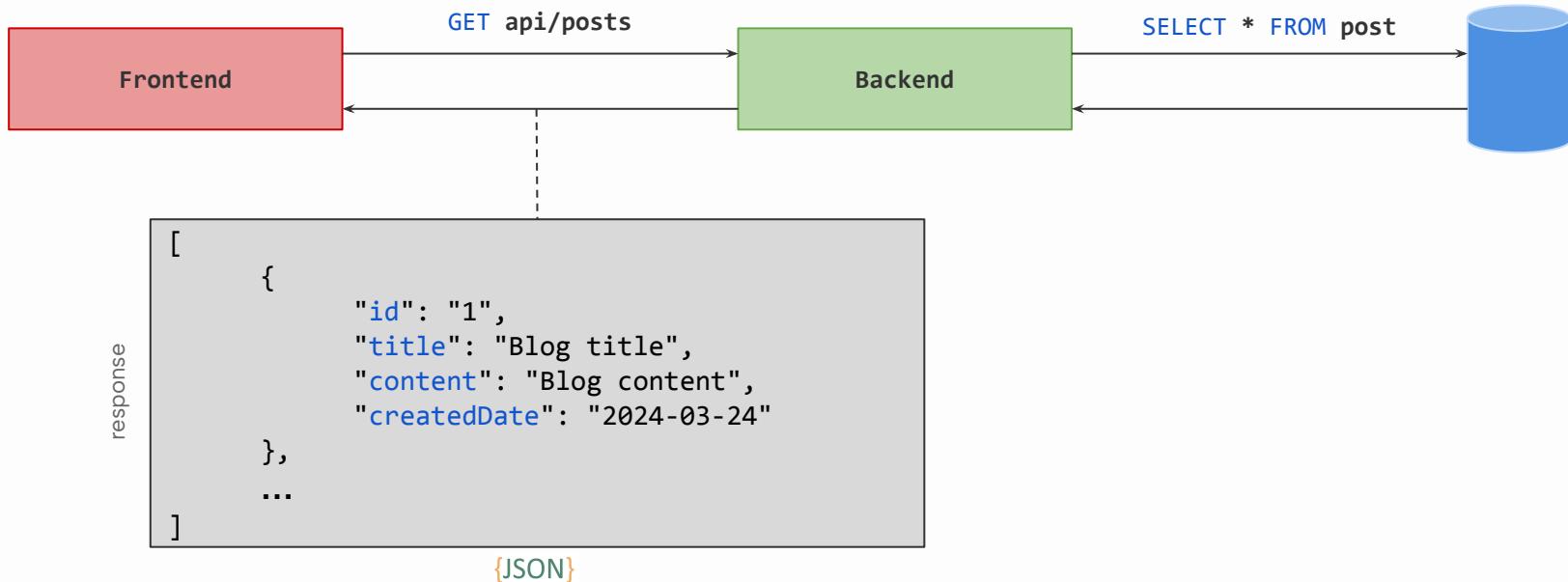
Flow frontend <-> backend



Flow frontend <-> backend



Flow frontend <-> backend



Session 03

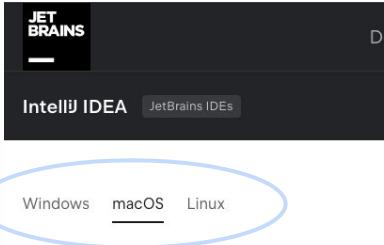
- Create a spring boot application **blogger-box-backend**
- Publish project on **Github** Repository
- Expose our first **endpoints**
- Document your endpoints with **Swagger**
- HTTP request **methods** with **conventions**
- Expose all **endpoints** for a blogger platform

Session 04

- Create a **database** remotely
- Connect backend to a **database** via **JPA**
- **Http code**
- **Exception** handling



Download a Java IDE (if not yet done)

| | |
|---------------|--|
| Step 1 | Head over to jetbrains.com/idea/download |
| Step 2 | Select your OS (Windows, macOS or Linux)  |
| Step 3 | Download IntelliJ IDEA Community Edition (free)  |

Create spring boot app

Head over to start.spring.io to create your spring boot application

| | |
|-------------------------|----------------------|
| Project | Maven |
| Language | Java |
| Spring Boot | 4.0.2 |
| Project Metadata | |
| Group | com.dauphine |
| Artifact | blogger-box-backend |
| Name | Blogger Box Backend |
| Description | Blogger Box Backend |
| Package name | com.dauphine.blogger |
| Packaging | Jar |
| Java | 25 |
| Dependencies | Spring Web |

Generate and unzip project
Place project in your workspace

Users > elie > Workspace > dauphine > blogger-box-backend

Project

 Gradle - Groovy Gradle - Kotlin Maven

Language

 Java Kotlin Groovy

Spring Boot

 4.1.0 (SNAPSHOT) 4.1.0 (M1) 4.0.3 (SNAPSHOT) 4.0.2
 3.5.11 (SNAPSHOT) 3.5.10

Project Metadata

| | | | |
|---------------|---|----------------------------|--------------------------|
| Group | com.dauphine | Artifact | blogger-box-backend |
| Name | blogger-box-backend | | |
| Description | Blogger Box Backend | | |
| Package name | com.dauphine.blogger | | |
| Packaging | <input checked="" type="radio"/> Jar | <input type="radio"/> War | |
| Configuration | <input checked="" type="radio"/> Properties | <input type="radio"/> YAML | |
| Java | <input checked="" type="radio"/> 25 | <input type="radio"/> 21 | <input type="radio"/> 17 |

Dependencies

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Create spring boot app

Head over to start.spring.io to create your spring boot application

| | |
|-------------------------|----------------------|
| Project | Maven |
| Language | Java |
| Spring Boot | 4.0.2 |
| Project Metadata | |
| Group | com.dauphine |
| Artifact | blogger-box-backend |
| Name | Blogger Box Backend |
| Description | Blogger Box Backend |
| Package name | com.dauphine.blogger |
| Packaging | Jar |
| Java | 25 |
| Dependencies | Spring Web |

Generate and unzip project
Place project in your workspace

 Users >  elie >  Workspace >  dauphine >  blogger-box-backend

Don't forget to add the dependency!

Project
 Gradle - Groovy Gradle - Kotlin Maven

Language
 Java Kotlin Groovy

Spring Boot
 4.1.0 (SNAPSHOT) 4.1.0 (M1) 4.0.3 (SNAPSHOT) 4.0.2
 3.5.11 (SNAPSHOT) 3.5.10

Project Metadata

| | | | |
|---------------|---|----------------------------|--------------------------|
| Group | com.dauphine | Artifact | blogger-box-backend |
| Name | blogger-box-backend | | |
| Description | Blogger Box Backend | | |
| Package name | com.dauphine.blogger | | |
| Packaging | <input checked="" type="radio"/> Jar | <input type="radio"/> War | |
| Configuration | <input checked="" type="radio"/> Properties | <input type="radio"/> YAML | |
| Java | <input checked="" type="radio"/> 25 | <input type="radio"/> 21 | <input type="radio"/> 17 |

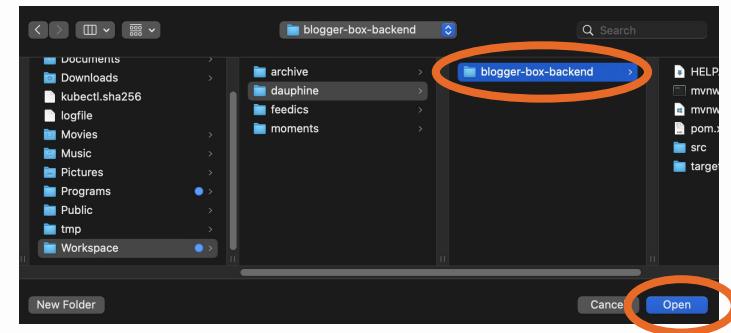
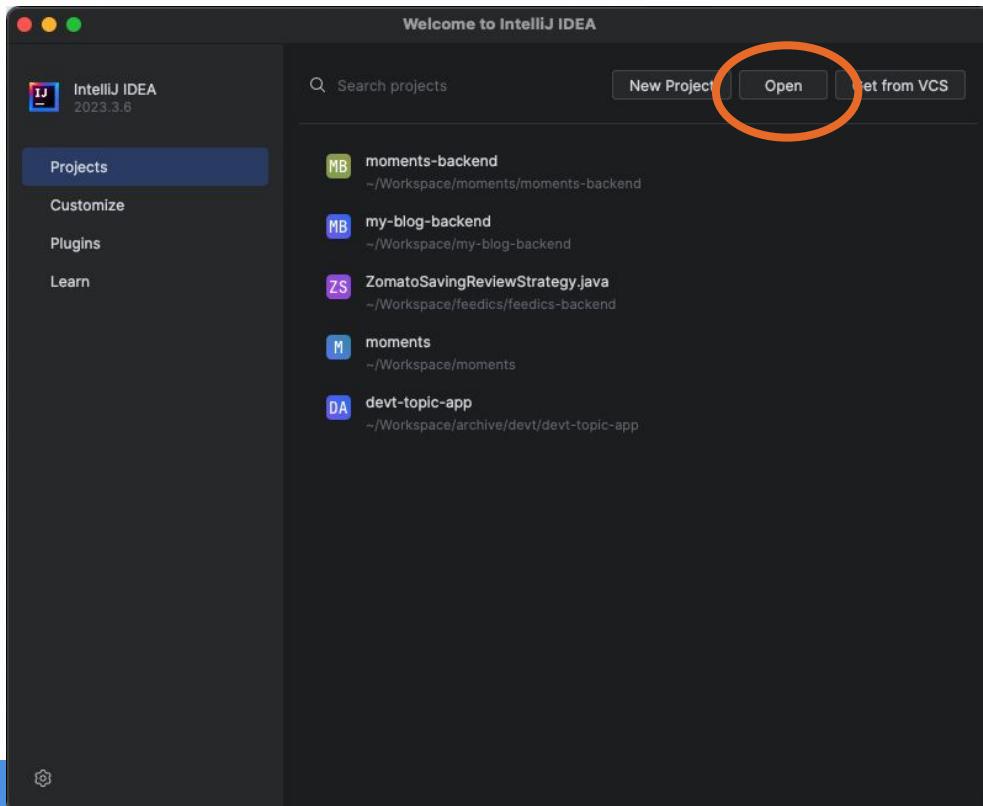
Dependencies **ADD ...**

Spring Web WEB

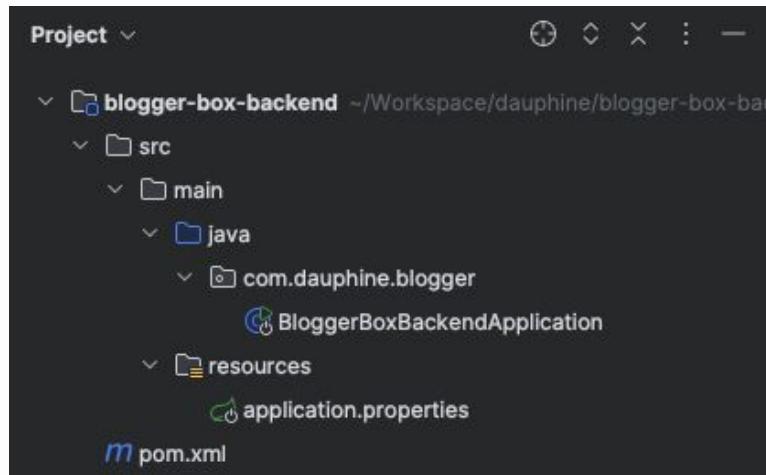
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

</>

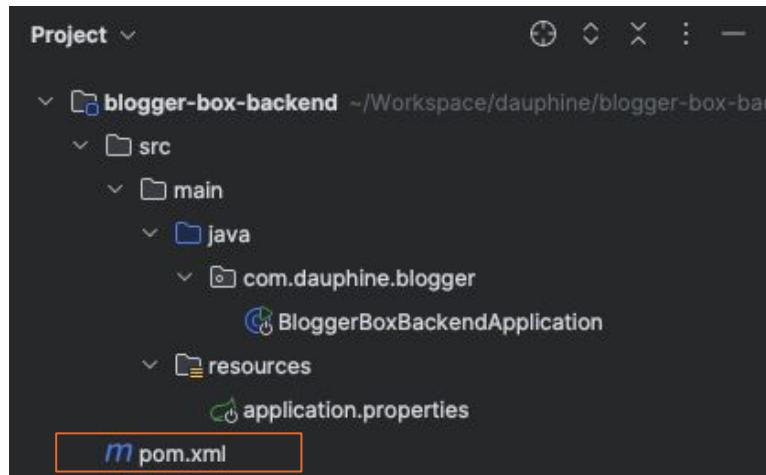
Open in IDE



Structure



Structure



pom.xml

pom.xml is a configuration file used by **Maven**, and will contain :

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven
5       4.0.0.xsd" modelVersion="4.0.0"> Compatible with Maven 3
6
7   <parent>
8     <groupId>org.springframework.boot</groupId>
9     <artifactId>spring-boot-starter-parent</artifactId>
10    <version>4.0.2</version>
11    <relativePath/>
12  </parent>
13
14  <groupId>com.dauphine</groupId>
15  <artifactId>blogger-box-backend</artifactId>
16  <version>0.0.1-SNAPSHOT</version>
17  <name>blogger-box-backend</name>
18  <description>Blogger Box Backend</description>
19
20  <properties>
21    <java.version>25</java.version>
22  </properties>
23
24  <dependencies> ⚡ Add Starters...
25  ⚡
26
27
28  ⚡
29  ⚡
30
31
32
33
34  <dependency>
35    <groupId>org.springframework.boot</groupId>
36    <artifactId>spring-boot-starter-webmvc</artifactId>
37  </dependency>
38  <dependency>
39    <groupId>org.springframework.boot</groupId>
40    ⚡
41    <artifactId>spring-boot-starter-webmvc-test</artifactId>
42    <scope>test</scope>
43  </dependency>
44 </dependencies>
45
46  <build>
47    <plugins>
48      <plugin>
49        <groupId>org.springframework.boot</groupId>
50        <artifactId>spring-boot-maven-plugin</artifactId>
51      </plugin>
52    </plugins>
53  </build>
```

pom.xml

pom.xml is a configuration file used by **Maven**, and will contain :

Project information : contains metadata details such as the project's **groupId**, **artifactId**, **version** and **name**

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven_4_0_0.xsd">
5   <modelVersion>4.0.0</modelVersion> Compatible with Maven 3
6
7   <parent>
8     <groupId>org.springframework.boot</groupId>
9     <artifactId>spring-boot-starter-parent</artifactId>
10    <version>4.0.2</version>
11    <relativePath/>
12  </parent>
13
14  <groupId>com.dauphine</groupId>
15  <artifactId>blogger-box-backend</artifactId>
16  <version>0.0.1-SNAPSHOT</version>
17  <name>blogger-box-backend</name>
18  <description>Blogger Box Backend</description>
19
20  <properties>
21    <java.version>25</java.version>
22  </properties>
23
24  <dependencies> ⚡ Add Starters...
25  <dependency>
26    <groupId>org.springframework.boot</groupId>
27    <artifactId>spring-boot-starter-webmvc</artifactId>
28  </dependency>
29  <dependency>
30    <groupId>org.springframework.boot</groupId>
31    <artifactId>spring-boot-starter-webmvc-test</artifactId>
32    <scope>test</scope>
33  </dependency>
34 </dependencies>
35
36  <build>
37    <plugins>
38      <plugin>
39        <groupId>org.springframework.boot</groupId>
40        <artifactId>spring-boot-maven-plugin</artifactId>
41      </plugin>
42    </plugins>
43  </build>
```

pom.xml

pom.xml is a configuration file used by **Maven**, and will contain :

Project information

Dependencies : contains all external libraries and framework that the project relies on

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven
5       4.0.0.xsd"> Compatible with Maven 3
6
7 <parent>
8   <groupId>org.springframework.boot</groupId>
9   <artifactId>spring-boot-starter-parent</artifactId>
10  <version>4.0.2</version>
11  <relativePath/>
12 </parent>
13
14 <groupId>com.dauphine</groupId>
15 <artifactId>blogger-box-backend</artifactId>
16 <version>0.0.1-SNAPSHOT</version>
17 <name>blogger-box-backend</name>
18 <description>Blogger Box Backend</description>
19
20 <properties>
21   <java.version>25</java.version>
22 </properties>
23
24 <dependencies> ⚡ Add Starters...
25 ⚡
26   <dependency>
27     <groupId>org.springframework.boot</groupId>
28     <artifactId>spring-boot-starter-webmvc</artifactId>
29 ⚡
30   </dependency>
31   <dependency>
32     <groupId>org.springframework.boot</groupId>
33     <artifactId>spring-boot-starter-webmvc-test</artifactId>
34     <scope>test</scope>
35   </dependency>
36 </dependencies>
37
38 <build>
39
40 ⚡
41   <plugins>
42     <plugin>
43       <groupId>org.springframework.boot</groupId>
44       <artifactId>spring-boot-maven-plugin</artifactId>
45     </plugin>
46   </plugins>
47
48 </build>
```

pom.xml

pom.xml is a configuration file used by **Maven**, and will contain :

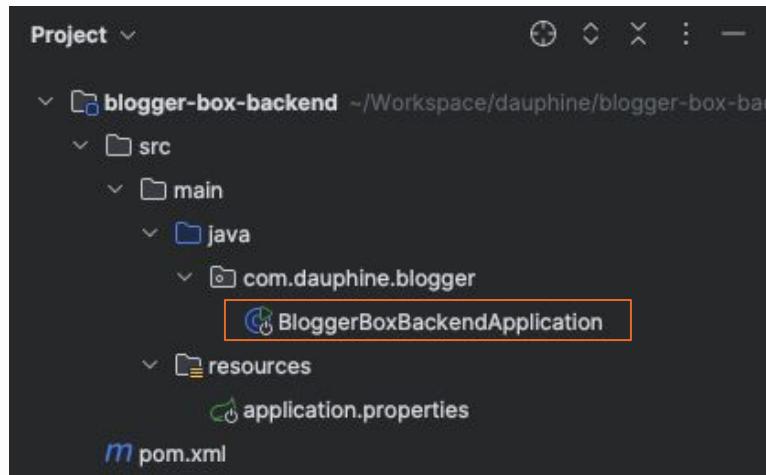
Project information

Dependencies

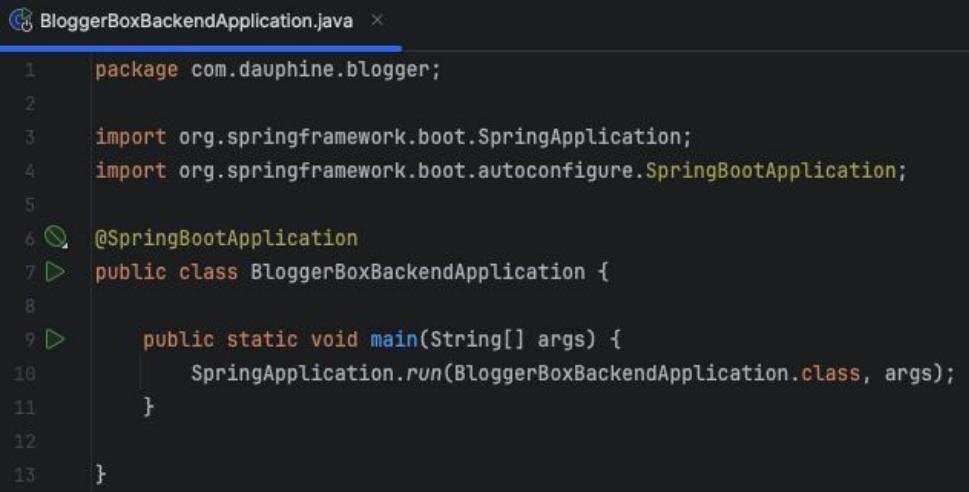
Build configuration : contains how the project is compiled, tested and packaged

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven
5   4.0.0.xsd"> Compatible with Maven 3
6
7 <parent>
8   <groupId>org.springframework.boot</groupId>
9   <artifactId>spring-boot-starter-parent</artifactId>
10  <version>4.0.2</version>
11  <relativePath/>
12 </parent>
13
14 <groupId>com.dauphine</groupId>
15 <artifactId>blogger-box-backend</artifactId>
16 <version>0.0.1-SNAPSHOT</version>
17 <name>blogger-box-backend</name>
18 <description>Blogger Box Backend</description>
19
20 <properties>
21   <java.version>25</java.version>
22 </properties>
23
24 <dependencies> ⚡ Add Starters...
25 ⚡
26   <dependency>
27     <groupId>org.springframework.boot</groupId>
28     <artifactId>spring-boot-starter-webmvc</artifactId>
29 ⚡
30   </dependency>
31   <dependency>
32     <groupId>org.springframework.boot</groupId>
33     <artifactId>spring-boot-starter-webmvc-test</artifactId>
34     <scope>test</scope>
35   </dependency>
36 </dependencies>
37
38
39
40 ⚡
41 <build>
42   <plugins>
43     <plugin>
44       <groupId>org.springframework.boot</groupId>
45       <artifactId>spring-boot-maven-plugin</artifactId>
46     </plugin>
47   </plugins>
48 </build>
```

Structure



Spring boot application (main)



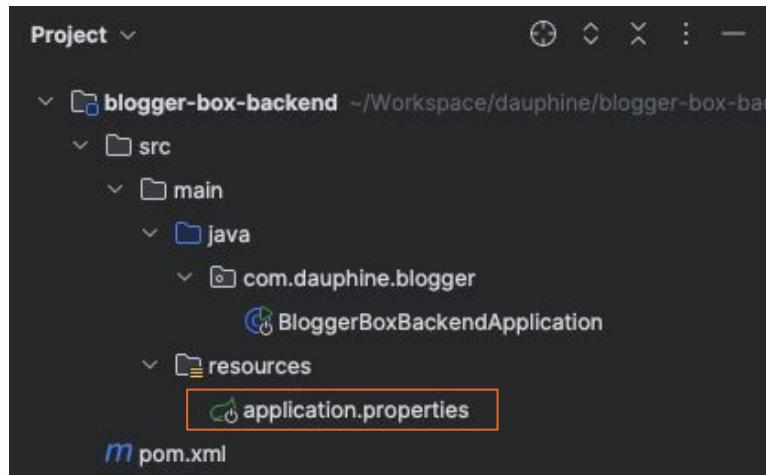
A screenshot of a code editor showing a Java file named `BloggerBoxBackendApplication.java`. The code defines a main entry point for a Spring Boot application:

```
1 package com.dauphine.blogger;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class BloggerBoxBackendApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(BloggerBoxBackendApplication.class, args);
11     }
12 }
13 }
```

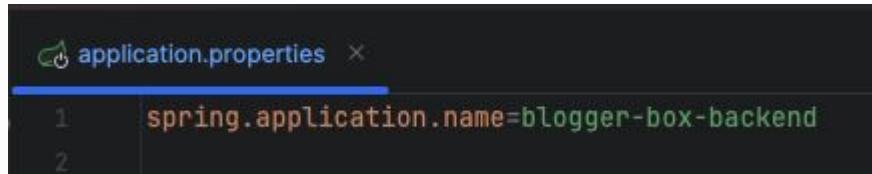
`BloggerBoxBackendApplication` contain the entry point of the application.

The annotation `@SpringBootApplication` allows to **auto configure** the application and will **start** an embedded server (by default Tomcat) and will **run** the application

Structure



Application properties



A screenshot of a code editor showing the `application.properties` file. The file contains the following configuration:

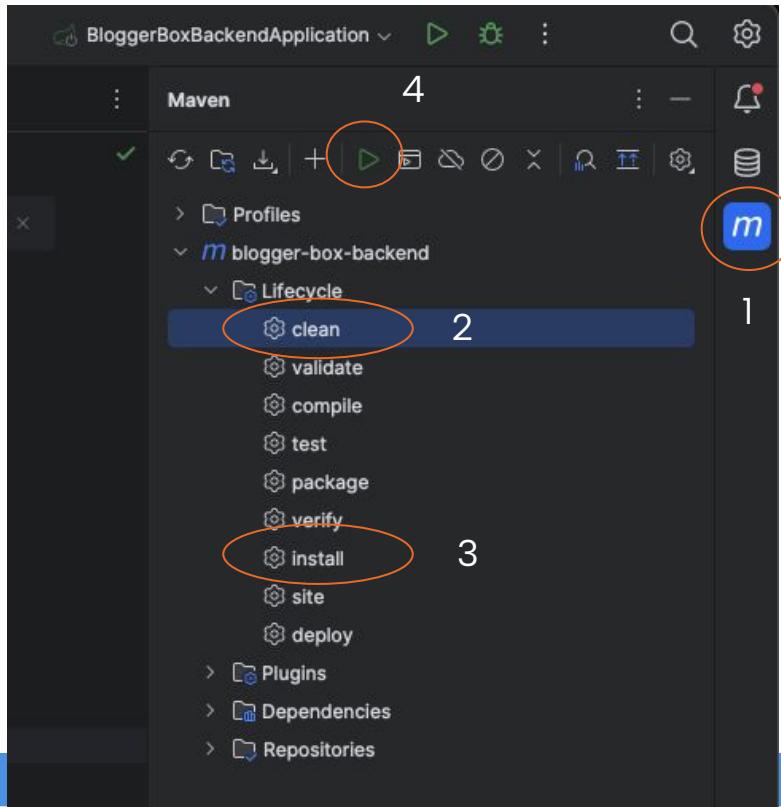
```
1 spring.application.name=blogger-box-backend  
2
```

The `application.properties` file is a configuration file used to configure various aspects of the application.

It will hold properties, which will control behaviors such as database connection setting, server port, logging level, etc...

It provides a way to externalize configuration from the codebase, which is useful for deploying the same application in different environments.

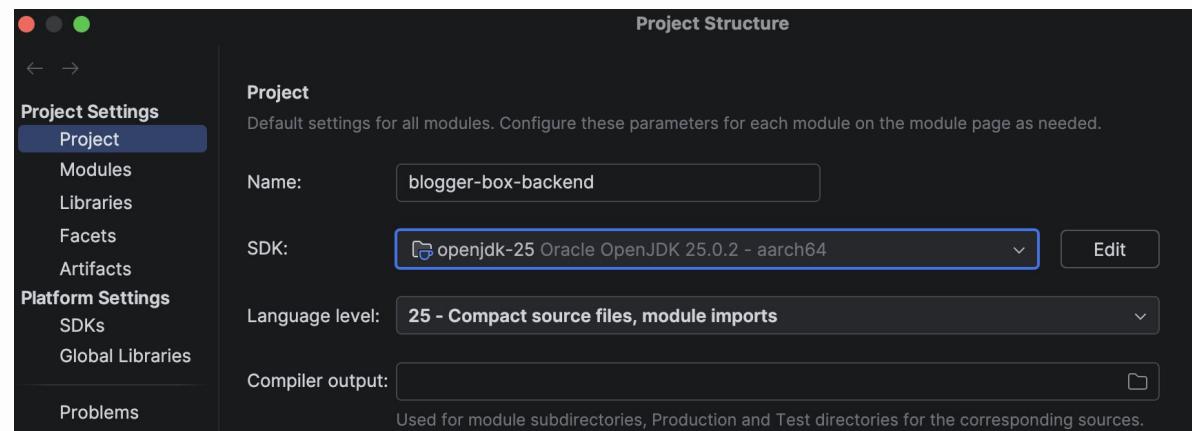
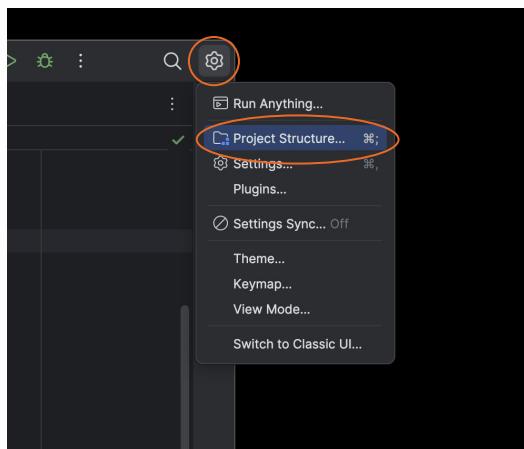
Compile



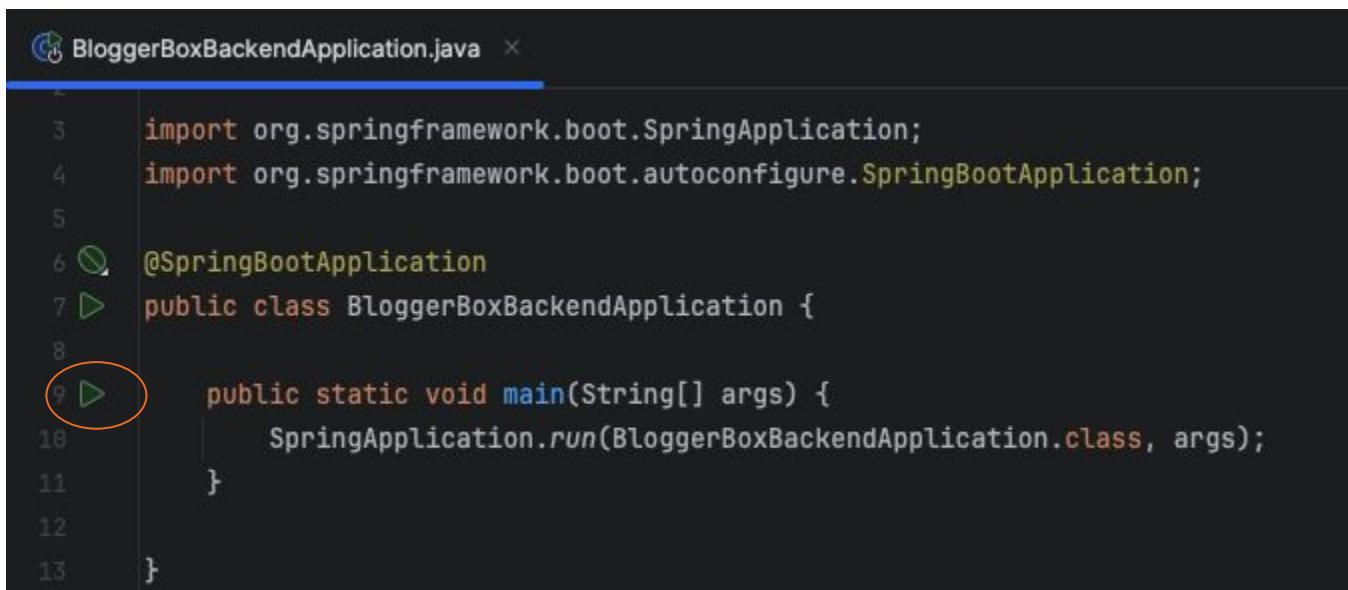
```
[INFO] Replacing main artifact /Users/elie/Workspace/2025-elie/blogger-box-backend/target/blogger-box-backend-3.1.3.jar with /Users/elie/Workspace/2025-elie/blogger-box-backend/target/blogger-box-backend-3.1.3.jar
[INFO] The original artifact has been renamed to /Users/elie/Workspace/2025-elie/blogger-box-backend-3.1.3.jar
[INFO]
[INFO] --- install:3.1.3:install (default-install) @ blogger-box-backend ---
[INFO] Installing /Users/elie/Workspace/2025-elie/blogger-box-backend/pom.xml to /Users/elie/Workspace/2025-elie/blogger-box-backend/target/blogger-box-backend-3.1.3.pom
[INFO] Installing /Users/elie/Workspace/2025-elie/blogger-box-backend/target/blogger-box-backend-3.1.3.jar to /Users/elie/Workspace/2025-elie/blogger-box-backend/target/blogger-box-backend-3.1.3.jar
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time:  1.007 s
[INFO] Finished at: 2025-03-16T08:43:19+01:00
[INFO]
```

Compile

In case you are facing an issue with compiling you might want to check your sdk version (it should be version 25)



Start application



```
BloggerBoxBackendApplication.java ×

  3 import org.springframework.boot.SpringApplication;
  4 import org.springframework.boot.autoconfigure.SpringBootApplication;
  5
  6 @SpringBootApplication
  7 public class BloggerBoxBackendApplication {
  8
  9     public static void main(String[] args) {
 10         SpringApplication.run(BloggerBoxBackendApplication.class, args);
 11     }
 12
 13 }
```

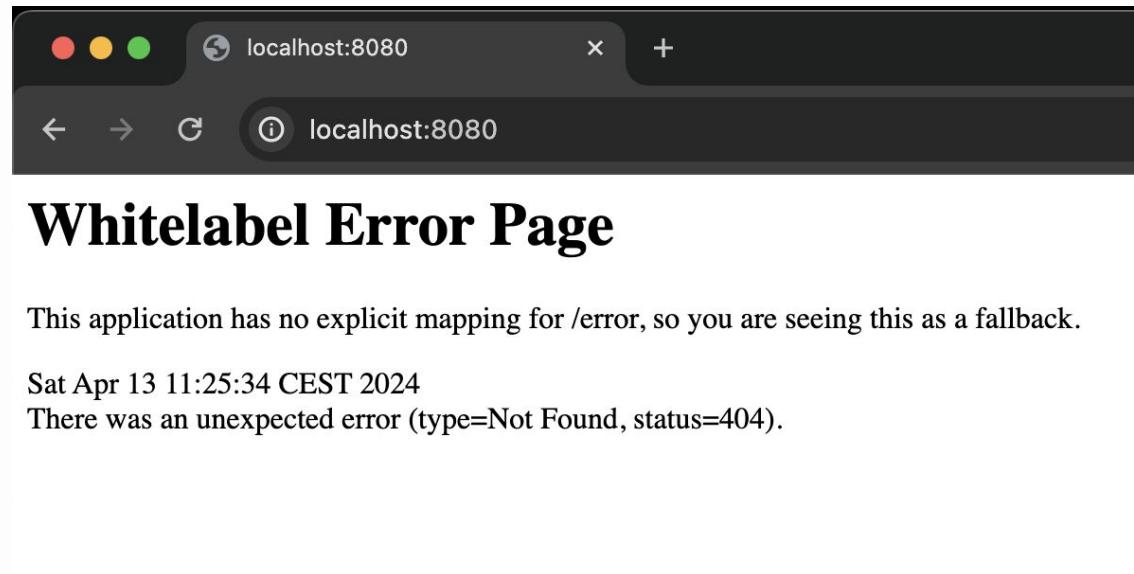
Application started

The application is up and running on port 8080 → <http://localhost:8080>

localhost:8080

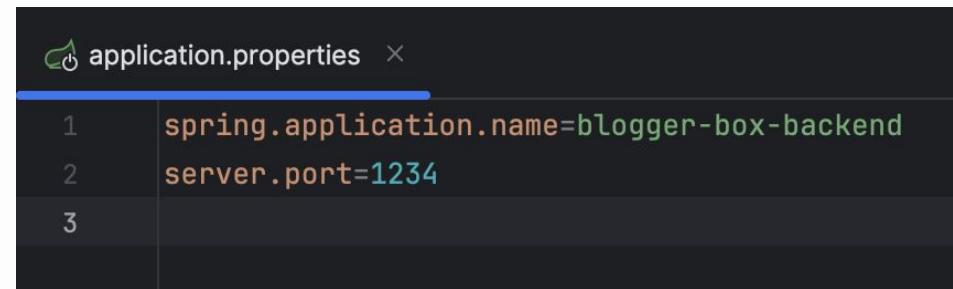
It's normal to have a Whitelabel Error page, since nothing was exposed yet!

The backend application is able to run 😊



localhost:1234

You can modify the port in the **application.properties** configuration file to whatever port that you want



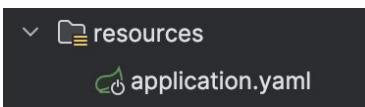
A screenshot of a code editor showing the `application.properties` file. The file contains the following configuration:

```
1 spring.application.name=blogger-box-backend
2 server.port=1234
3
```

.yaml vs .properties for properties

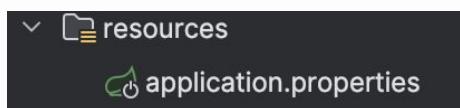
You can use .yaml file instead of .properties.

YAML is a convenient format for specifying hierarchical configuration data.



```
application.yaml
```

```
spring:
  application:
    name: blogger-box-backend
server:
  port: 1234
```



```
application.properties
```

```
spring.application.name=blogger-box-backend
server.port=1234
```

Using application.yml vs application.properties in Spring Boot

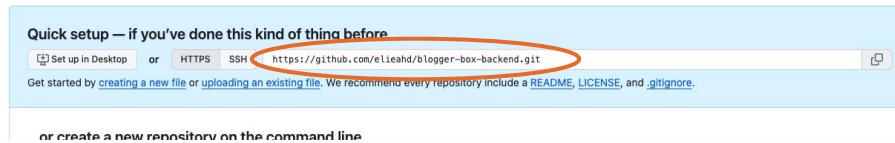
Publish to github

Create a new github repository : **blogger-box-backend**

do not initialize the new repository with README, license or gitignore files

Copy git url

will be used in the next slide



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Required fields are marked with an asterisk (*).

| | |
|---|-----------------------|
| Owner * | Repository name * |
| elieahd | / blogger-box-backend |
| <input checked="" type="checkbox"/> blogger-box-backend is available. | |

Great repository names are short and memorable. Need inspiration? How about `potential-goggles` ?

Description (optional)

Public
Anyone on the internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file
This is where you can write a long description for your project. [Learn more about READMEs](#).

Add .gitignore

Choose which files not to track from a list of templates. [Learn more about ignoring files](#).

Choose a license

A license tells others what they can and can't do with your code. [Learn more about licenses](#).

You are creating a public repository in your personal account.

Create repository



Open terminal and change directory to your project

```
cd Workspace/dauphine/blogger-box-backend
```

Initialize git repository

```
git init
```

Add all project files to the staging area

```
git add .
```

Commit to your local repository

```
git commit -m "COMMIT_MESSAGE"
```

Add git url (copied from the previous slide) to your local repository

```
git remote add origin REMOTE_REPOSITORY_URL
```

Push the changes to Github

```
git push -u origin BRANCH_NAME
```

The terminal window shows the following sequence of commands and their output:

```
cd Workspace/dauphine/blogger-box-backend
git init
Initialized empty Git repository in /Users/elie/Workspace/dauphine/blogger-box-backend/.git/
git add .
git commit -m "Init backend project"
[master (root-commit) 64d2fb] Init backend project
11 files changed, 682 insertions(+)
create mode 100644 .DS_Store
create mode 100644 .gitignore
create mode 100644 .mvn/wrapper/maven-wrapper.jar
create mode 100644 .mvn/wrapper/maven-wrapper.properties
create mode 100755 mvnw
create mode 100644 mvnw.cmd
create mode 100644 pom.xml
create mode 100644 src/main/java/com/dauphine/blogger/BloggerBoxBackendApplication.java
create mode 100644 src/main/java/com/dauphine/blogger/controllers/HelloWorldController.java
create mode 100644 src/main/resources/application.properties
create mode 100644 src/test/java/com/dauphine/blogger/BloggerBoxBackendApplicationTests.java
git remote add origin https://github.com/elieahd/blogger-box-backend.git
git push -u origin master
Enumerating objects: 28, done.
Counting objects: 100% (28/28), done.
Delta compression using up to 8 threads
Compressing objects: 100% (19/19), done.
Writing objects: 100% (28/28), 63.56 KiB | 10.59 MiB/s, done.
Total 28 (delta 0), reused 0 (delta 0)
To https://github.com/elieahd/blogger-box-backend.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```



Adding locally hosted code to Github

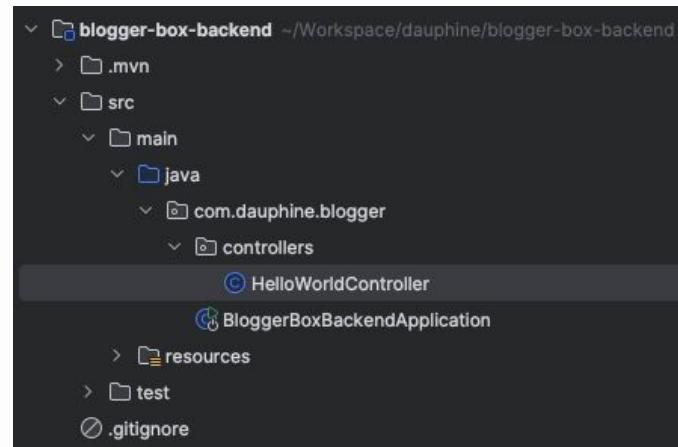
Github

The screenshot shows a GitHub repository page for the project 'blogger-box-backend' owned by user 'elieahd'. The repository is public. At the top, there are navigation links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below the header, there's a search bar and a pin/unwatch button. The repository name 'blogger-box-backend' is displayed again with a Public badge. The main content area shows a master branch with one commit. The commit details are as follows:

| File / Action | Description | Time |
|---------------|----------------------|------------------------|
| elieahd | Init backend project | 64d2fba · 1 minute ago |
| .mvn/wrapper | Init backend project | 1 minute ago |
| src | Init backend project | 1 minute ago |
| .DS_Store | Init backend project | 1 minute ago |
| .aitianore | Init backend project | 1 minute ago |

HelloWorldController

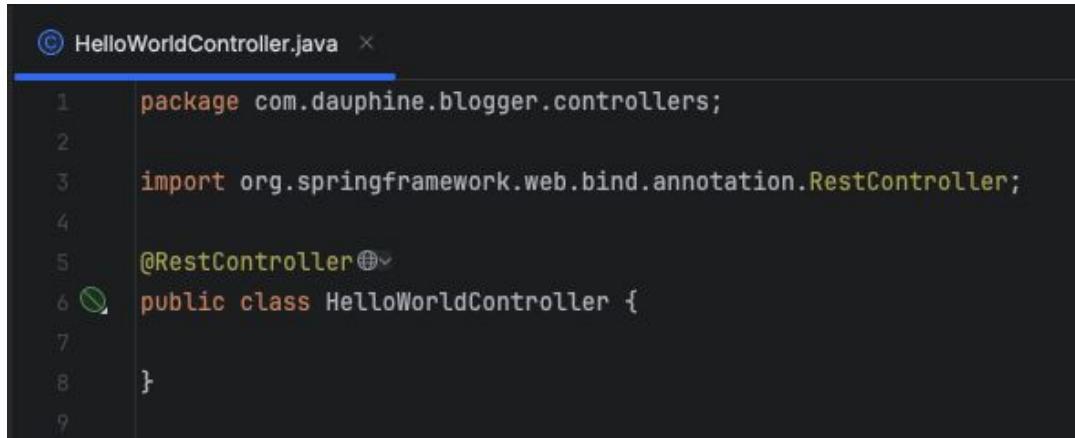
Let's create a new class **HelloWorldController** under controllers



HelloWorldController

`HelloWorldController` will be a Controller which will hold methods that **handle HTTP requests**

So we will annotate the class with `@RestController` which will allow us to automatically **return an HTTP response** (JSON format) in each of the response of the method



```
1 package com.dauphine.blogger.controllers;
2
3 import org.springframework.web.bind.annotation.RestController;
4
5 @RestController
6 public class HelloWorldController {
7
8 }
9
```

Expose an endpoint

Exposing our first **GET** endpoint `/hello-world` with annotation `@GetMapping`

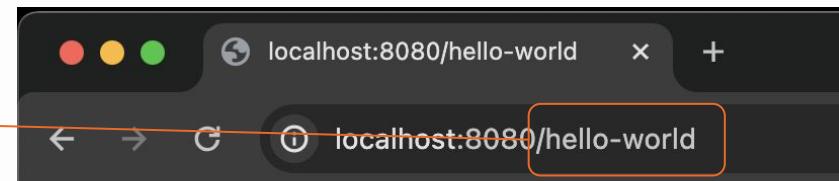
```
package com.dauphine.blogger.controllers;

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

@RestController
public class HelloWorldController {

    @GetMapping("/hello-world")
    public String helloWorld() {
        return "Hello World!";
    }
}
```

We can test GET Http request method in browser



Hello World!

Expose an endpoint with RequestParam

```
package com.dauphine.blogger.controllers;

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

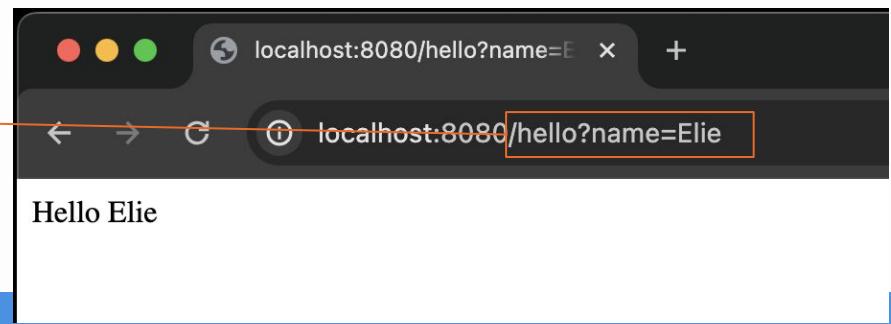
@RestController
public class HelloWorldController {

    @GetMapping("/hello-world")
    public String helloWorld() {
        return "Hello World!";
    }

    @GetMapping("/hello")
    public String helloByName(@RequestParam String name) {
        return "Hello " + name;
    }
}
```

`@RequestParam` allow us to extract query parameter from the URL in form of key-value pairs

We can test GET Http request method in browser



Expose an endpoint with Path Variable

```
package com.dauphine.blogger.controllers;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloWorldController {

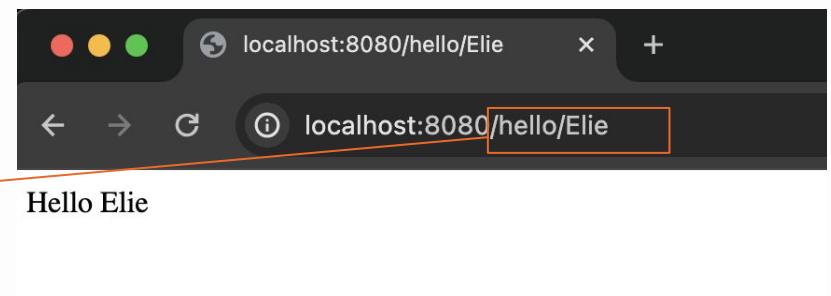
    @GetMapping("hello-world")
    public String helloWorld() {
        return "Hello World!";
    }

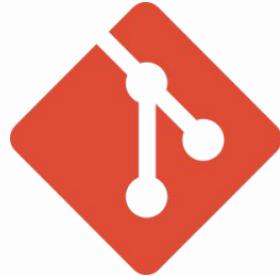
    @GetMapping("hello")
    public String helloByName(@RequestParam String name) {
        return "Hello " + name;
    }

    @GetMapping("hello/{name}")
    public String hello(@PathVariable String name) {
        return "Hello " + name;
    }
}
```

@PathVariable allow us to extract data from the URL path

We can test GET Http request method in browser





Sync with Github

expose my first endpoints

Endpoints

As of now we have exposed the following 3 endpoints

- GET /hello-world
- GET /hello?name={...}
- GET /hello/{name}

The more we evolve our backend, the more we are gonna expose endpoints, hence the need to have a proper **documentation tool**, that is informative, readable and easy to follow

</>

Swagger

Swagger is a tool that allow us to document and test our endpoints

Add following dependency in `pom.xml`

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>3.0.1</version>
</dependency>
```

```
<dependencies> ⚡ Add Starters...
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webmvc</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>3.0.1</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webmvc-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

</>

Swagger

Head over to

<http://localhost:8080/swagger-ui/index.html>

The screenshot shows the Swagger UI interface running in a web browser. The title bar indicates the page is titled "Swagger UI". The address bar shows the URL "localhost:8080/swagger-ui/index.html". The main content area displays an "OpenAPI definition" for version v0, OAS 3.1. The definition is located at the path "/v3/api-docs". A "Servers" dropdown menu is open, showing the URL "http://localhost:8080 - Generated server url". Below this, a section titled "hello-world-controller" lists three API endpoints:

- GET /hello
- GET /hello/{name}
- GET /hello-world



hello-world-controller

GET /hello

GET /hello/{name}

Parameters

Name Description

name * required string (path) Batman

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8080/hello/Batman' \
-H 'accept: */*'
```

Request URL

```
http://localhost:8080/hello/Batman
```

Server response

Code Details

200 Response body Hello Batman

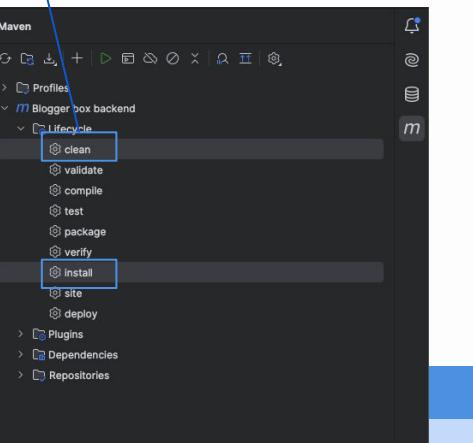
Response headers

Download

Swagger

If you are having issue with that step, that means the dependency was not properly installed

- > Reload project
- > Generate sources and update folders
- > mvn clean install



The screenshot shows the IntelliJ IDEA context menu for the 'pom.xml' file. The 'mvn clean install' option is highlighted with a blue box. The menu also includes other options like 'Reload project' and 'Generate Sources and Update Folders'. A red arrow points from the 'mvn clean install' step in the list above to this menu item.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.4.4</version>
    <relativePath/> 
  </parent>
  <groupId>com.dauphine</groupId>
  <artifactId>blogger-box-backend</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Blogger box backend</name>
  <description>Blogger box backend</description>
</project>
```

Write Application documentation

```
import io.swagger.v3.oas.annotations.OpenAPIDefinition;
import io.swagger.v3.oas.annotations.info.Contact;
import io.swagger.v3.oas.annotations.info.Info;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
@OpenAPIDefinition(
    info = @Info(
        title = "Blogger box backend",
        description = "Blogger box endpoints and apis",
        contact = @Contact(name = "Elie", email = "eliedhr@gmail.com"),
        version = "1.0.0"
    )
)
public class BloggerBoxBackendApplication {

    public static void main(String[] args) {
        SpringApplication.run(BloggerBoxBackendApplication.class, args);
    }
}
```

The screenshot shows a browser window displaying the Swagger UI at `localhost:8080/swagger-ui/index.html`. The URL bar also shows `/v3/api-docs`. The page title is "Blogger box backend 1.0.0 OAS 3.0". Below the title, there is a link to `/v3/api-docs`, a brief description of the API, and a "Contact Elie" link. A red box highlights the main title area. Below this, under the heading "Servers", is a dropdown menu set to `http://localhost:8080 - Generated server url`. The main content area is titled "Hello world API My first hello world endpoints" and lists three API endpoints:

- GET `/hello`
- GET `/hello/{name}`
- GET `/hello-world`

Write Controller documentation

```
3 import io.swagger.v3.oas.annotations.tags.Tag;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.PathVariable;
6 import org.springframework.web.bind.annotation.RequestParam;
7 import org.springframework.web.bind.annotation.RestController;
8
9 @RestController
10 @Tag(
11     name = "Hello world API",
12     description = "My first hello world endpoints"
13 )
14 public class HelloWorldController {
15
16     @GetMapping("/{name}")
17     public String helloWorld() {
18         return "Hello " + name;
19     }
}
```

The screenshot shows the Swagger UI interface running at `localhost:8080/swagger-ui/index.html#/Hello%20world%20API`. The top navigation bar includes the Swagger logo, the URL `/v3/api-docs`, and an `Explore` button. Below the navigation is the title **OpenAPI definition v0 OAS 3.0** with a link to `/v3/api-docs`. A section titled **Hello world API** with the description "My first hello world endpoints" is highlighted with an orange box. Under this section, three API endpoints are listed:

- GET /hello**
- GET /hello/{name}**
- GET /hello-world**

Write Endpoint documentation

```
@GetMapping("hello/{name}")
@Operation(
    summary = "Hello by name endpoint",
    description = "Returns 'Hello {name}' by path variable"
)
public String hello(
    @Parameter(description = "Name to greet")
    @PathVariable String name
) {
    return "Hello " + name;
}
```

Hello world API My first hello world endpoints

GET /hello Hello by name endpoint

GET /hello/{name} Hello by name endpoint

Returns 'Hello {name}' by path variable

Parameters

| Name | Description |
|-------------------------------------|---------------|
| name * required string (path) | Name to greet |
| | name |

Try it out



Sync with Github

add documentation via Swagger

HTTP request methods

| | |
|---------------|---|
| GET | Retrieve data (<i>should not modify data</i>) |
| POST | Create a new resource |
| PUT | Modify/Update an existing resource |
| PATCH | Modify part of an existing resource |
| DELETE | Delete an existing resources |

Best practices

Plural nouns

It helps ensure consistency and better reflects the possibility of the endpoint returning multiple resources



GET /category

GET /post



GET /categories

GET /posts

Best practices

Plural nouns

Separate words with hyphens

Use hyphens (-) to improve the readability of URLs, do not use underscores (_)



GET /managed_devices

GET /myFolders



GET /managed-devices

GET /my-folders

Best practices

Plural nouns

Separate words with hyphens

Use lowercase letters

lowercase letters should be consistently preferred in URI paths



GET /Categories

GET /POSTS



GET /categories

GET /posts

Best practices

Plural nouns

Separate words with hyphens

Use lowercase letters

Use path variables for singleton resource

```
GET /categories
```

*Will return a collection of resource **categories***

```
GET /categories/{id}
```

*Will return a singleton resource **a category***

Best practices

Plural nouns

Separate words with hyphens

Use lowercase letters

Use path variables for singleton resource

Use query param to filter collection



```
GET /categories/search-by-name/{name}
```

```
GET /posts/created-date/{date}
```



```
GET /categories?name={name}
```

```
GET /posts?created-date={date}
```

Best practices

Plural nouns

Separate words with hyphens

Use lowercase letters

Use path variables for singleton resource

Use query param to filter collection

Sub resources

`GET /categories/{id}/posts`

Will return the list of posts per a category

`GET /posts/{id}/categories`

Will return the list of categories per a post

Best practices

Plural nouns

Separate words with hyphens

Use lowercase letters

Use path variables for singleton resource

Use query param to filter collection

Sub resources

Version your endpoints

*helps to easily manage changes and updates to an API
while still maintaining backward compatibility*

`GET /v1/categories`

`GET /v2/categories`

Best practices

Plural nouns

Separate words with hyphens



POST /v1/categories/create

Use lowercase letters



POST /v1/categories

Use path variables for singleton resource

Use query param to filter collection

Sub resources

Version your endpoints

Do not use verbs in the URI

HTTP methods (GET, POST, PUT, DELETE, etc.) are used to perform actions on those resources, effectively acting as verbs

Use cases

Get all categories

```
GET /categories
```

Use cases

Get all categories

```
GET /categories
```

Get category by id

```
GET /categories/{id}
```

Use cases

Get all categories

```
GET /categories
```

Get category by id

```
GET /categories/{id}
```

Get all post of a certain categories

```
GET /categories/{id}/posts
```

Use cases

Get all categories

```
GET /categories
```

Get category by id

```
GET /categories/{id}
```

Get all post of a certain categories

```
GET /categories/{id}/posts
```

Search posts by created date

```
GET /posts?date=20-01-2024
```

Use cases

Get all categories

```
GET /categories
```

Get category by id

```
GET /categories/{id}
```

Get all post of a certain categories

```
GET /categories/{id}/posts
```

Search posts by created date

```
GET /posts?date=20-01-2024
```

Create a new category

```
POST /categories
```

Use cases

Get all categories

```
GET /categories
```

Get category by id

```
GET /categories/{id}
```

Get all post of a certain categories

```
GET /categories/{id}/posts
```

Search posts by created date

```
GET /posts?date=20-01-2024
```

Create a new category

```
POST /categories
```

Update an existing category

```
PUT /categories/{id}
```

Use cases

Get all categories

```
GET /categories
```

Get category by id

```
GET /categories/{id}
```

Get all post of a certain categories

```
GET /categories/{id}/posts
```

Search posts by created date

```
GET /posts?date=20-01-2024
```

Create a new category

```
POST /categories
```

Update an existing category

```
PUT /categories/{id}
```

Update a sub property of an existing category

```
PATCH /categories/{id}
```

Use cases

Get all categories

```
GET /categories
```

Get category by id

```
GET /categories/{id}
```

Get all post of a certain categories

```
GET /categories/{id}/posts
```

Search posts by created date

```
GET /posts?date=20-01-2024
```

Create a new category

```
POST /categories
```

Update an existing category

```
PUT /categories/{id}
```

Update a sub property of an existing category

```
PATCH /categories/{id}
```

Delete a category

```
DELETE /categories/{id}
```

Http method POST

POST request method accept data enclosed in the **body** of the request message to **create** a resource

Example of creation of a new resource

```
@PostMapping(@RequestMapping("/elements"))
public String create(@RequestBody ElementRequest body) {
    // TODO later, implement persistence layer
    // INSERT INTO ... (title, description) VALUES (${title}, ${description});
    return "Create new element with title '%s' and description '%s'"
        .formatted(body.getTitle(), body.getDescription());
}
```

```
public class ElementRequest { 1

    private String title; 2 usag
    private String description;

    // getters and setters ...
}
```

Http method PUT

PUT request method is used to **update/replace** an existing new resource.
It's similar to the POST method, in that it **sends data** to a server,

Example of updating an existing resource

```
@PutMapping("/elements/{id}")
public String update(@PathVariable Integer id,
                     @RequestBody ElementRequest body) {
    // TODO later, implement persistence layer
    // UPDATE ... SET title = ${title}, description = ${description} WHERE id = ${id}
    return "Update element '%s' with title '%s' and description '%s'"
        .formatted(id, body.getTitle(), body.getDescription());
}
```

Http method PATCH

PATCH request method is used to make a **partial changes** in an **existing** resource

Example of patching an existing resource

```
@PatchMapping("/elements/{id}/description")
public String patch(@PathVariable Integer id,
                    @RequestBody String description) {
    // TODO later, implement persistence layer
    // UPDATE ... SET description = ${description} WHERE id = ${id}
    return "Patch element '%s' with description '%s'".formatted(id, description);
}
```

Http method DELETE

DELETE request method is used to **delete** an **existing** resource

Example of deleting an existing resource

```
@DeleteMapping("/elements/{id}")
public String delete(@PathVariable Integer id) {
    // TODO later, implement persistence layer
    // DELETE ... WHERE id = ${id}
    return "Delete element '%s'".formatted(id);
}
```

Blogger box use cases

Identify all use cases/functionalities for the blogger box application

Blogger box use cases

Identify all use cases/functionalities for the blogger box application

- Retrieve all **categories**
- Retrieve a **category** by id
- Create a new **category**
- Update the name of a **category**
- Delete an existing **category**
- Create a new **post**
- Update an existing **post**
- Delete an existing **post**
- Retrieve all **posts** ordered by creation date (to show latest post, in home page)
- Retrieve all **posts** per a category

Blogger box use cases

Functionalities

- Retrieve all **categories**
- Retrieve a **category** by id
- Create a new **category**
- Update the name of a **category**
- Delete an existing **category**
- Create a new **post**
- Update an existing **post**
- Delete an existing **post**
- Retrieve all **posts** ordered by creation date
- Retrieve all **posts** per a category

Expose all endpoints (without implementation)



Don't forget swagger documentation
&
best practices

check slide 34 in Session 01, to get the attributes of a post and a category

Blogger box use cases



Tip #1 split those endpoints into 2 controllers : `CategoryController`, `PostController`



Tip #2 add versioning to endpoint (v1 😊)



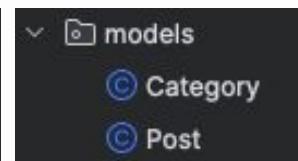
Tip #3 use `@RequestMapping` on the controller to start all endpoints of each controller the same way

```
@RestController  
@RequestMapping("/v1/categories")  
public class CategoryController {  
  
    @RestController  
    @RequestMapping("/v1/posts")  
    public class PostController {
```



Tip #4 differentiate between model and DTO classes

- **Model** classes are used throughout your applications
- **DTO** Data Transfer Object are only used only in controllers
(you can use the same DTO for both creation and update requests, if they share the same attributes)



Blogger box use cases



Tip #5 since we are not gonna implement persistence/dao layer just now, if you want to make the endpoints more interactive, you can add a temporary list of objects in the controller

Example :

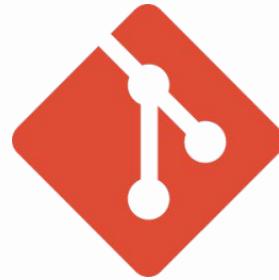
```
@RestController
@RequestMapping(@v"/v1/categories")
public class CategoryController {

    private final List<Category> temporaryCategories;  5 usages

    public CategoryController() {
        temporaryCategories = new ArrayList<>();
        temporaryCategories.add(new Category(UUID.randomUUID(), name: "my first category"));
        temporaryCategories.add(new Category(UUID.randomUUID(), name: "my second category"));
        temporaryCategories.add(new Category(UUID.randomUUID(), name: "my third category"));
    }

    @GetMapping@v
    public List<Category> retrieveAllCategories() {
        return temporaryCategories;
    }
}
```

In the creation endpoint, you can also add to that existing temporary category list



Sync with Github

expose all endpoints

Architecture and Structure

Dependency Injection is a fundamental aspect of the Spring framework, through which the Spring container “**injects**” objects into other objects or “**dependencies**”.

Simply put, this allows for loose coupling of components and moves the responsibility of managing components onto the container.

Architecture and Structure

Let's rely on **dependency Injection** to better structure our code.

We will create a service class for each model :

- CategoryService
- PostService

And each service will hold its use cases / functionalities

CategoryService

Annotating with `@Service` will allow spring to create a bean for `CategoryService`, so we can inject it in another class, like `CategoryController`

```
8  public interface CategoryService { 5 usages 1 implementation
9
10     List<Category> getAll(); 1 usage 1 implementation
11
12     Category getById(UUID id); 2 usages 1 implementation
13
14     Category create(String name); 1 implementation
15
16     Category updateName(UUID id, String name); 1 usage 1 implementation
17
18     boolean deleteById(UUID id); 1 usage 1 implementation
19 }
```

```
④ CategoryServiceImpl.java ×
1  package com.dauphine.blogger.services.impl;
2
3  import com.dauphine.blogger.models.Category;
4  import com.dauphine.blogger.services.CategoryService;
5  import org.springframework.stereotype.Service;
6
7  import java.util.ArrayList;
8  import java.util.List;
9  import java.util.UUID;
10
11  @Service
12  public class CategoryServiceImpl implements CategoryService {
13
14      private final List<Category> temporaryCategories; 9 usages
15
16      public CategoryServiceImpl() {
17          temporaryCategories = new ArrayList<>();
18          temporaryCategories.add(new Category(UUID.randomUUID(), name: "my first category"));
19          temporaryCategories.add(new Category(UUID.randomUUID(), name: "my second category"));
20          temporaryCategories.add(new Category(UUID.randomUUID(), name: "my third category"));
21      }
22
23      @Override no usages
24      public List<Category> getAll() { return temporaryCategories; }
25
26      @Override no usages
27      public Category getById(UUID id) {
28          return temporaryCategories.stream() Stream<Category>
29              .filter(category -> id.equals(category.getId()))
30              .findFirst() Optional<Category>
31              .orElse( other: null);
32      }
33
34      @Override no usages
35      public Category create(String name) {
36          Category category = new Category(UUID.randomUUID(), name);
37          temporaryCategories.add(category);
38          return category;
39      }
40
41      @Override no usages
42      public Category update(UUID id, String newName) {
43          Category category = temporaryCategories.stream() Stream<Category>
44              .filter(c -> id.equals(c.getId()))
45              .findFirst() Optional<Category>
46              .orElse( other: null);
47          if (category != null) {
48              category.setName(newName);
49          }
50          return category;
51      }
52
53      @Override no usages
54      public void deleteById(UUID id) { temporaryCategories.removeIf(category -> id.equals(category.getId())); }
55
56      @Override no usages
57      public void deleteByName(String name) { temporaryCategories.removeIf(category -> name.equals(category.getName())); }
58
59      @Override no usages
60      public void deleteAll() { temporaryCategories.clear(); }
61  }
```

CategoryController

Controller

Management of the REST endpoints

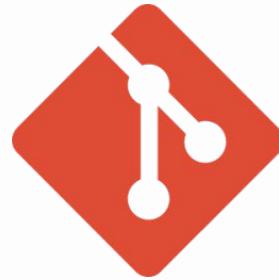
Service

Business Logic Implementations

Inject **CategoryService** in the constructor

The **implementation** and **business logic** of each method will be done at the service layer

```
CategoryController.java x
1 package com.dauphine.blogger.controllers;
2
3 import com.dauphine.blogger.models.Category;
4 import com.dauphine.blogger.models.Post;
5 import com.dauphine.blogger.services.CategoryService;
6 import org.springframework.web.bind.annotation.DeleteMapping;
7 import org.springframework.web.bind.annotation.GetMapping;
8 import org.springframework.web.bind.annotation.PathVariable;
9 import org.springframework.web.bind.annotation.PostMapping;
10 import org.springframework.web.bind.annotation.PutMapping;
11 import org.springframework.web.bind.annotation.RequestBody;
12 import org.springframework.web.bind.annotation.RequestMapping;
13 import org.springframework.web.bind.annotation.RestController;
14
15 import java.util.ArrayList;
16 import java.util.List;
17 import java.util.UUID;
18
19 @RestController
20 @RequestMapping(@v"/v1/categories")
21 public class CategoryController {
22
23     private final CategoryService service;  6 usages
24
25     @Q
26     public CategoryController(CategoryService service) {
27         this.service = service;
28     }
29
30     @GetMapping(@v
31     public List<Category> retrieveAllCategories() {
32         return service.getAll();
33     }
34
35     @GetMapping(@v"{id}")
36     public Category retrieveCategoryById(@PathVariable UUID id) {
37         return service.getById(id);
38     }
39
40     @PostMapping(@v
41     public Category createCategory(@RequestBody String name) {
42         return service.create(name);
43     }
44
45     @PutMapping(@v"{id}")
46     public Category updateCategory(@PathVariable UUID id,
47                                     @RequestBody String name) {
48         return service.update(id, name);
49     }
50
51     @DeleteMapping(@v"{id}")
52     public UUID deleteCategory(@PathVariable UUID id) {
53         return service.deleteById(id);
54     }
55 }
```

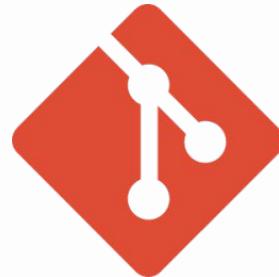


Sync with Github

add `category` service layer

Implement PostService and the changes in the controller

```
8 ④ ↗ public interface PostService { 8 usages 1 implementation
9
10 ④ ↗     List<Post> getAllByCategoryId(UUID categoryId); 1 usage 1 implementation
11
12 ④ ↗     List<Post> getAll(); 1 usage 1 implementation
13
14 ④ ↗     Post getById(UUID id); 2 usages 1 implementation
15
16 ④ ↗     Post create(String title, String content, UUID categoryId); 1 implementation
17
18 ④ ↗     Post update(UUID id, String title, String content); 1 usage 1 implementation
19
20 ④ ↗     boolean deleteById(UUID id); 1 usage 1 implementation
21
22 }
```



Sync with Github

add `post` service layer