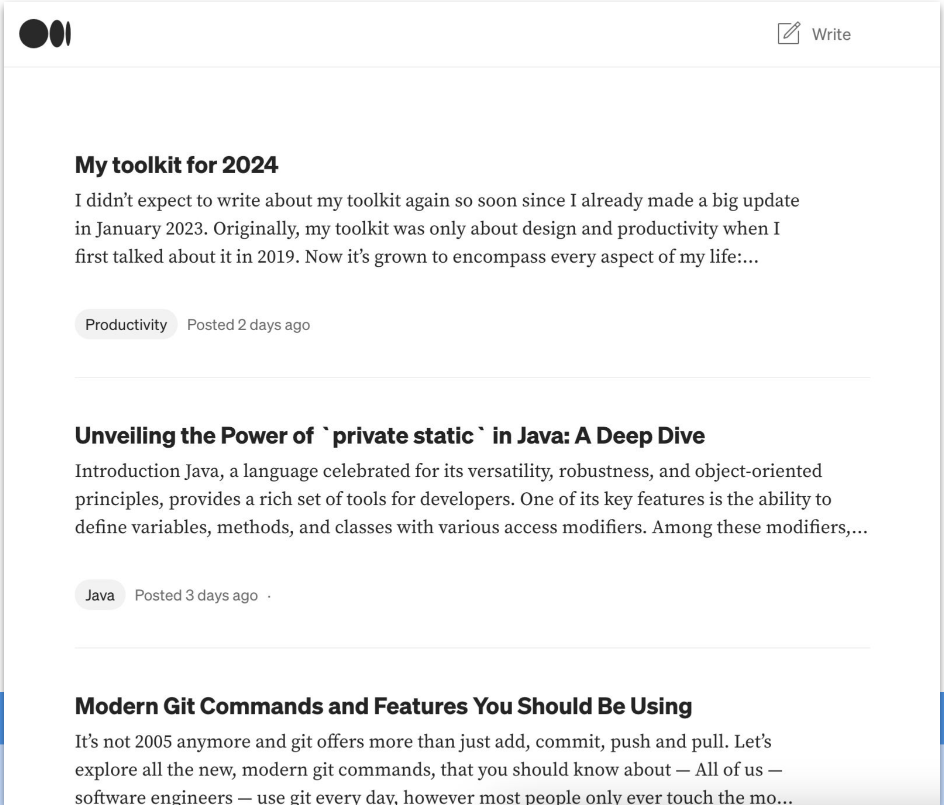
Session 03

# Building a backend application

With Java 21 and Spring Boot 3.2

# Blogger box



**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**

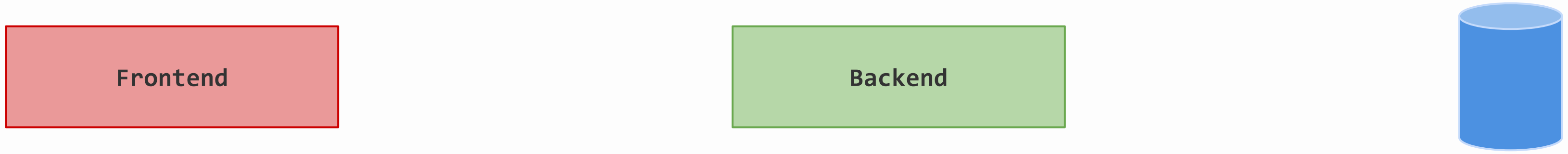
Introduction Java, a language celebrated for its versatility, robustness, and object-oriented principles, provides a rich set of tools for developers. One of its key features is the ability to define variables, methods, and classes with various access modifiers. Among these modifiers,...

Java Posted 3 days ago ·

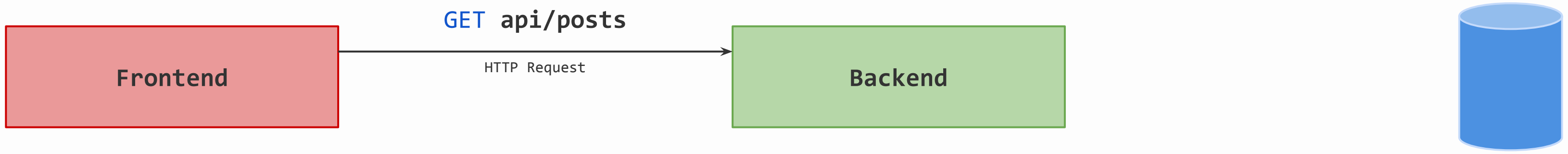
**Modern Git Commands and Features You Should Be Using**

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...

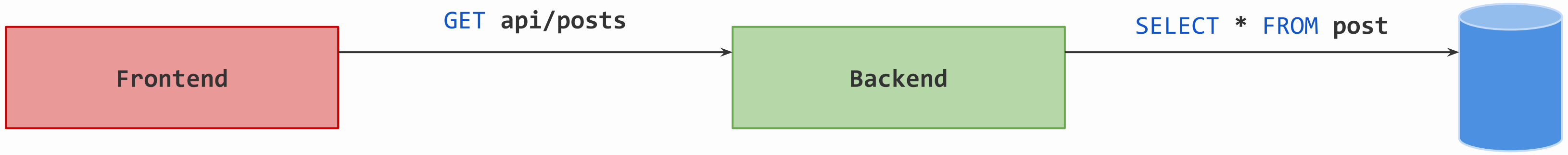
# Flow frontend <-> backend



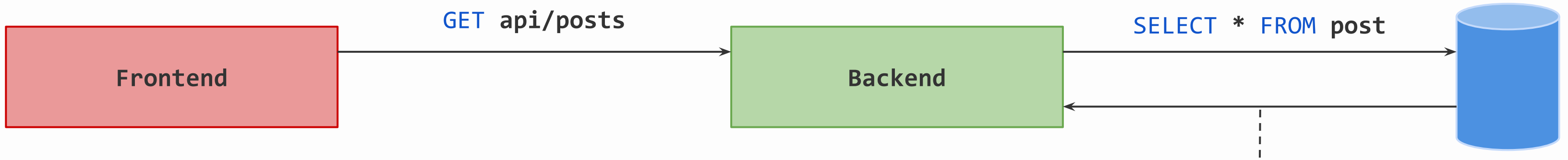
# Flow frontend <-> backend



# Flow frontend <-> backend

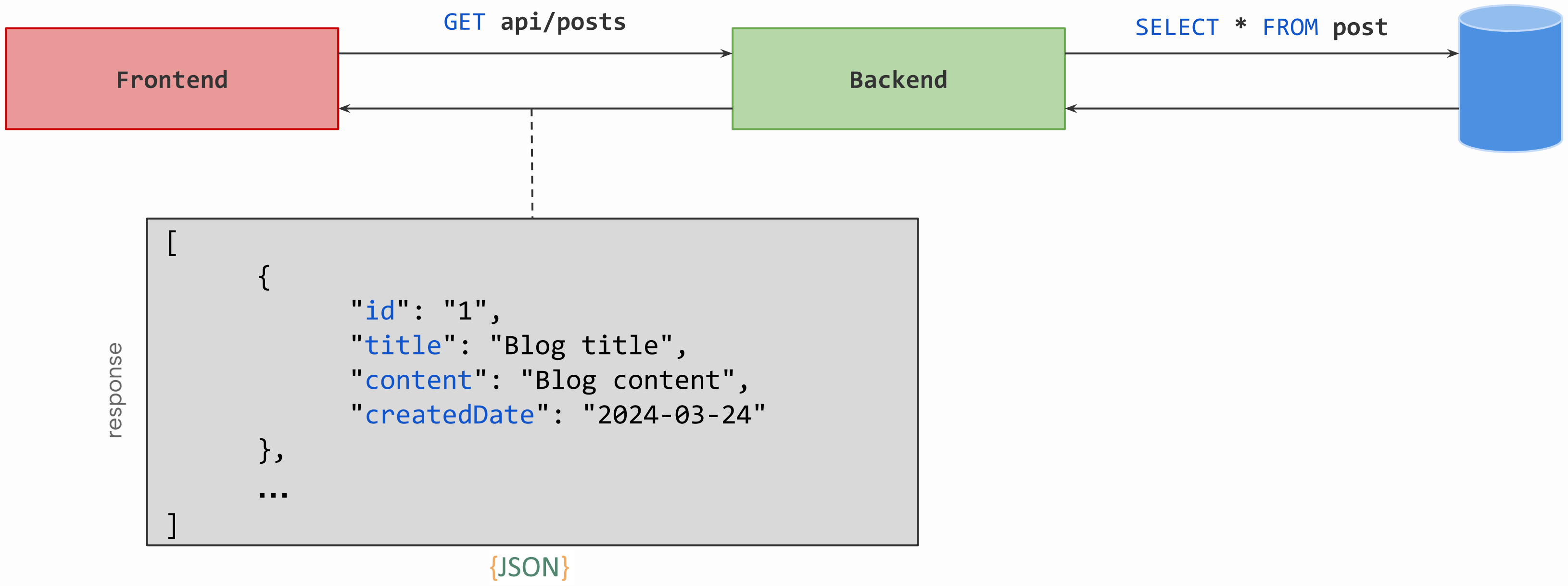


# Flow frontend <-> backend



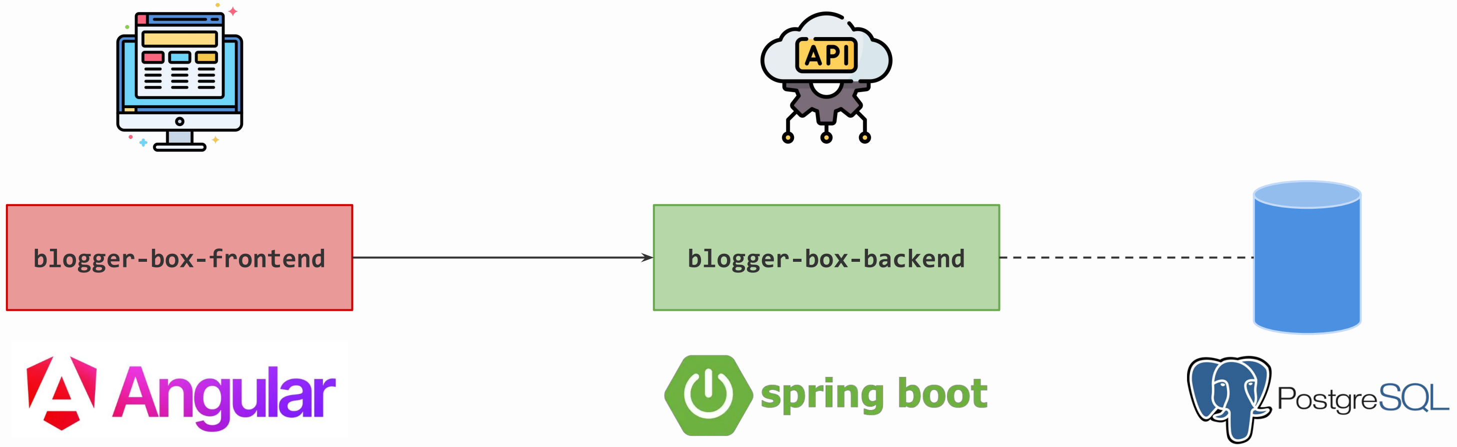
| post | | | | |
|---|---|---|---|---|
| id | title | content | created_date | category_id |
| 1 | ... | ... | 31/03/2024 | 1 |
| 2 | ... | ... | 30/03/2024 | 2 |
| 3 | ... | ... | 29/03/2024 | 1 |

# Flow frontend <-> backend



```
[
      {
            "id": "1",
            "title": "Blog title",
            "content": "Blog content",
            "createdDate": "2024-03-24"
      },
      …
]
```

## Blogger box architecture



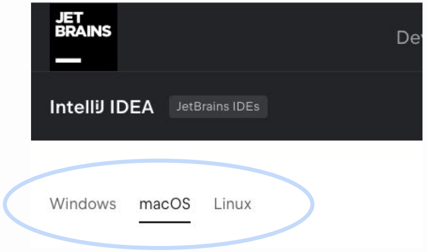
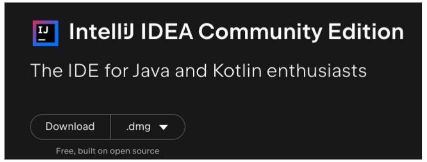
## Session 03

- 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**
- **Pagination** in an endpoint (performance)
- **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)  IntelliJ IDEA Community Edition The IDE for Java and Kotlin enthusiasts |

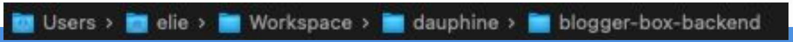
# Create spring boot app

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

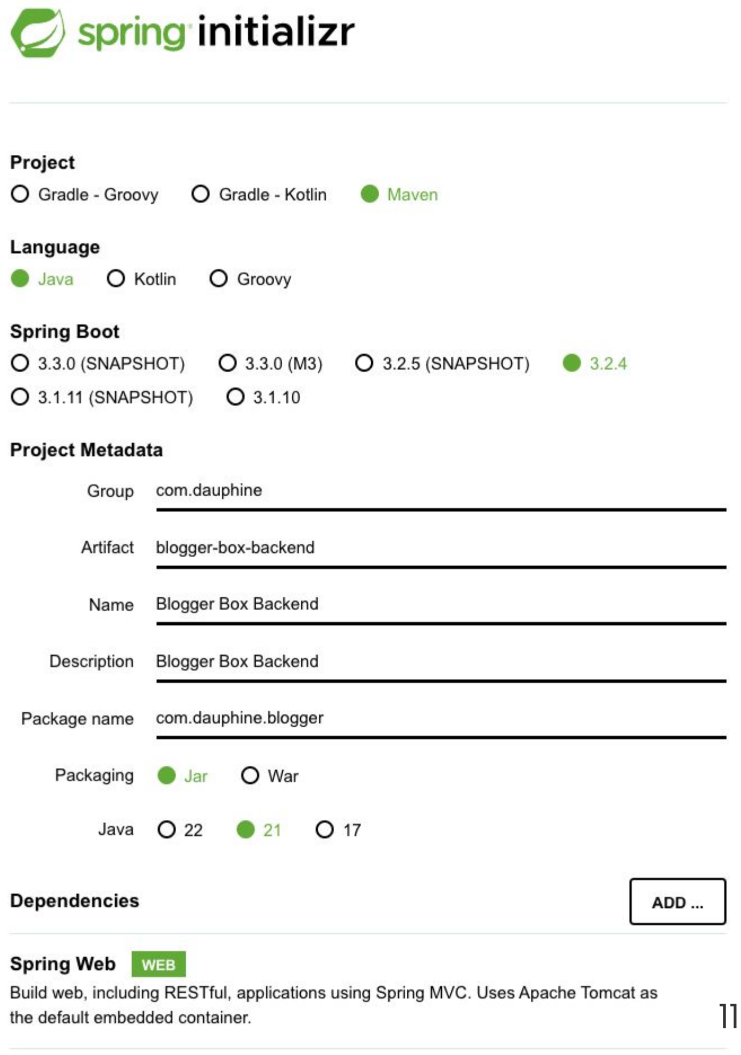
| | |
|---|---|
| **Project** | `Maven` |
| **Language** | `Java` |
| **Spring Boot** | `3.2.4` |
| **Project Metadata** | |
| **Group** | `com.dauphine` |
| **Artifact** | `blogger-box-backend` |
| **Name** | `Blogger Box Backend` |
| **Description** | `Blogger Box Backend` |
| **Package name** | `com.dauphine.blogger` |
| **Packaging** | `Jar` |
| **Java** | `21` |
| **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**

3.3.0 (SNAPSHOT)   3.3.0 (M3)   3.2.5 (SNAPSHOT)   [✓ 3.2.4]
3.1.11 (SNAPSHOT)   3.1.10

**Project Metadata**

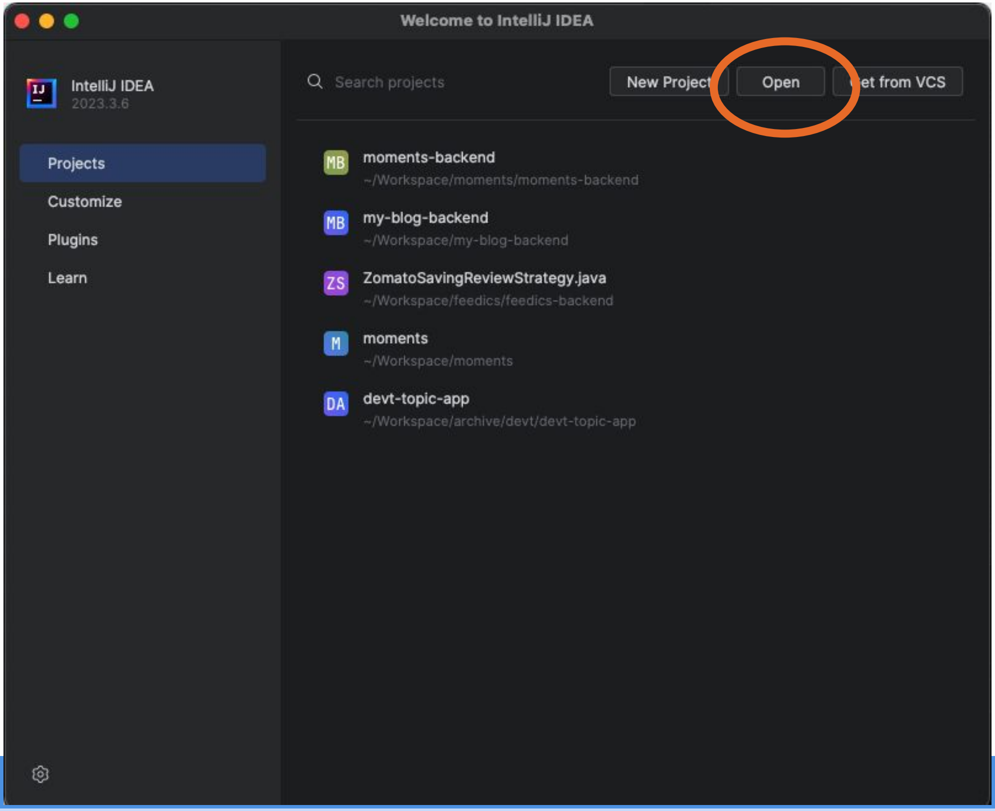
| | |
|---|---|
| Group | com.dauphine |
| Artifact | blogger-box-backend |
| Name | Blogger Box Backend |
| Description | Blogger Box Backend |
| Package name | com.dauphine.blogger |
| Packaging | [✓ Jar]   War |
| Java | 22   [✓ 21]   17 |

**Dependencies**

**Spring Web** WEB

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

## Open in IDE



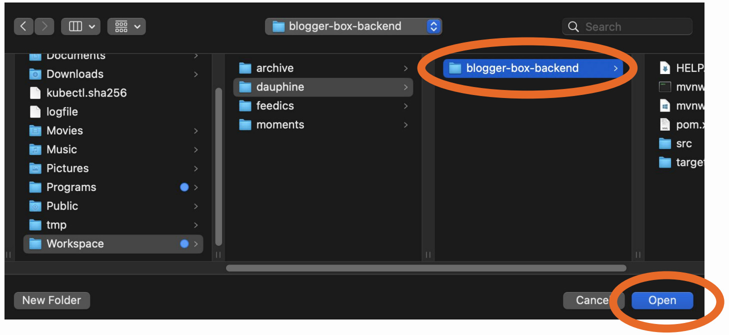
moments-backend
~/Workspace/moments/moments-backend

my-blog-backend
~/Workspace/my-blog-backend

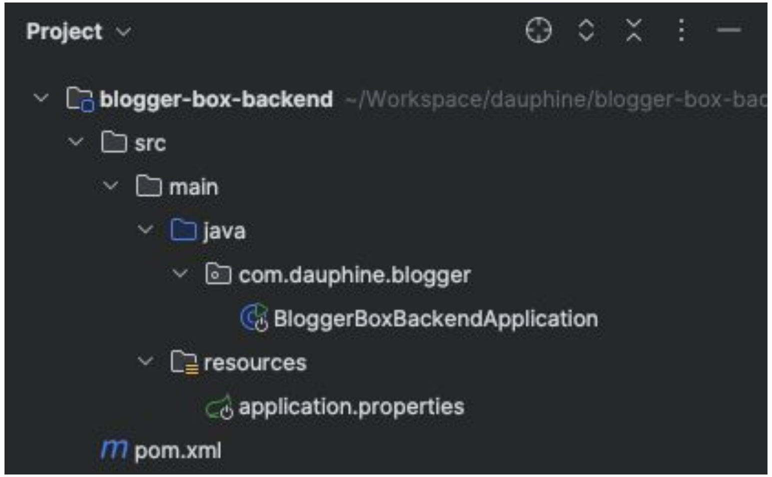
ZomatoSavingReviewStrategy.java
~/Workspace/feedics/feedics-backend

moments
~/Workspace/moments

devt-topic-app
~/Workspace/archive/devt/devt-topic-app

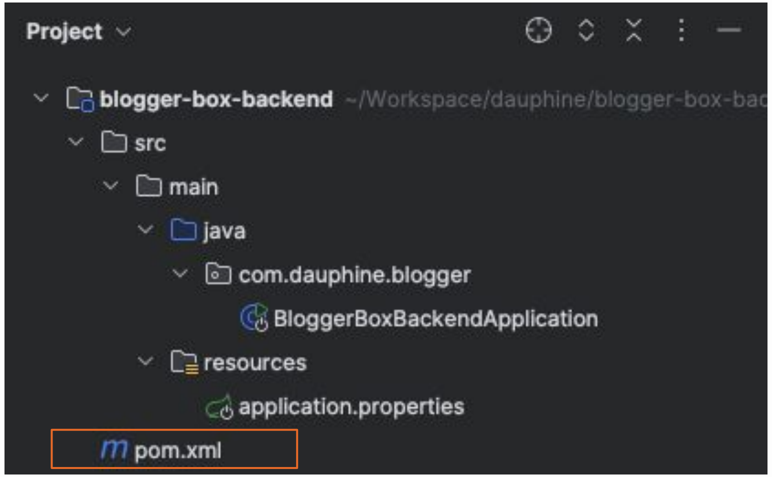


## Structure



```
Project
blogger-box-backend ~/Workspace/dauphine/blogger-box-bac
  src
    main
      java
        com.dauphine.blogger
          BloggerBoxBackendApplication
      resources
        application.properties
  pom.xml
```

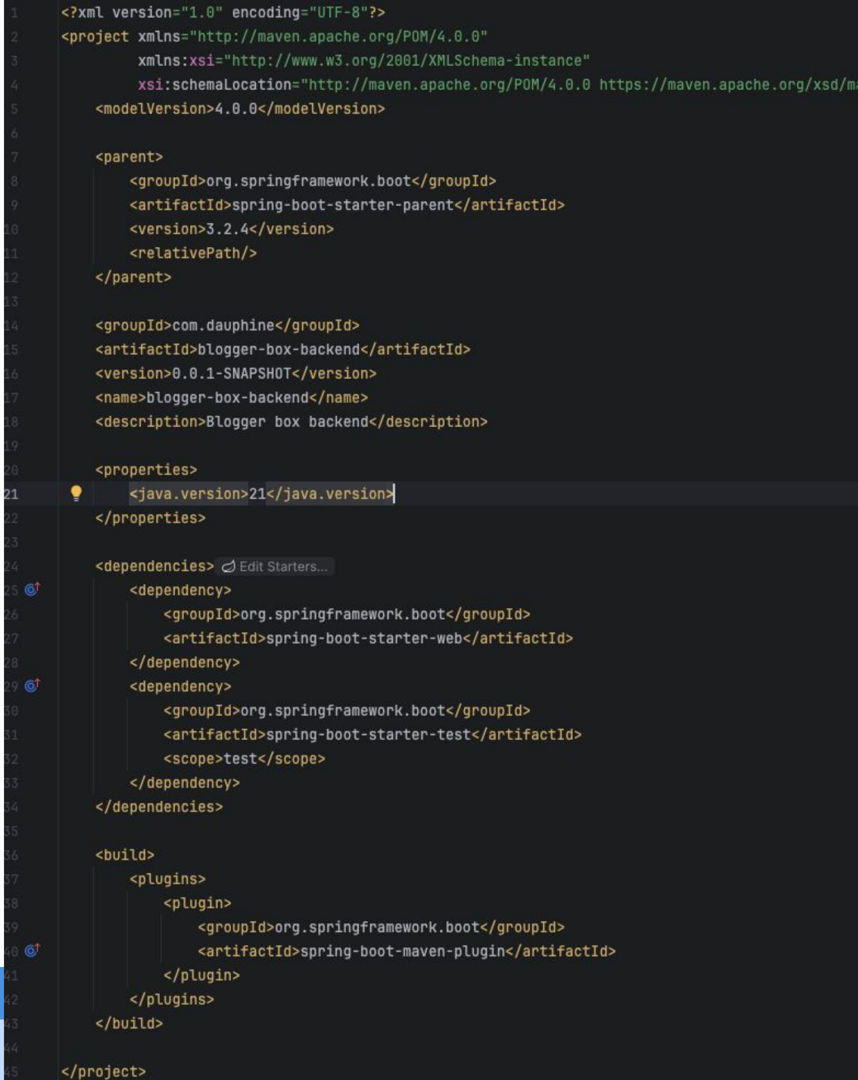
## Structure



```
blogger-box-backend ~/Workspace/dauphine/blogger-box-bac
  src
    main
      java
        com.dauphine.blogger
          BloggerBoxBackendApplication
      resources
        application.properties
  pom.xml
```

# pom.xml

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

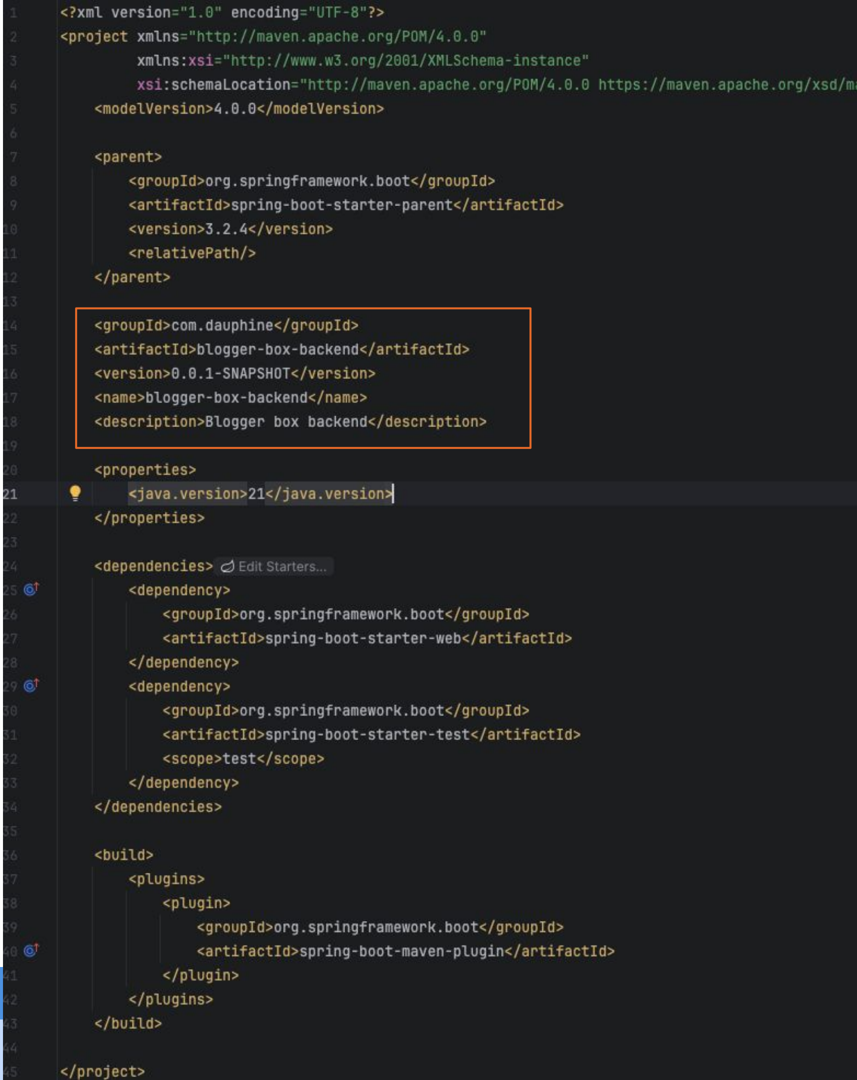


```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/ma
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.4</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>

    <properties>
        <java.version>21</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

# pom.xml

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

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



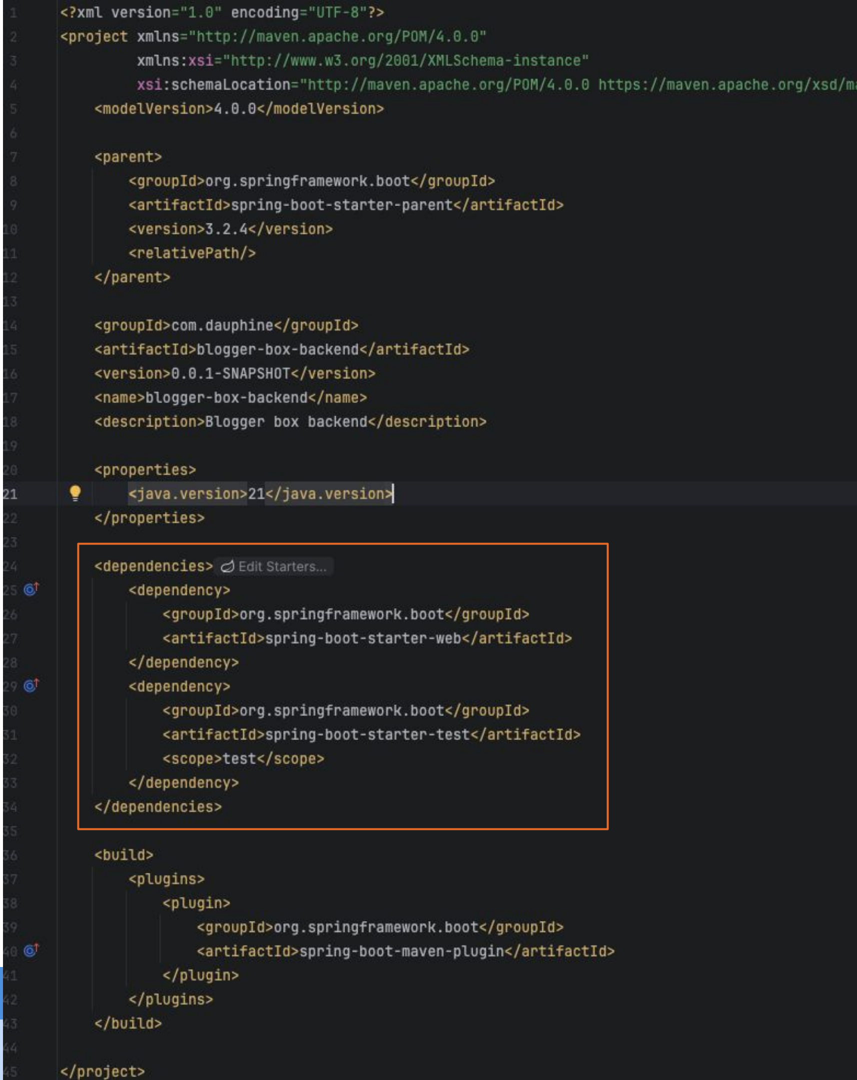
```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/ma
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.4</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>

    <properties>
        <java.version>21</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

## 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



```xml
<?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 https://maven.apache.org/xsd/m
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.4</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>

    <properties>
        <java.version>21</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

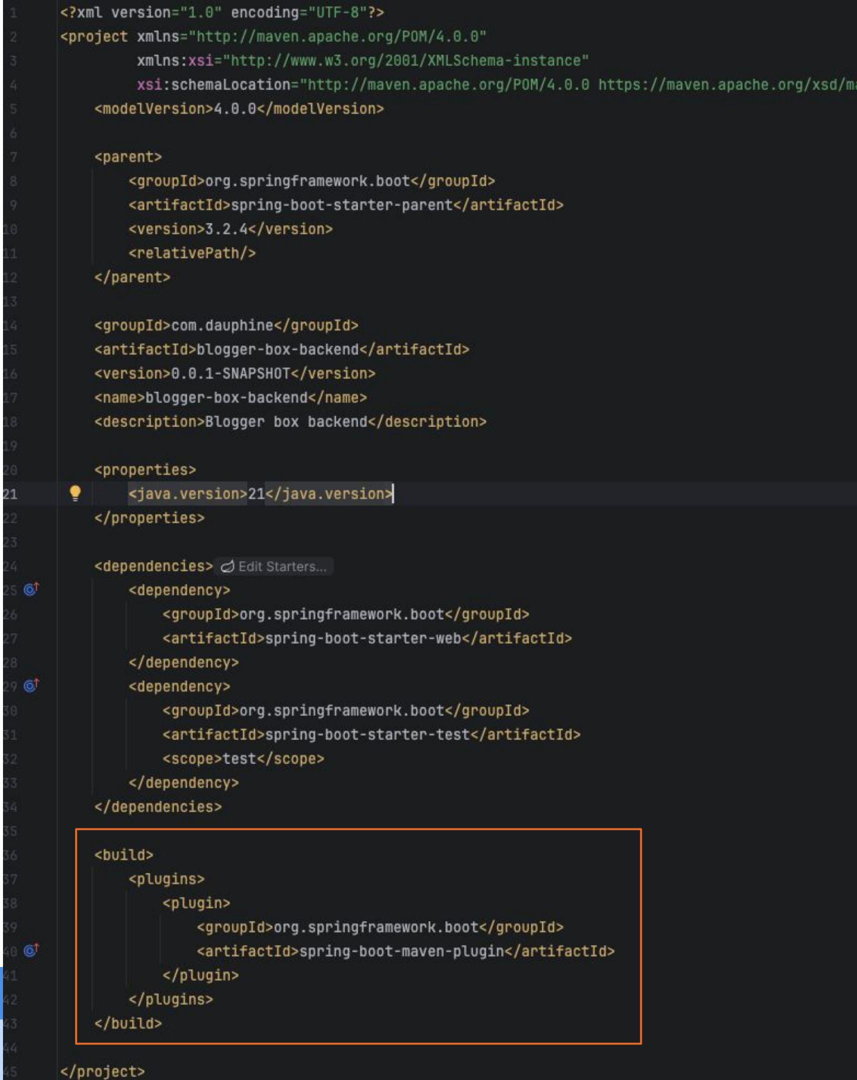
# pom.xml

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

**Project information**

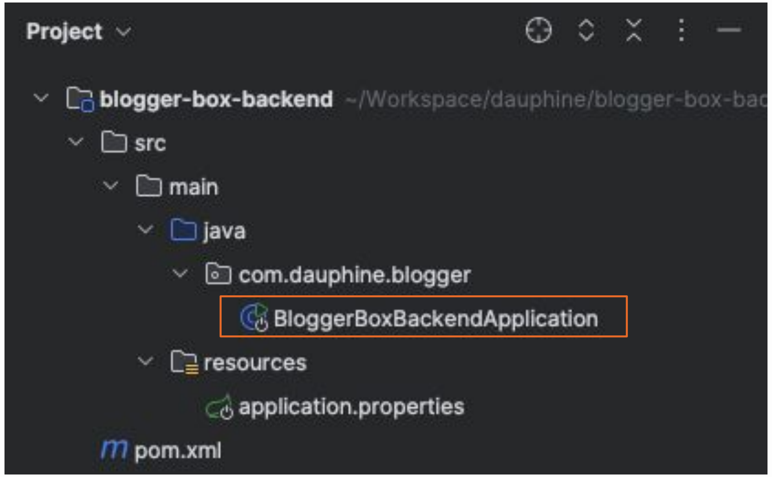
**Dependencies**

**Build configuration** : contains configuration settings related to the build process



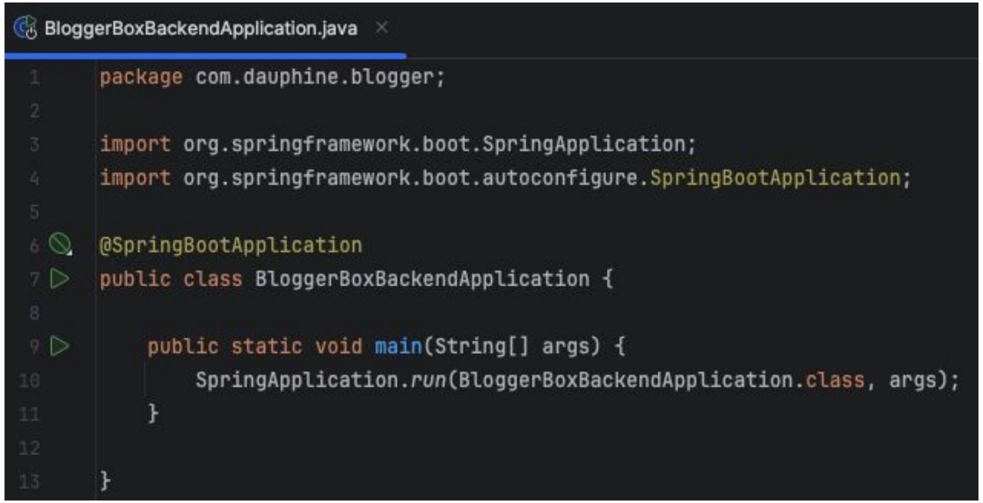
```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/m
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.4</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>

    <properties>
        <java.version>21</java.version>
    </properties>

    <dependencies> Edit Starters...
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

## Structure



```
Project
blogger-box-backend ~/Workspace/dauphine/blogger-box-bac
  src
    main
      java
        com.dauphine.blogger
          BloggerBoxBackendApplication
      resources
        application.properties
  pom.xml
```

# Spring boot application (main)



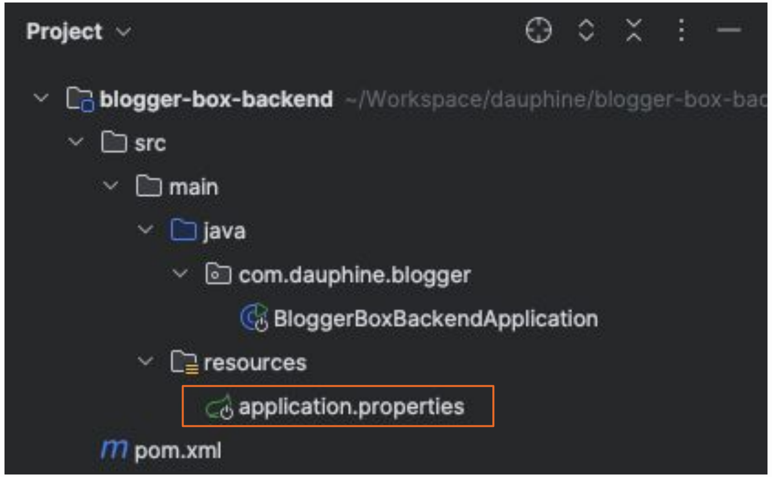
BloggerBoxBackendApplication.java

```java
package com.dauphine.blogger;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class BloggerBoxBackendApplication {

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

}
```

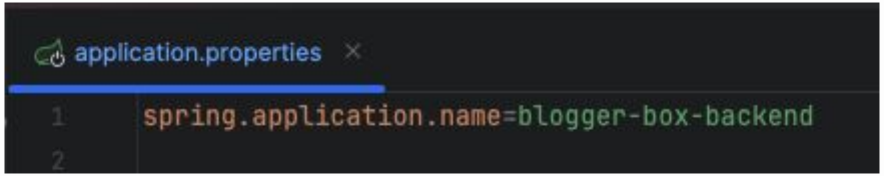
**`BloggerBoxBackendApplication`** contain the entry point of the application.

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

## Structure


```
Project
blogger-box-backend ~/Workspace/dauphine/blogger-box-bac
  src
    main
      java
        com.dauphine.blogger
          BloggerBoxBackendApplication
      resources
        application.properties
  pom.xml
```

## Application properties



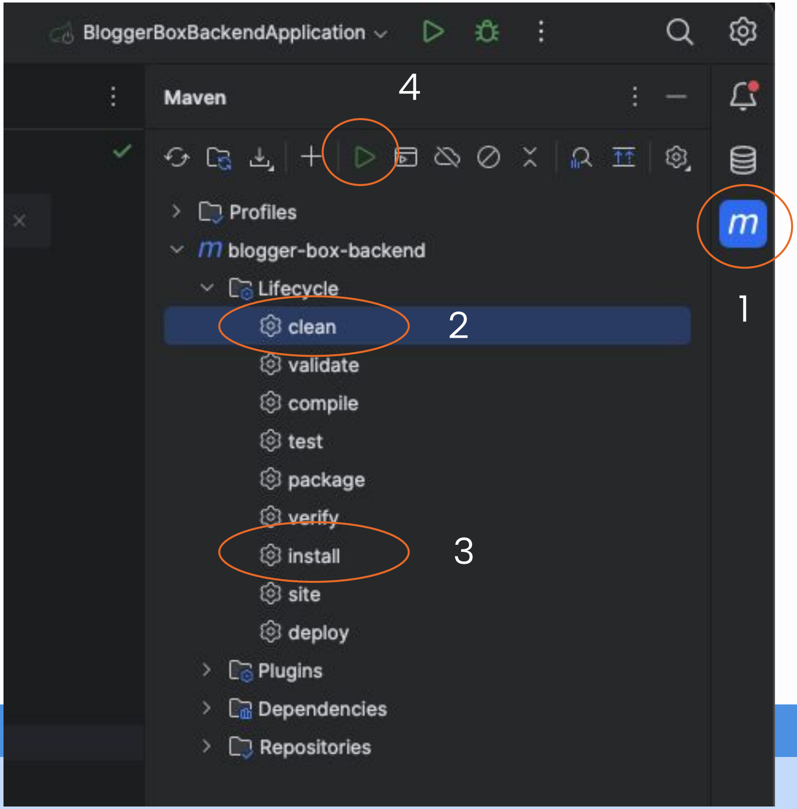
```
spring.application.name=blogger-box-backend
```

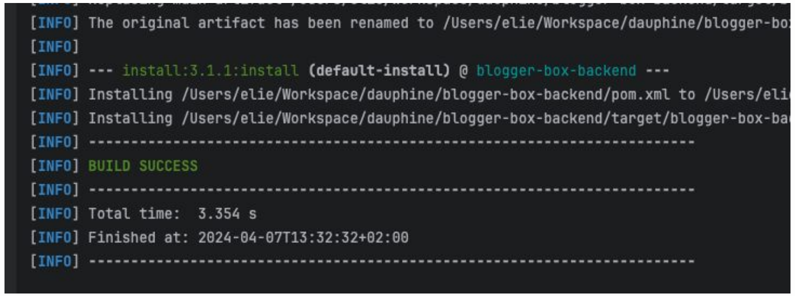
The **`application.properties`** file is a configuration file used to configure various aspect of the application.

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 environment.

# Compile

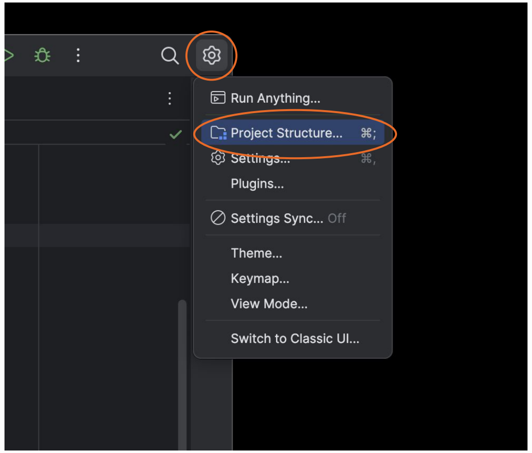


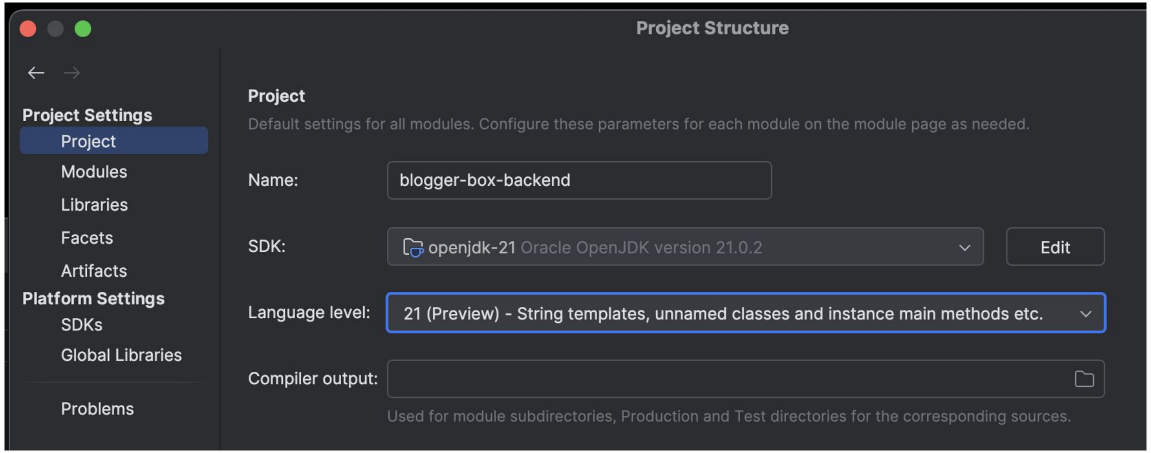


```
[INFO] The original artifact has been renamed to /Users/elie/Workspace/dauphine/blogger-bo
[INFO]
[INFO] --- install:3.1.1:install (default-install) @ blogger-box-backend ---
[INFO] Installing /Users/elie/Workspace/dauphine/blogger-box-backend/pom.xml to /Users/eli
[INFO] Installing /Users/elie/Workspace/dauphine/blogger-box-backend/target/blogger-box-ba
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.354 s
[INFO] Finished at: 2024-04-07T13:32:32+02:00
[INFO] ------------------------------------------------------------------------
```

## Compile

In case you are facing an issue with compiling you might want to check your sdk version (it should be version 21)





Project

Default settings for all modules. Configure these parameters for each module on the module page as needed.

Name: blogger-box-backend

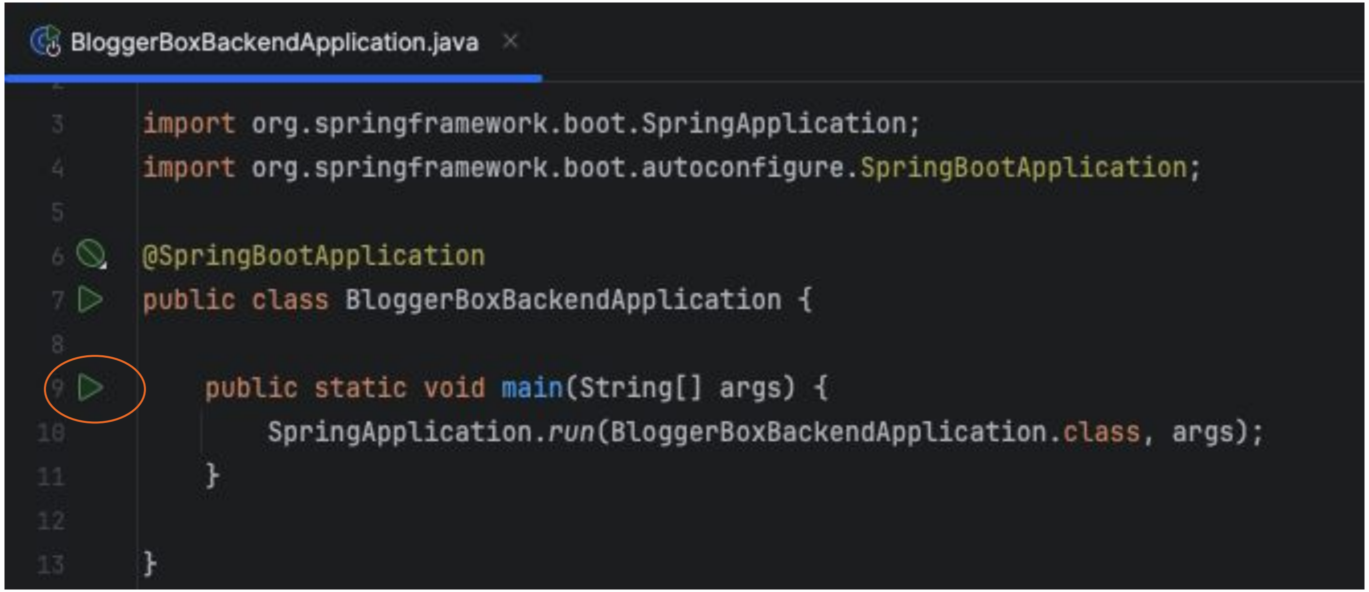
SDK: openjdk-21 Oracle OpenJDK version 21.0.2

Language level: 21 (Preview) - String templates, unnamed classes and instance main methods etc.

Compiler output:

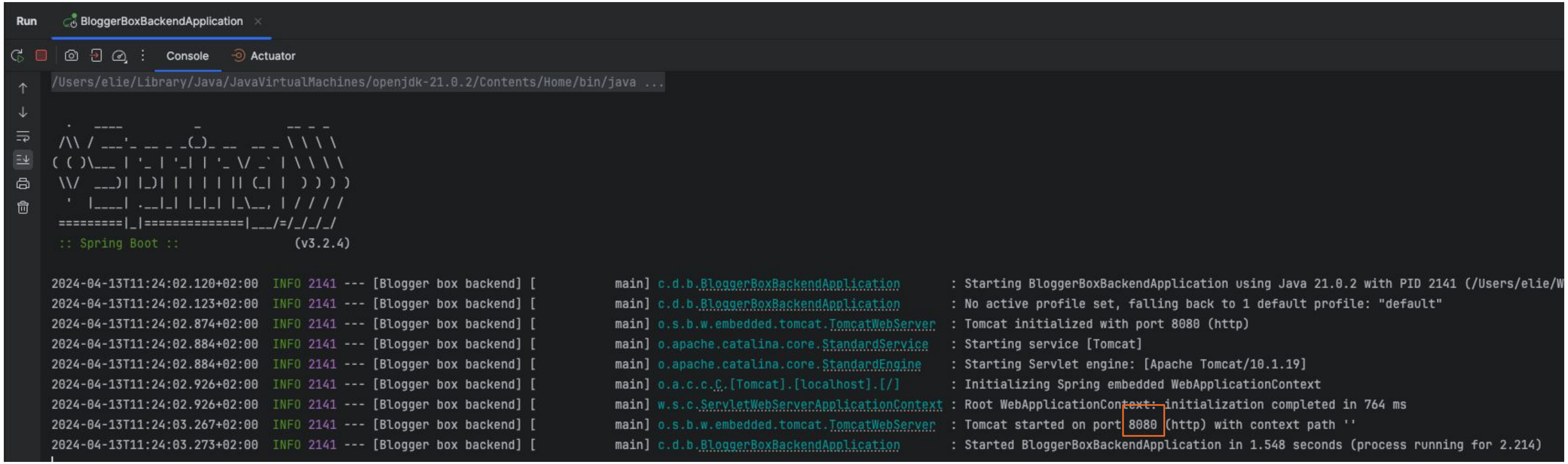
Used for module subdirectories, Production and Test directories for the corresponding sources.

## Start application



```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class BloggerBoxBackendApplication {

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

}
```

# Application started



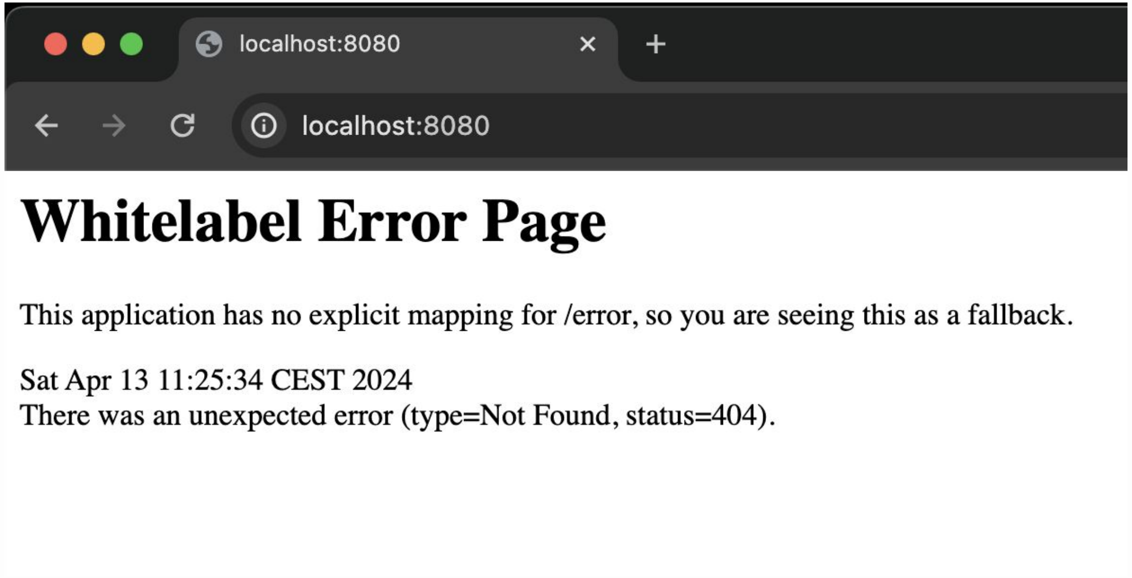
```
/Users/elie/Library/Java/JavaVirtualMachines/openjdk-21.0.2/Contents/Home/bin/java ...

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.4)

2024-04-13T11:24:02.120+02:00  INFO 2141 --- [Blogger box backend] [           main] c.d.b.BloggerBoxBackendApplication       : Starting BloggerBoxBackendApplication using Java 21.0.2 with PID 2141 (/Users/elie/W
2024-04-13T11:24:02.123+02:00  INFO 2141 --- [Blogger box backend] [           main] c.d.b.BloggerBoxBackendApplication       : No active profile set, falling back to 1 default profile: "default"
2024-04-13T11:24:02.874+02:00  INFO 2141 --- [Blogger box backend] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-04-13T11:24:02.884+02:00  INFO 2141 --- [Blogger box backend] [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-13T11:24:02.884+02:00  INFO 2141 --- [Blogger box backend] [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-13T11:24:02.926+02:00  INFO 2141 --- [Blogger box backend] [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-13T11:24:02.926+02:00  INFO 2141 --- [Blogger box backend] [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 764 ms
2024-04-13T11:24:03.267+02:00  INFO 2141 --- [Blogger box backend] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
2024-04-13T11:24:03.273+02:00  INFO 2141 --- [Blogger box backend] [           main] c.d.b.BloggerBoxBackendApplication       : Started BloggerBoxBackendApplication in 1.548 seconds (process running for 2.214)
```

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 🥳



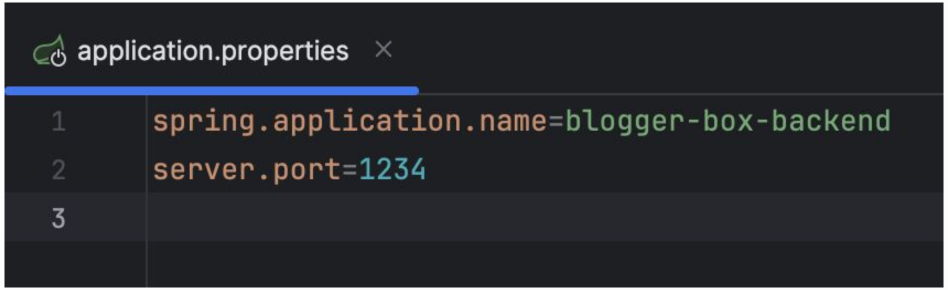
# Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sat Apr 13 11:25:34 CEST 2024
There was an unexpected error (type=Not Found, status=404).

## localhost:1234

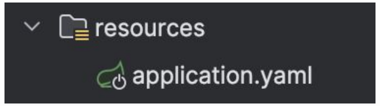
You can modify the port in the **`application.properties`** configuration file to whatever port that you want

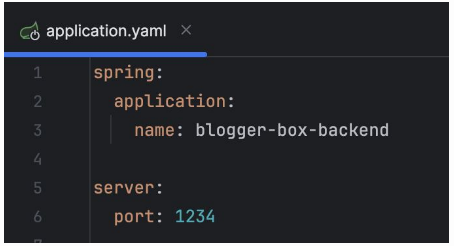


```
spring.application.name=blogger-box-backend
server.port=1234

```

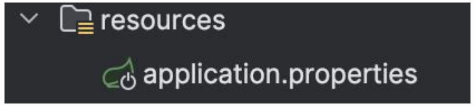
## .yaml vs .properties for properties

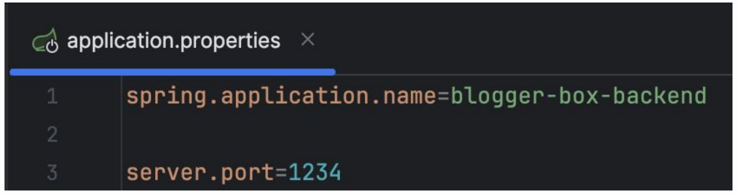
You can use `.yaml` file instead of `.properties`.
YAML is a convenient format for specifying hierarchical configuration data.





```yaml
spring:
  application:
    name: blogger-box-backend

server:
  port: 1234
```





```properties
spring.application.name=blogger-box-backend

server.port=1234
```

Using application.yml vs application.properties in Spring Boot

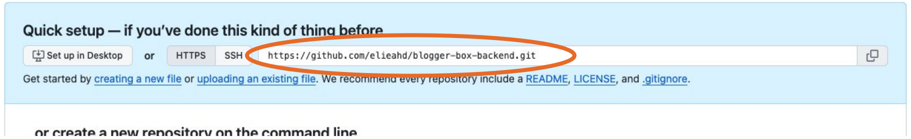
# Publish to github

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*

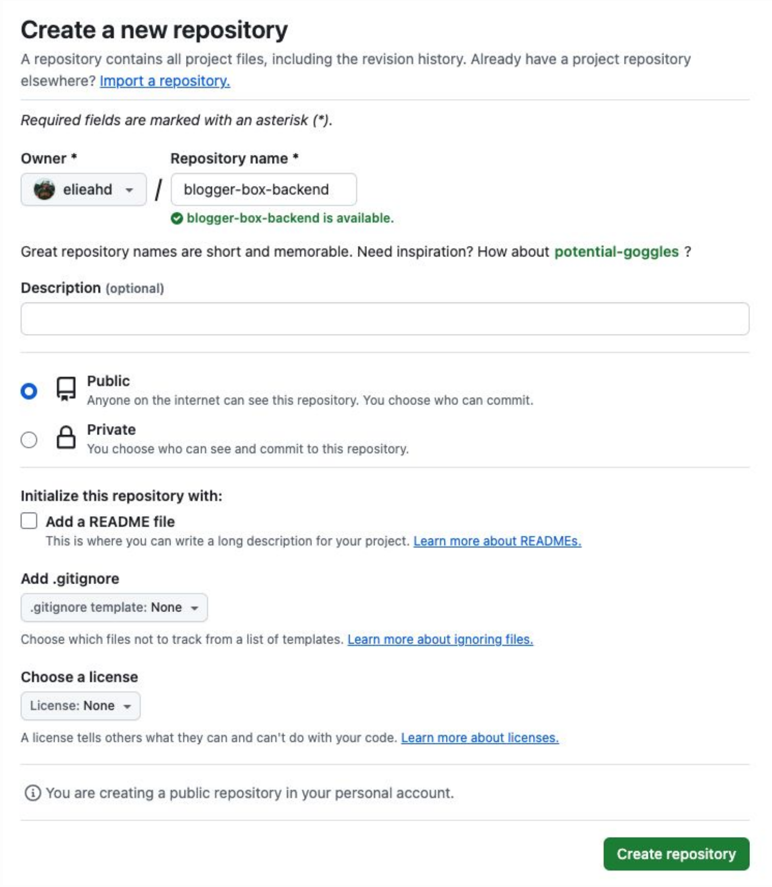


Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH `https://github.com/elieahd/blogger-box-backend.git`

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

or create a new repository on the command line



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 *: elieahd / Repository name *: blogger-box-backend

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: .gitignore template: None

Choose which files not to track from a list of templates. Learn more about ignoring files.

Choose a license: License: None

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

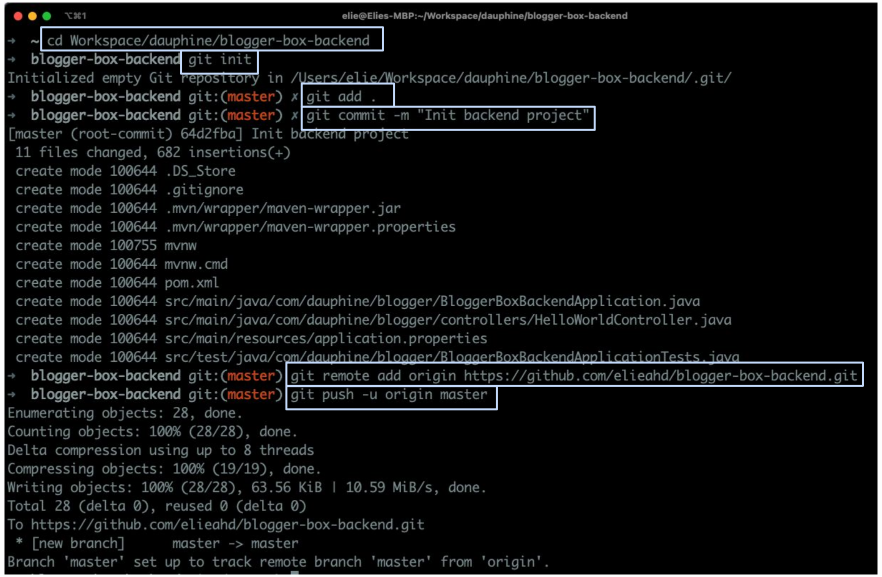
```
git commit -m “COMMIT_MESSAGE”
```

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

```
git remote add origin REMOTE_REPOSITORY_URL
```

Push the changes to Github

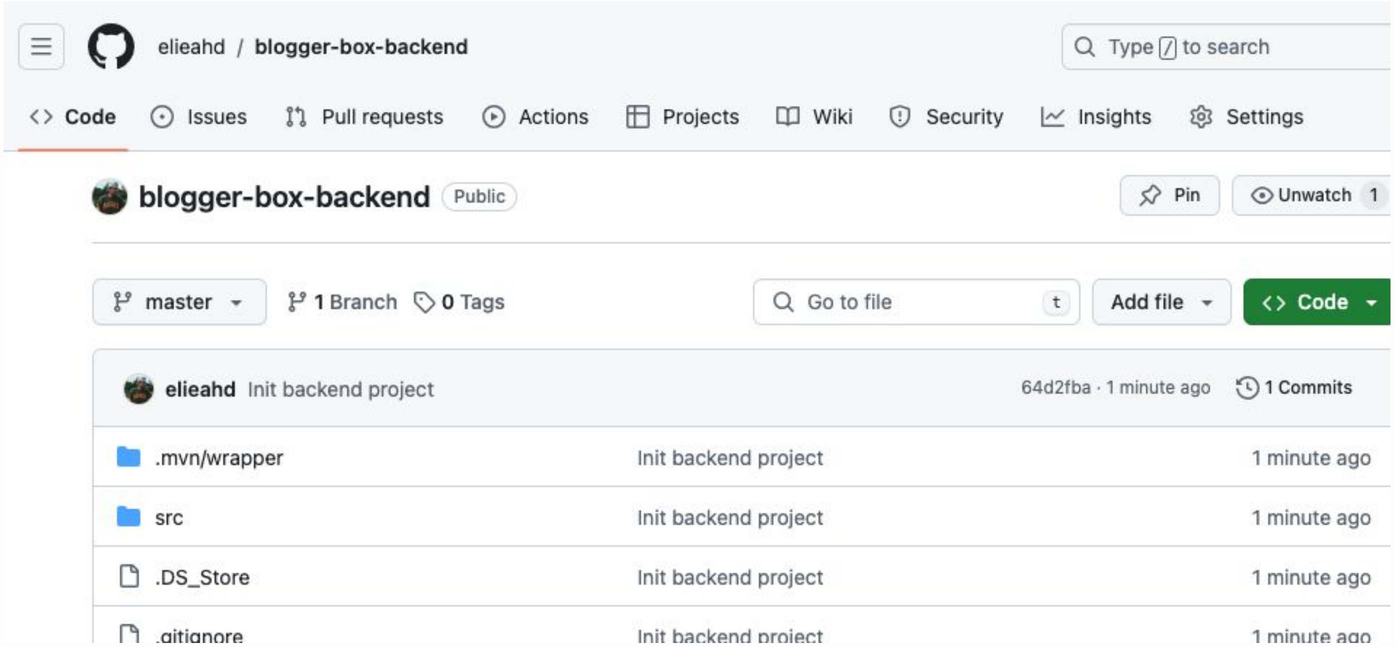
```
git push -u origin BRANCH_NAME
```



```
⌥⌘1                elie@Elies-MBP:~/Workspace/dauphine/blogger-box-backend
→  ~ cd Workspace/dauphine/blogger-box-backend
→  blogger-box-backend git init
Initialized empty Git repository in /Users/elie/Workspace/dauphine/blogger-box-backend/.git/
→  blogger-box-backend git:(master) ✗ git add .
→  blogger-box-backend git:(master) ✗ git commit -m "Init backend project"
[master (root-commit) 64d2fba] 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
→  blogger-box-backend git:(master) git remote add origin https://github.com/elieahd/blogger-box-backend.git
→  blogger-box-backend git:(master) 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

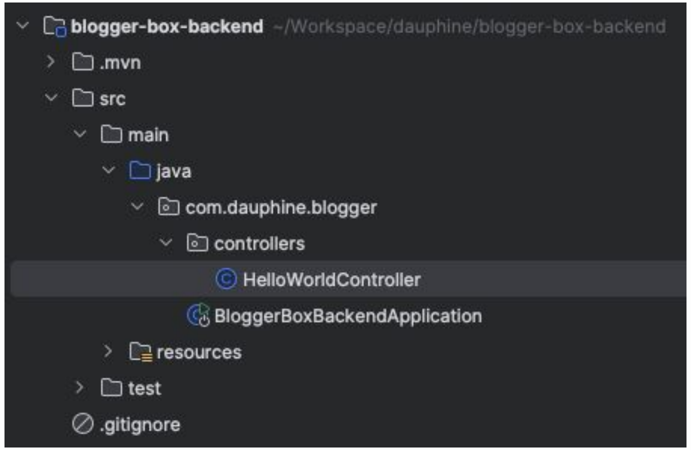


elieahd Init backend project 64d2fba · 1 minute ago 1 Commits

| | | |
|---|---|---|
| .mvn/wrapper | Init backend project | 1 minute ago |
| src | Init backend project | 1 minute ago |
| .DS_Store | Init backend project | 1 minute ago |
| .gitignore | Init backend project | 1 minute ago |

# HelloWorldController

Let’s create a new class **`HelloWorldController`** under controllers

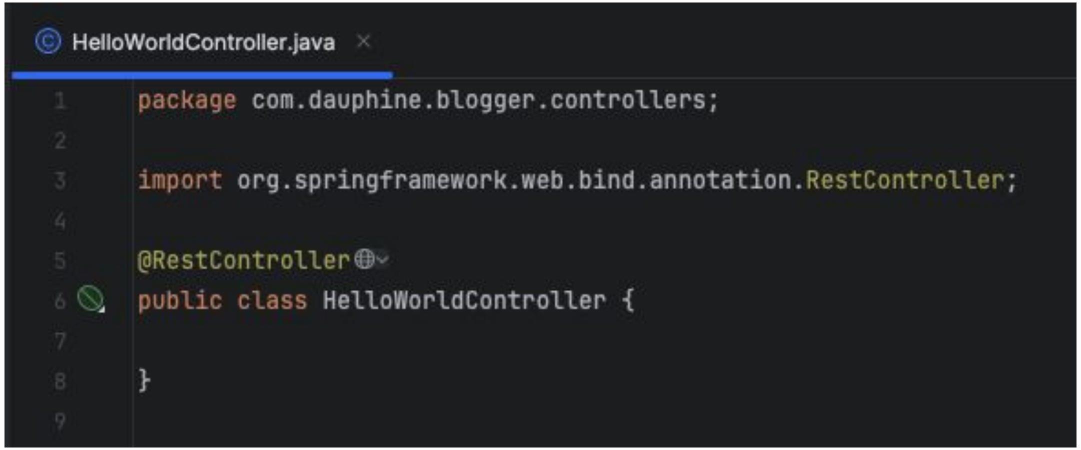


```
blogger-box-backend ~/Workspace/dauphine/blogger-box-backend
  .mvn
  src
    main
      java
        com.dauphine.blogger
          controllers
            HelloWorldController
          BloggerBoxBackendApplication
      resources
    test
  .gitignore
```

## 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

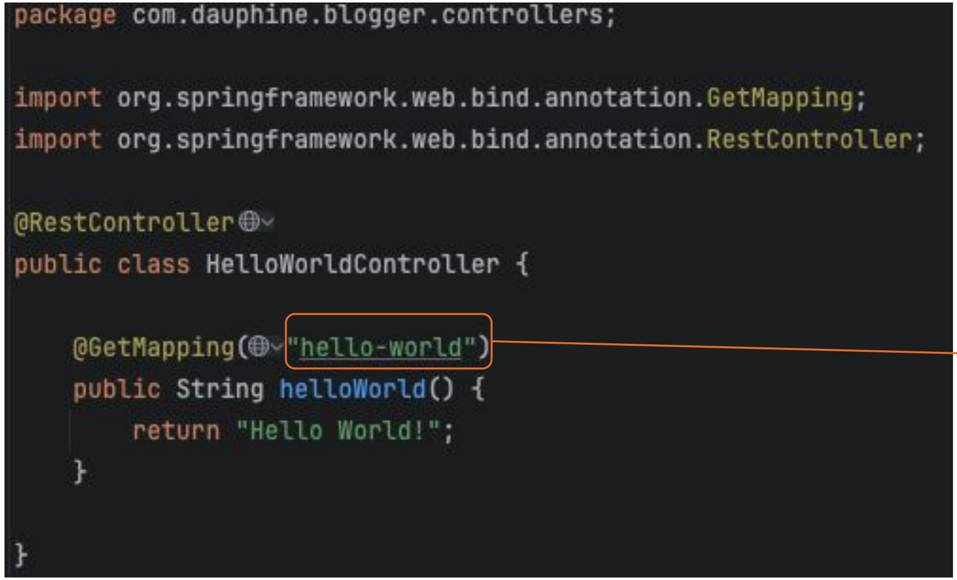


HelloWorldController.java

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

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

@RestController
public class HelloWorldController {

}
```

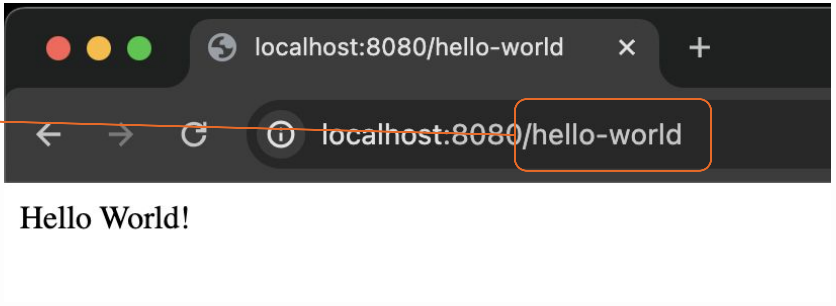
## Expose an endpoint

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



```java
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!";
    }

}
```

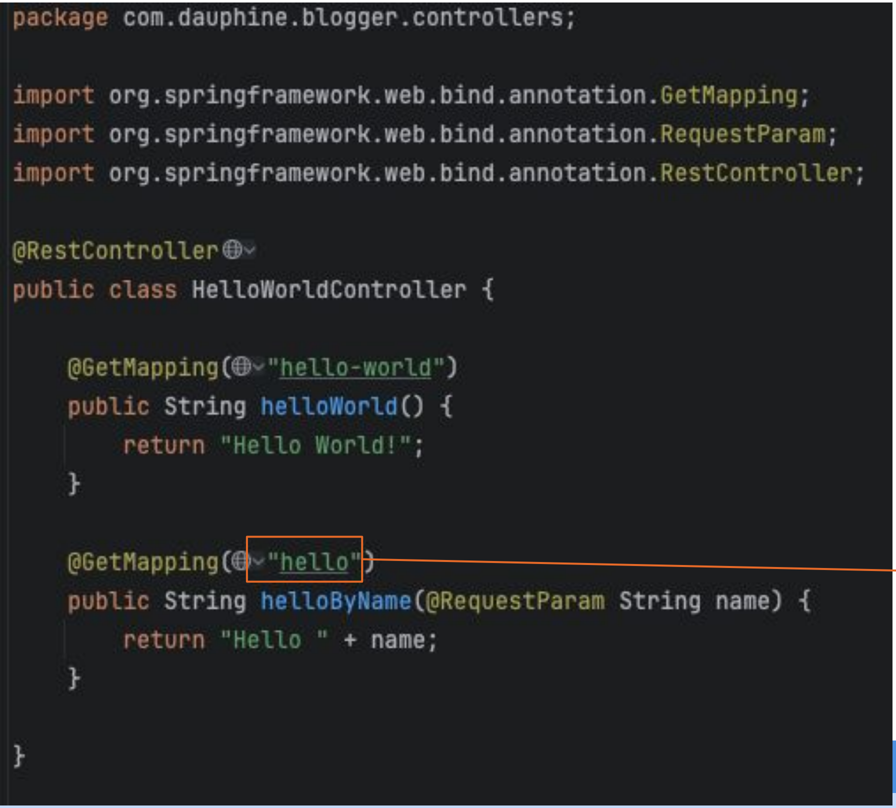
We can test GET Http request method in browser



```
localhost:8080/hello-world
```

Hello World!

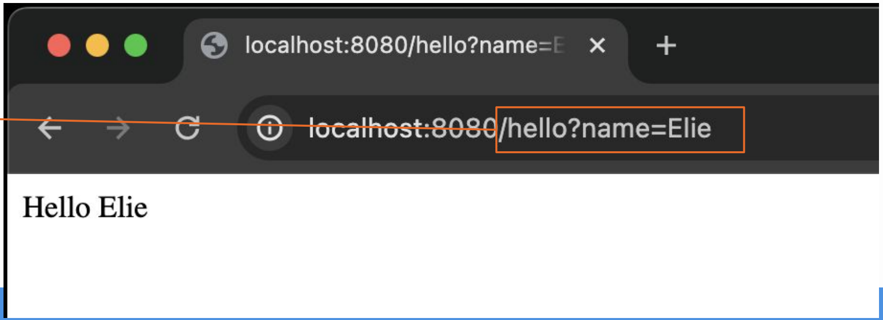
# Expose an endpoint with Request Param



```java
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 from of key-value pairs

We can test GET Http request method in browser

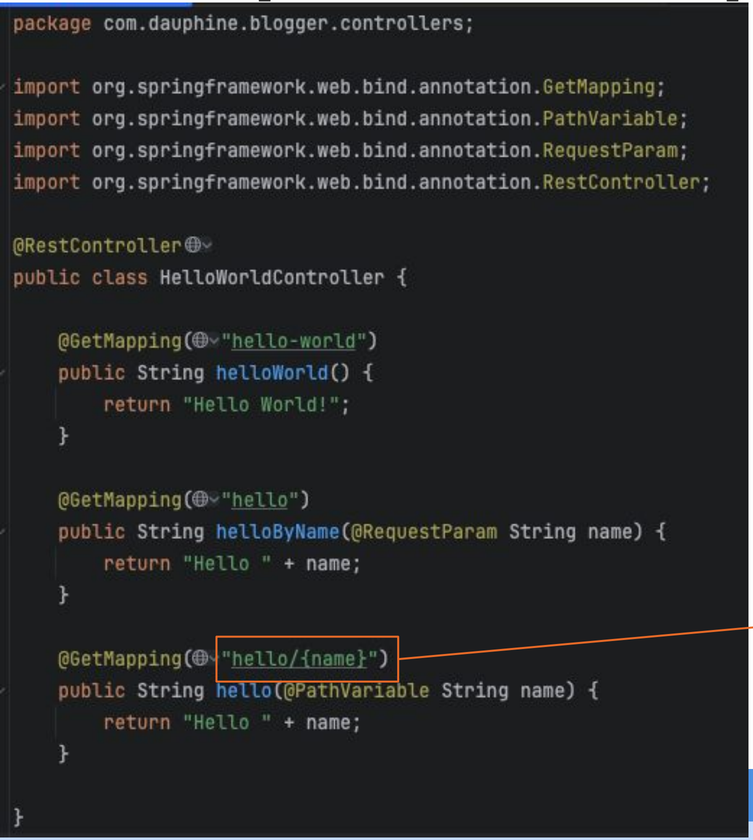


localhost:8080/hello?name=E

localhost:8080/hello?name=Elie

Hello Elie

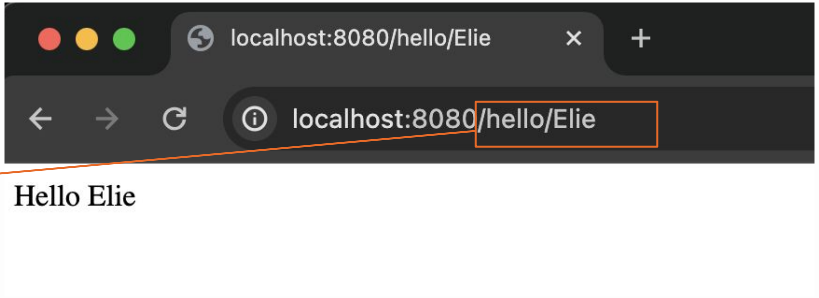
## Expose an endpoint with Path Variable



```java
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



Hello Elie

## Sync with Github

***expose my first endpoints***

## Endpoints

As of now we have exposed the following 3 endpoints

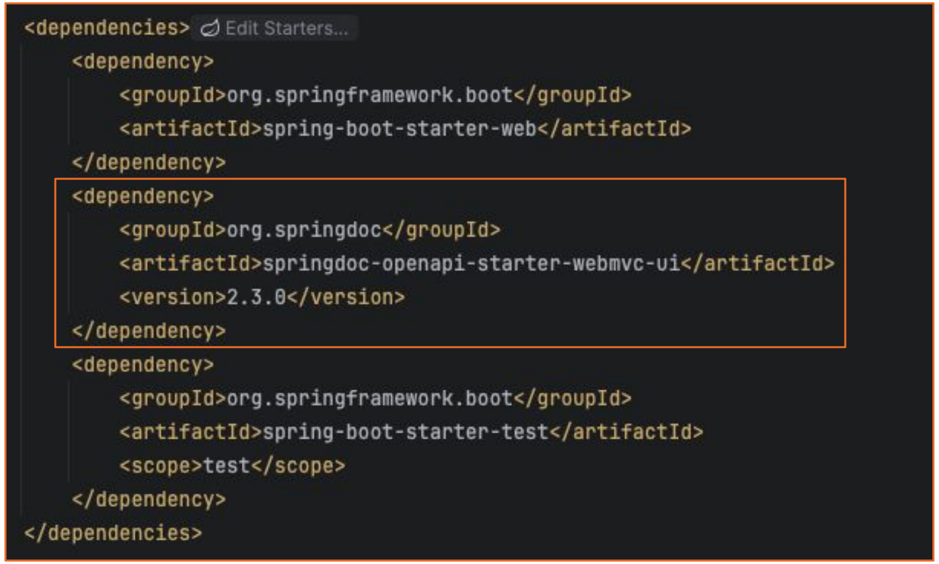
- `GET /hello-world`
- `GET /hello?name={...}`
- `GET /hello/{name}`

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 powerful tool that allow us to document and test our endpoints

Add following dependency in **pom.xml**



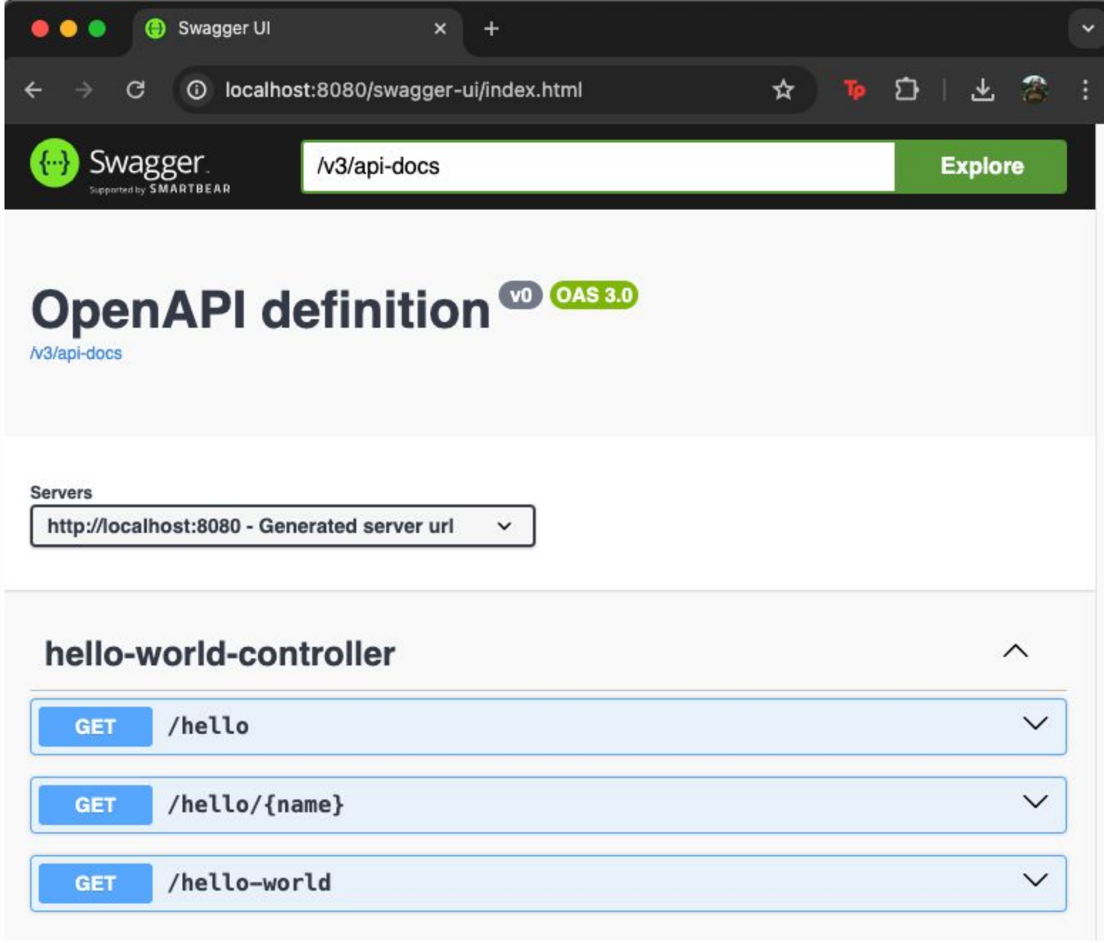
```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
        <version>2.3.0</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

```
<dependency>
   <groupId>org.springdoc</groupId>
   <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
   <version>2.3.0</version>
</dependency>
```

## Swagger

Head over to

```
http://localhost:8080/swagger-ui/index.html
```



OpenAPI definition v0 OAS 3.0

/v3/api-docs

Servers

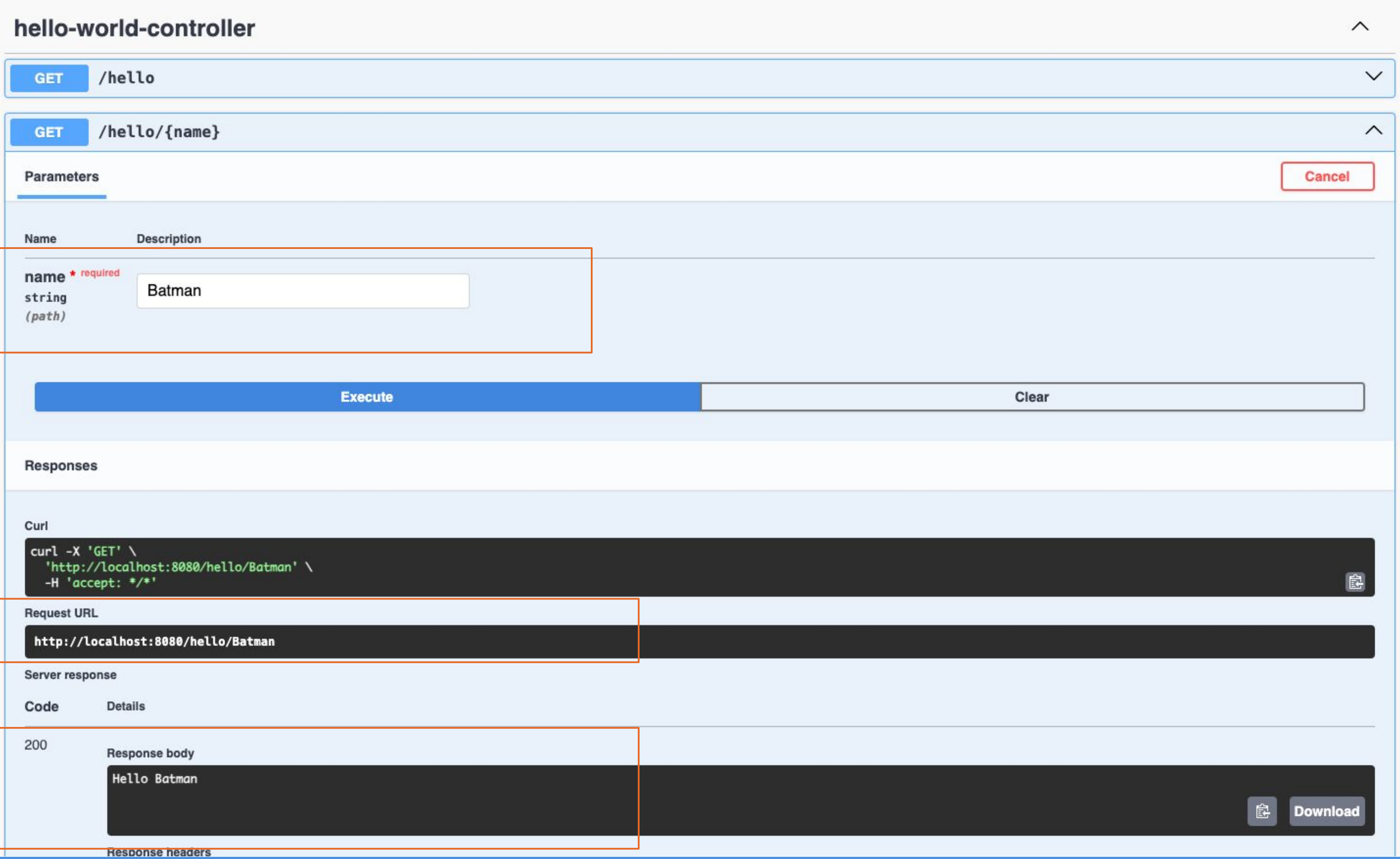
http://localhost:8080 - Generated server url

hello-world-controller

GET /hello

GET /hello/{name}

GET /hello-world



Batman

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

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

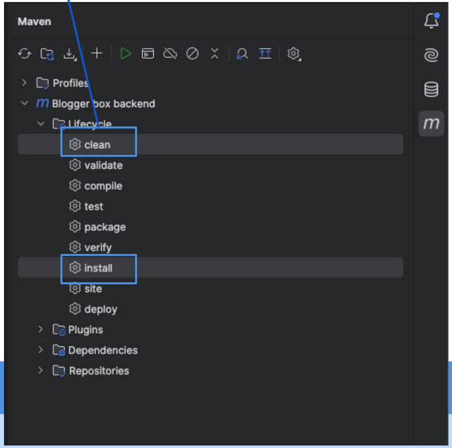
200

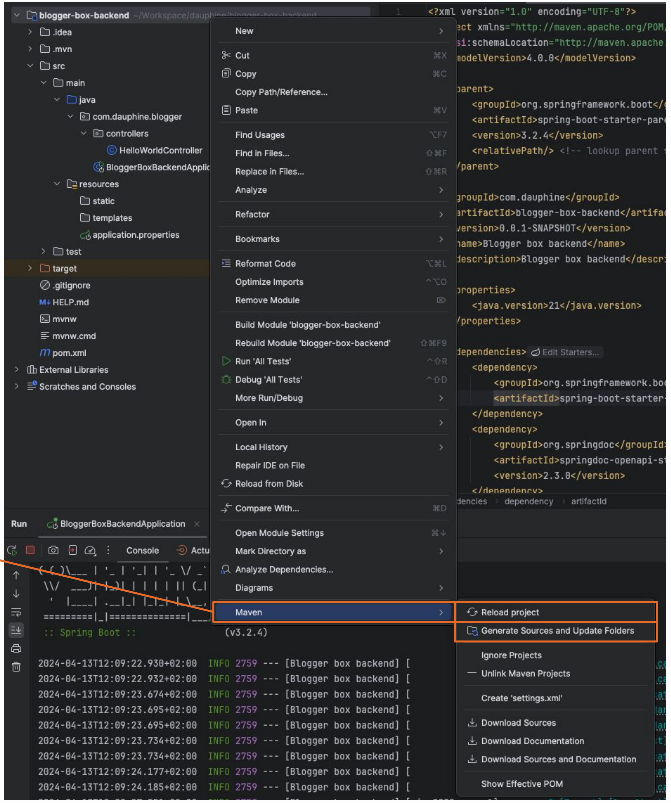
```
Hello Batman
```

# 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

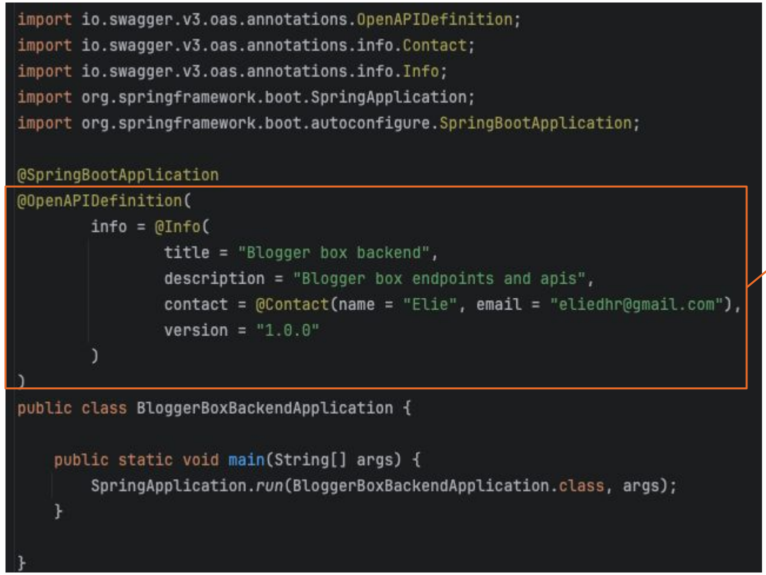




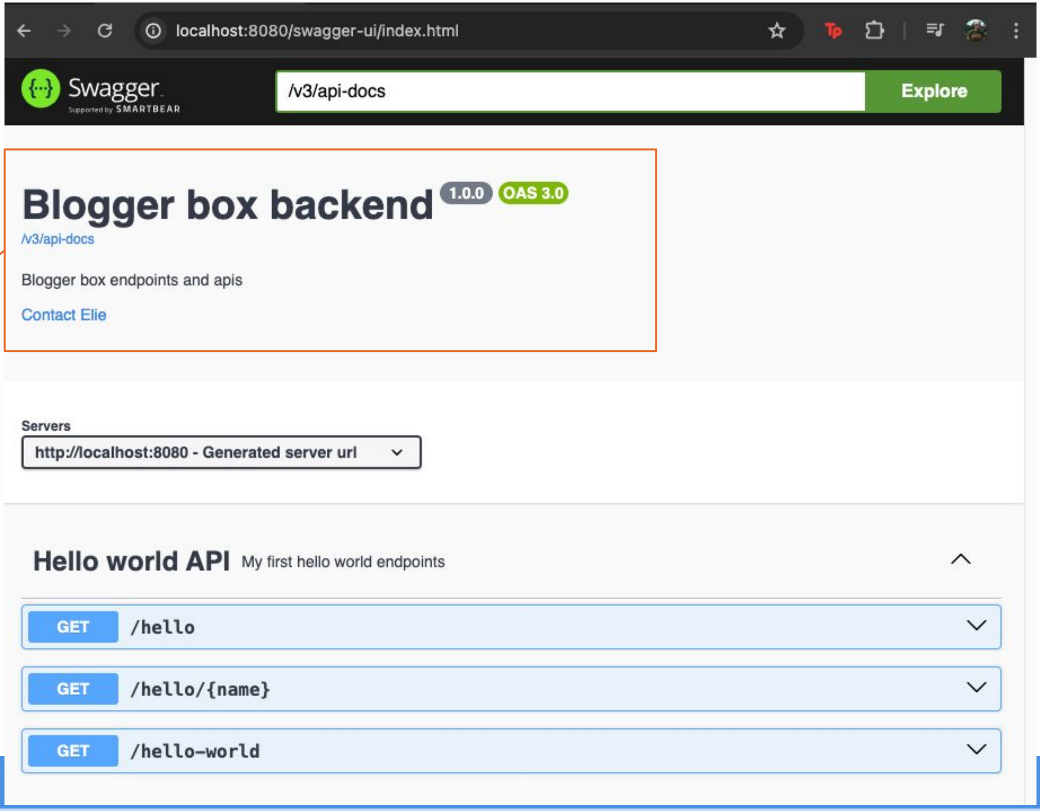
```
<?xml version="1.0" encoding="UTF-8"?>
ect xmlns="http://maven.apache.org/POM/
si:schemaLocation="http://maven.apache
modelVersion>4.0.0</modelVersion>

parent>
  <groupId>org.springframework.boot</
  <artifactId>spring-boot-starter-pare
  <version>3.2.4</version>
  <relativePath/> <!-- lookup parent
/parent>

groupId>com.dauphine</groupId>
artifactId>blogger-box-backend</artifac
version>0.0.1-SNAPSHOT</version>
name>Blogger box backend</name>
description>Blogger box backend</descr

properties>
  <java.version>21</java.version>
/properties>

dependencies>
  <dependency>
    <groupId>org.springframework.boo
    <artifactId>spring-boot-starter-
  </dependency>
  <dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-s
    <version>2.3.0</version>
  </dependency>
```

```
:: Spring Boot ::                (v3.2.4)

2024-04-13T12:09:22.930+02:00  INFO 2759 --- [Blogger box backend] [
2024-04-13T12:09:22.932+02:00  INFO 2759 --- [Blogger box backend] [
2024-04-13T12:09:23.674+02:00  INFO 2759 --- [Blogger box backend] [
2024-04-13T12:09:23.695+02:00  INFO 2759 --- [Blogger box backend] [
2024-04-13T12:09:23.695+02:00  INFO 2759 --- [Blogger box backend] [
2024-04-13T12:09:23.734+02:00  INFO 2759 --- [Blogger box backend] [
2024-04-13T12:09:23.734+02:00  INFO 2759 --- [Blogger box backend] [
2024-04-13T12:09:24.177+02:00  INFO 2759 --- [Blogger box backend] [
2024-04-13T12:09:24.185+02:00  INFO 2759 --- [Blogger box backend] [
```

## 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);
    }

}
```



localhost:8080/swagger-ui/index.html

Swagger Supported by SMARTBEAR

/v3/api-docs Explore

Blogger box backend 1.0.0 OAS 3.0

/v3/api-docs

Blogger box endpoints and apis

Contact Elie

Servers

http://localhost:8080 - Generated server url

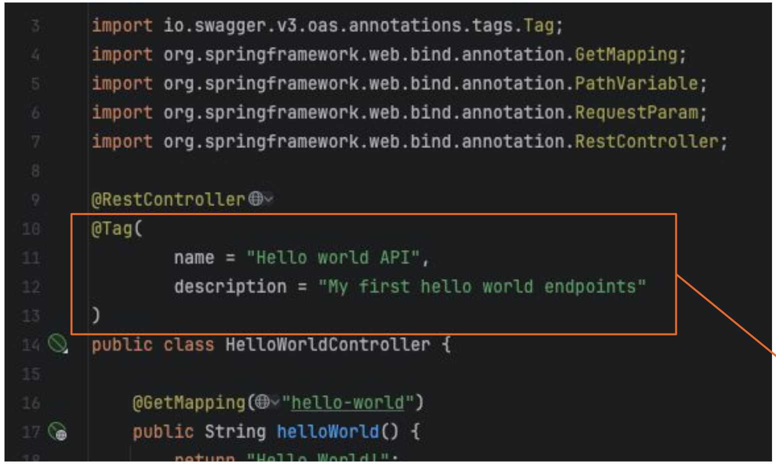
Hello world API My first hello world endpoints

GET /hello

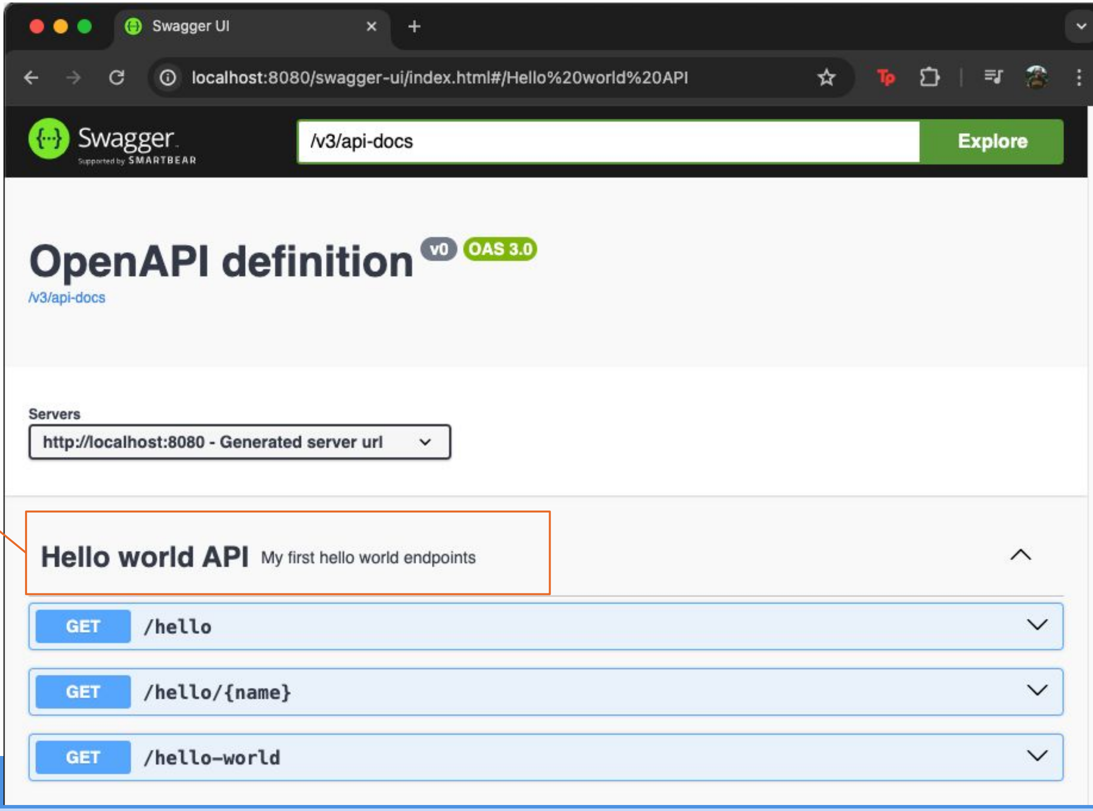
GET /hello/{name}

GET /hello-world

# Write Controller documentation



```
import io.swagger.v3.oas.annotations.tags.Tag;
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
@Tag(
        name = "Hello world API",
        description = "My first hello world endpoints"
)
public class HelloWorldController {

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



Swagger UI

localhost:8080/swagger-ui/index.html#/Hello%20world%20API

Swagger Supported by SMARTBEAR

/v3/api-docs Explore

OpenAPI definition v0 OAS 3.0

/v3/api-docs

Servers

http://localhost:8080 - Generated server url

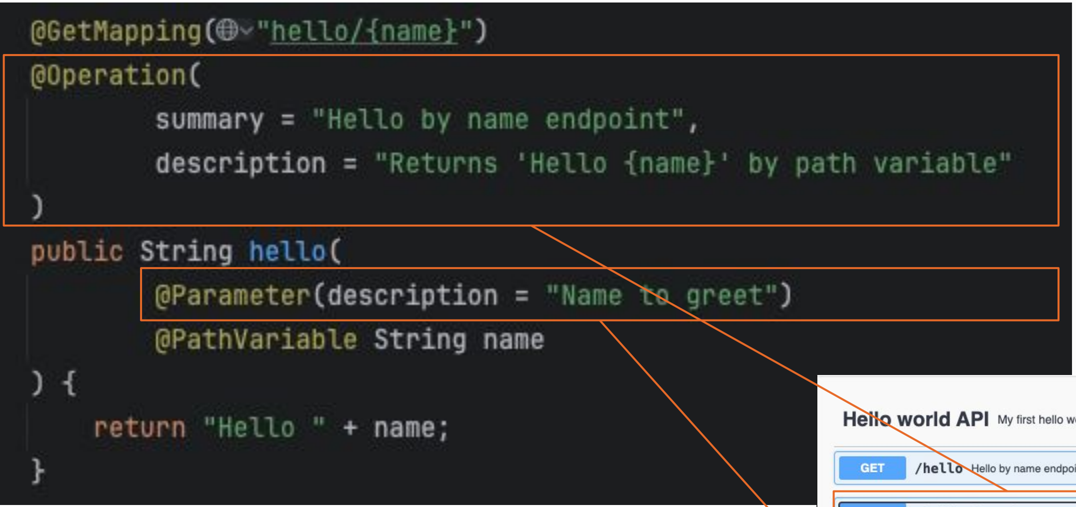
Hello world API My first hello world endpoints

GET /hello

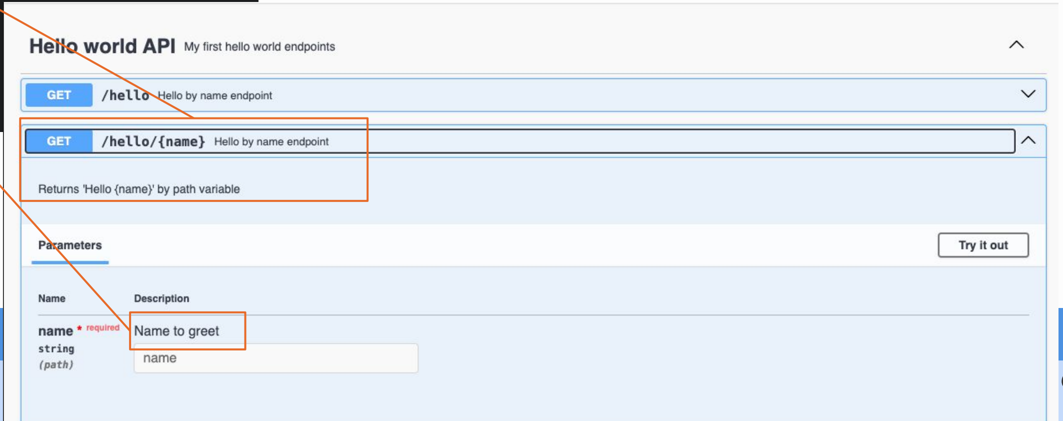
GET /hello/{name}

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 |

## Sync with Github

***add documentation via Swagger***

## HTTP request methods

| | |
|---|---|
| **GET** | Retrieve data *(should not modify data)* |
| **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 URIs, 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 /categories?name={name}` |
| `GET /posts/created-date/{created-date}` | `GET /posts?created-date={name}` |

## 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**

**Use lowercase letters**

**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*

✗
```
POST /v1/categories/create
```

✓
```
POST /v1/categories
```

# 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}
```

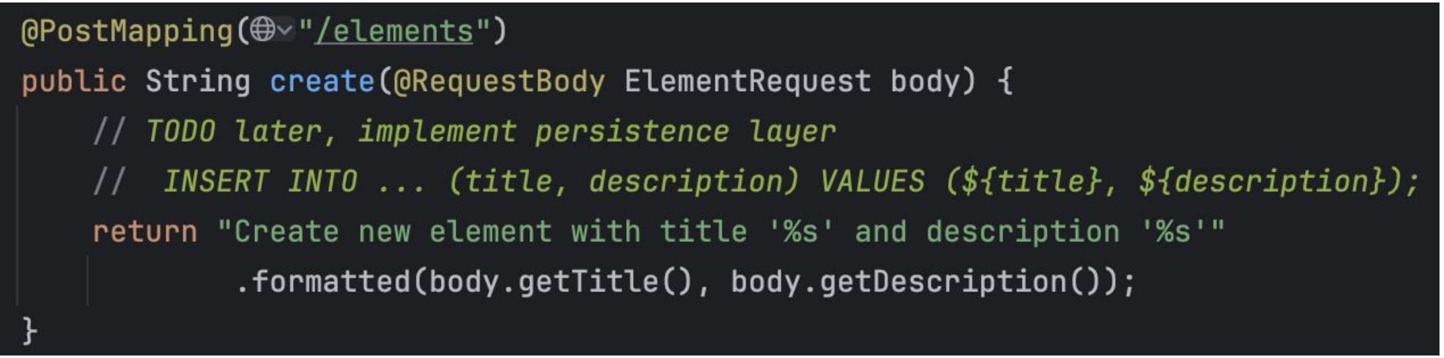
Delete a category

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

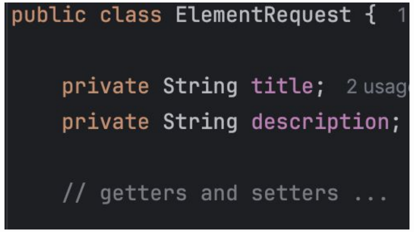
## Http method POST

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

**Example of creation of a new resource**



```java
@PostMapping("/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());
}
```

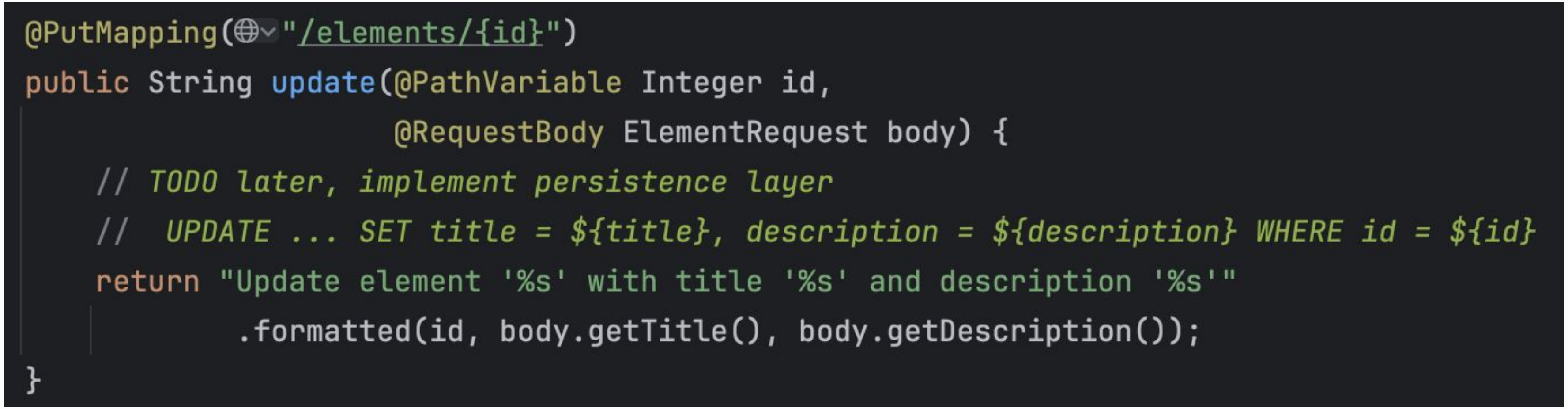


```java
public class ElementRequest {  1

    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**

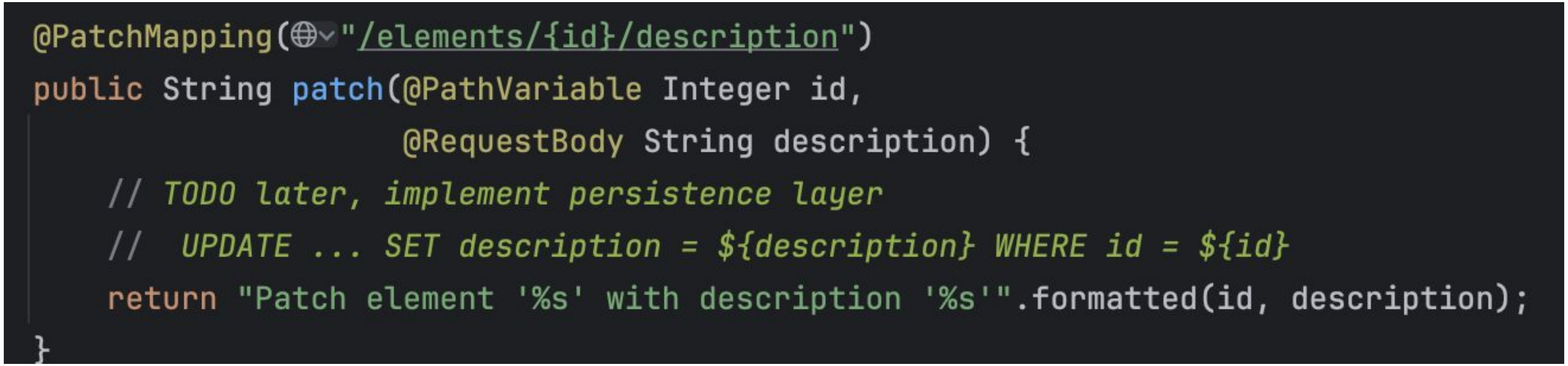


```java
@PutMapping("/elements/{id}")
public String update(@PathVariable Integer id,
                     @RequestBody ElementRequest body) {
    // TODO later, implement persistence layer
    //  UPDATE ... SET title = ${title}, description = ${description} WHERE id = ${id}
    return "Update element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

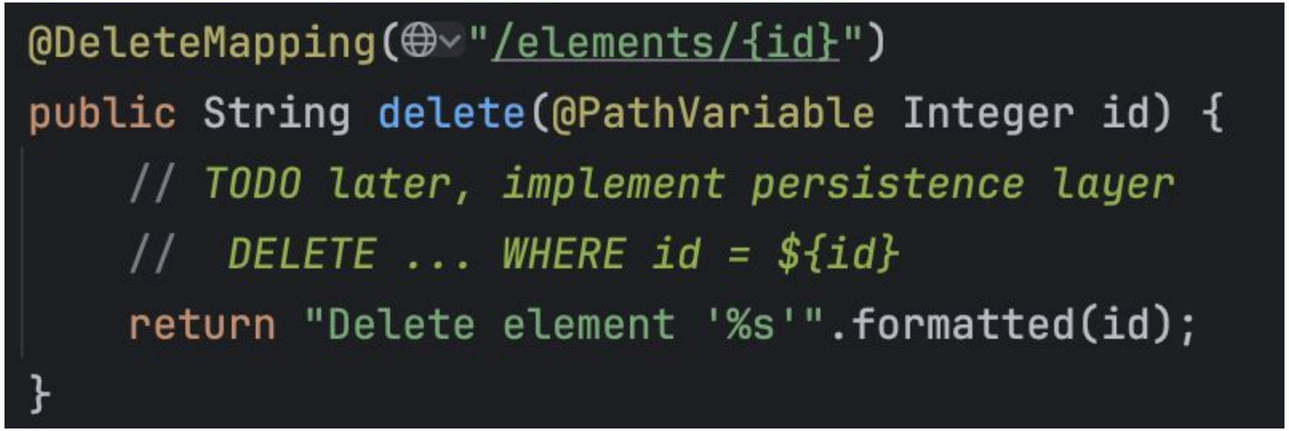


```
@PatchMapping("/elements/{id}/description")
public String patch(@PathVariable Integer id,
                    @RequestBody String description) {
    // TODO later, implement persistence layer
    //  UPDATE ... SET description = ${description} WHERE id = ${id}
    return "Patch element '%s' with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

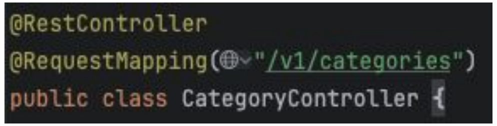
*check slide 30 in Session 01, to get the attributes of a post and a category*

## Blogger box use cases

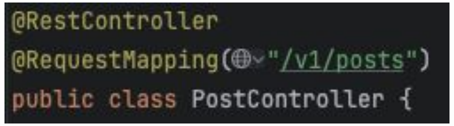
*Tip #1 split those endpoints into 2 controllers :* **`CategoryController, PostController`**

*Tip #2 add versioning to endpoint (*`v1` 😉*)*

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



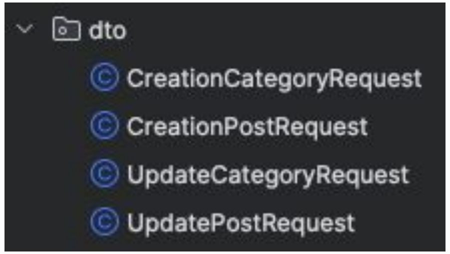
```
@RestController
@RequestMapping("/v1/categories")
public class CategoryController {
```



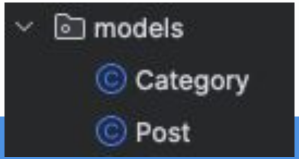
```
@RestController
@RequestMapping("/v1/posts")
public class PostController {
```

*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*



```
dto
    CreationCategoryRequest
    CreationPostRequest
    UpdateCategoryRequest
    UpdatePostRequest
```

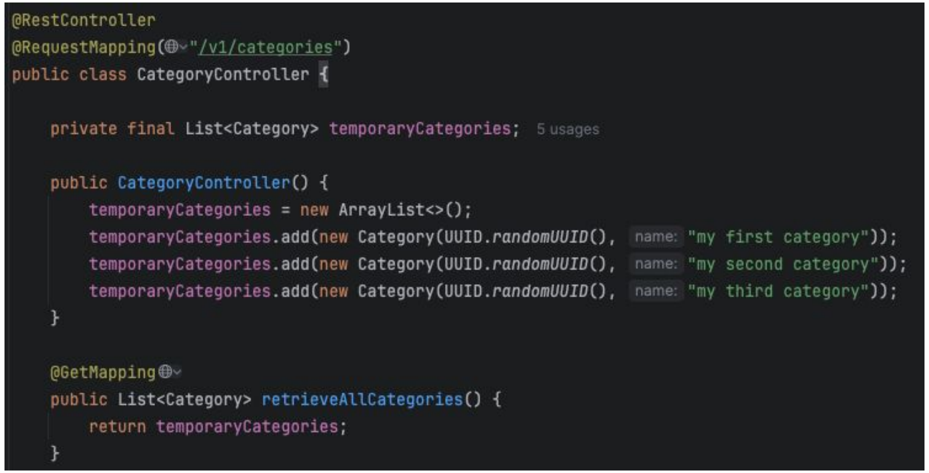


```
models
    Category
    Post
```

# Blogger box use cases

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

*Example :*



```java
@RestController
@RequestMapping("/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
    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***