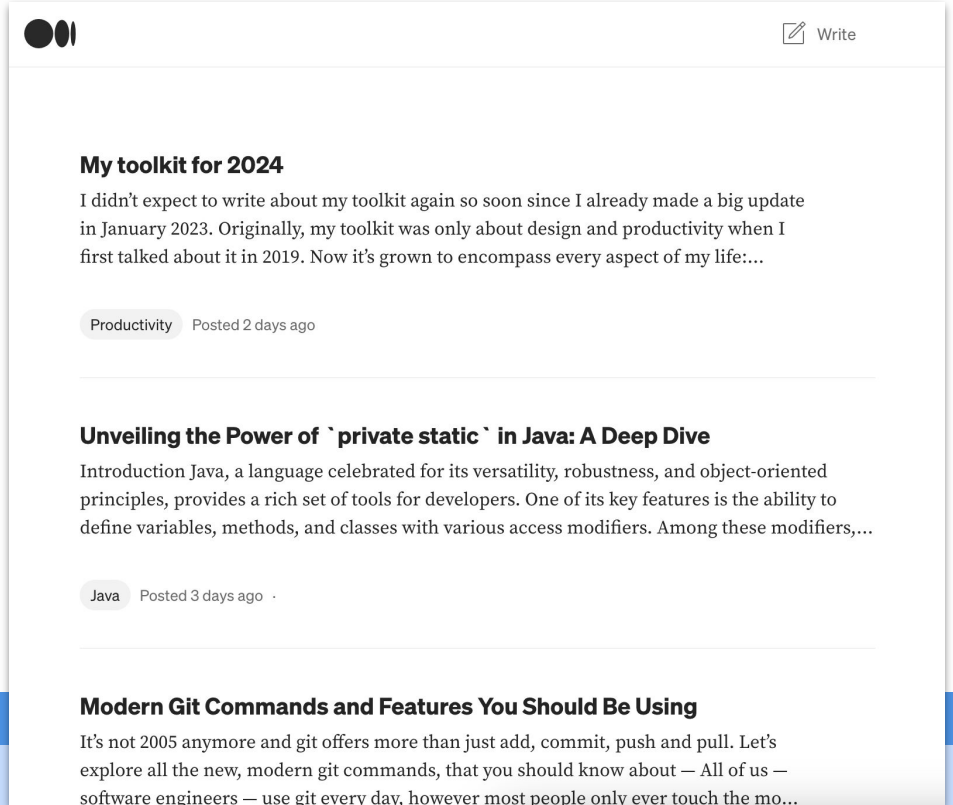
Session 03

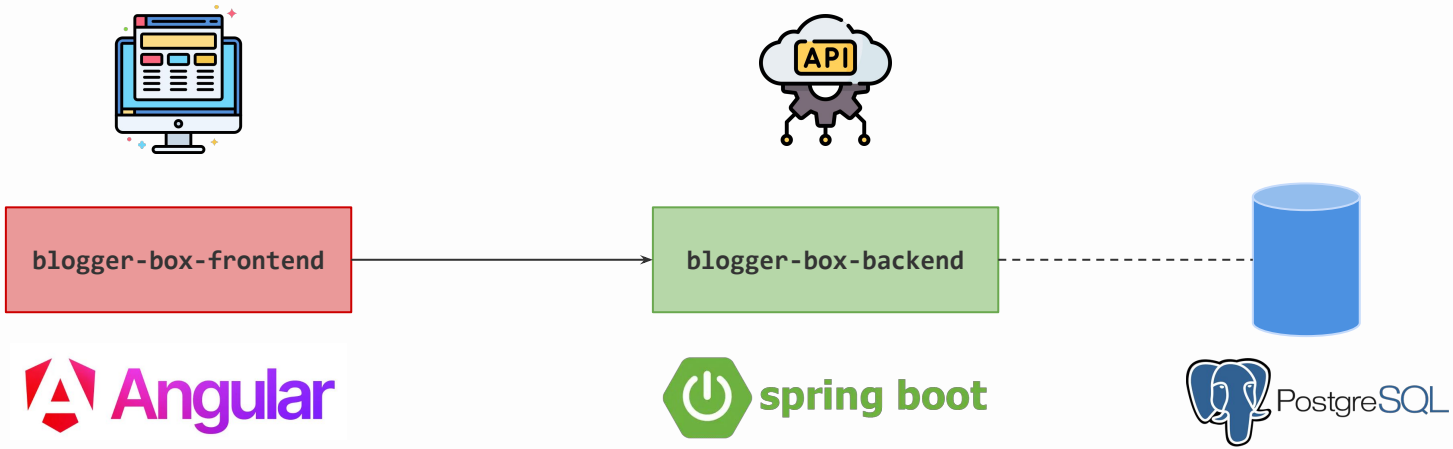
# Building a backend application

With Java 25 and Spring Boot 4.0

# Blogger box

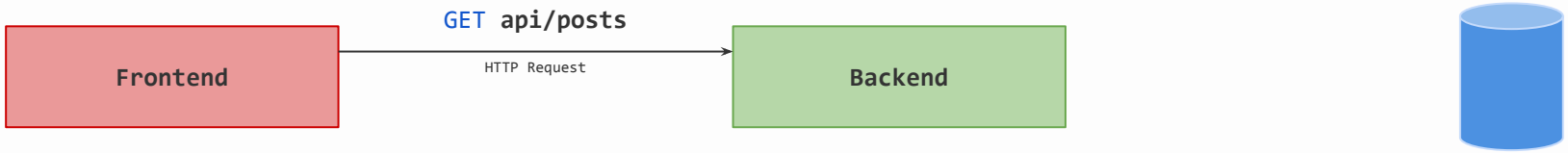

Write
My toolkit for 2024
I didn't expect to write about my toolkit again so soon since I already made a big update in January 2023. Originally, my toolkit was only about design and productivity when I first talked about it in 2019. Now it's grown to encompass every aspect of my life:...
Productivity
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

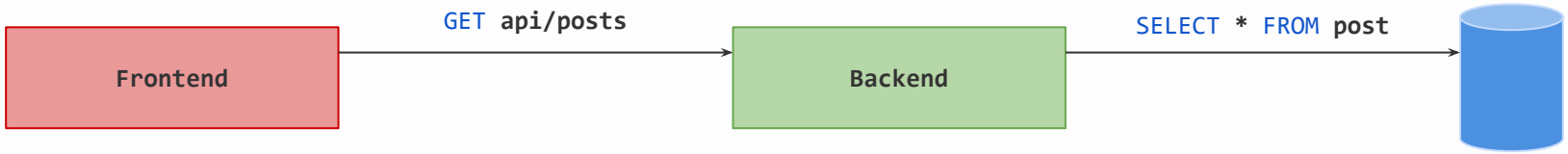


# Flow frontend <-> backend

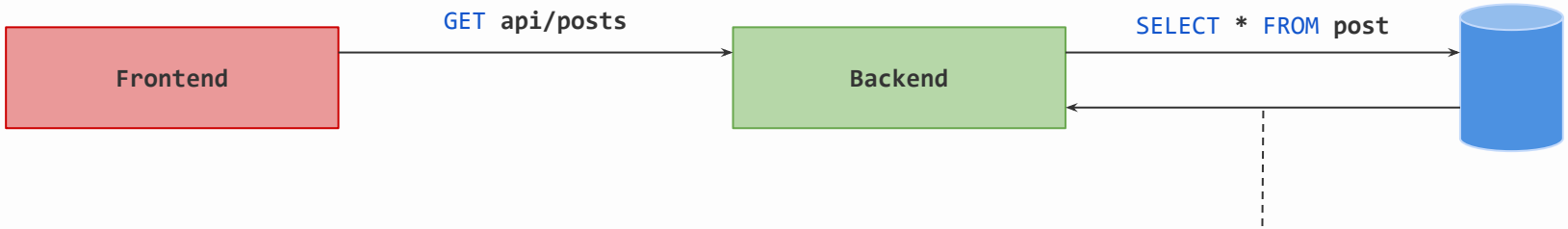
# Flow frontend <-> backend



# Flow frontend <-> backend

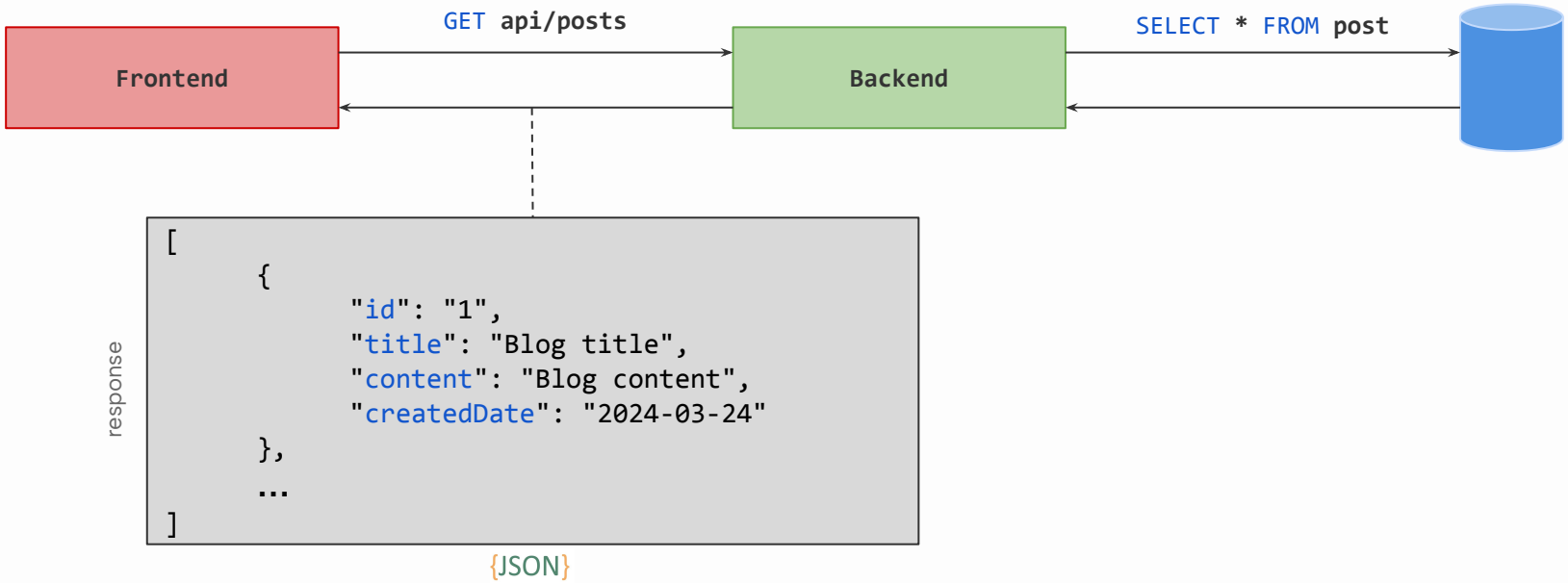


# Flow frontend <-> backend




| 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



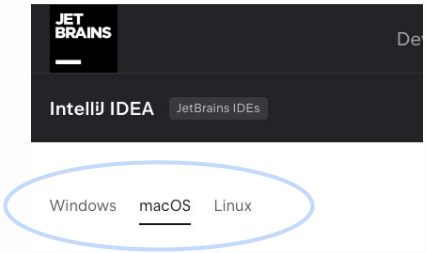
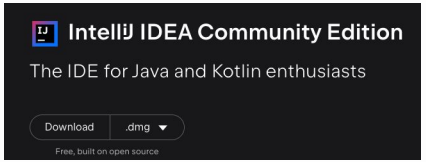
## Session 03

- 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)<br>  |
| **Step 3** | Download Intellij IDEA Community Edition (free)<br>  |

# 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


# Create spring boot app

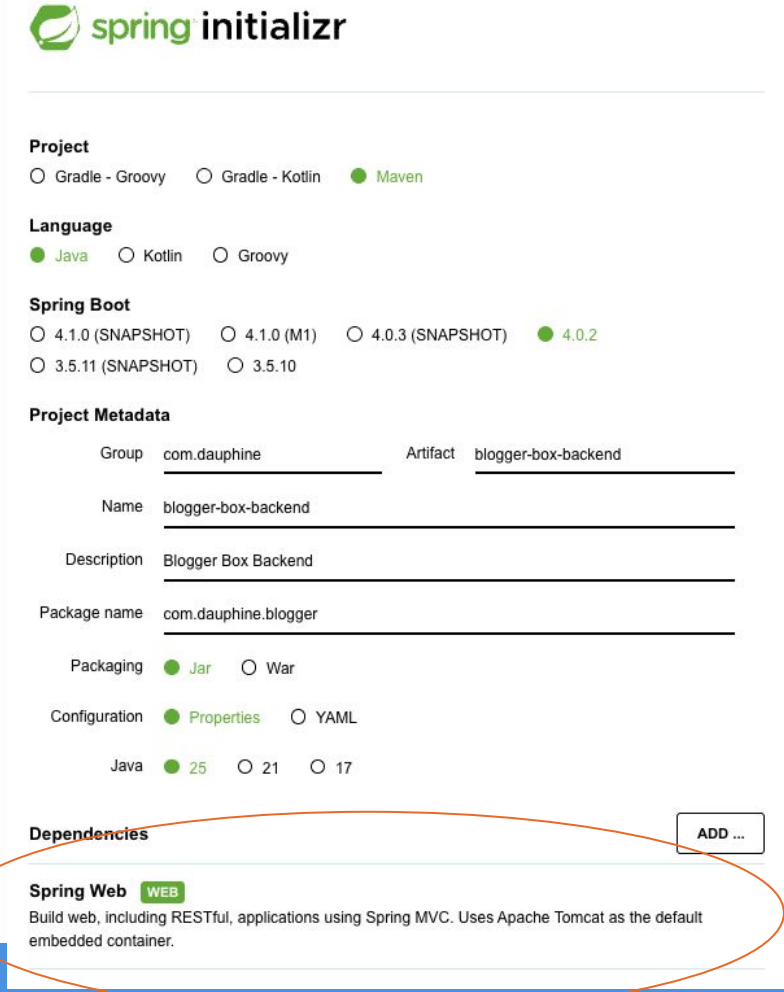
Head over to start.spring.io to create your spring boot application

| | |
|---|---|
| **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 |

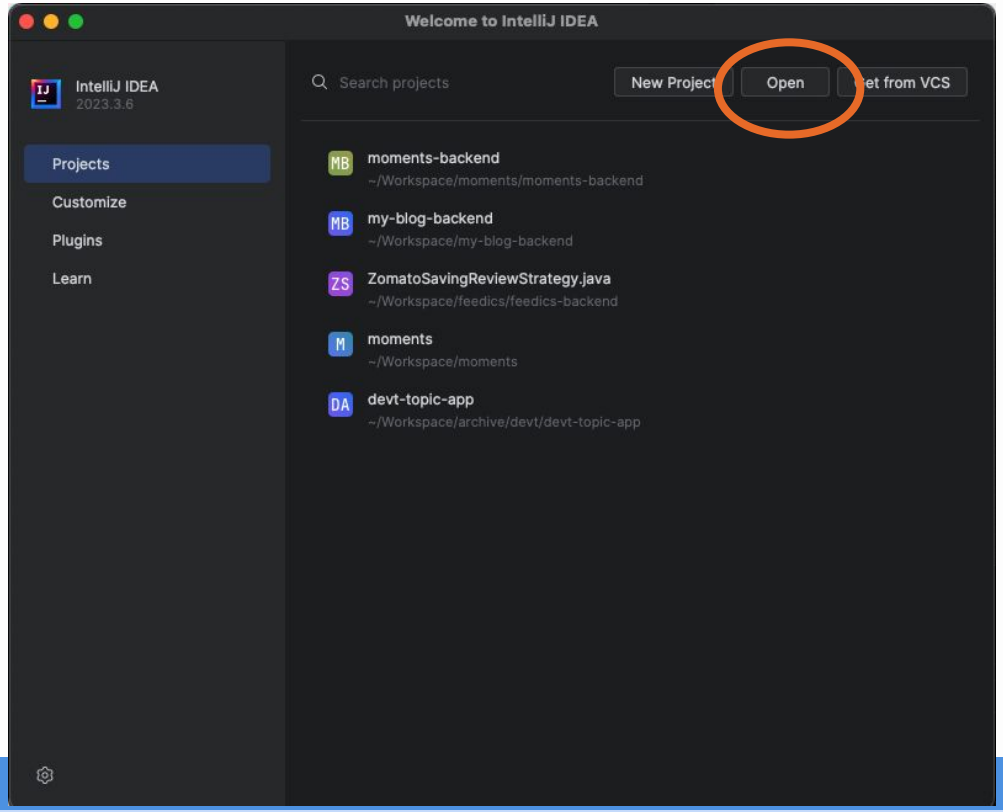
Don't forget to add the dependency

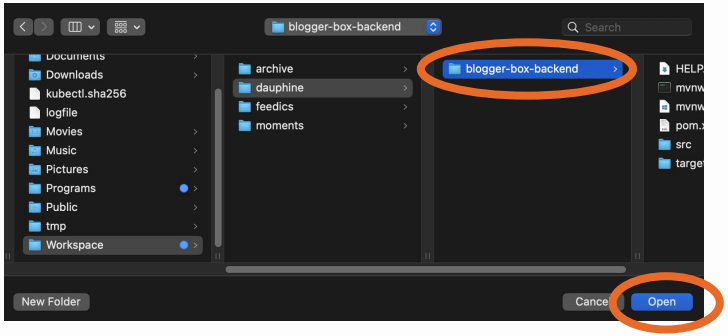
Generate and unzip project

Place project in your workspace

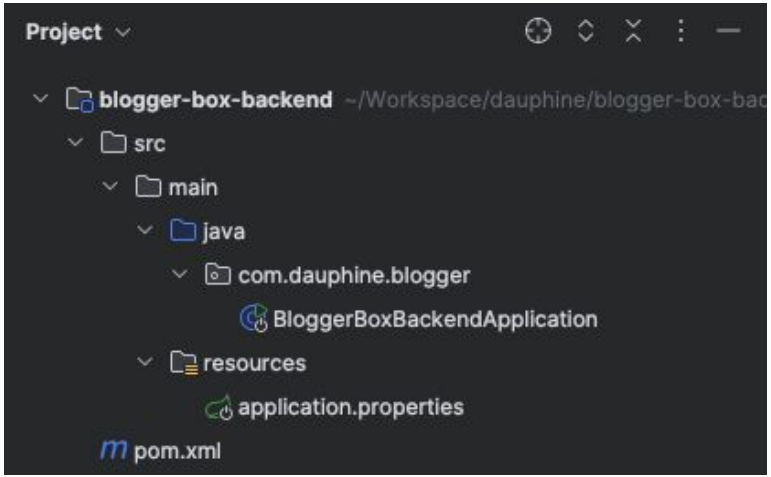



# Open in IDE

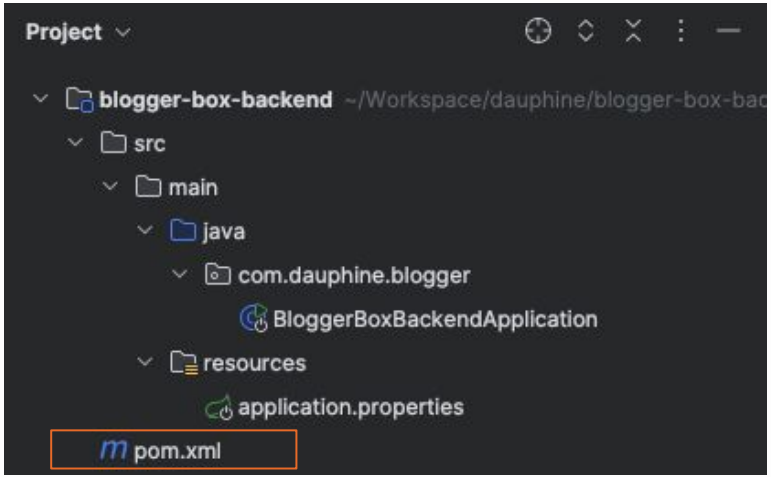

Welcome to IntelliJ IDEA
IntelliJ IDEA
2023.3.6
Projects
Customize
Plugins
Learn
Search projects
New Project
Open
Get from VCS
moments-backend
~/Workspace/moments/moments-backend
my-blog-backend
~/Workspace/my-blog-backend
ZomatoSavingReviewStrategy.java
~/Workspace/feedics/feedics-backend
moments
~/Workspace/moments
devt-topic-app
~/Workspace/archive/devt/devt-topic-app


blogger-box-backend
Search
Downloads
kubectl.sha256
logfile
Movies
Music
Pictures
Programs
Public
tmp
Workspace
archive
dauphine
feedics
moments
blogger-box-backend
New Folder
Cancel
Open

# Structure


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

# Structure


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

# pom.xml

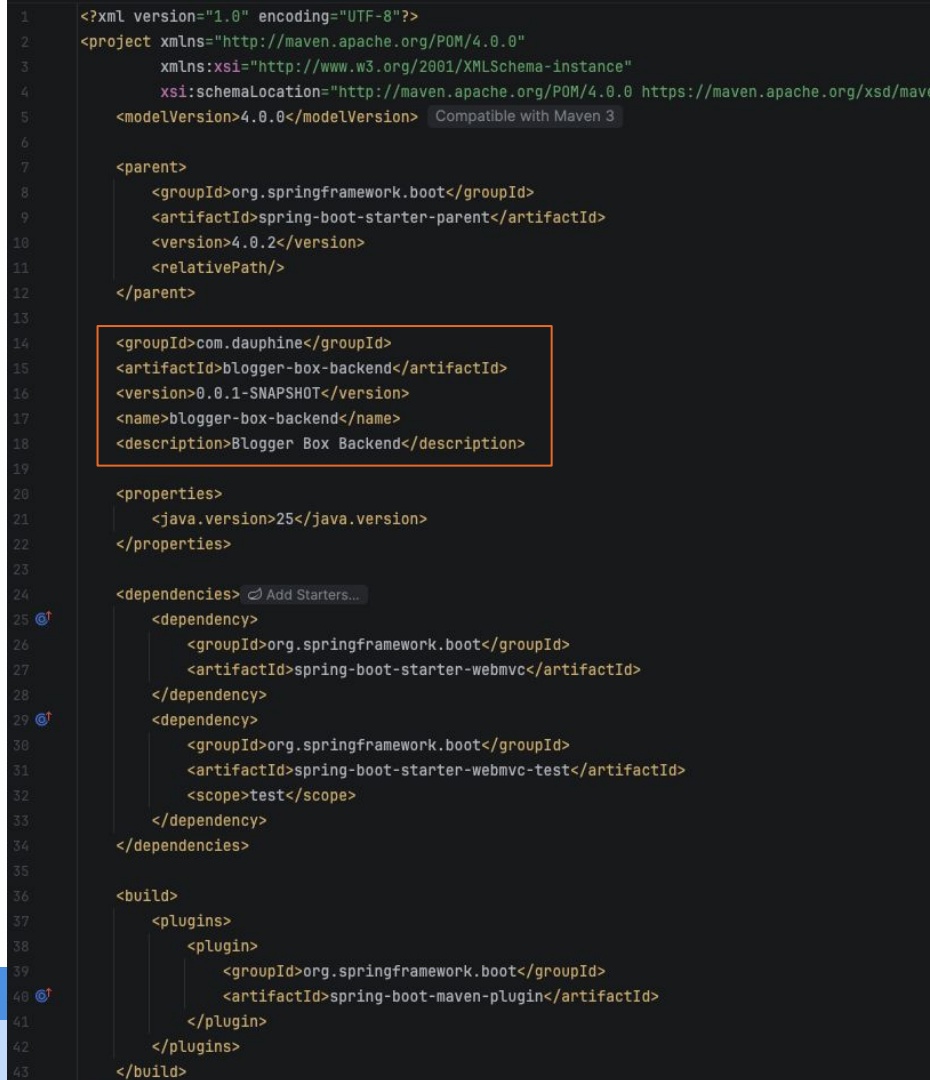
**`pom.xml`** is a configuration file used by **Maven**™, and will contain :

```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/mave
    <modelVersion>4.0.0</modelVersion> Compatible with Maven 3

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>4.0.2</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>25</java.version>
    </properties>

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

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



# 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/mave
    <modelVersion>4.0.0</modelVersion> Compatible with Maven 3

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>4.0.2</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>25</java.version>
    </properties>

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

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

# pom.xml

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

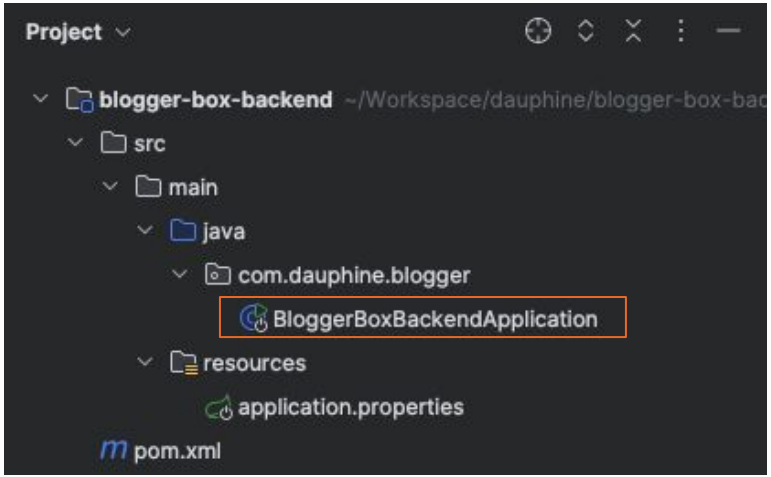
**Project information**

**Dependencies**

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

```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/mave
    <modelVersion>4.0.0</modelVersion> Compatible with Maven 3

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>4.0.2</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>25</java.version>
    </properties>

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

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

# 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)

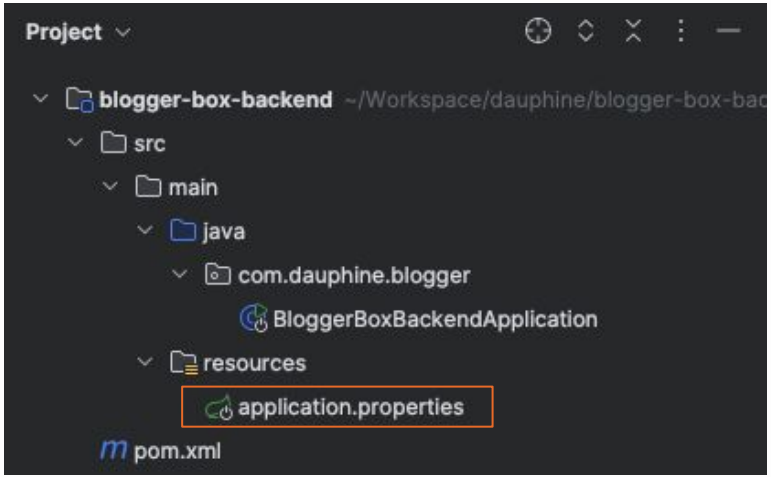
BloggerBoxBackendApplication.java

```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

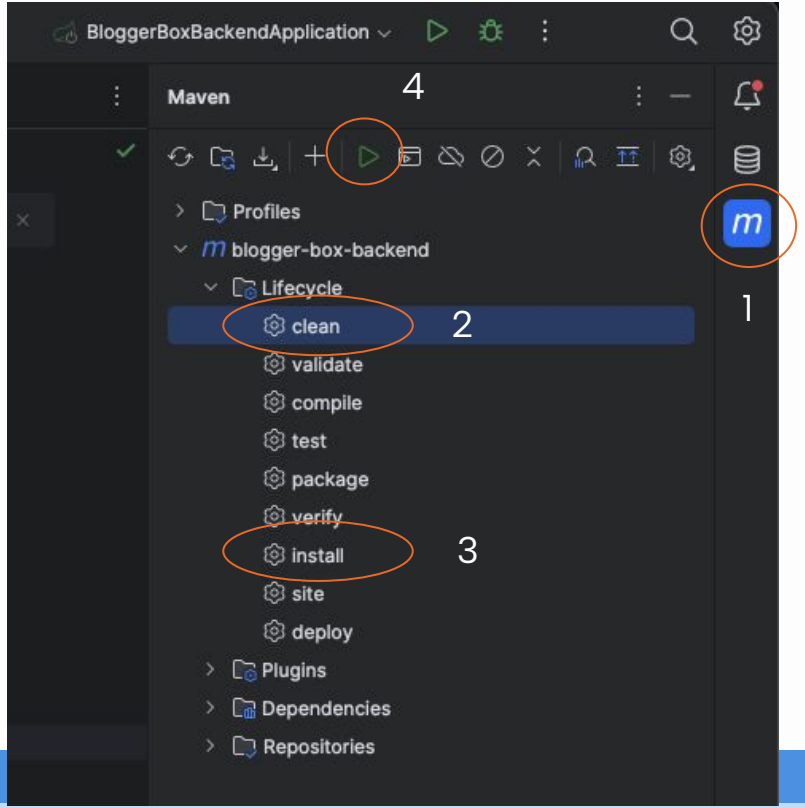
# Application properties

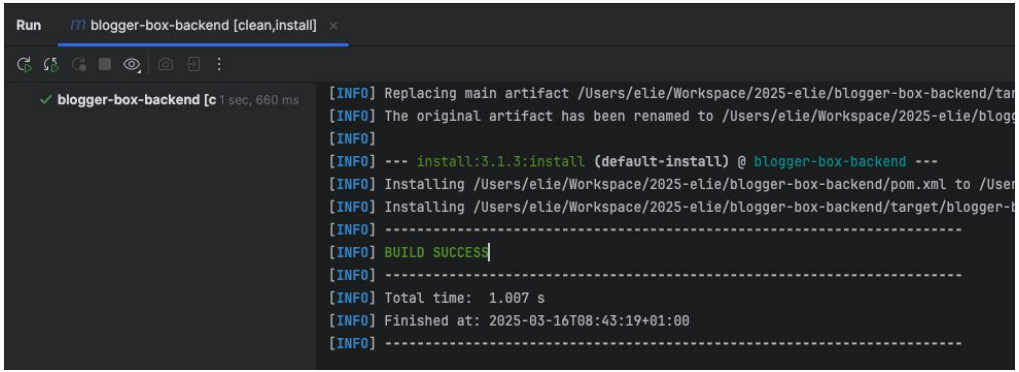

The **`application.properties`** file is a configuration file used to configure various aspect of the application.

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.

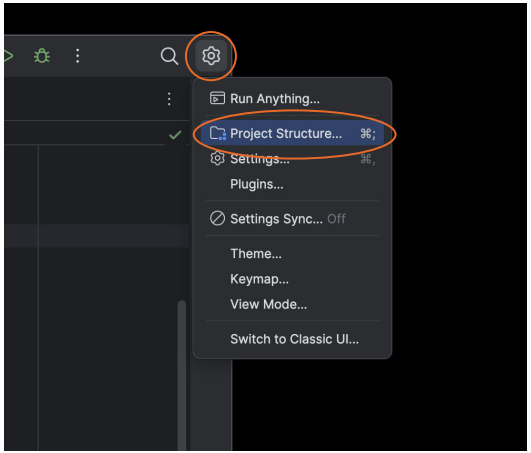
# Compile

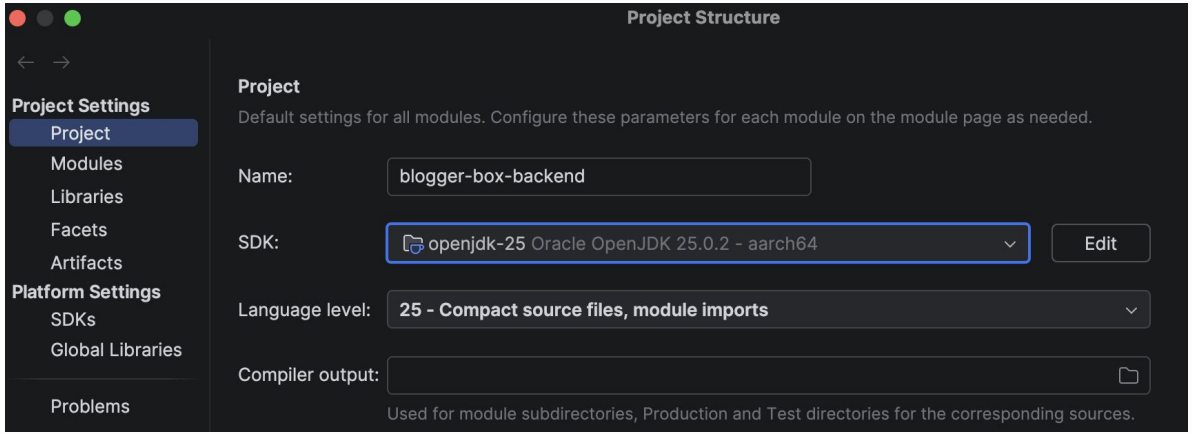

BloggerBoxBackendApplication
Maven
4
Profiles
blogger-box-backend
Lifecycle
clean
2
validate
compile
test
package
verify
install
3
site
deploy
Plugins
Dependencies
Repositories
1


Run
blogger-box-backend [clean,install]
blogger-box-backend [c 1 sec, 660 ms
[INFO] Replacing main artifact /Users/elie/Workspace/2025-elie/blogger-box-backend/tar
[INFO] The original artifact has been renamed to /Users/elie/Workspace/2025-elie/blogg
[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 /User
[INFO] Installing /Users/elie/Workspace/2025-elie/blogger-box-backend/target/blogger-b
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.007 s
[INFO] Finished at: 2025-03-16T08:43:19+01:00
[INFO] ------------------------------------------------------------------------

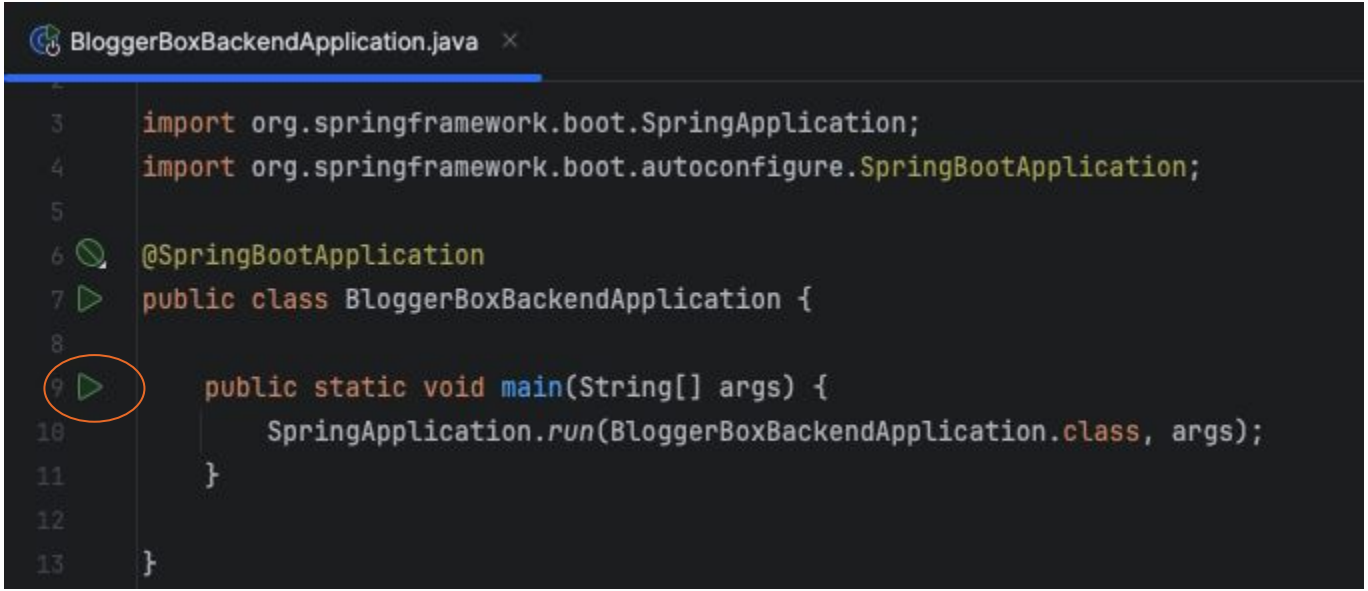
# Compile

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

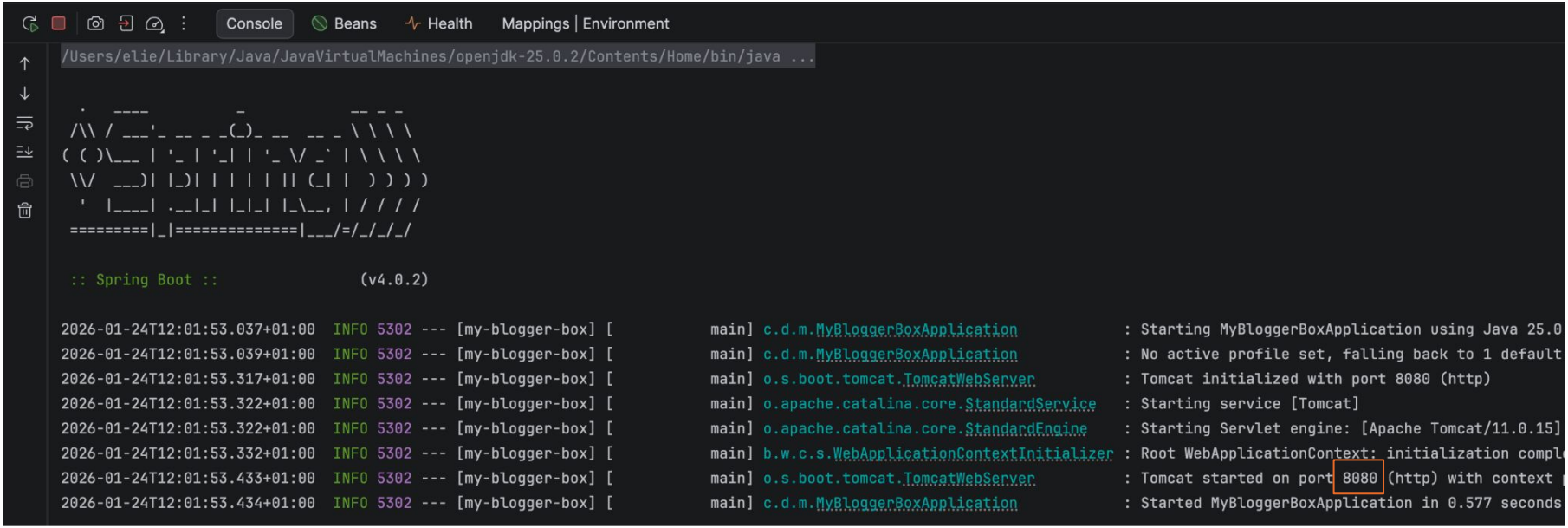


# Start application



# Application started

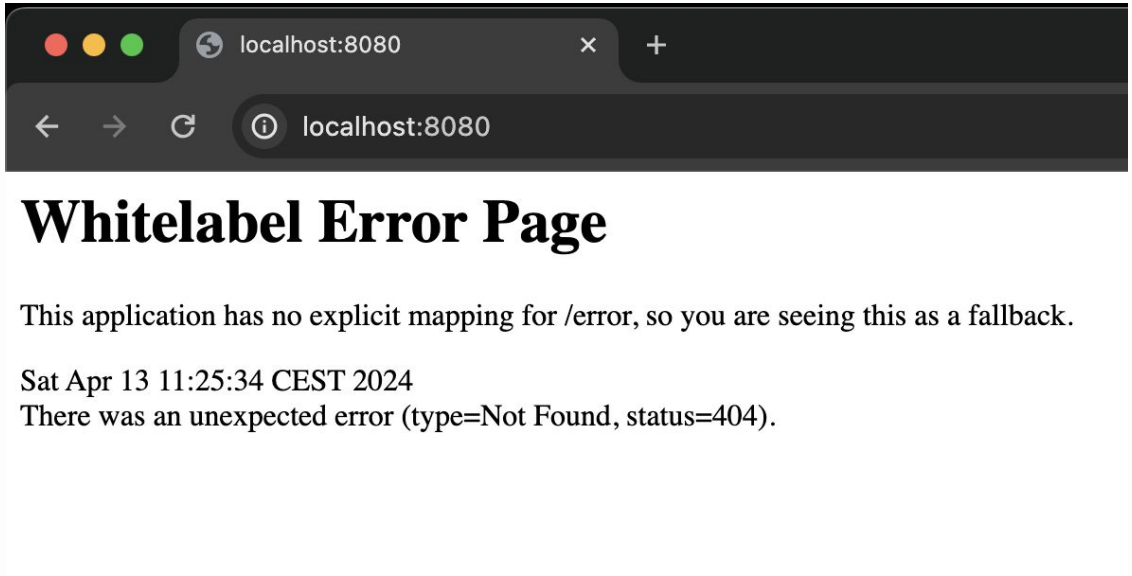



The application is up and running on port 8080 → http://localhost:8080

# localhost:8080

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

The backend application is able to run 🥳

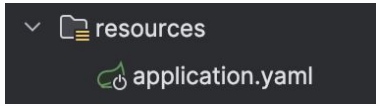


# localhost:1234

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

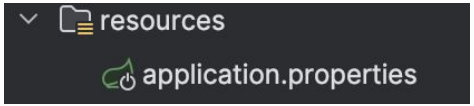
# `.yaml` vs `.properties` for properties

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




application.yaml

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

server:
  port: 1234
```




application.properties

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

server.port=1234
```

Using application.yml vs application.properties in Spring Boot

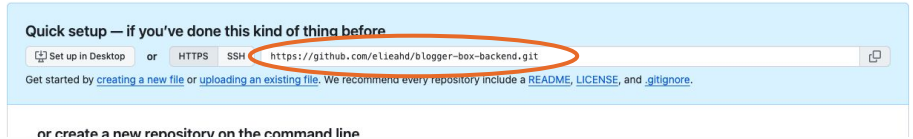
# Publish to github

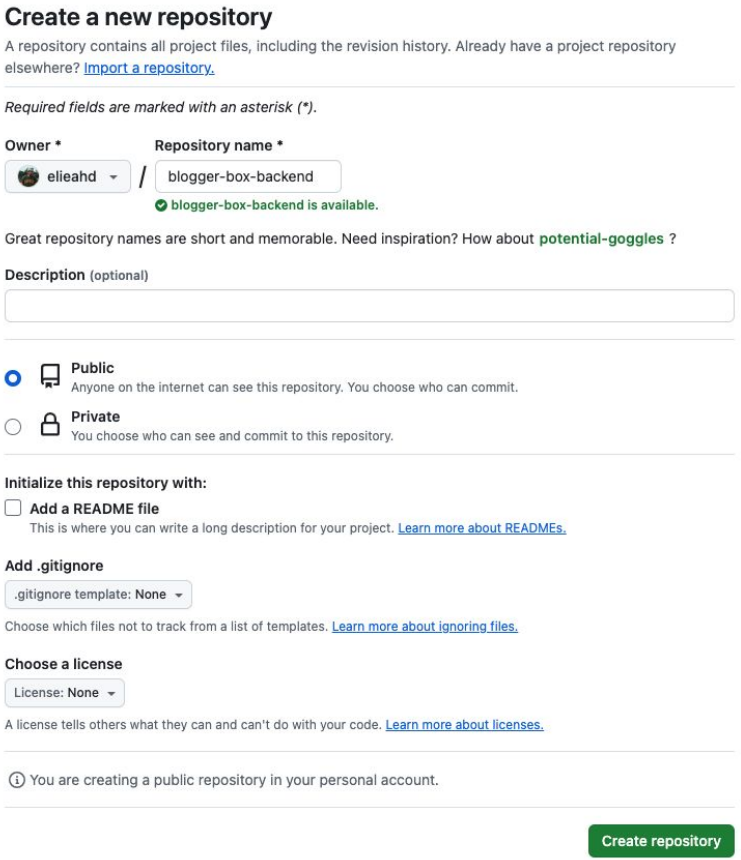
Create a new github repository : **blogger-box-backend**

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

Copy git url

*will be used in the next slide*



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

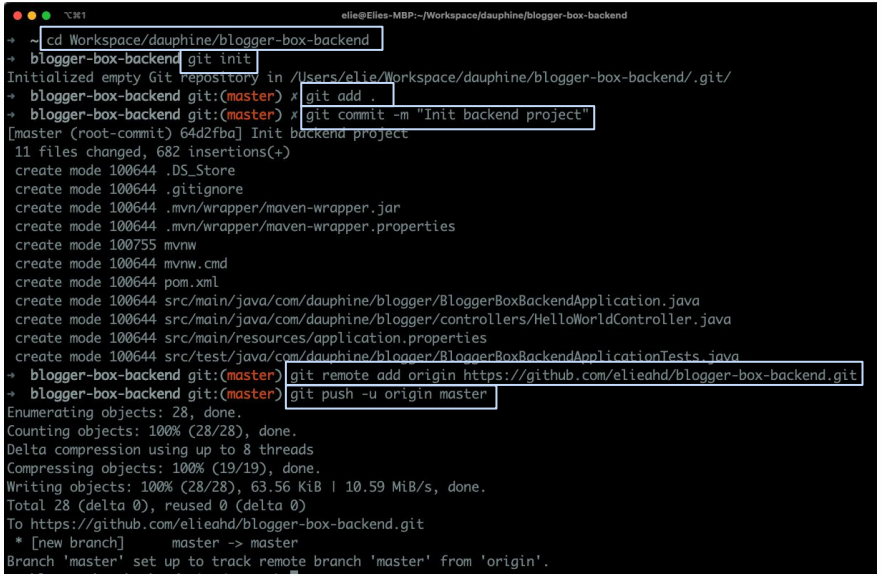
```
git commit -m “COMMIT_MESSAGE”
```

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

```
git remote add origin REMOTE_REPOSITORY_URL
```

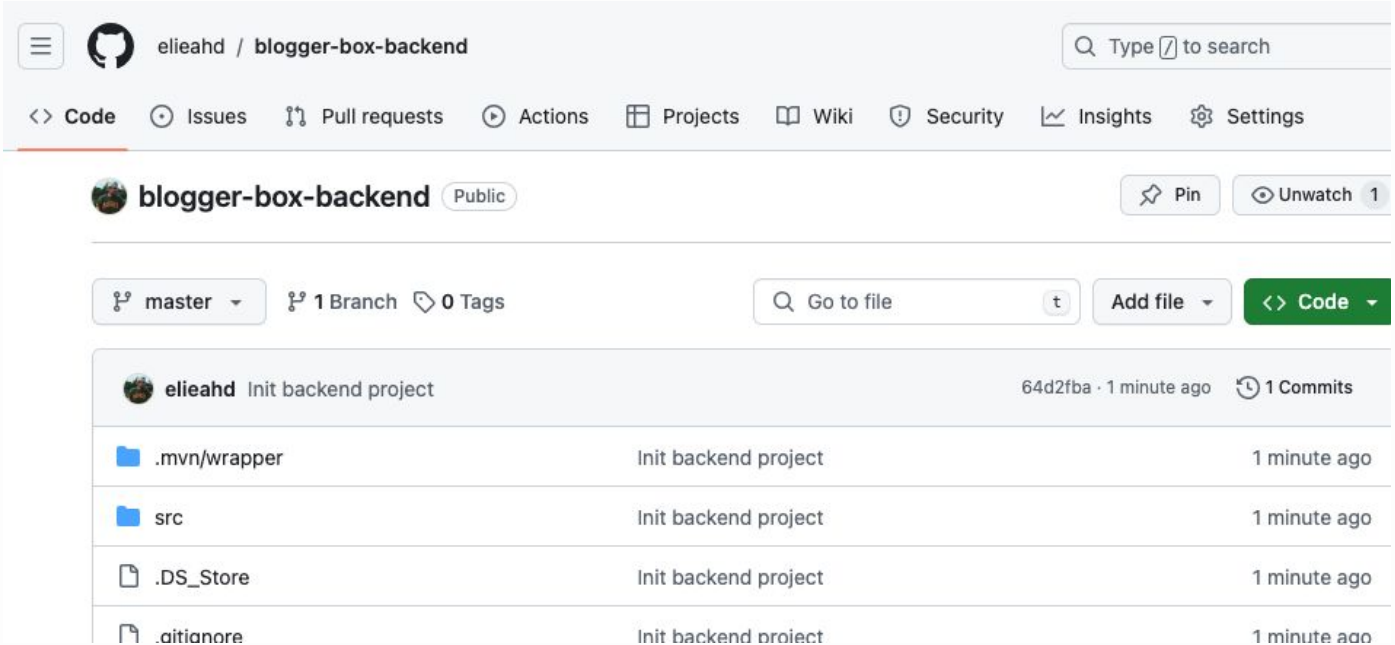
Push the changes to Github

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



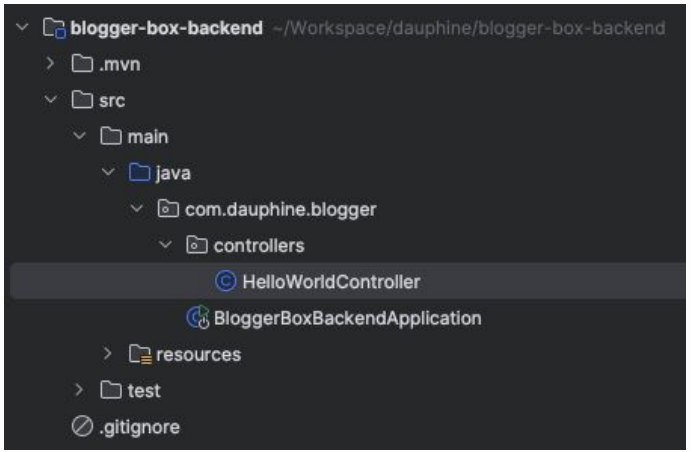

Adding locally hosted code to Github

# Github


elieahd / blogger-box-backend
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
blogger-box-backend
Public
Pin
Unwatch 1
master
1 Branch
0 Tags
Go to file
t
Add file
Code
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

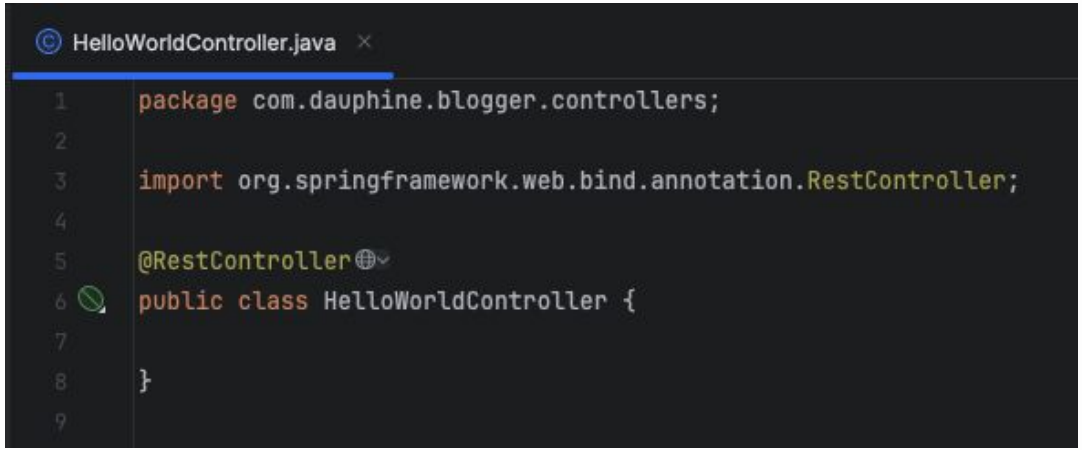
Let's create a new class **`HelloWorldController`** under controllers



# 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

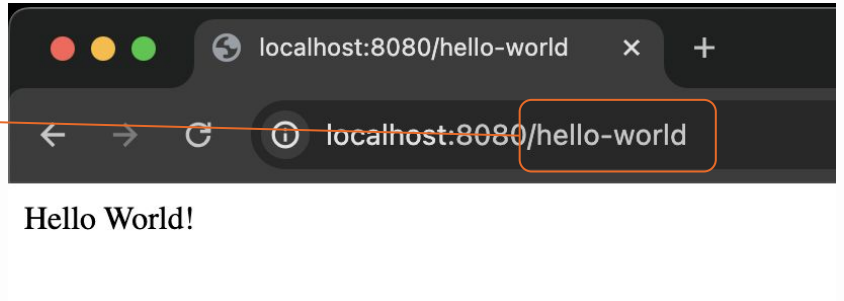


# Expose an endpoint

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

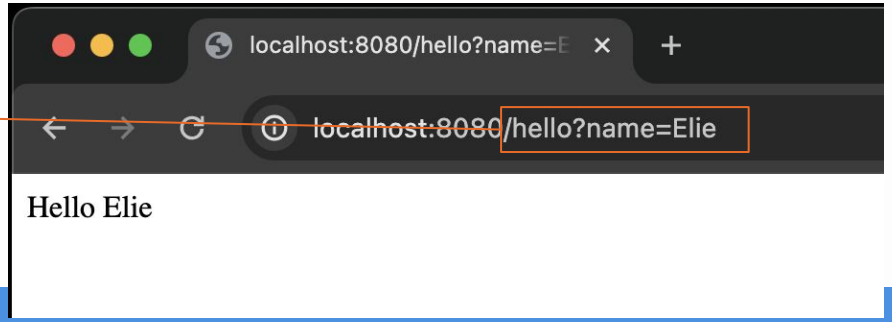


# Expose an endpoint with Request Param

```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

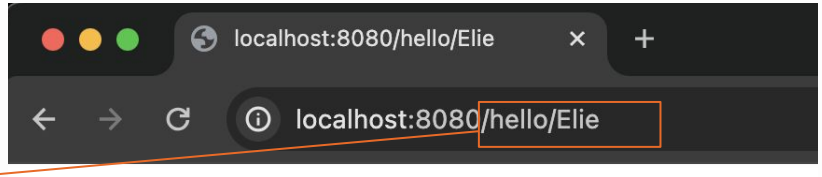


# Expose an endpoint with Path Variable

```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 tool that allow us to document and test our endpoints

Add following dependency in **pom.xml**

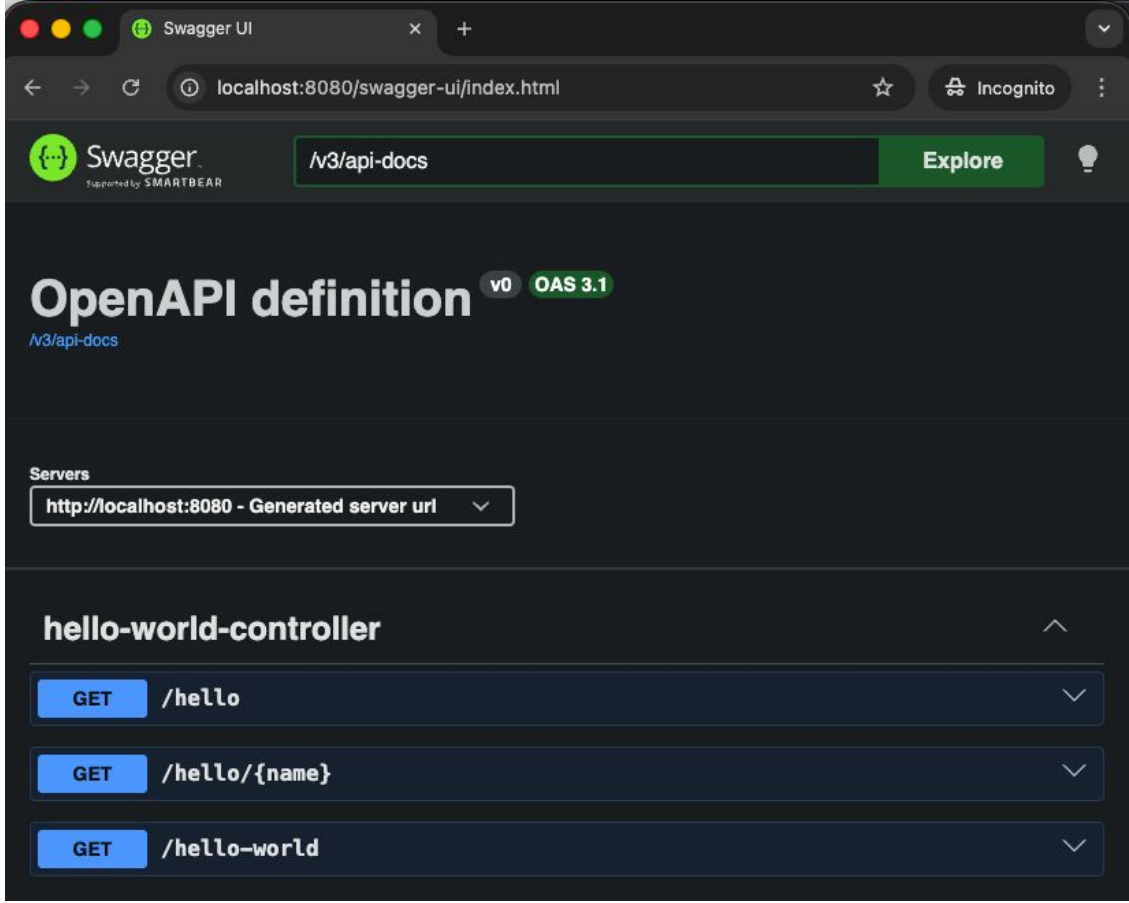
```
<dependencies> Add Starters...
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webmvc</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>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

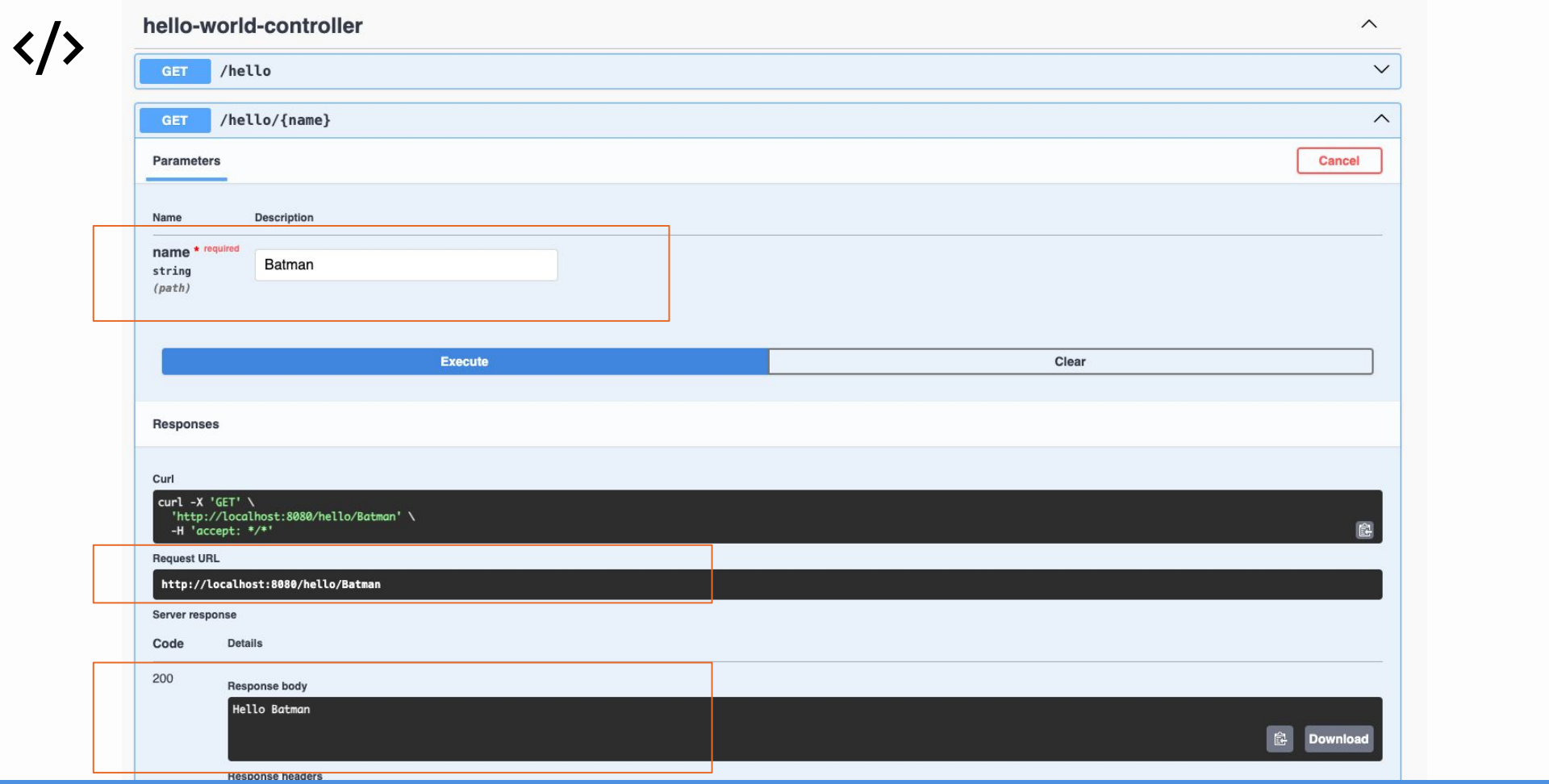
```
<dependency>
   <groupId>org.springdoc</groupId>
   <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
   <version>3.0.1</version>
</dependency>
```

# Swagger

Head over to

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

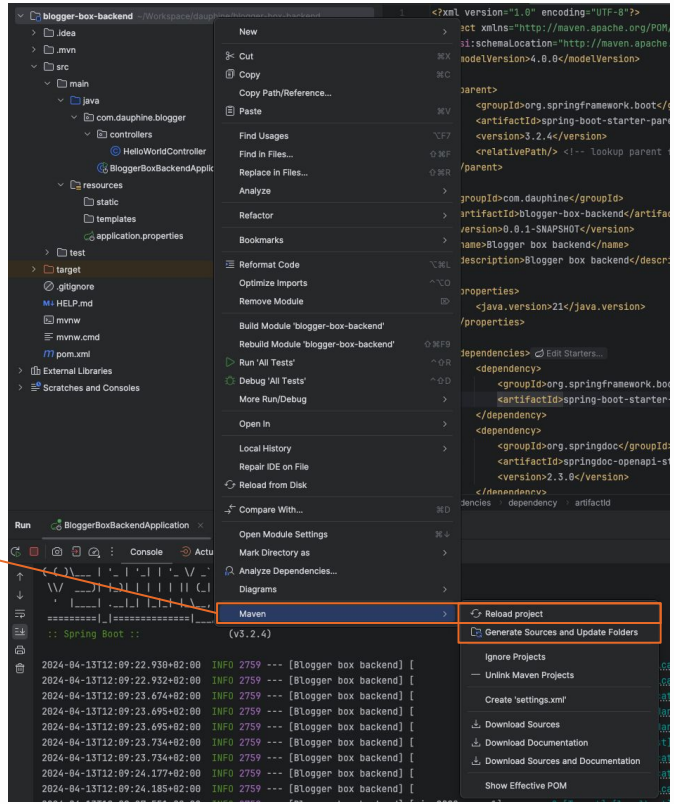


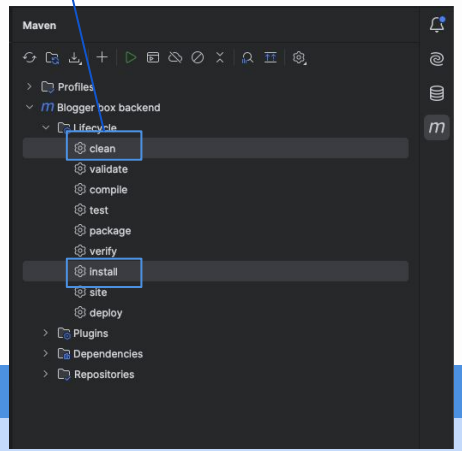

hello-world-controller
GET /hello
GET /hello/{name}
Parameters
Cancel
Name
Description
name * required
string
(path)
Batman
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost:8080/hello/Batman' \
-H 'accept: */*'
Request URL
http://localhost:8080/hello/Batman
Server response
Code
Details
200
Response body
Hello Batman
Download
Response headers

# Swagger

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

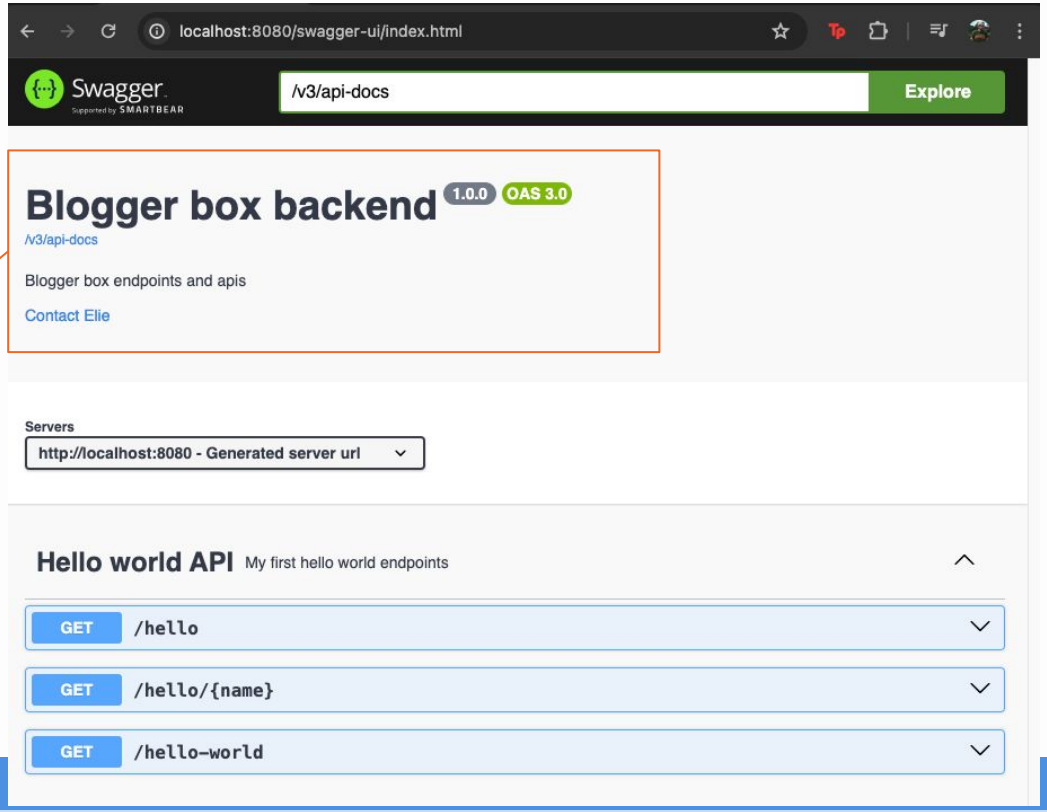
> Reload project
> Generate sources and update folders
> mvn clean install



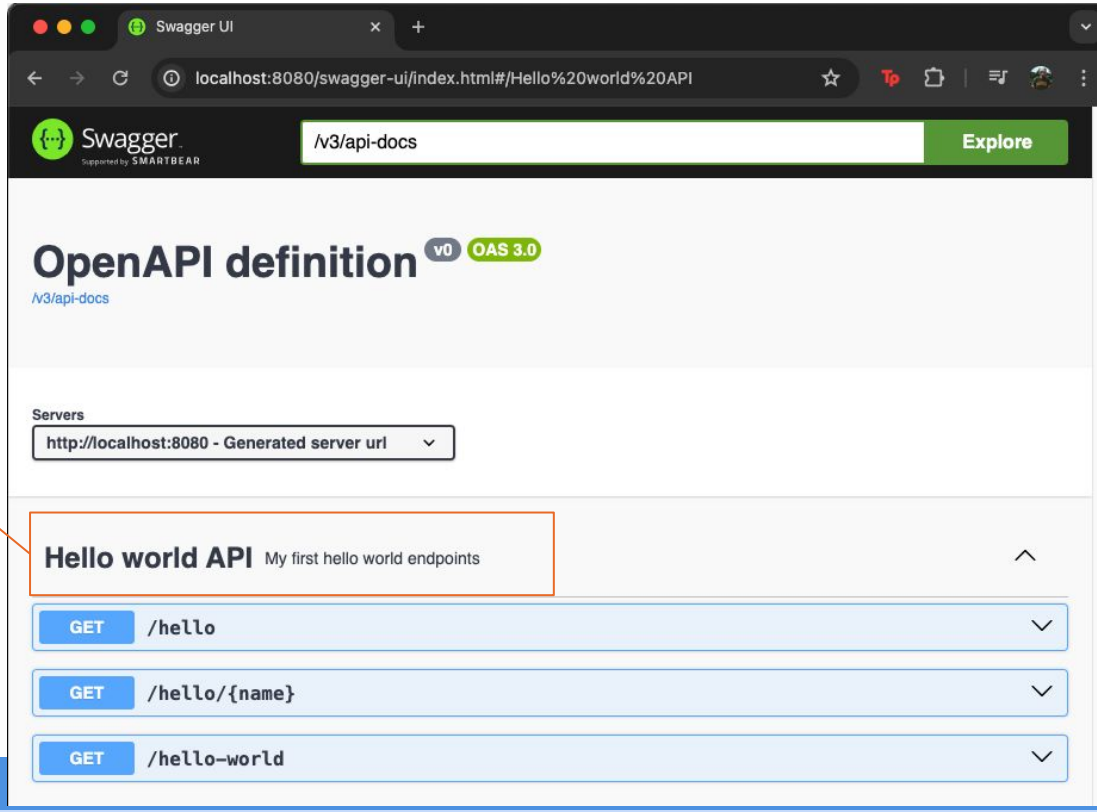
# Write Application documentation

```java
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);
    }

}
```



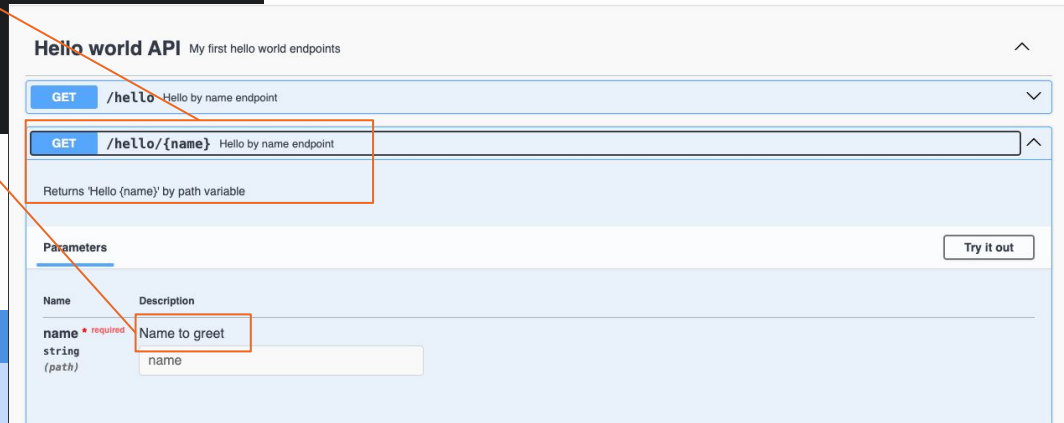
# Write Controller documentation

```java
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() {
```



# Write Endpoint documentation

```java
@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;
}
```



# 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 /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**

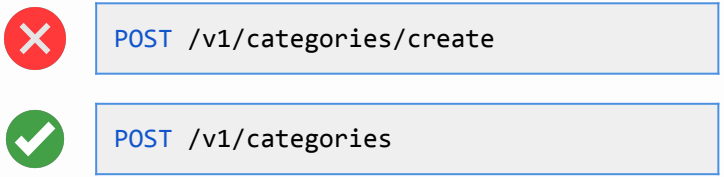
**Use lowercase letters**

**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*

# 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 {

    private String title;
    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` 😉*)*

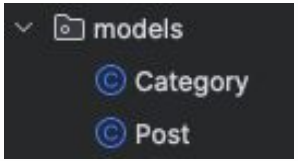
***Tip #3*** *use* `@RequestMapping` *on the controller to start all endpoints of each controller the same way*

```
@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("/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*

# 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 it 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`**

```java
public interface CategoryService {  5 usages  1 implementation

    List<Category> getAll();  1 usage  1 implementation

    Category getById(UUID id);  2 usages  1 implementation

    Category create(String name);  1 implementation

    Category updateName(UUID id, String name);  1 usage  1 implementation

    boolean deleteById(UUID id);  1 usage  1 implementation
}
```

CategoryServiceImpl.java

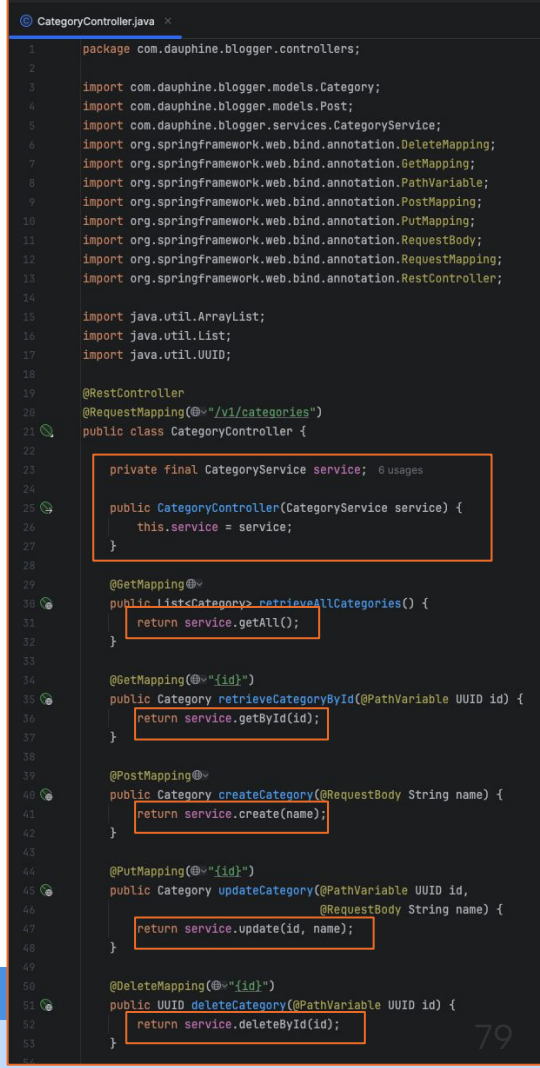
```java
package com.dauphine.blogger.services.impl;

import com.dauphine.blogger.models.Category;
import com.dauphine.blogger.services.CategoryService;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
import java.util.UUID;

@Service
public class CategoryServiceImpl implements CategoryService {

    private final List<Category> temporaryCategories;  9 usages

    public CategoryServiceImpl() {
        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"));
    }

    @Override  no usages
    public List<Category> getAll() { return temporaryCategories; }

    @Override  no usages
    public Category getById(UUID id) {
        return temporaryCategories.stream() Stream<Category>
                .filter(category -> id.equals(category.getId()))
                .findFirst() Optional<Category>
                .orElse( other: null);
    }

    @Override  no usages
    public Category create(String name) {
        Category category = new Category(UUID.randomUUID(), name);
        temporaryCategories.add(category);
        return category;
    }

    @Override  no usages
    public Category update(UUID id, String newName) {
        Category category = temporaryCategories.stream() Stream<Category>
                .filter(c -> id.equals(c.getId()))
                .findFirst() Optional<Category>
                .orElse( other: null);
        if (category != null) {
            category.setName(newName);
        }
        return category;
    }

    @Override  no usages
    public void deleteById(UUID id) { temporaryCategories.removeIf(category -> id.equals(category.getId())); }

}
```

# CategoryController

| | |
|---|---|
| Controller | Management of the REST endpoints |
| Service | Business Logic Implementations |

Inject **CategoryService** in the constructor

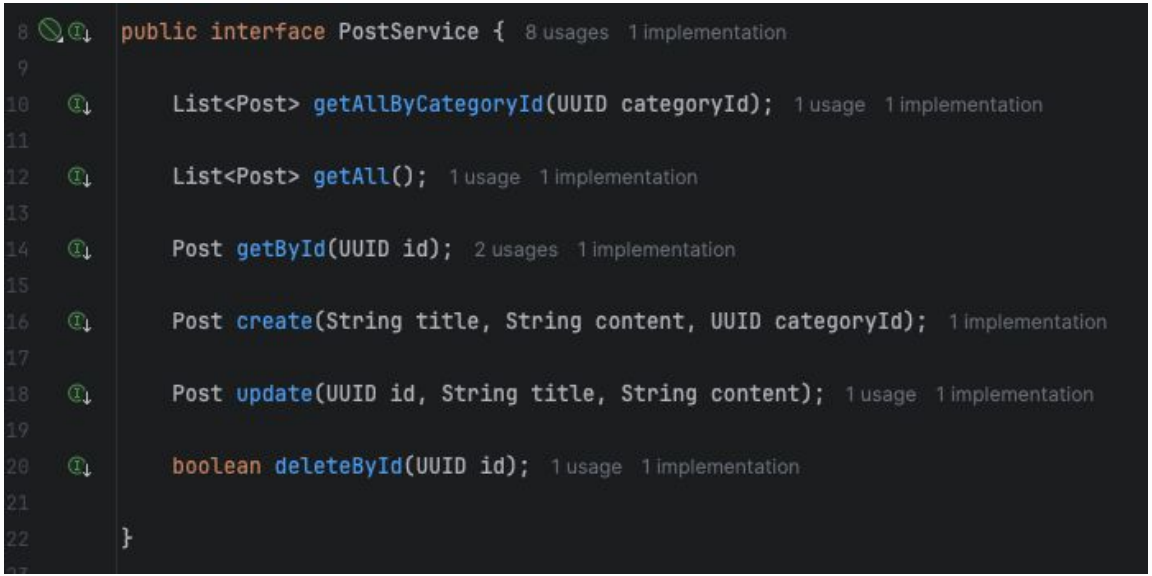
The **implementation** and **business logic** of each method will be done at the service layer



# Sync with Github

*add `category` service layer*

# Implement PostService and the changes in the controller



```java
public interface PostService {  8 usages  1 implementation

    List<Post> getAllByCategoryId(UUID categoryId);  1 usage  1 implementation

    List<Post> getAll();  1 usage  1 implementation

    Post getById(UUID id);  2 usages  1 implementation

    Post create(String title, String content, UUID categoryId);  1 implementation

    Post update(UUID id, String title, String content);  1 usage  1 implementation

    boolean deleteById(UUID id);  1 usage  1 implementation

}
```

# Sync with Github

*add `post` service layer*