

**Lab Manual- Spring Boot Zipkin , Swagger and SpringAdmin**

**Prepared for**:

**Date:** 18th June 2024

**Prepared by:**

Document Name: Lab Manual **Document Number** SDLab333

**Contributor:**

Contents

[1. Objective 3](#_Toc173068631)

[2. Clone the Code from Git Repository and Download Zipkin 3](#_Toc173068632)

[2.1 Clone the Git Repository 3](#_Toc173068633)

[2.2 Download the Zipkin 4](#_Toc173068634)

[3. Understand FruitController class 6](#_Toc173068635)

[1. Code Explanation 6](#_Toc173068636)

[2. Zipkin's Point of View 6](#_Toc173068637)

[3. Summary 7](#_Toc173068638)

[4. Update Code for Zipkin | get Endpoint | Monitor with Zipkin | Get Document with Swagger (SpringDoc) 7](#_Toc173068639)

[3.1 Customized Code for Zipkin 7](#_Toc173068640)

[3.1 Get the EndPoint 9](#_Toc173068641)

[1. Breakdown of the URL 9](#_Toc173068642)

[2. What Happens When You Access the URL 10](#_Toc173068643)

[2.1 Get the Traces with Zipkin 10](#_Toc173068644)

[2.1 Get the API Visual Documentation with Swagger 11](#_Toc173068645)

[3. Launch Swagger UI 13](#_Toc173068646)

[5. Update Code SpringAdmin 13](#_Toc173068647)

[5.1 Key Features of Spring Boot Admin 14](#_Toc173068648)

[5.2 How to Set Up Spring Boot Admin (Example) 14](#_Toc173068649)

[1. Set Up Spring Boot Admin Server 14](#_Toc173068650)

[2. Set Up Spring Boot Admin Client 14](#_Toc173068651)

[3. Enable Spring Boot Admin Server: 14](#_Toc173068652)

[5.3 How to Set Up Spring Boot Admin (Demo) 15](#_Toc173068653)

# Objective

# Clone the Code from Git Repository and Download Zipkin

## Clone the Git Repository

* Open **PowerShell** in Administrative Mode, Change Directory and Create New Directory

**Cd /**

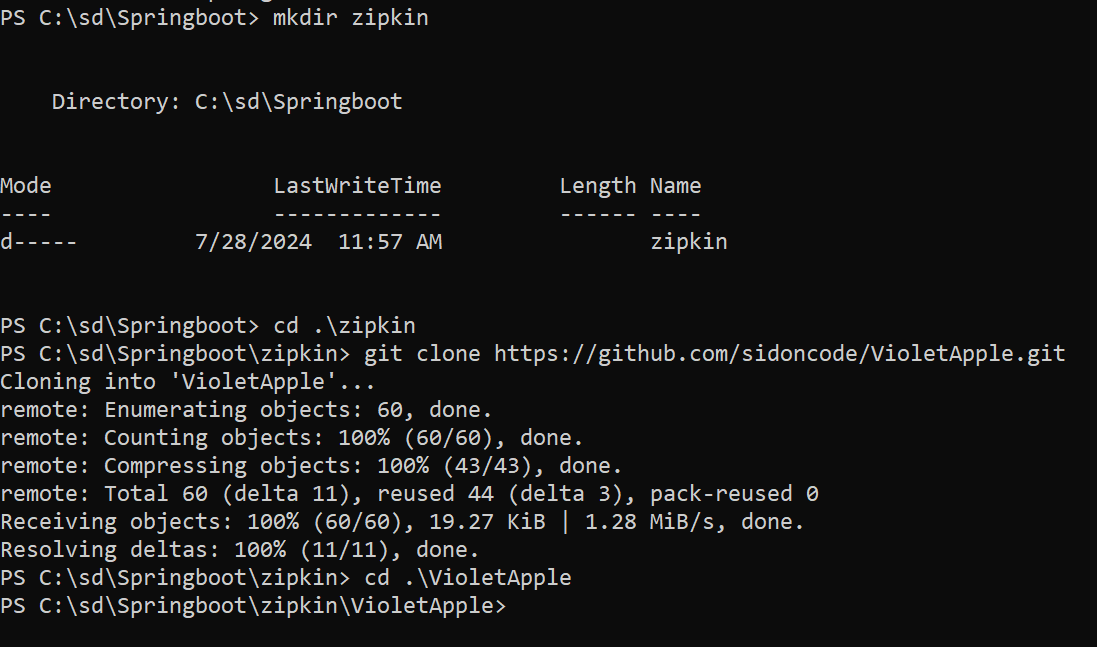
**Mkdir ZipKin**

**Cd ZipKin**

**Git clone** [**https://github.com/sidoncode/VioletApple.git**](https://github.com/sidoncode/VioletApple.git)

**Cd VioletApple**

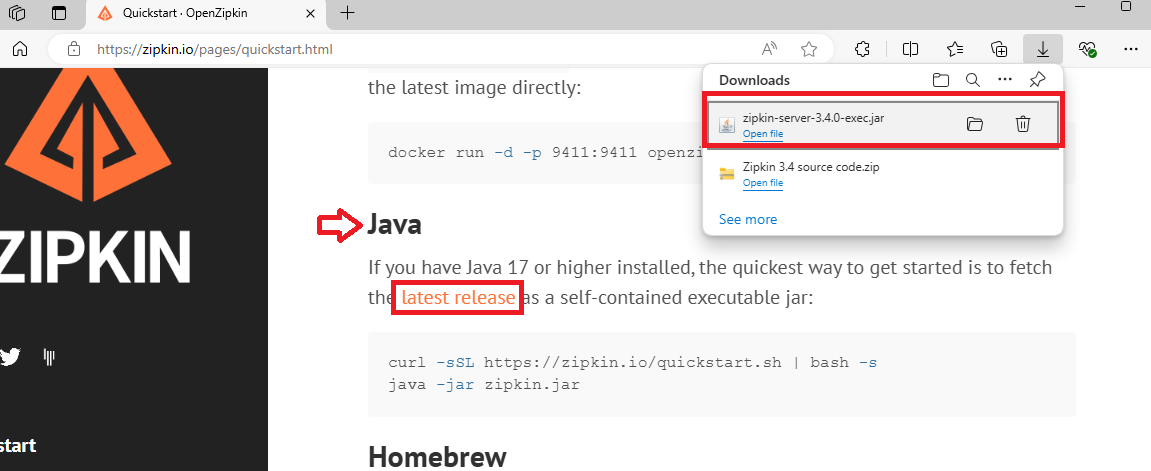




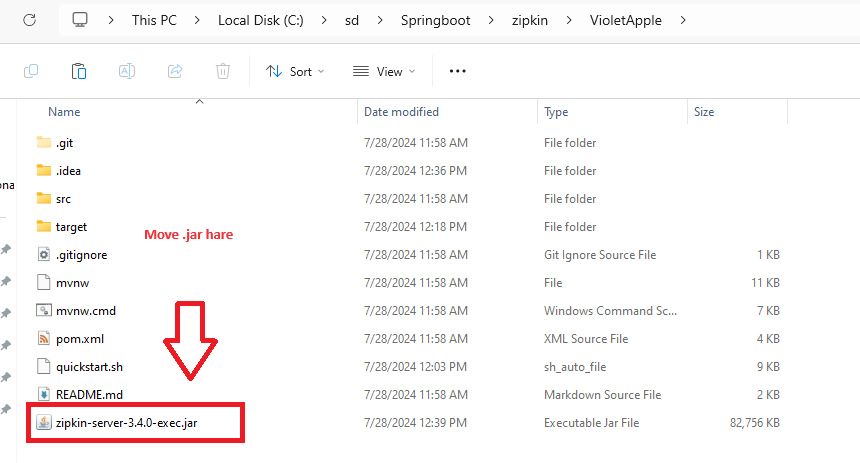
## Download the Zipkin

* Download **Zipkin** on windows under **Java – Latest Release**

<https://zipkin.io/pages/quickstart.html>



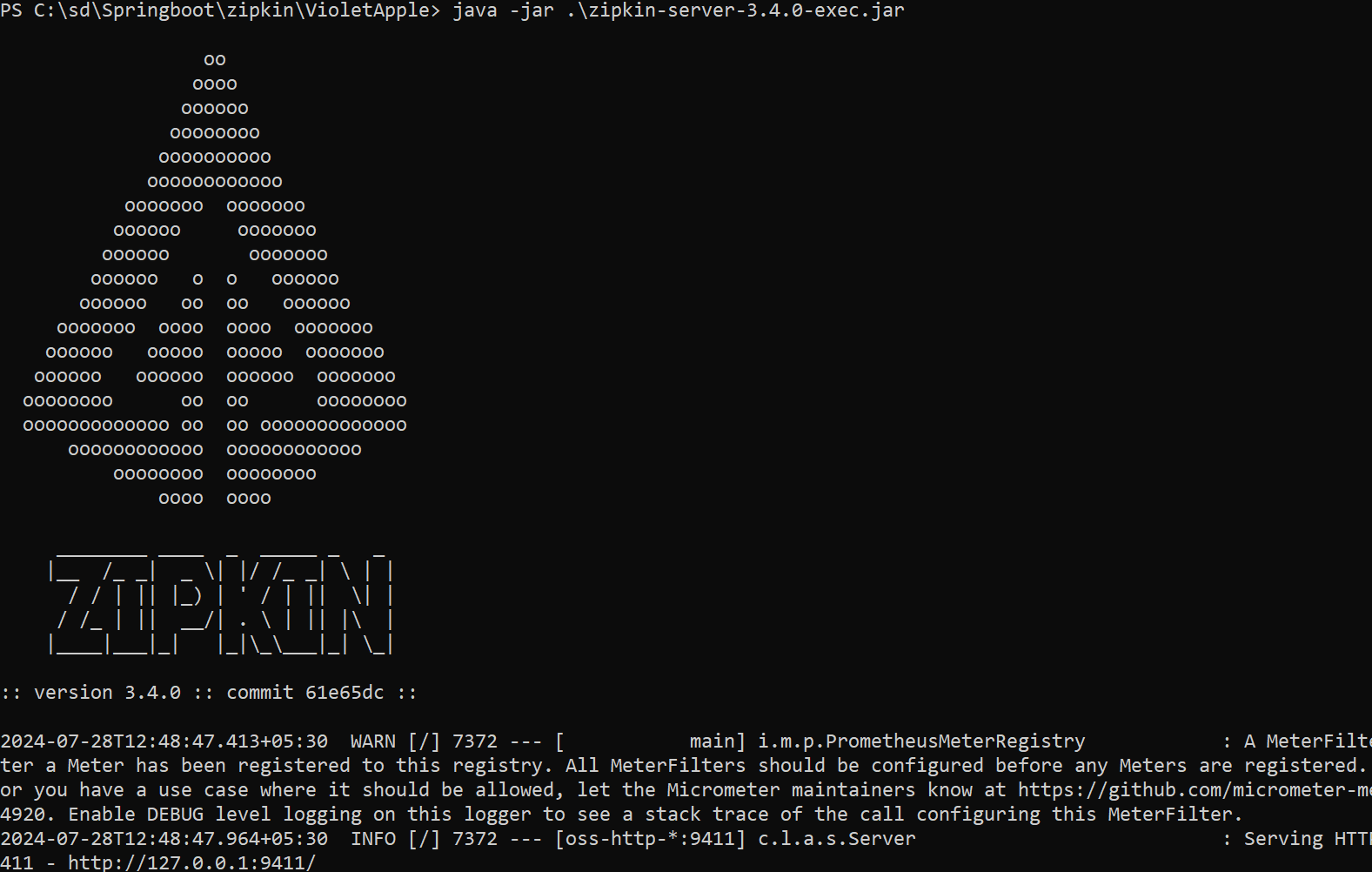
* Copy the downloaded **.jar** file to your git cloned project



* Execute the .**jar** file make Zipkin up inside VioletApple Directory .

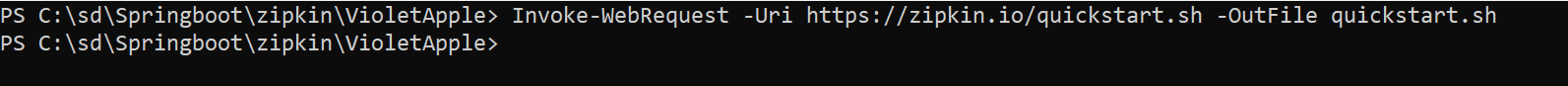
**java -jar .\zipkin-server-3.4.0-exec.jar**

Note : After Executing Jar File you should able to see the url in the last http://127.0.0.0/9411/



* Download **Zipkin** on windows

**Invoke-WebRequest -Uri https://zipkin.io/quickstart.sh -OutFile quickstart.sh**

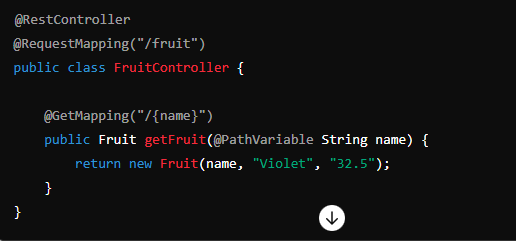


# Understand FruitController class

The **FruitController class** is a **REST controller** in a Spring Boot application.

When viewed from the perspective of **Zipkin**, which is a **distributed tracing system**, this class is significant because it **represents an endpoint** that can be **traced for performance** and **debugging purposes**.

Here is a breakdown of the code and its relevance to Zipkin:



### Code Explanation

1. **Annotations:**
   * @RestController: This annotation indicates that the class is a REST controller, meaning it handles HTTP requests and returns responses in a RESTful manner.
   * @RequestMapping("/fruit"): This annotation maps HTTP requests to /fruit to methods in this controller.
2. **Class Definition:**
   * public class FruitController: Defines a public class named FruitController.
3. **Method:**
   * @GetMapping("/{name}"): This annotation maps HTTP GET requests to /fruit/{name} to the getFruit method.
   * public Fruit getFruit(@PathVariable String name): This method takes a path variable name from the URL and returns a Fruit object. The Fruit object is created with the given name, a hardcoded color ("Violet"), and a hardcoded weight ("32.5").

### Zipkin's Point of View

Zipkin is a tool for collecting and visualizing traces in a distributed system. Here's how Zipkin interacts with this controller:

1. **Tracing HTTP Requests:**
   * When an HTTP GET request is made to /fruit/{name}, Zipkin can trace this request. The trace will include information about the request duration, the endpoint being called, and other relevant metadata.
2. **Span Creation:**
   * A span is a unit of work in a trace. Zipkin will create a span for the getFruit method call. This span will capture the time taken to execute the method, along with any tags or annotations added for additional context.
3. **Propagating Context:**
   * Zipkin propagates trace context across service boundaries. If the FruitController calls other services or methods that are also instrumented with Zipkin, the trace context will be passed along, creating a full trace of the request flow.
4. **Performance Monitoring:**
   * By tracing the getFruit endpoint, Zipkin helps monitor the performance of this method. If there are delays or errors, they can be identified and debugged using the trace information.
5. **Example Trace Flow**
6. **Client Request:**
   * A client makes a GET request to /fruit/apple.
7. **Trace Initialization:**
   * Zipkin starts a new trace and records the incoming request as the root span.
8. **Method Execution:**
   * The getFruit method is executed. Zipkin creates a span for this method call, recording the start time, end time, and any relevant tags (e.g., method name, path variable).
9. **Response:**
   * The method returns a Fruit object, and the span is completed.
10. **Trace Completion:**
    * Once the response is sent back to the client, the trace is completed and sent to the Zipkin server for storage and visualization.

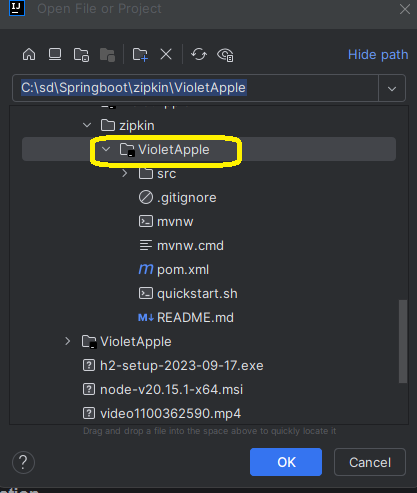
### Summary

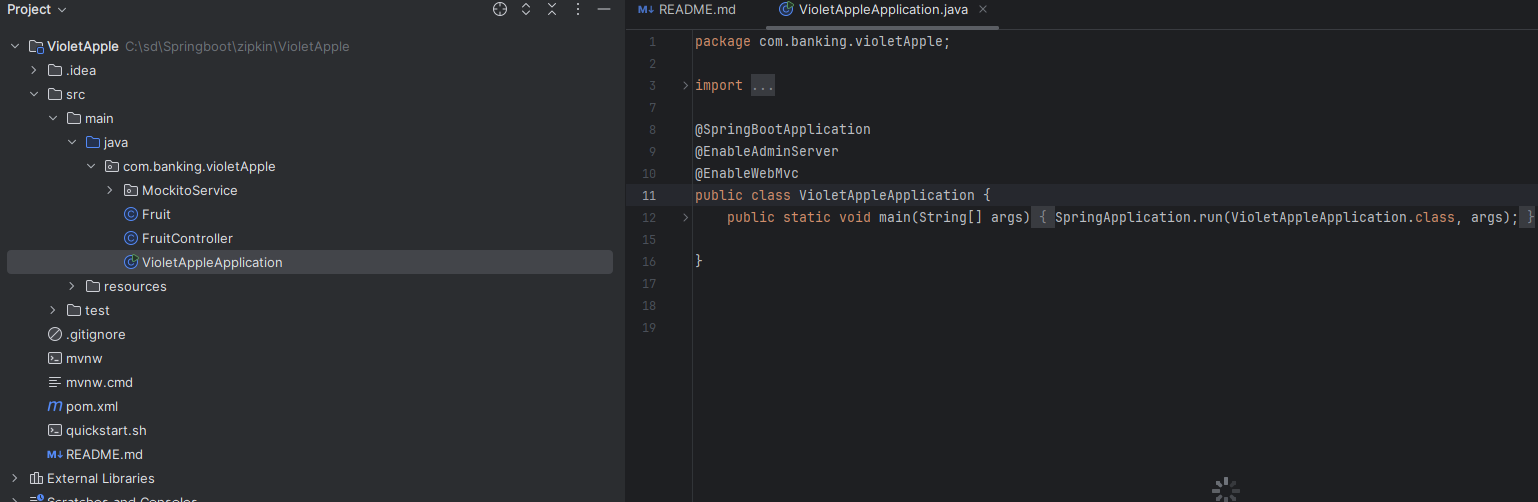
* + In summary, from Zipkin's point of view, the **FruitController class** represents an endpoint that can be traced for performance and debugging purposes. **Each request to this controller generates trace data that** can be used to **monitor** and analyze the behavior and performance of the application.

# Update Code for Zipkin | get Endpoint | Monitor with Zipkin | Get Document with Swagger (SpringDoc)

## Customized Code for Zipkin

* Open Project in **IntelliJ**

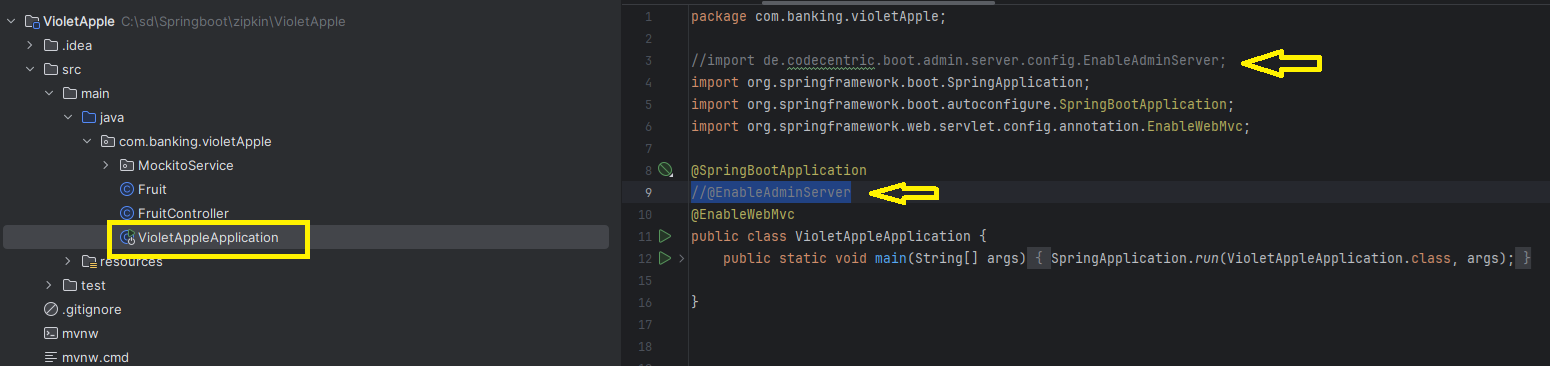




* Traverse to Main Application in Package **Src/Main/java/com.banking.violetAppie** and Comment Line no **#3** and Line **#9**

**//import de.codecentric.boot.admin.server.config.EnableAdminServer;**

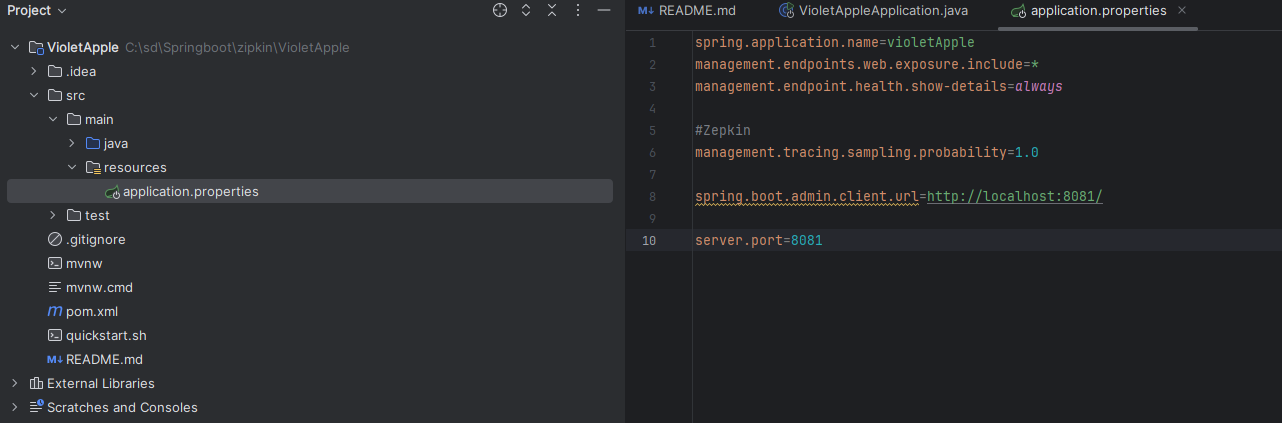
**//@EnableAdminServer**

****

* Then in **application.properties** Change localpost to 8081 and also add new server port 8081

**spring.boot.admin.client.url=http://localhost:8081/**

**server.port=8081**

****

* Now Save the File and Run Application

## Get the EndPoint

This demonstrates how a **RESTful endpoint** in a Spring Boot application processes an **HTTP GET** request and returns a resource representation based on the request parameters.

 The URL **http://localhost:8081/fruit/apple** triggers the **getFruit** method in the **FruitController**.

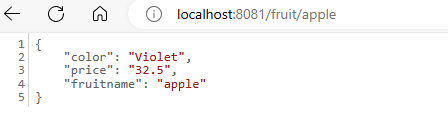
 It extracts the path variable **apple** and uses it to create a new **Fruit** object.

 The **Fruit** object is then returned as a **JSON** response to the client.

The URL **http://localhost:8081/fruit/apple** is used to access a **specific endpoint** in a Spring Boot application that is **mapped by** the FruitController class.

Type below URL to fire get data using Zipkin

<http://localhost:8081/fruit/apple>



### Breakdown of the URL

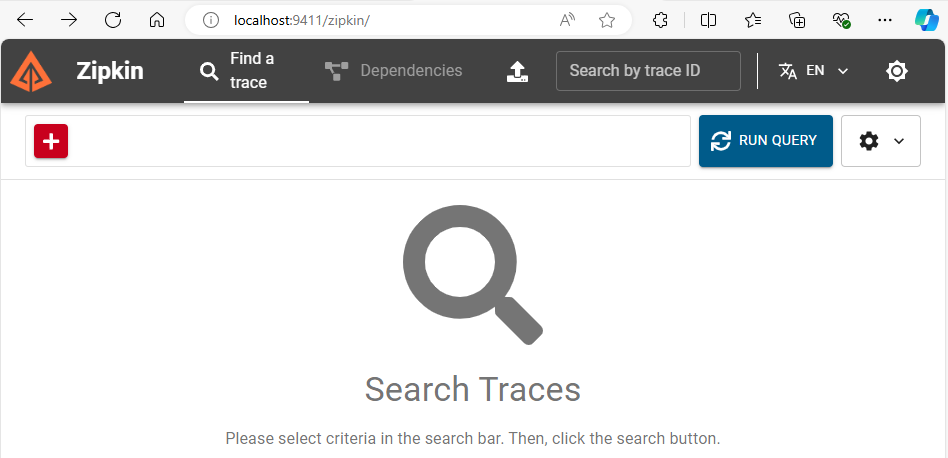
* **http**: The protocol used for the request.
* **localhost**: The hostname, indicating that the application is running on the local machine.
* **8081**: The port on which the Spring Boot application is listening.
* **/fruit/apple**: The path that specifies the endpoint in the FruitController.

### What Happens When You Access the URL

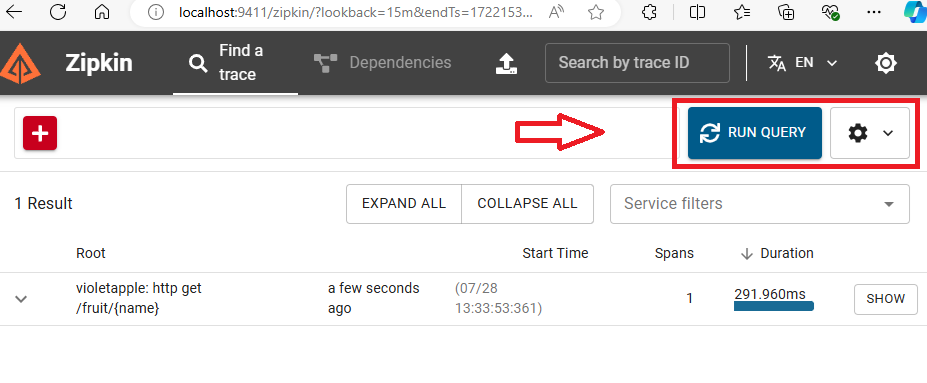
1. **HTTP GET Request:**
   * An HTTP GET request is made to the URL http://localhost:8081/fruit/apple.
2. **Request Mapping:**
   * The request is mapped to the FruitController class because of the @RequestMapping("/fruit") annotation.
3. **Path Variable Extraction:**
   * The @GetMapping("/{name}") annotation maps the apple part of the URL to the name parameter of the getFruit method.
4. **Method Execution:**
   * The getFruit method is called with the name parameter set to apple.
   * A new Fruit object is created with the name "apple", color "Violet", and weight "32.5".
5. **Response:**
   * The Fruit object is returned as the response, typically serialized to JSON by Spring Boot.

## Get the Traces with Zipkin

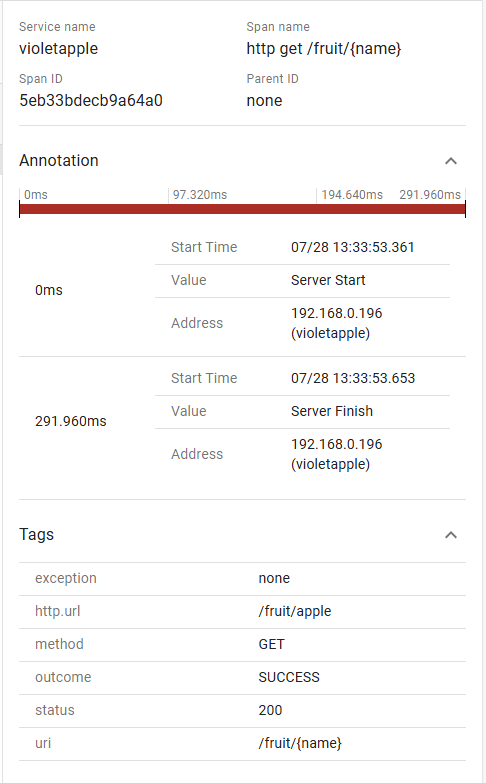
<http://localhost:9411/zipkin/>



Click on Run Query to get latest



* Click **Show** Button to Get Latest



## Get the API Visual Documentation with Swagger

Swagger UI is a tool for visualizing and interacting with the API endpoints defined in your application. When you navigate to http://localhost:8081/swagger-ui/index.html, you will see a **web-based interface** that **displays the documentation** for the **API endpoints** provided by your Spring Boot application.

To Design API use Swagger Editor :

<https://editor.swagger.io/>

There are other commonly used endpoints and related features when using **springdoc-openapi** . These endpoints provide the OpenAPI (Swagger) documentation in JSON format

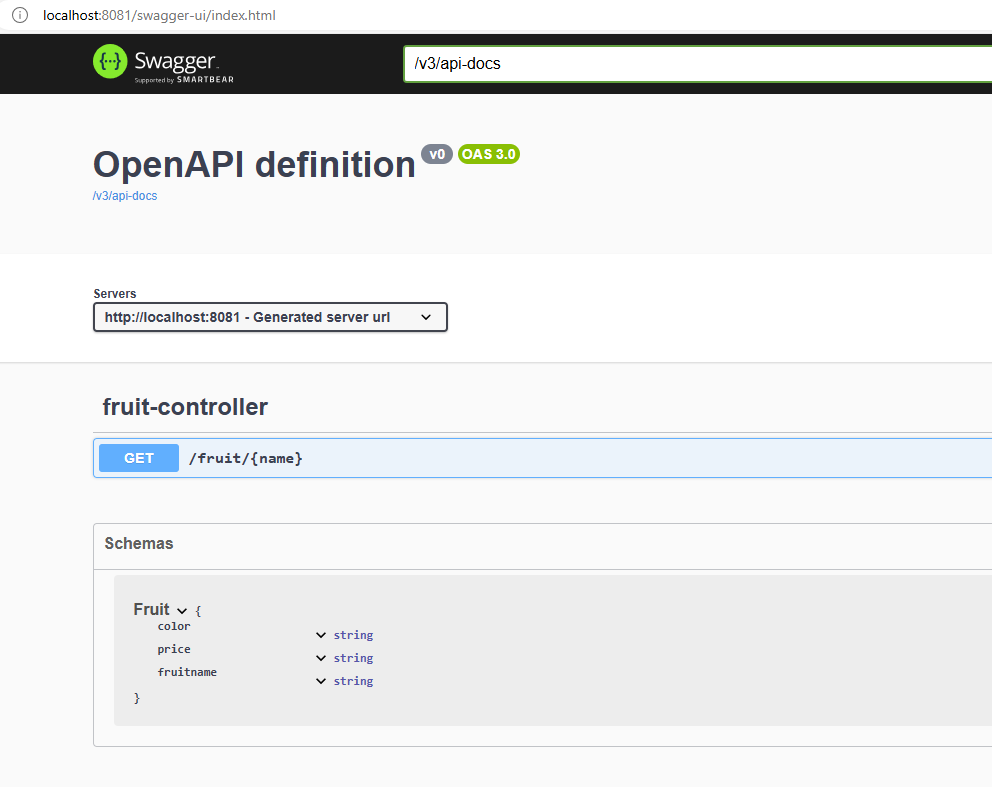
<http://localhost:8081/v3/api-docs>



Swagger is an open-source framework that allows developers to **design, build, document**, and **consume RESTful web services**.

### Launch Swagger UI

<http://localhost:8081/swagger-ui/index.html>



# Update Code SpringAdmin

Spring Boot Admin is a community project to **manage** and **monitor Spring Boot applications**.

It provides a web-based user interface to **view information** about running Spring Boot applications, such as **health status**, **metrics, environment properties**, and more.

It is particularly useful for administrators and developers to monitor the health and performance of their applications.

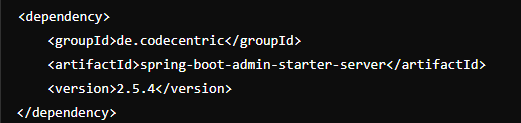
## Key Features of Spring Boot Admin

1. **Dashboard**:
   * Provides an overview of all registered applications.
   * Displays the status of each application.
2. **Application Details**:
   * Health: Shows health indicators such as disk space, database status, etc.
   * Metrics: Displays application metrics like memory usage, CPU usage, HTTP request statistics, etc.
   * Environment: Lists environment properties and configuration properties.
   * Loggers: Allows changing the log level of various packages at runtime.
   * Threads: Displays information about active threads.
   * Trace: Shows HTTP request traces.
   * JMX: Provides access to JMX beans.
3. **Notifications**:
   * Supports notifications for changes in the status of applications via various channels like email, Slack, and others.
4. **Security**:
   * Supports authentication and authorization to restrict access to the Spring Boot Admin UI.

## How to Set Up Spring Boot Admin (Example)

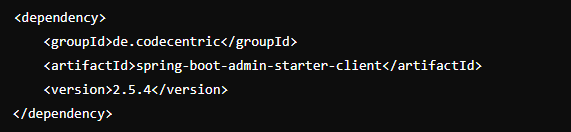
### Set Up Spring Boot Admin Server

* **Add Dependencies**:
  + Add the necessary dependencies to your Spring Boot Admin Server pom.xml



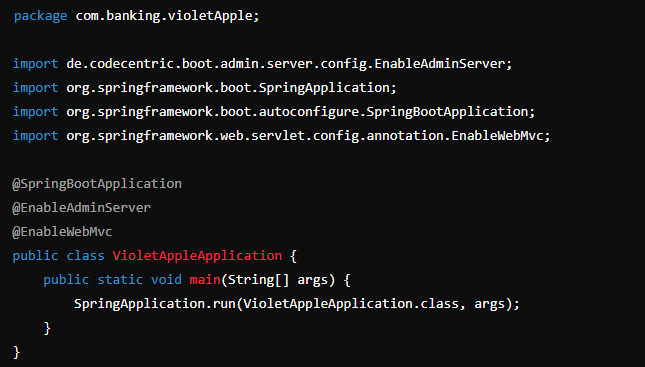
### Set Up Spring Boot Admin Client

* **Add Dependencies:**
  + Add the necessary dependencies to your Spring Boot Admin Client pom.xml:



### Enable Spring Boot Admin Server:

* + Create a **configuration class** to enable Spring Boot Admin Server:

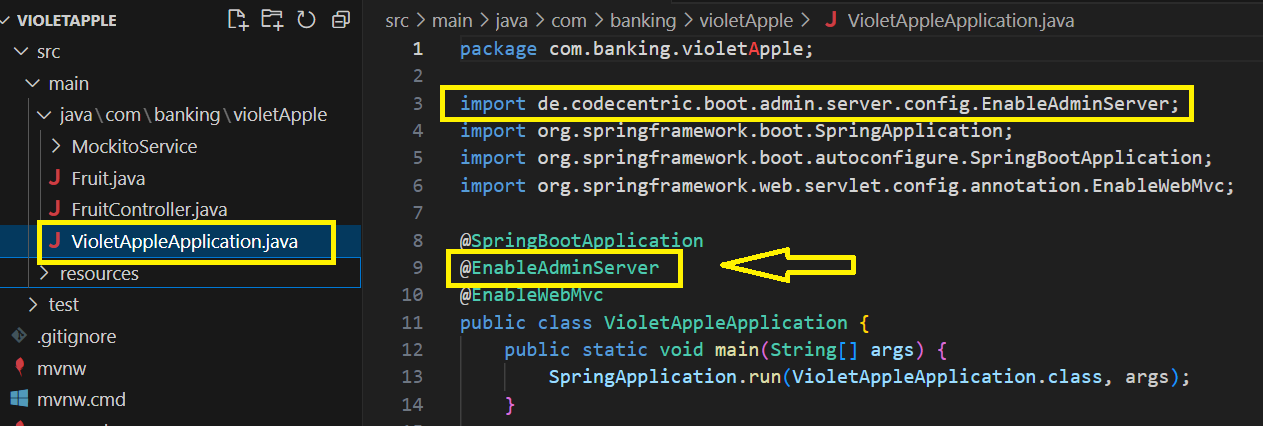


## How to Set Up Spring Boot Admin (Demo)

1. Enable Admin server in VioletAppleApplication.java at Line no #3 and Annotaion @EnableAdminServer at Line #9

import de.codecentric.boot.admin.server.config.EnableAdminServer;

@EnableAdminServer



1. Uncomment below line in **POM.xml**

<dependency>

            <groupId>de.codecentric</groupId>

            <artifactId>spring-boot-admin-starter-client</artifactId>

            <version>3.3.2</version>

        </dependency>

        <!-- https://mvnrepository.com/artifact/de.codecentric/spring-boot-admin-starter-server -->

        <!-- add @EnableAdminServer in main.java -->

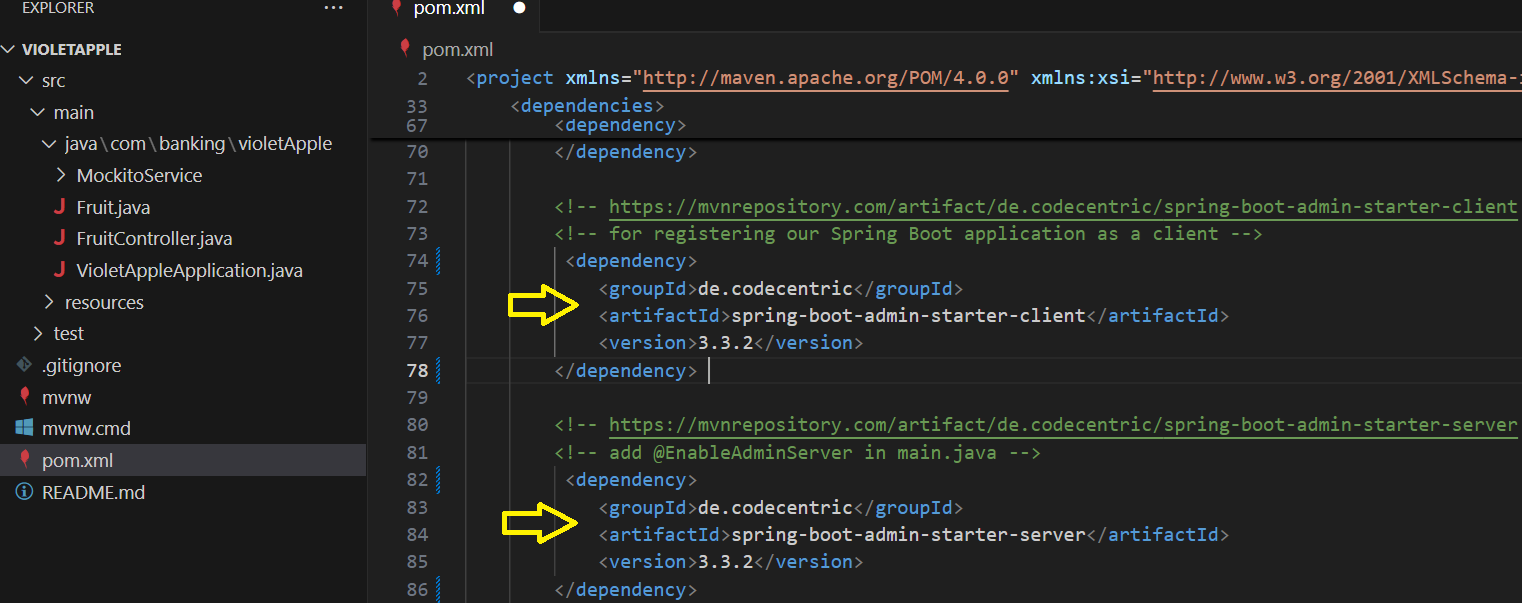
         <dependency>

            <groupId>de.codecentric</groupId>

            <artifactId>spring-boot-admin-starter-server</artifactId>

            <version>3.3.2</version>

        </dependency>



1. Disable Swagger in POM.xml

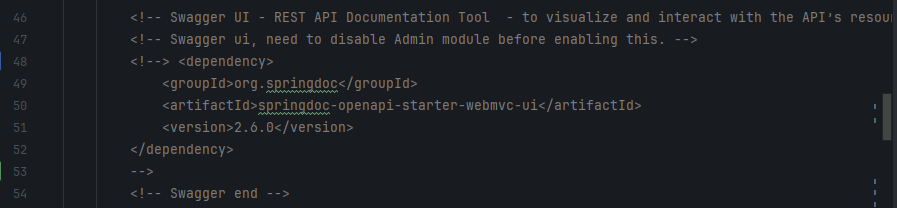
<dependency>

            <groupId>org.springdoc</groupId>

            <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

            <version>2.6.0</version>

        </dependency>



1. If Required Sync the Maven
2. Run the Application
3. Paste the URL in browser

<http://localhost:8080/applications>

