✅ **Case Study Title: Employee Info API using Spring Boot AutoConfiguration**

🎯 **Objective:**

To build a simple Spring Boot application that exposes an API endpoint to retrieve basic employee information using **Spring Boot AutoConfiguration**. The endpoint will be tested via a browser and Postman using only @GetMapping.

🧾 **Background:**

Spring Boot simplifies application setup with its **AutoConfiguration** feature. Instead of manually defining bean configurations, Spring Boot intelligently guesses what you need and configures it behind the scenes.

This case study helps you understand:

- What AutoConfiguration does.
- How to leverage it using minimal configuration.
- How to expose a basic REST endpoint with @GetMapping.

🧱 **Components Involved:**

1. **Spring Boot Starter Web** – Automatically brings in all dependencies for building REST APIs.
2. **AutoConfiguration** – Behind the scenes, it configures the DispatcherServlet, Tomcat server, and other beans automatically.
3. **REST Controller** – A simple Java class using @RestController and @GetMapping.
4. **Browser/Postman** – For testing the GET API.

💡 **Scenario:**

You are a developer working in the HR software team. Your task is to expose employee information (like name, ID, and department) through a simple HTTP GET API without manually configuring any server, servlet, or web.xml file.

🏗️ **Steps in the Case Study:**

1. **Create the Spring Boot Project**
   - Use Spring Initializr (https://start.spring.io)
   - Project metadata:

◦ Group: com.company ◦ Artifact: employee-api

   - Dependencies:
   - Spring Web

2. **Directory Structure AutoCreated by Spring Boot**

Spring Boot automatically generates the following:

```
src/
 main/    java/
   com.company.employeeapi/
     EmployeeApiApplication.java       controller/
      EmployeeController.java    resources/
   application.properties
```

3. **Understanding AutoConfiguration**
   - No need to configure DispatcherServlet, JSON converter, or server port.
   - When you add spring-boot-starter-web, it:
   - Configures embedded Tomcat server.

- Registers Jackson for JSON conversion.
- Sets up DispatcherServlet for handling REST requests.
- Starts server on port 8080.

4. **Creating a Simple GET Endpoint**
   - The @RestController and @GetMapping("/employee") annotations automatically expose a REST endpoint due to AutoConfiguration.
5. **Running the Application**
   - Just run the main class EmployeeApiApplication.java.
   - Spring Boot auto-starts the embedded server and makes the endpoint live.
6. **Testing the API**

Open browser or Postman.
Hit: http://localhost:8080/employee
Expected JSON output:

```
{
  "id": 101,
  "name": "John Doe",   "department": "Engineering"
}
```

EmployeeApiApplication.java:

```
package com.company.employeeapi;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class EmployeeApiApplication {
        public static void main(String[] args) {
                SpringApplication.run(EmployeeApiApplication.class, args);
        }
}
```

EmployeeController.java:

```
package com.company.employeeapi.controller;
import com.company.employeeapi.model.Employee;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class EmployeeController {
   @GetMapping("/employee")
   public Employee getEmployeeInfo() {
      return new Employee(101, "John Doe", "Engineering");
   }
}
```

Employee.java:

```
package com.company.employeeapi.model;
public class Employee {
   private int id;
   private String name;
   private String department;
```

```
    public Employee() {
    }
    public Employee(int id, String name, String department) {
        this.id = id;
        this.name = name;
        this.department = department;
    }
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getDepartment() { return department; }
    public void setDepartment(String department) { this.department = department; }
}
```

Application.properties:

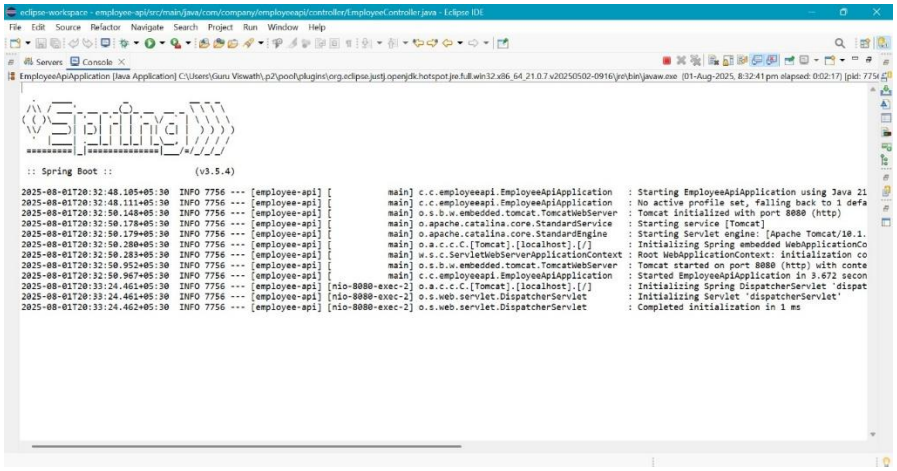
```
spring.application.name=employee-api
server.port=8080
```

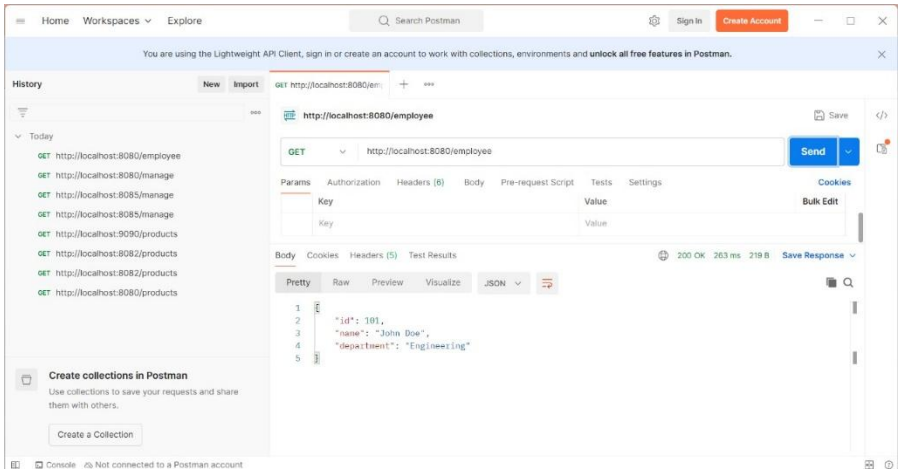
pom.xml:

```
<?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/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>3.5.4</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.company</groupId>
        <artifactId>employee-api</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>employee-api</name>
        <description>Company project for Spring Boot</description>
        <url/>
        <licenses>
                <license/>
        </licenses>
        <developers>
                <developer/>
        </developers>
        <scm>
                <connection/>
                <developerConnection/>
```

```
		<tag/>
		<url/>
	</scm>
	<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>
```

Outputs:





✅ **2. Spring Boot – Actuators**

🎯🎯**Case Study: Monitoring an Inventory System**

Problem Statement:

You deploy an Inventory Management app and want to **monitor** its health, memory usage, bean loading, and environment settings without building these endpoints manually.

💡 **Key Concept:**

Spring Boot **Actuator** exposes production-ready features like health checks, metrics, beans, and custom endpoints.

📝 **Scenario:**

You add the spring-boot-starter-actuator dependency, and enable the / actuator endpoint in application.properties.

With zero code changes, you get:

- /actuator/health → Health of the service.
- /actuator/beans → Beans created in the container.
- /actuator/metrics → JVM and HTTP metrics.
- /actuator/env → Current environment values.

InventoryMonitoringApplication.java:

```
package com.company.inventorymonitoring;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class InventoryMonitoringApplication {
	public static void main(String[] args) {
		SpringApplication.run(InventoryMonitoringApplication.class, args);
	}
}
```

application.properties:

```
spring.application.name=inventory-monitoring
management.endpoints.web.exposure.include=*
server.port=8082
```

InventoryMonitoringApplicationTests.java:

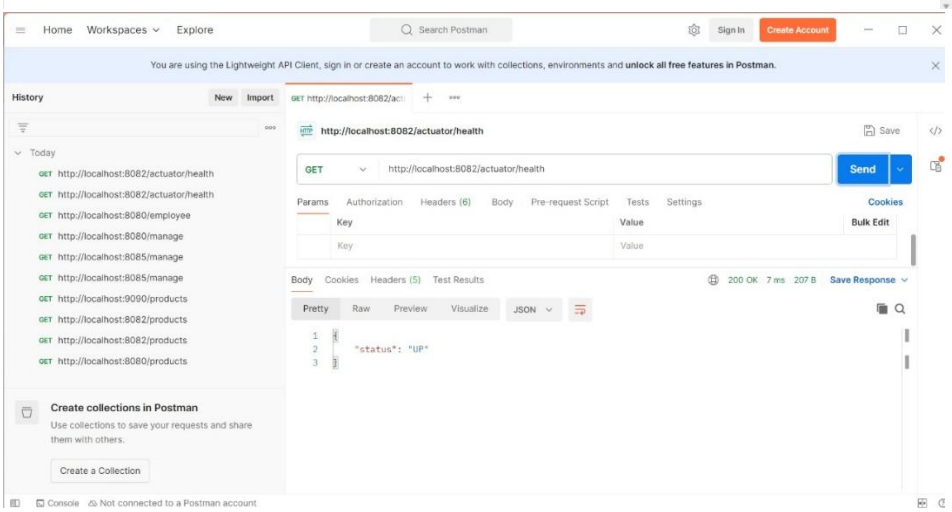
```
package com.company.inventory_monitoring;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
@SpringBootTest
class InventoryMonitoringApplicationTests {
	@Test
	void contextLoads() {
	}
}
```

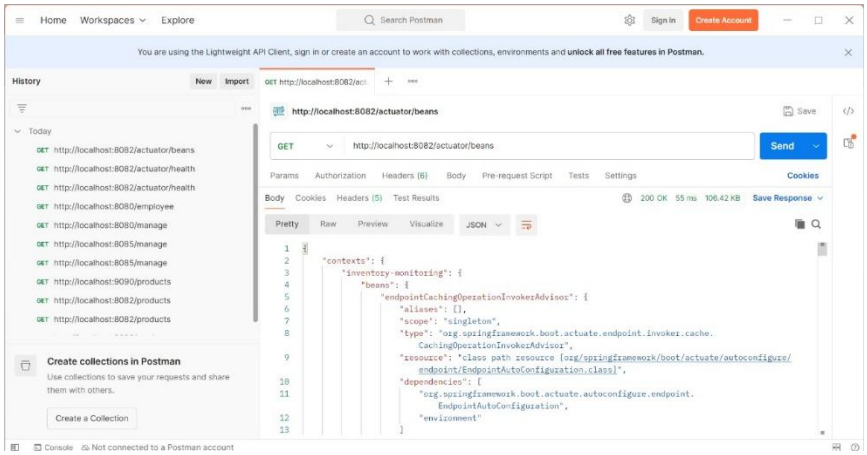
Pom.xml:

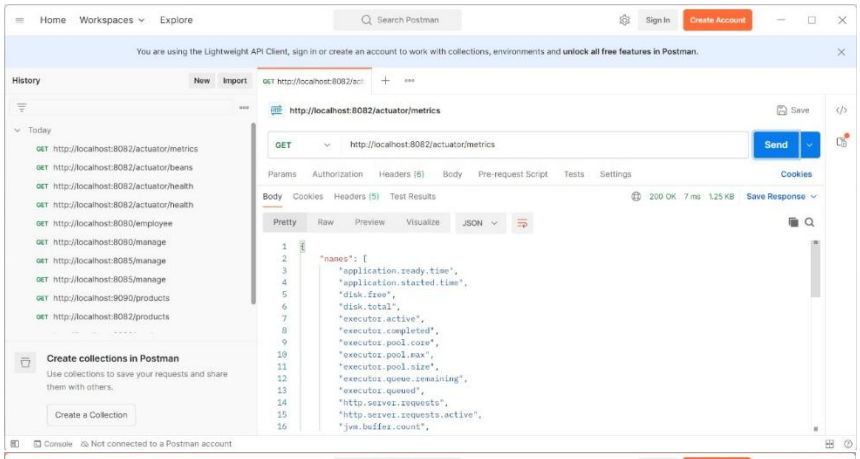
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>3.5.4</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.company</groupId>
	<artifactId>inventory-monitoring</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>inventory-monitoring</name>
	<description>Inventory monitoring project for Spring Boot</description>
	<url/>
	<licenses>
		<license/>
	</licenses>
	<developers>
		<developer/>
	</developers>
	<scm>
		<connection/>
		<developerConnection/>
		<tag/>
		<url/>
	</scm>
	<properties>
		<java.version>21</java.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-actuator</artifactId>
```

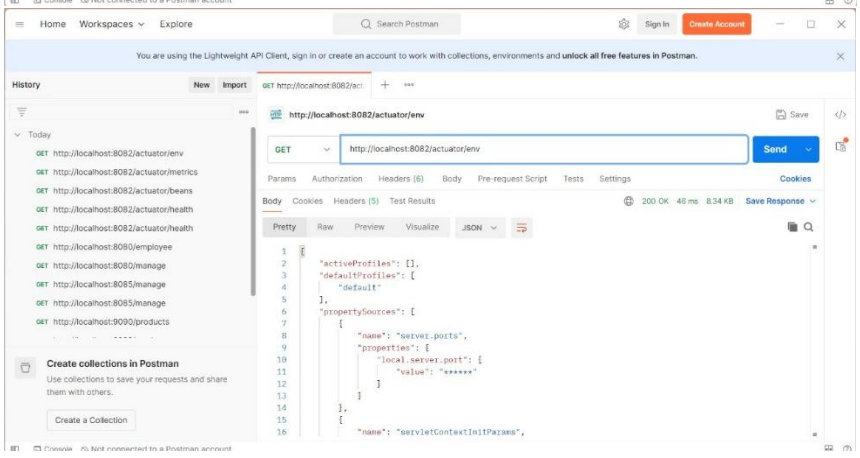
```
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<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>
```

Outputs:




Home
Workspaces
Explore
Search Postman
Sign In
Create Account
You are using the Lightweight API Client, sign in or create an account to work with collections, environments and unlock all free features in Postman.
History
New
Import
http://localhost:8082/actuator/beans
GET
http://localhost:8082/actuator/beans
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies
Headers (5)
Test Results
200 OK
55 ms
106.42 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"contexts": {
"inventory-monitoring": {
"beans": {
"endpointCachingOperationInvokerAdvisor": {
"aliases": [],
"scope": "singleton",
"type": "org.springframework.boot.actuate.endpoint.invoker.cache.
CachingOperationInvokerAdvisor",
"resource": "class path resource [org/springframework/boot/actuate/autoconfigure/
endpoint/EndpointAutoConfiguration.class]",
"dependencies": [
"org.springframework.boot.actuate.autoconfigure.endpoint.
EndpointAutoConfiguration",
"environment"
]
Today
GET http://localhost:8082/actuator/beans
GET http://localhost:8082/actuator/health
GET http://localhost:8082/actuator/health
GET http://localhost:8080/employee
GET http://localhost:8080/manage
GET http://localhost:8085/manage
GET http://localhost:8085/manage
GET http://localhost:9090/products
GET http://localhost:8082/products
GET http://localhost:8082/products
Create collections in Postman
Use collections to save your requests and share them with others.
Create a Collection
Console
Not connected to a Postman account


Home
Workspaces
Explore
Search Postman
Sign In
Create Account
You are using the Lightweight API Client, sign in or create an account to work with collections, environments and unlock all free features in Postman.
History
New
Import
http://localhost:8082/actuator/metrics
GET
http://localhost:8082/actuator/metrics
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies
Headers (5)
Test Results
200 OK
7 ms
1.25 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"names": [
"application.ready.time",
"application.started.time",
"disk.free",
"disk.total",
"executor.active",
"executor.completed",
"executor.pool.core",
"executor.pool.max",
"executor.pool.size",
"executor.queue.remaining",
"executor.queued",
"http.server.requests",
"http.server.requests.active",
"jvm.buffer.count",
Today
GET http://localhost:8082/actuator/metrics
GET http://localhost:8082/actuator/beans
GET http://localhost:8082/actuator/health
GET http://localhost:8082/actuator/health
GET http://localhost:8080/employee
GET http://localhost:8080/manage
GET http://localhost:8085/manage
GET http://localhost:8085/manage
GET http://localhost:9090/products
GET http://localhost:8082/products
Create collections in Postman
Use collections to save your requests and share them with others.
Create a Collection
Console
Not connected to a Postman account


Home
Workspaces
Explore
Search Postman
Sign In
Create Account
You are using the Lightweight API Client, sign in or create an account to work with collections, environments and unlock all free features in Postman.
History
New
Import
http://localhost:8082/actuator/env
GET
http://localhost:8082/actuator/env
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies
Headers (5)
Test Results
200 OK
46 ms
8.34 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"activeProfiles": [],
"defaultProfiles": [
"default"
],
"propertySources": [
{
"name": "server.ports",
"properties": {
"local.server.port": {
"value": "******"
}
}
},
{
"name": "servletContextInitParams",
Today
GET http://localhost:8082/actuator/env
GET http://localhost:8082/actuator/metrics
GET http://localhost:8082/actuator/beans
GET http://localhost:8082/actuator/health
GET http://localhost:8082/actuator/health
GET http://localhost:8080/employee
GET http://localhost:8080/manage
GET http://localhost:8085/manage
GET http://localhost:8085/manage
GET http://localhost:9090/products
Create collections in Postman
Use collections to save your requests and share them with others.
Create a Collection