Stage Technique

Documentation Back-End Avec Spring-Boot

Introduction

JAVA Spring-Boot is an open-source tool that is made to make working with Java-based Frameworks easier for developers to create microservices and Web Apps.

Chapiter 1: Pre-requirements

Before jumping off to start coding out project, it is necessary to check if these conditions are present.

- Solid Java SE fundamentals: Core JAVA syntax, OOP (classes, inheritance, interfaces) …
- HTTP/REST basics: Understand GET/POST/PUT/DELETE, status codes, headers, and Using JSON
- SQL & relational database CRUD: Be able to write simple SELECT/INSERT/UPDATE/DELETE queries and grasp primary/foreign keys

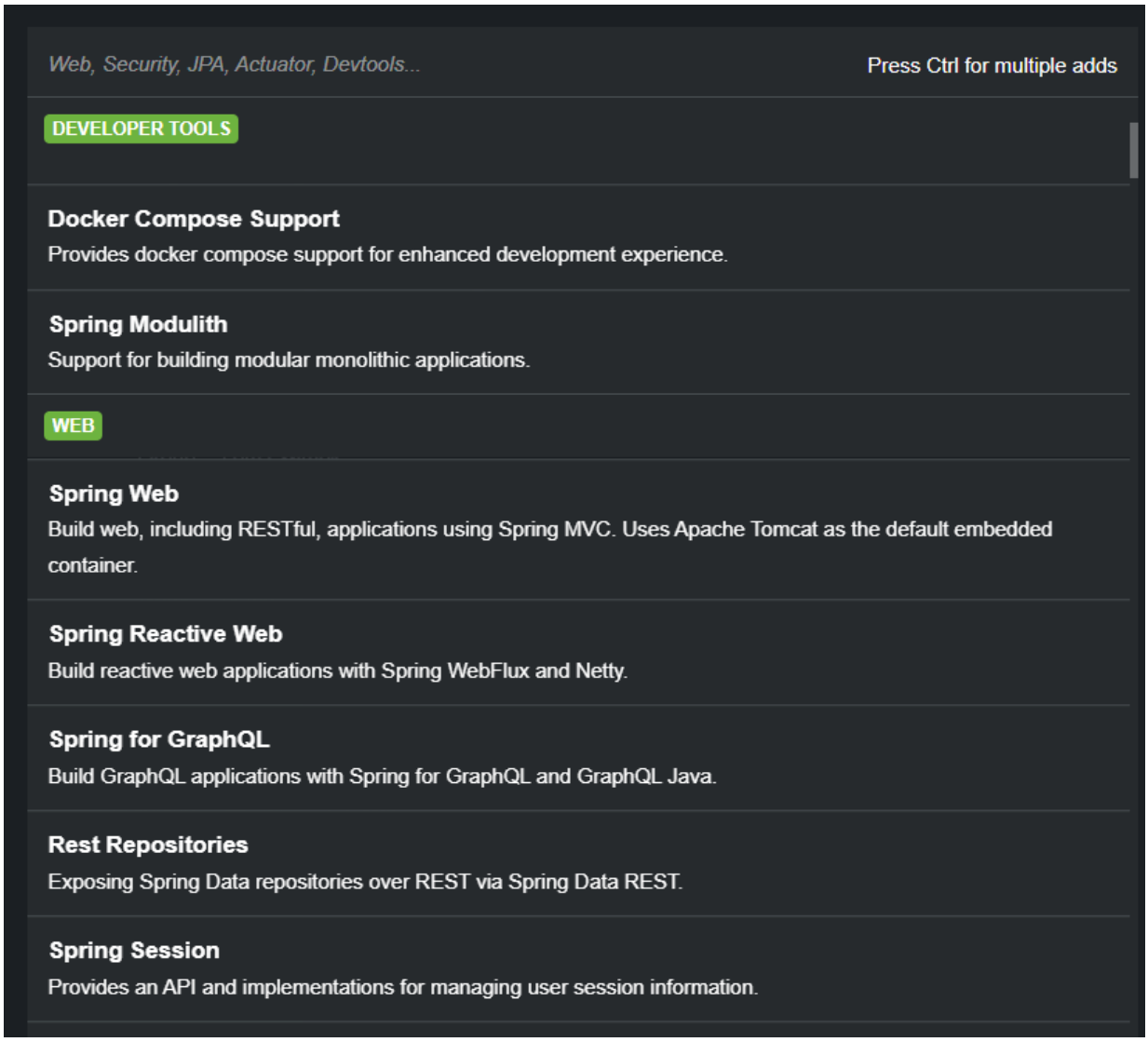
If this knowledge is here we can move on to the verification of the building tools and IDE Setup though we will use MAVEN for dependencies management and STS4 Eclipse Version with the JDK 11 or newer as IDE for Coding

Chapiter 2: Spring-Boot Basics

In this part we will talk about the basics and important steps to take for the Spring-Boot project Kick-off

1. Understanding Application Layers: There is 3 layers, when a user sends a request to the application the request handling layer is the controller who takes the request and sends it to the proper service which is the service layer or the business layer where we can find the business logic which is the application logic and then it sends it to the Model Layer which have the direct connection with the database
2. Maven: Is a build automation tool which serves to auto-download and configure libraries and auto-build and test project
3. Controller & RestController
4. Service & Component
5. Entity: The Entity represents the database table itself in the Model Layer

6. Repository : repository is the tool who makes the direct connection with a certain database (insert, update, select…)
7. Starter Dependencies are pre-configured dependencies (MAVEN ones in our case) that pull in to your project all the Spring and related technologies that you need without having to go through coding loads of dependencies (spring-boot-starter-web, spring-boot-starter-data-jpa, etc…)
8. Spring Initializr that is a web-based tool that generates Spring-Boot project with the chosen starters (below list of pre-configured Maven dependencies for the WEB

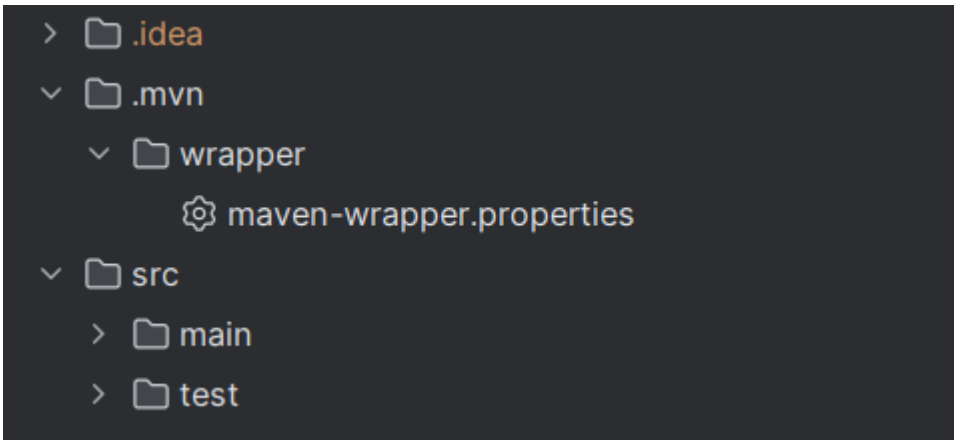


9. Dependency injection (IoC container and beans) : Inversion of Control (IoC) also known as dependency injection is a process whereby objects define their dependencies, that is, the other objects they work with. The Container (Core of Spring Framework that is the manager of objects from creation till destruction) then injects those dependencies when it creates

the bean (A bean is the object that forms the backbone of the application and being managed by Spring IoC Container

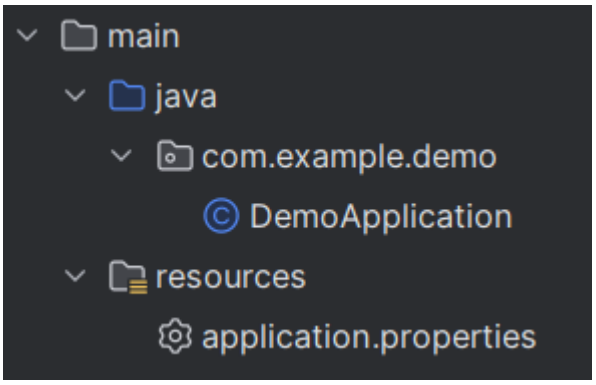
10. Auto-Configuration: Based on what's on your class path and your properties, Spring Boot automatically configures sensible defaults (e.g. Data Source, MVC setup).

Chapiter 3: Structure of Spring boot Project



The main folders that construct a spring boot project are this folder above

- The .idea: is a folder created by the IDE used and not included in the original project, .idea for Intellij IDEA as we can find other names for other IDEs
- The .mvn: is a folder for MAVEN and includes the Maven features as you can use all MVN commands from the IDE without the need for manual installation of it in the local CMD
- The src: is the principale folder where our project will be with its details below



So the main folder includes a JAVA folder which includes the main class which has the main() function in it and this is where all java code lives (controllers, services repos, entities)

The Main/Resources contains the static files and configuration files like application.properties (like .env file in Laravel) where the app configurations are .

The Test/java is where unit and integration tests are written (though spring boot comes with Junit by Default)

Chapiter 4 : Creating Our First REST API

Now, let's build something simple for the exemple and also for better understanding of the REST API Concept and learning how controllers, services, entities and repositories work together

1- Creating an Entity

```
@Entity  10 usages
public class Contact {
    @Id  no usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public Long id;
    public String name;  no usages
    public String email;  no usages

    // Getters and Setters
}
```

2- Create a Repository

```
public interface ContactRepository extends JpaRepository<Contact, Long> {  2 usages
}
```

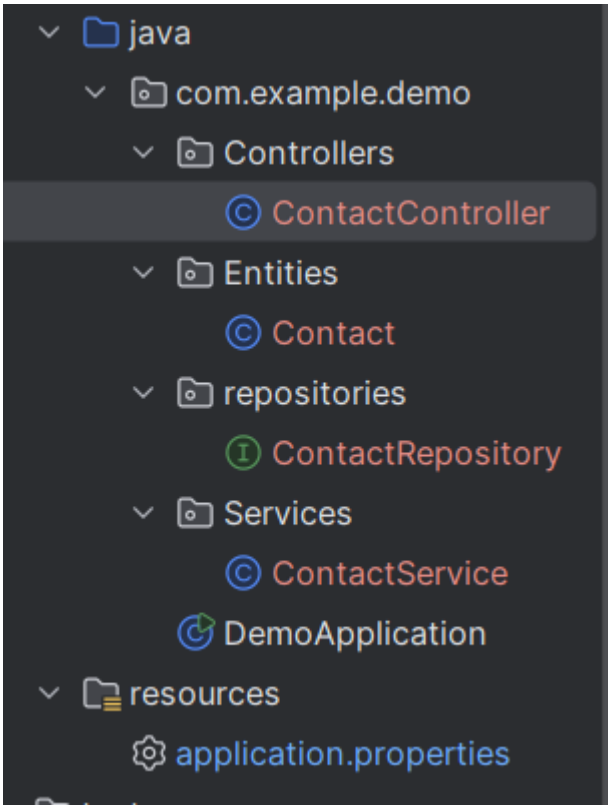
3- Create a Service

```java
@Service  2 usages
public class ContactService {
    @Autowired  2 usages
    private ContactRepository contactRepository;

    public List<Contact> getAllContacts() {  1 usage
        return contactRepository.findAll();
    }

    public Contact addContact(Contact contact) {  1 usage
        return contactRepository.save(contact);
    }
}
```

4- Create a Controller

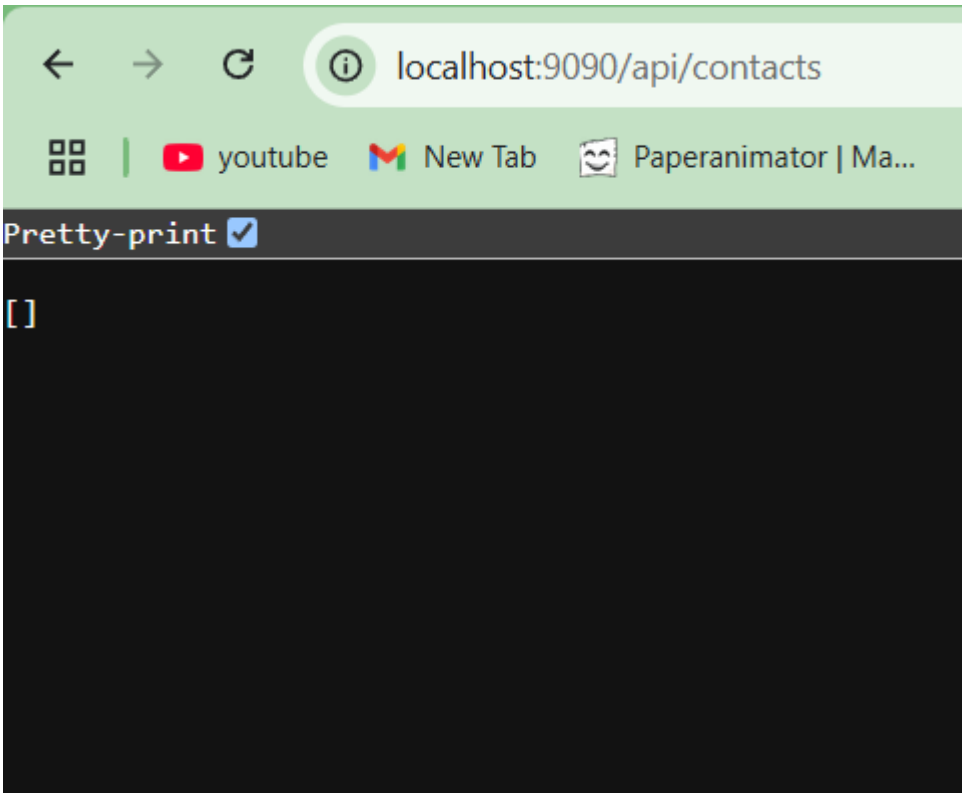
```java
@RestController  no usages
@RequestMapping("/api/contacts")
public class ContactController {
    @Autowired  2 usages
    private ContactService contactService;

    @GetMapping  no usages
    public List<Contact> getAllContacts() {
        return contactService.getAllContacts();
    }

    @PostMapping  no usages
    public Contact addContact(@RequestBody Contact contact) {
        return contactService.addContact(contact);
    }
}
```

5- Project Structure

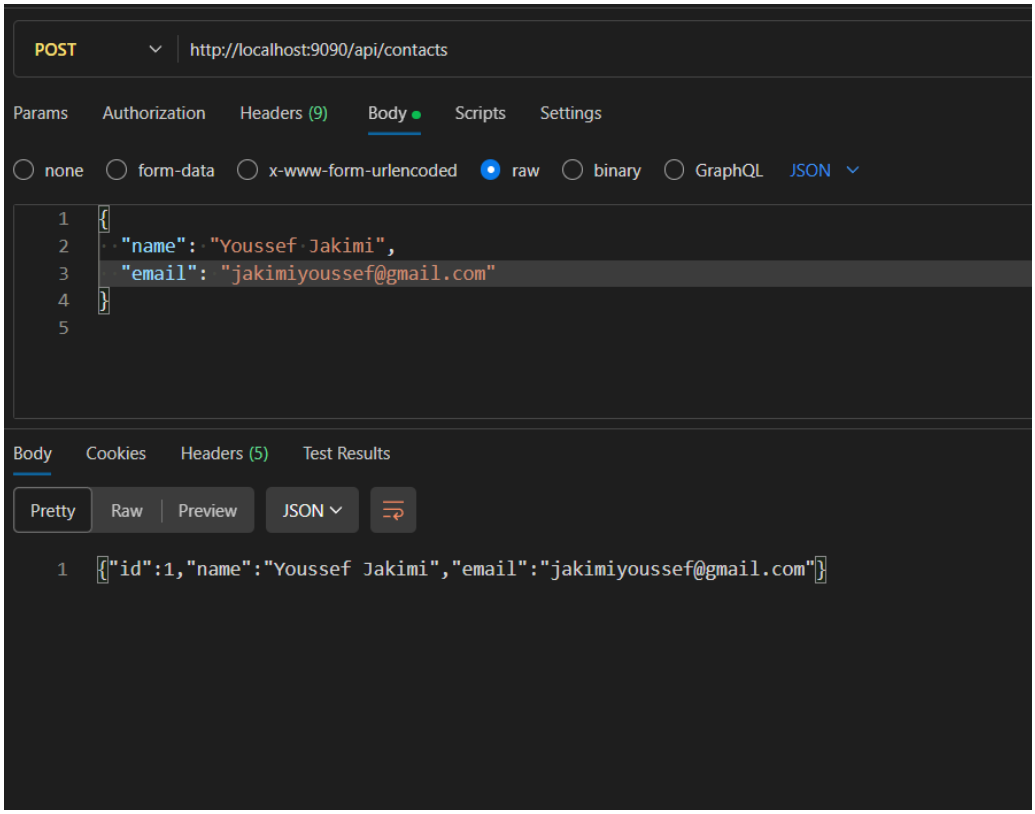


6- Results of the Project

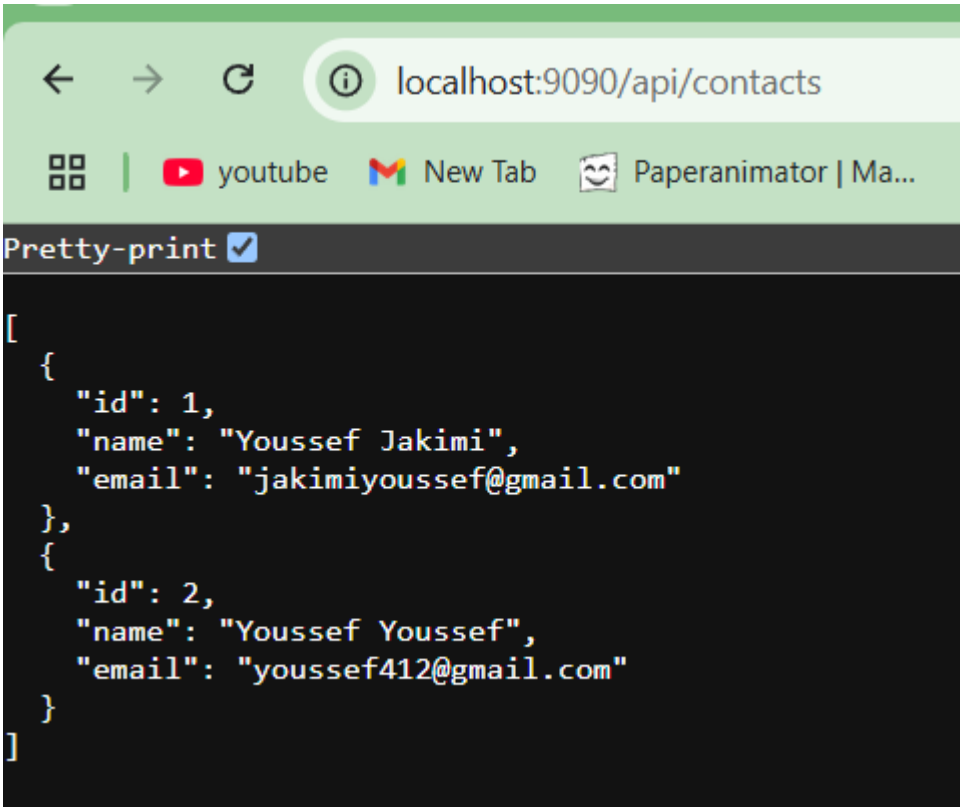
A – Get Request at (/api/contacts)



B – Post Request at (/api/contacts)



C – Another Get Request at (/api/contacts)



Conclusion

In this documentation, we covered the essential basics for getting started with Java Spring Boot development.
We explored the key concepts necessary before starting, such as understanding Java fundamentals and some HTTP/REST.
We also discussed the important building blocks of a Spring Boot application:

- How the Controller, Service, and Repository layers interact.
- How Maven helps manage dependencies.
- The role of Entities to represent database tables.
- The concept of Dependency Injection and how Spring manages your application's objects (beans).
- What Starter Dependencies and Spring Initializr are and how they make project setup faster.

Finally, we learned how a typical Spring Boot project is structured, with important folders like src/main/java and src/main/resources, and how the main application starts from the main() method.
Spring Boot is a powerful and flexible tool, and mastering these fundamentals is a great first step toward building real-world Java applications!