Spring boot

Framework that makes it easy to create Java applications, especially web apps and APIs.

Installation for VSC IDE

Prerequisites

1. JDK 8 or above (java - version)
2. Maven (mvn -v) --> Build tool
3. VSC extensions --> "Spring Boot Extension Pack" and "Extension Pack for Java"

Step-by-step guide

1. Configure Spring Boot Project (https://start.spring.io/)
   - Project Type (Maven)
   - Languaje
   - Spring boot version
   - Project metadata
     - Group
     - Artifact
     - Name
     - Description
     - Package name
     - Packaging (folder type compression) --> Jar
     - Java version --> 22
   - Dependencies --> Add additional Functionalities to the JSB project simplifying the APIRest, DDBBB and sending messages management.

     Functionalities
     - Developer tools
     - Web
     - Template engines
     - Security
     - SQL
     - NoSql

     Functionalities
     - I/O
     - OPS
     - Observability
     - Testing
     - Spring cloud
     - AI

        - Messaging

2. Download the project and import it to VSC.
    - Open as FOLDER. The project will have the following structure:

```
basic my-spring-boot-app
│
├── src
│   ├── main
│   │   ├── java
│   │   │   └── com
│   │   │       └── example
│   │   │           └── demo
│   │   │               └── DemoApplication.java   // Main class
│   │   ├── resources
│   │   │   ├── application.properties             // Configuration file
│   │   │   └── static                             // Static resources (HTML, JS, CSS)
│   │   │   └── templates                          // Thymeleaf or other template engine files
│   │   │   └── application.yml                    // Optional YAML config file (alternative to .properties)
│   ├── test                                       // Unit and integration tests
│       └── java                                   // Test code
│           └── com
│               └── example
│                   └── demo                       // Test packages
│                       └── DemoApplicationTests.java
│
├── pom.xml or build.gradle                        // Dependency and build configuration
└── README.md                                      // Documentation
```

- Summary of Components:
    - **Main class**: Entry point annotated with @SpringBootApplication.
    - **Configuration**: "application.properties" or "application.yml" for settings.
    - **Controllers**: Handle web requests, annotated with @RestController or @Controller.
    - **Models/Entities**: Represent data structures.
    - **Repositories**: For database interaction (if applicable).
    - **Services**: Contain business logic.
    - **Static/Template resources**: HTML, CSS, JS, etc.

- **Tests**: Pre-configured unit and integration tests.

3. Possible actions to implement
    1. Create REST controller
        - Go to the "src/main/java/com/example/demo" folder (replace com.example.demo with your package name)
        - Create a new Java class called "HelloController.java"
        - - Running TSB applications
        - *This will start the TSB application on defalut pot 8080.*

        ```
        mvn spring-boot:run
        ```

        - Test the function
            - On web browser or Postman --> http://localhost:8080/hello

```
##This code defines a very simple Spring Boot controller that listens for HTTP requests and responds with a message.

package com.example.demo (package);

import org.springframework.web.bind.annotation.GetMapping; ## Maps HTTP GET request
import org.springframework.web.bind.annotation.RestController; ## Define a class as a controller that retuns data as JSON

@RestController  ## Annotation whihc indicates that will handle incoming web requests.
public class HelloController {

        @GetMapping("/hello") ## This method will hanlde HTTP GET and send to URL "http://localhost:8080/hello". Then the method will be executed.
        public String sayHello() {
                return "Hello, Spring Boot!";
        }
}
```

3. Possible actions to implement
    1. Other implementations
        1. You can expand this project by adding more endpoints, services, and logic as needed
            - Adding bussines logic --> using models/entities, services and controllers

        - GET (All users/specific one)
        - POST --> Add new user

2. Create a model/entity
    1. In "src/main/java/com/example/demo", create a new package model (you can do this by creating a folder).
    2. Inside the model package, create a class User.java

```java
package com.example.demo.model;

public class User {
        private Long id;
        private String name;
        private String email;

        public User(Long id, String name, String email) {
                this.id = id;
                this.name = name;
                this.email = email;
        }
        // Getters and setters
        public Long getId() {
                return id;
        }
        public void setId(Long id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public String getEmail() {
                return email;
        }
        public void setEmail(String email) {
                this.email = email;
        }
}
```

3. Creates a Service --> Business logic
    - "In src/main/java/com/example/demo", create a new package service.
    - Inside the service package, create a class "UserService.java"

```java
package com.example.demo.service;

import com.example.demo.model.User;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Service
public class UserService {

    private List<User> users = new ArrayList<>();

    public UserService() {
        // Add some dummy users for testing
        users.add(new User(1L, "John Doe", "john@example.com"));
        users.add(new User(2L, "Jane Doe", "jane@example.com"));
    }
    // Get all users
    public List<User> getAllUsers() {
        return users;
    }
    // Get a specific user by ID
    public Optional<User> getUserById(Long id) {
        return users.stream().filter(user ->
user.getId().equals(id)).findFirst();
    }
    // Add a new user
    public void addUser(User user) {
        users.add(user);
    }
}
```

4. Update the controller
    - (In src/main/java/com/example/demo, open the HelloController.java (or create a new UserController.java if you prefer))

```java
package com.example.demo.controller;

import com.example.demo.model.User;
import com.example.demo.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/users")
public class UserController {

    @Autowired
    private UserService userService;

    // GET /users -> Get all users
    @GetMapping
    public List<User> getAllUsers() {
        return userService.getAllUsers();
    }
    // GET /users/{id} -> Get a specific user by ID
    @GetMapping("/{id}")
    public Optional<User> getUserById(@PathVariable Long id) {
        return userService.getUserById(id);
    }
    // POST /users -> Add a new user
    @PostMapping
    public String addUser(@RequestBody User user) {
        userService.addUser(user);
        return "User added successfully!";
    }
}
```

5. Testing
    - GET all users: http://localhost:8080/users
    - GET user by id: http://localhost:8080/users/1
    - POST new user --> Using POSTMAN (http://localhost:8080/users (url) + POST (method) + body (JSON))