**Create Spring Boot Hello World REST API**

Let’s create a spring project.

just type start.spring.io

This will bring up Spring Initializer.

Now, let's go and let's fill up the project information over here.

We're going to use the Maven project, so let's keep Maven project as selected.

We have three kinds of JVM languages Java, Kotlin, and Groovy. We are going to use Java.

So let's keep Java as selected.

Next, we are going to use Spring Boot Version [3.4](https://start.spring.io/)

So let's select Spring Boot [3.4.1](https://start.spring.io/) Milestone one.

So this is the latest version of spring boot as of now. Next, let's fill up the project metadata over here. So let's create a group as com Jpura, you can go any group that you want.

Next, artifact let's give Artifact as spring1.example

Next, let's give project name as spring1 and then description spring1 REST API and then package name is automatically generated based on the group and the artifact and then packaging as a jar and then Java version

Well, remember whenever you choose Spring Boot version three, then you have to choose Java version 17 or later.

Next, let's go ahead and let's select the dependencies.

Now coming to the dependencies, we need to define what dependencies we need for our microservices.

**The very first dependency that we need to add is web.**

Since we decided to build REST services by adding these starter project, my spring boot is going to bring all the dependencies and libraries that I need to build web applications, including REST services,

MVC based applications, and it is also going to use Apache Tomcat as a default embedded container.

So that's a beauty of starter projects inside Spring Boot.

You don't need to define the dependencies manually for all these libraries and dependencies by just selecting one of the starter project, which is spring web. I'm going to get all the dependencies that I need.

**going to go with the H2** database as the next dependency,

H2 database is an internal memory database which you can use inside spring boot web applications.

It does not require any installation of MySQL servers or Oracle. You can use this simple internal memory database, so let me select that.

**I want to add the Spring data JPA** as a dependency inside my Spring boot web application. Using this starter projects, we can store the data into the database,

we can retrieve the data so all the libraries related to database interaction will be available inside the spring data JPA. So let me select this after this spring Data JPA,

**I also want to go with the actuator**. Like I said before, Spring Boot provides inbuilt endpoints to monitor and manage our application, such as like health of our application metrics of the application.

Since we are building microservice and we want to monitor our microservices without writing any external code, I want to use this actuator dependency and the next dependency is dev tools.

So this spring boot dev Tools is a starter project which will improve the productivity of the developer. Like you can see, whenever we have this dependency inside our spring boot application, it is going to provide fast application restarts, live reload and configurations for enhanced development experience.

Once we start our application, we do some code changes like Java changes, our property changes. In such scenarios, we don't have to manually restart the server of spring boot by default.

Internally, this dev tools is going to restart very quickly by reloading only the changes that I have made. So let me select this as well.

**And the next dependency is Lombok**. So using Lombok we can avoid all the setters and getter methods that we need to write inside a pojo. (plain old java object) class. (<https://www.baeldung.com/java-pojo-class>)

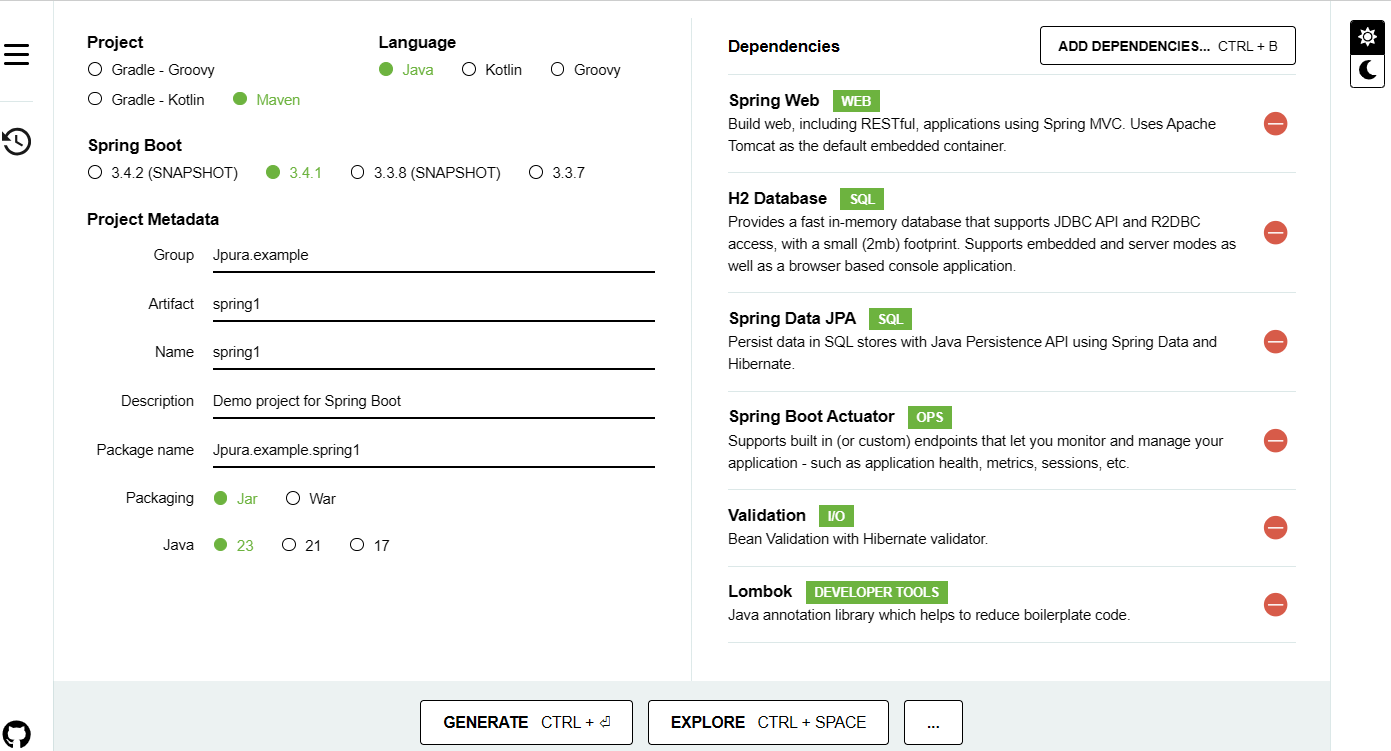
At the end of the day, all the data that we get from UI application or from other applications will get converted into a pojo class and to access the data from the Pojo class, we will use setter methods. So writing these getter and setter methods is very cumbersome and introduce a lot of boilerplate code.

So with the help of Lombok, we can reduce the boilerplate code.

**and the last validation that I want to add here is validation.**

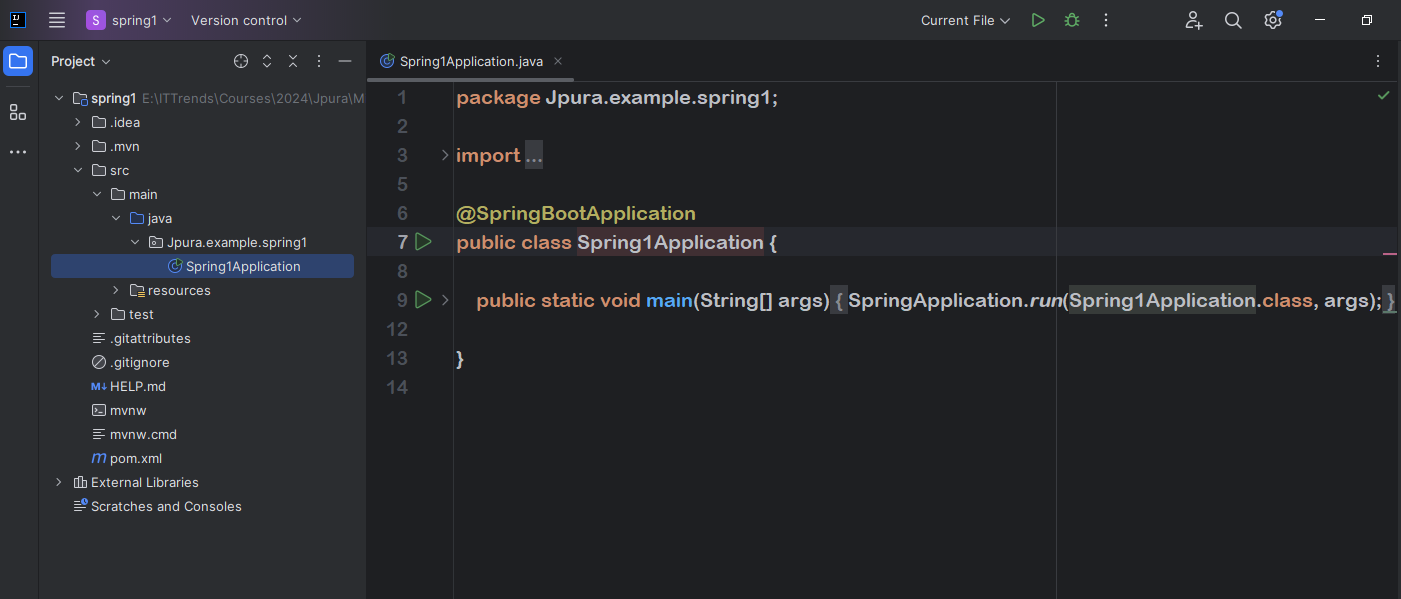
Since we decided to follow the standards of performing validations on the incoming requests, we need to add these starter project inside our spring boot application. So now I have added all the dependencies that I have needed.

Please note that whatever dependency that I have added related to spring boot dev tools, this will work only for local development.



Now as a next step, I can click on this generate button which will generate a Maven application. Unzip the file to a selected folder and observe the Maven project structure.

There is a source folder, there is a pom.xml and other maven related files inside my Maven project. Now I can import this into my IDE so that we can start working on it.



As soon as you open you'll get this pop up, which is Maven build script found. So you need to click on this load.

So this will recognize my accounts project as a Maven project. You can also confirm the same by clicking on this Maven tab.

Now, this will open Spring Boot project in IntelliJ Idea., if you can see the src folder within that, we have the main folder and within the main folder we have Java and resources.

Within the Java folder, we keep all the development-related source code and within a resource folder, we keep all the static resources as well as UI-related resources and all the application-level configuration in application dot properties file.

And here we have a test folder. Within that we keep all the Junit test cases.

Next, we have pom.xml file.

It contains all the Maven Dependencies and Maven plugins.

Now let me go to the source file, and under this source file, we'll be having a class with the name Spring1application.

And this is the main class inside your spring boot application because it has annotation @SpringBootApplication.

With these annotation, we are telling to the spring boot framework to perform auto configurations and to scan all the beans inside our spring boot application.

What is JavaBeans?

JavaBeans is a portable, platform-independent model written in Java Programming Language. Its components are referred to as beans.

In simple terms, JavaBeans are [classes](https://www.edureka.co/blog/java-objects-and-classes/#javaclass) which encapsulate several [objects](https://www.edureka.co/blog/java-object/) into a single object. It helps in accessing these object from multiple places. JavaBeans contains several elements like Constructors, Getter/Setter Methods and much more.

<https://www.edureka.co/blog/what-is-javabeans/#:~:text=Let's%20begin!-,What%20is%20JavaBeans%3F,these%20object%20from%20multiple%20places>

So SpringBootApplication annotation is a combination of all these annotations like enable auto configuration,

@SpringBootConfiguration and @ComponentScan.

So these are very basic to spring and spring boot framework.

**Explain cody as a plugin and usage**

our application does not have any REST APIs, so let's try to build a Hello World Rest API

**Let’s create simple hello word rest api**

In order to create a new basic Rest API, I'm going to create a new package with the name “controller”

Jpura.example.spring1.controller

inside this controller package I'm going to create a new class.

The class name is going to be accountscontroller. Inside this class only we are going to create all the Rest APIs related to the accounts.

Microservice First, let's try to put an annotation on top of this class.

The annotation is @RestController.

Whenever we mention this annotation on top of a class, we are instructing to the spring boot framework I'm going to write methods inside this class with the annotations related to the Http methods. Accordingly, please expose all these methods as a Rest APIs to the outside world.

So that is an indication that we are giving to the spring boot framework.

So here I'm going to create a method. So let me type the method like public.

It is going to return string and the method name is going to be sayhello and it is not going to accept anything. It is simply going to return a string saying that Helloworld.

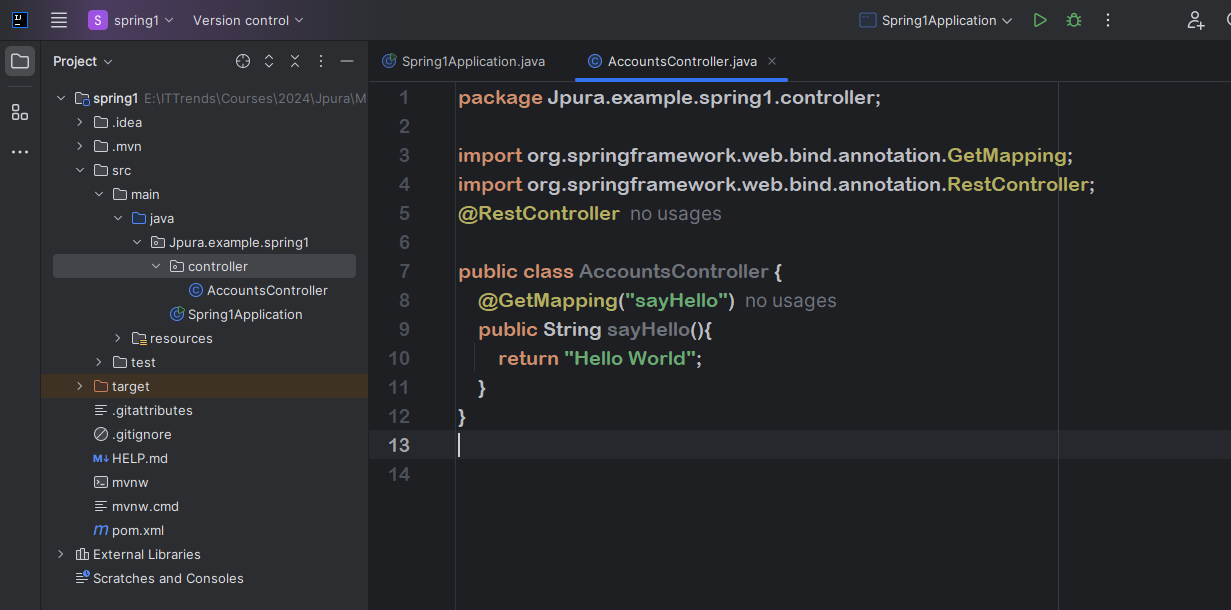
Now, on top of this method, I'm going to mention an annotation which is getmapping.

So whenever we are mentioning get mapping, we are telling to the spring boot framework that this method is going to support Http get method.

So whoever want to invoke my API, they need to invoke with the help of Http get method because this

method is simply going to return the data to the end user or to the client. That's why we should use the Get Mapping.

And since someone want to invoke this API, we should mention what is the path that this method is going to support.



So here I'm just going to define the path saying that sayhello. So if someone is trying to invoke my API, say hello, they'll get this response.

So let me save this changes. Let’s do a build.

When you're trying to do a build very first time, your IDE will ask you to enable annotation processing

because it identifies the Lombok inside your classpath. So please enable the annotations

After this, do build again.

Once the build is completed, I'm straight away going to the spring Boot main class, which is accounts

application. And here I'm trying to start my application in a debug mode.

So let's see the magic that it is going to do behind the scenes. Like you can see here, my accounts application started at the port 8080 and it also identified that

I have H2 database inside my dependency.

That's why it completed all auto configuration related to H2 console. The same applies for the actuator.

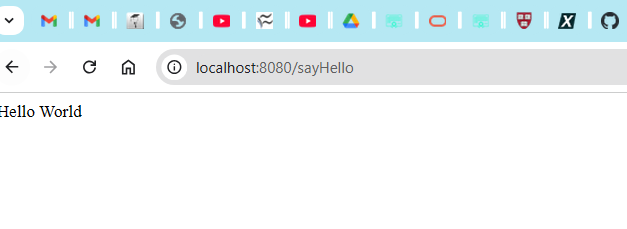
Since we have the actuator related dependency inside our pom.xml spring boot did all the auto configuration and apart from that it is also deployed our web application automatically into your Tomcat server at the port 8080 with the context path, this means my spring boot framework based upon dependencies that I have defined inside my web application.

It is able to do a lot of auto configuration. It is not asking me to provide what is a port number where it has to deploy, what is the server, where it has to deploy, what are the connection details to my H2 database?

It is not asking all those details and it is assuming and doing some default configurations.

Inside my browser, lets type localhost 8080 since our application started at the port 8080.

And the path that we need to invoke is sayhello. As soon as I try to invoke this API path, you will be able to see I'm getting Hello world response from my backend server.



So whenever we try to invoke this path from browser, it is going to invoke with the help of http get method only. That's why I'm getting the response. Hello world.

I hope you are seeing the beauty of the spring boot framework; how easy it is to build a microservice

if I'm aware of all these steps. I can create a micro service template or skeleton within one two minutes, post that I can update with my own business logic.

Try using Dev tools dependency to apply on the fly application changes by doing only the build without restarting the spring application.

By default, you may not have this look and feel in terms of this font and these colors and this dark theme. If you want to have the same theme like I'm showing here, please go to these IntelliJ ideas and open the preferences search under settings

Look for plugins under plugins. Search for dark purple theme from Jetbrains and also one dark theme from Mark Skelton please insall them separately.

Once you install this theme, you can enable these and you can search for theme inside your preferences.

Once you search for the theme.

Under these appearance and behavior, there is an appearance tab. Just click on that.

Under this theme, you should be able to change to whatever theme that you want. Also enable annotations processing because we are going to use a Lombok, Search for annotation processor and enable it. Otherwise you will not be able to use all the Lombok related annotations that we are going to discuss.

