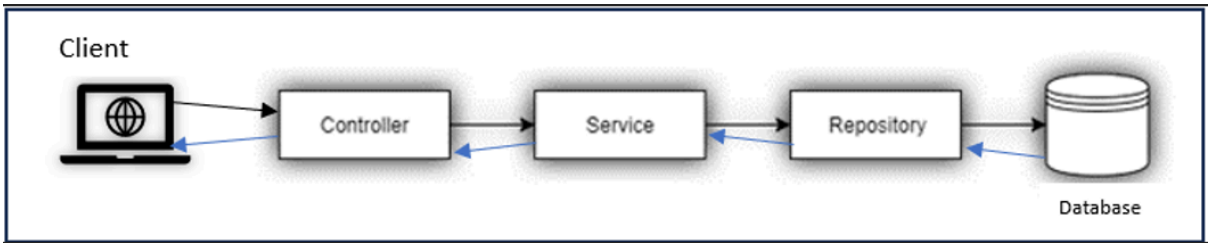
13-Spring Boot Web

- **Client-Server Relationship (CSR) Layer Model**



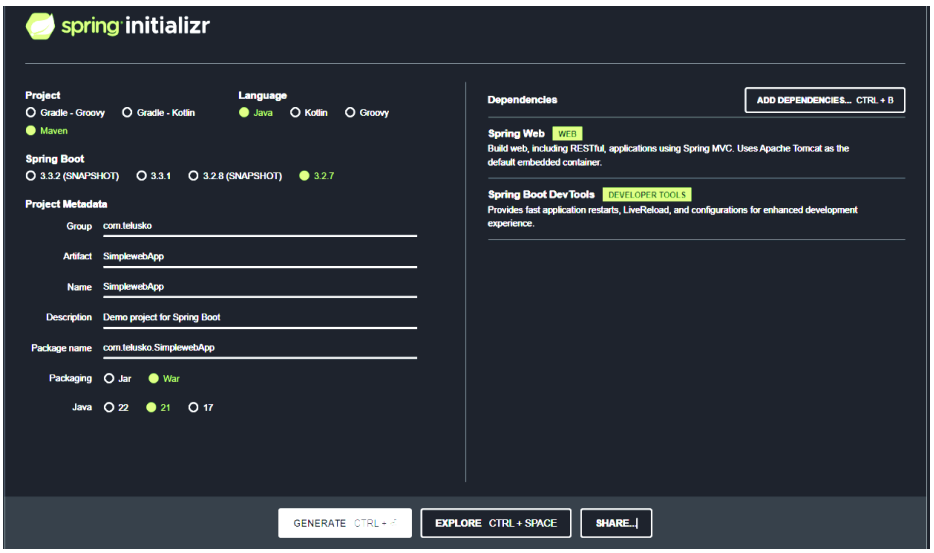
In our previous discussions, we explored the CSR layer model, where the client sends and receives requests over the network using a web container or server through the HTTP protocol. The primary purpose of this model is to share data from the database to the controller and, ultimately, to the client along with the layout using frontend technologies like Angular or React. Without data, the frontend is just a blank HTML page, so data is crucial.

- **Evolution of Data Sharing**

In the past, servers used to send both the layout and data simultaneously, with the layout using HTML and CSS. Initially, XML was used to store and share data over the network. Nowadays, JSON (JavaScript Object Notation) is preferred. JSON is a lightweight, easy-to-understand, and self-explanatory format that stores and shares data in key-value pairs.

- **Creating a New Spring Boot Web Project**

Let's create a new web project using the Spring Initializer with the following configuration:



Dependencies: Spring Web and DevTools (for auto-reloading the server upon changes).

- **Embedded Server**

We will use an embedded server in our project. While there are many popular servers in the enterprise market (like Tomcat, Jetty, and GlassFish), Spring Boot has an embedded Tomcat server running on the default port 8080. This can be modified through configuration.

- **Creating a Controller Layer**

Following our CSR model, let's create a controller layer, which is a Java class responsible for sending and receiving data to the client. We'll name our controller class HomeController and add some methods to it. In a corporate environment, certain rules and regulations for writing syntax must be followed to ensure code readability and maintainability.

- **Using annotations**

We will use annotations as in previous Spring Boot projects:

- **@Controller:** This is a class-level annotation, a specialization of @Component, marking a class as a web request handler. It often serves web pages and returns a string indicating which route to redirect by default.

- **@RequestMapping("/")**: This annotation maps web requests to handler methods. It has many optional elements like consumes, header, method, name, params, path, produces, and value.

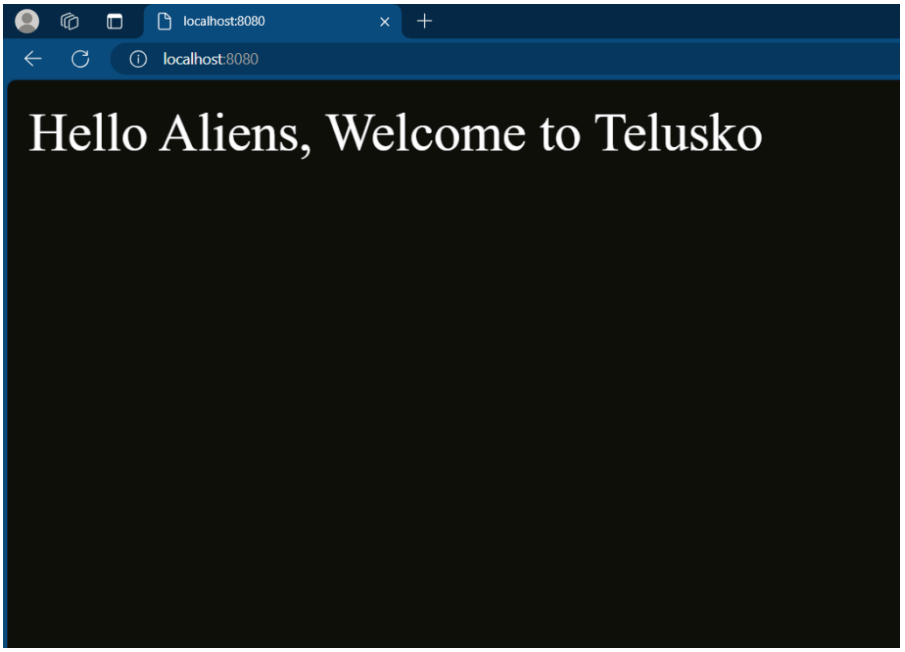
- A combination of @ResponseBody and @Controller is **@RestController,** which returns data from the server to the client, following the REST (Representational State Transfer) API concept.

- **Example Code**

```java
package com.telusko.SimplewebApp;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class Homecontroller {
    @RequestMapping("/")
    @ResponseBody
    public String greet() {
        return "Hello Aliens, Welcome to Telusko";
    }

}
```

Here, we're only sending data, not layout pages. If you want layout pages, we can create those using Thymeleaf or JSP (Java Server Pages). Multiple endpoints can be added to the project as needed.

- **Output:**



- **Service and Repository Layers**

Following our CSR model, we also have service and repository layers, which are Java classes. Multiple controllers can be included in the project. You might wonder how Spring Web or Spring MVC redirects requests to the correct pages. In our CSR model, a front controller is responsible for managing all requests coming from the controller. This front controller is created by Spring itself.

- **Sending data from the server**

Until now, we've focused on sending data from the server to the client. In the next chapter, we'll explore how data is sent from the client side to the server.

Keep Learning and Coding ☺