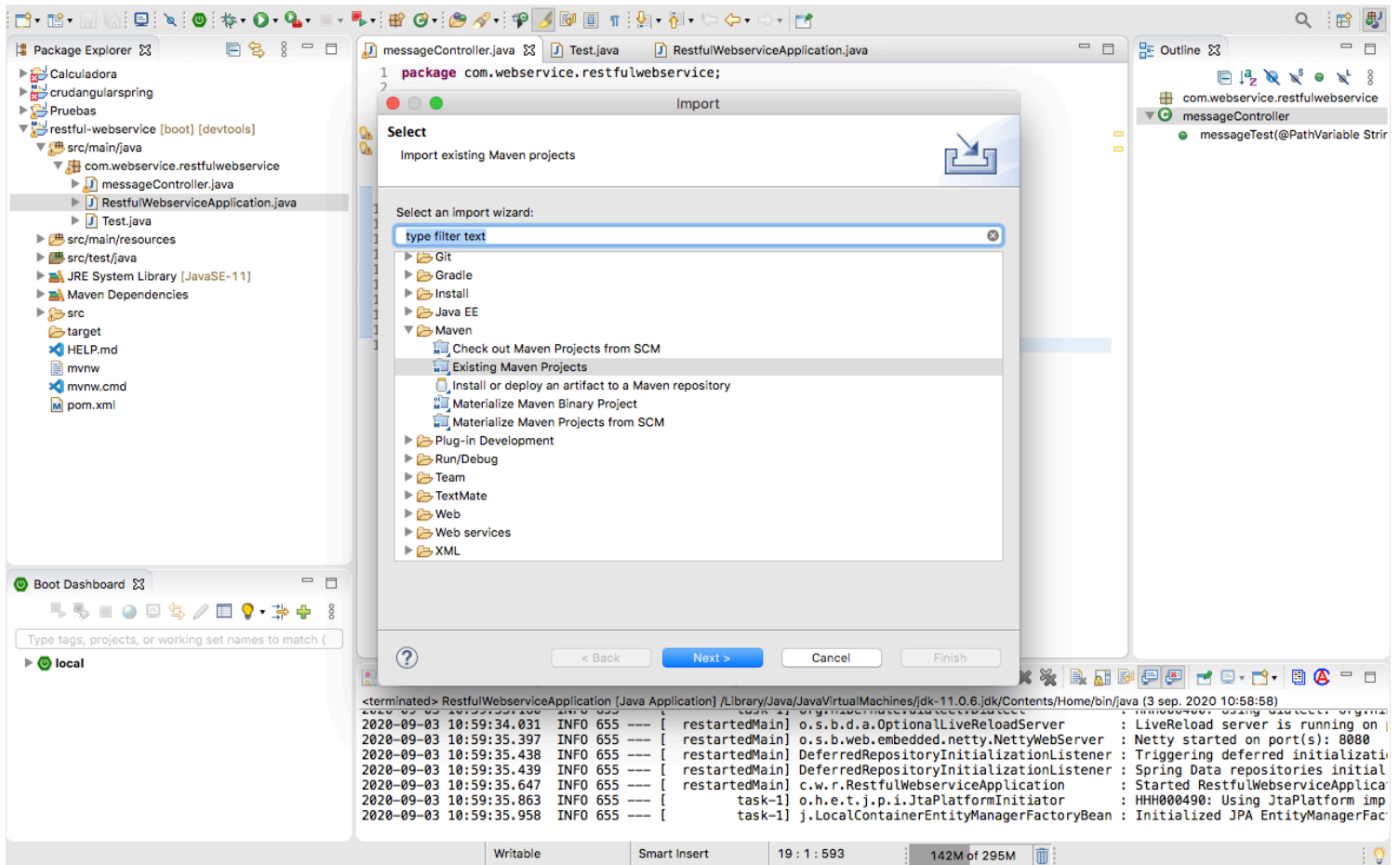


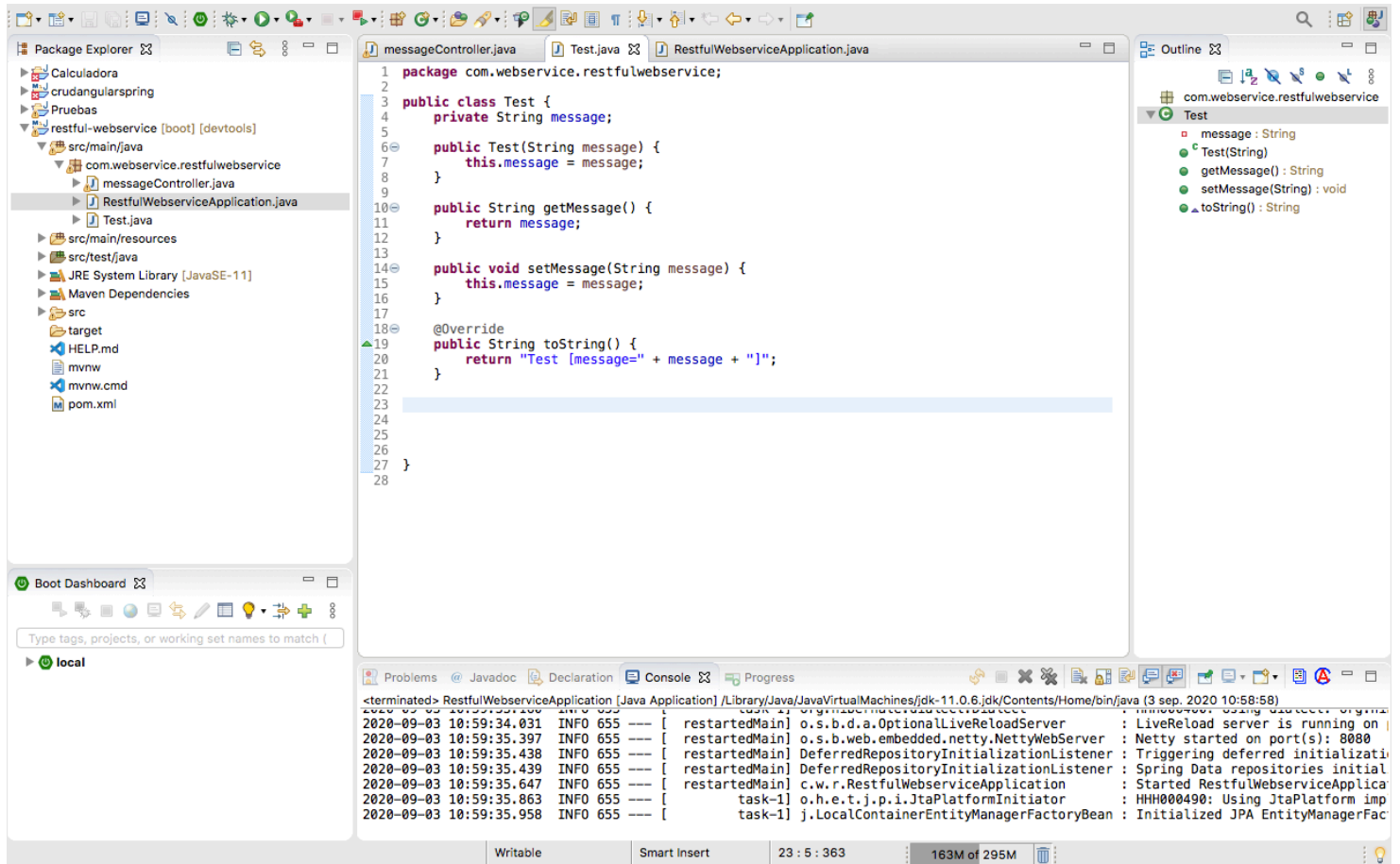
Paso 1.- Hemos seleccionado el lenguaje Java y el framework SpringBoot para poder crear nuestro micro servicio . Para crear nuestro proyecto nos dirigimos a la pagina <https://start.spring.io/>, en donde podremos nos facilitara toda la configuración del proyecto, ahí seleccionamos un nombre para nuestro proyecto, versión y lenguaje que quiéranos ocupar.

The screenshot shows the Spring Initializr web application interface. The browser's address bar displays 'start.spring.io'. The page features the Spring logo and the text 'spring initializr'. The interface is divided into several sections: 'Project' with radio buttons for 'Maven Project' (selected) and 'Gradle Project'; 'Language' with radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'; 'Spring Boot' with radio buttons for various versions, including '2.3.4 (SNAPSHOT)' (selected), '2.3.3', '2.2.10 (SNAPSHOT)', '2.2.9', '2.1.17 (SNAPSHOT)', and '2.1.16'; and 'Project Metadata' with input fields for 'Group' (containing 'com.example') and 'Artifact' (containing 'demo'). A 'Dependencies' section on the right includes a button 'ADD DEPENDENCIES... ⌘ + B' and the text 'No dependency selected'. At the bottom, there are three buttons: 'GENERATE ⌘ + ↵', 'EXPLORE CTRL + SPACE', and 'SHARE...'. The browser's taskbar at the top shows several open applications, including 'Aplicaciones', 'Trello', 'Go', 'Hand', 'Demos', 'Idiomas', 'https://colab.rese...', 'How to One Hot E...', 'Conceptos funda...', and 'Data Science for...'.

Paso 2.- ahora procedemos ingresar a nuestro IDE, en este caso Eclipse, nos dirigimos a archivo, importar y ahí seleccionamos “proyecto existente de maven”, seleccionamos nuestro proyecto y damos aceptar.



Paso 3.- Procedemos a crear una clase llamada Test con un constructor , getters y setters, en donde podremos asignarle un mensaje a nuestra llamada API, como podemos ver en nuestra imagen.



Paso 4.- procedemos a crear una clase llamada **messageController**, dentro de ella crearemos un método de tipo Test llamado **messageTest**, asignándole un endpoint por medio de la ayuda de la etiqueta **@GetMapping**, el cual devolverá un mensaje.

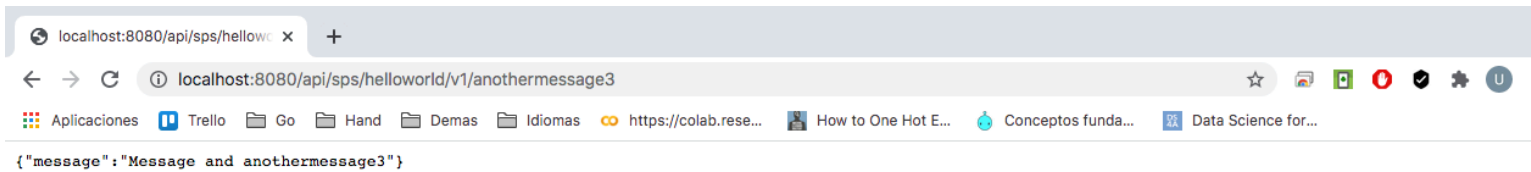
The screenshot displays an IDE interface with the following components:

- Package Explorer (Left):** Shows the project structure. The package `com.webservice.restfulwebservice` is expanded, showing `messageController.java` and `Test.java`.
- Editor (Center):** Displays the code for `messageController.java`. The code is as follows:

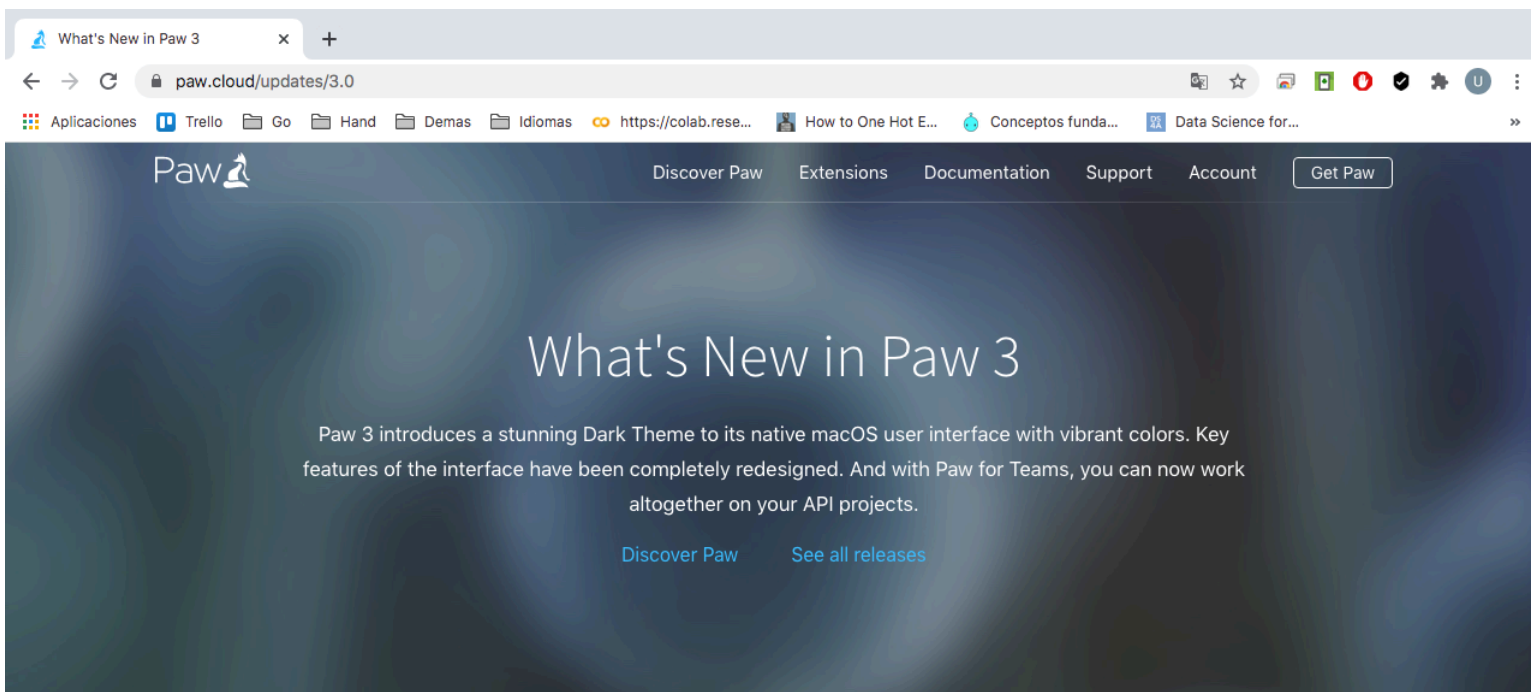
```
1 package com.webservice.restfulwebservice;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.PathVariable;
5 import org.springframework.web.bind.annotation.RequestMapping;
6 import org.springframework.web.bind.annotation.RequestMethod;
7 import org.springframework.web.bind.annotation.RestController;
8
9 @RestController
10 public class messageController {
11
12
13     @GetMapping(path="/api/sps/helloworld/v1/{newmessage}")
14     public Test messageTest(@PathVariable String newmessage) {
15         return new Test(String.format("Message and %s", newmessage));
16     }
17 }
18
19 |
```
- Outline (Right):** Shows the class `messageController` and its method `messageTest(@PathVariable Str`.
- Boot Dashboard (Bottom Left):** Shows a search bar and a list of local projects.
- Console (Bottom):** Displays the output of the application. The log shows the application starting successfully on port 8080.

```
<terminated> RestfulWebserviceApplication [Java Application] /Library/Java/JavaVirtualMachines/jdk-11.0.6.jdk/Contents/Home/bin/java (3 sep. 2020 10:58:58)
2020-09-03 10:59:34.031 INFO 655 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on
2020-09-03 10:59:35.397 INFO 655 --- [ restartedMain] o.s.b.web.embedded.netty.NettyWebServer : Netty started on port(s): 8080
2020-09-03 10:59:35.438 INFO 655 --- [ restartedMain] DeferredRepositoryInitializationListener : Triggering deferred initializat
2020-09-03 10:59:35.439 INFO 655 --- [ restartedMain] DeferredRepositoryInitializationListener : Spring Data repositories initia
2020-09-03 10:59:35.647 INFO 655 --- [ restartedMain] c.w.r.RestfulWebserviceApplication : Started RestfulWebserviceApplic
2020-09-03 10:59:35.863 INFO 655 --- [ task-1] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform imp
2020-09-03 10:59:35.958 INFO 655 --- [ task-1] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFa
```

Paso 5.- Corremos nuestra aplicación y nos dirigimos a nuestro navegador, ingresamos la dirección de nuestro endpoint y verificamos que funciona.



Paso 6.- Ingresamos a la pagina Paw.cloud para hacer uso de una herramienta de prueba de nuestra API, descargamos la app.



Paw for Teams

<https://paw.cloud/updates>

Keep everyone in sync with Paw for Teams. Create a team, invite your team and everyone gets

Paso 7.- Con la app ya descargada y abierta pegamos la dirección de nuestro endpoint y verificamos si funciona. Como se puede ver nos devuelve un mensaje 200, diciendo que todo funciona.

