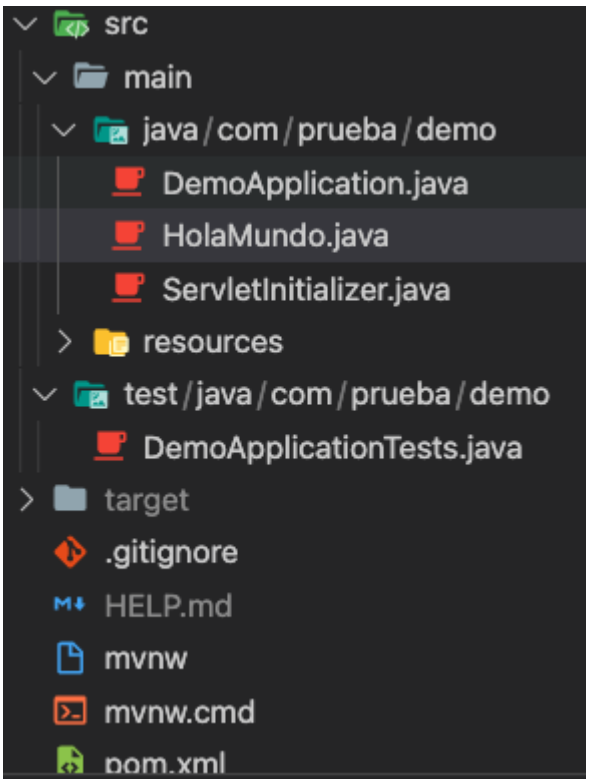
Proyecto colaborativo en servidor Parte II

NOTA: El proyecto **completo** está subido al repositorio siguiente de github:
https://github.com/kant003/transformaloTu.git

Añadiendo funcionalidades a nuestra aplicación

Por ahora nuestra aplicación web no hace nada, vamos a añadir un poco de funcionalidad.

Crea una carpeta llamada HolaMundo.java en la ruta que ves en la imagen



Añádele el siguiente contenido

```java
package com.prueba.demo;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HolaMundo {
    @RequestMapping("/")
    public String saludar(){
        return "Esta es tu primera página web backend";
    }

    @RequestMapping("/despidete")
    public String despidete(){
        return "Adios amigo";
    }
}
```

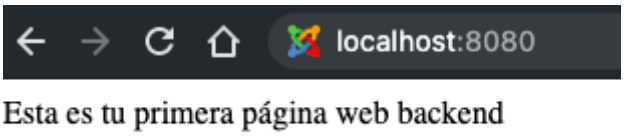
En la terminal para (stop) el servidor usando CTRL+C y vuelve a arrancarlo con el comando

Linux y mac:
./mvnw spring-boot:run

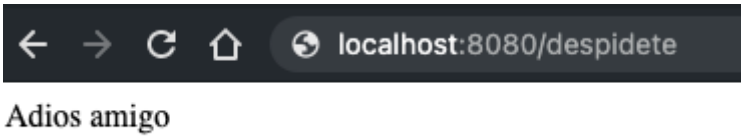
Windows:
mvnw.cmd spring-boot:run

Ahora si accedes a la rutas:

http://localhost:8080/



http://localhost:8080/despidete



Subiendo los cambios al servidor Heroku

Para que los cambios se vean reflejados en el servidor real Heroku realiza los siguientes pasos:

Vuelve a **comitear** los cambios a github (push)
Y vuelve a **deployar** la app a heroku (como ya hemos visto antes)

Comunicación del cliente (frontend) con el servidor (backend)

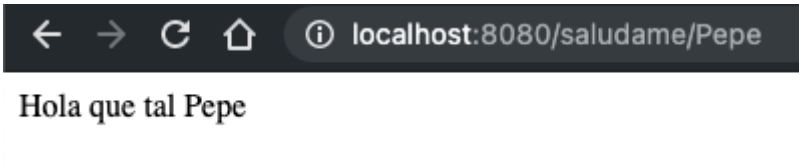
Enviando un parámetro por la url

Añadimos el siguiente fragmento de código al fichero SaludaController.java

```
@GetMapping("/saludame/{nombre}")
   public String saludame(@PathVariable String nombre){
       return "Hola que tal " + nombre;
   }
```

con esto estamos permitiendo que el cliente indique un nombre en la url de su navegador, que será enviado al servidor.

http://localhost:8080/saludame/Pepe



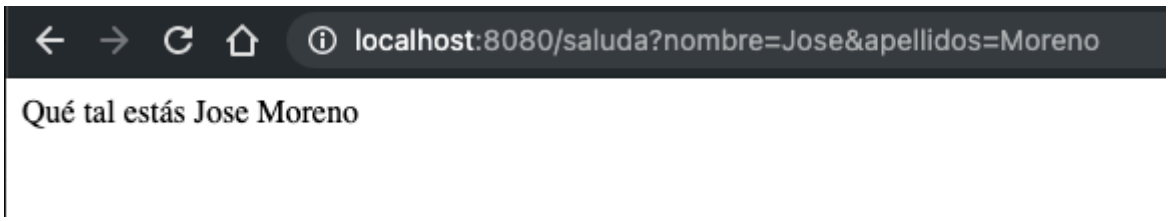
Enviando querys por la url

Añadimos el siguiente fragmento de código al fichero SaludaController.java

```
   @GetMapping("/saluda")
   public String SaludaPorQuery(@RequestParam String nombre,@RequestParam String
apellidos){
       //return "Hola que tal estás" + nombre + " " + apellidos;
       Object params[] = {nombre, apellidos};
       return MessageFormat.format("Qué tal estás {0} {1}", params);
   }
```

ahora el cliente podrá enviar información (del estilo clave/valor) usando la url de su navegador.

http://localhost:8080/saluda?nombre=Jose&apellidos=Moreno



Enviando peticiones REST

Por ahora hemos enviado peticiones de tipo GET, pero existen otros tipos de peticiones que podemos usar.

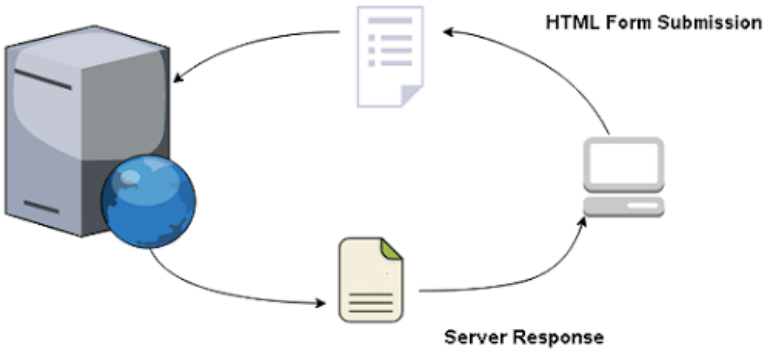
GET → la usamos cuando pedimos información al servidor (se usan cuando escribo una ruta en el navegador)
POST → la usamos cuando enviamos información al servidor, para crear algo (se usa en formularios)
PUT → la usamos cuando queremos modificar algo
DELETE → la usamos cuando queremos borrar algo
PATCH → la usamos cuando queremos modificar pero solo una parte de algo
… existen otras pero menos usadas...

Envío una petición por POST

Añadimos el siguiente fragmento de código al fichero SaludaController.java

```
@PostMapping("/guardar")
   public String Guardar( @RequestParam Map<String, String> body) {
       // Normalmente aquí guardamos algo en el sistema
       return "Datos enviados: " + body.get("nombre") + " y la edad: " + body.get("edad");
   }
```

ahora el cliente podrá enviar información (del estilo clave/valor) usando un formulario (que se ejecutará desde el cliente).
Con el anterior fragmento de código el cliente nos estaría indicando que desea añadir algo del sistema. Por ejemplo un registro a la base de datos, un fichero al disco duro del ordenador, ...



Este es el formulario (en html) que usaríamos para comunicarnos con el backend

```
<!DOCTYPE html>
<html lang="en">
   <html xmlns:th="http://www.thymeleaf.org" lang="en">

<head>
   <meta charset="UTF-8">
   <title>Document</title>
</head>
```

```
<body>
   <form action="http://localhost:8080/guardar" method="post">
       <input type="text" name="nombre">
       <input type="text" name="edad">
       <input type="submit" value="enviar">
   </form>
</body>
</html>
```

Envío una petición por DELETE

Añadimos el siguiente fragmento de código al fichero SaludaController.java

```
@DeleteMapping("/borrar/{id}")
   public String borrar (@PathVariable String id){
       // Aquí normalmente borramos algo en el sistema
       return null;
   }
```

Con el anterior fragmento de código el cliente nos estaría indicando que desea borrar algo del sistema. Por ejemplo un registro de una base de datos, un fichero del ordenador, ...

Uso de servicios

Como norma general (lo hacen todos los frameworks) cuando nos vamos a conectar con algo externo a nuestra aplicación (por ejemplo una base de datos, o un API, o nuestro disco duro, …) es recomendable que lo hagamos en un servicio, así nuestro código estará más ordenado.

Conexión a un API externo

Vamos a obtener una foto random de una persona usando un API gratuito.
Para ello creamos un servicio que se encargará de conectarse a ese api `(randomuser.me)` y realizar la petición para obtener la fotografía de la persona.

Creamos el fichero FotoService.java con el siguiente contenido.

```
package com.cebem.transformalotu.services;


import java.util.ArrayList;
```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

@Service
public class FotoService {

   @Autowired
   RestTemplate restTemplate;

   public String getFoto(){
       String url = "https://randomuser.me/api/";
       DatosPersona json = restTemplate.getForObject(url, DatosPersona.class );
       return json.results.get(0).picture.large;
   }
}
```

Por simplicidad en el código vamos a usar restTemplate para hacer la petición al API (existen mecanismos más modernos pero un poco más complicados de usar)
La petición nos va a devolver un fichero en formato JSON con la información de una persona aleatoria


```json
{
 "results": [
  {
   "gender": "male",
   "name": {
    "title": "Mr",
    "first": "Lawrence",
    "last": "Reynolds"
   },
   "location": {
    "street": {
     "number": 8402,
     "name": "Eason Rd"
    },
    "city": "Wagga Wagga",
    "state": "Victoria",
    "country": "Australia",
    "postcode": 6260,
    "coordinates": {
     "latitude": "-85.5425",
     "longitude": "-101.3515"
    },
    "timezone": {
     "offset": "-3:30",
     "description": "Newfoundland"
    }
   },
   "email": "lawrence.reynolds@example.com",
   "login": {
    "uuid": "9b5d3ed6-8bab-4463-9795-f96b87b8c81b",
    "username": "organicgoose953",
    "password": "hollywood",
    "salt": "GQMQrzne",
```

```
      "md5": "e688d76df674cf7abb69c737345ec3bf",
      "sha1": "ca8f98a732f51d612e3e5163c5aa37d71a8beb12",
      "sha256": "d72a164d1fc68814d6ea4309270882db23abdad7d8c9cdf2f86550613b584449"
     },
     "dob": {
      "date": "1975-10-09T22:15:07.704Z",
      "age": 46
     },
     "registered": {
      "date": "2009-07-27T00:16:30.890Z",
      "age": 12
     },
     "phone": "05-4757-5064",
     "cell": "0455-808-390",
     "id": {
      "name": "TFN",
      "value": "366812871"
     },
     "picture": {
      "large": "https://randomuser.me/api/portraits/men/72.jpg",
      "medium": "https://randomuser.me/api/portraits/med/men/72.jpg",
      "thumbnail": "https://randomuser.me/api/portraits/thumb/men/72.jpg"
     },
     "nat": "AU"
    }
   ],
   "info": {
    "seed": "946e2ead39d09417",
    "results": 1,
    "page": 1,
    "version": "1.3"
   }
}
```


Lo que realmente necesitamos de toda esta información es la foto que se accede a ella a través e results → picture → large

Para poder trabajar con estos datos, necesitamos mapear el objeto JSON a un objeto/clase de tipo Java. Por ello es necesario crear las siguiente clases java

```java
class Picture{
    public String large;
    public String medium;
}
class Result{
    public String gender;
    public String email;
    public String phone;
    public Picture picture;
}
class DatosPersona{
    public ArrayList<Result> results = new ArrayList<>();
}
```

Un último detalle a tener en cuenta para poder usar resTemplate es añador el siguiente bean al fichero TransformalotuApplication.java

```
    @Bean
    public RestTemplate getresttemplate(){
        return new RestTemplate();
    }

    public static void main(String[] args) {
        SpringApplication.run(TransformalotuApplication.class, args);
    }
```

Una vez el servicio está terminado, ya podemos añadir la funcionalidad al controlador. Este se conectará al servicio, obtendrá una foto y se la devolverá al cliente.

```
@Autowired
FotoService fotoService;
```

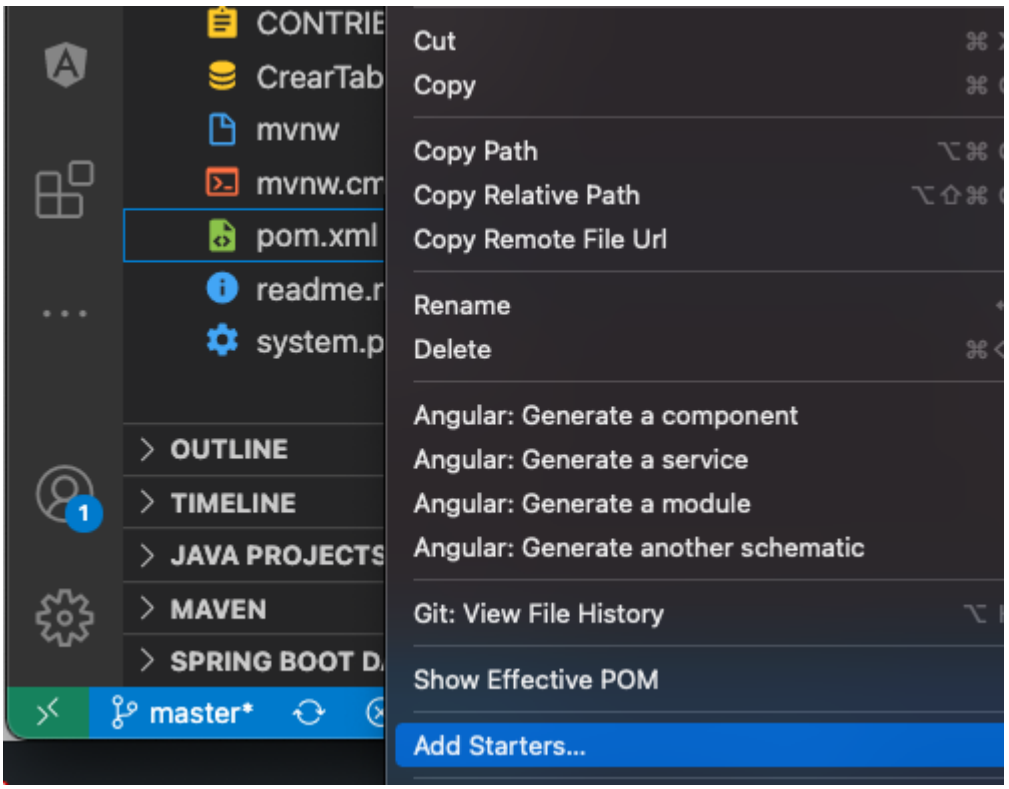
```
@GetMapping("/damefoto")
    public String dameFoto(){
        String foto = fotoService.getFoto();
        return "<img src='"+foto+"'/>";
    }
```

Usando un motor de vistas

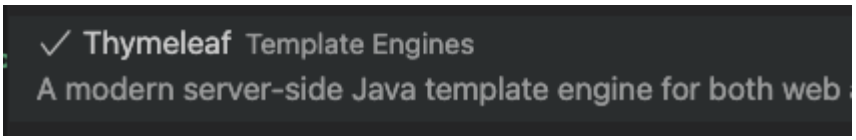
Cuando devolvemos información al cliente, es mucho más cómodo si usamos un motor de vistas (o templates) que construya la página web que voy a devolver al cliente. Para este ejemplo vamos a usar el Motor Thymeleaf.

Instalando la dependencia de Thymeleaf

Pulsamos con el botón derecho y seleccionamos la opción Add Starters...



Buscamos la dependencia llamada Thymeleaf y la instalamos



Si todo ha ido bien podemos observar como el fichero pom.xml se ha actualizado

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Creando la vista / template

En la carpeta resources → templates creamos un fichero cualquiera con extensión html. Nosotros para este ejemplo vamos a crear una página web donde mostremos el peso y foto de un pokemon determinado.

resources →
templates →
pokemon.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet" href="/style.css"/>
   <title>Document</title>
</head>
<body>
   <h1>Peso del pokemon</h1>
   <div class="container">
       <p>El pokemon <span th:text=${nombre}></span>
           pesa <span class="peso" th:text=${peso}></span> </p>
           <br/>
           <img th:src="${foto}"/>
   </div>
</body>
</html>
```

Observa que hemos usado etiquetas especiales (solo validas para thymeleaf) que permitirán sustituir esos valores por el nombre, peso y foto del pokemon adecuado. Esta información será enviada desde el controlador.

```
<span th:text=${nombre}></span>
```

```
<span class="peso" th:text=${peso}></span>
```

```
<img th:src="${foto}"/>
```

Mostrando la vista / template desde el controlador

Vamos a crear un controlador nuevo, creamos un fichero llamado WebController.java

```
package com.cebem.transformalotu.controllers;

import com.cebem.transformalotu.services.PokemonBDService;
import com.cebem.transformalotu.services.PokemonService;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
```

```java
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class WebController {

   @RequestMapping("/pesoPokemon")
   public String peso(@PathVariable String nombrePokemon,
                      Model modelo){
       modelo.addAttribute("nombre", "Pikachu");
       modelo.addAttribute("peso", 20);
       modelo.addAttribute("foto",
"https://i.pinimg.com/originals/dc/ab/b7/dcabb7fbb2f763d680d20a3d228cc6f9.jpg");
       return "pokemon";
   }
}
```

Observa que hemos anotado la clase como **@Controller**
El método posee un atributo Model que nos permitirá comunicarnos con la vista.
Le pasamos 3 valores a la vista usando el método **addAttribute** (el nombre, el peso y la foto)
Lo más interesante es que al devolver la palabra **"pokemon"** en realidad el sistema sustituye eso por la página web que hemos construido en el paso anterior pokemon.html.