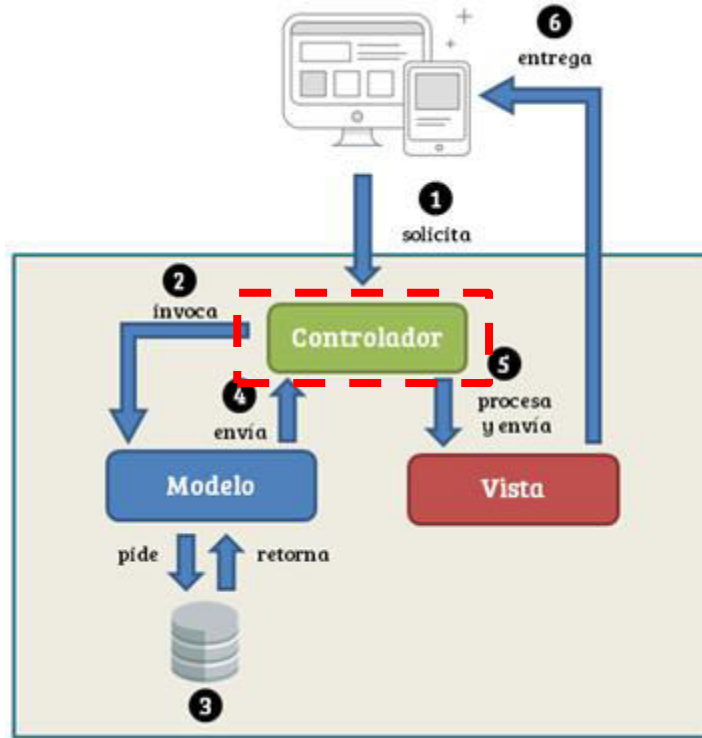


CONTROLADORES



CONTROLADOR

Clase que se va a encargar de atender peticiones y derivarnos a una vista adecuada.



¿CÓMO SE CREA UN CONTROLADOR?

- ▶ Clase POJO Java (*Plain Old Java Object*)
- ▶ Anotación `@Controller`
- ▶ Métodos anotados con `@RequestMapping` o sus derivados
 - ▶ `@GetMapping`
 - ▶ `@PostMapping`
 - ▶ `@PutMapping`
 - ▶ `@DeleteMapping`
 - ▶ `@PatchMapping`

ESTRUCTURA BÁSICA DE UN MÉTODO DEL CONTROLADOR

La clase es un pojo, no tiene que extender a ninguna otra obligatoriamente, tan solo anotada con `@Controller`

```
@Controller  
public class MainController {
```

```
    @GetMapping("/")  
    public String welcome() {  
        return "index";  
    }  
}
```

La anotación `@GetMapping` indica que este método se invoca cuando se produce una petición GET a /

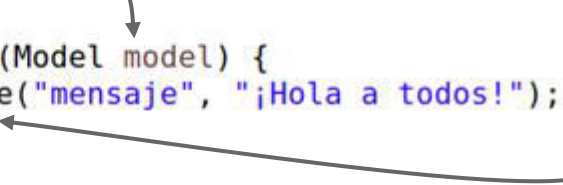
El método puede tener un nombre cualquiera, y no tiene por que recibir parámetros

El método devuelve un *String*. Ruta de la plantilla sin la extensión (que se supone es .html)

CÓMO ENVIAR DATOS A LA VISTA

La clase *Model* es un *Map*, que nos permite pasar objetos del controlador a la vista.

```
@GetMapping("/")  
public String welcome(Model model) {  
    model.addAttribute("mensaje", "¡Hola a todos!");  
    return "index";  
}
```



En nuestra plantilla Thymeleaf, podemos utilizar los datos recibidos.

```
<h1 th:text="${mensaje}">Mensaje</h1>
```



<https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/ui/Model.html>

POSIBLES ARGUMENTOS DE UN MÉTODO DEL CONTROLADOR

- ▶ *java.util.Map, org.springframework.ui.Model, org.springframework.ui.ModelMap*

Nos permite pasar datos a la vista

- ▶ *@ModelAttribute*

Permite acceder a un objeto del modelo (útil con formularios)

POSIBLES ARGUMENTOS DE UN MÉTODO DEL CONTROLADOR

- ▶ *@RequestBody*

Permite acceder a un objeto presente en el cuerpo de la petición

- ▶ *HttpEntity<?>*

Permite acceder a la petición (encabezado y cuerpo)

POSIBLES TIPOS DE RETORNO EN UN MÉTODO DEL CONTROLADOR

- ▶ *String*: es el más usual en las últimas versiones de Spring. Se trata del nombre de la plantilla, que será resuelto por el *ViewResolver* configurado (Spring Boot + Thymeleaf en el classpath)
- ▶ *@ResponseBody* + cualquier tipo de dato: se convierte el valor devuelto a través del conversor configurado.

POSIBLES TIPOS DE RETORNO EN UN MÉTODO DEL CONTROLADOR

- ▶ *HttpEntity<?>, ResponseEntity<?>*: se devuelve la respuesta HTTP completa (cabeceras y cuerpo).
- ▶ *ModelAndView*: modelo + vista en un solo objeto.

¿CUÁNTOS MÉTODOS PUEDE TENER UN CONTROLADOR?

- ▶ Puede tener cuantos necesitemos.
- ▶ No hay un límite determinado. Lo establece el diseño de clases de nuestra aplicación.

```
@Controller
public class MainController {

    @GetMapping("/")
    public String welcome(Model model) {
        model.addAttribute("mensaje", "¡Hola a todos!");
        return "index";
    }

    @GetMapping("/saludo")
    public String saludoCompleto(Model model) {
        model.addAttribute("saludo",
            "Seguro que has visto otras plataformas con miles de cursos, pero en OpenWebinars");
        return "saludo";
    }
}
```

ALGO PARA PRACTICAR

Un ejercicio para hacer por tu cuenta

CONTROLADOR DE UNA WEB CLÁSICA

- ▶ Las webs clásicas de pequeños negocios solían incluir:
 - ▶ **Portada.** Información sobre la organización (quiénes somos)
 - ▶ **Qué hacemos**
 - ▶ **Dónde estamos e información de contacto.**

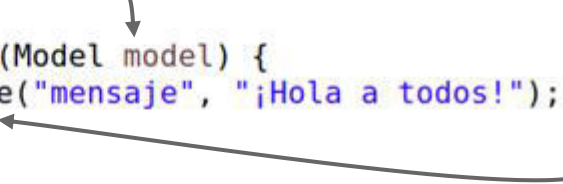
CONTROLADOR DE UNA WEB CLÁSICA

- ▶ Crea un nuevo proyecto, añadiendo los *starters* de WEB y THYMELEAF.
- ▶ Crea un controlador con 3 métodos, para que atiendan a las rutas `/`, `/que`, `/contacto`
- ▶ Crea las plantillas necesarias (*index.html*, *que.html* y *contacto.html*).
- ▶ Puedes pasar el contenido de las plantillas a través de un objeto *Model*.

CÓMO ENVIAR DATOS A LA VISTA

La clase *Model* es un *Map*, que nos permite pasar objetos del controlador a la vista.

```
@GetMapping("/")  
public String welcome(Model model) {  
    model.addAttribute("mensaje", "¡Hola a todos!");  
    return "index";  
}
```



En nuestra plantilla Thymeleaf, podemos utilizar los datos recibidos.

```
<h1 th:text="${mensaje}">Mensaje</h1>
```



<https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/ui/Model.html>