Clase grabada

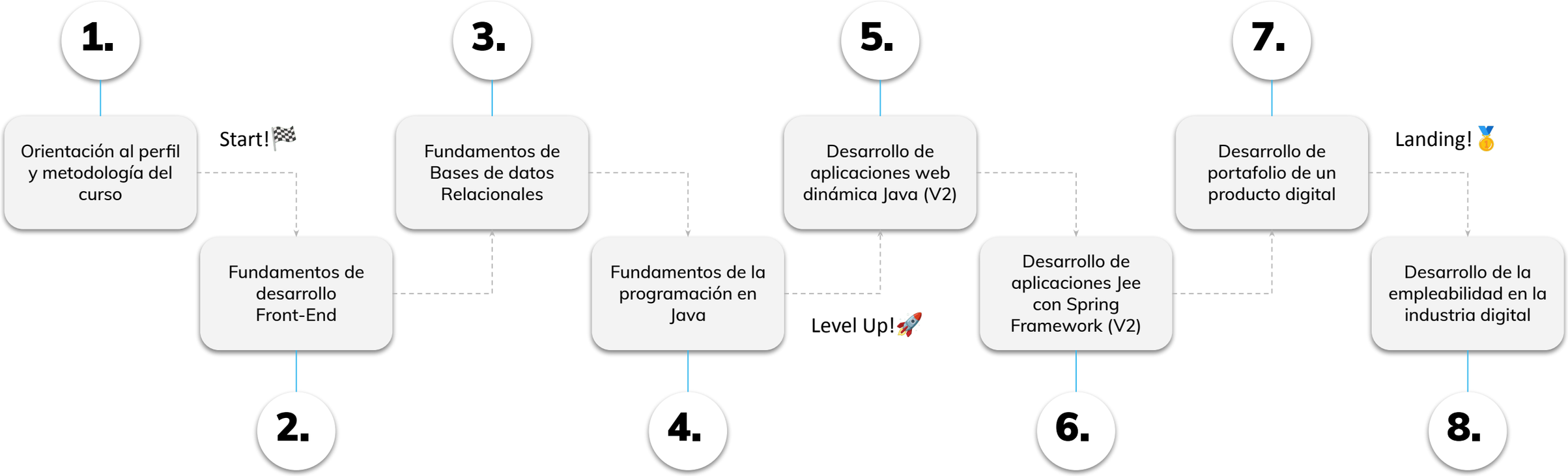
El Framework Spring MVC

Plan formativo: Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

English Always | alkemy

HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



English Always | alkemy

REPASO CLASE ANTERIOR

En la clase anterior trabajamos 📚:

- ✔ *Creación de proyecto Spring Boot*
- ✔ *Configuración de la tecnología de vista*
- ✔ *Configuración del datasource*

LEARNING PATHWAY

¿Sobre qué temas trabajaremos?

6.2

Start! 🏁

El Framework Spring MVC

Capa de Vistas y Controladores
Configurando las peticiones
Controladores multiacción

Controladores

Controlando

English Always | alkemy

OBJETIVOS DE APRENDIZAJE

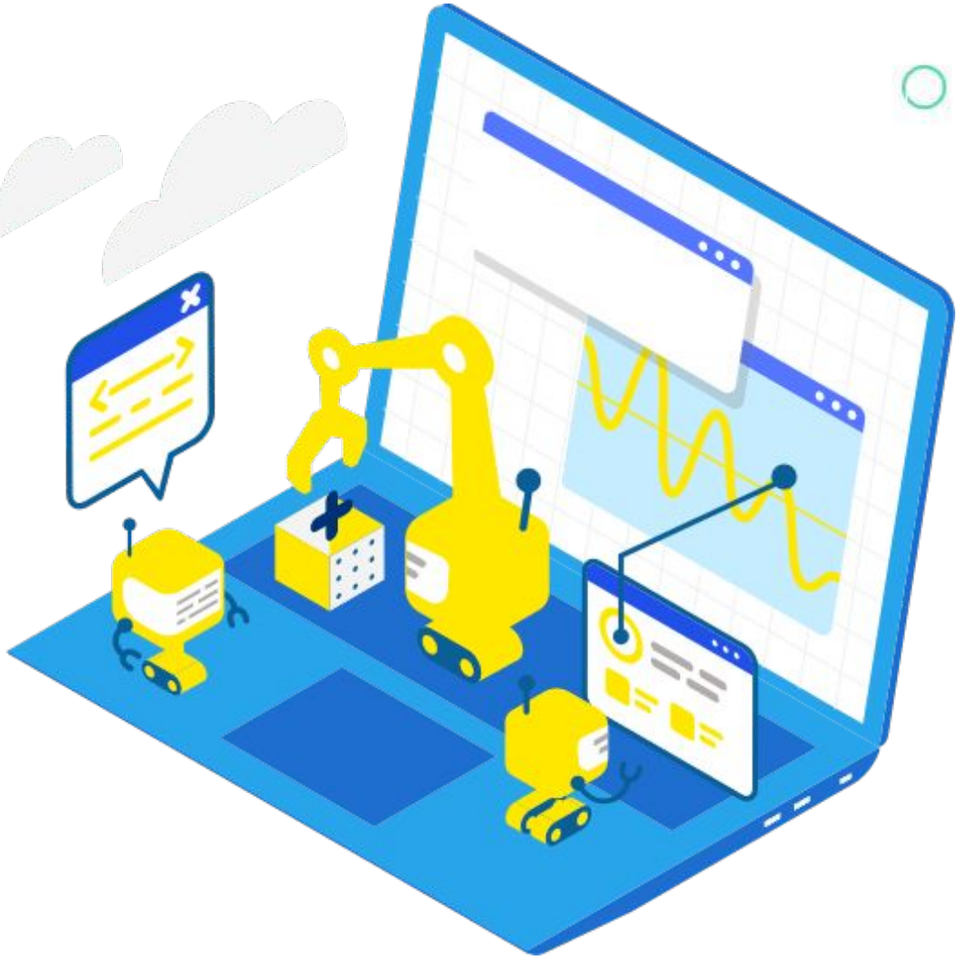
¿Qué aprenderemos?

- *Conocer la programación en capas con Vistas y Controladores*
- *Comprender la configuración de peticiones*

Controladores

Controladores

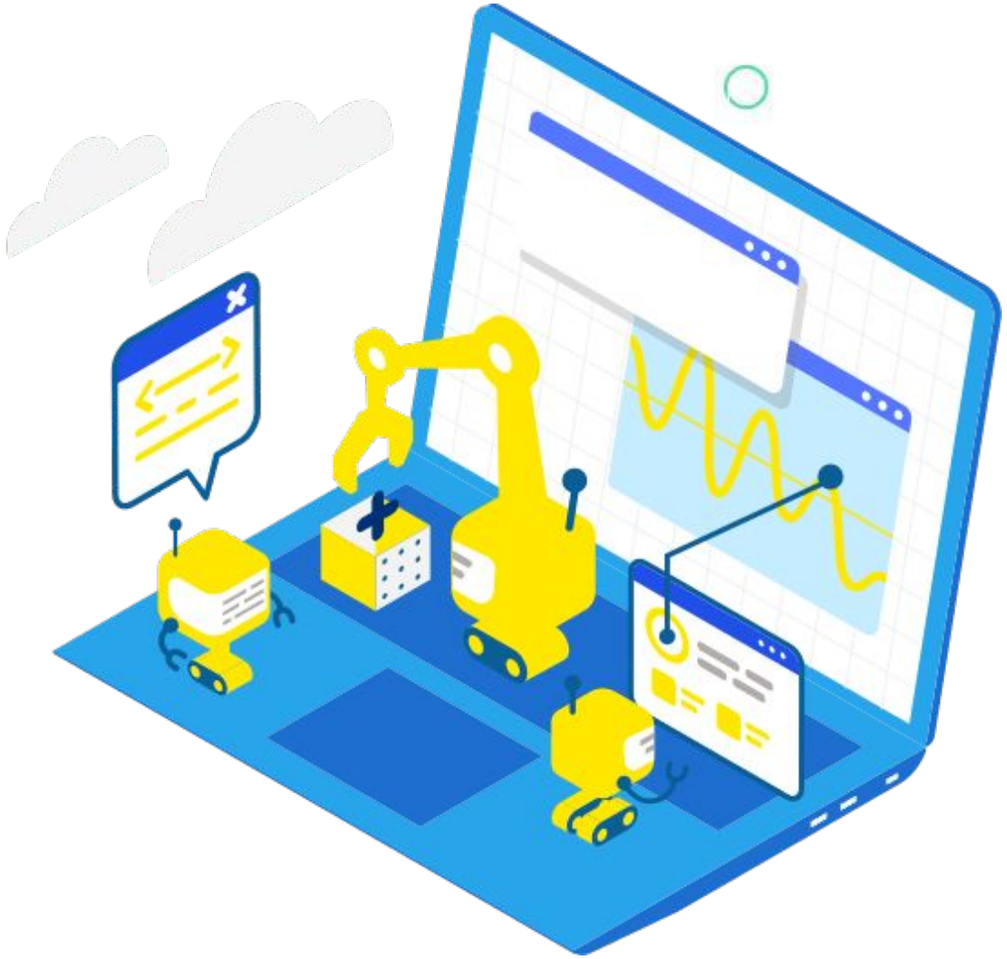
Un controlador en Java, específicamente en el contexto de desarrollo web, es una componente esencial que forma parte del patrón de diseño Modelo-Vista-Controlador (MVC). Un controlador se encarga de manejar las solicitudes del cliente, procesarlas y decidir cómo responder.



Controladores

Los **Controllers** son las clases Java que usaremos como controladores en nuestra aplicación web. Estas clases son muy parecidas a los Servlets de Java EE pero más sencillas de usar. Recuerda que usamos estas clases gracias a que estamos usando el framework Spring MVC (habiendo incluido la dependencia spring-web en nuestro proyecto).

No es necesario que los controladores en Spring MVC extiendan de ninguna otra clase ni que implementen ninguna interfaz. Únicamente deben incluir la anotación **@Controller** al principio de la clase.



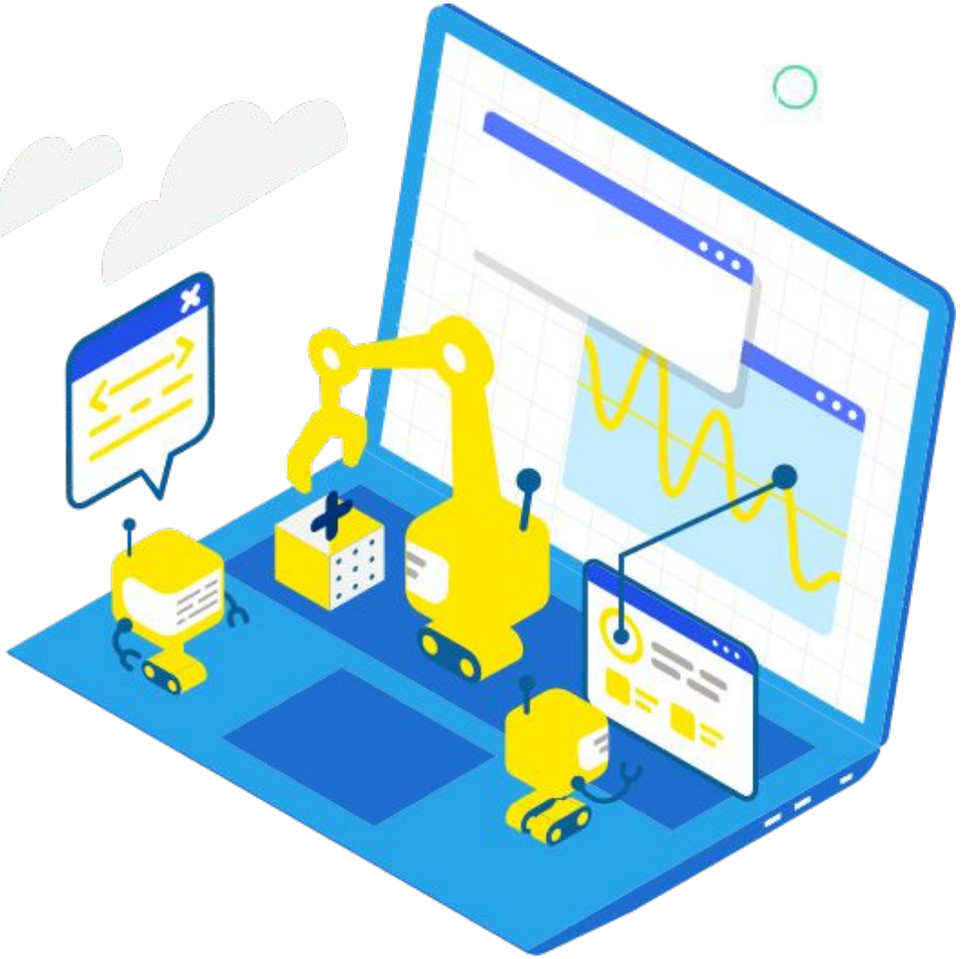
Controladores

Funciones y Responsabilidades:

- **Gestión de solicitudes**: son responsables de recibir las solicitudes HTTP entrantes, como solicitudes de página, envíos de formularios o llamadas a servicios web.

- **Procesamiento de solicitudes**: lo que puede incluir validación de datos, ejecución de la lógica de negocio y acceso a servicios o recursos necesarios.

- **Selección de modelo y vista**: Basado en la solicitud y el resultado del procesamiento, el controlador decide qué modelo utilizar y qué vista mostrar como respuesta.



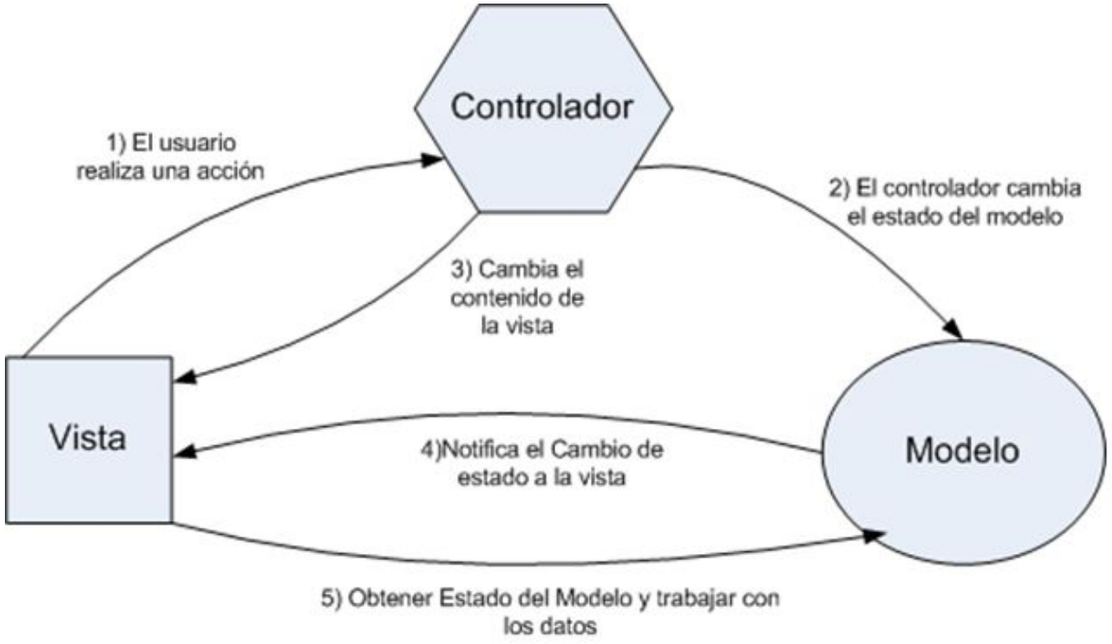
Controladores

Funciones y Responsabilidades:

- **Preparación de datos para la vista:** Los controladores pueden cargar datos desde el modelo y prepararlos para que sean representados en la vista. Esto puede incluir la recuperación de datos de la base de datos o la interacción con otros servicios.

- **Devolver respuestas al cliente:** retornan respuestas al cliente en forma de vistas, objetos JSON, archivos u otros tipos de datos, según la solicitud y los resultados del procesamiento.

Controladores



Controladores

Vamos a analizar por ejemplo el siguiente código

```java
@Controller
@RequestMapping({"/index.html"})
public class ProfessorController {
      @Autowired
        private ProfesorDAO profesorDAO;


        public ModelAndView read(HttpServletRequest request,HttpServletResponse response) {
            Map<String, Object> model = new HashMap<>();
            String viewName;

            try {
                Profesor profesor = profesorDAO.get(1001);
                model.put("texto", profesor.toString());
                viewName = "profesor";
            } catch (Exception ex) {
                model.put("msgError", "No es posible obtener los datos");
                viewName = "error";
            }

            return new ModelAndView(viewName, model);
        }

}
```

Controladores

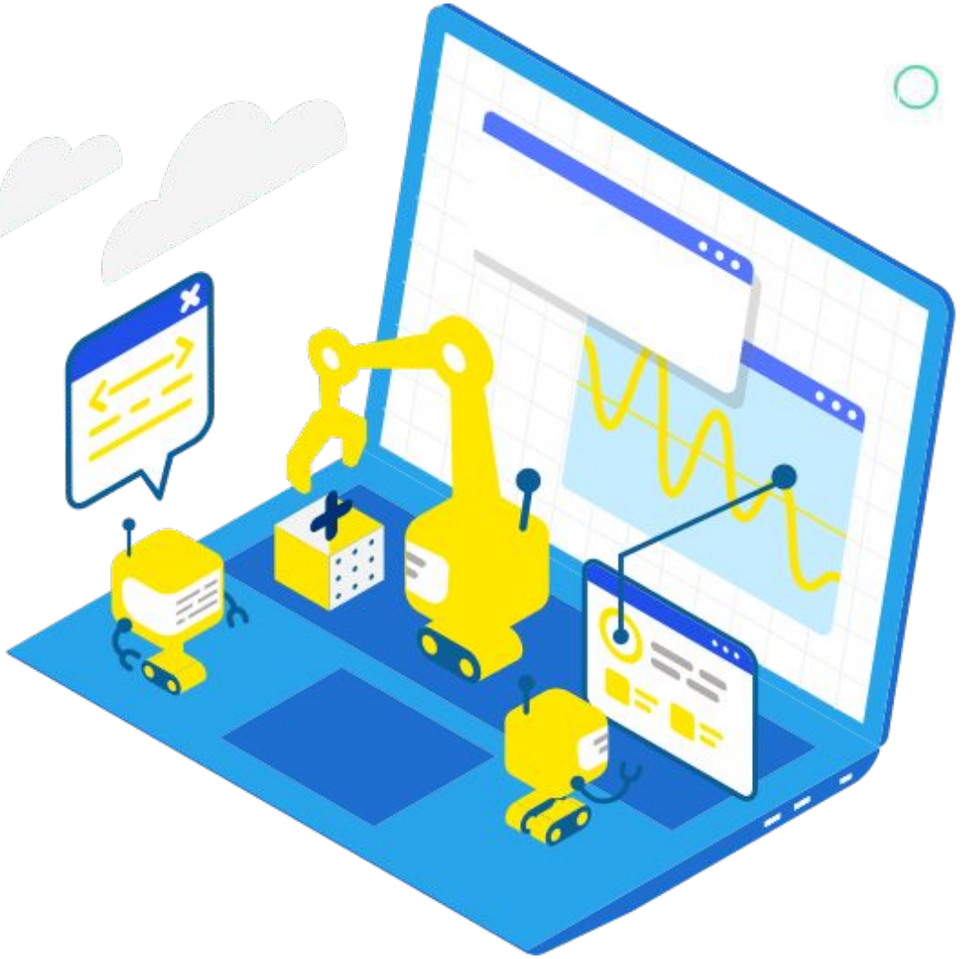
Como ya hemos dicho lo principal que debe tener un controller de Spring MVC es la anotación **@Controller** al principio de la clase (Línea 13). Con ésto Spring MVC sabrá que esta clase es un bean de tipo controlador.

```
13  @Controller
14  @RequestMapping({"/index.html"})
15  public class ProfessorController {
16      @Autowired
17        private ProfesorDAO profesorDAO;
18
19
20        public ModelAndView read(HttpServletRequest request,HttpServletResponse response) {
21            Map<String, Object> model = new HashMap<>();
22            String viewName;
23
24            try {
25                Profesor profesor = profesorDAO.get(1001);
26                model.put("texto", profesor.toString());
27                viewName = "profesor";
28            } catch (Exception ex) {
29                model.put("msgError", "No es posible obtener los datos");
30                viewName = "error";
31            }
32
33            return new ModelAndView(viewName, model);
34        }
35
36  }
```

Controladores

Manejo de peticiones

El **manejo de peticiones web se realiza a través de los métodos del controlador**. Cada método gestionará las peticiones de una o más URLs y puede haber tantos métodos como se desee. Ésto es una gran ventaja respecto a un Servlet ya que hace mucho más sencillo procesar una serie de URL relacionadas en un único controlador. Recuerda que en un Servlet había un único método para todas las peticiones.



Controladores

Cada método que controla peticiones Web tendrá la anotación **@RequestMapping** (Línea 14). Esta anotación tendrá como argumento un Array de Strings con todas las URLs que va a manejar. Por ello nótese que hay una llave “{ }” antes y después del String de la URL que maneja, ya que se permite más de una.

```
13  @Controller
14  @RequestMapping({"/index.html"})
15  public class ProfessorController {
16      @Autowired
17        private ProfesorDAO profesorDAO;
18
19
20        public ModelAndView read(HttpServletRequest request,HttpServletResponse response) {
21            Map<String, Object> model = new HashMap<>();
22            String viewName;
23
24            try {
25                Profesor profesor = profesorDAO.get(1001);
26                model.put("texto", profesor.toString());
27                viewName = "profesor";
28            } catch (Exception ex) {
29                model.put("msgError", "No es posible obtener los datos");
30                viewName = "error";
31            }
32
33            return new ModelAndView(viewName, model);
34        }
35
36  }
```

Controladores

Los argumentos de entrada del método son los objetos **HttpServletRequest** y **HttpServletResponse** al igual que en un Servlet.

Spring MVC tiene muchas funcionalidades relativas a las URL que controla: **parámetros de entrada y de salida**.

El método retorna una clase de Spring MVC llamada **ModelAndView**. Esta clase contendrá un String con la página **jsp** a mostrar (es decir la **Vista**) y un **Map** con los datos que necesita la vista para mostrarse (es decir el **Modelo**).

Controladores

Podemos ver en la línea 33: return new ModelAndView(viewName, model); como se crea un nuevo objeto **ModelAndView** que tiene como argumentos un String con el nombre de la página **JSP** a mostrar y el **Map** con los datos para la vista.

```
13  @Controller
14  @RequestMapping({"/index.html"})
15  public class ProfessorController {
16      @Autowired
17        private ProfesorDAO profesorDAO;
18
19
20        public ModelAndView read(HttpServletRequest request,HttpServletResponse response) {
21            Map<String, Object> model = new HashMap<>();
22            String viewName;
23
24            try {
25                Profesor profesor = profesorDAO.get(1001);
26                model.put("texto", profesor.toString());
27                viewName = "profesor";
28            } catch (Exception ex) {
29                model.put("msgError", "No es posible obtener los datos");
30                viewName = "error";
31            }
32
33            return new ModelAndView(viewName, model);
34        }
35
36  }
```

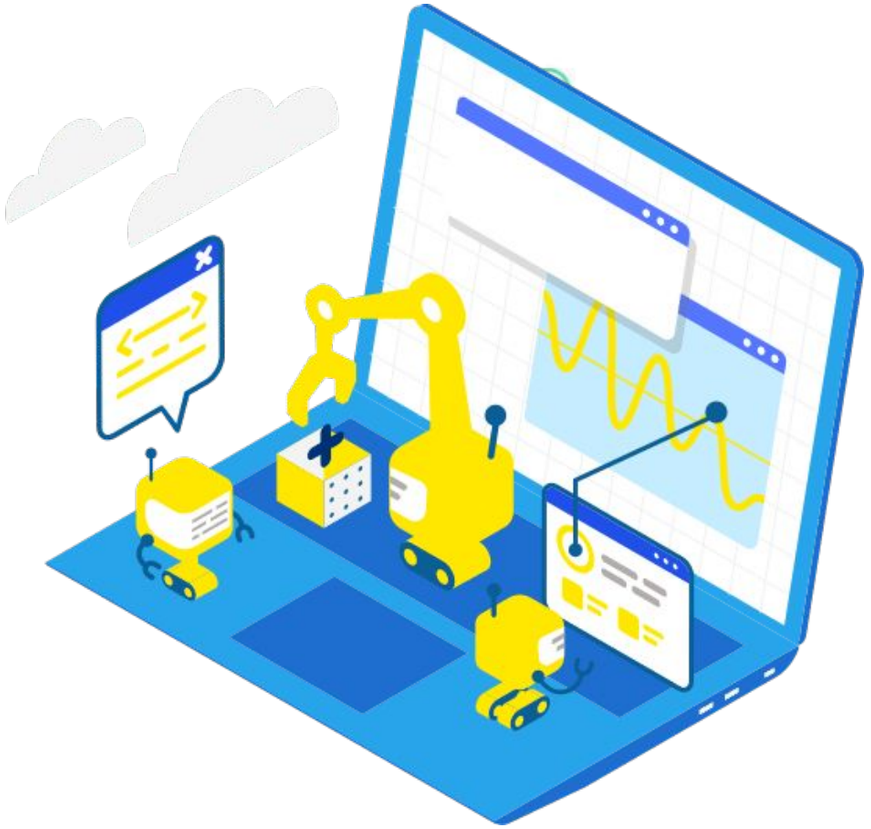
Controladores

Beneficios de un controlador en Java:

Separación de preocupaciones: La arquitectura MVC, con un controlador en el centro, permite una clara separación de las preocupaciones de presentación, lógica de negocio y acceso a datos. Esto hace que el código sea más mantenible y escalable.

Reutilización de código: Los controladores permiten reutilizar la lógica de negocio en diferentes vistas y acciones. Esto evita la duplicación de código y promueve la coherencia en la aplicación.

Flexibilidad y extensibilidad: Al utilizar un controlador, puedes cambiar fácilmente la vista que se muestra sin modificar la lógica subyacente. También es más sencillo agregar nuevas funcionalidades a la aplicación sin afectar otras partes del sistema.



Controladores

Beneficios de un controlador en Java:

Pruebas unitarias: Los controladores son componentes que se pueden probar de manera aislada, lo que facilita la creación de pruebas unitarias y la verificación de que la lógica funcione correctamente.

Escalabilidad: Al separar las preocupaciones y utilizar un controlador, es más sencillo escalar una aplicación, ya que puedes agregar nuevos controladores y vistas según sea necesario.

Mantenimiento más sencillo: La estructura organizada y la separación de preocupaciones hacen que el mantenimiento de la aplicación sea más manejable, ya que es más fácil identificar y solucionar problemas.

Vista JSP

Vista JSP

La página JSP

Las páginas JSP se deben colocar en la carpeta src/main/resources/templates.

La vista tiene acceso a los elementos del Map del modelo que se creó en la clase ModelAndView mediante el siguiente código:

```
Object texto=request.getAttribute("texto");
```

Siendo el String “texto” la clave del Map con la que se guardó el valor.

Vista JSP

Vemos en la línea 3 cómo se accede a los datos del modelo.

```
1  <%@page contentType="text/html" pageEncoding="ISO-8859-1"%>
2  <%
3  Object texto = request.getAttribute("texto");
4  %>
5  <!DOCTYPE html>
6  <html>
7  <head>
8  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
9  <title>Profesor</title>
10 </head>
11 <body>
12     <h1><%=texto%></h1>
13 </body>
14 </html>
```

Controladores Multiacción

Controladores Multiacción

En el contexto de Spring MVC, un controlador **multiaction** se refiere a una técnica en la cual **un único controlador puede manejar múltiples acciones** (métodos) basados en diferentes URL o parámetros de solicitud. Esto permite agrupar varias acciones relacionadas en un solo controlador, lo que puede mejorar la organización y la legibilidad del código.

A menudo se utilizan en aplicaciones web para simplificar la gestión de múltiples acciones relacionadas, como operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en recursos específicos.

Controladores Multiacción

Características de los Controladores Multiacción:

1. **Acciones basadas en parámetros**: En lugar de asociar una ruta específica a cada método en un controlador, los controladores multiacción analizan los parámetros de la solicitud para determinar qué acción debe llevarse a cabo.

2. **Centralización de la lógica**: Las acciones relacionadas se agrupan en una única clase de controlador, lo que facilita la organización y el mantenimiento del código.

3. **Reducción de duplicación de código**: Al permitir que un solo método maneje varias acciones, se evita la duplicación de código similar en diferentes métodos de controlador.

Controladores Multiacción

Beneficios de los Controladores Multiacción:

Simplificación de rutas: Se pueden utilizar rutas más generales en las solicitudes, lo que puede mejorar la legibilidad de las URL y simplificar la configuración de rutas.

Centralización de la lógica de control: La lógica relacionada se encuentra en un solo lugar, lo que facilita la comprensión y el mantenimiento del código.

Menos código repetitivo: Al manejar varias acciones en un solo método, se reduce la necesidad de duplicar código similar en diferentes métodos de controlador.

Controladores Multiacción

Supongamos que estamos desarrollando una aplicación web para administrar una lista de tareas. En lugar de crear un método de controlador separado para "crear tarea", "actualizar tarea" y "eliminar tarea", podríamos usar un controlador multiacción.

```
@Controller
public class TareasController {

    @PostMapping("/tareas")
    public String gestionarTarea(@RequestParam String accion
        if ("crear".equals(accion)) {
            // Lógica para crear una tarea
        } else if ("actualizar".equals(accion)) {
            // Lógica para actualizar una tarea
        } else if ("eliminar".equals(accion)) {
            // Lógica para eliminar una tarea
        }
        // Redirigir a la página de tareas
        return "redirect:/tareas";
    }
}
```

Controladores Multiacción

En este ejemplo, el controlador TareasController maneja las acciones relacionadas con las tareas. El parámetro en la solicitud determina si se debe crear, actualizar o eliminar una tarea. La lógica de cada acción se maneja dentro del mismo método, lo que simplifica el código y centraliza la gestión de tareas.

```
@Controller
public class TareasController {

    @PostMapping("/tareas")
    public String gestionarTarea(@RequestParam String accion
        if ("crear".equals(accion)) {
            // Lógica para crear una tarea
        } else if ("actualizar".equals(accion)) {
            // Lógica para actualizar una tarea
        } else if ("eliminar".equals(accion)) {
            // Lógica para eliminar una tarea
        }
        // Redirigir a la página de tareas
        return "redirect:/tareas";
    }
}
```

Controladores Multiacción

Consideraciones y Mejores Prácticas:

- Si bien los pueden simplificar la gestión de múltiples acciones, es importante mantenerlos organizados y asegurarse de que cada acción tenga su propia lógica específica.

- Debes tener cuidado de no sobrecargar un controlador con demasiadas acciones, ya que esto podría complicar la legibilidad y el mantenimiento del código.

- Si las acciones se vuelven complejas o numerosas, considera la posibilidad de dividir el controlador en múltiples clases para mantener la organización y la claridad del código.

- La elección de utilizar controladores multiacción o controladores convencionales depende de la arquitectura de tu aplicación y de tus preferencias personales. Ambos enfoques tienen sus propias ventajas y desventajas.

LIVE CODING

Ejemplo en vivo

WalletController:

Vamos a comenzar el diseño de una Billetera Virtual para un cliente bancario que desea brindar una nueva experiencia a sus usuarios, incluyendo en el catálogo de beneficios una Wallet donde puedan gestionar algunas de las principales transacciones bancarias.

Los usuarios deben poder ver su saldo disponible, realizar depósitos y retiros de fondos.

Tiempo: 20 minutos

LIVE CODING

Ejemplo en vivo

1. *En el proyecto AlkeWallet, crear una clase Cuenta*
2. *Agregar los atributos String nombreUsuario, Integer numeroCuenta, Double saldoActual, String verDatos*
3. *Crear una clase CuentaController*
4. *Crear los métodos verSaldo, depositar, retirar y verDatos*
5. *Crear una vista “cuenta.jsp”*
6. *Mostrar, a modo de ejemplo, como se puede enviar el saldo del usuario a la vista.*

¿Alguna consulta?

Ejercicio

Controlando

Controlando

Consigna: ✍️

Vamos a hacer un pequeño ejercicio de prueba para aplicar las anotaciones e importaciones necesarias.

En la carpeta src/main/java, desde el paquete principal (com.example.demo”) crea un nuevo paquete llamado “controladores” (com.example.demo.controladores).

En este nuevo paquete debes:

1- Crear una clase @Controller que responda a una petición @RequestMapping(“/index”)

2- Crear un método que retorne un String “nombre”

Tiempo🕐: *10 minutos*

RESUMEN

¿Qué logramos en esta clase?

- ✓ *Conocer la capa de vista y controladores*
- ✓ *Aprender a configurar un Controller*
- ✓ *Controladores Multiacción*

English Always | alkemy

#WorkingTime

Continuemos ejercitando

¡Antes de cerrar la clase! Te invitamos a: 👇👇👇

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
   a. *Lectura Modulo 6, Lección 2: páginas 21 - 26*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

¡Muchas Gracias!

Nos vemos en la próxima clase 👋🏻

English Always | alkemy