Clase grabada

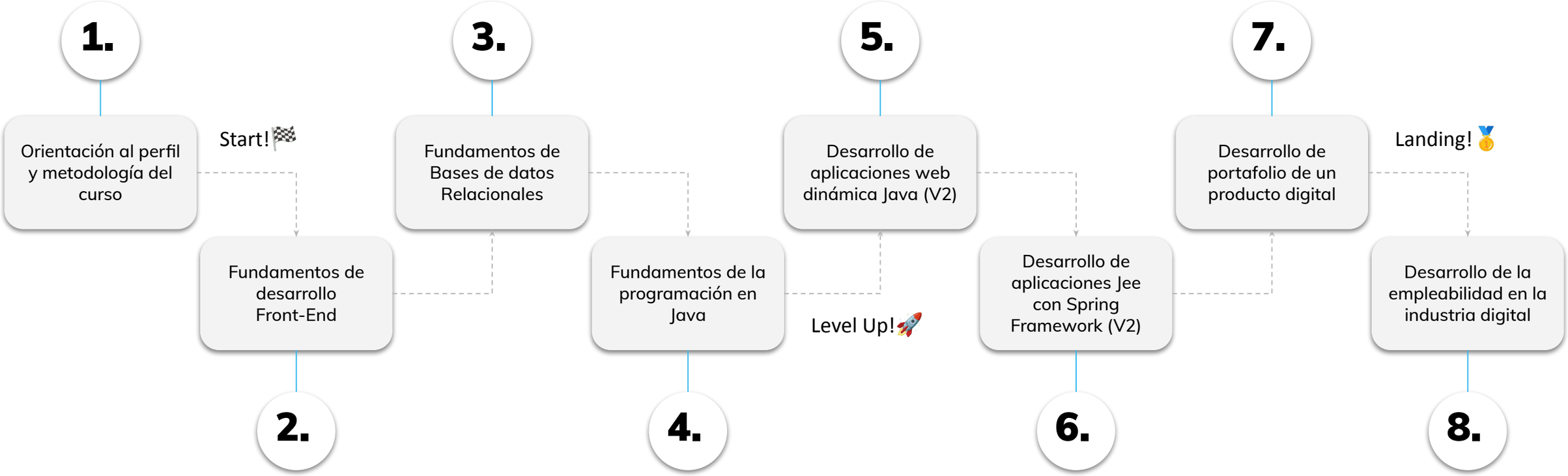
La interoperabilidad entre los sistemas

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 📚:

✓ *Aprender a crear una API REST con Spring MVC*

LEARNING PATHWAY

N° de la unidad . ¿Sobre qué temas trabajaremos?

6.5 Start!🏁

La interoperabilidad entre los sistemas

API REST

API REST

Creando una API REST con Spring MVC

English Always | alkemy

OBJETIVOS DE APRENDIZAJE

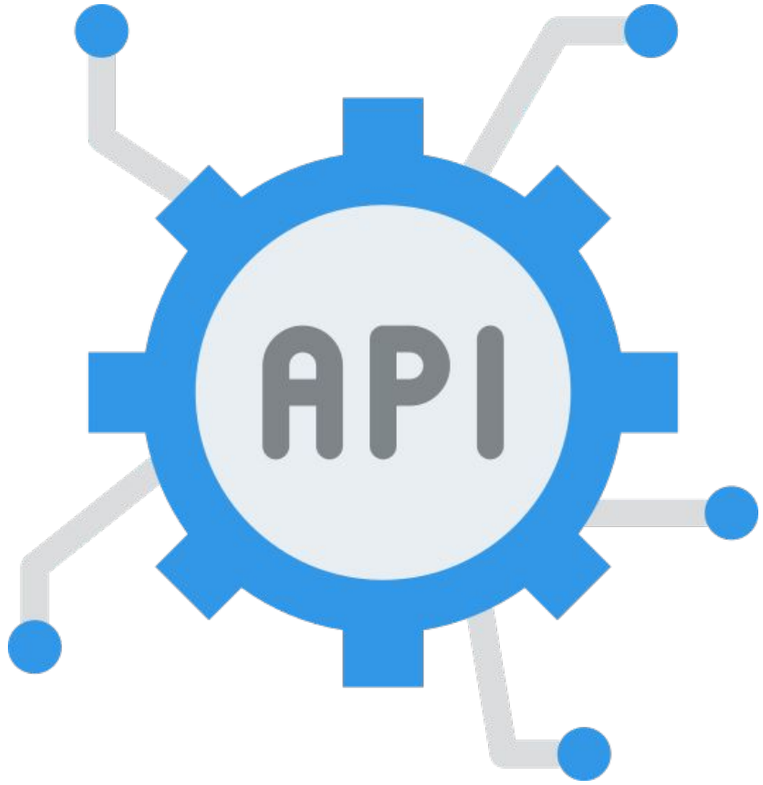
¿Qué aprenderemos?

- ***Crear una API REST con Spring MVC***

Creando una API REST con Spring MVC

Creando una API REST con Spring MVC

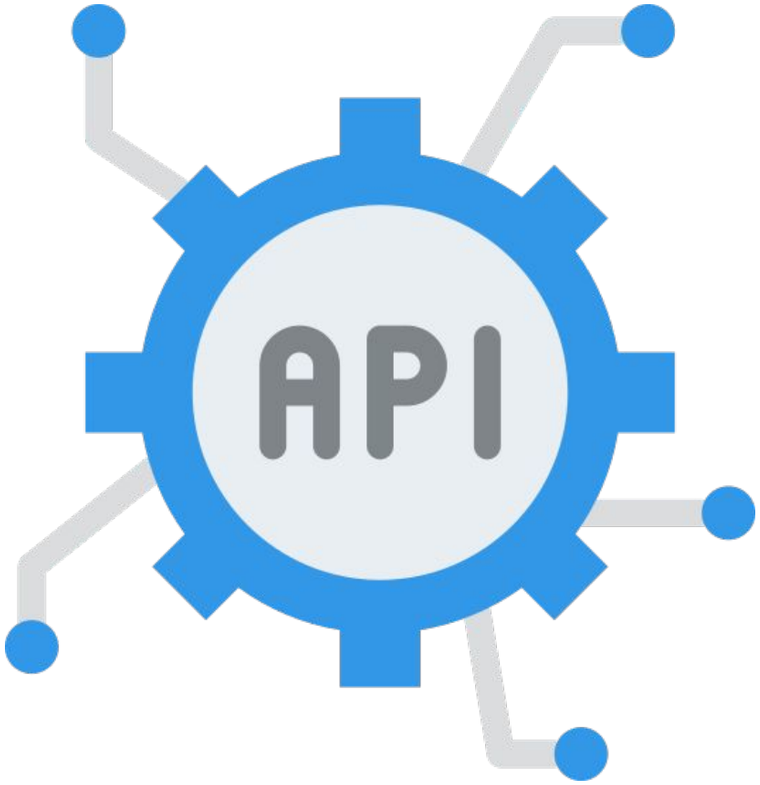
Spring MVC (Model-View-Controller) es un marco de trabajo basado en el patrón de diseño Modelo-Vista-Controlador. Facilita la creación de aplicaciones web y APIs REST mediante la separación de responsabilidades en capas distintas. En particular, Spring MVC proporciona un enfoque robusto para la creación de servicios web RESTful.



Creando una API REST con Spring MVC

Características de Spring MVC para APIs REST

1. Anotaciones para Mapeo de Solicitudes
Una de las características más destacadas de Spring MVC es su uso extensivo de **anotaciones** para mapear solicitudes HTTP a métodos específicos en controladores. Esto elimina la necesidad de configuraciones XML engorrosas y hace que el código sea más limpio y legible.



Creando una API REST con Spring MVC

En este ejemplo, la anotación @RestController indica que la clase es un controlador de Spring MVC que maneja solicitudes REST. La anotación @RequestMapping establece la ruta base para todas las operaciones definidas en la clase.

```java
@RestController
@RequestMapping("/api/usuarios")
public class UsuarioControlador {

    @GetMapping("/{id}")
        public ResponseEntity<Usuario> obtenerUsuario(@PathVariable Long id) {
            // Lógica para obtener y devolver un usuario por ID
        }

        @PostMapping
        public ResponseEntity<String> crearUsuario(@RequestBody Usuario nuevoUsuario) {
            // Lógica para crear un nuevo usuario
        }
    }
```

Creando una API REST con Spring MVC

2. **Conversión Automática de Datos a Formato JSON**

Spring MVC facilita la creación de APIs que envían y reciben datos en formato JSON de manera transparente. Utiliza la biblioteca Jackson para la conversión automática de objetos Java a formato JSON y viceversa. Aquí, la respuesta del método obtenerUsuario se convertirá automáticamente a JSON antes de ser enviada al cliente.

```
@GetMapping("/{id}")
    public ResponseEntity<Usuario> obtenerUsuario(@PathVariable Long id) {
        // Lógica para obtener y devolver un usuario por ID
     return ResponseEntity.ok(usuario);
    }
```

Creando una API REST con Spring MVC

3. **Soporte para Validación de Datos**

Spring MVC ofrece un sólido soporte integrado para la validación de datos. Al utilizar anotaciones de validación como @Valid y @RequestBody, los desarrolladores pueden validar automáticamente los datos recibidos en las solicitudes. En este ejemplo, la anotación @Valid indica que el objeto nuevoUsuario debe ser validado antes de que la lógica del controlador sea ejecutada.

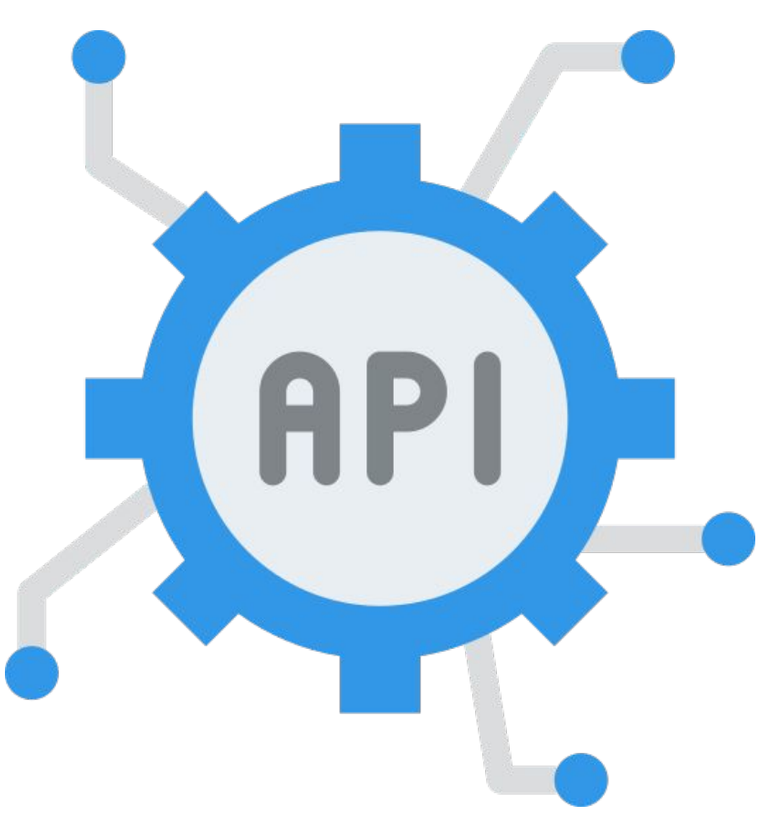
```
@PostMapping
public ResponseEntity<String> crearUsuario(@Valid @RequestBody Usuario nuevoUsuario) {
    // Lógica para crear un nuevo usuario
    return ResponseEntity.ok("Usuario creado correctamente");
}
```

Creando una API REST con Spring MVC

Ventajas de Crear una API REST con Spring MVC

Facilidad de Configuración y Desarrollo: Una de las principales ventajas de Spring MVC es su facilidad de configuración. Al aprovechar las convenciones de opinión y la configuración basada en anotaciones, los desarrolladores pueden crear APIs de manera rápida y eficiente.

Abstracción de Detalles de Bajo Nivel: Spring MVC abstrae muchos de los detalles de bajo nivel relacionados con la manipulación de solicitudes HTTP y la serialización de datos. Esto permite a los desarrolladores centrarse en la lógica de negocio en lugar de preocuparse por la manipulación directa de la entrada y salida HTTP.



Creando una API REST con Spring MVC

Extensibilidad y Personalización: La arquitectura de Spring MVC es altamente extensible. Los desarrolladores pueden personalizar y extender el comportamiento predeterminado mediante la implementación de interfaces y la configuración de componentes personalizados.

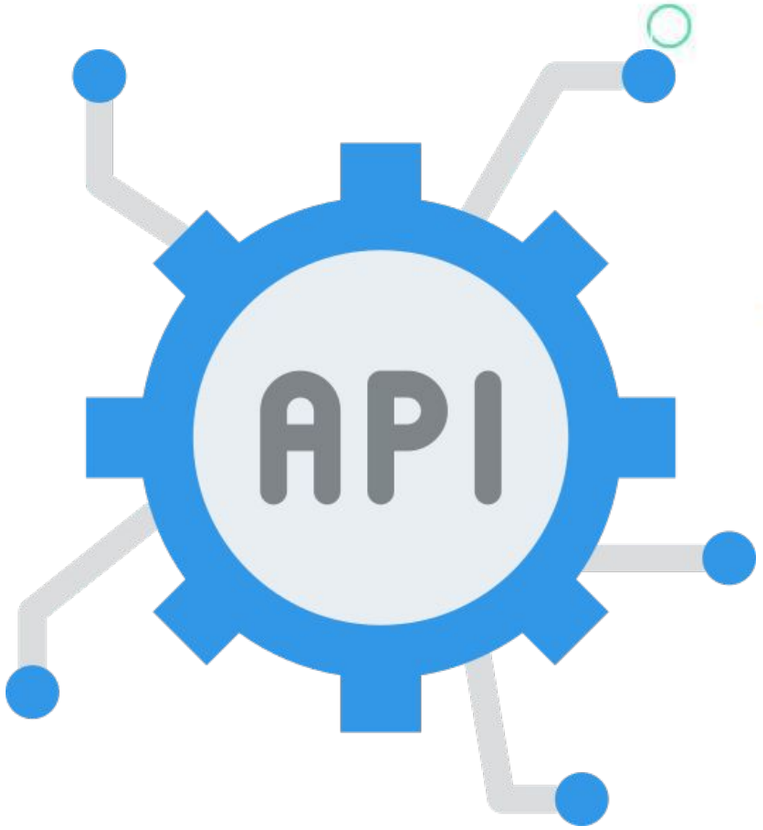
Integración: Spring MVC se integra de manera natural con otras características clave de Spring, como Spring Security para la implementación de seguridad, Spring Data para la integración con bases de datos, y Spring Boot para el desarrollo de aplicaciones autónomas.

Creando una API REST con Spring MVC

⛔Desventajas de Crear una API REST con Spring MVC

Curva de Aprendizaje Inicial: Para los desarrolladores nuevos en Spring MVC, la curva de aprendizaje inicial puede ser empinada, especialmente si no están familiarizados con los conceptos fundamentales de Spring. Sin embargo, la abundancia de recursos en línea y la documentación oficial de Spring pueden mitigar este desafío.

Complejidad en Proyectos a Gran Escala: A medida que los proyectos crecen en complejidad y tamaño, la gestión de configuraciones y la comprensión completa de la arquitectura de Spring MVC pueden volverse desafiantes. Se recomienda un diseño cuidadoso y la implementación de mejores prácticas para abordar este desafío.



Creando una API REST con Spring MVC

También permite hacer una llamada a una api externa y devolver una vista con la información a través del modelo, por ejemplo:

```
@GetMapping("/external")
public ModelAndView getExternalBooks() {

  String url = "https://api.example.com/books";
  List<Book> books = restTemplate.getForObject(url, List.class);

  ModelAndView mav = new ModelAndView("external");
  mav.addObject("books", books);

  return mav;

}
```

Creando una API REST con Spring MVC

También permite hacer una llamada a una api externa y devolver una vista con la información a través del modelo, por ejemplo:

```
@GetMapping("/external")
public ModelAndView getExternalBooks() {

  String url = "https://api.example.com/books";
  List<Book> books = restTemplate.getForObject(url, List.class);

  ModelAndView mav = new ModelAndView("external");
  mav.addObject("books", books);

  return mav;

}
```

Evaluación Integradora ✨

¿Listos para un nuevo desafío? En esta clase comenzamos a construir nuestro....

Trabajo Integrador del Módulo 💪🏽

Iremos completándolo progresivamente clase a clase.

English Always | alkemy

LIVE CODING

Ejemplo en vivo

AlkeWallet API Rest: Vamos a completar el proyecto de billetera virtual. Para comprobar nuestro avance, veremos algunas partes ya desarrolladas y otras por desarrollar.

Parte 1: Registro e inicio de sesión

1- Clase de entidad llamada Usuario que represente los datos del usuario.

2- Interfaz de repositorio llamada UsuarioRepository que permita interactuar con la base de datos.

3- Controlador llamado UsuarioController que implemente los métodos necesarios para el registro e inicio de sesión de usuarios.

LIVE CODING

Ejemplo en vivo

Parte 2: Administración de fondos

1- Clase de entidad llamada Cuenta que represente los datos de la cuenta del usuario.

2- Interfaz de repositorio llamada CuentaRepository que permita interactuar con la base de datos.

3- Controlador llamado CuentaController que implemente los métodos necesarios para la administración de fondos de los usuarios.

LIVE CODING

Ejemplo en vivo

Parte 3: Envío y recepción de fondos

Vamos a agregar una nueva entidad con su correspondiente implementación y controlador:

1- Clase de entidad llamada Transaccion que represente los datos de la transacción.

2- Crea una interfaz de repositorio llamada TransaccionRepository que permita interactuar con la base de datos.

3- Crea un controlador llamado TransaccionController que implemente los métodos necesarios para el envío y recepción de fondos entre usuarios.

LIVE CODING

Ejemplo en vivo

Parte 4: Historial de transacciones

1- Crea una clase de entidad llamada HistoricoTransacciones que represente los datos del historial de transacciones.

2- Crea una interfaz de repositorio llamada HistoricoTransaccionesRepository que permita interactuar con la base de datos.

3- Crea un controlador llamado HistoricoTransaccionesController que implemente los métodos necesarios para la consulta del historial de transacciones de los usuarios.

Momento:

Time-out!

⏰10 min.

¿Alguna consulta?

RESUMEN

¿Qué logramos en esta clase?

- ✓ *Aprender a crear una API REST con Spring MVC*

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 4: páginas 9 - 11*
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