Clase grabada

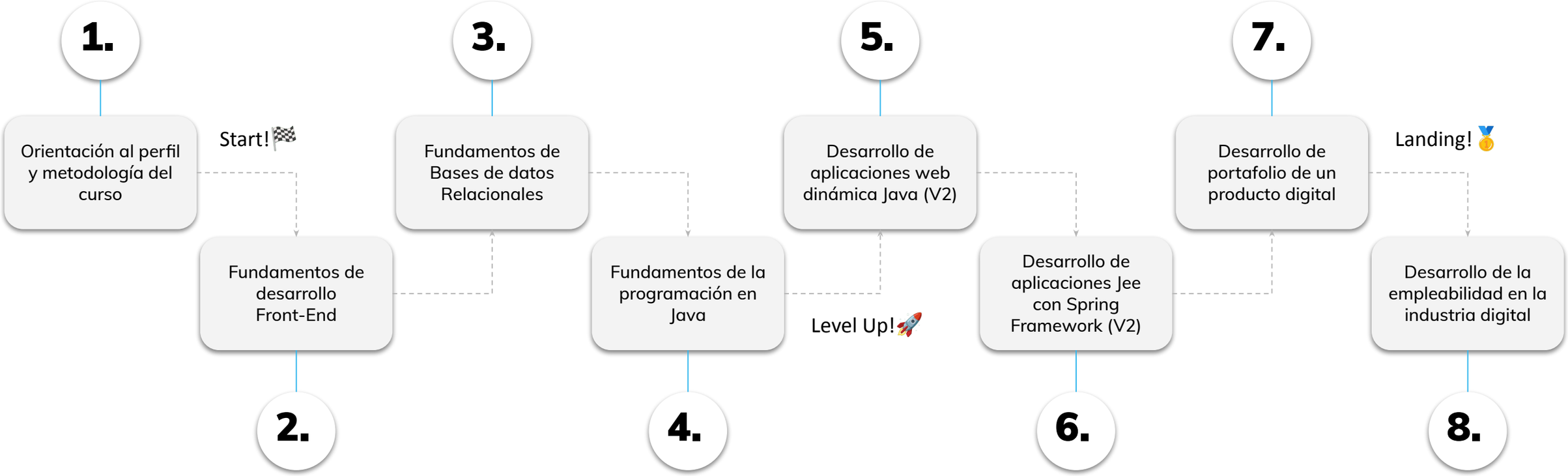
El Framework Spring MVC

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 📚:

- ✔ *Recibir datos en el controlador y enviar datos a la vista*
- ✔ *Realizar el despliegue de una vista JSP con datos entregados en el Controlador*

LEARNING PATHWAY

¿Sobre qué temas trabajaremos?

6.2 Start! 🏁

El Framework Spring MVC → Capa de Servicio → @Servicios

Capa de Servicios
El rol de la capa de servicio en el modelo MVC
Creando un servicio utilizando anotaciones
Inyectando el servicio a un controlador para su utilización

English Always | alkemy

OBJETIVOS DE APRENDIZAJE

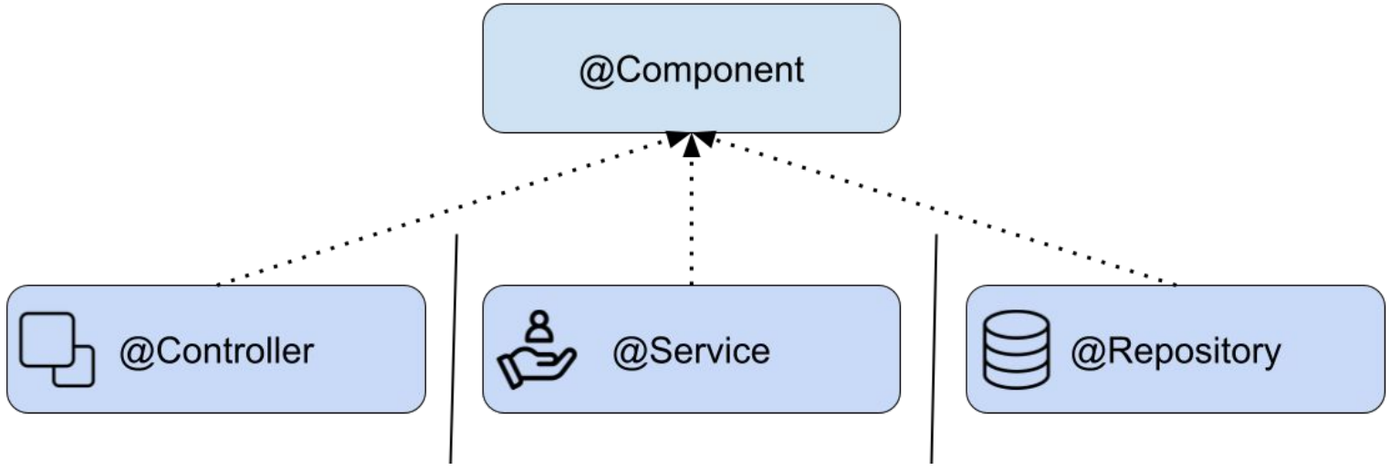
¿Qué aprenderemos?

- *Aprender a aplicar la capa de servicios en proyectos Spring*
- *Comprender la inyección de servicios en los controladores*

Capa de Servicios

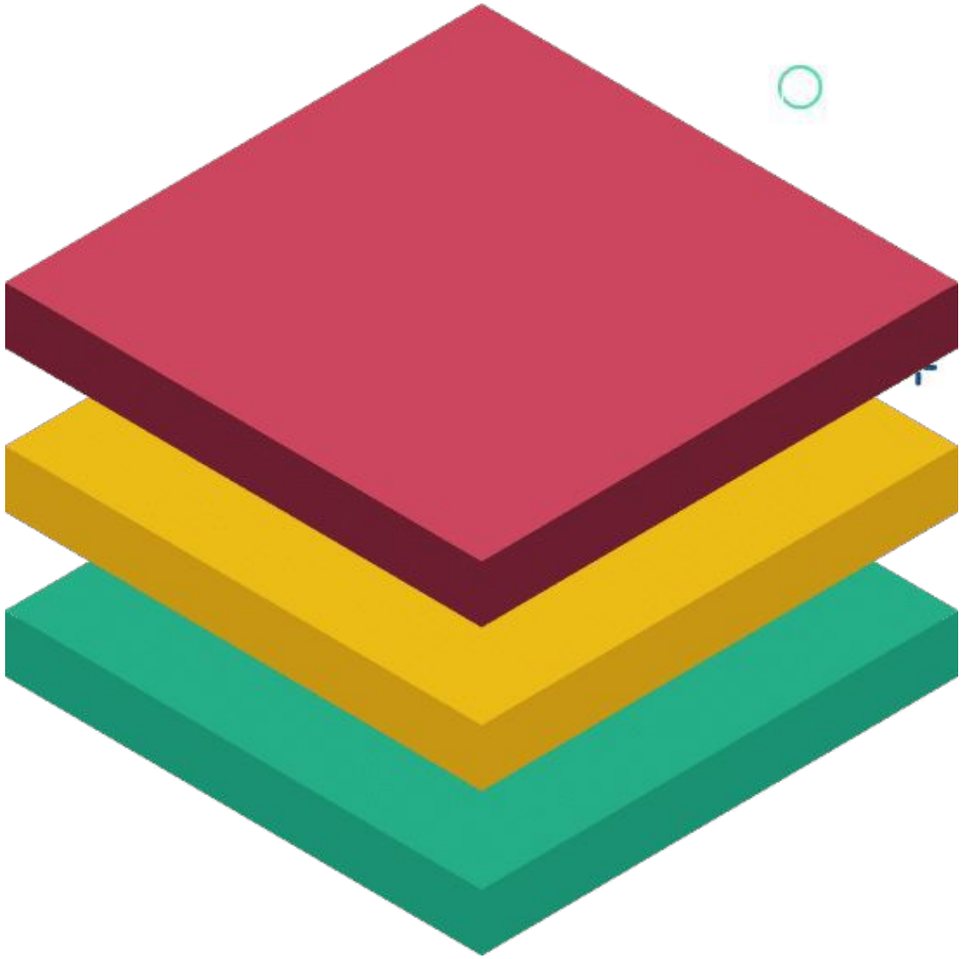
Capa de Servicios

En el patrón de arquitectura **Modelo-Vista-Controlador** (MVC), la **capa de servicio** (también conocida como capa de negocio o capa control) es responsable de **implementar la lógica del negocio** y coordinar las operaciones entre la capa de presentación (vista) y la capa de acceso a datos (modelo).



Capa de Servicios

El propósito principal de la capa de servicio es **encapsular la lógica del negocio y proporcionar una interfaz que la capa de presentación pueda utilizar para acceder y manipular los datos**. Esta capa se encarga de procesar las solicitudes recibidas desde la capa de presentación, realizar las operaciones requeridas y devolver los resultados correspondientes.

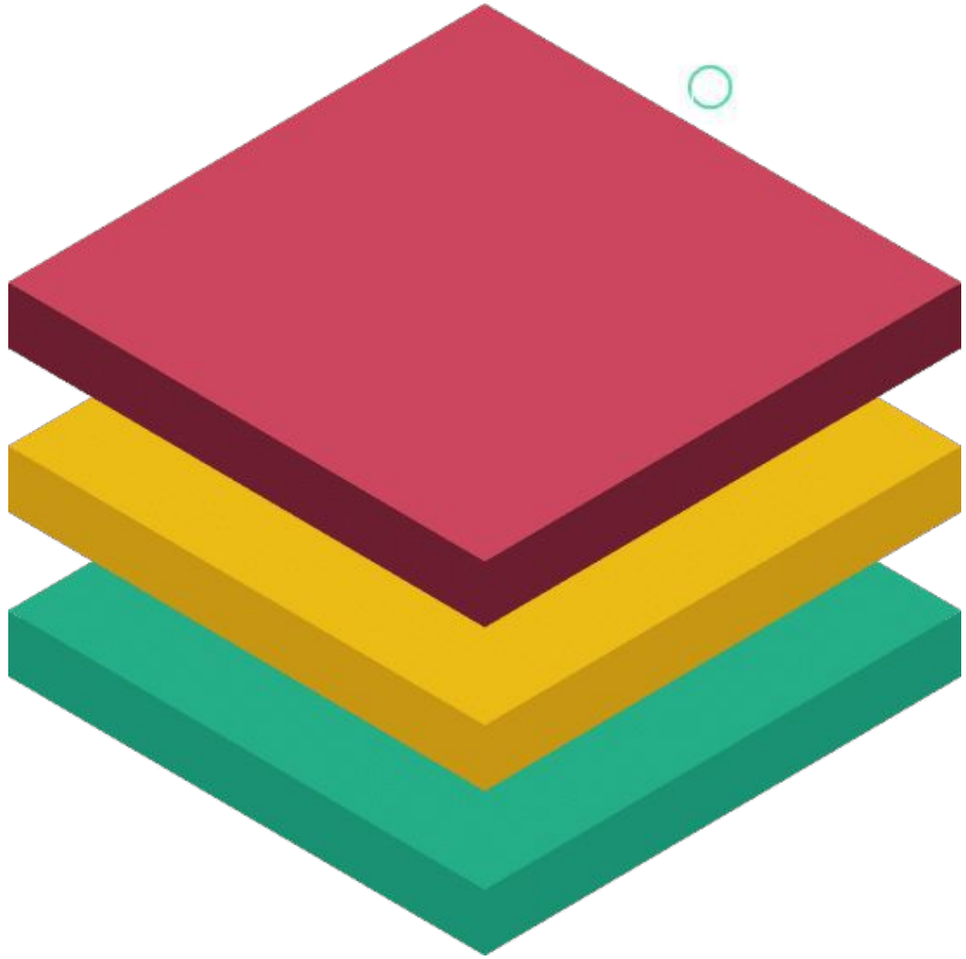


Capa de Servicios

Algunas de las **responsabilidades** típicas de la capa de servicio son:

Validación de datos: La capa de servicio valida los datos ingresados por el usuario antes de realizar cualquier operación en la capa de acceso a datos. Esto garantiza que los datos sean consistentes y cumplan con las reglas de negocio establecidas.

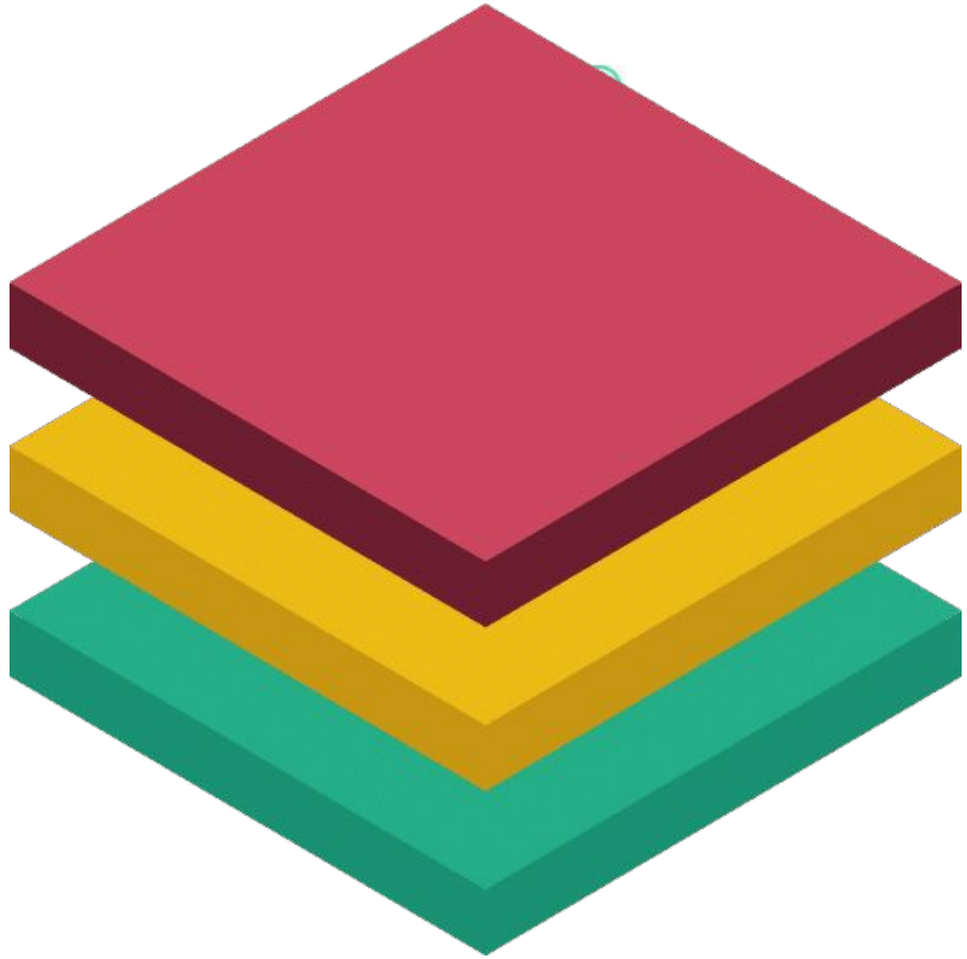
Lógica de negocio: Implementa la lógica del negocio de la aplicación. Esto puede incluir cálculos complejos, verificaciones de seguridad, reglas de flujo de trabajo, procesamiento de pagos, integraciones con servicios externos, entre otros.



Capa de Servicios

Coordinación de la capa de acceso a datos: La capa de servicio interactúa con la capa de acceso a datos para realizar operaciones de lectura y escritura en la base de datos u otros sistemas de almacenamiento. Puede orquestar múltiples operaciones de acceso a datos y garantizar la integridad de los datos.

Transacciones y gestión de errores: La capa de servicio puede administrar transacciones, asegurando la integridad de los datos en operaciones que requieren cambios en varias entidades o tablas. También puede manejar excepciones y errores, proporcionando mecanismos para recuperarse de fallos y comunicar mensajes de error adecuados a la capa de presentación.



LIVE CODING

Ejemplo en vivo

Wallet Service:

Las clases de servicio deberán tener su propio paquete en nuestro proyecto. Vamos a continuar el diseño de la AlkeWallet creando el paquete ‘servicios’ en la carpeta src/main/java.

Además, vamos a crear una clase llamada ‘UsuarioServicio’ que manipulará la lógica de los usuarios.

Tiempo: 10 minutos

LIVE CODING

Ejemplo en vivo

1. *Crear el paquete .service*
2. *Agregar al paquete la clase UsuarioServicio*

Creando un servicio utilizando anotaciones

Creando un servicio utilizando anotaciones

@Service es una de las anotaciones más habituales de Spring Framework . Se usa para construir una clase de Servicio que habitualmente se conecta a varios repositorios y agrupa su funcionalidad.

Veamos el ejemplo de la clase que creamos recién, "UsuarioServicio", que implementa la lógica de negocio para manejar operaciones relacionadas con usuarios.

Creando un servicio utilizando anotaciones

Para convertir esta clase en un servicio utilizando anotaciones deberías hacer lo siguiente:

1- Anotar la clase: Agrega la anotación @Service en la clase para indicar que es un servicio.

```
3  import org.springframework.stereotype.Service;
4
5  @Service
6  public class UsuarioServicio {
7
8  }
9
```

Creando un servicio utilizando anotaciones

2- Inyectar dependencias: Si el servicio necesita utilizar otras clases o componentes, puedes inyectarlos utilizando la anotación @Autowired.

En este ejemplo, se inyecta un objeto Usuario en el servicio. La anotación @Autowired se encargará de proporcionar automáticamente una instancia de Usuario cuando se cree una instancia del servicio.

```
8  @Service
9  public class UsuarioServicio {
10
11     @Autowired
12         private Usuario usuario;
13 }
14
```

LIVE CODING

Ejemplo en vivo

Wallet Service:

- *Vamos a anotar la clase 'UsuarioServicio' con @Service.*

- *Además, debemos anotar con @Autowired un atributo del tipo Usuario. De ésta manera, podemos colocar los métodos de creación de usuarios en este servicio.*

Tiempo: 20 minutos

LIVE CODING

Ejemplo en vivo

1. *Anotar como @Service la clase UsuarioServicio*
2. *Inyectar una instancia de Usuario con @Autowired*
3. *Generar la lógica de **creación** de usuarios en la clase UsuarioServicio.*

Momento:

Time-out!

⏰5 min.

Inyectando el servicio a un controlador para su utilización

Inyectando el servicio a un controlador para su utilización

Puedes utilizar el servicio en otras partes de tu aplicación inyectándolo de manera similar a como se hizo con las dependencias. Por ejemplo, puedes inyectar el servicio en un controlador utilizando la anotación @Autowired:

```
 8 @Controller
 9 public class UsuarioControlador {
10
11     @Autowired
12     private UsuarioServicio usuarioServicio;
13 }
```

Inyectando el servicio a un controlador para su utilización

Puedes utilizar el servicio en otras partes de tu aplicación inyectándolo de manera similar a como se hizo con las dependencias. Por ejemplo, puedes inyectar el servicio en un controlador utilizando la anotación @Autowired:
En este caso, el servicio UsuarioServicio se inyecta en el controlador UsuarioControlador para poder utilizar sus métodos y lógica de negocio.

```
 8 @Controller
 9 public class UsuarioControlador {
10
11     @Autowired
12     private UsuarioServicio usuarioServicio;
13 }
```

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

Wallet Service:

- *Ahora vamos a inyectar un objeto del tipo UsuarioServicio en el controlador. De esta manera podremos utilizar las llamadas a los métodos CRUD del servicio desde el controlador.*

- *Además, debemos crear un método en el servicio que retorne un String con el nombre del usuario.*

Tiempo: 20 minutos

LIVE CODING

Ejemplo en vivo

1. *Inyectar una instancia de UsuarioServicio en el controlador de usuarios.*
2. *Generar un método denominado “respuesta” que retorne el nombre del usuario en la clase UsuarioServicio.*
3. *Llamar al método ‘respuesta’ desde el controlador utilizando la instancia de usuarioServicio.*
4. *Mostrar el nombre por pantalla.*

¿Alguna consulta?

RESUMEN

¿Qué logramos en esta clase?

- ✓ *Aprender la implementación de la capa de Servicios*
- ✓ *Comprender la inyección de servicios en los controladores*

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 30 - 33*
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