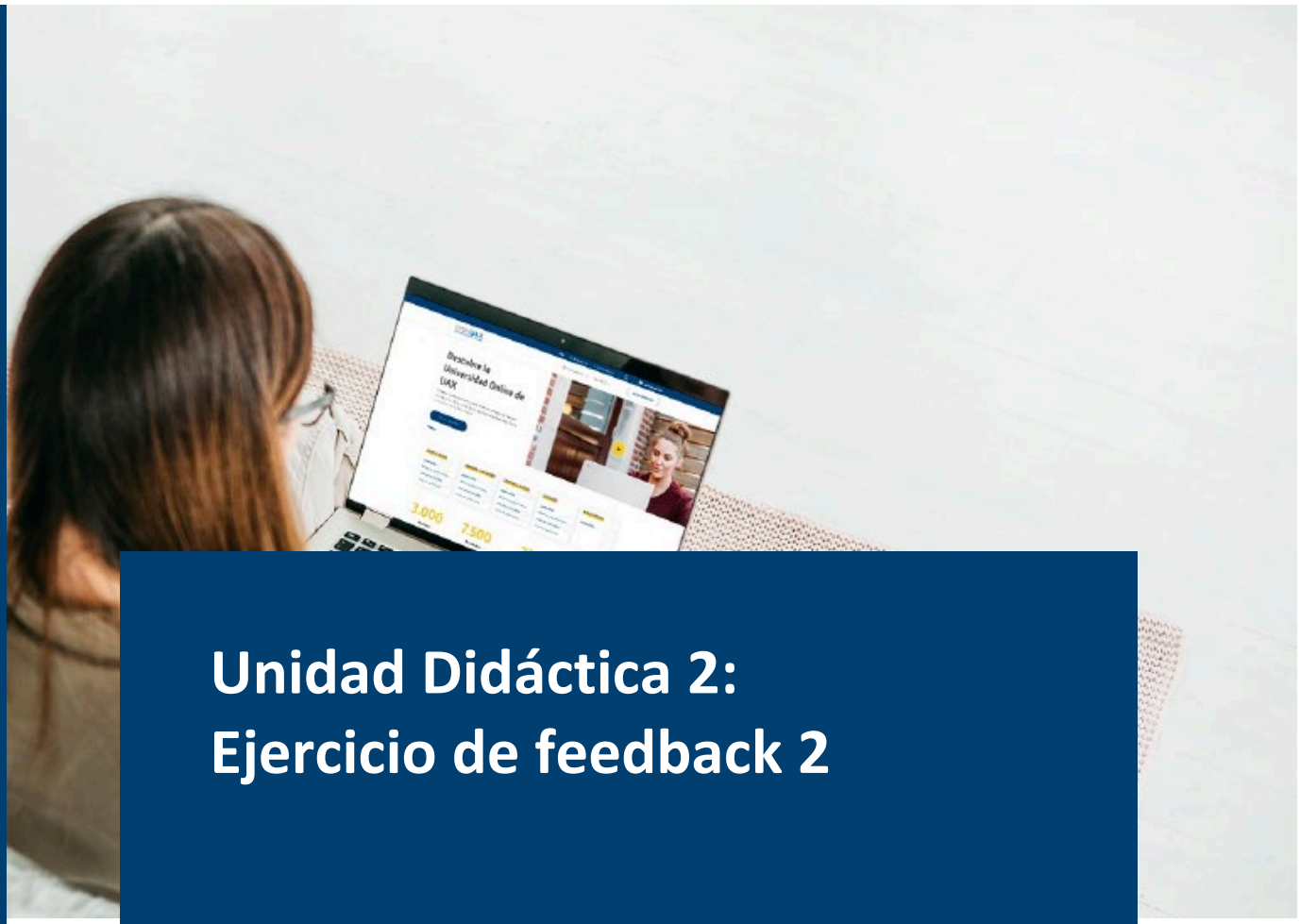

Unidad Didáctica 2: Ejercicio de feedback 2

Programación Concurrente

1. Enunciado

Contexto General: Usted forma parte de un equipo de desarrollo que trabaja en un proyecto de alto nivel de complejidad, encargado de integrar múltiples sistemas para monitorear y gestionar eventos en tiempo real. Estos sistemas abarcan desde la gestión de pedidos hasta el monitoreo de dinosaurios y misiones espaciales, combinando tecnologías como Spring, microservicios, y programación reactiva.

Enunciado del Proyecto:

La empresa donde trabaja está desarrollando un **Sistema de Monitoreo y Gestión Avanzada (SMGA)** que integra varios subsistemas independientes, cada uno especializado en un área crítica:

1. **Gestión de Pedidos en Tiempo Real:** Implementado con Spring y optimizado para transacciones concurrentes y la gestión de bases de datos utilizando HikariCP y @Transactional.
2. **Monitoreo de Dinosaurios en Jurassic Park:** Utiliza programación reactiva con Spring WebFlux para manejar grandes volúmenes de datos de sensores en tiempo real.
3. **Misión en Marte:** Un sistema de procesamiento por lotes con Spring Batch para analizar grandes volúmenes de datos de sensores en tiempo real.
4. **Monitoreo y Gestión de Hechizos en el Mundo Mágico:** Basado en microservicios y Spring Cloud, este sistema maneja eventos mágicos asegurando la resiliencia y la eficiencia del sistema.

El objetivo es integrar estos subsistemas en un único sistema robusto y escalable que sea capaz de manejar grandes volúmenes de datos, asegurar la consistencia transaccional, y responder eficientemente a eventos críticos en tiempo real.

Tareas del Proyecto:

1. **Diseño de la Arquitectura Integrada:**
    - Describa cómo integrar cada uno de los subsistemas en el SMGA. Considere aspectos de interoperabilidad, resiliencia, y escalabilidad.
    - Cree un diagrama de arquitectura que muestre la relación entre los diferentes subsistemas.
2. **Desarrollo y Optimización del SMGA:**

- **Gestión de Pedidos en Tiempo Real:** Implemente mejoras en la gestión de transacciones y bases de datos para manejar concurrencia utilizando Spring y HikariCP.
  - **Fragmento de Código de Ejemplo:**

```
java
Copiar código
@Transactional(rollbackFor = Exception.class)
public void updateOrder(Order order) throws Exception {
   orderRepository.save(order);
   if (someConditionFails()) {
      throw new Exception("Simulated error");
   }
}
```

- **Monitoreo de Dinosaurios:** Implemente flujos de datos reactivos utilizando WebFlux para asegurar una respuesta rápida a eventos de sensores.
  - **Fragmento de Código de Ejemplo:**

```
java
Copiar código
@GetMapping("/dinosaur-data")
public Flux<Integer> getDinosaurData() {
   return dinosaurMonitoringService.getDinosaurData();
}
```

- **Misión en Marte:** Optimice el procesamiento por lotes para manejar grandes volúmenes de datos utilizando Spring Batch.
  - **Fragmento de Código de Ejemplo:**

```
java
Copiar código
@Bean
```

```
public Step step1(ItemReader<SensorData> reader, ItemProcessor<SensorData, ProcessedSensorData> processor, ItemWriter<ProcessedSensorData> writer) {
   return stepBuilderFactory.get("step1")
         .<SensorData, ProcessedSensorData>chunk(10)
         .reader(reader)
         .processor(processor)
         .writer(writer)
         .build();
}
```

- **Monitoreo y Gestión de Hechizos:** Utilice microservicios y patrones como Circuit Breaker y API Gateway para asegurar la resiliencia del sistema.
  - **Fragmento de Código de Ejemplo:**

```
java
Copiar código
@HystrixCommand(fallbackMethod = "defaultSpell")
public String castSpell(String spellName) {
   return restTemplate.getForObject("http://spell-service/cast?spell=" + spellName, String.class);
}


public String defaultSpell(String spellName) {
   return "The spell " + spellName + " is currently unavailable. Please try again later.";
}
```

3. **Implementación de la Integración:**
   - Desarrolle un módulo central que gestione la comunicación entre los subsistemas.
   - Asegure la consistencia de datos y la sincronización entre los diferentes módulos.

- Implemente un mecanismo de logging y monitoreo centralizado para detectar y reaccionar ante posibles fallos.

4. **Pruebas y Validación del SMGA:**
   - Realice pruebas unitarias e integrales que validen la correcta operación de cada subsistema dentro del contexto del SMGA.
   - **Ejemplo de Prueba Unitaria:**

java

Copiar código

```
@Test
public void testCastSpell() throws Exception {
   mockMvc.perform(MockMvcRequestBuilders.get("/cast?spell=Lumos")
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(content().string("The spell Lumos is cast successfully."));
}
```

5. **Análisis Post-Implementación:**
   - Realice un análisis detallado del rendimiento del sistema tras la integración.
   - Compare el rendimiento y la capacidad de respuesta del SMGA antes y después de la integración, utilizando métricas específicas (tiempos de respuesta, tasa de éxito en la transmisión de datos, etc.).
   - Redacte un informe que incluya gráficos y estadísticas para mostrar la mejora lograda.

2. Instrucciones de entrega

- Formato: PDF con repositorio de Github
- Extensión máxima: 10 folios
- Incluir nombre del fichero: "EjercicioFeedback_NombreApellido.pdf"

UAX Universidad Alfonso X el Sabio

G R A C I A S

UAX.COM