Yalbani Aranda

PROYECTO FINAL

Este es un sistema modular para gestionar la participación de un invitado en un congreso de lunes a domingo, con un total de aprox. 100 horas. Este sistema intenta simular un entorno empresarial real donde se utilizan diversas tecnologías de manera compleja.

Objetivos del sistema:

La participación se contabilizará con el uso de una llave electrónica que registrará a que eventos ha entrado del congreso. Dependiendo de los eventos a los que asista, el participante ira acumulando puntos que puede intercambiar por freebies al finalizar el congreso.

Con una cantidad específica de puntos (25), se otorgará una constancia de asistencia.

Con una suma de cierta cantidad de puntos (30), el participante podrá ingresar a eventos especiales.

El progreso será notificado a los participantes por mensaje cada día del congreso.

Tecnologías Usadas:

- Spring Boot (3.5.6)
- Maven
- Spring Rest
- SpringData JPA
- Lombok
- MongoDB (8)
- MySQL
- Spring Batch
- Spring Modulith (1.3.0)
- Swagger (2.7.0)
- Testing (JUnit 5 y Mockito)
- Reportes automáticos con JaCoCo

Cada paquete o modulo mantiene la arquitectura modular con comunicación orientada a eventos, APIs REST y Maven para la gestión de dependencias y plugins.

La base de datos es hibrida (relacional MySQL y documental con MongoDB).

Los procesos se automatizan con la implementación de Spring Batch.

El testing es integral utilizando JUnit y Mockito, para buscar un coverage mínimo de 85%.

Se creó una conexión a MySQL a una base de datos llamada Congreso, con una conexión a un Contenedor en Docker llamado mysql-container2 por el puerto 3307. MongoDB funciona en el puerto 27017, también conectado a Docker.

Tomcat funciona utilizando el puerto 8080 (http) en localhost.

Estrategia de Datos Híbrida

MySQL (Datos Relacionales)

- Información de participantes
- Relaciones entre entidades
- Acumulación de puntos
- Creación de Constancias
- Reportes

MongoDB (Datos Documentales)

- Logs de acumulación y reducción de puntos
- Datos de Freebies y Conferencias manualmente ingresados a partir de 3 csv almacenados en resources. Se cargan de manera automática a la base de datos CongresoIntMongo
- Datos de movimientos de puntos
- Datos de logros desbloqueados

Csv

PointsDataInitializer y PointsDataLoader sirven para cagar manualmente los datos de Conferencias y del inventario de Freebies.

ARQUITECTURA

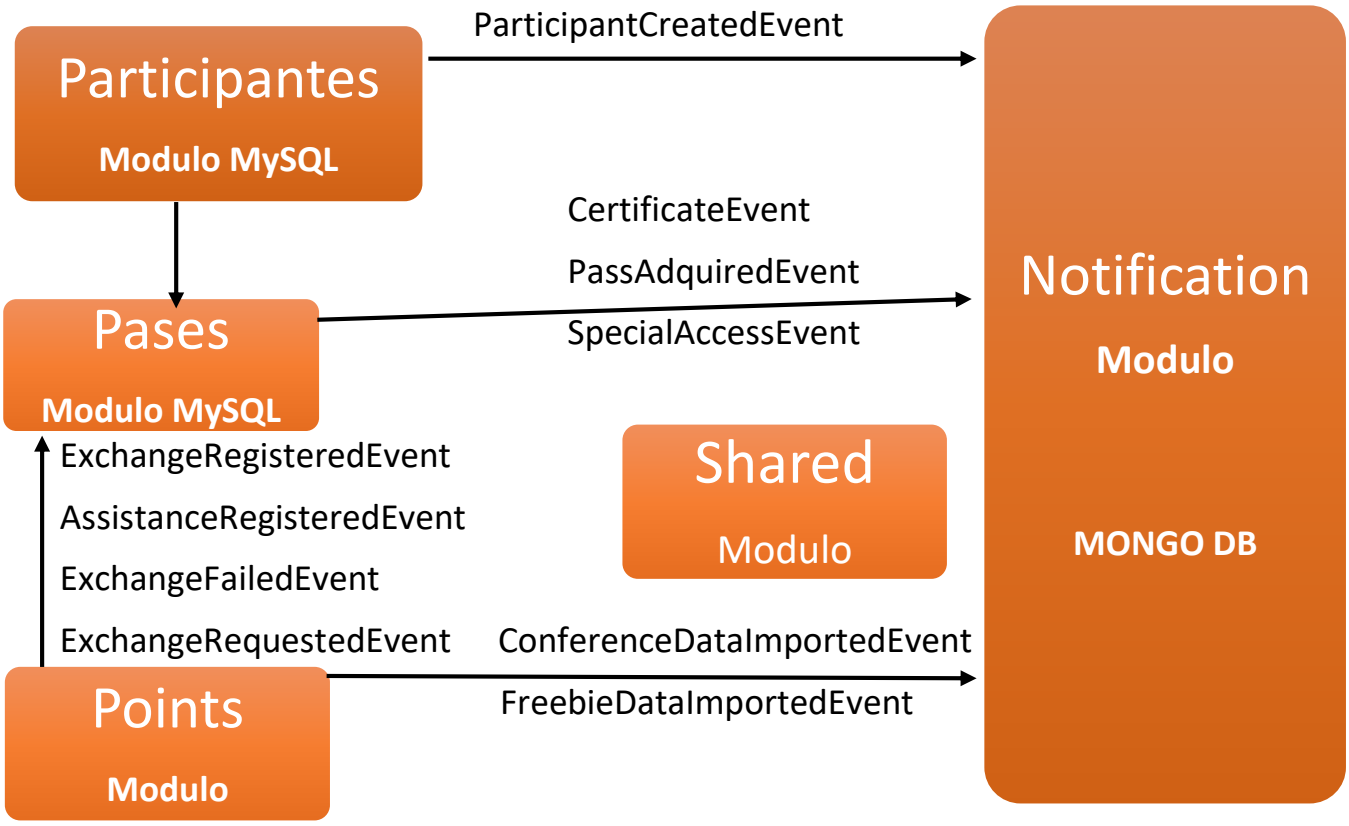
- Estructura del Proyecto

com.proyecto.congreso/

├── **notification/**
│ ├── controller/
│ ├── repository/
│ └── model/
│ ├── service/
├── **participantes/**
│ ├── controller/
│ ├── dto/
│ ├── events/
│ └── model/
│ ├── repository/
│ ├── service/
├── **pases/**
│ ├── controller/
│ ├── dto/
│ ├── events/
│ └── model/
│ ├── repository/
│ ├── service/
├── **points/**
│ ├── assistance/
│ ├── calculator/
│ ├── exchange/
│ ├── service/
└── **shared/**

Yalbani Aranda

DIAGRAMA MODULAR



MODULOS

ANOTACIONES EN MAIN

Aplicación Principal Del Sistema De Congreso Interactivo

@Modulithic: Habilita Spring Modulith para arquitectura modular

@EnableAsync: Permite procesamiento asíncrono de eventos

@EnableJpaRepositories: Configuración de repositorios JPA (MySQL)

@EnableMongoRepositories: Configuración de repositorios MongoDB

@SpringBootApplication

OTRAS ANOTACIONES

@Service contiene la lógica de negocio y se encarga de realizar operaciones específicas, como interactuar con la base de datos o procesar datos.

@Controller permite coordinar el flujo de trabajo. Recibe las solicitudes del usuario y decide qué hacer con ellas, como llamar a un servicio.

@Repository permite interactuar con la base de datos y almacenar y recuperar datos de manera organizada.

SERVICE y CONTROLLER funcionan principalmente para operaciones CRUD.

MODULO SHARED

- GlobalExceptionHandler. Maneja IllegalArgumentException (errores de negocio). Ejemplos: * - Pass no encontrado
  * - Conferencia no existe
  * - Asistencia duplicada
- MongoConfig. Configuración de MongoDB cuando MongoTemplate está disponible.
- OpenApiConfig. Configuración de OpenAPI/Swagger para la documentación de la API.

MODULO NOTIFICATION

- Notification funciona con:
  - Operaciones CRUD
  - Operaciones Query
  - Operaciones de negocio de Pases y de Participantes
- MovementPoints funciona de manera similar a Notification, pero enfocada a movimientos de puntos add y use en consola. Tambien escucha los eventos relacionados con agregar y disminuir puntos.

MODULO PARTICIPANTES

Gestiona la información personal de participantes, se mantiene desacoplada del método Pases para mantener la modularidad. Necesaria para la autorización de la Constancia que otorga un Certificado de Asistencia al Congreso.

Yalbani Aranda

MODULO PASES

En el módulo **Pases**, PassPointsEventHandler es el encargado de escuchar eventos e **asistencias** registradas y actualiza los puntos del Pass. Utiliza @Transactional para garantizar consistencia. Al no “conocer” el módulo de points, se asegura el bajo acoplamiento entre ambos módulos.

FLUJO DEL PROCESO:

* 1. En el módulo Points, al registrar una asistencia AssistanceService publica un AssistanceRegisteredEvent

* 2. Este *handler*(PassPointsEventHandler) lo escucha de forma asíncrona

* 3. Busca el Pass en MySQL

* 4. Suma los puntos

* 5. Verifica logros (certificado, acceso especial) y publica CertificateAchievedEvent y SpecialAccessEvent (si aplica)

* 6. Guarda la actualización del Pass en MySQL

* 7. Notification escucha y envía email

De manera similar funciona escuchando el evento FreebieStockReservedEvent, solo que, en vez de aumentar, **reduce** puntos.

FLUJO DEL PROCESO:

* 1. Points (ExchangeService) publica ExchangeRequestedEvent

* 2. Points (FreebieStockHandler) verifica stock y publica FreebieStockReservedEvent

* 3. Pases (PassPointsEventHandler) descuenta puntos

* 4. Si algo falla al intercambiar el Freebie, se publica ExchangeFailedEvent

* 5. FreebieStockHandler revierte stock

MODULO POINTS

En el módulo Points, se encuentra toda la información relacionada con la suma y reducción de puntos, para mantener lo más desacoplado posible la arquitectura del proyecto.

En la carpeta DTO se encuentra información relevante para transportar datos entre los steps del Batch Job. También incluye DTO de respuesta para consultas de certificados, ya que, al alcanzar los 25 puntos, en MySQL crea un registro permanente del logro del participante. Se crea utilizando un método Factory.

AssistancePointsData es procesado por el Reader, que lee la asistencia directamente; el Processor, que crea el movimiento a partir de Asistencia + Pass; y el Writer consume el movimiento para actualizar MySQL y MongoDB. Assistance también tiene una Request y un Response para definir los datos relevantes durante su procesamiento.

De igual manera hay un ExchangePointsData que maneja de misma manera los puntos, pero en sentido inverso. Los va reduciendo.

Para asistencia y Exchange, se creó un controller para hacer las simulaciones manualmente en Postman y en Swagger.

EVENTOS

- ParticipantCreatedEvent. Se notifica al participante cuando haya completado su inscripción al Congreso.
- PassAdquiredEvent. Se notifica al participante cuando haya adquirido su pase General, o ALL INCLUDED para obtener comida y transporte.
- FreebieDataImportedEvent también es record.
- ConferenceDataImportedEvent
- AssistanceRegisteredEvent - Evento que representa la acumulación de puntos por asistencia a conferencias. Este evento se publica cuando un participante asiste a una conferencia y debe recibir puntos en su Pass.
- PassPointsEventHandler es el *Handler* central para el módulo de Pases, ya que Escucha eventos de otros módulos y actualiza el estado de los Pases.
- ExchangeRequestedEvent
- ExchangeRegisteredEvent - Cuando se intercambian los puntos por freebies se lanza el evento y los puntos del Pass se restan, asi como el stock del freebie.
- ExchangeFailedEvent lanza el evento de fallo en intercambios y revierte el stock si es necesario
- SpecialAccessEvent – Se notifica al participante cuando haya desbloqueado el acceso especial. HandleSpecialAccessAchieved Escucha el evento y crea el registro en MySQL. Este Evento se maneja como *record*, publicado cuando un Pass alcanza los puntos necesarios para acceso especial (30 puntos). Es record para que sea

inmutable, y genera Constructor, getters, equals, hashCode y toString automáticos

- CertificateEvent. Evento publicado cuando un Pass alcanza los puntos necesarios para certificado (25 puntos). Se notifica al participante cuando haya alcanzado su certificado, y se genera en la base de datos para el registro interno del congreso y posterior impresión. HandleCertificateAchieved Escucha el evento y crea el registro en MySQL. El CertificateEventHandler escucha este evento para crear el registro en la base de datos.

CONCEPTOS IMPORTANTES UTILIZADOS

- Arquitectura y Diseño.
  Se utilizó una Arquitectura Orientada a Eventos, con el manejo de los mismos utilizando Eventos para comunicación, con sus Publishers y Listeners.
  Las características fundamentales (Funcionalidades Core) permiten que funcione correctamente como los cimientos de una casa. Se utilizaron Interfaces con múltiples implementaciones para ayudar al desacoplamiento, asi como Strategy Pattern implementado, que permite elegir entre diferentes estrategias, dependiendo de la situación. Esta aplicación tiene muchas decisiones que tomar en todo el proceso y el polimorfismo ayuda con esas estrategias de cálculo (add y use points).
- Inyección de dependencias. Se utilizó Constructor Injection en Todos los Services, es decir, las dependencias se pasan a través del constructor de la clase, asegurando que cada servicio tenga todo lo que necesita desde el principio. Esta manera de inyectar dependencias, permite que las clases las reciban en lugar de crearlas. Esto facilita el mantenimiento y las pruebas. La configuración con Anotaciones Spring, el uso de Maven y el Manejo de Beans permitieron una construcción limpia del proyecto, así como su fácil manejo.
- Spring Framework. El uso de Spring permite la integración de REST, Spring Core, Data JPA, Spring Batch, DTOs y Validaciones, así como la implementación de Swagger permite un fácil manejo de errores y excepciones, repositorios personalizados, Queries personalizados y agregaciones y operaciones complejas de manera armónica que combinan múltiples datos o entidades para obtener resultados más completos en la gestión del proyecto.
- Testing. JUnit y Mockito permitieron la implementación de tests unitarios completos en la funcionalidad y en la lógica de negocio. Se agregan para realizar

pruebas y medir la cobertura del código de manera aislada para probar cada unidad (tests unitarios), para comprobar que se llamaron ciertos métodos y definir comportamientos específicos con Mocks (stubbing o verification), o para comprobar como diferentes partes de la aplicación funcionan juntas (testing de integración). El testing se documentó directamente en el código y se alcanzó un coverage de 87% general.

IMPLEMENTACION DE SPRING BATCH

Se implementó Spring Batch para procesar los datos del congreso de manera automatizada y ordenada, ya que se espera una asistencia de 300 participantes, los cuales van a estar ingresando aproximadamente al mismo tiempo a las conferencias, agregando puntos a sus Pases de inscripción, que funcionan como una cuenta donde acumular puntos.

Se maneja con 2 JOB. Por el momento va a ser mediante un POST manual. Utiliza repositorios en Mongo y en MySQL.

EL primer **JOB**(pointsJob) que se va a utilizar se va a disparar cada que el participante acumule una asistencia a una conferencia. El **STEP 1**. Va a calcular los puntos que se van a agregar. El Reader (MySQL) va a usar un Processor(calcula) y el Writer lo va a devolver a MySQL con el contexto de SpringBatch.

Ese contexto va a ser el **Reader** en el STEP 2, que va a publicar los eventos, que puede ser escuchado por un Event Listener en MongoDB, que registre la información de los movimientos de puntos.

La implementación de las interfaces, permite utilizar la lógica de la Fábrica de cálculo como **Processor** de Puntos. Utiliza una lógica sencilla de agregar y utilizar puntos. Finalmente, el **Writer** actualiza la información en MySQL y utiliza el contexto para llamarlo inmediatamente como Reader del **STEP 2** y publica la información del evento lanzado. En MySQL se crean y actualizan automáticamente las tablas de instancias, ejecuciones y contexto de los Jobs, así como los parámetros de ejecución y la ejecución y contexto de los steps.

Yalbani Aranda

IMPLEMENTACION DE SPRING MODULITH

Para ayudar con el bajo acoplamiento, se implementó Modulith para dividir la aplicación en módulos que se comunican entre sí mediante eventos. Para que entre módulos no se conozcan entre sí, que no sepan nada del otro modulo.

El manejador de eventos de Spring Modulith permite escuchar el evento, la lógica de acumulación de puntos, y con el desacoplamiento de servicios se mantiene el código en orden. Al publicar un evento, olvidamos quien escucha. El trabajo se hace de manera asíncrona, pero no influye con los otros módulos directamente.

CASOS DE USO DEL NEGOCIO

Gestión de Participantes en un Congreso y registro con email, conectado a notificaciones. Estados del participante (activo/inactivo), creación de una inscripción al congreso a través de un Pase. Búsquedas, filtros y validaciones de consulta de puntos, manejo de ellos, programación del congreso, asistentes en conferencias, liberación de conferencias, stock de freebies para regalar en el congreso, etc.

PUNTOS A MEJORAR:

Implementar una llamada REST para obtener el participanteID en la asistencia. Se usaría REST en lugar de inyectar PassRepository para mantener el bajo acoplamiento entre módulos.

Usar un lector de CSV para cargar los datos externos.

Escalarlo de simulación de asistencias, intercambios y notificaciones a escenarios reales.