Trabajo Practico 2 Armado de Recital

Grupo iota

Integrantes:

Quimey Salinas

Ingeniería en Informática

qsalinasgarin657@alumno.unlam.edu.ar

Franco Checcacci

Ingeniería en Informática

fchecacci@alumno.unlam.edu.ar

Profesores:

Federico Gasior

Hernán Lanzilotta

Lucas Videla

Verónica Aubin

I. CONTEXTO

En la organización de un recital, cada canción requiere la participación de distintos roles, como guitarristas, voces principales, bajistas, bateristas u otros roles especializados. Los artistas base del evento pueden cubrir algunos de estos roles, pero con frecuencia resulta necesario contratar artistas adicionales para completar la formación requerida por cada canción.

La contratación de artistas externos implica costos y restricciones específicas: cada artista tiene habilidades particulares, pertenece o no a determinadas bandas, posee un costo por presentación y solo puede participar en un número limitado de canciones. Además, puede haber roles que no cubra ningún artista de los contratables y por ende, se deba realizar un entrenamiento a los mismos.

II. INTRODUCCIÓN

El presente informe describe el desarrollo de un sistema pensado para organizar un recital compuesto por varias canciones, donde cada una requiere distintos roles musicales. La idea principal es poder asignar artistas a esos roles respetando todas las restricciones, y al mismo tiempo mantener el costo total de contratación lo más bajo posible. Para eso se modelaron artistas, bandas, roles históricos, límites de disponibilidad y toda la información necesaria para que el sistema pueda decidir qué artista conviene asignar en cada caso.

El trabajo se estructuró siguiendo el enfoque de programación orientada a objetos: se definieron las clases del dominio (Artista, Canción, Recital, AsignacionRol) y luego se crearon servicios encargados de manejar las reglas más complejas del problema (ContratacionService, CalculoEntrenamientos). La parte más pesada del sistema es la contratación general de artistas para todas las canciones a la vez, se resolvió implementando un algoritmo de **Mín-Cost Máx Flow**, que permite encontrar una asignación completa respetando las capacidades y minimizando el costo final.

Además del módulo de optimización, el proyecto incluye funciones más directas como calcular los roles faltantes por canción, verificar qué cubre la base, controlar los límites de cada artista y manejar el entrenamiento cuando corresponde. Todo esto se apoya en datos externos cargados desde archivos JSON, lo que permite cambiar artistas y canciones sin modificar el código.

III. MÉTODOS

Para llevar adelante el proyecto se trabajó con un enfoque bastante directo: primero modelar bien los objetos principales (Artista, Canción, Recital) y después construir los servicios que operan sobre esos modelos. La lectura de los archivos JSON fue el punto de partida, ya que sin esos datos el sistema no tiene nada que procesar. Una vez cargados los artistas, el recital y los artistas base, el programa muestra un menú de opciones donde puede empezar a calcular los roles faltantes, entrenar artistas, intentar las contrataciones, etc.

En cuanto a la contratación masiva o general, que es la parte más pesada del trabajo, se implementó un algoritmo de mínimo costo con máximo flujo (MinCostMaxFlow) en ContratacionService. La idea fue armar un grafo donde cada artista es un nodo, cada rol requerido es otro nodo, y se conectan con aristas de capacidad 1 y un costo equivalente al costo real del artista. Después se ejecuta SPFA para encontrar caminos de menor costo e ir ampliando el flujo siempre que se pueda.

Para los casos más simples, como asignar artistas base, revisar roles faltantes, validar límites de canciones por artista, se usó lógica normal sin algoritmos raros. Las verificaciones incluyen: controlar si ya está asignado a cierta canción, verificar si no superó el máximo de canciones, y ver si requiere entrenamiento (y si se puede).

El entrenamiento también se maneja dentro del sistema, con la restricción de que no se puede entrenar a artistas base ni a artistas que ya estén contratados. El costo del artista sube un 50% cada vez que obtiene un rol nuevo por entrenamiento.

Las pruebas unitarias (JUnit) cubren las funciones principales: cálculo de roles faltantes, asignación manual de artistas, entrenamiento y la contratación completa con el algoritmo de flujo.

Por último, se decidió implementar una nueva funcionalidad que, al momento de realizar la contratación masiva, detecte los roles faltantes y de la posibilidad de realizar el entrenamiento de aquellos roles detectados. Esto mejora drásticamente la experiencia del usuario ya que, desde la misma opción de menú, puede visualizar los roles que faltan asignar y puede decidir que artistas debe entrenar para dichos roles (también los puede omitir).

IV. RESULTADOS Y OBJETIVOS

El sistema terminó resolviendo correctamente la contratación de artistas para un recital completo, respetando todos los límites que pide el dominio (roles, historial, maxCanciones, descuentos, etc.). Se puede consultar qué roles faltan por canción o en todo el recital, y el sistema es capaz de asignar automáticamente a los artistas base cuando corresponde. El algoritmo de flujo logró efectivamente minimizar el costo total de contratación en las pruebas, lo cual era uno de los puntos críticos del TP.

También funciona adecuadamente la parte de entrenar artistas, teniendo en cuenta las restricciones de que no se puede entrenar a un artista base, ni a uno ya contratado.

El objetivo principal, el cual era modelar la problemática completa con orientación a objetos, cargar datos externos, operar con colecciones y diseñar un sistema funcional, se cumplió. A futuro, si el proyecto siguiera, se podría mejorar un poco la interacción con el usuario y agregar más validaciones, o incluso optimizar la parte del algoritmo para manejar recitales más grandes sin perder eficiencia. También se podría mejorar algunas opciones de menú, ya que, en algunas el código puede ser mejorable u optimizable. Pero para los requisitos del TP, el proyecto está funcionando correctamente.

V. CONCLUSIONES

El trabajo permitió construir un sistema que resuelve un problema realista sin volverse imposible de mantener. La clave fue separar bien las responsabilidades: las clases representan el dominio y los servicios hacen el trabajo "pesado". La parte más compleja terminó siendo la contratación global, donde el algoritmo de min-cost max-flow

disminuyó la complejidad, porque si intentábamos hacerlo a mano o con lógica más simple directamente no daba los resultados esperados.

Durante el desarrollo quedó claro que muchas complicaciones vienen más de las restricciones del dominio que del código en sí: un artista que tiene límite de canciones, otro que no puede ser entrenado, el descuento por bandas compartidas, etc. Todo eso había que encadenarlo bien para que no se rompa nada.

En líneas generales el sistema cumple lo que la consigna pedía, maneja las opciones de menú solicitadas, incluyendo las contrataciones, entrenamientos, muestra de datos, exportación del estado actual, importación de un estado previo, entre otras funcionalidades, además, también incluimos datos de origen para tener distintos casos de prueba. Si bien se podría mejorar la eficiencia o la interfaz, el objetivo está completamente logrado.

VI. REFERENCIAS

Referencias bibliográficas utilizadas:

[1] FasterXML. Jackson JSON Processor Documentation: https://github.com/FasterXML/jackson

[2] Minimum Cost Maximum Flow, concepto base para resolver asignaciones con costo mínimo respetando restricciones de capacidad.

[3] Documentación oficial de SWI-Prolog – JPL (para integración con Java): https://www.swi-prolog.org/pldoc/package/jpl

[4] Stack Overflow, “Error: org.jpl7.JPLException: Unsupported blob type passed from Prolog,” 2021: https://stackoverflow.com/search?q=unsupported+blob+type+passed+from+Prolog