Desafio Tecnico “Susaeta Ediciones” para Programador Jr

Marvin Leonel García Lémus

leogarcia0027@gmail.com

La siguiente documentación presenta la solución al desafío técnico proporcionado por el Ingeniero Juan Castillo

Detallando la solicitud:
Desafío Técnico

Nuestra plataforma ofrece libros digitales escolares junto con recursos complementarios como quizzes, presentaciones, PDFs, audios y videos. Los usuarios canjean un libro en la plataforma y obtienen acceso a su contenido de manera digital.

Queremos optimizar nuestro sistema de recomendaciones para sugerir el próximo libro más relevante para cada usuario según su actividad.

Problema

Supongamos que tienes un catálogo de N libros digitales, donde cada libro tiene:

- Título
- Autor
- Lista de categorías (Ej: Matemáticas, Historia, Ciencias)
- Número de veces canjeado
- Tiempo promedio de lectura
- Cantidad de recursos digitales utilizados (quizzes, PDFs, audios, videos, etc.)

Objetivo

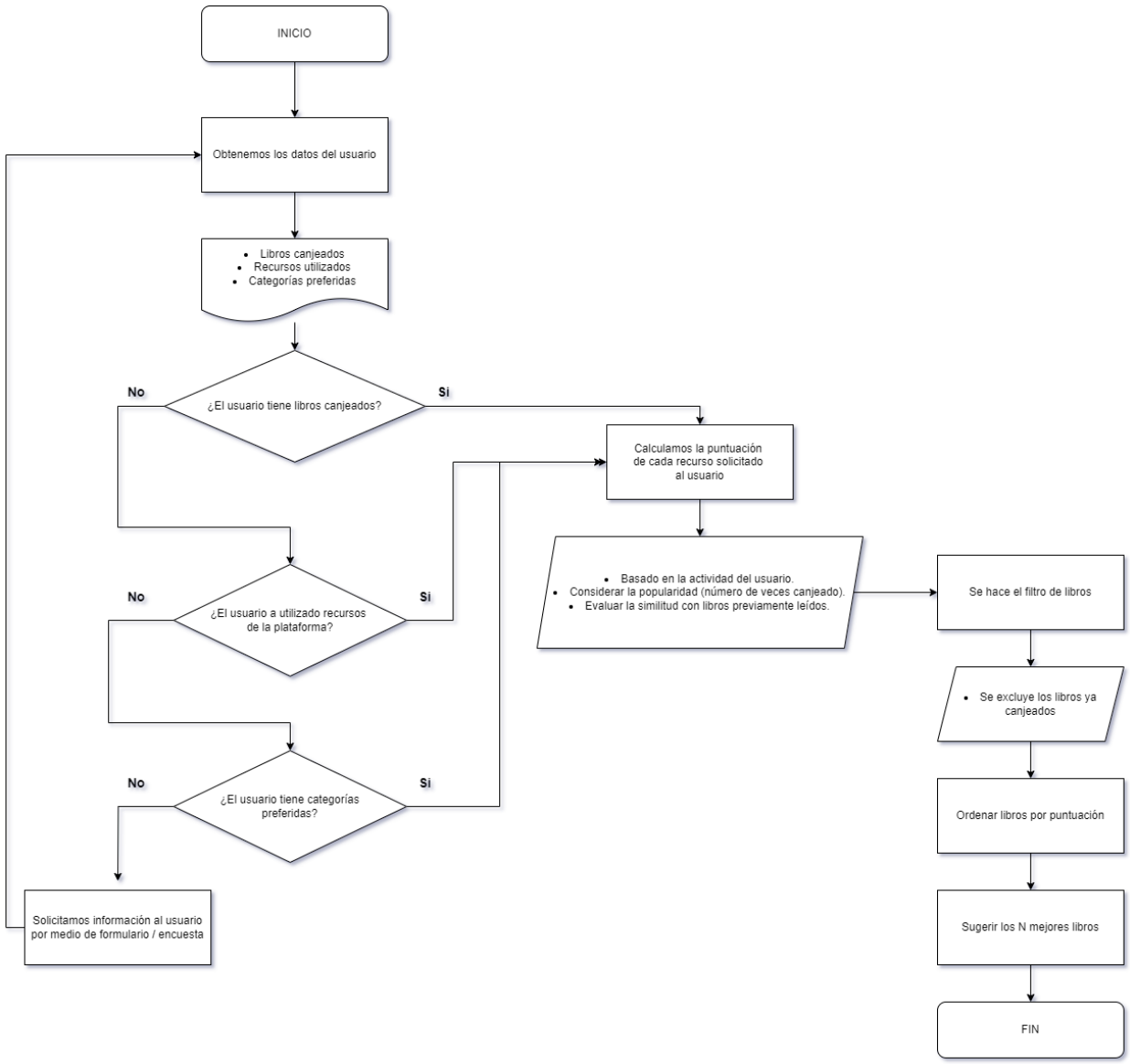
1. Diseña un algoritmo eficiente que permita recomendar el próximo libro digital que un usuario debería canjear, basado en sus hábitos de uso y en las interacciones de otros usuarios con recursos similares.
2. Explica la estructura de datos óptima para almacenar esta información y permitir consultas rápidas a gran escala.
3. Describe la complejidad computacional de tu solución en términos de Big-O.
4. Si el sistema tuviera millones de libros digitales y usuarios accediendo simultáneamente, cómo escalarías la solución.

1) **Diseñando el algoritmo:**

Pseudocodigo (Visual Studio Code):

```
función recomendar_libro(usuario):
    libros = obtener_libros_catalogo()
    puntuaciones = diccionario vacío

    por cada libro en libros:
        si libro ya canjeado por usuario:
            continuar

        puntuaciones[libro] = 0
        puntuaciones[libro] += popularidad(libro)  # Basado en número de canjeos
        puntuaciones[libro] += actividad_usuario(usuario, libro)  # Basado en recursos usados

    libros_recomendados = ordenar_por_puntuacion(puntuaciones)
    return libros_recomendados[:N]  # Retornar los N mejores libros

    ## Marvin Leonel García Lémus
    ## leogarcia0027@gmail.com
    ## Desafio Tecnico Susaeta Ediciones / Algoritmo
```

Diagrama de Flujo (realizado en Draw.io):



2) Estructura de datos optima:

1. Diccionarios/Tablas Hash

Uso:

- **Libros**: Cada libro se almacena en un diccionario donde la clave es un identificador único (ID del libro) y el valor es un conjunto de atributos del libro, como título, autor, categorías, canjeos, tiempo de lectura, etc. Lo cual permitirá acceder rápidamente a la información de un libro específico utilizando su ID.
- **Usuarios**: Similarmente, los usuarios se almacenan en otro diccionario donde la clave es el ID del usuario y el valor contiene información como nombre, libros canjeados y recursos utilizados. Esto permite un acceso rápido a los datos del usuario.

Ejemplificando:

- **Libro**: ID 1 → { título: "Matemáticas Avanzadas", autor: "Juan Pérez", categorías: ["Matemáticas", "Ciencias"], canjeos: 150 }
- **Usuario**: ID "usuario1" → { nombre: "Carlos", libros_canjeados: [1, 2], recursos_utilizados: 2 }

2. Listas

Uso:

- Cada categoría tiene una lista asociada que contiene los IDs de los libros que pertenecen a esa categoría. Esto facilita la búsqueda de libros similares o relacionados dentro de una categoría.

Ejemplificando:

- **Categoría "Matemáticas"**: [1, 3, 5] (donde 1, 3 y 5 son los IDs de libros en esta categoría)
- **Categoría "Ciencias"**: [1, 4]

3. Matrices Dispersas

Uso:

- Cada fila de la matriz representa a un usuario y cada columna representa un libro. En cada celda de la matriz, se guarda un valor que indica si el usuario ha canjeado o interactuado con el libro (por ejemplo, un 1 si lo ha canjeado y 0 si no)

Ejemplificando:

- **Matriz**:
    - Usuario 1 → [1, 0, 1] (ha canjeado el libro 1 y 3, no el 2)
    - Usuario 2 → [0, 1, 1] (ha canjeado el libro 2 y 3, no el 1)

Ahora ejemplificando en código (MySQL)

```
#libros: Almacena información de los libros
#usuarios: Almacena información de los usuarios
#categorias: Organiza los libros en categorías
#usuarios_libros: Relaciona usuarios con los libros que han canjeado

#Libros:
id (INT, PRIMARY KEY, AUTO_INCREMENT): #Identificador único del libro
titulo (VARCHAR): #Título del libro.
autor (VARCHAR): #Autor del libro.
categoria_id (INT, FOREIGN KEY): #Relaciona con la tabla de categorías
canjeos (INT): #Número de veces que el libro ha sido canjeado
tiempo_lectura (INT): #Tiempo estimado de lectura en minutos

#Usuarios:
id (INT, PRIMARY KEY, AUTO_INCREMENT): #Identificador único del usuario
nombre (VARCHAR): #Nombre del usuario.

#Categorías:
id (INT, PRIMARY KEY, AUTO_INCREMENT): #Identificador único de la categoría
nombre (VARCHAR): #Nombre de la categoría (ej. "Matemáticas", "Ciencias")

#Usuarios_Libros:
user_id (INT, FOREIGN KEY): #Relaciona con la tabla de usuarios
book_id (INT, FOREIGN KEY): #Relaciona con la tabla de libros
PRIMARY KEY (user_id, book_id): #Clave primaria compuesta para asegurar que un usuario no canjee el mismo libro más de una vez

#Consultando para obtener los libros de un usuario:
SELECT b.titulo, b.autor
FROM books b
JOIN user_books ub ON b.id = ub.book_id
WHERE ub.user_id = ?;  -- Reemplaza ? con el ID del usuario

#Consultando para obtener los libros de una categoría:
SELECT b.titulo, b.autor
FROM books b
JOIN categories c ON b.categoria_id = c.id
WHERE c.nombre = 'Matemáticas';  -- Cambia 'Matemáticas' por la categoría deseada

# Marvin Leonel Garcia Lemus
# leogarcia0027@gmail.com
# Desafio Tecnico Susaeta - Microservicios con Docker Compose
```

3) Complejidad computacional Big-O

```
## Marvin Leonel García Lémus
## leogarcia0027@gmail.com
## Desafio Tecnico Susaeta Ediciones / Big-O

## Obtener datos del usuario: O(N), donde N es el número de libros
## Calcular puntuaciones: O(M * C), donde M es el número de libros y C es el número de recursos digitales utilizados por el usuario
## Filtrar libros: O(N)
## Ordenar libros: O(K log K), donde K es el número de libros no canjeados
## La complejidad total del algoritmo puede ser O(N + M * C + K log K)
```

Complejidad Total

O(N + M * C + K log K):

Interpretación: El rendimiento del algoritmo será dominado por el término más grande a medida que las variables se vuelven grandes. Ejemplo:

- Si N es mucho mayor que M y C, la complejidad se aproximará a O(N).
- Si M y C son grandes, entonces O(M * C) será el término dominante.
- Si K es muy grande, O(K log K) dominará.

4) Si el sistema tuviera millones de libros digitales y usuarios accediendo simultáneamente, cómo escalarías la solución

Implementación de caché y microservicios dentro del código general y la base de datos

Caché:

```php
<?php
$redis = new Redis();
$redis->connect('localhost', 5000);

$redis->set('libros_populares', json_encode($libros_populares)); // Aqui indicamos el almacen de resultados en caché para los libros populares

$libros_populares = json_decode($redis->get('libros_populares'), true); // Aqui recuperamos los resultados de los libros populares

// Marvin Leonel Garcia Lemus
// leogarcia0027@gmail.com
// Desafio Tecnico Susaeta - Optimización Caché
```

Microservicios:

```yaml
# Microservicios
version: '3'
services:
  usuarios:
    image: usuarios-service:latest
    ports:
      - "5000:5000"

  libros:
    image: libros-service:latest
    ports:
      - "5001:5001"

  recomendaciones:
    image: recomendaciones-service:latest
    ports:
      - "5002:5002"

// Marvin Leonel Garcia Lemus
// leogarcia0027@gmail.com
// Desafio Tecnico Susaeta - Microservicios con Docker Compose
```

Una solución optima y escalable es la combinación de microservicios enlazados con la Nube (Amazon DynamoDB):

Se implementa un sistema donde cada microservicio (como usuario, libros, y recomendaciones) accede a los libros almacenados en la nube y utiliza caché para optimizar el rendimiento. Cada servicio se puede escalar de manera indipendiente y el almacenamiento en la nube proporciona la capacidad de gestionar un gran volumen de datos en simultaneo, sin la necesidad de intervenir concurrentemente en optimizar el flujo de datos.