Documentación del Proyecto

Descripción del Proyecto

Este proyecto implementa una API REST que permite analizar secuencias de ADN para determinar si una persona es un mutante o un humano, según las especificaciones del ejercicio propuesto. Además, la API mantiene estadísticas de las verificaciones realizadas, mostrando la cantidad de mutantes y humanos, así como el ratio entre ambos. Utiliza una arquitectura en capas para separar la lógica de negocio, la persistencia y la interacción con el cliente.

Tecnologías Utilizadas

Backend:

- **Lenguaje**: Java
- **Framework**: Spring Boot
- **Base de Datos**: H2 (en memoria)
- **Contenedor**: Docker
- **Despliegue**: Render (Plataforma de alojamiento en la nube)
- **Herramientas:**
    - **Spring Initializr**: Para la configuración inicial del proyecto.
    - **Gradle**: Para la gestión de dependencias.
    - **Postman**: Para pruebas de la API.
    - **JUnit & Mockito**: Para pruebas unitarias.

Arquitectura del Proyecto

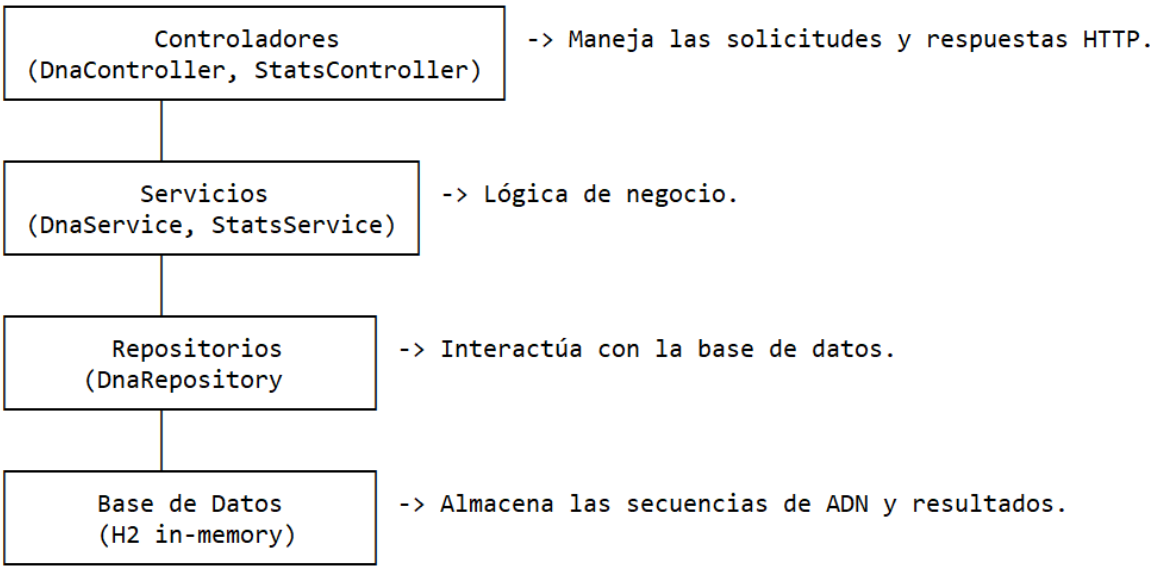
El proyecto sigue una arquitectura en capas para garantizar una adecuada separación de responsabilidades, facilitando su mantenibilidad y escalabilidad.

Backend:

La arquitectura está dividida en varias capas:

1. **Capa de Controlador**: Maneja las solicitudes HTTP y devuelve las respuestas correspondientes.
2. **Capa de Servicio**: Contiene la lógica de negocio, incluyendo la validación y los procesos de análisis de ADN.
3. **Capa de Persistencia**: Se encarga de interactuar con la base de datos mediante los repositorios de JPA.
4. **Capa DTO (Data Transfer Object)**: Define los objetos utilizados para transferir datos entre las diferentes capas.
5. **Capa Entities**: Define las entidades que representan las tablas en la base de datos.

Diagrama de Arquitectura del Proyecto



Comunicación de Datos en la Arquitectura

1. **Solicitud del Cliente (Controlador)**: El cliente realiza una solicitud HTTP (POST o GET) para enviar una secuencia de ADN o para obtener estadísticas.
2. **Controlador**: El controlador (por ejemplo, `DnaController`) recibe la solicitud, valida los datos de entrada y pasa la secuencia de ADN a la capa de servicio (`DnaService`).
3. **Servicio**: En `DnaService`, se ejecuta la lógica de negocio. Aquí se valida si el ADN es válido y se analiza si la secuencia corresponde a un mutante o no. Si la secuencia es nueva, se persiste en la base de datos.
4. **Persistencia**: La capa de persistencia (`DnaRepository`) interactúa con la base de datos para guardar o consultar datos sobre las secuencias de ADN.
5. **Respuesta al Cliente**: Finalmente, el controlador envía una respuesta HTTP adecuada al cliente, indicando si el ADN es mutante o no, o devolviendo estadísticas si es solicitado.

Documentación General del Código

Algoritmo Utilizado

El código que se ha implementado es parte de un sistema diseñado para detectar si una secuencia de ADN pertenece a un mutante. El algoritmo recorre una matriz de ADN NxN buscando más de una secuencia de 4 letras consecutivas iguales en tres direcciones: horizontal, vertical y diagonal. El enfoque principal del algoritmo utiliza técnicas de **Sliding Window** y búsqueda diagonal para optimizar la verificación de las secuencias de ADN.

Búsqueda de Secuencias

El algoritmo recorre la matriz de ADN buscando secuencias consecutivas de 4 caracteres iguales en tres direcciones:

- **Horizontales (filas)**: Se busca en cada fila de la matriz si hay 4 letras iguales consecutivas.
- **Verticales (columnas)**: Se busca en cada columna de la matriz de ADN.
- **Diagonales principales e inversas**: Se buscan secuencias en las diagonales principales (↘) y diagonales inversas (↗).

Tipo de Algoritmo

El algoritmo sigue un enfoque de búsqueda secuencial, utilizando las siguientes técnicas:

- **Sliding Window**: Esta técnica se utiliza en el recorrido de las filas y columnas. Se toma un grupo de 4 letras consecutivas y si se encuentra una secuencia válida, se salta a la siguiente posición relevante, optimizando las comparaciones.
- **Búsqueda Diagonal**: El algoritmo también recorre las diagonales principales e inversas de la matriz, verificando si hay secuencias de 4 caracteres consecutivos.

Salida Anticipada

El algoritmo está optimizado para detenerse en cuanto detecta más de una secuencia mutante, reduciendo el tiempo de ejecución en casos donde ya se puede determinar que el ADN es mutante.

Complejidad del Algoritmo

- **Complejidad Temporal**: La complejidad del algoritmo es **$O(n^2)$** en el peor de los casos, ya que tiene que revisar todas las filas, columnas y diagonales de la matriz NxN. Sin embargo, gracias a las optimizaciones (como la salida anticipada y los saltos en la secuencia), el tiempo de ejecución suele ser más bajo en la práctica.

Endpoints de la API

1. Verificación de ADN Mutante

- **Método**: POST
- **URL**: `/mutant/`
- **Descripción**: Detecta si una secuencia de ADN corresponde a un mutante.
- **Request Body**:

```
{
  "dna": ["ATGCGA", "CAGTGC", "TTATTT", "AGACGG", "GCGTCA", "TCACTG"]
}
```

- **Response**:
  - **200 OK**: Si la secuencia de ADN corresponde a un mutante.
  - **403 Forbidden**: Si la secuencia no corresponde a un mutante.

2. Estadísticas de ADN

- **Método**: GET
- **URL**: `/stats/`
- **Descripción**: Devuelve estadísticas de las secuencias verificadas, mostrando el número de mutantes, humanos y el ratio entre ambos.
- **Response**:

```
{
  "count_mutant_dna": 40,
  "count_human_dna": 100,
  "ratio": 0.4
}
```

Instalación y Configuración

Backend:

1. **Clonar el Repositorio**:

   ```
   git clone https://github.com/Lucas-Chavez/ProgramacionIII-Parcial-1.git

   cd ProgramacionIII-Parcial-1
   ```

2. **Configuración del Proyecto**: El proyecto utiliza Gradle para la gestión de dependencias. Asegúrate de tener Gradle instalado.

3. **Ejecutar el Proyecto**: Para ejecutar el proyecto localmente:

   ```
   ./gradlew bootRun
   ```

   Accede a la API en `http://localhost:8080`.

4. **Base de Datos H2**: El proyecto utiliza H2 como base de datos en memoria. Puedes acceder a la consola de H2 en:

   `http://localhost:8080/h2-console`

   Configuración para acceder:

   - **JDBC URL**: `jdbc:h2:mem:testdb`
   - **Username**: `sa`
   - **Password**: (dejar en blanco)

Despliegue en Render

El proyecto ha sido desplegado en la plataforma **Render**. Para acceder a la API en producción, utiliza la siguiente URL:

- **URL del Proyecto**: https://programacioniii-parcial-1.onrender.com

---

Pruebas Unitarias

Se han implementado pruebas unitarias para validar la funcionalidad principal del proyecto, utilizando JUnit y Mockito. Las pruebas cubren:

1. **Verificación de ADN Mutante**: Pruebas para validar diferentes secuencias de ADN, tanto mutantes como no mutantes.
2. **Estadísticas de ADN**: Pruebas que verifican el cálculo correcto de las estadísticas de secuencias almacenadas.

Para ejecutar las pruebas unitarias:

```
./gradlew test
```

Repositorio Utilizado

El código fuente completo está disponible en el siguiente repositorio:

- Repositorio en GitHub