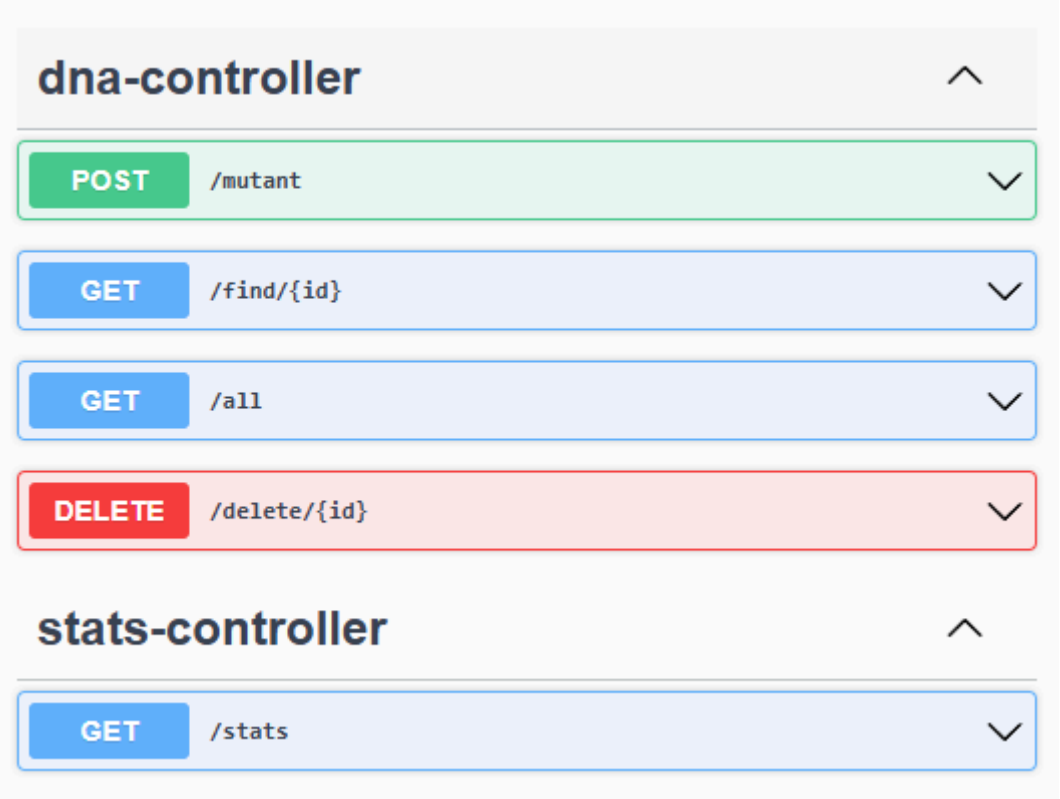
Documentación del proyecto

Documentación visual



Pruebas con JMeter

Resultados

Se realizaron pruebas de estrés mediante la utilización de JMeter, estas fueron realizadas en local y arrojaron los siguientes resultados.

- Test con 100 peticiones en un segundo:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	100	4	2	7	1.04	0.00%	100.6/sec	31.93	29.67	325.0
TOTAL	100	4	2	7	1.04	0.00%	100.6/sec	31.93	29.67	325.0

- Test con 500 peticiones en un segundo:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Thread Grou...	500	4	2	21	2.33	0.00%	450.0/sec	142.84	132.73	325.0
TOTAL	500	4	2	21	2.33	0.00%	450.0/sec	142.84	132.73	325.0

- Test con 1000 peticiones en un segundo:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Thread Grou...	1000	16	1	145	22.51	0.00%	827.1/sec	262.52	243.94	325.0
TOTAL	1000	16	1	145	22.51	0.00%	827.1/sec	262.52	243.94	325.0

Test-Automáticos

Se implementaron 15 tests los cuales dieron una efectividad del 100%. En estos se intento abarcar todos los escenarios posibles con matrices de distintas tamaños y comportamiento, se encuentran en el proyecto en el Folder `test`.

Diagrama de secuencia

Diagrama para el POST

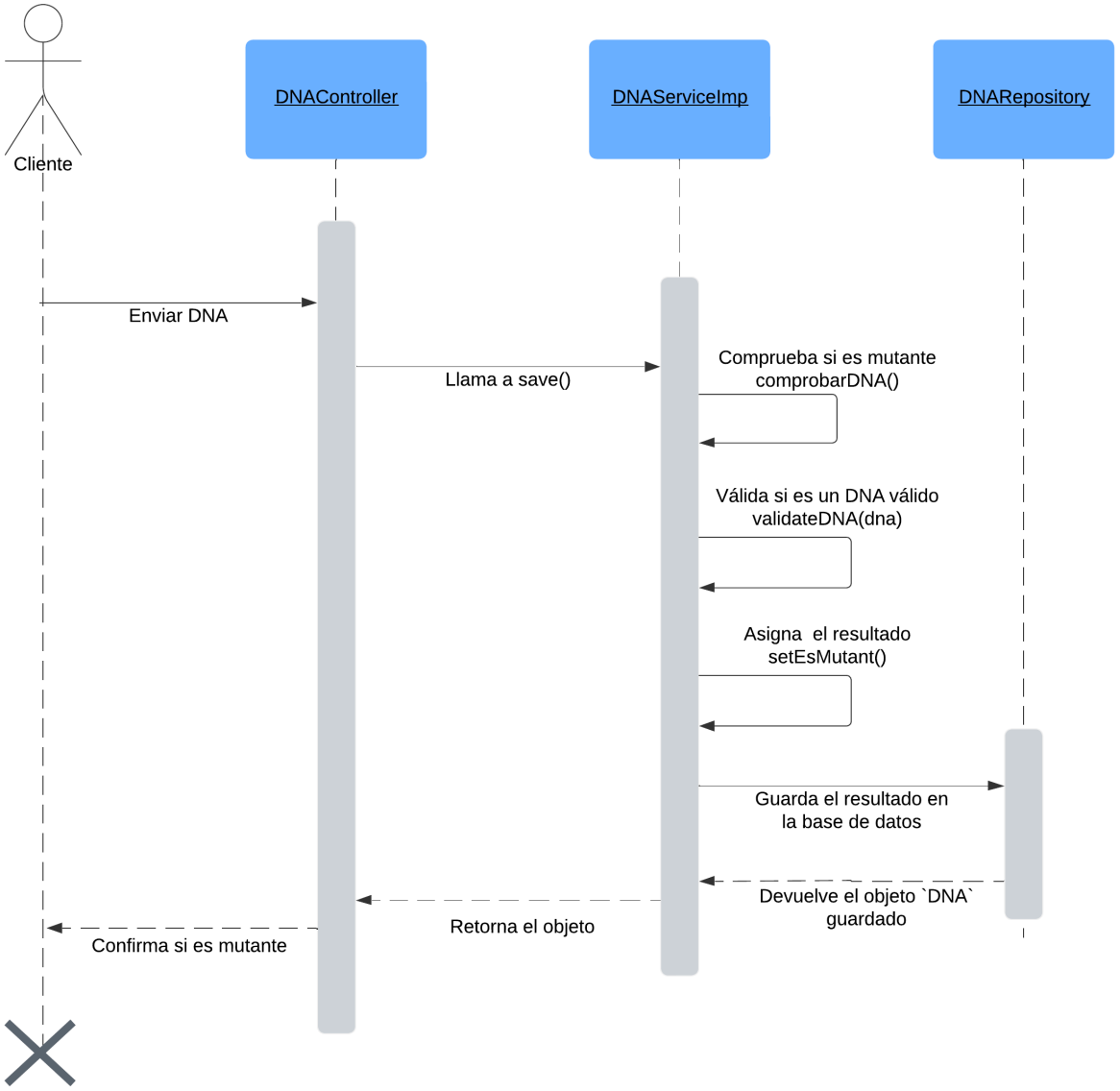
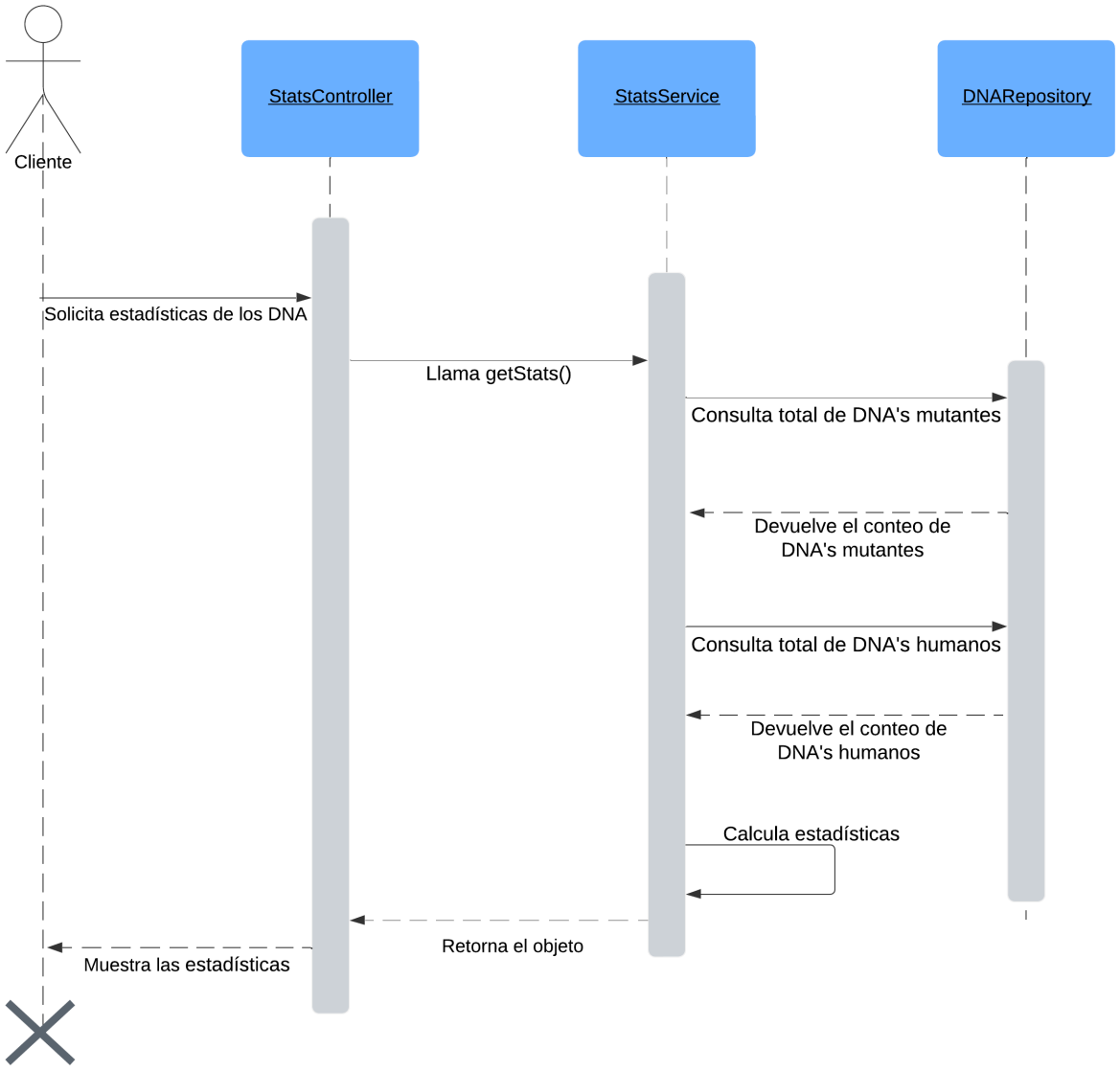
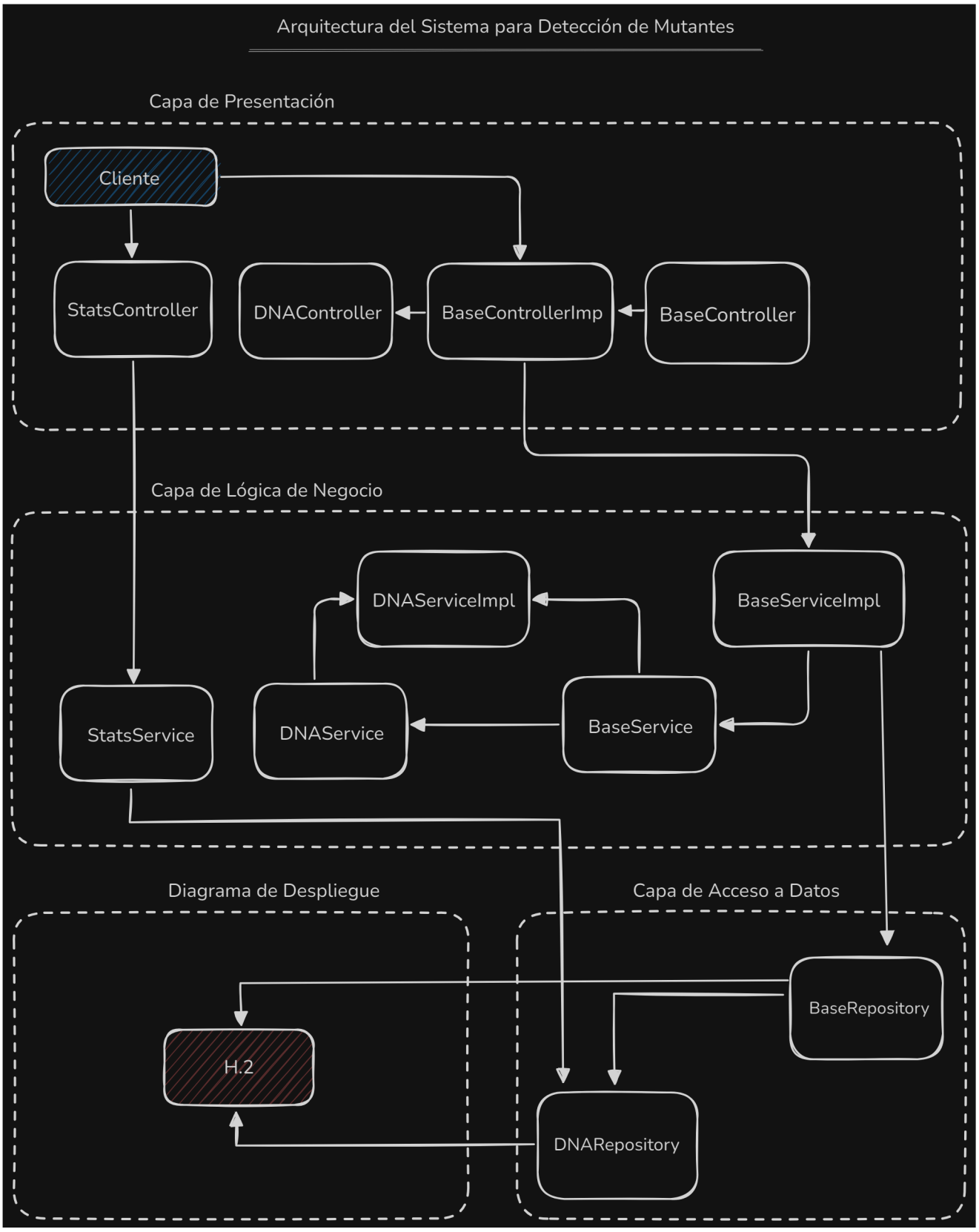


Diagrama para Stats



Arquitectura del sistema

Para nuestro sistema se decidió por optar código genérico y reutilizable para que en un futuro sea escalable, permitiendo en caso de que se quiera añadir funciones concretas al mutante o humano se puedan implementar de manera sencilla y limpia. Ademas de esta manera el código queda limpio y organizado.



Capa de Presentación

- **Cliente**: Nuestro cliente será quien represente el componente que interactúa. Este componente envía solicitudes y recibe respuestas del backend.
- **StatsController**: En este controlador se gestionara las solicitudes relacionadas con las estadísticas, como el número de mutantes y humanos en la base de datos. Recibe las peticiones del cliente y se comunica con el servicio correspondiente para procesarlas.

- **DNAController**: Controlador que maneja las solicitudes relacionadas al ADN mutante. Se agrego para que sea escalable y en un futuro contener el manejar de solicitudes de servicios en particular.
- **BaseController**: Interfaz que define métodos comunes para todas las entidades y servicios DNA.
    - **BaseControllerImpl**: Implementación de la interfaz `BaseController`, que puede contener la lógica común de los DNA.

Capa de Lógica de Negocio

- **BaseService**: Define métodos comunes a nivel de lógica de negocio para ser reutilizados por servicios específicos.
    - **BaseServiceImpl**: Implementa los métodos comunes para los DNA, es vital ya que contiene la implementación de la mayoría de la lógica del proyecto.
- **StatsService**: Servicio que contiene la lógica de negocio relacionada con las estadísticas. Este servicio puede interactuar con repositorios para acceder a los datos necesarios.
- **DNAService**: Servicio responsable de la lógica de negocio relacionada con el ADN, en caso de existir lógica adicional a futuro se contendría aquí.
    - **DNAServiceImpl**: Implementación de la interfaz `DNAService`, que define cómo se lleva a cabo la detección de ADN mutante.

Capa de Acceso a Datos

- **DNARepository**: Repositorio que se encarga de la interacción con la base de datos para los métodos específicos de los DNA.
- **BaseRepository**: Interfaz que define métodos comunes de los DNA para la interacción con la base de datos, como operaciones CRUD. Permite a otros repositorios heredar y reutilizar esta funcionalidad básica.

Diagrama de despliegue

- **H2 (base de datos)**: Es la base de datos en memoria que utilizara nuestro sistema para almacenar los datos relacionados con las solicitudes de ADN y las estadísticas.