UNIVERSIDAD ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

INFORME DE ARQUITECTURA

ECI-Bienestar

Equipo Ingenieros BackEnd

AUTORES

Grupo de Arquitectos y Desarrolladores Backend de Bismuto

PROGRAMA
Ingeniería de Sistemas

ASIGNATURA
Ciclos de Vida del Desarrollo de Software (CVDS)

PROFESORES
Ing. Rodrigo Humberto Gualtero Martínez
Ing. Andrés Martín Cantor Urrego

BOGOTÁ DC, COLOMBIA
Sábado, 4 de Mayo de 2025

Generalidades Módulos a Cargo de Bismuto

En el desarrollo de sistemas modernos, la elección y aplicación de tecnologías, patrones de arquitectura, bases de datos y herramientas debe responder a una visión estratégica, donde cada decisión esté alineada con los objetivos del negocio, la escalabilidad y la productividad del equipo. No se trata de adoptar tecnologías por tendencia, sino de utilizar aquellas que mejor se adapten a las necesidades específicas de cada módulo, considerando factores como los tiempos de entrega, el conocimiento del equipo y la interoperabilidad entre componentes.

Por esta razón, se plantearon las siguientes generalidades para cada uno de los módulos a cargo de nuestro grupo en este proyecto. Estas decisiones no fueron tomadas de manera individual, sino que corresponden a acuerdos establecidos por todo el grupo, considerando diversas razones técnicas que se explican más adelante. Este acuerdo permitió definir un marco común que guía el desarrollo del sistema, asegurando coherencia, eficiencia y calidad sobre el producto.

Tecnologías a usar

Comparativa entre Lenguajes y Framework

A continuación, presentamos un análisis y comparación de distintos lenguajes de programación y sus respectivos entornos, tanto para el desarrollo de backend como de frontend. Este comparativo destacan atributos clave como extensibilidad, ecosistemas de librerías, facilidad de uso y casos de aplicación, con el fin de ayudar a seleccionar la tecnología más adecuada para cada módulo.

Característica	Spring Boot	Express.js	Django	ASP.NET Core	Laravel
Lenguaje	Java	Javascript	Python	C#	PHP
Arquitectura REST	Soporte nativo con anotaciones	Muy flexible y manual por módulos	Basado en clases y serializadores	Nativo con controllers	Integrado vía controllers
Desempeño	Alto	Medio alto	Medio	Alto	Medio

ORM / Persistencia	JPA	Sequelize o Prisma	Django ORM	Entity Framework Core	Eloquent ORM
Manejo de errores	Excepciones + ResponseEntity	Manual con middlewares	Manejadores personalizados	Filtros y middlewares	Exception handling integrado
Autenticación y seguridad	Spring Security	Manual con middleware	Django Auth, JWT	Identity + JWT	Laravel Sanctum / Passport
Testing	Sí (JUnit, Mockito)	Sí (Jest, Mocha, Supertest)	Sí (Django Test)	Sí (xUnit, NUnit)	Sí (PHPUnit)
Soporte para microservicios	Excelente (Spring Cloud)	Parcial (requiere setup)	Limitado	Sí (modular y escalable)	Limitado
Ecosistema y comunidad	Enorme y consolidado	Muy activa y abundante	Activa en entornos científicos	Empresarial y estable	Muy activa y amigable

Java lleva más de dos décadas validándose en entornos empresariales de alta exigencia; su estabilidad y compatibilidad hacia atrás garantizan que nuestras librerías y código legado sigan funcionando sin interrupciones. Además, la JVM optimiza el rendimiento en tiempo de ejecución mediante compilación JIT y gestión automática de memoria, lo que se traduce en respuestas ágiles de nuestros endpoints de autenticación y consulta de usuarios, incluso bajo altos niveles de carga.

El tipado estático de Java detecta errores en fase de compilación y fomenta una disciplina de diseño robusta. En un módulo tan sensible como el de Gestión de Usuarios y autenticación donde manejamos validaciones críticas y control de roles, esta rigidez reduce defectos y facilita el mantenimiento a largo plazo. Por último, el ecosistema Java es inmenso: IDEs consolidados (IntelliJ, Eclipse), sistemas de construcción maduros (Maven, Gradle) y herramientas de calidad

como JaCoCo y SonarQube. Nuestra experiencia con estas tecnologías acelera la configuración de pipelines de CI/CD y garantiza altos estándares de cobertura y calidad de código.

Además, el ecosistema de Spring Boot es uno de los más completos y consolidados dentro del desarrollo de software moderno. Gracias a su integración nativa con múltiples herramientas como Spring Security, Spring Data, Spring Web, entre otros. Además, permite que los desarrolladores tengan acceso a soluciones listas para usar, bien documentadas y con una gran comunidad de soporte detrás. Esto reduce significativamente el tiempo de desarrollo y facilita el trabajo colaborativo entre equipos.

Uno de los pilares fundamentales del proyecto es la seguridad, y aquí Spring Boot destaca por ofrecer un enfoque robusto y personalizable a través de Spring Security. Esto permite implementar autenticación, autorización, protección contra ataques comunes como CSRF o inyecciones, y sistemas de roles de forma controlada y escalable.

En cuanto a la extensibilidad, Spring Boot permite construir sistemas que no están pensados solo para funcionar en este momento, sino también para crecer y adaptarse a nuevas necesidades en el futuro. Gracias a su enfoque modular y a su compatibilidad con la arquitectura de microservicios, cada parte del sistema puede desarrollarse, mantenerse y desplegarse de forma independiente, facilitando tanto la escalabilidad técnica como la organizacional. Esto resulta ideal para un proyecto dividido por módulos funcionales claros, como el nuestro. Otro aspecto clave es la eficiencia en el funcionamiento y en el desarrollo. Spring Boot automatiza muchas configuraciones complejas sin quitar flexibilidad, lo que nos permite enfocarnos en la lógica del negocio sin perder tiempo en aspectos técnicos repetitivos. Su rendimiento es alto y estable, lo que garantiza que el sistema responderá bien incluso cuando crezca en volumen de usuarios o datos.

Teniendo en cuenta lo expuesto, podemos afirmar que la adopción de Java en conjunto con Spring Boot representa la opción más adecuada, resultado de un análisis conjunto por parte de todo el equipo. Esta elección responde directamente a las necesidades reales de nuestro proyecto y se sustenta en múltiples ventajas técnicas y operativas frente a otras alternativas disponibles.

En conclusión, Spring Boot nos proporciona una base robusta, segura y adaptable para desarrollar un sistema que no solo cumple con los requerimientos actuales, sino que también está preparado para evolucionar conforme crecen las necesidades del módulo. Esta elección, fruto del consenso dentro del equipo, equilibra lo técnico con lo estratégico: garantiza calidad en el producto final y optimiza la experiencia de desarrollo.

Bases de Datos

En el desarrollo de aplicaciones bajo arquitecturas de microservicios, la elección entre bases de datos SQL y NoSQL es algo crucial. Cada tipo de base de datos ofrece ventajas específicas según el tipo de módulo o funcionalidad que se esté construyendo. A continuación, se presenta un cuadro comparativo que ilustra las diferencias clave entre ambas tecnologías.

Característica / Criterio	**SQL (Relacional)**	**NoSQL (No Relacional)**
Estructura de datos	Tablas con relaciones definidas, esquemas fijos	Documentos, grafos, pares clave-valor o columnas; esquemas flexibles
Transacciones ACID	Soporte total, es ideal para lógica financiera pues hay un alto nivel de precisión y confiabilidad, usando principios de atomicidad de transacciones, consistencia en los datos, aislamiento en las transacciones y registro de transacciones.	Soporte limitado o eventual, pues depende de la capacidad del motor utilizado, como MongoDB o Cassandra.

Escalabilidad	Vertical (más recursos al servidor)	Horizontal (añadir más nodos fácilmente)
Flexibilidad del esquema	Bajo, ya que se usan estructuras de datos fijas que dificultan la mutación de las tablas de datos.	Alto, ya que no hay una estructura determinada y cada documento, grafo o esquema varia su forma de organizar los datos insertados.
Velocidad en escritura masiva	Moderada, requiere validaciones para alterar las tablas o realizar transacciones.	Alta, se puede almacenar o variar grandes cantidades de datos en poco tiempo
Consulta de datos complejos	Potente, pues cuenta con SQL estándar, JOINs, agregaciones, vistas y demás	Limitada o especializada, pero escalable. No cuentan con herramientas complejas a comparación de las SQL
Casos de uso ideales	- Pagos y transacciones - Facturación - Gestión de usuarios - Auditoría	- Historial de actividad del usuario - Feed o timeline social - Contenido dinámico y cambiante (ej. posts, comentarios, productos)

Es común poder usar ambas en un mismo servicio, pues nos permite escoger lo mejor de ambos tipos de bases y sacar mayor rendimiento en el manejo de los datos. Esta información brindada, sirve como punto de partida para cada que squad decida como manejar los datos dentro de su sistema.

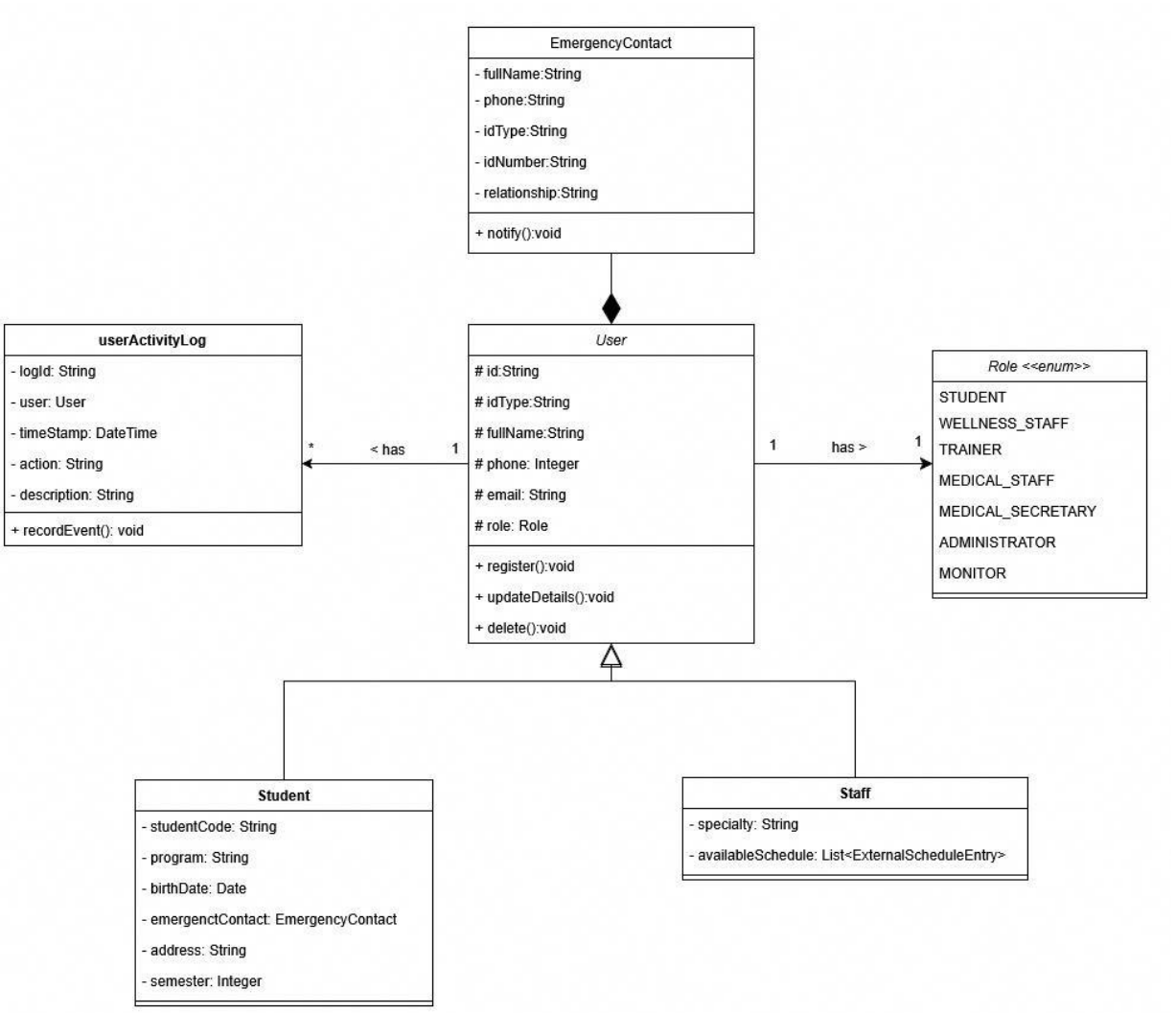
En una arquitectura de microservicios, es recomendable que cada servicio tenga su propia base de datos para garantizar su autonomía, escalabilidad y resiliencia. Esta separación permite que cada microservicio evolucione de forma independiente, sin necesidad de coordinarse con otros equipos al modificar su modelo de datos. Además, facilita la escalabilidad individual, ya que cada servicio puede ajustarse en capacidad según su propia carga, sin afectar el rendimiento del resto del sistema.

Módulo Gestión de Usuarios

Este módulo es parte fundamental de la aplicación, pues realiza el control y gestión de usuarios registrados. Su propósito principal es permitir la creación, edición, eliminación y organización de las cuentas de los usuarios que interactúan con la aplicación. A través de este módulo se definen roles, permisos y estructuras jerárquicas que determinan el nivel de acceso de cada usuario a las distintas funcionalidades del sistema. También permite llevar un registro de actividad asociado a cada perfil, facilitando la trazabilidad y el control interno. Este módulo es clave para mantener el orden, la seguridad operativa y una gestión eficiente de los usuarios dentro de la plataforma.

Para comenzar, teniendo en cuenta la información presente en la sección de generalidades, usaremos Spring Boot como framework backend estandarizado en este proyecto, pues como se explicó más a detalle en dicha sección, es conveniente que todos los desarrolladores involucrados dominen esta herramienta en caso de ser necesario una intervención de algún squad para ayudarnos en cualquier situación requerida. Además, nos permite crear un software muy seguro y estable que mantenga la información de los usuarios segura y protegida de cualquier amenaza, de modo que, garantizamos la estabilidad casi total de nuestro servicio. No obstante, no dejamos de lado la necesidad de mantener cohesión entre los distintos módulos, que facilita la tarea de mantenimiento general de la aplicación. Para esto, se planteó usar Spring Boot en su versión 3.2.X conjunto a Java 17, pues son las últimas versiones consideradas completamente estables sin renunciar a las novedades del framework ni sus complementos como plugins y dependencias.

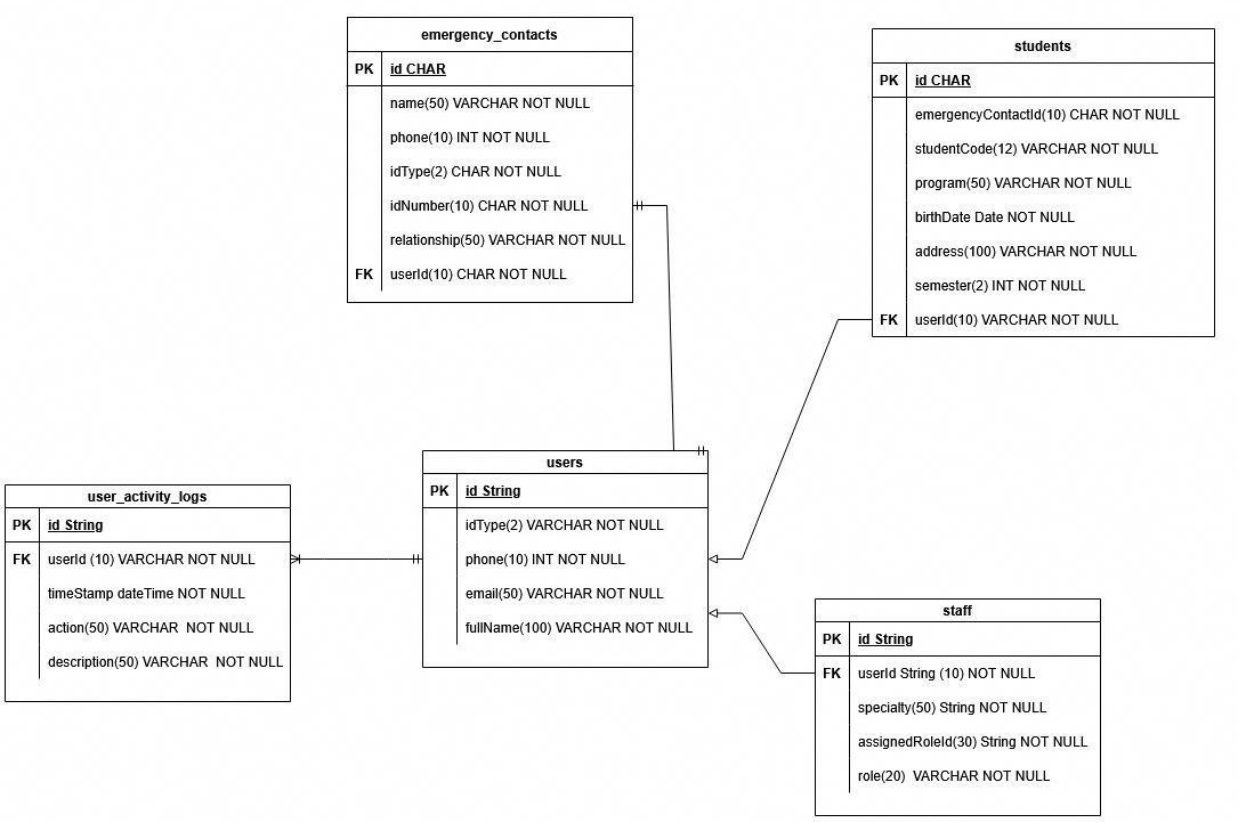
En primer lugar, se planteó el siguiente modelo de clases que permite representar las entidades principales del sistema, sus atributos, métodos y relaciones. A nivel general, ofrece una visión estructurada de los elementos que conforman la lógica del negocio, mientras que técnicamente facilita el diseño orientado a objetos y la identificación de responsabilidades entre clases.



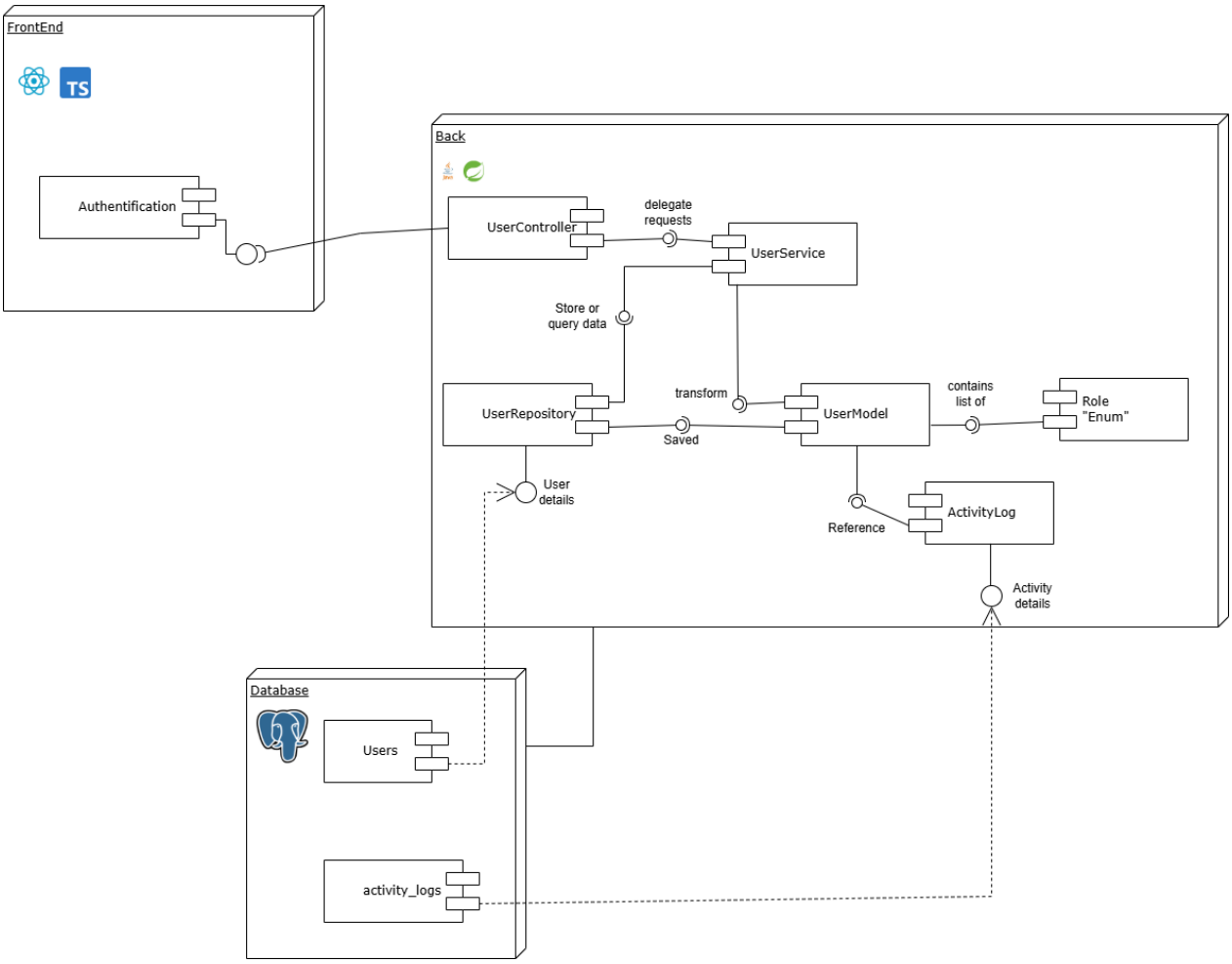
El diagrama detalla los elementos internos del sistema desde una perspectiva orientada a objetos. Por ejemplo, se representan clases como *User*, que incluye atributos como id, nombre, email, etc, y métodos como register() o delete(), apoyado de una clase de datos especificados por el usuario y que los puede editar a conveniencia. También se incluye la clase enumeradora *Rol*, que incluye todos los roles únicos disponibles para un usuario, además, una clase *ActivityLog*, que mantiene un registro de ingreso que hace el usuario dentro del sistema.

En segundo lugar, decidimos utilizar una base de datos relacional para este módulo, pues al requerir almacenar información indispensable y sensible de los usuarios, necesitamos que los datos tengan una estructura clara y un orden preestablecido. Por otro lado, al tratarse de un servicio destinado

únicamente al control de los usuarios, requerimos de realizar transacciones ACID para garantizar que los datos siempre sean correctos al manipularlos, acciones que no se ejecutan con frecuencia por la naturaleza de los requerimientos o necesidades del sistema. Además, al ser un CRUD básico, podemos hacer uso de las consultas para fortalecer el módulo por completo y que podamos apoyar el servicio de estadísticas con datos mejor filtrados. Los datos quedarían representados de la siguiente manera.



Para finalizar, tenemos que plantear un flujo general de nuestro modulo, pues de esta forma podemos visualizar con más a detalle cómo se relacionan los componentes internos de nuestro sistema junto a un diagrama que ilustra la secuencia de ingresar a la página de administración de usuarios. De este modo, validaremos que cada componente cumpla con su función dentro de nuestro sistema y evitamos que falten elementos a futuro.



Este diagrama de componentes representa la estructura general del módulo de manejo de usuarios, dividido en tres partes principales; frontend, backend y base de datos. Cada una de estas capas cumple funciones específicas y se relaciona con las demás para permitir la autenticación de usuarios, el manejo de sus datos y el registro de sus actividades.

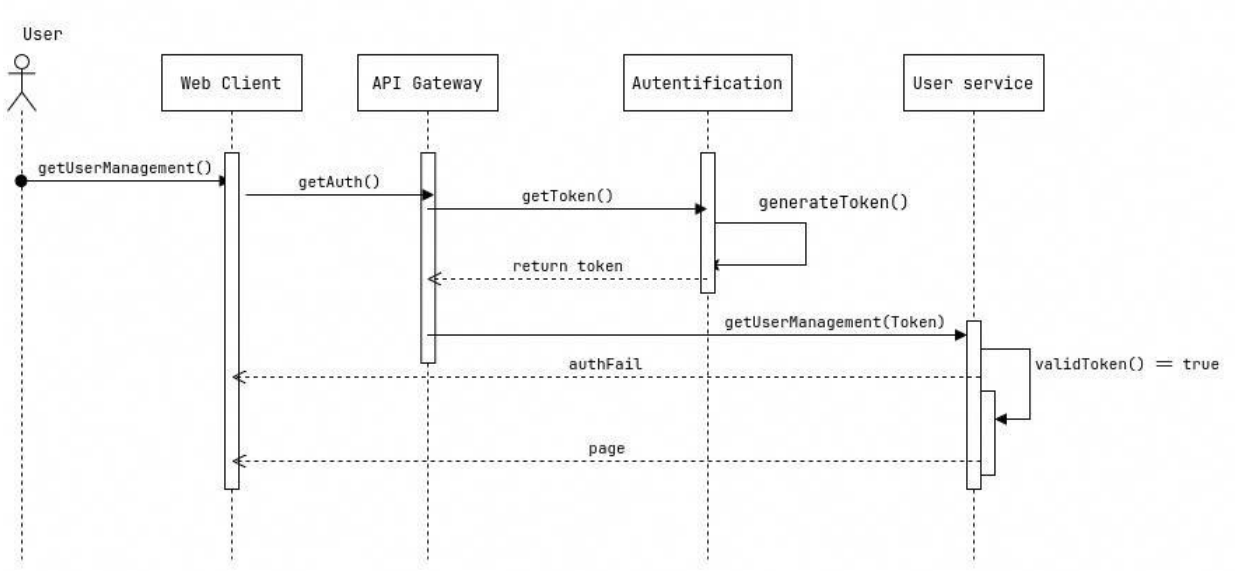
En el frontend se utiliza React junto con TypeScript. El único componente representado es el de autenticación, que se encarga de interactuar con el usuario y enviar solicitudes al backend para iniciar sesión. Esta capa es la encargada de ofrecer la interfaz gráfica con la que los usuarios finales interactúan.

El backend se organiza en varios componentes. El controlador de usuario recibe las solicitudes que llegan desde el frontend y las redirige al servicio correspondiente. Este servicio contiene la lógica principal del sistema, como la validación de credenciales o la gestión de roles. Para acceder a los datos almacenados, el servicio utiliza el repositorio, que es el componente encargado de

comunicarse con la base de datos. El modelo de usuario representa la estructura interna de los datos del usuario, incluyendo atributos como el correo o la contraseña, así como una lista de roles definidos por un enumerado.

La base de datos, almacena la información persistente del sistema. Cuenta con una tabla de usuarios donde se guardan los datos personales, y una tabla de registros de actividad, que contiene los eventos relacionados con las acciones de los usuarios. Estas tablas se conectan directamente con los componentes del backend encargados de gestionar la información.

Este diagrama permite visualizar de forma clara cómo se organiza el módulo de usuarios, mostrando los flujos entre el frontend, el backend y la base de datos, y explicando cómo cada parte contribuye al funcionamiento completo del sistema.



Este diagrama de secuencia muestra el flujo general de interacción entre un usuario y los distintos componentes del sistema durante el proceso de autenticación y acceso a una funcionalidad protegida. Participan cinco actores: el usuario, el cliente web, API Gateway, el servicio de autenticación y el servicio de usuarios.

El propósito principal del diagrama es ilustrar cómo se coordina la generación y validación de un token de acceso. El usuario realiza una solicitud desde el cliente web, la cual pasa por el API Gateway y llega al servicio de autenticación que genera un token. Este token luego se utiliza para acceder a los datos del sistema, y es validado por el servicio de usuarios antes de permitir el acceso.

Si el token no es válido, se devuelve un fallo de autenticación; de lo contrario, se permite el acceso al recurso solicitado.

Para el manejo de errores de este módulo, tendremos las siguientes convenciones:

Código HTTP	Mensaje de error	Causa probable
400	"Datos de entrada inválidos"	Validaciones fallidas en el formulario
401	"Usuario no autenticado"	Token inválido o ausente
404	"Usuario no encontrado"	algún usuario no existe
404	“Usuario ya existente en el sistema”	Usuario ya existente en la base
500	"Error interno del servidor"	Fallo inesperado

Con toda esta información, podemos decir que el servicio ya quedo preparado para pasar al desarrollo de código, pues se tiene claridad de los comportamientos internos y externos de los componentes y las relaciones con bases de datos y otros servicios como el de autenticación.

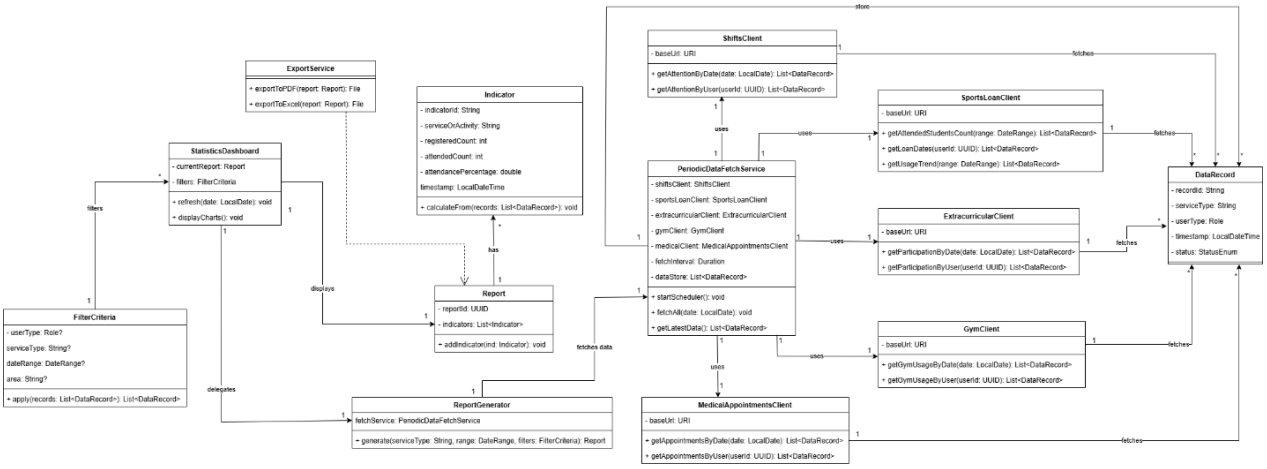
Módulo de Estadísticas

Este módulo, cumple la función de recopilar, procesar y presentar información relevante sobre el uso y funcionamiento del sistema. Su objetivo principal es ofrecer una visión general del comportamiento de los usuarios, el rendimiento de ciertas funcionalidades y otros datos clave que permitan tomar decisiones informadas. Además, organiza la información en métricas, gráficos o reportes, facilitando su análisis por parte de los administradores o responsables del sistema. Es una herramienta fundamental para el seguimiento, la mejora continua y la planificación estratégica dentro de la plataforma.

Para comenzar, teniendo en cuenta la información presente en la sección de generalidades, usaremos Spring Boot como framework backend estandarizado en este proyecto al igual que el

anterior modulo, pues como se explicó más a detalle en dicha sección, es conveniente que todos los desarrolladores involucrados dominen esta herramienta en caso de ser necesario una intervención de algún squad para ayudarnos en cualquier situación requerida. Además, nos permite crear un software muy seguro y estable que mantenga la información de los usuarios segura y protegida de cualquier amenaza, de modo que, garantizamos la estabilidad casi total de nuestro servicio. No obstante, no dejamos de lado la necesidad de mantener cohesión entre los distintos módulos, que facilita la tarea de mantenimiento general de la aplicación. Para esto, se planteó usar Spring Boot en su versión 3.2.X conjunto a Java 17, pues son las últimas versiones consideradas completamente estables sin renunciar a las novedades del framework ni sus complementos como plugins y dependencias.

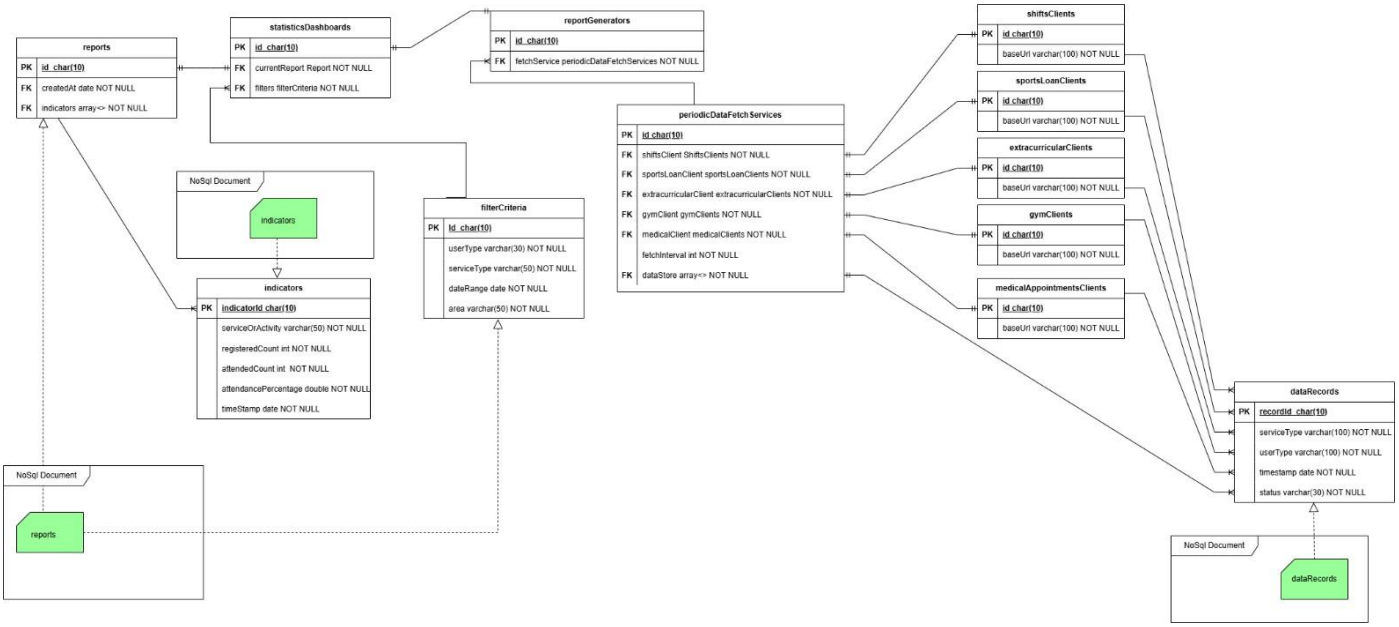
En primer lugar, se planteó el siguiente modelo de clases que permite representar las entidades principales del módulo de estadísticas, sus atributos, métodos y relaciones. A nivel general, ofrece una visión estructurada de los elementos que conforman la lógica encargada del análisis y visualización de datos, mientras que técnicamente facilita la organización orientada a objetos y la asignación clara de responsabilidades entre las clases.



El diagrama muestra componentes internos como la clase report, que sirve como base para distintos tipos de reportes, incluyendo atributos como identificadores, fechas o configuraciones, y métodos que permiten su generación o exportación. Se incluye también la clase statisticalData, que representa los datos ya procesados y listos para ser utilizados, y chartService, que transforma esos datos en gráficos o visualizaciones. Además, se presentan clases específicas como spReport, qmcReport o processChartService, cada una encargada de manejar estadísticas asociadas a

distintos contextos dentro del sistema. La clase configuration aparece de forma constante, ya que muchas de las operaciones dependen de parámetros definidos previamente por el usuario o el sistema.

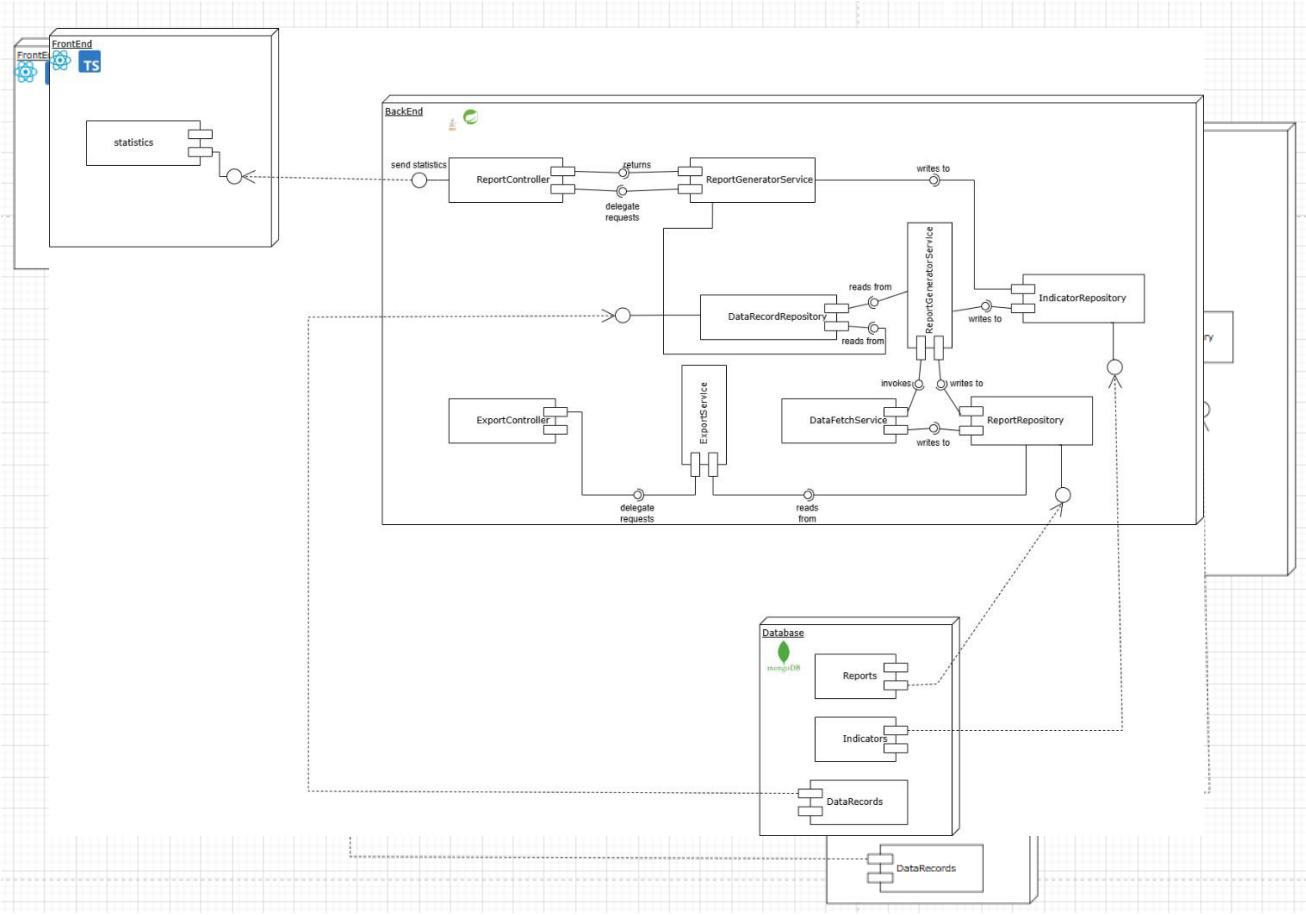
Por otro lado, considerando que este módulo maneja un historial de estadísticas obtenidas de distintos microservicios, se optó por utilizar una base de datos no relacional. Esta decisión se fundamenta en la necesidad de almacenar datos de manera flexible y eficiente, dado que las estadísticas pueden tener estructuras variables dependiendo de su origen. Una base de mongo maneja grandes volúmenes de información semiestructurada, permite una mayor escalabilidad y simplifica la integración de datos heterogéneos provenientes de múltiples servicios. A partir de estas decisiones, se planteó el siguiente modelo de clases.



Este modelo representa una base de datos no relacional orientada a documentos para almacenar los informes del módulo de estadísticas. La colección principal es reports, donde se guardan los informes generados con datos como el cliente, el autor, el tipo de informe y la fecha. Estos informes se relacionan con otras colecciones que contienen datos procesados o resultados específicos como statisticalClientDetails o chartDataPoints.

Cada colección almacena información concreta como datos por cliente, fechas o parámetros usados para generar el informe. También hay subdocumentos incrustados como dataStats o metrics que permiten acceder rápidamente a los datos sin depender de relaciones externas. Esta estructura hace más sencilla la consulta, permite almacenar todo el contenido del informe junto y mejora el rendimiento para lecturas rápidas o visualizaciones.

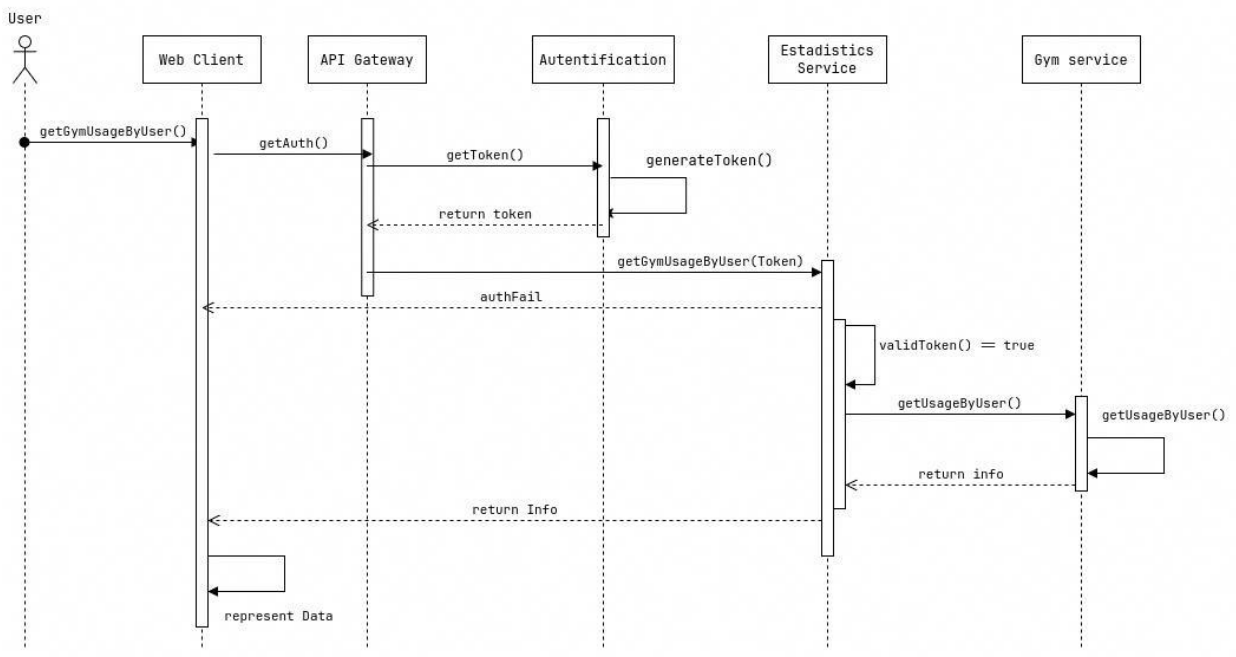
Para finalizar, tenemos que plantear un flujo general de nuestro modulo, pues de esta forma podemos visualizar con más a detalle cómo se relacionan los componentes internos de nuestro sistema junto a un diagrama que ilustra la secuencia de ingresar a una pantalla que ilustre cualquier tipo de información que el usuario desee. Así, tendremos una visualización clara de cómo el servicio actuara en base a los datos recibidos por los demás microservicios de nuestra aplicación y el cómo sería el proceso de almacenamiento y generación de reportes para cada tipo de estadística solicitada en los requerimientos de nuestro servicio.



Este diagrama de componentes muestra cómo está organizado el módulo de estadísticas. El frontend, desarrollado con React y TypeScript, cuenta con un componente llamado statistics que se encarga de enviar las solicitudes al backend para generar y visualizar los reportes.

En el backend, el controlador recibe estas solicitudes y las delega al servicio principal que se encarga de generar los reportes. Este servicio, a su vez, se apoya en otros componentes que consultan los datos necesarios desde los repositorios correspondientes y los procesan para crear la información que se mostrará. También existe un componente encargado de exportar los reportes a diferentes formatos.

La base de datos está dividida en colecciones que almacenan los reportes, los indicadores y las fuentes de datos utilizadas. Esta organización permite una separación clara entre la presentación, la lógica del sistema y el almacenamiento, lo que facilita su mantenimiento, escalabilidad y comprensión.



Este diagrama de secuencia representa el flujo general para obtener información de uso del gimnasio por parte de un usuario. El proceso inicia desde el cliente web, que realiza una solicitud para consultar los datos. Esta solicitud pasa por una pasarela de autenticación que se encarga de

generar un token de acceso. Una vez generado, se envía al servicio de estadísticas, que valida el token y luego consulta al servicio del gimnasio la información de uso correspondiente. Finalmente, los datos se devuelven al cliente para ser representados visualmente. El diagrama muestra la interacción entre los distintos componentes del sistema de forma ordenada y clara.

Para el manejo de errores de este módulo, tendremos las siguientes convenciones:

Código HTTP	Mensaje de error	Causa probable
400	"Datos de entrada inválidos"	Validaciones fallidas en el formulario
401	"Usuario no autenticado"	Token inválido o ausente
404	"información no disponible"	Algún servicio caído
404	“Información no encontrada”	Filtro para información inexistente
500	"Error interno del servidor"	Fallo inesperado

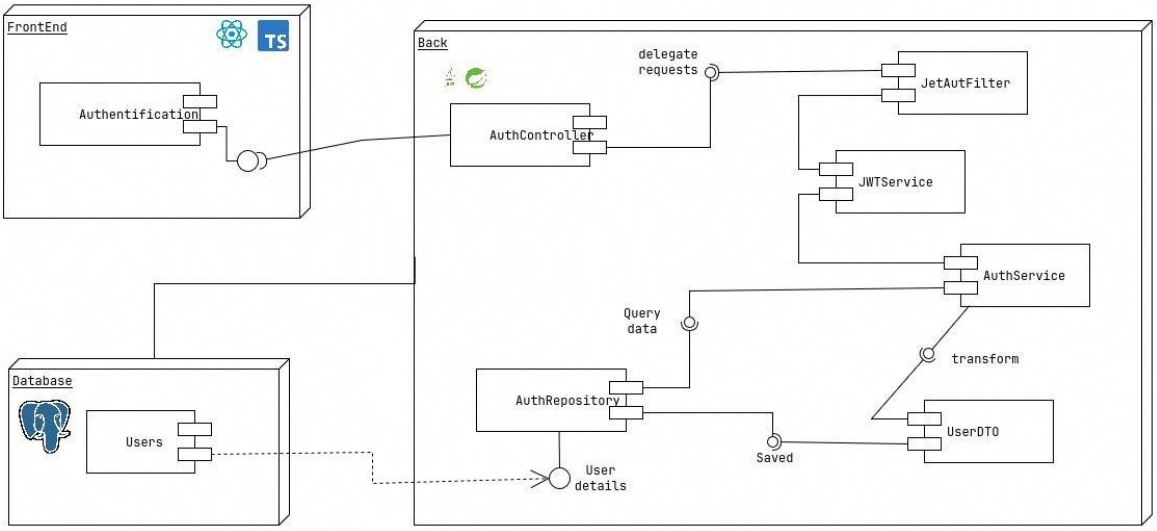
Con toda esta información, podemos decir que el servicio ya quedo preparado para pasar al desarrollo de código, pues se tiene claridad de los comportamientos internos y externos de los componentes, ya que haremos uso de otros microservicios para obtener la información objetivo y prepararla para su representación en el front, así las relaciones con bases de datos y otros servicios quedan cubiertas en nuestro sistema.

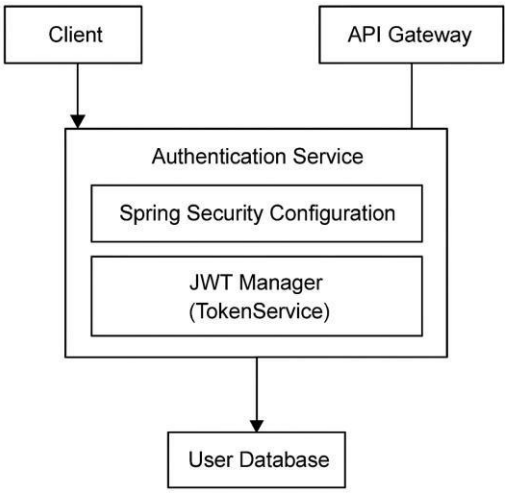
Módulo de Autentificación

Este módulo tiene como objetivo gestionar el acceso seguro de los usuarios al sistema. Este componente es esencial para proteger la integridad y confidencialidad de los datos, asegurando que solo usuarios autorizados puedan acceder a los recursos según sus permisos. Su diseño permite validar credenciales, emitir tokens de acceso y controlar sesiones, funcionando como punto de entrada para los demás módulos del sistema.

Para comenzar, se definió que Spring Boot será el framework estandarizado en este proyecto. Tal como se explicó anteriormente, se busca que todos los desarrolladores dominen esta herramienta para facilitar intervenciones en cualquier squad que lo requiera. Además, esta elección promueve la cohesión entre los distintos módulos y simplifica el mantenimiento general de la aplicación. Por otro lado, para este servicio que maneja la seguridad de la aplicación, podemos usar Spring Security junto a JWT y un encriptado de contraseña para garantizar que nuestra aplicación no sea vulnerable a fugas de datos, ingreso de usuarios no autorizados o manipulación de datos indebida.

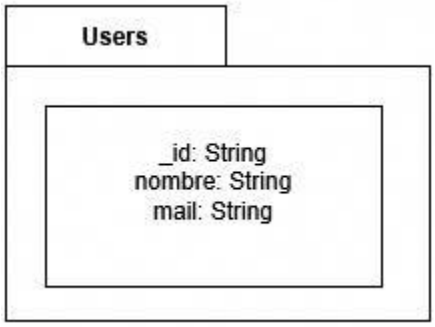
Para comprender la naturaleza de este servicio, se planteamos el siguiente diagrama que busca representar el comportamiento de los distintos componentes involucrados en el tema de seguridad.



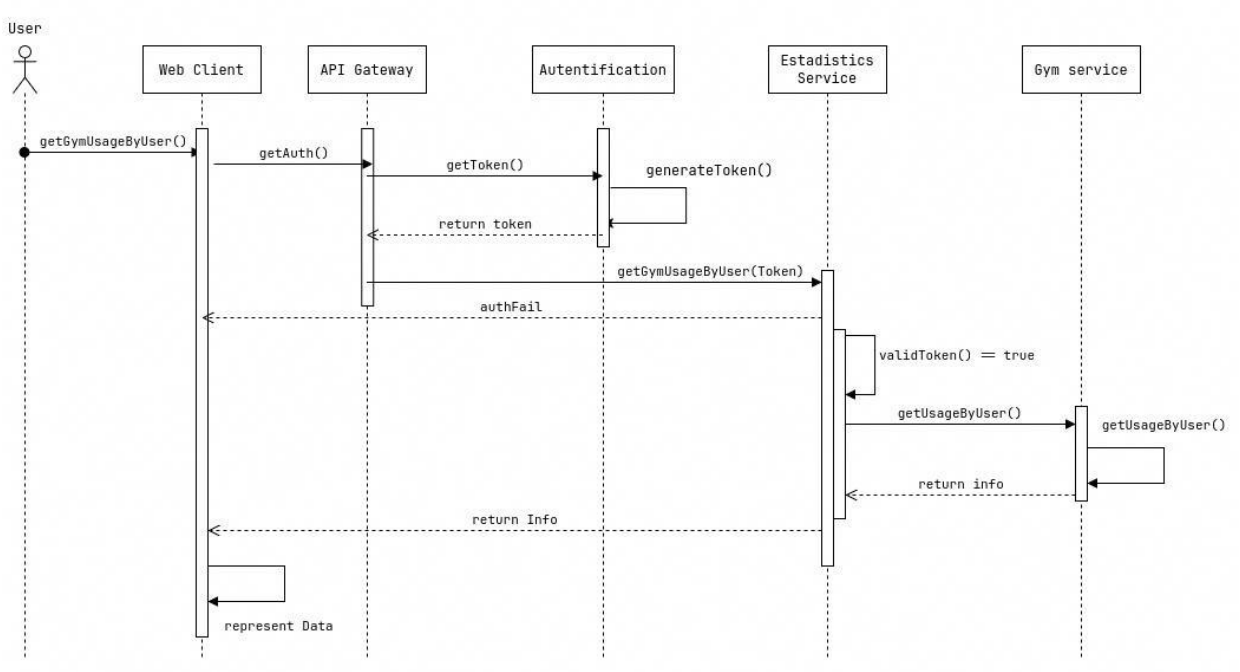


Este diagrama de componentes representa la arquitectura del módulo de autenticación. En el frontend, un componente se encarga de enviar solicitudes de acceso al backend. En el backend, el controlador de autenticación recibe estas solicitudes y las delega al servicio de autenticación, el cual gestiona la lógica principal, incluyendo la validación de credenciales y la transformación de datos de usuario a través de un objeto de transferencia. El repositorio accede a la base de datos para consultar o guardar información del usuario. Además, el servicio de autenticación se apoya en un servicio JWT para generar o validar tokens, y un filtro se encarga de aplicar las reglas de autorización en las solicitudes entrantes.

Se utiliza una base de datos en caché implementada con MongoDB para almacenar exclusivamente los accesos registrados de los usuarios. Esta base actúa como un complemento ligero y flexible a la base de datos principal de usuarios, permitiendo consultar rápidamente eventos de autenticación recientes sin afectar el rendimiento general del sistema. Gracias a la naturaleza schemaless de MongoDB, es posible adaptar fácilmente la estructura de los registros según las necesidades, como almacenar metadatos del acceso, ubicaciones, dispositivos o cualquier otra información útil para análisis o auditoría.



El diagrama de secuencia aplicado al módulo de autenticación sigue una estructura similar a la utilizada para solicitar información en el módulo de estadísticas, pero adaptado a la lógica de acceso seguro al sistema. En este caso, el usuario envía sus credenciales desde el frontend, que son recibidas por el backend a través del controlador de autenticación. Este, a su vez, delega la validación al servicio de autenticación, que consulta la base de datos para verificar que el usuario exista y que su contraseña sea correcta.



Si las credenciales son válidas, el sistema genera un token JWT, el cual se envía de vuelta al frontend para que lo utilice en futuras solicitudes. A partir de ese momento, cada petición del usuario incluirá este token, y será interceptada por un filtro que se encarga de validarlo antes de permitir el acceso al resto del sistema. Este flujo garantiza que el acceso a los recursos protegidos sea controlado de forma segura, manteniendo una experiencia consistente con la arquitectura general del sistema.

Para el manejo de errores de este módulo, tendremos las siguientes convenciones:

Código HTTP	Mensaje de error	Causa probable
400	"Datos de entrada inválidos"	Validaciones fallidas en el formulario
404	"Credencial invalida"	Los datos de credencial no son validos
404	“Usuario no existe”	No se encuentra al usuario
500	"Error interno del servidor"	Fallo inesperado

En este módulo, es fundamental garantizar una arquitectura segura, modular y de fácil mantenimiento. Dado que se gestionan datos sensibles como credenciales de usuario, se requiere un manejo estructurado y relacional de la información, por lo cual se utilizará una base de datos relacional para garantizar integridad, consistencia y facilidad en la gestión de relaciones entre usuarios, roles, permisos y tokens. Esta, es la misma de los usuarios comunes y corrientes, pues ambos servicios están muy relacionados.

Referencias

Anderson, B., & Nicholson, B. (2025, abril 17). SQL vs. NoSQL databases: What's the difference? *Ibm.com*. https://www.ibm.com/think/topics/sql-vs-nosql

Azure Event Hubs para Apache Kafka - Azure Event Hubs. (s/f). Microsoft.com. Recuperado el 5 de mayo de 2025, de https://learn.microsoft.com/es-es/azure/event-hubs/azure-event-hubs-apache-kafka-overview

ConoSurTech [@ConoSurTech]. (s/f). *#AzureFundamentals 2020 | introducción a azure service bus*. Youtube. Recuperado el 5 de mayo de 2025, de https://www.youtube.com/watch?v=Zq6j9apvvS4&t=17s

Framework Training. (2024, mayo 30). *Spring boot vs. Node.Js vs. .NET core vs. Django: A "celebrity face-off"*. Framework Training. https://www.frameworktraining.co.uk/news-insights/spring-boot-vs-node-js-versus-dot-net-core-vs-django/

Introducción a Azure Service Bus, un agente de mensajes empresarial - Azure Service Bus. (s/f). Microsoft.com. Recuperado el 5 de mayo de 2025, de https://learn.microsoft.com/es-es/azure/service-bus-messaging/service-bus-messaging-overview

¿Qué es Java Spring Boot? (2023, julio 17). *Ibm.com*. https://www.ibm.com/mx-es/topics/java-spring-boot

Smallcombe, M. (s/f). *SQL vs NoSQL: 5 critical differences*. Integrate.Io. Recuperado el 5 de mayo de 2025, de https://www.integrate.io/blog/the-sql-vs-nosql-difference/

Spring cloud OpenFeign. (s/f). Spring Cloud OpenFeign. Recuperado el 5 de mayo de 2025, de https://spring.io/projects/spring-cloud-openfeign

Spring security. (s/f). Spring Security. Recuperado el 5 de mayo de 2025, de https://spring.io/projects/spring-security

Written by Emorphis Technologies, & Emorphis Technologies. (2025, enero 14). *Django vs Laravel vs .NET – choosing the right framework for your project*. Blogs.Emorphis; Emorphis Technologies. https://blogs.emorphis.com/django-laravel-net/