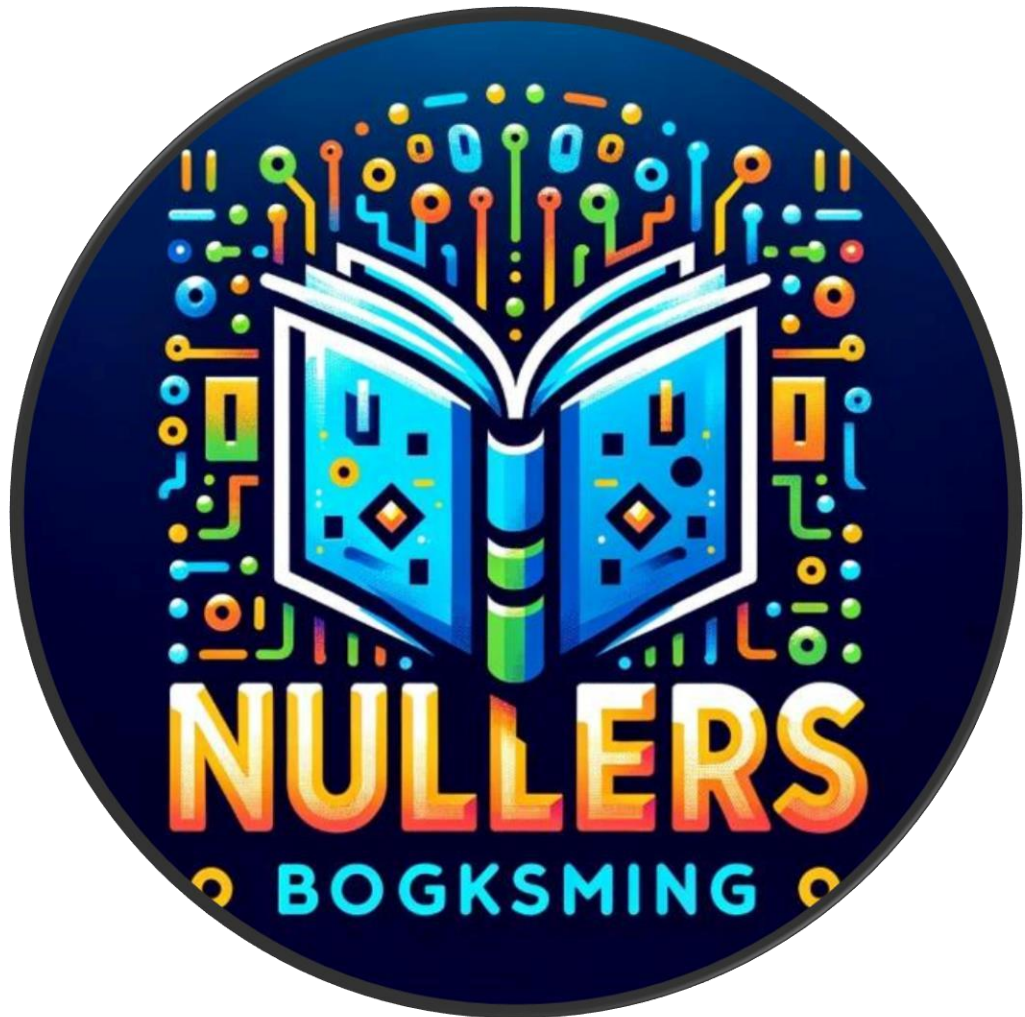


12 DE FEBRERO DE 2024



NULLERS BOOKS

POR NULLERS

NULLERS

Contenido

Descripción	4
Infraestructura	5
Elección de Tecnologías para el Modelo de Datos.....	6
Modelo Relacional para Pedidos y Líneas de Pedido.....	7
SQL para el Resto de Entidades	7
Dependencias	8
Lista de endpoints.....	9
Arquitectura	11
Estimación de gastos	13
Mano de Obra	13
Equipo de Desarrollo	13
Infraestructura	13
Plan AWS.....	13
Detalles del Plan AWS:.....	13
Servidor: T2.Medium	13
Configuración: Optimizada para el proyecto.....	13
Costo Total Estimado	13
Análisis tecnológico	14
Elección de PostgreSQL y MongoDB: Justificación para el Documento Profesional.....	14
PostgreSQL.....	14
Integridad y Confiabilidad:	14
Modelo Relacional Potente:	14
Soporte para Transacciones Complejas	14
MongoDB.....	14
Flexibilidad en el Esquema:.....	14
Escalabilidad Horizontal	14
Velocidad en Operaciones de Lectura/Escritura:	15

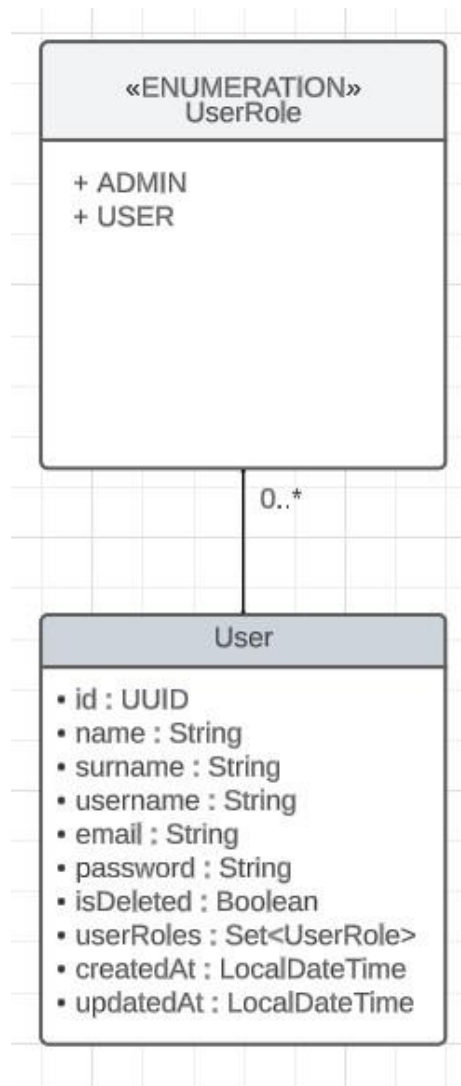
Descripción

API REST para una tienda de libros en línea que permita a los usuarios realizar operaciones como la consulta de libros, gestión de usuarios, administración de tiendas, y realización de pedidos. La API estará diseñada para ser segura, eficiente y escalable, proporcionando una interfaz robusta para interactuar con la plataforma de comercio de libros.

- **Controllers:** Se encargan de recibir las peticiones del usuario y devolver la respuesta correspondiente.
- **Exceptions:** Se encargan de definir las excepciones que se van a utilizar en la aplicación.
- **Models:** Se encargan de definir los objetos que se van a utilizar en la aplicación.
- **Services:** Se encargan de realizar las operaciones necesarias para que el controlador pueda devolver la respuesta.
- **Utils:** Se encargan de definir las clases útiles que se van a utilizar en la aplicación.
- **Main:** El programa que ejecutará la aplicación.



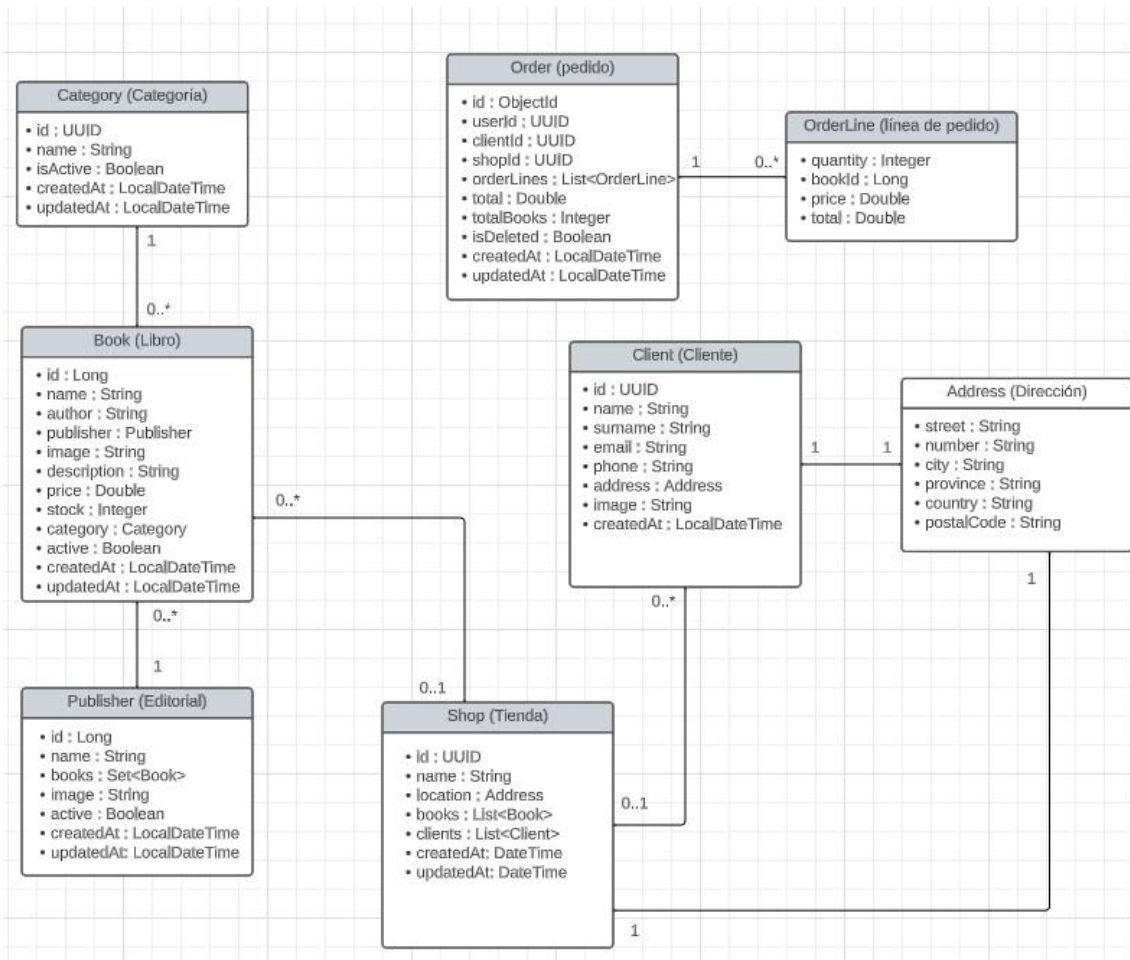
Infraestructura



El usuario tiene un `UserRole`, el cual define el tipo de usuario (si es Admin o User). Dependiendo del tipo de rol, se le otorgará la posibilidad de realizar ciertas peticiones. Los usuarios pueden realizar consultas GET en consultas como libros, tiendas o editoriales, pero no pueden realizar peticiones de actualización o eliminación. Los usuarios administradores tienen control para poder realizar estas peticiones.

El usuario cuenta con un email y un username, el cuál no se puede repetir.

Utilizamos el borrado lógico en `isDeleted` para la conservación de los usuarios.



Una tienda tiene una dirección y una lista de libros y clientes.

Un cliente también tiene una dirección.

Un Libro tiene asignada una categoría y una editorial. Las editoriales tienen de 0 a muchos libros.

La editorial, el libro y el cliente cuentan con una imagen.

El libro cuenta con un stock y un precio del libro, así como cada uno de los elementos que componen al libro: nombre, autor, descripción...

Borrado lógico:

- En book empleando *active*.
- En category empleando *isActive*.
- En publisher empleando *active*.
- En order empleando *isDeleted*.

Elección de Tecnologías para el Modelo de Datos

Modelo relacional: Hemos utilizado modelo relacional para los pedidos y líneas de pedido.

SQL: Para el resto de entidades, hemos utilizado SQL.

La elección de utilizar un modelo relacional para los pedidos y líneas de pedido, y SQL para el resto de entidades, se basa en consideraciones específicas relacionadas con la estructura y las operaciones previstas en el sistema.

Modelo Relacional para Pedidos y Líneas de Pedido:

Relaciones Complejas:

El modelo relacional es especialmente adecuado cuando existen relaciones complejas entre las entidades. En el caso de los pedidos y líneas de pedido, donde se pueden tener múltiples productos asociados a un solo pedido, el modelo relacional proporciona una representación clara y eficiente de estas relaciones.

Consistencia y Normalización:

La normalización inherente al modelo relacional ayuda a mantener la consistencia y la integridad de los datos. Al gestionar pedidos, donde es crucial mantener la coherencia de la información, la normalización contribuye a evitar redundancias y posibles incongruencias.

SQL para el Resto de Entidades:

Versatilidad y Escalabilidad:

El uso de SQL permite gestionar de manera eficiente una variedad de operaciones en las diferentes entidades del sistema, desde usuarios y tiendas hasta libros y categorías. SQL es conocido por su versatilidad y escalabilidad, lo que facilita la manipulación y consulta de datos en un amplio espectro de situaciones.

Consulta y Manipulación de Datos:

SQL proporciona un lenguaje poderoso para la consulta y manipulación de datos. Esto es esencial para operaciones como la obtención de información del perfil de un usuario, la actualización de datos de una tienda o la gestión de libros y categorías.

Dependencias

- NestJS TypeORM o Sequelize: NestJS integra ORM como TypeORM o Sequelize para el manejo de bases de datos relacionales, ofreciendo una experiencia similar a Spring Data JPA.
- NestJS Core Package: NestJS viene con capacidades incorporadas para desarrollar aplicaciones web y manejar solicitudes HTTP, similar a Spring Boot Starter Web.
- Cache Manager: NestJS puede utilizar cache-manager, un paquete de NPM, para implementar la caché en la aplicación.
- Class Validator: NestJS utiliza class-validator para la validación de datos, junto con class-transformer para transformar y validar objetos de datos basados en clases de TypeScript.
- Jest: NestJS utiliza Jest como su marco de pruebas predeterminado, permitiendo pruebas unitarias y de integración.
- Fast-XML-Parser: Para el manejo de XML en aplicaciones NestJS, se puede utilizar fast-xml-parser o bibliotecas similares disponibles en NPM.
- @nestjs/websockets: NestJS proporciona el paquete @nestjs/websockets para trabajar con WebSockets, facilitando la comunicación en tiempo real.
- Day.js o Luxon: Para manejo de fechas en NestJS, se pueden usar bibliotecas como Day.js o Luxon, que ofrecen una amplia gama de funciones para manejar fechas y horas.
- @nestjs/passport: Para la seguridad, NestJS se integra con Passport, utilizando el paquete @nestjs/passport para autenticación, junto con @nestjs/jwt para manejar JWT
- Supertest: Para pruebas que involucren la seguridad y la API, se puede utilizar Supertest junto con Jest en NestJS.
- @nestjs/jwt: Este paquete se utiliza en NestJS para la creación y validación de JSON Web Tokens, similar a la biblioteca Java JWT en Spring.
- @nestjs/swagger: NestJS utiliza el paquete @nestjs/swagger para generar documentación de la API compatible con OpenAPI.
- @nestjs/mongoose: Para integrar MongoDB en NestJS, se utiliza el paquete @nestjs/mongoose, que facilita el trabajo con MongoDB mediante Mongoose.
- mongodb-memory-server: Para pruebas que requieren MongoDB, mongodb-memory-server ofrece una base de datos MongoDB en memoria para Node.js.
- sqlite3: Para bases de datos en memoria en NestJS, se puede utilizar SQLite con TypeORM para un propósito similar al de ED en Spring.
- TypeScript Decorators: TypeScript en sí ofrece decoradores y otras características que reducen la necesidad de código boilerplate, similar a lo que Lombok hace en Java.

Lista de endpoints:

Shops:

- GET /api/shops/{id}: Obtiene una tienda por su ID.
- PUT /api/shops/{id}: Actualiza una tienda existente.
- DELETE /api/shops/{id}: Elimina una tienda por su ID.
- GET /api/shops: Obtiene todas las tiendas.
- POST /api/shops: Crea una nueva tienda.
- DELETE /api/shops/{id}/clients/{clientId}: Elimina un cliente de una tienda.
- PATCH /api/shops/{id}/clients/{clientId}: Añade un cliente a una tienda.
- DELETE /api/shops/{id}/books/{bookId}: Elimina un libro de una tienda.
- PATCH /api/shops/{id}/books/{bookId}: Añade un libro a una tienda.

Publishers:

- GET /api/publishers/{id}: Obtiene una editorial dado un ID.
- PUT /api/publishers/{id}: Actualiza una editorial.
- DELETE /api/publishers/{id}: Elimina una editorial.
- GET /api/publishers: Obtiene todas las editoriales.
- POST /api/publishers: Crea una editorial.
- PATCH /api/publishers/image/{id}: Actualiza la imagen de una editorial.

Orders:

- GET /api/orders/{id}: Obtiene un pedido dado un ID.
- PUT /api/orders/{id}: Actualiza un pedido.
- DELETE /api/orders/{id}: Elimina un pedido.
- PUT /api/orders/delete/{id}: Elimina un pedido de manera simulada.
- GET /api/orders: Obtiene todos los pedidos.
- POST /api/orders: Crea un pedido.
- GET /api/orders/user/{id}: Obtiene un pedido dado el ID de un usuario.
- GET /api/orders/shop/{id}: Obtiene un pedido dado el ID de una tienda.
- GET /api/orders/client/{id}: Obtiene un pedido dado el ID de un cliente.

Clients:

- GET /api/clients/{id}: Obtiene un cliente dado un ID.
- PUT /api/clients/{id}: Actualiza un cliente.
- DELETE /api/clients/{id}: Elimina un cliente.
- GET /api/clients: Obtiene todos los clientes.
- POST /api/clients: Crea un cliente.
- PATCH /api/clients/{id}/image: Actualiza la imagen de un cliente.
- GET /api/clients/email/{email}: Obtiene un cliente dado un email.

Categories:

- GET /api/categories/{id}: Busca una categoría por ID.
- PUT /api/categories/{id}: Actualiza una categoría.
- DELETE /api/categories/{id}: Borra una categoría.
- GET /api/categories: Obtiene todas las categorías.
- POST /api/categories: Crea una categoría.

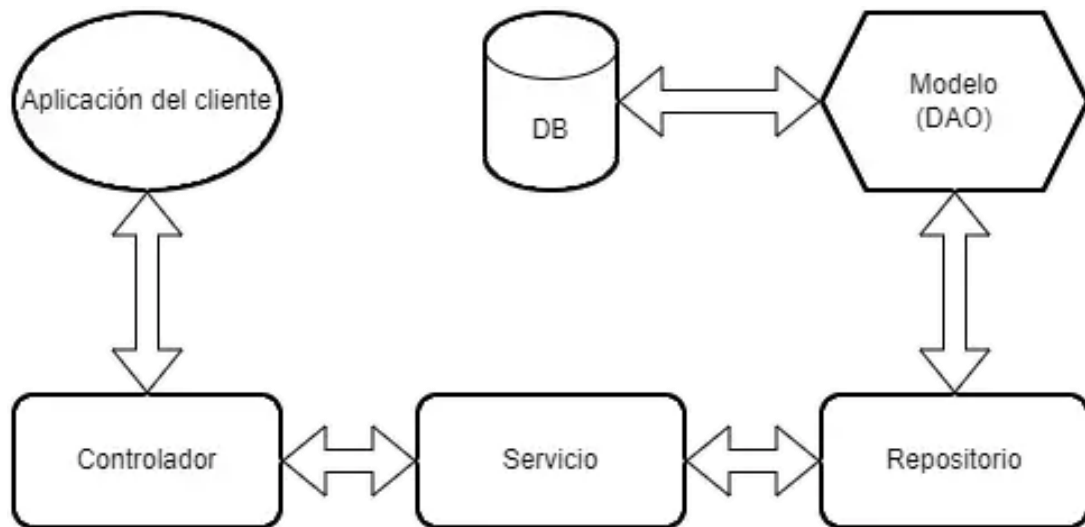
Books:

- GET /api/books/{id}: Busca un libro dado su ID.
- PUT /api/books/{id}: Actualiza un libro.
- DELETE /api/books/{id}: Borra un libro.
- PATCH /api/books/{id}: Actualiza un libro parcialmente.
- GET /api/books: Obtiene todos los libros.
- POST /api/books: Crea un libro.
- PATCH /api/books/image/{id}: Actualiza la imagen de un libro.

Authentication:

- POST /api/auth/signup: Crea una cuenta.
- POST /api/auth/signin: Inicia sesión.

Arquitectura



1. Controladores (controllers):

- Los controladores manejan las solicitudes HTTP y son responsables de interactuar con el cliente. En este caso, los controladores se encuentran en paquetes como `user-controller`, `shop-rest-controller-impl`, etc. Cada controlador maneja un conjunto específico de operaciones relacionadas con su entidad correspondiente (usuarios, tiendas, pedidos, etc.).

2. Servicios (services):

- Los servicios contienen la lógica de negocio de la aplicación. Separan la lógica del controlador para mantener un código limpio y modular. Cada controlador interactúa con los servicios correspondientes para realizar operaciones más complejas. En la arquitectura, podrías encontrar paquetes como `services` o `impl` que albergan las implementaciones concretas de los servicios.

3. Modelos (models):

- Los modelos representan las entidades del dominio de la aplicación y se utilizan para mapear datos entre la aplicación y la base de datos. Los modelos se encuentran en paquetes como `models` o `entities`. Pueden contener anotaciones de mapeo para trabajar con un sistema de gestión de bases de datos (como JPA en este caso).

4. Excepciones (exceptions):

- En este paquete, podrías encontrar clases que representan excepciones específicas de la aplicación. Estas excepciones pueden ser lanzadas y capturadas en distintas capas de la aplicación para manejar situaciones excepcionales de manera específica y proporcionar respuestas HTTP adecuadas.

5. DTO (Objetos de Transferencia de Datos) (dto):

- Los DTO son objetos que se utilizan para transferir datos entre las capas de la aplicación. Ayudan a desacoplar las representaciones internas de los datos (modelos) de las representaciones externas (como las respuestas HTTP). Pueden encontrarse en paquetes como `dto` o `requestresponse`.
6. **Mappers (`mappers`):**
- Los mappers se utilizan para convertir entre diferentes objetos, como convertir de modelos a DTO y viceversa. Estos ayudan a separar las preocupaciones de mapeo y proporcionan una forma estructurada de transformar datos.
7. **Utilidades (`utils`):**
- Este paquete podría contener clases de utilidades que proporcionan funciones comunes reutilizables en toda la aplicación, como operaciones de manipulación de fechas, generación de identificadores únicos, etc.
8. **Almacenamiento (`storage`):**
- Puede contener clases o utilidades relacionadas con el almacenamiento de archivos o datos fuera de la base de datos, como el manejo de archivos de imágenes para libros o imágenes de perfil.
9. **Notificaciones (`notifications`):**
- En algunos casos, puede haber un paquete dedicado a manejar notificaciones o eventos dentro de la aplicación, como el envío de correos electrónicos, mensajes de texto, o cualquier otro tipo de notificación.

Estimación de gastos

Mano de Obra

Para la ejecución del proyecto, se planifica una inversión en mano de obra basada en una estimación de 120 horas de trabajo, distribuidas entre cinco programadores. Se establece un costo promedio por hora de 20 € para el equipo completo.

Equipo de Desarrollo

- Desarrollador 1: 24 horas
- Desarrollador 2: 24 horas
- Desarrollador 3: 24 horas
- Desarrollador 4: 24 horas
- Desarrollador 5: 24 horas

Costo Estimado de Mano de Obra: 120 horas * 20 €/hora = 2.400 €

Infraestructura

Se busca la máxima escalabilidad y facilidad de uso, un servicio en la nube sería la mejor opción.

La infraestructura del proyecto implica el despliegue de la aplicación en un entorno cloud. Se ha decidido utilizar servicios de Amazon Web Services (AWS) para cumplir con los requisitos de hosting. El cálculo del costo de infraestructura se realiza en base al tipo de servidor necesario y la capacidad requerida. En esta estimación, se considera el uso de un servidor de tamaño medio.

Tipo de Servidor: T2.Medium.

Capacidad: Ajustada a los requisitos del proyecto.

Costo Estimado de Infraestructura en AWS: Aproximadamente 100 € mensuales.

Plan AWS

El plan de AWS se centra en la implementación de la aplicación en un servidor de capacidad adecuada para garantizar un rendimiento óptimo. Se elige un servidor de tamaño medio para cubrir las necesidades específicas del proyecto.

Tipo: T2.Medium

vCPUs: 2

Memoria: 4 GiB

Almacenamiento: EBS 80 GB SSD

Sistema operativo: Linux

Precio:

Costo por hora: \$0.0524

Costo mensual: \$37.71 (estimado para 730 horas)

\$X: Costo de la red (\$0.01 por GB de salida)

\$Y: Costo del almacenamiento adicional (\$0.05 por GB)

\$Z: Costo de la base de datos (RDS MySQL t2.medium: \$38.20 por mes)

Estimación del costo total:

Costo mensual: \$37.71 (servidor) + \$0.01 (red) + \$0.05 (almacenamiento adicional) + \$38.20 (base de datos) = \$75.97

Explicación de las variables:

\$X: Se estima un tráfico de 10 GB por mes, lo que resulta en un costo de \$0.10.

\$Y: Se estima un almacenamiento adicional de 20 GB por mes, lo que resulta en un costo de \$1.00.

\$Z: Se estima la necesidad de una base de datos RDS MySQL t2.medium, que tiene un costo mensual de \$38.20.

Costo Total Estimado:

La suma de los costos de mano de obra y de infraestructura proyecta un gasto total estimado de 2.475,97 €. Esta estimación proporciona una visión general de los recursos financieros necesarios para la ejecución exitosa del proyecto.

- **Desglose del Costo Total:**
 - Mano de Obra: 2.400 €
 - Infraestructura AWS: 75.97€

Análisis tecnológico

Elección de PostgreSQL y MongoDB: Justificación para el Documento Profesional

En la selección de las bases de datos para nuestro proyecto, hemos optado por integrar tanto PostgreSQL como MongoDB debido a sus características distintivas y complementarias.

PostgreSQL:

Integridad y Confiabilidad:

PostgreSQL destaca por su integridad y confiabilidad en entornos de datos críticos. Su implementación de transacciones ACID asegura la consistencia y durabilidad de los datos, elementos cruciales para aplicaciones empresariales.

Modelo Relacional Potente:

La estructura relacional de PostgreSQL proporciona flexibilidad en la organización de datos complejos, permitiendo consultas sofisticadas y optimizadas. Esto es esencial para aplicaciones que requieren un alto grado de coherencia en la representación de la información.

Soporte para Transacciones Complejas:

La capacidad de gestionar transacciones complejas es esencial en situaciones donde la integridad de los datos es primordial. PostgreSQL brinda soporte robusto para operaciones transaccionales avanzadas, asegurando la consistencia en escenarios de alta concurrencia.

MongoDB:

Flexibilidad en el Esquema:

MongoDB se distingue por su enfoque de esquema flexible basado en documentos BSON. Esta flexibilidad es crucial para proyectos donde la estructura de los datos puede evolucionar con el tiempo, facilitando adaptaciones rápidas a cambios en los requisitos.

Escalabilidad Horizontal:

MongoDB ofrece una excelente capacidad de escalabilidad horizontal, ideal para entornos que requieren crecimiento sin problemas a medida que la carga de datos aumenta. La capacidad de distribuir datos de manera eficiente garantiza un rendimiento sostenible a medida que la aplicación se expande.

Velocidad en Operaciones de Lectura/Escritura:

En situaciones donde la velocidad de lectura y escritura es crítica, MongoDB destaca al proporcionar operaciones rápidas y eficientes en entornos de alto rendimiento.

La combinación de PostgreSQL y MongoDB permite abordar de manera integral los requisitos específicos de nuestro proyecto, garantizando tanto la integridad y confiabilidad de los datos como la flexibilidad necesaria para adaptarse a futuras demandas. Esta elección estratégica está respaldada por la complementariedad de las fortalezas individuales de cada sistema de gestión de bases de datos.