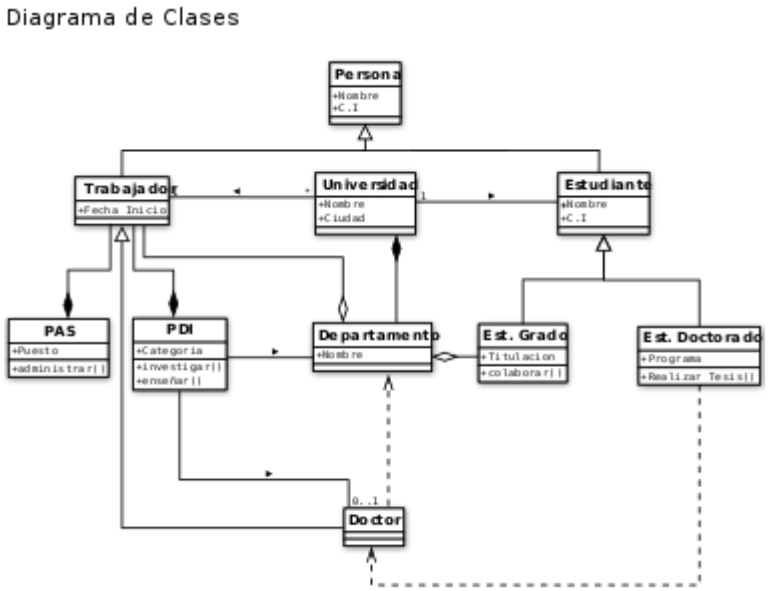
DISEÑO DE LA SOLUCIÓN

1. Diagrama de clases

Un diagrama de clases en Lenguaje Unificado de Modelado (UML) es un tipo de estructura estática que describe la estructura de un sistema mostrando las clases de este, sus atributos, operaciones (o métodos) y las relaciones entre los objetos[1].

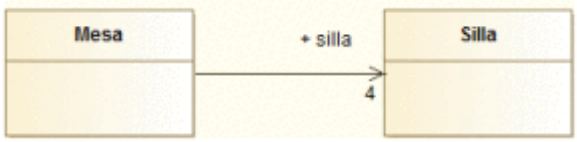


En cuanto a las clases, podemos definir las mismas como una plantilla para la creación de objetos de datos según un modelo predefinido. Cada clase es un modelo que define un conjunto de variables (estado) y métodos para operar con dichos datos (comportamiento). Cada objeto creado a partir de la clase se conoce como instancia de la clase[2].

En cuanto a las relaciones entre los objetos, son el tercer pilar fundamental del diagrama de clases. Pueden ser binarias o de orden superior. Si dos clases están relacionadas significa que esas clases tienen algo que ver entre sí[3].

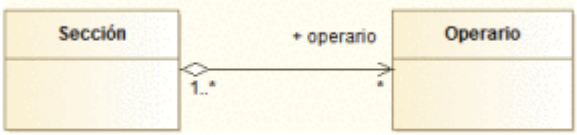
Hay varios tipos de relación:

- Asociación: Se utiliza para expresar simplemente que dos clases están vinculadas entre sí.

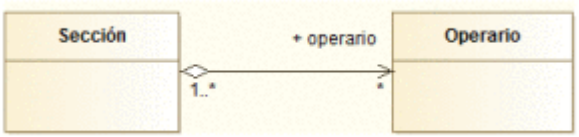


[1] https://es.wikipedia.org/wiki/Diagrama_de_clases
[2] https://es.wikipedia.org/wiki/Clase_(inform%C3%A1tica)
[3] https://joanpaon.wordpress.com/2013/06/06/uml-diagramas-de-clases-relacion/

- Pertenencia: Se apoya en el punto de vista del binomio [PARTE] – [TODO]. El componente [PARTE] se caracteriza porque es una pieza del componente [TODO], teniendo este último la capacidad de albergar al componente [PARTE], integrándolo dentro de sí mismo.



- Autonomía: La autonomía define qué ciclo de vida tienen los objetos de la clase [TODO] y cuál tienen los objetos de la clase [PARTE]. En concreto, la regla para determinar el tipo de asociación es fijarse en el ciclo de vida de los objetos de la clase [PARTE] cuando la clase [TODO] se destruye. ¿Qué ocurre con los objetos de la clase [PARTE] cuando la clase [TODO] desaparece?, la respuesta da lugar a dos tipos de asociación:
    - ➔ Agregación: Cuando el objeto [TODO] se destruye, los objetos [PARTE] pueden seguir existiendo de forma autónoma.

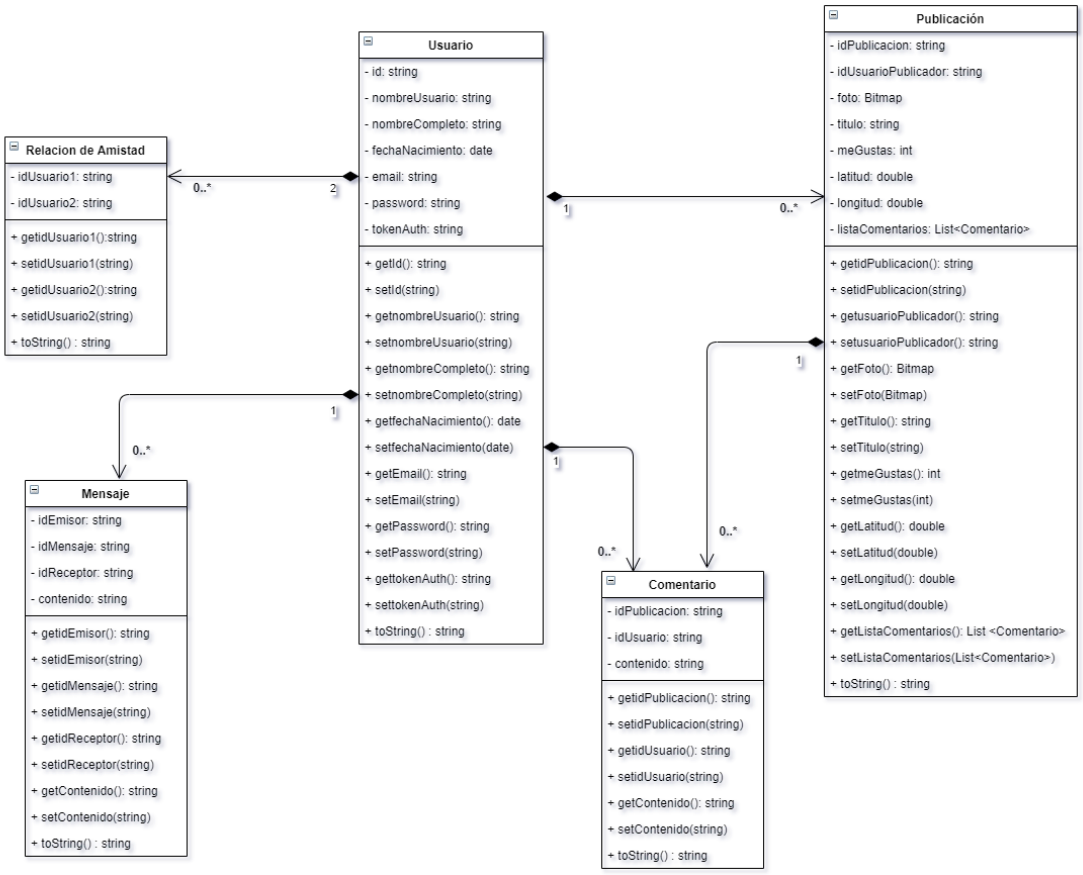


      En la agregación, la clase [TODO] se representa con un rombo en blanco y la clase [PARTE] se identifica con una flecha de navegación.
    - ➔ Composición: Cuando el objeto [TODO] se destruye, los objetos [PARTE] desaparecen igualmente, ya que su existencia no tiene sentido.

      En la composición, la clase [TODO] se representa con un rombo en negro y la clase [PARTE] se identifica con una flecha de navegación.

1.1. Diagrama de clases de Instadroid

A continuación, muestro el diagrama de clases de la aplicación Instadroid:



Como se puede observar, todas las relaciones de la aplicación Instadroid son relaciones de composición, porque ¿qué sentido tiene que permanezcan en la aplicación las publicaciones de un usuario, sus comentarios, sus mensajes y sus relaciones de amistad si el usuario es eliminado?

Voy a explicar una por una la naturaleza de estas relaciones.

Usuario ←→ publicación

En Instadroid, los usuarios suben publicaciones a la aplicación, por lo que la publicación está asociada al usuario que la sube.

- Tipo de asociación: Composición, ya que, si el usuario desaparece, sus publicaciones también.
- Cardinalidad: Una publicación pertenece a un usuario, pero un usuario puede subir (o no) varias publicaciones 1 → 0..*

Usuario ←→ Comentario ←→ Publicación

Los usuarios comentan las publicaciones subidas a Instadroid.

- Tipo de asociación: Composición, ya que, si el usuario o la publicación comentada desaparecen, los comentarios también.
- Cardinalidad: Un comentario es realizado por un usuario, pero un usuario puede realizar (o no) varios comentarios 1 → 0..*. Un comentario pertenece a una publicación, y una publicación puede tener ninguno o N comentarios 1 → 0..*.

Usuario ←→ Mensaje

En Instadroid, los usuarios envían y reciben mensajes (pueden ser emisores o receptores en un mensaje concreto).

- Tipo de asociación: Composición, ya que, si el usuario desaparece, sus mensajes también.
- Cardinalidad: Un mensaje es enviado/recibido por un usuario y un usuario puede enviar/recibir ninguno o varios mensajes 1→ 0..*.

Usuario ←→ relación de amistad

En Instadroid, los usuarios pueden enviar y recibir solicitudes de amistad. Las relaciones de amistad se componen por las id's de los dos usuarios que son amigos.

- Tipo de asociación: Composición, ya que, si el usuario desaparece, sus relaciones de amistad, también.
- Cardinalidad: Una relación de amistad pertenece a dos usuarios y los usuarios pueden tener a su vez varias relaciones de amistad 2 → 0..*.

1.2. Diseño de la persistencia de la información

Se denomina persistencia a la capacidad de guardar la información de un programa para poder volver a utilizarla en otro momento[4]. Esto puede significar guardar los datos en un fichero (de texto, binario, csv) o guardar los datos en una base de datos (y sus distintas alternativas)

[4] https://uniwebsidad.com/libros/algoritmos-python/capitulo-11/persistencia-de-datos

Para el desarrollo de Instadroid, distintos factores me han llevado a elegir la opción de utilizar la base de datos NoSQL de Google Firebase.

Antes que nada, debo contextualizar las cosas.

En primer lugar **¿qué es una base de datos NoSQL?**

NoSQL significa “not only SQL”, es decir, no sólo SQL. No es un modelo antagónico, si no un enriquecimiento y complemento útil de las tradicionales bases de datos SQL relacionales[5]. NoSQL plantea modelos de datos específicos de esquemas flexibles que se adaptan a los requisitos de las aplicaciones más modernas[6].

Este tipo de bases de datos surgió debido a las limitaciones y problemas de las bases de datos relacionales, que no son capaces de hacer frente a las exigencias del desarrollo moderno. En cambio, las bases de datos NoSQL utilizan novedades, como los servidores en la nube y estructuras de datos muy potentes y flexibles.

El funcionamiento de estas bases de datos parte de la premisa de no usar tablas tradicionales y rígidas para almacenar los datos. En su lugar, organizan grandes volúmenes de datos con técnicas más flexibles como por ejemplo documentos y pares de clave valor.

Una de las particularidades de los sistemas NoSQL es el escalamiento horizontal. Para entender este concepto es necesario saber que las bases de datos tradicionales escalan de manera vertical, es decir, toda su capacidad de rendimiento se basa en un solo servidor, por lo que para aumentar su capacidad hay que invertir en un servidor más potente (una opción muy cara en el largo plazo). El escalamiento horizontal supone que las soluciones NoSQL distribuyen sus datos en varios servidores, por lo que si necesitamos manejar más volumen de datos con un servidor austero sería suficiente. De esta manera, pueden almacenar grandes cantidades de datos a un precio menor.

Para terminar de entender bien el concepto de base de datos NoSQL, debemos atender a sus cuatro conceptos más importantes:

- Bases de datos orientadas a documentos

  Los datos se almacenan directamente en documentos de diferentes longitudes, sin ser necesario que los datos estén estructurados. Se les asigna distintos atributos sobre los cuales se puede rastrear los contenidos de los documentos. Son especialmente indicados para gestión de contenidos y blogs. Utilizan JSON como

[5] https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/nosql/
[6] https://www.grapheverywhere.com/bases-de-datos-nosql-marcas-tipos-ventajas/

formato de archivo ya que permite el intercambió de datos más rápido entre aplicaciones.

- Bases de datos de grafos

  Una base de datos de grafos forma relaciones entre datos mediante nodos. Así con contenidos de información muy entrelazada, ofrecen mejor rendimiento que las SQL relacionales. Se usan sobre todo en el ámbito de las redes sociales para representar relaciones entre seguidores.

- Bases de datos clave-valor

  Guardan los datos en pares clave-valor. Los valores individuales están asignados a claves específicas y el propio juego de datos funciona como clave y representa un valor. La clave crea un índice que se puede usar para buscar en la base de datos. Estas claves un unívocas y se pueden comparar a las Primary Key de las bases de datos relacionales.

- Base de datos orientadas a columnas

  A diferencia de SQL, no se guardan los registros en líneas, sino en columnas. Este cambio implica procesos de lectura mucho más cortos y una mayor capacidad de rendimiento. Este modelo se utiliza para minería y análisis de datos.

En segundo lugar **¿por qué decido utilizar una base de datos NoSQL?**

Para ello vamos a realizar un análisis de los conceptos más importantes de la persistencia de datos[7].

- **Integridad de datos**

Es la garantía de que los datos almacenados mantendrán su exactitud y consistencia en el tiempo.

➔ En SQL las tablas tienen estructuras rígidas, donde cada dato es de un tipo previamente definido, por lo que, si tu código funciona en un registro, debe funcionar en todos los demás

➔ En NoSQL no hay que definir el tipo de dato a almacenar, en un momento puedes almacenar un número y después almacenar una string o un array. Más que en los propios datos, NoSQL da la prioridad a cómo vamos a acceder a los mismos.

[7] https://medium.com/@eugeniomendoza/c%C3%B3mo-saber-si-necesitas-una-base-de-datos-nosql-b6cfd5bb7d9b

Si necesitamos que nuestros datos se mantengan exactos y consistentes, utilizaremos SQL. Esto es ideal en sistemas intolerantes a los fallos (por ejemplo, bancos, empresas).

Si nuestras estructuras de datos son propensas a cambiar, puede ser problemático en SQL ya que modificar la estructura rígida de la base de datos puede darnos más de un problema. En este caso será mejor utilizar NoSQL, ya que añadir claves nuevas a un documento no supone ningún problema y es sencillo.

- **Operaciones atómicas**

Se trata de cuando realizamos un cambio que afecta a múltiples entidades de la base de datos al mismo tiempo. Se acompaña del concepto de transacciones y los famosos commits y rollbacks.

- ➔ En SQL, como las tablas están conectadas, pueden "ponerse de acuerdo" para no aceptar nuevos cambios hasta que termine la transacción.
- ➔ NoSQL, al no existir las fuertes relaciones entre los datos que hay en SQL, no hay posibilidad de hacer una transacción atómica. Cuando queremos cambiar algo en 5 entidades diferentes, tendremos que hacer 5 llamadas a la base de datos.

Si nuestra aplicación maneja operaciones críticas como pagos a la vez del mismo producto, por ejemplo, SQL nos cubre las espaldas mejor, pero si no es el caso es mejor acudir a NoSQL porque la atomicidad no siempre es crucial… o ¿acaso es importante que dos usuarios difieran en la cantidad de likes de una foto?

- **Escalabilidad**

Se trata de lo que he expuesto justo antes de la escalabilidad horizontal de las bases de datos NoSQL. Esto viene muy bien principalmente en las primeras fases de crecimiento de una aplicación ya que expandir el volumen de datos a manejar será barato.

- **Velocidad**

¿Cómo de rápido leemos y escribimos en la base de datos?

- ➔ En SQL, las garantías que otorgan las relaciones conllevan un precio. En cuanto comenzamos a hacer consultas con muchas joins que involucran múltiples entidades aumentamos el tiempo de búsqueda.
- ➔ En NoSQL, se suele contar con mecanismos de búsqueda sumamente rápida para conseguir un dato específico entre millones. La principal

ventaja en la velocidad es que puedes diseñar la base de datos en función de las consultas que le haremos.

- **Consistencia vs redundancia**
    - ➔ En SQL, consiste en asegurarse de que un único dato esté una única vez en toda la base de datos.
    - ➔ En NoSQL, la redundancia es repetir adrede los datos a conveniencia en varias partes de la base de datos. Por ejemplo, en mi caso yo tendré un documento con usuarios y otro con publicaciones. Cuando se muestran las publicaciones en la app también tienen datos de los usuarios, por lo que podría repetir los datos del usuario que ha subido la publicación (nombre de usuario, foto de perfil) y meterlos también como datos de la propia publicación. De esta forma gano velocidad porque no tengo que realizar dos consultas para simular una "relación".

Conclusión

Después de esto, creo que se puede ver claro y justificado el porqué de mi elección.

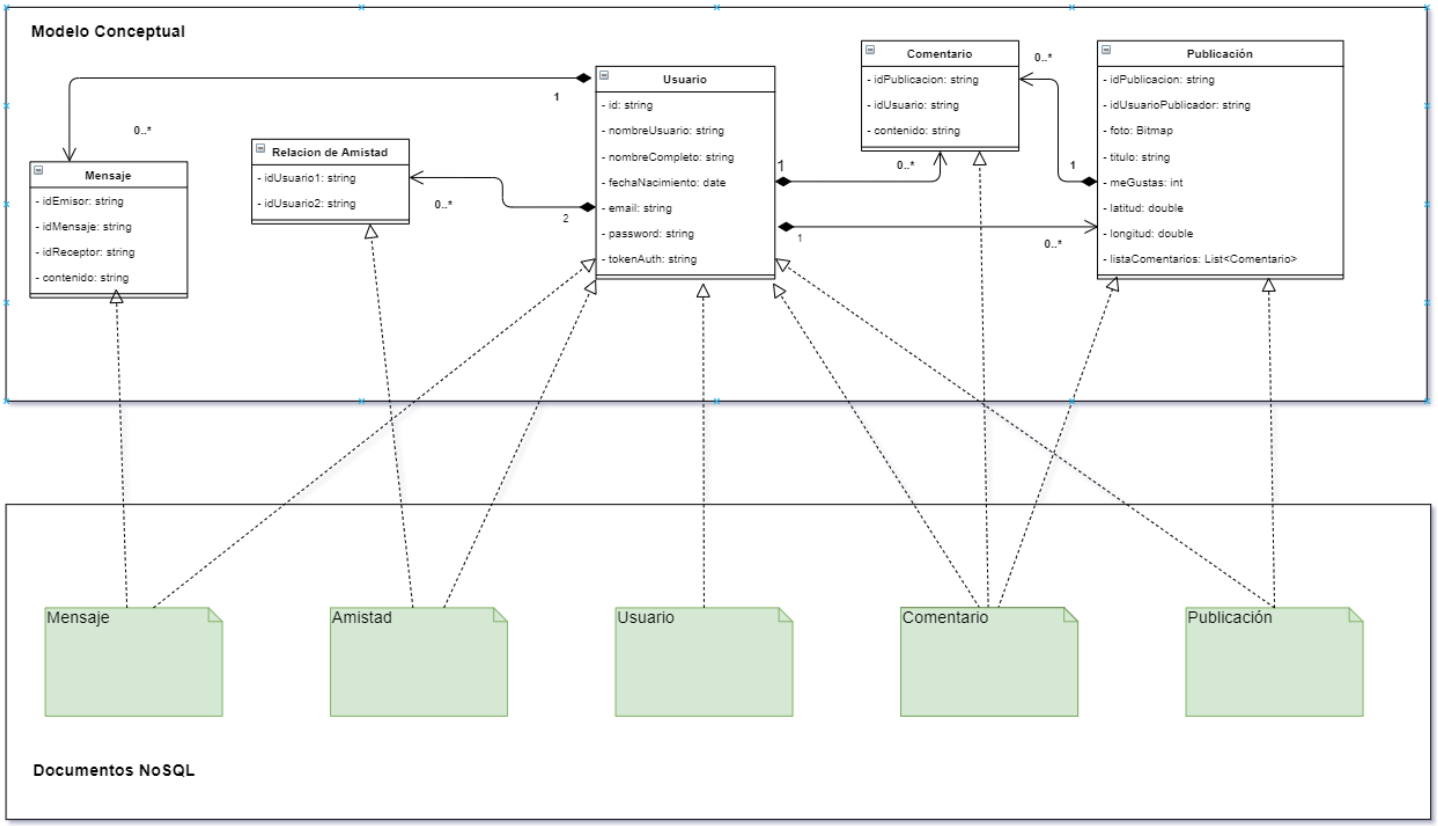
- Requisitos del propio proyecto. Si necesito ver información en tiempo real la mejor alternativa es Firebase y su Cloud Storage, por lo que aquí ya estoy condicionado a elegir NoSQL.
- No vamos a realizar operaciones sensibles y críticas en la aplicación por lo que la integridad no es un factor de principal importancia (priorizamos velocidad). Además, como estamos en etapas tempranas del desarrollo los tipos de datos pueden cambiar en algún momento y NoSQL nos dará facilidades para ello.
- No requerimos de operaciones atómicas que cambien muchas entidades, si cambia una publicación solo cambia la publicación, si el usuario edita su nombre, solo cambia el usuario, etc.
- Si la aplicación tiene buena acogida y necesitamos manejar más volumen de datos podremos hacerlo de forma austera gracias a la escalabilidad horizontal que nos ofrece NoSQL, comprando servidores baratos y aumentando la base de datos
- Con el objetivo de alcanzar la máxima velocidad posible de lectura y escritura nos serviremos de los mecanismos que ofrece NoSQL (recuerdo que debemos mostrar datos en tiempo real).

1.2.1. Diagrama de persistencia de la información

Este diagrama hay que entenderlo en 2 dimensiones. Por un lado, tenemos la dimensión de negocio que describe a los propios objetos/clases que se manejan a nivel de aplicación. Por otro lado, tenemos el grupo de documentos NoSQL físicos que se van a crear, indicando los datos de qué clases van a almacenar[8].

Las flechas muestran las clases de las que los documentos guardan datos, cuando hacen referencia a dos clases en este caso significa que el documento tendrá los datos de una y la referencia a otra (no quiero que por ejemplo cada vez que me traiga un usuario de la base de datos, forzadamente me tenga que traer sus publicaciones, por ejemplo, si no que en la publicación guardaré una referencia al usuario, por lo que tendré que realizar dos consultas a la base de datos, pero será más rápido).

Diagrama:



Como se puede observar, se reutiliza el propio diagrama de clases con los datos de cada clase para marcar qué es lo que va a contener cada documento.

[8] https://eaminds.com/2018/08/03/modelando-nosql-data-bases/

Relaciones

De izquierda a derecha:

- Mensaje: Contendrá los elementos de la clase mensaje y el idEmisor y el idReceptor serán referencias a datos de la clase Usuario, que tendremos en el documento de usuarios.
- Amistad: Contendrá dos referencias a datos de la clase Usuario que estarán en el documento de usuarios.
- Usuario: Contendrá los datos del usuario sin más referencias.
- Comentario: Contendrá los datos propios del comentario además de una referencia a datos de la clase Usuario (usuario que realiza el comentario), que estarán en el documento de usuarios y una referencia a los datos de la clase Publicación (publicación comentada), que estarán en el documento de publicaciones.
- Publicación: Contendrá los datos propios de la publicación y una referencia a la clase Usuario (usuario que sube la publicación) cuyos datos estarán en el documento de usuarios.