****

Contenido

[1. Introducción 2](#_Toc167994141)

[2. Tecnologías utilizadas 3](#_Toc167994142)

[3. UML 4](#_Toc167994143)

[Front-end 4](#_Toc167994144)

[Back-end 5](#_Toc167994145)

[4. Diagramas E/R 6](#_Toc167994146)

[5. Casos de uso, requisitos y funcionalidades 8](#_Toc167994147)

[6. Manual se usuario 15](#_Toc167994148)

[7. Comentarios 15](#_Toc167994149)

# Introducción

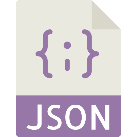


En la época actual, la conectividad y la facilidad de acceso a los servicios son aspectos fundamentales para el desarrollo y el éxito de cualquier aplicación. Con esto en mente, y siguiendo las especificaciones del enunciado del proyecto, la aplicación Kinétia se crea con el objetivo de facilitar la manera en que ofertantes y consumidores de actividades se ponen en contacto, proporcionando una plataforma accesible que facilita la interacción y el intercambio de servicios. Inspirada en modelos ya exitosos como Airbnb y Booksy, la aplicación pretende ser una solución para la gestión y promoción de actividades de cualquier tipo, desde eventos y talleres, hasta experiencias de ocio y entretenimiento.

Kinétia pretende ofrecer una experiencia intuitiva tanto para los ofertantes, quienes pueden listar y gestionar sus actividades de manera sencilla, como para los consumidores, quienes pueden descubrir y reservar actividades así como publicar anuncios de lo que buscan. Todo ello de la manera más accesible desde su dispositivo móvil.

A continuación, en esta documentación, se detallan los aspectos técnicos y funcionales de la aplicación.

# Tecnologías utilizadas



API REST







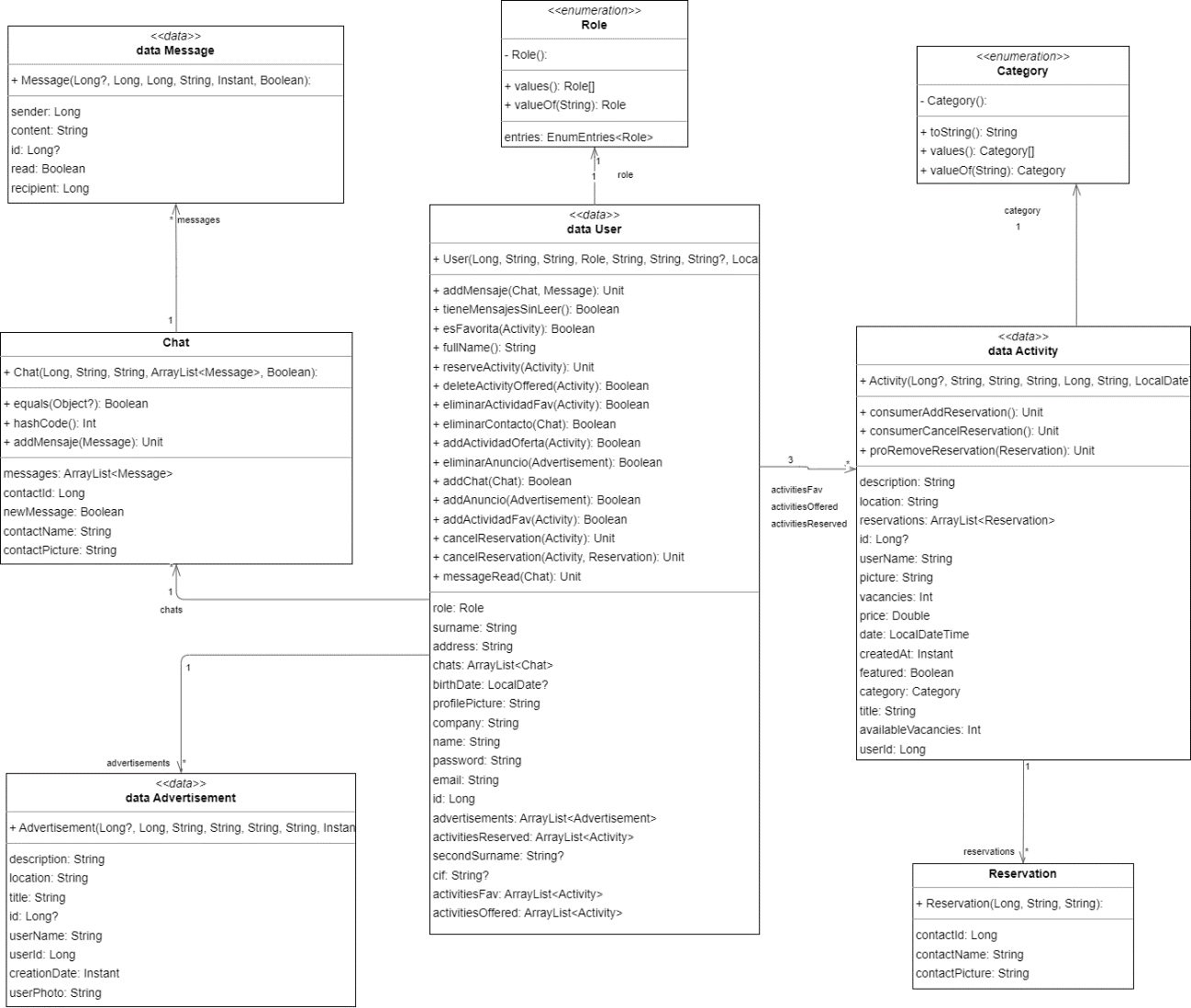
Como se ha mencionado anteriormente, para el proyecto integrado se ha optado por la creación de una aplicación móvil, más concretamente para el sistema operativo Android, siguiendo la arquitectura cliente-servidor.

El front-end ha sido programado en **Kotlin** mediante la librería **Jetpack Compose** y **Retrofit** para la comunicación con el servidor mediante **API Rest**, utilizando **JSON** como formato para la transferencia de datos.

Por otro lado, el back-end se ha desarrollado en **Java**, haciendo uso de la librería **Spring Boot**, con el módulo **JPA** para el mapeo y acceso a la base de datos relacional, la cual se gestiona con **MySQL**.

# UML

## Front-end



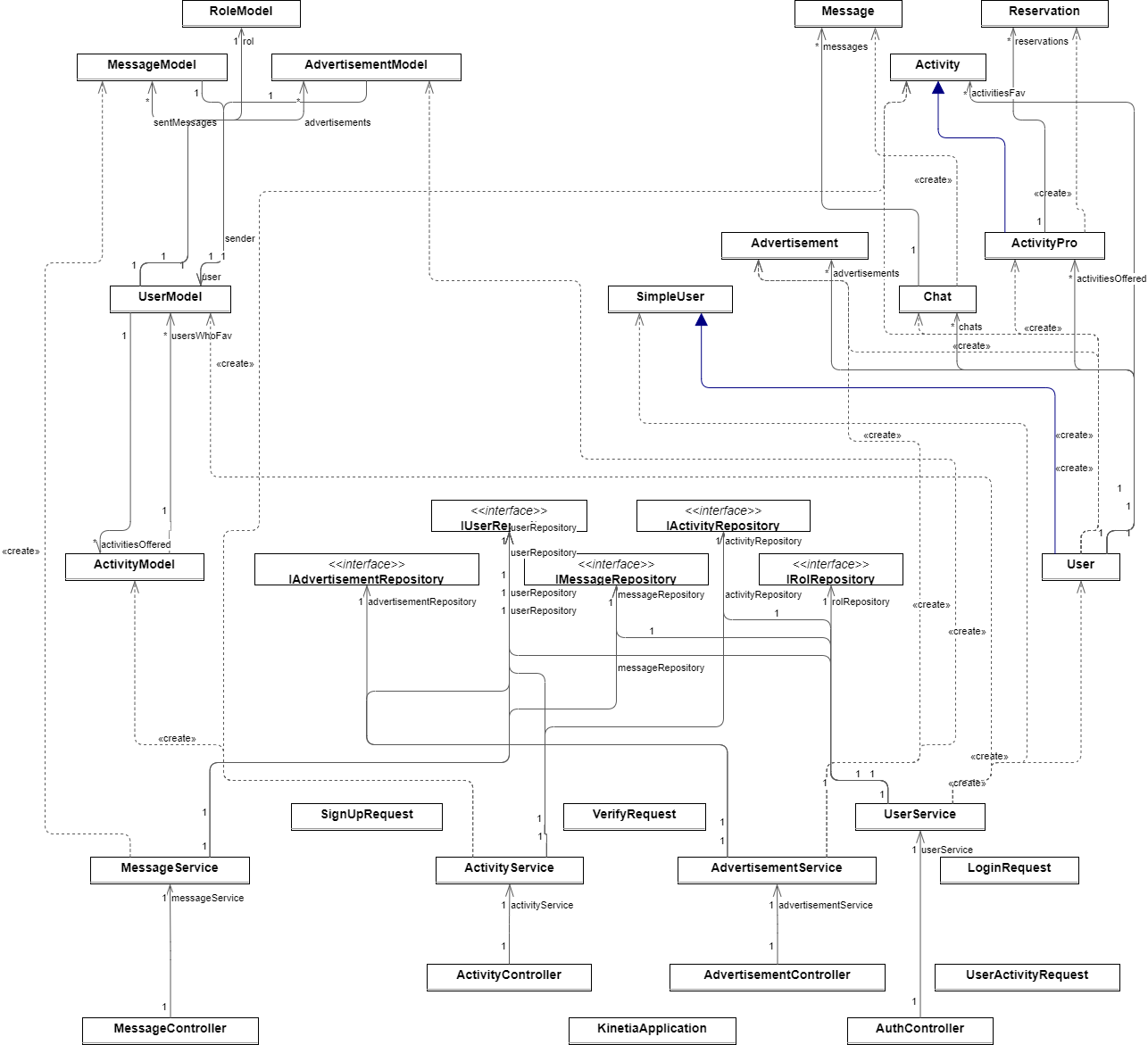
En el diagrama se aprecian los modelos utilizados en la aplicación, los cuales son semejantes a los utilizados en el servidor.

Todo parte del modelaje del usuario en la clase **User**, el cual, dependiendo de su rol (clase enum **Role**), tiene a su disposición distintas funcionalidades y atributos. El ofertante dispone de todas las funcionalidades, a diferencia del consumidor que no puede publicar ni gestionar actividades. De esta manera el usuario tiene varios listados de actividades, según sean ofertadas, favoritas o reservadas.

Se crea una clase **Chat**, la cual contiene una lista de instancias de **Message**, para manejar la comunicación entre los usuarios. Del mismo modo, el usuario gestiona las actividades a través de la clase **Activity**, la cual dispone de un atributo tipo enum **Category** para categorizar el tipo de actividad, así como los datos necesarios de la misma, además de una lista de instancias de **Reservation** que contiene los datos de los usuarios que han reservado la actividad.

Por último, la clase **Advertisement** contiene y gestiona los datos de los anuncios publicados por los usuarios. Los mismos se instancian en una lista en la clase User.

## Back-end



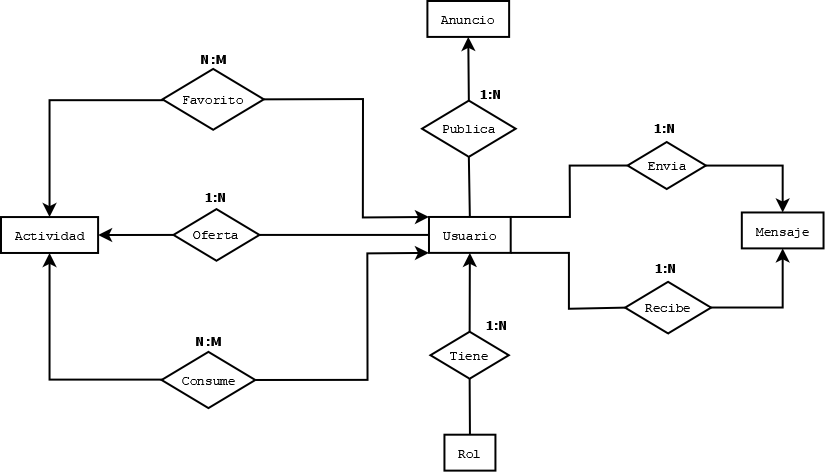
*\*Arquitectura del servidor API Rest*

El servidor se distribuye en varias capas jeráquicas, cada una con una funcionalidad específica. De este modo tenemos los siguientes paquetes:

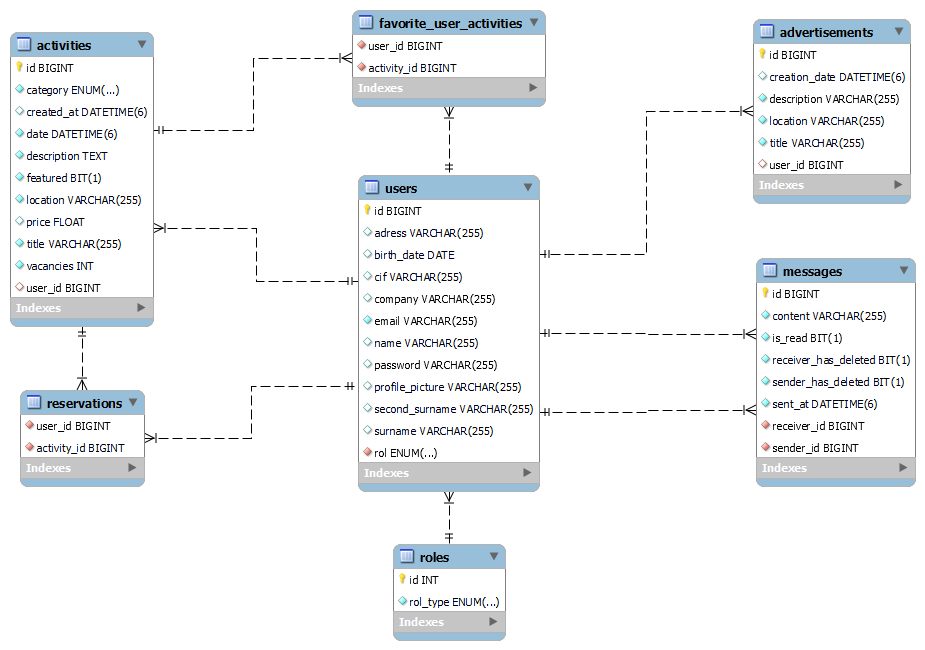
* **api**: Contiene los distintos controladores que manejan las peticiones y respuestas HTTP del servidor. En el subpaquete **request** se encuentran clases DTOs para los datos de las peticiones.
* **domain**: Aquí se encuentran las DTOs correspondientes a las entidades de la base de datos, así como otras necesarias para representar los datos que se envían al front-end.
* **models**: Ubicación de las entidades mapeadas de la base de datos con sus relaciones correspondientes.
* **repositories**: Interfaces que interactúan con la base de datos. Contienen las consultas predefinidas en Spring Data JPA así como otras creadas manualmente.
* **services**: Capa intermedia entre los controladores y repositorios con la lógica de negocio necesaria para manejar las peticiones.

# Diagramas E/R

Entidad-relación:



Entidad-relación extendido:



La base de datos ha sido diseñada con la pretensión de que sea clara y concisa, y sin redundancia de datos, siguiendo los criterios de las formas normales. A continuación se detallan los aspectos más importantes de la misma:

**1. Entidades:**

* **Actividades (activities):** Representa las diferentes actividades disponibles en la aplicación.
* **Anuncios (advertisements):** Representa los anuncios publicados por los usuarios.
* **Usuarios (users):** Representa a las personas registradas, tanto ofertantes como consumidores.
* **Mensajes (messages):** Representa las comunicaciones entre usuarios.
* **Roles (roles):** Representa los roles que pueden tener los usuarios en la aplicación, proveedor o consumidor.

**2. Atributos:**

Cada entidad tiene atributos que describen las características o propiedades de la misma. Por ejemplo, para la entidad activities, algunos de sus atributos son id, category, date, description, location o price.

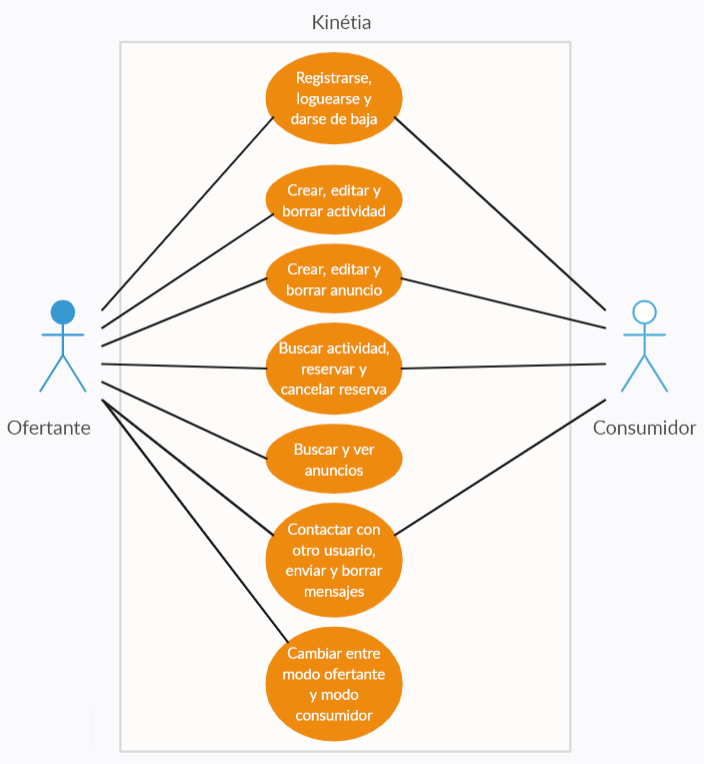
**3. Relaciones:**

Las entidades se relacionan entre sí conectadas entre sí mediante claves foráneas:

* **activities.user\_id** es la relación 1:N entre **activities** y **users**, que indica que cada actividad tiene asociado un usuario (el ofertante de la actividad).
* **advertisements.user\_id** es otra relación 1:N, esta vez entre **advertisements** y **users**, donde cada anuncio está asociado con un usuario (el creador del mismo).
* La tabla intermedia **favorite\_user\_activities** surgede la relación N:M entre usuarios y actividades marcadas como favoritas. Aquí encontramos dos foráneas: **user\_id** de la relación de la tabla con **users**, y **activity\_id** que define la relación con **activities,** de esta manera cada par valor relaciona un usuario con una actividad que ha marcado como favorita.
* **messages.sender\_id** y **messages.receiver\_id** son relaciones 1:N entre **messages** y **users**, indicando que cada mensaje tiene un remitente y un destinatario, ambos usuarios.
* De manera semejante a favorite\_user\_ativities, la tabla intermedia **reservations** relaciona usuarios y actividades mediante las foráneas **user\_id,** que apunta a **users**, y **activity\_id,** que hace lo mismo con **activities.**  Así cada registro relaciona el usuario con la actividad que ha reservado en una relación N:M entre usuarios y actividades.

# Casos de uso, requisitos y funcionalidades

A continuación se muestra un esquema con los casos de uso de la Kinétia, y posteriormente se detallan mismos así como sus requisitos.



































# Manual se usuario

# Comentarios