

Universidade de Vigo

ESCOLA SUPERIOR DE ENXEÑARÍA INFORMÁTICA

Memoria do Traballo de Fin de Grao que presenta

D. Gabriel Rodríguez Vidal

para a obtención do Título de Graduado en Enxeñaría Informática

NightTime: Desarrollo de una red social basada en la actividad nocturna

Xullo, 2021



Traballo de Fin de Grao Nº: EI 20/21-47

Titor/a: García Pérez-Schofield, Baltasar

Área de coñecemento: Linguaxes e Sistemas Informáticos

Departamento: Informática

Contenido

1.	Introducción.....	1
1.1	Uso y aplicación de las redes sociales.....	1
1.2	Android Jetpack	1
2.	Objetivos.....	3
3.	Resumen de la solución propuesta.....	4
3.1	Servidor Web	4
3.2	Aplicación móvil	4
3.3	Metodología utilizada	5
4.	Planificación y seguimiento	6
4.1	Planificación estimada	6
4.2	Tiempo invertido real.....	7
5.	Arquitectura.....	10
5.1	Arquitectura del servidor	10
5.2	Arquitectura de la aplicación móvil.....	12
6.	Tecnologías y Herramientas utilizadas.....	14
6.1	Tecnologías.....	14
6.2	Herramientas	17
7.	Especificación y análisis de requisitos	19
7.1	Actores.....	19
7.2	Requisitos funcionales	20
7.3	Requisitos no funcionales	21
7.4	Diagramas de caso de uso.....	21
7.5	Descripción de caso de uso	22
8.	Diseño del software	26
8.1	Diseño del servidor.....	26
8.1.1	Diseño estático.....	28
8.1.2	Diseño dinámico.....	31
8.2	Diseño de la aplicación.....	32
8.2.1	Diseño estático.....	32
8.2.2	Diseño dinámico.....	34
9.	Gestión de datos e información.....	36
10.	Pruebas llevadas a cabo.....	38
11.	Manual de usuario	40
11.1	Manual del servidor	40
11.1.1	Requerimientos	40
11.1.2	Manual de instalación	40

11.1.3	Manual de uso	41
11.2	Manual de la aplicación.....	41
11.2.1	Requisitos de la instalación	41
11.2.2	Manual de instalación	41
11.2.3	Manual de uso	42
12.	Principales aportaciones.....	46
13.	Conclusiones	47
13.1	Conclusiones técnicas.....	47
13.2	Conclusiones personales.....	47
14.	Vías de trabajo futuras.....	48
15.	Bibliografía	49
16.	Anexo.....	52
16.1	Extensión de la planificación del proyecto	52
16.2	Extensión de los diagramas de clase del servidor	53
16.3	Extensión de los diagramas de secuencia de sistema del servidor	65
16.4	Extensión de la descripción de la gestión de la información.	66

Índice de ilustraciones

Ilustración 1: Diagrama de Gantt estimado para el servidor API	6
Ilustración 2 : Diagrama de Gantt estimado para la aplicación móvil.....	7
Ilustración 3: Tiempo real de desarrollo del servidor	8
Ilustración 4: Tiempo real de desarrollo de la app.....	8
Ilustración 5: Diagrama de Gantt real para el servidor.	9
Ilustración 6:Diagrama de Gantt real para la aplicación móvil.	9
Ilustración 7: Arquitectura general del proyecto.....	10
Ilustración 8: Arquitectura del servidor	11
Ilustración 9: Arquitectura de la aplicación móvil.....	12
Ilustración 10: Arquitectura de la aplicación.....	13
Ilustración 11: Implementación de Pusher.....	17
Ilustración 12: Diagrama de caso de uso del usuario de la app	21
Ilustración 13: Diagrama de caso de uso del servidor	22
Ilustración 14: Diagrama de clase del controlador de bar	28
Ilustración 15: Diagrama de clase del servicio de bares.	29
Ilustración 16: Diagrama de clases de los modelos de datos	30
Ilustración 17: Diagrama de secuencia de sistema de mandar una petición de amistad	31
Ilustración 18: Diagrama de secuencia de sistema de aceptar una solicitud de amistad.....	31
Ilustración 19: Diagrama de clase de la aplicación móvil.....	33
Ilustración 20: Diagrama de secuencia de sistema de enviar una petición de amistad	34
Ilustración 21: Diagrama de secuencia de sistema de seleccionar un día para salir.....	35
Ilustración 22: Credenciales actuales para la base de datos.	40
Ilustración 23: Configuración de la base de datos	41
Ilustración 24: Pagina de login y registro de la app	42
Ilustración 25: Pagina de calendario	43
Ilustración 26: Pagina de los bares	44
Ilustración 27: Página de amigos y chat.....	45
Ilustración 28: Página de perfil y edición de perfil.....	45
Ilustración 29: Diagrama de Gantt del desarrollo del proyecto.....	52

Ilustración 30: Diagrama de clase del controlador de eventos	53
Ilustración 31: Diagrama de clase del controlador de mensajes	54
Ilustración 32: Diagrama de clase del controlador de usuarios.....	55
Ilustración 33: Diagrama de clase del servicio de amistades.....	57
Ilustración 34: Diagrama de clase del servicio de usuarios.....	58
Ilustración 35: Diagrama de clase del servicio de ciudades	59
Ilustración 36: Diagrama de clase del servicio de fechas marcadas.....	60
Ilustración 37: Diagrama de clase del servicio de eventos.....	61
Ilustración 38: Diagrama de clase del servicio de fotos de los bares.....	62
Ilustración 39: Diagrama de clase del servicio de almacenamiento de imágenes.....	63
Ilustración 40: Diagrama de clase del servicio de mensajes	64
Ilustración 41: Diagrama de secuencia de sistema de crear nuevo usuario	65
Ilustración 42: Diagrama de secuencia de sistema de seleccionar un día para salir	65
Ilustración 43: Relación y descripción de las tablas de la base de datos.....	69

NightTime: Desarrollo de una red social basada en la actividad nocturna

1. Introducción

Los sistemas móviles cada vez avanzan más y aparecen aplicaciones con objetivos más específicos. Siempre ha sido un problema estimar cuánta gente saldrá por la noche, sobre todo en fechas que no son festivas como puede ser el examen final de una titulación.

Actualmente es muy sencillo comunicarse con aquella gente que conocemos ya que hay una gran cantidad de redes sociales y servicios de mensajería instantánea que nos permiten hacer esto. Sin embargo, es muy difícil tener información de todos los usuarios ya que las redes sociales están centralizadas en grupos de amigos, por lo que no es difícil conocer las intenciones de la gente que esté más allá de este.

De la misma manera los nuevos locales nocturnos también tienen dificultades para ser conocidos y poder avisar al público de los eventos que va a albergar. Actualmente la mayor parte de establecimientos tiene una página web con la que comunicarse con los usuarios, sin embargo, la manera más común de transmitir estos eventos suele ser el boca a boca. Y no cabe duda de que este sistema puede verse mejorado en gran medida con el uso de un sistema centralizado por ciudades.

En este proyecto se desarrolla la implementación de una aplicación llamada *NightTime*. Esta permite de manera rápida obtener información sobre el día seleccionado, como el número de amigos y número total de usuarios que tienen intención de salir esa noche y los eventos disponibles en los bares de la ciudad.

1.1 Uso y aplicación de las redes sociales

Una red social [1] [2] es una interconexión de personas mediante el uso de ordenadores. De la misma manera que una red computacional es un conjunto de ordenadores unidos mediante cables, una red social es un conjunto de personas (u organizaciones u otras entidades sociales) conectadas por relaciones sociales, como amistad, compañeros de trabajo o intercambio de información.

Actualmente las redes sociales son utilizadas por la gran mayoría de la población y en muchos casos son consideradas esenciales como medio de comunicación y transmisión de las novedades de los grupos de amigos.

1.2 Android Jetpack

Esta tecnología hace uso de la librería de soporte “AndroidX” [3], que es la siguiente versión de la librería tradicional, ya obsoleta “Android Support Library”. Fue presentada en septiembre de 2018 y actualmente está en la fase alfa de la versión preliminar. Esto quiere decir que aún está en fase de desarrollo y está bajo actualizaciones constantes.

Jetpack [4] es un conjunto de librerías tanto de arquitectura como de interfaz de usuario, ya que permite la creación y mantenimiento de bases de datos locales con la librería *Room* y el desarrollo de interfaces gráficas con *Compose*, que deja de utilizar archivos XML [5] para describir la interfaz mediante código *Kotlin* [6] y necesita menos código genérico que las librerías tradicionales.

NightTime: Desarrollo de una red social basada en la actividad nocturna

Muchas de estas funcionalidades solo están disponibles bajo el lenguaje Kotlin, que es un lenguaje de programación dirigido a objetos que trabaja sobre la máquina virtual Java (JVM). Este lenguaje se caracteriza por la autogeneración de código, reducción de peso en la aplicación final y la compatibilidad con código Java, lo que permite el uso de librerías antiguas.

2. Objetivos

El objetivo de este proyecto es crear una aplicación para sistemas móviles que permite a los usuarios recopilar información y publicar los días que tiene intención de salir. Para lograr esto el usuario tiene que poder visualizar el número de usuarios que ya han confirmado sus intenciones de salir ese mismo día.

Como esta aplicación también tiene un fin de red social, es importante que los usuarios puedan personalizar sus perfiles para poder ser identificados mejor por el resto de usuarios.

Esto permite que entre dos usuarios se cree una relación de amistad, lo que se refleja a la hora de visualizar el número de usuarios con intención de salir, pues también se mostrará el número de usuarios amigos junto con el de usuarios totales.

También es necesario que los usuarios con este tipo de relación puedan comunicarse mediante un sistema de mensajería instantánea integrado en la aplicación cliente. Es decir, un chat en tiempo real entre dos usuarios de la aplicación.

Para llevar esta aplicación a cabo, hace falta un servicio que conecte todos los clientes, esto se logra a través de un servidor restfull, que controle el acceso, edición y obtención de información de la base de datos.

La metodología de creación de aplicaciones está actualmente en crecimiento, por lo que Google e IntelliJ han colaborado en el desarrollo de un conjunto de librerías y buenas prácticas conocido como Android Jetpack. Este conjunto ofrece prácticas y tecnologías en auge y promueve la creación de código más robusto y fácil de entender.

Entre las buenas prácticas destaca el uso de Material Design, una guía de diseño de interfaces de usuario para crear aplicaciones intuitivas a la par que visualmente agradables. Como Jetpack da soporte a Material hay componentes prefabricados que pueden usarse para la acelerar la implementación de las vistas.

3. Resumen de la solución propuesta

El desarrollo de esta aplicación debe utilizar los sistemas propuestos Jetpack ya que cada vez son más comunes y potentes y permiten un control más fluido del ciclo de vida de los componentes. Sin embargo, este nuevo marco solo está disponible para Kotlin, por lo que el desarrollo de la aplicación tendrá que ser en este lenguaje.

Kotlin es un lenguaje multiplataforma, es decir, funcionará como un lenguaje nativo cuando se esté ejecutando en la aplicación móvil o podrá ejecutarse bajo JVM en el servidor. Esto permite que el proyecto se desarrolle completamente en Kotlin.

La solución para este proyecto consta de dos sistemas interconectados, una aplicación móvil que mande peticiones, y un servidor web que responda a estas peticiones mediante objetos JSON.

3.1 Servidor Web

Para dar soporte a la aplicación de una manera eficiente la arquitectura API Rest [7] es la más indicada por la facilidad de implementación, limpieza de código y garantizar la compatibilidad en futuras ampliaciones del proyecto, como puede ser la creación de una página web para facilitar la administración de los bares, o el desarrollo de la aplicación cliente en otras tecnologías que permitan la comunicación HTTP [8].

Con el objetivo de garantizar la autenticidad del usuario se utiliza una seguridad basada en tokens, que son adjudicados al usuario en el momento de realizar el login. Esto permitirá comprobar si el usuario que realiza la consulta tiene permisos para llevarla a cabo.

Este proyecto se desarrolla sobre el framework Spring Boot [9] y así facilitar la creación y mantenimiento. Este servidor accede mediante JPA [10] a una base de datos donde se persiste la información.

3.2 Aplicación móvil

La aplicación para Android consistirá en una app con un panel de botones de navegación inferior que sirve para moverse entre las cuatro vistas diferentes. Siendo estas: un listado de los bares de la ciudad, donde se puede acceder a otra vista para ver los detalles más específicos del bar, un calendario con la información del día seleccionado, una lista de las conversaciones iniciadas y la vista del perfil de usuario.

Ya que la aplicación hace uso de Jetpack, siguiendo las prácticas recomendadas la arquitectura más adecuada es MVVM [11] (Modelo/Vista/Vista-Modelo). Esta arquitectura divide la aplicación en 3 componentes: Modelo (Model), encargado de la persistencia de datos, Vista (View), encargado de la definición de la interfaz de usuario y Modelo-Vista (View-Model), responsable de la información mostrada en la vista. En 5 se habla en más detalle del comportamiento de la arquitectura.

Para las vistas de la aplicación se utiliza Compose [12], que es un conjunto de herramientas para construir interfaces en Kotlin de manera nativa. Siendo la principal diferencia la manera de describir la interfaz, pues tradicionalmente esta se construía a partir de archivos XML y ahora se construye mediante código Kotlin. Esto ayuda a disminuir el peso de la aplicación final. Además, las vistas en Compose se organizan mediante un árbol interno, donde cada nodo es un elemento de la interfaz gráfica. Esto favorece la reutilización de componentes y permite que cualquier nodo se reconstruya sin afectar al resto de elementos de la IU.

3.3 Metodología utilizada

La metodología que se ha elegido para este proyecto ha sido Proceso Unificado, ya que el que sea una metodología dirigida por casos de uso y riesgos [13], iterativa e incremental nos permite centrarnos más en las funcionalidades de la app y como estas se relacionan con la arquitectura central.

Gracias al uso de esta metodología podemos dedicar más tiempo al diseño del núcleo de la aplicación y como interactuarán las distintas funcionalidades con este. De esta manera podemos prestar atención a que puntos de entrada permitirá el servidor y así tener una idea clara de cómo debe ser la comunicación establecida entre ambos sistemas. Además de esta manera se permite dividir la aplicación en funcionalidades e ir desarrollando estas individualmente. Así se facilita la gestión del proyecto y la medición de su duración.

Por otro lado, para expresar la arquitectura desde un punto de vista técnico se utiliza como lenguaje Unified Modeling Language (UML) [14], ya que es el recomendado por PU para diseñar el sistema y las interconexiones entre las distintas funcionalidades. Además, tener el esquema del sistema simplificará la ampliación de este, minimizará errores de lógica y facilitará el mantenimiento.

4. Planificación y seguimiento

Este proyecto se ha desarrollado mediante PU por lo que se ha llevado a cabo en 4 fases:

- Inicio: análisis de los objetivos y el alcance del proyecto, así como los casos de uso y factores de riesgo.
- Elaboración: se refinan los objetivos del proyecto y se comienza a desarrollar el núcleo de la aplicación con los casos de uso de la fase anterior, es decir, la arquitectura central para los casos de uso más básicos. Finalmente se reajustan las estimaciones.
- Construcción: implementación de la aplicación final con todas los requisitos formales y no formales. Al acabar esta fase el resultado debe ser una aplicación ejecutable con todos los casos de usos integrados.
- Transición: El proyecto puede considerarse como listo para lanzar al mercado y se comienza a trabajar en el diseño de los futuros cambios.

Este proyecto al estar dividido en dos partes se ha comenzado por el desarrollo del servidor. Ya que las tecnologías utilizadas por este son más comunes y existe una documentación más amplia. De esta manera, es posible marcar los puntos de entrada del servidor y hacer que la aplicación cliente se acomode a estos.

4.1 Planificación estimada

Los tiempos estimados para el desarrollo del proyecto son los siguientes:

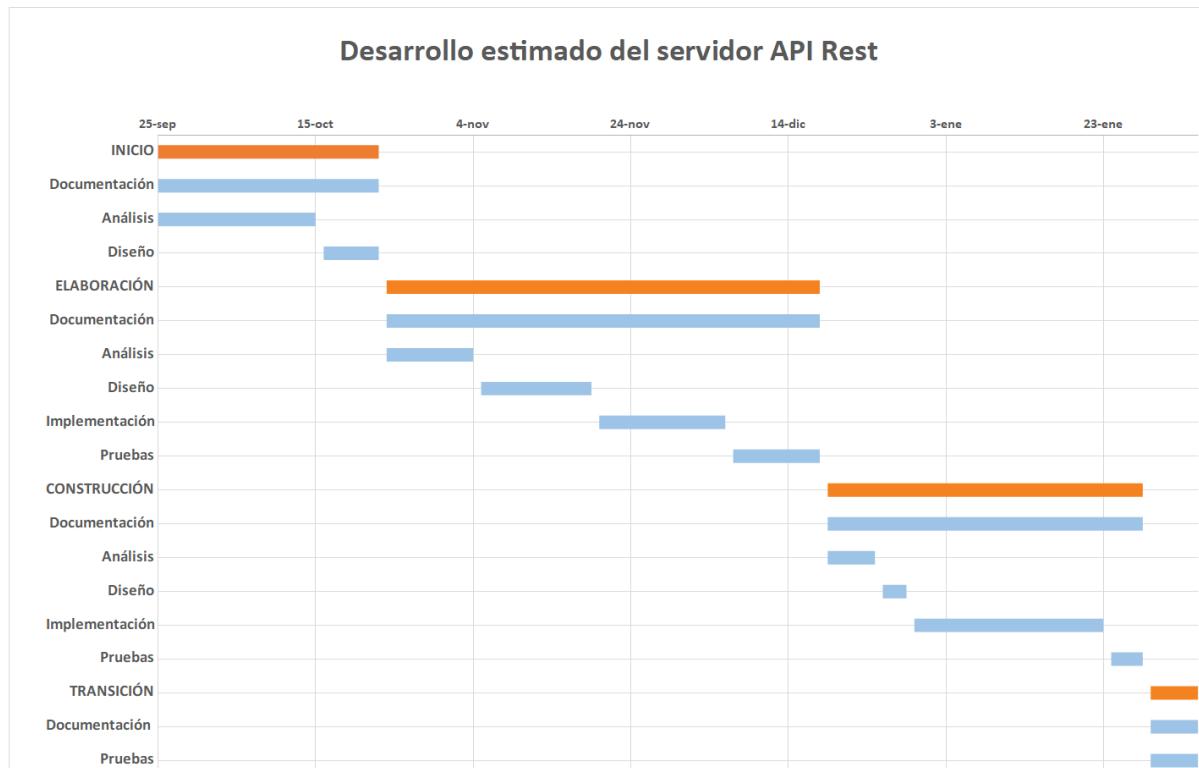


Ilustración 1: Diagrama de Gantt estimado para el servidor API .

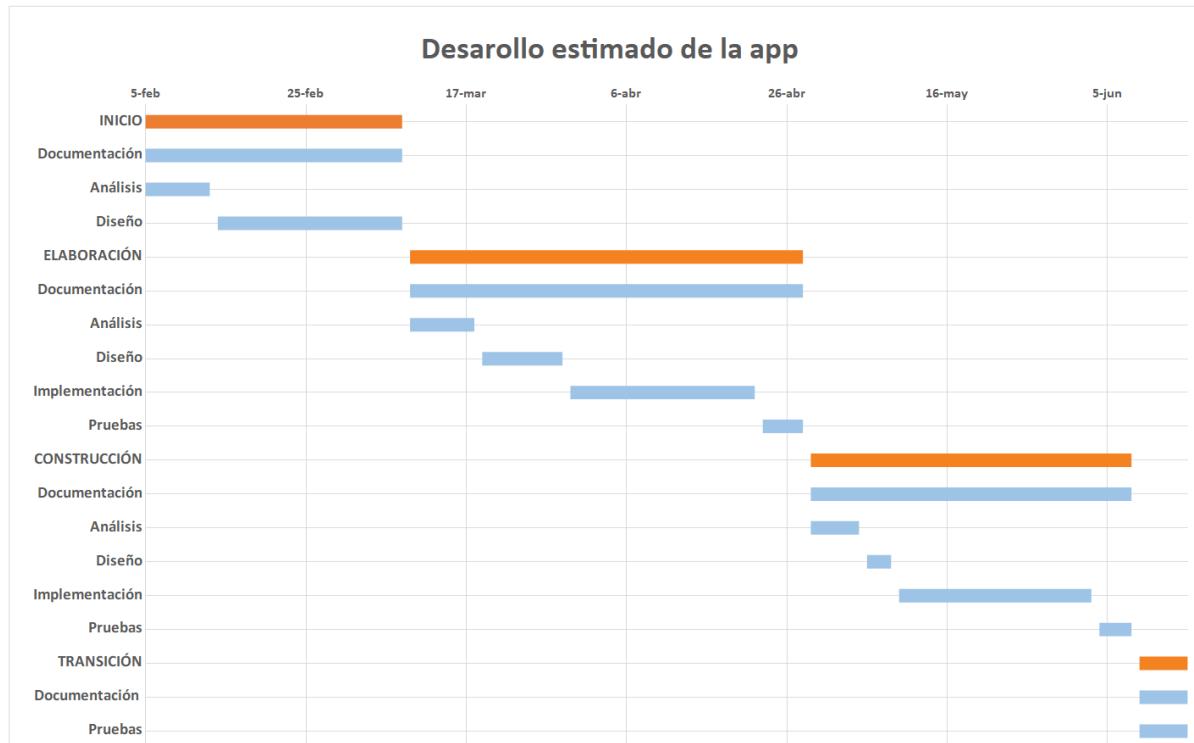


Ilustración 2 : Diagrama de Gantt estimado para la aplicación móvil.

4.2 Tiempo invertido real

Pese que este proyecto ha sido iniciado con gran antelación, el desarrollo ha incluido muchas tecnologías, herramientas y arquitecturas desconocidas previamente, que han impactado en los tiempos planificados. Ya que la elaboración un servidor API Rest es una práctica bastante común y se pudo agilizar el proceso gracias a la cantidad de información disponible online. Esto permitió finalizar el servidor antes de la fecha estimada y tras el mes de diciembre, en el que no se avanzó en el proyecto por motivos personales, se pudo comenzar con la app a mediados de enero.

Para el desarrollo de la aplicación móvil se ha utilizado tecnologías en fase de desarrollo y la información y buenas prácticas son más difícil de encontrar. Además, la creación de vistas e iconos personales necesitó más tiempo del estimado. Esto provocó que el desarrollo de la app se ralentizase y el proyecto se finalizó el 15/06.

El desarrollo de la documentación también alargó el tiempo de finalización del proyecto, ya que los sistemas fueron cambiando a lo largo del tiempo y mantener una documentación rigurosa durante todo el proyecto es muy delicado. Una vez completada la implementación muchos diagramas tuvieron que ser rehechos y algunos aspectos iniciales fueron cambiados. Finalmente, la documentación se finalizó el 22/07.

En el anexo 16.1 se puede encontrar un gráfico por con la planificación de ambos proyecto de manera conjunta.

NightTime: Desarrollo de una red social basada en la actividad nocturna

FASES	FECHA INICIO	DURACIÓN APROXIMAD		FECHA FIN
		DURACIÓN HORAS	A DÍAS	
INICIO	25-sep	28	54	23-oct
Documentación	25-sep	28	54	23-oct
Análisis	25-sep	20	40	15-oct
Diseño	16-oct	7	14	23-oct
ELABORACIÓN	24-oct	26	46	19-nov
Documentación	24-oct	26	46	19-nov
Análisis	24-oct	3	6	27-oct
Diseño	28-oct	8	16	05-nov
Implementación	06-nov	9	18	15-nov
Pruebas	16-nov	3	6	19-nov
CONSTRUCCIÓN	20-nov	12	18	02-dic
Documentación	20-nov	12	18	02-dic
Análisis	20-nov	2	4	22-nov
Diseño	23-nov	1	2	24-nov
Implementación	25-nov	5	10	30-nov
Pruebas	01-dic	1	2	02-dic
TRANSICIÓN	03-dic	2	4	05-dic
Documentación	03-dic	2	4	05-dic
Pruebas	03-dic	2	4	05-dic

Ilustración 3: Tiempo real de desarrollo del servidor.

FASES	FECHA INICIO	DURACIÓN EN DÍAS	DURACIÓN HORAS	FECHA FIN
INICIO	12-ene	32	62	13-feb
Documentación	12-ene	32	62	13-feb
Análisis	12-ene	8	16	20-ene
Diseño	21-ene	23	46	13-feb
ELABORACIÓN	14-feb	41	76	27-mar
Documentación	14-feb	41	76	27-mar
Análisis	14-feb	8	16	22-feb
Diseño	23-feb	10	20	05-mar
Implementación	06-mar	18	36	24-mar
Pruebas	25-mar	2	4	27-mar
CONSTRUCCIÓN	28-mar	76	146	12-jun
Documentación	28-mar	76	146	12-jun
Análisis	28-mar	16	32	13-abr
Diseño	14-abr	13	26	27-abr
Implementación	28-abr	40	80	07-jun
Pruebas	08-jun	4	8	12-jun
TRANSICIÓN	13-jun	2	4	15-jun
Documentación	13-jun	2	4	15-jun
Pruebas	13-jun	2	4	15-jun
Pruebas	13-jun	2	4	15-jun

Ilustración 4: Tiempo real de desarrollo de la app.

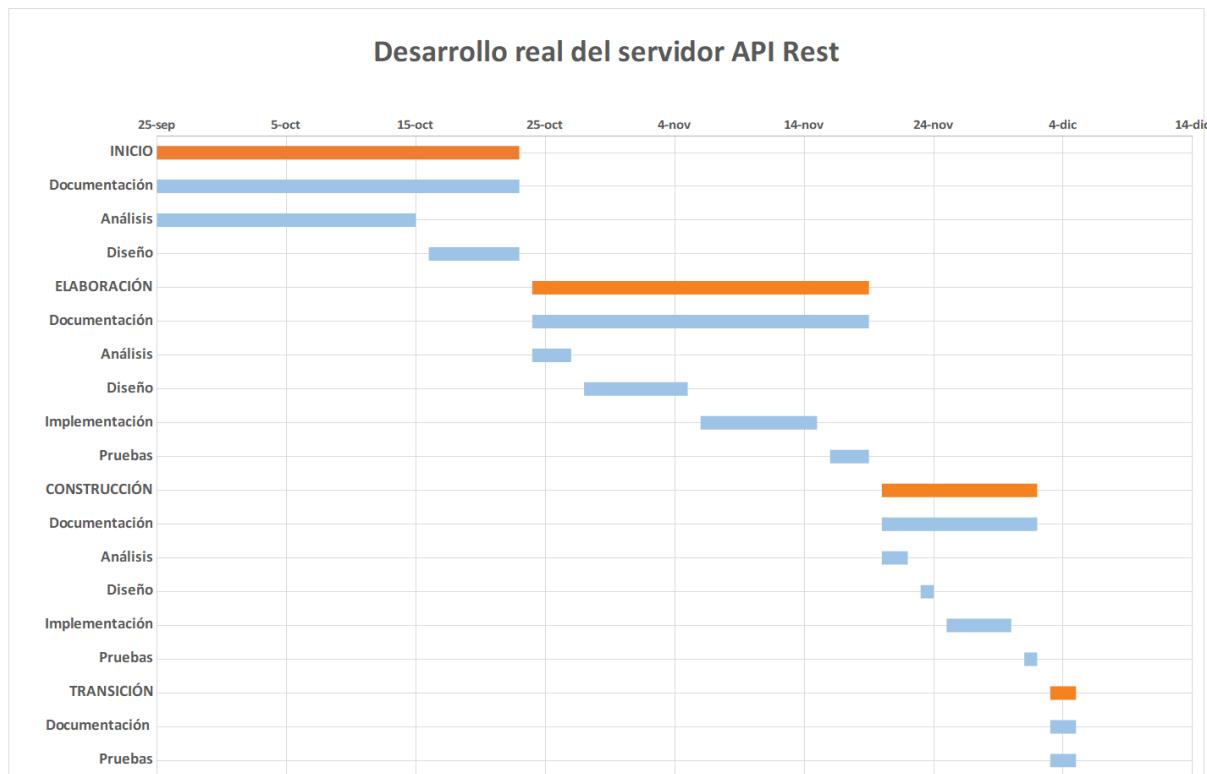


Ilustración 5: Diagrama de Gantt real para el servidor.

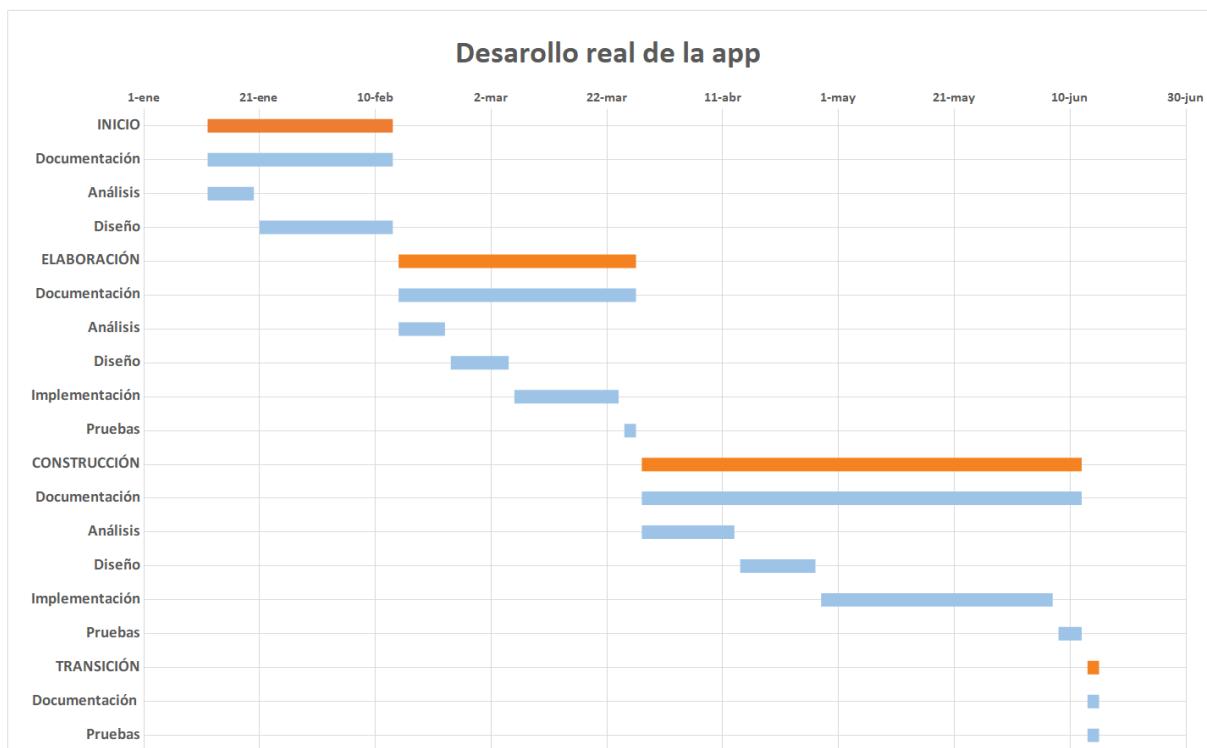


Ilustración 6: Diagrama de Gantt real para la aplicación móvil.

5. Arquitectura

Para dar cobertura total a una aplicación y que esta pueda mostrar datos actualizados, es necesario proporcionarle un punto de acceso donde pueda solicitar y actualizar información. Esto se logra a través de un servidor Rest el cual tras recibir peticiones HTTP, dependiendo de la petición, busca, actualiza o elimina información en la base de datos, y responde a la solicitud con otro mensaje HTTP con información codificada en JSON.

En la Ilustración 3 se muestra la arquitectura general, donde la aplicación móvil únicamente tiene acceso y conocimiento del servidor, y este tras realizar las operaciones solicitadas en la base de datos responde al usuario con los datos solicitados, o bien responde si la edición de los datos fue exitosa.

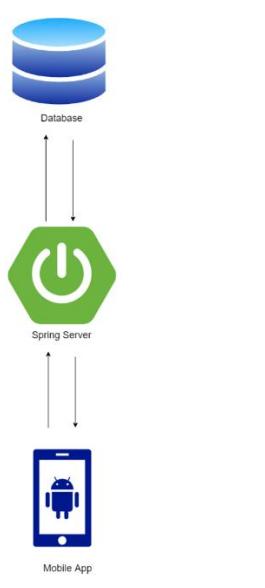


Ilustración 7: Arquitectura general del proyecto

5.1 Arquitectura del servidor

Como se ha mencionado anteriormente el servidor sigue el patrón Rest (transferencia de estado representacional, o representational state transfer), este patrón hace referencia a la comunicación establecida entre cliente y servidor, donde cada petición HTTP del cliente transmite toda la información necesaria para realizar una operación. Siendo el token de seguridad una excepción necesaria para asegurar la autenticidad y autorización del usuario sobre los datos.

Estas peticiones HTTP son un conjunto de una palabra clave u orden sobre una URL, además puede contener o no meta data como el id del usuario y su token privado. De esta manera cualquier sistema que tenga acceso a una red es un cliente potencial de este servicio, por lo que este sistema es perfectamente escalable a otros dispositivos más allá de lo planteado originalmente.

El patrón Rest generalmente se divide en 4 capas, y cada capa se encarga de hacer una operación específica y mandar la información a la siguiente capa. Normalmente la información solicitada es menor que la información almacenada en la base de datos, por ese motivo se utilizan objetos DTO, que transforman un objeto en otro más específico para que se pueda transmitir de manera sencilla y clara. Ya que el framework utilizado en el servidor es Spring, cuando recogemos información de la base de datos podemos hacer uso de lo conocido como Spring projections, que son unas interfaces que permiten encapsular solo la información solicitada. En la otra mano, cuando la información llega en formato JSON y es necesario convertirlo en un DTO, Kotlin ofrece un tipo de clase que ofrece ciertas ventajas frente a una clase normal de Java. Esta nueva clase es conocida como “*data class*” y permiten encapsular la información en clases muy simples, haciendo más sencillo el mantenimiento, y autogeneran código frecuentemente utilizado como constructores, *getters* y *setters*, funciones para equiparar dos clases iguales, etc.

En la ilustración 4 se muestran las 4 capas junto con una breve descripción del funcionamiento de cada una.

La capa superior (capa de presentación), es la primera en recibir la petición, confirma que el usuario está logueado y convierte la petición JSON en un modelo de datos, generalmente un DTO (*data transfer object*).

La segunda capa (capa de lógica) se encarga de las operaciones que hay detrás de cada petición, es decir, en primer lugar, confirma que el usuario tiene autorización para tratar los datos, después confirma que los datos recibidos como DTO coincidan con los datos necesarios para realizar la consulta tanto en tipo como otras especificaciones. En caso contrario lanzan una excepción a la capa superior.

La tercera capa (capa de persistencia), es la única capa que tiene acceso a la base de datos, por eso es la encargada de realizar peticiones de obtención o actualización de los datos. Para facilitar este proceso Spring ofrece una librería llamada “Data JPA” que reduce el esfuerzo necesario para establecer una comunicación segura y eficiente entre la base de datos y el sistema.

La cuarta y última capa es la propia base de datos, encargada de almacenar las tablas con la información.

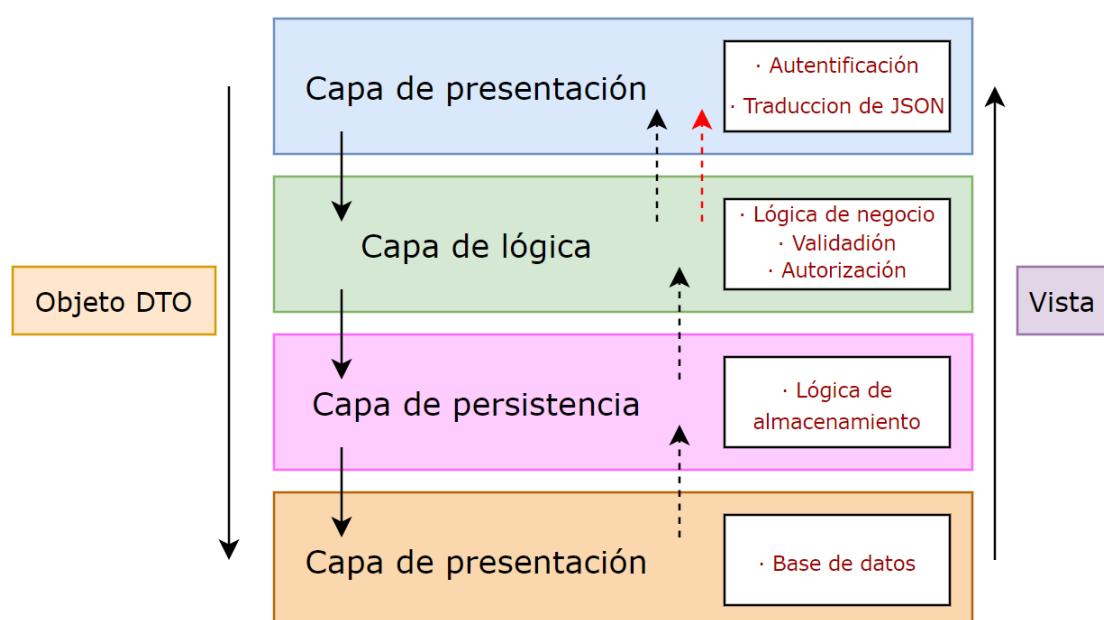


Ilustración 8: Arquitectura del servidor

5.2 Arquitectura de la aplicación móvil

La arquitectura de la aplicación sigue el diseño de la ilustración 5, donde hay una única actividad que dependiendo la vista que se muestra tiene en memoria un ViewModel diferente. Este modelo recoge la información de una fuente de datos diferente dependiendo si los datos que se van a mostrar podrían estar guardados en la memoria local o no. Sin embargo, esta operación no está implementada en el proyecto actual, pero se ha respetado el patrón para poder implementar una base de datos local del tipo *Room* en cualquier momento sin tener que eliminar código. En caso de que los datos a mostrar no pudiesen estar guardados en la memoria local, el modelo accede directamente al servidor, sin utilizar un repositorio como fachada.

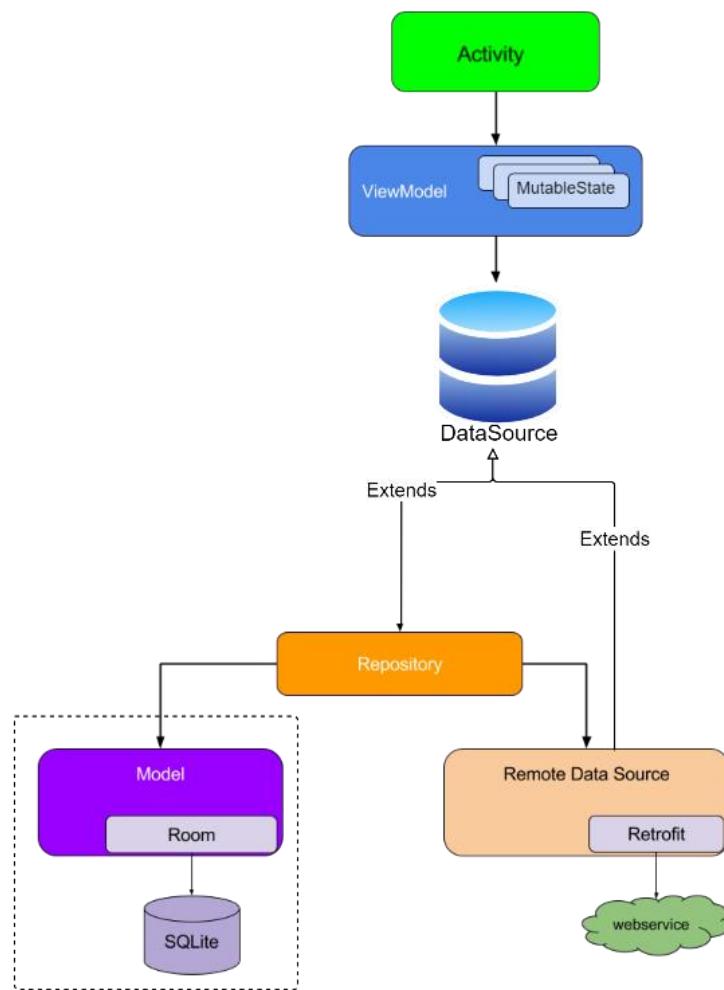


Ilustración 9: Arquitectura de la aplicación móvil

En este proyecto se respeta uno de los principios de Jetpack que recomienda el uso de una actividad única [15] y distintas vistas, generalmente estas vistas serían diferentes fragmentos que se van alternando en la vista principal a medida que el usuario avanza por la aplicación. En este caso como el framework utilizado para crear la interfaz de usuario es Jetpack Compose, no se utilizan fragmentos, sino funciones que describen una parte de la interfaz.

Cada vista tiene una función padre que o bien recibe por parámetro o bien crea mediante una factoría una clase ViewModel, que en el momento de la instanciaión obtendrá los datos propios de la vista. En la ilustración 6 se puede ver como los datos mostrados en la vista son obtenidos desde el modelo (memoria local u obtenidos de la red) y almacenados por el ViewModel, y cualquier evento que suceda será transmitido al ViewModel y en caso de ser necesario, este actualizará los datos en el modelo. Cabe destacar que el ViewModel almacena la información en un tipo de dato llamado MutableState (estado mutable) que proporciona la ventaja de ser observable, por lo que cualquier vista que esté haciendo uso de ese dato será actualizada con la actualización del dato en el modelo.

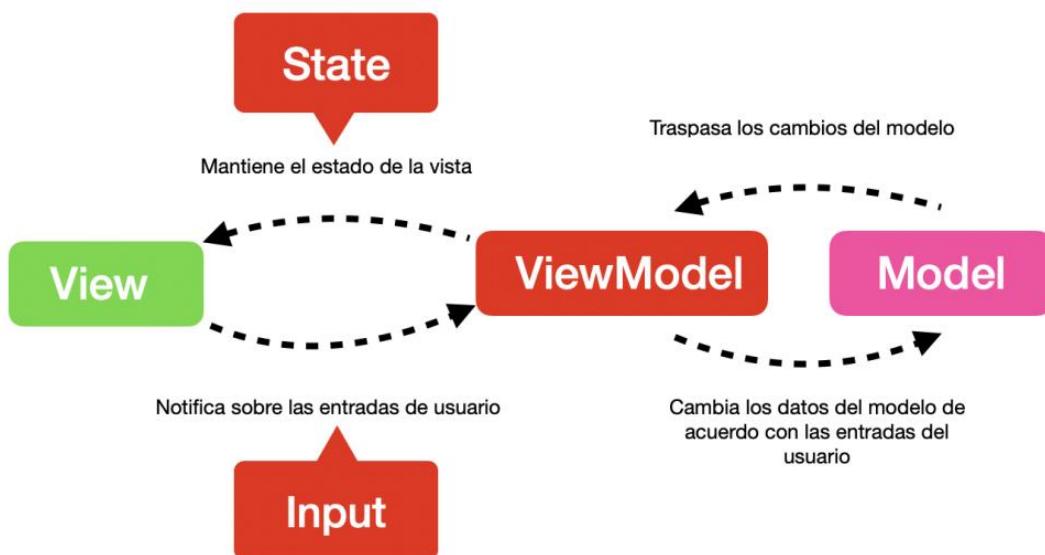


Ilustración 10: Arquitectura de la aplicación

6. Tecnologías y Herramientas utilizadas

Este proyecto incluye herramientas y tecnologías de terceros para facilitar el desarrollo y crear un código más limpio y mantenible. Estas herramientas están indicadas a continuación.

6.1 Tecnologías

- **Jetpack**

Conjunto de librerías y buenas prácticas para el desarrollo de aplicaciones móviles. Muchas de las prácticas utilizadas en este proyecto, como la arquitectura MVVM, pertenecen a este conjunto. Las librerías más importantes que se han utilizado para el desarrollo de este proyecto, son:

- **Compose**

Conjunto de librerías para el desarrollo de la interfaz gráfica por medio de código Kotlin. Implementa diseños prefabricados de Material Design.

- **Navigation**

Conjunto de librerías que permiten la navegación entre las vistas "Composables" junto con el envío de tipos de datos simples.

- **DataStore**

Librería de almacenamiento local persistente para tipos de datos simples. Los datos son guardados como mapas clave-valor, y pueden ser leídos tanto síncrona como asíncronamente.

- **Kotlin**

Principal lenguaje de programación utilizado para la implementación de este proyecto. Kotlin es un lenguaje de programación orientado a objetos (POO) que trabaja sobre la máquina virtual de Java. Gracias a esto es posible utilizar código en ambos lenguajes y hacer uso de cualquier framework que esté disponible para cualquiera de ellos. Por este motivo se puede utilizar tanto en el cliente como del lado del servidor. Este lenguaje agiliza el desarrollo ya que nos permite escribir menos aumentando así la limpieza y claridad del código [16].

- **Gradle**

Herramienta de automatización de compilación para Android [17]. Simplifica la gestión de las dependencias y permite la automatización de procesos antes y después de la compilación.

- **Maven**

Herramienta de gestión de proyectos para Java [18]. A través de un archivo de configuración se puede gestionar la construcción, dependencias y generación de informes.

- **Spring Framework**

Marco de desarrollo para servicios de internet [19]. Funciona como núcleo del servidor al que se le pueden ir añadiendo distintas librerías para facilitar el desarrollo de componentes, las librerías utilizadas son nombradas a continuación. Permite la ejecución de código inicial para poblar la base de datos y mantener logs sobre los eventos sucedidos.

- **Spring Data JPA**

Conjunto de librerías que permite el tratamiento de la base de datos como código [20]. De esta manera es posible autogenerar código SQL mediante Java o Kotlin. También permite insertar código SQL en crudo en caso de ser necesario.

- **Swagger**

Herramienta para la visualización de la estructura de APIs [21]. Es configurable mediante código y permite la simulación de clientes HTTP. Esto es utilizado tanto para testing como para permitir que otros desarrolladores puedan tener información del funcionamiento de la esta y así poder crear otras aplicaciones.

- **H2**

Herramienta utilizada en las fases más tempranas del desarrollo, permite la simulación de una pequeña base de datos, permitiendo el testeo de las funciones más simples sin tener que levantar y configurar una base de datos más pesada.

- **JSON**

Acrónimo de *JavaScript Object Notation* - Notación de Objetos de JavaScript [22]. Es un formato ligero de intercambio de datos por internet. Cualquier sistema puede hacer uso de esa notación para el envío y recepción de la información y resulta más liviano que el intercambio en archivos XML [23]. Como beneficios adicionales resulta sencillo de entender para los humanos simplificando así el proceso de corrección de errores.

- **HTTP**

Hypertext Transfer Protocol (Protocolo de transferencia de hipertexto) es el protocolo de transferencia de datos empleado por REST [8]. Está compuesto por un verbo (GET, PUT, POST, UPDATE ...), una URL, y opcionalmente cabeceras y un cuerpo.

- **REST**

REST (Representational State Transfer) es un estilo arquitectónico para desarrollar servicios web. Permite la comunicación a través de URLs, por lo que cualquier dispositivo que soporte el protocolo HTTP puede hacer uso del servidor.

- **Git**

Sistema de control de versiones ampliamente utilizado [24].

- **Picasso**

Librería para Kotlin que es utilizada para mandar peticiones HTTP y recogerlas como archivo multimedia [25]. Después de cada llamada guarda el archivo en una memoria interna para evitar tener que repetir llamadas a la misma URL. Esta imagen se guardada a tamaño completo y se adapta en el momento de mostrarla por pantalla.

- **Retrofit 2**

Librería de comunicación HTTP para Android [26]. Es un proyecto libre de la empresa Square. Permite el envío de mensajes de manera síncrona o asíncrona, además realiza la conversión de archivos JSON a tipos de datos sencillos. Para convertir los archivos JSON en objetos complejos es necesario el uso de un adaptador. En este proyecto se utiliza Moshi.

- **Moshi**

Adaptador de archivos JSON a objetos complejos [27]. Es un proyecto libre de la empresa Square. El uso de esta herramienta junto Retrofit 2 simplifica la recogida de datos los paquetes HTTP y permite la conversión automática a objetos o listas de objetos.

- **Pusher**

Servicio de terceros que permite la comunicación de datos en tiempo real entre un servidor y un cliente [28]. Se utiliza principalmente para crear chats en tiempo real. Inicialmente cada cliente tiene que registrarse a la escucha de un o más canales en el servicio Pusher. Entonces como se muestra en la ilustración 7, el cliente solo tiene que enviar los mensajes al servidor y este es el encargado de transmitirlos al servicio Pusher. Siendo este último el que tiene una referencia al usuario, por lo que es capaz de establecer una comunicación servidor-cliente.

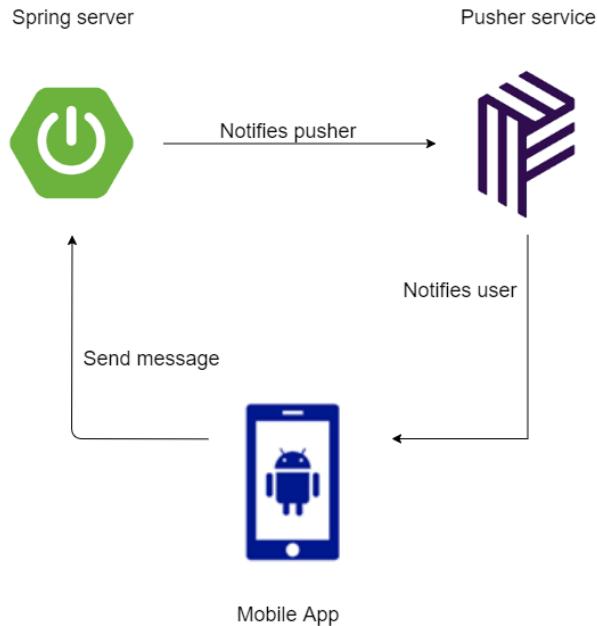


Ilustración 11: Implementación de Pusher

6.2 Herramientas

- **IntelliJ Idea**

Entorno de desarrollo integrado (IDE) propiedad de JetBrains [29]. Es un IDE que permite el testeo de aplicaciones Spring mediante la emulación de un servidor y da soporte para el manejo de los ficheros de estructura de datos (JSON). Además, tiene herramientas para el manejo de control de versiones, ejecución de fases de Maven, ejecución de pruebas, etc. Ha sido elegido por compatibilidad con todas las herramientas utilizadas, junto con preferencias personales.

- **Android Studio**

Entorno de desarrollo integrado (IDE) basado en IntelliJ Idea [30]. Es el IDE más popular el desarrollo de aplicaciones móviles y está especializado para esto. Pues permite la emulación de sistemas Android y la monitorización de los componentes de este (porcentaje de uso de batería, tráfico de red, y CPU).

- **Postman**

Es una herramienta de emulación de un cliente HTTP. Se utilizó para las fases de prueba de la API más tempranas por la característica que permite programar tests automáticos. Después se utilizó Swagger para este fin.

- **Microsoft Word 2019**

Word es un software de procesamiento de textos ampliamente conocido que se ha empleado para la generación de documentación del proyecto. [31]

- **Herramientas de comunicación**

Debido a la situación de pandemia por el COVID-19 no ha sido posible el contacto personal para la supervisión de este trabajo. Por lo que la comunicación alumno-tutor se ha realizado mediante herramientas como Gmail.

- **GitHub**

Web proveedora de un servidor de Git. Permite el control de versiones de código de proyectos de forma libre. De esta manera se asegura la coherencia del código, aunque el proyecto se haya desarrollado desde distintos ordenadores. Actualmente es una de las más utilizadas para compartir proyectos entre equipos.

- **Draw.io y Inkscape**

Herramientas para el diseño de elementos gráficos. Pese a ser herramientas diferentes comparten objetivos. Pues ambas se han utilizado para tanto para el diseño preliminar de las vistas de la aplicación como para el diseño de los gráficos e ilustraciones presentes en la documentación.

- **yFiles**

Conjunto de librerías compatibles con varios entornos de desarrollo (IDEs de JetBrains, Eclipse, NetBeans...), que permite la generación de diversos diagramas [32] que fueron usado como base para la creación de los diagramas encontrados en el punto 8.1 en adelante.

7. Especificación y análisis de requisitos

Siguiendo la metodología mencionada anteriormente, Proceso Unificado, el análisis de requisitos es el primer paso para la elaboración del proyecto, sin embargo, nuevos requisitos pueden ser descubiertos en la fase 2 y 3 (implementación de núcleo de la aplicación, e implementación de los casos de uso). Aquí se recogen todos los requisitos descubiertos durante el desarrollo del proyecto, y la descripción de los actores.

Los requisitos pueden considerarse de dos tipos:

- **Requisitos funcionales:** Expresan un comportamiento que el sistema debe cumplir y la información necesaria para lograrlo. [33]

Su abreviatura es RFxx (siendo xx el número que identifica el requisito)

- **Requisitos no funcionales:** Expresan criterios para juzgar la operación de un sistema.

Su abreviatura es RNFxx (siendo xx el número que identifica el requisito)

7.1 Actores

Se consideran dos actores principales: usuarios de la app y dueños de los establecimientos. Los usuarios de la app se comunican mediante la aplicación móvil, mientras con los dueños de los establecimientos hacen uso de una web de gestión. Sin embargo, por falta de tiempo para el desarrollo total de este proyecto, no se ha podido implementar esto último y la creación de la web es una vía de trabajo futuro. La definición de los actores finales es la siguiente:

- **Usuario de la app:** Cliente final que utilizará la app para su disfrute, con el fin de tener más información a la hora de tomar la decisión de que día salir de fiesta. Además, este usuario podrá mantener comunicación en tiempo real con los demás usuarios, convirtiendo la app cliente en una red social.
- **Dueños de los establecimientos:** Usuarios que utilizan la aplicación con fines comerciales, ya que se publicitan a través esta. Estos usuarios que han registrado sus establecimientos en la aplicación haciéndola visible al resto de usuarios pueden publicitar sus eventos en las fechas seleccionadas.

7.2 Requisitos funcionales

La aplicación móvil debe permitir:

RF1. Gestionar usuarios.

Un usuario podrá crear y modificar un perfil.

RF2. Consultar usuario.

Un usuario puede visualizar el perfil de otro usuario.

RF3. Consultar bares.

Un usuario puede consultar la lista de bares respecto a una ciudad, e información asociada a dicho bar como el contenido horarios, eventos y multimedia.

RF4. Consultar información de los días del mes.

Un usuario podrá consultar la información relativa a un día seleccionado, como los eventos disponibles y la cantidad total de gente y los amigos que han seleccionado ese día.

RF5. Registrar la intención de salir de fiesta.

Un usuario podrá seleccionar los días posteriores a la fecha en la que se encuentre, registrando así su intención de salir de fiesta ese día. De esa manera sus amigos y otros usuarios podrán tenerlo en cuenta.

RF6. Añadir y eliminar amigos.

Un usuario podrá enviar una solicitud de amistad a otro usuario. El otro usuario podrá aceptarla o rechazarla. En caso de ser aceptada, esta puede eliminarse más tarde.

RF7. Establecer una conversación en tiempo real con usuarios de la lista de amigos.

Un usuario podrá hacer uso del chat integrado en la aplicación, notificándose al momento al receptor si este está conectado.

De la misma manera, el servidor API Rest tiene que admitir y responder a todas las llamadas generadas por el cliente Android y además permitir la gestión de la información de los bares, es decir:

RF8. Gestionar bares.

A través de la API se podrá añadir, eliminar y modificar la información asociada a los bares.

RF9. Gestionar eventos.

A través de la API se podrá añadir y eliminar eventos asociados a los bares.

RF9. Añadir fotos como contenido multimedia a la información de los bares.

A través de la API se podrá añadir contenido multimedia a los bares como fotos.

7.3 Requisitos no funcionales

A continuación, se detallan los requisitos no funcionales que deben cumplir al menos en uno de los sistemas.

RNF1. Disponibilidad- La aplicación debe permitir una navegación ágil, devolviendo los resultados en un tiempo razonable.

RNF2. Compatibilidad- La herramienta debe ser compatible con distintas plataformas (Ordenadores de escritorio o dispositivos móviles) y distintos sistemas operativos.

RNF3. Robustez- La herramienta debe ser resiliente pudiendo recuperarse automáticamente de errores no críticos

RNF4. Seguridad- La herramienta debe respetar la confidencialidad de datos, teniendo niveles de acceso a los distintos tipos de datos.

7.4 Diagramas de caso de uso

El diagrama de caso de uso del cliente de la app se muestra a continuación, sin embargo, es necesario recordar que como requisito es necesario realizar la función de login antes de poder interactuar con el sistema.

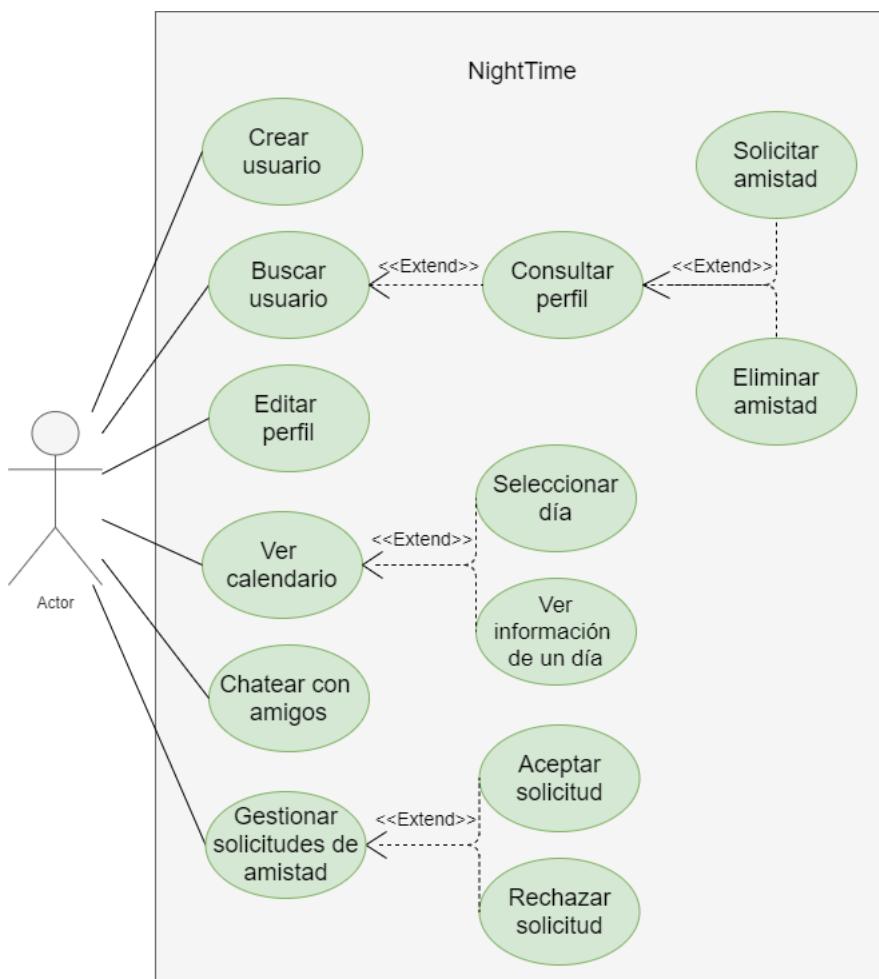


Ilustración 12: Diagrama de caso de uso del usuario de la app

El siguiente diagrama describe los casos de uso del dueño del establecimiento.

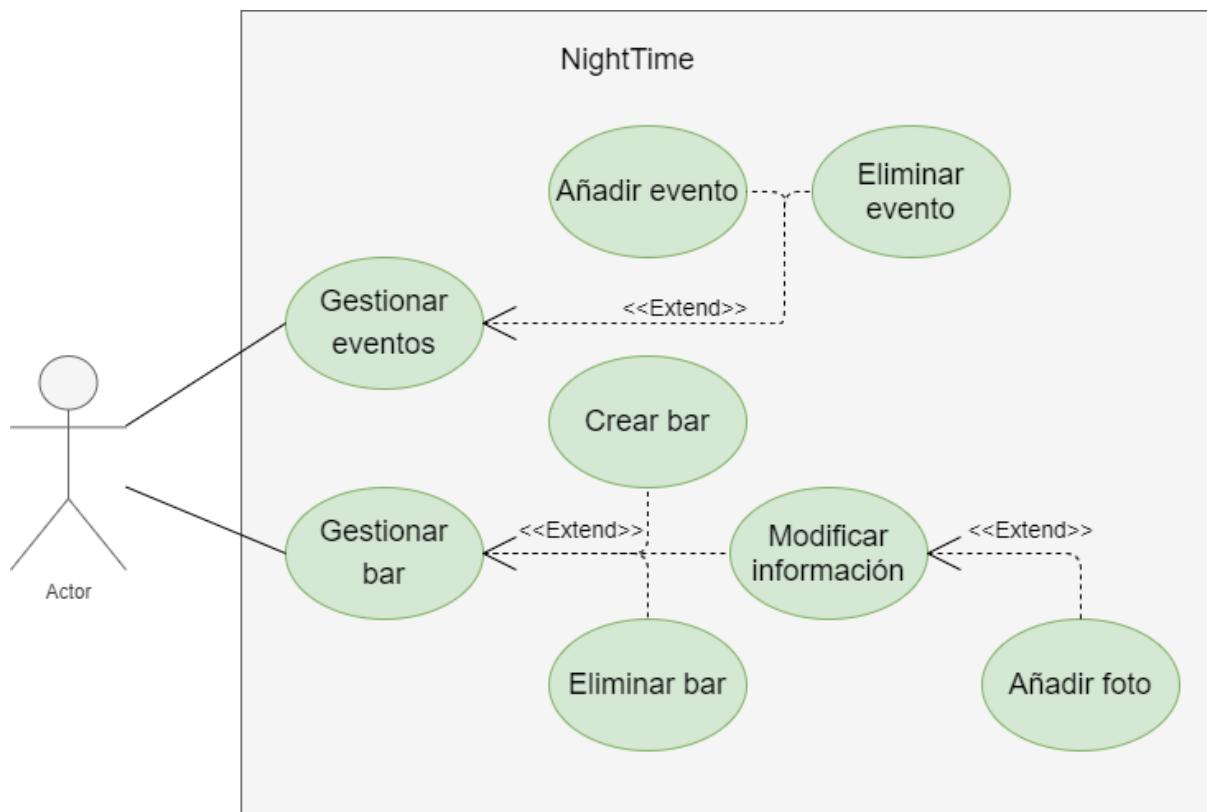


Ilustración 13: Diagrama de caso de uso del servidor

7.5 Descripción de caso de uso

A continuación, se describe el flujo de algunos casos de uso para el cliente.

Nombre	Crear usuario	
Actor	Cliente de la app	
Descripción	Un usuario se registra en la aplicación	
Precondiciones	El usuario no debe estar registrado previamente.	
Postcondiciones	El usuario quedará registrado en el servidor.	
Flujo de eventos		
	Cliente	NightTime
1	Abre la aplicación	
2		Muestra la página inicial de login
3	Pulsa sobre el ícono de añadir usuario	

4		Muestra el formulario de registro
5	Rellena el formulario y pulsa sobre el icono de añadir foto de perfil	
6		Abre la galería y permite la selección de una foto
7	Selecciona su foto favorita y pulsa "Confirmar"	
8		Notifica al usuario el éxito en la creación del perfil, realiza el login automáticamente y accede a la siguiente vista de la aplicación.

Nombre	Solicitar amistad.
Actor	Cliente de la app.
Descripción	Un usuario manda una petición de amistad a otro usuario
Precondiciones	No puede existir una relación de amistad previa entre ambos usuarios.
Postcondiciones	Existe una entrada en la tabla de amistad aun no aceptada.

Flujo de eventos

	Cliente	NightTime
1	Selecciona la página de chats.	
2		Muestra la lista de chats.
3	Pulsa en "Buscar usuario".	
4	Introduce uno o más caracteres iniciales del nombre de usuario o el nombre real.	
5		Muestra una lista con los usuarios que tengan esos caracteres en el nombre de usuario o el nombre real
6	Desliza hacia abajo para mostrar más resultados y pulsa sobre el usuario deseado.	
7		Muestra la vista del perfil del usuario seleccionado
8	Pulsa sobre "Enviar solicitud de amistad"	
9		Cambia el color del botón pulsado a gris e informa de que la solicitud ya ha sido enviada.

Nombre	Aceptar amistad.	
Actor	Cliente de la app.	
Descripción	Un usuario recibe una solicitud de amistad y la acepta.	
Precondiciones	Un usuario debe mandar previamente una solicitud de amistad.	
Postcondiciones	La notificación de solicitud desaparece y ambos usuarios son amigos.	
Flujo de eventos		
Cliente		NightTime
1	Selecciona la página de chats.	
2		Muestra la lista de chats, y sobre el ícono de peticiones de amistad, se muestra el número de peticiones aun no respondidas.
3	Pulsa sobre el ícono de solicitudes de amistad.	
4		Muestra una lista de los usuarios que han solicitado amistad. Mostrando sobre cada usuario el nombre y la foto y dos botones para rechazar o aceptar.
5	Pulsa sobre el ícono de aceptar solicitud	
6		Notifica al usuario de que la solicitud ha sido aceptada

Nombre	Seleccionar día.	
Actor	Cliente de la app.	
Descripción	Un usuario registra su intención de salir un día específico.	
Precondiciones	El día seleccionado ha de ser posterior o igual al día actual. El día no puede estar ya seleccionado.	
Postcondiciones	El día queda seleccionado por el cliente y el resto de usuarios pueden verlo.	
Flujo de eventos		
Cliente		NightTime
1	Selecciona la página de calendario.	
2		Muestra el calendario del mes actual e información sobre el día actual
3	Pulsa sobre el día deseado	

4		Muestra el número de personas totales y amigos que van a salir ese día, y los eventos disponibles de los bares de la ciudad.
5	Pulsa el botón “Seleccionar día”	
6		Cambia el color del día seleccionado indicando que ha sido registrado.

8. Diseño del software

Este punto presenta el diseño estático y dinámico del proyecto. Describiendo tanto el comportamiento, mediante diagramas de secuencia, como la estructura de clases y objetos del sistema, mediante diagramas de clases. [34]

8.1 Diseño del servidor

Los siguientes diagramas describen la estructura de clases y el comportamiento del servidor.

En esta API se ha creado un sistema por capas como se explica en la Ilustración 4, donde los hermanos de una misma capa no se comunican entre sí, y cada capa, solo tiene acceso a los elementos que están por debajo de ellos. En esta estructura encontramos 8 tipos de objetos

- **Controladores:** Encargados de recibir la llamada HTTP e invocar a los servicios correspondientes para completar la acción. Estas llamadas pueden ir acompañadas de objetos simples como pueden ser el identificador de usuario y una cadena a modo de token de seguridad, u objetos complejos, como los objetos DTO, que almacenan otros objetos y datos en ellos. Estas clases están marcadas con el cian.
- **Servicios:** Clases llamadas por los controladores que realizar acciones específicas, utilizan los repositorios para obtener datos de la base de datos y posteriormente realizan acciones sobre estos datos. La lógica del sistema está en este tipo de archivos. Son inyectados en los controladores de manera que estos solo tengan acoplamiento con la interfaz, permitiendo cambiar su implementación sin modificar código existente. Tanto los servicios como sus interfaces son de color verde.
- **DTOs de entrada:** Los objetos DTOs (Data Transfer Object) son objetos encargados de transportar datos entre capas [35]. Estas clases tienen la palabra “DTO” en el nombre y se utilizan como clases de entrada de datos, es decir, serán usadas por el controlador para mandar información específica a los servicios. Esto reduce el acoplamiento ya que los controladores no tienen que conocer la implementación de los modelos, sino que pueden transmitir información parcial que será completada por el servicio. Utilizarán el color amarillo.
- **Repositorios:** Son utilizados por los servicios y permiten realizar acciones sobre la base de datos, tales como adiciones o recogida de datos. Mediante los modelos mencionados anteriormente, podemos realizar consultas y ediciones sin utilizar código SQL, ya que Spring JPA crea de manera automática consultas en función del nombre de las funciones de los repositorios. De esta manera podemos utilizar los repositorios para transformar las devoluciones de la base de datos en objetos Kotlin. Su color en los diagramas será el rosa.

- **Proyecciones y vistas:** Estos son otro tipo de objeto DTO, sin embargo, estos serán utilizados exclusivamente como vistas, es decir, como salida de datos. Se pueden distinguir entre dos tipos diferentes. Las vistas, con el sufijo “View”, y las proyecciones, con el sufijo “Projection”. Las vistas son objetos creados a partir de otras vistas o mediante una factoría, aquí se hace uso de las clases “data” de Kotlin, que autogeneran código y simplifican la implementación, facilitando así el proceso de mantenimiento. Las proyecciones son interfaces que mediante Spring JPA es posible crear un objeto a partir de estas cuando se recogen datos mediante un repositorio. Su color representativo será el morado.
- **Útil:** Objetos que realizan funciones muy concretas como TokenSimple, que almacena y comprueba la validez de los tokens de los usuarios conectados, factorías para la creación de vistas y objetos enumerados. Están marcados con el color azul.
- **Excepciones:** Para la implementación de este proyecto se ha utilizado excepciones personalizadas para crear un sistema de comprobación de errores en la inserción/edición de elementos. Estas excepciones son principalmente lanzadas por los servicios, pero también hay otras clases que pueden lanzarlas, como TokenSimple. Están marcados con el color rojo.
- **Modelo:** La conexión con la base de datos se realiza mediante el framework JPA, esto permite la transformación de clases Java o Kotlin en modelos SQL, tables y columnas, agilizando el desarrollo y facilitando la comunicación entre sistemas. Estarán representados con el color naranja en los siguientes diagramas.

8.1.1 Diseño estático

Debido a que el diagrama de clase del sistema sería demasiado grande para una página, se ha decidido fragmentar el esquema. De esta manera se puede representar en detalle las relaciones que tienen los controladores, servicios y modelo por separado. En este punto solo se mostrará un diagrama de cada capa, el resto de esquemas se encuentran en el anexo 16.2

Este primer esquema presenta un diagrama de clase que abarca desde el punto de entrada BarRestController, controlador Rest que gestiona tanto la creación y eliminado de bares como la visualización de estos, hasta las relaciones de las interfaces de los servicios.

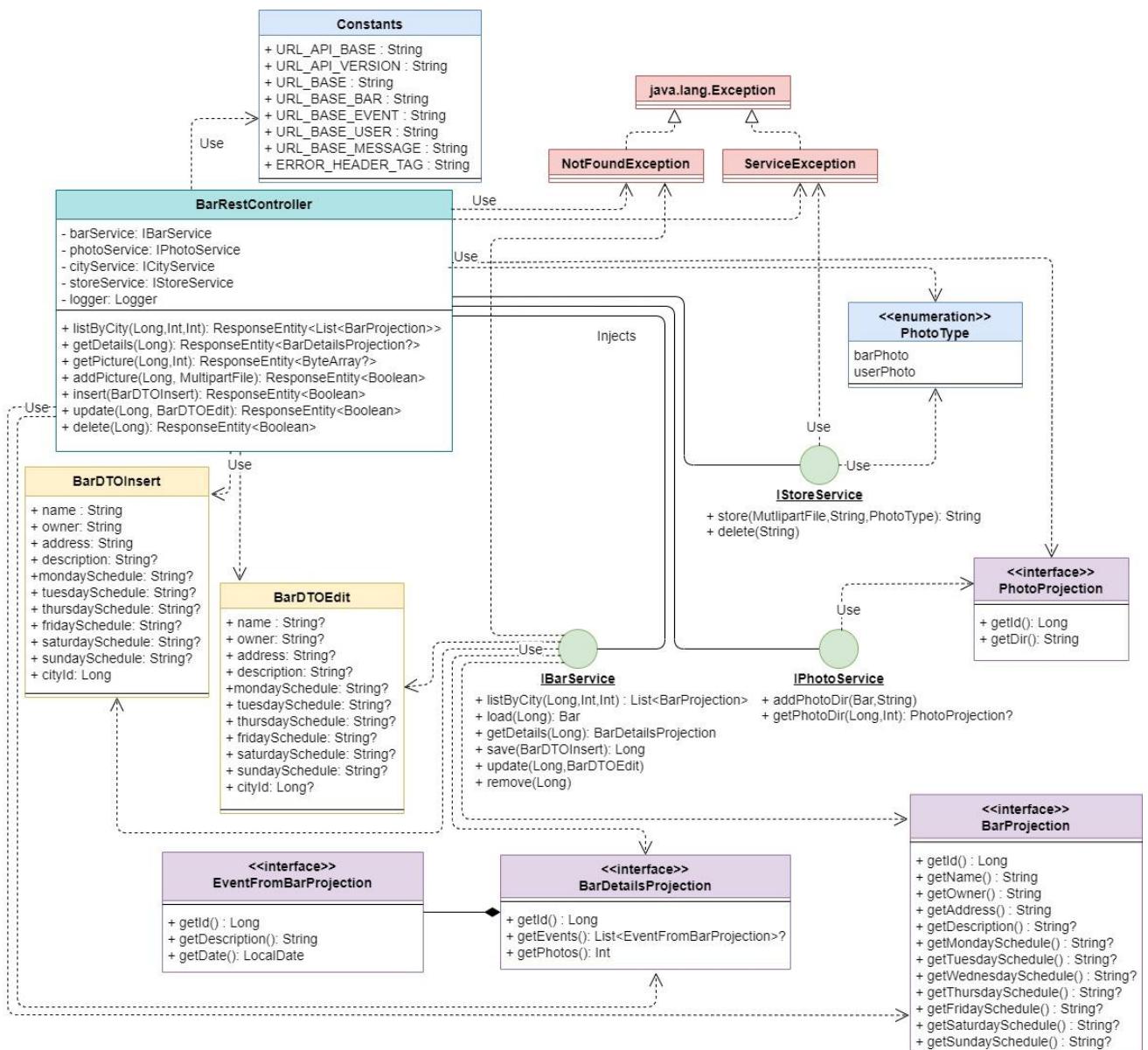


Ilustración 14: Diagrama de clase del controlador de bar

En el diagrama de clase del servicio de bares solo se muestran las relaciones de la clase modelo Bar con clases ya existentes en el diagrama. Se mostrará su representación completa en el diagrama de modelos del servidor.

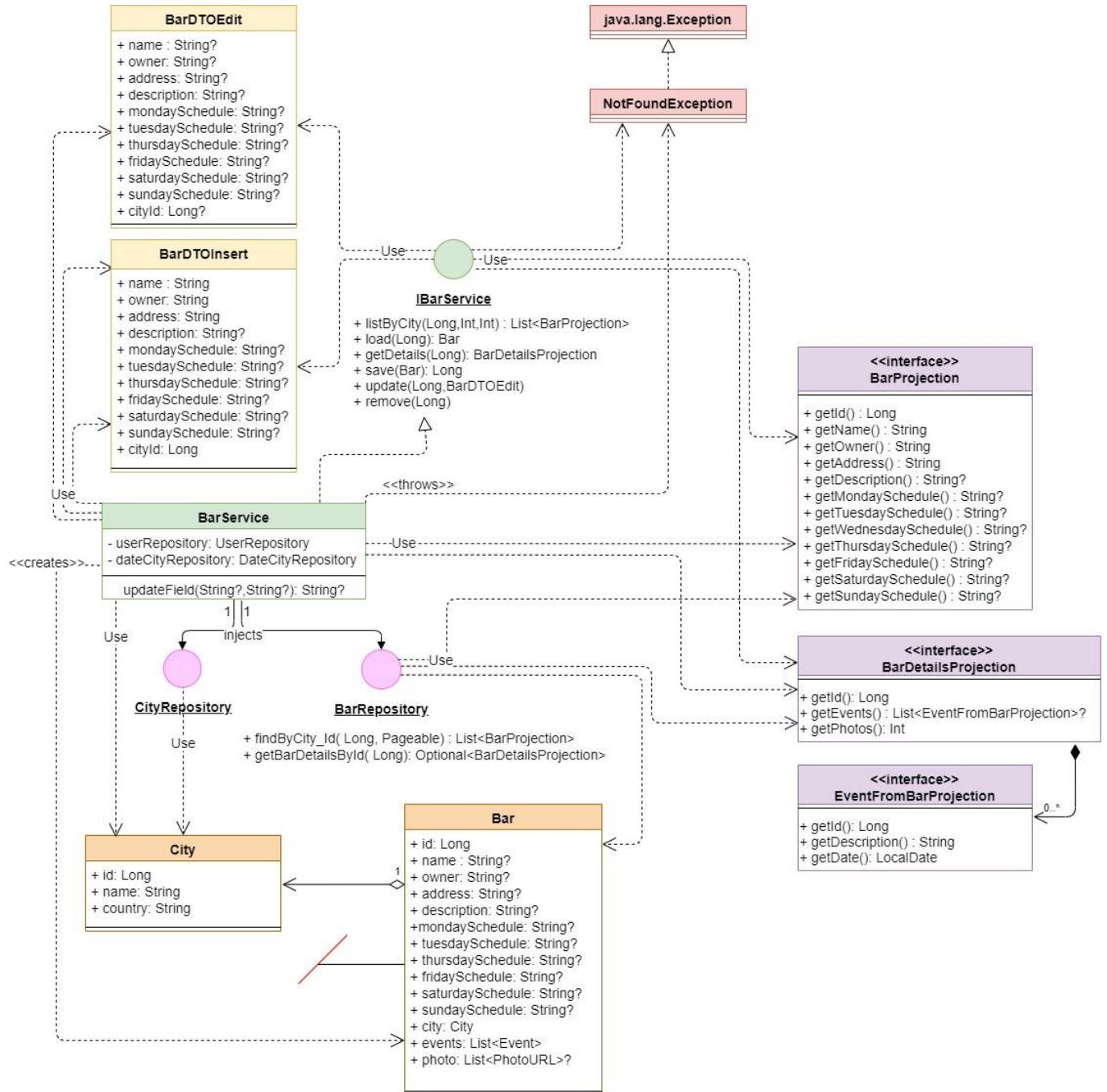


Ilustración 15: Diagrama de clase del servicio de bares.

Finalmente se describe la cuarta capa, el modelo de memoria persistente. Aquí se muestran las entidades de JPA.

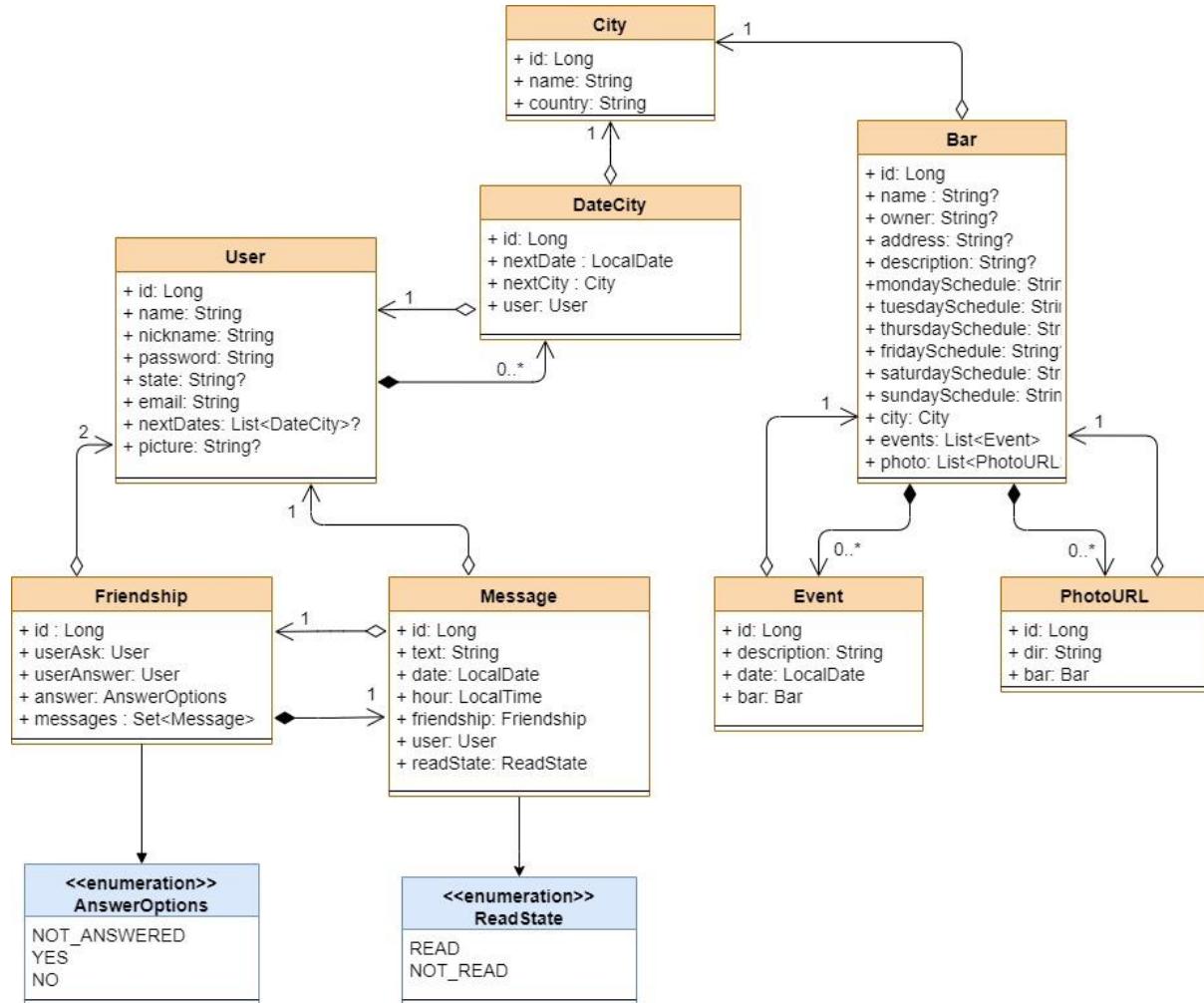


Ilustración 16: Diagrama de clases de los modelos de datos

8.1.2 Diseño dinámico

Aquí se describe el comportamiento del sistema a través del tiempo. Utilizando como medio los diagramas de secuencia de sistema, podemos describir cada paso de como las clases se comunican internamente para obtener la salida esperada.

Los dos diagramas muestran el proceso para formar una amistad entre dos usuarios.

El primer usuario propondrá la amistad, y como se puede ver en el esquema a la hora de guardar la nueva relación no se especifica el valor de AnsweredOptions (ilustración 23, el modelo de Amistades contiene un atributo especificando si la relación ha sido aceptada), esto es porque siempre se guardará como pendiente y solo puede ser actualizada, tanto aceptada como rechazada, por el usuario al que se mandó la petición.

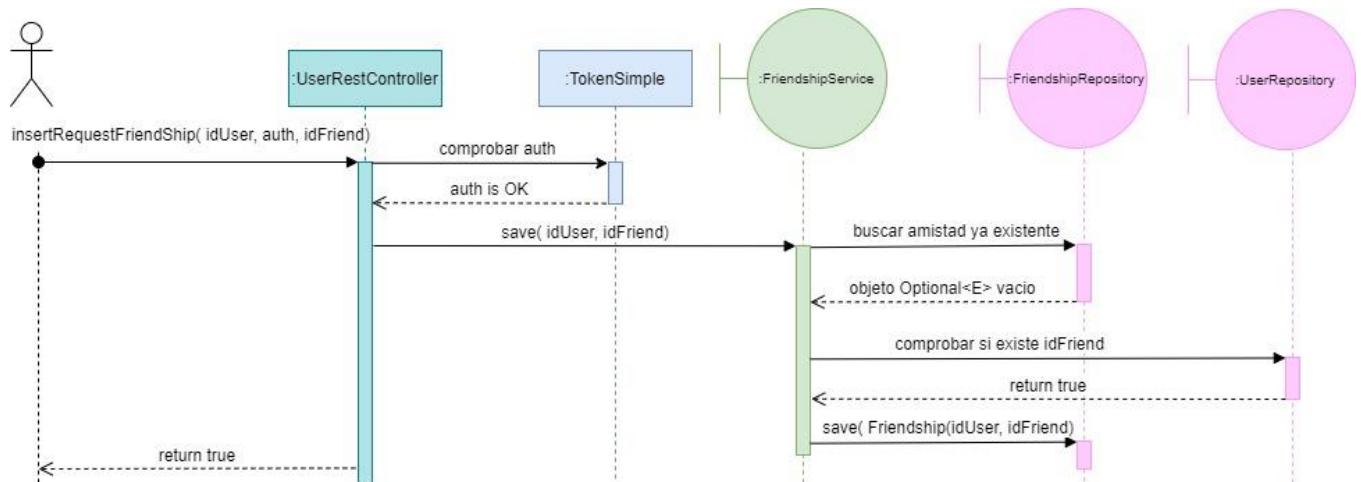


Ilustración 17: Diagrama de secuencia de sistema de mandar una petición de amistad

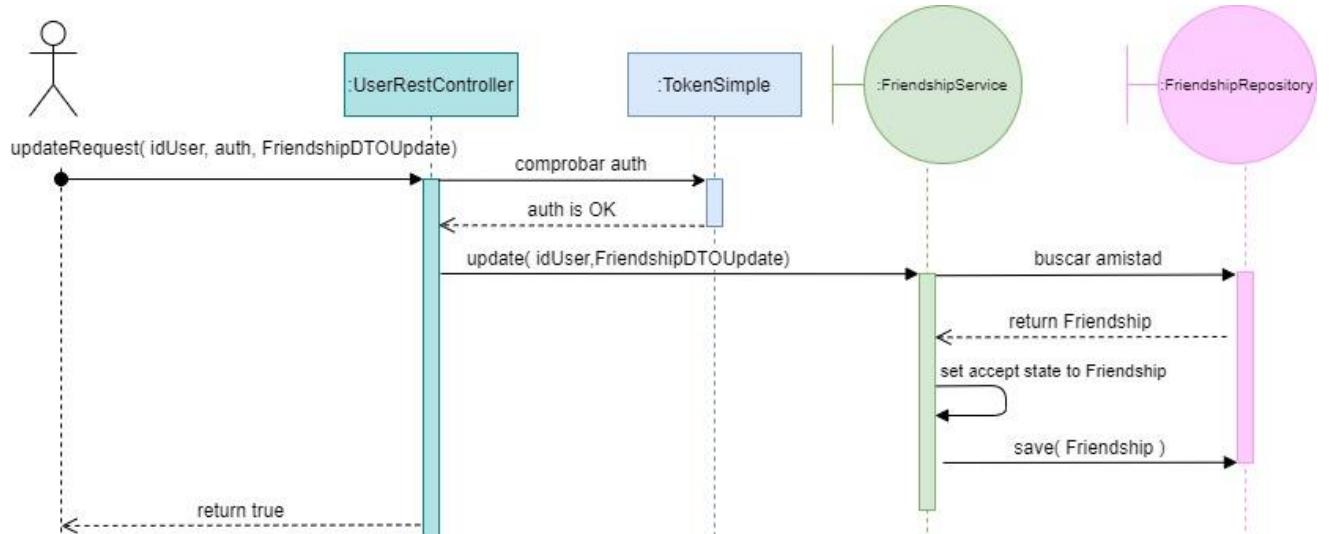


Ilustración 18: Diagrama de secuencia de sistema de aceptar una solicitud de amistad

El resto de diagramas de secuencia de sistema del servidor, se encuentran en el anexo 16.3.

8.2 Diseño de la aplicación

En este punto se describirá la aplicación móvil de la misma manera que se describió el servidor, es decir, el análisis estático y dinámico. [34]

8.2.1 Diseño estático

El siguiente diagrama describe la estructura de clases de la aplicación. En este esquema también hay un código de colores para distinguir las distintas partes que componen el sistema.

Las clases que contienen lógica básica, como puede ser la inicialización de la aplicación o la creación de los días del mes y enumerados, están definidas en color azul.

La interfaz encargada de realizar las llamadas a la API y transformar los datos de retorno en objetos DTO está marcada con el morado y se llama `NightTimeService`. Esta interfaz tiene un objeto *singleton* [36] que hace uso de Retrofit 2 y Moshi para realizar las solicitudes HTTP.

Los `ViewModels` son las clases de almacenamiento de datos que serán utilizados por las vistas. Estás clases contienen datos del tipo `State<E>` lo que permite observar los cambios y notificar a la vista para que sean actualizados. Además, estas clases también se encargan de recoger los datos del servidor necesarios para la vista. Por eso casi todas las clases tienen una referencia a `NightTimeService`. Excepto `ChatViewModel`, que se hace uso de una interfaz de un repositorio (marcado en rojo) para abstraer la implementación. De esta manera en un futuro añadir se puede añadir una vía de recogida de datos de la base de datos local además de la ya existente vía HTTP. Estas clases están marcadas en amarillo.

Los objetos DTO son objetos de transmisión de datos entre capas o almacenamiento de datos para los `viewModels`. Generalmente transmiten datos entre la interfaz de conexión con el servidor y los `viewModels`, o entre los `viewModels` y las vistas. Están resaltadas con el color naranja.

Finalmente, las vistas, o funciones *composables* están marcadas en verde. Estás funciones pueden ser de dos tipos: *statefull* o *stateless*. La diferencia está en que el primer tipo recibe tipos de datos complejos, como un `viewModel`, y llama a las funciones *stateless* enviando tipos de datos simples o DTOs. Por este motivo solo aparecen en el diagrama las funciones *statefull*.

El punto de inicio del sistema es el `MainActivity` que al inicio realiza la función de login en el servidor dando paso a la vista central de nuestra aplicación. En caso de que el login no se haya podido realizar porque no hay un usuario previo guardado o bien porque las credenciales no coinciden se mostrará la vista de login.

Esto es importante porque en el diagrama se puede ver como `MainActivity` crea `UserViewModel`, pues es la clase encargada de realizar el login, y tiene acceso a dos vistas, `NightTimeApp` y `LoginArchitecture`. Esto es destacable porque ambas funciones *composables* crean un gráfico de navegación, que es lo que permite moverse entre distintas vistas en un determinado alcance.

Además, `NightTimeApp` pese a ser una función *composable*, es capaz de crear objetos como `ChatListener` y llamar a otras funciones como `CalendarInit` que crearán una factoría de `CalendarViewModel` para construir este objeto con el token de usuario como variable privada.

Al utilizar Jetpack Compose, las vistas están definidas en Kotlin, por lo que sin ser clases son capaces de invocar otros objetos.

NightTime: Desarrollo de una red social basada en la actividad nocturna

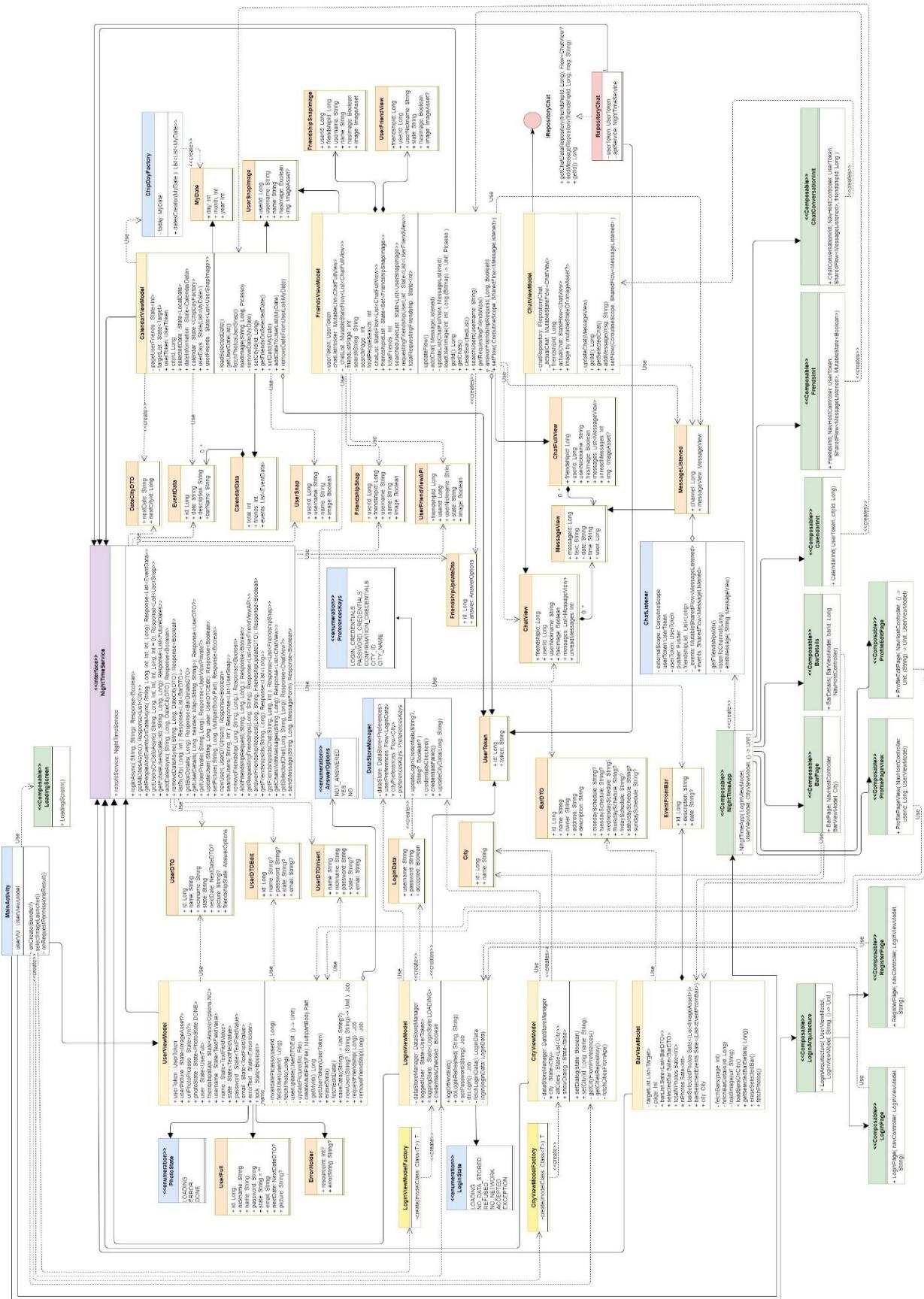


Ilustración 19: Diagrama de clase de la aplicación móvil

8.2.2 Diseño dinámico

Aquí se describe el comportamiento del sistema a través del tiempo. Utilizando como medio los diagramas de secuencia de sistema, podemos describir cada paso de como las clases se comunican internamente para obtener la salida esperada.

El primer diagrama muestra el proceso para solicitar una amistad. Esto empieza desde que se abre la aplicación y el usuario tiene que interactuar con los botones inferiores para cambiar a la pestaña de amigos. Donde a través de un buscador, se puede buscar a la persona tanto por nombre de usuario como nombre real. Una vez encontrado, tocándolo se accede a la vista de perfil, con un botón en la parte inferior que muestra el estado de la amistad. Si no hay una amistad el botón ofrece la opción a enviar una petición, si ya existía una amistad, el botón ofrecerá eliminar esta amistad.

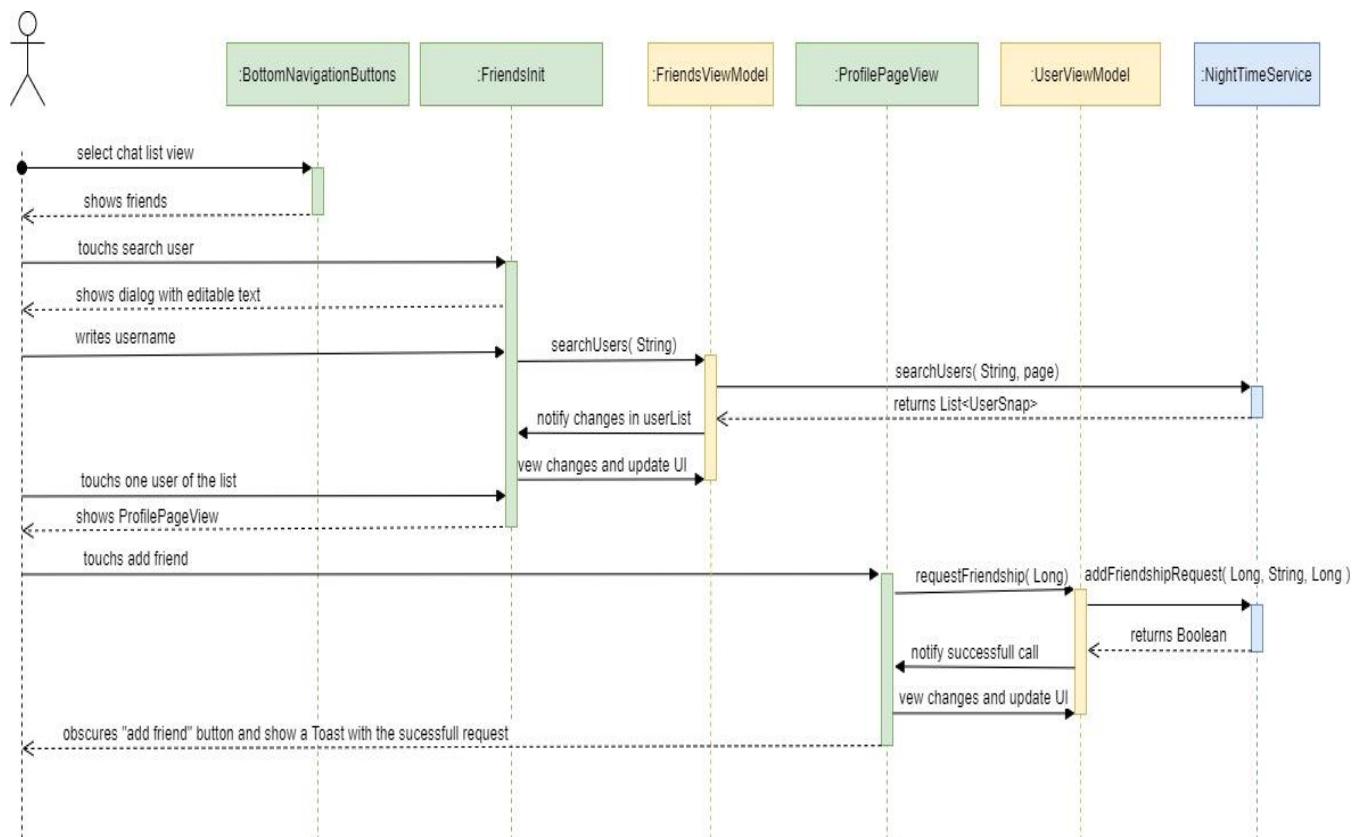


Ilustración 20: Diagrama de secuencia de sistema de enviar una petición de amistad

Este diagrama de secuencia de sistema muestra el proceso desde abrir la aplicación hasta seleccionar un día. Ya que la vista del calendario es la página inicial de la aplicación no es necesario interactuar con los botones de navegación. Una vez se abre la vista, CalendarInit crea el viewModel necesario y este en la inicialización hace la función setDate con la fecha actual, de

esta manera se realiza la llamada al servidor para obtener los datos del día a mostrar. Si el usuario selecciona un día lo primero que se confirma es si ese día pertenece al mes que se está mostrando actualmente, si esto no es cierto es necesario volver a cargar el calendario con el mes adecuado, y en ambas situaciones se vuelve a realizar otra llamada al servidor para obtener la información del día seleccionado. Una vez que el usuario encuentre el día deseado, este puede tocar el botón con la etiqueta “Seleccionar día” y automáticamente el día será marcado en otro color y actualizado en el servidor. Si hay algún problema actualizando el servidor el día se desmarcara solo.

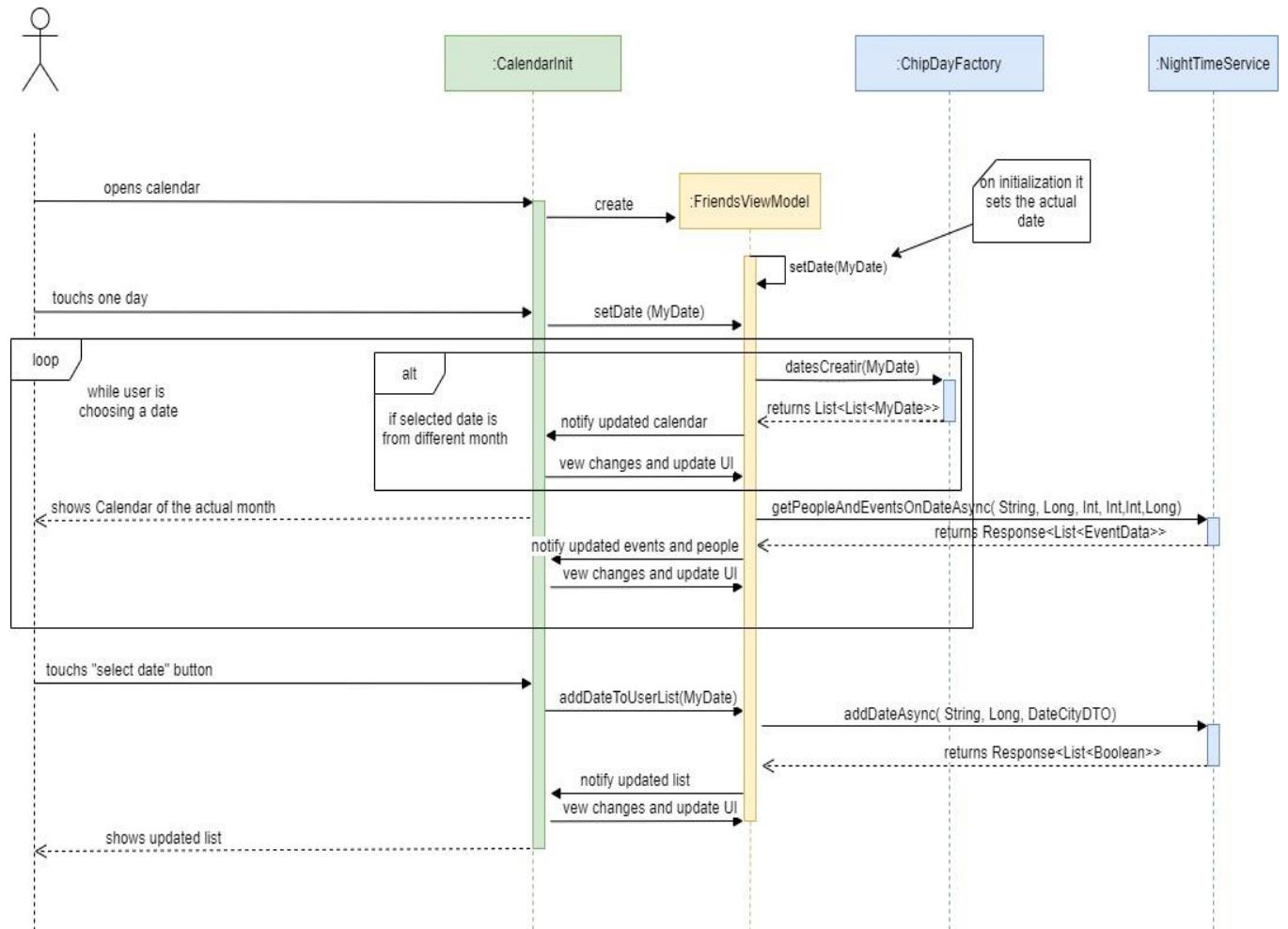


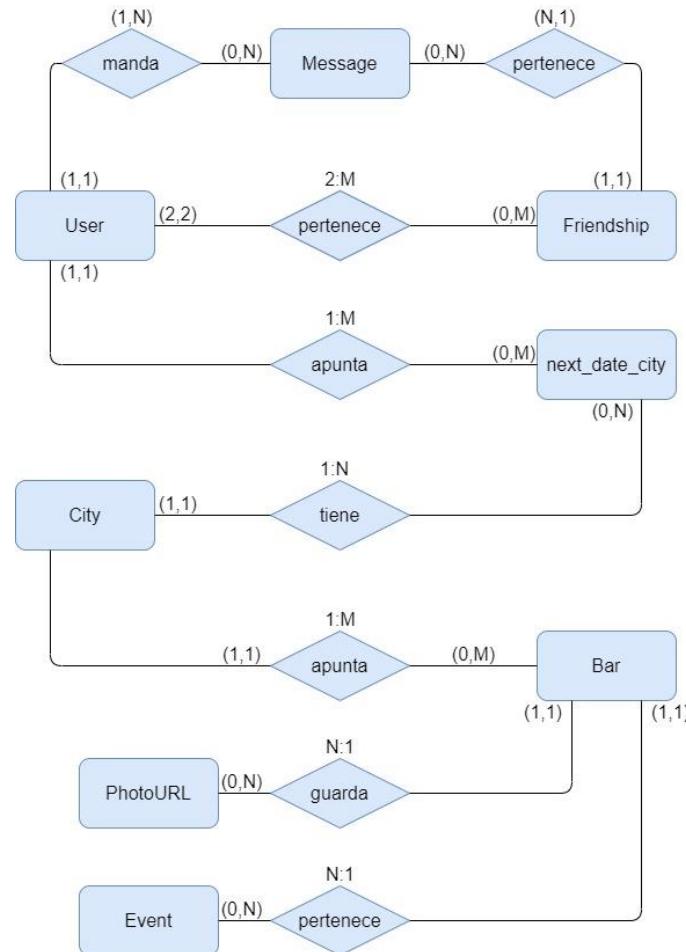
Ilustración 21: Diagrama de secuencia de sistema de seleccionar un día para salir

9. Gestión de datos e información

Este sistema almacena datos en la base de datos del servidor, en la ilustración 12 se pudo ver como los modelos JPA representaban las tablas así que ahora se mostrará el diagrama entidad-relación con la descripción de los atributos.

En primer lugar, se describirá brevemente que representa cada una de las entidades:

- **Usuarios:** Cada uno de los clientes de la app que tenga usuario y contraseña, referencia inequívocamente a una persona.
- **Friendship:** Relaciones entre dos usuarios, un usuario la solicita y el otro usuario debe aceptarla. En caso de ser rechazada, se eliminará la tupla.
- **Message:** Mensajes enviados entre dos usuarios que pertenecen a una amistad. Cada mensaje tiene, texto, hora y día de llegada al servidor y está firmado por un usuario.
- **Next_date_city:** Días seleccionados por el usuario en una ciudad determinada.
- **City:** Ciudades guardadas en el sistema.
- **Bar:** Entidades de los bares en cada ciudad con su descripción.
- **PhotoURL:** Relaciones entre los bares y las fotos guardadas en el servidor, se relaciona la dirección y el identificador del bar.
- **Event:** Eventos asociados a los bares. Están fijados para un día determinado y pueden contener una descripción.



Ya que hay dos entidades que guardan imágenes (Usuarios y Bares) es necesario estandarizar el formato de recogida y almacenamiento. Este formato es el “JPG” ya que es uno de los más livianos y es esperado que se guarden una gran cantidad de fotos.

A continuación, se describirán los atributos de dichas tablas.

Tabla *Users*:

CAMPO	TIPO DE DATO	DESCRIPCIÓN
Id	BigInt (20)	Identificador del usuario.
Email	VarChar (255)	Email.
Nombre	VarChar (255)	Nombre real del usuario.
Nickname	VarChar (30)	Nombre de usuario, no repetible.
Password	VarChar (255)	Contraseña encriptada con AES
Picture	VarChar (255)	Dirección de almacenamiento de la foto.
State	VarChar (255)	Estado público del usuario.

Tabla *Friendship*:

CAMPO	TIPO DE DATO	DESCRIPCIÓN
Id	BigInt (20)	Identificador de la amistad.
UserAsk	BigInt (20)	Id del usuario que mando la solicitud de amistad.
UserAnswer	BigInt (20)	Id del usuario que recibió la petición de amistad.
Answer	Enumerated of AnswerOptions	Estado de la petición, puede ser SI, No o “Aun no respondida”.

El resto de tablas junto con otro gráfico sobre la relación entre tablas se pueden encontrar en el anexo

Además, al ser un sistema que haga uso de mensajes HTTP es necesario el envío y recepción de datos en algún formato que sea soportado por este protocolo. Por ese motivo se hace uso de paquetes en JSON ya que se caracterizan por ser más liviano que XML y ser fácil de entender para las personas.

10. Pruebas llevadas a cabo

Para realizar el testeo del servidor se utilizaron distintas herramientas a lo largo del desarrollo. Una vez establecido el diseño y la arquitectura se implementaron los casos de uso más básicos, que son la gestión de las entidades, es decir, adiciones y actualizaciones.

Estos casos base fueron testeados con la herramienta automática Postman, que permite especificar peticiones HTTP, ejecutarlas en orden y comprobar el resultado de cada llamada. Esto permite la verificación del funcionamiento del sistema y asegurar que las siguientes iteraciones no interfieran con el diseño ya implementado.

Algunos de los test que se llevaron a cabo son los siguientes:

	Descripción	Endpoint utilizado	Condición de éxito	Resultado
Gestión de usuarios	El nombre de usuario tiene más de 35 caracteres.	Insertar usuario.	El usuario no se inserta y devuelve un mensaje de error explicativo.	Correcto.
	Editar un usuario y dejar el campo de la contraseña vacío.	Editar usuario	Se actualizan los campos que tienen el formato correcto y se devuelve un mensaje explicativo para los campos erróneos.	Correcto.
	Realizar una inserción y actualizar la foto	Insertar usuario y añadir foto de perfil	Se crea el nuevo usuario y se guarda la foto.	Correcto.

	Descripción	Endpoint utilizado	Condición de éxito	Resultado
Gestión de amistades	Insertar una amistad de un usuario consigo mismo	Insertar amistad.	La amistad no se inserta y se devuelve un mensaje explicativo.	Correcto.
	Insertar una amistad ya existente, pero con los usuarios invertidos.	Insertar amistad	La amistad no se inserta y se devuelve un mensaje explicativo.	Correcto.
	El usuario que mando la solicitud intenta aceptar la amistad.	Actualizar amistad	Se devuelve un código de error y un mensaje de que el usuario no tiene permisos.	Correcto.
	El usuario que recibe la petición la rechaza.	Actualizar amistad	Se elimina la amistad y se devuelve un código de éxito.	Correcto.

Una vez que el proyecto se encontraba en una fase más avanzada, las pruebas necesarias requerían mayor control sobre los datos mostrados. Por este motivo se utilizó el framework Swagger, que crea una endpoint donde se muestra documentación autogenerada y la posibilidad de probar directamente las funciones del servidor.

De esta manera se puede confirmar tanto que las listas paginadas como las vistas construidas con elementos de varias tablas muestran los datos correctos.

Pruebas realizadas.	Descripción	Endpoint utilizado	Condición de éxito	Resultado
	El recuento de usuarios totales y amigos en una fecha determinada es correcto	Conseguir amigos y total en una fecha.	Se devuelve el número correcto de usuarios totales y amigos y los eventos de la ciudad y la fecha.	Correcto.
	Los horarios de los bares se muestran adecuadamente si su estado es nulo.	Conseguir información de los bares.	Los horarios de los bares en estado nulo se muestran como "Cerrado" para ese día	Correcto.
	Los mensajes conservan el formato adecuado y bloquean las inyecciones de código.	Mandar mensaje.	Los saltos de línea y emojis llegan a su destino y las inyecciones SQL se tratan como texto plano.	Correcto.

La aplicación cliente no hizo uso de un testeo con herramientas, sino que las distintas funcionalidades se ejecutaban en un dispositivo real y en uno virtual. Asegurando así que el sistema funciona en distintas versiones de distintos sistemas operativos.

11. Manual de usuario

En este punto se explicará el uso del proyecto desarrollado junto con los requisitos de instalación y las instrucciones para instalarlo. Primero se expondrá el servidor, pues la aplicación no podrá pasar del sistema de inicio si no es capaz de conectarse con un endpoint y validar las credenciales.

11.1 Manual del servidor

Este sistema no está hecho para funcionar por sí solo, sino que se apoya en el cliente para que los usuarios finales puedan disfrutar de sus prestaciones. Aquí se explicará lo necesario para poder levantar el servidor y permitir a los dispositivos comunicarse entre sí.

11.1.1 Requerimientos

Para conseguir le servidor levantado y listo para el funcionamiento normal, es necesario ejecutar el programa en un servidor apache que al menos la versión 8 de la máquina virtual de Java y acceso a una base de datos.

11.1.2 Manual de instalación

Para instalar el servidor es necesario la descarga del código fuente desde un repositorio git. Para ello se utiliza el siguiente comando:

Git clone <https://github.com/minimerlyn/NightTimeAPIRest.git>

Al iniciar la aplicación, esta buscará una base de datos en el mismo sistema en el que se está ejecutando en el puerto 3306, el acceso a esta base de datos, se necesita sustituir las credenciales en el archivo “application.properties” por unas credenciales con permisos de creación y modificación de bases de datos. Este archivo se puede encontrar en el directorio NightTimeAPIRest/src/main/resources/application.properties y las credenciales se localizan en las líneas 8 y 10 como se muestran en la ilustración 18.

```
7 #user name  
3 spring.datasource.username = root1  
3 #password  
3 spring.datasource.password = toor
```

Ilustración 22: Credenciales actuales para la base de datos.

En caso de que no exista ninguna base de datos o que haya surgido un problema en la inicialización, el sistema hará uso de la dependencia H2 para crear una pequeña base de datos con una capacidad máxima de 4 GB

También es recomendable modificar la línea 18 con el valor “*none*” pues si se mantiene el valor actual la base de datos se volverá a crear cada vez que el servidor sea reiniciado, borrando así toda la información existente anteriormente.

```
16 # Values: none, validate, update, create, create-drop <= default
17 #When is ready to upload set to "none"
18 spring.jpa.hibernate.ddl-auto = create-drop
--
```

Ilustración 23: Configuración de la base de datos

Una vez modificado el sistema debe ser subirlo a un servidor apache y está listo para ejecutarse. El propio proyecto creará las tablas y entidades necesarias para el correcto funcionamiento del sistema.

11.1.3 Manual de uso

Este sistema está ampliamente explicado en la documentación ofrecida por Swagger en la dirección <http://xxx.xxx.xxx:8080/swagger-ui.html#> siendo xxx.xxx.xxx.xxx la dirección IP interna del servidor una vez levantado.

11.2 Manual de la aplicación

Aquí se detalla los requisitos, manual de instalación y uso de la aplicación móvil.

11.2.1 Requisitos de la instalación

El objetivo de este proyecto es el desarrollo de una aplicación para sistemas móviles con sistema operativo Android 5 [37]o superior y un servidor con el que se pueda comunicar para obtener y actualizar información.

11.2.2 Manual de instalación

Para la instalación de la aplicación solo es necesaria la ejecución, en un dispositivo móvil, del archivo descargado en el siguiente enlace.

<https://github.com/minimerlyn/NightTimeTFG/releases/download/1/night-time-app.apk>

Es posible que el sistema advierta de que la fuente no es fiable, pues la instalación de aplicaciones mediante el APK desde el navegador o desde un cable USB es considerado una posible fuente de malware y por lo tanto se requiere la confirmación del usuario.

11.2.3 Manual de uso

Esta aplicación hace uso de Material Design, por lo que el diseño de la interfaz de usuario está implementado de una manera intuitiva y haciendo uso de los colores y formas para transmitir acciones. La navegación dentro de la aplicación y las funcionalidades se explicarán a continuación.

Una vez instalada la app, al ejecutarla por primera vez aparecerá la página de login como se muestra en la ilustración 20. Desde aquí se puede crear una cuenta en el botón 1, o iniciar sesión si la cuenta ya ha sido creada. En caso de haber iniciado sesión previamente en el dispositivo, este guardará las credenciales para agilizar el proceso de verificación.

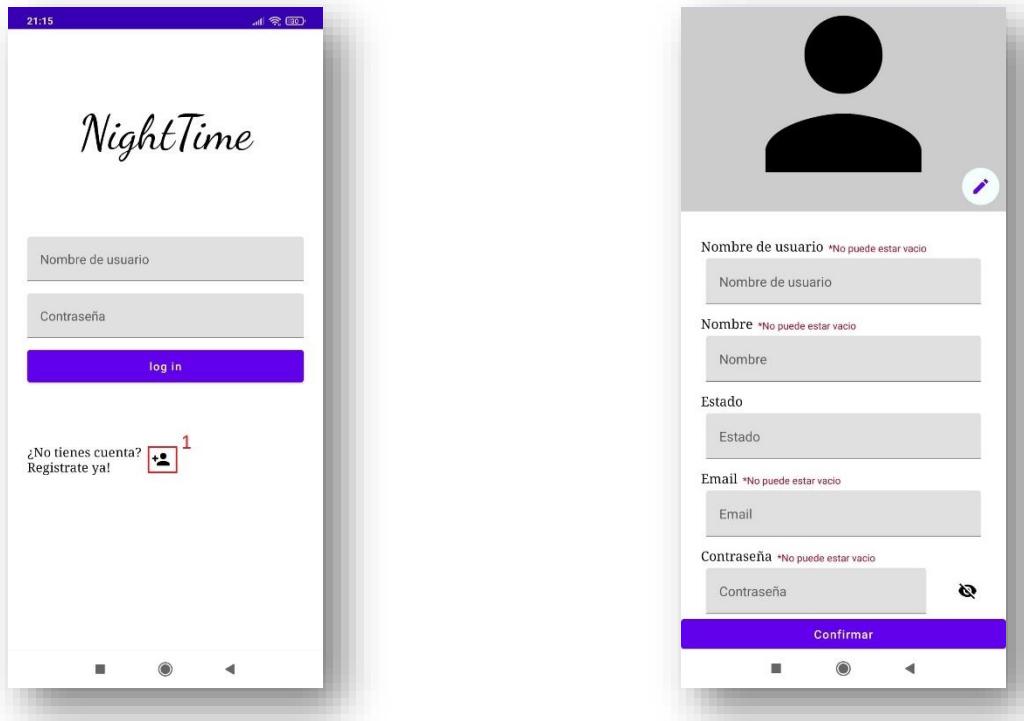


Ilustración 24: Pagina de login y registro de la app

En el caso de registrar una nueva cuenta, el sistema solicitará todos los datos necesarios marcando aquellos que no pueden estar vacíos con un asterisco, y ofreciendo la posibilidad de elegir una foto desde la galería interna del dispositivo.

Una vez autenticado en el servidor, se mostrará la vista de la ilustración 21 izquierda, donde se puede ver el calendario actual con los días marcados por el usuario. En la parte inferior se muestra en una pantalla deslizante los eventos de ese mismo día y un par de cuadros de texto mostrando los amigos y la gente total que ha marcado ese día. Tocando un día diferente esta información se actualizará. Pulsando el botón 1 de la imagen 21 se seleccionará o deselegionará el día, tocando el botón 2 se mostrará la lista de usuarios amigos que saldrán ese día y mediante el botón 3 de la parte superior, se mostrará la lista de ciudades de la app.

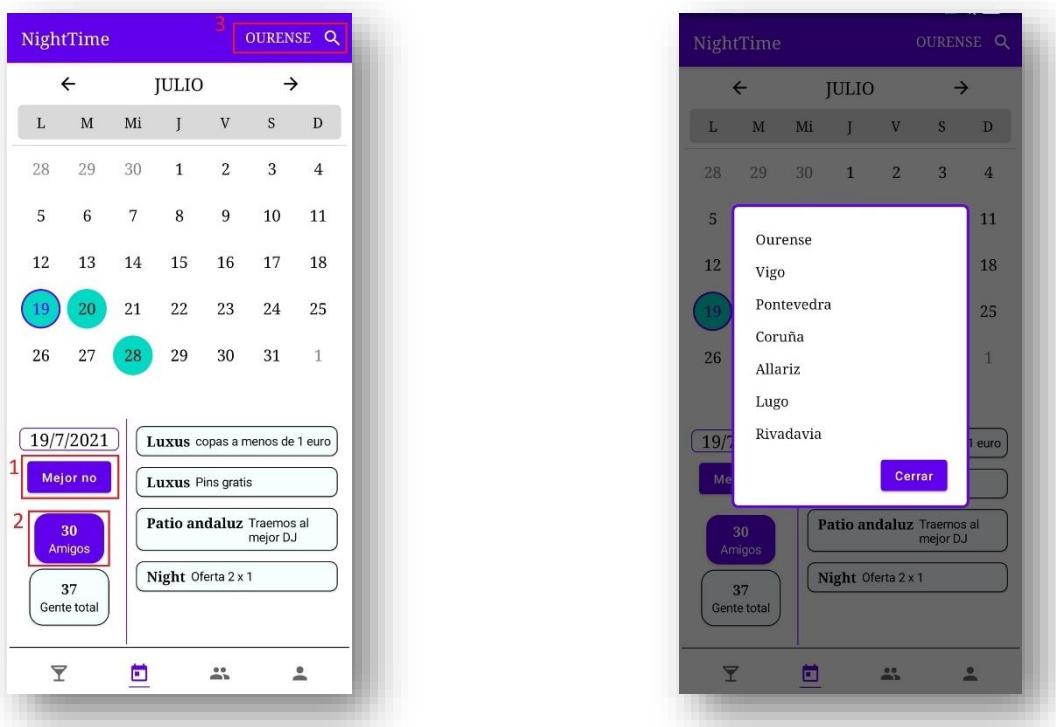


Ilustración 25: Pagina de calendario

En la parte inferior de la pantalla se muestra la barra de navegación con las 4 pantallas principales del dispositivo. En esta explicación se irán mostrando de izquierda a derecha empezando por la vista de los bares.

La ilustración 22 izquierda muestra la lista de los bares que hay en la ciudad seleccionada, junto con una descripción y una pequeña barra que muestra los días que el bar está abierto. Esta lista está paginada para evitar un consumo de memoria excesivo.

Tocando cada uno de los bares mostrados se accede la vista detalle de cada uno de los bares, esto se ve en la ilustración 22 derecha. Aquí se enseña toda la información asociada al bar seleccionado junto con el botón 1 que accede a Google Maps para mostrar la ruta más corta. Esta pantalla se puede deslizar verticalmente para mostrar la lista de los eventos futuros y las fotos de los bares. Las imágenes se cargan en una lista horizontal pudiéndose ampliar al tocarlas, y los eventos se muestran en una lista vertical ordenada por proximidad en el tiempo.

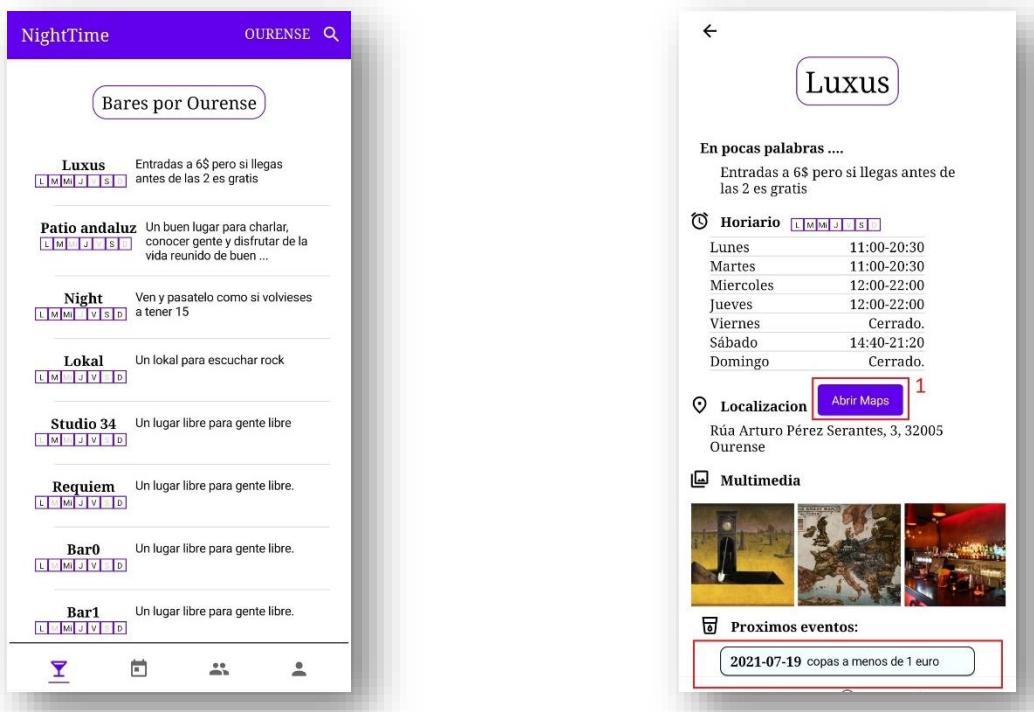


Ilustración 26: Pagina de los bares

La siguiente vista de la barra de navegación, es la vista de amigos, ilustración 23 izquierda, donde se puede ver las conversaciones de chat iniciadas y acceder a las funciones relacionadas con el contacto con otras personas. Las conversaciones activas muestran el último mensaje y el número de mensajes no leídos. Pulsando sobre la conversación se puede acceder a la vista del chat, ilustración 23 derecha, desde donde se pueden mandar mensajes al otro usuario, y acceder a su perfil pulsando sobre su nombre o foto en la parte superior de la pantalla.

Volviendo a la ilustración 23 derecha, mediante el botón 1 se accede al buscador de usuarios, donde haciendo una búsqueda por nombre real o de usuario se puede encontrar al resto de personas. La vista de usuario cambia dependiendo el estado de la amistad, es decir si ambos usuarios son amigos se mostrará la próxima fecha marcada y un botón ofreciendo eliminar dicha amistad.

El icono 2 muestra el número de peticiones de amistad recibidas y pendientes de respuesta. Y el botón flotante 3, permite acceder a la lista completa de usuarios donde se puede iniciar una convención con cualquiera de ellos.

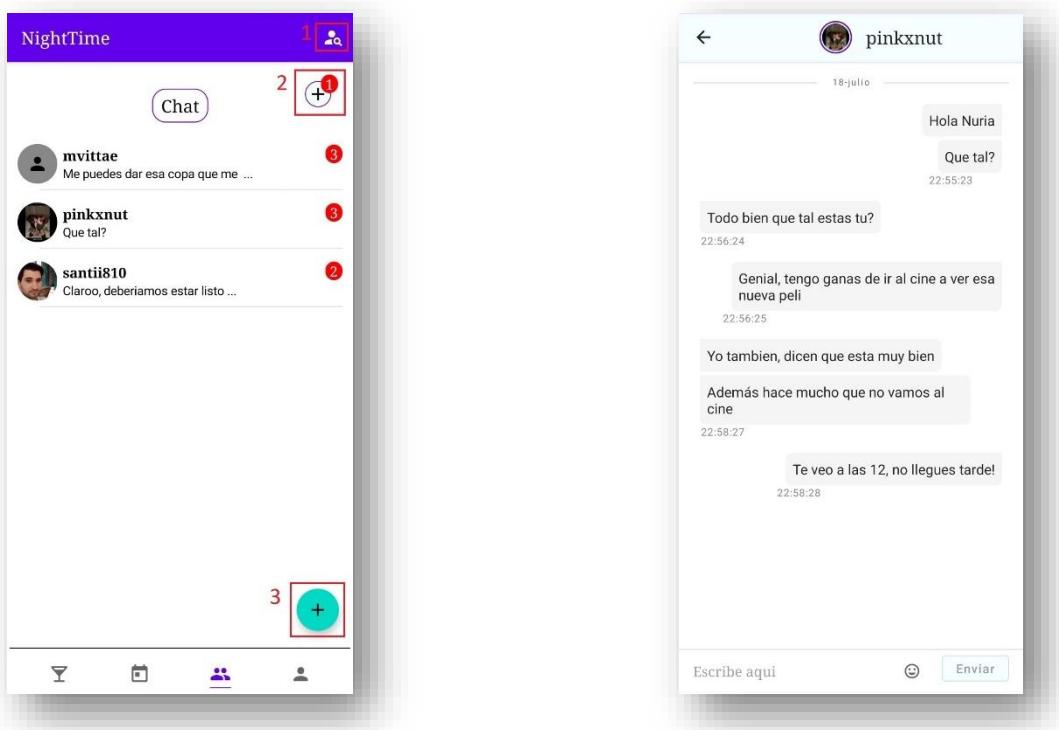


Ilustración 27: Página de amigos y chat

Finalmente la última vista muestra el perfil del usuario logueado, como se ve en la ilustración 24 izquierda, con un botón flotante que permite acceder a la edición de perfil, ilustración 24 derecha. En esta vista se enseñan y se permiten actualizar la mayor parte de campos que fueron solicitados en el registro menos el nombre de usuario. Finalmente hay un par de botones inferiores que permiten conservar o descartar los cambios.

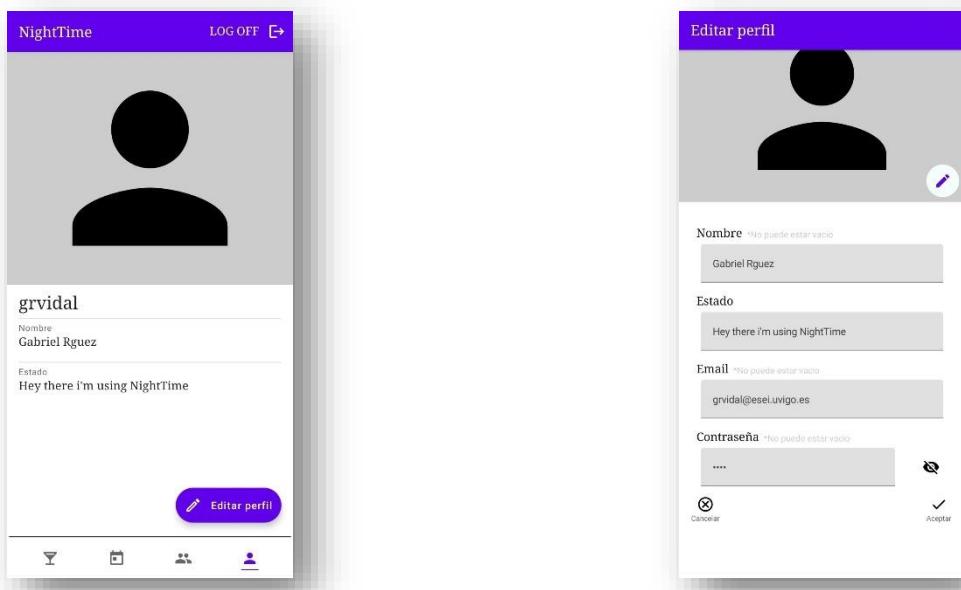


Ilustración 28: Página de perfil y edición de perfil

12. Principales aportaciones

Este proyecto permite la comunicación entre personas y la visualización de la información relativa a un día la manera más sencilla posible. Gracias a la combinación de distintas herramientas y códigos de terceros, este sistema es capaz de abarcar una gran cantidad de casos de uso y ofrecerlos al usuario de una manera intuitiva. Algunas de las principales aportaciones de la aplicación son las siguientes:

Utilidad: Mediante la visualización del calendario y los bares cercanos, es posible tener la información necesaria para tomar decisiones con más previsión que de la manera tradicional. Esto se suma al chat integrado, que permite la comunicación entre usuarios en tiempo real, minimizando así, el uso de aplicaciones involucradas en la tarea de organizar un evento o una reunión.

Intuitivo: Haciendo una interfaz que siga los principios de Material Design, ayuda a los usuarios a navegar por la aplicación de una manera fluida e intuitiva, minimizando la resistencia al cambio que suele observarse con la aparición de nuevas redes sociales.

Jetpack: Haciendo uso de la tecnología más actualizada, este sistema es capaz de funcionar en muchas versiones de distintos dispositivos de la manera esperada. Además, la librería Compose diseña las interfaces en código Kotlin, por lo que el tamaño de la app se reduce notablemente.

Ya que la aplicación necesita una fuente de información la implementación del servidor con arquitectura Rest es una ventaja. El hacer uso de una tecnología tan presentes en el mercado permite que este sistema sea exportado a otros dispositivos que hagan uso de este protocolo. Y ya que Swagger se ha incluido en el desarrollo, está ampliamente documentado para facilitar el mantenimiento y el testeo de futuras funcionalidades.

13. Conclusiones

El resultado de este proyecto, es un sistema estable que funciona como red social y planificador de salidas nocturnas. Pese a que pueda considerarse incompleto por la falta de un sistema de actualización de bares, el objetivo principal era el desarrollo de una aplicación móvil funcional, por lo que el desarrollo ha sido un éxito.

13.1 Conclusiones técnicas

Las tecnologías utilizadas han facilitado la implementación de muchas funcionalidades que de otra manera hubiese sido mucho más difícil o requerido demasiado tiempo de aprendizaje. Pusher es un claro ejemplo de esto, ya que la creación desde 0 de un sistema que establezca una comunicación bidireccional entre cliente y servidor, es mucho más complejo que la implementación de un framework.

El framework Spring, también ha sido uno de los pilares del servidor, ya que ha permitido la creación del esqueleto del servidor y sus modelos de una manera especialmente cómoda y ahorrando tiempo en muchos aspectos. Gracias a las consultas automáticas de los repositorios, solo cuando estas necesitan uniones con otras tablas es necesario el uso de consultas personalizadas.

También el lenguaje, que uno de los motivos por los que fue elegido es el aumento de su popularidad y la necesidad de utilizarlo para el framework Compose, resultó en un aprendizaje no duro, pero costoso. Sin embargo, una vez familiarizado hay muchos elementos que reducen la cantidad de líneas necesarias para llevar a cabo prácticas comunes y agilizan el desarrollo, especialmente las *data class*, la nulabilidad y la manera de crear *Singleton*.

13.2 Conclusiones personales

Este proyecto se ha iniciado con una gran ilusión y ganas de aprender nuevos lenguajes y herramientas, entre ellas Jetpack, ya que sin estar finalizada era bastante popular y proponía acercamientos interesantes al desarrollo de apps.

Este framework ofrece muchas facilidades a la hora de implementar y crear funcionalidades, ya que favorece la reusabilidad y el uso de estructuras predefinidas, pero por este mismo motivo el mantenimiento y cambio puede resultar algo más costoso ya que para satisfacer una nueva necesidad puede necesitarse modificar código compartido por varias entidades.

Después de haber trabajado con herramientas en fase alfa, considero que este framework es un gran apoyo para el desarrollo ágil de aplicaciones. A diferencia de la mayor parte de frameworks complejos, la documentación abunda y cada actualización está acompañada de toda la información relevante para actualizar el código obsoleto.

Finalmente, el diseño e implementación de este sistema ha sido un reto desde el principio, comenzando con el aprendizaje de un lenguaje y framework nuevo, el tiempo siempre ha sido un elemento muy presente, pero no asfixiante. Tras la creación de un sistema tan grande no pude hacer otra cosa que mirar atrás y darme cuenta de la cantidad de páginas y documentos que he tenido que leer para lograr que esto finalmente funcione. Al mismo tiempo, mirar adelante y ver la cantidad de información y conocimiento que me falta por aprender, y no son pocas las ganas de iniciar este camino.

14. Vías de trabajo futuras

Pese a que los objetivos hayan sido cumplidos, este proyecto puede ser ampliado de varias maneras. Como ya se mencionó en el punto 7.1, una vía de ampliación es la creación de una página web accesible para los dueños de los bares. Permitiendo en cualquier momento la actualización de sus datos y contenido multimedia.

Otro aspecto que permite una ampliación, es la implementación de un “modo noche” a la aplicación. Donde los usuarios puedan tener una vía de comunicación directa con el repertorio musical del local en el que esté presente, permitiendo al local saber que canción quieren los clientes. Esto también serviría para obtener información de la gente promedia que acude a cada bar y el estudio de mercado de los gustos de la población local.

Otro pequeño cambio es la inclusión de los usuarios en los eventos diarios. Ya que un usuario podría ofrecer su casa determinado día, visible únicamente para los amigos.

En el punto 5.2, se habló de la arquitectura de la aplicación, incluyendo el uso de un repositorio que actualmente solo tenía acceso al servidor Rest. Una mejora de este sistema, es la ampliación de fuentes de información, haciendo que el repositorio haga uso de la memoria local del dispositivo móvil, creando una base de datos local donde guarde las conversaciones de chat abiertas. De esta manera el servidor podría librarse de esta carga, reduciendo en gran medida el espacio esperado ocupado por la base de datos del servidor.

15. Bibliografía

- [1] J. Bruggeman, *Social Networks: An Introduction*, London, UK: Routledge, 2008.
- [2] L. Garton, C. Haythornthwaite y B. Wellman, *Studying online social networks*, Journal of Computer-Mediated Communication, 1997.
- [3] Android Developers, «AndroidX Overview,» 2020. [En línea]. Available: <https://developer.android.com/jetpack/androidx>. [Último acceso: 19 May 2021].
- [4] Android Developers, *Modern Android development: Android Jetpack, Kotlin, and more*, 2018.
- [5] Android Developers, «Layouts,» 2020. [En línea]. Available: <https://developer.android.com/guide/topics/ui/declaring-layout>. [Último acceso: 19 May 2021].
- [6] Kotlinlang, «Kotlin for Android,» 2021. [En línea]. Available: <https://kotlinlang.org/docs/android-overview.html>. [Último acceso: 19 May 2021].
- [7] L. Li y W. Chou, *Design and Describe REST API without Violating REST: A Petri Net Based Approach*, IEEE, 2011.
- [8] D. Connolly, «w3c.org,» 1999. [En línea]. Available: <https://www.w3.org/Protocols/rfc2616/rfc2616.html>. [Último acceso: 12 Junio 2021].
- [9] Spring, «Spring Boot 2.4.5,» 2021. [En línea]. Available: <https://spring.io/projects/spring-boot>. [Último acceso: 19 May 2021].
- [10] Spring, «Spring Data JPA,» 2021. [En línea]. Available: <https://spring.io/projects/spring-data-jpa>. [Último acceso: 20 May 2021].
- [11] J. Kouraklis, *MVVM as Design Pattern*, Apress, Berkeley, CA, 2016.
- [12] Android Developers, «Jetpack Compose: Beta overview,» Youtube, 2021. [En línea]. Available: <https://www.youtube.com/watch?v=Ef1xKWjA9E8>. [Último acceso: 20 May 2021].
- [13] I. N. Jim Arlow, *UML 2*, Madrid: Anaya multimedia, 2006.
- [14] M. Fowler y K. Scott, «UML gota a gota,» 2011. [En línea]. Available: https://books.google.es/books?hl=es&lr=&id=AL0YkFeaHwIC&oi=fnd&pg=PA37&dq=uml&ots=FxZRJ27mWr&sig=0fwV5mIS2FUyVFKZXSjb0qz2Rd4&redir_esc=y#v=onepage&q=uml&f=false. [Último acceso: 21 Mayo 2021].
- [15] Android Developers, «Single activity: Why, when, and how (Android Dev Summit '18),» 2018. [En línea]. Available: <https://www.youtube.com/watch?v=2k8x8V77CrU>. [Último acceso: 18 Mayo 2021].
- [16] Kotlinlang, «Comparison to Java,» 2021. [En línea]. Available: <https://kotlinlang.org/docs/comparison-to-java.html>. [Último acceso: 19 May 2021].

- [17] Gradle, «What is Gradle,» [En línea]. Available: https://docs.gradle.org/current/userguide/what_is_gradle.html. [Último acceso: 2021 Junio 10].
- [18] Apache, «What's Maven,» [En línea]. Available: <https://maven.apache.org/what-is-maven.html>. [Último acceso: 10 Junio 2021].
- [19] Spring IO, «Introduction to Spring Framework,» [En línea]. Available: <https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/overview.html>. [Último acceso: 10 Junio 2021].
- [20] Spring IO, «JPA Repositories,» [En línea]. Available: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference>. [Último acceso: 10 Junio 2021].
- [21] Swagger.io, «What is Swagger?,» [En línea]. Available: <https://swagger.io/docs/specification/2-0/what-is-swagger/>. [Último acceso: 10 Junio 2021].
- [22] JSON, «Json,» [En línea]. Available: <https://www.json.org/json-es.html>. [Último acceso: 12 06 2021].
- [23] M. B, «Stackoverflow,» [En línea]. Available: <https://stackoverflow.com/questions/2673367/how-does-json-compare-to-xml-in-terms-of-file-size-and-serialisation-deserialisa>. [Último acceso: 2021 06 12].
- [24] Git, «git-scm.com,» [En línea]. Available: <https://git-scm.com/>. [Último acceso: 12 Junio 2021].
- [25] S. Judd, «Github readme,» [En línea]. Available: <https://github.com/bumptech/glide>. [Último acceso: 2021 Junio 10].
- [26] Square, «Retrofit,» [En línea]. Available: <https://square.github.io/retrofit/>. [Último acceso: 12 Junio 2021].
- [27] Square, «Github,» [En línea]. Available: <https://github.com/square/moshi>. [Último acceso: 2021 06 12].
- [28] Pusher, «What is Pusher?,» [En línea]. Available: <https://pusher-community.github.io/real-time-laravel/introduction/what-is-pusher.html>. [Último acceso: 10 Junio 2021].
- [29] Jetbrains, «jetbrains.com/idea,» Jetbrains, [En línea]. Available: <https://www.jetbrains.com/es-es/idea/>. [Último acceso: 12 Junio 2021].
- [30] Google I/O, «YouTube 23:11,» 15 Mayo 2013. [En línea]. Available: https://www.youtube.com/watch?v=9pmPa_KxsAM. [Último acceso: 12 Junio 2021].
- [31] Microsoft, «www.microsoft.com,» [En línea]. Available: <https://www.microsoft.com/es-es/microsoft-365/word?SilentAuth=1&wa=wsignin1.0>. [Último acceso: 12 Junio 2021].
- [32] yWorks, «yworks.com/products/yfiles,» [En línea]. Available: <https://www.yworks.com/products/yfiles>. [Último acceso: 12 Junio 2021].

- [33] K. E. Wiegers, *Software Requirements: Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle*, Microsoft Press, 2003.
- [34] U. A. Metropolitana, «<http://academicos.azc.uam.mx/>,» [En línea]. Available: http://academicos.azc.uam.mx/jfg/diapositivas/adsi/Unidad_8.pdf. [Último acceso: 24 Junio 2021].
- [35] A. Hooshangi, «Reducing Development Time Using Automated Data Transfer,» 2012. [En línea]. Available: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.224.4562&rep=rep1&type=pdf>. [Último acceso: 01 07 2021].
- [36] A. Freeman, «The Singleton Pattern,» 2014. [En línea]. Available: https://link.springer.com/chapter/10.1007/978-1-4842-0394-1_6. [Último acceso: 2021 Junio 13].
- [37] Android Developers, «Android 5 Lollipop,» 11 03 2021. [En línea]. Available: <https://developer.android.com/about/versions/lollipop>. [Último acceso: 19 Junio 2021].

16. Anexo

Aquí se muestra una ampliación del contenido anterior.

Ambos sistemas pueden encontrarse en los siguientes enlaces:

Servidor: <https://github.com/minimerlyn/NightTimeAPIRest.git>

Aplicación móvil: <https://github.com/minimerlyn/NightTimeTFG.git>

16.1 Extensión de la planificación del proyecto

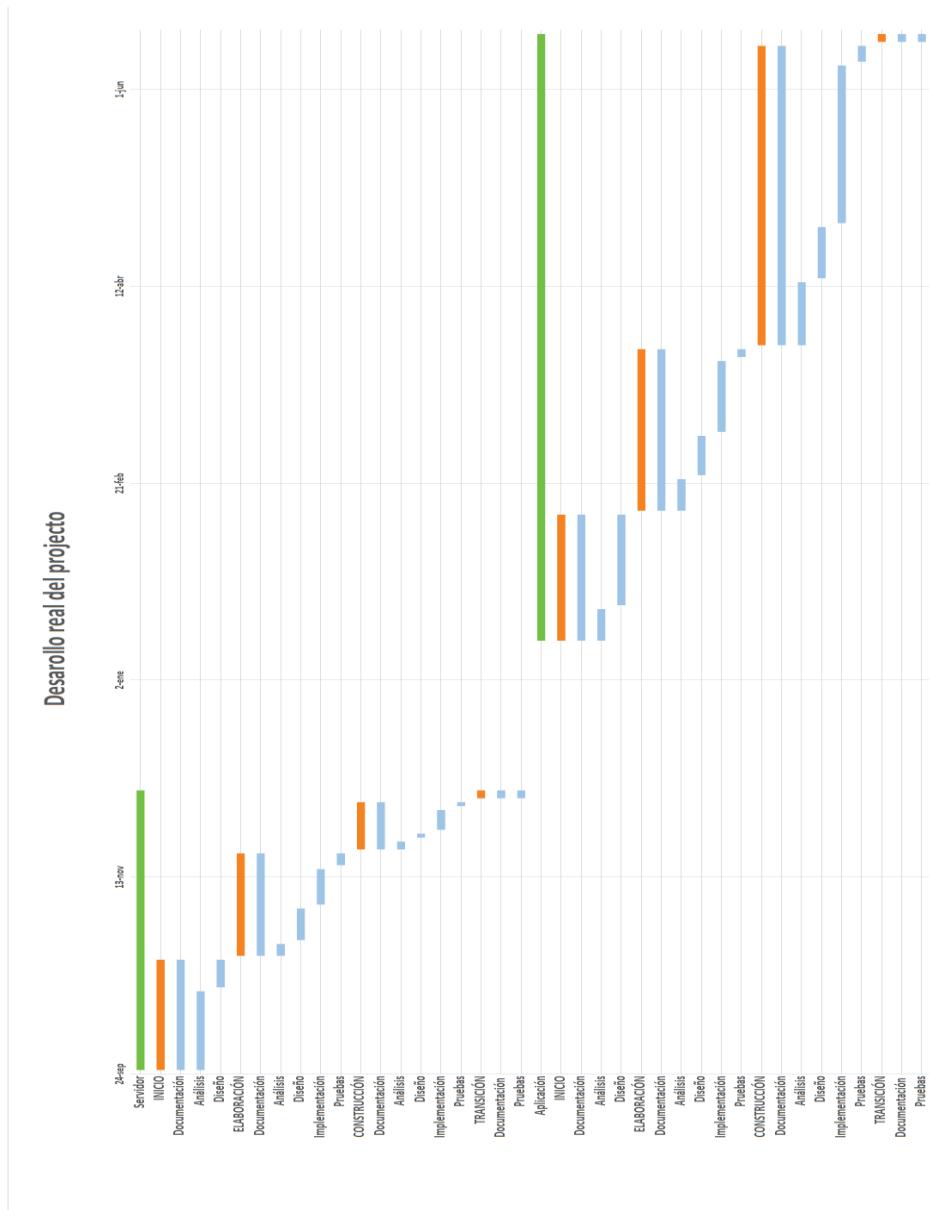


Ilustración 29: Diagrama de Gantt del desarrollo del proyecto

16.2 Extensión de los diagramas de clase del servidor

Este diagrama muestra el diagrama de clase del controlador de eventos. Este controlador se responsabiliza de la gestión de eventos de los bares.

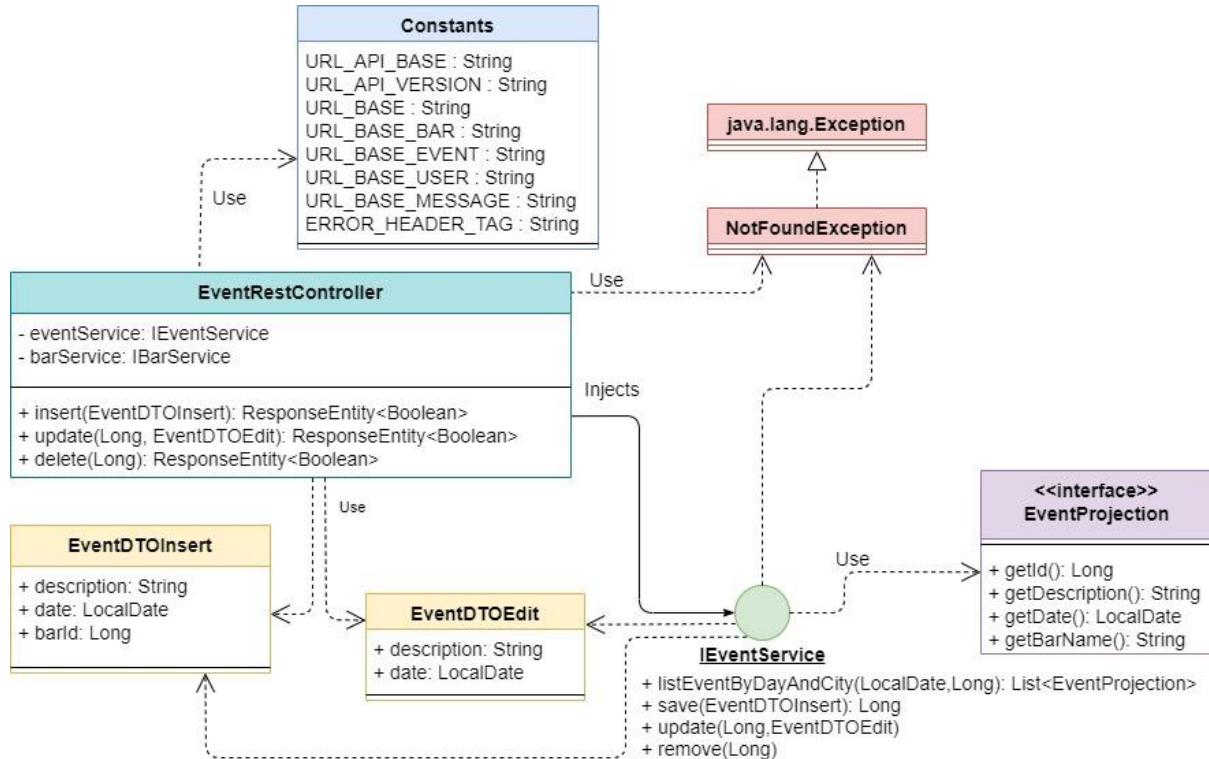


Ilustración 30: Diagrama de clase del controlador de eventos

A continuación, se muestra el diagrama de clase del controlador MessageRestController, el cual se encarga de permitir el envío de mensajes en tiempo real. Solo tiene un *endpoint*, el cual recibe un identificador con su token de seguridad y un DTO de un mensaje. Tras validar la identidad del usuario, el sistema guarda en la base de datos el nuevo mensaje recordando que aún no ha sido leído y se comunica con Pusher para mandar el usuario a su destino si este tiene la aplicación abierta.

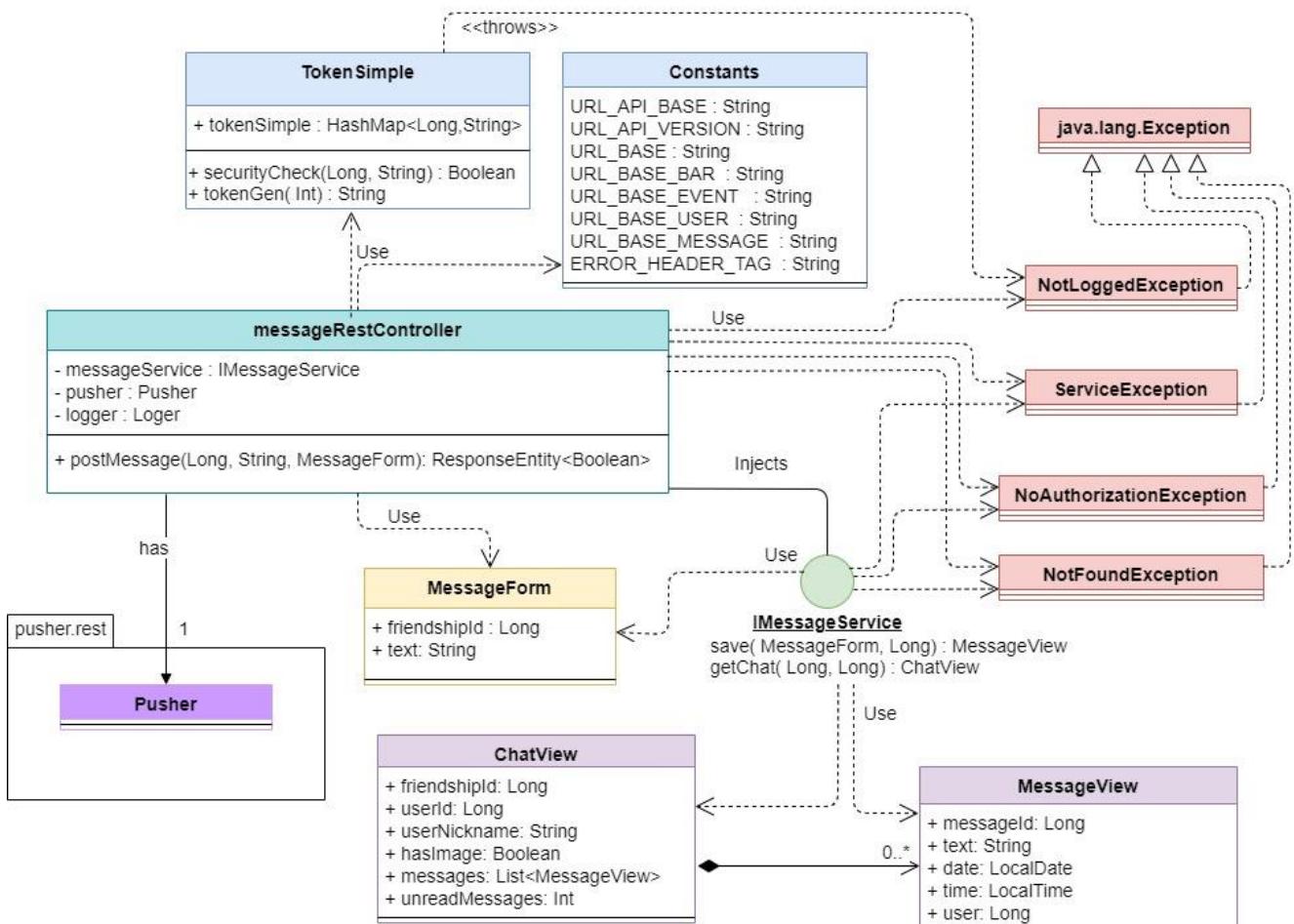


Ilustración 31: Diagrama de clase del controlador de mensajes

El ultimo controlador existente en el servidor se llama `UserRestController`, y es el controlador que mayoritariamente da cobertura a la aplicación móvil. Se encarga de permitir la gestión de usuarios y amistades, y las subclases derivadas, como pueden ser los días seleccionados por un usuario, o permitir ver el número de usuarios que va a salir un día determinado.

Ya que este controlador hace uso de 7 servicios del sistema, se omitirán las clases que no estén directamente relacionadas con `UserRestController`, a excepción de tres de las vistas de la derecha ya que completan el esquema de la respuesta del servidor.

NightTime: Desarrollo de una red social basada en la actividad nocturna

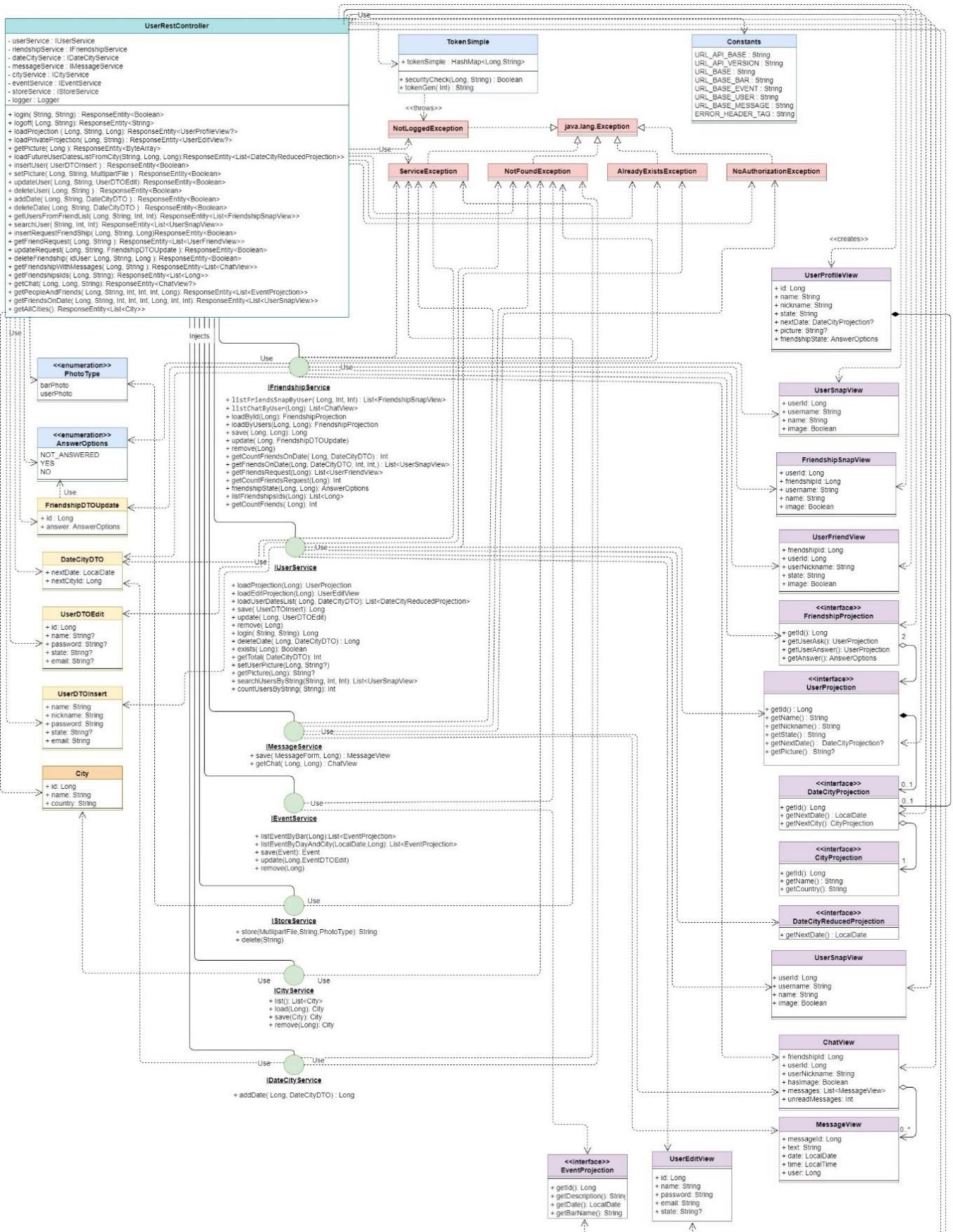


Ilustración 32: Diagrama de clase del controlador de usuarios

Una vez que hemos terminado de describir la primera capa del sistema, se describirán la segunda y la tercera juntas. Estas son los servicios y los repositorios. Se mostrarán siendo el inicio del esquema el servicio hasta mostrar todos o al menos las clases más relevantes que tengan interacción con los servicios.

El primero será el diagrama de clases del servicio de amistades. Este utiliza dos repositorios para realizar todas sus funciones, pues cuando realiza una nueva inserción crea los usuarios a partir de sus identificadores y después crea e inserta la relación. Cabe destacar que las relaciones de la clase Usuario han sido recortadas para minimizar la representación.

NightTime: Desarrollo de una red social basada en la actividad nocturna

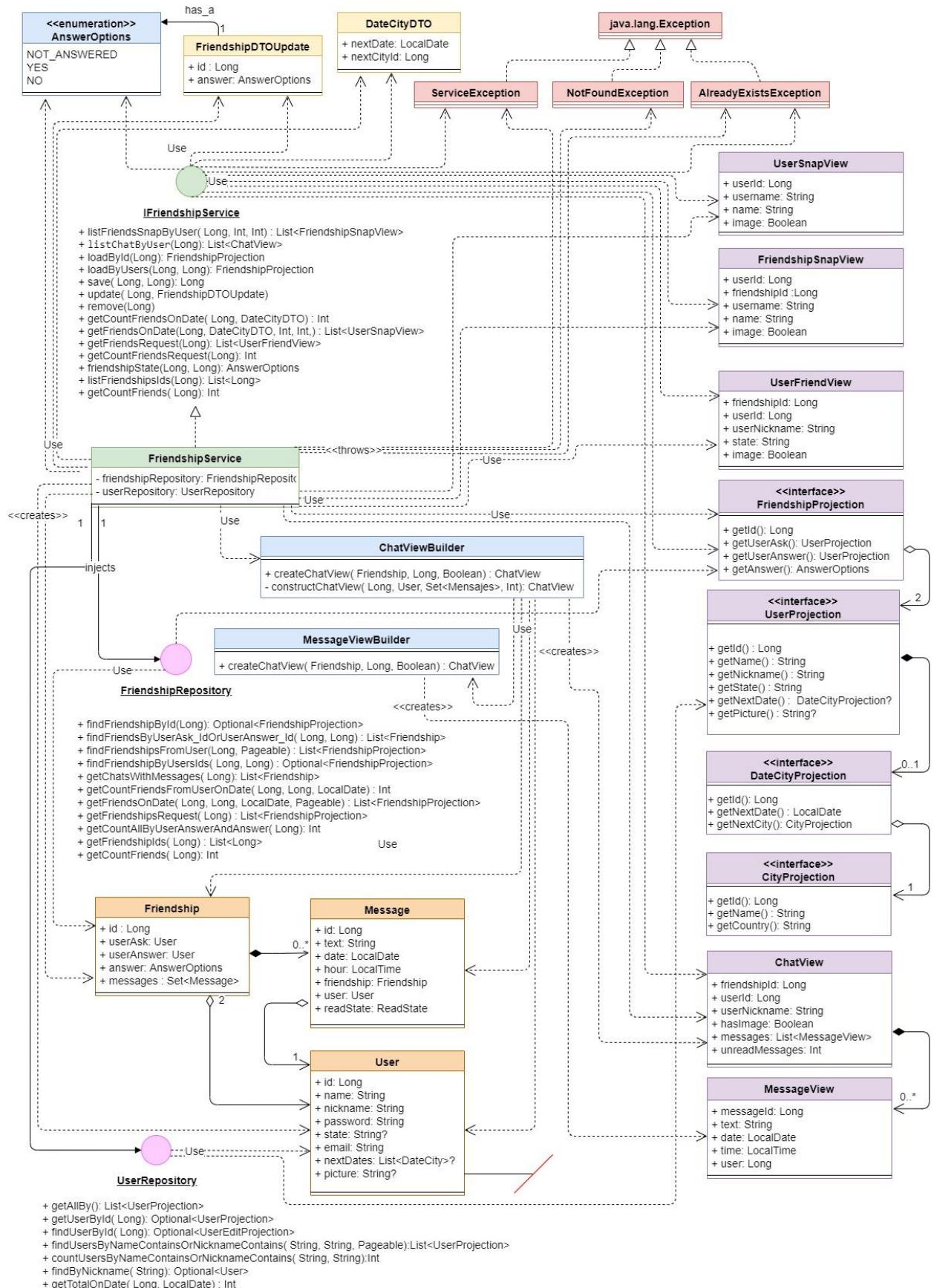


Ilustración 33: Diagrama de clase del servicio de amistades

El servicio de usuarios se encarga de la gestión de estos, pudiendo lanzar determinadas excepciones en situaciones especiales, como si las restricciones de tamaño del nombre de usuario no se cumplen, o si el identificador del usuario es inexistente.

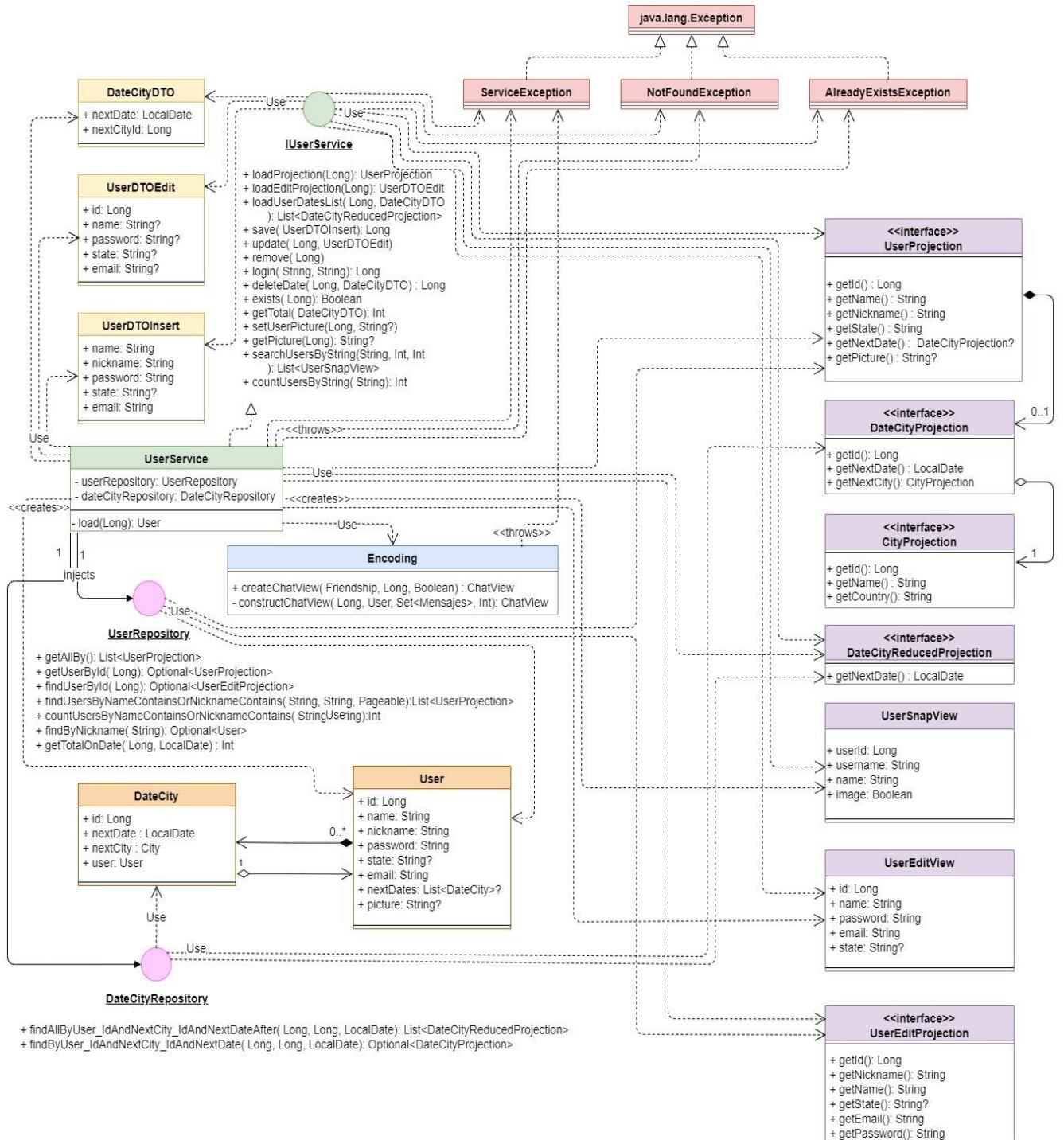


Ilustración 34: Diagrama de clase del servicio de usuarios

El servicio de gestión de ciudades es posiblemente uno de los más sencillos y se ha evitado el uso de DTOs y vistas para minimizar el tiempo de desarrollo. En cualquier caso, gracias al uso de una arquitectura definida se podría realizar cualquier modificación sin demasiado esfuerzo.

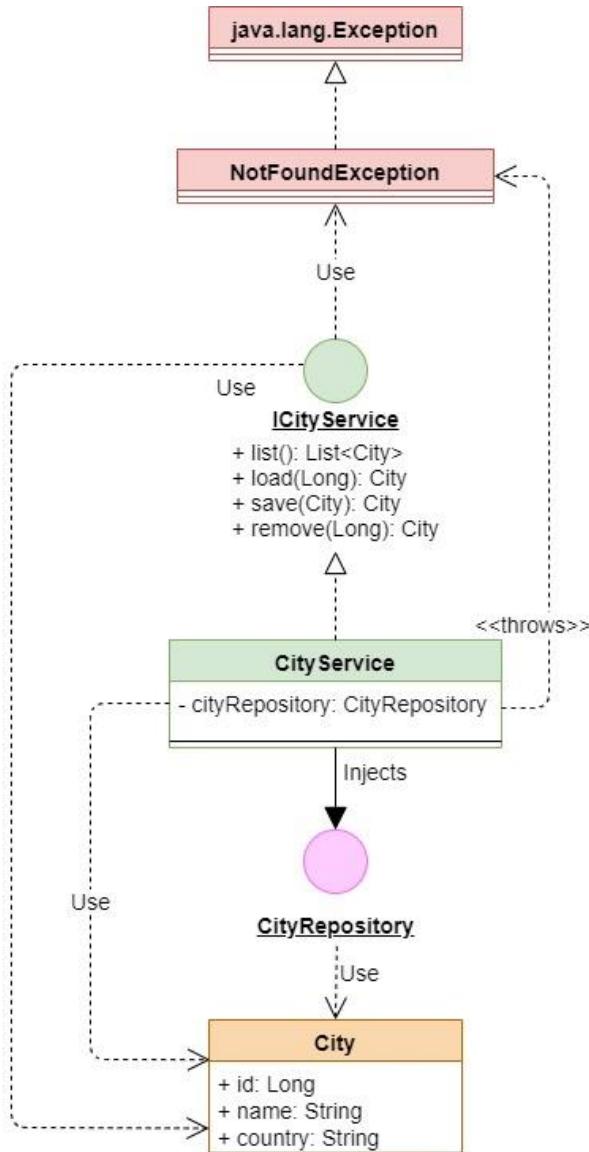


Ilustración 35: Diagrama de clase del servicio de ciudades

El siguiente diagrama representa el servicio encargado de almacenar los días que el usuario ha marcado para salir. Se han recortado las dependencias de DateCityRepository ya que no son utilizadas por este servicio, sino por el de usuarios.

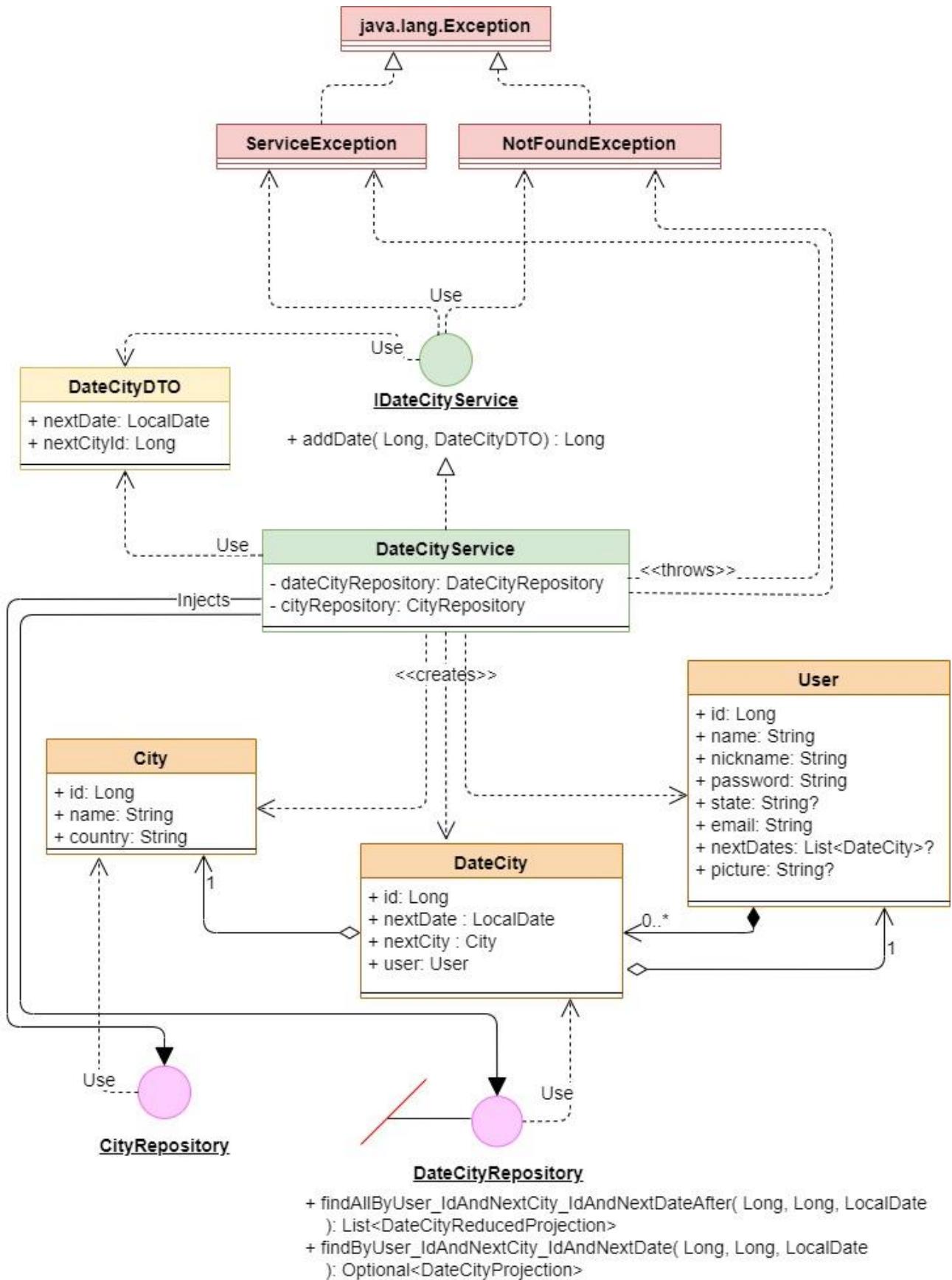


Ilustración 36: Diagrama de clase del servicio de fechas marcadas.

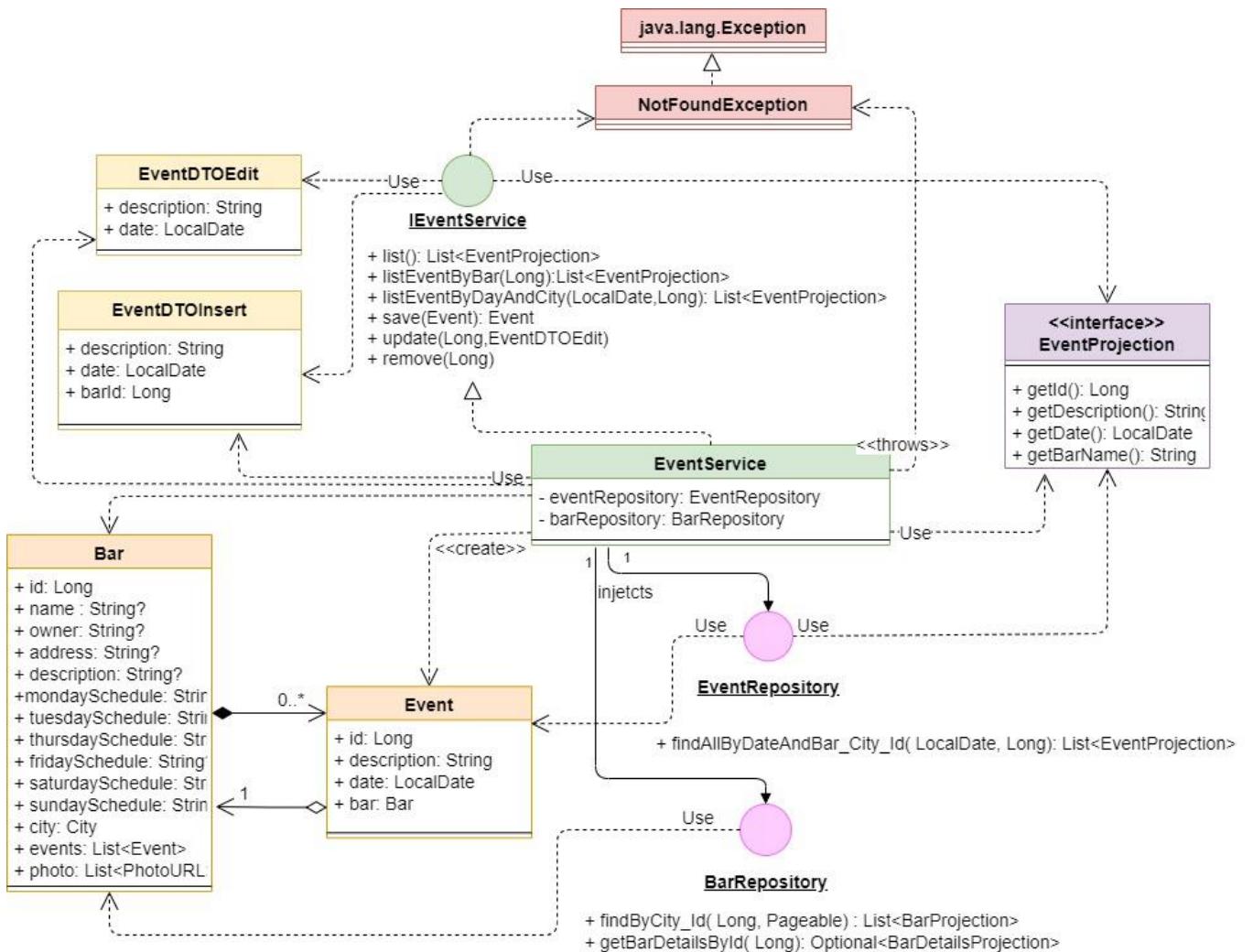


Ilustración 37: Diagrama de clase del servicio de eventos.

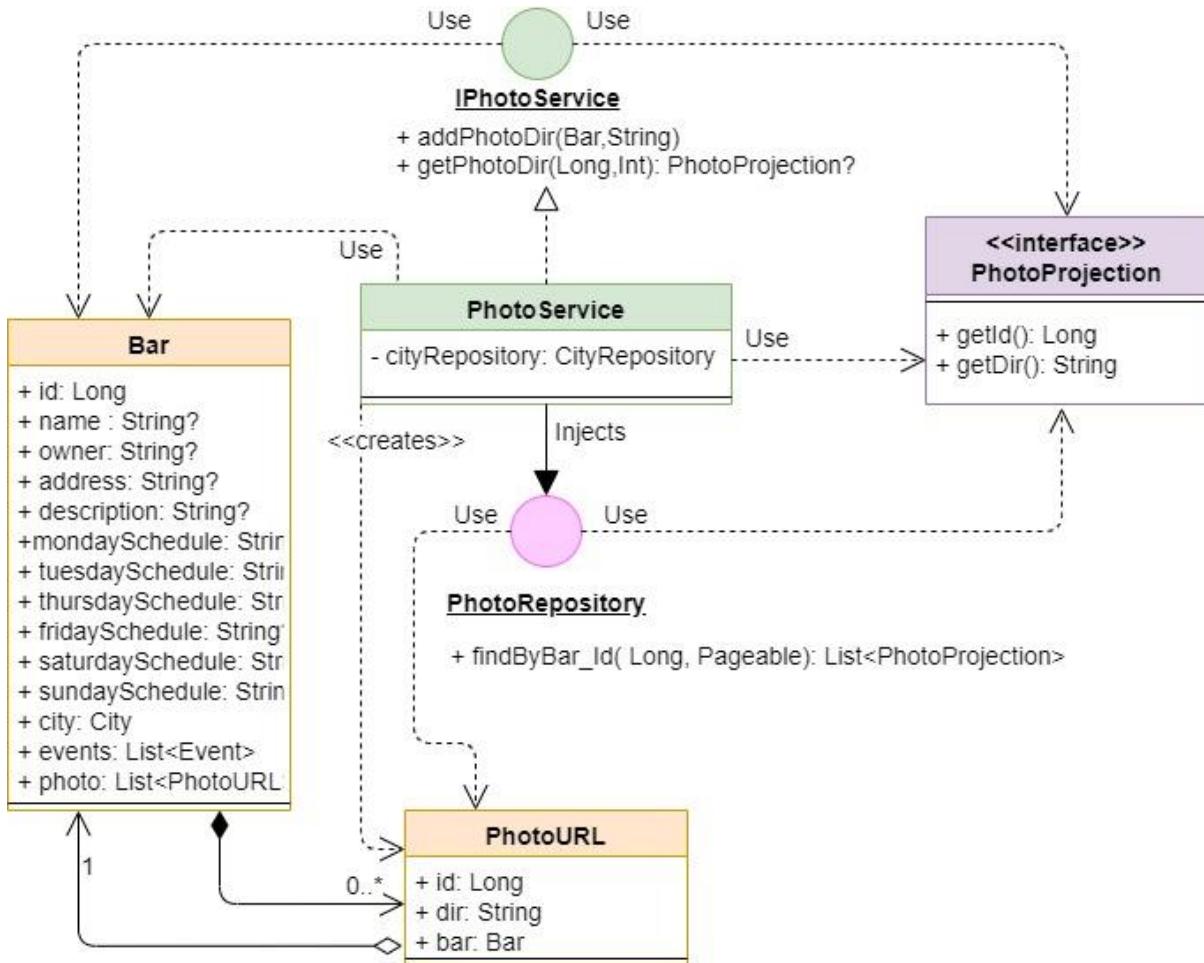


Ilustración 38: Diagrama de clase del servicio de fotos de los bares

El servicio que almacena imágenes en la memoria del servidor es llamado StoreService. Por este motivo no hace uso de repositorios, sino que utiliza librerías de java.nio para realizar sus operaciones, utilizando una clase enumerado para saber a qué directorio acceder.

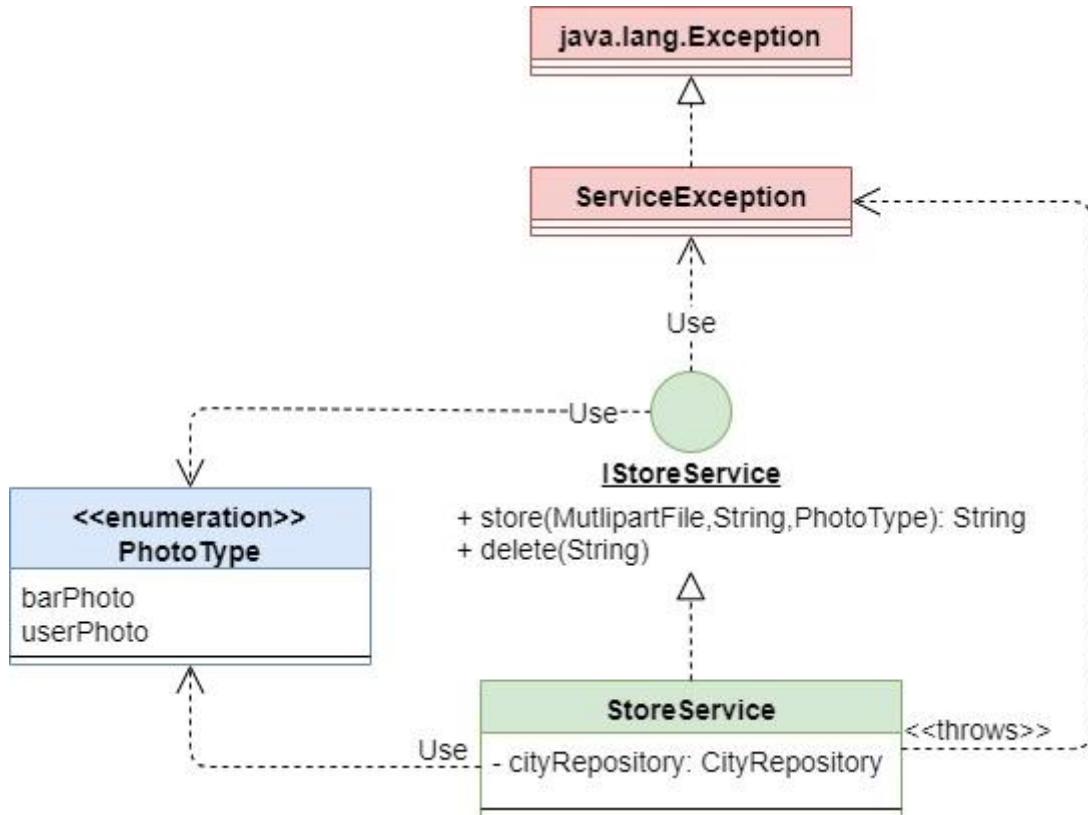


Ilustración 39: Diagrama de clase del servicio de almacenamiento de imágenes

El último servicio es el encargado de la gestión de los mensajes. En este esquema se recorta el repositorio de usuarios ya que muchas de sus relaciones no son utilizadas por este servicio. Cabe destacar el uso de dos clases constructores auxiliares, `ChatViewBuilder` y `MessageViewBuilder`. Estas clases utilizan los modelos para crear una vista personalizada de un chat.

NightTime: Desarrollo de una red social basada en la actividad nocturna

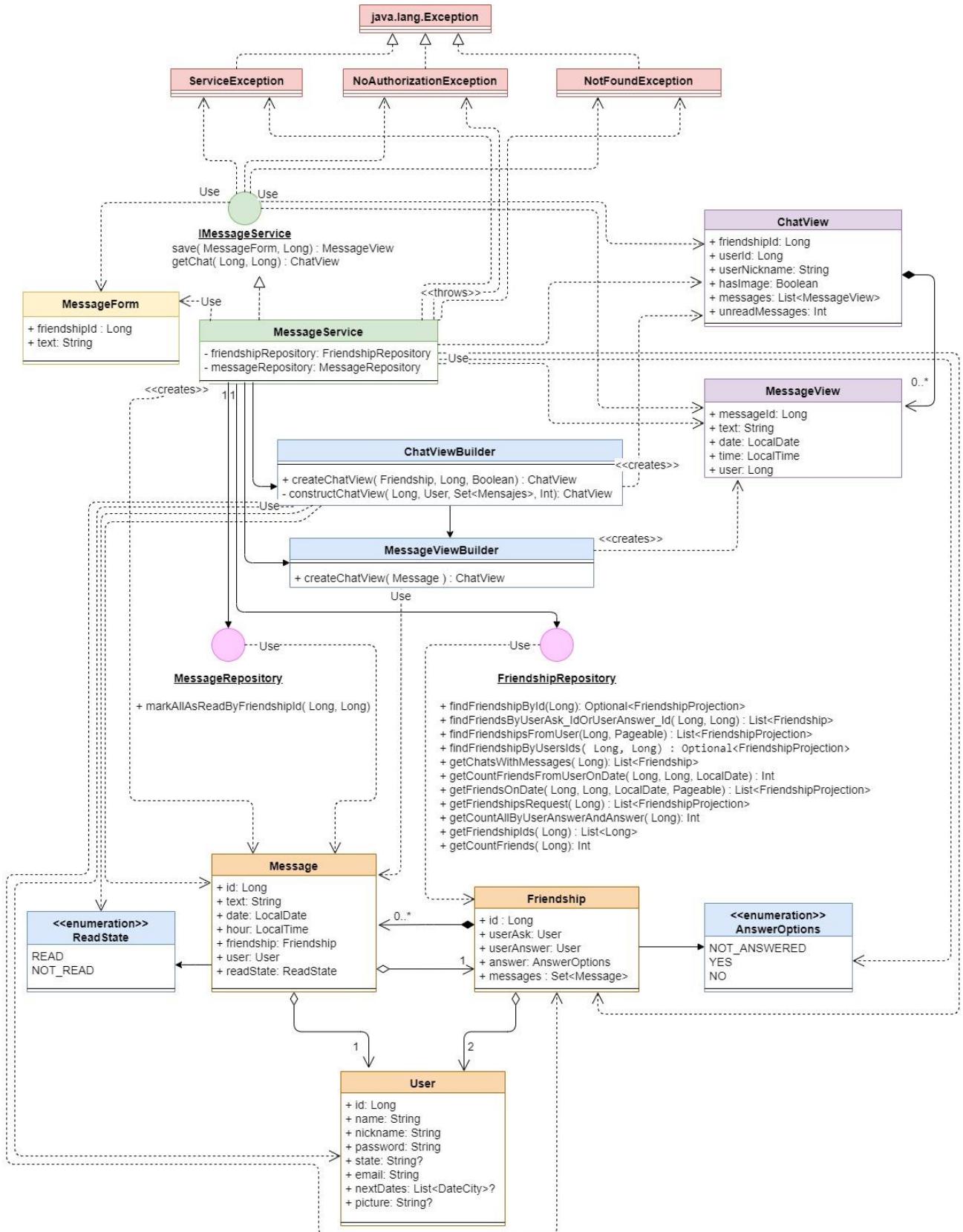


Ilustración 40: Diagrama de clase del servicio de mensajes

16.3 Extensión de los diagramas de secuencia de sistema del servidor

La ilustración 26 muestra el proceso para crear un nuevo usuario. El cliente manda los datos de registro al controlador, la contraseña se encripta antes de guardar en la base de datos utilizando el nombre de usuario como contraseña de la encriptación. Una vez guardado, se le adjudica un token automáticamente y se devuelve esta información.

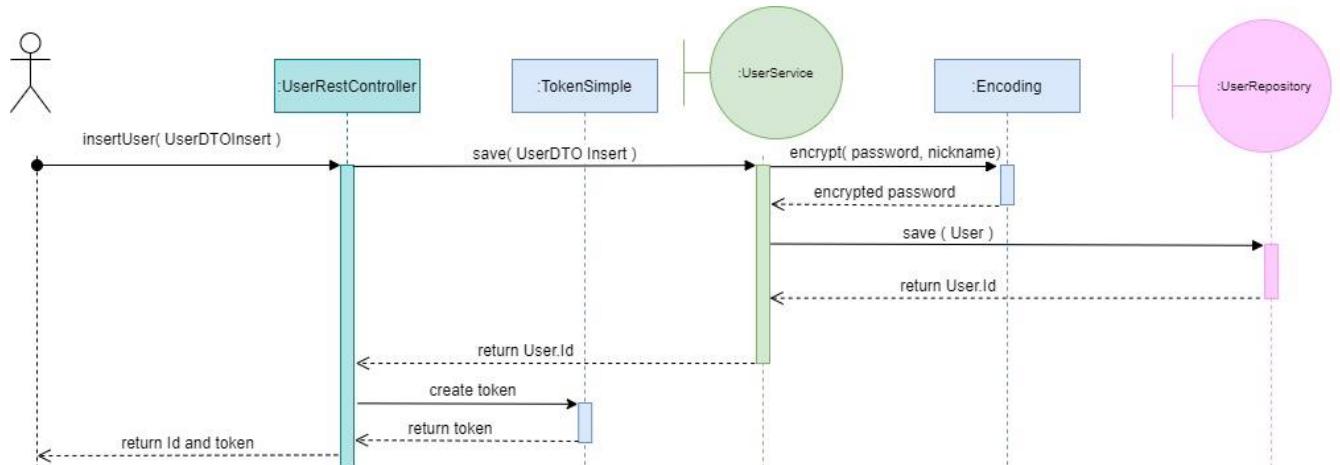


Ilustración 41: Diagrama de secuencia de sistema de crear nuevo usuario

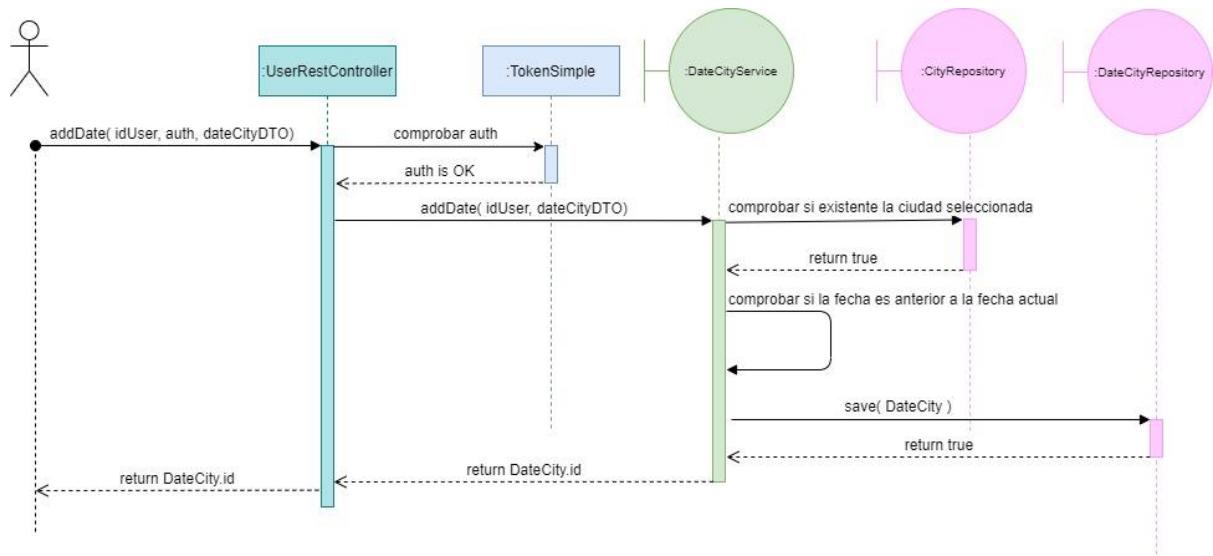


Ilustración 42: Diagrama de secuencia de sistema de seleccionar un día para salir

16.4 Extensión de la descripción de la gestión de la información.

Tabla *Message*:

CAMPO	TIPO DE DATO	DESCRIPCIÓN
Id	BigInt (20)	Identificador del mensaje.
Friendship_Id	BigInt (20)	Identificador de la amistad del mensaje.
User_Id	BigInt (20)	Identificador del usuario que mandó el mensaje.
Date	Date	Dia, mes y año en que el mensaje llegó al servidor.
Hour	Time	Hora y minuto en que el mensaje llegó al servidor.
Text	VarChar (255)	Texto del mensaje

Tabla *Next_date_city*:

CAMPO	TIPO DE DATO	DESCRIPCIÓN
Id	BigInt (20)	Identificador de la fecha marcada para salir de fiesta.
City_Id	BigInt (20)	Identificador de la ciudad seleccionada.
User_Id	BigInt (20)	Identificador del usuario que ha seleccionado dicha fecha.
NextDate	Date	Fecha seleccionada del calendario.

Tabla *City*:

CAMPO	TIPO DE DATO	DESCRIPCIÓN
Id	BigInt (20)	Identificador de la ciudad.
Country	VarChar (255)	Nombre del país al que pertenece.
Name	VarChar (255)	Nombre de la ciudad.

Tabla *Bar*:

CAMPO	TIPO DE DATO	DESCRIPCIÓN
Id	BigInt (20)	Identificador del bar.

City_Id	BigInt (20)	Identificador de la ciudad donde se encuentra el bar.
Address	VarChar (255)	Dirección del bar, se utilizará para abrir Google Maps.
Owner	VarChar (255)	Nombre del propietario del bar.
Description	VarChar (255)	Descripción del bar.
Name	VarChar (255)	Nombre del bar.
Monday_schedule	VarChar (255)	Horario del lunes, puede estar vacío para indicar que está cerrado.
Tuesday_schedule	VarChar (255)	Horario del martes, puede estar vacío para indicar que está cerrado.
Wednesday_schedule	VarChar (255)	Horario del miércoles, puede estar vacío para indicar que está cerrado.
Thursday_schedule	VarChar (255)	Horario del jueves, puede estar vacío para indicar que está cerrado.
Friday_schedule	VarChar (255)	Horario del viernes, puede estar vacío para indicar que está cerrado.
Saturday_schedule	VarChar (255)	Horario del sábado, puede estar vacío para indicar que está cerrado.
Sunday_schedule	VarChar (255)	Horario del domingo, puede estar vacío para indicar que está cerrado.

Tabla *PhotoURL*:

CAMPO	TIPO DE DATO	DESCRIPCIÓN
Id	BigInt (20)	Identificador de la foto del bar.
Bar_Id	BigInt (20)	Identificador del bar al que pertenece la foto.
Dir	VarChar (255)	Dirección de almacenamiento de la foto del bar

Tabla *Event*:

CAMPO	TIPO DE DATO	DESCRIPCIÓN
Id	BigInt (20)	Identificador del Evento del bar.
Bar_Id	BigInt (20)	Identificador del bar al que pertenece el evento.
Date	Date	Fecha en la que el evento está disponible.
Description	VarChar (255)	Descripción del evento.

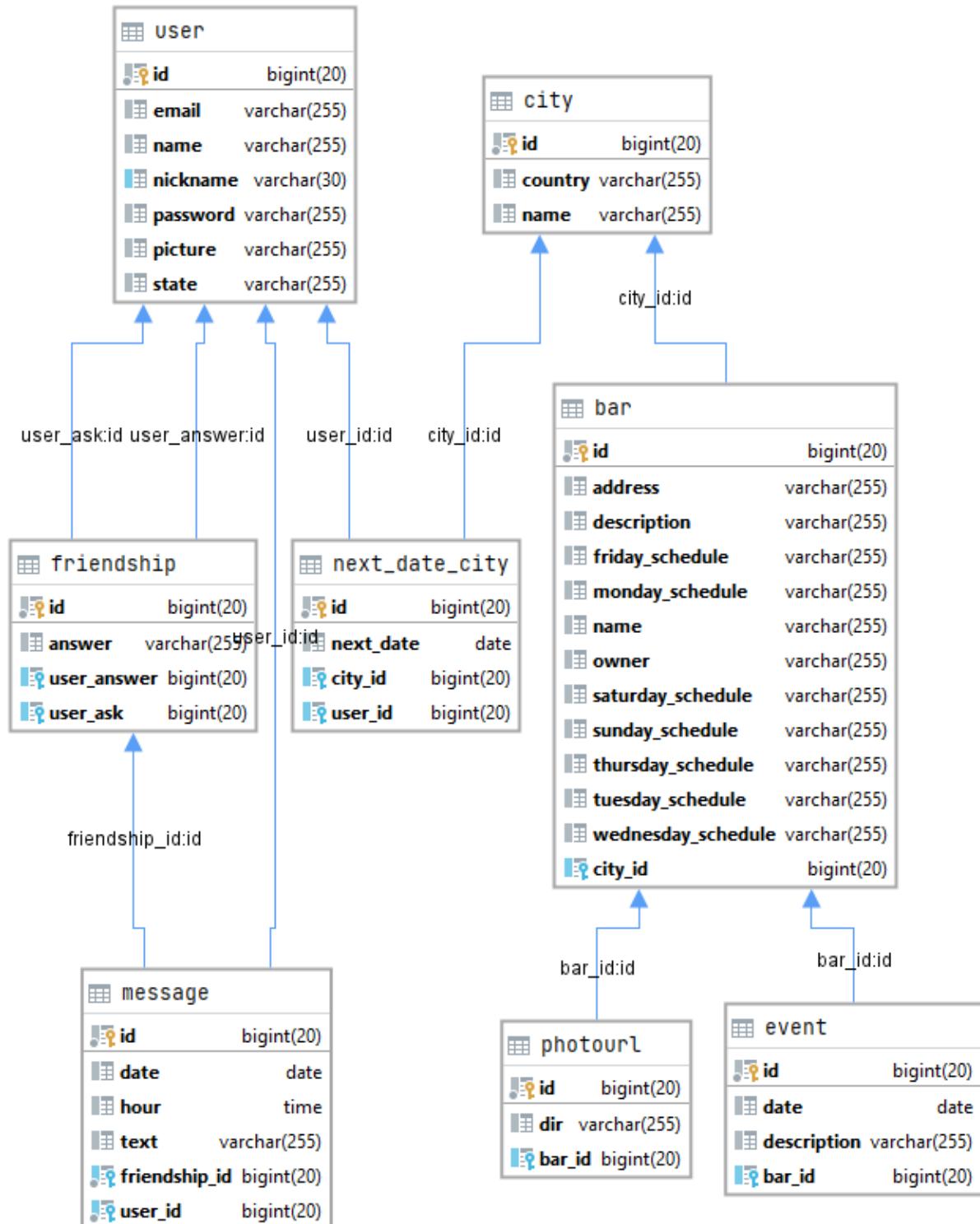


Ilustración 43: Relación y descripción de las tablas de la base de datos.