Elaborado por:
Equipo A - Clifford

Documento de Descripción de Arquitectura
Kwii Platform

Contenido

1 Introducción

1.1 Nombre del sistema de software

El nombre del sistema de software que está siendo desarrollado por el Equipo A - Clifford como proyecto final de la materia Arquitectura de Software para el periodo académico 2019-01 es:

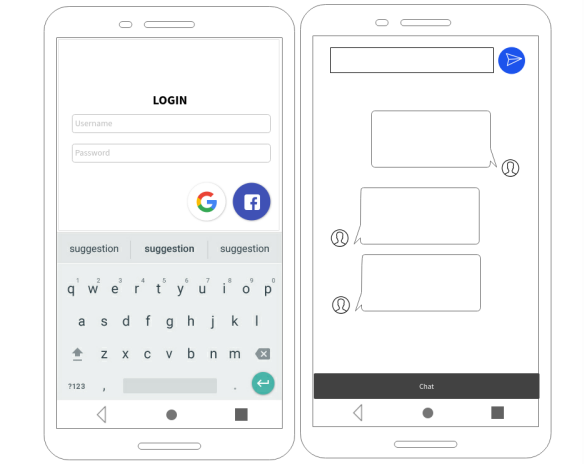
Kwii Platform

1.2 Descripción del sistema de software

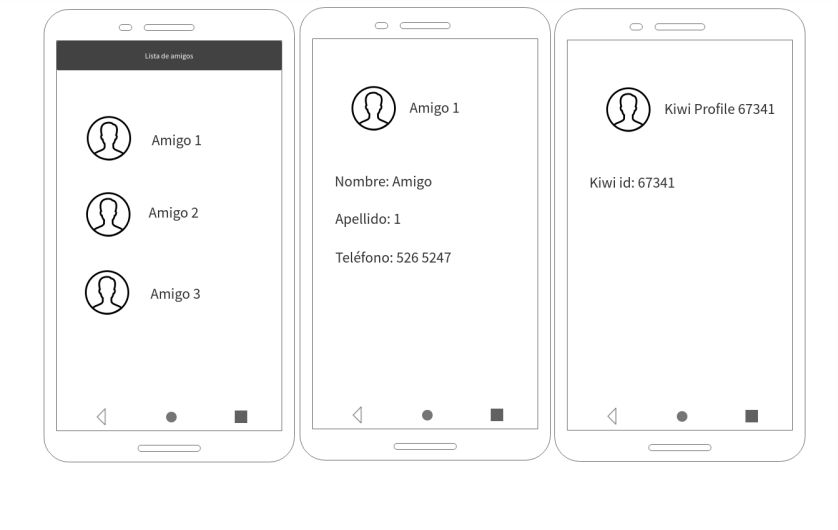
Kwii Platform es un sistema de software que permite a sus usuarios comunicarse por medio de chats. Dichos chats crearán puntos de encuentro entre pares de usuarios o entre grupos de tres o más usuarios, donde pueden enviar mensajes de texto. Para acceder a los chats, se cuenta con dos interfaces amigables a los usuarios (uno para web y otro para teléfonos inteligentes), que están encargadas de permitir a los usuarios utilizar el sistema. El sistema notifica a los usuarios sobre la interacción de otros usuarios con ellos en estos chats, o sobre información importante que la plataforma quiera comunicarles.

1.2.1 Descripción de idea del sistema de software para el curso de Arquitectura de Software

Los usuarios interactúan con el sistema a través de los Kwiiers (interfaces gráficas de usuarios de la Kwii Platform), que les permite personificar a un Kwii. Un Kwii es un ser capaz de comunicarse con otros Kwiis (otros usuarios que personifican un Kwii), ya sea con un único Kwii o con un grupo de tres o más Kwiis. Los usuarios personifican a un único Kwii, y un único Kwii identifica de manera única a un usuario. Esta identificación única de los Kwiis es lograda a través de un sistema de autenticación y administración de usuarios, que contempla la Kwii Platform.

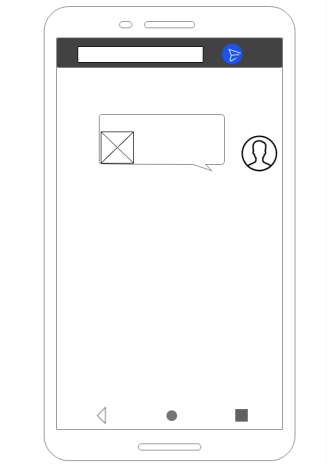


Las interacciones sociales de los Kwiis son bastante complejas, sus comunidades se construyen a partir de las amistades. Los Kwiis pueden construir sus propias listas de amigos en la Kwii Platform. Esta lista de amigos indica qué Kwiis considera un Kwii como su amigo. La amistad quiere decir, en la Kwii Platform, la capacidad de compartir información personal del Kwii (de su usuario asociado). Es así como se presentan dos tipos de perfiles, para todos los usuarios, uno que será compartido a nivel global, que indicará únicamente su identificador Kwii en la plataforma, y un usuario privado el cual podrá ser visto por los amigos del Kwii.

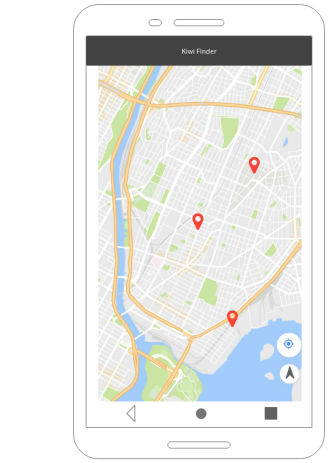


1.2.2 Descripción de ideas para el sistema de software para trabajo futuro

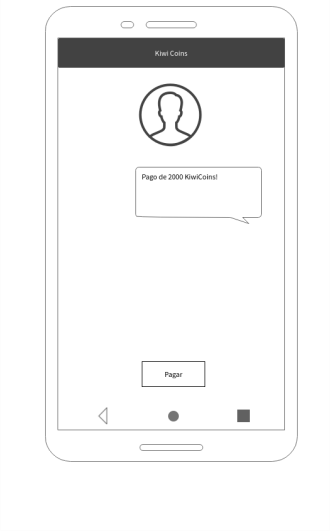
Los Kwiis son dotados de muy buena memoria, siendo capaces de recordar conversaciones que tuvieron con otros Kwiis hasta 5 años, sin embargo, también son muy buenos guardando secretos, ser grandes confidentes es la base de su compleja sociedad, así que si un amigo Kwii pide que olviden lo que le van a contar en una conversación en 5 minutos exactos, sus amigos Kwiis van a borrar toda evidencia de este evento pasados los 5 minutos exactos. En los chats individuales y grupales, los usuarios tienen la posibilidad de enviar mensajes con tiempo de expiración, y de eliminar, antes de pasados 10 minutos, los mensajes enviados sin expiración. Por defecto, todos los mensaje tienen una vigencia de 5 años.



Los Kwiis son seres muy sociales, les gusta reunirse y no solo hablar desde la distancia, es por ello que los Kwiiers permiten a los Kwiis acercarse y encontrarse en cualquier lugar del mapa. Los usuarios pueden compartir en tiempo real su ubicación. Además, habilitada la opción de ”encontrarse”, los usuarios pueden recibir notificaciones sobre otros Kwiis cerca.



Además, los Kwiis tienen sistemas económicos complejos, es por esto que los Kwiiers permiten a los Kwiis hacer transacciones en KwiiCoins, la moneda principal de los Kwiis. Los usuarios, a través de la aplicación, podrán hacer transacciones en la moneda virtual KwiiCoin que será manejada por la plataforma. Un KwiiCoin equivale a un Peso Colombiano. Dos mil (2.000) KwiiCoins equivalen a una piragua en el Comedor Central de la Universidad Nacional de Colombia.



La compleja sociedad de los Kwiis ha construido en la Kwii Platform a los KwiiBots, Bots creados por otros Kwiis que les facilitan la vida. Ellos son capaces de hacer muy buenas conversaciones, se comportan como cualquier otro Kwii, pero un Kwii puede ser dueño de un único KwiiBot. Los usuarios dentro de la Kwii Platform pueden crear y administrar un único bot dentro de la aplicación. Dichos bots se comportarán como otros usuarios teniendo perfiles, pero sus perfiles son únicamente públicos.

En las sombras de la sociedad de los Kwiis se esconden los KwiiAdmins, que son entidades grandiosas que velan por la seguridad, expansión y continua mejora de la Kwii Platform. Aunque para los demás Kwiis ellos parecen Kwiis normales y corrientes, éstos Kwiis tienen acceso al sistema Kwii Platform en su totalidad. En el mundo de los humanos, los KwiiAdmins se les llaman desarrolladores.

1.3 Ingeniería de requisitos

En el estudio de ingeniería de requisitos, se encuentran las diversas historias de usuario, y dichas historias son desglosadas, exponiendo sus respectivos casos en el diagrama de casos de uso, además son consolidadas en una tabla de casos de uso y también son presentadas al detalle en las tablas de descripción de casos de uso.

Finalmente es mostrado un conjunto de requisitos no funcionales, que garantizarán la calidad de la Kwii Platform.

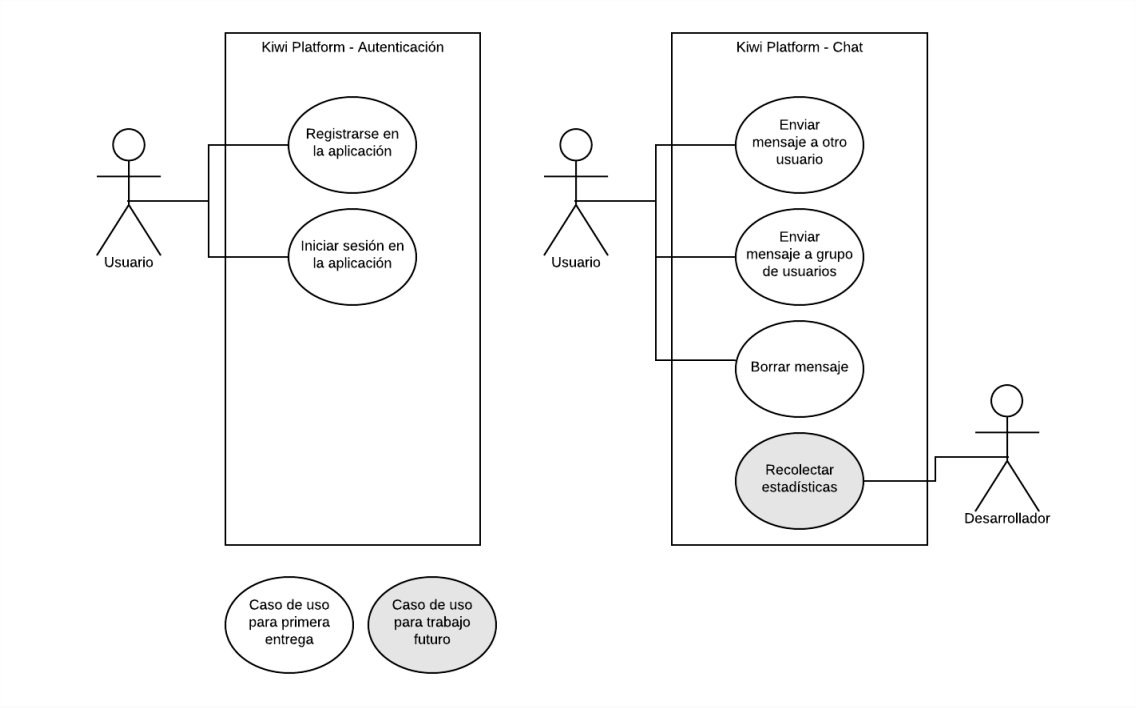
1.3.1 Identificación de los concerns y sus respectivos stakeholders

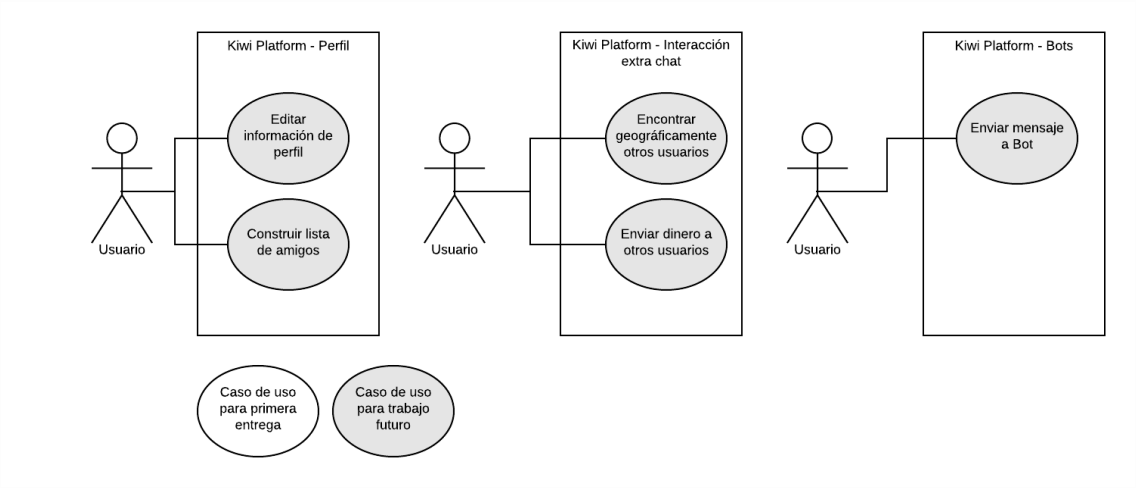
Para exponer los stakeholders y sus respectivos concerns, se exponene las siguientes historias de usuario:

- Usuarios de la aplicación (Kwiis)
  - Como Kwii, yo debo poder registrarme en la aplicación, con el objetivo de poder volver a utilizar la aplicación desde otro dispositivo.
  - Por supuesto, como Kwii, también debo poder iniciar sesión en la aplicación. Así podré recuperar mis datos cuando, por ejemplo, pierda mi celular.
  - Es importante que con la aplicación pueda entrar en contacto con otras personas (Kwiis), es por eso que debería poder enviar mensajes en la aplicación.
  - Para mí como usuario de la aplicación, es menester enviar mensajes no solo a otro usuario, sino a varios, para así fácilmente compartir mis ideas a varios otros usuarios a la vez.
  - Es importante que la aplicación me notifique cuando otro usuario me escriba, para así, si tengo la posibilidad, poderle contestar de manera rápída.
  - Como Kwii, usuario de la aplicación, debo poder hacer uso de los servicios de la aplicación desde mi computador o mi celular (inclusive desde ambos a la vez).
  - A veces, como usuario, puedo ser bastante descuidado con los mensajes que envío, es por eso que me encantaría poder borrar mensajes enviados por error o descuido.
  - Me gustaría que los demás usuarios puedan ver mi foto de perfil, para así poder ser identificado más fácil entre los otros usuarios.
  - Algo de infromación personal como perfiles, también estaría bien, para poder personalizar mi cuenta.
  - A veces, me es difícil encontrar personas cercanas con las que pueda compartir sin moverme demasiado de mi lugar de trabajo, mi casa, o donde simplemente esté en un momento dado, es por eso que me gustaría poder encontrar usuarios cercanos que estén usando la aplicación en el momento en el que yo la esté usando.
  - También me gustaría poder pagar mis deudas utilizando una aplicación de pagos, para así no tener que usar dinero en efectivo o sencillamente no tener que retirar en un banco.
  - Sería encantador poder hablar con algún tipo de bot o máquina dentro de la misma aplicación.
- Equipo de desarrollo (KwiiAdmins)
  - Para nosotros como equipo de desarrollo podría ser interesante extraer estadísticas de la aplicación, con el fin de observar el comportamiento de los usuarios y analizar dónde puede mejorar la aplicación.

1.3.2 Especificación de los requisitos funcionales

Se presentan a continuación los diagramas de casos de uso que resumen las historias de casos de uso presentadas anteriormente. En el diagrama, los casos de uso que no hacen parte de la implementación de la primera entrega son señalados.





Ahora, mostramos la tabla de casos de uso, para ellos, a cada uno le asignamos un identificador, su nombre y actor, además de lo que es considerado la ”complejidad” de implementar para el equipo de desarrollo, la ”prioridad” de implementar la funcionalidad y finalmente, si esta para esta funcionalidad, es implementada su lógica en la primer entrega, o si es considerada trabajo futuro.

ID	Nombre	Actor	Complejidad	Prioridad	Primera entrega
1	Registrarse en la aplicación	Usuario	Media	Alta	Sí
2	Iniciar sesión	Usuario	Media	Alta	Sí
3	Enviar mensaje a otro usuario	Usuario	Alta	Alta	Sí
4	Enviar mensaje a grupo de usuarios	Usuario	Muy alta	Alta	Sí
5	Recibir notificaciones de la aplicación	Usuario	Alta	Alta	Sí
6	Borrar mensajes	Usuario	Alta	Media	No
7	Editar información de mi perfil	Usuario	Media	Media	No
8	Construir lista de amigos	Usuario	Media	Baja	No
9	Encontrar usuarios geográficamente	Usuario	Muy alta	Muy baja	No
10	Enviar dinero a otros usuarios	Usuario	Muy alta	Muy baja	No
11	Enviar mensajes a bot	Usuario	Muy alta	Baja	No
12	Recolectar estadísitcas sobre el uso de la aplicación	Desarrollador	Media	Media	No

Ahora, presentaremos los casos de uso de forma más amplia, escribiendo la descripción de estos.

ID	1
Nombre	Registrarse en la aplicación
Descripción	Un usuario puede entrar a la aplicación y crear un perfil, lo que significa poder registrarse dentro de la plataforma para así poder llevar registro de las conversaciones que ha tenido y demás información importante dentro de su perfil

ID	2
Nombre	Iniciar sesión dentro de la aplicación
Descripción	Un usuario puede entrar a la aplicación con su usuario y contraseña si son correctos

ID	3
Nombre	Enviar mensaje a otro usuario
Descripción	Un usuario puede enviarle un mensaje de manera privada a otro usuario que esté registrado en la aplicación

ID	4
Nombre	Enviar mensaje a grupo de usuarios
Descripción	Un usuario puede enviarle un mensaje de manera privada a un grupo de usuarios registrado en la aplicación

ID	5
Nombre	Recibir notificaciones de la aplicación
Descripción	Un usuario puede recibir notificaciones en el cliente donde esté usando la aplicación

ID	6
Nombre	Borrar mensajes
Descripción	Un usuario puede borrar mensajes enviados por él y que ya no queden registrados en los historiales

ID	7
Nombre	Editar información de mi perfil
Descripción	Un usuario puede editar y actualizar la información relacionada a su cuenta

ID	8
Nombre	Construir lista de amigos
Descripción	Un usuario puede manejar una lista de amigos (usuarios registrados en la aplicación)

ID	9
Nombre	Encontrar usuarios geográficament
Descripción	Un usuario puede encontrar a otros usuarios por medio de la geolocalización

ID	10
Nombre	Enviar dinero a otros usuarios
Descripción	Un usuario puede enviar dinero a otros usuarios por medio de una cartera electrónica

ID	11
Nombre	Enviar mensajes a un bot
Descripción	Un usuario puede tener conversaciones con bots creados por otros usuarios

ID	12
Nombre	Recolectar estadísticas sobre el uso de la aplicación
Descripción	Los desarrolladores podrán tener la capacidad de recolectar información del uso de la aplicación y generar estadísticas

1.3.3 Especificación de los requisitos no funcionales

Para garantizar la calidad de la Kwii Platform, son listadas los siguientes requisitos no funcionales junto por qué es considerado un requisito para el sistema.

- El sistema debe está implementado haciendo uso de una arquitectura de microservicios para facilitar su escalabilidad e interoperabilidad.
- La funcionalidad de la plataforma se distribuye en 5 microservicios los cuales funcionan cada uno con una tecnologia diferente, estos son:
  - Componente Usuarios
    Lenguaje usado: Ruby
    Framework usado: Ruby on Rails
    Base de Datos: PosgreSQL
  - Componente Autenticación
    Lenguaje usado: Javascript
    Framework usado: Express
    Base de Datos: Redis
  - Componente Chat
    Lenguaje usado: Python
    Framework usado: Django
    Base de Datos: MongoDB

- Componente ChatRoom
  Lenguaje usado: Java
  Framework usado: Maven
  Base de Datos: MySQL
- Componente Notificaciones
  Lenguaje usado: Go
  Framework usado: Rush
  Base de Datos: Redis

2 Estilos y Patrones Arquitectónicos

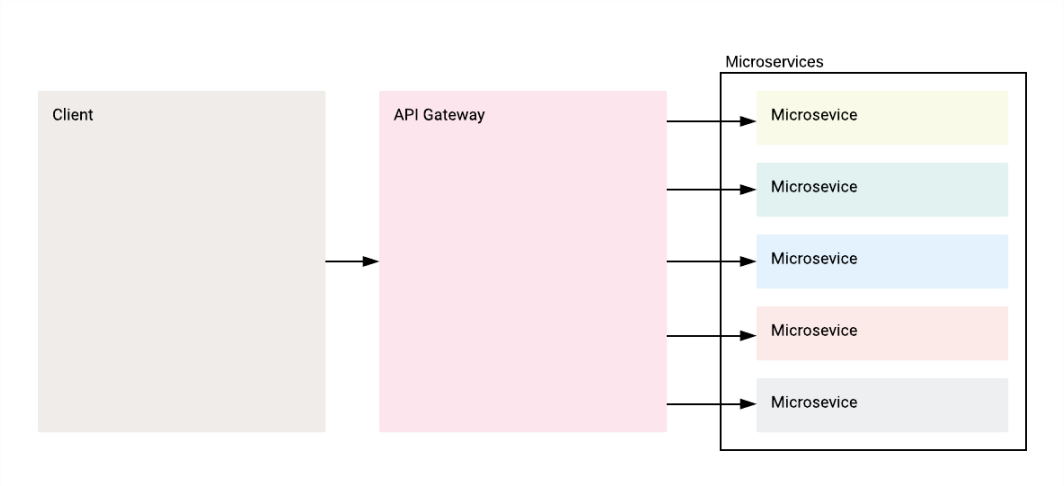
2.1 Estilos arquitectónicos

Para dar solución a los requerimientos, Kwii Platform adopta un estilo arquitectonico basado en microservicios, teniendo como componentes encargados de la presentación de nuestros servicios y el manejo de los usuarios, los clientes movil y web, además de una familia de microservicios construida especificamente para el manejo de la lógica de Kwii Platform, la conexion de estos familia microservicios es posible a traves del uso del conector REST.

2.1.1 Uso del estilo de microservicios

Consecuentemente, Kwii Platform aplica un Patron de microservicios estructurado de la siguiente manera:

5 componentes que encapsulan funcionalidades concretas de la aplicación (administracion de usuarios, sesion, salas de chat, mensajes y un componente de notificación), adicionalmente se tiene un componente adicional encargado de sincronizar los demas ademas de exponer la funcionalidad completa de la aplicación a los clientes.



2.1.2 Uso del estilo REST

REST se ha consolidado como el estilo más adecuado cuando el protocolo de comunicacion es HTTP para obtener datos y aún más si la representación de estos datos es en XML o JSON, siendo esta última la representación elegida para las entidades de esta aplicación.

2.2 Patrones arquitectónicos

Cada uno de los microservicios a su vez utiliza un patron arquitectonico para poder cumplir con su funcion especifica, el patron usado por cada uno de ellos se describe a continuación:

2.2.1 MVC (Modelo Vista Controlador)

Este modelo se utiliza en cuatro de los cinco microservicios (usuarios, autenticación y notificaciones y chat-room), estos servicios manejarán las entidades a través de controladores, los cuales podrán renderizar respuestas a peticiones HTTP en formato JSON, estas representaciones actuarán como vistas.

2.2.2 MTV (Model Template View)

En el caso del microservicio de chat este modelo no solo resulta ser el más adecuado para el framework en el que se desarrolla (Django), sino que accede a la base de datos a través del modelo, y en lugar de utilizar una vista para la capa de presentación, utiliza una plantilla. que permite determinar con mayor facilidad que cosas se muestran en las vistas, que en este caso hacen el rol de describir la lógica del negocio. Esto último resulta ser útil para el motor de bases de datos noSql que se usa en este caso (MongoDB).

3 Vistas Arquitectónicas

A continuación se muestran las diferentes vistas arquitectónicas que plasman los diagramas la Kiwi Platform.

3.1 Vista de Descomposición

La vista de descomposición encierra y describe toda nuestra aplicación en los siguientes módulos principales:

- api-gateway:

  Este módulo es el responsable de orquestar los flujos de procesos de la plataforma. En él se encuentra la lógica del manejo de peticiones, redirigiendo el trabajo de todos los microservicios de la aplicación.

- usuarios

  El módulo de usuarios es el encargado de llevar la lógica de los usuarios, éste a su vez es dividido en los siguientes dos submódulos:

  - user-ms

    El microservicio de los usuarios, módulo encargado de almacenar y exponer la información de los usuarios. Este submódulo permite la administración de los perfiles de usuario ademas de relacionarlos entre ellos.

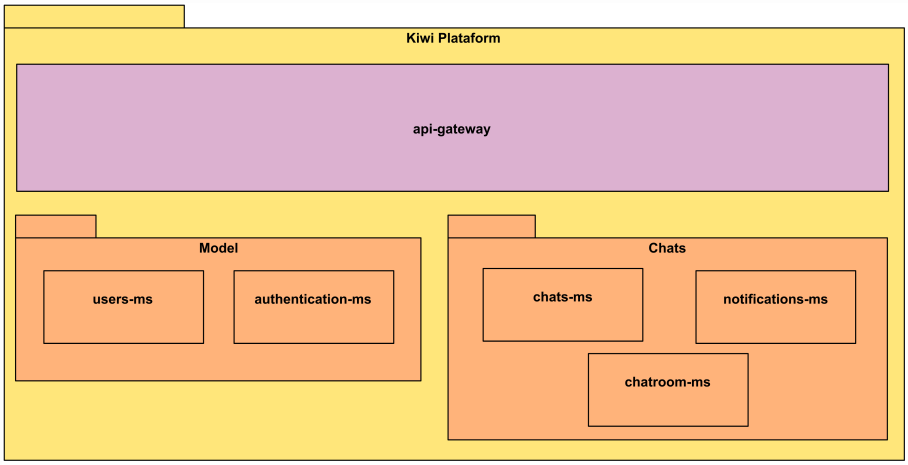
  - authentication-ms

    El microservicio de autenticación, permite a los usuarios tener sesiones dentro de la aplicación. Este microservicio es el principal para cuestiones de seguridad en la sesión de los usuarios en el sistema.

- chats

  El módulo de chats es el gran módulo que vela por la creación de chats y la entrega de mensajes entre ellos, además también se encarga de notificar a los usuarios sobre el uso de estos chats. Este gran módulo se divide en los tres siguientes submódulos:

  - chat-ms
    El submódulo chat-ms es el encargado de almacenar y exponer información sobre los mensajes enviados en los chats de la aplicación.
  - chatroom-ms
    El submódulo chatroom-ms es el encargado de la gestión de los chats, los chats son los lugares donde se intercambian mensajes. Estos chats pueden ser consumidos por los usuarios asociados a esos chats.
  - notifications-ms El submódulo encargado de la entrega de notificaciones a los usuarios es notification-ms, él vela por informar a los usuarios de la aplicación sobre el intercambio de mensajes de la aplicación.

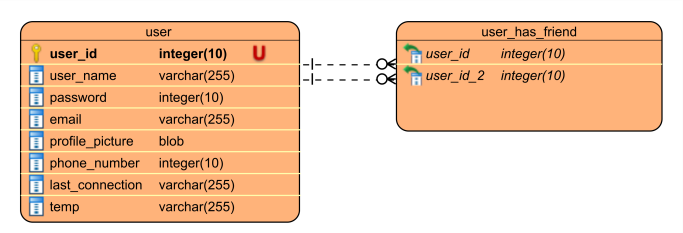


3.2 Vista de Modelos de Datos

Las vistas de los modelos de datos nos muestra la aplicación desde la perspectiva de sus datos, y cómo residen ellos en nuestra aplicación. Se evidencia en los diagramas cómo cada microservicio de la aplicación es dueño de sus datos y los manipula y expone a su conveniencia.

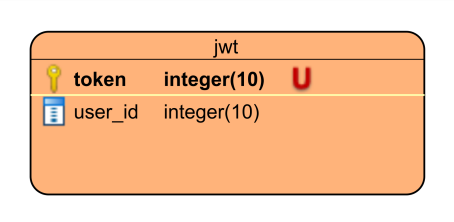
3.2.1 Microservicio usuarios

En el modelo de datos del microservicio usuarios encontramos una estuctura en la que se guardan cosas relevantes referentes a ellos, tales como el nombre o su contraseña secreta. Esta estructura puede ser ampliada cuando sea necesario, para almacenar más información acerca de los usuarios.
Además se presenta una relación de muchos a muchos de los usuarios con ellos mismos, para trazar la información de los amigos de un usuario; de esta forma, un usuario puede tener muchos amigos, y muchos amigos (otros usuarios) pueden ser amigos de un mismo usuario.



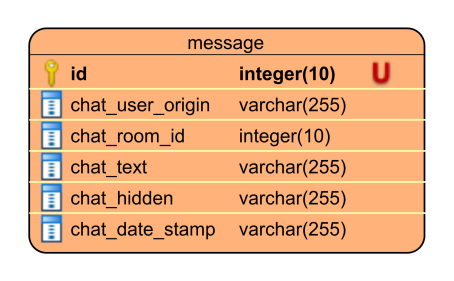
3.2.2 Microservicio autenticación

En esta vista, mostramos que el microservicio de autenticación almacena y manipula información sobre los JSON Web Token, estos JSON en realidad contienen bastante información sobre la sesión de usuario que es útil para la Kwii Platform.



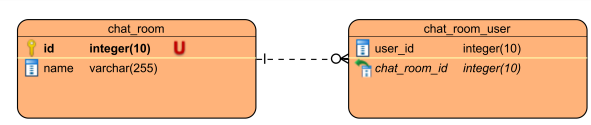
3.2.3 Microservicio chat

En esta vista, el microservicio de chat almacena información del usuario que envia el mensaje, a que sala de chat lo envía, el mensaje, y si este mensaje será oculto (el usuario decidió eliminarlo). También se guarda la hora en el que el mensaje fue enviado.



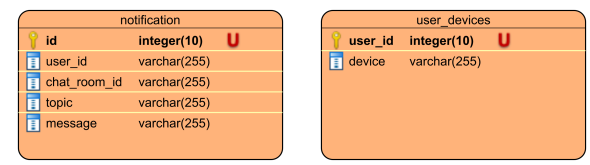
3.2.4 Microservicio chat-rooms

Los chat-room almacenan información referentes a las salas de chat como lo son el nombre de la sala y qué usuarios pertenecen a élla. Esta entidad puede ser ampliada para contener mas información.



3.2.5 Microservicio notificaciones

Para las notificaciones, es de interés guardar información sobre los dispositivos que usan los usuarios, además de la información de las notificaciones en si tal como qué usuario es el que le hace llegar la notificación, el título y el mensaje de esta.



3.3 Vista de Componentes y Conectores

Para explicar el diagrama de componentes se listarán los componentes usados, junto la explicación de su uso, además de la lista de sus relaciones.

- Componente de api_gateway
  Es el componente que se encargará de orquestar los servicios ofrecidos por la plataforma.
  Los elementos con los que se relaciona utilizando conectores REST son:
  - El componente de usuario:
    Para poder orquestar el manejo de usuarios.
  - El componente de autenticación:
    Para poder orquestar el manejo de sesiones y autenticación de la aplicación.
  - El componente de chat:
    Para orquestar el envío de chats en la aplicación.
  - El componente de chat room:
    Para orquestar la creación de los diversos chats.
  - El componente de notificación:
    Para orquestar el envío de notificaciones a los usuarios.

  Los elementos con los que se relaciona utilizando conectores GraphQL son:
  - El componente de Front End Web:
    Para poder exponer los servicios de la plataforma por medio de una interfaz web.
  - El componente de Font End Mobile:
    Para poder exponer los servicios de la plataforma por medio de una interfaz movil.
- Componente de Usuarios
  Este es el componente que manejará la información de los usuarios, pero poco tendrá que ver con el manejo de sus contraseñas. Este componente se relaciona con los siguientes:
  - El componente User DB por medio del conector pg:
    Para almacenar los datos sobre los usuarios.
  - El componente api _gateway por medio de un conector REST:
    Para exponer los servicios de los usurios a través de él.
- Componente User DB Este componente almacenará la información de los usuarios Los componentes que se relacionan con la base de datos de los usuarios son:
  - El componente User por medio del conector pg:
    Para poder ofrecer los servicios de los usuarios.
  - El componente Autentication por medio del conector pg node:
    Para adquirir las contraseñas de los usuarios.
- Componente Autenticación:
  El componente de Autenticación será el encargado del manejo de sesiones dentro de la aplicación. Las relaciones de este componente son:
  - El componente api_gateway por medio de un conector REST:
    Para exponer los servicios de la autenticación a través de él.
  - El componente de User DB por medio del conector pg node:
    Para poder recuperar las contraseñas de los usuarios.
  - El componente de Autentication DB por medio del conector ioredis:
    Para poder usar almacenar información de User DB en cache.

- Componente de Autentication DB:
  Las contraseñas son almacenadas en caché para evitar el uso de la base de datos User DB y la verificación del JWT usando la base de datos de Usuarios.
  Los componentes que se comunican con él son:
  - El componente de Autentication por medio del conector ioredis:
    Para poder usar la información de usuarios almacenada en caché
- Componente Chat
  Es el componente encargado del tránsito de la gestión de los mensajes en el sistema.
  Los componentes con los que se comunica son:
  - El componente api_gateway por medio de un conector REST:
    Para exponer los servicios de la gestión de los mensajes a través de él.
  - El componente Chat DB por medio del conector mongoengine:
    Es usado para almacenar los mensajes en una base orientada a documentos NoSQL.
- Componente Chat DB
  Es el componente que almacena los mensajes en el sistema.
  Los componentes con los que se comunica son:
  - El componente Chat por medio del conector mongoengine:
    El componente chat utilizará chat DB para almacenar los mensajes.
- Componente ChatRoom
  Es el componente que gestiona las salas de chat, donde los usuarios podrán enviar mensajes. Los componentes con los que se comunica son:
  - El componente api_gateway por medio de un conector REST:
    Para exponer los servicios de la gestión de las salas de usuario a través de él.
  - El componente ChatRoom DB por medio del conector jdbc:
    Para almacenar la infomación relacionada con las salas de chats.
- Componente ChatRoom DB
  Es el componente que almacena la información relacionada con las salas de chat. Los componentes con los que se comunica son:
  - El componente ChatRoom por medio del conector jdbc:
    El componente ChatRoom utilizará ChatRoom DB para almacenar la infomación relacionada con las salas de chats.
- Componente Notification
  Es el componente que administra las notificaiones hacia los usuarios en el sistema. Los componentes con los que se comunica son:
  - El componente api_gateway por medio de un conector REST:
    Para exponer los servicios de la gestión de notificaiones a través de él.
  - El componente Notification DB por medio del conector go_redis:
    Para almacenar la infomación relacionada con las notificaciones.
- Componente Notification DB
  Es el componente que almacena la información relacionada con las notificaiones en el sistema. Los componentes con los que se comunica son:

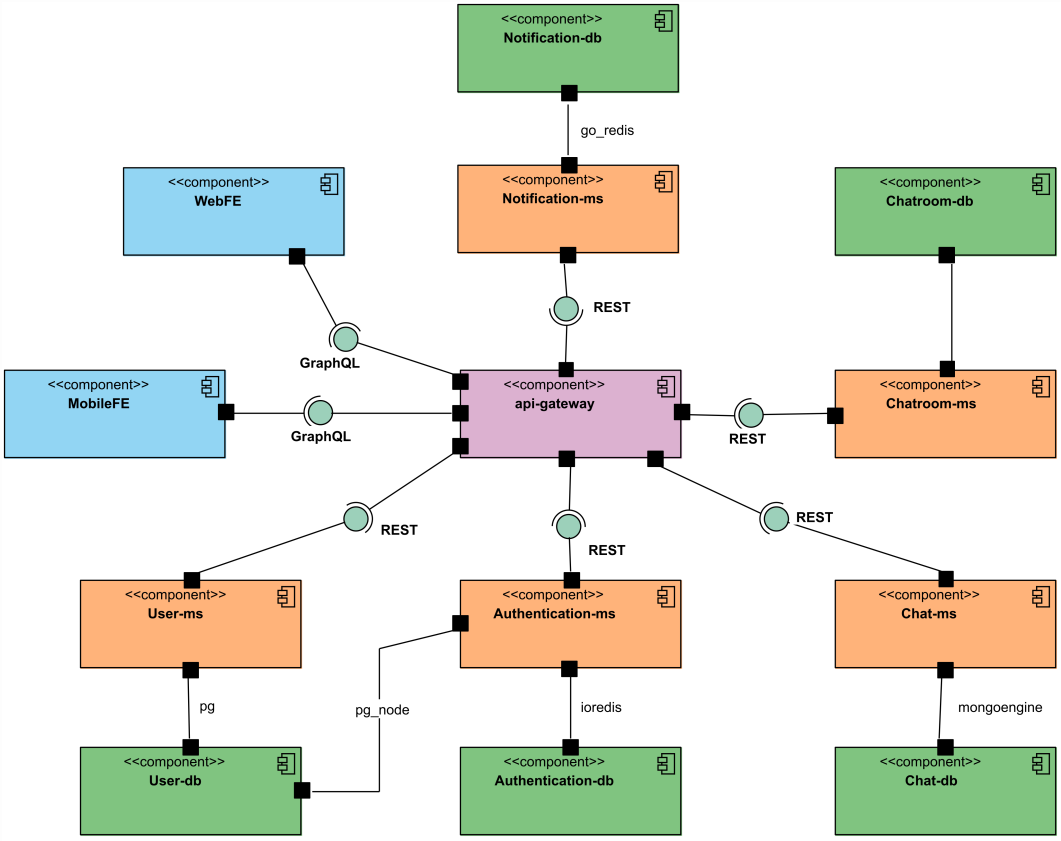
  - El componente Notification por medio del conector go_redis:
    El componente Notification utilizará Notification DB para almacenar la infomación relacionada con las notificaciones.

- Componente Web FE:
  Es el componente encargado de consumir los servicios expuestos por el api_gateway para la presentación al usuario por medio de la web:
  Los componentes con los que se comunica son:

  - El componente api_gateway por medio del conector GraphQL:
    Para consumir los servicios expuestos.

- Componente Mobile FE:
  Es el componente encargado de consumir los servicios expuestos por el api_gateway para la presentación al usuario por medio de una aplicación móvil:
  Los componentes con los que se comunica son:

  - El componente api_gateway por medio del conector GraphQL:
    Para consumir los servicios expuestos.



3.4 Vista de Capas

La vista de capas nos permite visualizar de manera precisa la secuencia lógica por la que pasa la información en la aplicación, para presentar esta vista se ha divido en 3 tiers los cuales son:

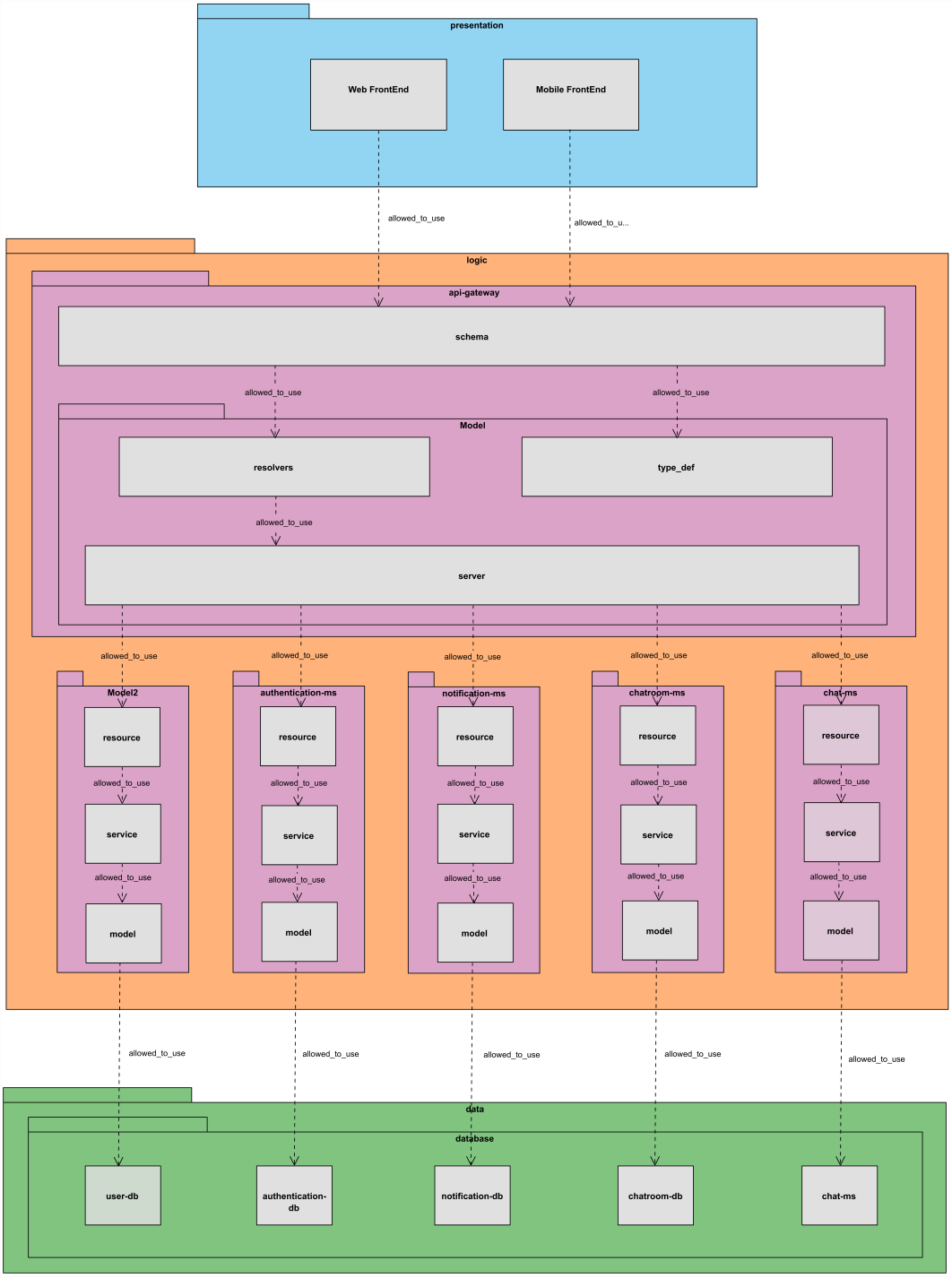
- Presentacion:
  El tier de presentación es el más cercano al usuario final y está compuesto por 2 capas las cuales componen el Front End de la aplicación (Web FrontEnd y Mobile FrontEnd).
  Ambas capas tienen permitido acceder al tier de lógica haciendo uso de la capa del api-gateway y de manera más precisa a la subcapa del esquema.

- lógica:
  El tier de lógica contiene, como su nombre lo indica, toda la lógica de la obtención y manipulación de los datos usados por la aplicación, este tier está compuesto por las siguientes capas:

  - Api-gateway
    Recibe y encamina los datos recibidos por los FrontEnd hacia las capas más internas, para esto se divide en 2 segmentos:
    - Esquema:
      Da formato a los datos recibidos haciendo uso del segmento de definición de tipos (type_def) y los remite hacia el segmento de resolvers.
    - Modelo:
      El segmento de modelo encierra a su vez 3 segmentos especializados.
      - Tipe_def:
        Define los tipos de datos permitidos en la aplicación.
      - Resolvers:
        Toma los datos entregados por el esquema y resuelve hacía que capa o capas debe redirigir los datos.
      - Server:
        Toma los datos y los encamina de manera lógica hacia el siguiente conjunto de capas para cumplir correctamente el flujo de la aplicación.
  - Users-ms
    Recibe las peticiones relacionadas a la administracion de usuarios, este componente da respuesta a las solicitudes haciendolas pasar por 3 segmentos:
    - Recurso:
      Expone la ruta y define los parametros necesarios para una accion concreta.
    - Servicio:
      Lleva a cabo la operaciones especificas para poder cumplir con la solicitud.
    - Modelo:
      Expresa los atributos e informacion necesaria que tiene cada uno de los datos para poder ser operados, es la capa lógica mas cercana a los datos en bruto.
  - Notifications-ms
    Recibe las peticiones e información necesaria para poder realizar el envio de notificaciones a los usuario, el flujo de esta informacion pasa por los siguiente segmentos:
    - Recurso:
      Expone la ruta y define los parametros necesarios para una accion concreta.
    - Servicio:
      Lleva a cabo la operaciones especificas para poder cumplir con la solicitud.
    - Modelo:
      Expresa los atributos e informacion necesaria que tiene cada uno de los datos para poder ser operados, es la capa lógica mas cercana a los datos en bruto.

- Authentication-ms
  Recibe las peticiones relacionadas a la autenticacion de usuarios, este componente da respuesta a las solicitudes haciendolas pasar por 2 segmentos:
  * Recurso:
    Expone la ruta y define los parametros necesarios para una accion concreta.
  * Servicio:
    Lleva a cabo la operaciones especificas para poder cumplir con la solicitud.
- Chatroom-ms
  Recibe las peticiones relacionadas a la administracion de salas de chat, este componente da respuesta a las solicitudes haciendolas pasar por 3 segmentos:
  * Recurso:
    Expone la ruta y define los parametros necesarios para una accion concreta.
  * Servicio:
    Lleva a cabo la operaciones especificas para poder cumplir con la solicitud.
  * Modelo:
    Expresa los atributos e informacion necesaria que tiene cada uno de los datos para poder ser operados, es la capa lógica mas cercana a los datos en bruto.
- Chat-ms
  Recibe las peticiones relacionadas a los mensajes que se dan en los chats, este componente da respuesta a las solicitudes haciendolas pasar por 3 segmentos:
  * Recurso:
    Expone la ruta y define los parametros necesarios para una accion concreta.
  * Servicio:
    Lleva a cabo la operaciones especificas para poder cumplir con la solicitud.
  * Modelo:
    Expresa los atributos e informacion necesaria que tiene cada uno de los datos para poder ser operados, es la capa lógica mas cercana a los datos en bruto.

• Datos:
  Es el tier mas alejado de el usuario final, contine un capa de base de datos en donde los diferentes modelos de la aplicación extraen la informacion que necesitan.
  Esta capa a su vez tiene 5 segmentos correspondientes a cada uno de los microservicios, estos son:
  - User DB
  - Notifications DB
  - Authentication DB
  - Chat Room DB
  - Chat DB

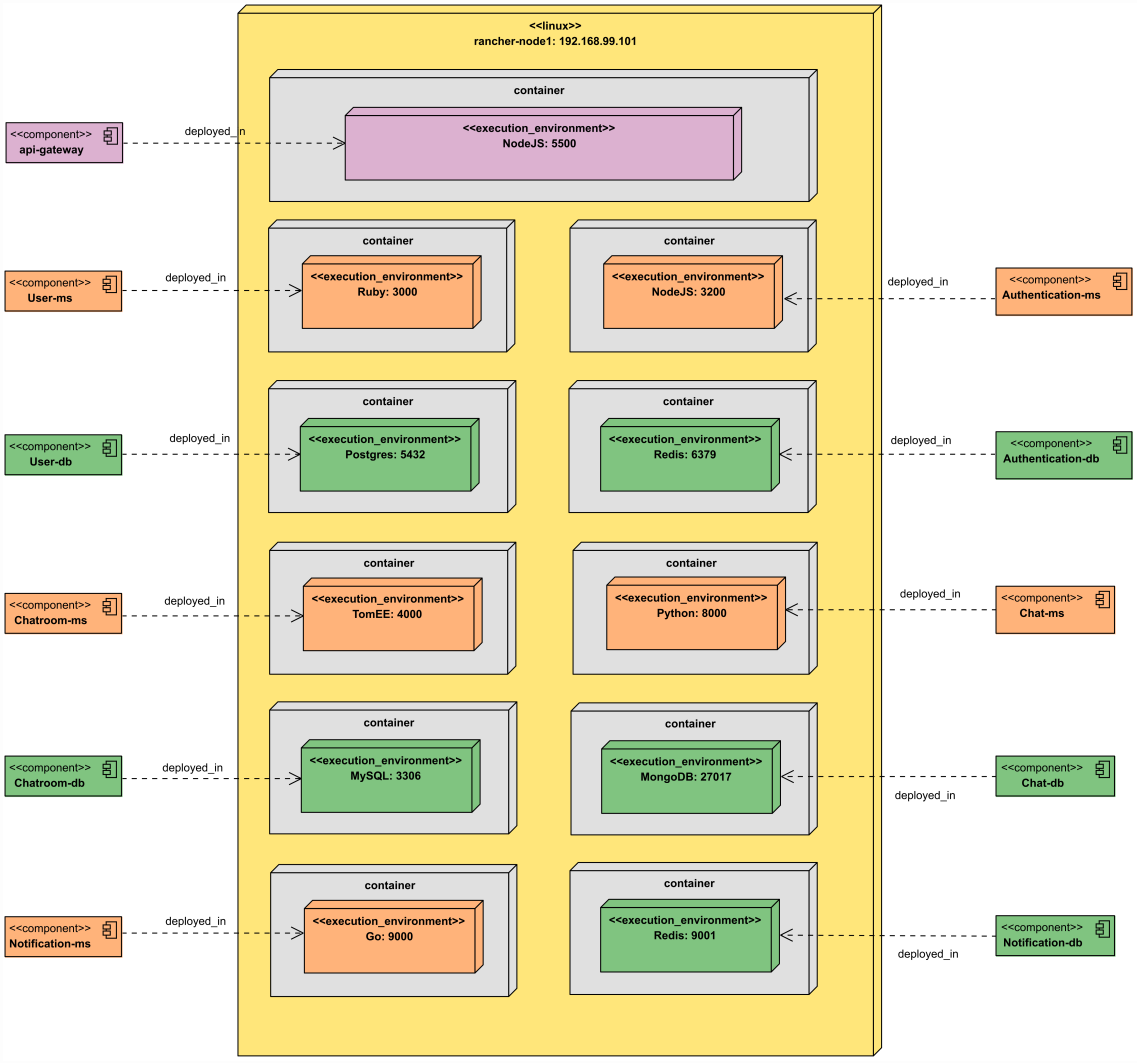


3.5 Vista de Despliegue

Para la vista de depliegue se tiene una máquina virtual que está ejecutando un ambiente Linux y es un nodo rancher. Esta máquina responde al host 192.168.99.101 y en él se despliegan los componentes de la aplicación en contenedores de la

siguiente manera:

- Un contenedor con el ambiente de ejecución Ruby que expone el puerto 3000 para el despliegue del componente User.
- Un contenedor con el ambiente de ejecución Postgres que expone el puerto 5432 para el despliegue del componente User DB.
- Un contenedor con el ambiente de ejecución TomEE que expone el puerto 4000 para el despliegue del componente ChatRoom.
- Un contenedor con el ambiente de ejecución MySQL que expone el puerto 3306 para el despliegue del componente ChatRoom DB.
- Un contenedor con el ambiente de ejecución Python que expone el puerto 8000 para el despliegue del componente Chat.
- Un contenedor con el ambiente de ejecución MongoDB que expone el puerto 27017 para el despliegue del componente Chat DB.
- Un contenedor con el ambiente de ejecución NodeJS que expone el puerto 3200 para el despliegue del componente Authentication.
- Un contenedor con el ambiente de ejecución Redis que expone el puerto 6379 para el despliegue del componente Authentication DB.
- Un contenedor con el ambiente de ejecución Go que expone el puerto 9000 para el despliegue del componente Notification.
- Un contenedor con el ambiente de ejecución Redis que expone el puerto 9001 para el despliegue del componente Notification DB.
- Un contenedor con el ambiente de ejecución NodeJS que expone el puerto 5500 para el despliegue del componente api_gateway.


<<linux>>
rancher-node1: 192.168.99.101
container
<<execution_environment>>
NodeJS: 5500
<<component>>
api-gateway
deployed_in
container
<<execution_environment>>
Ruby: 3000
<<component>>
User-ms
deployed_in
container
<<execution_environment>>
NodeJS: 3200
deployed_in
<<component>>
Authentication-ms
container
<<execution_environment>>
Postgres: 5432
<<component>>
User-db
deployed_in
container
<<execution_environment>>
Redis: 6379
deployed_in
<<component>>
Authentication-db
container
<<execution_environment>>
TomEE: 4000
<<component>>
Chatroom-ms
deployed_in
container
<<execution_environment>>
Python: 8000
deployed_in
<<component>>
Chat-ms
container
<<execution_environment>>
MySQL: 3306
<<component>>
Chatroom-db
deployed_in
container
<<execution_environment>>
MongoDB: 27017
deployed_in
<<component>>
Chat-db
container
<<execution_environment>>
Go: 9000
<<component>>
Notification-ms
deployed_in
container
<<execution_environment>>
Redis: 9001
deployed_in
<<component>>
Notification-db

4 Tácticas Arquitecturales a favor de la Seguridad

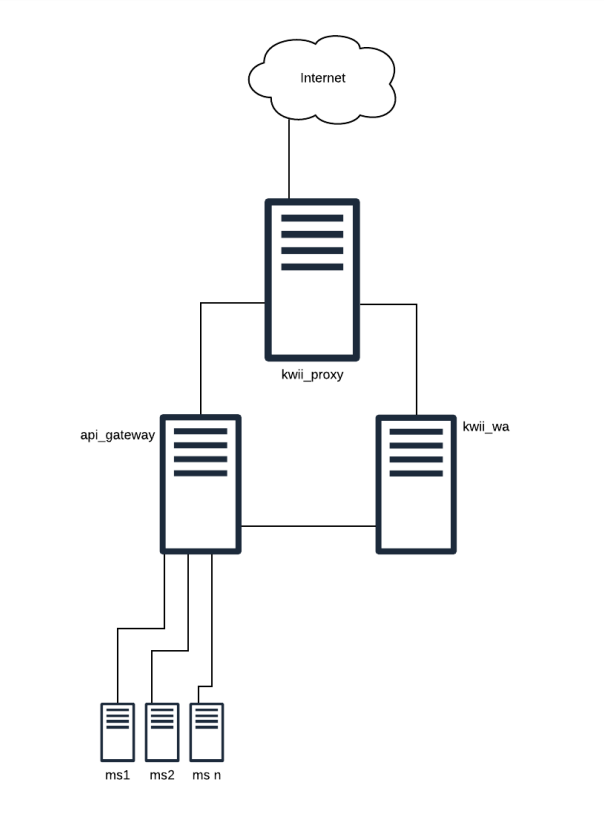
El diseño de la Kwii Platform vela por seguridad de la información de sus usuarios, es decir, las decisiones arquitecturales de la plataforma tuvieron en cuenta aspectos del menejo de la información de los usuarios con el fin de proveer al acceso a esta de manera fácil y eficaz, pero solo a los usuarios autorizados a obtenerla y además, las decisiones facilitan negar el acceso a esta información de manera rotunda a quien no esté autorizado. Es por ello que se plantea esta sección, donde se discutirán diferentes decisiones que fueron tomadas para hacer a la Kwii Platform un sistema seguro.

4.1 Uso de un Servidor de Proxy Inverso

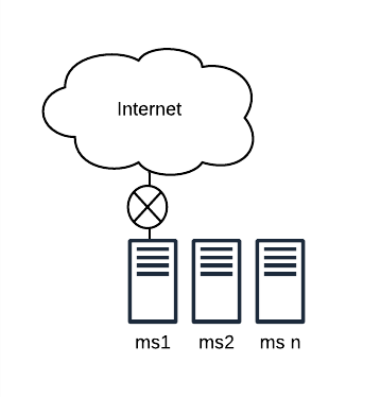
En la arquitectura de la Kwii Platform, se plantean diferentes componentes que son encargados de numerosas responsabilidades dentro de la plataforma. Dichos componentes satisfacen las responsabilidades a las que son encargadas de maneras muy diversad pero, en general, para cumplir dichas responsabilidades los componentes exponen servicios que son consumidos por un único componente central que hace las de mediador en la arquitectura. Para evitar el acceso no autorizado con posibles acciones malintencionadas y garantizar que los servicios que prestan estos componentes sean únicamente consumidos por el componente central (tal y como lo plantea la vista de capas), se propone el uso de un Servidor de Proxy Inverso.

4.1.1 Justificación del uso de un Servidor de Proxy Inverso

El Servidor de Proxy Inverso es el encargado de exponer los servicios compuestos del componente central, de exponer el componente Front End Web y de evitar el contacto de los demás componentes con el exterior exterior. La comunicación del mundo exterior con la Kwii Platform es representada en la siguiente figura:



Como se mencionó anteriormente, cualquier intento de comunicación de otro elemento de la plataforma con el exterior será descartado.



4.1.2 Características de Seguridad Protegidas

Las características de seguridad protegidos en la Kwii Platform a la hora de implementar un servidor de proxy inverson son las siguientes:

- Confidencialidad
  El servidor de proxy inverso potencia la confidencialidad de la información al disminuir los puntos de acceso a esta. Ahora, viéndonos limitados a acceder a la información de los demás componentes de la plataforma únicamente por

medio del componente central, la arquitectura puede ser reforzada en estos aspectos únicamente tratando los flujos de información que pasan por dicho componente.

- Integridad
  Así mismo como el servidor de proxy inverso asegura la confidencialidad, asegura también la integridad. Restringimos la cantidad de puntos de manipulación de la información.
- Disponibilidad Como trabajo futuro, el servidor de proxy inverso implementa dfunciones de balanceo de carga que garantizan la disponibilidad. (Próximamente)

4.2 Manejo de Sesión de Usuarios

Para el manejo de sesión de usuarios, la arquitectura de la Kwii Platform propone el uso de un esquema basado en tokens. En él, cualquier comunicación, solicitud de información, y en general, cualquier operación en la plataforma tiene que ser autorizada, por medio de la validación de tokens de usuario.

4.2.1 Justificación del uso de autenticación basada en tokens

Hay diversas características que hacen que la autenticación por tokens funcione bastante bien con la arquitectura propuesta de la Kwii Platform, entre ellas, se destacan las siguientes:

- Los tokens siguen el principio stateless de la comunicación HTTP
  Al contener en sí mismo la información de la sesión, los tokens nos permiten manipular la autenticación de los usuarios sin la necesidad de trazar su estado.
- Generación y manipulación centralizado
  La arquitectura de nuestra aplicación permite que la autenticación sea manejada únicamente por los componentes encargados de ello, así los tokens son manipuladas de manera centralizada y el resto de los componentes que sean agregados a la arquitectura no se tienen que responsabilizar por cuestiones de autenticación.
- Control de acceso
  El uso de tokens de usuario nos permite determinar de manera relativamente sencilla qué usuarios tienen acceso a qué. Y restringir de manera efectiva las solicitudes de información no autorizadas.

4.2.2 Características de Seguridad Protegidas

Las características de seguridad protegidos en la Kwii Platform a la hora de implementar un servidor de proxy inverson son las siguientes:

- Confidencialidad e integridad
  El uso de tokens garantiza la confidencialidad de información creando un mecanismo de acceso en el que solo pueden acceder a la información aquellos usuarios a los que se les conceden permisos autenticándose y recibiendo un token. De la misma manera, únicamente los usuarios a los que se les asigna un token al ser autenticados, son capaces de realizar cambios en la información.

4.3 Uso de Directorio Activo