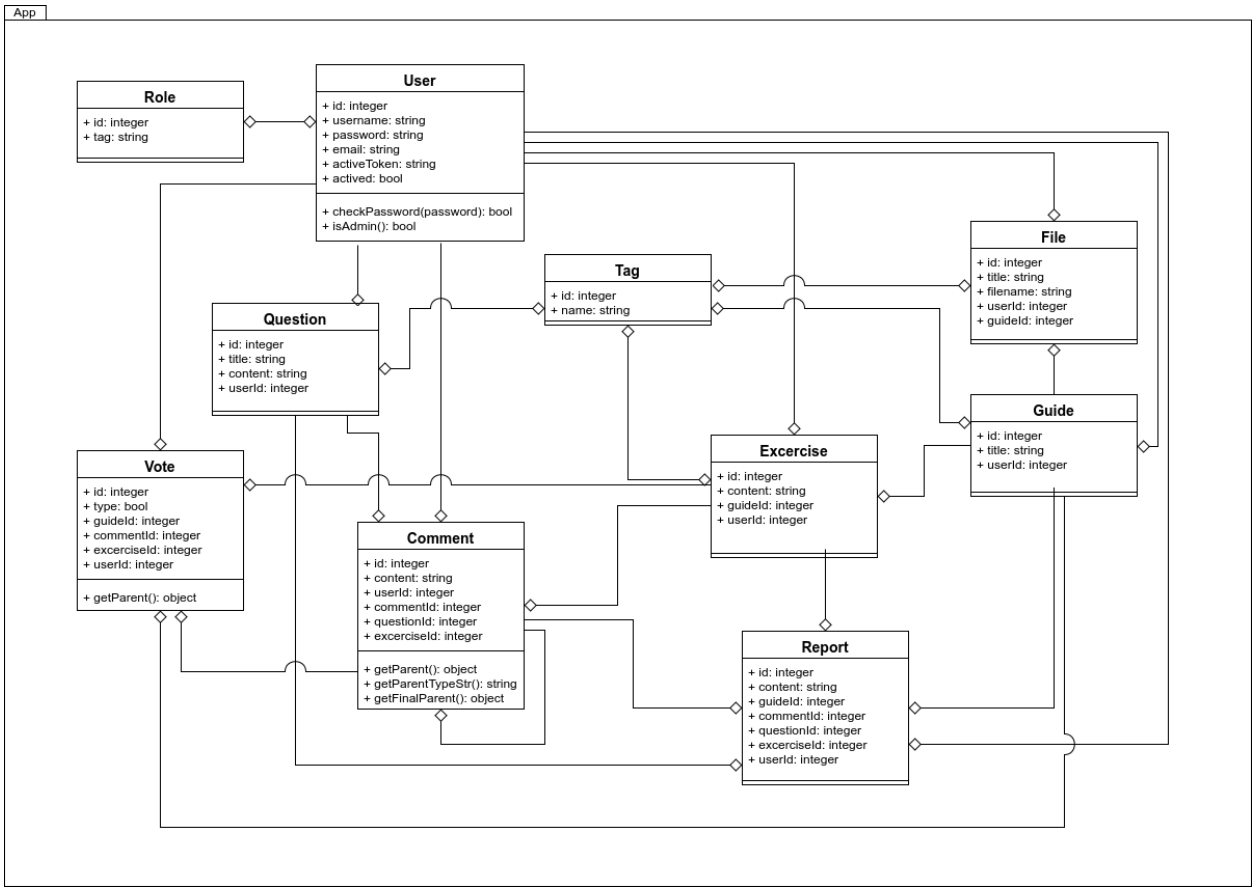
Documentación

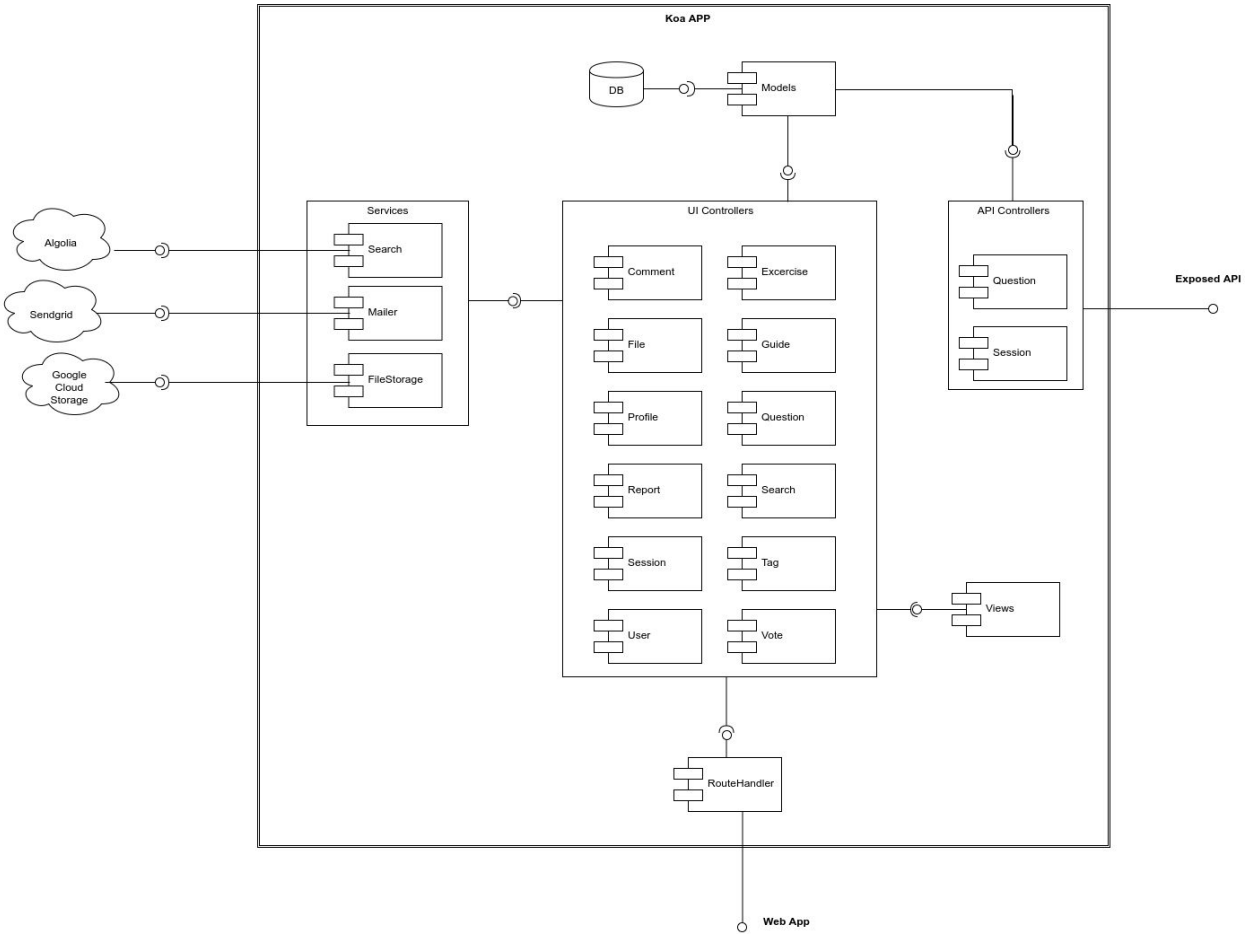
Modelo de Datos

A continuación se encuentra el diagrama UML que representa al modelo de datos usados en nuestro sistema.



Arquitectura del Sistema

A continuación se encuentra el diagrama UML de la arquitectura de nuestro sistema



Se puede notar que nuestra aplicación puede ser accedida de dos formas: consumiendo la API expuesta, o accediendo a la aplicación web desde un navegador.
Internamente, sigue una estructura MVC para la mayor parte del sistema, donde hay un *template engine* encargado de generar las vistas, un *route handler* encargado de guiar a un determinado controlador, y el modelo que se encarga de interactuar con la base de datos.
Además, creamos tres principales servicios que alimentan funcionalidades de nuestros controladores, estos son: búsqueda, mailer y almacenamiento de archivos. Estos se conectan a servicios externos para funcionar: Algolia, Sendgrid y Google Cloud Storage, respectivamente.
También podemos notar que están completamente separados los controladores para la API y para la aplicación Web, pero que están conectados al mismo modelo. Esto

permite tener funcionalidades específicas para cada punto, pero que ambos trabajen con los mismo datos.

Para consumir la API pueden entrar a:
https://gsulloa.github.io/sgfundamentals-api-consumer

Detalles de implementación

En cuanto a Algolia, nuestra aplicación su API para ofrecer una búsqueda inteligente y en tiempo real al usuario. Para esto, la aplicación está sincronizada con una base de datos de Algolia que lleva un índice de todas las entidades importantes en nuestra base de datos, y todo cambio que se haga en nuestra base de datos a través de la aplicación se verá reflejado en los índices de Algolia mediante requests hechos desde el servidor. Esto posibilita que al realizar una búsqueda el cliente envíe requests al servidor de Algolia y no al nuestro, y obtenga resultados instantáneos generados por un algoritmo más inteligente y personalizable.

Por otro lado, construimos un sistema interactivo para las preguntas de la aplicación. Este fue construido con ReactJS, que permitió manejar de manera más simple y eficiente las vistas del usuario.
Consta de principalmente 4 componentes: *Question, Answer, Voting* y *Report,* cada uno de estos con una funcionalidad específica.
Question, se encarga de renderizar la estructura general de la pregunta, carga el contenido y le entrega los datos necesarios a los otros componentes para hacer el *render.*
Answer, es el componente encargado de mostrar respuestas (y anidaciones de respuesta), y mostrar un componente de formulario, el cual maneja su propio estado para enviar el contenido nuevo de respuesta
Voting, es el componente encargado de mostrar los votos que posee una pregunta y sus respuestas, además de entregar la funcionalidad de votar sobre ellas.
Report, se encarga de entregar un acceso a reportar cada una de las entidades, ya sea pregunta o respuestas.
Todos estos componentes están desarrollados con patrones de *Dumb* y *Smart components*.