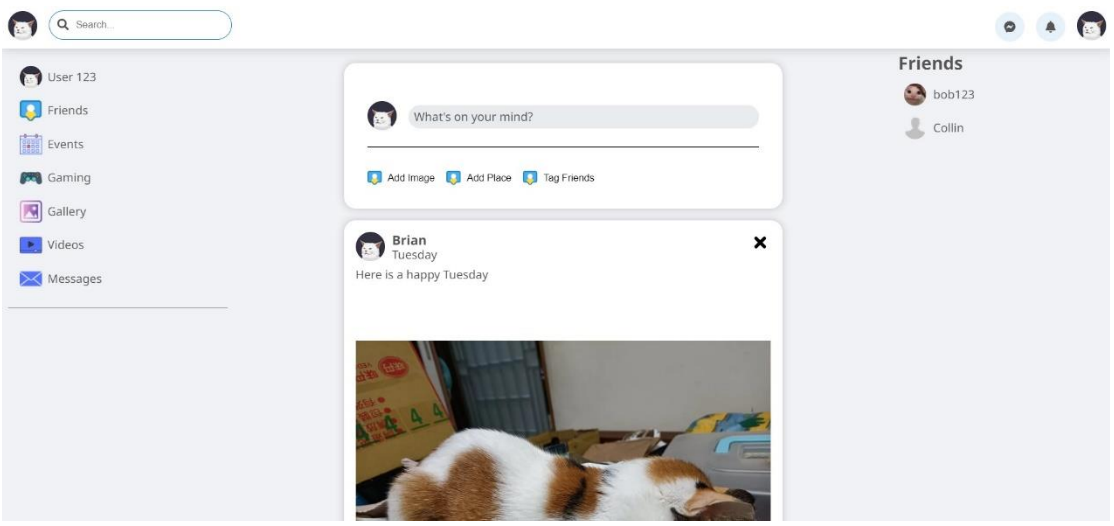
CatBook – A social app clone

Introduction

This project is focus on re-build the main functionality of the modern famous social media app(ex: Facebook, WhatsApp…). The app includes post and comment system, friendship system, and instant feedback chatting system. Below are the features this project implements:

1. User can write their post, with text and image
2. User can delete their own post
3. User can leave common on the post
4. User can like the post
5. User can add another user as friend
6. User can have a conversation with their friend
7. User's conversation will have immediately response

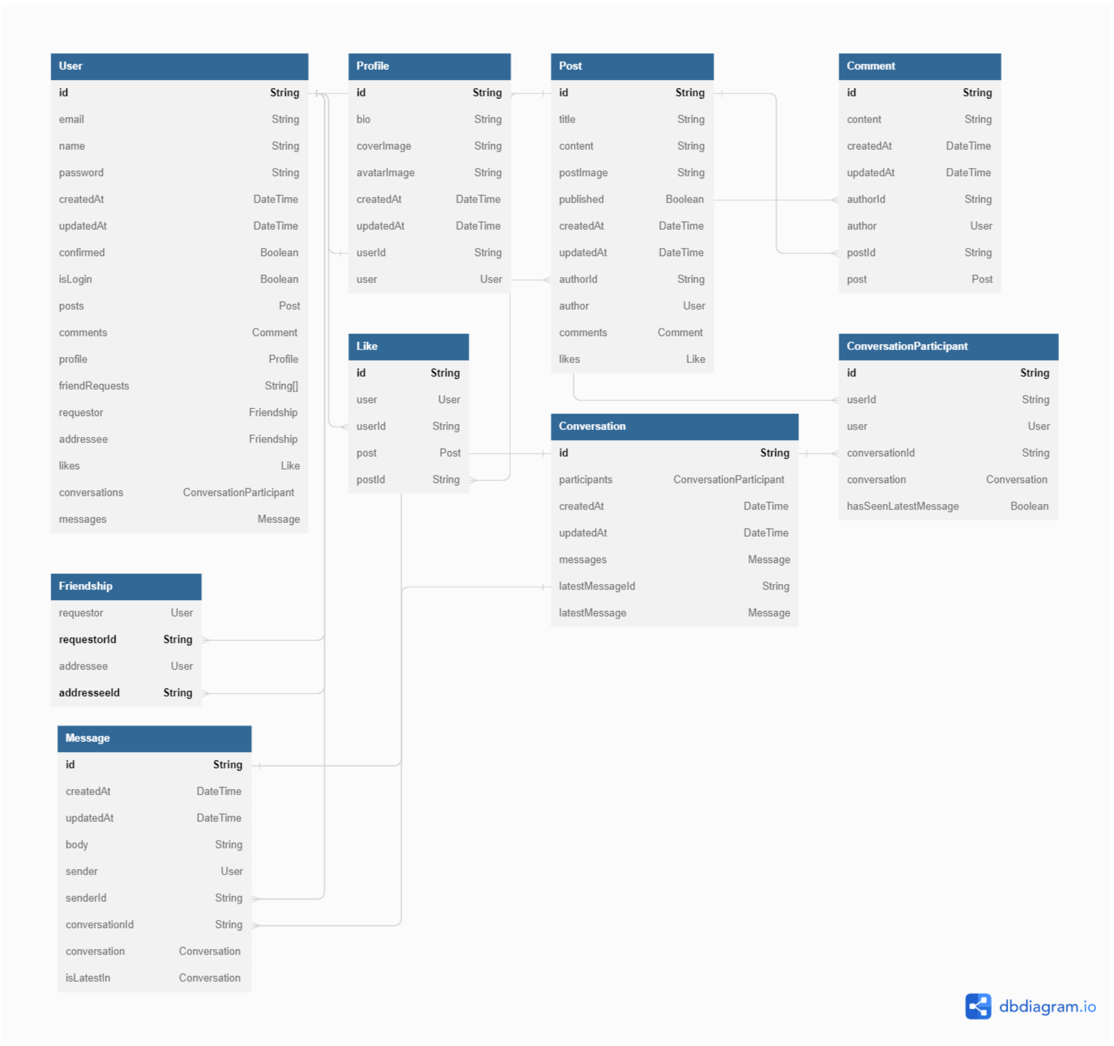


1- Short cut of the project home page

Structure and Design

Database

The database(DB) is built with Postgres SQL DB because there will involve some complex query operation in this app. So SQL DB is prefer than NoSQL DB. Next, the prisma library, which is a famous ORM library, is used to simplify the codes of database query. By using this ORM system, the DB schema is easier to build and maintain thus I can focus more on business logic and UI design. Finally, I choose firebase as an image storage due to it's well-define API and nice user interface.



User	
id	**String**
email	String
name	String
password	String
createdAt	DateTime
updatedAt	DateTime
confirmed	Boolean
isLogin	Boolean
posts	Post
comments	Comment
profile	Profile
friendRequests	String[]
requestor	Friendship
addressee	Friendship
likes	Like
conversations	ConversationParticipant
messages	Message

Profile	
id	**String**
bio	String
coverImage	String
avatarImage	String
createdAt	DateTime
updatedAt	DateTime
userId	String
user	User

Post	
id	**String**
title	String
content	String
postImage	String
published	Boolean
createdAt	DateTime
updatedAt	DateTime
authorId	String
author	User
comments	Comment
likes	Like

Comment	
id	**String**
content	String
createdAt	DateTime
updatedAt	DateTime
authorId	String
author	User
postId	String
post	Post

Like	
id	**String**
user	User
userId	String
post	Post
postId	String

ConversationParticipant	
id	**String**
userId	String
user	User
conversationId	String
conversation	Conversation
hasSeenLatestMessage	Boolean

Conversation	
id	**String**
participants	ConversationParticipant
createdAt	DateTime
updatedAt	DateTime
messages	Message
latestMessageId	String
latestMessage	Message

Friendship	
requestor	User
requestorId	**String**
addressee	User
addresseeId	**String**

Message	
id	**String**
createdAt	DateTime
updatedAt	DateTime
body	String
sender	User
senderId	String
conversationId	String
conversation	Conversation
isLatestIn	Conversation

2-database diagram

Backend server

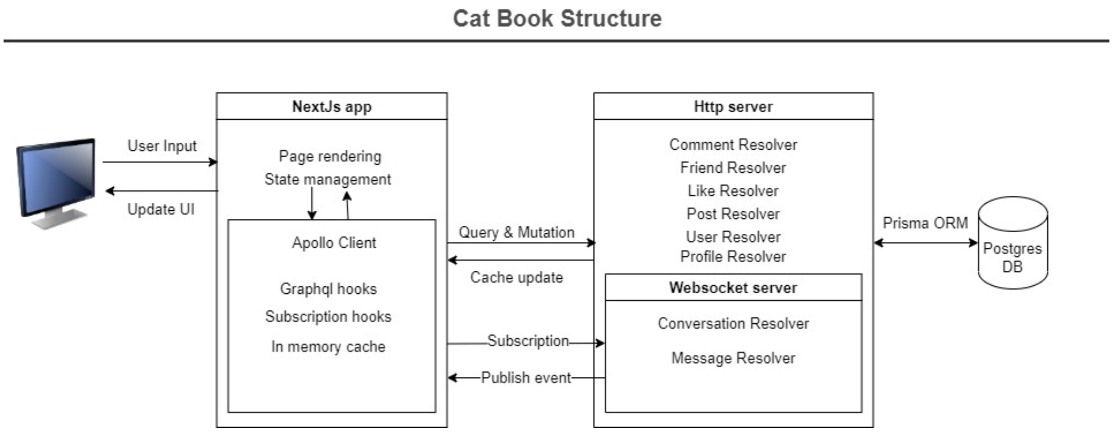
The server is built with Nodejs with Express and Apollo server framework. All server API is built with the graphql basis instead of Restful API because it is more type safety and well-structured. Moreover, the frontend developer can easily transfer the type from the graphql schema to nicely connect with typescript which significantly improve the code quality which restful API can't. Finally, it can provide some nested query operation and partial query field which can greatly reduce the number and complexity of API.

The subscription system is built with simple WebSocket pub-sub library. With the user subscribe to the server, when some events occur (ex: conversation is created, message is created), the server will publish the event to notice the user thus user can update their state instantly.

Frontend Application

The application is built with Javascript with NextJs, Apollo client frameworks. The NextJs is a famous UI library built on ReactJs, it has well-structure page routing system and Server-side rendering ability. Also, the mature ecosystem is another benefit of this react-base framework.

The data fetching part is relied on Apollo client because it is one of the famous and mature graqhql frontend framework. Also, it has a built-in cache system which can easily used to enhance the data fetching efficiency and give user the better using experience.



3-App structure

Functions

The app is composed of three main functions: Authentication system, Data query and mutation, and WebSocket subscription.

Authentication and Authorization

For the authentication, I use JWT method because comparing to cookie session strategy, token base method is stateless so I don't need to prepare another storage system to preserve the session data which lead to simple and fast performance. However, the token along will contain potential risks below: vulnerable of losing token , frequently re-authentication. So, the refresh mechanism set on http only cookies is used to keep the token only live in short period and re-validate the user behind the scenes which both improve the safety and user experience.

Data Query and Mutation

The data query and mutation are built with graphql system. For the query, thanks to the cache Apollo client provide, query result can be store in user memory so the same query can be access on cache for the duplicate fetching which boost the speed of the app. For the mutation, I use the response to change the cache data which keep the user from re-fetch the server and some cases the manually change cache to update UI which give user better using experiences.

Data Subscription

For the client app perspective, the fetch-more method and use subscription hook were used to subscribe the data event. The former one was used to handle the case which the user need the data first and wait for upcoming changes (ex: chatting room new message). The latter one is generally used, like conversation create event, message un-read event and so on.

The server side uses the simple pubsub system and keep the subscription data only on memory. For typical mutation event, like: message create or conversation create, the event will be publishing after the DB data mutation.