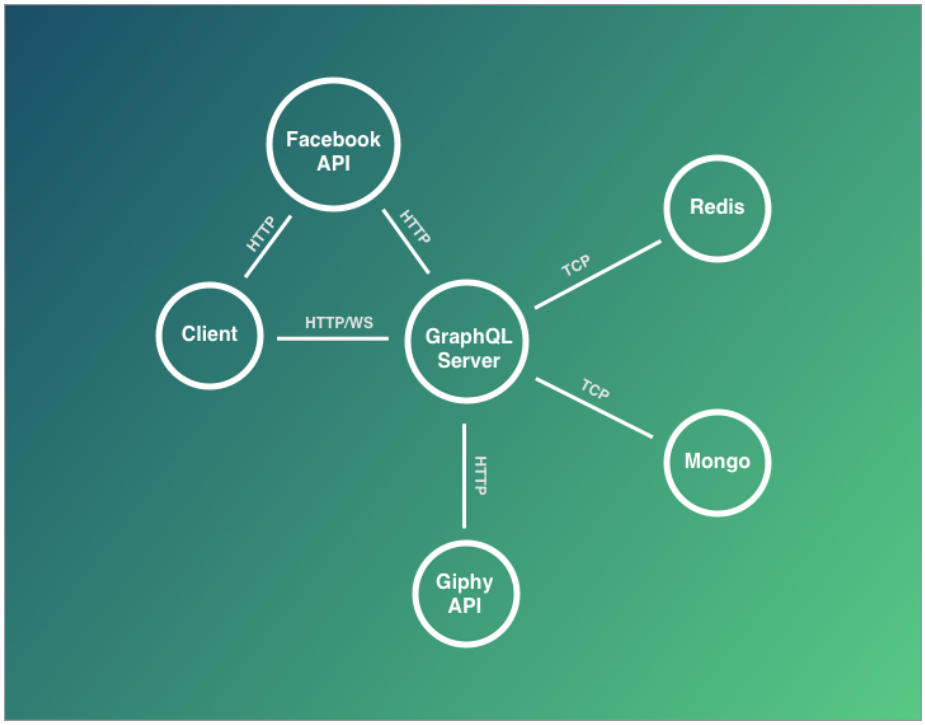
Rebus chat: a web project

Marcus Bertilsson, Alex Sundbäck, David Söderberg, Eric Wennerberg

March 13, 2018

1 Architecture



This chat is built using a client in React, a GraphQL server, Redis, MongoDB, Giphy API and Facebook API. The client communicates with the server over HTTP for non-reactive data (such as the users profile, the channels the user is currently a member of etc) and Websockets for reactive data (messages). The server exposes a GraphQL schema for the client, including the reactive and non-reactive types. The usage of GraphQL enables the client to exactly ask for the data it needs to render the view, unlike REST where the server dictates which data is returned from each endpoint.

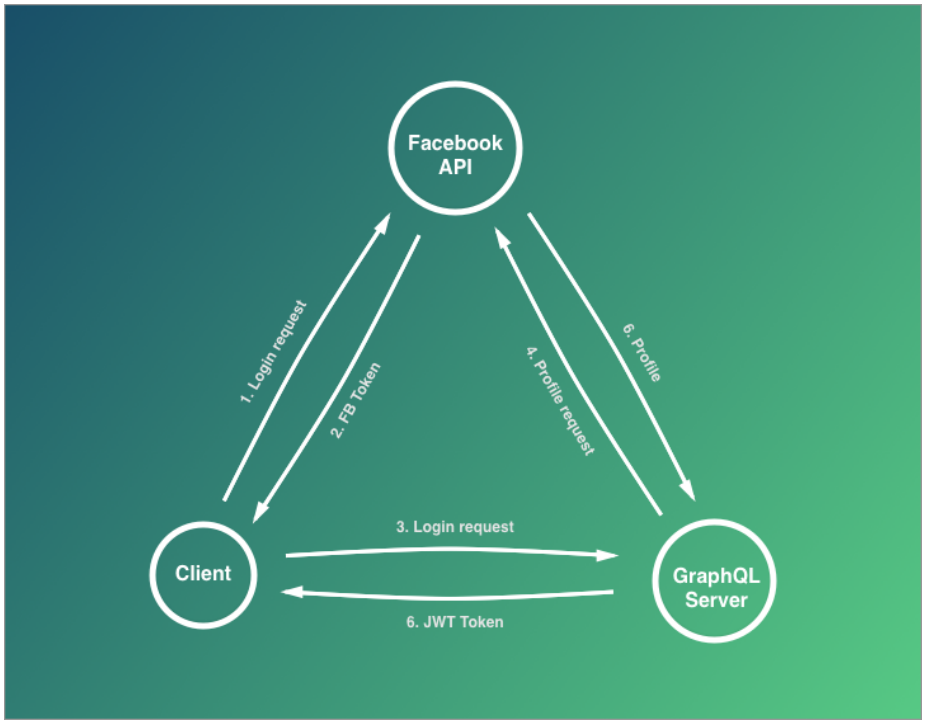
The server uses MongoDB for the persistence of all data, and uses Redis for its high-performance pub-sub implementation. In order to build a rebus the server uses Giphys API to fetch gifs that could make up the rebus. To avoid forcing the user to create an account Facebooks API is used instead.

2 Responsibilities

The client is responsible for all rendering for the end user. It has no data-management responsibility except keeping track of the current users token.
The server is responsible for all data-management, be that validation and persistance.

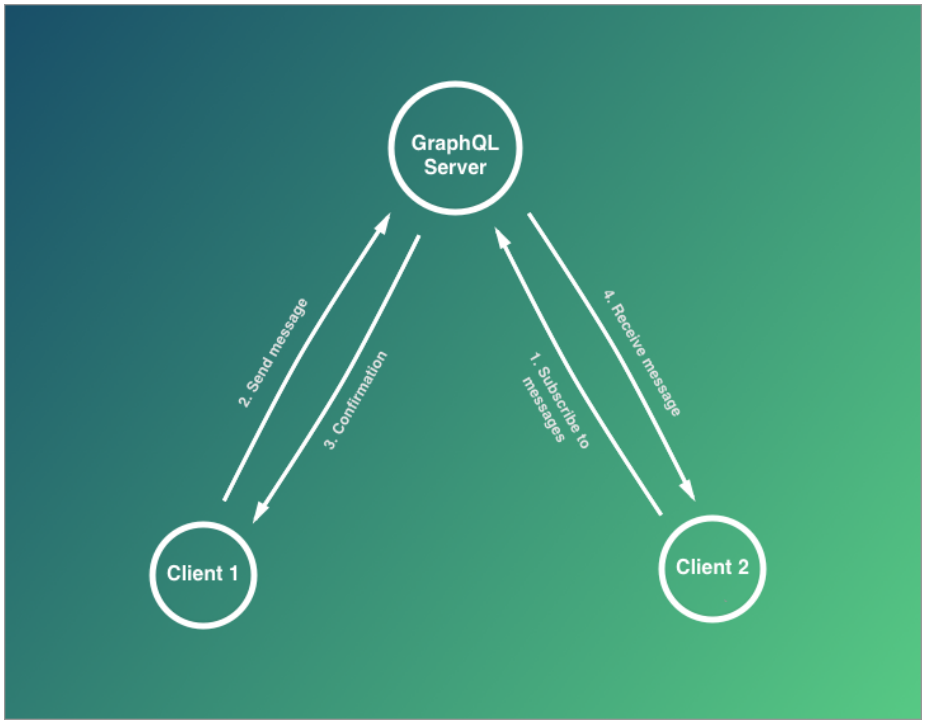
3 Flow

3.1 Logging in



For a user to login to the chat the client initiates a login request to Facebooks API. If successful the response contains a token that can be used across all Facebooks services to authenticate as this user. This token is then sent to our server as an identification for this user. The server requests the profile that belongs to the token from Facebook API. If the token is valid the response will contain the users profile (name, picture and id). This data is persisted in mongo and a JWT token is generated and sent back to the client.

3.2 Sending and receiving messages



For a user to receive real time updates of messages in a channel first a subscription is established over websockets. Then another user sends their message to a channel, and receives confirmation that the server has received it. Once the message has been received by the server it is propagated to all users that is subscribing to this channel.