Project 4 Part II: Websocket Interface and Json-Based API

January 6, 2018

1 Demo Video

Our demo video can be seen on the Youtube `https://www.youtube.com/watch?v=XQUl1OqNKtc`. Besides, we provide a demo website at `http://128.227.246.42:3000/` for readers to try our demo.

2 Installation

2.1 Install Phoenix Twitter Clone

To setup our project, we need to run (on Linux) as follows.

- sudo apt-get install postgresql postgresql-contrib
- sudo -u postgres createdb *phoenix_twitter_dev*
- sudo -u postgres psql -c "ALTER USER postgres PASSWORD 'postgres';"
- sudo apt-get install build-essential checkinstall
- sudo apt-get install libssl-dev
- curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.31.0/install.sh | bash
- source /.bashrc
- nvm install 8.9
- mix deps.get
- mix ecto.create
- mix ecto.migrate
- npm install
- mix phoenix.server

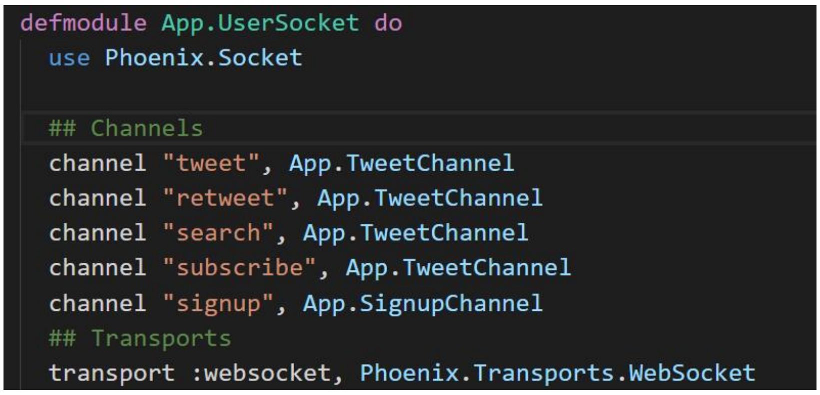
The above steps can setup the Phoenix Twitter well. If visiting to the website `localhost:3000`, we will see the homepage of Twitter.

2.2 Install wsta

Since we implemented the Phoenix Twitter in websocket interface, we need to install a websocket client to communicate with Phoenix Twitter. This client is wsta. The process of install it is as follows.

Install on Linux.

- echo 'deb http://download.opensuse.org/repositories/home:/esphen/Debian_8.0/ /' >
- /etc/apt/sources.list.d/wsta.list
- apt-get update



```
defmodule App.UserSocket do
  use Phoenix.Socket

  ## Channels
  channel "tweet", App.TweetChannel
  channel "retweet", App.TweetChannel
  channel "search", App.TweetChannel
  channel "subscribe", App.TweetChannel
  channel "signup", App.SignupChannel
  ## Transports
  transport :websocket, Phoenix.Transports.WebSocket
```

Figure 1: Websocket interface

- apt-get install wsta

Install on Mac OS X.

- brew tap esphen/wsta https://github.com/esphen/wsta.git
- brew install wsta

Install on Windows.

- Download from `https://github.com/esphen/wsta/releases`
- extract and open it in a command prompt with GNU libraries, for example the git prompt
- the command to run it is: ./wsta.exe ......

3 Functionality to Implement

Use Phoenix web framework to implement a WebSocket interface to your part I implementation. That means that, even though the Elixir implementation of your Part I project could use the Erlang messaging to allow client-server implementation, you now need to design and use a proper WebSocket interface. Specifically:

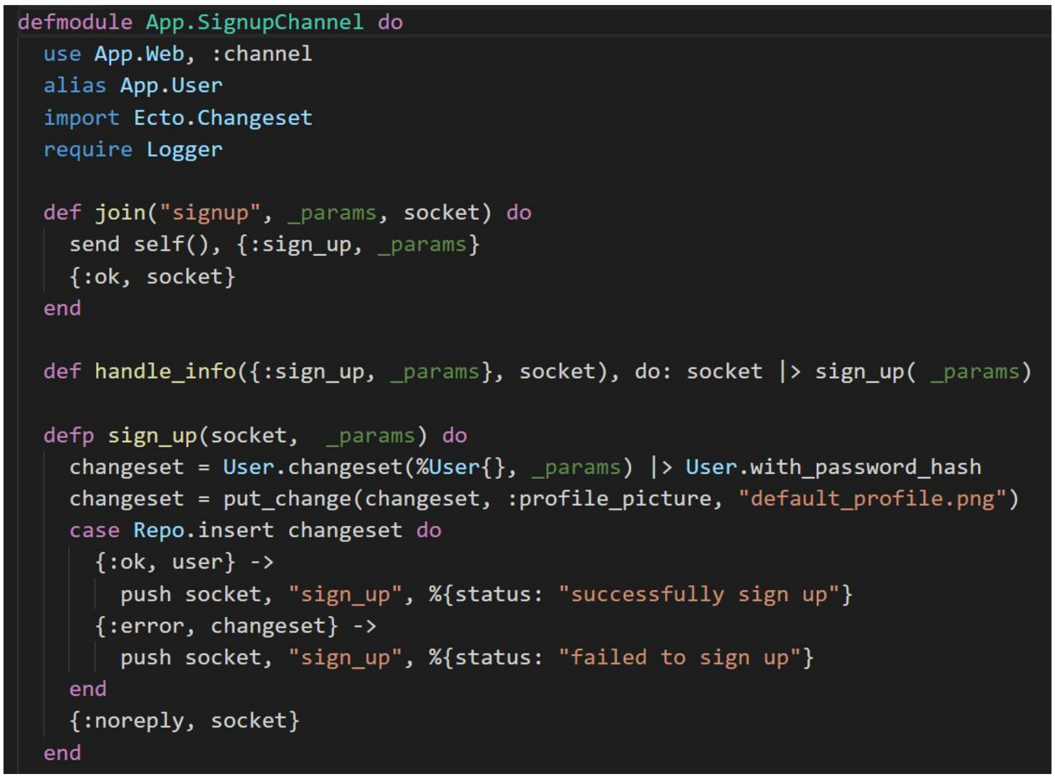
- You need to design a JSON based API that represents all messages and their replies (including errors)
- You need to re-write your engine using Phoenix to implement the WebSocket interface
- You need to re-write your client to use WebSockets.

4 Introduction

This Twitter Clone project is based on Phoenix framework, Node.js, and postgresql database. The user-interaction webpage is also included. The postgresql database is used to store the user informations and tweets informations. They include user id, user name, user password, user image, tweet id, tweet text and so on. A user using this website can sign up, log in, post tweets, follow other users, make certain tweets its favorite, and search some users, tweets or hashtags from the search box. In order to show the Websocket Json-based API, we have shown the corresponding operation in the video and will be explained below.

5 Implementation

We have already shown how to run the code and how to use those functionalities in the video `https://www.youtube.com/watch?v=XQUl1OqNKtc`. Here we will show the implementation details.



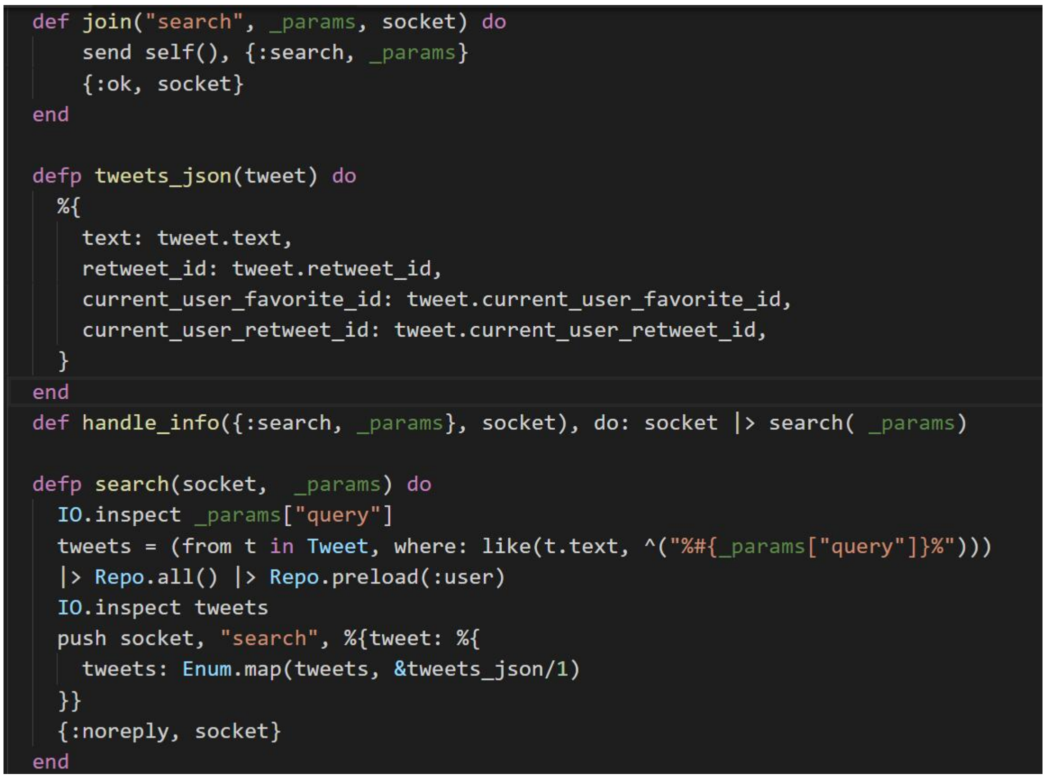
```elixir
defmodule App.SignupChannel do
  use App.Web, :channel
  alias App.User
  import Ecto.Changeset
  require Logger

  def join("signup", _params, socket) do
    send self(), {:sign_up, _params}
    {:ok, socket}
  end

  def handle_info({:sign_up, _params}, socket), do: socket |> sign_up( _params)

  defp sign_up(socket,  _params) do
    changeset = User.changeset(%User{}, _params) |> User.with_password_hash
    changeset = put_change(changeset, :profile_picture, "default_profile.png")
    case Repo.insert changeset do
      {:ok, user} ->
        push socket, "sign_up", %{status: "successfully sign up"}
      {:error, changeset} ->
        push socket, "sign_up", %{status: "failed to sign up"}
    end
    {:noreply, socket}
  end
```

Figure 2: Signup code details



```elixir
def join("search", _params, socket) do
    send self(), {:search, _params}
    {:ok, socket}
end

defp tweets_json(tweet) do
  %{
    text: tweet.text,
    retweet_id: tweet.retweet_id,
    current_user_favorite_id: tweet.current_user_favorite_id,
    current_user_retweet_id: tweet.current_user_retweet_id,
  }
end
def handle_info({:search, _params}, socket), do: socket |> search( _params)

defp search(socket,  _params) do
  IO.inspect _params["query"]
  tweets = (from t in Tweet, where: like(t.text, ^("%#{_params["query"]}%")))
  |> Repo.all() |> Repo.preload(:user)
  IO.inspect tweets
  push socket, "search", %{tweet: %{
    tweets: Enum.map(tweets, &tweets_json/1)
  }}
  {:noreply, socket}
end
```

Figure 3: Search tweets code details

5.1 Implementation of Websocket Interface

Fig. 1 shows the part where we use the Phoenix channel to send and receive message about topics. Phoenix.Transports.WebSocket is the default transport mechanism. And topics in this project includes signup, tweet, retweet, search and subscribe. In the next part, which is the implementation of Json-based API, we will take two examples to show how the channel is working and how the data is transferred to Json.

5.2 Implementation of Json-based API

We have implemented the functionality of register, tweet with mentions and hashtags, subscribe to user's tweets, retweet and search using the Json-based API. Here, we will show some of the code details of implementing the Json-based API.

Take the signup code as example shown in Fig. 2, after we type the code

```
$wsta 'ws://128.227.246.42:3000/socket/websocket' '{"topic":"signup","
    event":"phx_join","payload":{"login": "ABC", "name": "ABC", "
    password": "123", "password_confirmation": "123", "email": "ABC@ufl
    .edu"},"ref":"1"}'
```

in the terminal (on Windows, replace "wsta" with "./wsta.exe"), it passes the login, name, password, password_confirmation and email to the server using websocket transportation. The server will find the "signup" topic and pass the corresponding parameters to it. Then the sign up information is inserted into the database. If insert successfully, it will return a json message saying "signup successful". Otherwise, it would tell us "signup failed".

Another example is searching tweet. Fig. 3 shows the core code of implementing json-based api when searching tweets. When we type the command

```
$wsta 'ws://128.227.246.42:3000/socket/websocket' '{"topic":"search","
    event":"phx_join","payload":{"query": "DOS"},"ref":"1"}'
```

in the terminal, it means that we want to search all the tweets that contains "DOS". The server will find the "search" topic and pass "DOS" to it. After getting all the tweets that satisfy the requirements, the map-formatted tweets are converted to Json-formatted tweets in the function *tweets_json*. And this Json-based API would send the tweets in Json format to the websocket client.

We also list the other WebSocket APIs below:

```
$wsta 'ws://localhost:3000/socket/websocket' '{"topic":"tweet","event
    ":"phx_join","payload":{"login":"KeDOS", "password": "123", "tweet
    ":"Hello #DOS, class is over @keke and @pan"},"ref":"1"}'

$wsta 'ws://localhost:3000/socket/websocket' '{"topic":"retweet","event
    ":"phx_join","payload":{"login": "KeDOS", "password": "123", "
    tweet_id": "1"},"ref":"1"}'

$wsta 'ws://localhost:3000/socket/websocket' '{"topic":"subscribe","
    event":"phx_join","payload":{"login":"KeDOS", "password": "123", "
    follow_id":"1"},"ref":"1"}'
```

6 Conclusion

The whole above is the functionalities that we have implemented in this project, including sign up, send tweets, search, subscribe and retweet in the Phoenix Twitter. We not only implemented the UI where you can refer on the website, but also the websocket and Json-based API where it will return Json-format code and communicate through websocket.