Ernesto Martinez
CS 494 - RFC Document
December 2019.
ernes@pdx.edu

1. Introduction

i. This document refers to the protocol design and description for an Internet Relay Chat. The application is able to login/register users than can send/receive messages through created channels. All the users, channels, and messages are stored in a real time database supported in Firebase.

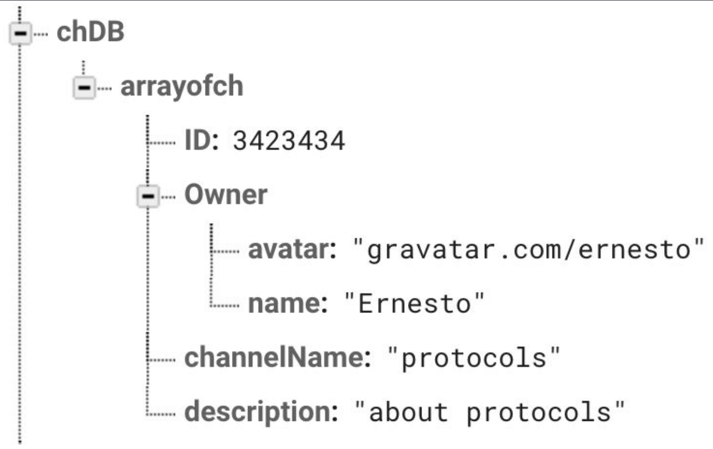
2. Tools

i. Semantic-ui-react for styles in the GUI.
ii. Redux for state management.
iii. ReactJS for FrontEnd Interactions
iv. Firebase for Database Management and Protocol
v. NodeJS for Backend Interactions

3. Database Design

a. Channels

i.
```
channelsRef: firebase.database().ref("channels"),
```



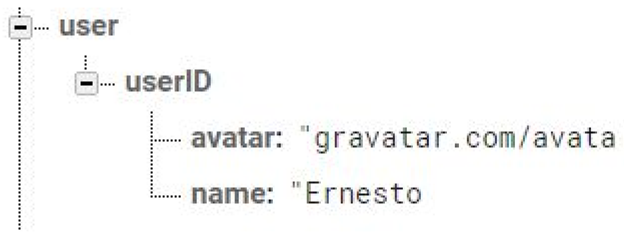
```
chDB
  arrayofch
    ID: 3423434
    Owner
      avatar: "gravatar.com/ernesto"
      name: "Ernesto"
    channelName: "protocols"
    description: "about protocols"
```

ii.

iii. The channels database consist of an array of channels, each channel has a unique ID, the channel name, a brief description, and the user who created the channel. The user has two children representing the name and avatar of that active user.

```
iv.    state = {
v.       activeChannel: "",
vi.      user: this.props.currentUser,
vii.     channels: [],
viii.    channelName: "",
ix.      channelDetails: "",
x.        channelsRef: firebase.database().ref("channels"),
```
xi.

b. Users

i. 
```
user
  userID
    avatar: "gravatar.com/avata
    name: "Ernesto
```

```
ii.       usersRef: firebase.database().ref("users")
```

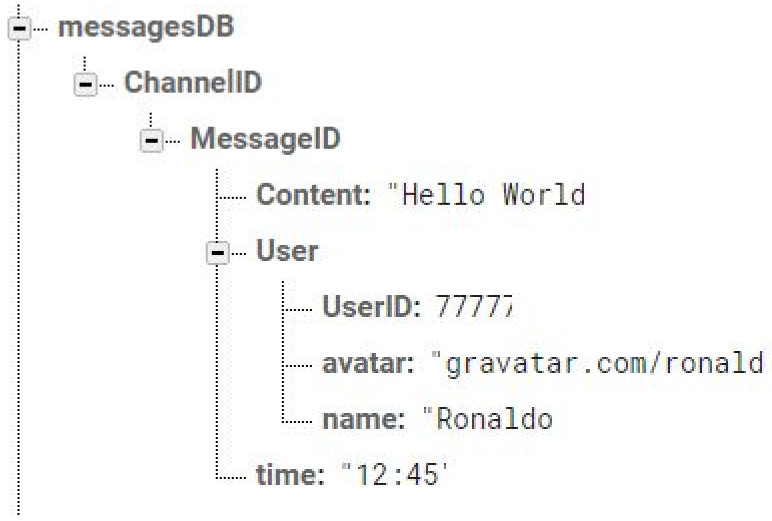
iii. The users database can be accessed by the usersRef reference shown above.

iv. The user database will consist of an array of users ID. Each user ID will have 2 children, corresponding to user name and an optional field for the user avatar.

v. This is an example of how to access the user database to create a new user.

```
vi.     saveUser = createdUser => {
vii.      return
        this.state.usersRef.child(createdUser.user.uid).set({
viii.       name: createdUser.user.displayName,
ix.         avatar: createdUser.user.photoURL
x.        });
xi.     };
```
xii.

c. Messages

i. 
```
messagesDB
  ChannelID
    MessageID
      Content: "Hello World
      User
        UserID: 77777
        avatar: "gravatar.com/ronald
        name: "Ronaldo
      time: "12:45'
```

ii. The message database will have a child with the channel ID which is active at the moment. (More Information about active channel is in “Client can join a channel” feature.
iii. The channelID will have a child with the messageID.
iv. The messageID will have 3 children:
    1. Content or text to be send
    2. Time in which the message was sent. Acquired by:
        a. PICTURE FROM MESSAGE FORM
    3. User information consisting of userID, avatar, and name.
v. This is the process on how to push a message to the database.

```
      messagesRef
        .child(channel.id)
        .push()
        .set(this.createMessage())
  createMessage = () => {
    const message = {
      timestamp: firebase.database.ServerValue.TIMESTAMP,
      user: {
        id: this.state.user.uid,
        name: this.state.user.displayName,
        avatar: this.state.user.photoURL
      },
      content: this.state.message
    };
    return message;
  };
```

4. Features

a. Server Process

i. The server runs in using NodeJs and ReactJS tools.
ii. It runs in localhost.
iii. When the app starts, it opens the login screen and immediately connects to firebase for validation after the client inputs the information.

```
import {
    BrowserRouter as Router,
    Switch,
    Route,
    withRouter
} from "react-router-dom";
```

b. Client can connect to a server

```
      usersRef: firebase.database().ref("users")
```

ii. In method register.js => handleSubmit() , firebase protocol is called with the method **createUserWithEmailAndPassword()** from the firebase API. It reaches to the users database and create a user.

iii. Length of password must be >= 4. It can be modified in the function **register.js/isPasswordValid()** and other restrictions can be added.

iv. In method login.js => handleSubmit() , firebase protocol is called with the method **signInWithWithEmailAndPassword()** from the firebase API. It reaches to the users database and verifies for the user.

c. Client can create a channel

i. Directory Path : ...src/components/SidePanel/Channels.js

ii. Method : addChannel()

iii. addChannel() will reach out to firebase to create a new channel within the channels collection. A channel is created in the state object. A unique key is created for each new channel. It uses the push method to get the key property and gives a unique identifier for every new channel.

d. Client can list all channels

i. Directory Path : ...src/components/SidePanel/Channels.js

ii. Method : DisplayChannels(channels)

iii. Display an array of channels by ID, and list their names.

iv. IDs are acquired from the **channels Database**

```
    displayChannels = channels =>
      channels.length > 0 &&
      channels.map(channel => (
        <Menu.Item
          key={channel.id}
          onClick={() => this.changeChannel(channel)}
          name={channel.name}
    style={{ opacity: 1.0 }}
    active={channel.id === this.state.activeChannel}
   >
    # {channel.name}
   </Menu.Item>
  ));
```

e. Client can join / leave a channel

i. When the user log in, a channel is set active by default.

```
 setFirstChannel = () => {
  const firstChannel = this.state.channels[0];
  if (this.state.firstLoad && this.state.channels.length >
0) {
   this.props.setCurrentChannel(firstChannel);
   this.setActiveChannel(firstChannel);
  }
  this.setState({ firstLoad: false });
 };
```

x. The user is able to switch channels at any time by clicking on one of the channels in the channel list.

```
 changeChannel = channel => {
  this.setActiveChannel(channel);
  this.props.setCurrentChannel(channel);
 };
```

f. Client can list members of a channel

i. Number of Users connected are shown in the top part of the messages interface.

```
     <span>
      {channelName}
     </span>

<Header.Subheader>{numUniqueUsers}</Header.Subheader>
```

g. Multiple Clients can connect to a server

i. By cloning the local host into another window of the browser, many clients can be connected at once. I recommend using incognito window so it can be redirected to the Login Screen every time its cloned.

h. Client can sends messages to a channel

i. In the messages form, at the bottom of the interface. A client can write text and click to send. The message will be sent to the channel that is currently active by the user that is currently signed in.

```
state = {
  message: "",
  channel: this.props.currentChannel,
  user: this.props.currentUser,
  loading: false,
  errors: []
};
```

ix. The message will have the time that it was sent, (Using Firebase timestamp method for this.) More details about the content of the message are in the Messages Database section.

x. The method createMessage is called within the sendMessage function.

xi. There is also a responsive error catch in case the message field is blank.

i. Client can join multiple channels

i. This action can be done by switching channels within the side panel section of the interface.

j. Client can send distinct messages to multiple (selected) rooms

i. This is possible by having two different windows where the same user is logged in. Each window can be in a different channel, therefore the user will be able to interact with multiple channels at the same time.

k. Client can disconnect from a server

```
key: "signout",
text: <span onClick={this.handleSignout}>Sign Out</span>
```

iii.

iv.–x.

```
    handleSignout = () => {
      firebase
        .auth()
        .signOut()
        .then(() => console.log("signed out!"));
    };

```

xi. In the side panel, there is a dropdown menu for the user, there is a clickable action button that permits the client to disconnect from the server. This action calls the signOut method in firebase.

I. Server can disconnect from a clients

i. This action is achieved by using react scripts and ejecting the server.

ii.

```
      "eject": "react-scripts eject"
```

5. Programming Style

i. There is a folder for each separate component.

ii. There is a folder for state actions and another folder for reducers, both these folders interact with the components.

iii. Package.json manages the dependencies of the application which can be installed by running Yarn Install.

iv. .gitignore file will avoid all the node modules installed to be pushed to the repository. Making the cloning way faster.

v. All the necessary imports are always at the top of the files.

vi. There are no weird variable names, therefore the code is readable.

vii. The application follows modern style conventions of Javascript ES6.

6. Extra Credit Features

a. Cloud Server

i. Channels, Messages, and Users are all handled by Firebase in the database. It lets the app to manage its information efficiently. It was the first thing I focused in the thinking process. The databases are key for the application and server communication. Firebase is easy enough to use for this type of feature. I used firebase and react during an internship, and I went for it as the main feature of the Internet Relay Chat.

b. File Upload

i. The file is registered as a new message and the content is stored in the Storage feature of firebase.

7. References

i. Redux Learning : https://medium.com/better-programming/redux-setup-for-your-react-app-d003ec03aedf

ii. Web-Sockets https://medium.com/@dominik.t/what-are-web-sockets-what-about-rest-apis-b9c15fd72aac

iii. Push / Set / Update Methods for Database in Firebase https://firebase.google.com/docs/database/admin/save-data

iv. Server Time https://firebase.google.com/docs/reference/js/firebase.database.ServerValue

v. Register, Login and Sign Out https://firebase.google.com/docs/auth/web/password-auth

vi. Server Environment with Firebase https://firebase.google.com/docs/cloud-messaging/server

vii. For a better understand about creating web chat apps https://www.udemy.com/course/build-a-slack-chat-app-with-react-redux-and-firebase/

viii. More help about Firebase https://www.udemy.com/course/starting-with-firebase/

ix. React + Redux interaction tutorial https://www.udemy.com/course/react-the-complete-guide-incl-redux/