Chapter 15: HTTP Requests

Section 15.1: Using Promises with the fetch API and Redux

Redux is the most common state management library used with React-Native. The following example demonstrates how to use the fetch API and dispatch changes to your applications state reducer using redux-thunk.

```
export const fetchRecipes = (action) => {
  return (dispatch, getState) => {
    fetch('/recipes', {
        method: 'POST',
        headers: {
          'Accept': 'application/json',
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({
          recipeName,
          instructions,
          ingredients
        })
    })
    .then((res) => {
      // If response was successful parse the json and dispatch an update
      if (res.ok) {
        res.json().then((recipe) => {
          dispatch({
            type: 'UPDATE_RECIPE',
            recipe
          });
        });
      } else {
        // response wasn't successful so dispatch an error
        res.json().then((err) => {
          dispatch({
            type: 'ERROR_RECIPE',
            message: err.reason,
            status: err.status
          });
        });
      }
    })
    .catch((err) => {
      // Runs if there is a general JavaScript error.
      dispatch(error('There was a problem with the request.'));
    });
  };
};
```

Section 15.2: HTTP with the fetch API

It should be noted that Fetch *does not support progress callbacks*. See: https://github.com/github/fetch/issues/89.

The alternative is to use XMLHttpRequest https://developer.mozilla.org/en-US/docs/Web/Events/progress.

```
fetch('https://mywebsite.com/mydata.json').then(json => console.log(json));

fetch('/login', {
  method: 'POST',
```

```
  body: form,
  mode: 'cors',
  cache: 'default',
}).then(session => onLogin(session), failure => console.error(failure));
```

More details about fetch can be found at MDN

Section 15.3: Networking with XMLHttpRequest

```
var request = new XMLHttpRequest();
request.onreadystatechange = (e) => {
  if (request.readyState !== 4) {
    return;
  }

  if (request.status === 200) {
    console.log('success', request.responseText);
  } else {
    console.warn('error');
  }
};

request.open('GET', 'https://mywebsite.com/endpoint/');
request.send();
```

Section 15.4: WebSockets

```
var ws = new WebSocket('ws://host.com/path');

ws.onopen = () => {
  // connection opened

  ws.send('something'); // send a message
};

ws.onmessage = (e) => {
  // a message was received
  console.log(e.data);
};

ws.onerror = (e) => {
  // an error occurred
  console.log(e.message);
};

ws.onclose = (e) => {
  // connection closed
  console.log(e.code, e.reason);
};
```

Section 15.5: Http with axios

Configure

For web request you can also use library axios.

It's easy to configure. For this purpose you can create file axios.js for example:

```
import * as axios from 'axios';

var instance = axios.create();
instance.defaults.baseURL = serverURL;
instance.defaults.timeout = 20000;]
//...
//and other options

export { instance as default };
```

and then use it in any file you want.

Requests

To avoid using pattern 'Swiss knife' for every service on your backend you can create separate file with methods for this within folder for integration functionality:

```
import axios from '../axios';
import {
    errorHandling
} from '../common';

const UserService = {
        getCallToAction() {
        return axios.get('api/user/dosomething').then(response => response.data)
            .catch(errorHandling);
    },
}
export default UserService;
```

Testing

There is a special lib for testing axios: axios-mock-adapter.

With this lib you can set to axios any responce you want for testing it. Also you can configure some special errors for your axois'es methods. You can add it to your axios.js file created in prevous step:

```
import MockAdapter from 'axios-mock-adapter';

var mock = new MockAdapter(instance);
mock.onAny().reply(500);
```

for example.

Redux Store

Sometimes you need to add to headers authorize token, that you probably store in your redux store.

In this case you'll need another file, interceptors.js with this function:

```
export function getAuthToken(storeContainer) {
    return config => {
        let store = storeContainer.getState();
        config.headers['Authorization'] = store.user.accessToken;
        return config;
    };
}
```

Next in constructor of your root component you can add this:

```
axios.interceptors.request.use(getAuthToken(this.state.store));
```

and then all your requests will be followed with your authorization token.

As you can see axios is very simple, configurable and useful library for applications based on react-native.

Section 15.6: Web Socket with Socket.io

Install *socket.io-client*

```
npm i socket.io-client --save
```

Import module

```
import SocketIOClient from 'socket.io-client/dist/socket.io.js'
```

Initialize in your constructor

```
constructor(props){
    super(props);
    this.socket = SocketIOClient('http://server:3000');
  }
```

Now in order to use your socket connection properly, you should bind your functions in constructor too. Let's assume that we have to build a simple application, which will send a ping to a server via socket after every 5 seconds (consider this as ping), and then the application will get a reply from the server. To do so, let's first create these two functions:

```
_sendPing(){
    //emit a dong message to socket server
    socket.emit('ding');
}

_getReply(data){
    //get reply from socket server, log it to console
    console.log('Reply from server:' + data);
}
```

Now, we need to bind these two functions in our constructor:

```
constructor(props){
    super(props);
    this.socket = SocketIOClient('http://server:3000');

    //bind the functions
    this._sendPing = this._sendPing.bind(this);
    this._getReply = this._getReply.bind(this);
}
```

After that, we also need to link _getReply function with the socket in order to receive the message from the socket server. To do this we need to attach our _getReply function with socket object. Add the following line to our constructor:

```
this.socket.on('dong', this._getReply);
```

Now, whenever socket server emits with the 'dong' your application will able to receive it.