Using Promises - ECMAScript 6 Tutorial

Promises have replaced callback functions as the preferred programming style for handling asynchronous calls. A promise is a holder for a result (or an error) that will become available in the future (when the async call returns). Promises have been available in JavaScript through third-party libraries (for example, jQuery and q). ECMAScript 6 adds built-in support for promises to JavaScript.

In this unit, you create a simple application called ratefinder that returns a list of available mortgage rates.

Part 1: Use a Promise

To illustrate the use of promises in this example, you use the new `fetch()` function. At the time of this writing, `fetch()` is available in the latest version of Chrome, Firefox, and Opera, but not in IE and Safari. You can check the current availability of `fetch()` here. You can read more about `fetch()` here.

1. Create a file named `ratefinder.html` in the `es6-tutorial` directory. implemented the file as follows:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
</head>
<body>
    <table id="rates"></table>
    <script src="build/ratefinder.bundle.js"></script>
</body>
</html>
```

2. Create a file named `ratefinder.js` in the `es6-tutorial/js` directory. implemented the file as follows:

```
let url = "rates.json";

fetch(url)
    .then(response => response.json())
    .then(rates => {
      let html = '';
      rates.forEach(rate => html += `<tr><td>${rate.name}</td><td>${rate.years}</td>
<td>${rate.rate}%</td></tr>`);
      document.getElementById("rates").innerHTML = html;
    })
    .catch(e => console.log(e));
```

To keep things simple, this code uses a static data file: rates.json. The application would work the same way with a URL pointing to a remote service.

3. Open `webpack.config.js` in your code editor. In `module.exports`, modify the entry and output items as follows:

   ```
   entry: {
       app: './js/main.js',
       ratefinder: './js/ratefinder.js'
   },
   output: {
       path: path.resolve(__dirname, 'build'),
       filename: '[name].bundle.js'
   },
   ```

   The **webpack** script will now compile two applications: **main.js** and **ratefinder.js**. It will create two compiled files based on the entry name: **app.bundle.js** and **ratefinder.bundle.js**.
4. On the command line, type the following command to rebuild the application:

   ```
   npm run webpack
   ```

Part 2: Create a Promise

Most of the time, all you'll have to do is use promises returned by built-in or third-party APIs. Sometimes, you may have to create your own promises as well. In this section you create a mock data service to familiarize yourself with the process of creating ECMAScript 6 promises. The mock data service uses an asynchronous API so that it can replace an actual asynchronous data service for test or other purpose.

1. Create a new file named `rate-service-mock.js` in the `js` directory.

2. In rate-service-mock.js.js, define a `rates` variable with some sample data:

   ```
   let rates = [
       {
           "name": "30 years fixed",
           "rate": "13",
           "years": "30"
       },
       {
           "name": "20 years fixed",
           "rate": "2.8",
           "years": "20"
       }
   ];
   ```

3. Define a `findAll()` function implemented as follows:

   ```
   export let findAll = () => new Promise((resolve, reject) => {
       if (rates) {
   ```

```
        resolve(rates);
    } else {
        reject("No rates");
    }
});
```

4. Open `ratefinder.js`. Change the implementation as follows:

```
import * as service from './rate-service-mock';

service.findAll()
    .then(rates => {
        let html = '';
        rates.forEach(rate => html += `<tr><td>${rate.name}</td><td>${rate.years}
</td><td>${rate.rate}%</td></tr>`);
        document.getElementById("rates").innerHTML = html;
    })
    .catch(e => console.log(e));
```

5. On the command line, type the following command to rebuild the application:

```
npm run webpack
```

6. Open a browser, access http://localhost:8080/ratefinder.html.