Introduction to programming Using React 16.8

1 Getting started

React is a software library to create browser based and mobile applications. The library is used to develop views that can be used in PC, tablet or mobile environment. Initially React was developed in FaceBook at 2013 by Jordan Walke. In Finland the usage of React is very wide (https://www.itewiki.fi/yritykset/react). In this chapter the tools to use React are presented.

The React apps are started by entering “npm start” from terminal/console in the project folder. If there is an error, try “npm install” to install the missing components. Also you can check the error and install some component that is not listed in package.json like “npm install –save missing_component_name”.

1.1 Installing Node.JS

During the development of React application the solution is tested on a local PC. A specific tool called npm (node package manager) is used for various activities. This is the reason why Node.JS is needed, both to run the server for testing during development and provide the tool npm (and npx) to support the development. There is an option to use a web based service like https://codepen.io/ but it serves quite limited features.

Download and install the Node.JS version depending your OS https://nodejs.org/en/. After installation one needs to check that the commands npm and npx can be found on the PATH environment variable. To be able to run the command in any system folder update the PATH environmental variable(This is a maven example but does the same for npm). Command npm is used to automatically download and install (=manage packages) for various libraries. The first library needed to be downloaded and installed is a library that will create react application folder and files structure so that one doesn’t need to start from scratch while developing an app.

1.2 Installing create-react-app

Open terminal (cmd on Windows, terminal for Ubuntu) and use npm to install react templates

npm i -g create-react-app@1.5.2

The above command ensures that the actual 1.5.2 -version of the template is used

1.3 Using terminal

The terminal is used to run npm commands like to install new components, start the application, test and deploy it. The terminal can be used as an separate app besides the IDE or inside the IDE. The Windows OS command to start terminal is cmd.

1.4 Visual Studio Code

You may use any (UTF8) text editor but basically you need a professional IDE like Eclipse, Visual Studio Code or IntelliJ. Most of the developers use Visual Studio Code so to install it first download https://code.visualstudio.com/download

Then install the application and run it for the first time. Normally, when we create the project (using npm create-react-app test) we start the editor by

code . (in the project folder)

to start the Visual Studio Code in the correct folder.

2 Creating the very first React app

In this example we create a small React application with the same idea that was in the pre-task. The application will show two drop-downs, the first one lists the names of months and the latter one list the dates on the month. Ones a month is selected, the latter list is edited to have only the right number of days available.

To create a React application from template use the following command in projects -folder

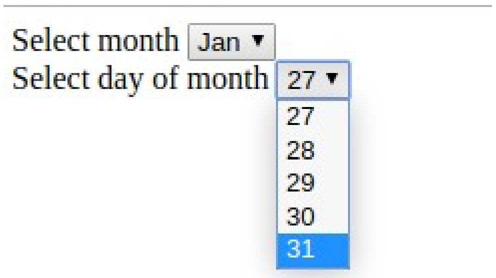
```
npx create-react-app month
```

The last word is the name of the application to be built. To edit the code you need to move to that folder and start your favorite editor, like.

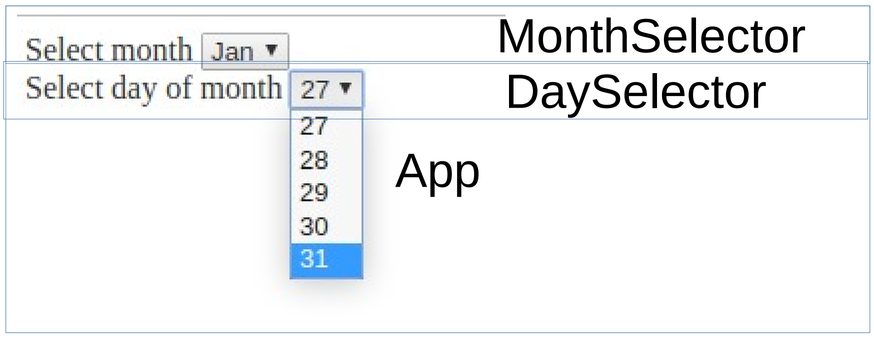
```
cd month
code .
```

2.1 Plan the app

The very first thing you need to do it to make the user interface mockup and based on that plan later decide the components to be developed. The following presents an idea what we are building. An app to select the month and then the right day of month. The app limits selection of non-valid days (like Feb 30).



While developing React application, we create new components. Components that render (=draw) themselves to the user interface. There are two different types of compnonents: classes and functional components. One should prefer using functional components. Component can include other components. In the above example we could initially create a App -component that will draw the whole page. Then we could add two new components; MonthSelector and DaySelector. The new components are usually created to a folder named components. There is no realtion between the months and days, every month has 31 days initially.

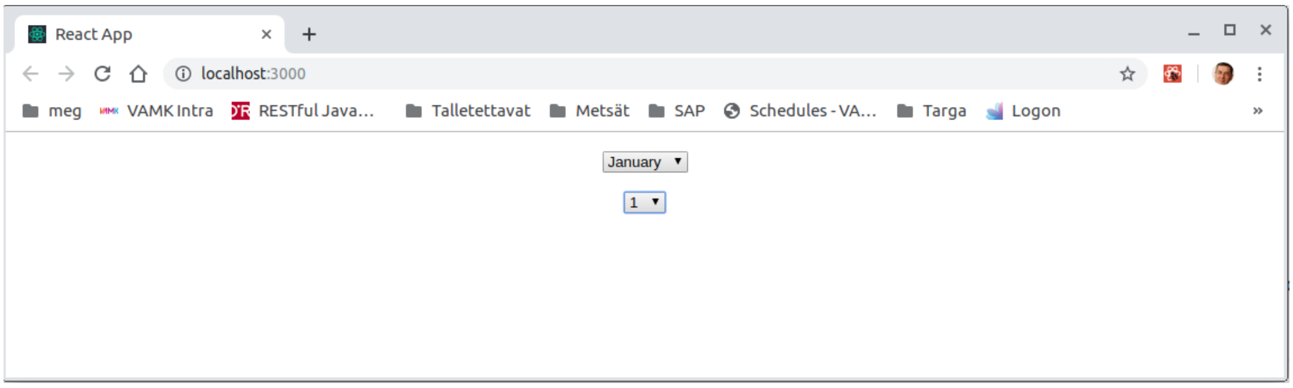


2.2 Create a first version

Initially we could just use the App.js -file, create two arrays, one for the data of the months and one for the days of the months. The initial version just introduces us to the use of arrays, initialize the data outside the render()-method and then use the created variables to draw the user interface.

```
import React from "react";
import "./App.css";

function App() {
 let months = [
   { name: "January", days: 31 },
   { name: "February", days: 28 },
   { name: "March", days: 31 }
 ];

 var monthOptions = [];
 for (const [index, value] of months.entries()) {
  monthOptions.push(<option key={index}>{value.name}</option>);
 }

 var dayOptions = [];
 for (let i = 1; i < 32; i++) {
  dayOptions.push(<option key={i}>{i}</option>);
  }

  return (
    <div className="App">
      <p>
        <select name="month">{monthOptions}</select>
      </p>
      <select name="day">{dayOptions}</select>
    </div>
  );
}

export default App;
```

The initial result looks like



The first version has many limitations. There is not any link between the select-elements, selecting the month does not trigger the number of days in days. The labels are missing, the page is poorly named by the template etc. But we managed to create the array, loop the arrays and use the result to build the two select-elements.

2.3 Create components

Next we will move the month and day select -elements to their own components. The idea is that when a month component is rendered, it renders also the day component with the argument number of days needed. So if we change the month to February the day-component should be re-rendered with the correct max number of days available.

The React is planned to support this kind of responsive behavior. We just need to split the code from App to components (earlier planned) MonthSelector and DaySelector.

The React components can be either classes and functional components. Most of the existing React code and examples are class -based but you should use functional components whenever it is possible.

The functional component is initialized as follows (in folder src with name ComponentName.js):

```
import React from "react";

const ComponentName = () => {
        return(
                <div>Hello world!</div>
        )
}

export default ComponentName;
```

Note that ones part of the implementation is moved to another file, that file need to be imported. To import the code, it must be exported like in the above example. The functional component can also be created with command **function ComponentName(){…}**. The arrow function [=()=>] is most modern way. Check React videos 13 and 20 to understand better the arrow functions. The App application below shows an example how the component is imported and used. While using the component both style `<ComponentName />` and `<ComponentName>This is the component</ComponentName>` are valid options. Usually the components are created to a sub folder called "components" but here we for simplicity just create the component to the same folder where the App.js locates (=src).

```
import React from "react";
import "./App.css";
import ComponentName from "./ComponentName";

function App() {
  return (
    <div className="App">
      <ComponentName />
    </div>
  );
}
export default App;
```

Next we will plan how to split the original mockup implementation into components. Basically it means that we create new files and move and update part of the code there. How the components are name and where should they locate is based on the application design. Normally the components are named with capitalized first letter and using gamel notation (ThisIsHow). Also normally the folder components is added to the src -folder of the projects. The components are then stored there. The suffix of the component should be as .js (or .tsx if Typescript is used).
The component can be created as follows

```
import React,{Component} from ‘react’;
class MyComponent extends Component{
   //code here
}
```

or

```
import React from 'react';
function MyComponent {
   //code here
}
```

or

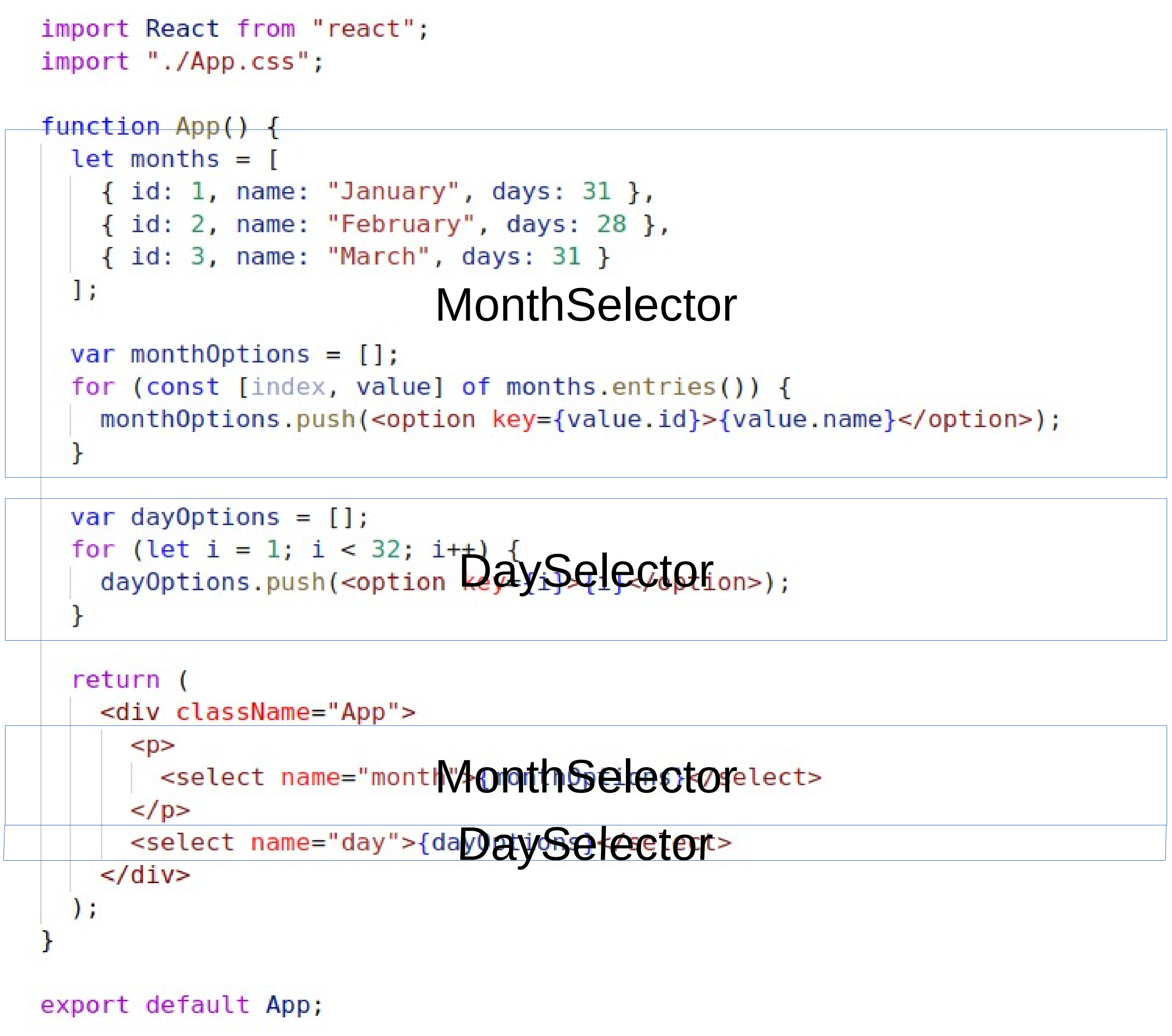
```
import React from "react";
const MyComponent = () => {
  //code here
};
```

The last two examples above are simular functional components. Because React components usually are used to render (=show) some elements in browser the return() -method is a must to have to render the returned elements.

```
import React from "react";
const MyComponent = () => {
  return(
    <div>
      <p>Hello world!</p>
    </div>
  );
};
```

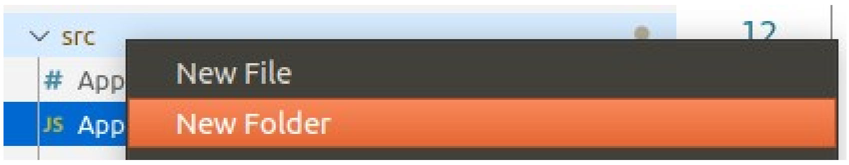
The return function mixes JavaScript and HTML-like code and has limitations because of that. Using normal if/for/while might cause difficulties and create the code extremely complex. That is why in many cases some of the HTML elements are created in a variable, like the li -elements for the unordered list in the following example.

```
import React from "react";
const MyComponent = () => {
  const names = ["Tauno", "Jorma", "Juha", "Antti"];
  const ulNames = names.map((name, index) => {
    return <li key={index}>{name}</li>;
  });
  return (
    <div>
    <p>Hello world!</p>
    <p>
     <ul>{ulNames}</ul>
    </p>
   </div>
 );
};

export default MyComponent;
```



```
import React from "react";
import "./App.css";

function App() {
  let months = [
    { id: 1, name: "January", days: 31 },
    { id: 2, name: "February", days: 28 },
    { id: 3, name: "March", days: 31 }
  ];

  var monthOptions = [];
  for (const [index, value] of months.entries()) {
    monthOptions.push(<option key={value.id}>{value.name}</option>);
  }

  var dayOptions = [];
  for (let i = 1; i < 32; i++) {
    dayOptions.push(<option key={i}>{i}</option>);
  }

  return (
    <div className="App">
      <p>
        <select name="month">{monthOptions}</select>
      </p>
      <select name="day">{dayOptions}</select>
    </div>
  );
}

export default App;
```

The code from previous App-component will be splitted into three parts. The role for App will be to render MonthSelector -component and the MonthSelector will render the months and call the DaySelector to render itself. Ones the month is changed in user interface, the MonthSelector will re-render itself and the number of the days.



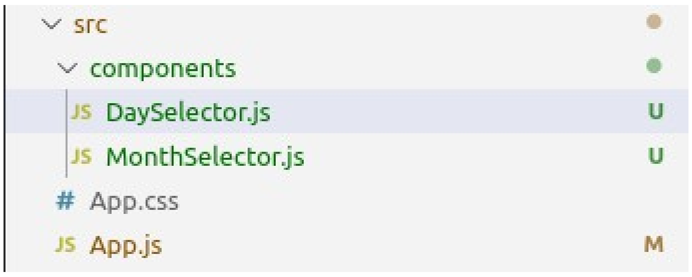
To follow good design pattern a new sub-folder is created to and the new components are created there. After creating the folder and files (components, MonthSelector.js and DaySelector.js) we will move the code from App to the components as follows:

```
import React from "react";
import "./App.css";
import MonthSelector from "./
components/MonthSelector";

function App() {


  return (
    <div className="App">
      <MonthSelector />
    </div>
  );
}

export default App;
```

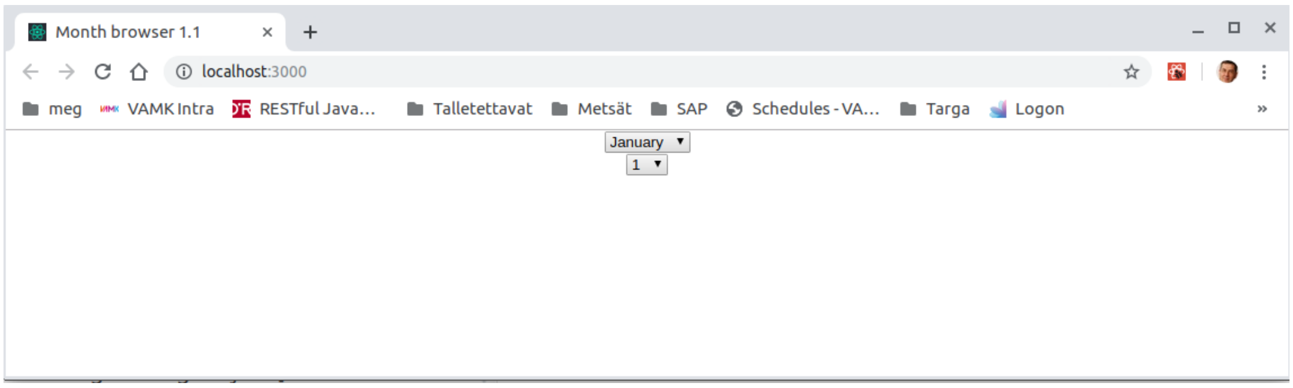
```
import React from "react";
import DaySelector from "./DaySelector";

const MonthSelector = () => {
  let months = [
    { id: 1, name: "January", days: 31 },
    { id: 2, name: "February", days: 28 },
    { id: 3, name: "March", days: 31 }
  ];

  var monthOptions = [];
  for (const [index, value] of months.entrie
s()) {
    monthOptions.push(<option key={valu
e.id}>{value.name}</option>);
  }

  return (
    <div>
      <select name="month">{monthOptio
ns}</select>
      <DaySelector />
    </div>
  );
};

export default MonthSelector;
```

```
import React from "react";

const DaySelector = () => {
  var dayOptions = [];
  for (let i = 1; i < 32; i++) {
    dayOptions.push(<option key={i}>{i}</
option>);
  }
  return (
    <div>
      <select name="day">{dayOptions}</select>
    </div>
  );
};

export default DaySelector;
```

The folder structure should look like



The result looks the same, still we miss the activity to update the number of days while changing the month.



2.4 Props between the components and useState

The easiest way to pass arguments between the components is to pass properties while callin the components. The properties are passed as attributes is the element. Thus the call <DaySelector/> does not pass are arguments but <DaySelector dayInMonth=31/> passes.

The idea is that whenever the month changes the DaySelector is automatically called with the right month to list the days. To catch the change in the MonthSelector select -element we need to use onChange-listener.
To render the list of days after every month change the concept state must be used. React has a easy way of using useState concept. We just define a const table with two variable: the first one is the variable having the state, the next one is the variable we use to change the state (see the yellow highlighted code). Note also that useState need to be imported.

```
import React from "react";
import "./App.css";
import MonthSelector from "./components/MonthSelector";

function App() {
 return (
   <div className="App">
    <MonthSelector />
   </div>
 );
}

export default App;
```

```
import React from "react";
import DaySelector from "./DaySelector";
import { useState } from "react";
import "./components.css";

const MonthSelector = () => {
 const [days, setDays] = useState(31);
 let months = [
   { id: 1, name: "January", days: 31 },
   { id: 2, name: "February", days: 28 },
   { id: 3, name: "March", days: 31 },
   { id: 4, name: "April", days: 30 }
 ];

 var monthOptions = [];
 for (const [index, value] of months.entries()) {
   monthOptions.push(
    <option value={value.days} key={value.id}>
      {value.name}
    </option>
   );
 }

 function monthChangeHandler(event) {
   //console.log("works " + event.target.value);
   setDays(event.target.value);
 }

 return (
   <div>
```

```
import React from "react";

const DaySelector = props => {
 var dayOptions = [];
 for (let i = 1; i <= props.daysInMonth; i++) {
   dayOptions.push(<option key={i}>{i}</option>);
 }
 return (
   <div>
    <label>Day</label>
    <select name="day">{dayOptions}</select>
   </div>
 );
};

export default DaySelector;
```

```
      <form>
        <label>Month</label>
        <select name="month" onChange={e
=> monthChangeHandler(e)}>
          {monthOptions}
        </select>
        <DaySelector daysInMonth={days} />
      </form>
    </div>
  );
};

export default MonthSelector;
```

2.5 Fine tuning the outlook

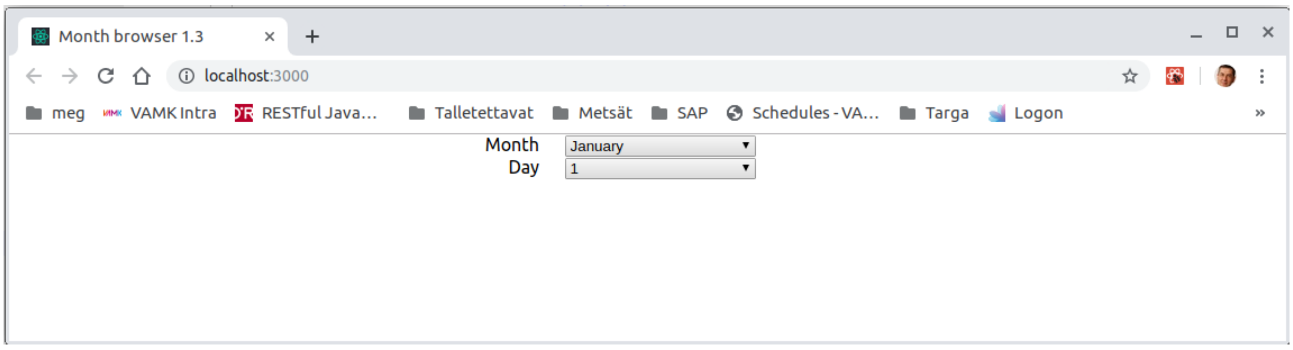
Finally we can add some styling to the created components to layout them in more convenient way. Let’s add a new stylesheet file to components -folder called components.css and edit it like:

```
form {
  width: 50%;
  margin: 0 auto;
}

label,
select {
  /* in order to define widths */
  display: inline-block;
}

label {
  width: 30%;
  /* positions the label text beside the input */
  text-align: right;
}

label + select {
  width: 30%;
  /* large margin-right to force the next element to the new-line
     and margin-left to create a gutter between the label and input */
  margin: 0 30% 0 4%;
}

/* only the submit button is matched by this selector,
 but to be sure you could use an id or class for that button */
input + input {
  float: right;
}
```

The update the MonthSelector and DaySelector functional components to use the stylesheet

```
import React from "react";
import "./components.css";

const DaySelector = props => {
```

After the changes the application looks like



Month January
Day 1

2.6 Future concerns

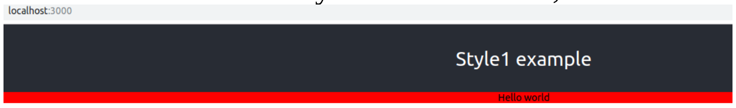
As usual out small app has it own limitations. To keep the example short we listed only four first months, list the rest months there as well.
In case of leap year February should have 29 days. So to make the app more complete we should ask the user the year (in drop down listing last/next 100 years) and in case the year number divided by four gives zero as remainder is is a leap year. But there is an exception if the year number divided by 100 gives no remainder it is not leap year but if the year divided by 100 AND 400 gives zero reminder it is leap year. So add a year drop down and fix the bug. It means also that you need to move the months array as useState data and in case of leap year, change the February days to 29.

3 Styling the app

Initally the react app imports the styles from App.js (if used) or index.css. To overwrite these style setting we can add new classes to these files or use embedded or inline styles. It is important to notice that the **name for class style in React must be className="..."**

3.1 Inline style

Inline style is useful if we want to specify the style only in one place and never reuse it. Just create a constant variable and specify the style as the value of it. To use the style, place the style -attribute to the element where the style should be used, like the div in the example below.



Style1 example

Hello world

```
import React from "react";
import "./App.css";

function App() {
 const style = {
   backgroundColor: "red"
 };
 return (
   <div className="App">
```

```
      <header className="App-header">Style1 example</header>
      <div style={style}>Hello world</div>
    </div>
  );
}

export default App;
```

We can use as many variables as needed to specify the styles for various elements

4 Testing React App

4.1 Needed components

There are a number of testing tools, here we use Enzyme and Jest. The libraries needed for testing need to be installed:

npm install --save enzyme

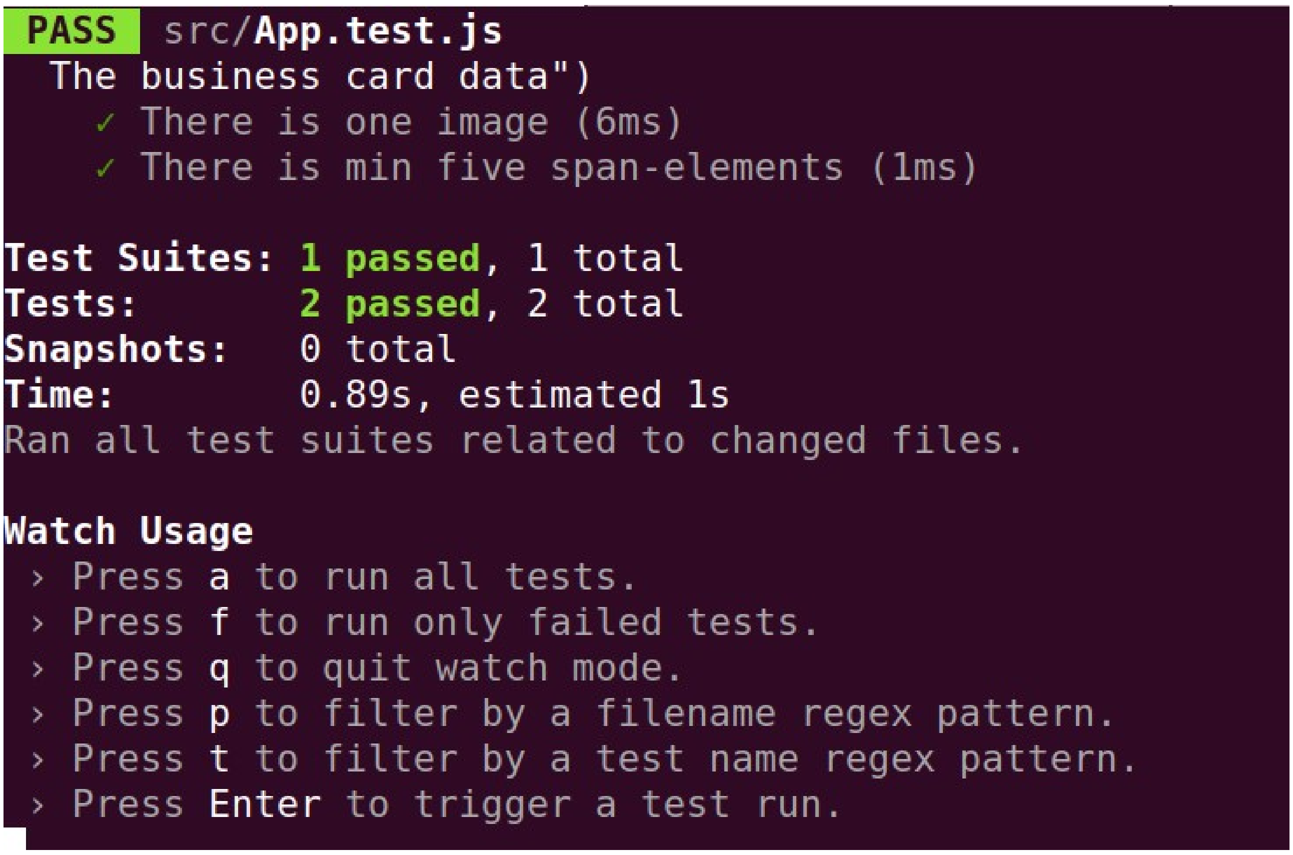
npm install --save enzyme-adapter-react-16

(npm install --save react-test-renderer) not needed

The create-react-app -template already created a test case. The test case is named as Component**.test**.js where the Component is the actual component to be tested. In the example below we test that the table has eigth table rows (tr). Here we test App.js -component so the test file should be named App.test.js

```
import React from "react";
import App from "./App";
import { configure, shallow, assert } from "enzyme";
import Adapter from "enzyme-adapter-react-16";

configure({ adapter: new Adapter() });
describe('Testing table")', () => {
  it("Table has eight tr:s", () => {
    const wrapper = shallow(<App />);
    expect(wrapper.find("tr")).toHaveLength(8);
  });
});
```

The tests are carried out in terminal/console window by command npm test



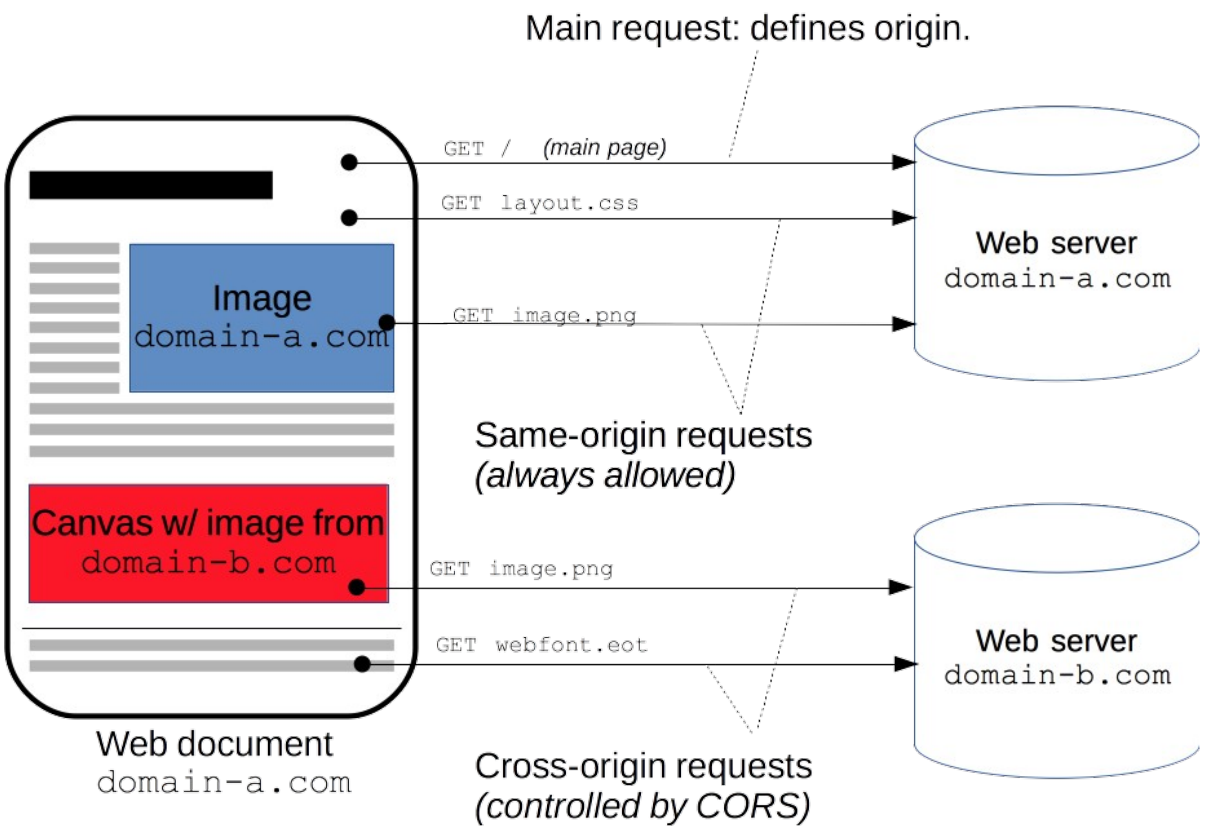
```
 PASS  src/App.test.js
  The business card data")
    ✓ There is one image (6ms)
    ✓ There is min five span-elements (1ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.89s, estimated 1s
Ran all test suites related to changed files.

Watch Usage
 › Press a to run all tests.
 › Press f to run only failed tests.
 › Press q to quit watch mode.
 › Press p to filter by a filename regex pattern.
 › Press t to filter by a test name regex pattern.
 › Press Enter to trigger a test run.
```

5 Fetching the data from the back end

In this chapter we modify the month-day example so that instead reading the month data from a variable in the code we fetch the month data from the back end. This enables us to create easily multi-language application. Based on the used language, we fetch the URL that presents the month names right.

5.1 Avoiding CORS

Ones the data is requested from server it very common to face cross-origin HTTP request problem(CORS). The browser normally expects all data in one page to come from the same server. This is called same origin policy. Sometimes (=usually) we load the application from application server (like http://localhost:3000) and the data from a back end server like Firebase, a Java Spring Boot, ASP.NET Core or Python Django back end developed by the back end team.

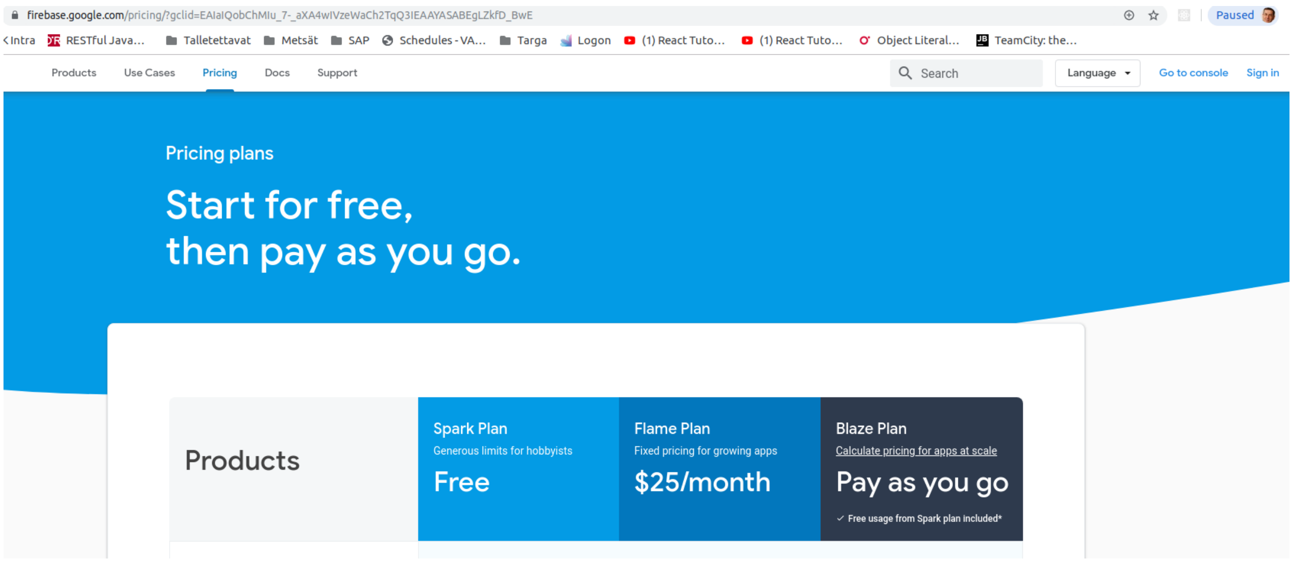

Main request: defines origin.
GET / (main page)
GET layout.css
GET image.png
Image domain-a.com
Canvas w/ image from domain-b.com
Web server domain-a.com
Same-origin requests (always allowed)
GET image.png
GET webfont.eot
Web server domain-b.com
Web document domain-a.com
Cross-origin requests (controlled by CORS)

https://mdn.mozillademos.org/files/14295/CORS_principle.png

The response from backend server that provides data must include CORS headers. To avoid this problem we use initially a serverless back end developed by Google. In case you create your own back end for the data the CORS header information must be correct in the service settings (`@CrossOrigin` -annotation for Java Spring Boot, `services.AddCors` for ASP.NET Core or `CORS_ORIGIN_ALLOW_ALL = True` in Python Django).

5.2 Preparing the backend

To acces some data from server in JSON-format Firebase is very convenient option. Using Firebase we just define a real time database and then we can send or retrieve data from data database. Firebase also has *Spark Plan* free prizing strategy.

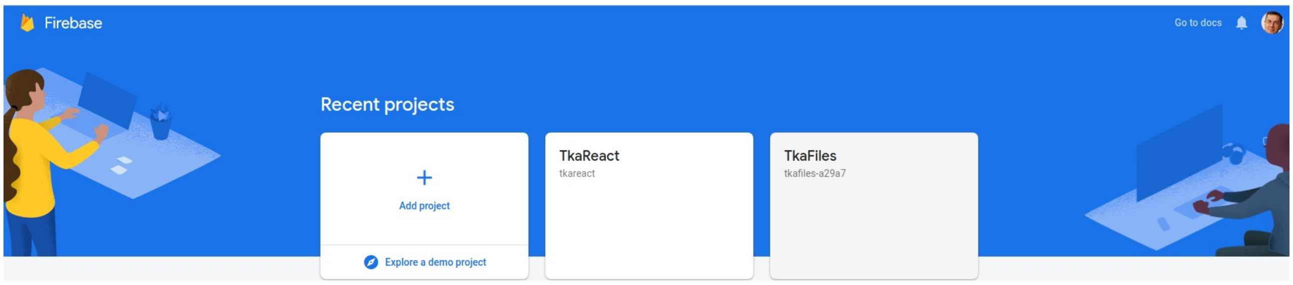

firebase.google.com/pricing/?gclid=EAIaIQobChMIu_7-_aXA4wIVzeWaCh2TqQ3IEAAYASABEgLZkfD_BwE
Products
Use Cases
Pricing
Docs
Support
Search
Language
Go to console
Sign in

Pricing plans

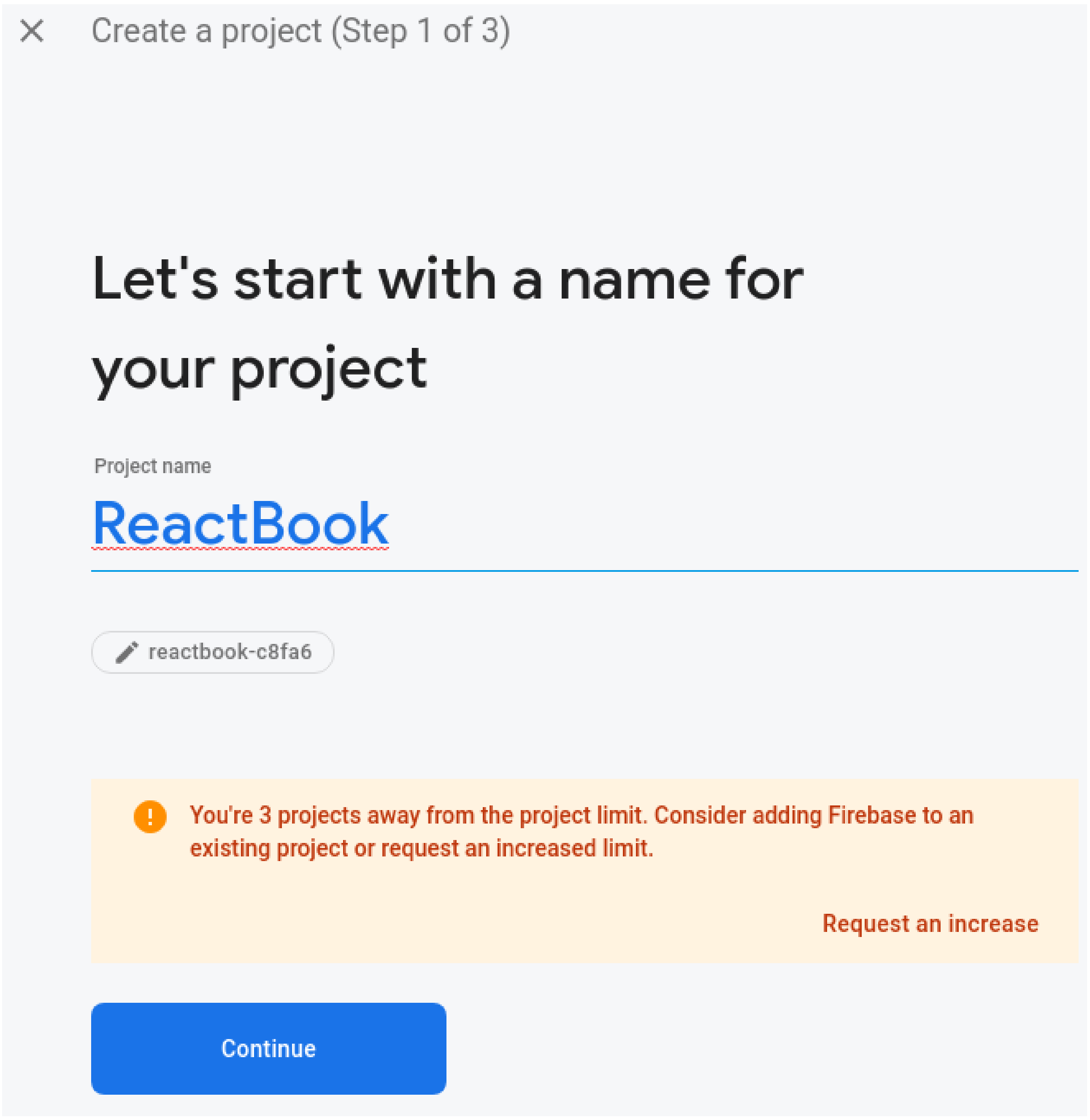
Start for free,
then pay as you go.

Products	Spark Plan Generous limits for hobbyists Free	Flame Plan Fixed pricing for growing apps $25/month	Blaze Plan Calculate pricing for apps at scale Pay as you go ✓ Free usage from Spark plan included*

After logging in with Google account Go to Console link is used to access the console.



Add a project, name it, accept following steps and wait until the project is ready. Then select Database, Create database, to access it easily use Start in test mode, select the server location anw wait for Firestore to initialize. Ones the database is ready, start your first collection. After naming the collection decide which data fields and types are needed for it initially. One needs also to identify the collection or use autoid to do it. Finally change the database type to realtime. Now you may access the database using https and CRUD-commands for the given uri like https://reactbook-c8fa6.firebaseio.com. Select the Rules tab and enable reading and writing to the database by changing true to the values.



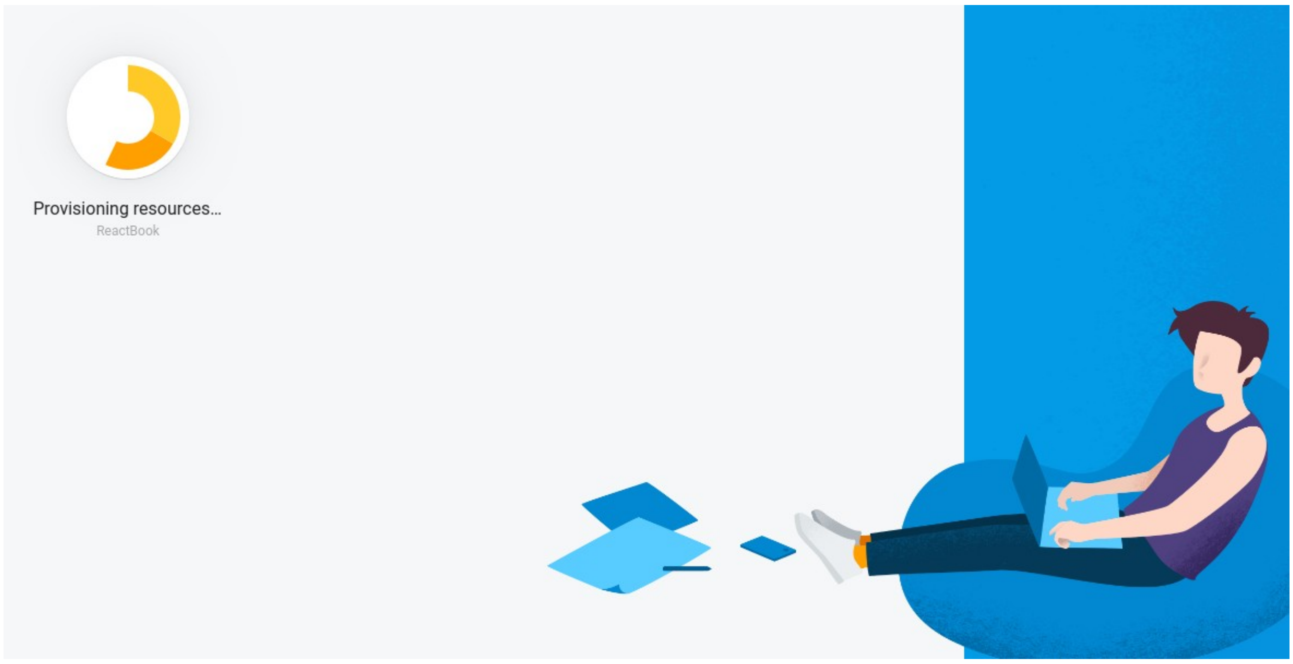
Let's start with a name for your project

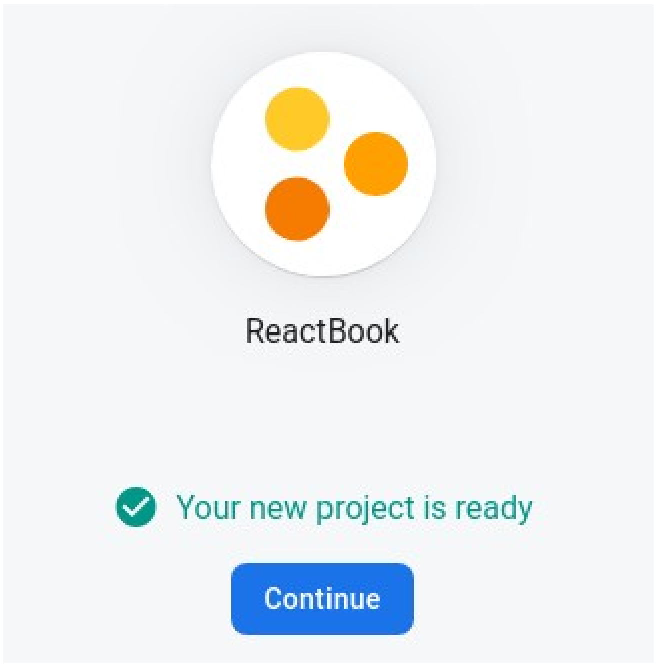
Project name

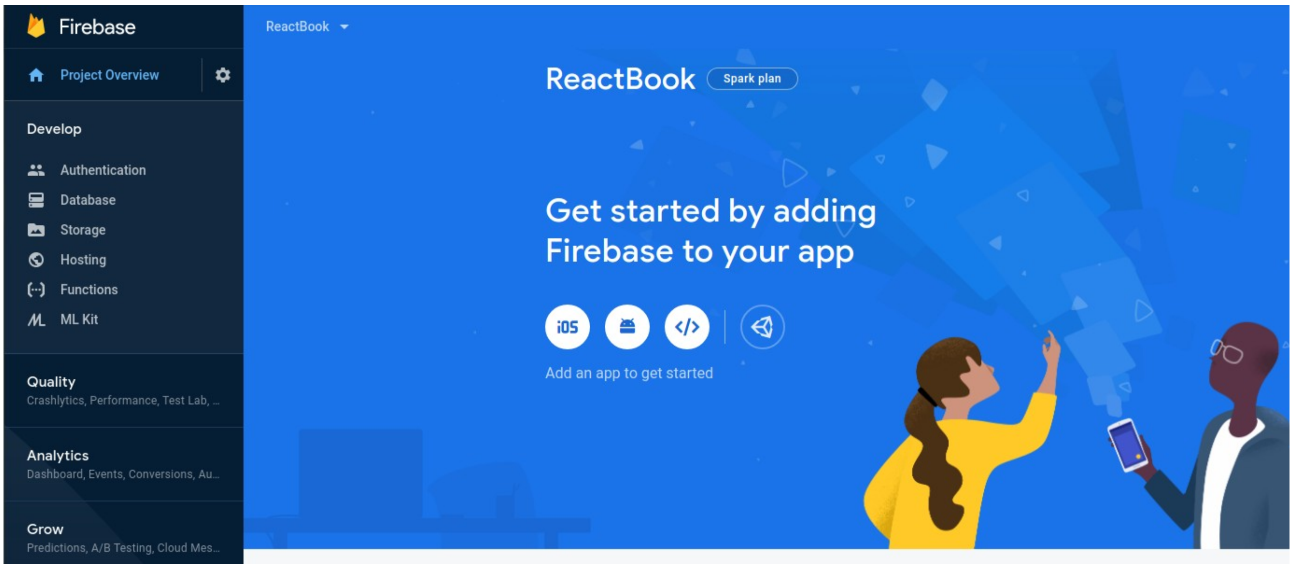
ReactBook

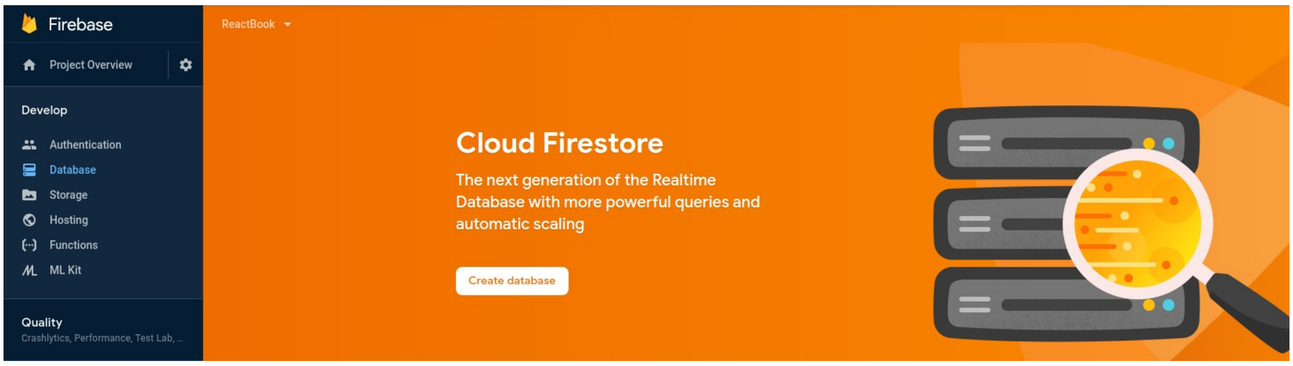
reactbook-c8fa6

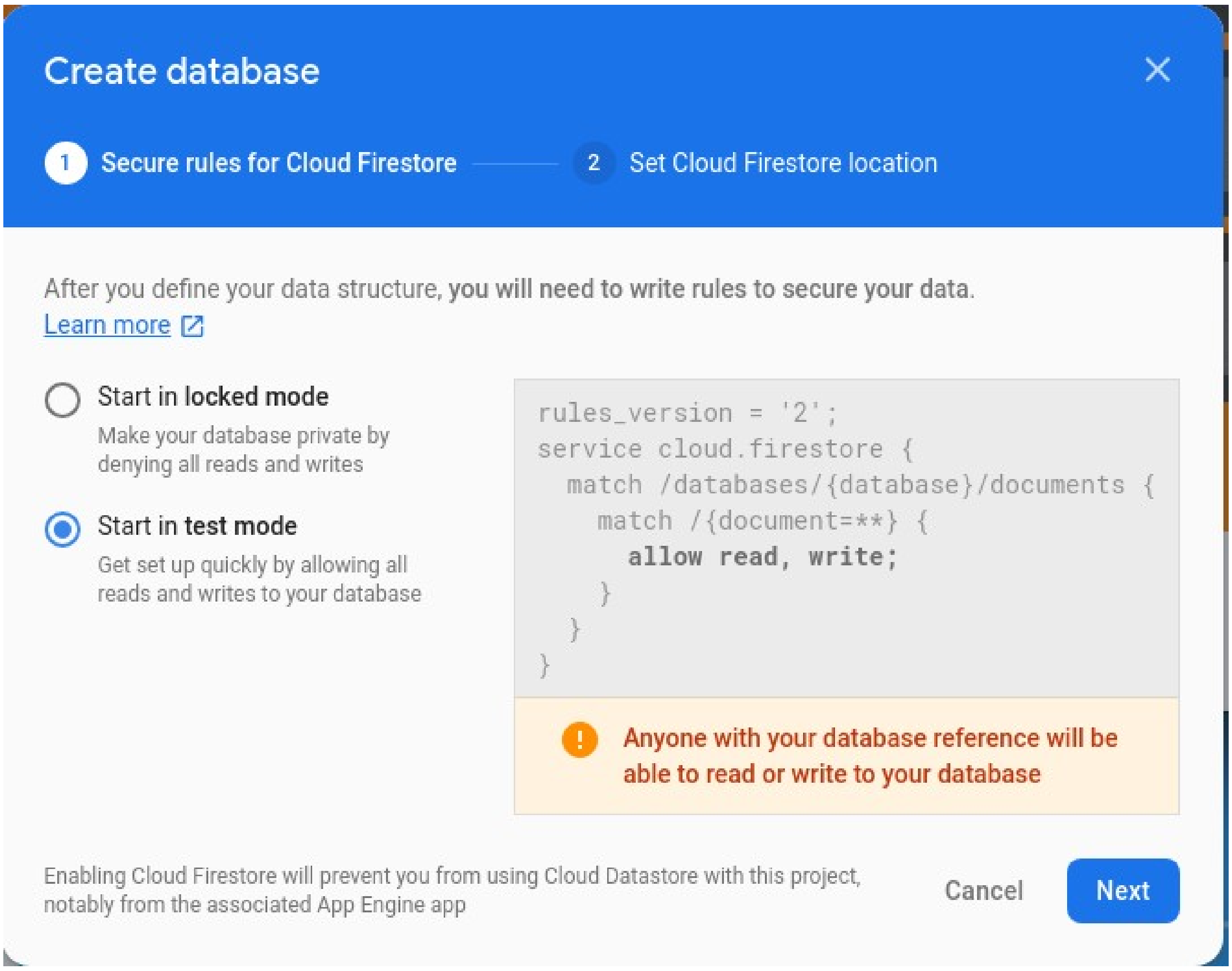
You're 3 projects away from the project limit. Consider adding Firebase to an existing project or request an increased limit.


Provisioning resources...
ReactBook


ReactBook
Your new project is ready
Continue


Firebase
Project Overview
Develop
Authentication
Database
Storage
Hosting
Functions
ML Kit
Quality
Crashlytics, Performance, Test Lab, ...
Analytics
Dashboard, Events, Conversions, Au...
Grow
Predictions, A/B Testing, Cloud Mes...
ReactBook
ReactBook
Spark plan
Get started by adding Firebase to your app
iOS
Add an app to get started


Firebase
Project Overview
Develop
Authentication
Database
Storage
Hosting
Functions
ML Kit
Quality
Crashlytics, Performance, Test Lab, ...
ReactBook
Cloud Firestore
The next generation of the Realtime Database with more powerful queries and automatic scaling
Create database


Create database

1 Secure rules for Cloud Firestore — 2 Set Cloud Firestore location

After you define your data structure, **you will need to write rules to secure your data.**
Learn more

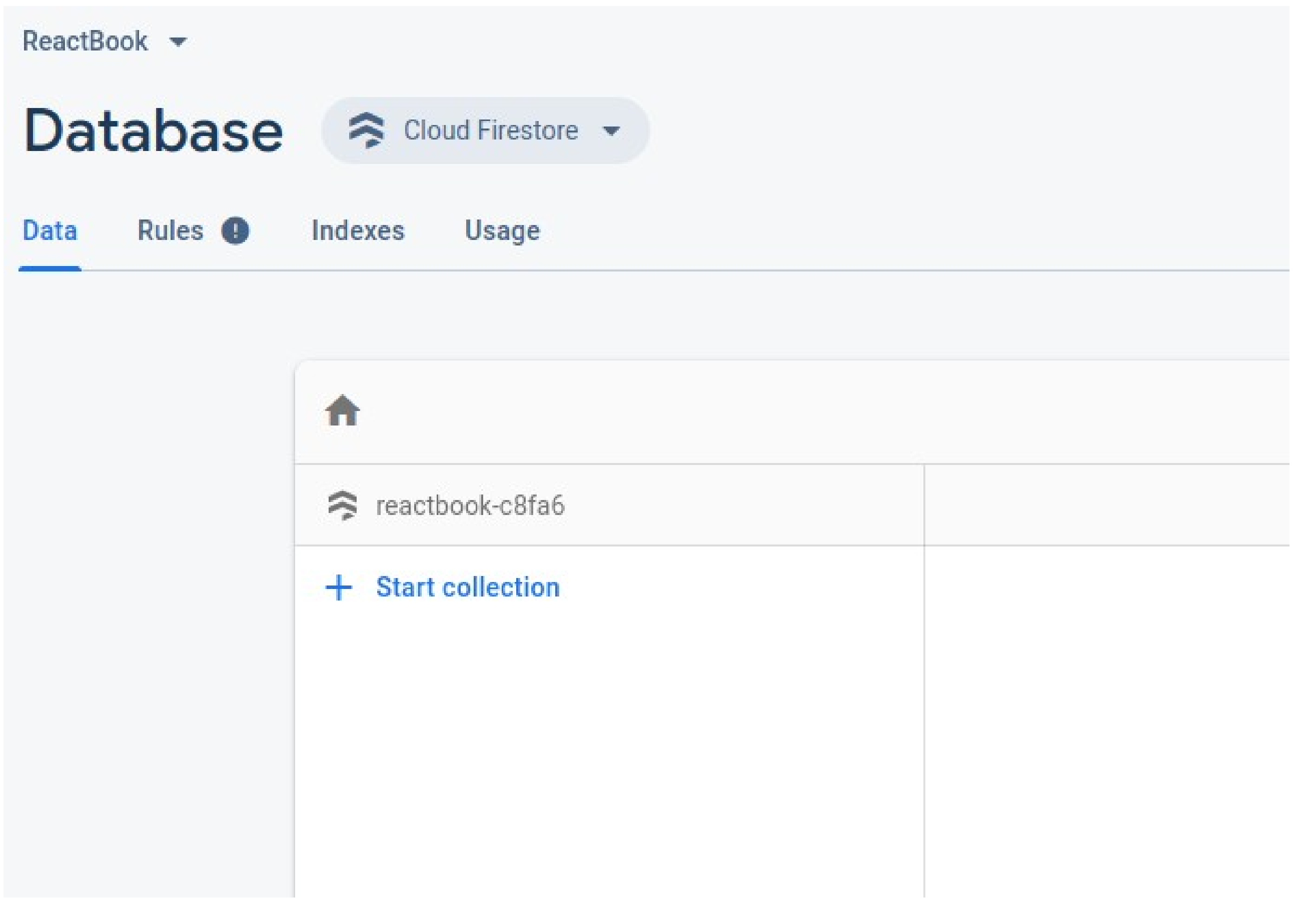
Start in **locked mode**
Make your database private by denying all reads and writes

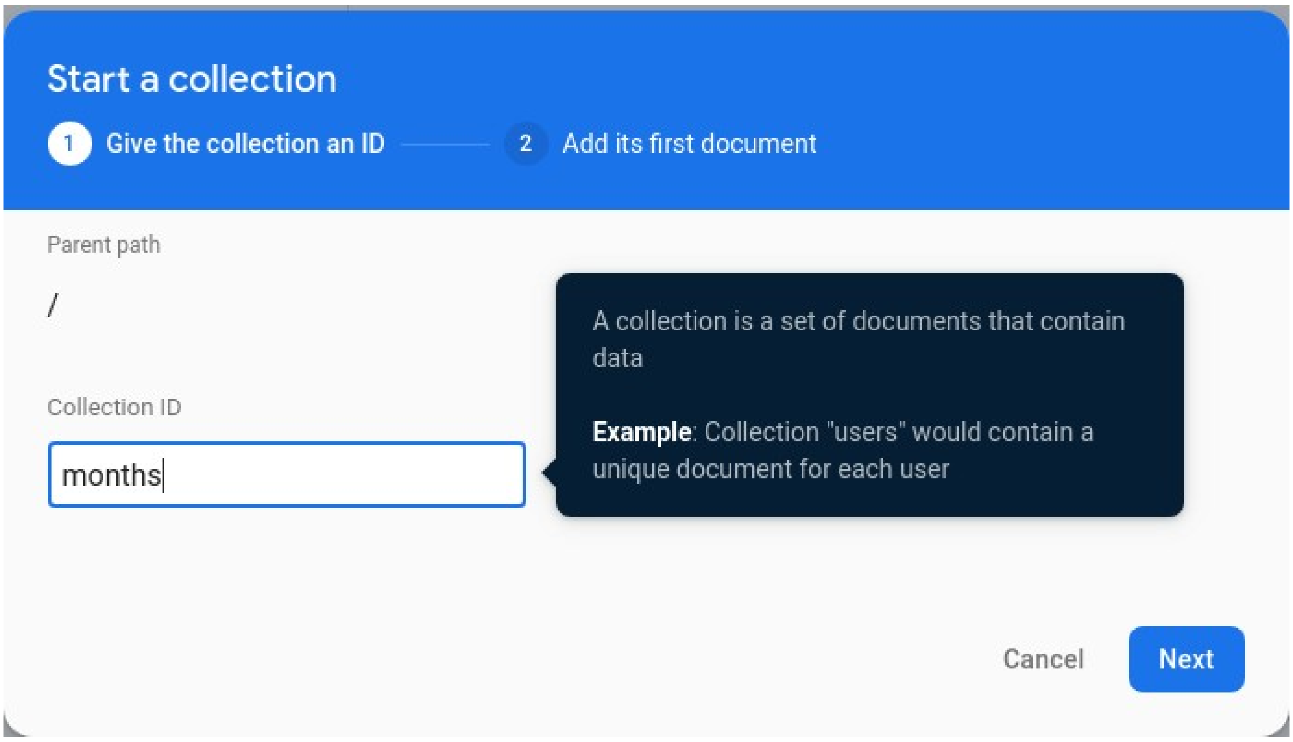
[✓ Start in **test mode**]
Get set up quickly by allowing all reads and writes to your database

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write;
    }
  }
}
```

Anyone with your database reference will be able to read or write to your database

Enabling Cloud Firestore will prevent you from using Cloud Datastore with this project, notably from the associated App Engine app





Parent path

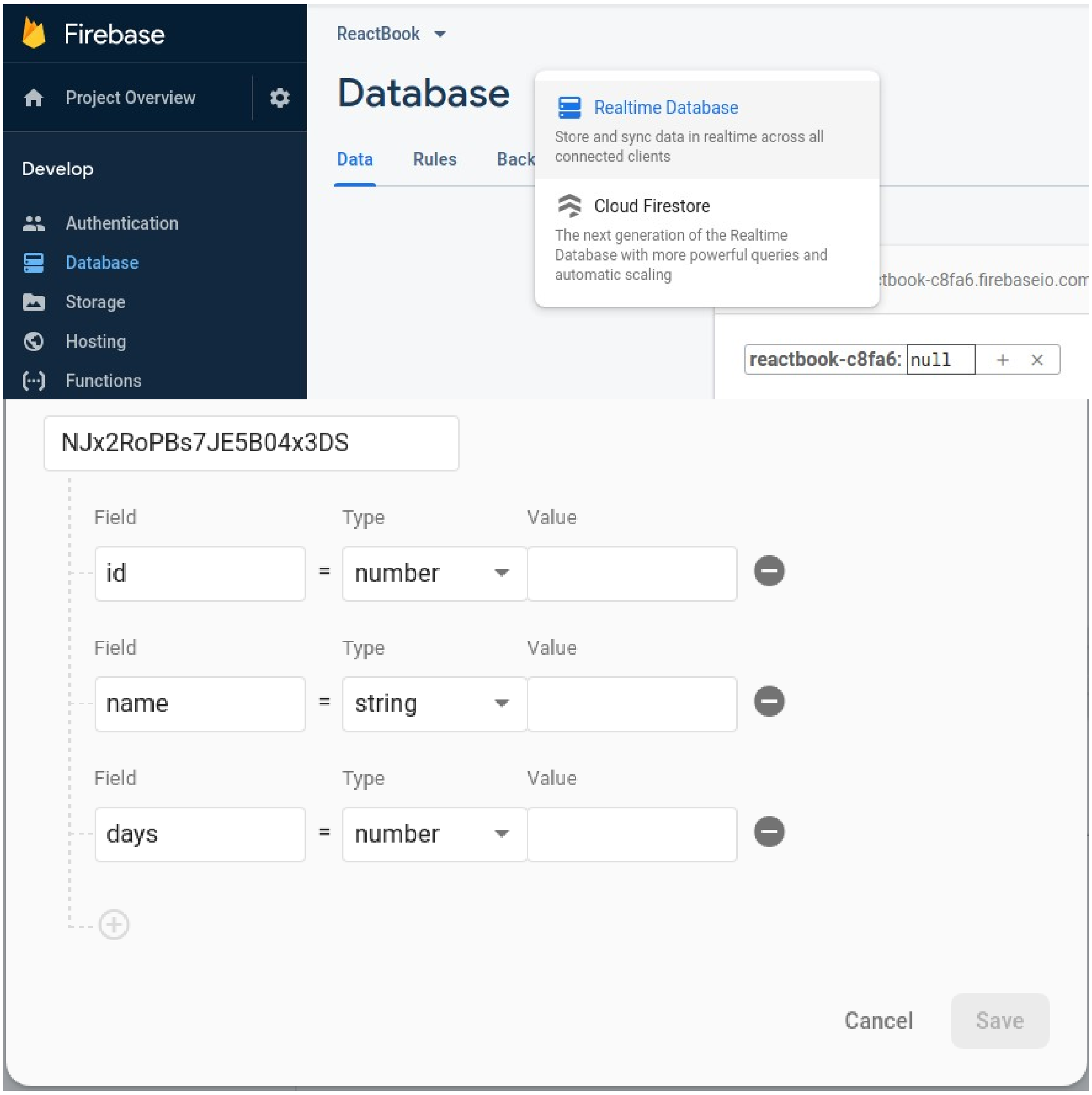
/

Collection ID

months

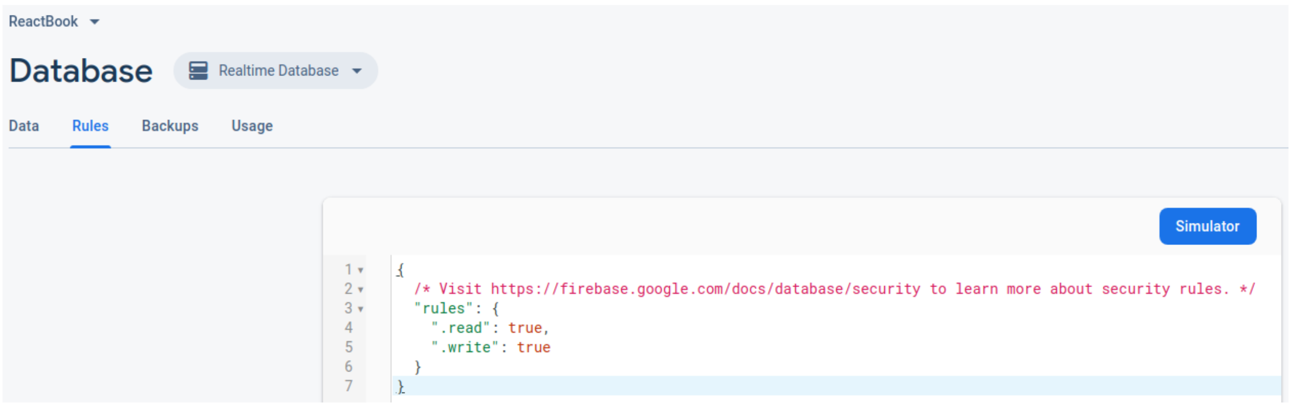
A collection is a set of documents that contain data

Example: Collection "users" would contain a unique document for each user


Firebase
Project Overview
Develop
Authentication
Database
Storage
Hosting
Functions
ReactBook
Database
Data
Rules
Back
Realtime Database
Store and sync data in realtime across all connected clients
Cloud Firestore
The next generation of the Realtime Database with more powerful queries and automatic scaling
tbook-c8fa6.firebaseio.com
reactbook-c8fa6: null
Cancel
Save

NJx2RoPBs7JE5B04x3DS

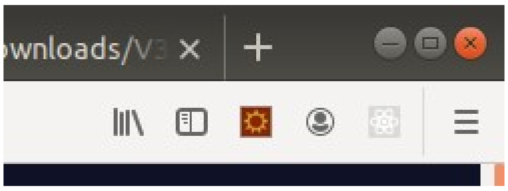
Field		Type	Value
id	=	number	
name	=	string	
days	=	number	



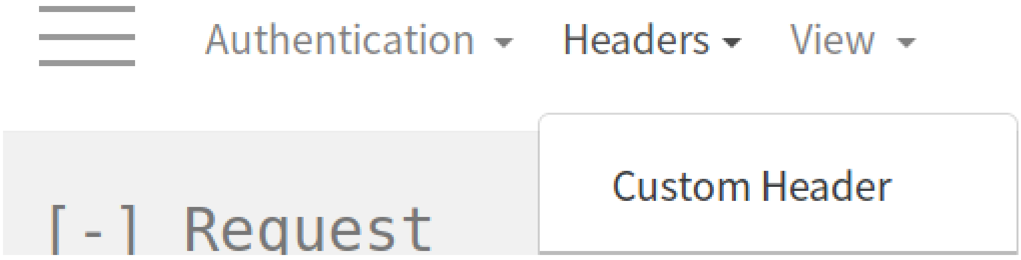
```
{
  /* Visit https://firebase.google.com/docs/database/security to learn more about security rules. */
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

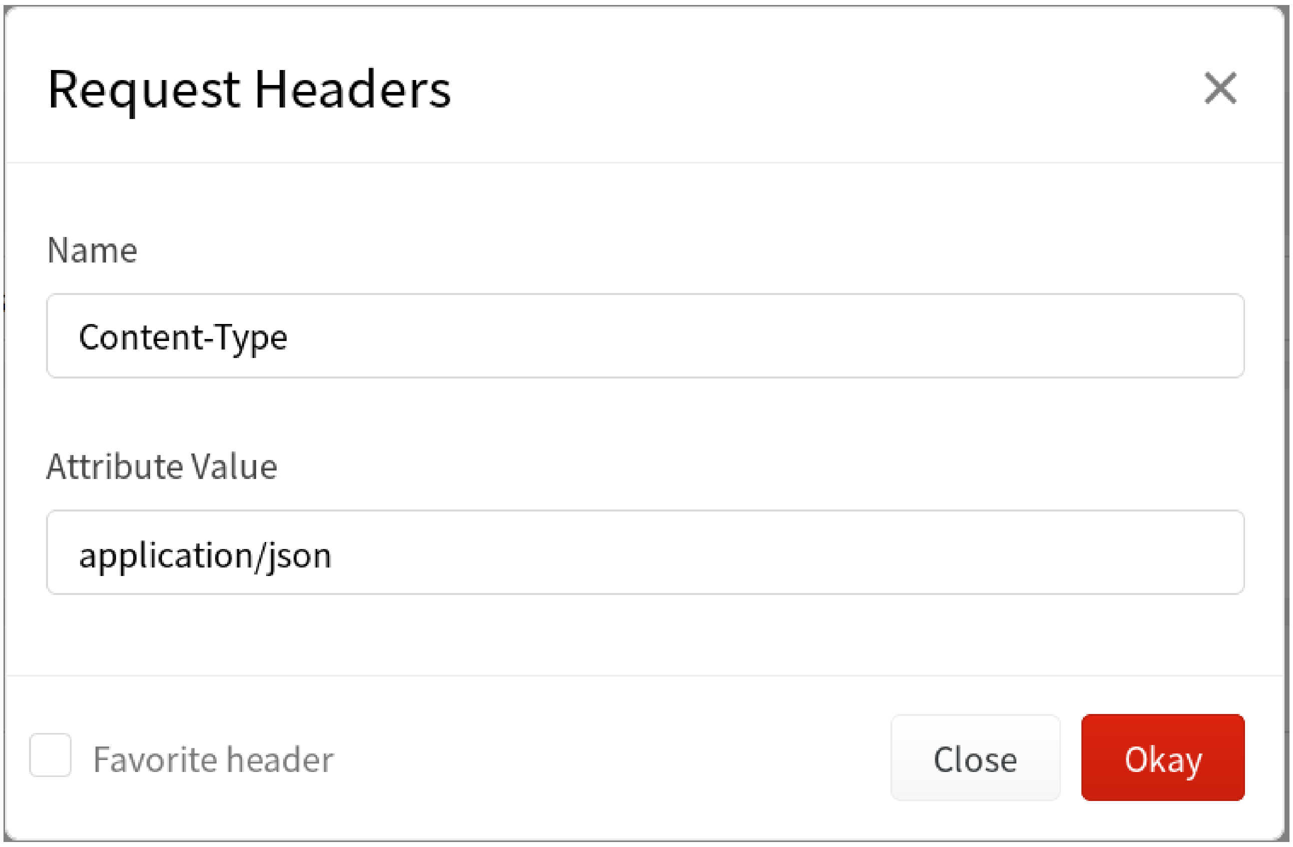
5.3 Adding some data to the backend

To add some data we need a REST client like Firefox RestClient, curl or Postman. To use Firefox a RESTClient add-in component must be installed (normal web browser can only GET data). Google firefox restclient and follow the first link install and Add to FireFox and install. After installation you see a new icon in toolbar of FireFox.



Click the RESTClient icon and start using the client. The client-server communication should happen with JSON content so first thing to do is to add from Header a Custom header and specify Content-Type-Application/JSON.




Request Headers

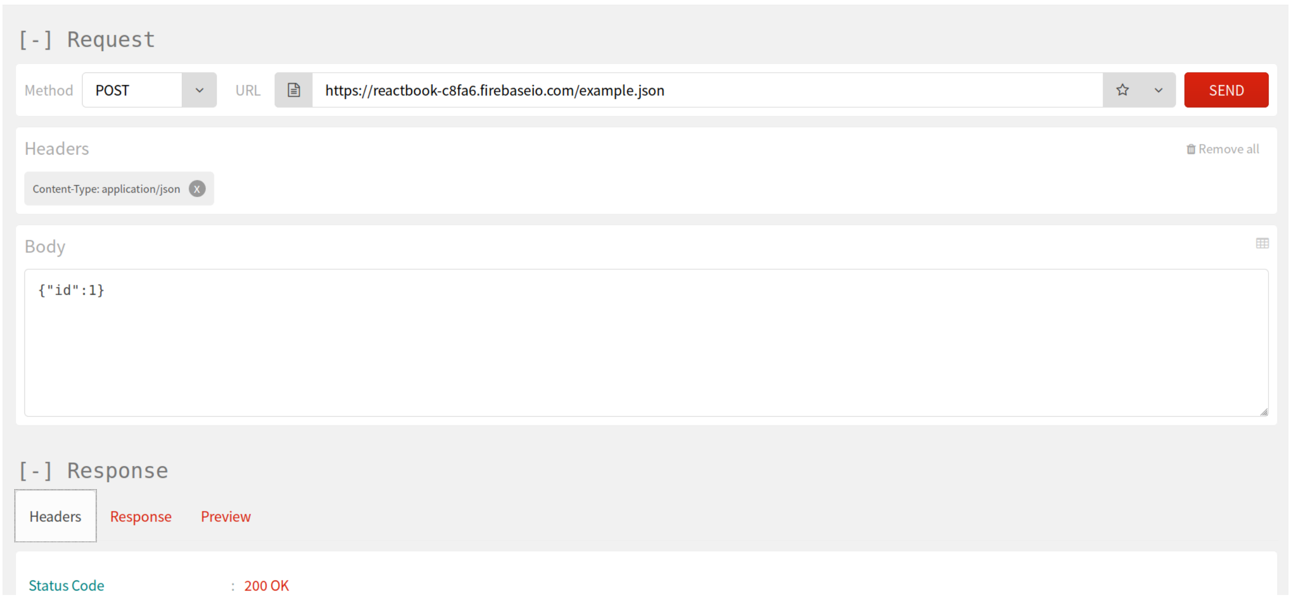
Name

Content-Type

Attribute Value

application/json

After setting the headers (and Authentication if needed) you can send some data to the back end by POST-http-method. Most simpliest JSON has only one key-value pair so let's try that. For the URL you need to enter the Firebase URL and the name of the JSON -file (here exmple.json) where the data is stored. By clicking SEND you will see 200 (OK - success) as RESPONSE Headers.


[-] Request

Method POST URL https://reactbook-c8fa6.firebaseio.com/example.json

Headers

Content-Type: application/json

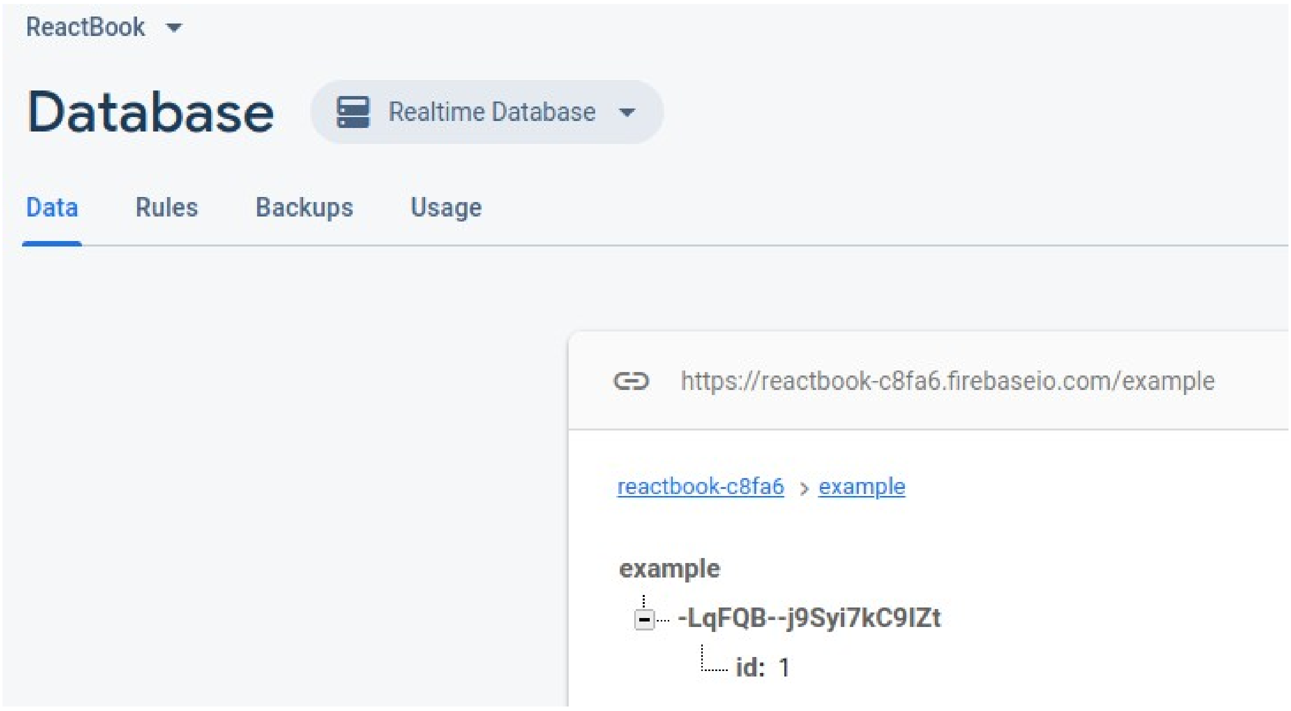
Body

```
{"id":1}
```

[-] Response

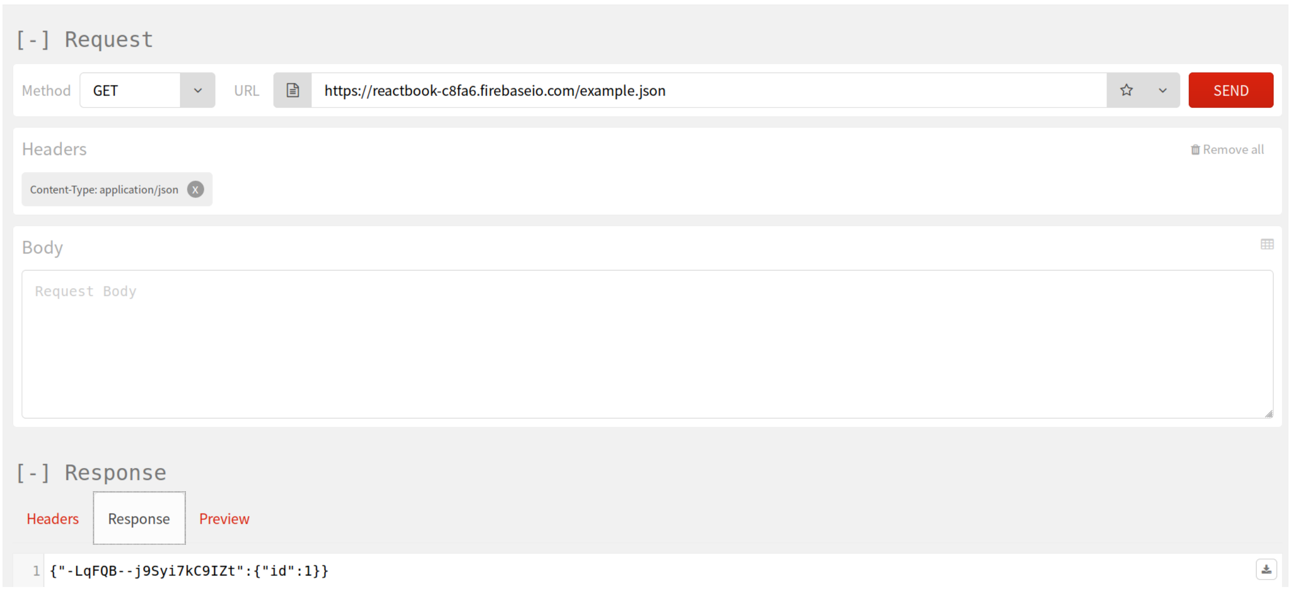
Status Code : 200 OK

To validata the result you can use the URL in RESTClient and paste is to address bar of the browser. You can see that the example was created (without JSON suffix) and data {“id”:1} is also there. Firebase will return an unique id given to the JSON object also.



```
https://reactbook-c8fa6.firebaseio.com/example
reactbook-c8fa6 > example
example
-LqFQB--j9Syi7kC9IZt
    id: 1
```

We can also use RESTClient to validate if the data really was saved to Firebase by GETting the same URL.

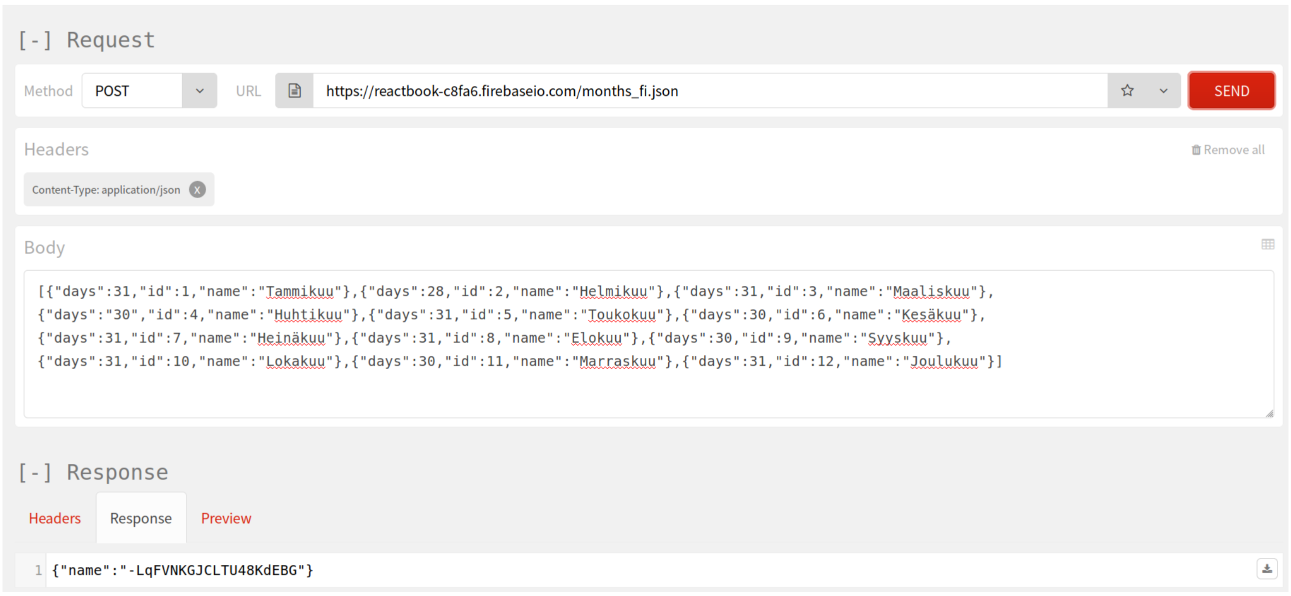


Method: GET
URL: https://reactbook-c8fa6.firebaseio.com/example.json
Content-Type: application/json

```
{"-LqFQB--j9Syi7kC9IZt":{"id":1}}
```

Now it’s time to create a URL to hold the data of months - id, name and the number of days in each month. Here is a JSON -array of Finnish months:

```
[{"days":31,"id":1,"name":"Tammikuu"},{"days":28,"id":2,"name":"Helmikuu"},
{"days":31,"id":3,"name":"Maaliskuu"},{"days":"30","id":4,"name":"Huhtikuu"},
{"days":31,"id":5,"name":"Toukokuu"},{"days":30,"id":6,"name":"Kesäkuu"},
{"days":31,"id":7,"name":"Heinäkuu"},{"days":31,"id":8,"name":"Elokuu"},
{"days":30,"id":9,"name":"Syyskuu"},{"days":31,"id":10,"name":"Lokakuu"},
{"days":30,"id":11,"name":"Marraskuu"},{"days":31,"id":12,"name":"Joulukuu"}]
```



Method: POST

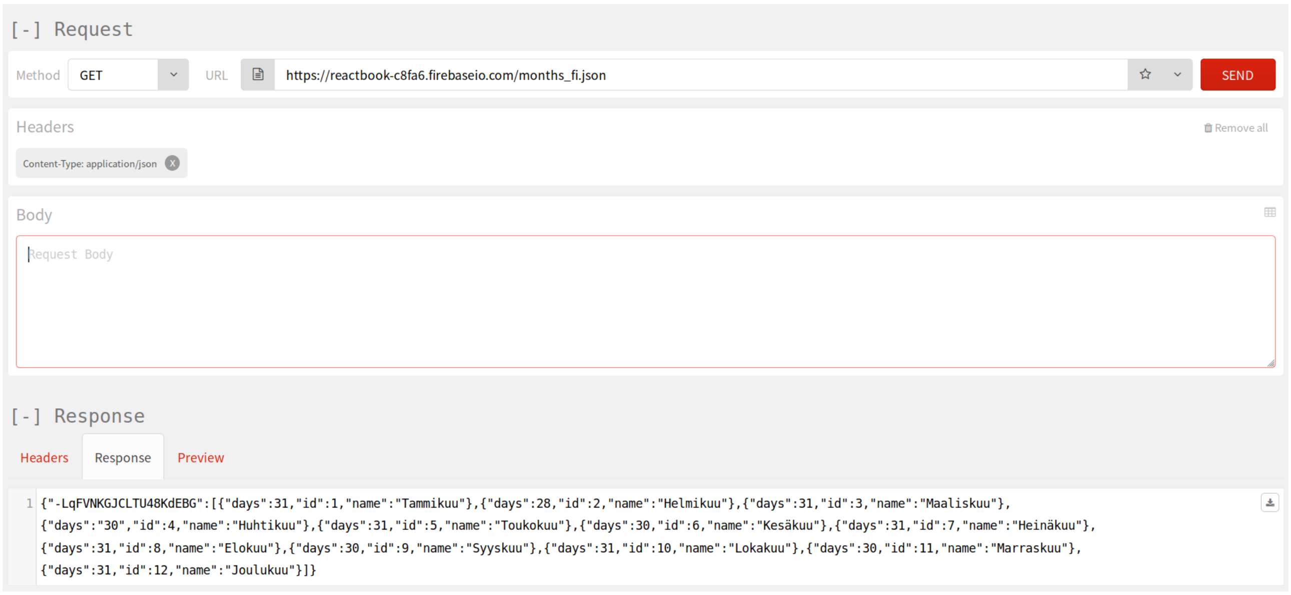
URL: https://reactbook-c8fa6.firebaseio.com/months_fi.json

Content-Type: application/json

```
[{"days":31,"id":1,"name":"Tammikuu"},{"days":28,"id":2,"name":"Helmikuu"},{"days":31,"id":3,"name":"Maaliskuu"},
{"days":"30","id":4,"name":"Huhtikuu"},{"days":31,"id":5,"name":"Toukokuu"},{"days":30,"id":6,"name":"Kesäkuu"},
{"days":31,"id":7,"name":"Heinäkuu"},{"days":31,"id":8,"name":"Elokuu"},{"days":30,"id":9,"name":"Syyskuu"},
{"days":31,"id":10,"name":"Lokakuu"},{"days":30,"id":11,"name":"Marraskuu"},{"days":31,"id":12,"name":"Joulukuu"}]
```

```
{"name":"-LqFVNKGJCLTU48KdEBG"}
```

And, again, the data can be fetched to validate that it was really saved by GET-method



Method: GET

URL: https://reactbook-c8fa6.firebaseio.com/months_fi.json

Content-Type: application/json

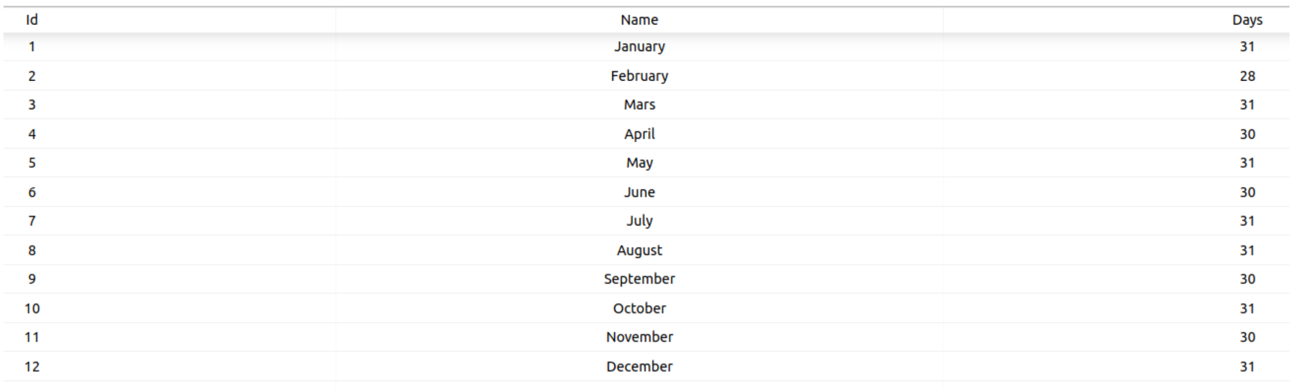
```
{"-LqFVNKGJCLTU48KdEBG":[{"days":31,"id":1,"name":"Tammikuu"},{"days":28,"id":2,"name":"Helmikuu"},{"days":31,"id":3,"name":"Maaliskuu"},
{"days":"30","id":4,"name":"Huhtikuu"},{"days":31,"id":5,"name":"Toukokuu"},{"days":30,"id":6,"name":"Kesäkuu"},{"days":31,"id":7,"name":"Heinäkuu"},
{"days":31,"id":8,"name":"Elokuu"},{"days":30,"id":9,"name":"Syyskuu"},{"days":31,"id":10,"name":"Lokakuu"},{"days":30,"id":11,"name":"Marraskuu"},
{"days":31,"id":12,"name":"Joulukuu"}]}
```

5.4 Fetching the data with React client

Let's create a new React project to you folder you build your React skills

```
$ npx create-react-app table
$ cd table
```

The target of this learning session is to build an UI like:



Id	Name	Days
1	January	31
2	February	28
3	Mars	31
4	April	30
5	May	31
6	June	30
7	July	31
8	August	31
9	September	30
10	October	31
11	November	30
12	December	31

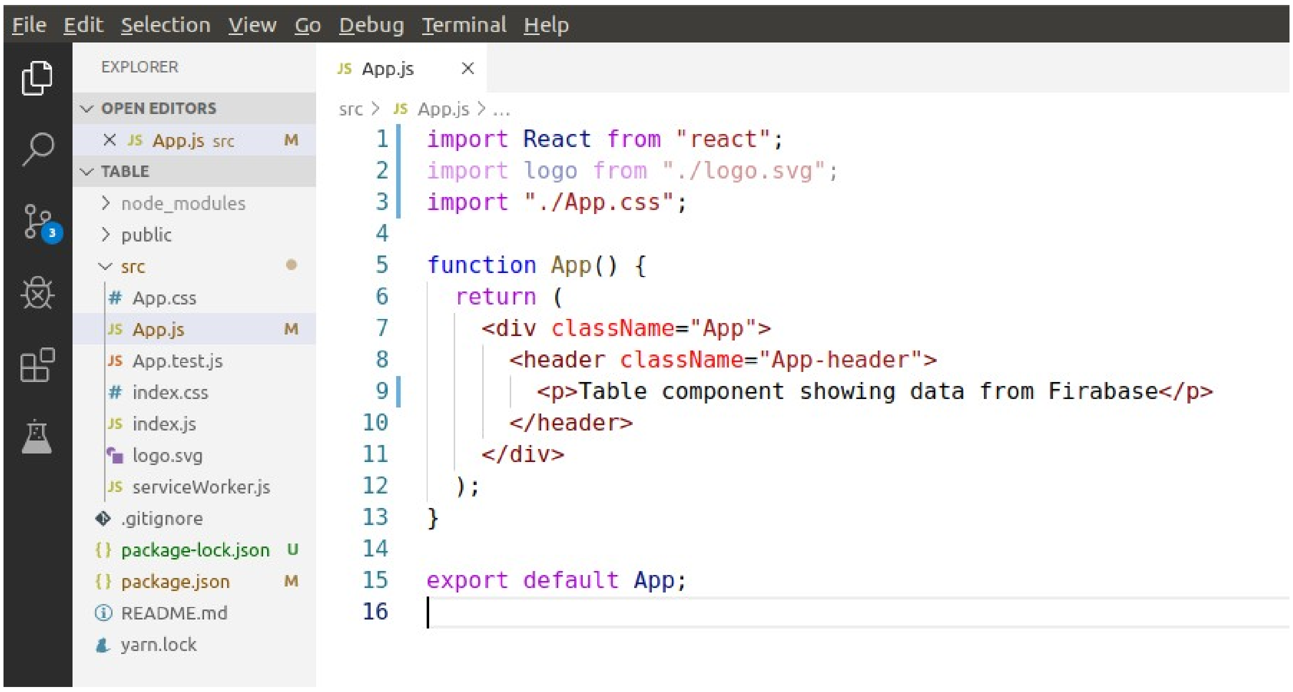
The data for the table in the UI is fetched from the Firebase back end.

To access the data at the back end and show the result in React app we need to install a new component https://www.npmjs.com/package/react-table. During the installation we want to add the component permanently to the project, thus save it to package.json -file as well

```
npm install --save react-table
```

(see the change in package.json file, new line "react-table": "^6.10.3")

To focus on data fetching and using the new component, the implementation is simplified only in App.js. Initially let's modify the header and remove not needed content.



```
import React from "react";
import logo from "./logo.svg";
import "./App.css";

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <p>Table component showing data from Firabase</p>
      </header>
    </div>
  );
}

export default App;
```

Update also the App.css to move the header at the top of the page (justify-content: top;)

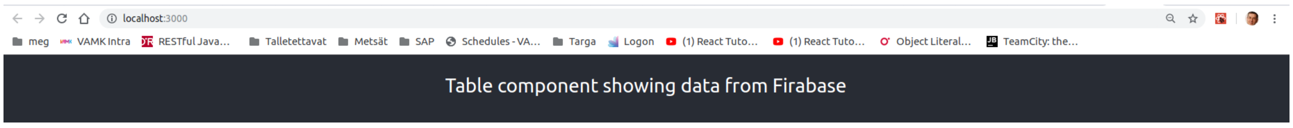


Table component showing data from Firabase

5.4.1 Using useState

To manage various states while the data is fetched from the server useState is the modern solution. Note that hooks don't work with classes so you need to use functional components (function App()..). The idea is to fetch the JSON -data from server and then store is locally into a state variable. State variable is used because the UI will automatically update itself ones the state is updated (rows added/deleted/updated).

To activate useState, import it:

```
import React useState  from "react";
```

The state variable must be inside a function, here we add the variable (=table) to the main function creating the functional component:

```
function App() {
  // the state is an array with two parts first is the variale, second is the variable
 //used to change the value of first variable
  const [months, setMonths] = useState([]);
```

The function useState return two components, the variable to store the data and the function that updates it. It's up to us how to name then but a good way is to name the variable accordingly and the function like **setSomething**. The code useState([]) initializes the month-variable with empty array. Whenever we want to overwrite the empty array we use the second argument in the array [months, **setMonths**]

To use the data for user interface (fill the table) we can just refer to the variable that is the first argument [**months**, setMonths].

5.4.2 Fetch

Fetch is a modern way of making a asynchronous request of resource (e.g. JSON-content) in a server. The challenge for the fetch is that the action for the response need to be carried out during .then sequence, here we first convert the response into JavaScript object and then store the object (array) to the variable defined as part of the state of the component

```
  fetch("https://tkareact.firebaseio.com/rest/months_en.json", {
    method: "GET",
    headers: { "Content-Type": "application/json" }
  })
    .then(response => {
      return response.json();
    })
    .then(responseData => {
```

```
    setMonths(responseData); //here we use the hook to save the data to state
   });
```

5.4.3 Using useEffect

While downloading the data from back end server it might take some time. The downloading process can cause some side-effects like downloading locally is updating the screen properly but if downloading from slower back end might cause screen without data. This is solved with useEffect -hook.

```
import React, { useState, useEffect } from "react";
import logo from "./logo.svg";
import "./App.css";

function App() {
 const [months, setMonths] = useState([]);
 useEffect(() => {
  fetch("https://tkareact.firebaseio.com/rest/months_en.json", {
    method: "GET",
    headers: { "Content-Type": "application/json" }
  })
   .then(response => {
    return response.json();
   })
   .then(responseData => {
    //let res = responseData;
    setMonths(responseData);
   });
 }, []);
 console.log(months);
 return (
  <div className="App">
   <header className="App-header">
    <p>Table component showing {months.length} data from Firebase </p>
   </header>
   <p>
    {months.map(month => (
     <p>{month.name}</p>
    ))}
   </p>
```

```
    </div>
  );
}

export default App;
```

The function useEffect has two arguments. The first one is (usually) to fetch the data and the second one in the state(s) that will ask the useEffect to retrieve the data again, thus update the screen. TODO:So in case we want to render when the variable month is updated, add variable to the second argument to useEffect.

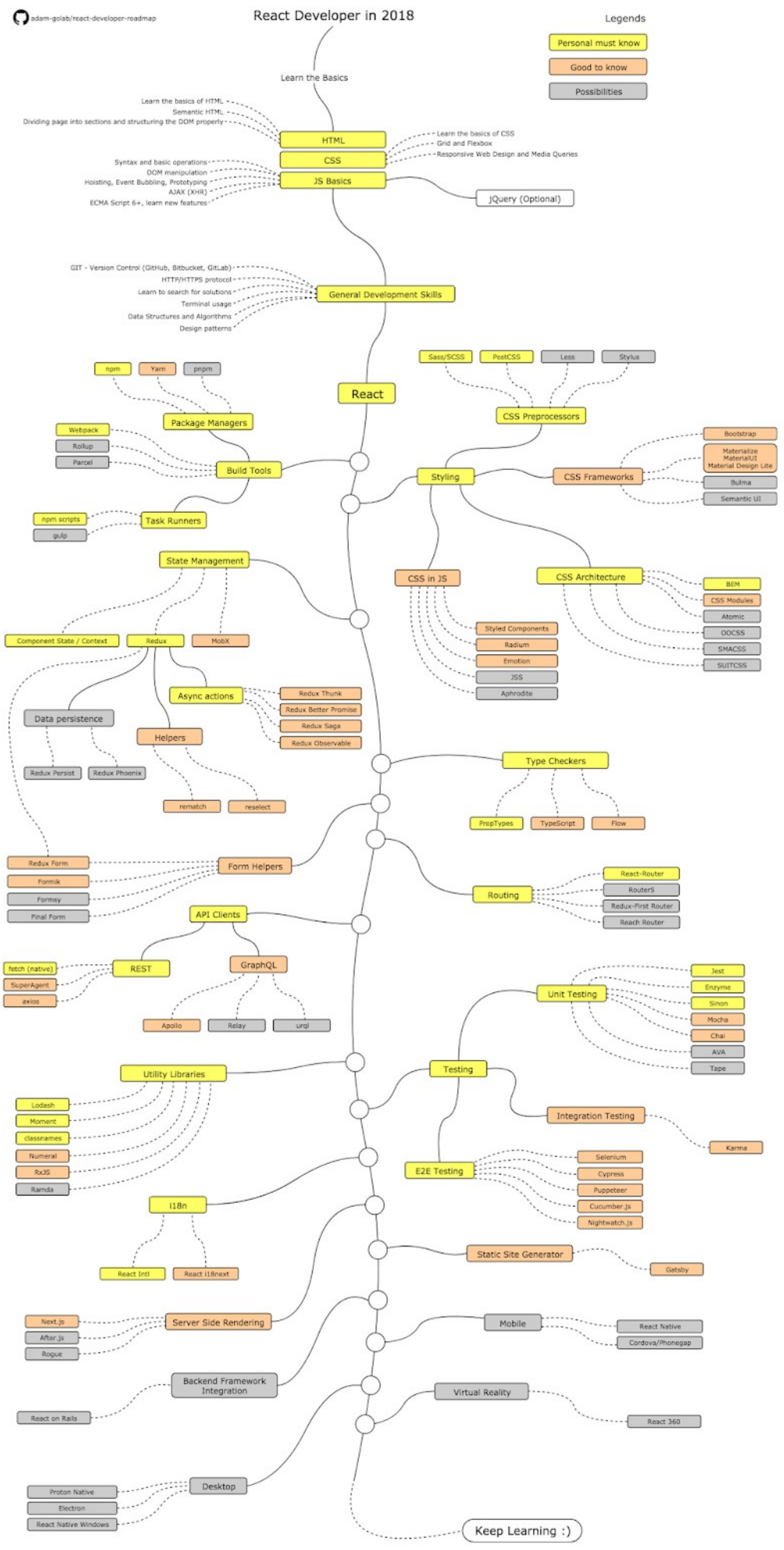
```
  useEffect(() => {
    fetch("https://tkareact.firebaseio.com/rest/months_en.json", {
      method: "GET",
            ...................................................
  }, [months]);
```

6 References

React - The Complete Guide (incl. Hooks, React Router and Redux)

https://git.vamk.fi/tka/reactbook/archive/1.3.zip

The small React book by Timo, 2019



adam-golab/react-developer-roadmap
React Developer in 2018
Legends
Personal must know
Good to know
Possibilities
Learn the Basics
Learn the basics of HTML
Semantic HTML
Dividing page into sections and structuring the DOM properly
HTML
CSS
JS Basics
Learn the basics of CSS
Grid and Flexbox
Responsive Web Design and Media Queries
Syntax and basic operations
DOM manipulation
Hoisting, Event Bubbling, Prototyping
AJAX (XHR)
ECMA Script 6+, learn new features
jQuery (Optional)
GIT - Version Control (GitHub, Bitbucket, GitLab)
HTTP/HTTPS protocol
Learn to search for solutions
Terminal usage
Data Structures and Algorithms
Design patterns
General Development Skills
React
npm
Yarn
pnpm
Package Managers
Webpack
Rollup
Parcel
Build Tools
npm scripts
gulp
Task Runners
Sass/SCSS
PostCSS
Less
Stylus
CSS Preprocessors
Styling
CSS Frameworks
Bootstrap
Materialize
MaterialUI
Material Design Lite
Bulma
Semantic UI
State Management
Component State / Context
Redux
MobX
Async actions
Redux Thunk
Redux Better Promise
Redux Saga
Redux Observable
Data persistence
Redux Persist
Redux Phoenix
Helpers
rematch
reselect
CSS in JS
Styled Components
Radium
Emotion
JSS
Aphrodite
CSS Architecture
BEM
CSS Modules
Atomic
OOCSS
SMACSS
SUITCSS
Type Checkers
PropTypes
TypeScript
Flow
Form Helpers
Redux Form
Formik
Formsy
Final Form
Routing
React-Router
Router5
Redux-First Router
Reach Router
API Clients
REST
fetch (native)
SuperAgent
axios
GraphQL
Apollo
Relay
urql
Unit Testing
Jest
Enzyme
Sinon
Mocha
Chai
AVA
Tape
Utility Libraries
Lodash
Moment
classnames
Numeral
RxJS
Ramda
Testing
Integration Testing
Karma
E2E Testing
Selenium
Cypress
Puppeteer
Cucumber.js
Nightwatch.js
i18n
React Intl
React i18next
Static Site Generator
Gatsby
Server Side Rendering
Next.js
After.js
Rogue
Mobile
React Native
Cordova/Phonegap
Backend Framework Integration
React on Rails
Virtual Reality
React 360
Desktop
Proton Native
Electron
React Native Windows
Keep Learning :)