React Eshop

Easily reusable Eshop in React, ES6, and Firebase

Manav Sehgal

reacteshop.com

React Eshop

Easily reusable Eshop in React, ES6, and Firebase. First few chapters only available at no cost. Code available at GitHub.

Manav Sehgal

This book is for sale at http://leanpub.com/reacteshop

This version was published on 2019-12-15

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Also By Manav Sehgal

React Speed Coding

Data Science Solutions

Contents

Easy Start React

Goal of this book is to recommend the shortest path to writing real-world React apps. We will code along a single page web app creating an easily reusable Eshop in React, ES6, and Firebase.

In each chapter we will complete certain feature goals for your Eshop app. We will also learn important React concepts along the way.

Eshop feature goals

In this chapter we will add first set of features to our Eshop.

- Browser tab title to reflect our intended app
- Responsive navigation bar with brand and links
- App introduction or main call-to-action area
- Device specific icons
- Meta tags targeting Facebook, Twitter, and search engines

React learning goals

We will also learn following concepts in this chapter.

- Scaffold a React app in three steps
- Integrate React Bootstrap
- Add ESLint in-place notifications to Atom editor
- Integrate Flow static type checking
- Generate a production ready build for our app
- Deploy our app to a local server
- Deploy our app to Firebase static hosting

Demo and Repo

Live demo of final completed Eshop is available on ReactEshop.com[1] website. Source for this chapter is available in our GitHub repo[2]. Please feel free to support us by starring this repo.

Download and run complete app

You can clone and run the final app we complete at the end of this book.

[1] https://reacteshop.com
[2] https://github.com/manavsehgal/react-eshop

```
git clone https://github.com/manavsehgal/react-eshop.git
cd react-eshop
npm install
npm start
```

Code along chapter by chapter

You can clone the source for this book, change to app directory, checkout just the code for a chapter, install and start the app to launch the local version in your default browser.

```
git clone https://github.com/manavsehgal/react-eshop.git
cd react-eshop
git checkout -b c01 origin/c01-easy-start-react
npm install
npm start
```

Start React app in three easy steps

Getting started with React is now easier than ever.

Step 1: All you need to do is install Create React App[3], the official scaffold generator provided by Facebook, to create a React app boilerplate.

Fire up your Mac Terminal (Applications > Utilities > Terminal) to start installing the scaffold generator.

The Create React App scaffold generator can be installed using Node Package Manager (NPM).

```
npm install -g create-react-app
```

Step 2: You can now scaffold a React app boilerplate.

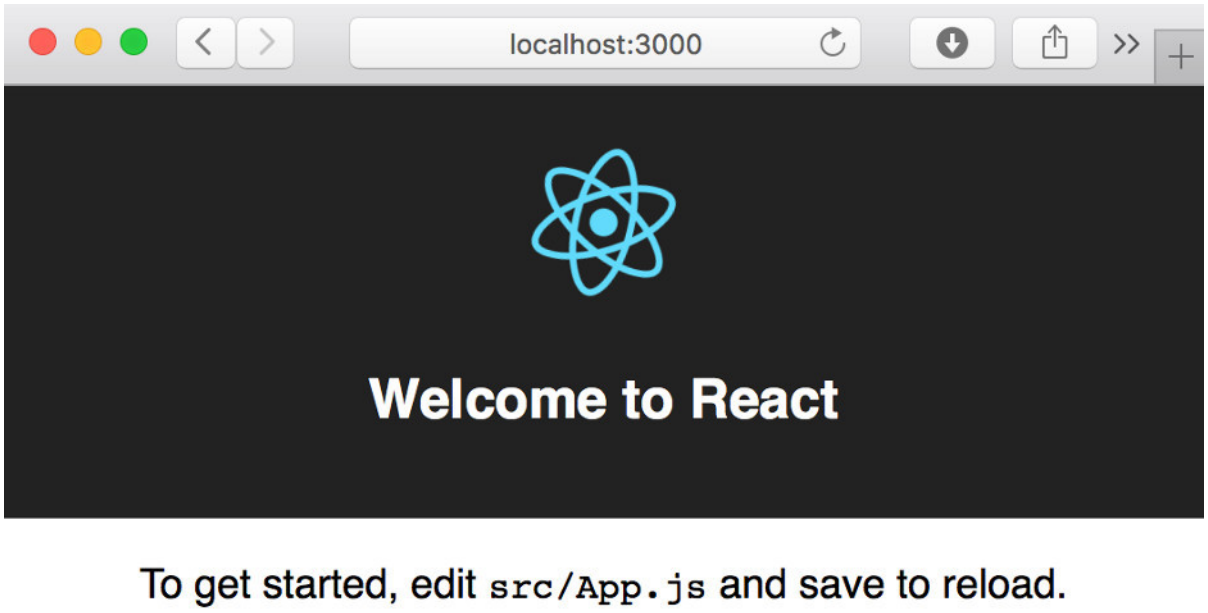
```
create-react-app react-eshop
```

The Create React App scaffold generator finishes with this message.

```
Success! Created react-eshop at .../react-eshop.
...
Happy hacking!
```

Step 3: That's it! Now you can run `npm start` command in the Terminal. This in turn fires up your browser to render your app at http://localhost:3000 local address.

[3] https://facebook.github.io/react/blog/2016/07/22/create-apps-with-no-configuration.html



Default React App

Windows and Node

Are you using Windows?

Unfortunately Windows users face several challenges using Node. Read one of the largest threads on these Windows-Node issues here[4]. We are not suggesting you invest in a Mac though.

Instead you may want to consider using Cloud based IDE like Cloud9[5]. Cloud9 comes pre-installed with Node, features intuitive browser based IDE, and offers terminal access over Ubuntu OS. You can start Cloud9 on a generous free plan.

Do not have Node?

If you get an error running the NPM command or your Node version is not current you can install or upgrade using installer available at the official Node website[6].

[4] https://github.com/nodejs/node-gyp/issues/629
[5] https://c9.io
[6] https://nodejs.org

Structure of the React app

The React app scaffold contains minimal number of files required for a root component defined in `index.js` and another custom component called App defined in the `App.js` file.

Each component imports its own CSS styles. Images can also be imported within the components.

```
react-eshop/      <== App Root
  README.md
  package.json    <== The only app config file
  .gitignore      <== Recommended folders/files to ignore in git

  node_modules/   <== Installed by Create React App

  public/
    index.html    <== Default page template
    favicon.ico   <== Static assets not processed by Webpack

  src/            <== App source lives here
    App.css       <== Component specific CSS
    App.js        <== Custom component definition
    index.css
    index.js      <== Root component definition
    logo.svg      <== Import image
```

The scaffold also contains minimal configuration within `package.json` and only one development dependency in the form of `react-scripts` package.

App.js component definition

Let us learn React ES6 fundamentals from the generated code.

The App component lives in `App.js` file. The component is defined using ES6 syntax.

Import. First statement is importing dependencies from React core. The `React` dependency is exported using `export default` so does not require { } braces when importing. There can only be one such dependency. The `Component` dependency is imported using { } braces. There can be multiple such dependencies.

Next statement imports the logo image used within the App component. Following statement imports the CSS styles used by the App component.

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
```

Class definition. ES6 `class` statement defines the custom App component and extends `Component` we import from core React.

```
class App extends Component {
```

JSX. The `render` method returns JSX which looks like HTML. JSX structures React component hierarchies, connects UI with event handlers, and specifies data flow between components.

```
render() {
  return (
    <div className="App">
      <div className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <h2>Welcome to React</h2>
      </div>
      <p className="App-intro">
        To get started, edit ...
      </p>
    </div>
  );
}
```

Native components. The HTML-like tags are actually native React components wrapping the respective HTML tag features. Yes, even `<p>` is a native React component wrapping the actual HTML DOM tag for paragraph.

```
<p className="App-intro">
  To get started, edit ...
</p>
```

Note the subtle differences in use of `className` instead of `class` which is not used as it is a JavaScript reserved keyword.

Composition. React is all about component hierarchies. You will notice two kinds of component compositions in this example. App custom component owns `div` native React component. The `div` component in turn is a parent to children `div` and `paragraph` components. Subsequent div component identified by className `App-header` is parent to `img` and `h2` child components.

```
<div className="App-header">
  <img src={logo} className="App-logo" alt="logo" />
  <h2>Welcome to React</h2>
</div>
```

Children. Components can be self-closing like the `img` component or have children components or text like the `paragraph` and `h2` components, with enclosing tags. These components can then refer to the enclosed components or text using `this.props.children` within their own component definition.

```
<h2>Welcome to React</h2>
```

Props. React components can pass data from owner components like custom App component in this case, into their owned components, like the `img` native component. This data is passed using what are known as props or properties. Examples of props passed in this file include `className`, `src`, and `alt` properties.

```
<img src={logo} className="App-logo" alt="logo" />
```

We will learn more React ES6 concepts as we code along Eshop app in this book.

index.js root component

Root component within a folder is identified by `index.js` file instead of file name by component class name as in case of App component.

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import './index.css';

ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

The root component is importing App component, React, and ReactDOM to render to HTML DOM target identified by the element id, `root` in this case.

package.json dependencies

The `package.json` configuration file represents the project dependencies used when you run the app or build.

```
"devDependencies": {
  "react-scripts": "0.2.0"
},
"dependencies": {
  "react": "^15.2.1",
  "react-dom": "^15.2.1"
},
```

There is only one development dependency, react-scripts. Runtime dependencies which are loaded in the browser include react and react-dom core libraries.

Integrating React Bootstrap

React Bootstrap[7] is a mature library of React components delivering Bootstrap[8] CSS styles for React apps. It completely rewrites Bootstrap JS using custom React components.

You can integrate React Bootstrap in three easy steps.

Step 1. Install React Bootstrap and Bootstrap CSS using NPM.

```
npm install react-bootstrap --save
npm install bootstrap@3 --save
```

This will add bootstrap version 3 and react-bootstrap as runtime dependencies within package.json file.

Step 2. Import Bootstrap CSS and optionally Bootstrap theme within `index.js` file.

```
...
import './index.css';
import 'bootstrap/dist/css/bootstrap.css';
import 'bootstrap/dist/css/bootstrap-theme.css';
...
```

Step 3. Import and use required React Bootstrap components within your app component.

Let us rewrite `App.js` using React Bootstrap custom components replacing all of React native components.

We start by importing the required components from react-bootstrap. Note that selectively importing components reduces the runtime size of our app and in turn improves performance.

[7] https://react-bootstrap.github.io/
[8] http://getbootstrap.com/

```
import React, { Component } from 'react';
import {
  Grid,
  Navbar,
  Nav,
  NavItem,
  Jumbotron,
  Button } from 'react-bootstrap';
```

We no longer need `App.css` as all styles will come from React Bootstrap.

Next we rewrite the render method. If you are familiar with Bootstrap, you will notice many similarities in the component hierarchy and naming. You will also appreciate that the React JSX code is way more concise than plain HTML + Bootstrap code. This is thanks to good React design patterns following by React Bootstrap.

```
class App extends Component {
  render() {
    return (
      <div>
        <Navbar inverse fixedTop>
          <Grid>
            <Navbar.Header>
              <Navbar.Brand>
                <a href="/">React Eshop</a>
              </Navbar.Brand>
              <Navbar.Toggle />
            </Navbar.Header>
            <Navbar.Collapse>
              <Nav pullRight>
                <NavItem href="//github.com/manavsehgal/react-eshop">
                  Code
                </NavItem>
                <NavItem href="//leanpub.com/reacteshop">
                  Book
                </NavItem>
              </Nav>
            </Navbar.Collapse>
          </Grid>
```

```
        </Navbar>
        <Jumbotron>
          <Grid>
            <h1>Easily Reusable Eshop in React</h1>
            <p>Eshop written in React, ES6, and Firebase.</p>
            <p><Button bsStyle="success" bsSize="large">
              Learn more
            </Button></p>
          </Grid>
        </Jumbotron>
      </div>
    );
  }
}

export default App;
```

If you are unfamiliar with Bootstrap, we suggest starting with their well documented website, followed by React Bootstrap documentation to note the similarities and differences.

Let us walk through the important areas to understand the new App component.

We are adding two important components `Navbar` and `Jumbotron` to deliver the Eshop feature goals for this chapter.

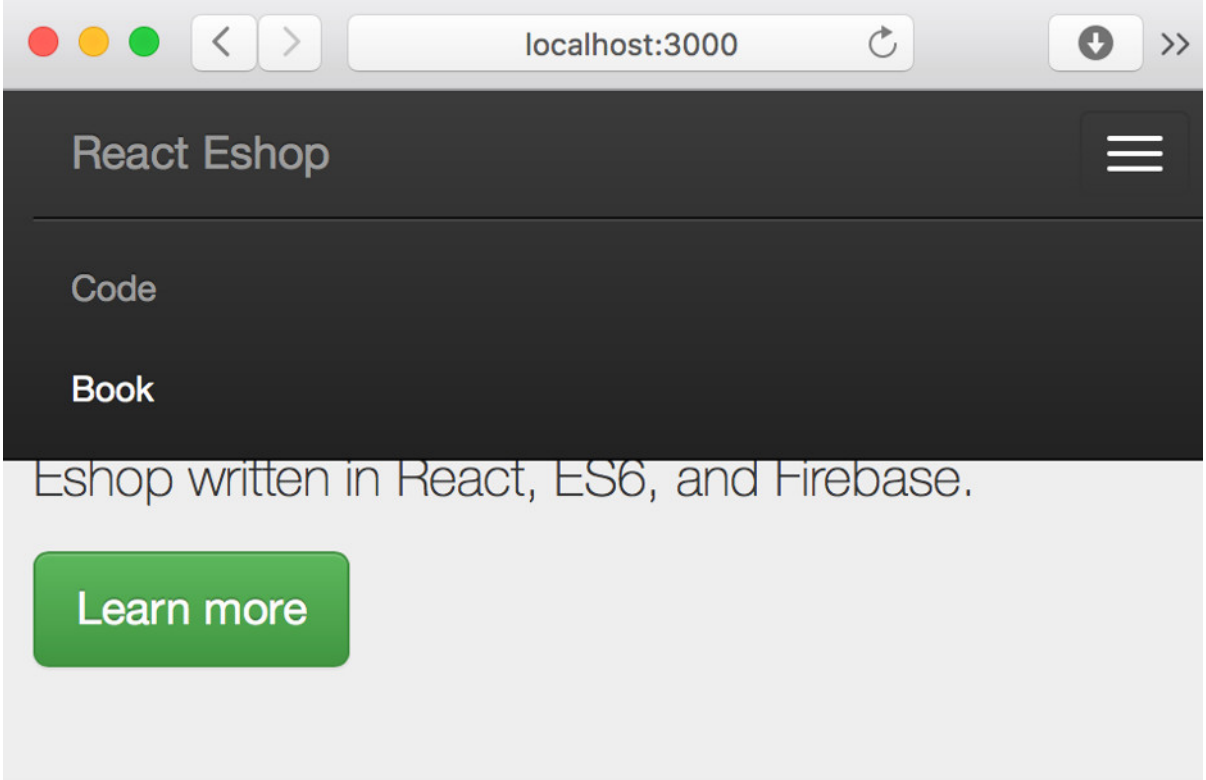
The `Navbar.Header` and `Navbar.Brand` are React namespaced components[9] representing children of `Navbar` parent component.

Note that the `Grid` component is only available in React Bootstrap, however it is actually wrapping `container` component from Bootstrap using `bsStyle='container'` default property.

The boolean properties without values `inverse`, `fixedTop`, and `pullRight` can be read as props set to `true` value. When these properties are not specified they default to `false` value.

Note that the new App component renders a mix of React Bootstrap custom components and React native components. Effectively mixing two component libraries written by different vendors to create your custom app, using a single component hierarchy. This is the true power of React. Composition is so intuitive that you miss how easy this integration is compared with other solutions.

[9] https://facebook.github.io/react/docs/jsx-in-depth.html#namespaced-components



Responsive navigation and call-to-action

index.html template

To change the browser tab title we need to change the page template defined in the `/public/index.html` template.

The page template can be used to add webfonts, meta tags, or analytics, just like you would do in the default HTML file.

You can also use EJS template engine[10] syntax to make really powerful templates for your React apps.

Let us add some EJS to configure meta tags targeting Facebook, Twitter, and search engines.

```
<%
  var title = 'React Eshop | Easily reusable Eshop in React';
  var description = 'Reusable Eshop app in React, complete with a shopping ca\
rt, product catalog, and search.';
%>
<title><%= title%></title>
<meta name=”twitter:title” content="<%= title%>">
<meta property=og:title” content="<%= title%>">

<meta name="description" content="<%= description%>">
<meta name=”twitter:description” content="<%= description%>">
<meta property=og:description content="<%= description%>">

<meta name=”twitter:card” content=”summary” />
<meta property="og:site_name" content="React Eshop" />
<meta property=og:type content="website">
```

In the HTML code for our template's head section we are using EJS tags `<% %>` to add JavaScript. The EJS `<%= %>` tag inserts result of JavaScript expression following the equal sign within the HTML.

[10] http://www.embeddedjs.com/

ESLint for JS guidelines and syntax checking

Using `npm start` or `react-scripts start` also runs ESLint on your code. ESLint checks your JavaScript code against naming conventions, best practices, and syntactical bugs. Let us try and break one of the rules of ESLint in our app and see how the error shows up in our Terminal.

Let us replace `Component` with `React.Component` in the class definition, without changing the import statements.

```
class App extends React.Component {
```

As you save `App.js` after making the above change, you will notice the Terminal starts displaying warnings.

```
Compiled with warnings.

Warning in ./src/App.js

.../react-eshop/src/App.js
  1:17  warning  'Component' is defined but never used  no-unused-vars

▫ 1 problem (0 errors, 1 warning)
```

This warning is somewhat helpful. You can get better insights by integrating ESLint warnings within your editor.

To integrate ESLint within Atom editor use the following instructions.

Step 1: Add global dependencies required for ESLint editor integration. This workaround is required until ESLint fix it.

```
npm install -g eslint-config-react-app eslint babel-eslint eslint-plugin-reac\
t eslint-plugin-import eslint-plugin-jsx-a11y eslint-plugin-flowtype
```

You may note some EPEERINVALID warnings thrown by `eslint-config-react-app` however these can be ignored. If you want to do away with these warnings you can install specific versions like so.

```
npm install -g eslint-config-react-app@0.2.1 eslint@3.5.0 babel-eslint@6.1.2 \
eslint-plugin-react@6.3.0 eslint-plugin-import@1.12.0 eslint-plugin-jsx-a11y@\
2.2.2 eslint-plugin-flowtype@2.18.1
```

Step 2: Install Linter ESLint Atom package. We also install base linter package for helpful in-place notifications.

```
apm install linter-eslint
apm install linter
```

Step 3: Edit Linter ESLint settings to check *Use Global ESLint installation* option.

Step 4: Update `package.json` with eslint configuration.

```
"eslintConfig": {
  "extends": "react-app"
}
```

Now as you make coding mistakes which ESLint catches, you will start getting hints right within your editor. This approach is more intuitive than console warnings.

ESLint Atom Integration

Flow static type checking

Just like ESLint checking, you can optionally setup Facebook Flow[11] to perform static type checking within your JavaScript code and avoid really hard to fix bugs right within your code editor.

These may be a category of bugs that are not even picked up by the compiler or ESLint.

Follow the instructions on Flow GitHub repo[12] for installing this useful tool.

Once installed, change to your app root folder `/react-eshop` and run `flow init` just once. This will create a `.flowconfig` file in your app root. Add ignore section within this file.

```
[ignore]
<PROJECT_ROOT>/node_modules/fbjs/.*
```

Facebook promises they will consider integrating more tightly with Flow in the future to avoid this workaround.

To help understand the value of Flow let us introduce a type casting bug into our app. Open the `App.js` file.

We can mock a const called `version` and try to multiply an integer with a string.

```
render() {
  const version = 1 * "beta";
```

We also display this version within our app title like so.

```
<h1>Easily Reusable Eshop in React {version}</h1>
```

To have Flow examine this file we need to add a `/* @flow */` comment at the top of the file.

Next we go to our Terminal and run `npm start` command if it is not already running. You will notice that the app runs just fine without any errors or warnings. However, the title displays `NaN` where version is supposed to be rendered.

Flow can help us catch this kind of bug. Now open another Terminal window or stop your app and run `flow check` command.

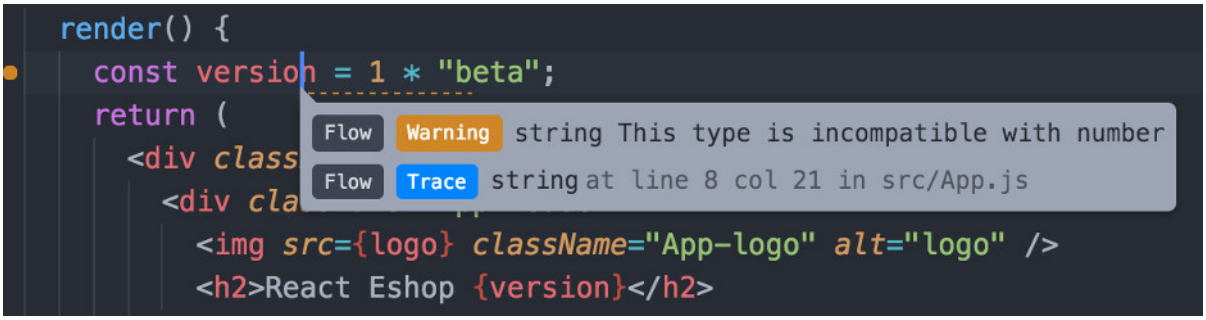
[11] https://flowtype.org/
[12] https://github.com/facebook/flow#installing-flow

```
react-eshop: $ flow check
src/App.js:13
 13:     const version = 1 * "beta";
                             ^^^^^^ string. This type is incompatible with
 13:     const version = 1 * "beta";
                           ^^^^^^^^^^ number
```

If you want to catch such type checking errors in-place within the editor, all you need to do is install another package. In case of Atom editor we can install the linter-flow package.

```
apm install linter-flow
```

Now if you introduce type checking errors in your code, your Atom editor will provide in-place Flow notifications. How cool is that!



Flow Atom Integration

Build and deploy

When you are ready to deploy your app, you can use `npm run build` to generate a production optimized version of your app.

The scaffold generator performs several optimizations behind the scenes including bundling, minifying, gzip compression, and generating file name hashes for facilitating browser caching among other things.

```
Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

  92.31 KB  build/static/js/main.7bddf9bd.js
  21.17 KB  build/static/css/main.406970a0.css
...
```

The build folder now contains production ready files. These include glyphicons as part of Bootstrap, source map files for browser debugging, minified and gzipped js/css files, and generated index.html from our EJS template.

```
build/
  static/
    js/
    css/
    media/
  favicon.ico
  index.html
```

You can now use the recommended static server to deploy the build on localhost.

```
npm install -g pushstate-server
pushstate-server build
open http://localhost:9000
```

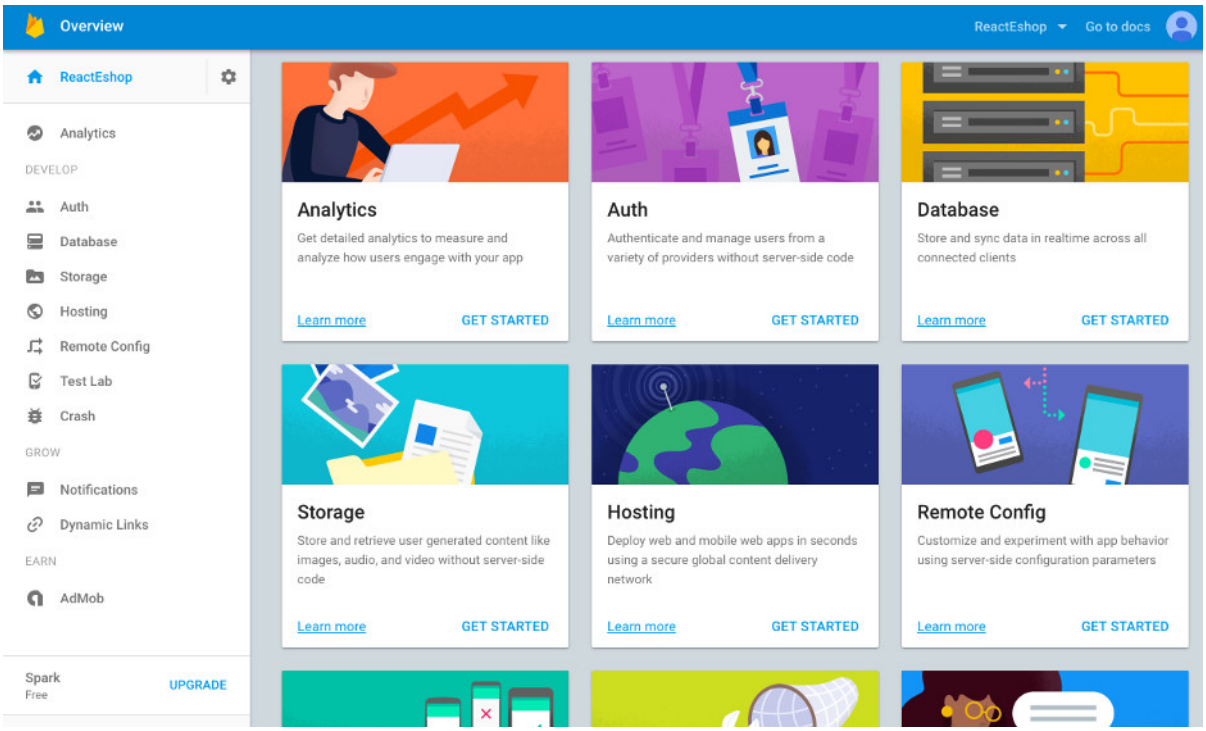
Optionally, you can even deploy your app to Github[13] following instructions provided in the Create React App docs.

[13] https://github.com/facebookincubator/create-react-app/blob/master/template/README.md#deploy-to-github-pages

Deploy using Firebase

You can deploy your app using Firebase static hosting.

To start with you need a gmail account. Login to your gmail account and access the Firebase website[14]. Once you are on their website you will notice **Go to console** link on the top right. As you click on this link you will be directed to create a new project. Once you create a new project you will land on their administration dashboard with more than 10 feature menus on the left. You are now active on Firebase free Spark plan.



Firebase Features

To get started with deploying your app, you need to install the Firebase CLI.

```
npm install -g firebase-tools
```

Next you need to type `firebase login` in your Terminal to login to your Gmail account. Once you are logged in using Terminal, run the `firebase init` command within your app root to create the `firebase.json` configuration file interactively. Select default options when prompted with exception of following questions.

[14] http://firebase.google.com/

- What do you want to use as your public directory? **build**
- Configure as a single-page app (rewrite all urls to /index.html)? **Y**
- File build/index.html already exists. Overwrite? **N**

Once the initialization is complete Firebase CLI will write three new files in your app root.

- **database.rules.json** contains security rules for your Firebase database
- **.firebaserc** connects the Firebase project with your app
- **firebase.json** sets up rewrite rules and identifies the app directory to deploy to hosting

Now all you need to do is run `npm run build` to create a fresh build for your app. Next run `firebase deploy` to deploy the build to firebase hosting and you are done. Hit `firebase open` on your Terminal to open the Firebase hosted app in your browser.