ToyBlock

Webpage explaining the concept of blockchain in a pedagogic way.

Programmer's Guide

Août 2021

Antoine Van Aelst

Contents

1 Introduction

1.1 What's ToyBlock

ToyBlock is a webpage popularizing how a blockchain works in a pedagogic way. The user follows the story of an animal village on their way to create a new currency and learns how to build a blockchain technology, step by step by resolving all the problems encountered by the animals of the village.

1.2 Deployed version

You can check a deployed version of this webpage at this address:
https://ushien.github.io/ToyBlock/

1.3 Repository

Here is the github repository of the project:
https://github.com/Ushien/ToyBlock

1.4 Author

This project is developed by Antoine Van Aelst as part of the Unamur INFOB318 Projet Individuel class.

Author: Antoine Van Aelst
Project manager: Jérôme Fink
Teacher in charge: Vincent Englebert

1.5 Environment

ToyBlock is developed on a Windows 10 version of Visual Studio Code using the Node.js library.

1.6 Technology

ToyBlock is written with the help of React, Create React App and React-Bootstrap.

1.6.1 Websites

- React: https://fr.reactjs.org/
- React-Bootstrap: https://react-bootstrap.github.io/
- Create React App: https://create-react-app.dev/

2 Project structure

2.1 Main files

The project is based on the structure provided by Create React App. The code is split into 3 main files:

1. App.js: Contains the structure and the main components of the project.
2. Blocs.js: Contains the whole text of the website distributed into different blocks.
3. blockchain.js: Contains the specific functions and components coding the different interactive machines within the story.

React and React-Bootstrap make coding the entire project only by using javascript possible.

2.2 Why React ?

I chose React because this programming language is the perfect tool for coding blocks structured webpages which was exactly what I planned to do. I wanted the story blocks of the page appearing and disappearing in a dynamic way and I also wanted to code completely independent interactive machines.

3 Understanding the Code

3.1 Blocs.js

3.1.1 Structure

Each block of text is represented by a function returning a JSX expression.

```
function introtext1(){
        return(<div>
                Au beau milieu de la forêt se trouve un village...
        </div>
        )
}
```

You can also add a parameter to the function to use it in the text. Don't forget to send the parameter when you call the function.

```
function text7(moneyname){
        return(<div>
                <div>
                Le chef du village réagit : Après chaque cycle, l'animal ay
                </div>
                <h2>
                Problème:
                </h2>
                <div>
                Le village grandit et le système {moneyname} a beaucoup de
                </div>
        </div>
        )
}
```

The export instruction must contain every block of text you want to use in your story.

```
export {introtext1, introtext2}
```

3.1.2 How to add a block of text

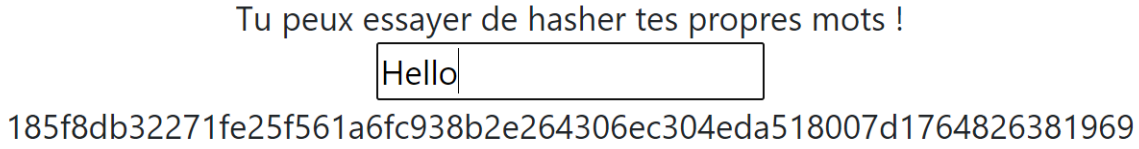
- First create a function and name it the way you want.
- Pass the parameters you may need.
- Add the function to the export list at the end of the file.

3.2 blockchain.js

Here you can organize and write components of the interactive machines of the story. Just don't forget to add everything you want to import in App.js in the export list at the end of the file.

3.2.1 Hashing machine

The hashing machine uses the SHA256 hashing algorithm provided by the crypto.js library to hash a text in real time.



You can change the default hashed String in App.js.

3.3 App.js

3.3.1 Imports

This section contains all the imports of the projects. The project currently includes components from the following libraries:

- React
- React-Bootstrap
- crypto.js

3.3.2 Constants declarations

This section contains the basic parameters of the webpage, including default values for many elements.

- startdistance : (from 1 to 13) Indicates at which point you want to start the story. This variable mainly exists for testing purposes.
- defaultname : Indicates the default suggested name of the currency.
- baseword : Indicates the default word that will be hashed in the hashing machine.

3.3.3 HandlingBlock

This is the main component of the project, containing all the sub-components. This component contains the main state of the program:

- distance represents the current progress of the story. A higher distance means you are further in the story.
- moneyname : Indicates the name of the story, which is an input of the user in the introduction of the story.

Then you have the high-level methods of the program.

- incrementDistance() must be used to progress in the story.
- changeName(newname) must be used to modify the name of the currency.

Finally you can find the render method of the component. It defines the order of the different blocks of the story and render them depending on the distance variable. To add a new block, define a new condition, and call the associated component inside a JSX expression.

```
if (this.state.distance === 1){···
}
if (this.state.distance ===2){
  console.log("Entered Intro Block 2")
  return (
  <div>
    <BlocIntro2 onDistanceChange={this.incrementDistance} distance={this.state.distance}/>
  </div>
  )
}
if (this.state.distance ===3){···
}
```

Note that the conditional structure of the 5 to 13 blocks makes them appear without making the previous one disappear. The intro blocks are meant to appear alone.

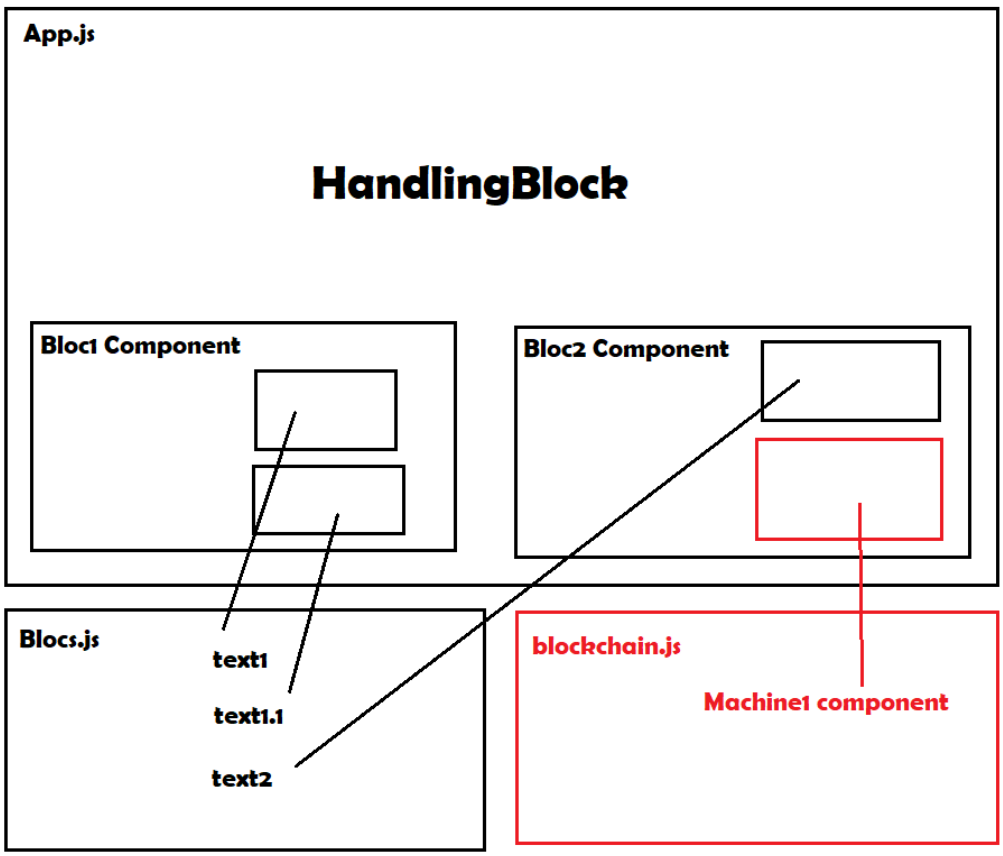
You can send all the additional informations or methods a block could need by passing them arguments.

3.3.4 Blocks components

Each of these components encapsulate one of multiple text blocks from Blocks.js, blockchain.js into their its render method. One component represents what will be rendered if you take a step forward in the story.

You can also bind your methods to the ones provided by the HandlingBlock to interact with the state of the program.

Here is a summary of the structure of the program:



4 Run the code

Before anything, please get the code from the ToyBlock repository.

4.1 Test the code

First please make sure your device meets all the requirements needed for the lauching of the app. ToyBlock is running on top of React so make sure you have Node.js installed on your device. Please follow this link if you need to install it: Node.js

Then you may need to install scripts and configurations used by Create React App with the following command:

```
npm install react-scripts --save
```

After that, go to code/toyblock/ and then type

```
npm start
```

That will launch the app on localhost:3000.
To apply any modification, just save the file and reload the the page in your browser.

4.2 Deploy the page

Multiple methods can be used depending on the type of server and website you want to add ToyBlock to. Please refer to the official Create React App documentation for any specific case.

4.2.1 Deploy on a static server

Again, make sure Node.js is installed on your server, then install serve with this command:

```
npm install -g serve
```

Then run

```
serve -s build
```

to deploy your page on the port 5000 of your server.

Or run

```
serve -s build -l 3000
```

to freely choose the port of the server you prefer, by replacing 3000.

5 Future updates

The difficulties of the development pushed me to prefer a limited but functional application over a ambitious and buggy code. These are then possible improvements that could still be implemented.

- Reworking the visual aspect of the webpage, including improved transitions, better positioning of the elements, and colorful illustrations.
- More interactive machines to illustrate different key moments of the story: Public-key cryptography illustration, transactions mails sending within the village, etc.

6 Contact

Any question or remark ? Send an email to antoinevaelst@gmail.com.