ToyBlock

Webpage explaining the concept of blockchain in a pedagogic way.

Programmer's Guide

Juin 2022

Antoine Van Aelst

Contents

1 Introduction

1.1 What is 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, JSDoc 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/
- JSDoc: https://jsdoc.app/

2 Project structure

2.1 Main files

The project is based on the structure provided by Create React App. The code is split into 4 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. Components.js: Contains the components coding the different interactive machines within the story.
4. Classes.js: Contains the JavaScript classes with which the different machines interact

Additionally to that, the following files and directories handle every visual aspect of the website:

1. App.css: Contains all the CSS code of the project.
2. ./visuals: Contains every image file necessary to the website.

React makes coding the entire project only by using JavaScript and CSS possible.

2.2 Why React ?

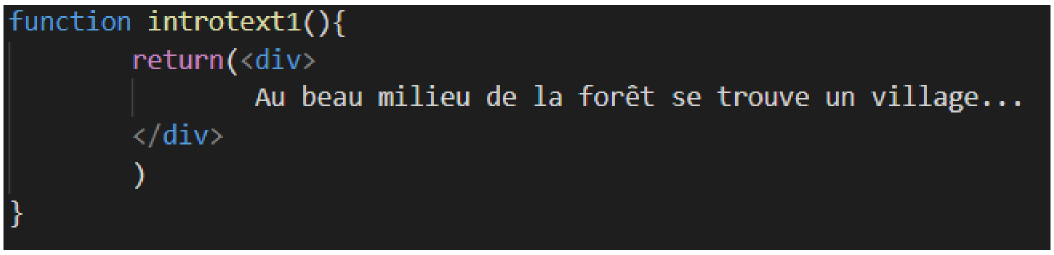
I chose React because this JavaScript framework is the perfect tool for coding blocks structured webpages which was exactly what I planned to do. I wanted the story blocks of the page appearing and disappearing in a dynamic way and I also wanted to code completely independent interactive machines I can easily encapsulate into each other.

3 Understanding the Code

3.1 Blocs.js

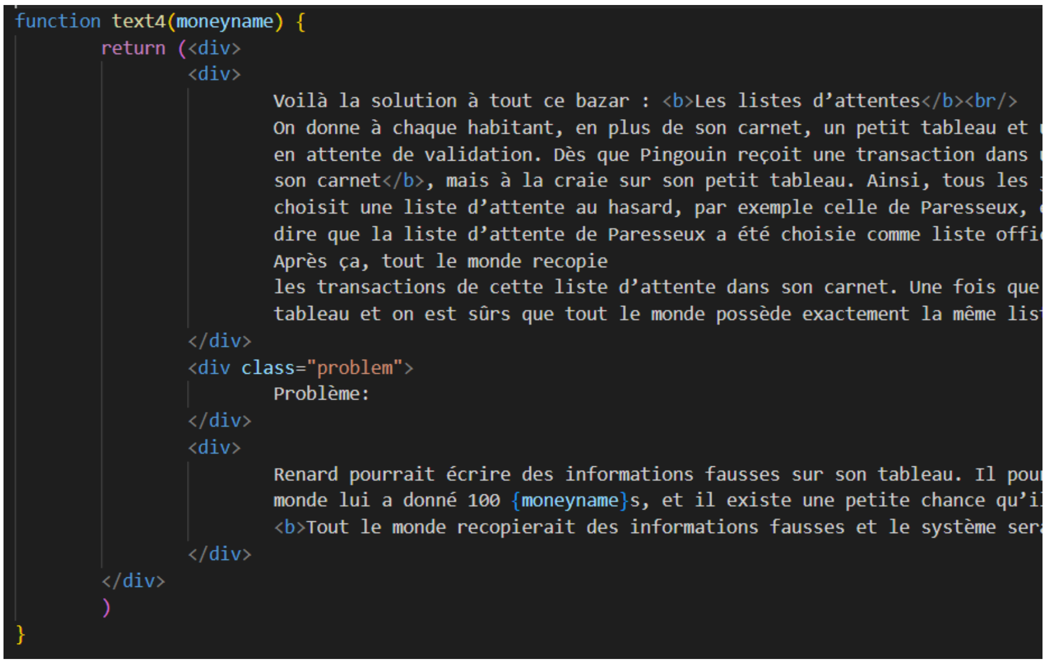
3.1.1 Structure

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



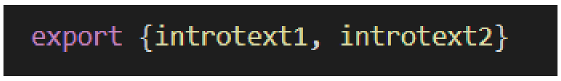
```
function introtext1(){
        return(<div>
                Au beau milieu de la forêt se trouve un village...
        </div>
        )
}
```

JSX expressions can integrate html tags to manipulate the rendering of the text. You can also add a parameter to the function to use it in a block. Don't forget to send the parameter when you call the function.



```
function text4(moneyname) {
        return (<div>
                <div>
                        Voilà la solution à tout ce bazar : <b>Les listes d'attentes</b><br/>
                        On donne à chaque habitant, en plus de son carnet, un petit tableau et
                        en attente de validation. Dès que Pingouin reçoit une transaction dans
                        son carnet</b>, mais à la craie sur son petit tableau. Ainsi, tous les
                        choisit une liste d'attente au hasard, par exemple celle de Paresseux,
                        dire que la liste d'attente de Paresseux a été choisie comme liste offi
                        Après ça, tout le monde recopie
                        les transactions de cette liste d'attente dans son carnet. Une fois que
                        tableau et on est sûrs que tout le monde possède exactement la même lis
                </div>
                <div class="problem">
                        Problème:
                </div>
                <div>
                        Renard pourrait écrire des informations fausses sur son tableau. Il pou
                        monde lui a donné 100 {moneyname}s, et il existe une petite chance qu'i
                        <b>Tout le monde recopierait des informations fausses et le système ser
                </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 Components.js and Classes.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.
This guide will cover the global functioning of the components. For additionnal implementation details please consider reading the JSDoc documentation delivered with the project.

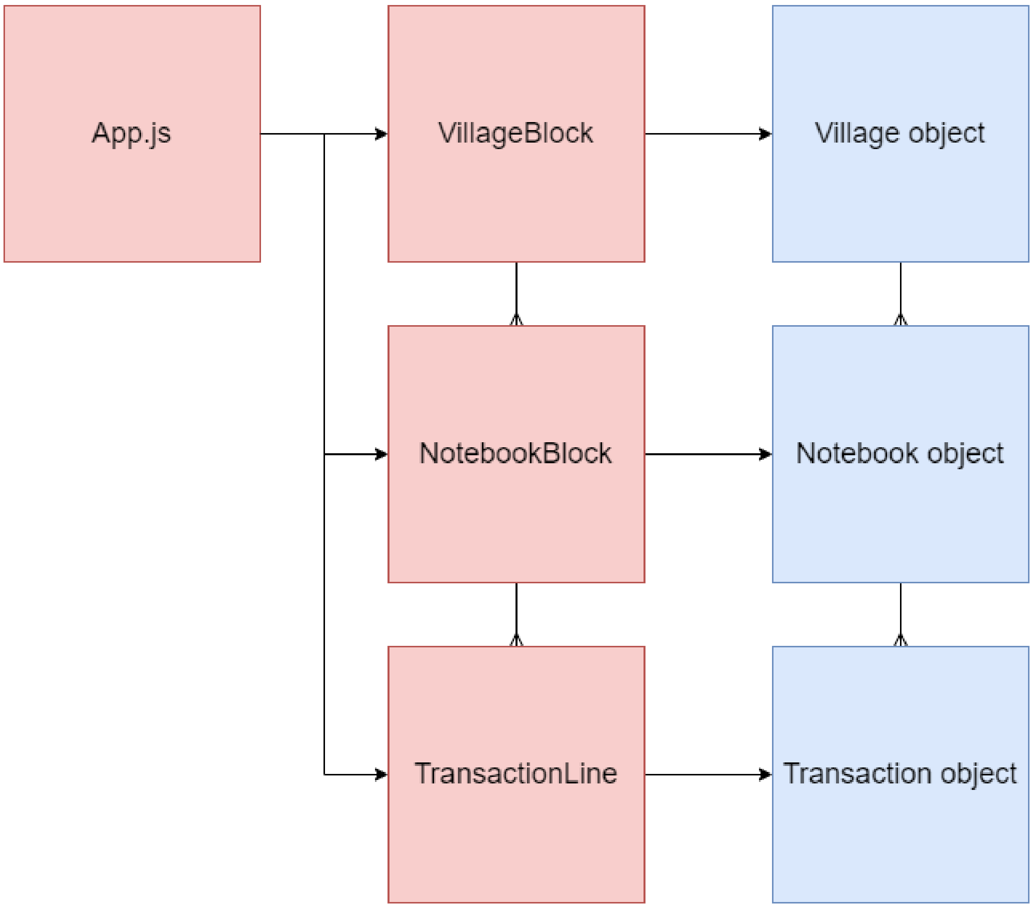
3.2.1 Organization

Classes.js contains the implementation of the necessary JavaScript classes. Components.js contains the implementation of the React components intended for interacting with the user.

The current version of the software defines 3 types of objects to represent the story:

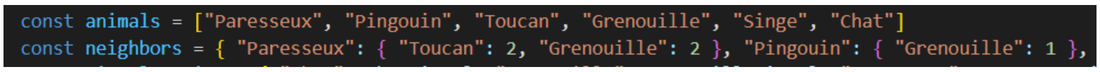
- A **Transaction** represents a transaction of money from one villager to another
- A **Notebook** is held by one villager and represents a set of Transactions among the villagers.
- A **Village** represents a set of Notebooks held by multiple villagers.

Here is a summary of how the component system works:



3.2.2 Managing the villagers

Don't forget to define the different villagers composing the village inside of Components.js too, respecting the following syntax:



```
const animals = ["Paresseux", "Pingouin", "Toucan", "Grenouille", "Singe", "Chat"]
const neighbors = { "Paresseux": { "Toucan": 2, "Grenouille": 2 }, "Pingouin": { "Grenouille": 1 },
```

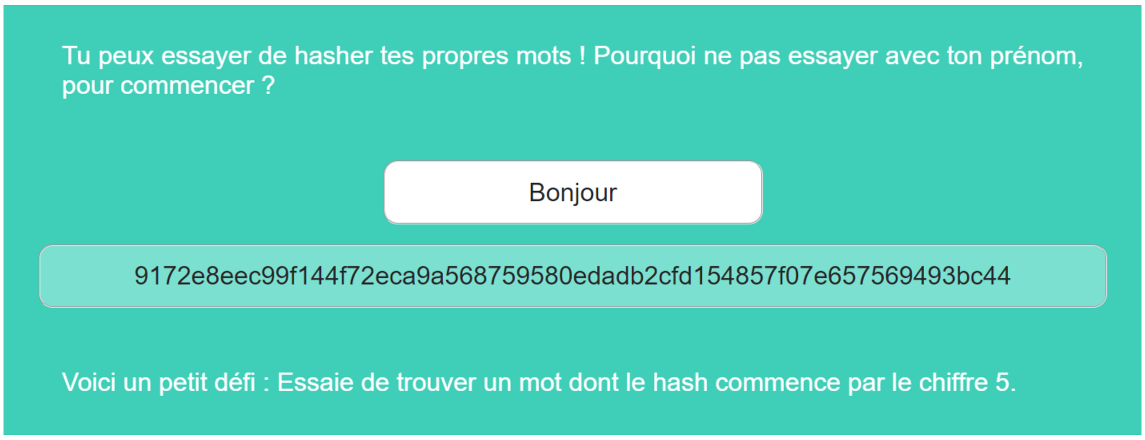
- animals defines a simple list of the different animals composing the village
- neighbors defines which villager is the neighbor of which villager. The associated integer defines the distance between the two neighbors.

3.2.3 Transaction, Notebook, Village machines

To add a new machine, create a new component and pass it the necessary arguments. You can freely use the different methods to manipulate your objects before passing them. Read the official code documentation to use theses functions.

3.2.4 Hashing machine

The hashing machine is independant of the previously discussed components and uses the SHA256 hashing algorithm provided by the crypto.js library to hash a text in real time.



Tu peux essayer de hasher tes propres mots ! Pourquoi ne pas essayer avec ton prénom, pour commencer ?

Bonjour

9172e8eec99f144f72eca9a568759580edadb2cfd154857f07e657569493bc44

Voici un petit défi : Essaie de trouver un mot dont le hash commence par le chiffre 5.

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

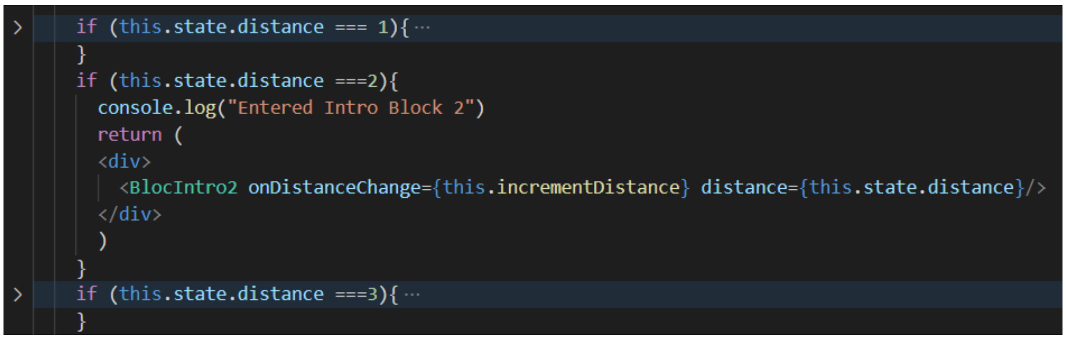
This is the main component of the project, containing all the sub-components. This component contains the main state of the program:

- 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.

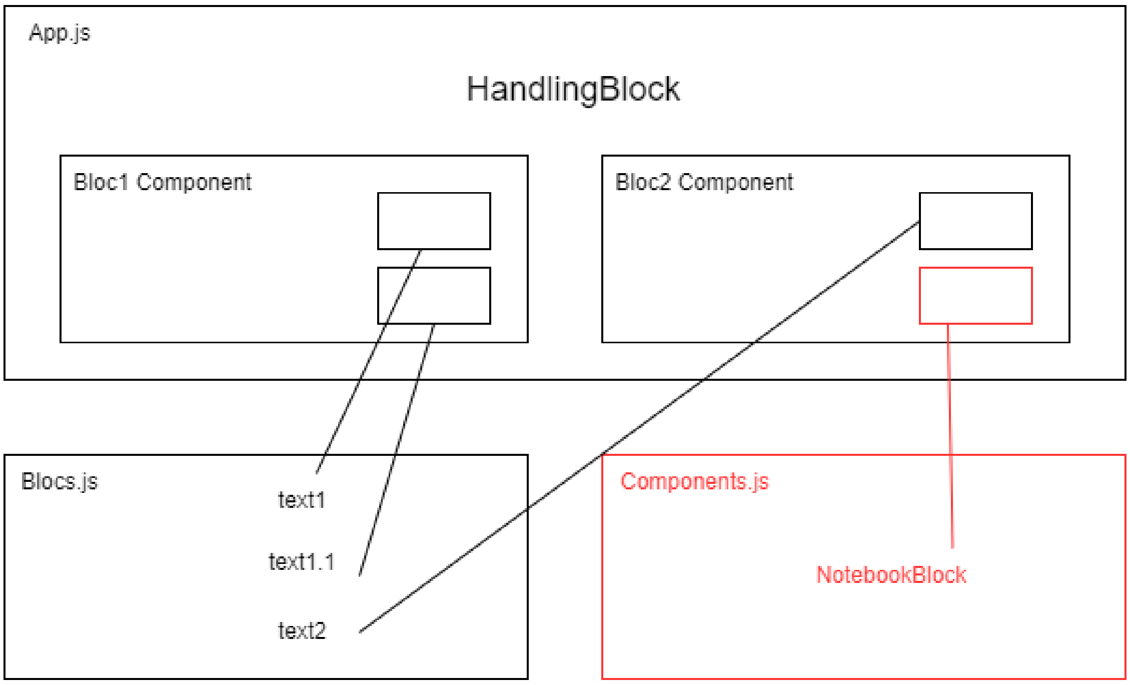
You can send all the additional informations or methods a block could need by passing them arguments.

3.3.4 Blocks components

Each of these components encapsulate one or multiple text blocks from Blocs.js, Components.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 App:



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.

4.1.1 Test a specific part of the code

To test a specific component of the website, edit the Test.js file of the project. You can setup your own environment and add any component you want. Then just set the 'test' boolean in App.js to true to launch the app in test mode.

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 Specification of the classes and components

A documentation of the code was generated using JSDoc. It is available as an attached file. Just open the *index.html* file from the *doc/JSDoc* folder in your web browser.

6 Future updates

The current website already meets many of the challenges. However, these are multiple aspects that still can be improved:

- Organizing and cleaning the css code of the project which contains a small amount of hardcoded behaviour.
- On the visual aspect too: Making the entire website fully responsive.
- An additional interactive machine implementing the hashing mechanics into the current village.

7 Visual credits

Pixel art animals credit goes to Tristan De Meyere and Antoine Van Aelst.
Other illustrations credit goes to Antoine Van Aelst.

8 Contact

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