React & D3

Esteban Sotillo
CPNV-ES

12 avril 2018

1 Introduction

Documentation de l'utilisation de React et D3 afin de pouvoir commencer un projet avec ces technologies de manières simple et rapides.

Pour faire ce projet j'ai travaillé sur une version de linux donc certaines manipulations risque de changer un peu. Les possibles manipulations supplémentaires sont laissées au lecteur.

2 Mise en place

2.1 Prérequis

— npm v5.x+
— Syntaxe ES6+
— React

2.2 Création de la base du projet

Pour la base du projet il est plus pratique d'utiliser le CLI de création d'app React afin d'avoir une base de code consistante avec tous les modules principaux.

```
1  npm i -g create-react-app
2  create-react-app my-app
3  cd my-app
```

2.3 Installation des dépendances

Pour ce projet j'ai utilisé différents packages dont j'expliquerais l'utilité dans les prochaines lignes.

react-move	Utilisé pour aider à l'animation de composants react.
react-chord-diagram	Composant react simplifiant l'utilisation du diagram chord de D3.
d3-ease	Librairie de fonctions servant à définir le flux des transitions.

Pour les installer il suffit d'entrer la commande suivante :

```
1  npm i react-move react-chord-diagram d3-ease
```

2.4 Structure de fichier

Voici un rapide descriptif des fichiers utiles pour commencer le projet.

```
1 + public : fichiers statics modifiable
2 + src : dossier de travail
3   - index.js : fichier js de base
4   - App.js : Composant react originel
```

3 Utilisation

Dans cette partie, je vais faire un petit projet afin d'expliquer plus facilement l'utilisation des composants.

3.1 Importation des composants

Dans le fichier **src/App.js** ajouter ces trois lignes au début du fichier.

```
1  import ChordDiagram from 'react-chord-diagram'
2  import { Animate } from 'react-move'
3  import { easeExpInOut } from 'd3-ease'
```

Puis supprimer le contenu de la première div.

3.2 Composant ChordDiagram

Le composant ChordDiagram possède 4 propriétés que nous allons utiliser qui sont "matrix", "componentId", "groupLabels" et "groupColors".
La propriété "matrix" est un tableau à 2 dimensions comprenant en x la quantité partant et en y la quantité de réception.
La propriété "componentId" est un ID pour reconnaître le diagramme.
La propriété "groupLabels" est un tableau de string utilisé comme label pour le diagramme.
La propriété "groupColors" est un tableau de couleurs css utilisé pour le diagramme.

Exemple :

```
1  <ChordDiagram
2    matrix={[[1,2],[3,4]]}
3    componentId={"ID"}
4    groupLabels={["foo", "boo"]}
5    groupColors={["#333","#567"]}
6  />
```

3.3 Composant Animate

Le composant Animate sers à aider à l'animation des composants react. Il possède 2 propriétés requises qui sont "start" et "update". Ces deux propriétés sont des références de fonction définissant respectivement les status initiaux et les status après execution.

Exemple :

```
<Animate
  start={() => ({
    x: 0
  })}
  update={() => ({
    x: [1],
    timing: { duration: 1500, ease: easeExpInOut },
  })}
>
{state => null /* do something with state.x */}
</Animate>
```

4 Dans le code

4.1 Configuration du composant React

Pour commencer, il faut définir des "state" sur notre composant app (à défaut d'en avoir créer un autre). nous aurons besoin de trois variables différentes qui sont "x", "matrix" et "oldMatrix". Il faut le définir dans le constructeur de notre composant.

```
constructor() {
  this.state = {
    x: 0,
    matrix: [[]],
    oldMatrix: null,
  }
}
```

Il faut ensuite ajouter un bouton (dans la partie rendu) qui sera utilisé pour modifier les données de notre matrice.

```
<div className="App">
  <input type="button" onClick={()=>{}} value="update" />
</div>
```

Ajouter le composant Animate a la suite du bouton.

```
<Animate
  start={() => ({
    x: 0,
  })}
  update={() => ({
    x: [this.state.x],
    timing: { duration: 1500, ease: easeExpInOut },
  })}
>
  {state => {
```

```
      let {x} = state
      // la suite se passera ici
    }}
  </Animate>
```

4.2 Calcul des différences entre les matrices

Afin d'avoir une transition jolie entre la nouvelle et l'ancienne matrice, il faut calculer a l'aide de l'indice x (qui évoluera entre 0 et 1).
Le code se trouve dans le contenu du composant Animate.

```
let {x} = state
let a = this.state.matrix
// si il n'y a pas de matrice pas besoin de calculer
if (this.state.oldMatrix !== null) {
  // transformation de x
  x = this.state.do === 0 ? 1 - x : x

  let base = this.state.oldMatrix
  let change = this.state.matrix
  let isNormal = true

  if (base.length < change.length) {
    [base, change] = [change, base]
    isNormal = false
  }

  a = base.map((row, ri) =>
    row.map((cell, ci) => {
      let cellChange = 0
      let cellBase = cell
      if (change[ri] && change[ri][ci])
        cellChange = change[ri][ci]
      if (!isNormal) {
        cellBase = cellChange
        cellChange = cell
      }
      return Math.round(cellBase + x * (cellChange -
          cellBase))
    })
  )
}
// quand c'est termine, force la matrice
if (x === 1) {
  a = this.state.matrix
}
```

4.3 Rendu du diagramme

Une fois que la matrice est calculée, il suffit de retourner le composant ChordDiagram pour l'afficher.

```
1  return (
2    <ChordDiagram
3      matrix={a}
4      componentId={1}
5      groupLabels={this.labels}
6      groupColors={this.colors}
7    />
8  )
```

NB : les labels et couleurs n'ont pas encore été définies mais le seront dans une prochaine partie.

4.4 Création des données de matrices

Étant donné qu'il n'y a pas de vraies données, il va falloir les générer aléatoirement. Voici une fonction qui va créer une matrice. Il suffit d l'ajouter dans les méthodes de la classe App.

```
1  generateMatrix (size = 4, min = 100, max = 10000) {
2    let item = []
3    for(let i=0; i<size; i++) item.push(0)
4    return item.map(_=> item.map(_=>
5      Math.round(Math.random()*(max-min)+min)
6    ))
7  }
```

4.5 Méthode de màj de matrice

Pour simplifier l'utilisation, il est intéressant de créer une méthode qui sera exécutée quand on veux mettre à jour les données de la matrice.

```
1  updateMatrix () {
2    this.setState({
3      oldMatrix: this.state.matrix,
4      matrix: this.generateMatrix(),
5      x: this.state.x === 0 ? 1 : 0,
6    })
7  }
```

4.6 Finitions

Pour que le tout fonctionne il faut modifier un peu le constructeur et la fonction onClick du bouton.
Constructeur :

```
1  constructor() {
2    this.state = {
3      x: 0,
4      matrix: this.generateMatrix(),
5      oldMatrix: null,
6    }
7    this.labels = ['a', 'b', 'c', 'd']
8    this.colors = ['red', 'yellow', 'green', 'blue']
9  }
```

Bouton :

```
1  <input type="button" onClick={()=> this.updateMatrix() }
       value="update" />
```

4.6.1 Documentations

— https ://reactjs.org/docs/jsx-in-depth.html
— https ://www.npmjs.com/package/react-move
— https ://www.npmjs.com/package/react-chord-diagram