|  |
| --- |
| **Sorting Process Visualizer** |

**React Software Design Document**

**Bo-Hong, Jiang**

**2021/12/13**

# 1.Introduction

## 1.1 Purpose

When learning data structure, Sorting methods are always an important part of the subject. The algorithm itself is easy to find on the internet, such as Google search engine, YouTube… Despite there being tons of resources that teach about math, it is hard to find some websites where you can try the method yourself and give you the animation of the sorting process. So, I decided to write a website that shows the sorting process by animation and make both myself and other software learners get a better understanding of these common sorting methods.

## 1.2 Scope

The program is built with two parts:

1. Toolbar: the toolbar contains four kinds of settings. The first one is to create a new set of random data for the sorting test. The second part is to control data size and animation speed. The third part is the place to switch each algorithm and there will be four methods for the user to choose. The last one is a button that will sort data with user setting and meanwhile play the sorting process animation

2. Sorting Visualizer: there are two functions in this part: a display block displays the sorting process to the user and a timer calculates the animation time which makes the user have a more general understanding of the algorithm efficiency.

## 1.3 Definitions, Acronyms, and Abbreviations

### 1.React-redux:

A JavaScript library that can create a store to manage each state.

### 2.Components:

An element, like JavaScript functions, accepts arbitrary inputs and returns React elements describing what should appear on the screen. There are two kinds of react components: function components and class components. Here I used the function component only.

### 3.Store:

A root component to manage states centrally.

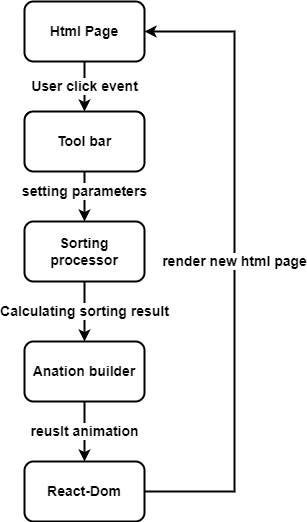
### 4.Slice:

A new function from the redux-toolkit module contains both states and reducers at once and make redux code easier to read.

# 2.System architecture and architecture design

## 2.1 System workflow

Users can click or drag the buttons and input on the HTML page to change the sorting method, sorting speed, data size… and so on. Then Toolbar will accept user click events and transform them to setting parameters and pass them to the processor. When the user clicks Sorting Button, the processor will calculate the result and pass the result animation to the animation builder. Finally, the animation builder will change either the element height or color and render a new HTML page to show the animation of the sorting process.



2-1 System Workflow Diagram

## 2.2 Software Architecture Description

The software was built with two function components and one store which contains three slices.

### Components:

|  |  |  |
| --- | --- | --- |
| Component name: ToolBar | | |
| Brief description: The ToolBar component contains two kinds of event trigger:  1. Buttons: switching sorting method, reset data and start the sorting process, and showing the animation of the sorting process.  2. Range inputs: controlling data size and animation speed. | | |
| **State** | **Description** |
| Size | A state to control array size, value is from 4 to 100 |
| Speed | A state to control animation show speed, value is from 10 to 1000ms per frame |
| isRunning | A state to record whether the program is running or not and keep the user from changing settings during the process |
| **Button/Input** | **Description** |
| Reset Array | A button that can create a new random data set to the array |
| Size | A range input can set the data size of the array and will create a new random array simultaneously |
| Speed | A range input which can change the animation speed |
| Merge Sort | A button that can switch sorting algorithms to merge sort |
| Quick Sort | A button that can switch sorting algorithm to quick sort |
| Heap Sort | A button that can switch sorting algorithm to heap sort |
| Bubble Sort | A button that can switch sorting algorithm to bubble sort |
| Sort | A button that can sort the data in an array and play the sorting process on the screen | |

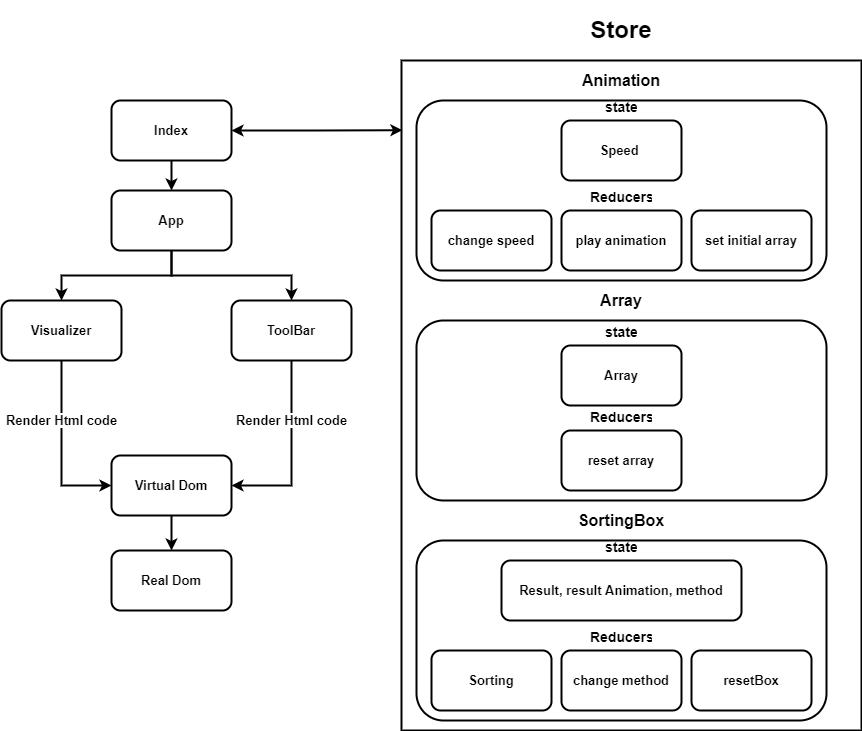
|  |  |
| --- | --- |
| Component name: Visualizer | |
| Brief description: The Visualizer component contains two-part:  1. Body container: Place to show the data and play the animation.  2. Timer: A timer to show how much time it takes to show animation. | |
| **Element** | **Description** |
| Body container | A container contains a set of bar elements with different key, height, and background color |
| Timer | A label that can show much time it takes to show animation |

### Slice

|  |  |
| --- | --- |
| Slice name: Animation | |
| Brief description: A slice to deal with animation, contains one state value and three reducers. The function of this slice is focus on dealing with the sorting process animation. | |
| **State** | **Description** |
| speed | The value is the control of the animation playing speed, it ranges from 10ms to 1010ms. |
| **Reducer** | |
| changeSpeed(state, action) | **Method Description** |
| The method simply changes the state value-speed of the slice itself. |
| **Program Description Language** |
| state.value = action.payload |
| playAnimation(state, action) | **Method Description** |
| The method is to show the animation of the sorting process, it will call the changeBarheight method to play the animation. |
| **Program Description Language** |
| changeBarHeight(payload, speed)  changeBarHeight(animation, timeDelay){  [action, data] = animation  For(i=0;i<animation.length();i++){  Switch(action){  Case ‘situation’:  Element.style.backgroundColor = ‘color’  Element.style.height = ‘{xx}px’  }  } |

|  |  |
| --- | --- |
| Slice name: Array | |
| Brief description: | |
| **State** | **Description** |
| Value | The value is to store the set of random test data. |
| **Reducer** | |
| resetArray(state, action) | **Method Description** |
| The method is to create a new random data set with user design size and store it in the Array slice itself. |
| **Program Description Language** |
| state.value = createRandomArray(payload)  createRandomArray(arraySize){  for(i=0;i<array.length();i++){  array.push(randomIntFromInterval(5, 500))  }  return array  } |
| setInitialArray(action) | **Method Description** |
|  | The method is designed to recover the screen to the initial situation(the original bar height, color), and it will execute when every time the sort button is clicked. |
|  | **Program Description Language** |
|  | recoverArray(payload)  recoverArray(array){  for(i=0;i<array.length();i++){  array[i].style.height = ‘{xx}px’  array[i].style.backgroundColor = ‘color’  }  } |

|  |  |
| --- | --- |
| Slice name: SortingBox | |
| Brief description: | |
| **State** | **Description** |
| sortedResult | The value is to store the data after the sorting process. |
| sortedAnimation | The value is to store the sorting process animation. |
| Method | The value is to store which algorithm was chosen. |
| **Reducer** | |
| sorting(state, action) | **Method Description** |
| This method is to sort the test data with four algorithms: merge sort, quick sort, bubble sort, and heap sort. It will switch the method depending on the value of SortingBox and return the result to both sorted result and sortedAnimation in the slice itself. |
| **Program Description Language** |
| sorting(state, action){  [result, animation] = [[], []]  data = action.payload  Switch(state.value.method){  Case ‘merge’:  [result, animation] = mergeSort(data)  Case ‘quick’:  [result, animation] = quickSort(data)  Case ‘bubble’:  [result, animation] = bubbleSort(data)  Case ‘heap’:  [result, animation] = heapSort(data)  }  state.value = {  sortedResult: result,  sortedAnimation: animation,  method: state.value.method,  }  } |
| changeMethod(state, action) | **Method Description** |
| The method simply changes the state value-method of the slice itself. |
| **Program Description Language** |
| state.value.method = action.payload |
| resetBox(state) | **Method Description** |
| The method simply changes the whole state to the initial state( empty value for all three values). |
| **Program Description Language** |
| state.value = initialStateValue |



2-2 Software Architecture Diagram

# Reference

[1] [React–用來實作使用者介面的JavaScript 函式庫](https://zh-hant.reactjs.org/)

[2] [React Crash Course for Beginners 2021](https://www.youtube.com/watch?v=Dorf8i6lCuk&list=PL0BV7dLxRnWE_frjHBB4YjruagjYGyoev&index=39&t=9630s)

[3] [React Redux Tutorial For Beginners | Redux Toolkit Tutorial 2021](https://www.youtube.com/watch?v=k68j9xlbHHk&list=PL0BV7dLxRnWE_frjHBB4YjruagjYGyoev&index=44)

[4] [Sorting Visualizer Tutorial (software engineering project)](https://www.youtube.com/watch?v=pFXYym4Wbkc&list=PL0BV7dLxRnWE_frjHBB4YjruagjYGyoev&index=38&t=653s)

[5] [Learn useState In 15 Minutes - React Hooks Explained](https://www.youtube.com/watch?v=O6P86uwfdR0&t=775s)

[6] [Data Structures & Algorithms Tutorial in Python](https://www.youtube.com/watch?v=_t2GVaQasRY&list=PLeo1K3hjS3uu_n_a__MI_KktGTLYopZ12)

[7] [Example - XML Legal Document Utility Software Design Document](https://www.oasis-open.org/committees/download.php/24846/Example-SoftwareDesignDocument-LegalXMLUtility.pdf)