Tuto2: Developing User Interfaces with React

Nicolas Médoc   Luxembourg Inst. of Science and Technology

Outline

1. Recall of React concepts

React principles: components

- What is a component ?

React principles: components

- What is a component ?
  - A React **component** is a function responsible to render a part of the user interface
  - the function returns **JSX code**, a combination of HTML and Javascript generating the final HTML page displayed in the browser
  - the **component tree** is the tree structure resulting from the recursive declaration of all components from the root component (App)
  - **pure components:** try to build pure components, i.e. the same input (props + state) render the same output (HTML)

React principles: props

- Why using **props** ?
  <MyChildComponent myProps={myPropsValue}>

React principles: props

- Why using **props** ?
  <MyChildComponent myProps={myPropsValue}>
  - to declare a piece of data sent by the parent component and used by the child component to render its DOM structure
  - to declare the child component to be an observer of any update of this data

React principles: props example

- Declaring props in a component:

```
function MyChildComponent({myProps}){
    // do something with myProps
    return (
        // return JSX component
    )
}
```

- Sending props data to the child component

```
<MyChildComponent myProps={anyValue}>
```

React principles: states

- Why using **states**?
  const [state,setState] = useState(initialState);

React principles: states

- Why using **states**?
  const [state,setState] = useState(initialState);
  - to memorize the data between the rendering cycles
  - to notify the child components to re-render when the state is updated with the setter method
  - to store data which is reactive (updated by external events)
  - avoid storing in a state any data that can be computed with a pure logic (same input => same output) from another state
  - declare a state as deeply as possible in the component tree

React principles: synchronizing components

- How to synchronize multiple components?

React principles: synchronizing components

- How to synchronize multiple components?
  - declare a state in the closest common ancestor
  - declare an event handler function to set the data state in the closest common parent
  - send the handler function to the child components as props
  - the child component that triggers the event call the event handler function to update the state
  - when the state setter function is called in the common ancestor, all the child components are notified to re-render (with updated props values according to the new parent states)
- **Examples:** See exercises 3 and 4 of the first tutorial.

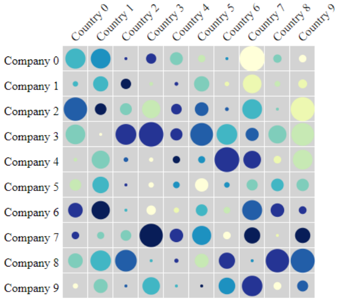
Exercise 5: Visual encoding in a matrix visualization

*Choose an appropriate visual encoding to represent the numerical values in **nbProductSold** and **salesGrowth** (quantity and rate in [0,1] respectively) in each data item.*

Exercise 5: Visual encoding in a matrix visualization

*Choose an appropriate visual encoding to represent the numerical values in **nbProductSold** and **salesGrowth** (quantity and rate in [0,1] respectively) in each data item.*

`https://github.com/nicolasmedoc/TD2-React2`



Immutability of state: how to update arrays and objects?

Anti-pattern: never change object or array content directly to update the state.
Solution: create a copy of objects/arrays and update the copy:

```
const newObject = {...oldObject, key:newValue};
const newArray = [...oldArray, newValue];
const updatedArray = oldArray.map(item=>{
    if (item.id === itemIdToUpdate){
        return {...item, key:newValue};
    }else{
        return item;
    }
});
```

Exercise 6: interactivity in Matrix component

Highlight/unhighlight a matrix cell on click

- In Cell component, declare an event onClick on <g> element. This allows to trigger the click event over the whole area covered by the child elements (<rect> and <circle>).
- In App component, update the state matrixData to change the property 'selected' of the cellData corresponding to the clicked Cell in the list.
- In Cell component, change the attribute stroke and stroke-width of the <circle> element to show a red border if the cell is selected.

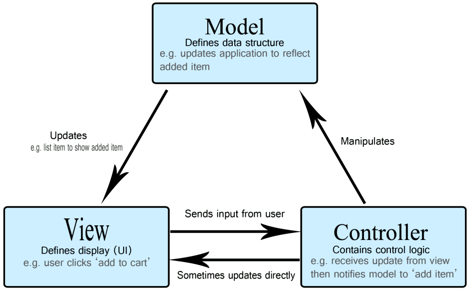
Exercise 7: synchronization of ControlBar with mouse hover interaction in the Matrix

In the Control panel, show the row and column positions of the cell hovered by the cursor, e.g. "hovered cell (1,3)" and "Nothing hovered" otherwise.

- In Cell component, declare an event onMouseEnter and onMouseLeave on <g> element.
- Send the hovered cellData to the ControlBar component and render the message accordingly. You can add a third attribute 'hoveredCell' in genConfig state, initialized with and taking the hovered cellData.

Model View Controller (MVC) design pattern

Does React props and states allows to implement MVC design pattern?


Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'

`https://developer.mozilla.org/en-US/docs/Glossary/MVC`

Limitation of using only props and states

- **No separation of responsibilities:** the model, the controller and the view are all defined in the same component function, *e.g. the data structure (matrixData and matrixConfig) and the event handlers appear together with the UI rendering logic in the App component function.*
- **Complexity and readability of the code:** the declaration of the states used to synchronize components are all declared in the closest common ancestor, which can makes it difficult to maintain and read, *e.g. App.js with many states and functions to update them.*
- **Performance issues for heavy components**: updating a state in the common ancestor triggers unnecessary re-rendering in the children branches, *e.g. updating the states in App.js triggers the re-render of Matrix component on hovering interaction or when the user is typing nbRows/nbCols.*