Tuto3: Developing User Interfaces with React

Nicolas Médoc Luxembourg Inst. of Science and Technology

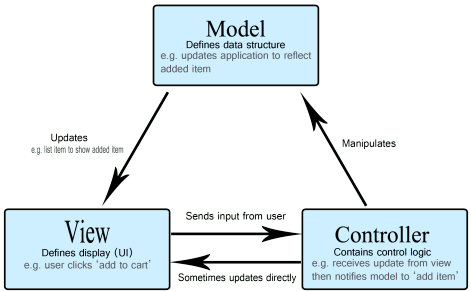
Outline

1. limitations of using only states and props

2. Redux: passing data deeply through the component tree

Model View Controller (MVC) design pattern

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



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 and the event handler of genData and genConfig appear together with the UI rendering logic in the App component function.*
- **Complexity and readability of the code:** the declaration of the states used for the synchronization of the components are all declared in the closest common ancestor, which can makes it difficult to maintain and read, *e.g. App.js.*
- **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 over the cells or on change of nbRows/nbCols.*

Analyzing rendering sequence with logs

Get the branch init-redux of the repository TD2-React2:

```
git clone https://github.com/nicolasmedoc/TD2-React2/
git checkout init-redux
cd TD2-React2
npm install
npm start
```

or

```
https://github.com/nicolasmedoc/TD2-React2/archive/refs/heads/init-redux.zip
cd project_from_extracted_folder
npm install
npm start
```

Check the log to see the sequence of rendering. When we change NbRows/NBCols or when the mouse hover over the cell, the matrix re-render while matrixData doesn't change.

Redux: Why

- for a better separation of responsibilities between the Model, the Controller and the View
- to avoid passing the states and/or event handler as props through all intermediate components when common ancestor is far from the components using them.
- to build a global state (model) with a tree structure closely following the component tree
- to avoid unnecessary re-rendering of all child components from the common ancestor holding the state.
- redux memorizes the state structure and provide reducer actions to update a part of the state. React triggers a re-render only on the components that use this part of the state.

Setup redux (done in branch init-redux)

- Install redux:

```
npm install @reduxjs/toolkit  react-redux
```

- Declare an empty store in store.js:

```
import { configureStore } from '@reduxjs/toolkit'

export default configureStore({
  reducer: {}
})
```

Declare the store provider in index.js (done in branch init-redux)

```
import React from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App'
import store from './app/store'
import { Provider } from 'react-redux'

const root = createRoot(document.getElementById('root')!)

root.render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>,
)
```

Create reducers: states and actions

Create the file src/redux/ConfigSlice.js with:

```
import { createSlice } from '@reduxjs/toolkit'
// createSlice declares the init state and reducer actions for a data slice
export const configSlice = createSlice({
  name: 'config',
  initialState: {nbRows: 4,nbCols: 4, hoveredCell:{}},
  reducers: {
    updateNbRowsAndCols: (state, action) => {
      return {...state, nbRows:action.payload.nbRows, nbCols:action.payload.nbCols};
    },
    updateHoveredCell: (state, action) => {
        return {...state, hoveredCell:action.payload};
    },
  }
})
// Action creators are generated for each case reducer function
export const { updateNbRowsAndCols, updateHoveredCell } = configSlice.actions
// return the reducer by default
export default configSlice.reducer
```

Add the reducer in store.js

```
export default configureStore({
  reducer: {
    config: configReducer,
    }
})
```

useSelector() to get the state in a component

In ControlBar.js:

```
import { useSelector} from "react-redux";
...
function ControlBar(){
    // config being the name used in store.js
    const genConfig = useSelector(state=>state.config);
...
```

useDispatch() to trigger actions

In ControlBar.js:

```
...
import { useSelector, useDispatch } from "react-redux";
import { updateNbRowsAndCols } from "../../redux/ConfigSlice";

function ControlBar(){
    const dispatch = useDispatch();
    const genConfig = useSelector(state=>state.config);
    const handleOnChangeNbRows = function(event){
        const nbRows = parseInt(event.target.value);
        // the paramter object will be payload in the reducer
        dispatch(updateNbRowsAndCols({ ...genConfig, nbRows }))
    }
    const handleOnChangeNbCols = function(event){
        const nbCols = parseInt(event.target.value);
        dispatch(updateNbRowsAndCols({ ...genConfig, nbCols }))
    }
...
```

Test: check the rendering sequence in the logs

- update NbRows/NbCols
- only ControlBar rerenders

Exercise 8: Use dispatch and clean unnecessary props and states

- Use dispatch to handle hoveredCell in Matrix.js.
- The props genConfig is not needed anymore in the ControlBar component, you can remove it.
- The state genConfig is not needed anymore in App.js, you can remove it.
- The props handleHoveredCell is not needed anymore in Matrix and Cell. The function can also be removed in App.js.

Exercise 9: declare Matrix reducer, use it and clean unnecessary props and state

- Open the reducer in src/redux/MatrixSlice.js and see how it is defined
- add this reducer in store.js

```
import matrixReducer from './redux/MatrixSlice'
import configReducer from './redux/ConfigSlice'
export default configureStore({
  reducer: {
    config: configReducer,
    matrix: matrixReducer,
    }
})
```

- use a selector to get matrix data in Matrix component
- use dispatch in ControlerBar to generate new data
- use dispatch in matrix to select a Cell for highlighting
- clean the props and states that are not necessary anymore

App.js is only responsible to render the child components

```
function App() {
  useEffect(()=>{
    console.log("App useEffect");
  })
  return (
    <div className="App">
        {console.log("App rendering")}
        <div id="control-bar-container">
          <ControlBar/>
        </div>
        <div id="view-container">
          <Matrix/>
        </div>
    </div>
  );
}
```

The Matrix component re-renders only when the MatrixData changes

See the sequence of component rendering in the logs:

- when modifying NbRows/NbCols
- when hovering over the cells
- when clicking the cell
- when generating the matrix