React Hooks 3
State Management

Tessema Mengistu (Ph.D.)
Department of Computer Science
Virginia Tech
Mengistu@vt.edu

Outline

- Introduction
- useReducer
- Context
  - useContext

Introduction

- State
  - JavaScript object that represents information about the component's current situation
    - Example: Error or loading
  - A change to a component state causes the component to refresh – re-rendering
  - React supports the management of states
    - Using hooks
      - useState - simple state management
      - useReducer – complex state management
    - Third-party libraries
      - Redux
      - Mobx
      - …

useReducer

- useReducer
  - Store, modify, and access a state variable within a component
  - Uses reducer function
    - Allows a variety of actions to perform on the state value
    - Useful when the next state depends on the previous state

useReducer

- **Accepts** two arguments:
  - A reducer function
  - An initial state value

```
const [state, dispatch] = useReducer(reducer, initialState);
```

useReducer

- ***reducer*** function
  - Specifies how the state gets updated
  - Accepts
    - state
    - action
  - Returns a single value
    - The next state

```
function reducer(state, action) {

   switch (action.type) {

     case '':

        return state1 ;

    case '':

          return  state2;
    default:
         //  do something

    }
```

useReducer

- Returns an array with two values
  - The current state value
  - A special function that updates the state value
    - **dispatch**

```
const [state, dispatch] = useReducer(reducer, initialState);
```

useReducer

- ***dispatch*** function
  - Update the state to a different value and trigger a re-render
  - The **action** object should be passed as a parameter
    - **action** object with a **type** property identifying the action and optionally, other properties with additional information
    - Example:

```
function handleClick() {

      dispatch({ type: 'action type' });

}
```

Context

- Components share data (state) among each other
  - Passing props
    - From parent to child
    - Problematic
      - Props drilling
  - Component composition
  - Using **Context**

Context

- Context
  - provides a way for components to access information stored higher in the component tree without needing to pass props

Context

- Four steps to using React context:
  - Create context using the **createContext** function
  - Take your created context and wrap the context **provider** around your component tree
  - Put any value you like on your context provider using the **value** prop
  - Read that value within any component by using the context **consumer**
    - **useContext**

Context

- **createContext**
  - Returns a context object that contains
    - A Provider
    - A Consumer
  - Example

```
export const ExampleContext = createContext();

export default function App() {
  return (
    <ExampleContext.Provider value="Dr M">
      <User />
    </ExampleContext.Provider>
  )
}
```

Context

```
function User() {

  return (
    <ExampleContext.Consumer>

      {value => <h1>{value}</h1>}
        {/* prints: Dr M */}

    </ExampleContext.Consumer>
  )
}
```

useContext

- The useContext hook:
  - Takes one argument
    - Context object
  - Returns the context value for the calling component
  - Example:

```
const value = useContext(ExampleContext);
```

- Reads and subscribes to a context
  - Should be called at the top level of the component
  - Object destructuring to target the specific context values the component needs

useContext

```
function User() {

  const value = useContext(ExampleContext);

  return <h1>{value}</h1>;
    {/* prints: Dr M */}
}
```

References

- https://react.dev/reference/react/useReducer
- https://react.dev/learn/updating-arrays-in-state
- https://legacy.reactjs.org/docs/context.html