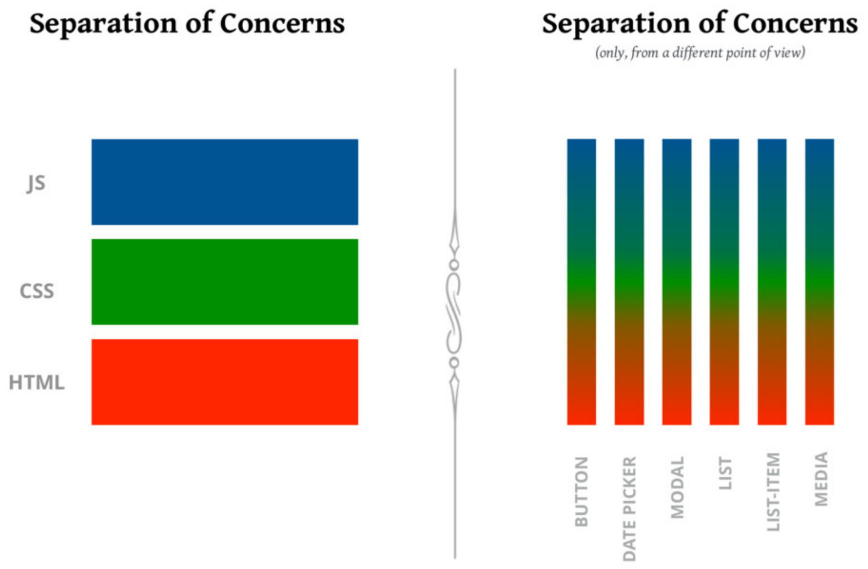
Intro to React (Lecture Notes)

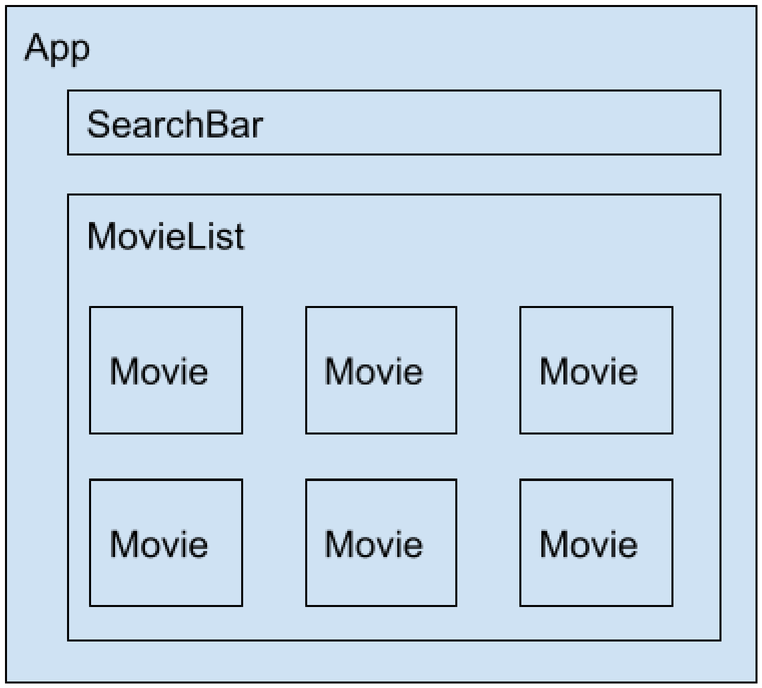
- What is React?
  - Popular, powerful front-end framework for building user interfaces
    - Framework => layer of abstraction; provides a “blueprint” for apps
  - Developed/sponsored by Facebook
  - Opinionated => specified conventions to follow; “rules” for writing code
  - Modular => code is separated into components that encapsulate logic and UI into a single function (vs. vanilla JavaScript that separates logic and UI)
    - Easier to navigate
    - Easier to debug
    - Allows for code re-use (e.g. a navbar that’s used on multiple pages)



- Component Hierarchy
  - Single responsibility principle => a component should ideally only do one thing
  - It’s conventional for the top-level component to be named App
  - Flixster example (if made using React):

```
App
├── SearchBar
└─┬ MovieList
  └── Movie
```

Shown visually as:



- DEMO => create-react-app
  - `npm start` to run app
  - File walkthrough

```
├── README.md               README, can edit or delete
├── package-lock.json       Lock file, don't edit directly
├── package.json            Can edit, as usual
│
├── public                  Rarely need to edit these
│   ├── favicon.ico
│   ├── index.html          Main HTML page of site
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
│
└── src                     Where React stuff goes
    ├── App.css                 CSS for example component
    ├── App.js                  Example component
    ├── App.test.js             Tests for App component
    ├── index.css               Site-wide CSS
    ├── index.js                Start of JS
    ├── logo.svg                React logo
    ├── reportWebVitals.js      Tool for measuring performance
    └── setupTests.js           Starter test configuration
```

- Building our first component
    - Again, a component is a function that includes logic and renders JSX
    - JSX => similar to HTML but is not legal JavaScript
        - Gets transpiled into JavaScript using a compiler
        - More strict than HTML
            - Elements must be explicitly closed to avoid syntax errors
        - Slightly different attribute names than HTML (to avoid JavaScript reserved words)
            - 'className' instead of 'class'
            - 'htmlFor' instead of 'for' (forms)
    - Import and export default vs. non-default data/functions
- File organization (component folders with JavaScript file and associated CSS file)

- Data Flow in React
    - Unidirectional data flow => data flows in one direction, from parent component to child component
        - Properties (props) are used to pass data to components
            - They allow us to configure components so they can be reused
            - Data from props is immutable (cannot be changed)
        - State is used to manage data within a component
            - State is mutable (can be changed) => whenever state changes, the component re-renders
            - Avoid storing data in state which can be calculated from props => React best practice to avoid bugs
            - useState hook => returns a pair of values: the current state and a function that updates it. State is initialized with a starting value.
              ```
              const [count, setCount] = useState(startVal)
              ```

            - Changing State
                - If your new state depends on the previous state, you should use the callback pattern
                  ```
                  const [num, setNum] = useState(0);
                  function clickUp() {
                      setNum(n => n + 1);
                  }
                  ```

                - When working with mutable data structures (array, object, etc.) make a new copy of the data structure, using the spread operator
                  ```
                  const [circles, setCircles] = useState([]);

                  function addCircle(newColor) {
                      setCircles(circles => [
                        ...circles,
                        { color: newColor },
                      ]);
                    }
                  ```

- Conditionals in JSX
    - Ternary operator (instead of if/else blocks)
      `condition ? exprIfTrue : exprIfFalse`

    - JavaScript logical && operator
      `condition && exprIfTrue`

- Creating JSX via Iteration
    - `map()` is used to iterate through items in an array and returns a new array

      ```
      const nums = [1, 4, 9, 16];
      const numsDouble = nums.map(x => x * 2);

      console.log(numsDouble);
      // output: [2, 8, 18, 32]
      ```

    - In React, each child in an array should have a unique "key" prop to avoid reconciliation issues => allows the UI to update more efficiently

- Events in React
    - Any event you can listen for in JavaScript, you can listen for in React
        - Mouse events: `onClick`, `onMouseOver`, etc.
        - Form events: `onSubmit`, etc.
        - Keyboard events: `onKeyDown`, `onKeyUp`, `onKeyPress`
        - Full list
    - Event listeners expect to receive functions as values => do not invoke your function when you pass it to an event listener

- React Forms
    - HTML form elements work differently than other DOM elements in React
    - Controlled components => a technique to access the input field values in React
        - In HTML, form elements typically maintain their own state and update it based on user input
        - In React, form state is maintained in the state of the component and is updated with the setter function returned from `useState()`
    - Having an uncontrolled element/component in your React application can lead to unwanted behavior and bugs
    - `handleChange` runs on every keystroke and updates React state; the displayed value will update as the user types
        - To handle multiple controlled inputs:
            - Add HTML name attribute to each JSX input element
            - Keys in state must match input name attributes
    - `handleSubmit` normally includes a function to update parent component state
        - Parent passes a state setting function as a prop to child component
        - Child component calls this method, updating the parent's state