

CS 312 Software Development

Introduction to React: Components

Component based web design

CS 312 Software Development

- Home
- Course Info
- Lectures
 - Lecture 00 - Intro
 - Lecture 01 - JS
 - Lecture 02 - Tools
- Assignments
- Practicals
- Project
- Resources

CS 312 - Course Information

Professor Christopher Andrews
Office 215 75 Shannon Street
Email candrews@middlebury.edu
Office hours M 1:30p-3:30p, Th 2:30p-4:30p, F 1:30p-2:30p or by appointment
Website <http://go.middlebury.edu/cs312>
Class meetings Rm 224 75SHS -- Section A: TTh 9:35a-10:50a, Section B: TTh 11:10a-12:25a

Course Objectives

At the completion of the course you should be able to:

1. Describe and employ modern methodologies for managing SW development, specifically Agile and Scrum
2. Use tools and services that support those processes, such as version control, GitHub, continuous integration, etc.
3. Describe and employ SW development principles, patterns and best-practices, such as design patterns, SOLID, test-driven development (TDD), etc.
4. Describe, evaluate and employ technologies for full stack web development and single page web applications (SPAs)
5. Complete a large software development project as part of a team

Class deliverables

There will be four different kinds of deliverables in this class:

Practical exercises: Throughout the semester, there will be a collection of in-class exercises where you will work through examples yourselves. The purpose of these is to give you practice working through concrete examples of the things you see in lecture.

Assignments: In the first half of the semester there will also be a collection of homework assignments. These will be longer, more involved, and will require more thought and synthesis on your part. These are designed to both deepen your understanding and allow you to demonstrate your understanding of the material.

Weekly assessments: Every week, we will do a short assessment exercise. These will primarily consist of short answer questions about the material covered up to that point in the class. Each will be marked as belonging to one of the three major topic areas of the class: "skills" (technical implementation, e.g., writing a short function or debugging an implementation), "concepts" (more theoretical questions, e.g., demonstrating knowledge of different patterns, describe pros and cons of different kinds of databases), and "process" (questions about the software development process, e.g., writing user stories, describing the pros and cons of different kinds of testing).

Project: The focus of this class is the final project. In the second half of the semester you will undertake a large software development project as part of team of approximately 6 students (depending on class size). You will need to be in frequent contact with your group and actively contributing as a software developer each week.

Component based web design

Listing for individual books

Pager

multi-item carousel

small item view

Color picker example

red: 151

green: 187

blue: 214

Frameworks

 BACKBONE.JS



- Event based (e.g., Backbone, Vue)
 - Changing the data triggers an event
 - Views register event handlers

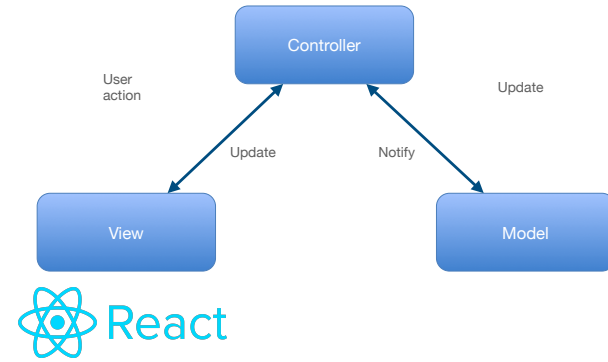
 ANGULAR

- Two-way binding (e.g. Angular)
 - Assigning to a value propagates to dependent components and vice versa

 React

- Efficient re-rendering (e.g. React)
 - Re-render all subcomponents when data changes

Model View Controller (MVC)

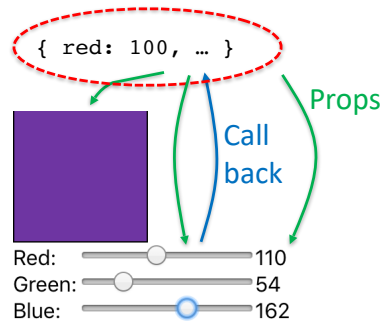


CS 312 Software Development

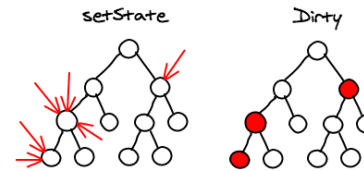
Introduction to React: Fundamentals

Philosophy of React

- There is a single source of truth (the state)
- Render the UI as it should appear for any given state of the application
- Update the state as a result of user actions
- Repeat (i.e., re-render the UI with the new state)

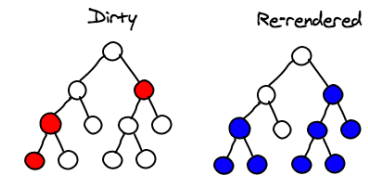


React mechanics



Batched updates

Sub-tree re-rendering



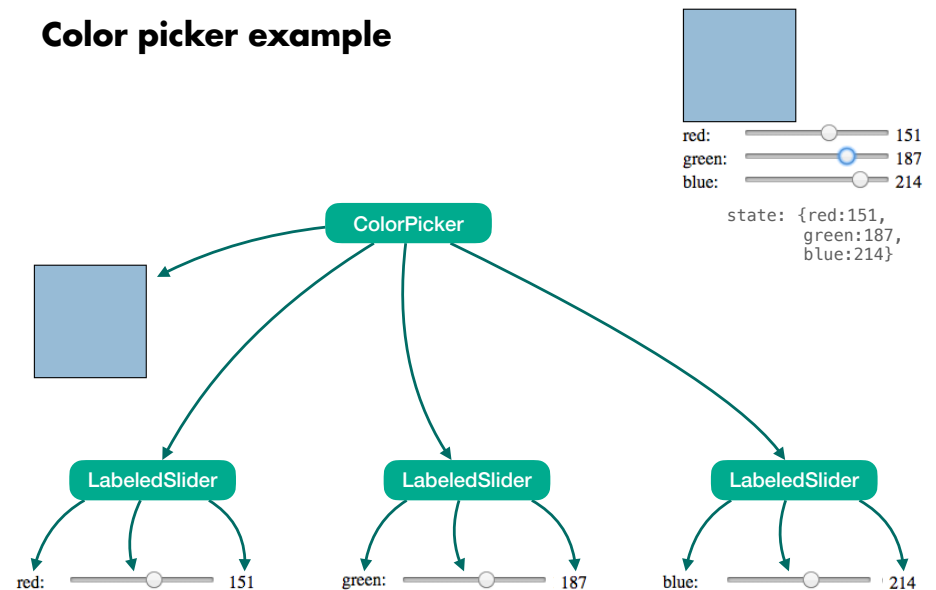
<https://calendar.perfplanet.com/2013/diff/>

Thinking in React

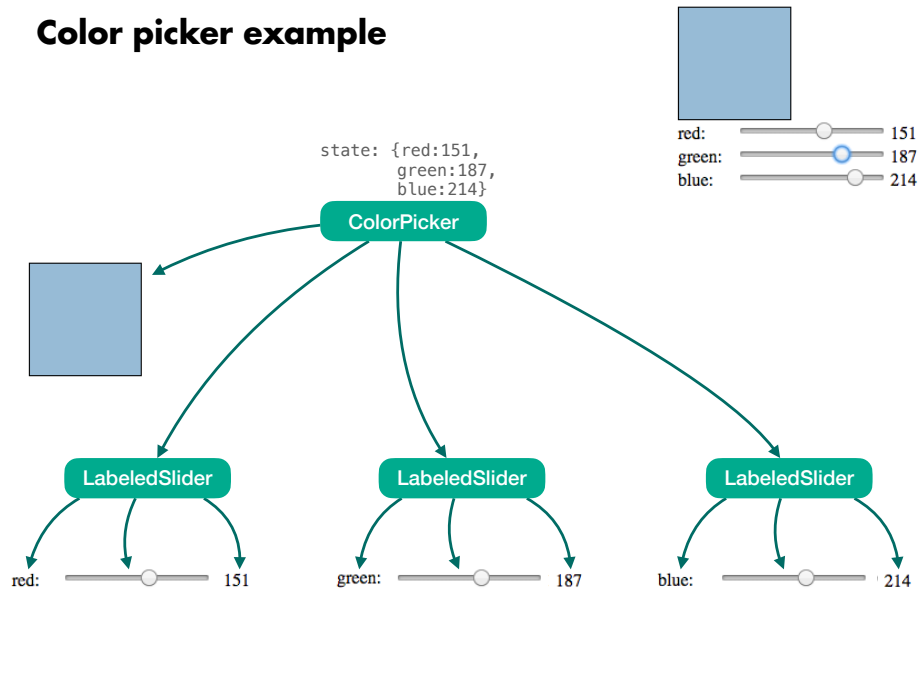
1. Break the UI into a component hierarchy
2. Build a static version in React
3. Identify the minimal (but complete) representation of state
4. Identify where your state should live
5. Add “inverse” data flow (data flows down, callbacks flow up)

<https://reactjs.org/docs/thinking-in-react.html>

Color picker example



Color picker example



CS 312 Software Development

Introduction to React: Writing code

ColorPicker

```
function ColorPicker() {
  const [red, setRed] = useState(0);
  const [green, setGreen] = useState(0);
  const [blue, setBlue] = useState(0);

  const color = {background: `rgb(${red}, ${green}, ${blue})`};
  return (
    <div className="color-picker">
      <div className="color-swatch" style={color} ></div>
      <LabeledSlider label="red" value={red} setValue={setRed}/>
      <LabeledSlider label="green" value={green} setValue={setGreen}/>
      <LabeledSlider label="blue" value={blue} setValue={setBlue}/>
    </div>
  );
}
```

ColorPicker

State with React Hooks (useState)

```
function ColorPicker() {
  const [red, setRed] = useState(0);
  const [green, setGreen] = useState(0);
  const [blue, setBlue] = useState(0);

  const color = {background: `rgb(${red}, ${green}, ${blue})`};
  return (
    <div className="color-picker">
      <div className="color-swatch" style={color} ></div>
      <LabeledSlider label="red" value={red} setValue={setRed}/>
      <LabeledSlider label="green" value={green} setValue={setGreen}/>
      <LabeledSlider label="blue" value={blue} setValue={setBlue}/>
    </div>
  );
}
```

`const [currentValue, setter] = useState(initial value)`

destructuring assignment

ColorPicker

```
function ColorPicker() {
  const [red, setRed] = useState(0);
  const [green, setGreen] = useState(0);
  const [blue, setBlue] = useState(0);

  const color = {background: `rgb(${red}, ${green}, ${blue})`};
  return (
    <div className="color-picker">
      <div className="color-swatch" style={color} ></div>
      <LabeledSlider label="red" value={red} setValue={setRed}/>
      <LabeledSlider label="green" value={green} setValue={setGreen}/>
      <LabeledSlider label="blue" value={blue} setValue={setBlue}/>
    </div>
  );
}
```

template literal

ColorPicker

```
function ColorPicker() {
  const [red, setRed] = useState(0);
  const [green, setGreen] = useState(0);
  const [blue, setBlue] = useState(0);

  const color = {background: `rgb(${red}, ${green}, ${blue})`};
  return (
    <div className="color-picker">
      <div className="color-swatch" style={color} ></div>
      <LabeledSlider label="red" value={red} setValue={setRed}/>
      <LabeledSlider label="green" value={green} setValue={setGreen}/>
      <LabeledSlider label="blue" value={blue} setValue={setBlue}/>
    </div>
  );
}
```

ColorPicker

Passing props

```
function ColorPicker() {
  const [red, setRed] = useState(0);
  const [green, setGreen] = useState(0);
  const [blue, setBlue] = useState(0);

  const color = {background: `rgb(${red}, ${green}, ${blue})`};
  return (
    <div className="color-picker">
      <div className="color-swatch" style={color} ></div>
      <LabeledSlider label="red" value={red} setValue={setRed}/>
      <LabeledSlider label="green" value={green} setValue={setGreen}/>
      <LabeledSlider label="blue" value={blue} setValue={setBlue}/>
    </div>
  );
}
```

passing state and setter as props

in JSX, we surround JS with {}

LabeledSlider

```
function LabeledSlider({ label, value, setValue }) {
  return (
    <div>
      <div className="color-label">{label}</div>
      <input type="range"
        min="0"
        max="255"
        value={value}
        onChange={(event) => {setValue(parseInt(event.target.value, 10))}} />
      <span>{value}</span>
    </div>
  );
}
```

LabeledSlider

Props

single destructured argument "props"

```
function LabeledSlider({ label, value, setValue }) {
  return (
    <div>
      <div className="color-label">{label}</div>
      <input type="range"
        min="0"
        max="255"
        value={value}
        onChange={(event) => {setValue(parseInt(event.target.value, 10))}} />
      <span>{value}</span>
    </div>
  );
}
```

LabeledSlider

Controlled components

```
function LabeledSlider({ label, value, setValue }) {
  return (
    <div>
      <div className="color-label">{label}</div>
      <input type="range"
        min="0"
        max="255"
        value={value}
        onChange={(event) => {setValue(parseInt(event.target.value, 10))}} />
      <span>{value}</span>
    </div>
  );
}
```

Form elements (like input) are **controlled**

We exploit React's re-render loop to get interaction while maintaining a single source of truth