

ReactJS.

Fundamentals

Agenda

- **Background.**
- **The V in MVC**
- **JSX (JavaScript Extension Syntax).**
- **Developer tools..**
- **React Component basics.**
- **(Material Design.)**

ReactJS.

- **A Javascript framework for building dynamic Web User Interfaces.**
 - **A Single Page Apps technology.**
 - **Open-sourced in 2012.**



- **Client-side framework.**
 - **More a library than a framework.**

Before ReactJS.

- MVC pattern – **The convention for app design. Promoted by market leaders**, e.g. **AngularJS (1.x), EmberJS, BackboneJS**.
- **React is not MVC, just V.**
 - **It challenged established best practice (MVC).**
- Templating – **widespread use in the V layer.**
 - **React based on components.**

	Templates	(React) Components
Separation of concerns	Technology (JS, HTML)	Responsibility
Semantic	New concepts and micro-languages	HTML and Javascript
Expressiveness	Underpowered	Full power of Javascript

Components

- **Philosophy:** *Build components, not templates.*
- **All about the User Interface (UI).**
 - **Not about business logic or the data model (Mvc)**
- **Component - A unit comprised of:**
 - UI description (HTML) + UI behavior (JS)*
 - **Two aspects are tightly coupled and co-located.**
 - **Pre-React frameworks decoupled them.**
 - **Benefits:**
 - 1. Improved Composition.**
 - 2. Greater Reusability.**

Creating the UI description

- React.createElement() – **create a HTML element.**
- ReactDOM.render() – **attach an element to the DOM.**
- React.createElement() **arguments:**
 1. **type (h1, div, button etc).**
 2. **properties (style, event handler etc).**
 3. **children (0 -> M).**
 - **We never use createElement() directly – too cumbersome.**
- ReactDOM.render() **arguments:**
 1. **element to be displayed.**
 2. **DOM node on which to mount the element.**

Code Demos

A screenshot of a web browser window displaying a file directory listing. The address bar shows the URL `127.0.0.1:5500`. The directory structure is as follows:

```
~ /  
  js  
  02-UIDescription.html  
  05-simpleComponent.html  
  README  
  node_modules  
  03-JSX-error.html  
  package-lock.json  
  style.css  
  01-UIDescription.html  
  04-JSX.html  
  package.json
```

Below the file listing, a blue callout box contains the text:

See Archive accompanying these slides.
Run code using VS Code Live Server extension

UI description implementation

(the imperative way)

- **See the demos:**
 - **Ref.** 01-UIDescription.html.
 - **Nesting createElement() calls - Ref.** 02-UIDescription.html
-

Imperative programming is a programming paradigm that uses statements that change a program's state

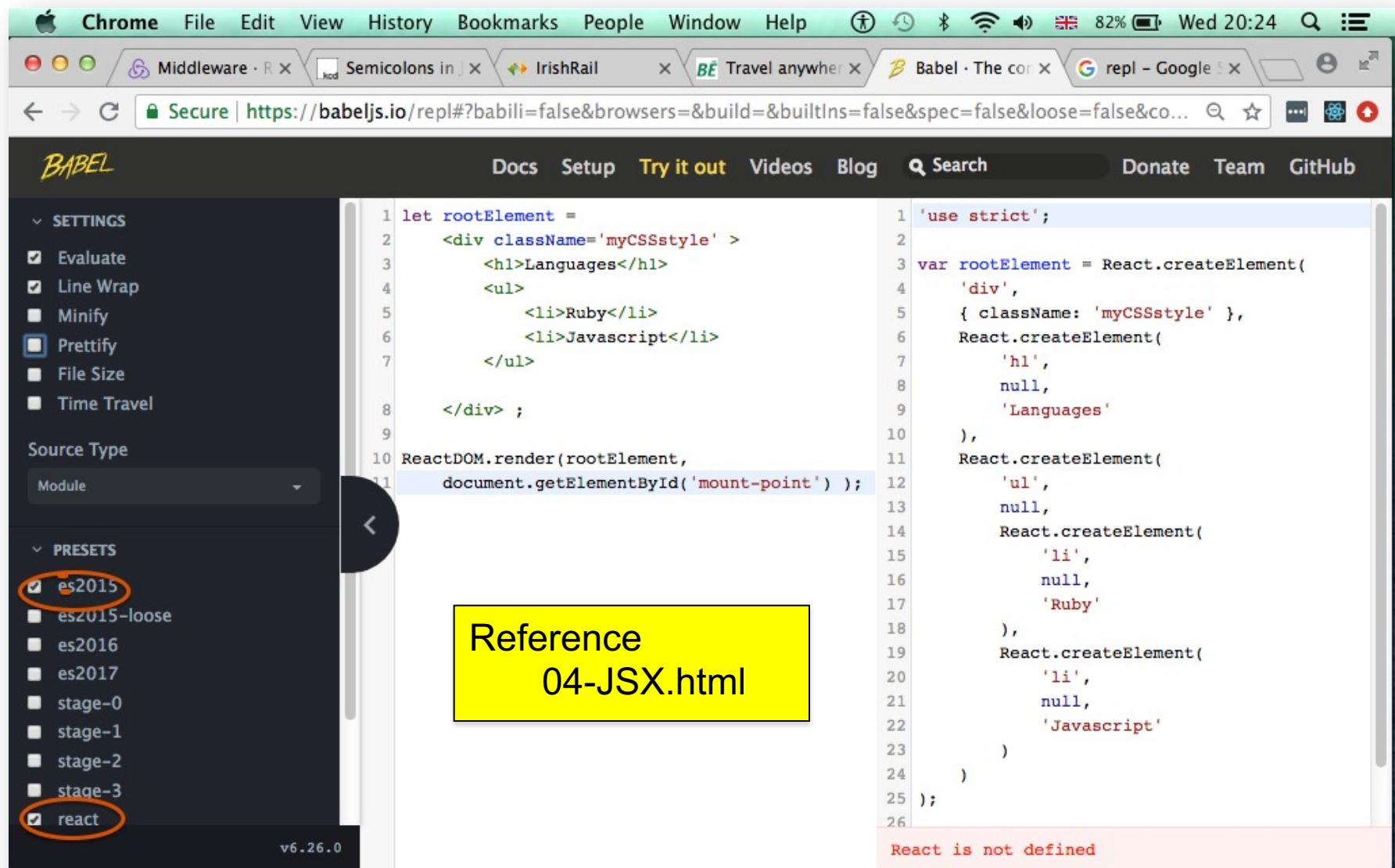
Declarative programming is a programming paradigm ... that expresses the logic of a computation without describing its control flow.

UI description implementation

(the declarative way)

- **JSX – JavaScript extension syntax.**
- **Declarative syntax for coding UI descriptions.**
- **Retains the full power of Javascript.**
- **Allows tight coupling between UI behavior and UI description.**
- **Must be transpiled before being sent to browser.**
 - The Babel tool
- **Reference** 03-JSX-error.html and 04-JSX.html

REPL (Read-Evaluate-Print-Loop) transpiler.



JSX.

- **HTML-like markup.**
 - It's actually XML code.
- Some minor HTML tag attributes differences, e.g. `className` (`class`), `htmlFor` (`for`).
- Allows UI description to be coded in a declarative style and be inlined in JavaScript.
- Combines the ease-of-use of templates with the power of JS.

Transpiling JSX.

- **What?**
 - The **Babel** platform.
- **How?**
 - 1. Manually, via REPL or command line.**
 - When experimenting only.
 - 2. Using specially instrumented web server during development mode - the Webpack library..**
 - 3. Using bundler tools as part the build process before deployment – Webpack again.**

React Components.

- **We develop COMPONENTS.**
 - A JS function **that returns a UI description**, i.e. JSX.
- **Can reference a component like a HTML tag.**
e.g. `ReactDOM.render(<ComponentX />, . . .)`
- **Reference** 05-simpleComponent.html

React Developer tools.

- **create-react-app (CRA) - Features:**
 - **Scaffolding/Generator.**
 - **Development web server: auto-transpilation on file change + live reloading.**
 - **Builder: build production standard version of app, i.e. minification, bundling.**
- **Storybook - Features:**
 - **A development environment for React components.**
 - **Allows components be developed in isolation.**
 - **Promotes more reusable, testable components.**
 - **Quicker development – ignore app-specific dependencies.**



- **Installation:**

```
$ npm install @storybook/react
```

- **Tool has two aspects:**

1. **A web server.**

```
$ ./node_modules/.bin/start-storybook -p 6006 -c ./storybook
```

- **Performs live re-transpilation and re-loading.**

2. **Web browser user interface.**



- **Storybook User interface.**

A screenshot of a web browser displaying the Storybook user interface. The title bar shows multiple tabs: "Storybook", "Middleware · R", "Semicolons in J", and "Travel anyw". The address bar shows the URL: "localhost:9001/?selectedKind=DynamicLanguages&selectedStory=nor".

The main content area has a header "STORYBOOK" and a "Filter" input field. On the left, there's a sidebar with a list of components: "DynamicLanguages", "OtherComponentX", and "OtherComponentY". A red arrow points from the "DynamicLanguages" item in the sidebar to a list of languages on the right: "Python" and "Javascript".

A blue callout bubble labeled "Component catalogue" points to the sidebar. Another blue callout bubble labeled "Component Rendering." points to the list of languages.

Dynamic Languages

- Python
- Javascript

Component catalogue

Component Rendering.



- **What is a Story?**
- **A component may have several STATES → State effects how it renders.**
 - **Each state case termed a STORY.**
 - **Stories are a design consideration.**
- **EX.: DynamicLanguages component.**
 - **States might be:**
 - Default – **5 or less languages → Render full list**
 - Boundary – **empty list → Render ‘No languages’ message**
 - Exceptional – **More than 5 languages → Render first 5 and a ‘See More...’ link to display next 5.**

STORYBOOK

- List a component's states/stories under its name:

The screenshot shows a web browser window displaying the Storybook application at the URL `localhost:9001/?selectedKind=DynamicLanguages&selectedStory=default`. The browser has a back button, forward button, and a refresh button. The main content area has a header "STORYBOOK" and a "Filter" input field. On the left, there is a sidebar menu with the following items:

- DynamicLanguages
 - default
 - boundary
 - exceptional** (highlighted with a blue arrow from a callout bubble)
- OtherComponentX
- OtherComponentY

A blue callout bubble labeled "Set of Stories" points to the "exceptional" item in the sidebar. On the right, the main content area has a large heading "Dynamic Languages" and a bulleted list of languages:

- Python
- Javascript
- Ruby
- PHP
- Groovy

[See more ...](#)



- Define component groups when component library is large.
 - helps others team members with searching.

A screenshot of a web browser displaying the Storybook application. The title bar shows multiple tabs: "Storybook", "Middleware · R", "Semicolons in J", and "Tr". The main content area has a header "STORYBOOK" and a "Filter" input field. On the left, there's a sidebar with a tree view of components:

- Group A
 - OtherComponentX
 - DynamicLanguages
 - default
 - boundary
 - exceptional** (highlighted in grey)
- Group B
 - OtherComponentY

The "exceptional" node under Group A is highlighted with a grey background. To the right, the main content area displays the heading "Dynamic Languages" and a bulleted list of dynamic languages:

- Python
- Javascript
- Ruby
- PHP
- Groovy

Below the list is a link "[See more ...](#)".

Aside – JS Modularity.

- Split application code into multiple files, termed **modules**.
- Reusability - make modules available to other modules..
- Pre-ES6 provided module system via separate library;
- ES6 modules built into language.
 - Two main options: Default exports; Named exports.

- Also Mixed exports.

```
29 // lib.js
30 export default function() {
31   ...
32 }
33 // -----
34 // bar.js
35 import myFunc from './lib';
36 myFunc();
37
```

```
// lib.js (Named exports)
export let myStr = 'important string';
export const pi = 3.14159526;
export function myFunc() {
  ...
}
export class myClass {
  ...
}
// -----
// bar.js
import {myFunc, myStr} from './lib';
myFunc();
console.log(myStr)
```

Writing stories

- **.stories.js file extension (convention)**
- **1 Stories file per component**

```
import React from "react";
import DynamicLanguages from "../components/dynamicLanguages";

export default {
  title: "Dynamic Languages",
  component: DynamicLanguages,
};

export const Default = () => {
  const list = ["Javascript", "Python", "Java", "C#"];
  return <DynamicLanguages languages={list} />;
};

export const Exceptional = () => {
  .....
};

export const Error = () => {
  .....
};
```

default export; Metadata; How Storybook lists components.

- Story implemented as a function.
- Named exports.
- UpperCamelCase
- 3 stories for this component

Writing stories (the old way)

- Fluent-style syntax for writing stories.
 - Method chaining programming style.



```
1 import React from 'react';
2 import { storiesOf } from '@storybook/react';
3 import DynamicLanguages from '../components/dynam:
4
5 storiesOf('DynamicLanguages', module)
6   .add('default',
7     () => {
8       let languages = ['Python', 'Javascript', 'Ruby']
9       return <DynamicLanguages list={languages} />
10      }
11    )
12   .add('boundary',
13     () => . . . .
14   )
15   .add('exceptional',
16     () => . . . .
17   )
18
19 storiesOf('OtherComponentX', module)
20   .add('state 1',
21     () => . . . .
22   )
23   . . . .
```

Grouping stories.

- Use directory pathname symbol (/) to indicate component grouping (i.e. group/subgroup/....).

```
export default {  
  title: "Group A/ Component 1",  
  component: Component1,  
};  
  
... stories ...
```

```
export default {  
  title: "Group A/ Component 2",  
  component: Component2,  
};  
  
... stories ...
```

```
export default {  
  title: "Group B/ Component X",  
  component: Component1,  
};  
  
... stories ...
```

... back to components . . .

Demo Samples

The image shows two side-by-side views. On the left is the Storybook interface, displaying a sidebar with 'EXERCISES' and 'SAMPLE' sections. Under 'SAMPLE', 'Basic' is selected, showing sub-options like '02 - JSX embedded variable'. On the right is a file explorer showing a project structure named 'BASICREACTLAB'. The structure includes a '.storybook' folder, a 'node_modules' folder, and a 'src' folder containing 'components' and 'samples' sub-folders. Inside 'samples' are files named '01_staticComponent.js', '02_embeddedVar.js', '03_props.js', '04_iteration.js', '05_hierarchy.js', and '06_state.js'. Below these is a 'stories' folder with 'exercises' and 'samples' sub-folders. At the bottom are files for version control and dependencies: '.gitignore', 'package-lock.json', and 'package.json'. Callout boxes with arrows point from specific elements in the Storybook sidebar and file explorer to their corresponding counterparts in the other view.

- Configuration – boilerplate** points to the '.storybook' folder in the file explorer.
- Lab exercise** points to the 'exercises' folder in the 'samples' section of the file explorer.
- Sample Components** points to the files in the 'samples' section of the file explorer.
- Stories** points to the 'stories' folder in the file explorer.

Storybook Sidebar:

- Find components search bar
- EXERCISES
 - 01 - static component
 - Basic
 - 02 - JSX embedded variable
 - 03 - component with props
 - 04 - iteration
- SAMPLE
 - 01 - static component
 - Basic
 - 02 - JSX embedded variable
 - 03 - component with props
 - 04 - Component collection (Iteration)
 - 06 stateful component

File Explorer:

- .storybook
- node_modules
- src
 - components
 - exercises
 - samples
 - 01_staticComponent.js
 - 02_embeddedVar.js
 - 03_props.js
 - 04_iteration.js
 - 05_hierarchy.js
 - 06_state.js
 - stories
 - exercises
 - samples
- .gitignore
- package-lock.json
- package.json

JSX - embedded variables.

- Dereference variable embedded in JSX using {} braces.
 - Braces can contain any valid JS expression.
- Reference samples/02_embeddedVariables.js

```
JS 02_embeddedVar.js ×  
components > samples > JS 02_embeddedVar.js > ...  
1 import React from "react";  
2  
3 const Demo = () => {  
4   const languages = ["Go", "Julia", "Kotlin"];  
5   const header = "Modern";  
6   return (  
7     <div>  
8       <h1>{` ${header} Languages`} </h1>  
9       <ul>  
10         <li>{languages[0]}</li>  
11         <li>{languages[1]} </li>  
12         <li>{languages[2]} </li>  
13       </ul>  
14     </div>  
15   );  
16 };  
17  
18 export default Demo
```

Reusability.

- **We achieve reusability through** parameterization.
- **props – Component properties / attribute / parameters.**
 1. **Passing props to a component:**
`<CompName prop1Name={value} prop2Name={value} . . . />`
 2. **Access inside component via props object:**
`const ComponentName = (props) => {
 const p1 = props.prop1Name

}`
 3. **Props are Immutable.**
 4. **Part of a component's design.**
- **Reference** samples/03_props.js (and related story).

Aside – Some JS features

- When an arrow function has only ONE statement, which is its return value, then you may omit:
 - Body curly braces; ‘return’ keyword.

```
const increment = (num) => {  
    return num + 1  
}
```

```
const increment = (num) => num + 1
```

Aside – Some JS features

- The **Array map method** – returns a new array based on applying the function argument to each element of the source array.

```
1 let frameworks = [
2   {name: 'React', url : 'https://facebook.github.io/react/'},
3   {name: 'Vue', url : 'https://vuejs.org/'},
4   {name: 'Angular', url : 'https://angularjs.org/'}
5 ];
6 const names = frameworks.map((f,index) => `${index+1}. ${f.name}`)
7 console.log(names)
8 // [ '1. React', '2. Vue', '3. Angular' ]
9
```

Aside – Some JS features.

- We can assign a single JSX element to a variable.

```
9
0  const demo = <div>
1      <h1>Something</h1>
2      <h2>Something else</h2>
3  </div> ;
```

- Why?

```
const demo = React.createElement(
  "div",
  null,
  React.createElement("h1", null, "Something"),
  React.createElement("p", null, "Some text ...")
);
```

Component collection - Iteration

- **Use case:** We want to generate an array of components (or JSX elements) from a data array.
- **Reference** samples/04_iteration.js

```
▼<div id="root">
  <h2>Most Popular client-side frameworks</h2> == $0
  ▼<ul>
    ▼<li>
      <a href="https://facebook.github.io/react/">React</a>
    </li>
    ▼<li>
      ▶<a href="https://vuejs.org/">...</a>
    </li>
    ▼<li>
      ▶<a href="https://angularjs.org/">...</a>
    </li>
  </ul>
</div>
```

Required HTML produced by component.
(From Chrome Dev Tools)

Component return value.

- **Examples:**
 1. return <MyComponent prop1={.....} prop2={.....} /> ;
 2. return (

```
<div>
  <h1>{this.props.type}</h1>
  <MyComponent prop1={.....} prop2={.....} />
  <p>
    . . .
  </p>
</div>
```

) ;
- **Must enclose in () when multiline.**

Component return value.

- **Must return only ONE element.**
- **Error Examples:**
 - ```
return (
 <h1>{this.props.type}</h1>
 <MyComponent prop1={.....} prop2={.....} />
 <p>

 </p>
) ;
```
  - **Error** – ‘Adjacent JSX elements must be wrapped in an enclosing tag’
  - **Solution: Wrap elements in a <div> tag.**

# Component return value.

- **Old solution:**

```
return (
 <div>
 <h1></h1>
 <MyComponent />
 <p> </p>
 </div>
) ;
```
- **Adds unnecessary depth to DOM → effects performance.**
- **Alternative solution:**

```
return (
 <>
 <h1></h1>
 <MyComponent />
 <p> </p>
</>
) ;
```
- **<> </> – special React element, termed Fragment.**
  - **No DOM presence.**

# Component *Hierarchy*.

All React application are designed as a hierarchy of components.

- Components have children – nesting.
- Ref. 05\_hierarchy.js.

The screenshot shows the Storybook interface with the sidebar navigation expanded. The 'Samples' section is selected, and the '05 - component hierarchy' item is highlighted in blue. The main content area displays two sections: 'Ranked client-side frameworks' and 'Ranked Server-side Languages'. The 'Ranked client-side frameworks' section lists React, Vue, and Angular, with a note that data is sourced from npm-stat.com. The 'Ranked Server-side Languages' section lists Javascript, Python, and Java, with a note that data is sourced from StackOverflow. Red circles numbered 1, 2, and 3 are overlaid on the right side of the content blocks, likely indicating a nesting or parent-child relationship.

**Ranked client-side frameworks**

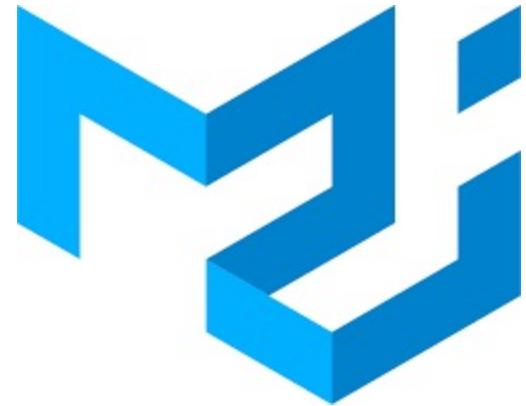
- React
- Vue
- Angular

Data sourced from [npm-stat.com](#)

**Ranked Server-side Languages**

- Javascript
- Python
- Java

Data sourced from [StackOverflow](#)



# Material UI.

A 3<sup>rd</sup> party component library to build  
high quality digital UIs

# Material Design.

- Material (Design) **is a** design system **created by Google to help teams build** high-quality digital experiences **for Android, iOS, and web.**
- A visual language **that synthesizes classic principles of good design with the innovation and possibility of technology and science.**
- **Inspired by:**
  - **the physical world and its textures, including how they reflect light and cast shadows.**
  - **the study of paper and ink.**
- **Material is a metaphor.**
  - **Material surfaces reimagine the mediums of paper and ink.**

# Material Components.

- Material Components are **interactive building blocks** for creating a **digital user interface**.
- They cover a range of interface needs, including:
  1. **Display:** Placing and organizing content using components like cards, lists, and grids.
  2. **Navigation:** Allowing users to move through an application using components like navigation drawers and tabs.
  3. **Actions:** Allowing users to perform tasks using components such as the floating action button.
  4. **Input:** Enter information or make selections using components like text fields and selection controls.
  5. **Communication:** Alerting users to key information and messages using snackbars, banners and dialogues.

# Theming.

- **Material Design does not mean copy Google design.**
- Material Theming **makes it easy to customize Material Design to match the look and feel of your brand, with built-in support and guidance for customizing colors, typography styles, and corner shape.**
- Color - **Material's color system is an organized approach to applying color to a UI. Global color styles have semantic names and defined usage in components – primary, secondary.**
- Typography - **The Material type system provides 13 typography styles for everything from headlines to body text and captions.**
  - **Each style has a clear meaning and intended application within an interface.**

# Material UI.

- A React component library based on the Material Design system.
- Components include `<Card />`, `<Box />`, `<Grid />`, `<Menu />`, `<Button />`, `<Icon />`, `<Snackbar />`, `<Typography />` .....
- Build your own design system, or start with Material Design.
- The CSS-in-JS model.

# CSS-in-JS

- **Plain CSS**

```
.my-header {
 background-color: lightblue;
 padding: 10px;
}

import 'app.css'
```

```
<header
 className="my-header">

</header>
```

Must be  
CamelCase

- **CSS-in-JS**

```
.import { makeStyles } from
 "@material-ui/core/styles";
const useStyles = makeStyles(({
 myHeader: {
 backgroundColor: "lightblue",
 padding: "10px"
 });
```

```
const classes = useStyles();
<header
 className={classes.myHeader}>
 </header>
```

1

2

3

# Summary.

- **JSX.**
  - UI description and behaviour **tightly coupled**.
  - **Can embed variables/expressions with braces.**
- **All about components.**
  - A function that takes a **props argument** and returns a single **JSX element** .
  - **Components can be nested.**
- **Storybook tool.**
  - **Develop components in isolation.**
  - **Story – the state (data values) of a component can effect its rendering (and behaviour).**
- **Material Design – The Material UI React library.**

