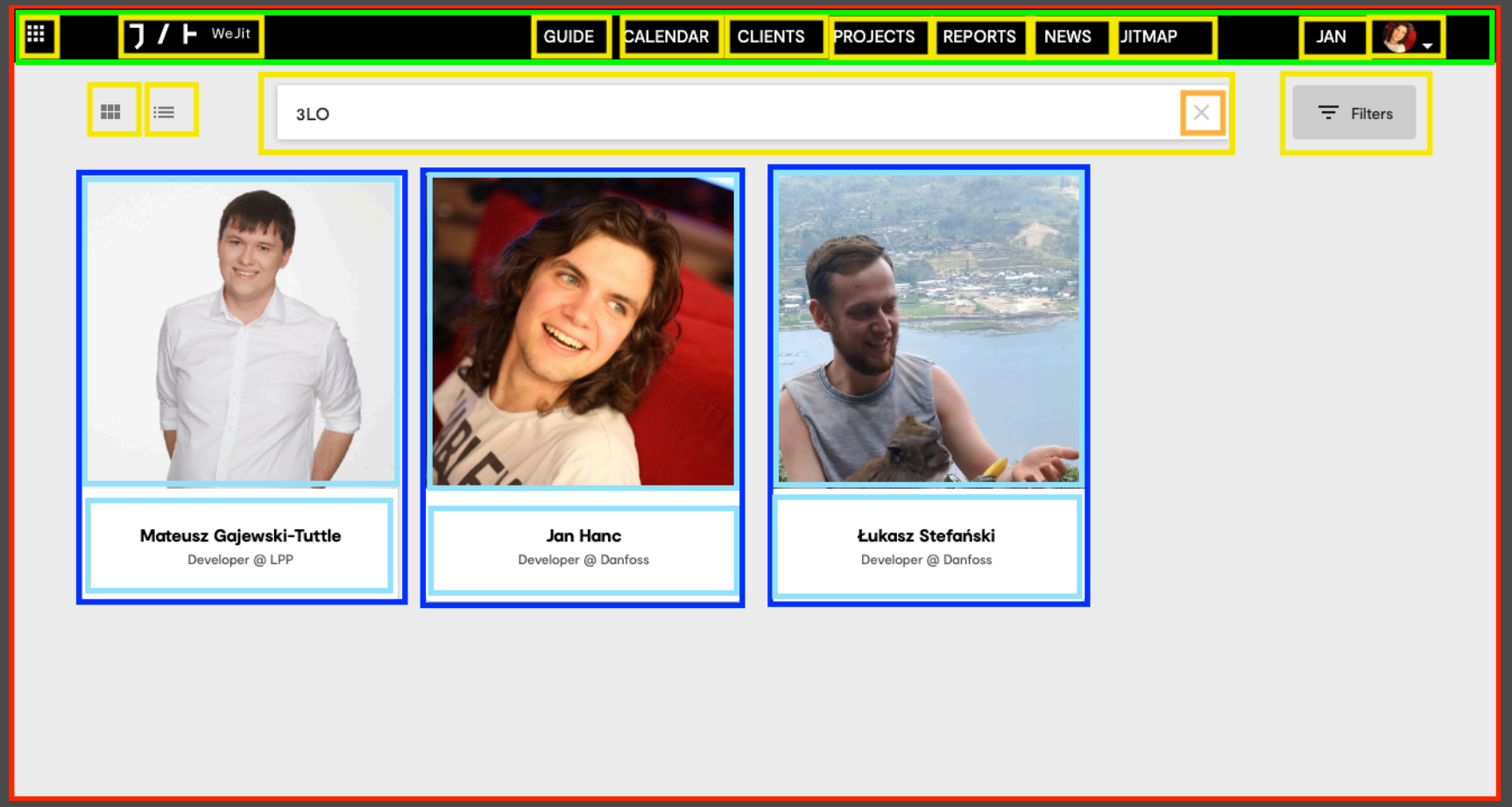
< React />

React Components

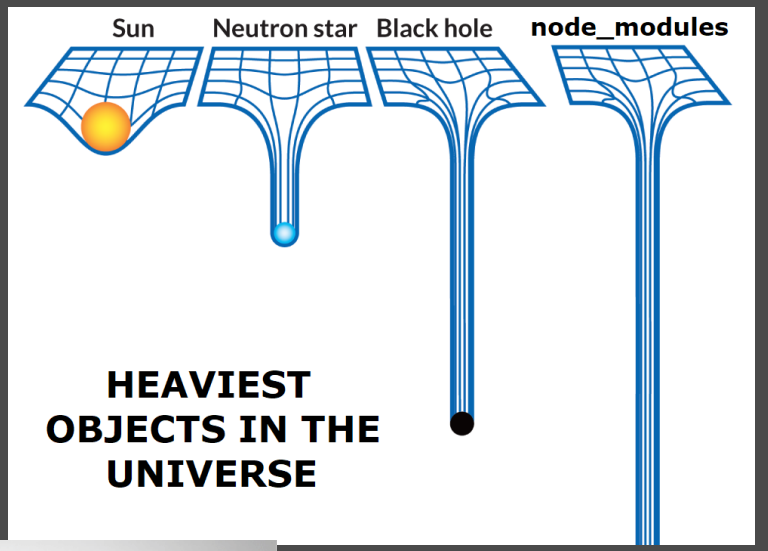
- Small isolated pieces of code
- Combined into larger components
- Create entire application

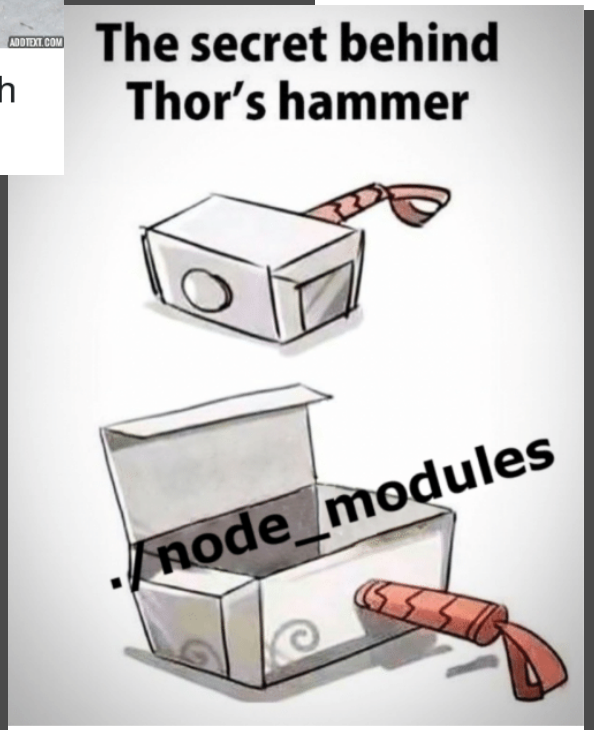


Tools

- React devtools
- Npm
- Node modules
- Webpack
- Babel
- create-react-app

A simple modern web app with npm install





Is your hard drive worthy?

JSX

```
const name = '3LO';

const element = <h1>Hello, {name}</h1>;
```

```
const element = <img src={user.avatarUrl} />
```

Virtual DOM (ReactDOM)

- Virtual representation of DOM is kept in memory and synchronized with the actual DOM
- We tell React what we want the state of UI to be – no need for attribute manipulation, event handling, and manual DOM updating
- We use React elements to represent pieces of user interface
- Once React knows which virtual DOM objects have changed, then React updates only those objects, in the real DOM
- ReactDOM is a library used by React to handle this operation

Lifecycle hooks

Mounting

- constructor()
- *static* getDerivedStateFromProps()
- render()
- componentDidMount()

Updating

- *static* getDerivedStateFromProps()
- shouldComponentUpdate()
- render()
- getSnapshotBeforeUpdate()
- componentDidUpdate()

Unmounting

- componentWillUnmount()

Error handling

- *static* getDerivedStateFromError()
- componentDidCatch()

Class components

- Render method
- Lifecycle hooks
- JSX
- Props
- State
- https://codesandbox.io/s/sweet-newton-l4rrt

```
class Counter extends React.Component {
  state = {
    count: 0
  };
  //constructor(){
  //   super()
  //   this.state = {
  //     count: 0
  //   }
  // }

  add = () => {
    this.setState(state => ({ ...state, count: this.state.count + 1 }));
  };
  substract = () => {
    this.setState(state => ({ ...state, count: this.state.count - 1 }));
  };
  componentDidMount() {
    console.log("mounted!");
  }
  componentDidUpdate() {
    console.log("updated!");
  }
  componentWillUnmount() {
    console.log("Unmounted!");
  }
  render() {
    return (
      <div className="App">
        <button onClick={this.add}>+</button>
        <button onClick={this.substract}>-</button>
        <h1>{this.state.count}</h1>
      </div>
    );
  }
}
```

Props - children

```jsx
const Container = ({ children }) => {
  return <div>{children}</div>;
};

const ContainerWithChildren = () => {
  return (
    <Container>
      <h1>Hello Janusz!</h1>
    </Container>
  );
};

ReactDOM.render(<ContainerWithChildren />, rootElement);
```

Function components

```
function Example(props) {
  return <h1>Hello {props.name}</h1>;
}
```

Higher order component (HOC)

```
const App = ({ name, color }) => {
  return <h1 style={{ color }}>Hello {name} !</h1>;
};

const HigherOrderComponent = Component => props => {
  const name = "Janusz";

  return <Component {...props} name={name} />;
};

const NewApp = HigherOrderComponent(App);
ReactDOM.render(<NewApp color="tomato" />, rootElement);
```

Render children prop

```
const ChildrenPropComponent = ({ children }) => {
  const name = "Janusz";
  return children(name);
};

const AppWithRenderProp = () => {
  return (
    <ChildrenPropComponent>
      {name => <h1>Hello {name}!</h1>}
    </ChildrenPropComponent>
  );
};

ReactDOM.render(<AppWithRenderProp />, rootElement);
```

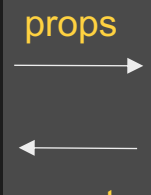
React hooks

- Complete overhaul of building react applications
- State hook
- Effect hook
- Callback hook
- Memoisation hook
- Custom hooks
- https://reactjs.org/docs/hooks-overview.html
- https://codesandbox.io/s/xenodochial-kirch-ltyg9

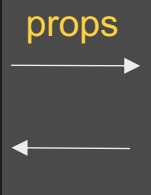
- Basic Hooks
  - `useState`
  - `useEffect`
  - `useContext`
- Additional Hooks
  - `useReducer`
  - `useCallback`
  - `useMemo`
  - `useRef`
  - `useImperativeHandle`
  - `useLayoutEffect`
  - `useDebugValue`

One way data flow

```
function Grandparent(props) {
 const handleNameClick = (arg)=>{...}
 return (
   <Parent
   name="Sara"
   onNameClick={handleNameClick}
   />
 )
}
```




```
function Parent(props) {
 return (
   <Child
   name={props.name}
   onNameClick={props.handleNameClick}
   />
 )
}
```




```
function Child(props) {
 return (
   <div className="itemContainer">
     <p>{props.name}</p>
     <input onClick={props.onNameClick} />
   </div>
 )
}
```

React higher order components

- Memo
  - PureComponent and shouldComponentUpdate for functional components
  - memoizes the rendered output then skips unnecessary rendering

```
function propsAreEqual(prevMovie, nextMovie) {
  return prevMovie.investor === nextMovie.investor
    && prevMovie.provider === nextMovie.provider;
}

export default memo(SecurityTransactionsTable, propsAreEqual);
```

- Portal
  - allows us to render children in a DOM node which doesn't necessarily exist in the parent's DOM hierarchy
  - usually used when we want a child to visually “break out” of its container. For example, dialogs, hovercards, and tooltips.

```
const PortalDialog = (props: IProps): ReactElement => {
  const { children, ...rest } = props;
  return createPortal(
    <Dialog cancelOnBgClick={false} allowEsc={false} {...rest}>
      {children || <div />}
    </Dialog>,
    document.body,
  );
};
```

Rendering lists

```
const names = ["Janusz", "Stefan", "Basia"];

const RenderList = () => {
  return (
    <ul>
      {names.map(name => (
        <li key={name}>{name}</li>
        ))}
    </ul>
  );
};
ReactDOM.render(<RenderList />, rootElement);
```

Forms

```
const Form = () => {
  const [value, setValue] = useState('');
  return (
    <form>
      <input value={value} onChange={setValue} />
    </form>
  );
};

ReactDOM.render(<Form />, rootElement);
```

Context

Initialize new context

Consume context

Provide context

```
const Context = React.createContext();

const ContextProvider = ({ children }) => {
  const [value, setValue] = useState("");
  return (
    <Context.Provider value={{ value, setValue }}>{children}</Context.Provider>
  );
};

const ContextWithHooks = () => {
  const context = useContext(Context);
  const handleChange = event => {
    context.setValue(event.target.value);
  };
  return (
    <>
      <input value={context.value} onChange={handleChange} />
      <h1>{context.value}</h1>
    </>
  );
};

ReactDOM.render(
  <ContextProvider>
    <ContextWithHooks />
  </ContextProvider>,
  rootElement
);
```

Create provider

Styling

- CSS
- CSS modules
  - local and global classes – no more name conflicts,
  - all class names and animation names are scoped locally by default – they apply to this component and nowhere else
- CSS in JS
  - camelCased properties e.g. fontWeight instead of font-weight,
  - possibility to include logic e.g. height: Platform.OS === 'ios' ? 200 : 100,
  - hooks compatible https://cssinjs.org/?v=v10.0.0#react-jss-example

Exercise #1 Pokemon Pokedex

- http://pokemontcg.io/

Exercise #1 – spacex api