MOBA2

MOBILE WEB:
REACT AND IONIC

OVERVIEW

- React Recap
- Styling React components
- Component Architecture
- React Integration
- React and Ionic

OVERVIEW

- React Recap
- Styling React components
- Component Architecture
- React Integration
- React and Ionic

JAVASCRIPT AND JSX

```
// ES5
var HelloComponent = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>;
  }
})

// ES6
class HelloComponent extends React.Component {
  render() {
    return <div>Hello {this.props.name}</div>
  }
}

// Function Component
const HelloComponent = (props) => {
  return (<div>Hello {props.name}</div>)
}
```

JAVASCRIPT AND JSX

- Component is a mix of JS and HTML code
- Idea: Everything belonging to the component is in one place
- Pure JS notation:

```
render () {
  return React.createElement("div", null, "Hello ", this.props.name)
}
```

FEEDBACK

- „A huge step backwards“
- „Rethink established best practices“

A talk by Pete Hunt:

React: Rethinking Best Practices

JSX ADOPTION

- Others have since adopted JSX
- Example: Stencil – a compiler for Web Components

```
@Component({
  tag: 'ds-text',
  styleUrl: 'ds-text.css',
  shadow: true,
})
export class Text {
  render() {
    return (
      <slot name="”ds-corp-text”">
        Your Text Rendered Here
      </slot>
    )
  }
}
```

Speaker notes

Stencil is an open source project started by the Ionic core team, with contributions also coming from the community

SO: WHAT IS REACT?

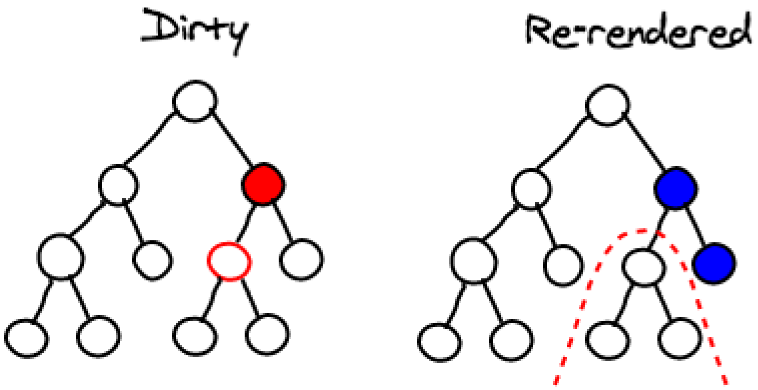
- React is the „View“ in the application
- It is not (only) a framework, it is mainly a concept
- Helps you organize your templates in components
- Separation of concerns – where to separate?
- Markup and display logic both share the same concern
- Virtual DOM makes rendering fast

Short: We have components and fast rendering

Speaker notes

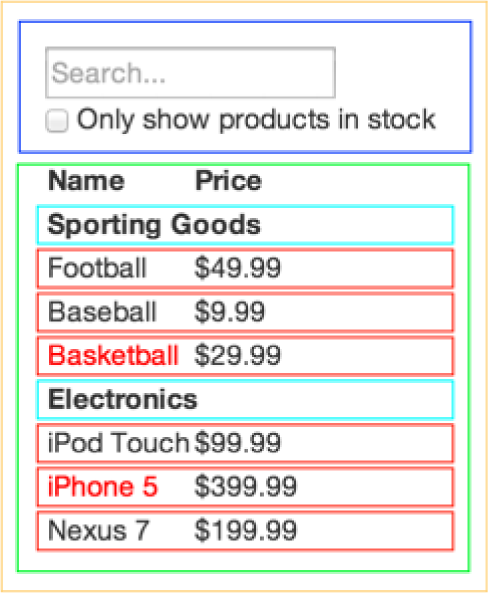
VIRTUAL DOM

- React builds a tree representation of the DOM in memory
- It calculates which DOM element should change using a diffing algorithm
- Whole application can be re-rendered when data changes
- Managing applications state much simpler



- Common alternative: Many small updates using direct DOM manipulation (jQuery ...)

COMPONENT-DRIVEN DEVELOPMENT



Search...

□ Only show products in stock

Name	Price
Sporting Goods	
Football	$49.99
Baseball	$9.99
Basketball	$29.99
Electronics	
iPod Touch	$99.99
iPhone 5	$399.99
Nexus 7	$199.99

- Power of thinking in smaller pieces
- Work with less responsibility
- Makes things easier to understand, to maintain and to test
- Design your components to be responsible for only one thing

CONCEPTUAL DIFFERENCES

```
// React
items.map(item =>
  <div key={item.id} />
)

<!-- Angular -->
<div *ngFor="let item of items">

<!-- Polymer -->
<template is='dom-repeat' items='{{items}}'></template>

<!-- Vue -->
<div v-for="item in items">
```

https://jeffcarp.github.io/frontend-hyperpolyglot/

Speaker notes

REACT SUMMARY

- Clear and simple flow of data: data is passed down and events flow up
- Data is passed down using properties
- While properties should never be changed, state is mutable
- State is owned by a component, can be passed down using properties
- Also to be passed down: Functions to change the state
- It’s best to keep most of your components stateless

Existing Frameworks Influenced:
All of them

THINK ABOUT / DISCUSS

What are the key differences between
Web Components and React

https://reactjs.org/docs/web-components.html

OVERVIEW

- React Recap
- Styling React components
- Component Architecture
- React Integration
- React and Ionic

CLASS NAMES

```
render() {
  let className = 'menu'
  if (this.props.isActive) {
    className += ' menu-active'
  }
  return <span className={className}>Menu</span>
}
```

PACKAGE CLASSNAMES

- Utility for conditionally constructing classNames
- https://www.npmjs.com/package/classnames

```
classNames('foo', 'bar')                     // => 'foo bar'
classNames({ foo: true }, { bar: true })     // => 'foo bar'
classNames({ foo: true, bar: true })         // => 'foo bar'
classNames('foo', { bar: true, duck: false }, 'baz', { quux: true })
                                             // => 'foo bar baz quux'
let buttonType = 'primary'
classNames({ [`btn-${buttonType}`]: true })  // => 'btn-primary'
```

The last example uses *computed keys* that are supported in JavaScript object literals since ES2015:

```
// Computed property names
let i = 0
const a = {
  [`foo${++i}`]: i,
  [`foo${++i}`]: i,
  [`foo${++i}`]: i,
}

console.log(a.foo1) // 1
console.log(a.foo2) // 2
console.log(a.foo3) // 3
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Object_initializer

PACKAGE CLASSNAMES

```
var classNames = require('classnames')

class Button extends Component {
  // ...
  render () {
    var btnClass = classNames({
      'btn': true,
      'btn-pressed': this.state.isPressed,
      'btn-over': !this.state.isPressed && this.state.isHovered
    })
    return <button className={btnClass}>{this.props.label}</button>
  }
}
```

INLINE STYLES

- The `style` attribute accepts an object with camelCased properties
- CSS classes are generally more efficient than inline styles, however

```
const divStyle = {
  color: 'blue',
  backgroundImage: 'url(' + imgUrl + ')',
}

function HelloWorldComponent() {
  return <div style={divStyle}>Hello World!</div>
}
```

https://reactjs.org/docs/dom-elements.html#style

INLINE STYLES

- A `px` suffix is appended automatically to certain numeric properties
- If you want to use othe units, specify the value as a string

```
// Result style: '10px'
<div style={{ height: 10 }}>
  Hello World!
</div>

// Result style: '10%'
<div style={{ height: '10%' }}>
  Hello World!
</div>
```

CSS-IN-JS

- When CSS is composed using JavaScript
- Various third-party libraries available
- Used in React Native, too

*„React does not have an opinion about how styles are defined; if in doubt, a good starting point is to define your styles in a separate *.css file as usual and refer to them using className.“*

https://reactjs.org/docs/faq-styling.html
https://css-tricks.com/a-thorough-analysis-of-css-in-js/

Speaker notes

THE DEBATE ABOUT CSS

„Do We Even Need CSS Anymore?“

- Everything is global
- CSS grows over time
- You can be more dynamic with styles in a programming language

https://speakerdeck.com/vjeux/react-css-in-js
https://css-tricks.com/the-debate-around-do-we-even-need-css-anymore/

OVERVIEW

- React Recap
- Styling React components
- Component Architecture
- React Integration
- React and Ionic

IMMUTABLE DATA

- Immutable data cannot be changed once created
- Much simpler application development
- No defensive copying necessary
- Immutable.js provides persistent immutable data structures
- Collections should be treated as values rather than objects
- Efficient on modern JavaScript VMs by using structural sharing

https://immutable-js.com

IMMUTABLE DATA

- `fromJS` deeply converts JS objects and arrays to immutable maps and lists
- Data setter and getter provide access in our component

```
state = {
  data: fromJS({ ...  }),
}

// Getter for "Immutable.js" state data...
get data() { return this.state.data }

// Setter for "Immutable.js" state data...
set data(data) { this.setState({ data }) }
```

IMMUTABLE DATA

- State is a plain JS object
- It's `data` attribute contains an immutable map
- For rendering, the data is converted back to plain JS

```
render() {
  const {
    articles,
    title,
    summary,
  } = this.data.toJS()

  return (
    <section> ... </section>
  )
}
```

FETCHING DATA

```
class UserListContainer extends Component {
  state = {
    data: fromJS({
      error: null,
      loading: 'loading...',
      users: [],
    }),
  }

  // getter, setter, componentDidMount, ...

  render() {
    return (
      <UserList {...this.data.toJS()} />
    )
  }
}
```

LIFECYCLE METHOD

```
componentDidMount() {
  users().then(      // users(true) to reject Promise
    (result) => {
      this.data = this.data
        .set('loading', null)
        .set('error', null)
        .set('users', fromJS(result.users))
    },
    (error) => {
      this.data = this.data
        .set('loading', null)
        .set('error', error)
    }
  )
}
```

USERLIST COMPONENT

```
const UserList = ({ error, loading, users }) => (
  <section>
    <ErrorMessage error={error} />
    <LoadingMessage loading={loading} />
    <ul>
      {users.map(i => (
        <li key={i.id}>{i.name}</li>
      ))}
    </ul>
  </section>
)
```

ERRORMESSAGE COMPONENT

```
const ErrorMessage = ({ error }) =>
  ImmutableMap()
    .set(null, null)
    .get(
      error,
      (<strong>{error}</strong>)
    )
```

```
// or... ??
const ErrorMessage = ({ error }) =>
  error ? (<strong>{error}</strong>) : null
```

OPTIMIZE RENDERING EFFICIENCY

```
class MyList extends Component {
  ...
  shouldComponentUpdate (nextProps, nextState) {
    return this.data !== nextState.data;
  }
}
```

- Using Immutable.js data, we can simply check for equality
- If `nextState.data` is the same, nothing has changed

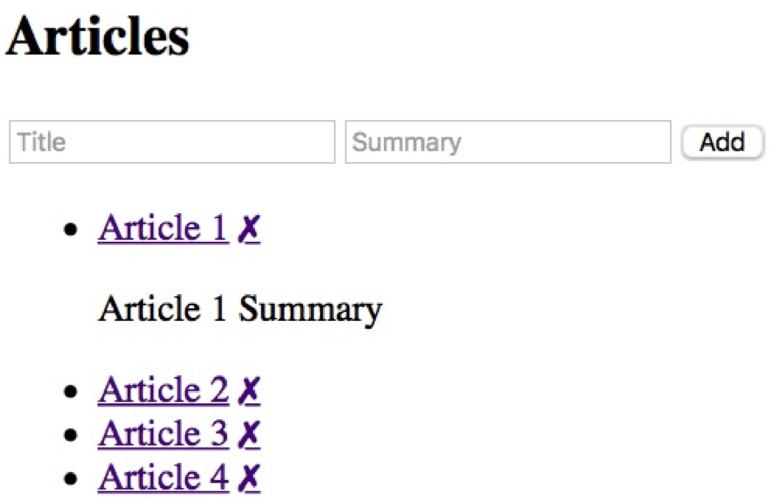
MONOLITHIC COMPONENTS

- Implement just one component for any given feature
- There wouldn’t be many components to maintain
- There wouldn’t be many communication paths

But

- Difficult to coordinate team development
- Difficult to refactor into something better
- Problem of feature overlap
- A lot of state packaged up

EXAMPLE (FROM WBE)



Articles

- Article 1 ✗

  Article 1 Summary
- Article 2 ✗
- Article 3 ✗
- Article 4 ✗

- Articles can be added
- To show/hide summary click on article
- To remove article click on ✗
- Can be implemented using just one component

EXAMPLE: STRUCTURE

```
render() {
  const { articles, title, summary, } = this.data.toJS()

  return (
    <section>
      <header>
        <h1>Articles</h1>
        <input placeholder="Title" value={title} onChange={this.onChangeTitle} />
        <input placeholder="Summary" value={summary} onChange={this.onChangeSummary} />
        <button onClick={this.onClickAdd}>Add</button>
      </header>
      <article>
        <ul> {articles.map(i => ( ... ))} </ul>
      </article>
    </section>
    )
  }
}
```

Speaker notes

EXAMPLE: LIST ITEMS

```
{articles.map(i => (
  <li key={i.id}>
    <a
      href="#"
      title="Toggle Summary"
      onClick={this.onClickToggle.bind(null, i.id)}
    > {i.title} </a>

    <a
      href="#"
      title="Remove"
      onClick={this.onClickRemove.bind(null, i.id)}
    > X </a>

    <p style={{ display: i.display }}>
      {i.summary}
    </p>
```

EXAMPLE: STATE MANAGEMENT

```
state = {
  data: fromJS({
    articles: [
      {
        id: cuid(),
        title: 'Article 1',
        summary: 'Article 1 Summary',
        display: 'none',
      },
      ...
    ],
    title: '',
    summary: '',
  }),
}
```

EXAMPLE: EVENT HANDLER

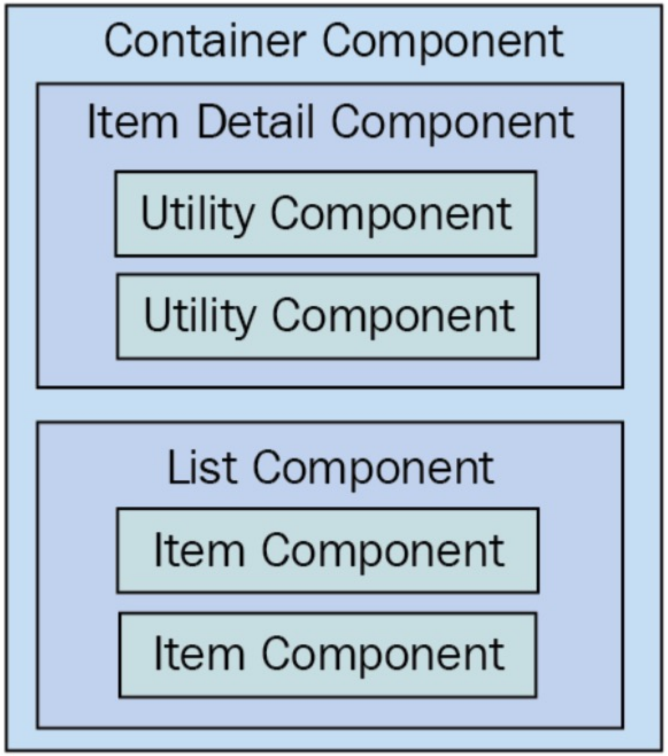
```
// When the title of a new article changes, update the state
// of the component with the new title value, by using "set()"
// to create a new map.
onChangeTitle(e) {
  this.data = this.data.set(
    'title',
    e.target.value,
  )
}
```

- The `set` method creates a new immutable map
- Assignment to `this.data` activites setter method
- Setter method calls `setState` and triggers re-rendering

EXAMPLE: EVENT HANDLER

```
// Creates a new article and empties the title
// and summary inputs. The "push()" method creates a new
// list and "update()" is used to update the list by
// creating a new map.
onClickAdd () {
  this.data = this.data
    .update(
      'articles',
      a => a.push(fromJS({
        id: cuid(),
        title: this.data.get('title'),
        summary: this.data.get('summary'),
        display: 'none',
      }))
    )
    .set('title', '')
    .set('summary', '')
```

REFACTORING COMPONENT STRUCTURE



Container Component

Item Detail Component

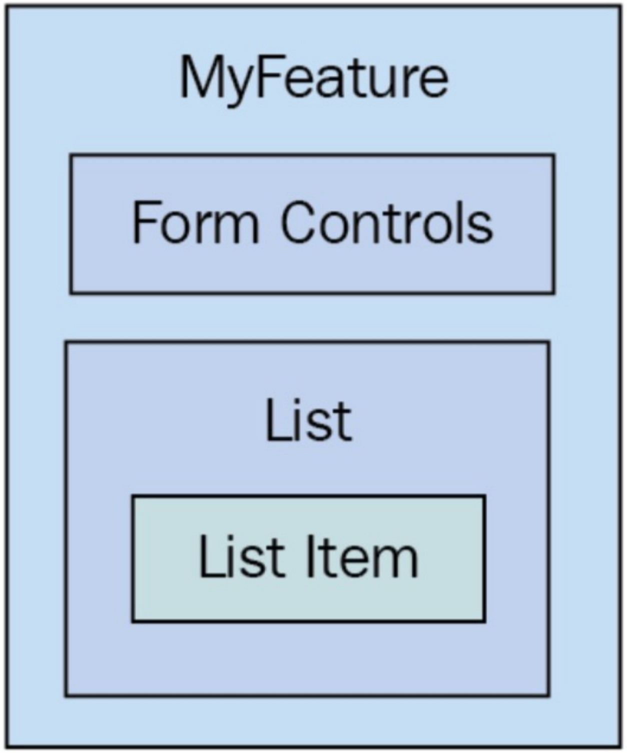
Utility Component

Utility Component

List Component

Item Component

Item Component



MyFeature

Form Controls

List

List Item

REFACTORING COMPONENT STRUCTURE

```
class MyFeature extends Component {
  render() {
    const { articles, title, summary } = this.data.toJS()
    return (
      <section>
        <AddArticle
          name="Articles" title={title} summary={summary}
          onChangeTitle={this.onChangeTitle}
          onChangeSummary={this.onChangeSummary}
          onClickAdd={this.onClickAdd} />
        <ArticleList
          articles={articles}
          onClickToggle={this.onClickToggle}
          onClickRemove={this.onClickRemove} />
      </section>
    )
  }
}
```

Speaker notes

REFACTORING COMPONENT STRUCTURE

- The `MyFeature` component maintains state
- The event handlers are also located here
- The other components are pure function components

THE ADDARTICLE COMPONENT

```
const AddArticle = ({
  name,
  title,
  summary,
  onChangeTitle,
  onChangeSummary,
  onClickAdd,
}) => (
  <section>
    <h1>{name}</h1>
    <input placeholder="Title" value={title} onChange={onChangeTitle} />
    <input placeholder="Summary" value={summary} onChange={onChangeSummary} />
    <button onClick={onClickAdd}>Add</button>
  </section>
)
```

THE ARTICLELIST COMPONENT

```
const ArticleList = ({
  articles,
  onClickToggle,
  onClickRemove,
}) => (
  <ul>
    {articles.map(i => (
      <ArticleItem
        key={i.id}
        article={i}
        onClickToggle={onClickToggle}
        onClickRemove={onClickRemove} />
    ))}
  </ul>
)
```

THE ARTICLEITEM COMPONENT

```
const ArticleItem = ({ article, onClickToggle, onClickRemove }) => (
  <li>
    <a href="#" title="Toggle Summary"
      onClick={onClickToggle.bind(null, article.id)} >
      {article.title} </a>

    <a href="#" title="Remove"
      onClick={onClickRemove.bind(null, article.id)} >
      X </a>

    <p style={{ display: article.display }}>
      {article.summary}
    </p>
  </li>
)
```

REFACTORING COMPONENT STRUCTURE

- Feature component now focuses almost entirely on the state
- It handles the initial state and handles transforming the state
- Would handle network requests, if there were any
- This is a typical container component in a React application
- The other components are the recipients of this data
- They only care about data snapshots at a particular point in time

Speaker notes

FEATURE COMPONENTS AND UTILITY COMPONENTS

- We started off with a single component that was entirely focused on a feature
- The component has very little utility elsewhere in the application
- Stateful components are difficult to use in any other context
- The further your components move from stateful data, the more utility they have
- Their property values could be passed in from anywhere in the application

FORMS AND CONTROLLED COMPONENTS

```
<input placeholder="Title" value={title} onChange={onChangeTitle} />
```

- Here, every change in the input element changes the state
- Conseqently, React state is used as the single source of truth
- Input elements like this are called controlled components

Forms in React
https://reactjs.org/docs/forms.html

Uncontrolled Components
https://reactjs.org/docs/uncontrolled-components.html

COMPONENT INHERITANCE

- Class components are just classes extending the base class `Component`
- You can create your own base components
- Components can inherit state, properties, JSX markup, event handlers

INHERITING STATE

```
class BaseComponent extends Component {

  state = {
    data: fromJS({
      name: 'Mark',
      enabled: false,
      placeholder: '',
    }),
  }

  get data() { return this.state.data }
  set data(data) { this.setState({ data }) }

  render() {
    return null
  }
}
```

```
class MyComponent extends BaseComponent {
  constructor(props) {
    super(props)
    this.state.data = this.data
      .merge({
        placeholder: 'Enter a name...',
        enabled: true,
      })
  }

  render() {
    const { enabled, name, placeholder } = this.data.toJS()
    return (
      <label htmlFor="my-input"> ... </label>
    )
  }
}
```

Don’t use this.data = ... here, since that would call the setter and also this.setState (which is not allowed in the constructor)

INHERITING PROPERTIES

Possible *static properties* in the base class:

- Default property values
- Property types

INHERITING JSX AND EVENT HANDLERS

```
class MyComponent extends BaseComponent {
  constructor(props) {
    super(props)
    this.state.data = this.data
      .merge({
        items: [
          { id: 1, name: 'One', done: false },
          { id: 2, name: 'Two', done: false },
          { id: 3, name: 'Three', done: false },
        ],
      })
  }
}
```

- MyComponent adds some state
- Rendering and event handlers defined in the base class

HIGHER-ORDER COMPONENTS

- Higher-order components are functions that take a component as input and return a new component
- Preferred over inheritance to compose new behavior
- Comparable to higher order functions in Functional Programming

CONDITIONAL COMPONENT RENDERING

```
const cond = (Component, predicate) =>
  props => predicate(props) && (<Component {...props} />)

const MyComponent = () => (
  <p>My component...</p>

const Composed = cond(MyComponent, (props) => props.show)

render((
    <Composed show={true} />
  ),
  document.getElementById('app')
)
```

PROVIDING DATA SOURCES

- Use a higher order function to connect to a data store
- The data store wraps a given component with a data source
- This pattern is used by React libraries such as Redux

Redux

- Predictable state container for JavaScript apps
- Can be used with React, or with any other view library
- It is tiny (2kB, including dependencies)
- https://redux.js.org

NAVIGATION WITH ROUTES

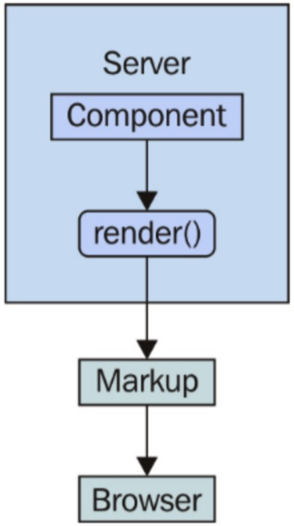
- Almost every web application requires routing
- Process of responding to a given URL
- Based on a set of route configurations
- De facto routing tool for React: react-router package

https://reactrouter.com/en/main

NAVIGATION WITH ROUTES

```
import * as React from "react"
import { createRoot } from "react-dom/client"
import { BrowserRouter } from "react-router-dom"

const root = createRoot(document.getElementById("root"))

root.render(
  <BrowserRouter>
    {/* The rest of your app goes here */}
  </BrowserRouter>
)
```

SERVER-SIDE REACT COMPONENTS



- React isn’t confined to the browser for rendering
- Components can be rendered on a Node.js server
- Isomorphic JavaScript: JavaScript code that can run in the browser and in Node.js without modification

SERVER-SIDE REACT COMPONENTS

- Better initial load performance
- Sharing code between the backend and frontend

https://reactjs.org/docs/react-dom-server.html

REACT AND EXPRESS

```
import React from 'react'
import { renderToString } from 'react-dom/server'
import express from 'express'
import App from './App'

const doc = content =>
  `
  <!doctype html>
  <html>
    <head>
      <title>Rendering to strings</title>
    </head>
    <body>
      <div id="app">${content}</div>
    </body>
  </html>
  `
```

```
const app = express()

// The root URL of the APP, returns the rendered React component.
app.get('/', (req, res) => {
  const props = {
    items: ['One', 'Two', 'Three'],
  }
  const rendered = renderToString((
    &lt;App {...props} />
  ))
  res.send(doc(rendered))
})

app.listen(8080, () => {
  console.log('Listening on 127.0.0.1:8080')
})
```

The 'renderToString()' function is like 'render()', except it returns a rendered HTML string instead of manipulating the DOM.

OVERVIEW

- React Recap
- Styling React components
- Component Architecture
- React Integration
- React and Ionic

REACT INTEGRATION EXAMPLES

- jQuery: encapsulate imperative code
- Bootstrap: pre-defined components
- Ionic: mobile optimized React UI components

RENDERING IMPERATIVE COMPONENTS

- Example: a simple jQuery UI button React component
- Lifecycle methods help us encapsulate imperative code
- The `button()` method has to be called on the DOM element

When the *ref* attribute is used on an HTML element, the ref callback receives the underlying DOM element as its argument. This allows you to have direct access to the DOM element or component instance.

In a functional component, a *useRef* hook would be used.

RENDERING IMPERATIVE COMPONENTS

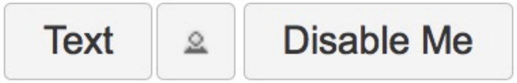
```
import $ from 'jquery'
import 'jquery-ui/ui/widgets/button'
import 'jquery-ui/themes/base/all.css'

class MyButton extends Component {
  componentDidMount() {
    $(this.button).button(this.props)
  }
  componentDidUpdate() {
    $(this.button).button('option', this.props)
  }
  render() {
    return (
      <button
        onClick={this.props.onClick}
        ref={(button) => { this.button = button }} />
    )
  }
}
```

Speaker notes

jQuery UI is no longer widely used. The main purpose of the example is to show how an imperative API can be wrapped in a declarative API.

RENDERING IMPERATIVE COMPONENTS

```
render((
  <section>
    <MyButton label="Text" />
    <MyButton label="My Button" icon="ui-icon-person" showLabel={false} />
    <MyButton label="Disable Me" onClick={onClick} />
  </section>
  ),
  document.getElementById('app')
)
```

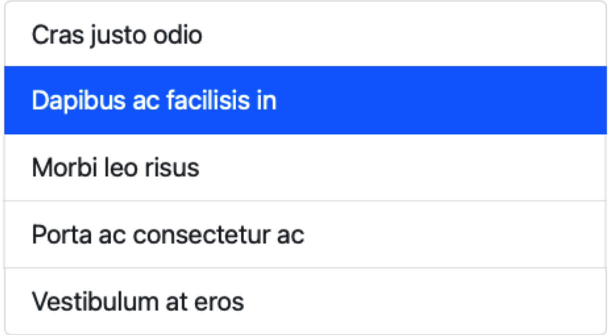


BOOTSTRAP

- Popular frontend toolkit
  - https://getbootstrap.com
- React/Bootstrap component libraries
  - https://react-bootstrap.github.io
  - https://reactstrap.github.io/

EXAMPLE: REACTSTRAP

```
<ListGroup>
  <ListGroupItem>
    Cras justo odio
  </ListGroupItem>
  <ListGroupItem active>
    Dapibus ac facilisis in
  </ListGroupItem>
  <ListGroupItem>
    Morbi leo risus
  </ListGroupItem>
  <ListGroupItem>
    Porta ac consectetur ac
  </ListGroupItem>
  <ListGroupItem>
    Vestibulum at eros
  </ListGroupItem>
</ListGroup>
```



Cras justo odio

Dapibus ac facilisis in

Morbi leo risus

Porta ac consectetur ac

Vestibulum at eros

OVERVIEW

- React Recap
- Styling React components
- Component Architecture
- React Integration
- React and Ionic

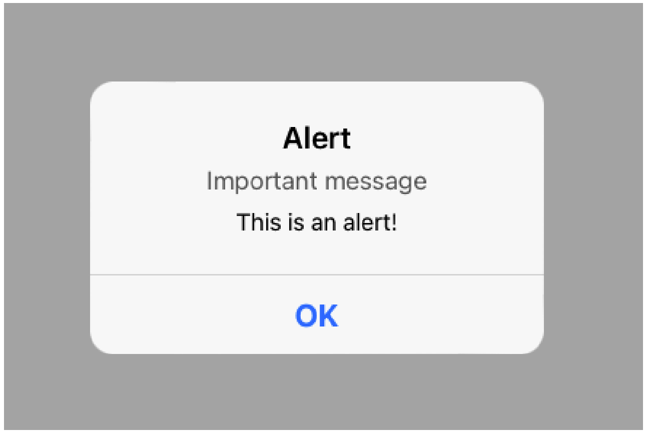
IONIC

- UI toolkit for building mobile apps
- Based on web technologies — HTML, CSS, and JavaScript
- Integrations for Angular, React, and Vue
- Can also be used standalone
- Focus: UI controls, interactions, gestures, animations

https://ionicframework.com

PLATFORM STYLES

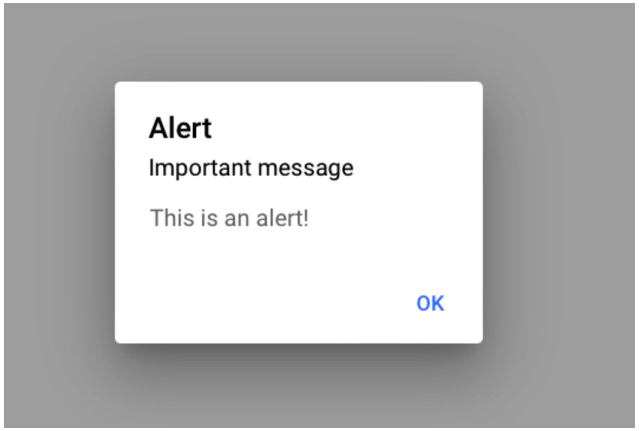
Platform specific styles: iOS and Android



Alert

Important message

This is an alert!



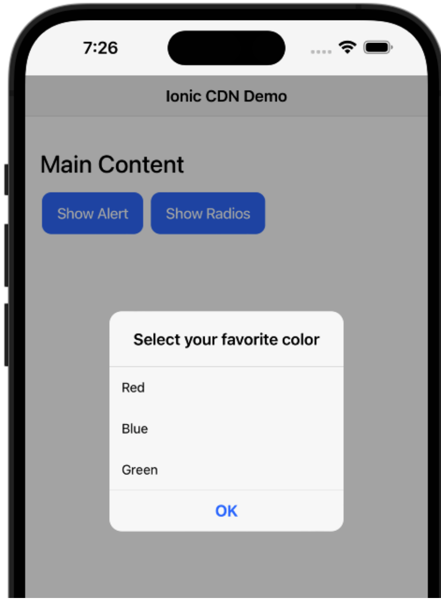
Alert

Important message

This is an alert!

FIRST TRY: CDN

```
<body>
  <ion-app>

    <ion-header>
      <ion-toolbar>
        <ion-title>Ionic CDN Demo</ion-title>
      </ion-toolbar>
    </ion-header>

    <ion-content class="ion-padding">
      <h1>Main Content</h1>
      <ion-button onclick="presentAlert()">Show Alert</ion-button>
      <ion-button onclick="presentRadios()">Show Radios</ion-button>
    </ion-content>

  </ion-app>
...
</body>
```



INSTALLATION

```
# install Ionic cli
npm install -g @ionic/cli

# create an app
ionic start myApp blank --type=react

# start app
cd myApp
ionic serve
```

TEMPLATES

```
$ ionic start --list

Starters for @ionic/react (--type=react)

name         | description
-------------------------------------------------------------------
blank        | A blank starter project
list         | A starting project with a list
my-first-app | A template for the "Build Your First App" tutorial
sidemenu     | A starting project with a side menu with navigation
tabs         | A starting project with a simple tabbed interface

...
```

EXAMPLE: index.tsx

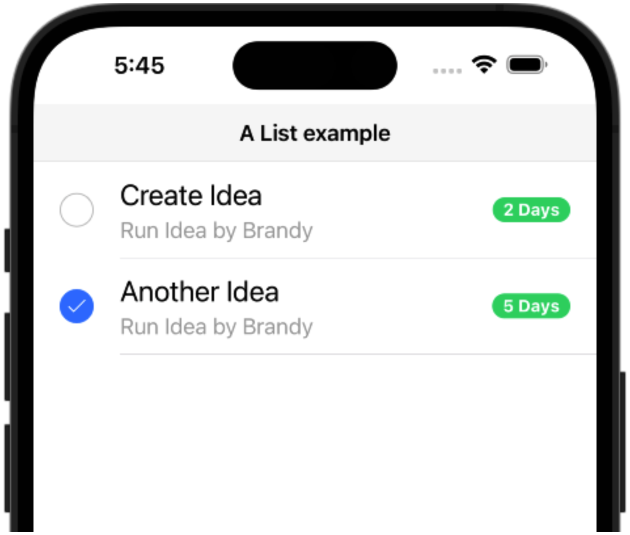
```
import React from 'react'
import ReactDOM from 'react-dom'
import App from './App'

ReactDOM.render(<App />, document.getElementById('root'))
```

EXAMPLE: App.tsx

```
import React from 'react'
import { Route } from 'react-router-dom'
import { IonApp, IonRouterOutlet } from '@ionic/react'
import { IonReactRouter } from '@ionic/react-router'
import Home from './pages/Home'
import NewItem from './pages/NewItem'

/* Core CSS required for Ionic components to work properly */
import '@ionic/react/css/core.css'

const App: React.FC = () => (
  <IonApp>
    <IonReactRouter>
      <IonRouterOutlet>
          <Route path="/home" component={Home} />
          <Route path="/new" component={NewItem} />
          <Redirect exact from="/" to="/home" />
      </IonRouterOutlet>
    </IonReactRouter>
  </IonApp>
)
```

EXAMPLE: Home.tsx (1)

```
<IonContent>
  <IonList>
    <IonItem>
      <IonCheckbox slot="start" />
      <IonLabel>
        <h1>Create Idea</h1>
        <IonNote>Run Idea by Brandy</IonNote>
      </IonLabel>
      <IonBadge color="success" slot="end">
        5 Days
      </IonBadge>
    </IonItem>
  </IonList>
</IonContent>
```



5:45

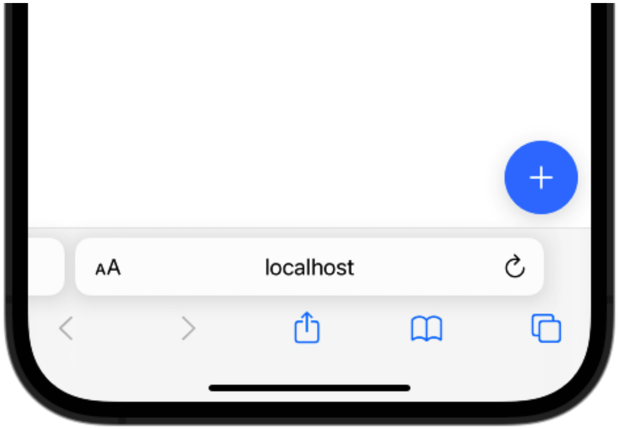
A List example

○ Create Idea
Run Idea by Brandy
2 Days

● Another Idea
Run Idea by Brandy
5 Days

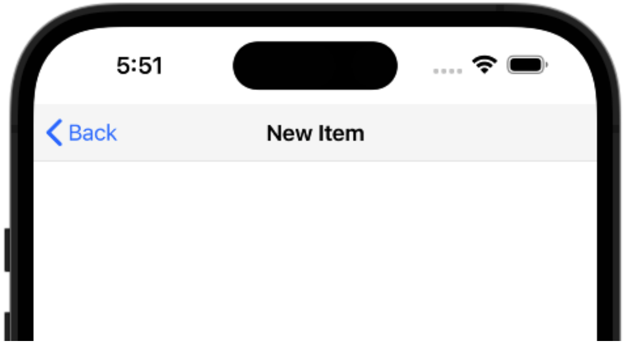
EXAMPLE: Home.tsx (2)

```
<IonContent>
  <IonList>
    ...
  </IonList>

  <IonFab vertical="bottom" horizontal="end" slot="fixed">
    <IonFabButton onClick={() => props.history.push('/new')} >
      <IonIcon icon={add} />
    </IonFabButton>
  </IonFab>

</IonContent>
```



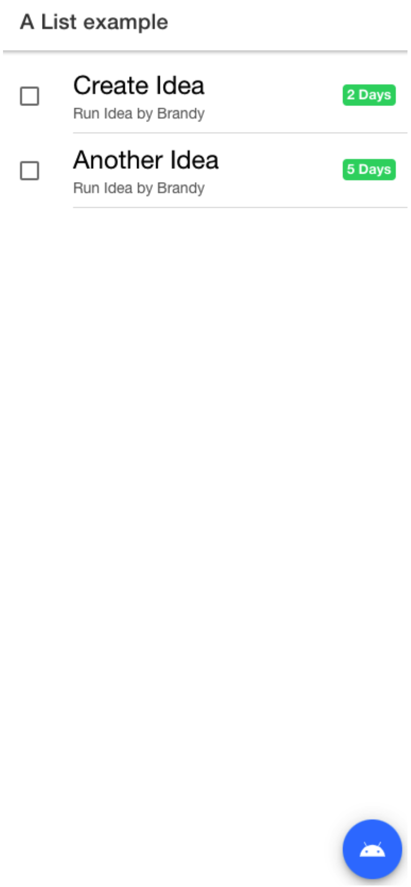
EXAMPLE: NewItem.tsx

```
const NewItem: React.FC = () => {
  return (
    <IonPage>
      <IonHeader>
        <IonToolbar>
          <IonButtons slot="start">
            <IonBackButton defaultHref="/home" />
          </IonButtons>
          <IonTitle>New Item</IonTitle>
        </IonToolbar>
      </IonHeader>
      <IonContent></IonContent>
    </IonPage>
  )
}
```



PLATFORM SPECIFIC ICON

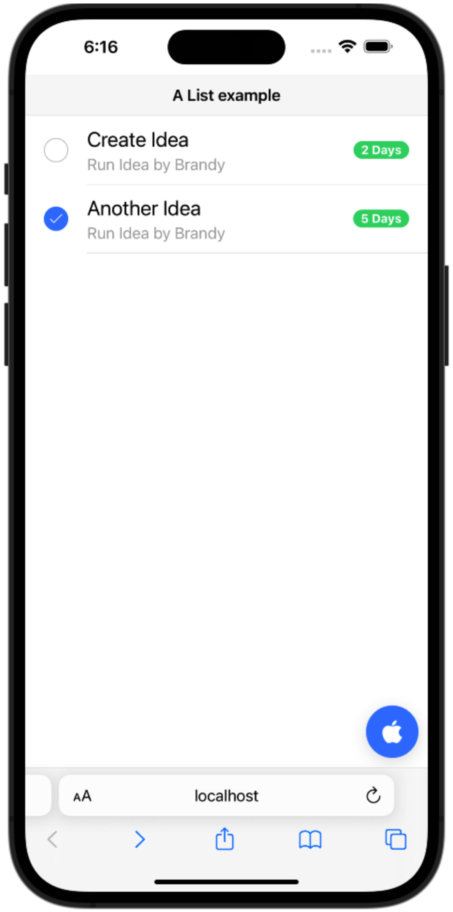
```
<IonIcon md={logoAndroid} ios={logoApple} />
```



A List example

□ Create Idea 2 Days
Run Idea by Brandy

□ Another Idea 5 Days
Run Idea by Brandy



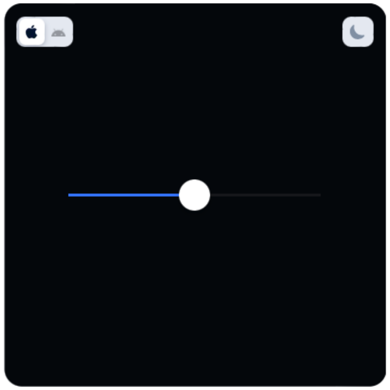
6:16

A List example

○ Create Idea 2 Days
Run Idea by Brandy

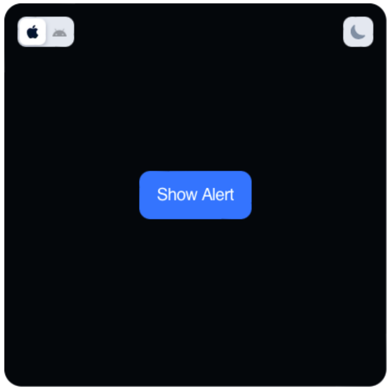
● Another Idea 5 Days
Run Idea by Brandy

MANY PRE-BUILT COMPONENTS



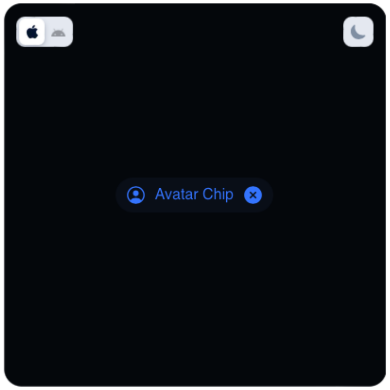
Slider

Supports keyboard and touch input, step interval, multiple thumb, and RTL direction



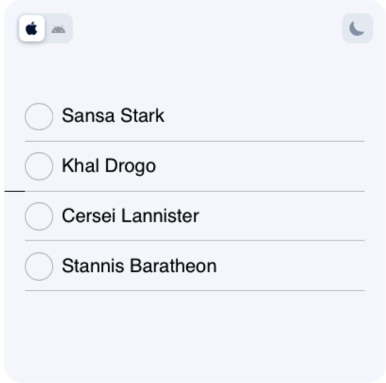
Alert

Stock with two platform modes, fine-grained focus control, accessible to screen readers



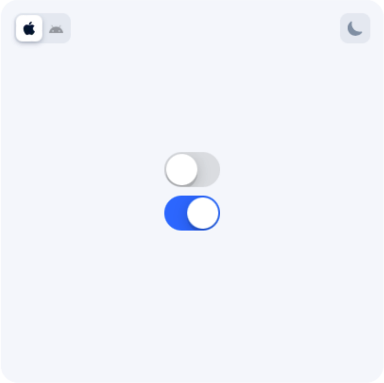
Chip

Can contain several different elements such as avatars, text, and icons



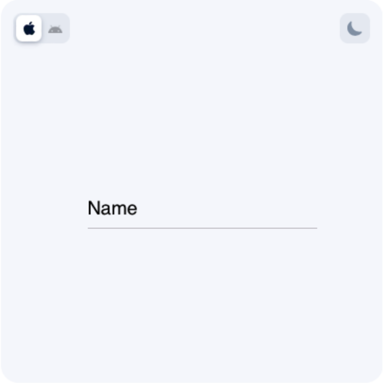
Checkbox

Allow for the display and selection of multiple options from a set of options



Toggle

Can be switched on or off by pressing or swiping and can also be checked programmatically



Input

A wrapper to the HTML input element with custom styling and additional functionality

NATIVE APP

- Access native device functionality
- Camera, Maps, Geolocation, Bluetooth, ...
- Cordova (older) or Capacitor (newer)

https://capacitorjs.com
https://cordova.apache.org

READING MATERIAL AND SOURCES

SOURCES

- React – A JavaScript library for building user interfaces
  https://reactjs.org
- Adam Boduch: React and React Native
  Second Edition, Packt Publishing, 2018
  Packt Online Shop