REACT FUNDAMENTALS

Len Nguyen - Solution Architect

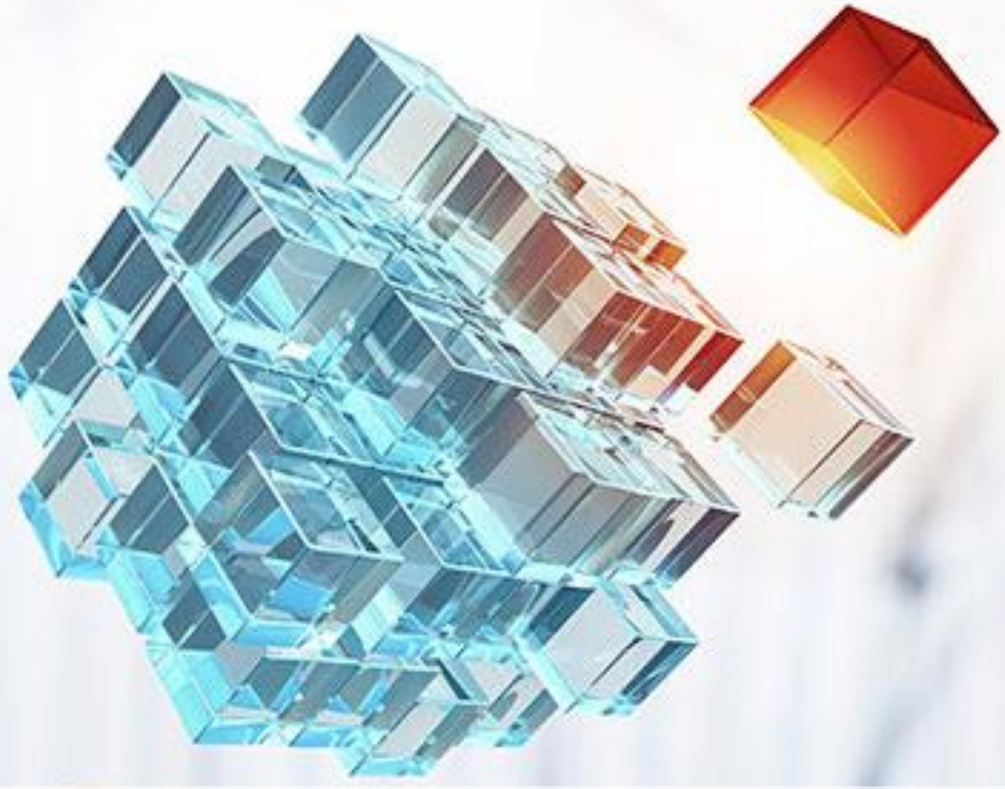
01-11-2016

NASH TECH
The Power to Innovate

AGENDA

- Objectives
- What is React?
- Why React?
- Hello React
- JSX
- Components
  - Props
  - State
  - Lifecycle
- Virtual DOM
- React API
- ReactDOM API
- Events
- Forms
- Router
- Animations
- Flux
- Redux
- Testing
- Starter Kit
- References
- Assignment
- Q&A

NASH TECH
The Power to Innovate



OBJECTIVES

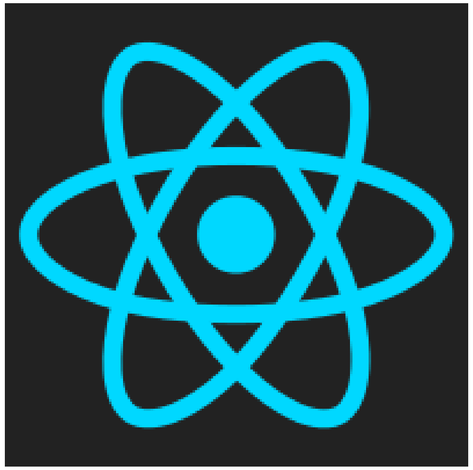
Objectives

- Have an overview of React ecosystem.
- Understand the core concepts of React and how it works.
- Understand React components, props, state and lifecycle.
- Understand of JSX.
- Understand the concepts of unidirectional data flow.
- Understand Flux pattern.
- Understand Redux.
- Learn React best practices.

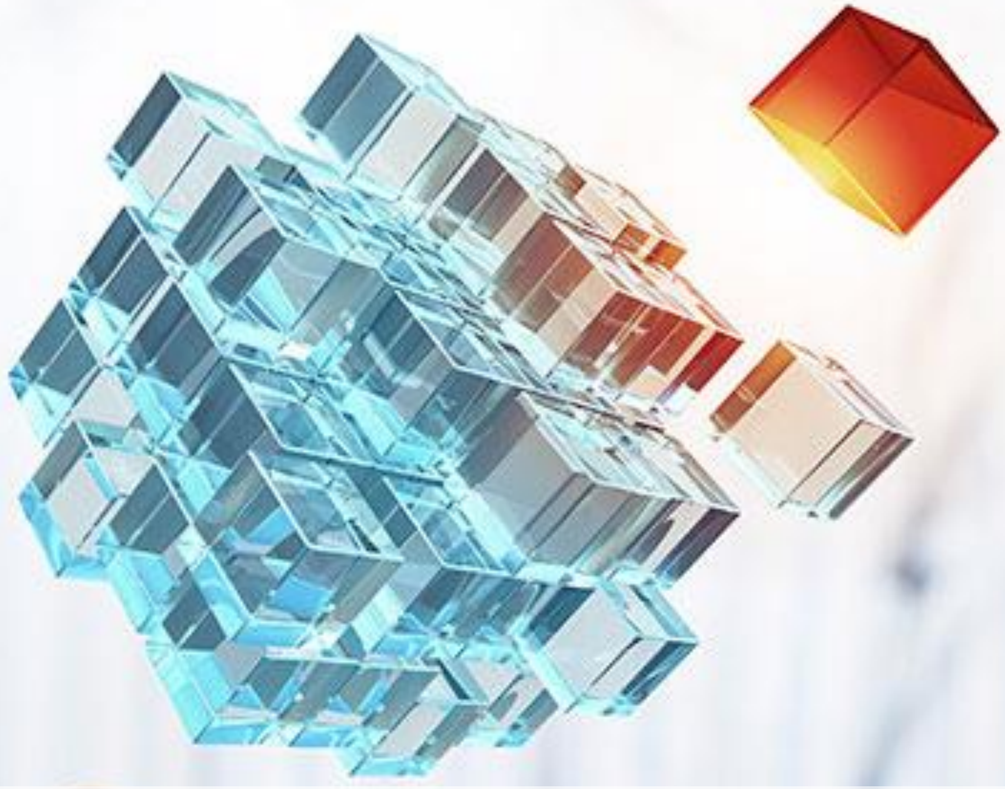
WHAT IS REACT?

What is React?

- React is a declarative, efficient, and flexible JavaScript library for building user interfaces.
  - Declarative: React uses a declarative paradigm that makes it easier to reason about your application.
  - Efficient: React computes the minimal set of changes necessary to keep your DOM up-to-date.
  - Flexible: React works with the libraries and frameworks that you already know.
- It is a V in MVC.



NASH TECH
The Power to Innovate



WHY REACT?

Why React?

- Solve one problem: building large applications with data that changes over time.
- Simple
  - Simply express how your app should look at any given point in time, and React will automatically manage all UI updates when your underlying data changes.
- Declarative
  - When the data changes, React conceptually hits the "refresh" button, and knows to only update the changed parts.
- Composable Components
  - React is all about building reusable components. In fact, with React the only thing you do is build components. Since they're so encapsulated, components make code reuse, testing, and separation of concerns easy.
  - Components are the future of web development.

Why React?

- Learn once, write anywhere.
  - Use React for both client side and server side
  - Use React Native for native mobile application

Hello React

- React ES5
  - var Hello = React.createClass({
  -   render: function() {
  -     return React.createElement('div', null, 'Hello ', this.props.name);
  -   }
  - });

  - ReactDOM.render(
  -   React.createElement(Hello, {name: 'React'}),
  -   document.getElementById('app')
  -);

Hello React

- React ES5 using JSX
  - var Hello = React.createClass({
  - render: function() {
  - return Hello {this.props.name} ;
  - }
  - });

  - ReactDOM.render(
  - <Hello name="React" />,
  - document.getElementById('app')
  -);

Hello React

- React ES6 using JSX
  - function Hello(props) {
  - return Hello {props.name} ;
  - }

  - ReactDOM.render(
  - <Hello name="React" />,
  - document.getElementById('app')
  -);

Hello React

- React ES6 using JSX
  - class Hello extends React.Component {
  - render() {
  - return Hello {this.props.name} ;
  - }
  - }

  - ReactDOM.render(
  - <Hello name="React" />,
  - document.getElementById('app')
  -);

NASH TECH
The Power to Innovate

JSX

JSX

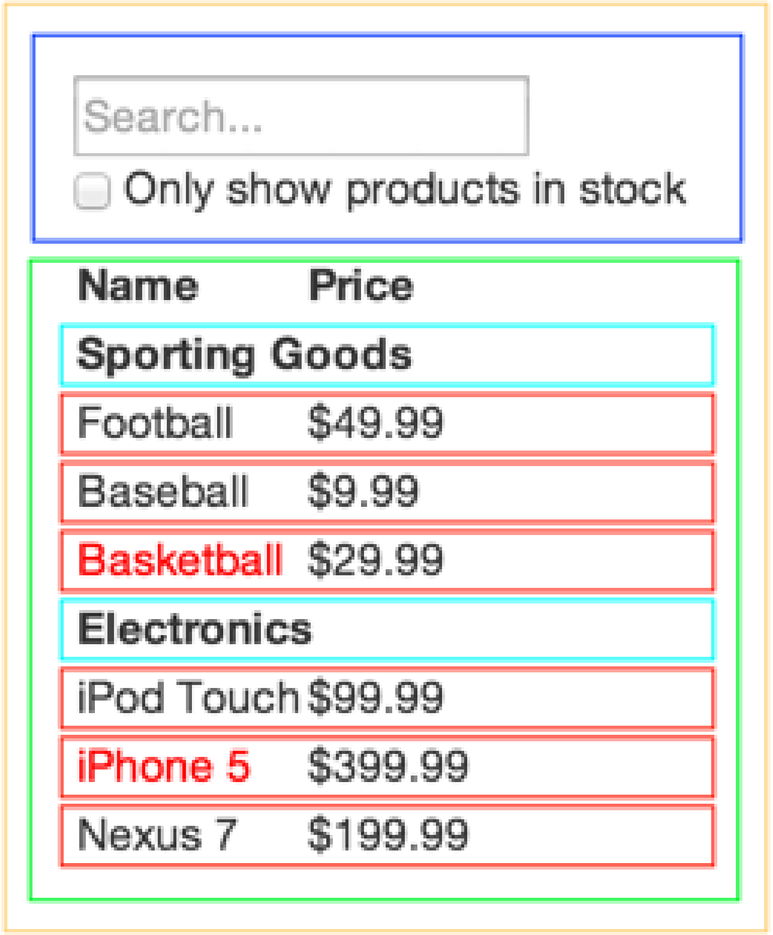
- JSX is a syntax extension to JavaScript that looks similar to XML.
  - JSX: const element = <h1>Hello, world!</h1>;
- JSX just provides syntactic sugar for the React.createElement(component, props, ...children) function.
  - JSX: <Button color="blue">Click Me</Button>
  - React Element: React.createElement(Button, {color: 'blue'}, 'Click Me’)
- JSX tags have a tag name, attributes, and children. If an attribute value is enclosed in quotes, the value is a string. Otherwise, wrap the value in braces and the value is the enclosed JavaScript expression.
  - JSX: <MyCounter count={3 + 5} />;
- JSX is optional and not required to use React.
- It helps in marking writing code easier and faster.

NASH TECH
The Power to Innovate

COMPONENTS

Components

- In React, the whole application is break into the components.
- Components let you split the UI into indep pieces, and think about each piece in isol
- Example:
  - FilterableProductTable
    - SearchBar
    - ProductTable
      - ◇ProductCategoryRow
      - ◇ProductRow



Components

- Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.
- Functional Components
  - function Welcome(props) {
  - return <h1>Hello, {props.name}</h1>;
  - }
- Class Components
  - class Welcome extends React.Component {
  - render() {
  - return <h1>Hello, {this.props.name}</h1>;
  - }
  - }

Components

- Class Properties
  - defaultProps can be defined as a property on the component class itself, to set the default props for the class.
  - Example:
    - class CustomButton extends React.Component {
    - // ...
    - }

    - CustomButton.defaultProps = {
    - color: 'blue'
    - };

Components

- Class Properties
  - propTypes can be defined as a property on the component class itself, to define what types the props should be.
  - Example:
    - class CustomButton extends React.Component {
    - // ...
    - }

    - CustomButton.propTypes = {
    - color: React.PropTypes.string
    - };

NASH TECH
The Power to Innovate

PROPS

Props

- Props are a way of passing data from parent to child.
- Use props (this.props) to access parameters passed from the parent.
- Props are read-only (immutable).
- Example:
  - function Welcome(props) {
  - return <h1>Hello {props.name}</h1>;
  - }
  - ReactDOM.render(
  - <Welcome name="World" />,
  - document.getElementById('app')
  -);

Props

- Define defaults and initial values for this.props by implementing getDefaultProps().
- Example:

```
var Counter = React.createClass({
  getDefaultProps: function() {
    return {
      initialValue: 1
    }
  },
  ...
});
```

NASH TECH
The Power to Innovate

STATE

State

- State is similar to props, but it is private and fully controlled by the component.
- Use states (this.state) to manage dynamic data.
- State is reserved only for interactivity, that is, data that changes over time.
- State should always start with a default value and then the state is mutated internally by the component using setState()
- State should be avoided if at all possible.
- Define defaults and initial values for this.state by implementing getInitialState().

State

```
var Counter = React.createClass({
  getInitialState: function() {
    return {
      count: 0
    };
  },
  handleClick: function() {
    this.setState({
      count: this.state.count + 1
    });
  },
  render: function() {
    return <div onClick={this.handleClick}>Clicked {this.state.count}</div>;
  }
});

ReactDOM.render(
  <Counter />,
  document.getElementById('app')
);
```

State

```
class Counter extends React.Component {
        constructor(props) {
          super(props);
          this.state = { count: props.count }
          this.handleClick = this.handleClick.bind(this);
        }
        handleClick() {
          this.setState({count: this.state.count + 1});
        }
        render() {
          return <button onClick={this.handleClick}>Click me! Number of clicks: {this.state.count}</button>;
        }
      }
      Counter.propTypes = {
        count: React.PropTypes.number
      };
      Counter.defaultProps = {
        count: 0
      };
      ReactDOM.render(
        <Counter count={10} />,
        document.getElementById('app')
      );
```

NASH TECH
The Power to Innovate

LIFECYCLE

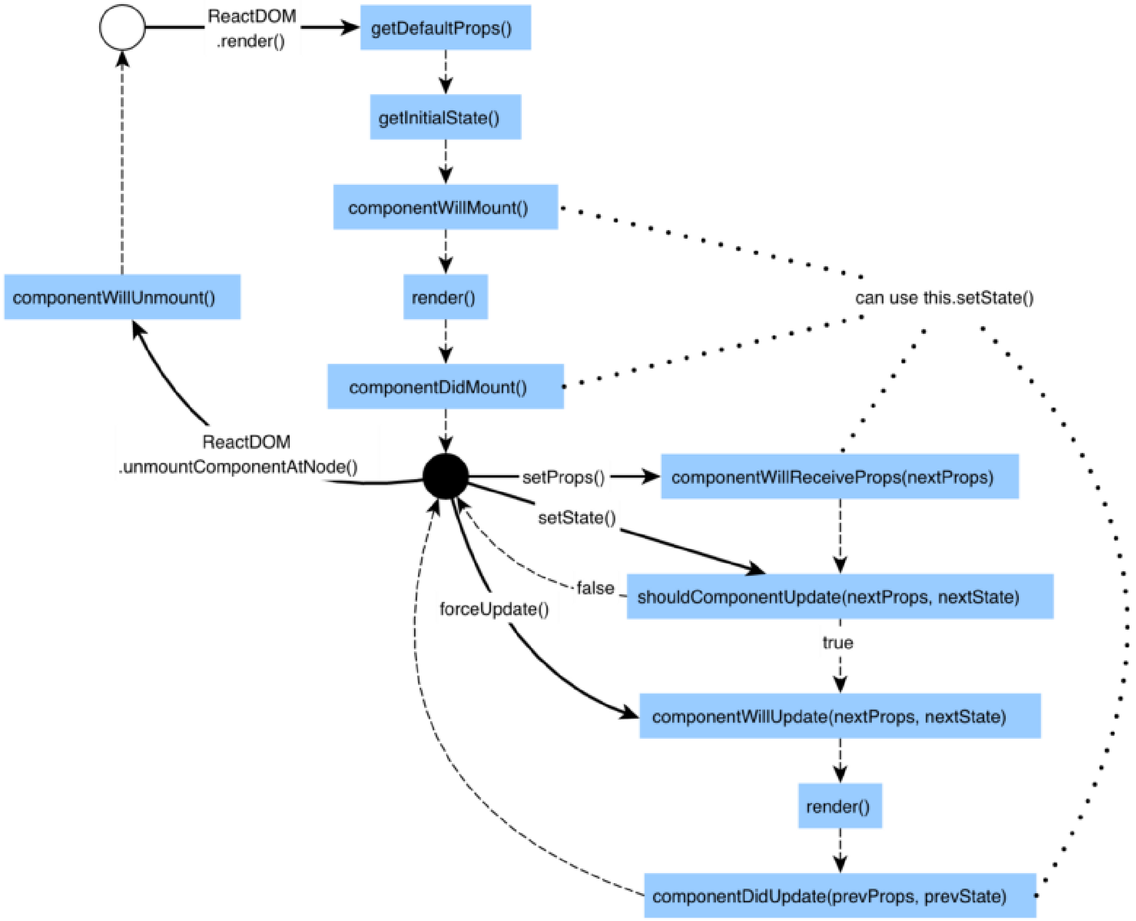
Lifecycle

- Each component has several "lifecycle methods" that you can override to run code at particular times in the process.
  - Methods prefixed with will are called right before something happens.
  - Methods prefixed with did are called right after something happens.
- Mounting
  - These methods are called when an instance of a component is being created and inserted into the DOM
    - constructor()
    - componentWillMount()
    - render()
    - componentDidMount()

Lifecycle

- Updating
  - An update can be caused by changes to props or state. These methods are called when a component is being re-rendered:
    - componentWillReceiveProps()
    - shouldComponentUpdate()
    - componentWillUpdate()
    - render()
    - componentDidUpdate()
- Unmounting
  - This method is called when a component is being removed from the DOM
    - componentWillUnmount()

Lifecycle


ReactDOM
.render()
getDefaultProps()
getInitialState()
componentWillMount()
render()
componentDidMount()
componentWillUnmount()
ReactDOM
.unmountComponentAtNode()
can use this.setState()
setProps()
componentWillReceiveProps(nextProps)
setState()
false
shouldComponentUpdate(nextProps, nextState)
forceUpdate()
true
componentWillUpdate(nextProps, nextState)
render()
componentDidUpdate(prevProps, prevState)

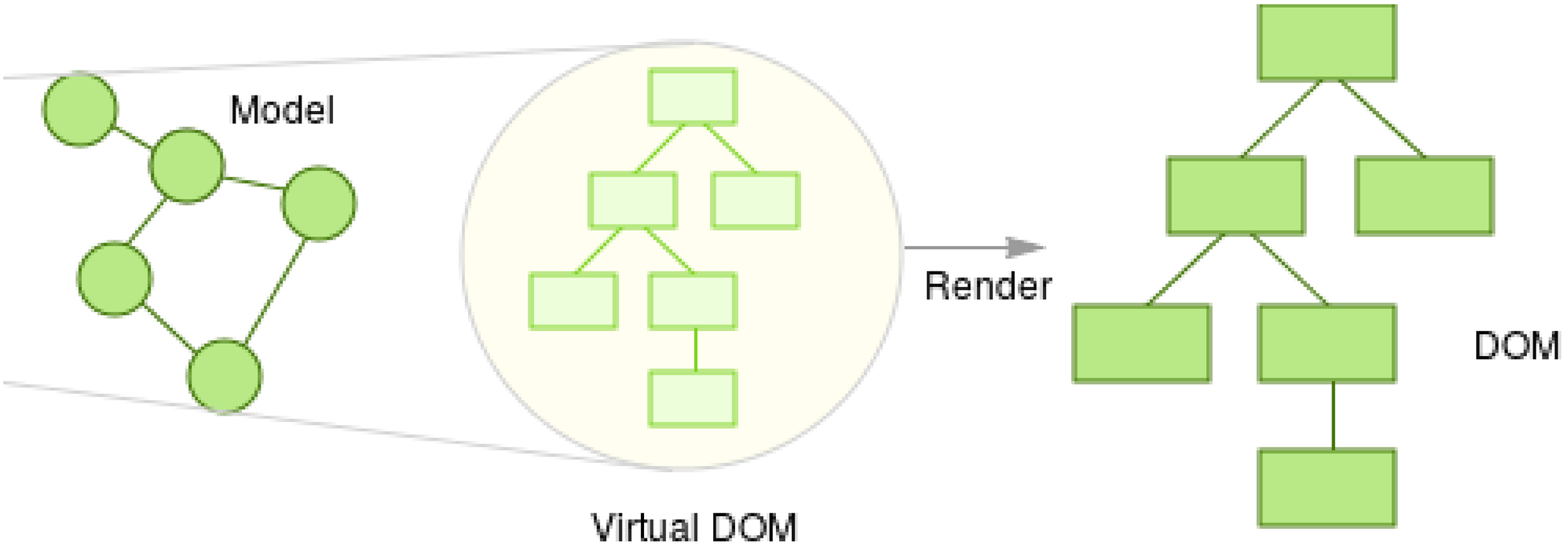
NASH TECH
The Power to Innovate

VIRTUAL DOM

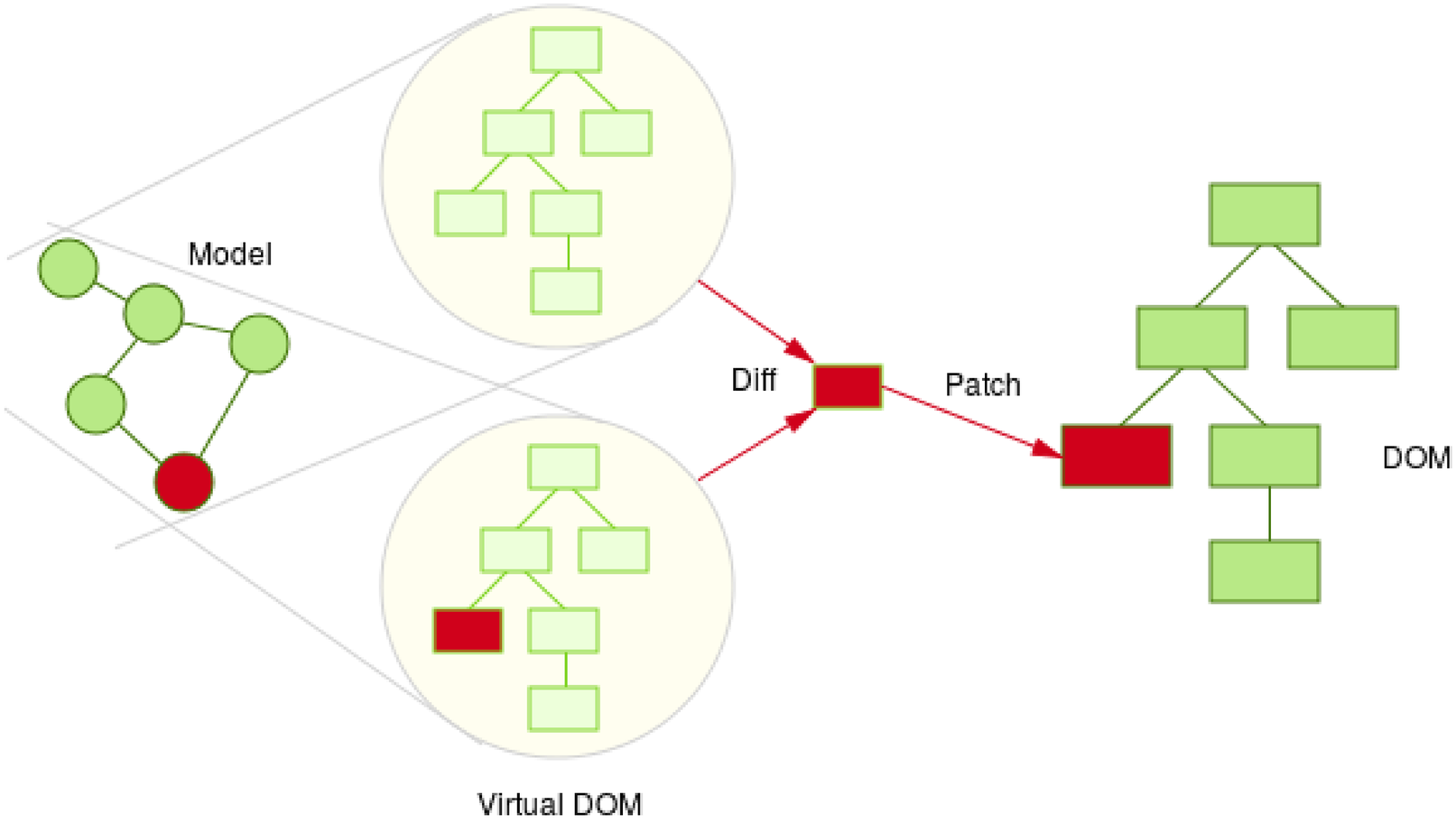
Virtual DOM

- React is very fast because of a technique called virtual DOM. It maintains a fast, in-memory representation of the DOM and never directly talks to the real DOM.
- Whenever anything may have changed, re-render everything to a virtual DOM representation.
- Diff the previous representation with the next one.
- Only update the real DOM with what actually changed.
- On every update
  - React builds a new Virtual DOM subtree
  - Diffs it with the old one
  - Computes the minimal set of DOM mutations and puts them in the queue
  - Batch executes all updates

Virtual DOM



Virtual DOM



NASH TECH
The Power to Innovate

REACT API

React API

- Components
  - React.createClass(specification)
  - React.Component
  - React.PureComponent
- Creating React Elements
  - React.createElement(type, [props], […children])
- Transforming Elements
  - cloneElement(type, [props], […children])
  - isValidElement(object)
  - React.Children (map, forEach, count, only, toArray)
- Typechecking with PropTypes
  - React.PropTypes
  - React.PropTypes.array
  - React.PropTypes.bool
  - React.PropTypes.func
  - React.PropTypes.number
  - React.PropTypes.object
  - React.PropTypes.string
  - React.PropTypes.any
  - …

NASH TECH
The Power to Innovate

REACTDOM API

ReactDOM API

- ReactDOM.render(element, container, [callback])
- ReactDOM.unmountComponentAtNode(container)
- ReactDOM.findDOMNode(component)
- ReactDOMServer.renderToString(element)
- ReactDOMServer.renderToStaticMarkup(element)

NASH TECH
The Power to Innovate

EVENTS

Events

- Handling events with React elements is very similar to handling events on DOM elements. There are some syntactic differences:
  - React events are named using camelCase, rather than lowercase.
  - With JSX you pass a function as the event handler, rather than a string.
- React defines these synthetic events according to the W3C spec.
- React uses SyntheticEvent, which is a cross-browser wrapper around the browser's native event. So all the event handlers in the react applications will be passed instances of SyntheticEvent.
- React normalizes events so that they have consistent properties across different browsers.
- Supported events: Clipboard Events, Composition Events, Keyboard Events, Focus Events, Form Events, Mouse Events, Selection Events, Touch Events, UI Events, Wheel Events, Media Events, Image Events, Animation Events, Transition Events

NASH TECH
The Power to Innovate

FORMS

Forms

- HTML form elements work a little bit differently from other DOM elements in React, because form elements naturally keep some internal state.
- Controlled Components
  - An input form element whose value is controlled by React in this way is called a controlled component.
  - Example:
    - <input type="text" value={this.state.value} onChange={this.handleChange} />
    - <textarea value={this.state.value} onChange={this.handleChange} />
    - <select value={this.state.value} onChange={this.handleChange}>…</select>

Forms

- Uncontrolled Components
  - An input that does not supply a value is an uncontrolled component.
  - To write an uncontrolled component, instead of writing an event handler for every state update, you can use a ref to get form values from the DOM.
  - To initialize the component with a non-empty value, you can supply a defaultValue prop.
  - Example:
    - <input type="text" defaultValue=“Hello" ref={(input) => this.input = input} />

NASH TECH
The Power to Innovate

ROUTER

Router

- React Router is a complete routing library for React.
- React Router keeps your UI in sync with the URL.
- React Router provides three components to get started:
  - Router and Route: Used to declaratively map routes to your application's screen hierarchy.
  - Link: Used to create a fully accessible anchor tag with the proper href.

Router

- Example:
  -
    - <Link to="/about">About</Link>
    - <Link to="/repos">Repos</Link>
  -
  - ReactDOM.render((
    - <Router>
      - <Route path="/" component={App}>
      - <Route path="about" component={About}/>
      - <Route path="repos" component={Repos}/>
    - </Router>
  -), document.getElementById('app'));

NASH TECH
The Power to Innovate

ANIMATION

Animation

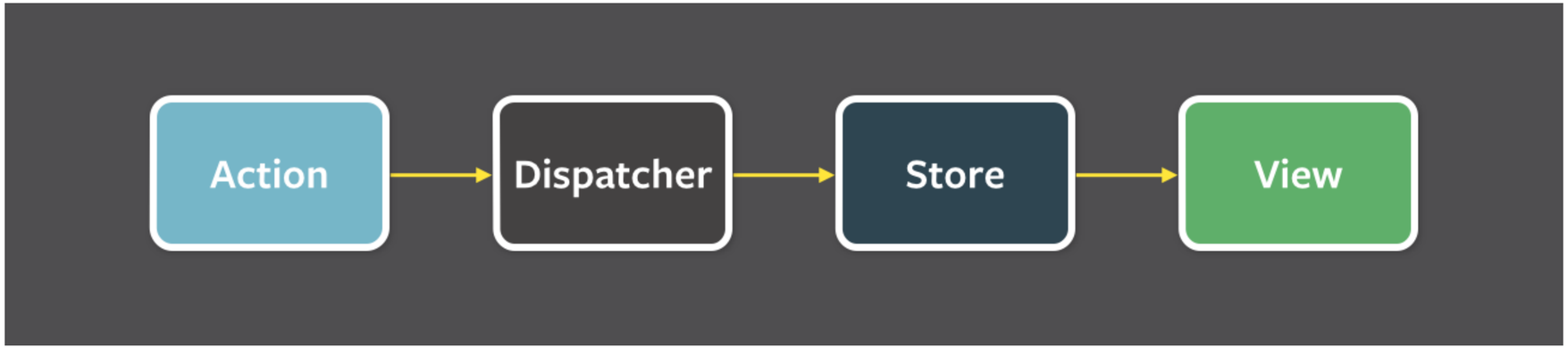
- The ReactTransitionGroup add-on component is a low-level API for animation.
- The ReactCSSTransitionGroup is an add-on component for easily implementing basic CSS animations and transitions.

NASH TECH
The Power to Innovate

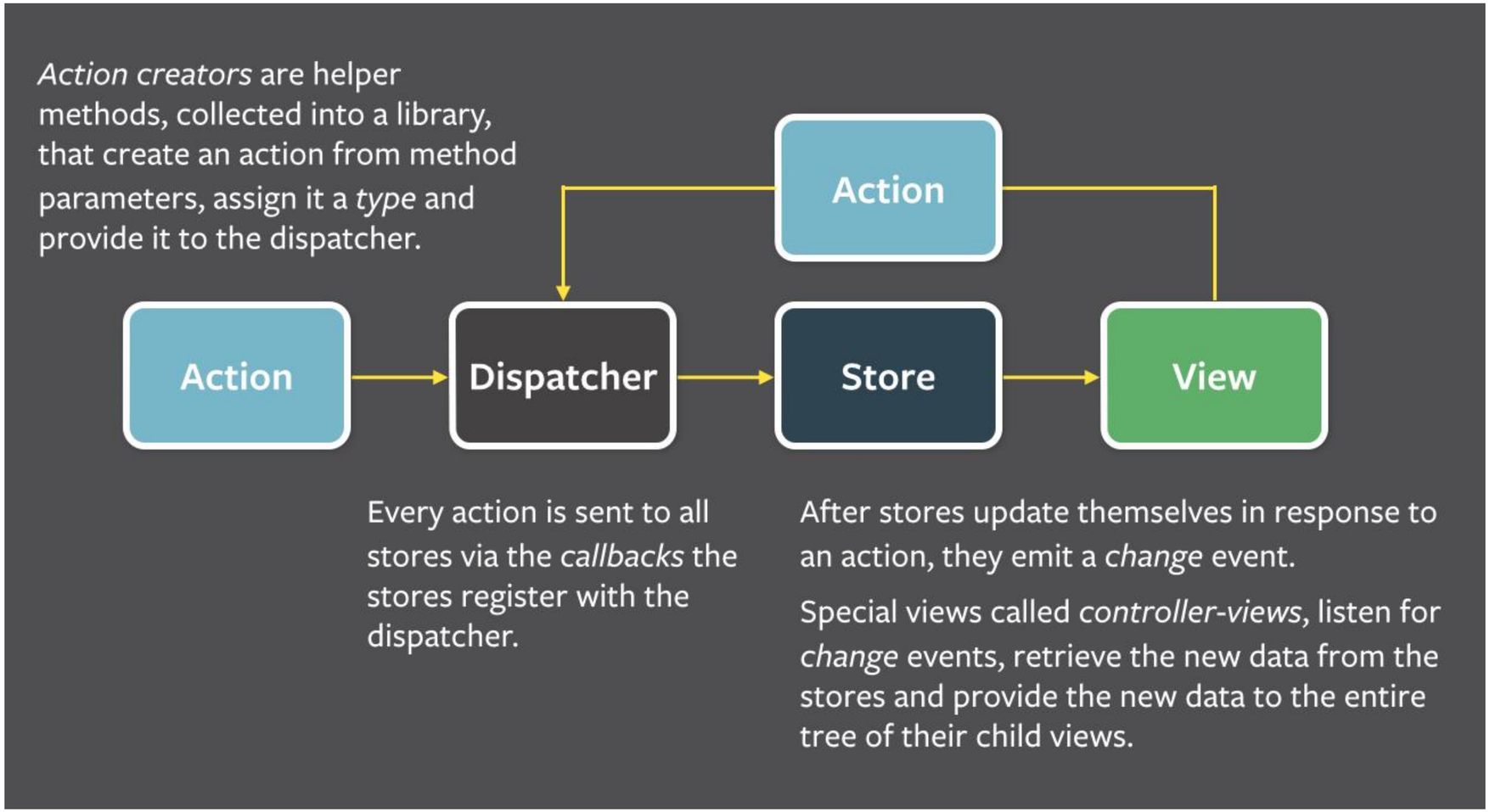
FLUX

Flux

- Flux is the application architecture for building user interfaces.
- It complements React's composable view components by utilizing a unidirectional data flow.
- Flux applications have three major parts: the dispatcher, the stores, and the views.
- Data in a Flux application flows in a single direction



Flux


Action creators are helper methods, collected into a library, that create an action from method parameters, assign it a *type* and provide it to the dispatcher.
Action
Action
Dispatcher
Store
View
Every action is sent to all stores via the *callbacks* the stores register with the dispatcher.
After stores update themselves in response to an action, they emit a *change* event.
Special views called *controller-views*, listen for *change* events, retrieve the new data from the stores and provide the new data to the entire tree of their child views.

NASH TECH
The Power to Innovate

REDUX

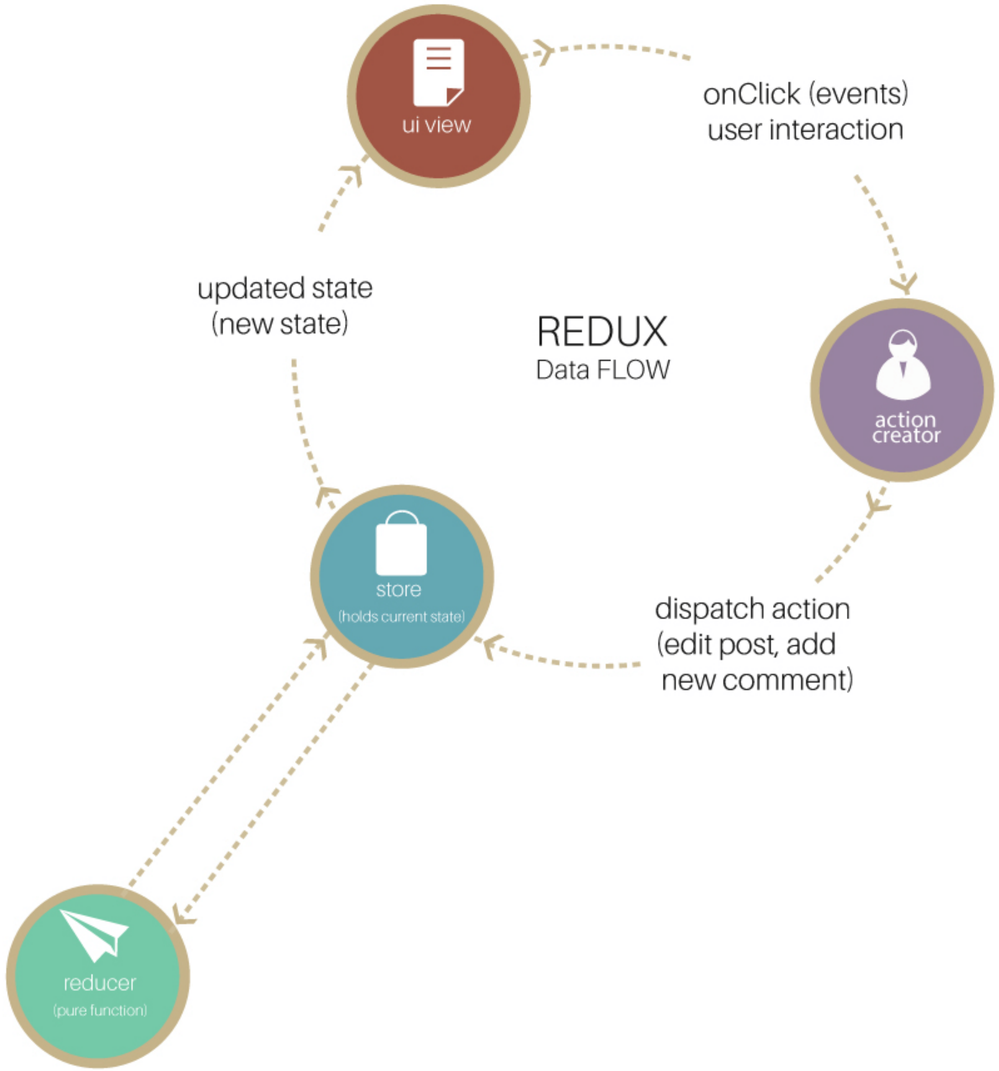
Redux

- Redux is a predictable state container for JavaScript apps.
- Redux can be described in three fundamental principles:
  - Single source of truth: The state of your whole application is stored in an object tree within a single store.
  - State is read-only: The only way to change the state is to emit an action, an object describing what happened.
  - Changes are made with pure functions: To specify how the state tree is transformed by actions, you write pure reducers.

Redux

Redux

- Redux data flow



Redux

- Actions
  - Actions are payloads of information that send data from your application to your store. They are the only source of information for the store. You send them to the store using store.dispatch().
  - Actions are plain JavaScript objects. Actions must have a type property that indicates the type of action being performed.
  - Example:
    - {
    - type: ADD_TODO,
    - text: 'Build my first Redux app'
    - }
  - Actions are created with action creators.

Redux

- Action creators
  - Action creators are exactly functions that create actions.
  - Example:
    - function addTodo(text) {
    - return {
    - type: ADD_TODO,
    - text
    - }
    - }

Redux

- Reducers
  - The reducer is a pure function that takes the previous state and an action, and returns the next state.
  - (previousState, action) => newState

Redux

- Reducers
  - Example:

```
function todoApp(state = initialState, action) {
  switch (action.type) {
    case SET_VISIBILITY_FILTER:
      return Object.assign({}, state, {
        visibilityFilter: action.filter
      })
    default:
      return state
  }
}
```

Redux

- Store
  - Store is the object that holds the application state and provides a few helper methods to access the state, dispatch actions and register listeners.
  - The entire state is represented by a single store.
  - The only way to change the state inside it is to dispatch an action on it.
  - The store has the following responsibilities:
    - Holds application state;
    - Allows access to state via getState();
    - Allows state to be updated via dispatch(action);
    - Registers listeners via subscribe(listener);
    - Handles unregistering of listeners via the function returned by subscribe(listener).

Redux

- Store
  - Example:
    - import { createStore } from 'redux'
    - import todoApp from './reducers'
    - let store = createStore(todoApp)
    - let unsubscribe = store.subscribe(() =>
    - console.log(store.getState())
    -)
    - store.dispatch(addTodo('Build my first Redux app’))
    - unsubscribe()

Redux

- Middleware
  - Redux middleware provides a third-party extension point between dispatching an action, and the moment it reaches the reducer.
  - Use middleware to manage async / side effects.
  - Use Redux Thunk https://github.com/gaearon/redux-thunk
  - Use Redux Saga https://github.com/yelouafi/redux-saga

NASH TECH
The Power to Innovate

TESTING

Testing

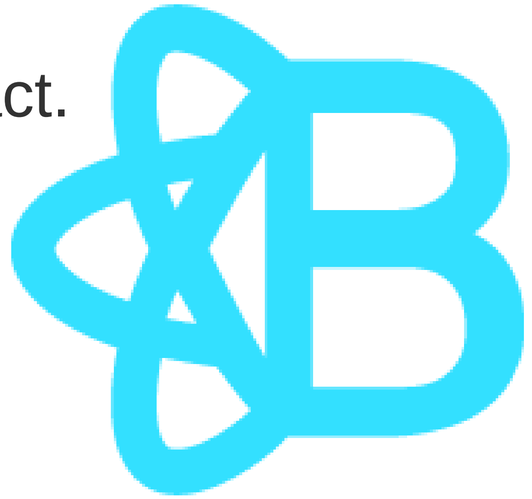
- Jest
  - Jest is a JavaScript testing framework, used by Facebook to test all JavaScript code including React applications.
- Enzyme
  - Enzyme is a JavaScript Testing utility for React that makes it easier to assert, manipulate, and traverse your React Components' output.

NASH TECH
The Power to Innovate

STARTER KIT

Starter Kit

- React Bootstrap
  - Github @ https://react-bootstrap.github.io/
  - The most popular front-end framework, rebuilt for React.



Starter Kit

- React Boilerplate
  - Github @ https://github.com/mxstbr/react-boilerplate
  - A highly scalable, offline-first foundation with the best developer experience and a focus on performance and best practices.

Starter Kit

- Create React App
  - Github @ https://github.com/facebookincubator/create-react-app
  - Create React apps with no build configuration.

Starter Kit

- React Redux Starter Kit
  - Github @ https://github.com/davezuko/react-redux-starter-kit
  - Get started with React, Redux, and React-Router!

Starter Kit

- Redux Minimal
  - Github @ https://redux-minimal.js.org/
  - A minimalist react-redux starter kit (boilerplate) which lets you build rich real world apps.

NASH TECH
The Power to Innovate

REFERENCES

References

- Tutorials
  - https://facebook.github.io/react/
  - https://medium.com/react-tutorials
  - https://www.tutorialspoint.com/reactjs/
  - https://blog.risingstack.com/the-react-way-getting-started-tutorial/
  - https://www.codecademy.com/courses/react-101/
- React Style Guide
  - https://github.com/airbnb/javascript/tree/master/react

References

- Books
  - Introduction to React
  - React.js Essentials
  - Pro React
- Resources
  - https://github.com/enaqx/awesome-react
  - https://github.com/xgrommx/awesome-redux
  - https://github.com/markerikson/react-redux-links
  - https://react-bootstrap.github.io/
  - http://www.material-ui.com/
  - http://elemental-ui.com/

NASH TECH
The Power to Innovate

ASSIGNMENT

Assignment

- Write a simple product listing and shopping cart using React, React Router and Redux.
  - Homepage with welcome message and menu
  - Product listing page to show all products
  - Product detail page to show product information, choose quality and add to cart
  - Shopping cart page to show all items, can update quality and remove items from cart
  - Checkout page to fill the shipping forms and click Order button to submit the data
  - Thank you page to show the ordered information
- See more @ https://scotch.io/tutorials/creating-a-simple-shopping-cart-with-react-js-and-flux and https://react.rocks/example/Jekyll_Store

NASH TECH
The Power to Innovate

Q&A

THANK YOU

www.nashtechglobal.com