React

Pre-requisities

Before we start React, we should have a basic understanding of

- HTML/CSS
- JAVASCRIPT
- JQuery
- Basic Programming Logic

Agenda

After completing this session you would be able to understand the below topics.

- React and basic tools
- React concepts
- Environmental setup
- Demo

What is REACT?

- React is open source library that came out of Facebook.
- It is a view layer that allows you to create components that are in turn composed of other components.
- Its greatest contribution to the JS landscape is the solidifying and popularizing the concept of one-way data flow as applied JS user interface construction.
- It does use a virtual DOM to achieve great performance in the browser. While really great feature and well implemented, this only makes React feasible , not desirable.

Advantages and Limitations

Advantages

- React uses virtual DOM which is JavaScript object. It will improve apps performance since JavaScript virtual DOM is faster than the regular DOM.
- Component and Data patterns improve readability which helps to maintain larger apps.
- React can be easily integrated with other frameworks.

Limitations

- React covers view layer of the app so we still need to choose other technologies to get a complete tooling set for development.

Companies

Flipboard NETFLIX

Adobe YAHOO! Dropbox

PayPal The New York Times airbnb

reddit BBC Pinterest

Core Technologies

Server-side JavaScript

Node packages in the browser

Components

Routing

Unidirectional data flows

Task runner

Node

- Server-side JS
- Uses V8 Engine
- Include npm package manager

Browserify

- Use Node modules in the browser
- Bundle dependencies

React

- Component library
- Utilize virtual DOM
- Can render on client and server

React Router

- Nested views map to nested routes
- Used at Facebook
- Inspired by Ember

Flux

- Uni –directional data flows
- More a pattern than a library

Gulp

- Task runner
- Rich plugin ecosystem
- Stream based

Versions

- React latest version
- React-Router latest version
- Flux latest version

Testing

Jest
Docs
API
Help
Blog
Search
GitHub
Jest
Painless JavaScript Testing
TRY OUT JEST
GET STARTED
LEARN MORE
Star
8,695

ES2015

- ES2015(ES6)
- Approved June 2015
- Browser support is growing
- Can transpile to ES5 via Babel

Babel

Babel is a JavaScript compiler.

Use next generation JavaScript, today.

Environmental Setup

Installation

- Install node
- Create package.json
- npm install
- Type gulp

Gulp

- Compile React JSX
- Lint JSX and JS via ESLint
- Bundle JS and CSS file
- Migrate the build app to the dist folder
- Run a dev web server
- Open the browser at your dev URL
- Reload the browser upon save

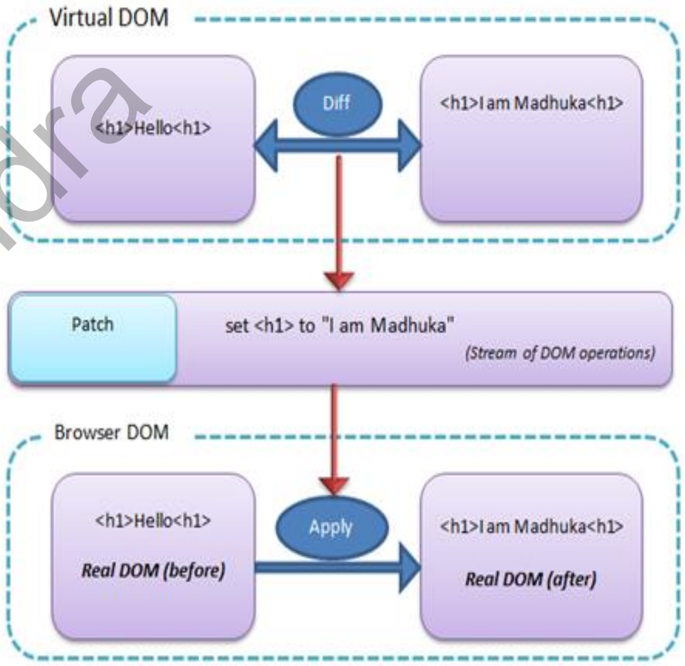
Note: Global libraries: %AppData%\npm\node_modules

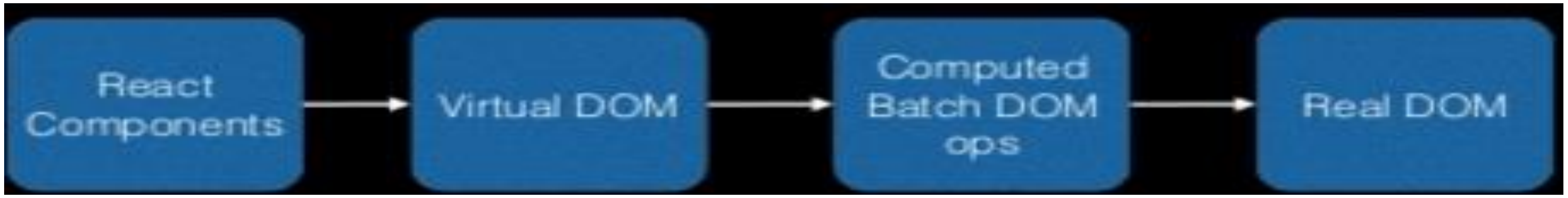
Data Flow

- React views are made of component, which is in turn made of components, which is in turn made of up of more components. Turtles all the way down.
- Data only flows from the parents to the children
- Children thus only have the logic to display the data, not modify it.
- Children also handle events and then inform parents via callback or events. Parents then modify their own state.
- Thus when you unexpected behavior it can only live in a select few places.
- This comes at the cost of being a bit verbose.

Virtual DOM

- React maintains a pure JavaScript version of the DOM.
- All changes are done in the JS virtual DOM and very fast
- React calculates the changes and updates the real DOM in a sequence of fast and optimized transactions
- Can run on Node.js without any browser
- Highly testable

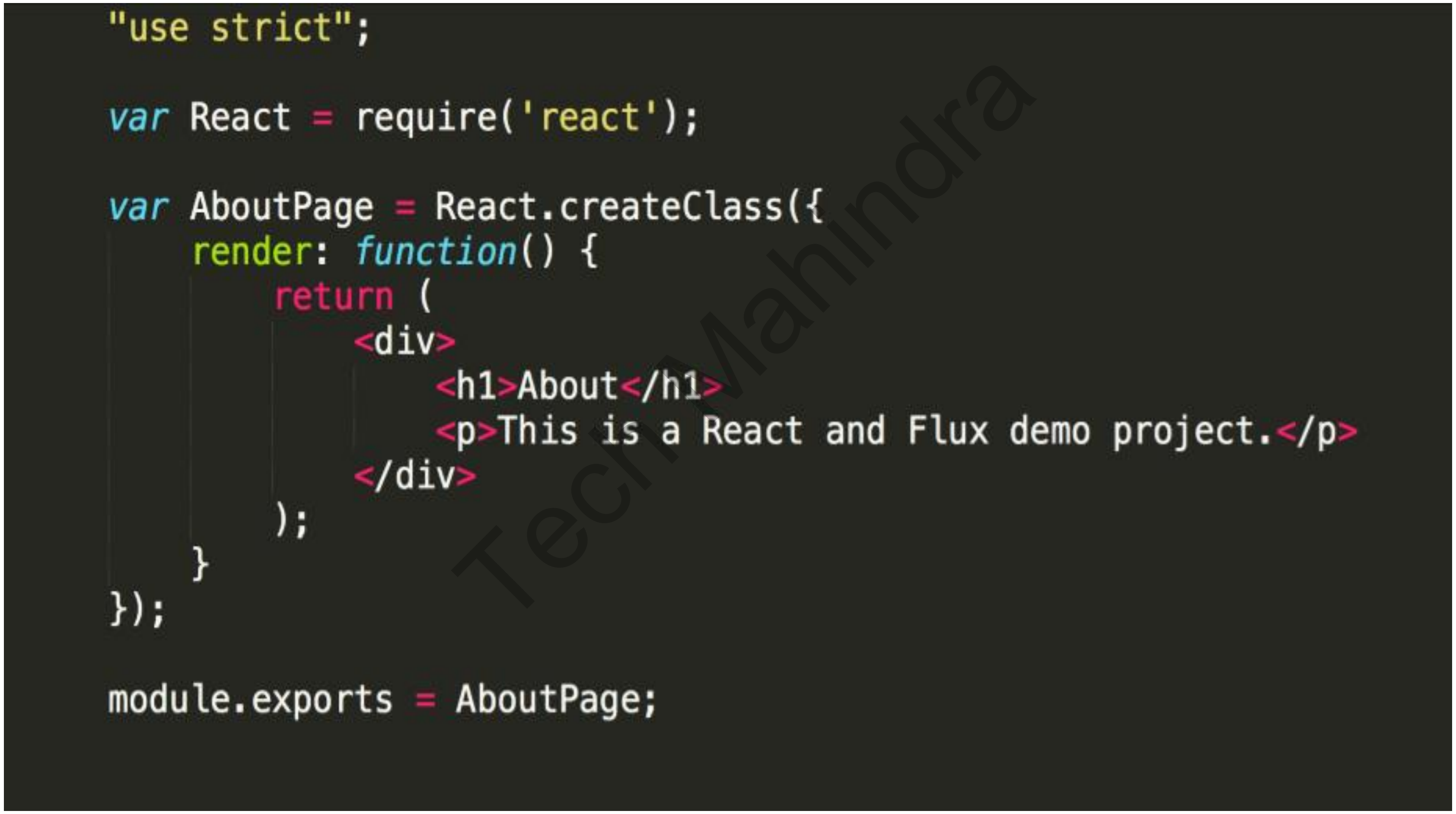




JSX

- In a nutshell, JSX lets you write markup(HTML)straight in your JavaScript.
- Babel(our ES6 transpiler)already has built in JSX support
- For production environment we use the Gulp plugin(JSX->JS)

JSX



```
"use strict";

var React = require('react');

var AboutPage = React.createClass({
    render: function() {
        return (
            <div>
                <h1>About</h1>
                <p>This is a React and Flux demo project.</p>
            </div>
        );
    }
});

module.exports = AboutPage;
```

JSX Compiles to JS

```
<h1 color="red">Heading here</h1>
```

↓

```
React.createElement("h1", {color: "red"}, "Heading here")
```

JSX editors

ATOM

2015

Hands on

Let us create our first React component using JSX

Why Lint?

- Avoid errors
- Enforce best practices
- Maintain code consistency
- Many to choose from
  - JSLint
  - JSHint
  - ESHint

Linting with ESLint

1

```
npm install --save-dev gulp-eslint
```

2

```
"eslintConfig": {
  "ecmaFeatures": {
    "jsx": true
  },
  "env": {
    "browser": true,
    "node": true
  },
  "rules": {
    //rules here…
  }
},
```

3

```
gulp.task('lint', function () {
  return gulp.src(config.paths.js)
    .pipe(eslint())
    .pipe(eslint.format())
});
```

Hands on

Comparision

	Angular	**Backbone**	**React**
Type	MV*	MVC	V
Technology	HTML/CSS/JS/Angular	HTML/CSS/JS/backbone	JSX
Core	MVC	MVC	Components
View	HTML	HTML	Virtual DOM
Data Flowing	2 way binding	-	Unidirectional
Creator	Google	-	FB* & Instagram
Architchure	-	-	React Native & Flux
SEO Support	Phantom js	-	SEO friendly

Reference

- https://www.youtube.com/watch?v=x7cQ3mrcKaY
- https://github.com/enaqx/awesome-react
- http://facebook.github.io/react/docs/getting-started.html
- http://facebook.github.io/react/docs/videos.html
- https://speakerdeck.com/pedronauck/reactjs-keep-simple-everything-can-be-a-component
- http://javascript.tutorialhorizon.com/2014/09/13/execution-sequence-of-a-react-components-lifecycle-methods/

Thank you

Visit us at www.techmahindra.com

Please write to us your suggestion
Sasirekha.Perumalvijayan@techmahindra.com

Tech Mahindra

React

Pre-requisities

Before we start React, we should have a basic understanding of

- HTML/CSS
- JAVASCRIPT
- JQuery
- Basic Programming Logic

Agenda

After completing this session you would be able to understand the below topics.

- Props
- State
- Dynamic child components
- Lifecycle methods
- Demo
- Composition
- Controller views
- Prop validation via PropTypes
- Mixins

Props and State

- Props – Looks like HTML attributes, but immutable

this.props.username

- state – Holds mutable state

this.state.username

Initial State

- getInitialState

Default Prop Values

- getDefaultProps

Life Cycle Methods

Handle bootstrapping and third party integrations

- componentWillMount
- componentDidMount
- componentWillReceiveProps
- shouldComponentUpdate
- componentWillUpdate
- componentDidUpdate
- componentWillUnmount

componentWillMount

When
Before initial render, both client and server

Why
Good spot to set initial state

componentDidMount

When
After render

Why
Access DOM, integrate with frameworks, set timers, AJAX requests

componentWillReceiveProps

When
When receiving new props. Not called on initial render.

Why
Set state before a render.

shouldComponentUpdate

When
Before render when new props or state are being received. Not called on initial render.

Why
Performance. Return false to void unnecessary re-renders.

componentWillUpdate

When
Immediately before rendering when new props or state are being received. Not called on initial render.

Why
Prepare for an update

componentDidUpdate

When

After component's updates are flushed to the DOM. Not called for the initial render.

Why

Work with the DOM after an update

componentWillUnmount

When
Immediately before component is removed from the DOM

Why
Cleanup

Keys for Dynamic Children

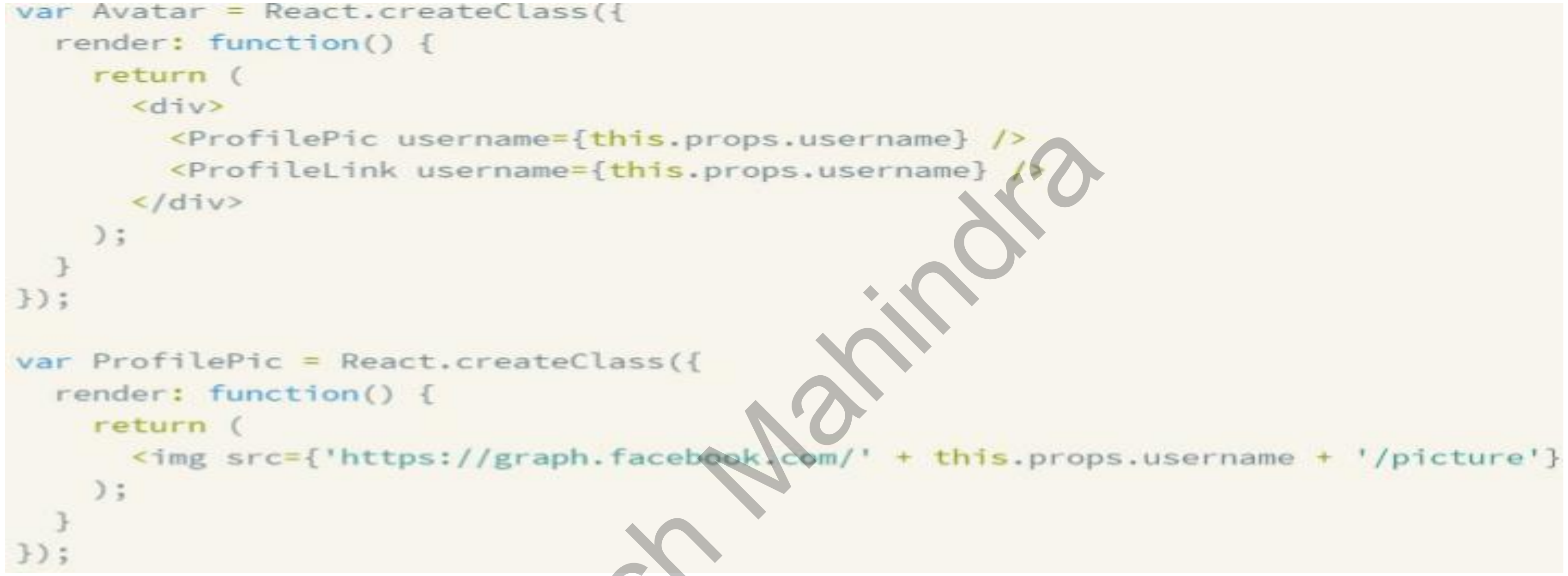
Add a key to dynamic child elements

```
<tr key={author.id}
```

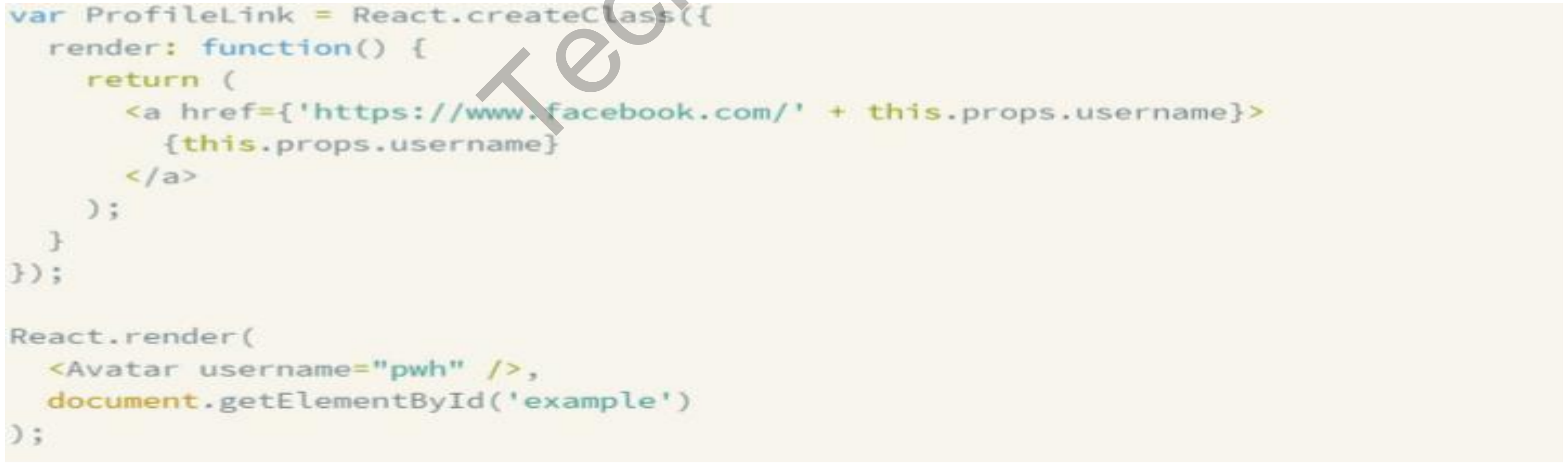
Summary

- Props – pass data to child components
- State – Data in controller view
- Lifecycle – Handle bootstrapping and third party integrations

Composing components



```
var Avatar = React.createClass({
  render: function() {
    return (
      <div>
        <ProfilePic username={this.props.username} />
        <ProfileLink username={this.props.username} />
      </div>
    );
  }
});

var ProfilePic = React.createClass({
  render: function() {
    return (
      <img src={'https://graph.facebook.com/' + this.props.username + '/picture'}
    );
  }
});
```



```
var ProfileLink = React.createClass({
  render: function() {
    return (
      <a href={'https://www.facebook.com/' + this.props.username}>
        {this.props.username}
      </a>
    );
  }
});

React.render(
  <Avatar username="pwh" />,
  document.getElementById('example')
);
```

Controller view

Top level component

Set props on children

Interact with stores

Prop Validation

```
propTypes: {

author:    React.PropTypes.object.isRequired,
onSave:    React.PropTypes.func.isRequired,
validate:  React.PropTypes.func.isRequired,
errors:    React.PropTypes.object,
hasErrors: React.PropTypes.func.isRequired,

 },
```

Prop Validation

optionalArray: React.PropTypes.array,
optionalBool: React.PropTypes.bool,
optionalFunc: React.PropTypes.func,
optionalNumber: React.PropTypes.number,
optionalObject: React.PropTypes.object,
optionalString: React.PropTypes.string,

Mixins

For cross-cutting concerns

Share code between multiple components

```
varManageAuthorPage= React.createClass({
mixins: [
Router.Navigation,
Router.State,
], …
```

Summary

Controller Views - “Smart” components that pass data down via props

PropTypes - Validate props

Mixins - Share behavior among multiple components

Statics - Define static methods

State

- Every component has a State object.
- Can be set by using setState.
- setState triggers UI updates and to get the initial state before the setState : getInitialState. getDefaultProps

Thank you

Visit us at www.techmahindra.com

Please write to us your suggestion
Sasirekha.Perumalvijayan@techmahindra.com

React

Pre-requisities

Before we start React, we should have a basic understanding of

- HTML/CSS
- JAVASCRIPT
- JQuery
- Basic Programming Logic

Agenda

After completing this session you would be able to understand the below topics.

- React Router
- React Forms

React Router

- Nested views map to nested routes
- Declarative
- Used at Facebook
- Inspired by Ember

Route Configuration

Route – Declaratively map a route

DefaultRoute – For URL of "/." Like "index.html"

NotFoundRoute – Client-side 404

Redirect – Redirect to another route

Params and Querystrings

```
// Given a route like this:
<route path="/course/:courseId" handler={Course} />



// and a URL like this:
'/course/clean-code?module=3'



//The component's props will be populated
var Course = React.createClass({
   render: function() {
      this.props.params.courseId; // "clean-code"
      this.props.query.module; // "3"
      this.props.path; // "/course/clean-code/?module=3"
      // ...
   }
});
```

Links

URL: /user/1

Route:
```
<route name="user" path="/user/:userId" />
```

JSX:
```
<Link to="user" params={{userId: 1}}>Bobby Tables</Link>
```

↓

```
<a href="/user/1">Bobby Tables</a>
```

Redirects

Need to change a URL? Use a Redirect.

Alias Redirect	var Redirect = Router.Redirect;
Create a new route	<Redirect from="old-path" to="name-of-new-path" />

Handling Transitions

willTransitionTo – Determine if page should be transitioned to

willTransitionFrom – Run checks before user navigates away

Handling Transitions

```
var Settings = React.createClass({
    statics: {
        willTransitionTo: function (transition, params, query, callback) {
            if (!isLoggedIn) {
                transition.abort();
                callback();
            });
        },

        willTransitionFrom: function (transition, component) {
            if (component.formHasUnsavedData()) {
                if (!confirm('Sure you want to leave without saving?')) {
                    transition.abort();
                }
            }
        }
    }

    //...
});
```

Transitions

- Handle transition using willTransitionTo and willTransitionFrom

Hash vs. History URLs

Hash Location

yourUrl.com#/courses

- Ugly URLs
- Works in all browsers
- Not compatible with server-render

History Location

- yourUrl.com/courses
- Clean URLs
- IE 10+
- Works for server-render

Mixins

- For cross-cutting concerns
- Share code between multiple components

```
Var ManageAuthorPage=React.createClass({

Mixins:[

Router.Navigation,
Router.State,

],
…
```

Navigation Mixin

//Go to a new route
This.transitionTo('contact')

//Replace current route
This.replaceWith('contact')

//Go back
This.goBack

//create a URL to a route
makePath(routeName,params,query)

React Forms

Forms should be easy ,Right?

- Validation
- Redirects
- Reusable inputs
- User notifications
- Saving an Population on load

Controlled Components

- Any <input> with a value set is a controlled component.
- The element's value *always* matches the value of the assigned prop.

Robust Forms in React…complete!

- Any input with a value is a controlled component
- Create reusable inputs
- Saved data via AJAX calls in controller view
- Programmatically redirected after save via ReactRouter
- Used Toastr to display notification
- Added validation
- Used PropType to declare expectations
- Used willTransitionFrom to protect from losing work
- Populated form in willComponentMount

Summary

- Declarative routing configuration
- “Normalized” anchors via Link
- Navigation Mixin
- Form designing in React

Tech Mahindra

Thank you

Visit us at www.techmahindra.com

Please write to us your suggestion
Sasirekha.Perumalvijayan@techmahindra.com

TLS

React - Flux

Pre-requisities

Before we start React, we should have a basic understanding of

- HTML/CSS
- JAVASCRIPT
- JQuery
- Basic Programming Logic

Agenda

After completing this session you would be able to understand the below topics.

- Flux

Flux

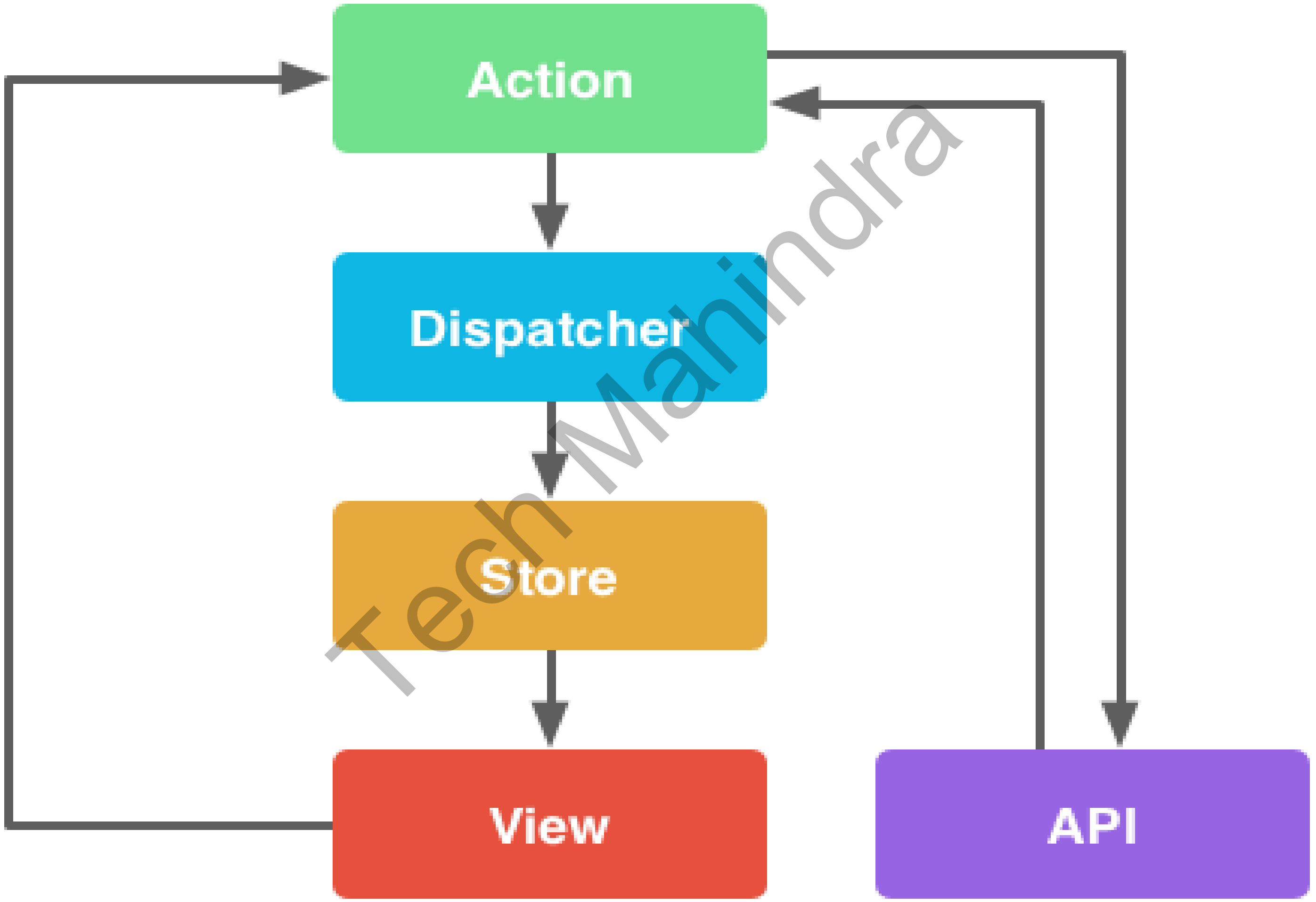
- Flux is an architecture that Facebook uses internally when working with React.
- It is *not* a framework or a library.
- It is simply a new kind of architecture that complements React and the concept of Unidirectional Data Flow.
- Facebook does provide a repo that includes a Dispatcher library. The dispatcher is a sort of global pub/sub handler that broadcasts payloads to registered callbacks.
- A typical Flux architecture will leverage this Dispatcher library, along with NodeJS's EventEmitter module in order to set up an event system that helps manage an applications state.

Flux Components

Flux is probably better explained by explaining its individual components:

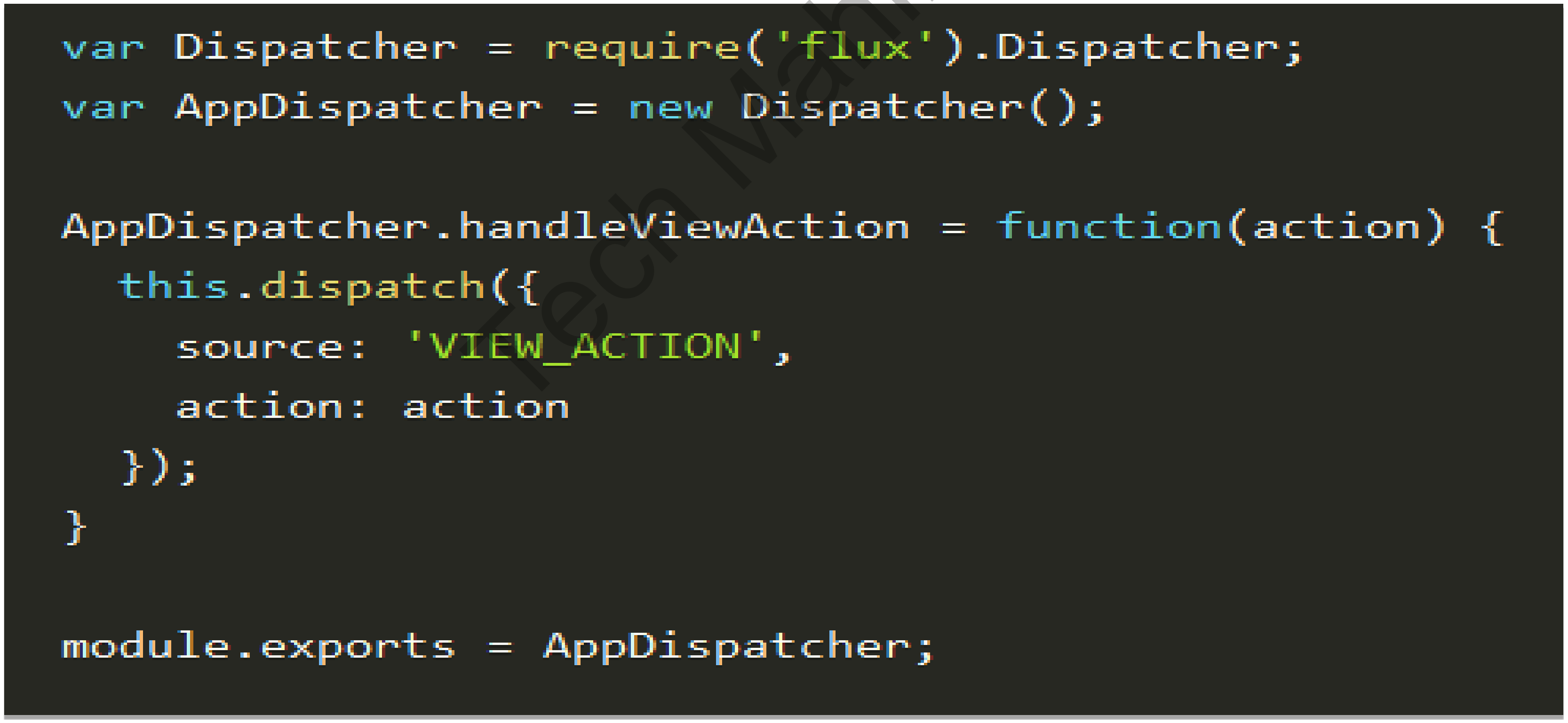
- **Actions** – Helper methods that facilitate passing data to the Dispatcher
- **Dispatcher** – Receives actions and broadcasts payloads to registered callbacks
- **Stores** – Containers for application state & logic that have callbacks registered to the dispatcher
- **Controller Views** – React Components that grab the state from Stores and pass it down via props to child components.

Flux Components



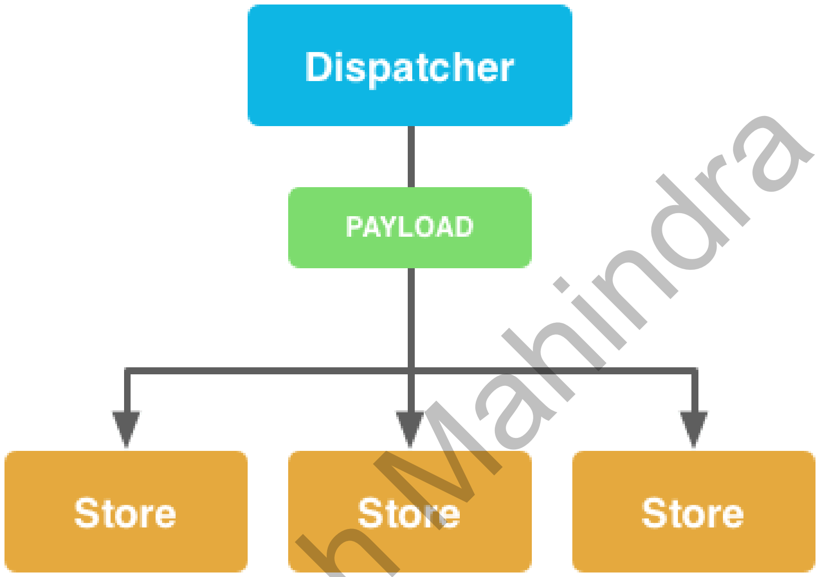
Dispatcher

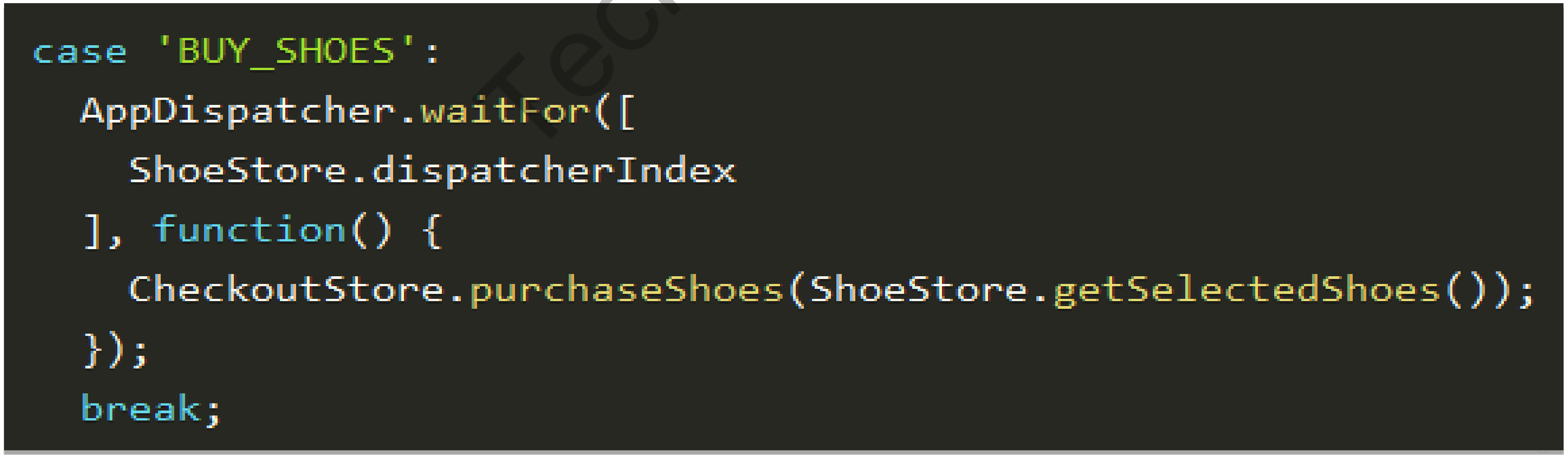
The Dispatcher is basically the manager of this entire process. It is the central hub for your application. The dispatcher receives actions and dispatches the actions and data to registered callbacks.



```
var Dispatcher = require('flux').Dispatcher;
var AppDispatcher = new Dispatcher();

AppDispatcher.handleViewAction = function(action) {
  this.dispatch({
    source: 'VIEW_ACTION',
    action: action
  });
}

module.exports = AppDispatcher;
```

Dispatcher



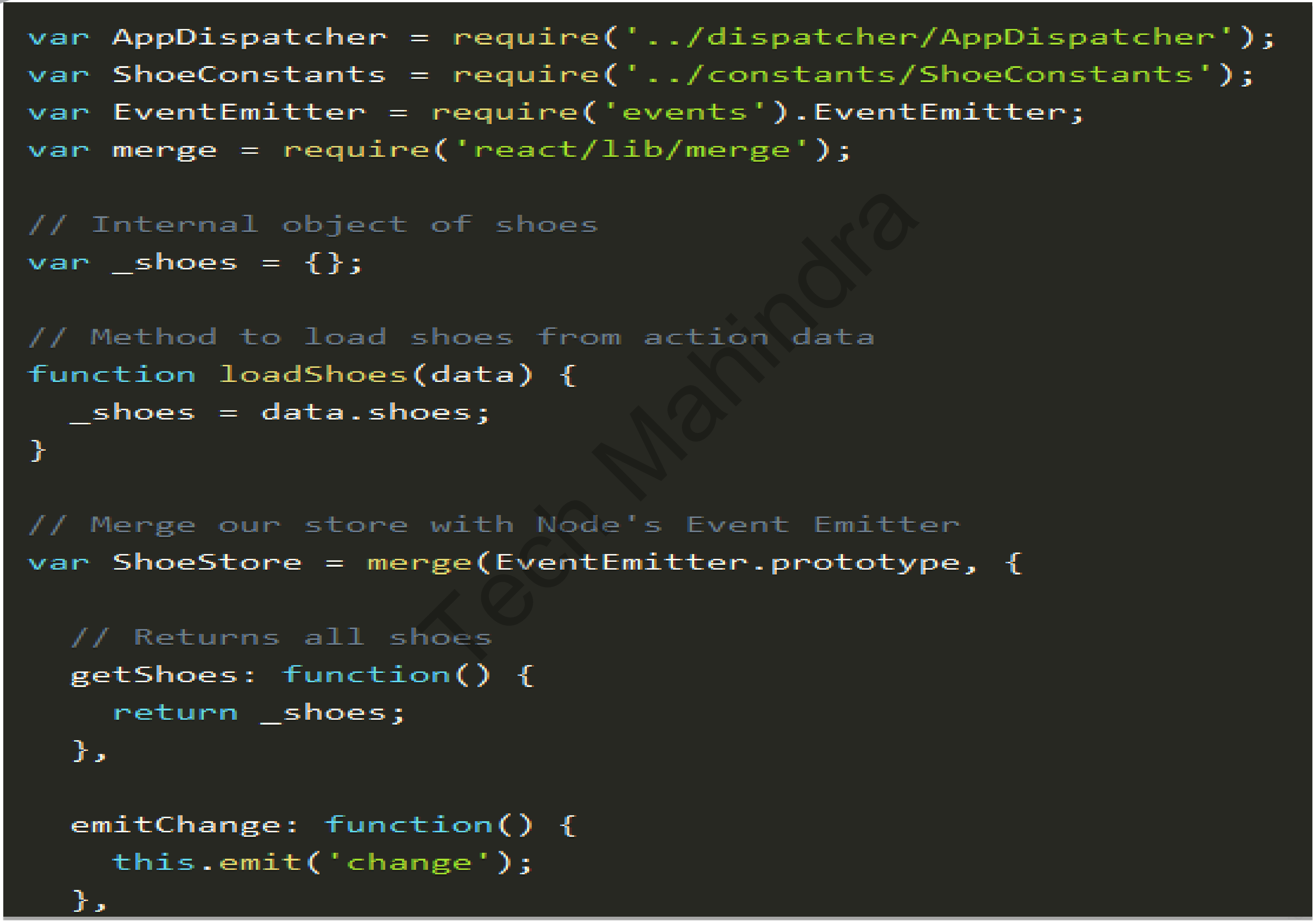


```
case 'BUY_SHOES':
  AppDispatcher.waitFor([
    ShoeStore.dispatcherIndex
  ], function() {
    CheckoutStore.purchaseShoes(ShoeStore.getSelectedShoes());
  });
  break;
```

Stores

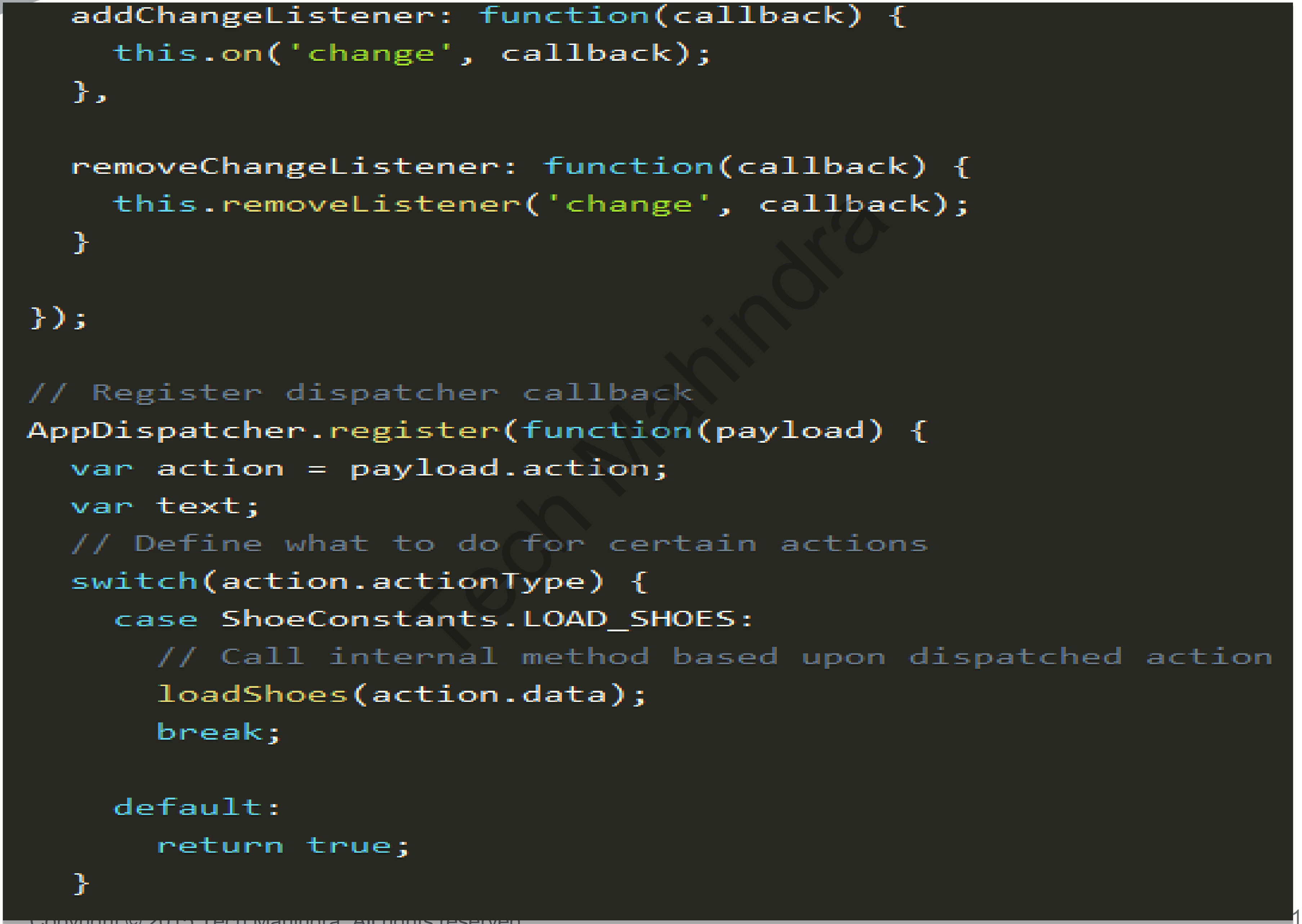
In Flux, Stores manage application state for a particular domain within your application. From a high level, this basically means that per app section, stores manage the data, data retrieval methods and dispatcher callbacks.

Stores



```
var AppDispatcher = require('../dispatcher/AppDispatcher');
var ShoeConstants = require('../constants/ShoeConstants');
var EventEmitter = require('events').EventEmitter;
var merge = require('react/lib/merge');

// Internal object of shoes
var _shoes = {};

// Method to load shoes from action data
function loadShoes(data) {
  _shoes = data.shoes;
}

// Merge our store with Node's Event Emitter
var ShoeStore = merge(EventEmitter.prototype, {

  // Returns all shoes
  getShoes: function() {
    return _shoes;
  },

  emitChange: function() {
    this.emit('change');
  },
```

Stores



```
   addChangeListener: function(callback) {
     this.on('change', callback);
   },

   removeChangeListener: function(callback) {
     this.removeListener('change', callback);
   }

});

// Register dispatcher callback
AppDispatcher.register(function(payload) {
   var action = payload.action;
   var text;
   // Define what to do for certain actions
   switch(action.actionType) {
     case ShoeConstants.LOAD_SHOES:
       // Call internal method based upon dispatched action
       loadShoes(action.data);
       break;

     default:
       return true;
   }
```

Stores

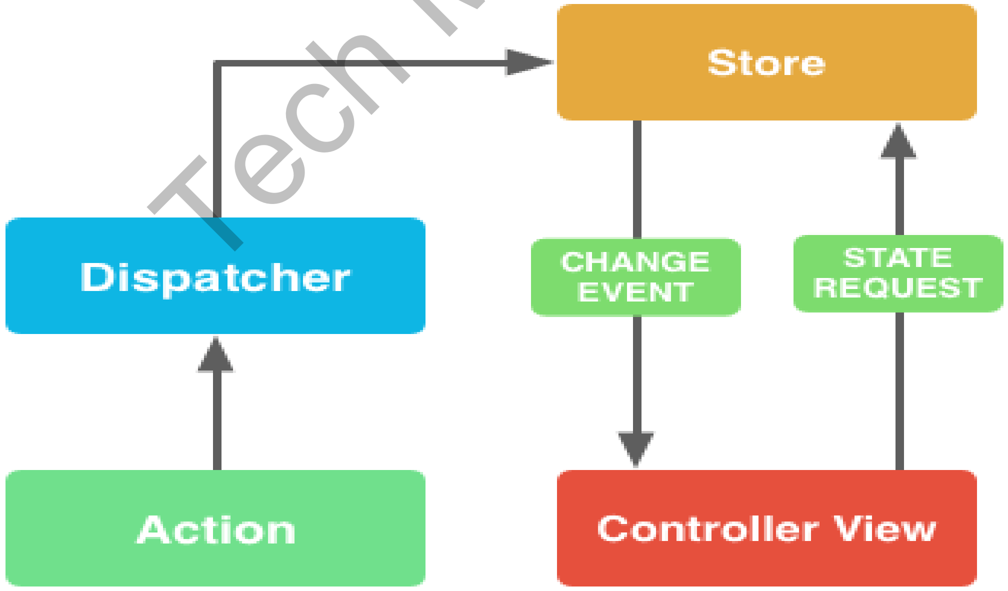
```
  // If action was acted upon, emit change event
  ShoeStore.emitChange();

  return true;

});

module.exports = ShoeStore;
```

The most important thing we did above is to extend our store with NodeJS's **EventEmitter**. This allows our stores to listen/broadcast events. This allows our Views/Components to update based upon those events. Because our Controller View listens to our Stores, leveraging this to emit change events will let our Controller View know that our application state has changed and its time to retrieve the state to keep things fresh.

Stores

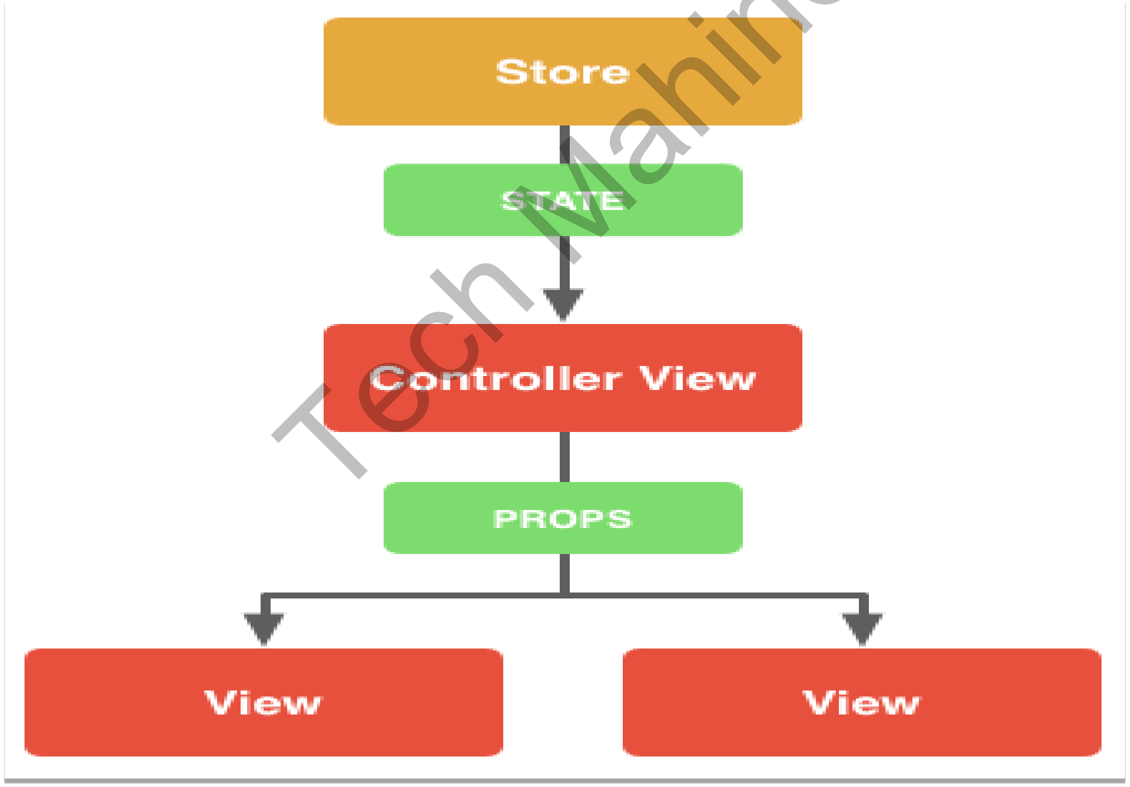
We also registered a call-backs with our AppDispatcher using its register method. This means that our Store is now listening to AppDispatcher broadcasts. Our switch statement determines whether, for a given broadcast, if there are any relevant actions to take.

If a relevant action is taken, a change event is emitted, and views that are listening for this event update their states.

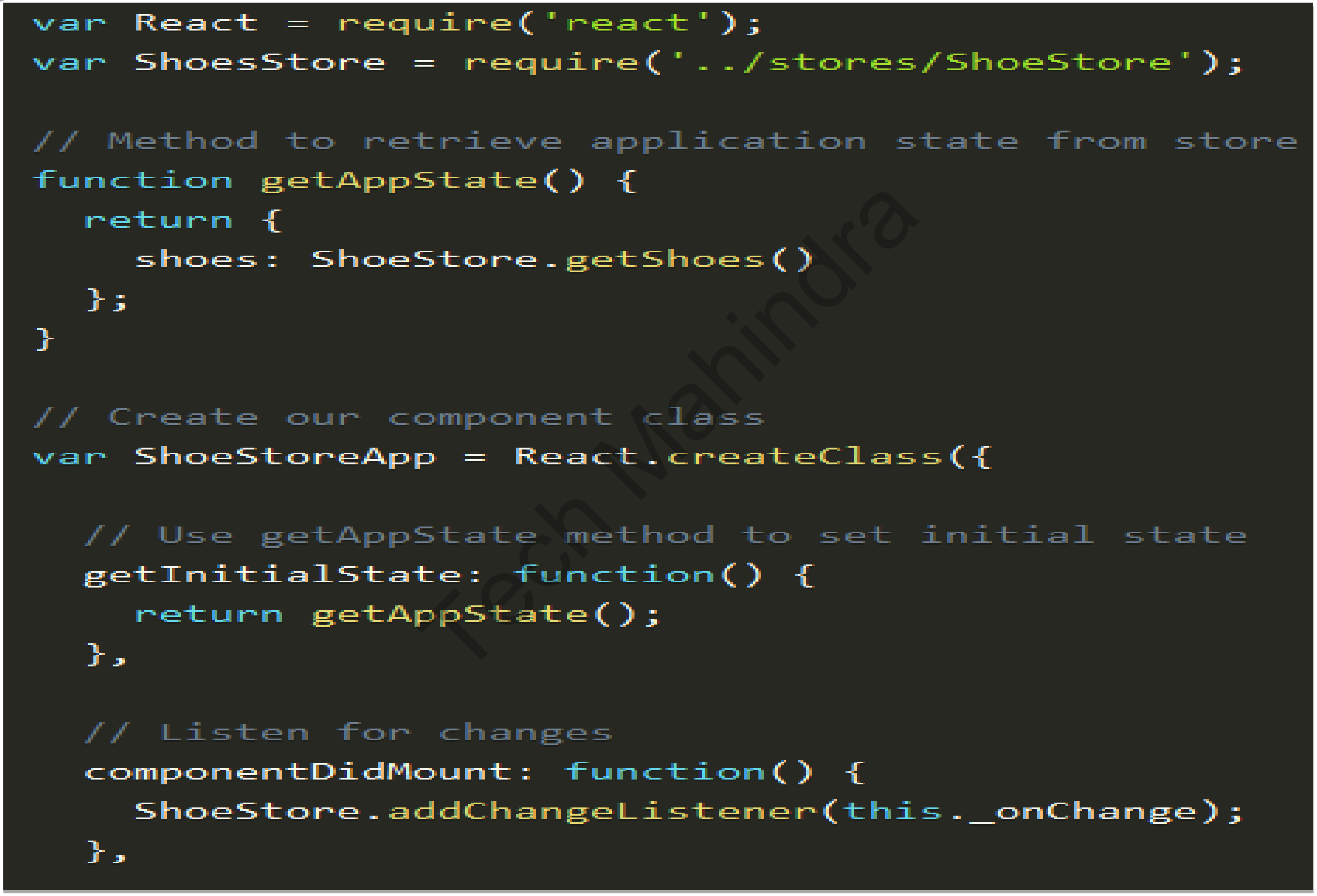


Controller Views

Controller views are really just React components that listen to change events and retrieve Application state from Stores. They then pass that data down to their child components via props.

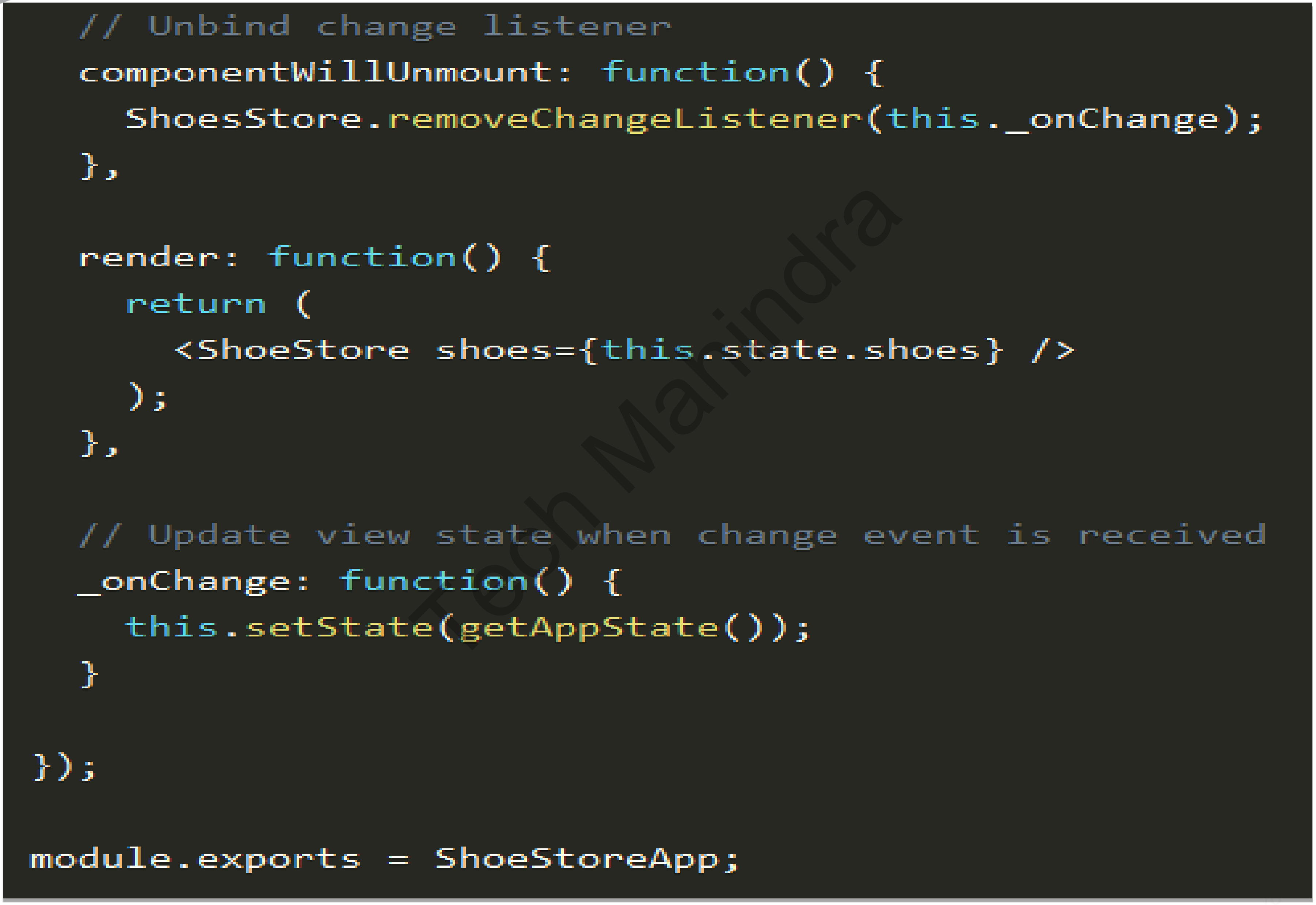


Controller Views



```
var React = require('react');
var ShoesStore = require('../stores/ShoeStore');

// Method to retrieve application state from store
function getAppState() {
  return {
    shoes: ShoeStore.getShoes()
  };
}

// Create our component class
var ShoeStoreApp = React.createClass({

  // Use getAppState method to set initial state
  getInitialState: function() {
    return getAppState();
  },

  // Listen for changes
  componentDidMount: function() {
    ShoeStore.addChangeListener(this._onChange);
  },
```

Controller Views

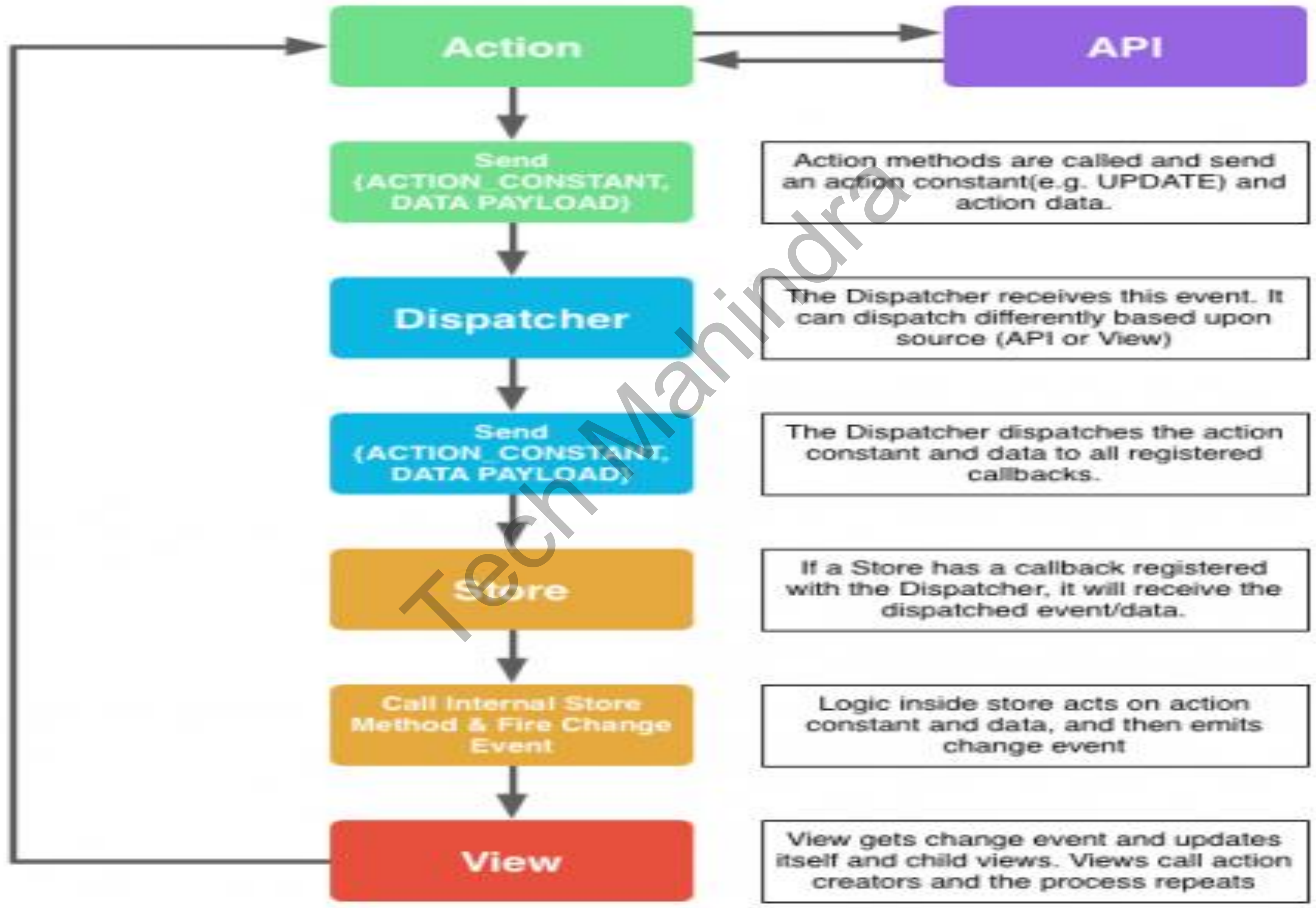


```
  // Unbind change listener
  componentWillUnmount: function() {
    ShoesStore.removeChangeListener(this._onChange);
  },

  render: function() {
    return (
      <ShoeStore shoes={this.state.shoes} />
    );
  },

  // Update view state when change event is received
  _onChange: function() {
    this.setState(getAppState());
  }

});

module.exports = ShoeStoreApp;
```

Controller Views

1. In the example above, we listen for change events using addChangeListener, and update our application state when the event is received.
2. Our application state data is held in our Stores, so we use the public methods on the Stores to retrieve that data and then set our application state.

Flux ALL



Action
API
Send {ACTION_CONSTANT, DATA PAYLOAD}
Dispatcher
Send {ACTION_CONSTANT, DATA PAYLOAD}
Store
Call Internal Store Method & Fire Change Event
View

Action methods are called and send an action constant(e.g. UPDATE) and action data.

The Dispatcher receives this event. It can dispatch differently based upon source (API or View)

The Dispatcher dispatches the action constant and data to all registered callbacks.

If a Store has a callback registered with the Dispatcher, it will receive the dispatched event/data.

Logic inside store acts on action constant and data, and then emits change event

View gets change event and updates itself and child views. Views call action creators and the process repeats

Thank you

Visit us at www.techmahindra.com

Please write to us your suggestion
Sasirekha.Perumalvijayan@techmahindra.com

Tech
Mahindra