**State –** like a save point where all data of the program at the specific time is saved in. (Note: if the program/script is not expected to change, non-dynamic, then we shouldn’t be using setState in React. Props should be used instead.)

**Data flow** in React is one way, from parent to child.

**Prop** – things that are passed down through your component

**Routing –** technique of setting an url to match a certain part of the page.

**Bot** – can set up a bot by going to dev.botframework.com

**React**

{} – use when we want to use js code in html

**Create Component**

//create a component named MessageComponent

Var MessageComponent = React.createClass({ // create a class in react

Render: function(){ //what we want the this component to do

Return (

<div>{this.props.message}</div> // we want it to return the message attached

);

}

});

// Render an instance of MessageComponent into document.body (note: this part is always needed to activate the component we want

ReactDOM.render(

< MessageComponent message =”hello!” />, //we want to display the message hello!

document.body // we want to display the component in the <body>

);

**Interactivity and dynamic UIs**

//create a component named MessageComponent

Var likeButton = React.createClass({ // create a class in react

getInitialState: function(){ //state is set inside the component

Return {liked: false}; //set initial state to be not liked

},

handleClick: function(event){

this.setState({liked: !this.state.liked}); //change the sate to be the opposite of what it was

},

Render: function(){

Var text = this.state.liked ? ‘like’ : ‘havent\’t liked’; //test if the state is liked or not

Return {

<p onClick={this.handleClick}>

You {text} this. Click to toggle, // toggle the handleClick when cliked on

</p>

);

}

});

ReactDOM.render(

< LikeButton />,

document.getElementbyId(‘example’) // we want to display the component in the id “example”

);

Note: with react, you can use ref=”name” and use this.refs.name.value to get the reference input.

**Routing**

The Router, itself is a component.

* Each path is imbedded in route which shall be match with it component.

To start routing, first you must npm install the react-route package at <https://github.com/ReactTraining/react-router> and set the following:

var ReactRouter = require('react-router');

var browserHistory = ReactRouter.browserHistory;

var Route = ReactRouter.Route;

var Router = ReactRouter.Router;

var Link = ReactRouter.Link;

To set the route link, React Render must be changed to:

ReactDOM.render(

<Router history={browserHistory}>

<Route path="/" component={App}>

<Route path="/conversation/:human" component={ConversationPane}></Route> //this will change the web link to abc.com/conversation/(name). When it goes to that link it will call the ConversationPane component after the App component since this route is in the route above.

</Route>

</Router>, document.getElementById('main'));

**Route-Mixin**

This will increase the efficiency of react as it will:

* Skip the re-rendering of components that are fix. (no additional info during update)

To include Mixin:

1. Add the mixin by:  
   var PureRenderMixin = require('react-addons-pure-render-mixin');
2. Add the mixing to the component by inserting:  
   mixins:[PureRenderMixin],

**React Developer Tools**

* Extension from the browser (need to be dled)
* Connect browser to React by adding a react tab

**Connecting to bot framework**

* First, you need to require the bot builder by:  
   var builder = ('botbuilder');
* Then, link it to the bot api. (where the bot is)by:  
   var model = 'https://api.projectoxford.ai/luis/v1/application?id=' + process.env.LUIS\_ID + '&subscription-key=' + process.env.LUIS\_KEY //link of the apir

var dialog = new builder.LuisDialog(model) ; //what we want from the link

var bot = new builder.BotConnectorBot({ appId: process.env.APP\_ID, appSecret: process.env.APP\_SECRET }); //connect to the bot

bot.add('/', dialog); //tell the computer to access the dialog section in the bot link.