REACT & REACT NATIVE TRAINING

DAY 2

BY

KIRAN KUMAR ABBURI

QUESTIONS DISCUSSION

GIT SETUP TO SHARE CODE

ES2015 REFRESHER

- New features of javascript
- Useful for writing concise code
- Need to use babel for ES2015 to ES5 transpilation
- We anyway need babel for JSX to Js transformation

ARROW FUNCTIONS

- => syntax for function shorthand

ES2015

```
const completedTodos = todos.filter(todo => todo.completed)
```

ES5

```
var completedTodos = todos.filter(function (todo) {
    return todo.completed
})
```

ARROW FUNCTIONS

- arrows share the same lexical this as their surrounding code

ES2015

```
{
  counter: 0,
  incrementCounter() {
    setInterval(() => this.counter = this.counter + 1, 1000)
  }
}
```

ES5

```
{
  counter: 0,
  incrementCounter:  function() {
    var that = this
    setInterval(function () {
      that.counter = that.counter + 1
    }, 1000)
  }
}
```

CLASSES

- The constructor method for creating and initializing an object created with a class
- Static methods are called without instantiating their class
- Instance methods are run on class

```
class Fruit {
  constructor(weight, price) {
    this.weight = weight;
    this.price = price;
  }
  calculatePrice() {
    return this.weight * this.pricePerUnit
  }
}
class Mango extends Fruit {
  constructor(weight, price) {
    super(weight, price)
    this.name = 'Mango'
  }
}
```

TEMPLATE STRINGS

Multiline strings

ES2015

```
`line1 text
 line2 text`
```

ES5

```
'line1 text' + '\n' + 'line2 text'
```

TEMPLATE STRINGS

Interpolate variables

ES2015

```
const msg = `Hello ${firstName} ${lastName}`
```

ES5

```
var msg = 'Hello ' + firstName + ' ' + lastName
```

DESTRUCTURING

Array destructuring

```
const [a, ,b] = [1,2,3];
a === 1;
b === 3;
```

Object destructuring

```
const values = {a: 1, b: 2, c: 3}
const {a, b} = values
a === 1;
b === 3;
```

DEFAULT FUNCTION PARAMETERS

```
const f(x = 2) {
  return x
}
f() === 2
f(5) === 5
```

REST OPERATOR

```
const f(x, ...y) {
  // x === 1
  // y === [2, 3, 4]
}

f(1, 2, 3, 4)
```

SPREAD OPERATOR

```
function f(x, y, z) {
  // x === 1
  // y === 2
  // z === 3
}

const data = [1, 2, 3]
f(...data)
```

LET & CONST

- let is block scoped
- use let instead of var
- Const is for Single Assignment

```
const x = 1
x = 2 // Throws error
```

ES2015 MODULES

- Modules help us organize the code in separate files
- Avoid global namespace collision
- Easy to share code across projects
- Simplifies using opensource code in our project

ES2015 MODULES

- Exporting single property

```
export default function calculator() {

}
```

- Exporting multiple properties

```
export function add(a, b) {
  return a + b
}
```

```
export function multiply(a, b) {
  return a * b
}
```

ES2015 MODULES

- Importing default property

```
import calculator from './calculator'
```

- Importing multiple properties

```
import {sum, multiply} from './calculator'
```

REACT BOOTSTRAP

TODO APP DEMO

REDUX

- Redux is a library for state management
- Very useful for efficient state management of large apps

REDUX

- The state of your whole application is stored in an object tree within a single store
- State is read-only
- Changes are made with pure functions

PURE FUNCTIONS

- Should not have side effects
- Should not mutate input data
- Should compute results based on inputs only

REDUX REDUCERS

- Pure functions

```
function reducer(prevState, action) {
  // Modify based on action
  return newState
}
```

REDUX

REDUX STORE

```
import { createStore } from 'redux'
import rootReducer from './reducers'
let store = createStore(rootReducer)
```

```
store.getState()  // To access state
store.dispatch(action) // To update state
store.subscribe(listener) // Listen to state changes
```

ACTIONS

- Actions are plain javascript objects
- They contain information that is sent from your application to your store
- Actions must have a type property that indicates the type of action being performed
- We send action to the store using store.dispatch()

```
{
  type: ADD_TODO,
  text: 'Build my first Redux app'
}
```

ACTION CREATORS

- functions that create actions

```
function addTodo(text) {
  return {
    type: ADD_TODO,
    text
  }
}
```

MIDDLEWARE

- It provides a third-party extension point between dispatching an action, and the moment it reaches the reducer
- Usecases
  - Logging
  - Crash Reporting
  - Asyncronous API calls

LOGGER MIDDLEWARE

```
const logger = store => next => action => {
  console.log('dispatching', action)
  let result = next(action)
  console.log('next state', store.getState())
  return result
}
```

CRASH REPORTING MIDDLEWARE

```
const crashReporter = store => next => action => {
  try {
    return next(action)
  } catch (err) {
    console.error('Caught an exception!', err)
    Raven.captureException(err, {
      extra: {
        action,
        state: store.getState()
      }
    })
    throw err
  }
}
```

ADDING MIDDLEWARES TO REDUX

```
import { createStore, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';
import rootReducer from './reducers/index';

const store = createStore(
  rootReducer,
  applyMiddleware(thunk)
);
```

REDUX THUNK

- Useful for handling Async actions

```
export function fetchPosts(subreddit) {
  return function (dispatch) {
    dispatch(requestPosts(subreddit))

    return fetch(`http://www.reddit.com/r/${subreddit}.json`)
      .then(response => response.json())
      .then(json =>
        dispatch(receivePosts(subreddit, json))
      )
    }
}
```

FETCH

- Fetch is a standards api for making AJAX calls
- Need to use a polyfil until window.fetch is available on all browsers
- To use with webpack

```
entry: ['whatwg-fetch', ...]
```

- Need es6-promise pollyfill as well for older browsers

FETCH

- fetching JSON data with fetch

```
fetch('/users')
  .then(function(response) {
    return response.json()
  }).then(function(json) {
    console.log('parsed json', json)
  }).catch(function(ex) {
    console.log('parsing failed', ex)
  })
```

FETCH

- Response metadata

```
fetch('/users.json').then(function(response) {
  console.log(response.headers.get('Content-Type'))
  console.log(response.headers.get('Date'))
  console.log(response.status)
  console.log(response.statusText)
})
```

FETCH

- Posting data to server

```
fetch('/users', {
  method: 'POST',
  headers: {
    'Accept': 'application/json',
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    name: 'Hubot',
    login: 'hubot',
  })
})
```

TODO APP WITH REDUX

REACT ROUTER

- Route

```
const routes = (
  <Route component={App}>
    <IndexRoute component={Home}/>
    <Route path="groups" component={Groups} />
    <Route path="users" component={Users} />
  </Route>
)

// for /       <App><Home /></App>
// for /groups <App><Groups /></App>
// for /users  <App><Users /></App>
```

REACT ROUTER

- Router
  - Router is primary component of React Router.
  - It keeps your UI and the URL in sync

```
import { browserHistory } from 'react-router'
ReactDOM.render(<Router history={browserHistory} routes={routes}/>, el)
```

REACT ROUTER

- Link
  - Usefull for navigation across app
  - it will render a fully anchor tag with the proper href

```
<Link to={`/users/${user.id}`}>{user.name}</Link>
```

REACT ROUTER

- Nesting Routes

```
const routes = (
  <Route component={App}>
    <IndexRoute component={Home} />
    <Route path="blog" component={Blog} />
      <IndexRoute component={BlogHome} />
      <Route path="posts" component={Posts} />
    <Route>
  </Route>
)
```

REACT ROUTER

- dynamic segments to capture ids from URL

```
<Route path="/" component={App}>
  <Route path="user/:userID" component={User}>
    <Route path="tasks/:taskID" component={Task} />
    <Redirect from="todos/:taskID" to="tasks/:taskID" />
  </Route>
</Route>
```