Universal Web

With React and Express

Azat Mardan

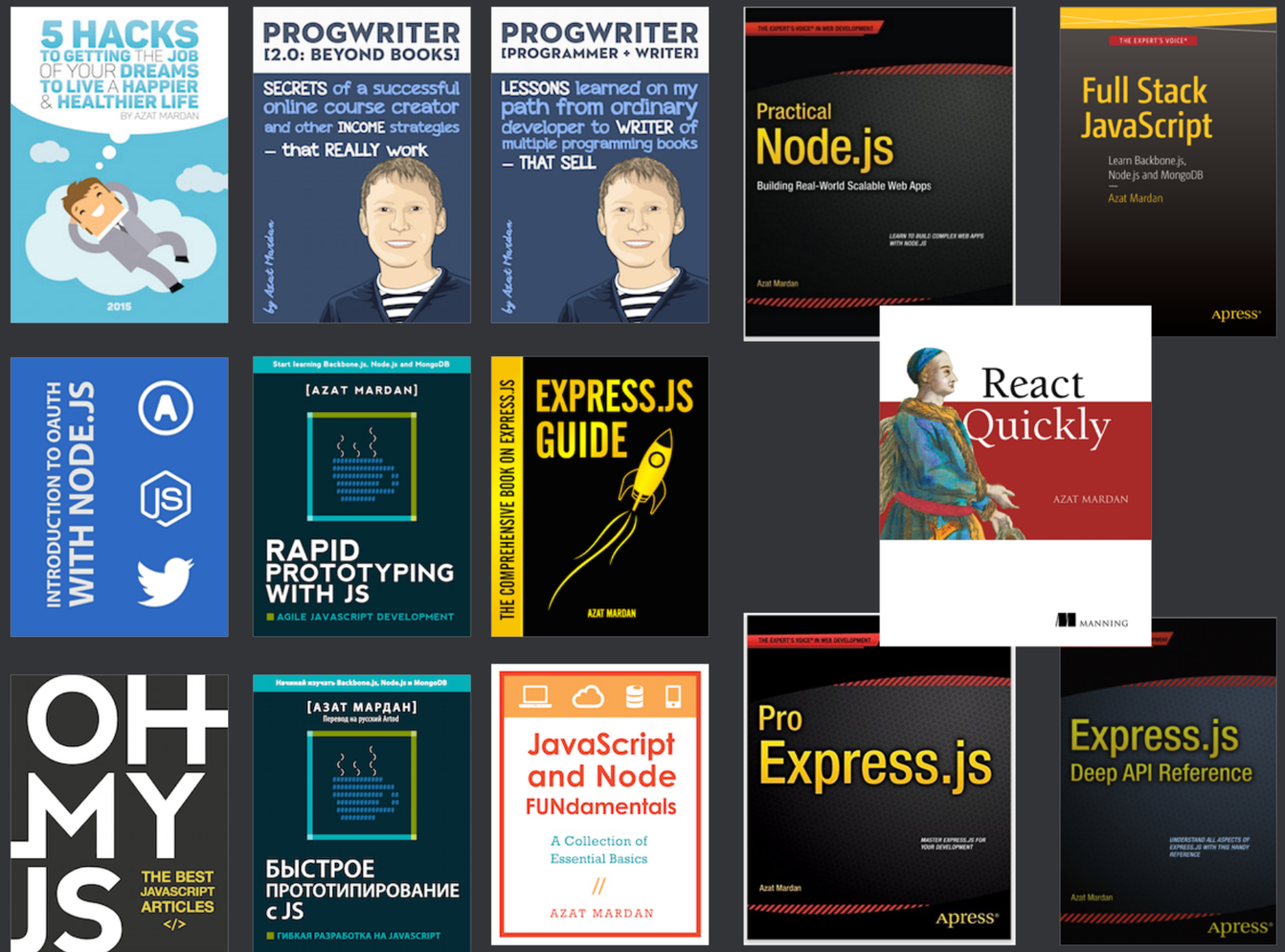
About Presenter

Azat Mardan

Twitter: @azat_co
Blog: webapplog.com

About Presenter

- Work: Technology Fellow at Capital One
- Experience: FDIC, NIH, DocuSign, HackReactor and Storify
- Books: Practical Node.js, Pro Express.js, Express.js API and 8 others
- Teach: NodeProgram.com


5 HACKS TO GETTING THE JOB OF YOUR DREAMS TO LIVE A HAPPIER & HEALTHIER LIFE
BY AZAT MARDAN
2015
PROGWRITER [2.0: BEYOND BOOKS]
SECRETS of a successful online course creator and other INCOME strategies – that REALLY work
by Azat Mardan
PROGWRITER [PROGRAMMER + WRITER]
LESSONS learned on my path from ordinary developer to WRITER of multiple programming books – THAT SELL
by Azat Mardan
Practical Node.js
Building Real-World Scalable Web Apps
Azat Mardan
Full Stack JavaScript
Learn Backbone.js, Node.js and MongoDB
Azat Mardan
Apress
INTRODUCTION TO OAUTH WITH NODE.JS
Start learning Backbone.js, Node.js and MongoDB
[AZAT MARDAN]
RAPID PROTOTYPING WITH JS
AGILE JAVASCRIPT DEVELOPMENT
THE COMPREHENSIVE BOOK ON EXPRESS.JS
EXPRESS.JS GUIDE
AZAT MARDAN
React Quickly
AZAT MARDAN
MANNING
OH MY JS
THE BEST JAVASCRIPT ARTICLES
</>
Начинай изучать Backbone.js, Node.js и MongoDB
[АЗАТ МАРДАН]
Перевод на русский Artsd
БЫСТРОЕ ПРОТОТИПИРОВАНИЕ С JS
ГИБКАЯ РАЗРАБОТКА НА JAVASCRIPT
JavaScript and Node FUNdamentals
A Collection of Essential Basics
//
AZAT MARDAN
Pro Express.js
Azat Mardan
Apress
Express.js Deep API Reference
Azat Mardan
Apress

Why Care?

- SEO
- Loading...
- Code re-use
- Performance

Many Names

- Full stack
- Isomorphic
- Universal

Problem

Code re-use, i.e., templates between server and browser

Storify Front-End App

https://www.npmjs.com/package/jade-browser

Problem

Make SPA load data faster

`SPA<->node+cache<->API`

DocuSign Web App

- Backbone
- Node
- Express
- CoffeeScript
- Grunt
- Browserify

Solution

Prefetch data from API, save in cache for SPA.

Backbone Models

```
# home page - dashboard
mapping['/home'] =
  name: 'dashboard'
  bodyClass: 'dashboard'
  pageTitle: 'Home'
  helpLink: 'http://www.docusign.com/DocuSignHelp/DocuSignHelp.htm'

  clientAssets:
    js: [ '/js/bundle-home.js', '/js/bundle-send.js', '/js/bundle-documents-page.js'
          '/js/tutorials.js'
        ]

  layoutView: 'HomeLayoutView'

  data:
    signatures:
      type: 'SignatureList'
      views: [ ]
      order: 1

    profile:
      type: 'Profile'
      views: [ 'ProfileView' ]
      reuse: false
      attachRefs: [ 'signatures' ]
      order: 2
```

Universal is not exclusive to React

But React makes it easier

Definition

React is a UI library

React Way

React UIs are simple functions where input generates HTML elements in a predictable way (input->output)

React component just work on server

One-way binding: views are simple functions of props

Demo

https://github.com/azat-co/react/tree/master/ch8/board-react2

React for server-side

```
let React = require('react')
let ReactDOMServer = require('react-dom/server')

class MyComponent extends React.Component {
  render() {
    return <div>Hello World</div>
  }
}

ReactDOMServer.renderToString(<MyComponent />)
```

TK check

react-express-view

```
let app = express()

app.set('views', __dirname + '/views')
app.set('view engine', 'jsx')
app.engine('jsx', require('express-react-views').createEngine())
```

Problem

Flash of content

Solution

```
<script>
  window.__data=${JSON.stringify(data)}
</script>
```

Browser React

```
ReactDOM.render(<MyComponent messages={window.__data}/>, document.getElementById('content'))
```

Browser React Example

```
// app.js - loaded only on  browser
React = require('react')
ReactDOM = require('react-dom')

{Header, Footer, MessageBoard} = require('./components.js')

ReactDOM.render(<Header />, document.getElementById('header'))
ReactDOM.render(<Footer />, document.getElementById('footer'))
ReactDOM.render(<MessageBoard messages={messages}/>, document.getElementById('message-board'))
```

Express Route Example

```
app.get('/', function(req, res, next){
    req.messages.find({}, {sort: {_id: -1}}).toArray(function(err, docs){
      if (err) return next(err)
      res.render('index', {
        header: ReactDOMServer.renderToString(Header()),
        footer: ReactDOMServer.renderToString(Footer()),
        messageBoard: ReactDOMServer.renderToString(MessageBoard({messages: docs})),
        props: `<script type="text/javascript">var messages=${JSON.stringify(docs)}</script>`
      })
    })
  })
```

```
<div>{{{props}}}</div>
<div class="row-fluid">
  <div class="span12">
    <div id="content">
      <div class="row-fluid"  id="message-board" />{{{messageBoard}}}</div>
  </div>
</div>
```

Problem

Routes, i.e., **`/users`** not **`/#users`**

Solution

Serve **`index.html`** for all routes

```
const express = require('express')
const path = require('path')
const port = process.env.PORT || 8080
const app = express()

app.use(express.static(__dirname + '/public'))

app.get('*', function (request, response){
  response.sendFile(path.resolve(__dirname, 'public', 'index.html'))
})

app.listen(port)
console.log("server started on port " + port)
```

Alternatively: React Routers

```
import { renderToString } from 'react-dom/server'
import { match, RouterContext } from 'react-router'
import routes from './routes'

serve((req, res) => {
  match({ routes, location: req.url }, (error, redirectLocation, renderProps) => {
    if (error) {
      res.status(500).send(error.message)
    } else if (redirectLocation) {
      res.redirect(302, redirectLocation.pathname + redirectLocation.search)
    } else if (renderProps) {
      res.status(200).send(renderToString(<RouterContext {...renderProps} />))
    } else {
      res.status(404).send('Not found')
    }
  })
})
```

On the client, instead of rendering

```
render(<Router history={history} routes={routes} />, mountNode)
```

You need to do

```
import { match, Router } from 'react-router'
import routes from './routes'

match({ history, routes }, (error, redirectLocation, renderProps) => {
  render(<Router {...renderProps} />, mountNode)
})
```

What about data?

Server side Redux

```
import { createStore } from 'redux'
import { Provider } from 'react-redux'
import counterApp from './reducers'
import App from './containers/App'

const app = Express()

app.use((req, res) =>{
  const store = createStore(counterApp)
  const html = renderToString(
    <Provider store={store}>
      <App />
    </Provider>
  )
  res.send(renderHtml(html, store.getState()))
})

app.listen(3000)
```

Pushing to Client

```
let renderHtml = (html, initialState) => {
  return `<div id="root">${html}</div>
    <script>
      window.__INITIAL_STATE__ = ${JSON.stringify(initialState)}
    </script>
    <script src="/static/bundle.js"></script>`
}
```

Redux Client

```
const initialState = window.__INITIAL_STATE__
const store = createStore(counterApp, initialState)

render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
)
```

Pass Params to Redux Store

```
const counter = parseInt(req.params.counter, 10) || 0
let initialState = { counter }
const store = createStore(counterApp, initialState)
```

Summary

- Render Components on the server
- Make sure you get the **same** data to the browser
- Use nice URLs with React Router
- Use Redux on the server

The End

Example: https://github.com/azat-co/react-quickly/tree/master/projects/autocomplete

Slides: https://github.com/azat-co/universal-web

React book: http://reactquickly.co

Workshop

https://github.com/azat-co/react-quickly/tree/master/ch16/message-board