UNIVERSAL REACT APPS USING NEXT.JS

Sia Karamalegos

@thegreengreek

WHY DO ELEVATORS HAVE MIRRORS?

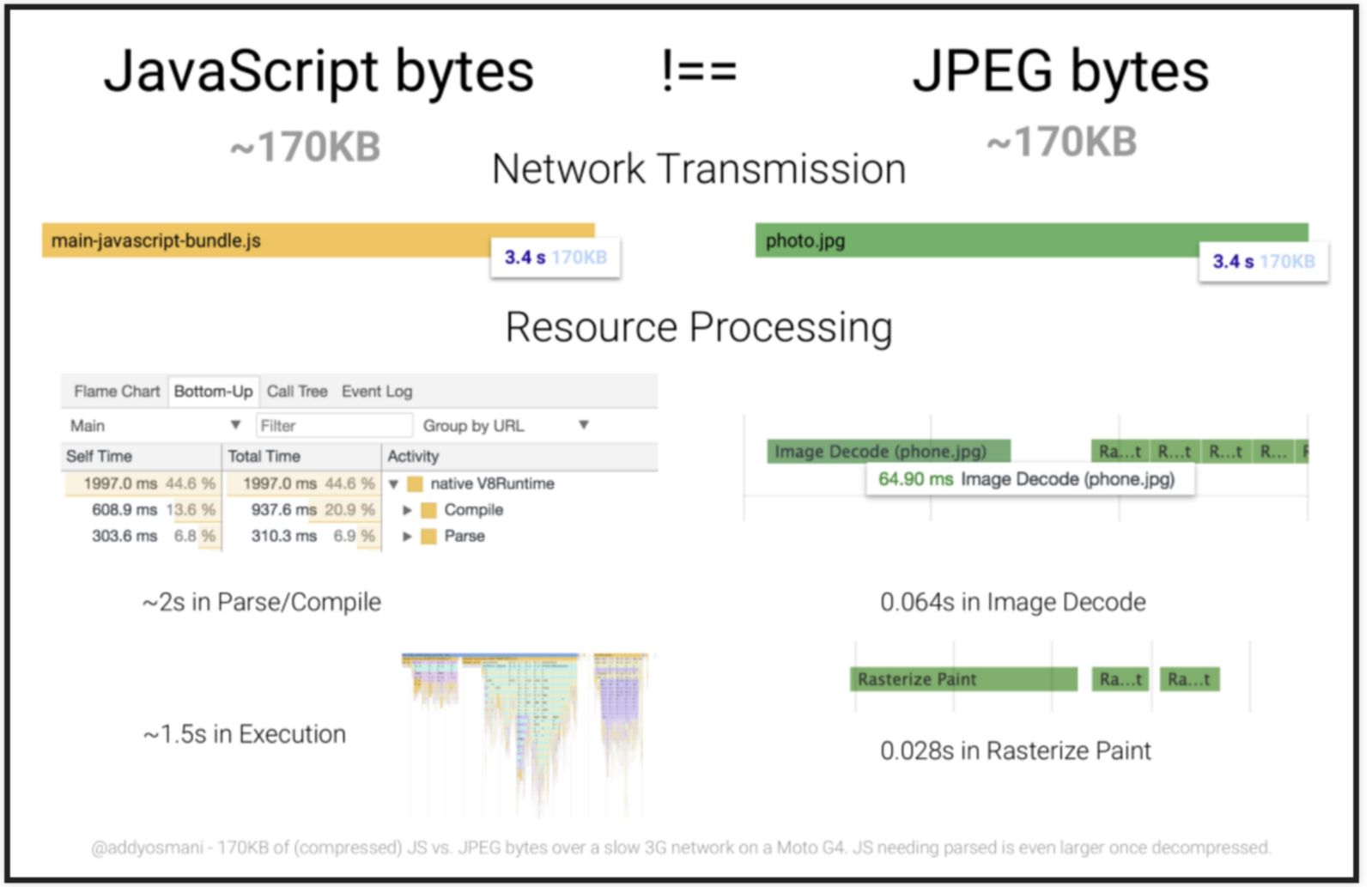
Why am I talking about Universal apps and Next.js?

1. Performance matters
2. Shipping code matters

53% of mobile site visits are abandoned if pages take longer than 3 seconds to load.

--*DoubleClick by Google, 2016*

JavaScript is your most expensive asset



And users have bad phones on bad networks

- 2-5x difference in fastest vs slowest phones
- 75% of worldwide mobile connections on 2G or 3G

https://infrequently.org/2017/10/can-you-afford-it-real-world-web-performance-budgets/

Yet we are pushing more and more bytes to the client!

Take back your rendering!

- For initial load, render and fetch data on the server
- For future navigation paths, "remove" loading and rendering time with prefetch

__Universal__ JavaScript means JavaScript that can run in both the client and the server.

NEXT.JS

Next.js is a minimalistic framework for universal, server-rendered (or statically pre-rendered) React applications.

It creates a React app using just one command:

```
$ npm install --save next react react-dom
```


Next.js Benefits

Faster page loads

- Server-rendered by default for initial load
- Can enable prefetching future routes
- Automatic code splitting

Next.js Benefits

No build configuration

- Simple page-based client-side routing
- Babel, Webpack, and hot module replacement

Next.js Benefits

Customizable

- Use your own Babel and Webpack configurations
- Customize the server API with Express or any other Node.js HTTP server

HOW DOES IT WORK?

No build config needed!

1. Create project and install:

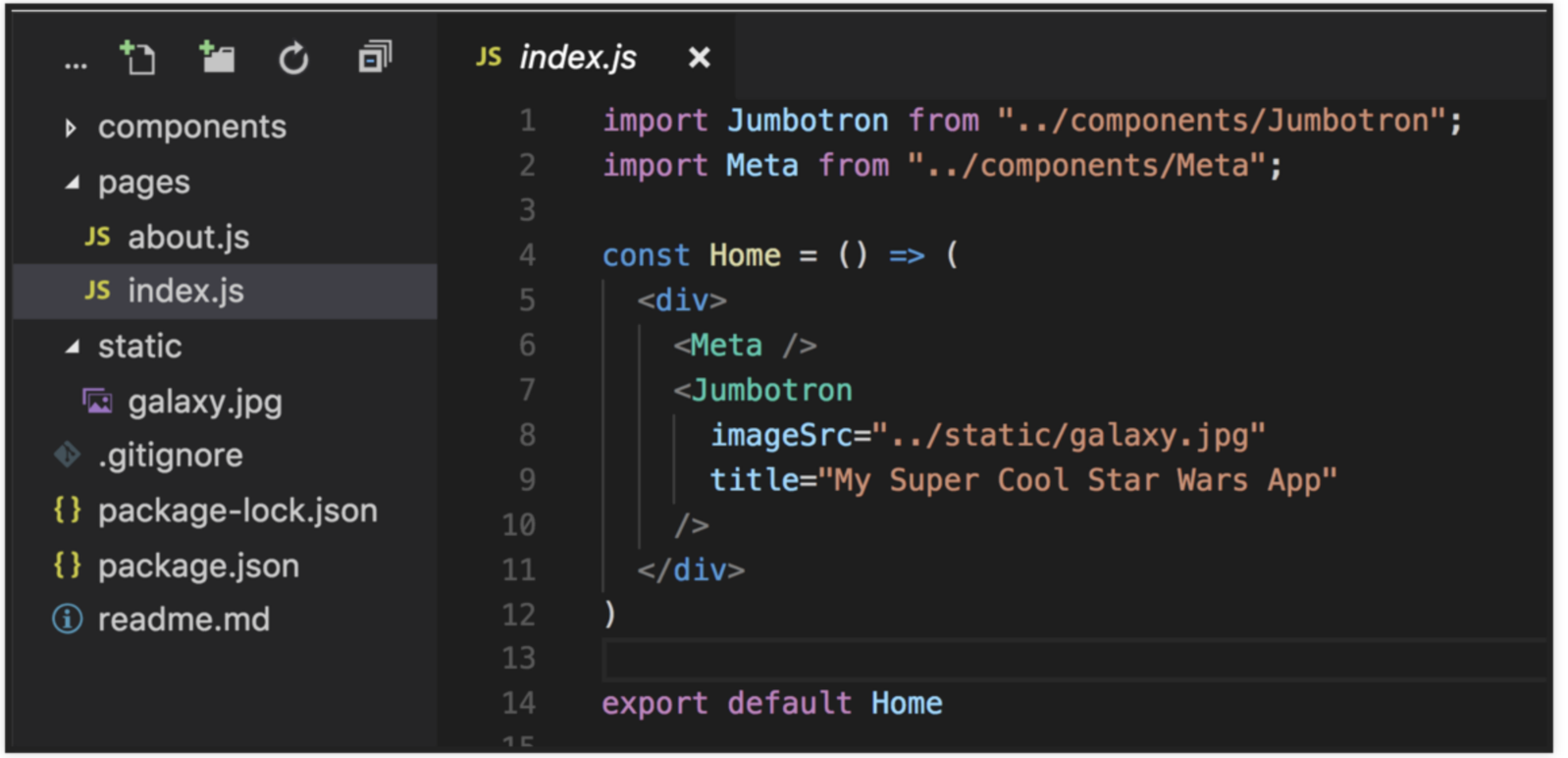
```
$ npm install --save next react react-dom
$ mkdir pages
```

2. Add scripts to `package.json`:

```
{
 "scripts": {
   "dev": "next",
   "build": "next build",
   "start": "next start"
 }
}
```

3. `npm run dev` to see your formatted 404 page.

The API is mostly the folder system



Link

To create links to pages, instead of using an anchor tag, use the API's `<Link />` component.

- Client-side navigation
- Client-side rendering of the target component

```
const MyLink = () => (
  <Link href="/about">
    <a style={{margin: '10px'}}>About</a>
  </Link>
)
```


Hydrate the client with future navigation paths!!

Prefetch

- `<Link prefetch />` pre-fetches the component's JSON representation in the background, via a ServiceWorker.
- `prefetch` does not trigger unnecessary data fetching due to a special `getInitialProps` method (*to be continued...*).

Note that `prefetch` only works in production.

What we get for free

- Automatic transpilation and bundling (with webpack and babel)
- Hot code reloading
- Server rendering and indexing of `./pages`
- Static file serving. `./static/` is mapped to `/static/`
- Built-in CSS support with `styled-jsx` (if you want to use it)
- Back button works!

LOADING DATA

Load data on page load with `getInitialProps`

- Asynchronous, static method
- Fetches anything that resolves to a JS plain object
- Populates the component's props with the returned object
- On initial load, executed on the server
- On navigation (not prefetch), executed in the client

`getInitialProps` API

- Receives a context object which contains `pathname`, `query`, `req`, `res`, `err`, etc.
- Can **only** be used on page components, not child components

```
MyPage.getInitialProps = async ({ pathname, query, req, res, err }) => {
  // do something like this to return a plain object
  const res = await fetch(myApiUrl)
  const json = await res.json()
  return { myThings: json.things }
}
```


DYNAMIC ROUTES

Dynamic routes can be created using query strings

- Set up Link path with the query string:

```
(title) => <Link href={`/post?title=${title}`}><a>{title}</a></Link>
```

- Create the target page component, grabbing the query from `props.url`:

```
export default (props) => (
  <div>
    <h1>{props.url.query.title}</h1>
    <p>This is the blog post content.</p>
  </div>
)
```


Every page receives the `url` prop

The `url` API includes these properties:

- `pathname` - current path excluding the query string (string)
- `query` - object with the parsed query string. Defaults to {}
- `asPath` - full actual path including the query (string)
- `push(url, as=url)` - performs a pushState call with the given url
- `replace(url, as=url)` - performs a replaceState call with the given url

The page-based API has no shared global state

To share state, you have several options:

- Wrapping all components with shared state in a single page/route then create a custom routing API
- Overriding the `<App>` API with a custom one that keeps state across navigating pages (API)
- Using it with Redux (example)
- And more... check out all of the examples for more possibilities, like Apollo + Redux

Want pretty URLs?

Use route masking...

```
// `as` attribute masks route
(title) => (
  <Link href={`/post?title=${title}`} as={`/post/${title}`}>
    <a>{title}</a>
  </Link>
)
```


Custom Server API

```
const express = require('express')
const next = require('next')

const dev = process.env.NODE_ENV !== 'production'
const app = next({ dev })
const handle = app.getRequestHandler()

app.prepare()
  .then(() => {
    const server = express()

    server.get('/films/:id', (req, res) => {
      const pathname = '/films'
      const queryParams = { id: req.params.id }
      app.render(req, res, pathname, queryParams)
    })
    // ...
```


ROUTER EVENTS API

You can hook into these events in the Router's lifecycle:

- `onRouteChangeStart(url)` - a route starts to change
- `onRouteChangeComplete(url)` - a route changed completely
- `onRouteChangeError(err, url)` - an error occurs when changing routes
- `onBeforeHistoryChange(url)` - just before changing the browser's history

Usage:

```
Router.onRouteChangeStart = url => { console.log('Routing to: ', url)}
```


STATIC EXPORTS

Don't want to use a server?

Next.js can also build static web apps that you can then host on Github pages, AWS S3, etc.

Static exports in Next.js:

- Generate a set of pre-rendered HTML pages
- Support dynamic urls, prefetching, preloading and dynamic imports
- Do **not** support dynamic (at runtime) pages (at build time is okay)

HOUSTON'S BAGGAGE CLAIM COMPLAINTS

RESOURCES

Continue your learning with these resources and tutorials:

- About Next.js https://zeit.co/blog/next
- Next.js GitHub repo
- Official Tutorial by Zeit
- New version 5.0 details on the Zeit blog
- 7 Principles of Rich Web Applications by Guillermo Rauch, the inspiration for Next.js

THANKS!

Slides, resources, and more: bit.ly/siaspeaks