SERVER SIDE RENDERING

Frank Walsh

AGENDA

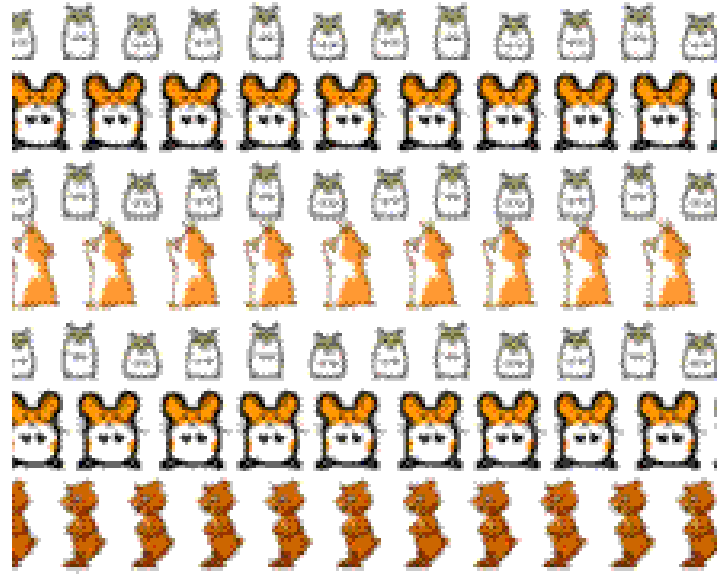
- Quick history
- Isomophism
- Server-Side vs. Client Side Rendering
- React and Server-Side Rendering

WEB APPS HISTORY STATIC

- Web App definition:
  - Highly interactive and dynamic
  - Adaptable and transient
- Web Site
  - Informational and static

WEB APP HISTORY DYNAMIC

- Static (to 1995)
  - HTTP GET is King
  - Gopher
- DHTML, Flash and Javascript (1995-)
  - Some functionality
  - No personalisation
- Simple and easy to create
- Classic examples opposite…



WEB APP HISTORY
SERVER SIDE SCRIPTING

- Need to create HTML content Dynamically
  - Common Gateway Interface
  - Perl
- Generally needed to invoke new process to serve request
- Not designed for web.

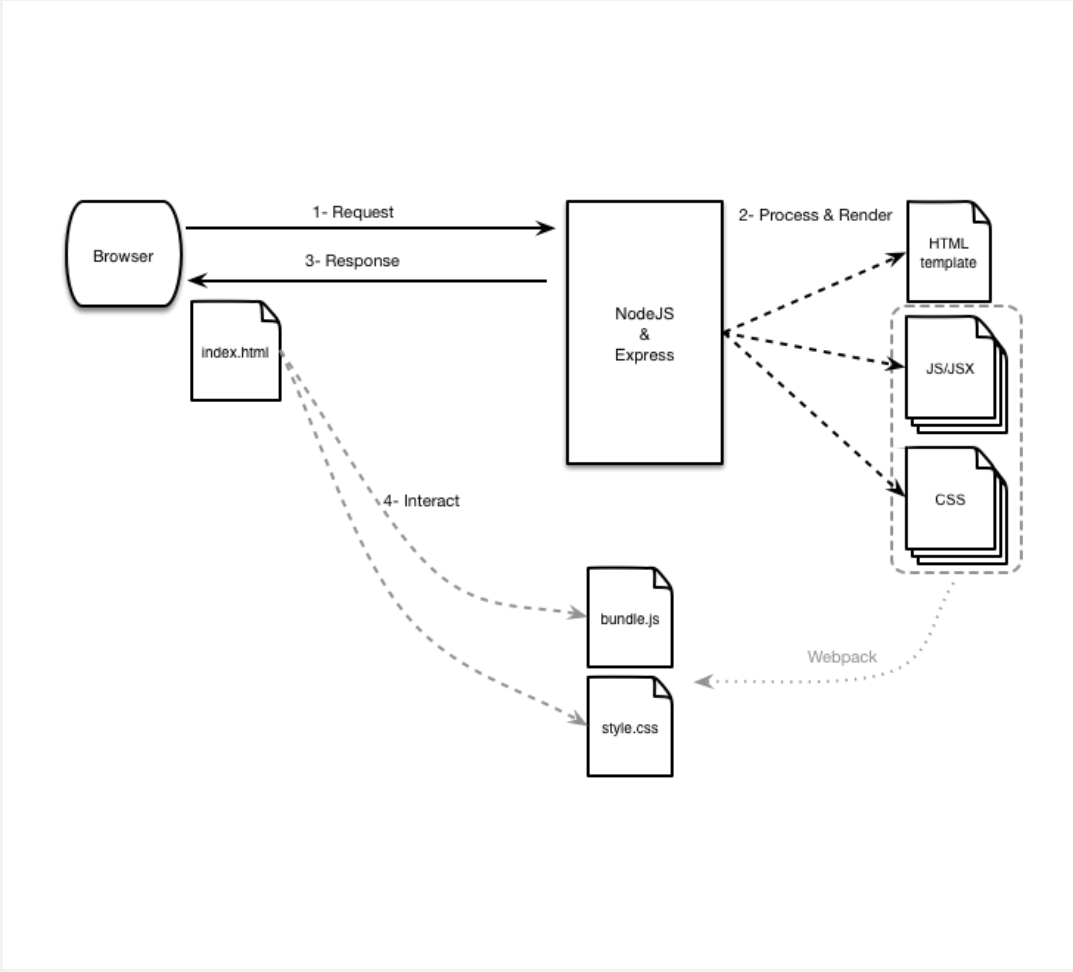
WEB APP HISTORY SERVER PAGES

- Server Page characteristics
  - write server-side code directly in the HTML
  - better for developers
- PHP, ASP, JSP….
- Server-side modules allowed you to run code inside containers
- JEE

```
<!DOCTYPE html>
  <html>
  <head>
  <title>PHP - Hello, world!</title>
  </head>
  <body>
  <?php echo '<h1>Hello, world!</h1>'; ?>
  </body>
  </html>
```

UNIVERSAL WEB APPLICATION

- Some probs with server pages:
  - spreading presentation logic between client and server
  - Complicated skill sets evolved around certain stacks (JEE, Spring)
- Universal web applications (isomorphism)
  - Ability to run same code on both client and server
  - What we're kind of approaching with react.



UNIV. WEB APPS CONSIDERATIONS

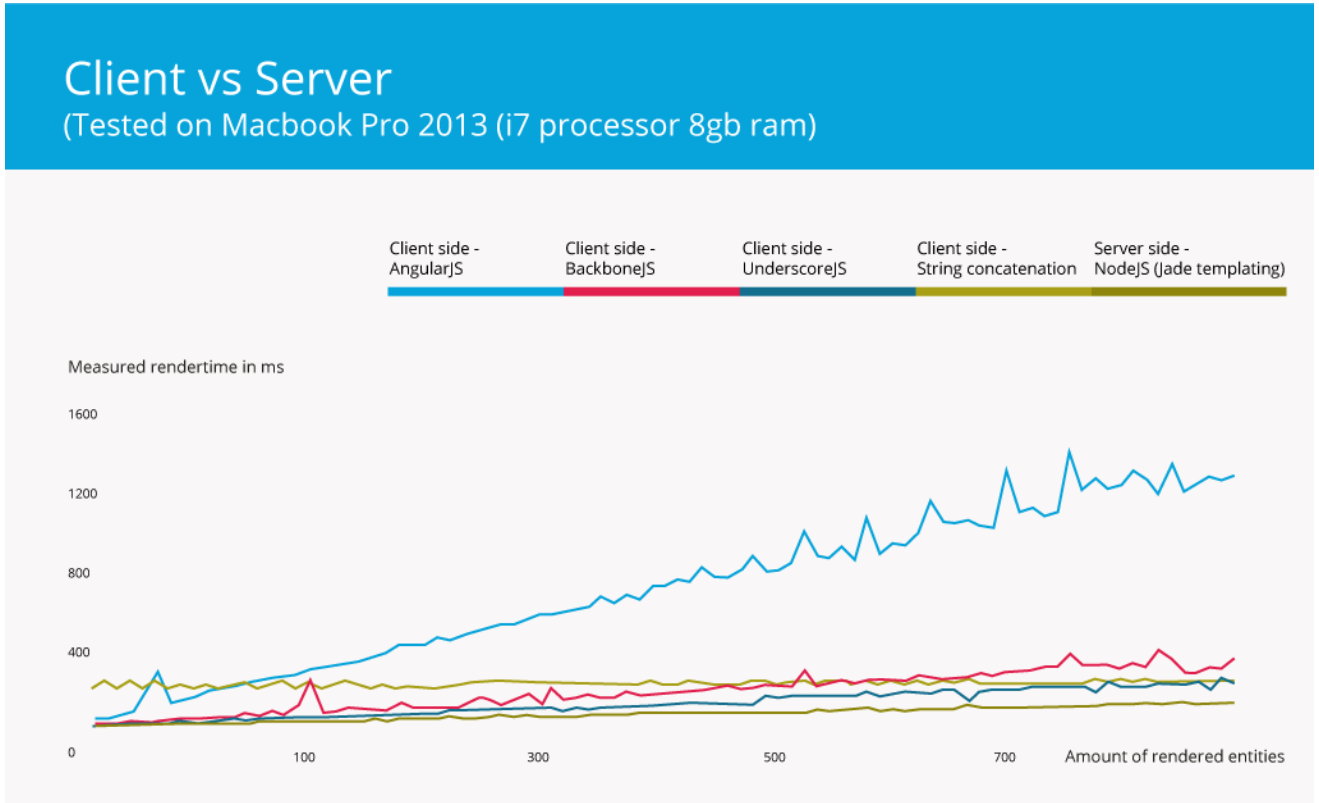
- Performance
  - Initial load time can be an issue
  - Mobile Devices
  - Increasing RAM/CPU power but also increasingly complex apps
- Search Engine Optimisation
  - Apps maybe should be written for machines as well as humans
  - Discoverability and Rank depend on content
  - Many SEO machines apparently aren't willing to run JavaScript
- Maintenance
  - UWA promotes1 solution for all.
  - Do not need diverse set of skills to maintain different versions.

SERVER SIDE RENDERING
WHY

- SPA applications (e.g. React apps) are rendering data from the server using JavaScript.
  - No Javascript – no data
  - Try it by switching off javascript in your browser.
- Most search engines will see this when they request the app
  - Can simulate this by doing a curl on your app you pushed to Bluemix last week
- Would be good if machined(i.e. search engines) could "see" what we see when we request the app.
  - Need to pre-render all components with data on server on initial request.

SERVER SIDE VS CLIENT SIDE TEST

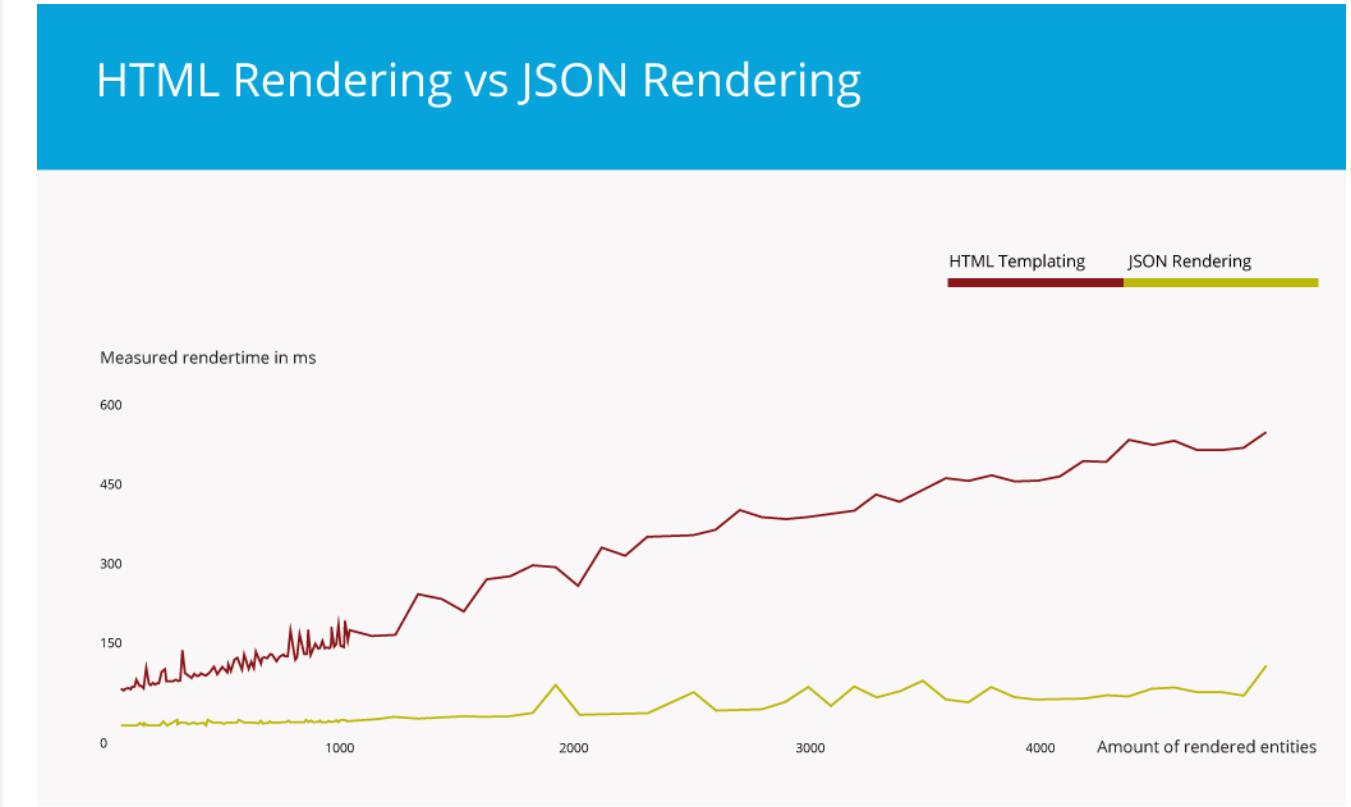
- Server side rendering needed almost a constant amount of time for rendering
- Client side shows almost linear growth.*



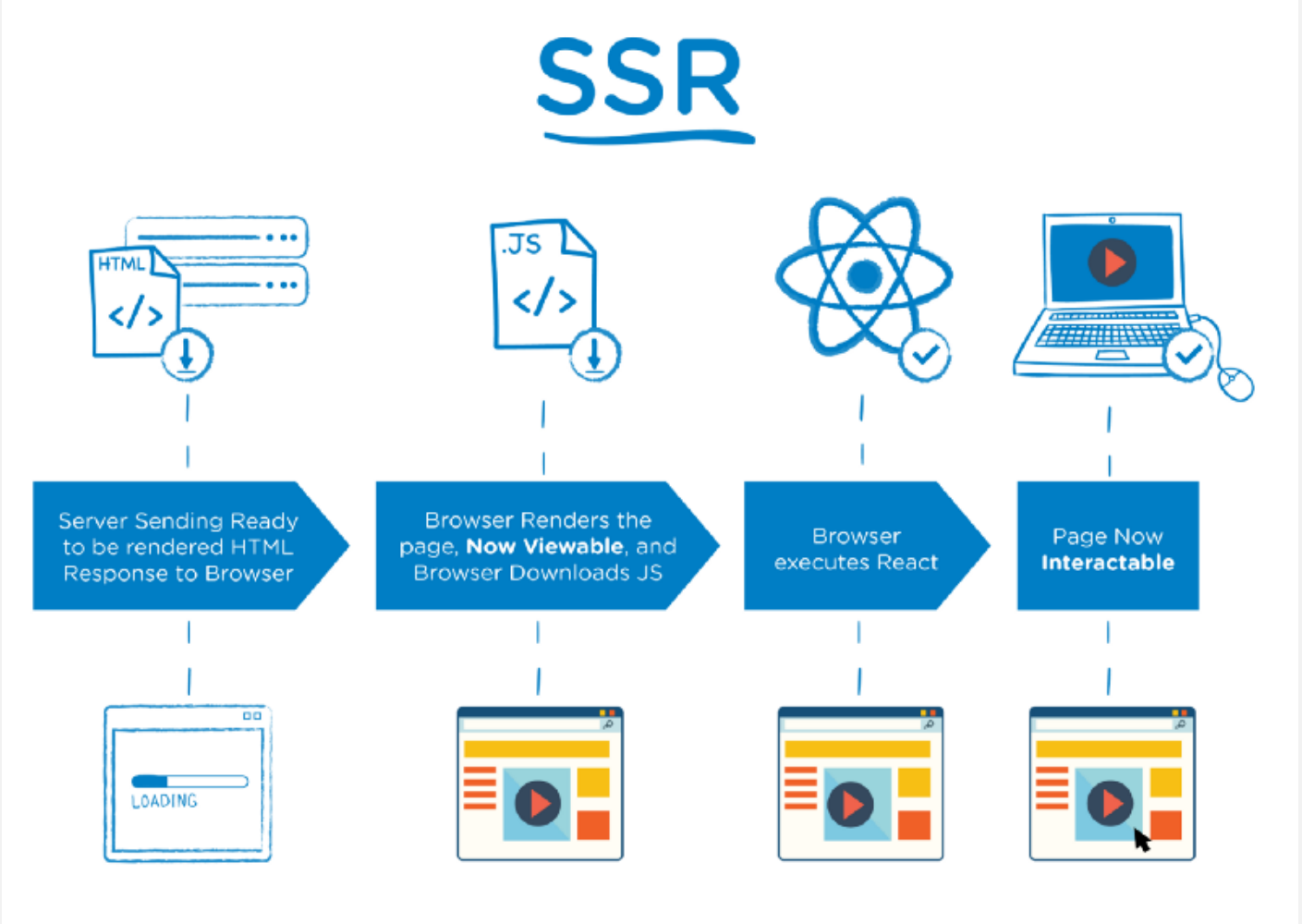
source: https://blog.mwaysolutions.com/2013/11/08/client-vs-serverside-rendering-the-big-battle-2/

JSON RENDERING VS HTML RENDERING

- An advantage of client side rendering is that the server can generate JSON faster than a complete HTML template.
- JSON is also smaller than HTML
- Can reduce server and traffic costs by sourcing out the rendering on the client.
- For Smartphone access, less traffic by serving a JSON object instead of a HTML document is an advantage

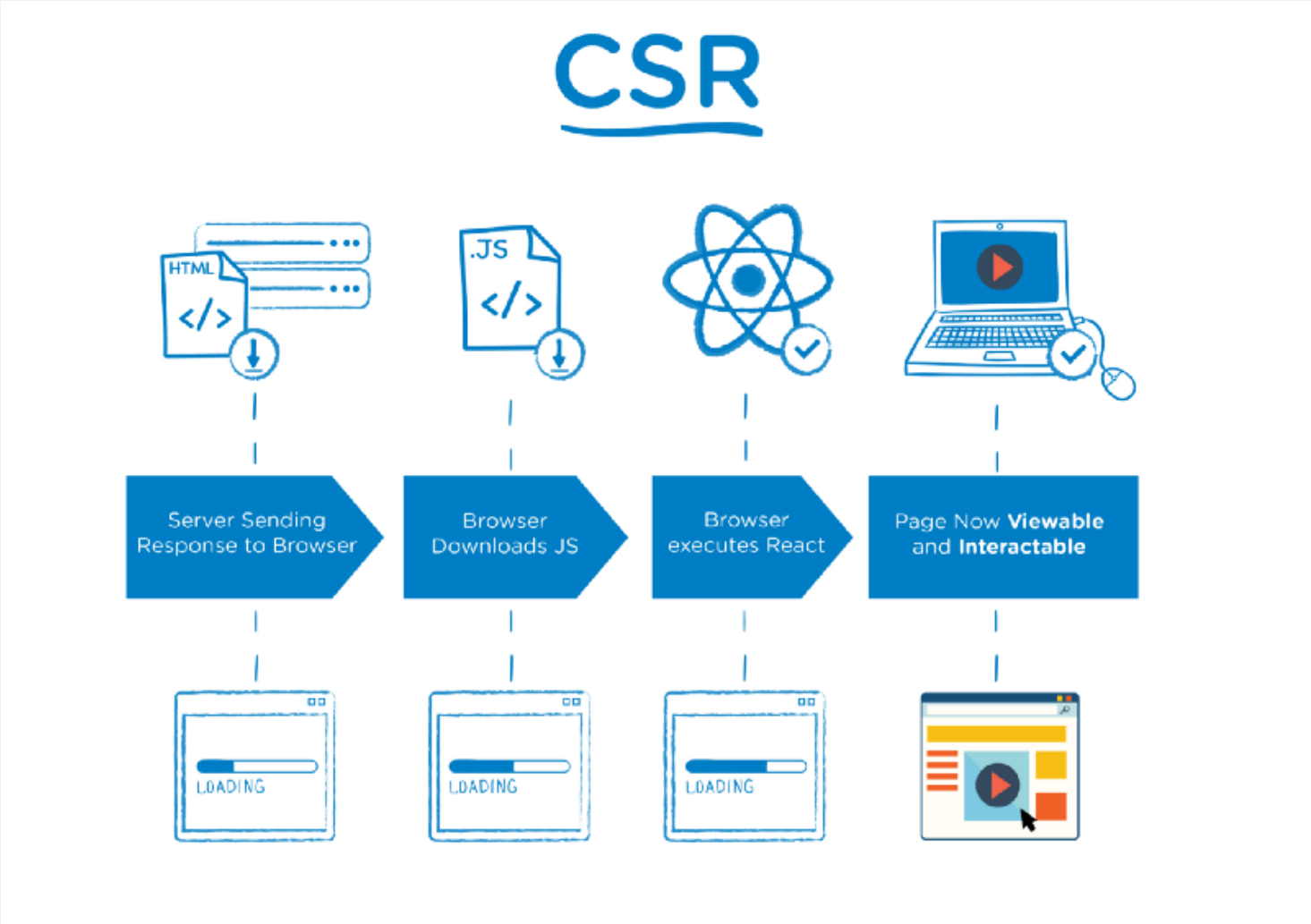


SERVER SIDE RENDERING



https://medium.com/walmartlabs/the-benefits-of-server-side-rendering-over-client-side-rendering-5d07ff2cefe8

CLIENT SIDE RENDERING



SERVER-SIDE RENDERING OPTIONS FOR REACT

- React-redux
  - https://github.com/reactjs/redux/blob/master/docs/recipes/ServerRendering.md
- Flux
- reactDomServer.
  - Comes with React
- React-Server

REACTDOMSERVER

- Allows you to render your components on the server.
  - renderToString() renders a React element to its initial HTML.
  - renderToStaticMarkup() doesn't create extra DOM attributes, can save lots of bytes.

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

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

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

SERVER-SIDE RENDERING WITH EXPRESS

- For server-side rendering, you need to add the relevant routes in to the express server.
- Requires install babel-preset-react
- Also works well with Templating Engines

```
server.get('/', (req, res) => {
    const html = ReactDOMServer.renderToHTML(<App posts={data}/>);
    res.send(html);
});
```

TEMPLATING USING EJS

- Lets you generate HTML markup with plain JavaScript
- EJS files are regular HTML files, but we can embed JavaScript using template tags
- Opposite ejs returns a random number.

```
...
<body>
  <%= math.random() %>
</body>
...
```

EJS AND EXPRESS

- Requires EJS dependency
- Configure your express app to use EJS as the view engine
- Use the response object to render EJS views
- Can pass variables from express into EJS as second argument of render() method

index.ejs

```
...
<body>
  <%= content %>
</body>
...
```

server.js

```
server.get('/',(req,res)=>{
  res.render('index',{
    content: 'hello EWD2017'
  });
})
```

EJS AND EXPRESS

- Can also pass in HTML
  - <%=…> will escape content
  - <%-…> will display HTML in template

index.ejs

```
...
<body>
  <%- content %>
</body>
...
```

server.js

```
server.get('/',(req,res)=>{
  res.render('index',{
    content: '<h1>hello EWD2017</h1>'
  });
})
```

EJS PARTIALS

- Can combine several partial EJS files
- For example header.ejs can be used in several templates
- Can also create a footer.ejs
- Use the <%- include(‘ ’) -%> statement to include partials

header.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-wid
  <meta http-equiv="X-UA-Compatible" content="ie=
  <title>Naming Contests</title>
  <link rel="stylesheet" href="https://maxcdn.boo
</head>
<body class="container">
```

index.ejs

```
<%- include('header') -%>
  <%- content %>
<%- include('footer') -%>
```

OTHER TEMPLATING FRAMEWORKS

- PUG
- Handlebars

GET INITIAL DATA ON THE SERVER

- For server side rendering, we need to request the initial data from API
- We've been doing this from the client so far (proxied through the web server)
  - Use axios again
  - PreRender all components with initial data

ServerRender.js

```
axios.get(getApiUrl(postId))
  .then(resp => {
    console.log(resp.data)
    const initialData = getInitialData(postId, resp.data);
    return {
      initialMarkup: ReactDOMServer.renderToString(
        <App initialData={initialData} />
      ),
      initialData
    };
  });
```

RETURN MARKUP AND INITIAL DATA

- Pass mark-up and initial data to the templating engine
- Can define/use initialData property on the window object in Javascript
- Pass in the initialMarkup into the root element.

index.ejs

```
<%- include('header') -%>
  <div id="root"><%- initialMarkup -%></div>

  <script type="text/javascript">
    window.initialData = <%- JSON.stringify(initialData) -%>;
  </script>
<%- include('footer') -%>
```

Server.js

```
server.get(['/'], (req, res) => {
  serverRender().then(({ initialMarkup, initialData }) => {
      res.render('index', {
        initialMarkup,
        initialData
      });
    })
    .catch(error => {
      console.error(error);
      res.status(404).send('Bad Request');
    });
});
```

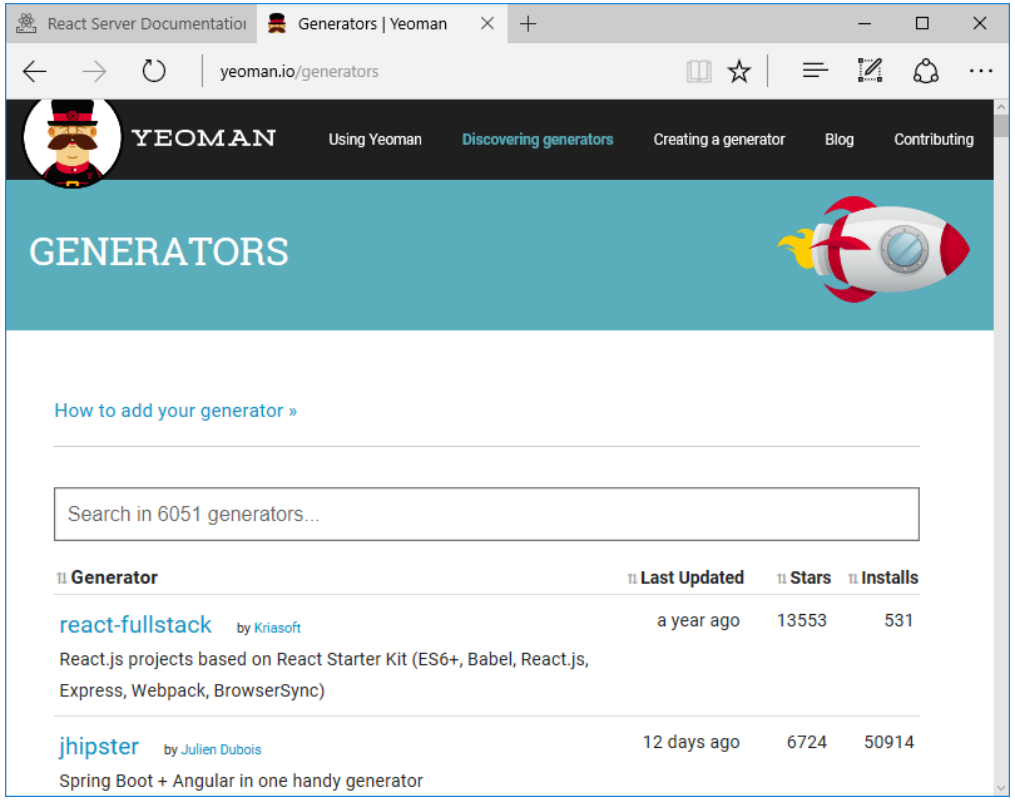
YEOMAN

- Node.js package for application scaffolding and workflows
- Key components:
  - Yo
  - Bower
  - Grunt
- Many build in generators for common types of applications
  - AngularJS, MEAN, Express…
  - Each generators install both needed Node.js packages and client-side JavaScript libraries
  - Generated Gruntfile.js for build/test/serve
  - Takes care of structuring application

SCAFFOLDING REACT-SERVER USING YEOMAN

YEOMAN GENERATORS

- Node Modules
- Typically available from NPM
- Can write your own generator



YEOMAN INSTALL

- To Install:

```
npm install -g yo
```

- To Install a generator:

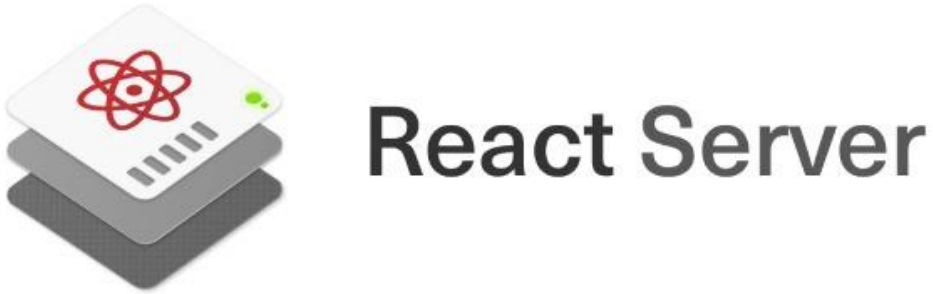
```
npm install -g generator-express
```

- Create a scaffold for express application(run this in the application top level directory)

```
yo express
```

REACT SERVER APP WITH YEOMAN

- “Blazing fast page load with seamless navigation”
- Framework designed to make universal (née isomorphic) React easier to write
- Concentrate on your React components.



REACT SERVER PAGE

- A hypertext document suitable for the world wide web and a web browser
- Roughly match one-to-one to urls for a web site.
- Pages have lifecycle methods that are called on them by react-server, which produce the html, either on the server or in the browser.
- Written as classes

```
export default class SimplePage {
    getElements () {
        return <h1>Hello react-server</h1>;
    }
}
```

PAGE LIFECYCLE METHODS

Once we reach the above, starts sending javascript.

EXAMPLE PAGE

Example in class…

```
1   import React from 'react';
2   import {ReactServerAgent, RootElement, TheFold, logging} from 'react-server';
3   import Header from '../components/header';
4   import Footer from '../components/footer';
5
6   import App from '../components/App'
7
8   //import '../node_modules/bootstrap/dist/css/bootstrap.css';
9
10
11  const logger = logging.getLogger(__LOGGER__);
12
13
14
15  export default class IndexPage {
16      handleRoute(next) {
17          logger.info('handling index route');
18          this.data = ReactServerAgent.get('http://localhost:8081/api/posts').then(d => d.body);
19          return next();
20      }
21
22      getTitle() {
23          return 'Hacker News';
24      }
25
26      getHeadStylesheets() {
27          return [
28              "https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
29          ]
30      }
31
32
33      getElements() {
34          return [
35              <RootElement key={0}>
36                  <Header/>
37              </RootElement>,
38              <RootElement when={this.data} key={1}>
39                  <App />
40              </RootElement>,
41              <TheFold key={2}/>,
42              <RootElement key={3}>
```

Line 15, Column 1

REFERENCES

- ReactDomServer: https://facebook.github.io/react/docs/react-dom-server.html
- Templating: http://www.embeddedjs.com/
- Scaffolding: http://yeoman.io/
- Isomorphic React: https://react-server.io/