Server-Rendered React Components

COURSE OVERVIEW

Daniel Stern
CODE WHISPERER

@danieljackstern

Server-Rendered React Components

UNDERSTANDING SERVER RENDERING

Daniel Stern
CODE WHISPERER

@danieljackstern

Course Roadmap

Understand server rendering

- Where it is / isn't appropriate
- Costs and benefits
- Required tools

Create scaffolding for server rendering

- Create a server with Express
- Compile JSX with React

Develop with server-rendered components

- Send client pre-rendered components
- Understand and apply hydration (restoring interactivity)

A Cutting Edge Startup Application Scenario

You are the lead developer
at small tech startup.

Your boss decides you'll be building
a new app for their corporate clients, an
appointment book for senior executives.

Always pressed for time, these executives
want an app that loads extremely fast on
their phone, tablet or laptop.

Even though an application this big and
complex will take time to load in its
entirety, these clients want
instant results.

What do you do?

Solution: Server-Rendered Applications

The server does a lot of work usually reserved for the client – slow client devices are not a problem

Time spent downloading React, loading the app into memory and drawing interactive components are pushed until the end – client sees app first

Full functionality of the app (completing forms, saving data to server, etc.) will activate after React is loaded

Costs and Benefits of Server-Rendered React Applications

Server Rendered Apps – Costs and Benefits

Costs

Additional back-end logic results in more code than standard application

Additional tooling (express, webpack) creates additional vectors for bugs to appear

Some code may run differently on the server, resulting in an inconsistent application

Additional server workload results in increased (usually minimal) costs

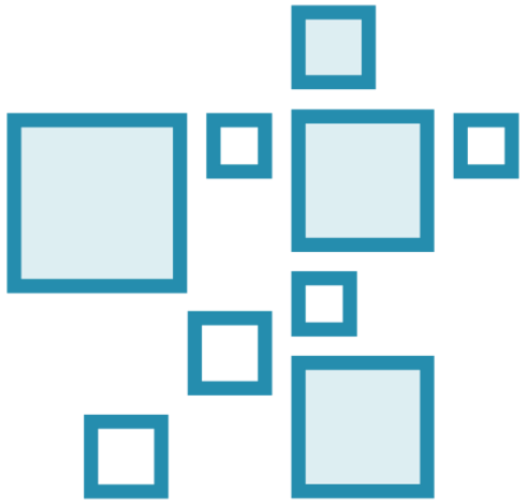
Benefits

More logic can be localized on server, reducing code sent to client

Application loads faster, with most marked difference on mobile devices

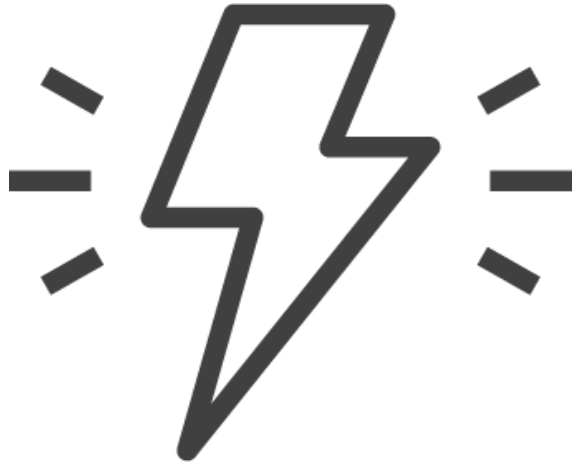
Instances of users giving up on using the app because it loads too slowly are mitigated

Server Rendered Applications Trade Complexity for Performance



Complexity

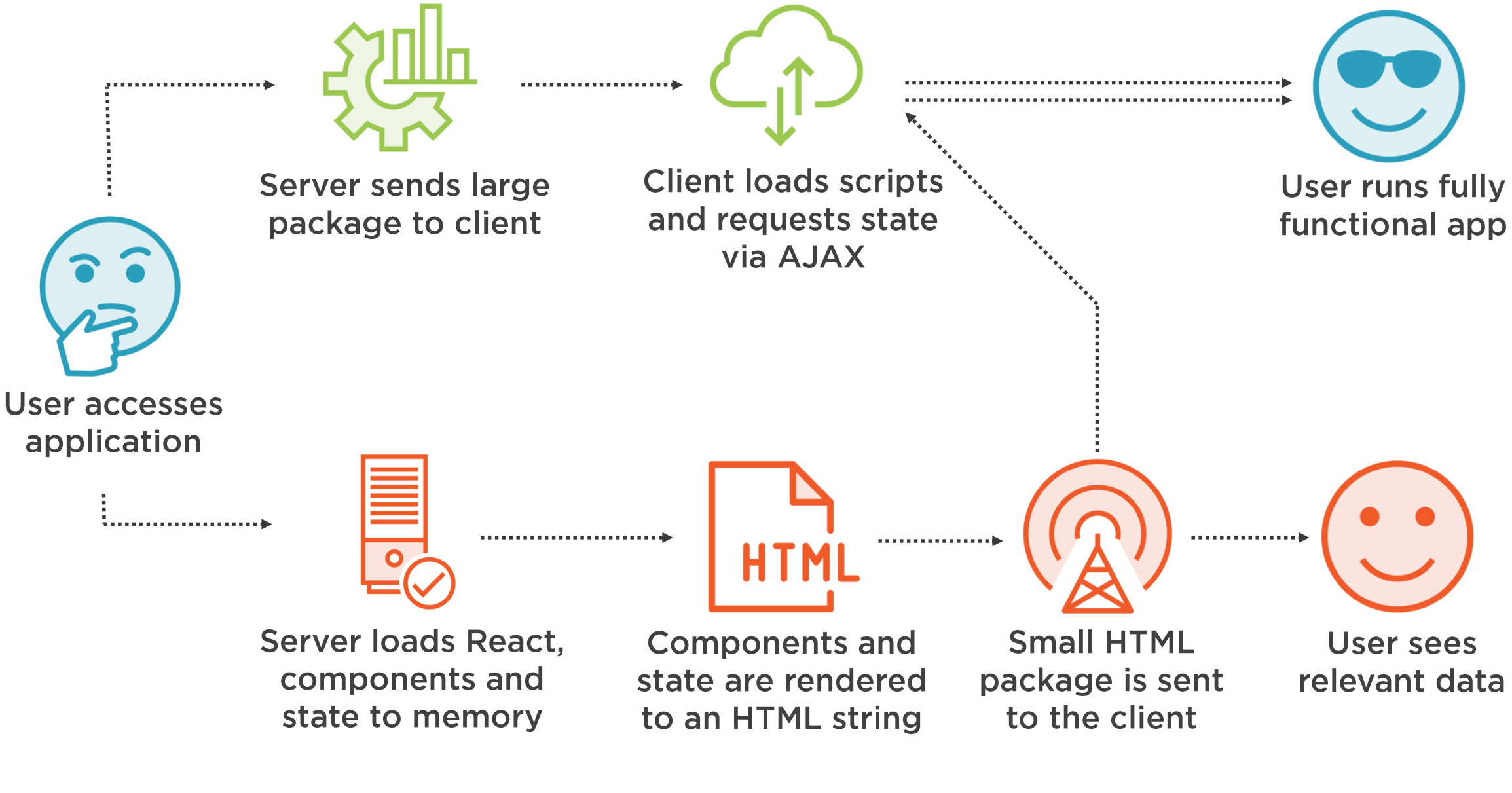
More code, sophisticated libraries, convoluted troubleshooting



Performance

Application appears faster on all devices, much faster on slow devices

How Server Rendering Works


User accesses application
Server sends large package to client
Client loads scripts and requests state via AJAX
User runs fully functional app
Server loads React, components and state to memory
HTML
Components and state are rendered to an HTML string
Small HTML package is sent to the client
User sees relevant data

Understanding Relevant Tools

“The mechanic that would perfect his work must first sharpen his tools.”

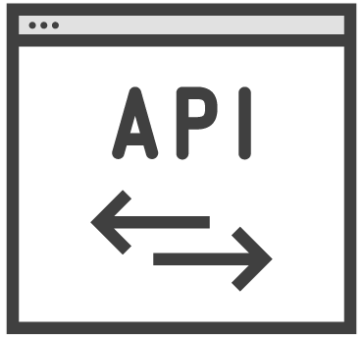
Confucius

Understanding Relevant Tools

Effective tooling saves time, and is needed for server-rendered React apps.

React

Renders JSX to HTML on both server and client



Express

Sends server-rendered HTML to the client

Babel

Allows JSX code to be loaded into server script

A Look at the Completed Application

Demo

A brief overview of the application and its component files

- Application will be built over the coming modules
- Code along at home (recommended) or just learn at your own pace

Note relevant application features:

- Delivery of server rendered content
- Interactivity
- Persistent state

Summary

Server rendered React applications trade more complexity for higher performance

- More tooling, technical knowledge and code are needed
- As a result, client sees relevant data on their device much faster

Server rendered applications first send a simple HTML package to client, then the majority of the code later

- Client is able to interact only after the majority of the code is loaded

Express, React, Webpack and Babel are the main tools needed

Coming Up in the Next Module

Using Node and Express to scaffold a program capable of server-rendering

Learn about the advantages and disadvantages of Create React App

Write React code using JSX

Scaffolding an Environment for Server Rendering

Daniel Stern
CODE WHISPERER

@danieljackstern

Learning Objectives

Fluently use Webpack and Babel to transform JSX code into JS

Use Express to create HTTP server where custom logic can be written

Create React components that have no internal logic and can be server or client rendered

Scaffolding Decisions: Using Create React App

What is Create React App?

Create React App is a command line utility that scaffolds React apps. Best practices are arrived at by consensus of a diverse and senior cast of developers.

Automatically generates express, babel and webpack configuration

Includes command line utilities for updating and maintaining project

Meant for interoperability within and even between teams

Advantages and Disadvantages of Using Create React App (CRA)

Advantages	Disadvantages
Little to no understanding of full stack web development needed	Little to no understanding of full stack web development needed
Constantly being revised by experts	Negligible educational value
Industry standard tool – developers are usually already familiar with it	Very large stack makes troubleshooting problems complicated
Automatically creates directory structure based on best practices	Intricate structure can only be modified from defaults by expert developers
Tools used are based on best practices	No choice of tooling
Easily implement supported features – linting, server-side rendering, etc.	Difficult to implement features not already supported

Creating a Project and Installing Dependencies

Troubleshooting

If you get stuck, verify the following things.

`^16.12.2` Correct React version

No local / global conflicts

Correct source code:
`https://github.com/danielstern/server-rendered-react-app`

Demo

Create `package.json` **and install dependencies**

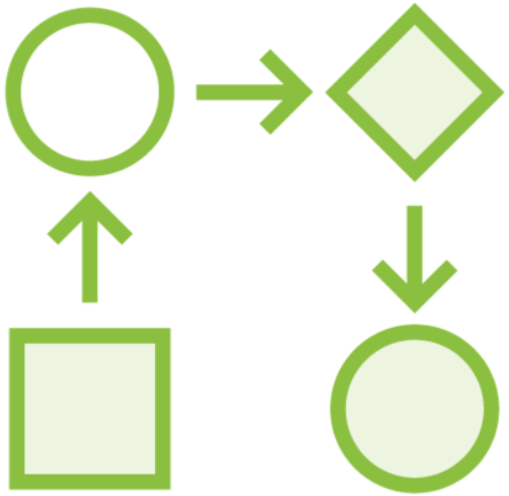
- Express, React, Babel and Webpack

Create simple "hello world" server

- More functionality will be added later

Setting up Babel

What is Babel?



Node utility which converts code from one language to another (usually outputs JavaScript)

Uses plugins (i.e., `babel-react-plugin`) to add functionality in a modular fashion

Demo

Create `.babelrc`

- Defines JSX transformation

Specify `npm start` **script**

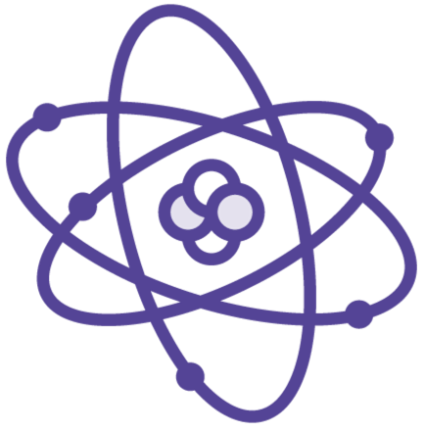
- Runs express server with `babel-node`

Verify and explore `babel` **functionality**

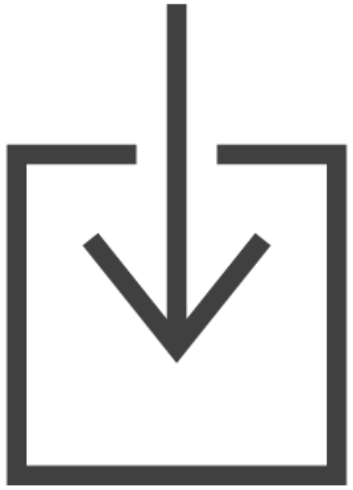
- Use import statements in express file
- Write JSX code inside server code

Creating the Main React Component

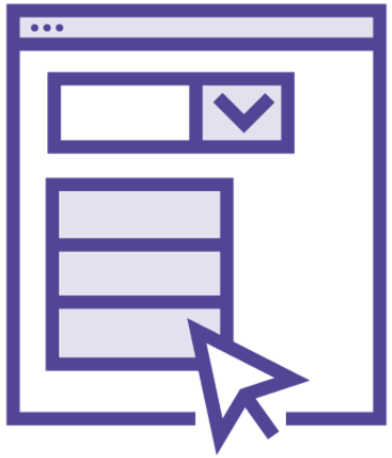
Creating Server-Render Friendly Components



State comes from external props only



No async methods or AJAX on init



Methods also only come from external props

Pure function which outputs HTML

Demo

Create a React component which will comprise application

- Code can be modular or all in a single file as desired

Configure Webpack and Babel

- Webpack will load file and pass it through Babel, creating Javascript file

Render component on client

- `ReactDOM` used to render component in browser
- Component will run just like without server rendering
  - All server rendered components must also work on client

Summary

Summary

Environment can be scaffolded automatically or manually

- Automatic scaffolding with Create React App allows for easier collaboration by teams
- Manual scaffolding with Babel, Express and Webpack allow for minute control

Babel used to convert JSX to Javascript

Webpack creates client version of application as single JS file

Main React component accepts external props and outputs pure HTML

Coming Up in the Next Module

Loading React components on the server

Using `renderToString` to create HTML output from React components, server-side

Sending pre-rendered React markup from server to client

Server Rendering
Basic React Components

Daniel Stern
CODE WHISPERER

@danieljackstern

Client and Server Rendering Comparison

Rendered on Server

Component sent as pure HTML and need to be rehydrated

AJAX requests are server-to-server (hidden from client)

Most of required processing power provided by server

Rendered on Client

Rendered components are fully functional

AJAX requests are client-to-server (viewable by technically skilled users)

Most of required processing power provided by client

Server Rendering Workflow

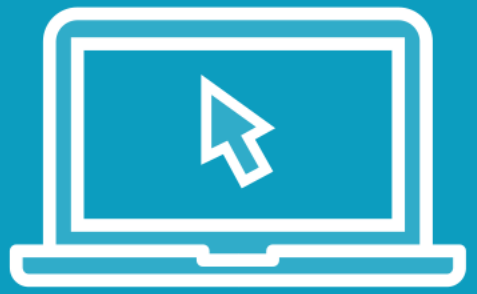
Server Rendering Workflow

Rendering a component on the server comprises three simple steps.

1. Load the React component and application state into server memory
2. Using the state, render the component to an HTML string
3. Send just the HTML string to the client

Rendering a React Component on the Server

Demo



Load React component into Express app and render to string in Node

Insert rendered string into HTML file

Send HTML with non-interactive React component to client

- Interactivity added next module

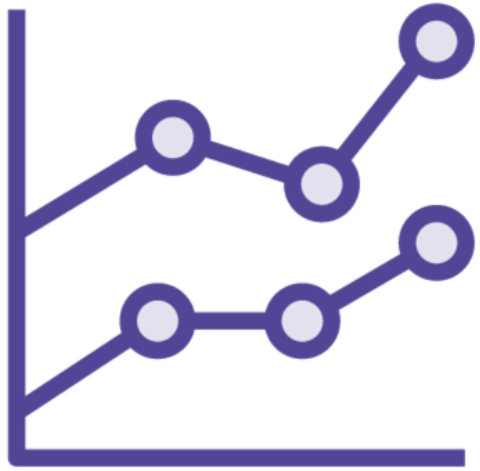
Note effects on performance / user experience

Understanding the Challenges of Server Rendering

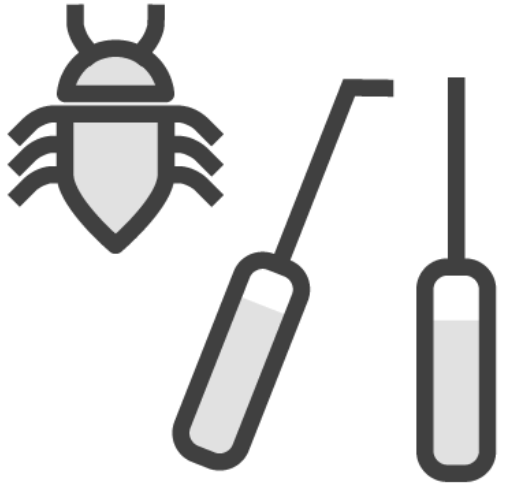
Understanding the Challenges of Server Rendering

Server, client communicate over HTTP differently

Code must work and be tested both on client and server



Sophisticated development process needed



More and unusual bugs require expert time to solve

Adding Functionality to Our Application

Demo

Create default state for our application

- Consists of `question` collection and `answer` collection

Create React component that renders newly created state

- Combines questions and answers into single, hierarchal list

Render component on server and send HTML to client

- Fast performance, but no interactivity yet

Summary

Summary

Server file can access state, templates etc. quickly and synchronously

Components are rendered on the server with the `renderToString` method

Components which have been rendered to string can be sent to client as HTML but have no interactivity

Coming Up in the Next Module

- Discover new tool – rehydration
- Apply rehydration to React application to add interactive functionality
- Completing the methods and styling of the React application
- Sharing code between client and server

Rehydrating Interactive React Components

Daniel Stern
CODE WHISPERER

@danieljackstern

Limitations of Server Rendered Components

Facade

A construction whose sole purpose is to provide a pleasing visual impression, though it may mask a structure that is crumbling, or not there.

Appears as though fully functional

Creates positive impression

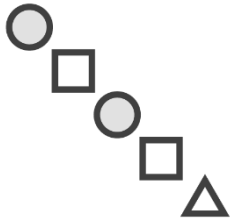
May be mistaken for the real thing by those in a rush

Actually does nothing

Limitations of Server Rendered Components

Server-rendered components are just bits of HTML with little real functionality.

Using buttons or forms does not have any result on state



Tooltips, sorting or other interactions will not function

Automatic communications with server will not take place

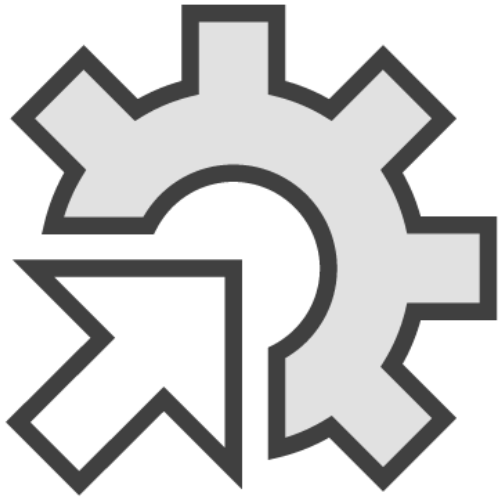
Understanding Rehydration

rehydration

The process of restoring interactivity and functionality to server rendered components

What is Rehydration?

React running on client recognizes server output as React and “binds” to it



React instantly and seamlessly substitutes fully functional app for server-rendered façade

Hydrated vs Non-Hydrated Components

Not Hydrated	Hydrated
Extremely lightweight HTML, No JS	Lightweight HTML, JS Libraries Needed
Can’t be interacted with	Fully interactive
Does not need React (or even JavaScript) to work	React and JavaScript must both be running on client’s device
Cannot update self in response to model changes	High-performance updates based on changing state, interactions

Adding Interactivity to Server Rendered Components through Rehydration

Demo

Create REST API allowing client to access exact same state as server

Update client script:

- Get state from API using AJAX

Rehydrate application on client

- Note effect on forms on buttons
- Note result of rehydrating app with non-matching dataset

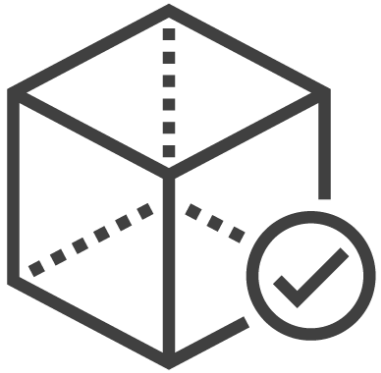
Sharing Code Between Client and Server

Sharing Code Between Client and Server

When the back-end of a server rendered application is not written with JavaScript, code often needs to be duplicated in multiple coding languages.

Libraries

Exact same React binary used on server and on client



Utilities

Custom code that works with your dataset is easily shred

Tests

Same specifications can test code on client and server

Demo

Create a simple JavaScript utility

- Modifies the value of one answer in an array of answers

Load and use the utility on the client

Update script to modify upvotes on server as well

- Use same utility as client
- Note how code is not duplicated
- Note how change to code base has identical effect on front and back end

Debugging Server Rendered React Applications

Server / Client Mismatch – A Unique Category of Error

Server prefers to access database directly

Client cannot communicate directly, uses API instead

If API modifies data or uses a different data set, app cannot be rehydrated

Server and client code usually get access to state in different ways. If the state does not match exactly, the app will not function.

Summary

- Server rendered components that have not been rehydrated are not interactive
- Rehydration adds interactivity to components without redrawing them
- Rehydrated applications can benefit from the same utility methods as server code
- Failing to match data sets exactly between server and client will cause errors

Coming Up in the Next Module

- Summarize overall ideas
- Extra assignments
- Courses to watch next

Conclusion

Daniel Stern
CODE WHISPERER

@danieljackstern

“We live on a placid island of ignorance in the midst of black seas of infinity, and it was not meant that we should voyage far.”

H.P. Lovecraft

Executive Summary

Server rendered React components deliver a great user experience

- Small download
- Low strain on slow devices
- Correct appearance

Express is used to send server-rendered components to client

- Babel allows server to load `.jsx` files
- `renderToString` method gives server an HTML-only component
- HTML-only components have no interactivity

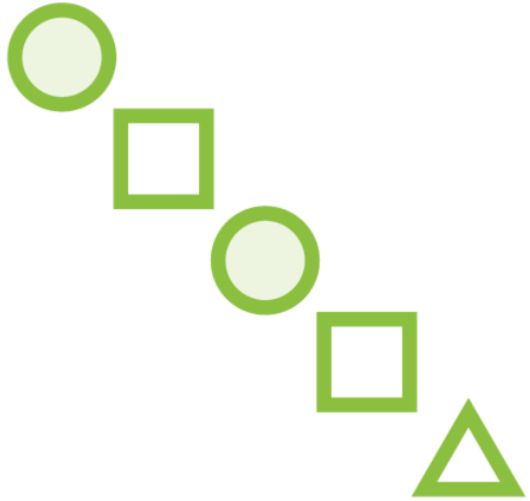
Executive Summary

Rehydration adds missing interactivity to server-rendered components

- Rehydration happens on client
- React must be loaded on client to rehydrate components
- React binds to server markup and restores interactivity
- Redraws are avoided when necessary

Extra Assignments

Insert form to add new questions / answers



Sort questions / answers by popularity

Limit upvotes / downvotes to one per user

More Resources to Explore

Build a Quiz Component with React
By Jon Friskics
https://www.pluralsight.com/projects/build-a-quiz-component-with-react

Building Applications with React and Redux
By Cory House
https://www.pluralsight.com/courses/react-redux-react-router-es6

Building a Full Stack App with React and Express
By Daniel Stern
https://www.pluralsight.com/courses/react-express-full-stack-app-building

Thank you!