Thomas Jackson – Code Creative on YouTube

30/6/2022

Presentation on React

Designed to be an "unpretentious" look! Tom said to take everything with a pinch of salt – he's biased and only started in the industry in 2018.

Which Framework & Rendering Method to choose:

Create-react-app. Next.JS and Gatsby are options.

React is a client-side renderer. Gatsby does it statically in build. Next.JS is Server side.

How do you choose?

In 2021, Google changed the way they rank websites for search, placing an emphasis on fast load times. This brought user experience to the front. So, there is a shift to SSG (Static Site Generation) and SSR.

Client-Side Rendering (CSR)

People were building with JQuery during 2010s as developer experience took a front seat. But this can be slow for the user. React grew in popularity as it gave the user a quick response as well. But, React has to run on the client's computer (which is what CSR means) which can be slow on old hardware and slow connections. Drawbacks – user can get a blank page for along time. This affects Largest Contentful Paint (LCP – how long it takes to draw the biggest thing on your page) and First Contentful Paint (FCP – how long it takes to draw the first thing on the page – e.g. page load time) poorly. Google uses LCP and FCP to rank search results now so your site could be ranked poorly.

Static Site Generation (SSG)

HTML built at build time before deploy, files uploaded to server, at run time the client is served html. But this is unsuitable for projects that are expensive to build or update a lot (due to the static nature of it!) – such as e-commerce where you add lots of products per day. This doesn't get rid of the complexity – it moves it to the build rather than client.

Static Site Rendering (SSR)

Server does the rendering upon client request, builds DOM elements, runs API requests, then sends HTML to client. Client doesn't receive a blank page, which is good for SEO as scores good for LCP/FCP, but has some disadvantages: Executing React twice (once on server then again on client to make interactive).

When to choose

CSR – when you have no SEO considerations. More for Proof of Concepts and the like (Tom admitted he was a bit biased against CSR though…)
SSG – when you want good SEO and your website isn't expensive to build and doesn't need updating frequently.
SSR – When you want good SEO and your website is expensive to build and needs updating frequently

Which rendering framework you should choose

Tom admitted bias towards Next.js

Next is capable of all rendering methods discussed as well as ISR and ISR on demand. ISR is Incremental Static Regeneration and allows you to mark a static page (that needs updating a lot – e.g. blog posts) with code and add some SSR to it so it can be updated dynamically.

Building with Next will make your app more flexible going forward if you ever want to change rendering method.

Styling Solutions

UI libraries are a good option, especially if you don't have time or aren't a designer. UI libraries are made by professionals so don't be afraid to use them. Example include Material UI and Chakra UI.

Component libraries – use these when you want a consistent design system and code base across multiple websites/projects. Styled Components etc.

CSS in JS – it's a mix of CSS and JS. The major advantage is that you don't have to write logic that changes an element's class based on something like a prop being passed to your component. Gives you access to JS variables in CSS.

Content Management Systems (CMS)

Headless CMS is the buzzword in the industry at the moment and where Tom thinks the industry is heading. It's where your content is decoupled from your presentation layer. Your CMS is a separate website that manages content on your production website. The CMS is effectively an API that you can customer with GUI/CLI and can be consumed by any front-end.

This means content editors can do simple things like capitalise a word without needing a developer to do it (because it's not in code anymore). Allows things like WordPress to be used.

Types of Headless CMS:

Full featured – e.g. Contentful. Have loads of features out of the box so easy to spin up, but hard to build features for.
Starters – e.g. Strapi and Sanity. Have a limited set of features to begin with but you have full access to the code that actually builds the CMS, so they are easy to build features for. They are also easy to test.

Tom recommends Strapi as it's cheap and where he thinks the future is heading.

Data Fetching & GraphQL vs. REST

REST has lots of endpoints and HTTP methods and uses URLs and HTTP to tell the server what data you want. Drawbacks can be over fetching (you get everything from the /posts when you only want the title) or under fetching (If you want information at /posts and at /authors you have to make 2 requests).

GraphQL has a single endpoint and only uses the POST HTTP method. Uses a custom query to tell the server what you want back. Drawbacks – you have to learn the query language and more to maintain on your front end.

Fetch vs. AXIOS

Built in fetch method is good for basic things, but Axios is a library that allows you to build complex features that rely on info from HTTP requests more quickly. If you have features like download spinners or progress bars it might be worth reaching for AXIOS. Other libraries are also available.

State Management

Types of state (some people argue about these but this is Tom's definitions and the important):

Local State – Very basic – could be something simple like a Boolean for a select. Generally solved in React with something like useState(). Lifting the state refers to placing your state higher in the component tree – but at what point does it become global!?
Global State – Where the piece of state is needed in places where it would be inconvenient to pass it as a prop. Don't want to do "prop drilling" (passing props down 4 or 5 levels deep), so we can handle this with the Context API or third-party libraries. Other options like Redux but Tom wouldn't invest much time in Redux in 2022 as it seems a legacy thing
External State – State associated with data from outside our app, typically from a GraphQL or REST API. If you have a REST API you could use React Query to help, or use Apollo Client for GraphQL API.

Form State – State associated with forms! Tom says if you ever have to build a form in React, please don't do it on your own, use a third party library like Formik.

Testing

Why test?

- To help you find where errors are in your app and why they are happening.
- To make sure you're confident your app works.

100% test coverage is good to strive for but often unrealistic. Prioritise the highest value

End to End Tests

Trying to mimic a user's interaction with your app

Could use Cypress or another headless browser.

Unit Tests

Tests for the smallest parts of our app. Could be things like util functions or components. Could use a library like Jest or React Testing Library

Integration Tests

Tests how the units of our app interact with each other. Could be front end or back end.

Q & A

How do we as Junior Devs find good examples of production code or apps so we can learn from them to be ready in the industry? Check out a big open-source repo and see how they write their commit messages, branch their code and use pull requests.

How do we avoid over-engineering our portfolio projects? Don't worry about this, use lots of tests and be able to explain why you've done something (e.g. Overkill to learn that thing)

What should people who are interested in Front End Development look in to next? Next.JS is a good idea and isn't just a front-end framework (does back end too!). But frameworks will change so don't worry about trying to keep "up to date" and remember different tools for different use cases (e.g. Svelte for smartwatches and small card payment terminals). Watch some talks from "You gotta love front-end".