Front-End Developer Challenge and Questions

We realize there are often multiple solutions or answers to the same questions and to the challenge in particular. We do not necessarily have “right or wrong” in mind, we are simply looking to learn a bit about you, and your approach to development.

Questions:

What front-end component libraries or frameworks have you used besides bootstrap?

As with many of these answers, there is a legacy component and a modern one. The legacy component began with using libraries and frameworks in approximately 2006 such as Python/TurboGears, then I began to focus on Ruby/Ruby on Rails and touched on ASP.net. I spent some time with PHP before all of this and way back in the late 1990s even used DreamWeaver. If I factor in my time with Google Web Toolkit (GWT) and Yahoo User Interface (YUI) libraries along with Spring/Maven web development using JSP/Freemarket templates, I’ve experienced a lengthy legacy compilation of various component libraries and frameworks over the years as a software engineer. It would be a challenge to recall all of them in detail.

Pushing forward to more modern times, I’ve been involved in multiple Python/Django projects as it's a legacy favorite. For the bulk majority of my focused frontend development in the past 5 years (most modern), I’ve focused on the following:

(This is intended to be used in conjunction with my resume without too much repetition.)

At The CollegeBoard, we developed a custom version of Bootstrap named Apricot and in addition to assisting with this, I wrote a custom ES5 revealing module pattern library with clean, straightforward HTM5/CSS3 design. This was as close ‘to the metal’ as possible with only jQuery and bootstrap used to assist.

At K12, I supported a modern (for the time) stack utilizing Backbone.js with Marionette.js appended to help us create more layered and dynamic views.

At REDLattice, I developed the technology for the full web stack and cloud cluster. I determined AngularJS to be the best for the frontend of this solution and later re-rewrote the frontend to utilize Angular4.

At OB1, I was responsible for our proofs of concept and technology decisions for our next generation responsive web version. I carefully weighed our team’s needs and chose ReactJS in combination with Redux. Before this I translated our current generation web technology to various iOS prototypes before developing a Swift iOS application client. This led to our ultimate decision to use ReactNative and ReactJS as future technologies.

What tools would you implement to enforce code quality rules and standards?

The most important part of enforcing code quality, rules and standards begins with mutually agreed upon agile processes for code reviews, documentation and high-level project management. Everyone on the team needs to be on the same page for maintaining and upholding the core elements of that process without it bogging them down too much. I find that it helps if that process is very open to dynamic, evolving changes taking into account feedback from individual developers. I’ve been deeply involved in designing these processes and also simply following them and taking notes from mentors and leaders.

Aside from the higher-level, more generic PM aspects, I find it most beneficial to:

1) Have a mutually agreed upon source control workflow (preferably using git/gitlab).
2) Use Linters, IDE Plugins, Automated Test Suites and other solutions that mitigate introduction of errors that may slip by undetected or cause code reviews to consist of wasted time.
3) I really like prettier.io (I used it in my solution) in conjunction with ESLint. I have it set up to automatically ensure that my code is always in prettier’s opinionated format on save in the IDE. An additional step to take here would be adding it to the git pre-commit hook. I offered this up as a solution to our frontend team a little while ago and we all agreed it would save us some time in addition to regular linting.

What HTML templating languages or engines have you used?

As I mentioned previously in the question about frameworks and libraries I have used throughout my career - the list is lengthy, therefore I will focus on the modern examples and stay brief on the legacy ones.

The most important thing I can say about templates of legacy years is that many were heavily focused on coupling logic and markup and were for rendering static HTML on the backend, then served to clients with minimal JavaScript. Most of these solutions were highly performant back in the days when it mattered most; these types of server-side solutions still have their benefits today, but mostly as an efficiency boost to rendering bits of client JS on the server to speed up slower clients. Writing HTML should stay as close to HTML as possible and never be performed in a traditionally server focused language like Java or Python. It should also mostly stay decoupled from logic.

Templating languages like handlebars.js formed a nice minimal templating solution. There were and are an almost limitless amount of frontend client-side solutions for web development; I've experimented with many, but in my experience Backbone, Angular and ReactJS have been the best web frameworks. Handlebars.js, Angular's templating and JSX with ReactJS are three of my modern favorites.

Which code related blogs or sites do you read regularly?

I personally prefer to run through courses on Udemy as my preferred method to try out a new methodology/technology. For reading, I have Google Inbox bundle up Quora, Medium and other collections of various development related topics I can peruse daily. I prefer full tech books, especially on Kindle, but it's challenging for writers and teachers to keep up to date. The best place to get the most up to date answers is through trial and error using the most up to date toolset docs, repository discussions and solutions from coworkers, friends and Stack Overflow/Google searches.

Frontend development in particular is always challenging itself with gauging best-practice solution sets because everything is constantly a moving target. It's exciting; it can also be a bit frantic to read three different books, watch three different 'professional' courses on Udemy and then read ten articles before concluding that there are only 'best attempts' at best practices in relation to many frameworks or methodologies.

In Sass, what is the significance of using an underscore at the beginning of a sass file name?

One of the features of SASS is the ability to create little modularized snippets of CSS that are easily reused. These are called partials. The underscore before the filename tells your SASS preprocessor to ignore those files when creating CSS, because they're intended to be included in your other larger SASS files.

Name a few (npm) node modules that you think are most useful?

There are plenty of node modules that are necessary for a given framework or its dependencies and add-ons. Honestly, the most important ones are the ones needed as dependencies to these frameworks such as all of the core pieces of a ReactJS app: react, react-dom, react-router, react-redux, redux-thunk, etc.

Listing these would take some space and be totally dependent upon the chosen framework. In a more specific direction, here are a few this year that surprised me in utility.

lodash and underscore were favorites on my last team to help with various utilitarian/ comfort of life in JavaScript areas.

request and async / promise modules are useful in many forms (request-promise, request-promise-native, request-promise-any). I found this useful in ReactJS proof of concepts and technology demos while we were trying to develop async/await code that wasn't littered with callback chain nightmares.

uuid is a nice polished little utilitarian unique identifier module.

axios I found useful when coming from my iOS swift codebase where I chose to use it, then continued it for web demos. Its for promise-based HTTP requests.

base64-js is useful for dealing with binary data in base64 rather than just text-based base64.

There are a slew of cryptographic libraries in which I found utility in the past year including bip32, bip39, libp2p, peer-id, etc. Protocol Labs has some great cutting edge implementations of IPFS (Interplanetary FileSystem) in javascript.

Given the option to choose Angular, React or Vue.js for a new project which would you choose and why? What type of requirements might affect your decision to use one over the other?

AngularJS and ReactJS are probably the two best supported, most capable modern frameworks for large projects. You can create a simple project with them, but there are plenty of lighter-weight solutions. Both of them are best suited for brand new projects and can be a challenge when trying to gradually modernize a legacy application.

AngularJS is great in my opinion for projects where you may have fewer development resources (fewer team members). This is due to its prescriptive nature. There's generally one prescribed way to do something in the stack and it means there's less disparity out there in solutions and best practices. The AngularJS team and its documentation are also generally under one umbrella for the tech stack. This is also its downside. Due to the prescriptive nature; if you don't like something about the Angular stack (such as its fondness for TypeScript), it doesn't play well with modularity. If you wished to not use AngularJS' template solution, you don't have a choice. One other small advantage of using this framework is an easier springboard for utilizing Google Material Design immediately within Angular Components. If TypeScript is preferred over Babel for modern JavaScript, AngularJS prefers it.

ReactJS is a modular framework. Almost every part of it can be swapped out for another solution in general. If you want or need a custom router or don't wish to use JSX for templating, you can choose another solution. The most popular state management solution for React is Redux, which has no actual tie or relation to React. This type of modularity means the community and its best practice solutions can be disparate, but it also has the benefit of allowing you to more easily integrate other solutions. I find this is a better solution in general for teams that have the resources to handle larger projects and it gives those projects more potential capabilities. Choosing each in the past for different reasons, I tend to prefer the career development and learning in the ReactJS space due to the wide variety and modularity of potential solutions.

Vue.js is a framework that has cropped up for those in-between solutions where a project might have significant legacy server generated code or need to be gradually modernized. It's not as battle hardened as the other two and hasn't had the benefit of as much experience, but it's a newer solution that seems to be a good stop-gap for modernization solutions. I would likely lean toward this for upgrading in steps if I didn't have the resources to do a complete new refresh with a new team.

If you were referred to an unfamiliar node project repo, what file or files would you first look to, to determine the commands to run, build and test the project?

Before anything else, one should always consult the README along with any other documentation on the repository's GitHub page.

Package.json is the first place to look in the project itself for the project's dependencies, the available scripts and all other configuration. Generally, I always check first to see which scripts are available and then dig deeper into what they do. For example, its typical for a frontend project to have some type of command to start up a development server that updates lives with code changes, to watch/re-run/re-build tests/css, or to build and deploy.

package-lock.json is also important and should be checked into source control; it has additional information in it to help npm manage semantic versioning, upgrading and is automatically updated/managed for you. I mention it here because I recently commented on a code review where someone attempted to edit it manually; It isn't for that, but it can be useful to read through for debugging/trying to understand npm issues.

What is the benefit of using the HTML 'nav' element vs. a div with the class of .nav?

The simplest real world example of the nav element being used instead of a div with a class of .nav is its interpretation to screen readers that parse out navigation and present it to the user on demand. Accessibility is mandated by law for many web applications.

Any markup language is designed to be parsed by its client and turned into some type of presentation and meaning. Markdown is a popular format for quickly writing blogs. There are markup languages for writing music and writing screenplays, etc.

HTML is used for two main purposes; it provides meaning(semantics) and it provides an ability to define the presentation for the web. Historically, many of the tags in HTML say something about what type of content is inside. A form tag contains some type of user editable form. An image tag contains an image. Tags such as div and span don't have any associated semantics.

HTML gained additional semantic tags to aid a variety of different web browsing clients having better abilities to interpret the meaning of HTML content to users. Accessibility and web standards use new semantics such as header, footer, nav, section, etc. to provide additional meaning where div and span were lacking. This additional meaning is a standards-based approach to ensuring that accessibility plugins such as screen readers or alternative browsers can present information in a meaningful way without need for presentational cues.

What code related framework, library or language do you want to learn next?

Primarily, I would like to dig much deeper into ReactJS, keep determining the best-practice stack and continue enhancing my frontend skills in all areas. I like focusing on the solutions that are closest to native JavaScript/HTML/CSS, especially supersets like Babel and TypeScript can provide. With Babel, being able to write for ES2017 before the features become standard across browsers is a great place to focus efforts. I think native JavaScript, HTML and CSS are going to rapidly progress and many of these frameworks will contribute to a growing number of standards being accepted by the standards committees at a rapidly increasing rate.

There are a slew of upcoming new, small, lightweight web 'frameworks' to always keep up with for the occasional hack.

I would like to learn more about Vue.js

Hobby-wise:
I try to keep up with iOS and Swift development; I particularly like the direction in which Apple is going with iOS apps coming to MacOS.

I have been gradually prodding forth at Unity game development and have spent some time on VR dev for the Oculus Rift.

Some Notes on My Solution for the Challenge:

I chose to use create-react-app in combination with reactstrap after some initial research. I definitely noticed that there is no mention in any of these requirements that ReactJS should be used, but the folder is called ‘React Developer Challenge’. Therefore, ReactJS has been chosen. The requirements can easily be met with a much lighter solution.

Reactstrap seemed like an interesting module to try for the solution as it allows you to write much of the Bootstrap 4 with JSX without wrapping it yourself in components. The obvious downside is the extra disconnect from Bootstrap. How well will reactstrap be maintained? How quickly is it updated? I found the disconnect in documentation to add a bit of nuisance in using it as well; overall I would probably skip the library for a serious projected write components in React that wrap Bootstrap. One huge pro is that it definitely makes your JSX templates super clean.

The default create-react-app app sticks everything into one div, losing all of the semantic HTML. I changed that by having it render into a header, main and footer. In a real, complex application, the solution wouldn’t be quite that simple as it would involve routing and a more complex breakdown of components.

I did a basic ‘break things down’ into components and tried to follow all requirements and get it fairly pixel perfect. I showed that Bootstrap’s Sass was imported, but didn’t really need to override it for this simple demonstration.

I hosted the solution on GitHub and ensured its accessible for you on GitHub pages at the following link. I will remove it after you’ve had a chance to see it.

Thanks for the opportunity to complete the challenge and if you have any additional questions for me, please send them my way: jashot7@gmail.com

Challenge:

Using the latest version of Bootstrap, create a single, web page that collapses responsively from desktop to mobile and where the desktop and mobile breakpoints match the wireframe images provided.

Utilize the Bootstrap grid system, components, and classes. Take care to demonstrate the use of modern, semantic HTML elements.

The carousel at the top of the page should contain three slides – which can all display the same content, but the carousel itself should be functional in that clicking left or right will animate to the previous or next slide.

Additionally, in the mobile view, the "hamburger" (three lines icon) menu button should collapse and expand the navigation.

All other components are only present for layout purposes and do not need to function - navigation links do not need to lead anywhere, search does not need to work, and so on.

Challenge Setup:

At a minimum, you must utilize Sass to import Bootstrap via your choice of any build tool (Gulp, Webpack, Parcel etc) or CLI to create the HTML and CSS output. The output should be locally viewable by simply loading an index.html page into the browser.

Use whatever tools you are most comfortable with to accommodate this, you do not have to reinvent the wheel.

Combine all of your challenge directory/files as well as this document with completed answers in a single .zip file and deliver back to us as soon as you have completed them.

Please provide your entire directory structure (with the exception of node_modules if it is generated by your toolset) to keep totally delivery size small.

Feel free to contact us with any questions and thank you for taking the time to complete these items!