JavaScript CASE STUDIES:

- We did a lot of work over the last year for a venture-funded startup focused on adaptive commerce with the goal of personalizing shopping beyond what standardly exists today. Their project was an e-commerce overlay system where they augmented existing catalog data with personalized filters and recommendations for the data sets of their client websites. It handled injected JavaScript, had Cross-origin resource sharing (CORS) compatibility, had NoSQL data stores (Mongo and ElasticSearch), and used lightweight custom JavaScript objects in their newer implementation. More information regarding some of the items we helped with:
  - Worked on a tool that could be injected into a customer website to determine if there were any glaring issues with how the toolset would run in that environment.
  - Helped analyze and document how their primary configuration generation tool operated.
  - Architected and organized how their new version of their platform would ultimately be used to manage a new site implementation from start to completion.
  - Debugged a series of performance issues with an older deployment of their client-side tools.
- For one client, who offers omnichannel personalization for clients like Best Buy, CDW, Target, Wine.com, Neiman Marcus, Costco, Burton, and Walmart, we helped write up a set of tools and tests to help ensure stability of the client-side libraries that their customers use. More information about how we contributed:
  - Built out a series of comprehensive behavior and unit tests for their client-side injectable component (custom JavaScript objects), which were executed through Jasmine. The component provided all of the access into their core system. These tests ensured that stability would be preserved when the component was modified (it would trip a test either from a function signature issue or behavior change). If there was an error or bug in the component, it would break all integrations with the backend solution so it was an important set of tests. There were approximately 1,500 assertions that went into the test suite. QUnit is another viable alternative that we explored for test execution.
  - Built out an injectable detection tool that would execute a series of validation checks to determine if the customer had their local environment properly configured and was using the injectable component correctly.
  - Designed the detection tool to accept and deal with CORS issues.
  - Set up the detection tool to be library light so a minimalistic set of components were required to make the tests/validation work.
  - Created the tests so they could be used in a harness where the page could be loaded and the tools injected and the test results parsed so the harness could be run on a schedule to keep an eye on customer sites to ensure that they continued to stay compatible with the client-side tool.
- We also did a lot of work for a Fortune 10 customer over the last couple of years. Their system dealt with nuclear power plant inventory and procurement, and it utilized both AngularJS and

Twitter Bootstrap in its construction. It used a typical Java enterprise stack (jBoss, Java, and Oracle) for the service layer. Most of the functionality was handled client-side in AngularJS however. More information about the project:

- AngularJS helped as it allowed good separation of concerns to be enforced on the client side. We were able to model the data for users, plants, inventory, parts, etc., all on the client side. This information would be saved and managed there and synchronized with the service processes. It resulted in a streamlined experience for the user that was exceptionally fast, one of their core requirements. Along with that, the breakdown of services and controllers and ensuring that DOM manipulation stayed in directives allowed easy testing of the application's code.
- Twitter Bootstrap was also used to normalize a lot of presentation items in the application. Since it had to support a wide array of browsers (IE8 support included), as much as possible was used from the frameworks to ensure consistency.
- A feature requirement was mobile support so the groundwork was laid for responsive design through Bootstrap’s use of device-specific classes.
- D3 was used for a series of reports and pie-chart graphing in the core UI, showing break-out classifications for various parts.
- We were called on to help the client’s internal systems team troubleshoot environments and issues unrelated to the work that was performed (the code worked well in a newly-built environment). We used our systems experience and troubleshooting skills to identify issues with application and database configurations from a standpoint of what REST endpoints were doing in the application and how the service layer was handling those requests. Since access to infrastructure was restricted, the solution-discovery approach was a matter of reconstructing test cases and coding out a race condition situation to demonstrate how session data was getting stored inconsistently from one instance to the next in their jBoss cluster. We also assisted with the execution of patch scripts and debugging when different environments went out of sync with the datasets they supported.

- A startup selected us to be their interim CTO in 2014 because of how well that project went. It was a very enjoyable project because we were involved from the start to where they are now. Things have been going so well that the founder was able to quit his day job due to the success of the business he created and bootstrapped. The toolset consists of a widget toolset that can be injected into third-party websites by web administrators. It handles API calls to a centralized service to serve up survey configurations and capture form data submissions that get posted back. The widget has a lot of flexibility in its behavior and how it renders for a given site. The application build out was approached in a Minimum Viable Product (MVP) manner, allowing for the main application to rapidly be prototyped and made public for paying-customer use. Many of the latest features have evolved from specific requests by existing customers' usage, which was the intent of the approach. More information:
  - JavaScript-specific object structure for rendering the widget and providing a data model for the widget to utilize on the client side.

  - Use of closures for code encapsulation.
  - A basic initialization signature was used so the project could use a version of jQuery that was already available on the site. jQuery is the only dependency of the toolset.
  - CORS configuration in place to allow the cross-domain API calls from the JavaScript classes.
  - Flexible client-side data embedding that can be sent along with form submissions. This enables detailed data analysis of different demographic lines for specific websites.
  - Implementations of different form factors of sites (desktop versus mobile).
  - Use of ElasticSearch on the backend for survey storage and form post handling.
- For one client, we used websockets to connect a number of browsers together to control the presentation and answering of survey material. A controlling interface would show questions and results and use websockets to communicate state changes, which would route though the app system running on Node.js, and in turn push out the notifications to the other clients. Clients could register/remove themselves from the app and subscribe to a certain channel. The system also handles if a client drops off and comes back online, tracking the socket ID signatures for different clients and tracking their connection status.

NEXT STEPS:

We hope the information provided above is useful and that we can explore working together further. If you have questions, requests, or feedback, we'd be pleased to hear from you at any time.

Thank you and Respectfully,

Peter Hanson

Peter Hanson, Founder & CTO
Up and Running Software, Inc.
Cell: 906-281-1178
Email: peter@upandrunningsoftware.com