Experiential Learning Through an Internship at Capital One

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

David Dimmett

Spring, 2022

ADVISORS

Rosanne Vrugtman PhD, Department of Computer Science

Daniel Graham PhD, Department of Computer Science

Experiential Learning Through an Internship at Capital One

CS4991 Capstone, 2021

David Dimmett
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
dcd2sg@virginia.edu

ABSTRACT

Whether classroom learning supports success in experiential learning environments can be an important matter for computer science students wanting to work in industry. To examine the relationship of education and work, I use my recent internship at Capital One, which was to extend the functionality of a web application using JavaScript and Vue.JS. I also implemented functional tests to verify that our work met business requirements. Computer science courses at UVA like Advanced Software Engineering and Algorithms gave me substantial background knowledge that I could adapt to the requirements of my job. However, more collaboration in some of these courses would have better prepared the communication and teamwork skills needed throughout my internship. Finally, the project I worked on could be improved by refactoring the code to increase readability and maintainability. Doing so would make future extensions to the application easier.

1 INTRODUCTION

"Responding to change," is one of the core principles of the Agile Manifesto [1]. My summer project at Capital One reflects the need to embrace this principle. My team's task was to introduce a new user type for an application that monitors and assesses call center agent performance. When a manager logs into the system, they can listen to their team's calls, address performance needs, and assign their team to another manager if they need to be absent. We added a senior manager user to improve upon the team delegation functionality. Previously, a non-delegated team would have to be manually assigned by the engineering team via a database entry. Our general approach was to update the backend API and frontend interface to support these changes. The team I worked consisted of three other interns. Two worked on API changes, while two (including me) worked on the frontend.

Frontend web development can be challenging for many reasons, one of the most difficult being, implementing an application which meets functional requirements and is also pleasing to use. When adding new features to an existing project, there is an added challenge of preserving previous work while deciding which segments of older code need to be changed to support the new features. This problem is emblematic of the challenges of iteration in software development.

2 RELATED WORKS

Matthew Wood describes integration testing in "Integration Testing with React and Enzyme [2]." While we did not use React or Enzyme and did functional instead of integration testing, we followed a similar approach to API mocking. "Micro Frontends," by Cam Jackson, gives an introduction to the micro frontend architecture which the application used [3].

3 PROJECT DESIGN

The following subsections describe various aspects of the project's designs. Sections 3.1, 3.2, 3.3, and 3.4 describe the technology stack, implementation, testing strategy, and challenges respectively.

3.1 Technology Stack

The application was just one container in an overarching system which uses the micro frontend architecture. Micro frontends are analogous to the microservices architecture [3]. The primary library used for the frontend was Vue.js. Vue is similar to other frontend frameworks in that it allows websites to be designed using a component structure. HTML and JavaScript define components using a declarative template syntax. The HTML section defines the appearance of the component and can contain embedded JavaScript variables. If the JavaScript code modifies a variable, the template will be automatically redrawn, so the changes are visible [4]. This property, along with components being reusable much like functions, is one of the reasons frontend frameworks like Vue are so useful.

Vue also provides several extensions to its core library. The two that the codebase used were Vue Router and Vuex. Vue Router introduces routing that turns a Vue application into a Single Page Application [5]. An SPA mimics a typical multipage application, but the appearance of moving between different URLs is a clever manipulation of the DOM [6]. Vue router is particularly useful because it allows for further organization of the code base through page views.

Vuex is another extension for Vue which implements a one-way data flow state management library. In this architecture, there is a global, centralized store which holds the data for the application. The view layer can access the store and dispatch actions that mutate the store [7]. Figure 1 shows the flow of this design pattern.

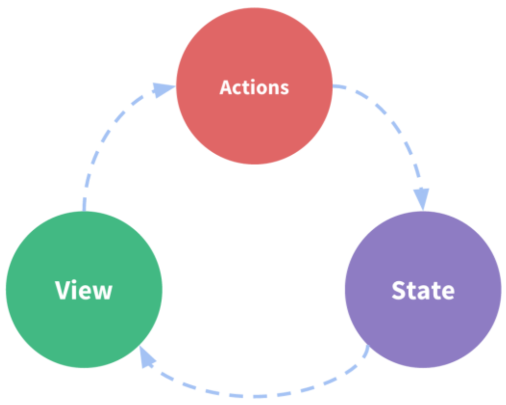

Figure 1: Vuex architecture [7]

The frontend consumed a variety of APIs. The two most important being the API to collect direct report information and to make team delegations. The existing delegation API consisted of a JavaScript AWS Lambda and Elasticsearch database.

3.2 Implementation

We divided the extension of the frontend into two phases. The first was to update the view after logging in so that the senior manager could view their team. On the backend team, the API had to be modified to collect the managers reporting to the senior manager and then the agents reporting to each manager. The frontend received this data as a large nested object. The presence of nesting determined whether the user was a senior manager. There were several alternatives proposed for determining if the user was a senior manager.

We initially tried fetching the direct reports of the user and then iterating over the result as a list to find the direct reports of each manager. The performance of this approach was slow for even modestly sized teams. However, the alternatives were not stable in the results they returned. For example, some high-level individual contributors were counted as managers, even though they were not while using other APIs. Our team did not own the direct reports API either, so modifications to suit our needs were not immediately possible. In the interest of time, we decided to stick with our original approach, but made some performance enhancements using Promise.all. Before, we were waiting for each API call to finish before starting the next. Promise.all accepts several asynchronous functional calls and finishes when all the API calls have completed [8]. This method sped up the team data fetching process enough to be acceptable for our scope.

To display this new team data for senior managers, we changed several aspects of the application’s design. The first was to change the language on the main team dashboard to reflect a senior manager’s perspective. For example, words like “Agents,” were changed to, “Associates.” We changed the breadcrumb navigation bar to display “Manager:” before the names of users that were managers. We also added a new page route in View Router for the manager team page. Likewise, we added dynamic links, so a user could click through senior manager, and manager team levels to view an individual agent.

The final modification was to enable senior managers to make delegations. On the backend, we added a single field to indicate who made a delegation. On the frontend, we added this field to the API call and changed the wording to indicate the user was delegating on behalf of another team.

3.3 Testing Strategy

The frontend already had excellent unit and functional test coverage before work, so maintaining the coverage levels was essential. I was only assigned functional tests, so I focus on these tests here.

The functional test suite used Cucumber and Cypress to execute the tests. Cucumber is a behaviour-driven testing framework which uses a language called Gherkin to define test cases. Gherkin uses a plain language like syntax to define test cases based on the acceptance criteria of a user story. These are called steps, which are then backed up by code that actually executes the assertions and mocks common in test cases [9]. Cypress is a test runner which simulates a live web browser to run tests against. Most of the functional test cases assert some condition in the web page has been met. For example, one user story required the page to display an error if the team data API did not work. The test case then mocked the API for failure and asserted that the error component was on the page.

3.4 Challenges

There were two major blockers that occurred during development. The first was an issue related to feature toggles and an API call. Throughout development, we used feature toggles to guard our work from going into the production environment too soon. The frontend fetched the feature toggles via an API call in the top-level component and stored it in Vuex. In the breadcrumb navigation component, rendering would sometimes finish before the API call returned. Even if the API call eventually finished, the page would not update accordingly. We found that assigning the feature toggle variable to a watcher property in Vue allowed us to poll if the toggle was ready. Although this solution was somewhat verbose, it prevented an awkward inconsistency from appearing to the user.

Setting up API mocking in the functional test also proved difficult. The version of Cypress needed to mock network requests was incompatible with several other JavaScript packages used in the application. We had to carefully upgrade those packages and remedy any issues caused by obsolesce. This change also had to be approved by another team. The benefit was that we could better test our changes.

4 RESULTS

We were able to complete all user stories for implementing the senior manager user type. While I do not have quantitative data on our work, our team did present our work before several managers and the team's product manager. These stakeholders positively received the work. The completion of our work also meant that the engineering team will no longer have to shift focus away from normal work to handle database management tasks.

Besides the material results discussed above, I learned more about frontend web development, software testing, and Agile. Most of my frontend experience has been with React, but learning Vue showed me some different approaches to the problem. Despite overlapping in some areas, Vue is much more declarative than React. Vue also provides more built-in functionality that would otherwise have to be achieved through third-party packages in React.

Software testing and Agile development are both discussed in classes like Advanced Software Development Techniques; however, I had never experienced these topics in such a structured way. On my team, there was a dedicated Agile delivery lead and product manager. The product manager was the primary stakeholder of our project, so being able to refer questions to them during stand up was very beneficial. I also gained experience using the Jira software to manage sprint tickets. Most of the code I wrote was test code. The functional testing experience I gained taught me about the importance of crafting precise test cases. I also learned different approaches to browser simulation and API mocking.

5 Conclusion

Software is an iterative process. During daily stand-up meetings, requirements were discussed and updated, showing the need to embrace the iterative process. In the end, this iteration led to a finished product that satisfied our stakeholders.

6 FUTURE WORK

In retrospect, the method for determining if a user is a senior manager, was brittle. The check is repeated on multiple pages, and is not cached. In the future, new pages wishing to incorporate the senior manager user data will have to repeat the API call. The user-type checking logic could be moved into Vue Router as a beforeEach navigation guard. Navigation guards are functions that are run each time the user navigates from one view to another [10]. We would move the checking logic to a navigation guard during the initial load of the application. This modification would simplify the codebase and make it easier to add new user types in the future.

7 Evaluation

Nothing in the CS curriculum explicitly taught me Vue, JavaScript, or functional testing with Cucumber. However, I felt well-equipped to learn all these technologies on my own. The background knowledge I have gotten from CS courses like Algorithms, Program and Data Representation, and Advanced Software Development is therefore adaptable to many scenarios. However, students would benefit from more group work experience. Many classes require students to work independently and ban collaboration, but we were expected to collaborate frequently in the work environment. More group work in CS classes would improve communication and project planning skills.

REFERENCES

[1] Manifesto for Agile Software Development. Retrieved October 23, 2021 from https://agilemanifesto.org/

[2] Matthew Wood. 2018. Integration Testing with React and Enzyme. Retrieved October 23, 2021 from https://tech.ebayinc.com/engineering/integration-testing-with-react-and-enzyme/

[3] Cam Jackson. Micro Frontends. *martinfowler.com*. Retrieved October 23, 2021 from https://martinfowler.com/articles/micro-frontends.html

[4] Introduction | Vue.js. Retrieved October 24, 2021 from https://v3.vuejs.org/guide/introduction.html#what-is-vue-js

[5] Vue Router. Retrieved October 24, 2021 from https://next.router.vuejs.org/introduction.html

[6] SPA (Single-page application) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. Retrieved November 13, 2021 from https://developer.mozilla.org/en-US/docs/Glossary/SPA

[7] What is Vuex? | Vuex. Retrieved October 25, 2021 from https://vuex.vuejs.org/#what-is-a-state-management-pattern

[8] Promise.all() - JavaScript | MDN. Retrieved October 24, 2021 from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/all

[9] Introduction - Cucumber Documentation. Retrieved October 25, 2021 from https://cucumber.io/docs/guides/overview/

[10] Vue Router. Retrieved October 25, 2021 from https://next.router.vuejs.org/guide/advanced/navigation-guards.html