Smart Suggestions

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

Michael Asare

Spring, 2023

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Daniel Graham, Department of Computer Science

Smart Suggestions

Technical Report on Summer 2021 Internship

Michael Asare
Computer Science
University of Virginia
Charlottesville Virginia USA
msa8wsy@virginia.edu

ABSTRACT

Amazon, as an old company, had an issue with usability and security compliance with their outdated internal permissions site. I was an intern at Amazon working on an AWS Security team, who was tasked with making smart suggestions to an internal authorization system to improve Amazonians' compliance with security standards. In order to address this problem, two new suggested features were created for the internal site. This was done after designing the features fully, analyzing the other potential approaches, and then altering the model-view-controller model for the website. Eventually these features were rolled out to production, with massive usage being seen instantly. Overall, the project had given me a technical challenge to solve, had improved the experience of thousands of employees internally, and finally allowed Amazon to properly evaluate me after the internship was finished.

1 Nature

Amazon's influence in modern society cannot be understated -- the company's valuation has tripled over the last five years, reaching a peak market cap of 1.73 trillion dollars. As a result of this swift growth, Amazon has placed an incredible amount of emphasis on workforce expansion. My internship with Amazon was curated to put me in the best position possible to convert to a full-time job offer, with the curated mentor-mentee relationship, major support structure, and high sense of ownership. Primarily speaking, the nature of my internship was the equivalent of an extended interview. I will admit that the internship did have an underlying focus on learning new technologies, however I would be lying if I didn't emphasize that the internship was a test run of living an Amazon SDE lifestyle.

Regardless of the semantics behind the internship, I still had my own thoughts on how I viewed the internship's work. Every day was a learning experience, which my mentor had highlighted every day that I had gone to work. The project that I was working on had very few specifically outlined plans on how I would complete my project. My manager gave me an end goal (giving suggestions to our 'customers') and told me what the makeup of the systems I was going to be working on. Outside of that, the suggestions project was a tale of myself, an inexperienced developer, learning what I should be doing as I was doing it. The nature of this project, or better yet, the internship overall, was an experience of me learning to deal with ambiguity as Amazon vetted me during that process.

2 Purpose

I have a fiery passion to learn. Despite this strong desire and in ironic fashion, I had not gotten an internship at any company beforehand. So, when it came to this internship in 2021, I wanted to learn; learn as much as I possibly could. With that motivation in mind, working at Amazon felt more like learning at Amazon. I cherished the opportunity as much as I could as I knew that you can only be taken as an apprentice for so long. Personally, this internship had a dual purpose -- to teach me things I couldn't find in a classroom textbook and to immerse myself in an unfamiliar system tasked to understand the tools at hand. I had experienced work-specific events that simply cannot be properly simulated in a lecture hall. I had never heard of a daily SCRUM style meeting, retro, or weekly design meetings. On top of that, many technical tools were at my disposal, both familiar and unfamiliar. However, there was an evident learning curve in turning functional knowledge into real application. Who would have thought that I would have to fully understand the advantages and disadvantages of certain git commands, such as rebase, fixup, or a squash, when I was so accustomed to simply pulling and merging at will? All in all, this internship had significance to me as a complete learning experience, from top to bottom.

Additionally, past my own passion to learn, the project I had done at the internship served the purpose to display what I could do for Amazon. I was on a security team, which had the job of authorizing persons to certain roles that would then grant permissions to certain resources. Having a role meant that the people who fell underneath a role had access to resources at Amazon that were deemed vital to the work they were doing for that role. Authorizations were decentralized, so any Amazonian was capable of organizing permissions (we called these organizations teams). This process of organizing permissions had been at the company for a long time, which would mean that there were many archaic processes still in use today. The outdated nature of these processes were then exacerbated by the low usability of the fundamental service we provided. Many

complaints were made in effort to stir attention towards bettering the user experience. My project, which was a project to provide various suggestions to users, was focused on improving that user experience. Because of the widespread impact of our team, and the perceived impact found in user experience improvements, my project had a high potential sphere of influence.

3 Work Accomplished

As stated earlier, I was given a project on a longstanding security team. On this security team, there was an internal site based on role-based access control principles. So, someone could set up a 'role' where if you were given that role, you would have all the bundled permissions with that role. Or to put it simply, people could make teams, add people to these teams, and then associate resources and permissions with that team. Our team handled two services. The first service was an extremely old service that used a 17-year-old tech stack, which was unfortunately coupled with a terrible UX. The second service was modeled after the first service; however, it had a new UX, with novel features such as automatic team membership rules that would allow for hands-off management of teams (no more hand adding!). I was tasked with creating suggestions for people who used our second service, where we'd suggest making team edits that would help adhere to security standards, e.g., minimizing manual management, least-privilege (minimum privileges needed to complete work), and reducing the number of unneeded members on teams (reduce redundancies).

Commonly on teams, people were assigned a team automatically through rules. Rules were a way of automatically assigning team membership. Rules could be defined by a multitude of attributes, including location, building, and whom one reports to. Team owners could also hand manage team membership by specifying exactly who would be added to a team list and vice versa. However, automatic rule management was the heavily favored method of team membership by our team as membership automation via rule management aligned with many more of our security standards. For example, people who were by hand added to a team would also have to be by hand removed from that team, meaning that there would be possibilities of people having extra permissions to resources they should not have (as we tried to adhere to least-privilege whenever possible).

Frequently, people were given overlapping permissions (granted by both rule management and hand management). These overlapping permissions were redundant, and did not adhere to our recommendation of strictly using rule management whenever possible. As a result, I devised a feature for the internal team membership site that would suggest removal of the redundant permissions.

To begin with this redundancy removal suggestion, I created a design document outlining the feature being proposed. I documented the potential impact of the feature by finding data on how many teams or members could be influenced by the feature. Then, the actual algorithm of determining if a member had redundant permissions given to them. Because this algorithm was quite trivial, and only required knowing the set of permissions for a given team, I did not have to extensively document the potential resource impact on the current systems if added. This is mostly because of the fact that many of these checks were already in place, and I simply had to add another step in the pipeline of steps to detect redundancies. I then outlined how the model-view-controller architecture would be altered to allow for http requests that would check for redundancies in membership. Finally, I would collaborate with a team of UX designers to come up with an interface to the frontend of the feature that would both be consistent with the current design of the internal site and also be designed to be intuitive to use.

This design stage was then followed by a design review stage by my teammates, who then approved of the feature after some minor critiques and future prospects discussion.

Implementation of the actual feature was not too difficult after the extensive planning of the feature. Testing of the feature was all done manually, so unfortunately there were no specific unit tests to test the correctness of my solution. Regardless of the lack of pure Blackbox testing, there was still extensive testing done through manual testers and time spent on beta testing. After the beta stage had ended, the feature was then pushed to production servers, and was quickly used.

Rule management had been a stressed feature throughout the internship as my team wanted to make sure that team owners were using rules whenever possible. Unfortunately, my team ran into a common issue with rules; people who should have been using rules, weren't well educated enough to quickly create clever rules for team membership. It was simply easier to add an override to a team that added a member to a team than to make sure that you can create a consistent rule for all the team's membership needs. So, as a result, I proposed a new suggestions feature, that would suggest rules to add to a team that would match 100% of the current team membership.

The process of designing, implementing, testing, and deployment was then repeated for the 2nd of my two features. The secondary of my two features was planned to be less deterministic to implement, and focus on setting a clear direction despite ambiguity. In contrast with the first feature I created, the implementation of the second feature was much less straightforward to propose.

There were many roadblocks that I had gone through while making a design document for the rule suggestions feature. For one, there was no existing API for rule suggestion creation, so I was also responsible for making a rule suggestions API that would interface with the database we had used (which is unlike the first feature, where there wasn't necessarily any api changes that needed to be made). There was an older deprecated suggestions

API that used a brute force approach to suggesting rules for teams, but I had to go through a vetting process of determining if that API was even still feasible for use. This meant testing the API on a large range of team sizes, as team sizes could range from the tens to the tens of thousands. After testing the runtimes of this older API, it was clear that this API was not very feasible to be used on a large scale. However, the alternatives to using this older API did not look that great as well. Using a complex machine learning approach was one thing that I had pondered, as this seemed to be the perfect machine learning problem. However, after implementing a demo machine learning algorithm for making a perfect matching rule for a team, there was shown to be a significant amount of server ram and resources used when generating the suggestion. Another potential avenue I had went down was an offline computation of rule suggestions using machine learning. The issue with this approach however came from the feasibility of the feature as a feature with very little use would not be deemed worthy of use at all. This is because of the offline generation, which would not guarantee generated suggestions on page load. Team owners already are spending a short amount of time on the site, so there is no need to try to optimize for a perfect rule with perfect resources allocation if after the API eventually generates a rule suggestion that the user would already be onto another task. Eventually it was clear that the choices left for my implementation were between making good rules (very accurate and will not be outdated quickly), fast rules (very quick to generate), and cheap rules (very easy to generate). I eventually settled on fast, cheap, and fairly good rules. Generally, the user experience is done in a fast manner, and the target audience of these rule suggestions is not one who is experienced in rule making, but it is for one who hasn't been introduced to rules at all, and needs a jump start to becoming a better team owner by using role management. So, I thought that creating the most complex rules was not a priority, and that the real priority was in making good enough rules that can help get the low-hanging fruit of teams in terms of not abiding by recommended security standards.

To fulfill this goal, I went with a compromised algorithm for rule generation. I decided to go with a brute-force based rule suggestion procedure. Using a metrics-focused approach, I determined which few attributes were frequently used by teams, and also what team sizes were commonly used. With this data at hand, I knew I could optimize the brute force approach API to be great at brute forcing extremely common team membership scenarios.

After settling on an algorithm and how I'd use the older not in use (but soon to be!) API, I would end up designing the UX. Even though this system was much more complex to create on the backend, the frontend was far simpler than the frontend for the first feature, as I simply had to create rule suggestions that the user may use.

Implementing the algorithm and the design was a fairly short process with the given template of the last feature I had made in hand. Once that was done, manual testing was done again, and then eventually turned into beta stage testing. After that was all said and done for a week, my feature was pushed to live servers.

4 Significance

The two suggestion features were an integral step in creating an improved user experience. Both of these features streamlined processes on the internal site, greatly improving the usability of the service. On average, just over 18,000 users used any one of these features per day. Additionally, 76% of people who moved from the older service to the newer service (named 'migrators') also had used one of the two features during their migration process. These data points demonstrate the ease of access of these features, which was one of the main motivators for creating these features in the first place. Additionally, with the evident frequency of use by a large portion of the site users, the feature was deemed successful in promoting clearer resolutions of security concerns. Technically speaking, the features that I developed were significant because of their impact on Amazonians' daily routines.

In terms of personal success, the projects were significant in that they displayed many attributes that Amazon had looked for in potential returning candidates. They were able to see my tendency to tinker, in order to fine tune the project that I was given to its max potential. My technical expertise was also shown, as I was able to quickly develop multiple features that were thought to be significant for the team. This success eventually led to a return offer back to Amazon, that even though I did not end up accepting, signified that I was qualified to be a full-time engineer at a serious software development company.

5 Preparation

Even though all my coursework proved useful to me in preparing me for this opportunity, there were a few keys stand out courses that best set myself up for this class. Data Structures and Algorithms I & II helped prepare me for this opportunity as fundamental knowledge of algorithms, complexity analysis, data structure usage, etc. were fundamental in my feature designs and as well as the actual implementation of my features. Software Development Essentials was by far the most helpful course, in terms of preparing myself for Amazon, that I have taken. Version control basics were almost necessary for all development. Knowing the gist of the entirety of the development cycle helped to get me up to speed quickly with day-to-day meetings and stand-ups. Many of the tools used for Java development specifically were frequently used by Amazon backends. Knowledge of the model-view-controller architecture was essential for debriefing the stack that I was a primary contributor to. Outside of things that I was actually responsible for creating, classes like Cloud Computing, Programming Languages for Web Applications, and Database Systems were essential for basic knowledge for the systems that I was using. Knowing how different database systems worked in our specific use case, how the cloud works for general storage, and also basic web programming were all things

that without prior background knowledge would have taken me a far longer time to develop and use on a day-to-day basis.

REFERENCES