Amazon.com Delivery Options: Repaving the Last Mile

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

Jaden Kyler-Wank

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

Advisor

Briana Morrison, Department of Computer Science

Technical Report

Amazon.com Delivery Options: Repaving the Last Mile

CS4991 Capstone Report, 2022

Jaden Kyler-Wank
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
jmk7jw@virginia.edu

Abstract

The Amazon.com checkout page is built on antiquated architecture due to the inefficient and illegible nature of the old codebase. Rebuilding the workflow architecture for the way in which delivery options were retrieved and rendered for the customer through various remote-render proprietary systems provided promising results. By compartmentalizing several operations into two distinct backend components of the new codebase, the process of generating eligible delivery options followed by retrieving the front-end structure for said delivery options was greatly simplified. To accompany the separation of operations, it was imperative that new architecture was written in a malleable fashion, such that the code could easily be modified in the future. The project proved to be a success, as the old delivery option widget was switched with the newly remote-rendered widget without any difference to the end consumer. As the old architecture is phased out, the newly completed codebase can be further expanded upon by other Amazon teams.

1. Introduction

What happens behind the scenes when the checkout page of Amazon.com is loaded? Amazon's proprietary codebase is (understandably) strictly confidential. Prior to my start date, I had no idea what to expect–as an intern, except that I would be on a team that deals with the checkout page of the retail site. My team was a small part of the category known as the "Last Mile" at Amazon—encapsulating any team working on the processes that must occur after a customer has added the desired items to their cart. To my surprise, I was tasked with taking the lead on an effort that had been abandoned the summer prior due to time constraints. From day one it was clear that I was an intern, though I was *not* doing intern work. I was not only told that my project would eventually be on the live website, but also that I was the first person on the team to approach the problem in over a year.

The problem was: How can we fundamentally change the architecture from which delivery options are retrieved and rendered such that it makes the old architecture obsolete? Even further, the nature of the problem required that the newly implemented workflow of rendering delivery options be completely seamless from the eyes of the customer. That is, the *only* thing the end-user should notice would be a decrease in page loading time. The opportunity to build the new workflow from the ground up for a part of the site that millions of people use every day was both a dream come true and a nightmare. The project was as interesting as it was intricate, but from the beginning it was clear that I would need to approach this task with three tenets in mind—operational speed, adaptability, and logical flow.

Amazon teams own certain services that make up the larger retail site, with my team owning what we will call the "Checkout Webapp." To further break down my eventual solution, the workflow of my project can be classified into three parts: retrieval of the proper delivery options, processing delivery option data, and rendering delivery options. Prior to the completion of my project, the retrieval and processing portions of the workflow were done on Amazon's old architecture, known as Gurupa. My solution

began with handling delivery option retrieval in the Checkout Webapp, which required significant design considerations and retooling. The data processing aspect of the project was by far the most complex, involving the integration of an entirely different service. Finally, the processed data was rendered from another service outside of Gurupa and Checkout Webapp such that my team had complete control over what the end user saw.

2. Related Works

In the same vein of the deprecation of the Gurupa architecture, Amazon has made numerous architectural migrations in the past. Prior to Gurupa, Amazon used a platform known as Obidos, which was built in such a way that everything is funneled into one location [1]. Hundreds of engineers working on a single behemoth code base becomes a huge bottleneck for efficiency. The problem of ownership also factored into my project's goal since my team did not actually "own" the code that was eventually rendered for the end user on Gurupa. According to Amazon (2016), the newer architecture migration is an evolution of the motivation behind Gurupa, which was to have an infrastructure in which independently developed modular services can be deployed and managed independently [1].

Quiambao (2021) makes the point that innovation with speed is key at Amazon, along with the ability to grow, two ideals which were hindered by the Odibos architecture [2]. While the migration of Odibos to Gurupa ended around 2007 [1], the elevated focus of modularity and independent ownership of code as well as ease of deployment all factored into my contribution towards the new migration to another architecture.

3. Project Design

In order to comprehensively explain my design approach from start to finish, I have included a greatly simplified diagram of the relevant architecture in Figure 1. From the understanding gained via Figure 1, I will be able to dive deep into each individual section and the design philosophy I had for each one. However, the nature of this paper and my professional relationship to Amazon will prevent me from describing many of the complexities of the project.

3.1 Review of Relevant Architecture

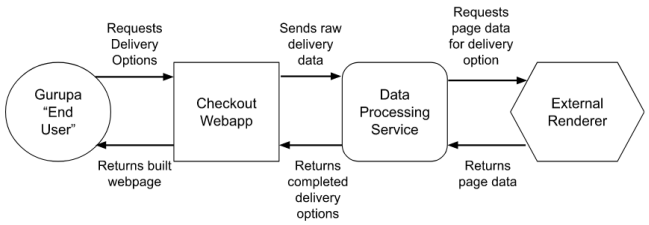

Figure 1: Diagram of Relevant Architecture

Gurupa is synonymous with the final webpage the end user can interact with. Upon the checkout page being loaded, Gurupa will send a request to the Checkout Webapp for several delivery options for the item(s), defined by Amazon's internal algorithm. At this point in the process, the Webapp sends each of the delivery options as information along to the Data Processing Service (DPS). The functionality of the DPS was to essentially convert and bundle the raw delivery option data into a format that would be legible for the External Renderer (ER). The DPS is an essential step due to the necessary types of data needed by the ER, which are different from those sent from the Webapp to the DPS.

Once the processed data reaches the ER, a block of page data recognizable by Gurupa is generated for the delivery option. This data propagates back to the Webapp, which assembles the full page from each block of page data the ER had constructed. The process just described is the final version of the workflow I came up with, which went through many changes during the internship. Prior to my project, the workflow for rendering delivery options only involved Gurupa and the Webapp with different functionalities.

3.2 Checkout Webapp Redesign

The Checkout Webapp is tied to several teams at Amazon and manages the requests of the Checkout page. The purpose of the Webapp itself is to serve as an abstraction layer for Gurupa with the idea of being more modular. However, when I was brought onto the project, there was an absence of integration from the Webapp with the DPS or ER. While the ER technology existed on

its own for a few years, it needed to be connected to the Webapp. To achieve this connection, I had to create a mechanism that could convert sent/received data into a format readable by both the Webapp and the ER.

Additionally, a large amount of setup and initialization were required for the Webapp to even integrate the ER. For these reasons, I implemented a multitude of functions within the Webapp to account for these necessities. The redesign process entailed re-routing much of the existing workflow within the Webapp to work with the new initialization and preparation done for the purpose of communicating with the ER. Following the completion of re-routing the workflow, I was ready to solidify how I implemented the DPS to convert the needed data.

3.3 Data Processing Service (DPS)

The best way to describe the DPS would be as an intermediary between the Webapp and the ER. The Webapp passes along the delivery option data to the DPS that formulates the desired request to the ER, and the DPS just makes that desired request into a format the ER can understand. More specifically, the DPS reads the Webapp's request and inserts the proper parameters that the ER accepts to know exactly what it is rendering. Regarding the DPS's purpose on the opposite flow from the ER to Webapp, it is similar in that the returned data is formatted for the Webapp. The biggest difference between the two directions is that the DPS must convert the response from the ER as a data type that can be stored as render-friendly page data. Once the processed data gets back to the Webapp, most of the complex operations have been performed.

3.4 Linking the External Renderer (ER)

Contrary to most of the other systems present in this project, the ER was designed to have functionality appended to it. Thus, the only design choices I needed to make were strictly on how the delivery options should be rendered. Luckily, the answer to the question of display was quite simple. One of the end goals of the project was for the end user to not notice any visible difference in their checkout experience. This provided me with the ability to simply recreate the HTML page data that had existed already in Webapp into the ER. Then all that needs to be returned to the DPS (and finally the Webapp) is this built page data chunk to be bundled and converted. Additionally, all the page data is owned directly by my team, giving us the ability to change what is displayed at any time without having to go through a lengthy pipeline.

4. Results

The most significant outcome of this project is the creation of new mechanisms that enable people to replicate my project in the future. As the internship was ending, my manager made sure that precedent and modularity were at the forefront of my design process. In terms of setting a streamlined procedure for implementing different checkout widgets into these new systems, I would say the project was successful. While I unfortunately never got to personally move my project into production, through all the testing and feedback I received, the new workflow was working properly. Without a doubt, my project will save my team a vast amount of development time when there is a need to update the end-user's view. Previously whenever a change to the HTML displayed was made, my team needed to go through numerous meetings and synchronizations before the changes ever became exposed. Now the proper team simply owns it.

5. Conclusion

My summer project was not at all what I expected but I loved constructing it anyway. Although I am biased, this project was a success in my team's eyes as well as mine. The system has a lot of potential to be improved upon, which can be worked on in a shorter period than previously. The design experience I gained was invaluable, and I genuinely believe I significantly helped my team.

6. Future Work

There is certainly room for improvement on my implemented solution, specifically with how the DPS processes more complex data. The top priority for my team in the future is to incorporate further logic into the new system such that it can cover all use cases. Furthermore, other teams

within the larger checkout page organization can adopt the rendering technology to decentralize from Gurupa even more so.

7. Acknowledgments

I would like to thank my manager and mentor, Atul Singh, and Yousef Mahmoud, respectively, for their continued support throughout the summer.

References

[1] Aaren Quiambao. Blog | Ex-Amazon IT manager shares how they migrated from a monolith to SOA - Toro Cloud. *Toro Cloud*. Retrieved October 30, 2022 from https://www.torocloud.com/blog/amazon-monolith-to-soa-migration

[2] 2016. Why Amazon Retail Went to a Service Oriented Architecture - High Scalability -. *High Scalability*. Retrieved October 30, 2022 from http://highscalability.com/blog/2016/7/13/why-amazon-retail-went-to-a-service-oriented-architecture.html