Efficiency and Transparency: How a Small Intern Project Can Save Days of Compute Time

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 in Computer Science

Winston Zhang

Spring, 2024

Rosanne Vrugtman, Department of Computer Science

Efficiency and Transparency: How a Small Intern Project Can Save Days of Compute Time

CS4991 Capstone Report, 2023

Winston Zhang
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
wyz5rge@virginia.edu

ABSTRACT

Fannie Mae, a Fortune 500 finance company dealing in the mortgage market, found itself disembarking from a several year-long journey that many tech companies have undertaken: migrating enterprise software assets to the cloud. After this migration, a particular software product that I had worked on the previous year, had difficulty processing uploaded user input, while also offering the user no means to check on the status of their upload. I provided a window of transparency by implementing a new status board on the existing frontend UI page with which users interacted. I utilized a combination of Angular and AWS Lambda to implement this full-stack solution. After deployment to the production environment, my implementation is expected to eliminate a significant source of redundancy in compute time while also minimizing future emails and support tickets from customers. Future work on this project may involve the implementation of a comprehensive suite of unit test cases, configuration of my feature for automatic testing and deployment using CI/CD pipelines, as well as standardization of frontend UI/UX elements in line with company standards.

1. INTRODUCTION

In previous iterations of the Google Chrome web browser, clicking on the taskbar icon would appear to open a window, which then disappeared. Most users would then click dozens more times to no avail, only for dozens of windows to open. My development team faced a problem of a similar nature with the software product we maintained, but with latencies of several days instead of minutes.

The software product is a web application hosted on an internal website that returns data relevant to the loan underwriting process. The application is commonly used by employees in the finance and business divisions. Due to it being designed for users who are not engineers, the application's primary purpose is to provide a user-friendly interface on the front end so that users can query for and access data without needing to know or use SQL. Common use cases saw users enqueueing high-workload jobs, and with most of the logic and computation hidden from view in the backend, as well as neither confirmation nor status update being displayed on the frontend, users would frequently assume that their submission was either not received or rejected due to formatting errors, leading them to either enqueue duplicate jobs, or to email status inquiries to the development team. Duplicate jobs would result in wasted cloud compute time, and thus wasted money, and email inquiries required the development team to pause their work and log into the AWS console to manually check on the status of a user's job, hindering productivity.

2. RELATED WORKS

The software product that I updated for Fannie Mae had been built by my father when he was on my team before I joined the company, so I was already familiar with the application and its uses. I gained more experience with the app in the previous year by supporting my team in its migration from on-premises IT infrastructure to AWS cloud infrastructure. Because of my familiarity with the project, as well as the relatively simple scope of the feature to be implemented, I did not consult any related works, instead opting to mimic on already implemented code to build my feature.

The resources I relied on were mainly tutorials and publicly available documentation for the frameworks and packages that I used in my code. The resources I consulted to become familiar with Angular for the front-end were the official documentation available on the Angular website, as well as a Linkedin Learning course (Angular, n.d.a; Schwarzenberger, 2019). Angular also maintains a UI element module for the framework called Angular Material, whose documentation I consulted as I built my feature's frontend (Angular, n.d.b). My backend used *psycopg2*, a Python package used for interfacing with PostgreSQL databases, whose documentation I also consulted (Di Gregorio et al. n.d.).

3. PROJECT DESIGN

Due to the web application being already deployed to the live production environment, the implementation process of my feature involved interfacing with and building around pre-existing application code. Thus, the set of requirements and constraints was clearly defined, as well as how the implementation would be split between front and backend.

3.1 Review of Application Functionalities

The web application offers two similar functionalities to users. The front end, implemented in Angular, prompts users for input through an HTML form, where loan information would be entered. The frontend extracts values from the HTML UI elements, and passes them to the AWS backend, which parses the input and translates it into a SQL query, fetching the requested data from the database. The backend passes the returned data back to the frontend as a JSON object, whose contents are displayed in the webpage, below the input form. This UI-based use case performs a query for a single loan - a process that takes place almost instantly.

The application also allows users to specify multiple loans at once by typing their input values into a *.dat* file, which they upload to the front end. The backend processes the uploaded files line-by-line and generates a *.dat* output file to be emailed to the user. This upload-based use case, often involving input files with thousands or millions of lines, caused the backend to spend several hours, or even days, processing each file. The problem arose when users erroneously uploaded duplicate files, backing up the job queue, resulting in wasted cloud compute time. Sometimes, a single user clogging up the queue would have a cascading effect, as the processing latency would cause other users to follow suit, uploading duplicate files of their own.

3.2 Requirements

The requirements for my implementation were quite straightforward. Because the application was used internally, and the problem faced by users was already known to my team, I had no need to perform requirements elicitation with any customers or stakeholders. Instead, the requirements were informally explained-to me by my team.

The implied functional requirements are:

1. Users can see processing status of their enqueued jobs

2. Processing status for jobs is displayed in tabular form
3. Status board displays several types of pertinent information in separate columns, such as file name, date uploaded, and processing status
4. Users can filter through status board table by each of the columns, in any combination
5. Users can navigate to status board from already existing menu on frontend portal.

Due to the limited scope of my implementation, no nonfunctional requirements were implied.

The constraints are:
1. Frontend is implemented using the Typescript-based Angular framework, and interfaces with existing frontend code
2. Backend is implemented using Spring Boot in Java (later amended to Python code run on AWS Lambda)

3.3 Key Components

My project implementations were divided into an Angular frontend and Python backend in accordance with the project constraints dictated by pre-existing application code.

3.3.1 Frontend

The frontend of my implementation was implemented as a single Angular component. Angular components consist of a Typescript class, an HTML template, and a CSS stylesheet, closely resembling the Model-View-Controller (MVC) design pattern widely used in web application development, where the class, which handles data and logic, is the model and controller, and the template, which displays data passed from the class, is the view.

To maintain UI consistency with existing components on the frontend, I chose to reuse existing CSS stylesheets in the repository.

The template, used to render the webpage, used HTML elements to populate the user's web browser window. A form was implemented at the top that allowed the user to enter values to filter by, such as a file name to search for, a range of dates, and processing status. The input values given to these HTML elements by the user would be saved to variables defined in the class. Below the input form for filter values was the status board itself. The status board was implemented using an Angular Material Table, more commonly referred to as a "*mat-table*" (Angular, n.d.b).

The class, used to handle data and logic, had several variables defined to store the input values supplied by the user to the template with which they wanted to filter data. The class took in data from the backend, passed in the form of a JSON object, which was stored in an array. The class had several Boolean functions defined to determine if each array element's data properties conformed to the filter conditions set by the user via the template and would exclude elements if they did not conform to all filter conditions. The array of now-filtered data would then be passed to the *mat-table* in the template for rendering.

3.3.2 Backend

Initially, the backend of the entire application ran on Spring Boot, a Java framework. While learning the framework, I had been informed of a sudden plan to re-platform the backend to AWS Lambda. Because of the re-platforming, the backend of my implementation became quite straightforward. A simple Python script was created which made a connection to the relational database hosted on AWS. Using *psycopg2*, it performed an SQL query on the metadata table, retrieving file names, dates of upload, and processing status. Because the filtering logic had already been handled by the frontend, the backend code did not need to perform any further filtering and could simply retrieve the entire table. The query response was then returned as a JSON object, which the frontend would access via URL.

4. ANTICIPATED RESULTS

By the end of my internship, I did not have the time to develop a way for my front and backend implementations to interface with each other, which was the keystone implementation for my feature to be functional. After deployment to production, my implementation is expected to save a substantial amount of cloud compute time and money, as well as man-hours, as users can check on the status of their enqueued jobs, eliminating the need to queue duplicate jobs, or email the development team. Due to the company's confidentiality policy, I cannot provide a figure on how much money, and by extension how much compute time was saved by my implementation. Additional cost savings are expected when changes are implemented to the data query process so that processing jobs can be performed faster.

5. CONCLUSION

My project sought to address a fault in a system easily over-encumbered by the erroneous actions of misinformed users. I addressed this fault by implementing a new component on an existing web application that gave status updates on the jobs uploaded by users, providing transparency to the previously black-box user experience.

My project demonstrates the importance of considering human-computer interaction, as well as the importance of clear and informative feedback to the user in response to their actions. Although my implementation could be seen as a "band-aid" solution to an underlying problem of system throughput, it also exemplifies the importance of agility in the software development process, by addressing faults quickly while a more comprehensive fix is under development.

6. FUTURE WORK

Due to time constraints, I was not able to complete several implementations that were important for my feature to be considered polished. Through manual testing, I was able to verify that both my front and backend implementations were functional, but I could neither create standalone automated unit tests for the front or backend components, nor tests to ensure proper interfacing between the two. Additionally, although the same CSS styling was used, I had implemented my frontend templates using generic HTML elements and did not have the time to change them to proprietary webpage assets normally used for company websites. Future work would involve creating a comprehensive test suite for verification and validation purposes as future features are implemented, as well as revising HTML design to bring my implementation in line with company UI/UX standards.

Other future work on the project would involve integrating this new feature into the new system of automated continuous integration and continuous deployment pipelines, along with the aforementioned test suite. Throughout the course of this project, I was working on another project that involved the re-platforming of CI/CD from Jenkins to Terraform. While my work on this second project involved the migration of another app that my team managed, it is anticipated that my new feature, along with the application, will be migrated in the future, in the same way that all the company's software applications were migrated in waves to AWS in previous years.

7. ACKNOWLEDGMENTS

I thank my manager, Sandeep Nadkarni, as well as the rest of my team, for their help and support over the past two summers. My development as a programmer and engineer would not have been possible without them. I also thank my department director, Jeffrey Penner, and my organization VP, Daniel Seeley. I am beyond thankful to my upper management for providing me with the resources to grow, even paying for my AWS Cloud Practitioner Certification exam.

I thank my fellow FNMA interns, who made my summers go by in a flash, who were some of the best friends I have made, and who made the 9-to-5 feel much shorter than it actually was.

I thank my parents, for being the first programmers I ever knew, for providing me with the computers I grew up playing with, and for providing me with the opportunities and support to get me where I am now.

I thank Anthony Petras, my high school chemistry teacher, and Steven Barber, my high school computer science teacher, for challenging me to strive to better understand the world and its state of being, and to appreciate failure instead of avoiding it, for unforeseen outcomes tell a story as important as the expected.

REFERENCES

Angular. (n.d.). Understanding Angular. Retrieved September 28, 2023, from https://angular.io/guide/understanding-angular-overview

Angular. (n.d.) Angular Material. Retrieved September 28, 2023, from https://material.angular.io/

Schwarzenberger, J. (2019, July 13). Angular Essential Training [Video series]. Linkedin Learning. https://www.linkedin.com/learning/angular-essential-training-2/why-use-angular?

Di Gregorio, F., Varazzo, D., & The Psycopg Team. (n.d.). Psyopg 2.9.8 documentation. Retrieved September 28, 2023, from https://www.psycopg.org/docs/index.html