Service Operation Management: Internal Web Application and API for Service Development and Production Support

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

Matthew Morelli

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

Rosanne Vrugtman, Department of Computer Science

Service Operation Management: Internal Web Application and API for Service Development and Production Support

CS4991 Capstone Report, 2022
Matthew Morelli
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
mjm7ngb@virginia.edu

ABSTRACT

A finance-area product team at a used car retailer and financer based out of Richmond, Virginia decided to improve access and usability for various elusive, confusing, and time-consuming functionalities for service development and production support. As an intern for the company, I achieved this using a custom API and web application that combined said functionalities into a centralized suite of tools. The project, titled Service Operation Management (SOM), was built first as an ASP.NET API and later integrated with an ASP.NET web application. Using React and an internal design library, my mentor and I developed a usable and functional user interface so these tools could be readily available for software developers in the finance area. The functionality of the tools in the SOM comes from both custom company systems and Microsoft Azure, and it has effectively increased the agility and efficiency of the team that developed it. Plans include adding more tools, building out the administration system for customization purposes, as well as rolling the system out to other finance product teams and possibly even teams outside of the finance area.

1 INTRODUCTION

Software engineers often use a variety of tools and technologies provided and accessed through different media and platforms; this can hinder the agility, efficiency, and organization of a product team. Not only does this introduce complexity within a single product team's developer "toolkit" and development stack, but different teams within the same company may leverage wildly different tools and technologies for their own unique purposes.

During my summer 2022 internship this problem was very evident within the finance area product team that I was assigned to work with. The finance team had devised a solution: a centralized hub of tools useful to finance area developers. This hub would be implemented in the form of an application programming interface (API) and a web application, the latter of which was my primary intern project, which we called Service Operation Management (SOM).

2 BACKGROUND

At the retailer, there are systems and other software tools that are developed internally, as well as software development tools and technologies that are external and require purchase of a license for use. Since the company has over 70 product teams working on different software and using different tools, efforts to keep track of them and remember how to use them could become quite convoluted. While both the internal and external tools are well documented, that documentation is usually stored in a GitHub repository (internal) or on a website (external). Finding the tools and reviewing documentation to properly use them wastes lots of time that could be better spent programming and debugging.

3 RELATED WORKS

The SOM was given this name because its purpose is for service development and production support. Information technology service management (ITSM) describes any method that a technology team uses to deliver their software services to end users. This includes the infrastructure, organizational plan, and other activities required to keep their systems working for the customer. The process of ITSM can be influenced by a team's DevOps process as well, which ties together service delivery and agile practices. DevOps has a focus on agility and

collaboration, and the SOM is certainly an ITSM system with DevOps influence, as defined by Atlassian (2022) [1]. Atlassian, a company that creates products for software teams and developers, provides enterprise solutions for ITSM, whereas the SOM is a solution developed internally.

Important decisions had to be made on the form and function of the SOM frontend. In software development, there are many programming languages and frameworks for building beautiful and responsive user interfaces (UI) and user experiences (UX). React is a JavaScript library that has gained popularity over the past decade. Developed by Facebook in 2013, it has been continuously improved up to the present day.

Our team at the retailer likely chose React for similar reasons companies like Netflix, Reddit, Airbnb, and Dropbox choose it for a web application. As Galik (2021) points out, React has essentially automated the rendering process so that less time is spent debugging complex programs, and more time can be spent focusing on the minute details that set a UI apart. Websites with lots of dynamic behavior can take lots of time to render on a screen. React also implements a virtual Document Object Model in the browser so that only components that have changed must be updated, leading to faster render times and smoother motion.

Arguably the most important feature to the retailer is the concept of custom components. In React, one can create reusable components that fit a particular need, creating consistency between the company's many systems. The company I worked with has built a suite of React components that can be used by any team in the department. Beyond these reasons, React is open source with a very large contributing community and a wealth of documentation on the internet, making it quite easy to learn and debug [3].

The SOM also began as a generic API to allow the tools to be accessible to other teams. From my perspective as an intern, API development appeared very common and popular within the retailer. As Gino (2021) describes, modern software development has become very reliant on APIs. APIs create an interface for certain function or data access that exists within an organization's systems. This allows companies to create an environment of functionality that can be leveraged by their own developers as well as those outside of the organization, depending on the permissions for their use. Creating uniform communication channels between external and internal systems, especially for functions or databases that are used in various tasks or systems, gives a company a competitive advantage and helps produce better products for the consumer [2].

4 PROJECT DESIGN

The functionalities that sparked discovery and development of the SOM were related to debugging and testing software for financial data and systems. The financial data that my team works with is in the form of credit applications for pre-approval and pre-qualification on vehicles, and these applications are used for performing hard and soft credit inquiries for customers. They contain a bulk of personal information like age, address, name, the last four digits of a Social Security number, annual salary, and more. This data gets sent to various systems, both internal and external, and must be encrypted before leaving the initial system where the application is submitted and stored in a database. While hackers can steal data within a database, it is far easier to intercept in transit between systems. As it is exposed to many high security threats in transit, it must be encrypted for both purposes. Once it arrives in a system where it must be reviewed by a human or processed in plaintext, it must be decrypted.

4.1 Key Components and Inspiration

For the finance team to improve their existing application submission and data transfer systems, they often need to decrypt encrypted application data, or vice versa, and inspect it. An example of this could be that a submitted credit application caused an error when being processed. Perhaps there is an aspect of the submission interface that led the customer to enter some information incorrectly, or a bug that changed the data into an incorrect format. A software engineer will need to inspect the unencrypted data by copying the encrypted version out of the database, sending it to the company's internal decryption API, and inspecting the response. This API is not intuitive to use, and its use requires a web browser or platform like Postman to send a web request, neither of

which is ideal for the inspection of application data.

The finance team also leverages instances of Azure Service Bus, an enterprise messaging broker, for various systems that communicate. Systems publish and/or subscribe to messages that pass through a queue. If these messages cannot be delivered to a subscribing system or processed, they fall into the dead-letter queue (DLQ). The DLQ holds these messages and they can be requeued, purged from the queue, or their payload can be inspected. All the functionality lies within Microsoft Azure, which can be interacted with via a command line interface or the Azure Portal. The finance team was dissatisfied with the usability of these interfaces and experienced difficulty navigating these tools.

The encryption, decryption, and dead-letter message managing functions are needed often, and often in conjunction with each other. The lead developer on my team built the SOM API that combined these three tools, and planned to extend that into a beautiful web interface with user and team specific customization capabilities.

4.2 Web Application Architecture

The login process and hierarchical structure of the SOM web application was kept fairly basic to allow for ease of use. Since the system is meant for internal use by the retailer's associates, I requested an access key for the company's single sign on and multifactor authentication partner from the identity and access management team. I then used this to integrate the associate login process with the SOM backend code. Once integrated, it was configured to authenticate a user upon accessing any URL associated with the SOM.

As for the hierarchical structure, the decisions my team made were fairly intuitive to anyone familiar with basic web design principles. The home page includes a description of the system's uses. The navigation bar, persisting across all pages in the web application, included two tabs: "Tools" and "Admin." Upon hovering over "Tools," a dropdown menu appears including the tabs "Encryptor," "Decryptor," and "Dead-Letter," which when clicked bring the user to their respective pages. Additional tools added to the SOM will be placed in this dropdown menu. The "Admin" tab, when clicked, brings the user to the admin configuration page. This page currently shows options for adding groups of credit application sample payloads, as well as adding specific instances of application data payloads for use in the encryptor and decryptor tools. It also includes options for adding service bus connection strings to use for accessing their respective dead-letter management functions. Future developments in team-based customization will also be present in the administration page.

4.3 Web Interface Design

The retailer has teams dedicated to developing internal design libraries and frameworks. These are similar to Bootstrap, a popular frontend development framework, but are built with the retailer's branding in mind. The underlying code is largely written in CSS (or variants thereof) and JavaScript.

Since the SOM web application's frontend uses React, my team opted to use the retailer's React-based component library to build out the interface. This accelerated the development process, as I was able to install the internal package into the SOM's dependencies, and access various component libraries through simple JavaScript import statements.

I was able to avoid the work of referencing company branding documentation to write code for stylesheets and React components solely for the SOM. This not only reduced the development time, but created a consistent look and feel across all systems developed by the retailer's various product teams, including applications for web, desktop and mobile.

5 RESULTS

The primary goals and achieved outcomes of this system's development were cohesion and consolidation of finance development tools, eliminating tedious steps from commonly used tools, and increased efficiency in debugging, managing, and developing finance software. Regarding cohesion and consolidation, the SOM now serves as a hub of tools that are commonly used among developers in the finance area. Rather than having tools that reside in Microsoft Azure Cloud, Azure DevOps, and separate internal systems and APIs, they are all accessible to developers and can be leveraged either from the SOM API or the web interface.

Decryption and encryption routines can take a tremendous amount of time. Before the SOM, the user would first need to obtain the SSL certificate that performed the cryptography, which can take between 5 minutes to a few days depending on the current workload and the response time of the associate who has the certificate. Once obtained, the user would need to either build a simple application using pre-built NuGet packages, or use an API to access an existing encryption/decryption application. Building a new application could take 1-3 days, and would be difficult to share with other engineers and difficult to scale due to the variety of systems that the retailer's engineers use. Tapping into an existing application would require tedious 5–30-minute setup to access only the relevant cryptography-related code. With the SOM, this work has been reduced to just obtaining the SSL certificate and adding it to the SOM backend. Once that is done, encryption and decryption time becomes negligible for that certificate, as you simply need to copy a payload of data, and paste it into the SOM web application.

The service bus DLQ manager also reduced time spent on tedious tasks. Viewing dead letter message payloads can be done through Azure Portal or other third-party applications, but our engineers found that many of those applications are paged very small (only showing a small number of messages at a time) and require installation. Purging or requeuing messages on existing solutions is often limited to around 10 messages at a time, and the retailer's retry handling practices require that data be modified in a certain way before a requeue. The SOM effectively shortened time spent managing DLQs as the manager can be accessed via a web browser and there are no limits to the number of messages that can be listed on the page, purged, or requeued at once. The extra modification that needs to be done before requeuing is also handled in the backend. It is difficult to give an estimate of time saved, but it is likely hours over long periods of DLQ management.

6 CONCLUSION

The initial version of the SOM has benefitted the retailer in numerous ways. It successfully centralized a variety of tools to provide finance area software engineers with a single point to access many development and support tools. It also eliminated tedious, mundane, and repetitive tasks, effectively increasing the team's agility, efficiency, and productivity. It has become a commonly used system among finance developers for managing service bus messages between communicating systems, as well as decrypting data for inspection or testing different encryption certificates. The benefits of the SOM are difficult to measure given its short existence, but it has and will continue to ease and enable the work of engineers at the retailer given its great potential for continued development.

7 FUTURE WORK

Existing plans for future work on the SOM include enhancing the admin page to allow for team-based customization and the addition of dashboards for Azure service health and service build and release pipeline statuses. The admin page enhancements would primarily include modifications to the underlying database. This would allow for organization of users by team. These users could then add specific DLQs, certificates, pipelines, and services to their team profile that would only be visible to members of the team(s) responsible for them. Service health notifies developers of issues impacting availability such as errors, planned maintenance, or outages. Having these in a centralized dashboard would give developers more time to respond accordingly. The SOM is also ideal for creating a more readable and customizable build and release pipeline dashboard, which currently exists in Azure DevOps.

REFERENCES

[1] Atlassian. 2022. What is ITSM? A guide to IT service management. Retrieved September 7, 2022 from https://www.atlassian.com/itsm

[2] Gino, I. 2021. Why Almost Every Company Is Now An API Company. (September 2021). Retrieved September 7, 2022 from https://www.forbes.com/sites/forbestechcouncil/2021/09/16/why-almost-every-company-is-now-an-api-company/?sh=26083d001e29

[3] Galik, O. 2021. When And Why You Should Use React. (March 2021). Retrieved September 7, 2022 from https://www.uptech.team/blog/why-use-react