Software Development Internship: Replacing Legacy Software

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

Harshil Pareek

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

Briana Morrison, Department of Computer Science

Software Development Internship: Replacing Legacy Software

CS4991 Capstone Report, 2023

Harshil Pareek
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
hp3be@virginia.edu

ABSTRACT
The main objective during my tenure as a Java Developer Intern at SAIC was to convert legacy software systems into cloud-integrated modern systems to protect critical user information and promote SAIC as a technology-driven and modernized firm. As Java was the required language for development, I chose the Spring Boot Framework as the most optimal codebase for the backend operations and React Typescript for the frontend and UI operations. A major outcome of development was the completion of a fully functional web application that authenticates and authorizes users based on their credentials. One such security measure that was developed was the generation and decryption of JWTs (JSON Web Tokens) in order to distinguish between individual users and allow for SSO (Single Sign-On) processes. Future work needed would be to continue with publication of the website on a web domain using AWS as the application was a prototype running on local computers. In addition, continuous integration would need to be added in order to keep the website functioning bug-free at all times.

1. INTRODUCTION

During the summer of 2022, I was an intern working at SAIC in their headquarters of Reston, VA. My role was to develop a full-stack web application that replaced legacy software with modernized solutions. The overarching problem with this task was finding a service and domain that kept confidential contractor information private and secure. This could not be done through a simple Cloud-based service like Heroku that would host the website, as this would leave many security vulnerabilities and constantly be backed up from multiple requests at once. I immediately understood that this architecture needed to be complex and would require numerous services running simultaneously to prioritize client confidentiality and real-time user interactions without delay.

The next biggest challenge to tackle in the solution would be to discover a method for restricting unauthorized user access. As I learned from my CS 3710 Cybersecurity class at UVA, this should be able to reject DoS, Malware, Phishing, and Spoofing attacks. Using a simple sign-on page that many web applications use would not suffice and would lead to further implications down the line with leaked contractor data. This would require hours of research and an in-depth analysis of secure domains to gain an understanding of how to develop a truly secure web application.

2. RELATED WORKS

There are several tutorial modules available online that go through developing such a service that I developed for my internship, but they all mainly fail to include the relevancy for developing a JWT token.

Singh (2018) configures Spring Security as well as JWT authentication to allow users to register and login to the application. My project utilized Singh's approach to build an Authentication Manager which configured a password encoder and JWT filter, along with other functionality to build the application. In addition, my project also incorporates Singh's algorithm to generate and verify the JWT coming in from requests. This process essentially receives the token from the request, validates it, loads the user associated with the token and then passes it to Spring Security to authorize the said user. My project also borrowed a portion of Singh's design to build a login What differs between my project and his is that I utilize a front-end interacting with the user, while he utilizes Postman to send GET and POST requests to the backend.

Aytin (2022) develops an intuitive React application that incorporates JWT Authentication with login and logout parameters. I took inspiration from Aytin in applying "Axios" API calls to call my backend server. In addition, my project also incorporated Aytin's approach to define private routes with RouteGuard Components for access control management of pages accessing contractor data, as I intended for certain contractors to only access their own data and not modify others. What differs between my project and Aytin's is that they use local storage to store their JWT token, while I take it beyond that and apply the JWT token into various endpoints that users can access, and either be authorized/unauthorized to enter. They also don't have a real server to make requests, so they access a fake API to handle the server side of their demo application. I developed a backend server that handles the security aspect of the token and returns data back to the user on React.

3. PROJECT DESIGN

3.1 Review of System Architecture

The application first needed a functional backend that operates the requests made by the user and processes the data into a database that holds contractor data. To interact with this backend and process the requests, there needed to be an attractive User Interface (UI) that met the demands and expectations of the user, predominantly contractors purchasing the software from SAIC. With the development of the application complete, this service will need to be hosted on a cloud-operated instance for clients to access, which was managed and given security provisions by SAIC outside of my internship. The application will follow a MVCS protocol (Model, View, Controller, Service) within the backend operations that will allow for greater code readability.

3.2 Requirements

The clients that require this web application are contractors assigned to the United States Department of Defense in the Border Control division. The work I was tasked with did not entirely relate to their division as I did not possess a security clearance. Despite this, I needed to develop secure provisions within the application so contractor data can be stored efficiently and readily available when requested. The application needed to function similar to a Microsoft-owned service with the domain of jwt.ms. In addition, the manager overseeing this development required the backend services to be run with Spring Boot, a web framework used to enhance security protocols.

3.3 Back-End Operations

The backend of the web application is coded with the Java Spring Boot framework and contains most of the functionality needed for the clients. The

Spring Boot application was locally run from the 8080 port and was listening to and sending requests from the front-end (Port 3000).

The initial functionality developed was the ability to generate and decode a JWT given parameters from the user via the Frontend. I implemented a JWT Utility package that configured tokens according to various claims hardcoded, as well as an expiration date and HS-256 signature to make the token truly private and encrypted. This token is now returned to the front-end via a POST request to fulfill the generation aspect of the project. The decoding aspect is essentially the same process but reversed with the decrypting of the token to reveal user claims, which is also returned to the front-end via a POST request. A Request Filter is embedded into the Backend which works synchronously to any request being transmitted. Within this, a series of checks to ensure the JWT generated matches the User Details when logged in and authenticated before returning the token back to the front-end. This filter is only applicable to the aspect of the application to view contractor data.

An MVCS architecture was followed throughout development. The models developed consisted of Token, Employee as well as Authentication Request (handles the username and password during login) and Response (handles the token component) models. The views consist of the JWT utility functions that build or deconstruct the token, as well as the security configure protocol that builds the filter, encodes the password, and oversees the authentication process. The controller is simply the file that handles the route management to the front-end, both to receive and transmit data back and forth to the front-end via Port 3000. The service loads a specific user by a username (used during login) and builds granted authorities based on the role assigned within the JWT token.

3.4 Database Service

The database for this application is an in-memory data storage platform called H2. It is embedded within the Spring framework and is configured with the Java programming language. Originally, the plan was to use a cloud-based service like PostgreSQL, but the issue of hosting confidential information online presented itself. For development, it was found to be more optimal and fully functional with the H2 database, especially with the issues my company computer was giving.

The SQL queries were developed and placed in the codebase, varying by the functionality needed for the CRUD application (Create, Read, Update, Delete). The Create functions applied the INSERT INTO statement of the SQL language, the Read functions applied the SELECT FROM statement, the Update functions applied the UPDATE function, and the Delete functions applied the DELETE FROM function. Possible CRUD functions were regarding contractor names, addresses, roles, etc. With the main Contractor table, there exists a supplemental reference table which possesses IDs of Addresses and Phone numbers that were repeated. This allowed for easier table readability and quicker load-up times.

3.5 Front-End Operations

The front-end framework for this web application was React and written in Typescript, a variant to JavaScript that requires defining variable types and greater code readability. The React application was locally run from the 3000

port, and requests for data from the backend (Port 8080) was done through the Axios library.

The UI needed to be simplified and easily updated with future revisions to applied packages. When accessing the page, the user meets a login page where they needed to input a valid email and password to proceed. This would synchronize with SAIC's credential process in the future to authenticate SAIC employees and contractors. There were four accessible tabs on the web application once logged in; two pages to generate and decode a JWT token, a page to view all Contractor data in a table format, and a page to update specific information that would also check to make sure the user possessed authority to access the page (as determined by their Roles in their designated JWT token).

The page to generate a JWT token contained a dynamic form to input email, first and last names, their Role (limited to Admin, Customer, or Other), and other keys and values the user could add as they will. Once the user submitted the form with a submit button, they received a token on the bottom of the page which was acquired from the request being sent to the backend and returned almost instantaneously. The page to decode a JWT token contains a simplified form with an input box for the JWT token. The user would then submit their information and receive their claim on the bottom of the page, which is essentially the opposite result as the generate page.
The contractor table page presents a React Table containing all Contractor data as well as Pie and Bar charts describing the data. The fourth and final page allows the user to edit or delete a contractor entry that they have authority to do so. This is determined by their Role designation in their JWT token assigned earlier in the generation page of the application. Even if the user authenticated, they need to be authorized to enter specific endpoints.

4. RESULTS

The newly-designed system allows contractors to input their required data into a form and save it into a relational database. It also ensures that their information will be kept confidential in the interests of the U.S. Department of Defense, while also being easily accessible for their own records. Mainly, however, the replacement of this new application to legacy software is the most surprising result. What would have taken 5 minutes with the old process would now take less than a minute with this application. The process for a user to be authenticated and authorized to enter an endpoint is instantaneous and only depends on how fast a user inputs the required fields. After speaking with potential clients as well as Project Managers, they stressed how time-consuming the old process of generating JWTs were and how practical and effortless it is now with this newly developed software.

5. CONCLUSION

Largely, this web application built an accessible and seamless approach towards SAIC employees and contractors to input their necessary information and provision it with security measures like a uniquely generated token that proves one's authentication and authorization to enter an endpoint and modify data. It incorporated various technologies and software to come together as an intuitive web application that processes requests sent by the user instantaneously, renders it through a multi-layered database system, and provides inner functionality to decompose and compute a simplified and legitimate solution back to the user.

Working on this project gave me a prolific sense of how working in the Software Engineering industry will operate, with weekly Sprint checks and on-demand learning to generate effective solutions and the ability to debug and decipher through countless error messages. I learned to stay composed and patient in my hunt for a solution, as I understood that I needed to present the most optimal solution to the customers.

6. FUTURE WORK

Future improvements to this application include hosting the site on a public domain for clients to use. This would be embedded into SAIC's client application and would also include SAIC authorization protocols, as previously mentioned with additional login parameters to ensure only SAIC employees can access the site. Hosting this site on a cloud-integrated service would likely cost a minimal amount and could be provisioned with an Elastic Load Balancer to cut costs to the company.

Another possible improvement in the future would be to add more security provisions to the SQL database and eliminate any threat of a SQL Injection Attack. Without these suggested improvements, however, the application I built over the summer was complete and fully functional for early usage and organization of data.

REFERENCES

Aytin, E. (2022, August 7). *JWT Authentication in React*. Permify. Retrieved on April 21, 2023, from https://www.permify.co/post/jwt-authentication-in-react

Singh, R. (2018, February 18). *Spring Boot + Spring Security + JWT + mysql + react full stack polling app - part 2*. CalliCoder. Retrieved on April 21, 2023, from https://www.callicoder.com/spring-boot-spring-security-jwt-mysql-react-app-part-2/.