Modernizing Existing In-Store Legacy Stipulation Document Approval System

CS4991 Capstone Report, 2023

Matthew Y. Samuel
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
ms3wcw@virginia.edu

ABSTRACT
Sales Managers at CarMax, the nation's leading used car retailer, were utilizing a legacy system platform in-stores to perform the final approval of customer-submitted stipulation documents, which was expensive to maintain, inefficient and incompatible with other systems. To modernize this legacy system, I designed the microsite with Figma and developed the microsite and service using the ASP.NET Core back-end web framework and C# programming language, the ReactJS front-end web framework, and the Azure web hosting service. I utilized the agile software development methodology, which entailed iteratively developing the microsite and service in two-week sprints. This new stipulation document approval tool is being used by sales managers at select CarMax test stores. Before releasing the tool to all stores nationwide, a feedback loop will be created so sales managers can provide feedback on the new tool, which will enable CarMax Technology team(s) to iterate on and improve upon the product.

1. INTRODUCTION

Sales Managers at CarMax have been using a legacy system in-stores to perform the final approval of customer-submitted stipulation documents before they are sent to the lender that the customer chooses to obtain their car loan from. This system has been in place since CarMax opened in 1993. Ultimately, this legacy application is still in use today because it still fulfills its intended purpose. However, with how fast business needs, consumer needs, and software are evolving now, maintaining a legacy system platform such as this one is not viable for many reasons.

First, legacy system platforms are expensive to maintain. As legacy systems get older, the amount of technical debt (implied cost incurred when organizations do not fix problems with software that will need to be fixed in the future) increases, which, in turn, causes higher maintenance costs. Additionally, legacy systems are inefficient. Over time, software becomes outdated and slows down since it is no longer receiving frequent updates. These inefficiencies negatively impact employee productivity, which ends up causing a negative experience for the consumer in some way. Finally, legacy systems are not compatible with newer systems. In particular, it is not easy to integrate a legacy system with a newer system, especially if the system was not originally designed to integrate with other pieces of software.

CarMax is facing these issues with the legacy software they use to approve customers' stipulation documents. This is why they have been implementing initiatives to begin modernizing not only this software, but all of their legacy software.

2. RELATED WORKS

Fanelli et al. (2016) developed a framework for modernizing legacy application systems that treats the application code, the information system, and the infrastructure as three different units that can be modernized independently. Their framework can be applied consecutively to one section of code or in parallel to multiple sections of code. The primary programming languages that this modernization framework targets are Java, C#, and C++, and the three phases that this framework consists of are Pre-Processing, Processing, and Post-Processing. After the development of their modernization strategy, Fanelli et al. (2016) conducted a case study involving a large financial services institution. The study's results showed that the institution reduced its costs by 80%, lines of code by 89%, operational complexity by 99%, and their project launch timelines by about 66%.

Chiang and Bayrak (2006) describe a legacy system modernization technique that they implemented enabling organizations to perform business rules extraction from legacy code, the process of understanding and separating the software's business logic from other logic. It also allows these organizations to convert their legacy code into reusable components in a way that conforms to the component interconnection model, which enables all of the components to smoothly communicate with one another. Chiang and Bayrak define reusable components and how the component interconnection model works and then go on to define business rules, as well as a program slicing method that allows organizations to perform business rules extraction. Finally, they provide a brief example of performing program slicing on a program.

3. PROJECT DESIGN

To modernize this legacy system, I utilized various web frameworks, programming languages, and tools to develop a minimum viable product (MVP) that enables sales managers to approve stipulation documents from within an existing site that allows associates to do the initial approval of these documents. This new sales manager experience consists of multiple components: the stipulation card stepper page, the document viewer, and the final submission confirmation modal/popup and submission confirmation page. The MVP consists of a microsite front-end developed using the ReactJS web framework and a microservice on the back-end developed using the ASP.NET Core framework and the C# programming language. The web application is hosted using Microsoft Azure App Services.

The stipulation card stepper page consists of multiple sub-components. First, it contains order details, including the vehicle stock number, the primary buyer's name, the co-buyer's name (if applicable), the store number, and vehicle information, including year, make, and model. Additionally, this page includes one card for each stipulation type (e.g., Proof of Identity, Proof of Income, Proof of Insurance) that can be expanded to show the specific documents required for that stipulation type (e.g., Pay Stub for Proof of Income).

From there, the sales manager can click on a specific document within the stipulation category, which will take them into the document viewer, where they can navigate through the files within that document and verify additional information about the buyer, including, but not limited to, income, contact information, and address information. When the sales manager returns to the stepper page from the document viewer, they can click the "Verify Stipulation" button to confirm that these documents will satisfy the requirements for the particular stipulation type.

Once the sales manager has verified documents for all stipulation types required

for a particular vehicle order, they can click the "Submit to Business Office" button at the bottom of the stepper page, which will prompt them with a popup asking if they are sure they want to submit the order, because after submission they will no longer be able to edit it. If the sales manager clicks "Submit" on this modal, they are routed to a confirmation page, and all the input boxes on the stepper page and document viewer become read-only. There is also a similar process in which the sales manager can click the "Decline" button to decline the order and blacklist the customer.

The front-end of this microsite was developed with the ReactJS framework, which utilizes an extension of the JavaScript programming language called JSX that allows HTML to be embedded within a JavaScript file. Within the front-end of the site, there is a back-end that adheres to the Backend for Frontends (BFF) pattern, which is a variation of the API Gateway pattern. This pattern creates a separate API gateway for each type of client (e.g., web app, mobile app, etc.), which allows for separation of concerns and easier maintenance as the size of the application grows.

The back-end of this microsite was developed with the ASP.NET Core framework, which utilizes the C# programming language. The back-end adheres to the Orchestrator Pattern, which involves the controller, orchestrator, service, and repository layers. When the user interacts with the UI of the site, input is first sent to the controller through the BFF layer. From there, it is passed onto the orchestrator layer through a Data Transfer Object (DTO) that is used to move data between the UI and the API. The orchestrator calls the service layer and passes it data from the DTO, and the service layer contains business logic such as input validation, grabbing model objects from the repository layer, and modifying these objects and sending them back to the repository layer to update records in the database, if necessary. Finally, the repository layer takes in data from the service layer and retrieves and/or updates data from the database.

This web application is hosted on the cloud using Microsoft Azure App Services, where the resources are automatically scaled up/down and the load balancer distributes network traffic among multiple servers, both done to ensure a very high rate of availability for the site.

4. ANTICIPATED RESULTS

This new stipulation document approval tool is being used by sales managers at select CarMax test stores, and although the tool will solve the problems associated with the existing legacy system platform, as an MVP, it is bound to have many issues that need to be ironed out before it is released to all stores nationwide. First, it does not support all the stipulation types that the legacy system supports. Furthermore, it does not support orders that contain co-buyers. As a result, the MVP is estimated to only support 15-35% of orders. This will make it harder to achieve the rapid feedback loop that is supposed to occur after the MVP release.

It is expected that there will also be issues with features on the microsite not working as expected. When testing the new sales manager experience during development, we put a lot of time and effort into end-to-end testing. However, since we were testing using fake orders that we set up, there will inevitably be things we did not account for that will come up once sales managers use the site to approve documents for real orders.

5. CONCLUSION

The modernization of CarMax's legacy stipulation document approval tool is crucial since the current legacy system platform used for document approval is expensive to maintain, inefficient, and incompatible with

other systems. The approval tool MVP involves a modern microsite and microservice that utilizes various frameworks and programming languages including ASP.NET Core, ReactJS, and C#, and is hosted on the Microsoft Azure cloud platform. This tool offers CarMax sales managers a smooth approval experience with features such as the stipulation card stepper page, document viewer, and the final submission confirmation modal/popup and submission confirmation page. This product will be much more easily maintainable and scalable by CarMax. As the tool currently undergoes testing in select CarMax stores, its release nationwide will allow for a more efficient document approval process, both for the customer and for sales managers.

6. FUTURE WORK

Before releasing the new stipulation document approval tool to sales managers all around the United States, the Document Center team at CarMax needs to implement a few additional features as well as bug fixes. First, they will need to add support for additional stipulation types. Additionally, they will need to have the new tool support orders with co-buyer(s) in addition to the primary buyer. Furthermore, the feedback the team receives from the sales managers at the test stores who are currently using the new tool will allow them to fix bugs with the product. Once these features and bug fixes have been implemented, the product will have a much higher adoption rate among sales managers nationwide.

REFERENCES

C. -C. Chiang and C. Bayrak, "Legacy Software Modernization," *2006 IEEE International Conference on Systems, Man and Cybernetics*, Taipei, Taiwan, 2006, pp. 1304-1309, doi: 10.1109/ICSMC.2006.384895.

T. C. Fanelli, S. C. Simons and S. Banerjee, "A Systematic Framework for Modernizing Legacy Application Systems," *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Osaka, Japan, 2016, pp. 678-682, doi: 10.1109/SANER.2016.40.