Using React-Awesome-Query-Builder for Marketing Campaigns

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

Benjamin Ainley
Spring, 2023
Technical Project Team Members
Akhil Chinnakotla
Shafali Gupta
Bruce Nguyen

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

Abstract

The process for filtering out groups of customers to target during a marketing campaign can be tedious work. During my internship at Capital One, my team developed an app to be used by business professionals which would allow for the creation of rulesets that outline desired specifications of customers to be targeted. This app was developed using ReactJS for the frontend. Rulesets are created using the React-Awesome-Query-Builder library and stored in JSONLogic format. We leveraged AWS Lambda as a serverless backend, allowing us to minimize the cost of computing and for automatic scalability. This rules builder application is not yet ready for production, however. Features that would improve ease of use need to be added, including better search functionality through the use of tags. On a larger scale, the application needs user privileges, allowing for different levels of authorization.

1. Introduction

Currently there is no app for business professionals with a streamlined user interface for creating rulesets and processing customer data. A lot of what is done in the line of business I was in at Capital One is done through command line tools, or two separate applications. This rule building app we worked on aims to change that by giving marketers a single interface to work with. Since business rulesets are stored in JSONLogic format, marketers without technical experience may have difficulty understanding and creating these rulesets. Because of this, our app displays rulesets in natural language for easier understanding.

2. Review of Research

Both React-Awesome-Query-Builder and JSONLogic are relatively new so there is very little research on those topics.

Robinson (2020) discussed the benefits and downsides of using a serverless backend such as AWS Lambda. We considered these issues when developing our app and decided the pros outweighed the cons.

Hamdani, et. al., (2022) aim to bridge both ends of GDRP compliance checking by using machine learning. When developing their GUI, they decided to use JSONLogic to serialize obtained rules as a JSON file. This method is similar to our app's use case, and supports our use of JSONLogic for our app.

3. Project Design

Our team's Rule Builder App design process is divided into three main parts: the backend, frontend, and API to link the two. When discussing how we should approach this project, our team agreed that it would be beneficial to begin working on the backend and API first. This would allow us to understand how the data is stored and set up how calls to this data will be processed. This would make it easier for us when designing the user interface since we will know ~~what~~ the format in which the data will be transferred.

3.1 System Architecture

Our team utilized Amazon S3 as a simple way to store our JSON ruleset objects. Amazon S3 is widely used by enterprises because of its scalability, data availability, security, and performance. We felt that S3 fit the use case for our app very well since we simply needed JSON rulesets stored, and decided that using it was the best approach for us. When setting up our backend, we instantiated two S3 buckets, one hosted in the east and the other in the west. We enabled data replication for these buckets, so if an object is added to a bucket in the east, it will also be added to the west bucket, and vice versa. This helps prevent data loss and keeps data more available. If one server goes down,

the data is backed up in the other server, and traffic will be directed there.

To retrieve the data stored in S3 from the website, a user makes an API request. The tool we decided to use as a communicator between the frontend and backend is AWS Lambda, a serverless, event-driven computing service that runs code in response to events. We decided to use AWS Lambda due to cost and efficiency. It is cheap because you only pay for what you use. Price is based on the number of requests and the time of execution. The developer also does not have to worry about provisioning resources, improving productivity, and scaling, all of which are handled automatically (Robinson 2020). Because we will be handling millions of customer records, our computation needs to be scalable and cheap, making AWS Lambda a good choice for the project. The Lambda function would receive HTTP requests, and perform different actions based on the type of request. The main type of requests required for our app are GET, POST, PUT, DELETE, and a modified GET (multi-get). Based on what request the Lambda receives, it will perform that action on the S3 bucket to get, post, put, or delete a ruleset stored.

The frontend of our app needs a way to send an HTTP request to our Lambda function. To make this happen, we set up two Application Load Balancers (ALB), one server in the west and the other in east. These load balancers act a point of contact for the frontend, and route HTTP requests to the Lambda function. Postman, an API platform, was used to test our API calls to ensure that we could make requests through the ALB to the Lambda functions. This allowed us to ensure that our Lambda functions were behaving as expected.

Once we verified that our API was working correctly with our database, we began working on the frontend. We created the frontend using ReactJS which is responsible for sending HTTP requests to our ALB, which will route it to the Lambda and then perform the corresponding function on the S3 bucket. This update is then made visible to the user through the webpage.

3.2 Website Requirements

The website my team worked on aims to create a streamlined interface for marketers to build rulesets for marketing campaigns. In addition to just creating rules, marketers should be able to view, update, and delete these rulesets. While building or viewing them, marketers should also be able to input a customer's record in order to test the results of the ruleset they created.

3.3 Key Components of Website

Our frontend's main functions involved the four basic CRUD functions: create, read, update, and delete, which correspond with the Lambda functions previously mentioned. Depending on which button is pressed, the corresponding HTTP request will be sent to our ALB. The website's interface largely consisted of a button to create rulesets and a table with rows containing a ruleset name and buttons for viewing, updating, and deleting that ruleset. In addition to those main functions, our website also includes a search bar to make it easier for marketers to look up rulesets, and another button to validate a customer record against a ruleset.

Figure 1: User Interface

When a button such as "create" is pressed, a pop-up implementing React-

Awesome-Query-Builder appears on screen. The user can now customize their ruleset by adding different fields and/or groups of fields. React-Awesome-Query-Builder is highly configurable, and what types of rules are available is determined by the way the config file is set up. This allows the project to be flexible and allows React-Awesome-Query-Builder to be used for other types of projects with different use cases. When the user is finished building and naming their ruleset, they can save it and the query will be stored in JSON format and saved in the S3 bucket.

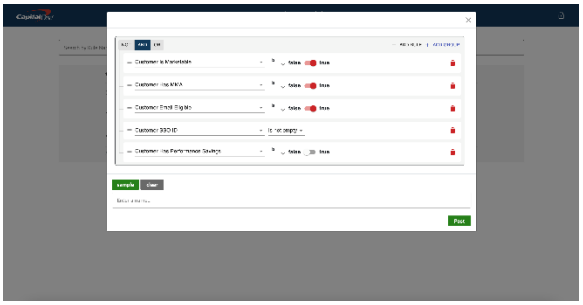

Figure 2: Create Ruleset Pop-up

When the validate button is clicked, a pop-up of the ruleset the user wishes to use appears. In a text area, the marketer can paste in a customer's record and compare it against the ruleset. If the customer passes the validation, the screen would display that the customer record passes; otherwise if it fails. The marketer is also able to edit the ruleset on this screen. While validating a customer's record, the marketer may want to make some changes to the ruleset and this allows them to do so. This validation process is made easier thanks to JSONLogic's apply() function. We are able to easily take a rule and apply it to a customer's record, which are both stored in JSON format.

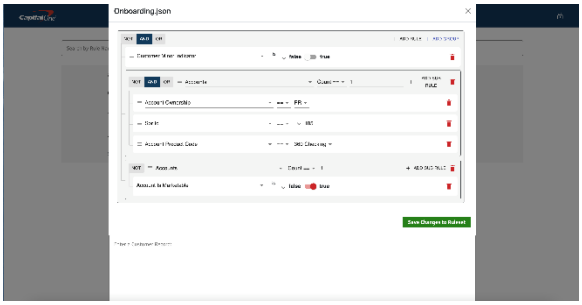

Figure 3: Validation Screen

3.4 Challenges

The greatest challenge we ran into while developing our app involved our continuous integration and continuous deployment pipeline. Capital One uses a CI/CD pipeline called OnePipeline where our code is built and run through tests. If the tests pass, our app is then deployed. Although our test cases were passing, the pipeline failed due to the version of ReactJS not being up to date. However, we needed to use this version of ReactJS when using React-Awesome-Query-Builder, and this caused difficulty when trying to deploy our app. We reached out to a number of Capital One employees who were in charge of OnePipeline, but our issue was never resolved. Due to this we had to host our app locally.

4. Results

The rules builder app that my team worked on was built from scratch and aims to provide something completely new for the company. Because of this our app was never pushed to production but is still in the development stages since there are additional features that need to be fleshed out. However, after the end of my internship, we onboarded my manager's team and they picked up where we left off and are continuing to further develop our app. As of right now, they have built their first official rule for the company. The plan now is to enable a UI for the marketers in quarter two of 2023.

5. Conclusion

This project has potential to make an impact on the marketing line of business at Capital One by making the process of building marketing campaign rulesets simpler for business professionals. Currently the process of building these rulesets and filtering customers can be tedious work. There is not an app that offers a streamlined interface for this task, and much of the work is done through command line tools or two separate applications. These tools are not always intuitive and can be difficult to understand if marketers do not have a lot of technical knowledge regarding query languages. This app provides a single interface that makes this whole process simpler. A business professional can now create rulesets, view existing ones, edit them, and run customer records against these rulesets all in one place. The interface uses a React component which displays queries in a more natural language, allowing for an easier understanding by business professionals. With this app marketers won't have to worry as much with the technical aspect of campaigns and can focus on the business side.

6. Future Works

The app is still in the early phases of development and still needs additional features before it is ready for production. There are a few different directions the application could head, which has not been fully determined yet. A feature that could be added is a more advanced search functionality which would improve ease of use through the use of tags. Another small feature that could be added is the ability to clone a ruleset and modify it. On a larger scale, the application needs user privileges. Right now, there is no functionality of signing in. This needs to be added with different levels of authorization in order to increase security. The app also only processes one customer record at a time. To improve upon this, we plan to add the ability to import large batches of records to be processed.

References

Robinson, D. (2020) *Serverless: Weighing up the pros and cons for enterprises*, *ComputerWeekly.com*. Available at: https://www.computerweekly.com/feature/Serverless-Weighing-up-the-pros-and-cons-for-enterprises

Hamdani, R.E. *et al.* (2021) "A combined rule-based and Machine Learning Approach for automated GDPR compliance checking," *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Law* [Preprint]. Available at: https://doi.org/10.1145/3462757.3466081