Work Based Project Summary: Creating an SLO Generator UI with

React and Backstage

## Eleanor Green

## **[Contents](#_rs71yeyvaw9o)** [**2**](#_rs71yeyvaw9o)

[**Summary**](#_6fvwpgsftqju) **3**

[**Introduction**](#_mkafe6uxlnqi) **4**

[**Project Scope**](#_dqdui3dwktyo)

(Complete) **4**

[**Project Plan**](#_d8oqd3bnpwxz)

(Complete) **5**

[**Consideration of legislation, regulation, industry and organisational policies, procedures and requirements**](#_q4ps499aonuq)

(B2, B3, S10, K6) **6**

[**Analysis and problem solving**](#_inyzbreuxjuv)

(B2, S6) **7**

[**Research and findings**](#_im62wov1z68v)

(B2, B3, S7) **7**

[**Project outcomes**](#_ky6953lf7c9x) **(**K2, K6) **8**

[**Planning and Analysis**](#_byvbeyv2wqw7)

(B2, S11, S12, K11) **8**

[**Design**](#_379ayylqxgyv)

(S12, S16, K9, K11) **10**

[**Implementatio**](#_8jxh3gihdr5a)**n**

(S1, S7, S12, S16, K9) **11**

[**Testing**](#_w6pnt8utz3xg)

(S4, S6) **13**

[**Deployment**](#_ysbad19wcz9f)

(B2, B3, S10) **14**

[**Maintenance**](#_r48p4s92ew8z)

(B3, S1, S7, S10) **1**[**5**](https://docs.google.com/document/d/1wfamzA98DjwBS3g6FiW7LBtdj0YppyBqR75jiyHK5Mo/edit#heading=h.r48p4s92ew8z)

[**Recommendations and conclusions**](#_onj3c17l8oef) **1**[**5**](https://docs.google.com/document/d/1wfamzA98DjwBS3g6FiW7LBtdj0YppyBqR75jiyHK5Mo/edit#heading=h.nu18lv9wu6zu)

[**Appendix**](https://docs.google.com/document/d/1wfamzA98DjwBS3g6FiW7LBtdj0YppyBqR75jiyHK5Mo/edit#heading=h.7j1zhwf73trl) **1**[**6**](https://docs.google.com/document/d/1wfamzA98DjwBS3g6FiW7LBtdj0YppyBqR75jiyHK5Mo/edit#heading=h.7j1zhwf73trl)

[Appendix A - Project Plan](#_d8oqd3bnpwxz) 5

[Appendix B - Extra Resources Used](#_huw8bxeszqpr) 16

[Appendix C - Artefacts](#_du3ia4ozem8r) 1[7](https://docs.google.com/document/d/1wfamzA98DjwBS3g6FiW7LBtdj0YppyBqR75jiyHK5Mo/edit#heading=h.vfts02f2ac9m)

[Declaration](https://docs.google.com/document/d/1wfamzA98DjwBS3g6FiW7LBtdj0YppyBqR75jiyHK5Mo/edit#heading=h.wquty8nu14ui) 25

## 

# Summary

Proposal

Over a 9-week period, I will develop a new UI component for my team’s product; The SLO (Service Level Objective) Framework. This backend service is responsible for generating SLOs for various internal applications within the Expedia Group. The project leverages modern web technologies, utilising Typescript as the front-end framework and hosting the UI with Backstage.

SLOs are essential for monitoring and ensuring the reliability and availability of critical services, making this project crucial for optimising internal operations, monitoring downtime and maintaining a high level of service quality. To enhance the current UI’s utility, I will be building an “SLO Generator” accordion. This will allow us to increase SLO adoption across Expedia Group, whilst also having all SLO tools in one central place.

Project Areas Include

* Building an intuitive and user-friendly interface for defining and monitoring SLOs.
* Integration with Backstage: The UI will be a plugin for Backstage, providing a consistent and centralised environment for SLO management.
* Establish seamless communication between the UI and the backend service responsible for SLO generation. This integration will enable data retrieval and ensure the UI reflects the most up-to-date SLO information.
* Adding to any existing developer documentation for future maintenance and creating user guides with links to FAQs or Slack channels to help with any issues they experience. There is a need to provide comprehensive documentation for end-users and developers, making it easy for stakeholders to understand and utilise the new UI.

Scope

Some of the technologies I will be using within this project will be:

* Java – I will need to make calls to the SLO Reader API (a piece of the SLO Framework).
* Figma – Figma is a design and prototyping tool that allows designers and developers to create user interfaces and visual designs. I will employ Figma to create wireframes, incorporating Material UI assets, which will then be shared with my team for feedback.
* Constructing React Components with Material UI/Backstage Default UI Theming – React allows developers to build reusable UI components. This will allow me to construct a responsive page comprising scalable components capable of accommodating SLO data for customers’ applications. Implementing Logic using Typescript – TypeScript, a statically typed superset of JavaScript, introduces strong typing to the language. This will be useful during local testing, as it significantly enhances the likelihood of detecting type-related errors, especially when interacting with databases.

# Introduction

This project takes place within the SLO Framework Team, in the Performance Reliability Engineering (PRE) department of the Expedia Group (EG).

“Service Level Objectives” or the SLO Framework, is an internal framework used to help teams throughout EG monitor how well their products perform. Our current team has a line manager, a product manager, two senior engineers, two level II engineers, one level I engineer, and myself.

Each validated application at EG has a backstage page. Within this Backstage page is the “Reliability Tab” where application owners can find useful information about the status of their applications. This tab was in need of a re-design and some additional utility. My project is a part of an overarching initiative to increase SLO adoption across Expedia Group, by updating the overall appearance of the Reliability Tab, and adding new useful features.

## Project Scope

In this project, my main responsibility will be to design and implement a form that will be used to Generate SLOs for customers’ applications. This form will render inside an accordion component underneath existing components within the Reliability Tab.

The process started with our product manager, who gathered requirements and wants from the product team. This information was then presented in a meeting within our team where we discussed the features, if we could implement them, how we would implement them, LODO work that would need to be done once the requirements had been delivered, and a realistic timeframe that we could deliver in.

Following that initial meeting, we had refinement sessions where we would split the discussed work into tickets and make sure that the tickets were readable and contained all the information required for any DEV to pick up and complete.

The SLO adoption rate is our main measure for success; we have now onboarded most teams’ applications with SLOs, but there are still some remaining. The SLO Generator form should allow the generation of SLOs to be more accessible, allowing for those last few applications to be onboarded.

Other ways success could be measured are:

* The number of visits made to the Reliability Tab each day after the changes are deployed – Has this form increased the usefulness of the Reliability Tab?
* The amount of customer questions we would get in our Slack channel – Is the UI clear and easy to use? Are there bugs or issues on the user’s side?
* The amount of error alerts we get in our team chat – Are the changes causing bugs? Are the customers entering data into the form that our backend cannot appropriately handle?

Things that are out of scope for the project are:

* Any low priority LODO work. This would include things like refactoring code, however, if bugs were found, then this would be expected to be fixed whilst working on the project.
* Any work that relates to other sections of the Reliability tab that are not the Generator Form section.

## Project Plan

|  |  |
| --- | --- |
| Role | Responsibilities |
| Program Manager/ Scrum Master | *Organises meetings, assigns tickets and removes any blockers.* |
| SDE II | *Contribute to design discussions and assist with implementation and debugging. Review PRs and offer guidance concerning backend functionality.* |
| SDE I | *Reviewing PRs and offering guidance concerning frontend functionality, and implementation of existing Reliability Tab features.* |

Communication and Collaboration

The requirements and wants were documented and expanded upon on a confluence page. Tickets are documented and maintained on Jira. Ticket refinement sessions will make sure that the tickets are still needed, if any further work needs to be done before starting the ticket and that the work is still within the scope of the initiative. Collaboration can happen in the office or via Slack if WFH. If bigger things need to be discussed, sessions can be set up in meeting rooms or over zoom, and then documented in a ticket or a confluence page.

## Consideration of legislation, regulation, industry and organisational policies, procedures and requirements

### (KSBs: B2, B3, S10, K6)

Data Minimization:

* Only the necessary data for the SLO generation is collected. (B3)

Authentication, Authorisation and Employee Access Control:

* This UI will only be accessible through Expedia Group’s (this also includes Ips inside EG such as VRBO) instance of Backstage. This is only accessible through an SSO whilst connected to a VPN, which will require the correct levels of access within EG. (B3)

Accessibility Compliance (e.g., WCAG):

* Making sure that the components and elements inside the parent accordion component have the correct labels, are inside the correct form groups and are aptly named. (B2)

Code Review and Approval(B3, S10, K6):

* Code will not be approved unless it has an approving review from another DEV in my team.
* Changes will not be deployed unless they adhere to the standards set by the EG-Reliability-Tab team.
* Changes will not be deployed unless they adhere to the standards set by the EG-Backstage team.

Documentation:

* Any changes in functionality to the SLO Generator or other methods of SLO generation should be added to the documentation when deployed. This includes any FAQs related to such changes. (B2, K6)
* Process for setting up a local development environment should also be added to maintainers documentation. (B2)

## 

## Analysis and problem solving

### (B2, S6)

Some of the potential challenges were identified, concerning local development:

* Will only be able to use Safari to test our live endpoint as we would need to be able to disable CORS for the whole page and changes to fully render.
* Setting up a local development environment would be challenging as it wasn’t fully documented how it was done previously. It would also be different from dev to dev as some of us had intel based MacBooks and others had Apple Silica MacBooks. This would cause different issues during set up.
* Not able to see how the actual page looked because running it locally would always be different to how it looked when it was finally deployed. This would present difficulties when correcting spacing and broken rendering on some components.

Some ways to overcome some of the aforementioned challenges could be to:

* Have Safari downloaded and set up. Make sure to always adjust the dev tools when first opening the browser so that CORS would not be a problem. (B2)
* Document every step I took to set up my MacBook and then add it to the documentation. Also note down the weirder errors, what caused them and how to solve them. (B2)
* Ask colleagues to check my changes on their local machines to try and account for anything that may not have shown up on my machine. If issues cropped up when deployed, figure out what could be causing the issue and deploy it alongside any further changes. (B2, S6)

## Research and Findings

### (B2, B3, S7)

In order to get a better understanding of the frontend components that I was using and their features, I used the Material UI docs. I had to use the docs for the MUI Core, MUI API and MUI System components as some of the features that were used in this project cme from different parts of the MUI framework. I always had these open when working on any part of this project. It was also useful to have a description and example of how the component is supposed to be used and how it should be labelled for better accessibility.

# Project outcomes (K2, K6, K11)

Our works using the agile methodology:

* Planning and Analysis
* Design
* Implementation
* Testing
* Deployment
* Maintenance

This project is a part of a larger, constantly evolving product, so we will repeat this cycle many times even after my project is complete.

* When new user requirements are put forward, we will start with planning meetings, where the product manager will explain what new functionality we need to implement. The output of these meetings are documented on a Confluence page within the SLO Framework Team’s documents.
* These features will be turned into a proof of concept (POC) which took the form of wireframes for this project. These will be referred to when implementing logic and testing to make sure that the components still adhere to the requirements and functionality.
* Using the aforementioned wireframes and confluence pages, the product will be built using existing components or creating new ones if needed.
* Using our use requirements and use cases that were written off the back of those requirements, the product will be tested end-to-end.
* After passing in-team checks, as well as checks from depending teams, a series of PRs across multiple repos will be raised. Once approved, the changes will need to be pushed through a Spinnaker pipeline and monitored with a manual judgement section to make sure that each step of deployment is completed successfully.
* After the product is deployed, bugs, errors and missed use cases will always crop up. This will be taken on using our LODO initiative and will be put into tickets, in order to maintain our service.

## Planning and Analysis (B2, S11, S12, K11)

Object-Oriented Programming (OOP) and Event-Driven Programming are the main programming models present in this project. The Generator form is a custom React component and React is a JavaScript library that employs both OOP and Event-Driven concepts to build interactive user interfaces efficiently. When building UIs with React, modularity is encouraged so that single components and functions can be reused. This means that React components encapsulate both its own state and its behaviour.

The main page that the Generator accordion is rendered on will be checking the state of the user’s application’s SLOs. The Generator accordion will then change it’s behaviour depending on that state.

The following conditions will affect how the accordion renders:

* The user has existing SLOs that are properly configured.
* The user has existing SLOs that need to be reconfigured.
* The user has existing SLOs that cannot be found.
* The user has no existing SLOs.

The accordion can render in the following ways:

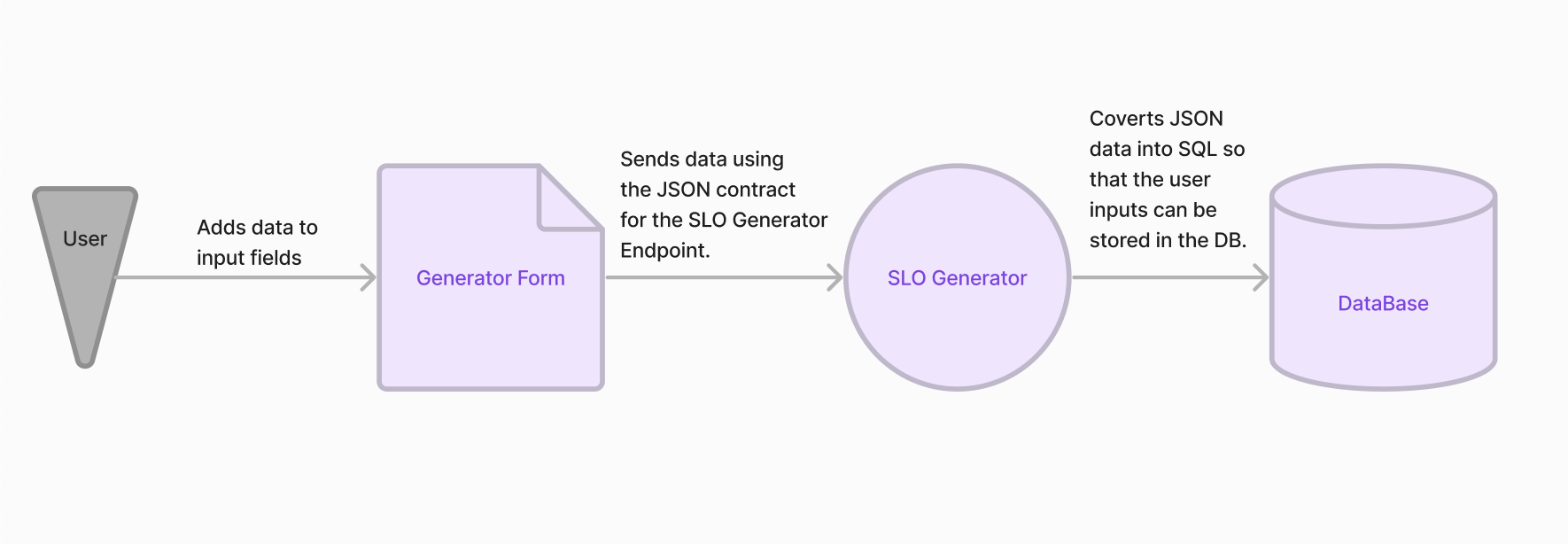
* If SLOs exist and are properly configured, the accordion will render at the bottom of the page with no additional message and be closed by default.
* If SLOs exist but need to be reconfigured, the accordion will render at the bottom of the page, but a tooltip will appear in the recommendations accordion further up on the page. When clicked on, this tooltip will scroll the page down to the Generator accordion and open it to be used.
* If SLOs cannot be found or the user has no existing SLOs, the Generator will be the only accordion that renders on the page, and a card will appear next to it prompting the users to “sync” their SLOs if they’ve recently generated them or generate some SLOs through the adjacent SLO Generator form.

This functionality allows the Generator Form component to be its own object that can be rendered and reused as many times as it needs to be. This also leaves room for the form to be used again elsewhere if features are added, promoting modularity and reusability, essential tenets of OOP.

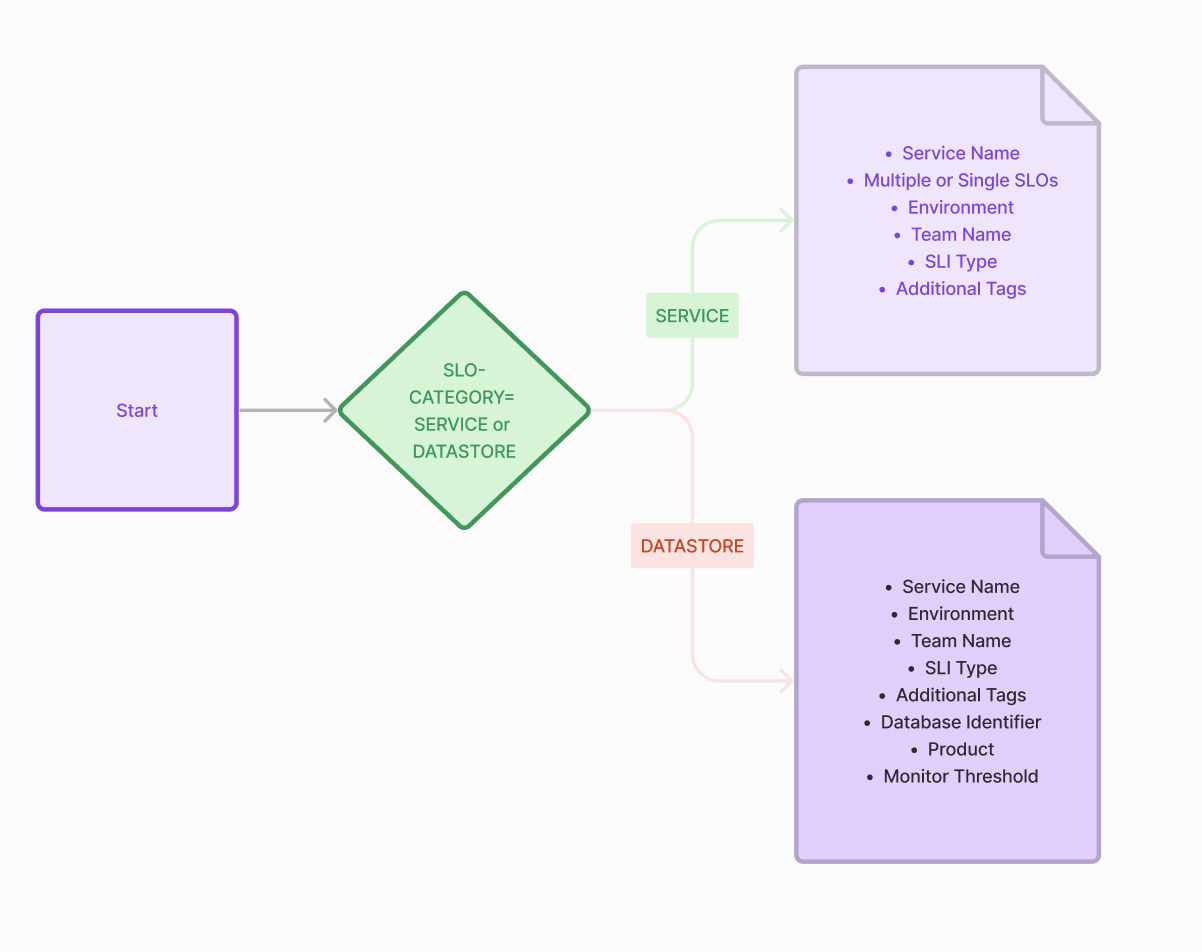
The aforementioned conditions for rendering the generator form also describe an event-driven approach. The rendering of the Generator form is triggered by events related to the customer’s application’s status and SLO existence. The Generator form’s visibility and content are event-driven, responding dynamically to the state of the application. This aligns with the event-driven model, where actions or changes in one part of the system trigger responses in other parts.

## 

## Design ( S12, S16, K9, K11)

The above diagram shows a basic flow for the Generator Form:

* The user enters the data in the form required to generate an SLO for their application.
* The data is sent as a JSON request body, maintaining the expected data types and order (JSON contract).
* The JSON is sent to the slo-generator which will use the data in the JSON to generate an SLO. The SLO is saved to the DB.

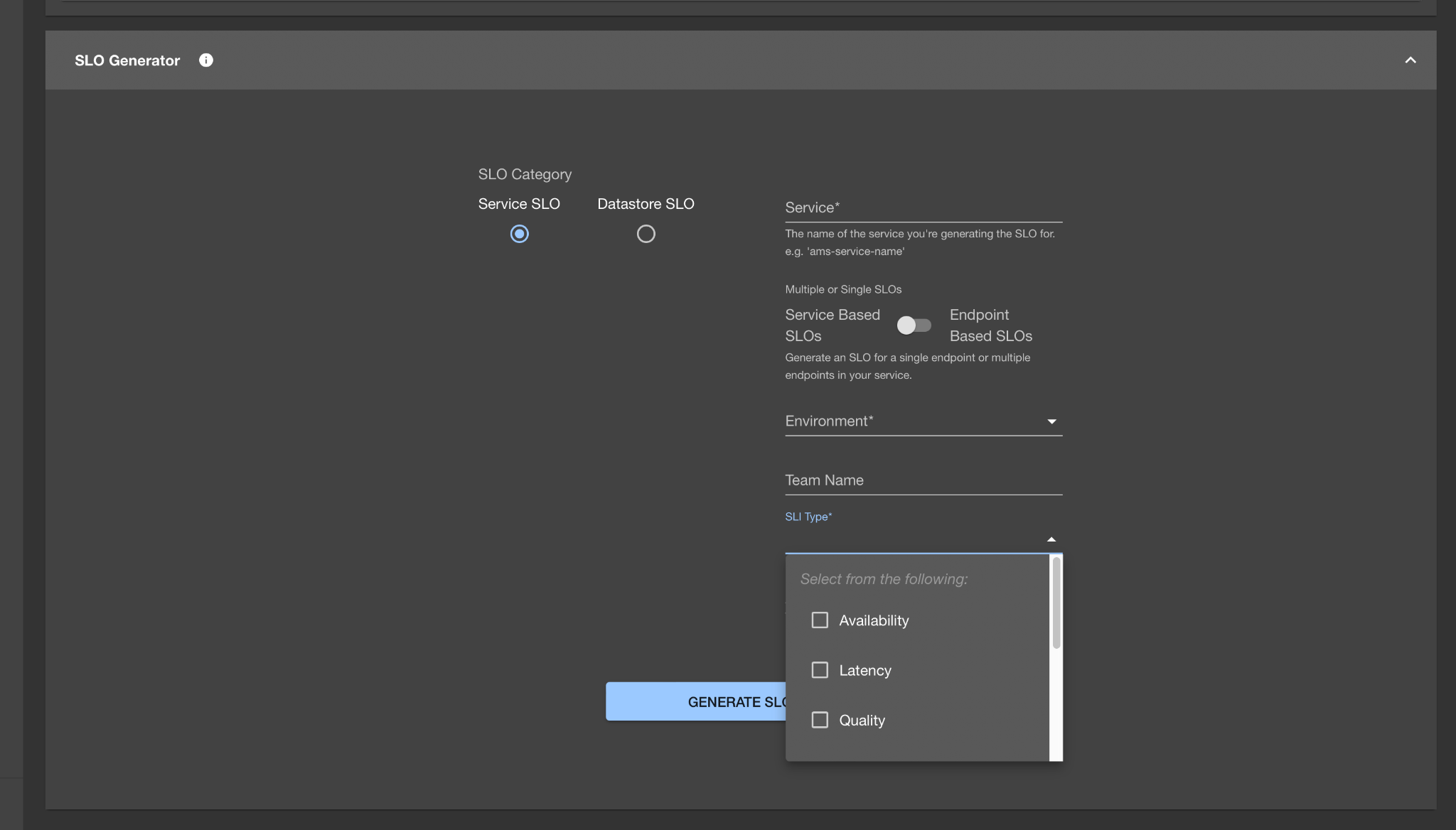


The above diagram shows the top part of the decision tree for the logic of the SLO Generator (a separate project to the Generator Form and Generator accordion). The decisions made in the SLO Generator will feedback to the Generator form, changing the state of certain components within the accordion and form.

The first decision in the decision tree is SLO-CATEGORY. This category can be either SERVICE or DATASTORE. A SERVICE SLO, is an application or product that executes tasks, and a DATASTORE SLO is an application or product that stores data or acts as a database. If the user selects a certain category, the form will change the input fields that are rendered. This makes it so that the correct data is sent back to the SLO Generator, and the JSON contract is not broken. (K9)

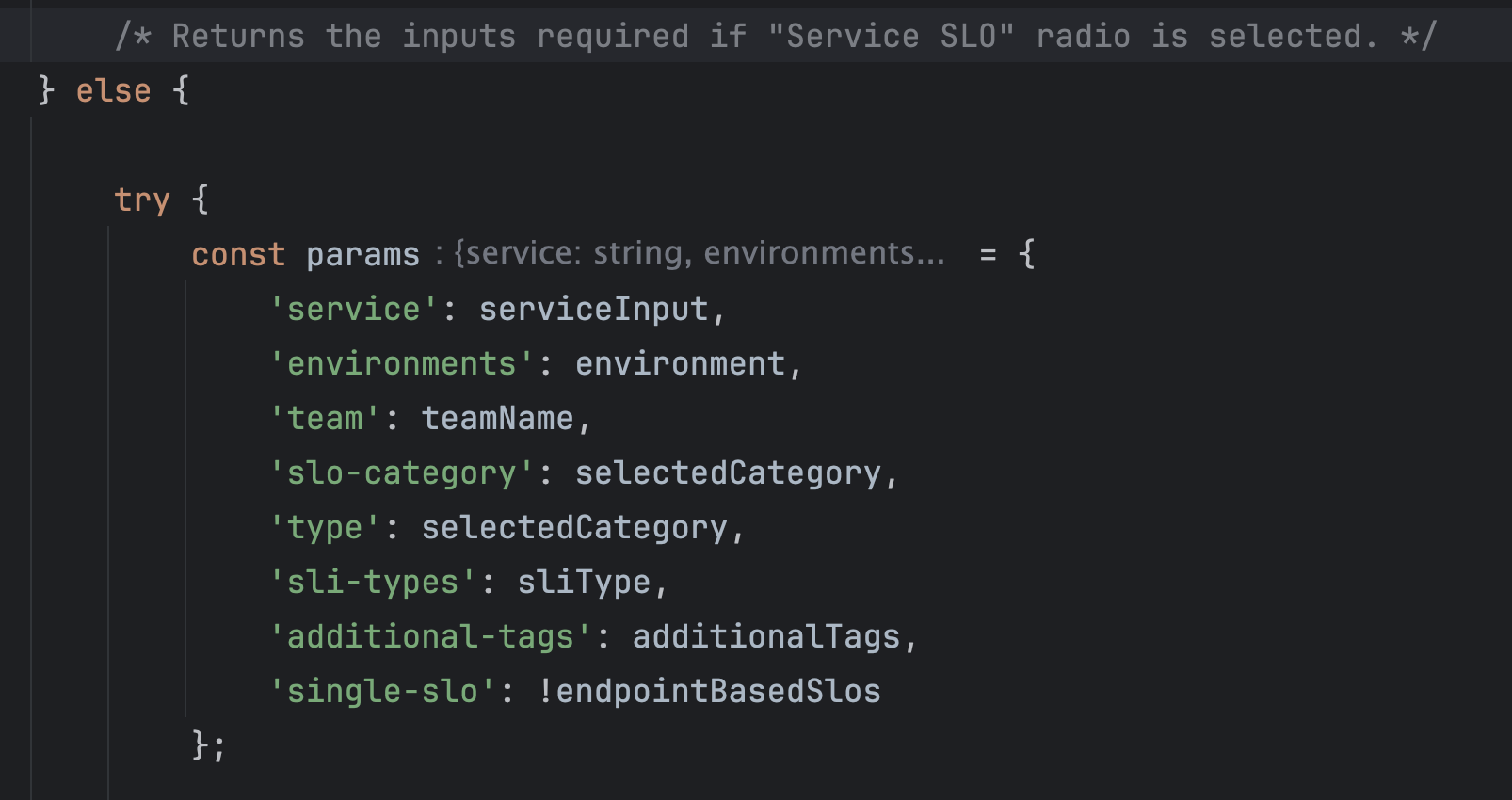
The State Method design pattern manifests in how the generator accordion adapts its appearance and behaviour according to the changing conditions. In this pattern, an object changes its behaviour when its internal state changes. In the React context, components frequently use local state to manage such variations, dynamically updating their rendering based on changes in state.

## Implementation (S1, S7, S12, S16, K9)



Above is an excerpt from the overall GeneratorForm component. This part of the component is for the category options “SERVICE” or “DATASTORE” that will determine the type of SLO that will be generated for the user. The rendered component looks like this:

* Each input field is wrapped within a <FormControl> component. This is a custom Material UI component that allows the components within to relate to one another. E.g. label “SLO Category” will be automatically attached and related to the two radio buttons beneath it. This is useful for adjusting layouts and making sure that components are visually related to one another. More importantly, it allows screen readers to associate certain sections of the page with the options available, helping my code to adhere to accessibility standards.
* The <RadioGroup> component also does something similar, but is specifically for radio buttons. It has built in styling for labels, so that you can make sure that the label is visually sitting in the correct place relative to the button.
* <Stack> Is another MUI component that acts as a container with built in styling, that forces components to stack on top of one another. Here, I have change the direction to “row” so that the components are forced to sit side-by-side instead. <Stack> was necessary to use as it was the only way for me to get the buttons to sit next to eachother without the labels breaking or disappearing.
* <FormControlLabel> is a versatile component that will render an input element with a label automatically attached to it. This component will change depending on the “control={<(component)/>}” value that you give it. I gave it “control={<Radio/>}” which caused it to render itself as a label with a radio button underneath. This component is why I had to use <Stack> instead of just rendering it alone with some labels inside the <RadioGroup> container. The buttons were stacking on top of one another, and when I showed this to my team for feedback, it was deemed as a little bit unclear; they wanted the layout of the buttons to look different to the rest of the form so that it stood out as the first set of options that should be filled in.
* Within the <FormControlLabel>s, there are onChange functions that can be used to determine what should happen when the state of the radio button changes.

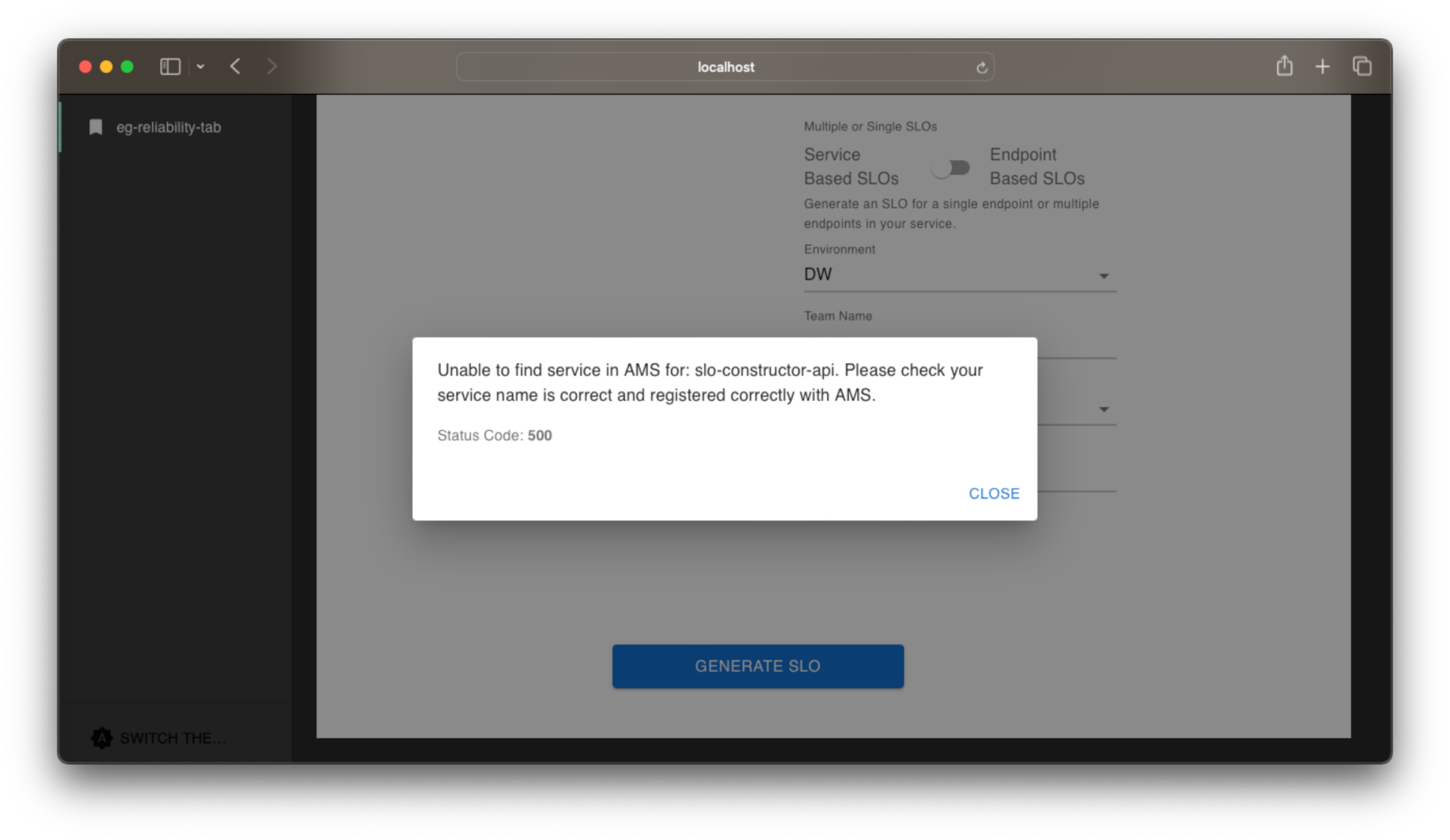


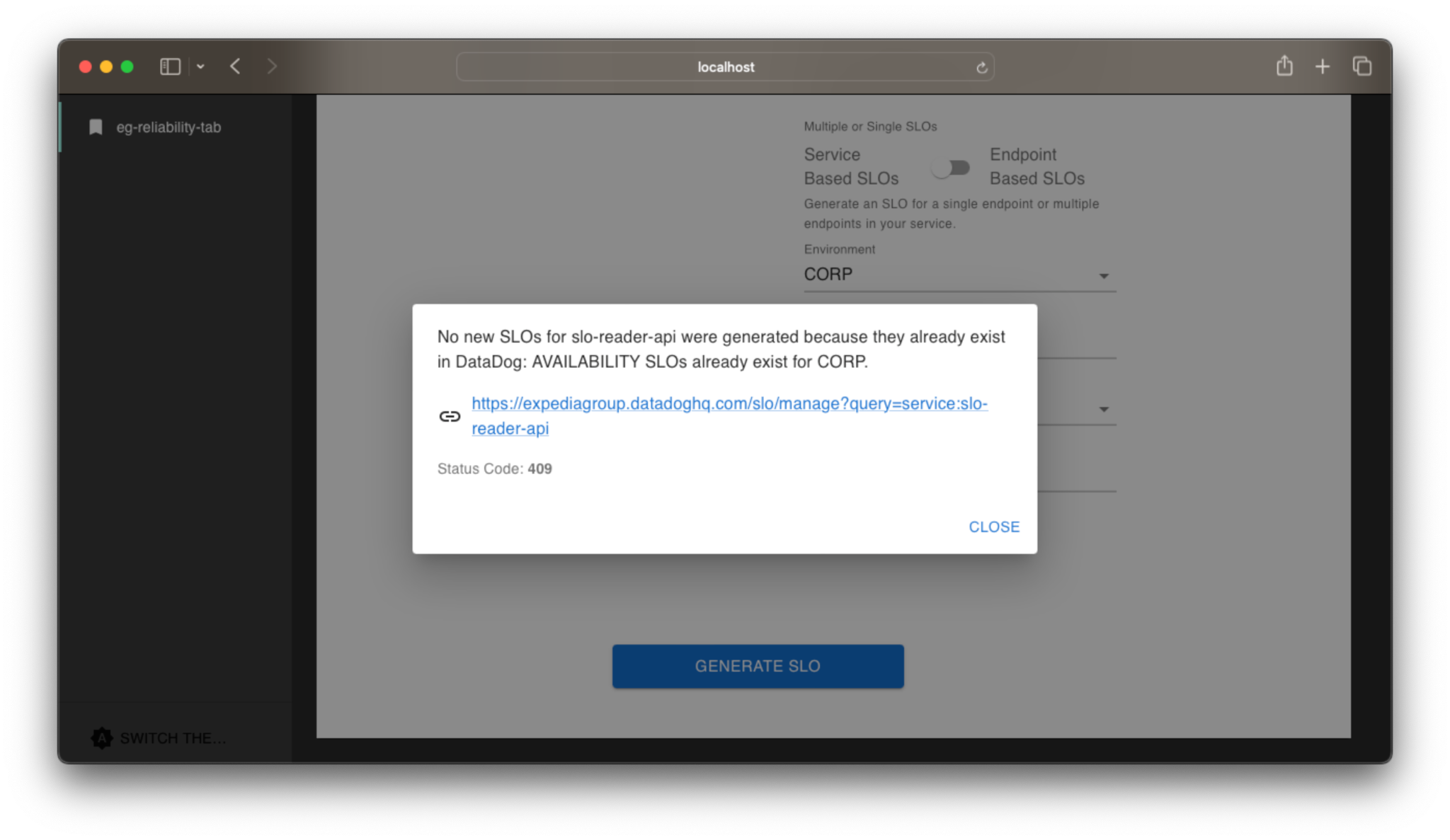
* The components have “value” properties that can be used to pass values through the components. This is an important part of how the entire form works. The input fields need to send back the correct data types in their values, so that the values can be correctly mapped to the properties within the JSON response.

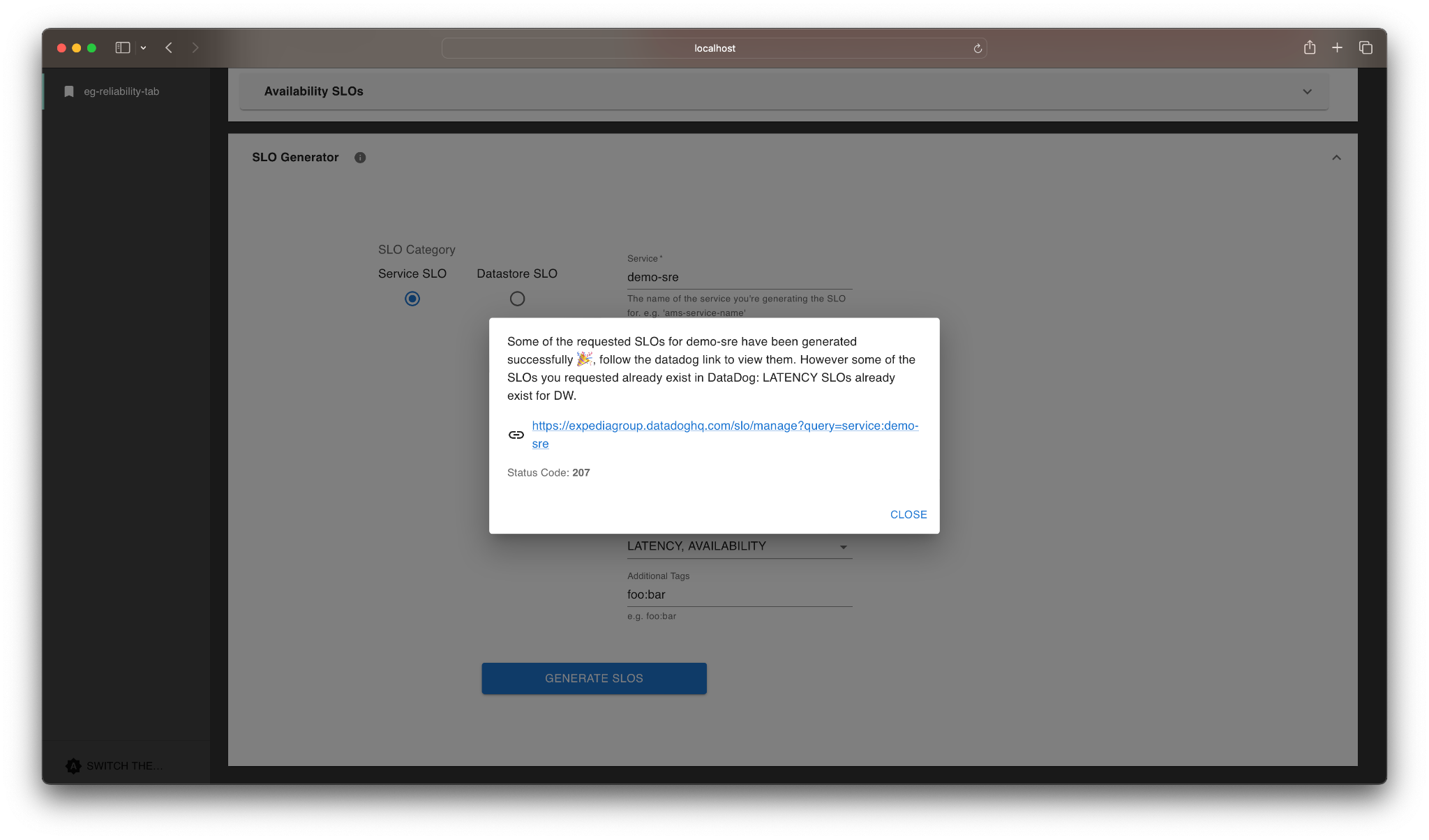
## Testing (S4, S6)

No testing framework swerve set up for this repo, so I had to manually test and check responses by running the frontend locally whilst using a production SLO Generator endpoint. The following are some of the use cases that I tested for:

* User tries to generate SLOs for an application that is not registered in AMS – they get a message telling them app can’t be found in AMS.



* User tries to generate SLOs for an application that already has one Availability and one Latency SLO in DataDog – they get a message telling them SLOs already exist in DataDog.
* User tries to generate SLOs for an application that has only one Latency SLO – one Availability SLOs is created, based on env and single-slo parameters.



## Deployment (B2, B3, S10)

The deployment process for this project was quite complicated. I was creating this Generator form within a plugin called “sre-slo-plugin”. This plugin needs to be run through another product called the “ReliabilityTab”. The ReliabilityTab needs to be run through “eg-backstage” which is Expedia’s instance of Backstage. My team own the sre-slo-plugin, but not the parents for the plugin. ReliabilityTab and eg-backstage are owned by other teams, so this required 3 PRs to 3 separate repos:

* A PR to the sre-slo-plugin; my team would need to take a look at my code, suggest any changes, and verify that I had tested the necessary use cases using the screenshots that I added to my PR. Once that was approved,
* A PR to the ReliabilityTab; in this PR, I would need to make sure that I updated the version number in the inner package.json file and then update the dependencies using npm install. Then I would raise the PR and contact someone from the team via Slack and wait for them to approve and merge it.
* Finally, I would raise a PR to the eg-backstage repo. This PR would also require me to change the version number, but in the root package.json instead of the inner package.json. I would then need to wait for this to be approved and merged.

The other two teams were based in India and America, so I had to make sure that I was asking for approvals at the right time of the day, otherwise I would have to wait until they were next awake. I also had to make sure that I was changing the version number in the correct jsons, as there were multiple across each repo.

## Maintenance (B3, S1, S7, S10)

The project was being built on top of a codebase that needed to be refactored. However, refactoring the code before starting work on the Generator Tab and other additions to the overall page, we would not have been able to complete in a realistic timeframe. There are plans and tickets being created and worked on as part of a LODO initiative for the Backstage UI and refactoring is a priority in this work. Some of this work includes:

* Separating larger components and functions into smaller modules, so that they can be reused across the codebase more efficiently.
* Configuring a test framework for the entire codebase.
* Adjusting and reconfiguring some of the response handling for edge cases.

The endpoints used in this project (the slo-generator and the slo-reader-api) have their own SLOs with alerts set up in our team slack channel. Whenever there are problems or errors with the frontend form, our alerts will pop up, usually showing us the related error code. Upon further investigation, we can see where the error was thrown from and whether or not it was through backstage. We also have our ask-channel on Slack where customers can ask questions if there are any problems.

# Recommendations and conclusions

Some of the things that I think really helped with the feedback loop and the implementation process were the wireframes. They were definitely one of the most important parts of this entire project. I didn’t feel that way before starting the project, but I think it would have been much more difficult to communicate with my team and get any concrete suggestions, additional requirements and corrections out of them if I couldn’t simply show them what I meant. Images and diagrams seem to be the most efficient way to communicate with my team, and I ended up referring back to those wireframes and even using them to plan out what would be added to the documentation when the changes were deployed. Screenshots are also very important and annotating the screenshots seemed to make it clearer when referring to a specific error or issue that I was facing whenever I asked for help or wanted to share a concern.

I managed to complete everything that I had set out to do by making sure to set time aside for everything. This included things like; the inevitable breaking of other components, having to wait on backend changes to APIs and other dependencies and tasks that were out of scope for the project and out of my control. I also set aside time to prepare and brainstorm any aftermath tickets and LODO work that could be done after the project and once everyone was back from various holidays.

In the future, I would lean more heavily on visual communication. I think a lot of problems came from miscommunication and misunderstandings. There are diverse levels of experience within my team and sometimes this can create a “language barrier”. I would also make sure that if frontend work was ever taken on again, that I asked for a frontend developer to work alongside us. It’s very difficult to work on a project that no one is experienced with, because you know that no one really knows the answer.

Now that this project is completed, LODO tickets for the Generator Project as well as some additional changes to the Reliability Tab are going to be worked on. The main goal is to refactor the base code and convert components that are too long into smaller, more modular components. More in-depth tests also need to be added to the codebase as well.

## Extra Resources Used

## <https://mui.com/material-ui/getting-started/>

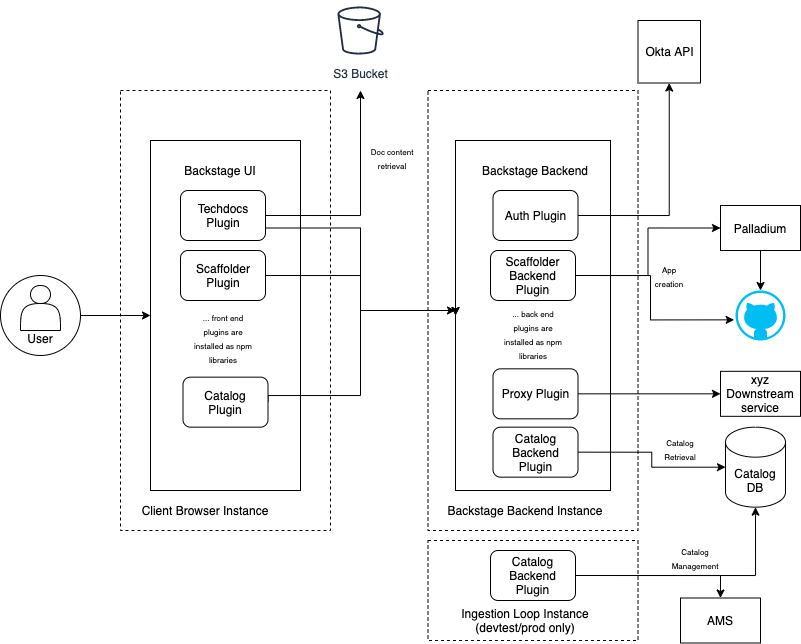
## <https://mui.com/system/getting-started/>

## <https://mui.com/base-ui/getting-started/>

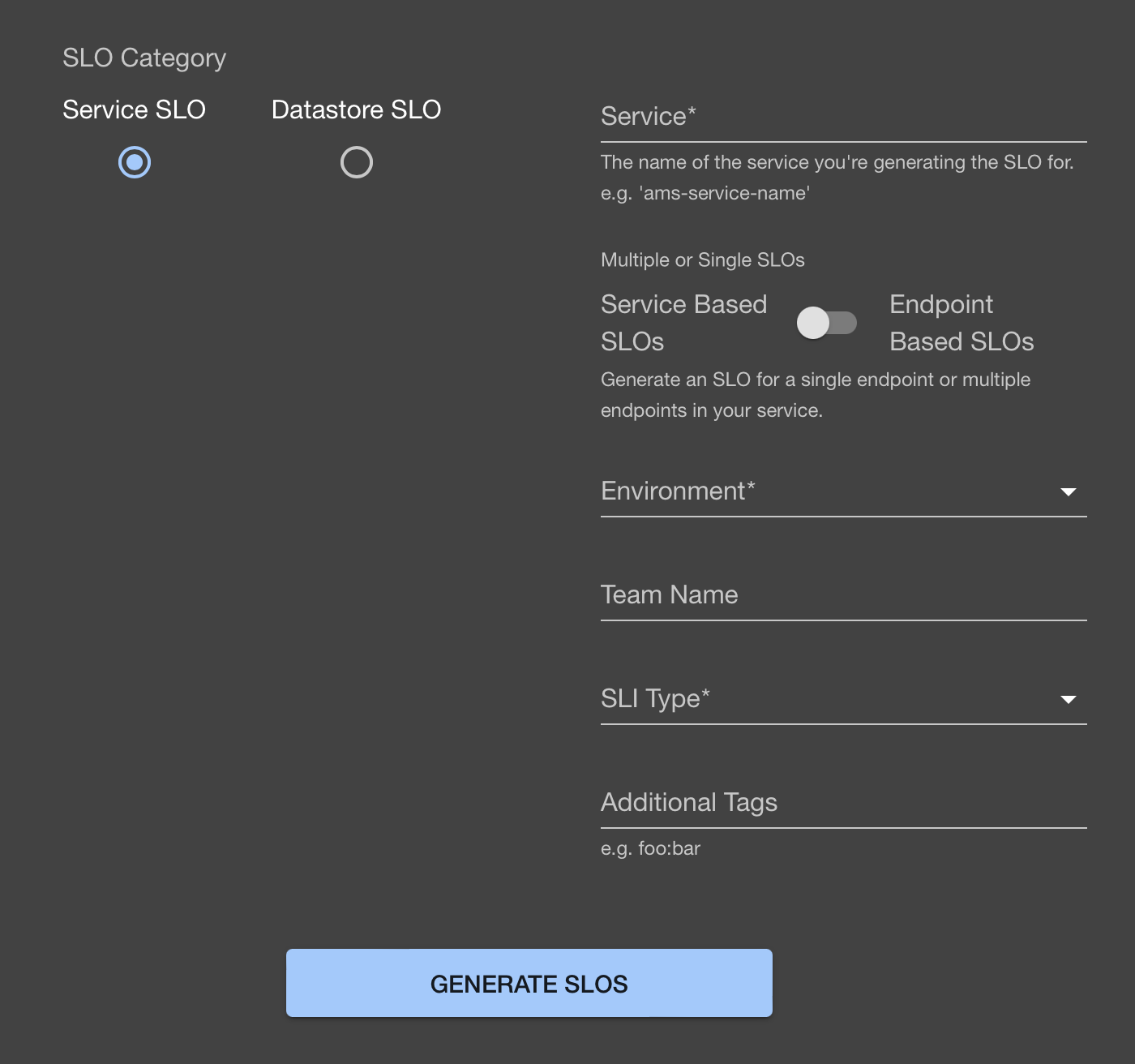
## 

## Artefacts

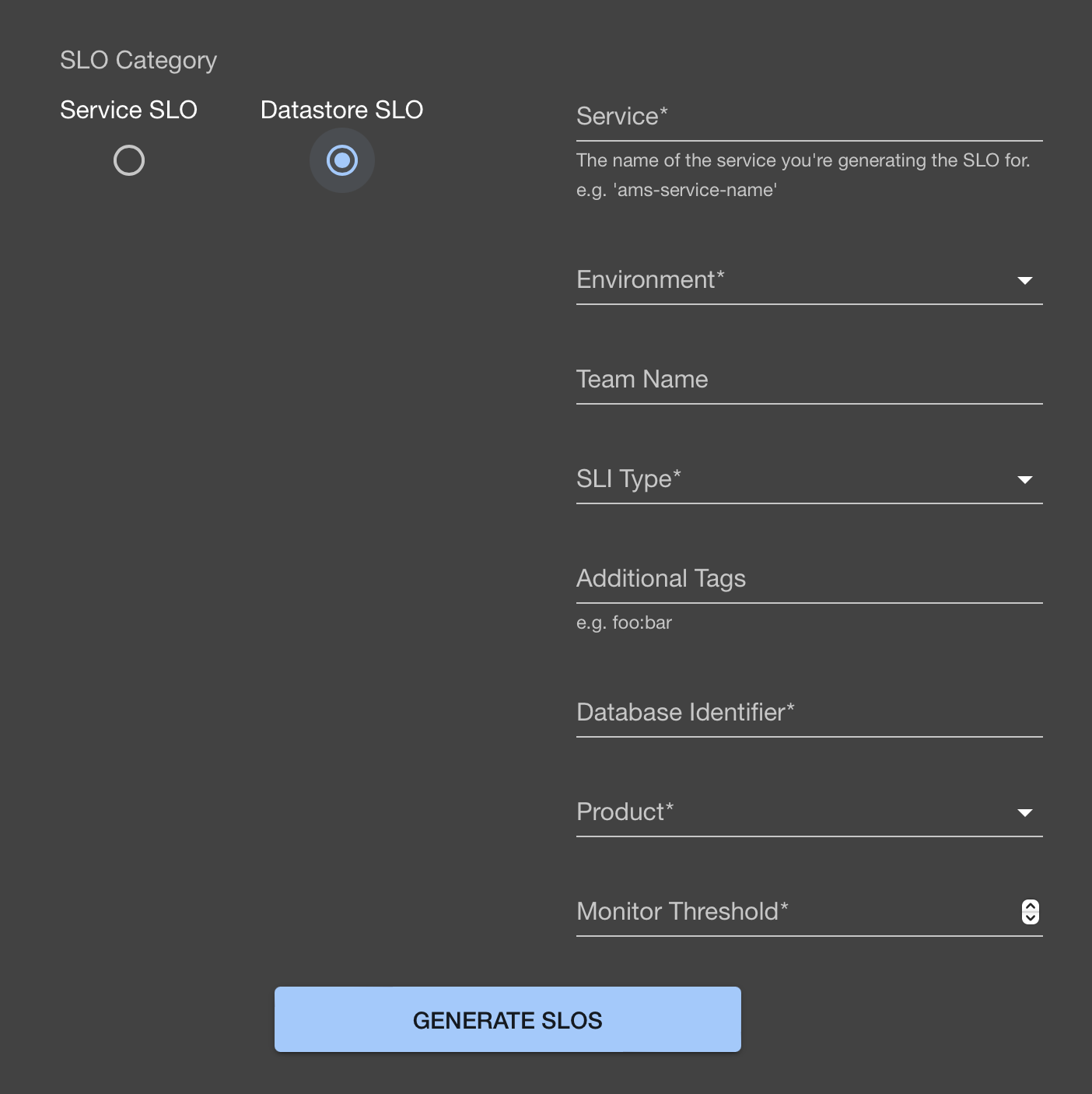
POC for Generator Form and sync button.

Diagram of sre-slo-plugin, ReliabilityTab and eg-backstage

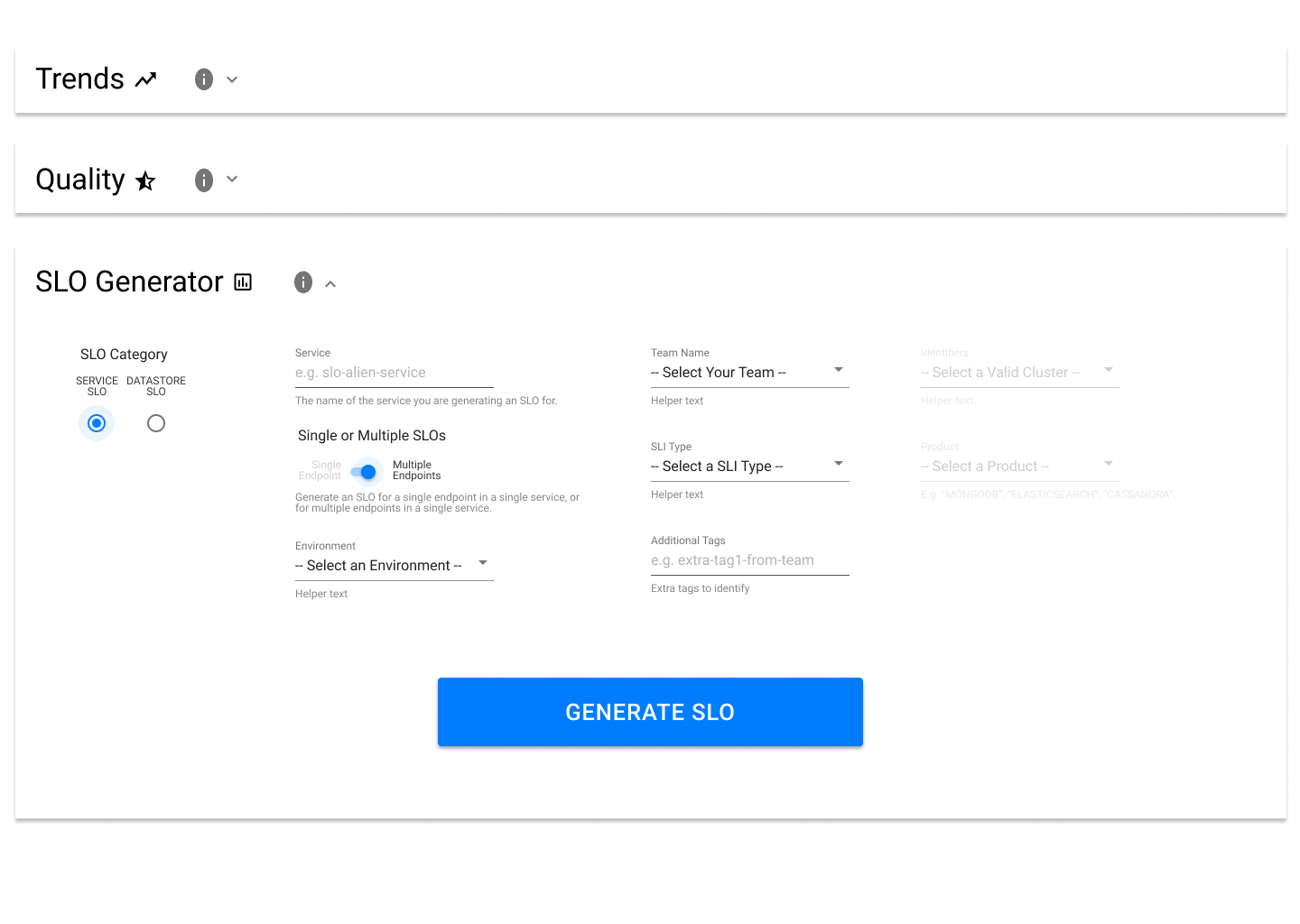
Deployed view of Generator Form with “Service” selected.



Deployed view of Generator Form with “Datastore” selected.



POC Generator Form with “Service” selected.



POC Generator Form with “Datastore” selected.

