Error Handling

Merln makes use of multiple third party APIs. However, as of right now, there is no consistent system to handle errors outputted by third party APIs. This is problematic for both developers, and the end users of Merln. In this document, I will detail how I handled errors caused by third party APIs, on both the backend, and the frontend.

Backend Changes

Most of the code that makes calls to external APIs are stored in Python classes, separate from the APIs that use the external APIs. These classes are stored in the folder api_manager. In particular, the file google_api_manager.py contains many such classes.

I started by modifying this file, adding code to catch these errors. The code to catch these errors came in form of classes, for example the class BadGateway, which addresses 502 errors. Once these errors have been caught, they are detected by the function handler exception_handler in shared.py file. If a third party error is detected while running an external API, the exception_handler outputs an error message. The list of all possible error messages can be seen below:

```
        except GoogleAnalyticsAPIOverload:
            return Response("failed to download data via google Analytics API", status
= 429)

        except DownloadReportError:
            return Response("failed to download data via google Analytics API",
status=500)

        except ServiceUnavailable:
            return Response('Google Analytics API Service unavailable', status = 500)

        except ValueError:
            return Response("Value Error", status = 500)

        except BadGateway:
            raise Response('Bad Gateway error from Google Analytics', status = 502)

        except:
            return Response("There's been an error with " + func.__name__, status =
500)
```

There exists a second function handler in shared.py – report_exception_handler. This function handler handles external API errors that occur in viewsets related to report

builder. Instead of outputting an error message once an external API error occurs, it raises an error instead.

Frontend Changes

On the frontend, we decided that end users needed less detail than developers working with the backend. Simply highlighting the source of the error is sufficient. I implemented this by catching the error status code outputted by the backend, in the related service.ts files.

The issue then becomes one of implementing a user interface to inform the user of the error. Two approaches were considered in achieving this. The first was to add a caption to the existing error image. This was achieved using the errorMessage property of the <app-no-data-img> component. This caption would appear at the bottom of the image, in a textbox. However, the issue with this approach was that the placing the error message in caption made it difficult to read.

After discussing the issue with management, we decided that the error message was to appear directly below the text, “Data Not Available”. This led to the second approach we took. The design team generated a new error image for our purposes, which can be seen below:

Testing

It is difficult to trigger external API errors. Raising an error directly in the code, however, should be enough to test how the code behaves once a external API error is caught.

Future Considerations

The error message should be dynamically generated in the future, instead of relying on a static image, as is the case right now.