**Cooper Platform Frontend Application**

**UI Development -Technical Design Document**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Prepared By** | **Reviewed by** | **Approved By** |
| **Name** | Naveen Kumar A |  |  |
| **Role** | Senior UI Developer | Project Leader | Project Manager |
| **Signature** |  |  |  |
| **Date** |  |  |  |

Table of Contents

[1.0 Introduction 3](#_Toc117797722)

[1.1 Purpose of this document 3](#_Toc117797723)

[1.2 Scope 3](#_Toc117797724)

[1.3 Intended Audience 3](#_Toc117797725)

[1.4 Definition & Acronyms 3](#_Toc117797726)

[2.0 Overview 4](#_Toc117797727)

[3.0 Detailed Documentation 4](#_Toc117797728)

[3.1 Application Framework 4](#_Toc117797729)

[3.2 Application folder structure 6](#_Toc117797730)

[3.2.1 Atoms 6](#_Toc117797731)

[3.2.2 Rules of Atoms 6](#_Toc117797732)

[3.2.3 Molecules 6](#_Toc117797733)

[3.2.4 Rules of Molecules 7](#_Toc117797734)

[3.2.5 Organisms 8](#_Toc117797735)

[3.2.6 Rules of Organisms 8](#_Toc117797736)

[3.2.7 Templates 8](#_Toc117797737)

[3.2.8 Rules of Templates 8](#_Toc117797738)

[3.2.9 Pages 9](#_Toc117797739)

[3.2.10 Rules of Pages 9](#_Toc117797740)

[3.3 Application Coding standards 10](#_Toc117797741)

[3.4 Application Network interaction 11](#_Toc117797742)

[3.5 Development environments 11](#_Toc117797743)

[4.0 References 11](#_Toc117797744)

# Introduction

## Purpose of this document

This document is aimed at providing the guidelines for creating a UI (User Interface) Development to ensure that:

* Information related to the application UI development is documented for better understanding of the process flow.
* Key and relevant details for on-screen data capture and data view are available for downstream architecture and design activities.
* Client business gets a chance to validate the UI and information layout for the application.

## Scope

The scope of this document is to provide the User Interface is development information, related to application initial setup, dependencies, usage and interaction of application at different stages ex:(development, staging, production), data flow and data storage in the application, application core framework details, application network interaction, application deployment, application testing, application folder structure, application building blocks, application coding standards, application tasks branching strategy with respect to VCS (Version Control Systems).

## Intended Audience

The intended audience for this document is

* Software developers
* Software testing group

## Definition & Acronyms

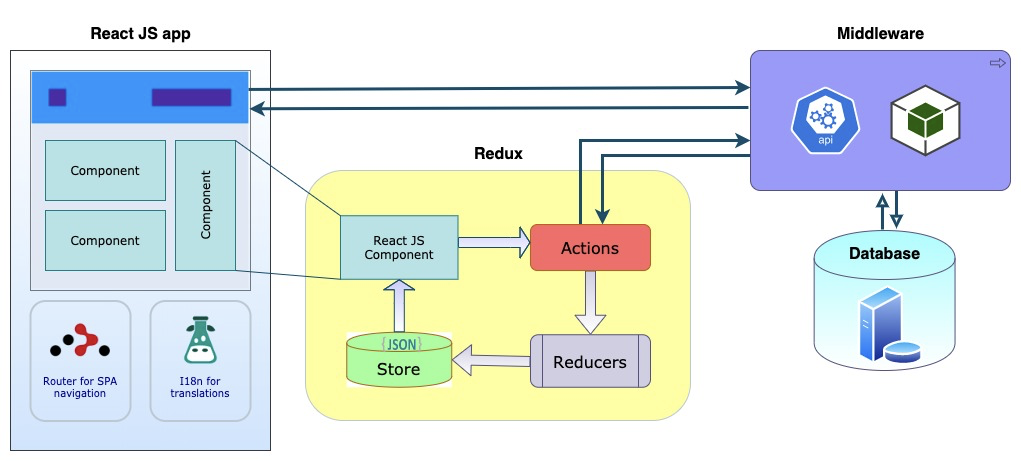
UI – User interface

Application – referring to Frontend application (React Framework)

VCS – Version Control System

# Overview

The frontend application is developed on React Framework. The framework is built using the language JavaScript (JS). For the development of the application JavaScript is used as the primary language along with HTML and CSS. The application data flow diagram is as follows.



*Frontend Application Data flow*

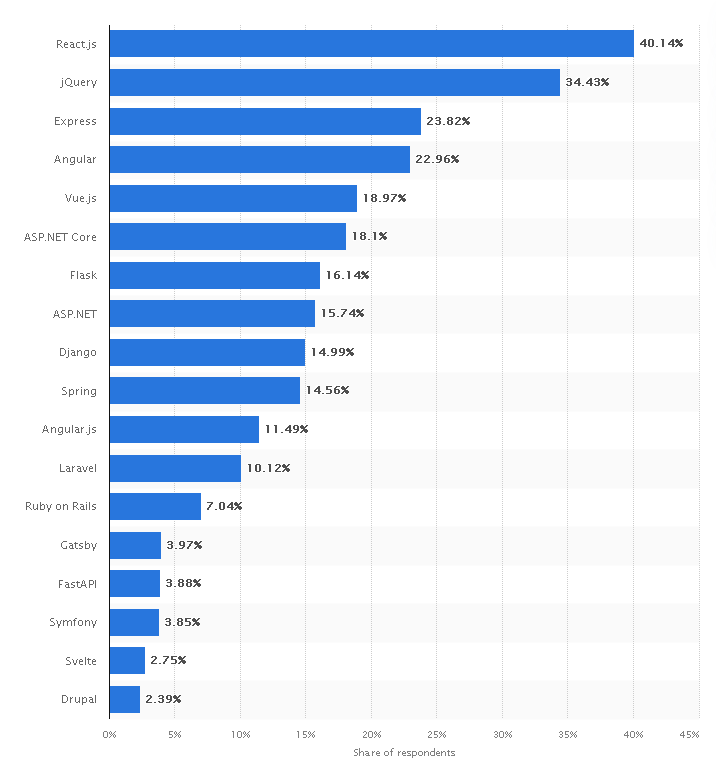
The application has a building block called React Components which can be reused as per the requirement and optimization. The data flow in the application is handled at the Redux store which acts as the single source of truth for the data storage for the application on the browser.

# Detailed Documentation

## Application Framework

The Application framework for frontend development used is ReactJS, the primary language for development is JavaScript. React.js or React in general can be considered an open-source JavaScript library that is used to create a user interface (UI) for web applications. React Native on the other hand is used for building cross-platform mobile applications. The underlying framework is more or less the same as React but instead of using web components, it uses native mobile components.

**Popularity of ReactJS over other frameworks**



The popularity of usage of React framework over other UI development frameworks is shown above.

The creation of massive applications with temporal data changes is the fundamental difficulty that React solves. The data in these apps isn’t static, it evolves over time. So, using React you have a better way to display data that is continually changing in an application.

React may be thought of as the V layer in the model-view-controller (MVC) software development pattern. You divide your application into three pieces in the MVC approach. In a database, the model implements data storage and retrieval. The view is responsible for displaying the user interface, whereas the controller is in charge of receiving user input and passing it on to the model. React just allows you to utilize the view component; you’ll need to use different tools for the model and controller sections.

There are some advantages that made React.js so popular:

* 1. It is straightforward. You only need to describe the app’s look and React will handle the user interface updates as the data changes.
  2. It is declarative in nature. When data changes, React understands how to update the changes.
  3. React is a component-based framework. This facilitates the creation of testable, reusable code. You create a component once and reuse it anywhere, any number of times.

For other information refer to React official website

<https://reactjs.org/docs/getting-started.html>

## Application folder structure

Atomic design works on the idea that everything can be abstracted into a singular module, which can then be used to construct bigger compositions of UI entities such as search bars and card groups. It is Brad Frost’s methodology for building design systems.

Brad Frost defines five main stages of UI components:

1.Atom

2.Molecules

3.Organisms

4.Templates

5.Pages

### Atoms

The **atom** is the most basic component, as generic as can be.

Icons, buttons, links, and labels are good examples of atoms. They don't do much on their own, and many other components on a site will typically depend on using these in one way or another. They can be used virtually anywhere throughout a site, so they have a lot of flexibility. Let's look at a very basic example, the Button:



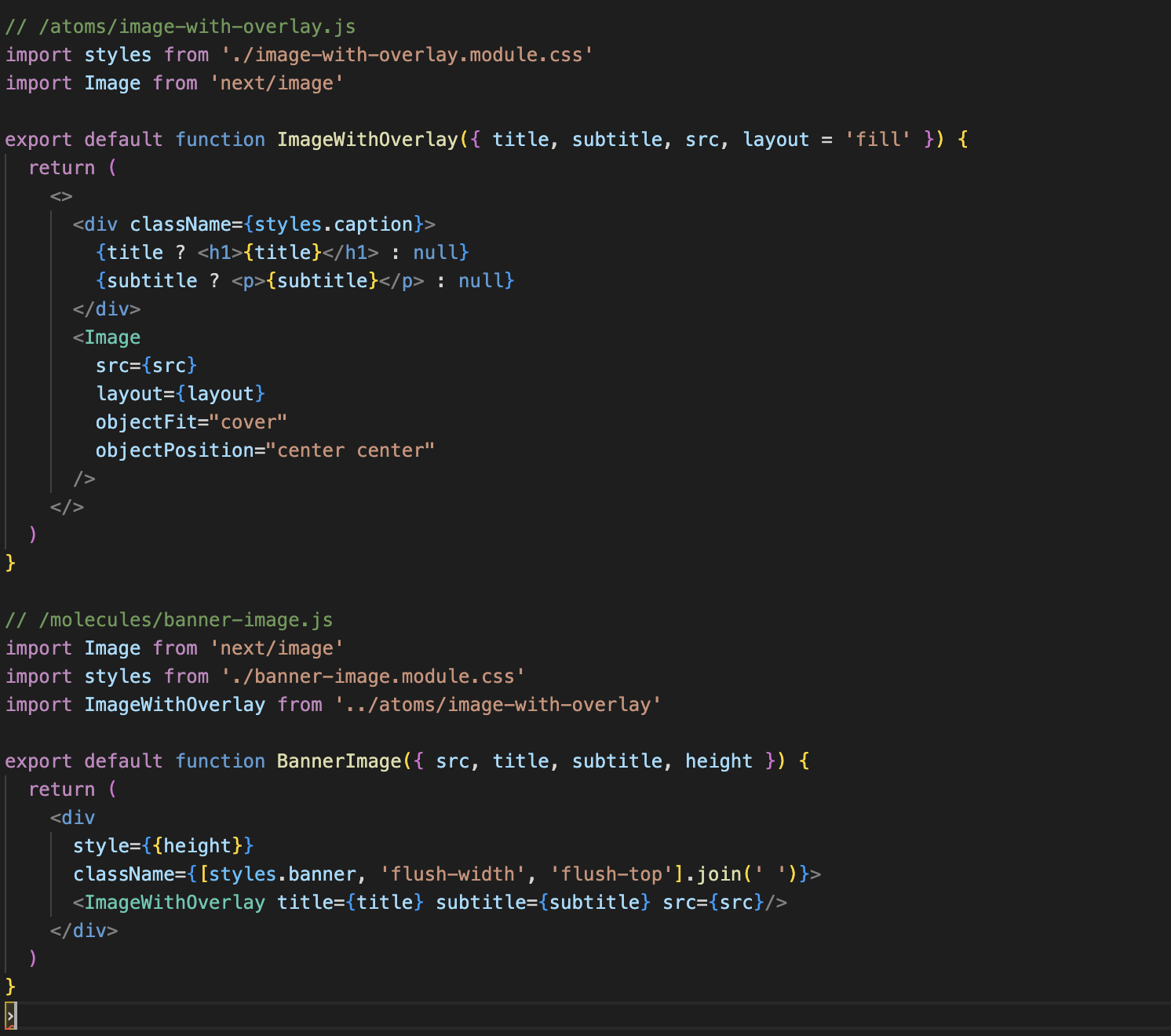
This is about as basic as it gets. Not even any styles needed here.

### **Rules of Atoms**

* Should not compose other components/only uses native elements or framework-specific components like native elements
* Can have its own markup & styles
* Can maintain its own internal state
* Should not be tightly coupled to specific UI or Logic areas
* Should not access application (or higher level) state directly
* Should not have any non-ui/layout related side effects
* Should not implement any application-specific business logic

### Molecules

Molecules are a combination of atoms that form more complex pieces of our UI, such as a search field with a submit button. Below is a molecule BannerImage that composes ImageWithOverlay Atom to create a banner designed specifically to be added at the start of content, flush with the top & side margins of its parent element.



### **Rules of Molecules**

* A component made up of one or more Atoms
* Can have its own markup & styles
* Can maintain its own internal state
* Should not be tightly coupled to specific UI or Logic areas
* Should not access application (or higher level) state directly
* Should not have any non-ui/layout related side effects
* Should not implement any application-specific business logic

### Organisms

Organisms build on top of molecules and orchestrate larger parts of the UI. This can include a list of products, a header, forms, etc. Organisms can even include other organisms.

One of the more basic organisms in the sample app is the SidebarNavigationItem. At two molecules and one atom,

SidebarNavigationItem component contains IconHeading as SidebarTitle, and VerticalListLayout as SidebarLinkList



### **Rules of Organisms**

* A complex component made up of multiple atoms and/or molecules and/or other organisms
* Can have its own markup & styles
* Can fetch application-specific data
* Can implement application-specific business logic
* Can be connected to application (or higher level) state
* Can be tightly coupled with a specific area (UI and/or Logic) of the app
* Can be organized into sub-folders by logical categorization (feature, page, etc...)

### Templates

*Templates* are where our pages start to come together, giving context to all of our organisms and molecules by giving them a unified purpose.

For Example, a template for a contact page will have organisms for headers and forms, and molecules for text fields and navigation bars.

### **Rules of Templates**

* A component that facilitates the layout of multiple organisms
* Can have its own markup & styles.
* Can accept & pass props as required.
* Should not access application (or higher level) state
* Should not have any non-ui/layout related side effects
* Should not implement any application-specific business logic

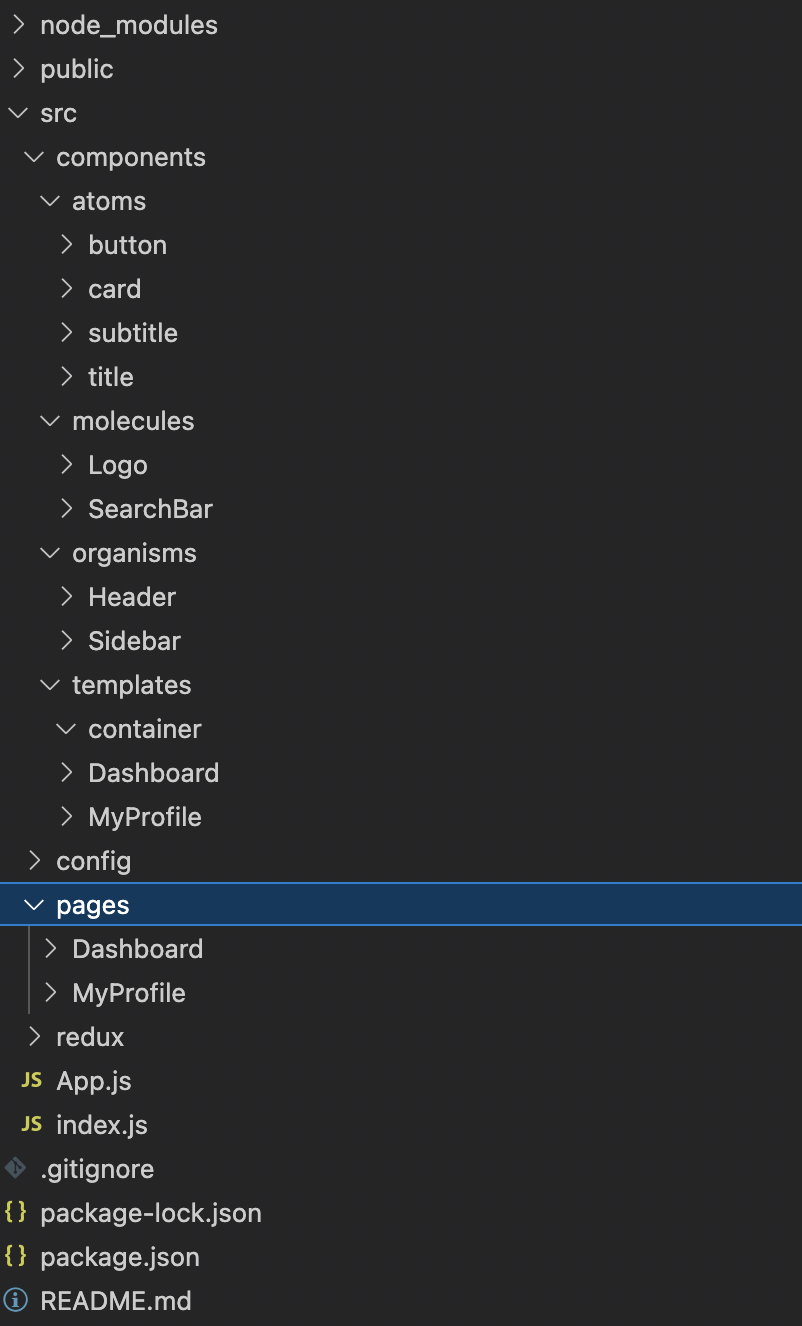
### Pages

Pages, as the name implies, is our final page with all its content. The difference between pages and templates is that templates don’t provide any content.

### **Rules of Pages**

* A component that implements a particular template
* Can fetch application-specific data
* Can implement application-specific business logic
* Can be connected to application (or higher level) state
* Should not have its own markup & styles

Sample Folder Structure:



## Application Coding standards

The application coding standards are drawn from the standard principles mentioned in the Airbnb standards of naming and structuring of the components.

The following <https://airbnb.io/javascript/react/>

The above link has the details of the components naming other ways of sending the props to functional and class components.

As per Robert Cecil Martin standards of clean code principles would be adopted for the further non React components logical implementation.

## Application Network interaction

The application interacts with the Backend server application with HTTP1.0 protocol and requests the APIs and gets the response and stores the data in the redux store and does the respective manipulation under the session instance in the browser.

The ‘axios’ is the respective npm dependency which is used for network call and embedded in the react saga framework.

## Development environments

The application is development is done with three environments

1. Development Environment
2. Stage / Testing Environment
3. Production Environment

**Development Environment**: In the development environment the developers use the dev dependencies for easy hot reload of application for quick updates and the environmental settings related to remote apis, secret variables are protected.

**Stage / Testing Environment**: The staging environment is used by the QA for testing the functionalities of the application. The respective staging environment variables are configured by the environment files. The respective deployment builds are also mapped to the environment configurations.

**Production Environment**: The production environment is the which is been thoroughly tested by QA and all quality checks passed, and finally released.

## Application Overview

# We have two applications as part of development. They are

# Cooperai\_admin\_application

# Cooperai\_user\_application

**3.6.1 Cooperai\_admin\_application**

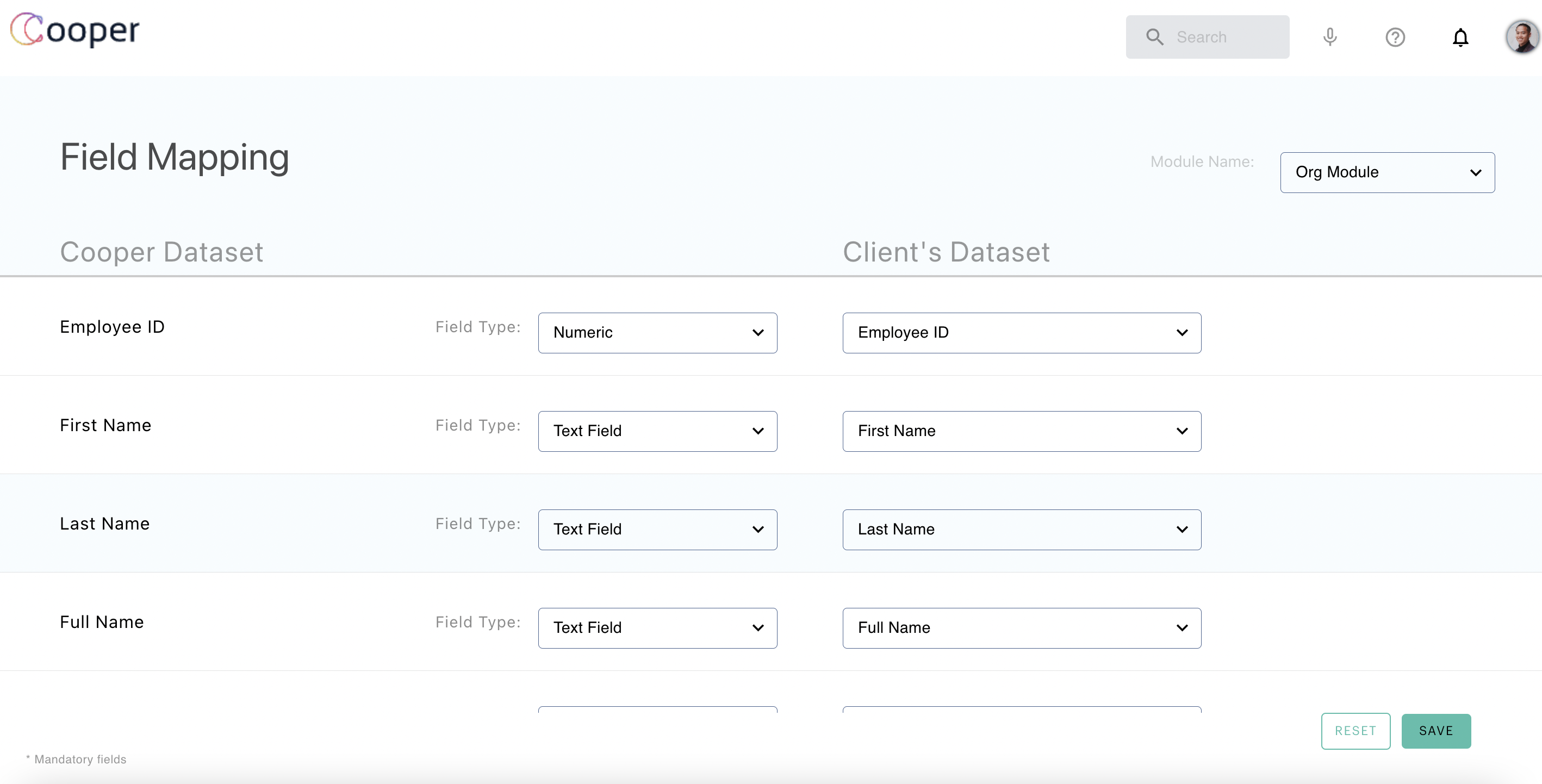
As part of Cooperai\_admin\_application we have developed screens for below modules.

1. Org Module
2. P & L Module
3. Margin Module
4. Vendor Module

**1. Org Module**

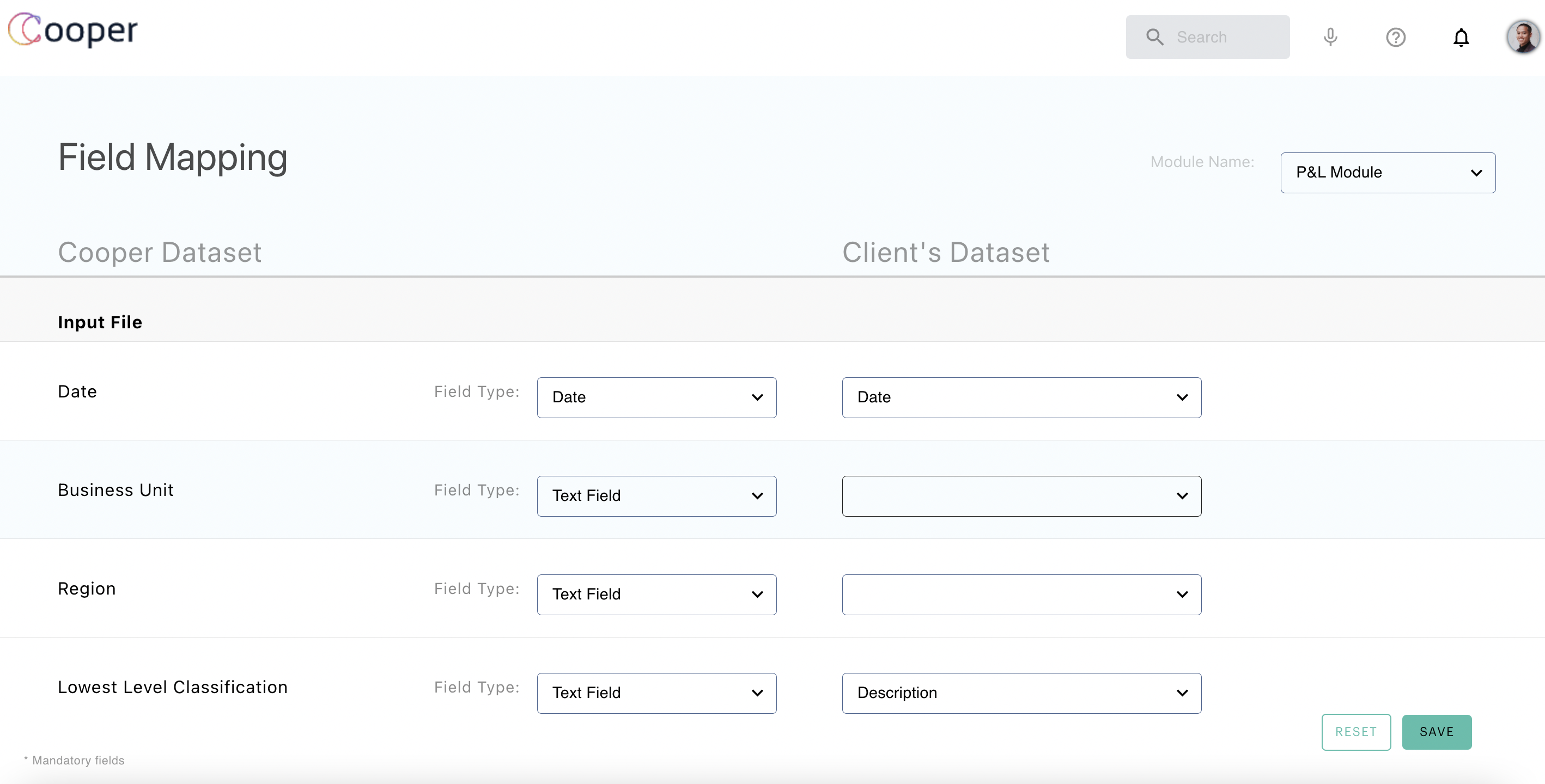
Org Module contains Field Mapping Screen in which admin will map Client datasets to copper datasets and the mapped data will saved into the database on click save.

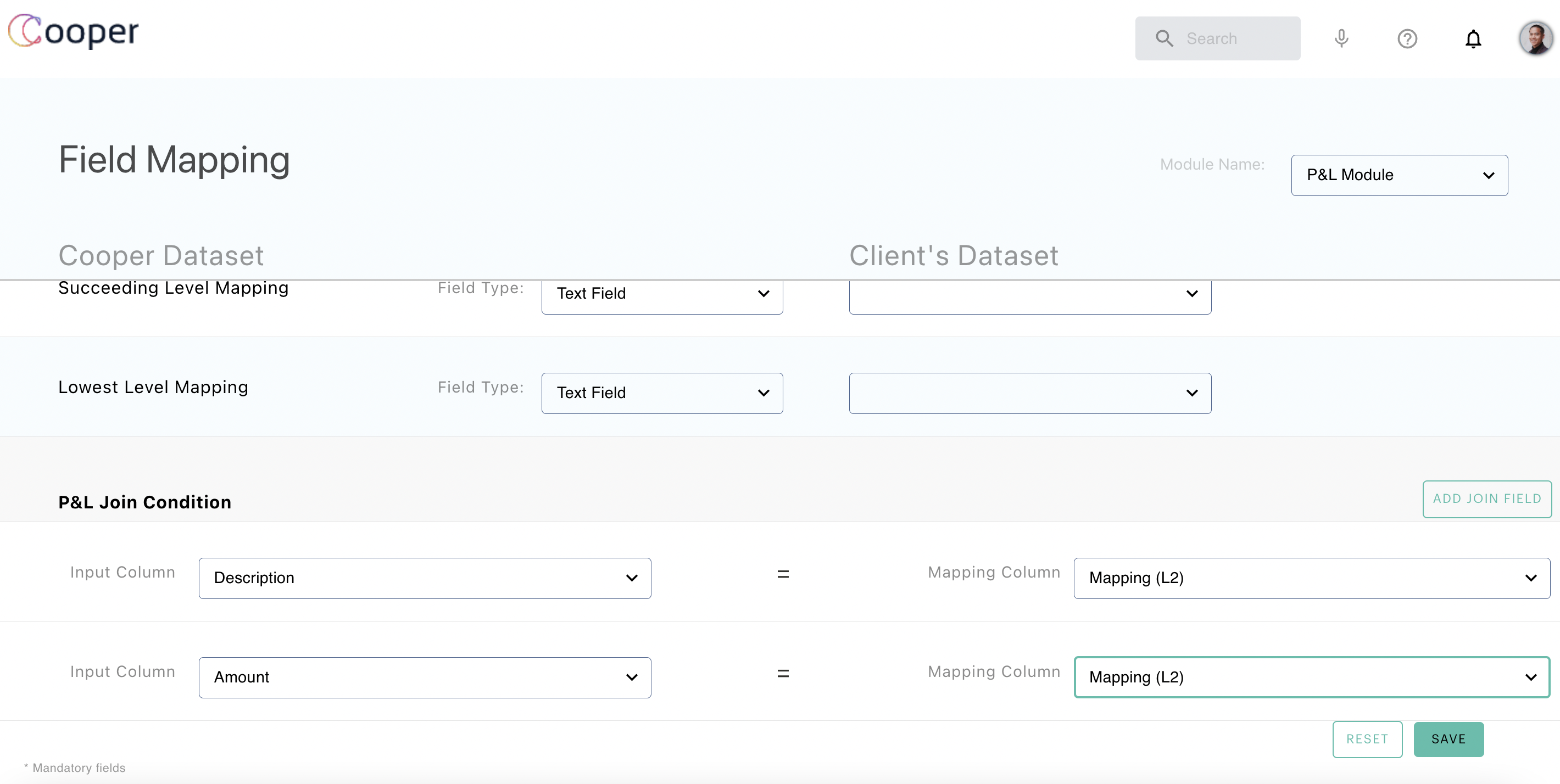
Sample screenshot:



**2. P & L Module:**

P & L Module also contains Field Mapping Screen but with some categorised data.  
 It also has Join Field functionality by which admin can Join Input columns to Mapping columns. The mapped data will be saved into DB on click of Save functionality.





**3. Margin Module:**

Margin module contains below screens.

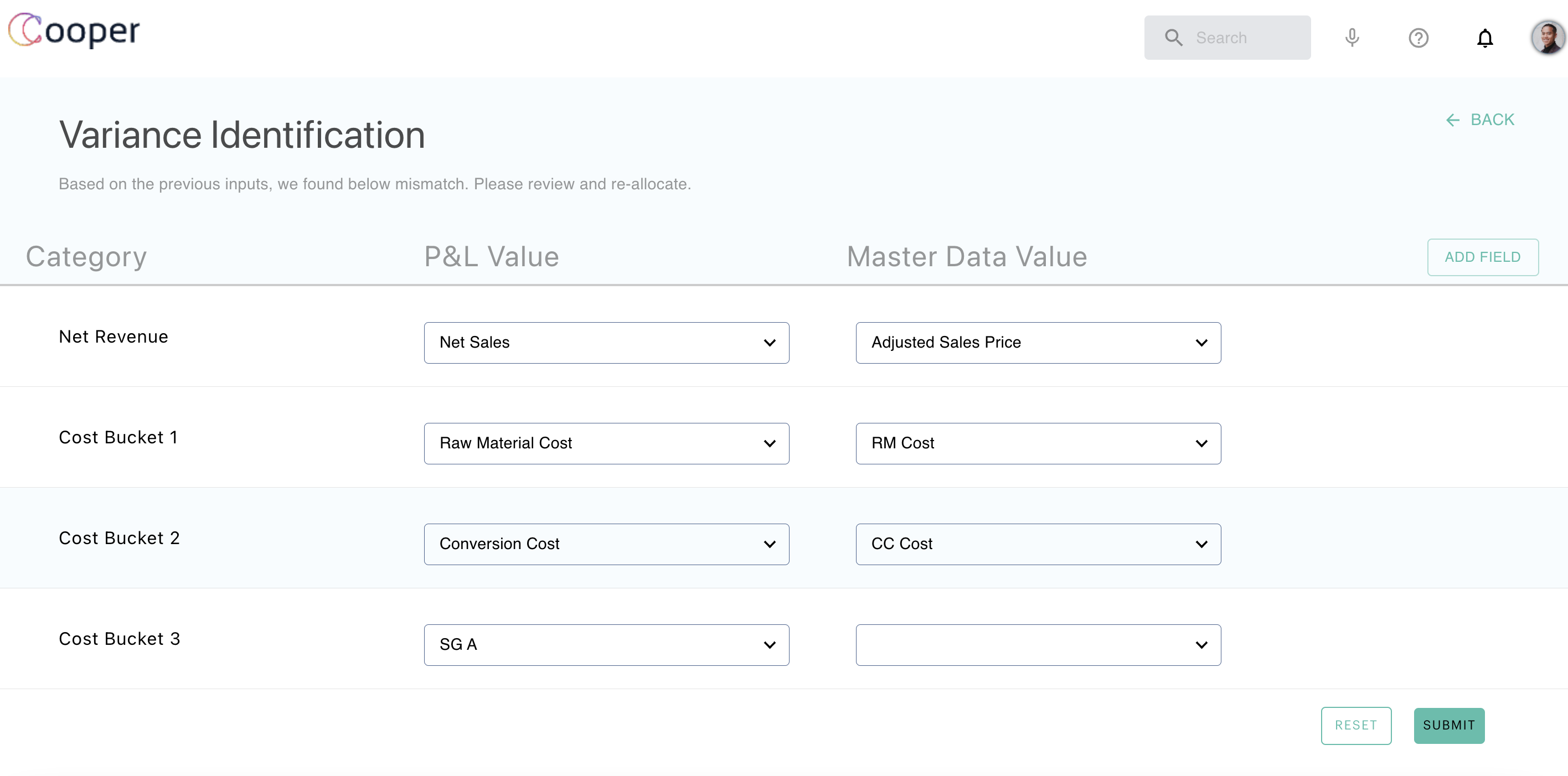
1. Field Mapping
2. Variance Identification
3. Variance Data
4. Performance Allocation

1. **Field Mapping**

In Field Mapping Screen We will map the fields as we did in above modules. Here Extra ScrollSpy header will be there, but the functionality is same as the previous modules.

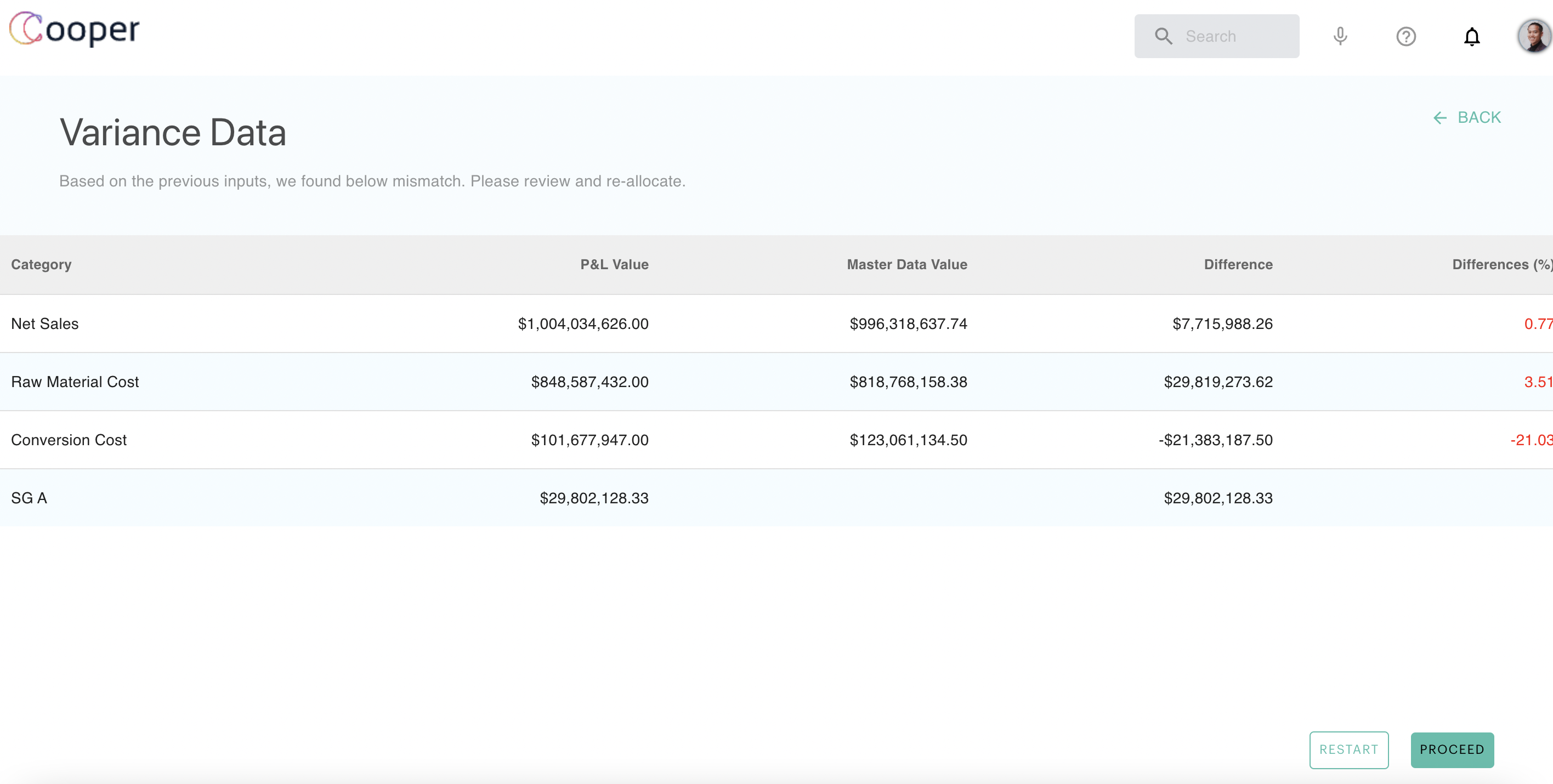
1. **Variance Identification**

In Variance Identification Screen Admin will map P and L values to Master data values with required Category. By default, three Cost buckets will be there. A new Cost bucket will be added by clicking the Add Field button.  
   
On click of Submit button the data will be saved into DB and User will navigate to the Variance Data Screen.



1. **Variance Data:**

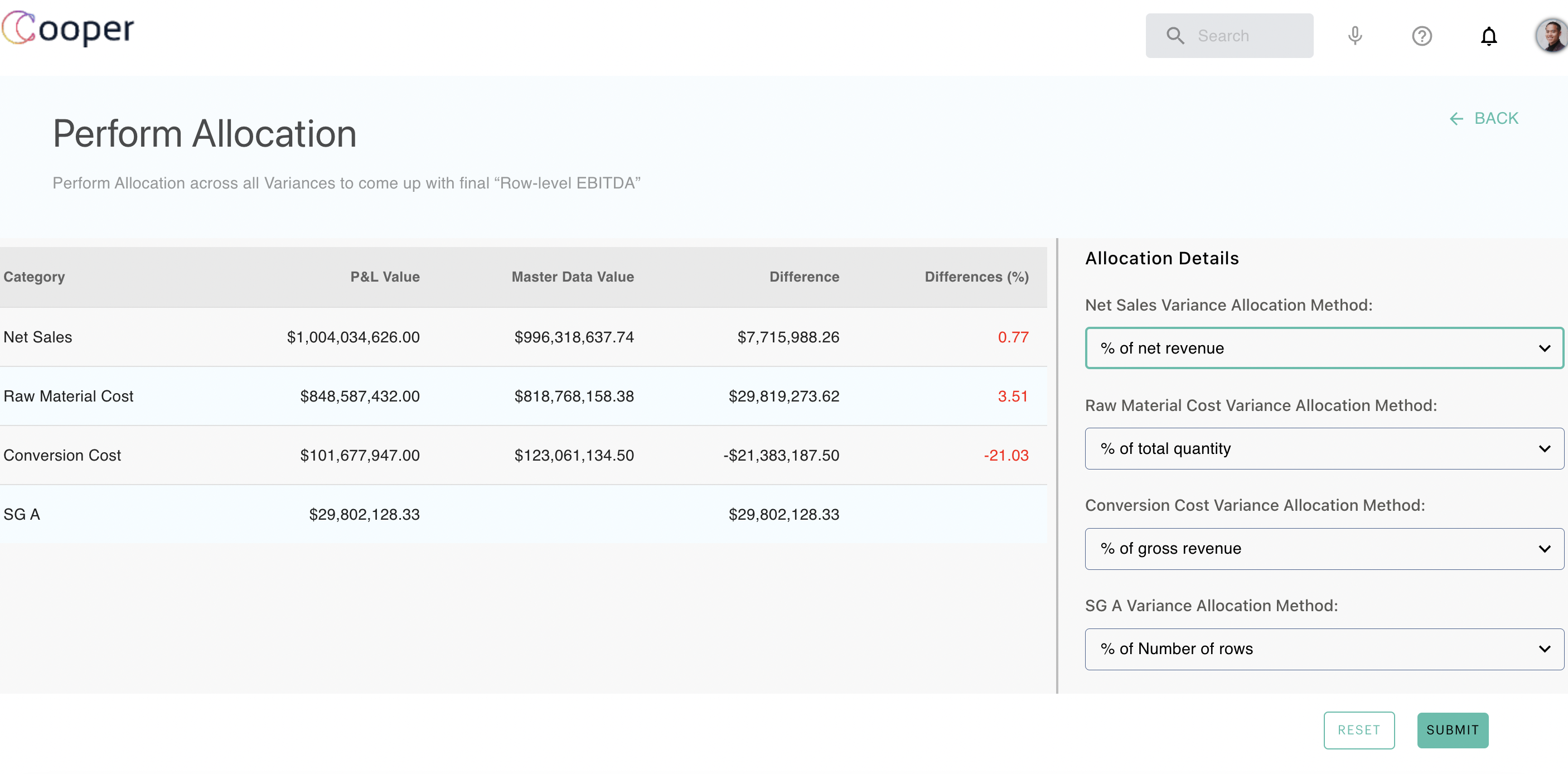
Variance Data screen will Provide the metrics based on the mappings done in Variance Allocations screen.  
   
Here admin will have a look at the data and if it looks fine, he will click on Proceed button to go to the Performance Allocation Screen.

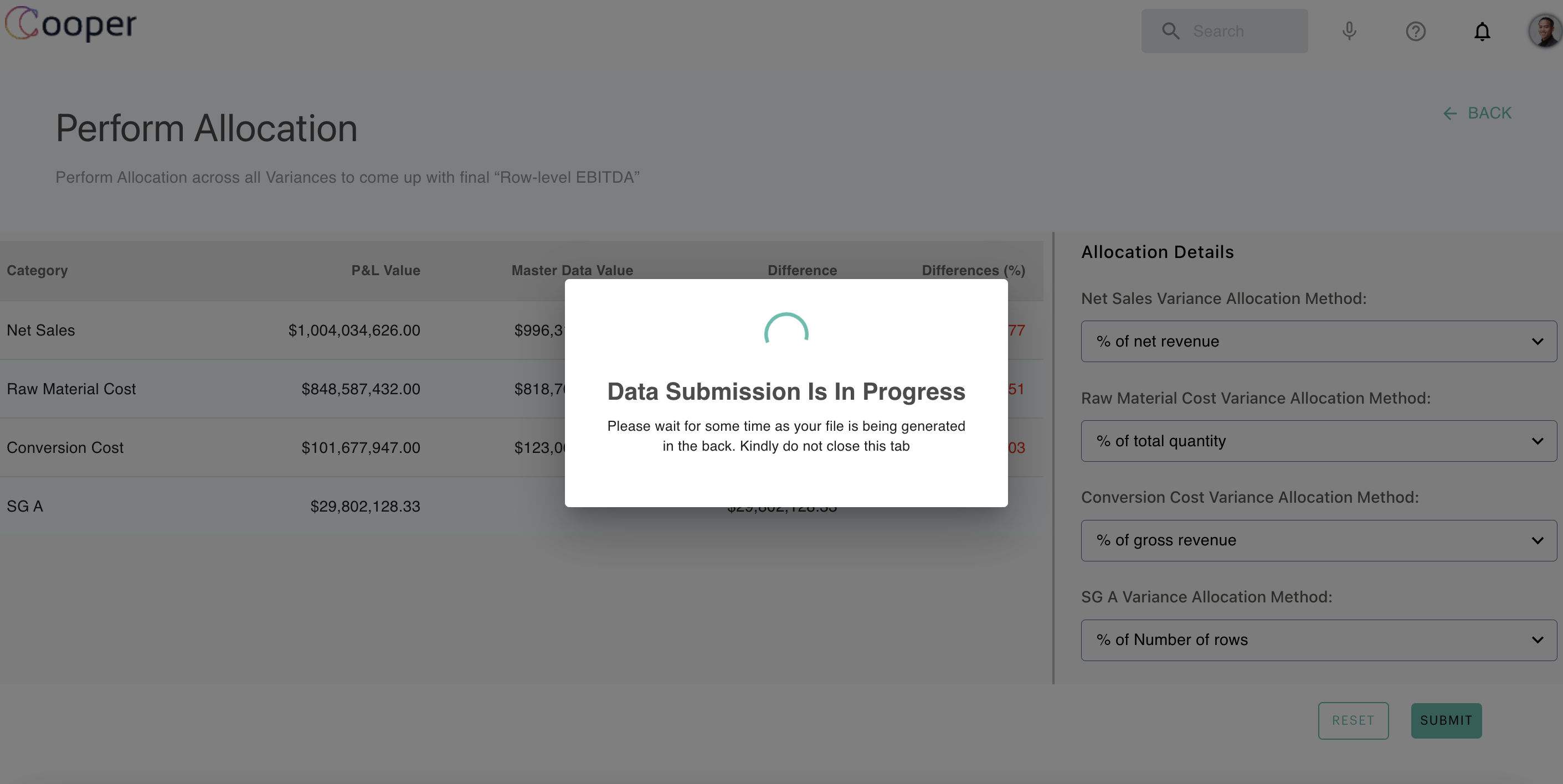


1. **Performance Allocation**

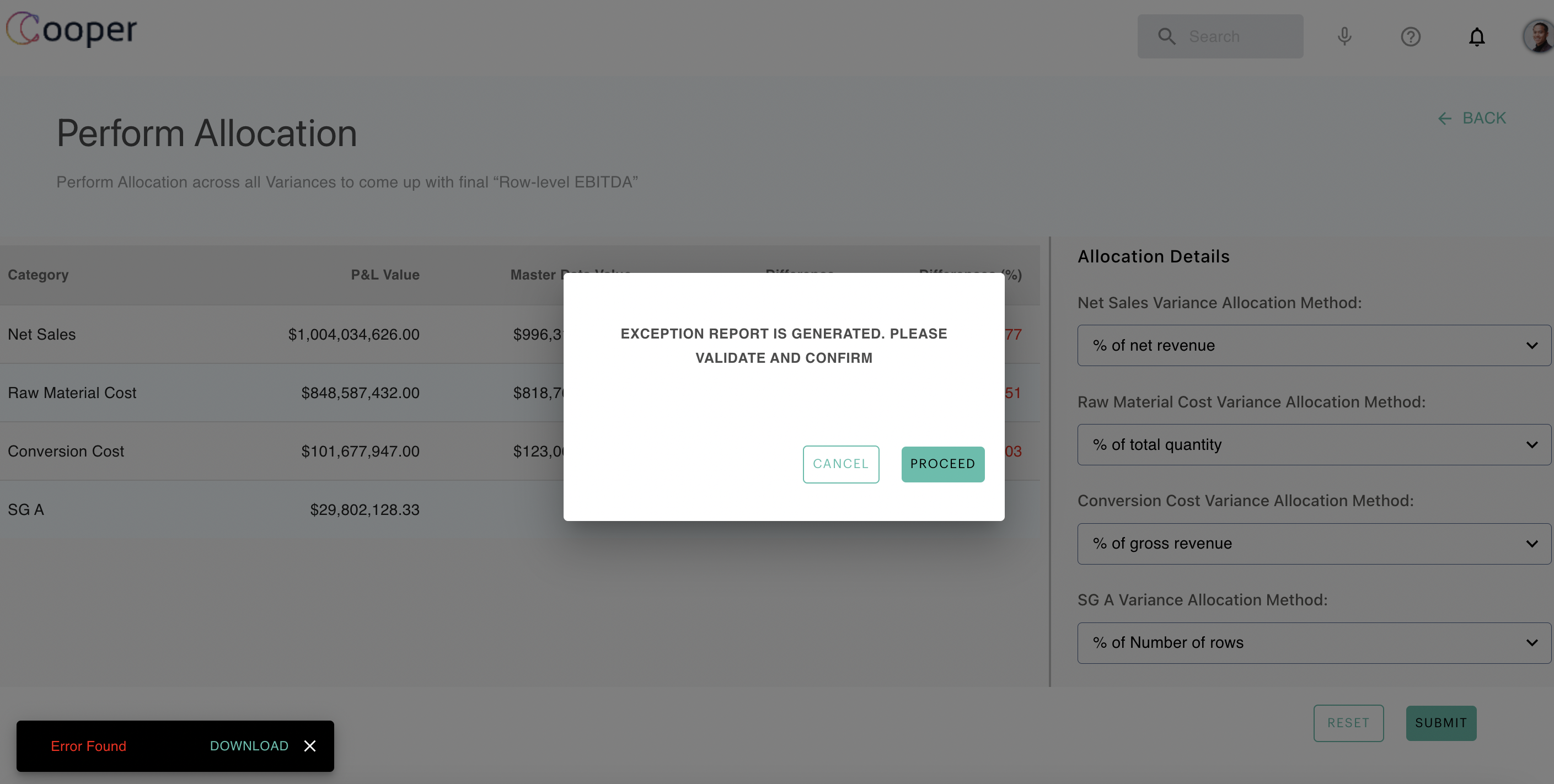
As part of this Screen Admin will select the allocations methods for the selected categories and do submit to proceed for Calculations.  
   
On click of Submit option below steps will be performed.

* Calculations
* Sanity
* Analytics

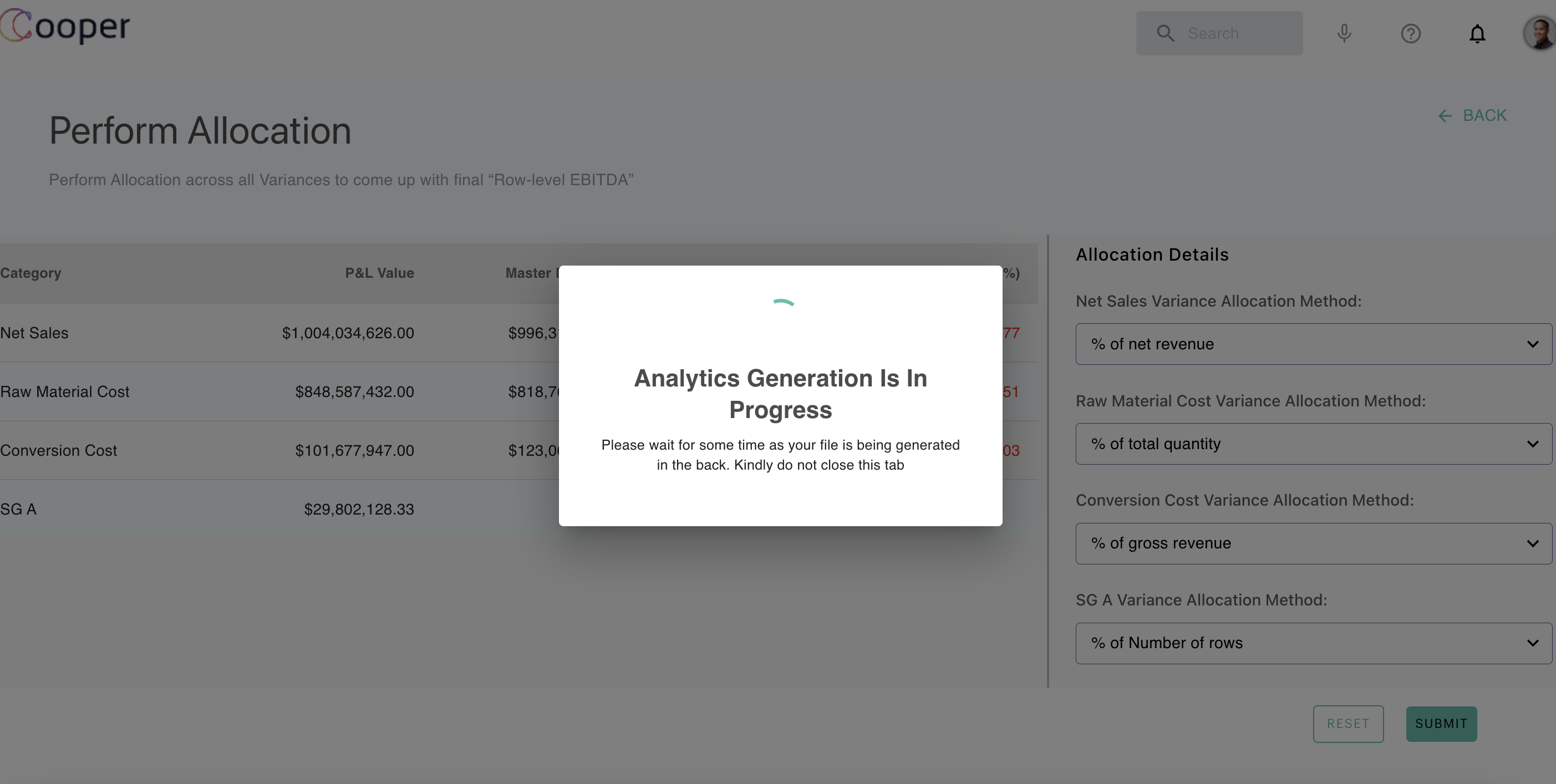




After Submission In most of the cases we will get Exception. Exception reports can be downloaded by clicking on the Download option.  
   
At this step Admin can Proceed or Cancel. If he click on Proceed the next steps can be performed otherwise admin will stay in same page.



Once Admin clicks on Proceed button Sanity checks and Analytics will be performed.



**4. Vendor Module:**

Vendor module has two categories (tables) of data. They are Spend Cube and Accounts Payable.  
   
Here We have two individual save actions are there for Field Mappings.

**Error Log File Download Feature**:  
   
For All the modules After the submission of Field mapping there may be chances of having errors related to the data. We have a feature of providing Error log file for admin to understand the error and resubmit the data with required modifications.

**3.6.2 Cooperai\_user\_application**

As part of User application, we have developed Login, Signup and Dashboard Screens.  
 Here the Dashboard screen displays the list of top 5 decisions.

# References

1. <https://reactjs.org/docs/getting-started.html>

2. <https://medium.com/the-research-nest/everything-you-need-to-know-about-react-ab24da4275ea>

3. https://airbnb.io/javascript/react/