

# Liberty Dash

## High Level Design and Architecture

*Team Iceberg*

**Project Manager:**  
Maya Bergandy

**Team Members:**

Bryce Bodley-Gomes

Tony Gao

Kevin Silva

Matthew Hinsley

Bhavik Jain

Adrian Poveda McKay

Chenhao Huang

Zachary Tousignant

Michael Schiller

# Table of Contents

## [Table of Contents](#)

### [1 - Introduction](#)

### [2 - Architecture](#)

#### [2.1 Client-Server Architecture](#)

##### [2.1.1 Rationale](#)

##### [2.1.2 Client](#)

##### [2.1.3 Server](#)

##### [2.1.4 Database](#)

###### [2.1.4.1 Internal Liberty Dash Database](#)

###### [2.1.4.2 External Liberty Mutual Database](#)

#### [2.2 Model View Controller Architecture](#)

##### [2.2.1 Rationale](#)

##### [2.2.2 Model](#)

##### [2.2.3 View](#)

##### [2.2.4 Controller](#)

#### [2.3 Libraries and Software Packages](#)

##### [2.3.1 Node.Js](#)

##### [2.3.2 Bootstrap](#)

##### [2.3.3 Log4JS](#)

##### [2.3.4 MySQL](#)

###### [2.3.4.1 MySQL Database](#)

###### [2.3.4.2 Oracle JDBC](#)

##### [2.3.5 Azure Active Directory Authentication](#)

###### [2.3.5.1 ADAL-Node](#)

### [3 - High Level Design](#)

#### [3.1 Client](#)

#### [3.2 Databases](#)

##### [3.2.1 Internal Liberty Dash Database Schema](#)

### [4 - GUI Design Mockups](#)

### [5 - Glossary](#)

# 1 - Introduction

The purpose of this document is to specify the high-level design for the Liberty Dash system. This document will outline our architectural strategies, discuss our design rationale, and display our current GUI mockups.

## 1.1 Overview

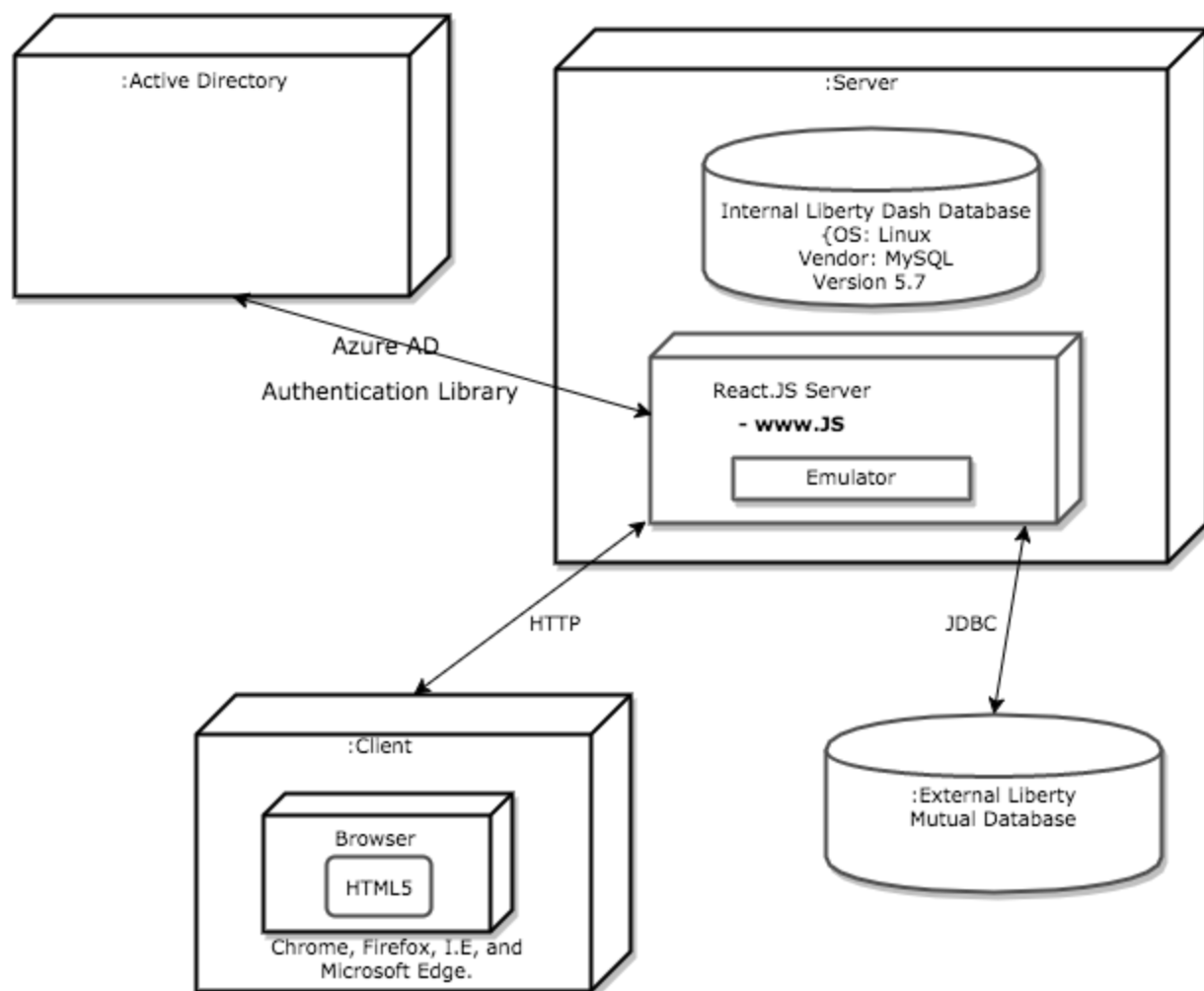
The Liberty Dash is an application built on a React.JS server with the purpose of simplifying and automating the process of executing macros on the External Liberty Mutual Database which contains Liberty Mutual Customer metadata. The server will communicate with the Liberty Dash client over an HTTP connection to receive macro requests. The server will use JSX to validate users with the Microsoft Azure Active Directory library and to interact with the External Liberty Mutual Database's metadata with SQL macros. Additionally, the server will also house the Internal Liberty Dash Database where the Changelog is stored (see Glossary for data the Changelog contains).

## 2 - Architecture

This section explains our architectural decisions and the rationale behind them.

### 2.1 Client-Server Architecture

This section will explain why we chose a client-server model for our system.



**Figure 2.1 - Client/Server Architecture**

### 2.1.1 Rationale

Using client-server architecture allows for:

- **Easier Separation of Logic:** Separating the system into a server application and a client application provides easier testing on the client side.
- **Diversified Endpoints:** Having a majority of the processing done on the server allows for an increased diversity of endpoints. The client-server model's decreased computational requirements for the client allows devices with lower processing powers to access the application with ease.
- **Faster Runtime:** The client-server model only transmits the representation of the data to the client. This results in faster access and manipulation rate for the data associated with the External Liberty Mutual Database.
- **Increased Reliability/Access:** The client-server model is easier to track and lock access to resources as they are in use. This ensures the validity of the resource access. Having a central server for all data guarantees users the ability to access data as long as server connectivity is available.

### 2.1.2 Client

The clients for this application are remote computer users who are running our application. These clients have the capability to interface with the server by sending requests to view processes (running, on hold, pending, failed, successful), create processes, view and manage Peer Reviews.

### **2.1.3 Server**

The server will maintain the Internal Liberty Dash Database and service requests created by the client. It will provide the capability to query the External Liberty Mutual Database, access the Internal Liberty Dash Database and view the Changelog. The server will process creation requests and manage the Peer Review request email notifications.

### **2.1.4 Database**

Our architecture involves two separate databases.

#### **2.1.4.1 Internal Liberty Dash Database**

The Internal Liberty Dash Database is the internal database of our Liberty Dash system. This database stores the Changelog in its own table. The Changelog is a table that logs all changes made to the External Liberty Mutual Database and the relevant information associated with these changes (see Glossary for full definition). This database also contains a table storing the Peer Review requests waiting to be processed.

#### **2.1.4.2 External Liberty Mutual Database**

The External Liberty Mutual Database is Liberty Mutual's existing database that stores metadata. The Liberty Dash application interacts with this external database through JDBC.

## 2.2 Model View Controller Architecture

This section will discuss what are the advantages of using the MVC model for our system and how this model relates to our system's components.

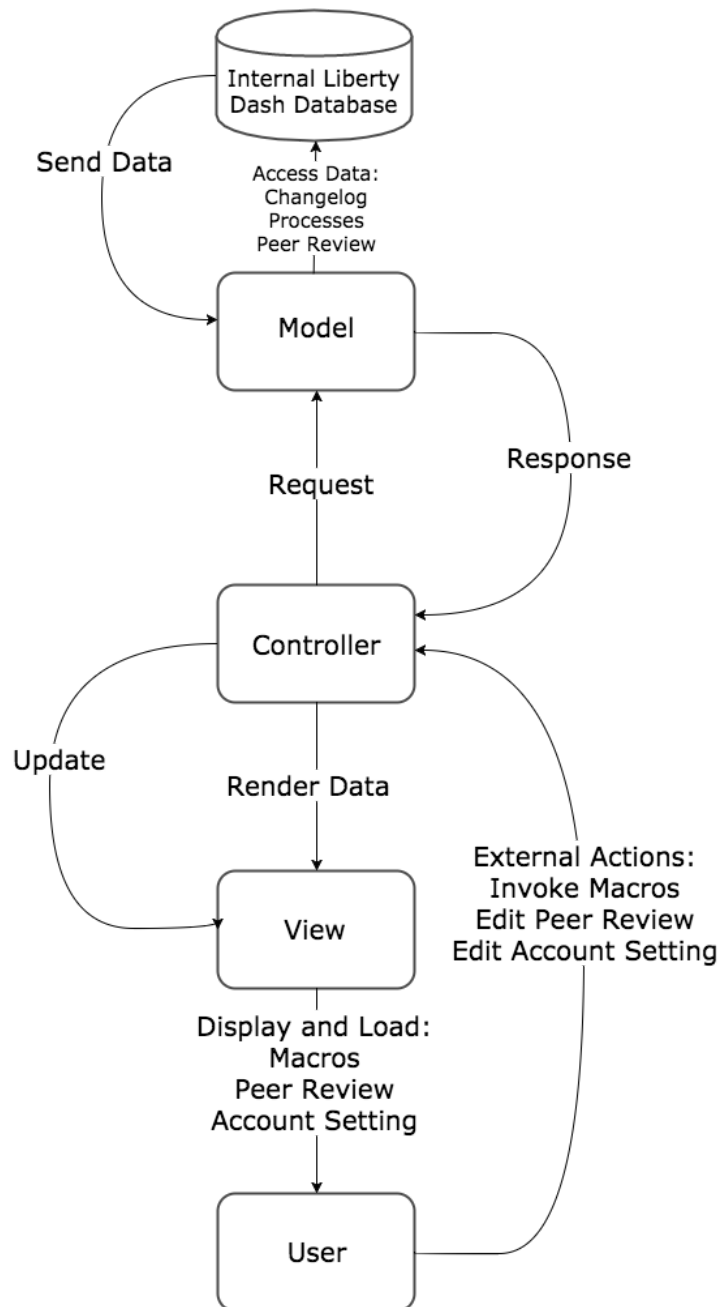


Figure 2.2 - Model View Controller

### **2.2.1 Rationale**

Using a Model-View-Controller (MVC) architecture is commonly used for web applications because it separates the logic from the views. The user interface is concerned with the visual aesthetic and presentation of information, while the logic side is concerned with accessing, transferring, and manipulating this information. Generally, the programming involved with these components are very different and not done concurrently. MVC allows these components to be divided, so that the controller interacts with the model and the view separately. For development purposes, one component is not dependent on the other to function and each can be written and modified more simply; this in turn provides an easier and more reliable method for the code to be tested.

In this system, the User instructs the Controller which sends requests to the Model to be forwarded to the Internal Liberty Dash Database (where we can access the ChangeLog table and Peer Reviews). From here, the Model responds to the Controller which in turn updates the View.

### **2.2.2 Model**

The Model is part of the backend of our system that contains and queries the Internal Liberty Dash Database. This database will return the requested data, which the Model then passes to the Controller to be handled.

### **2.2.3 View**

The View is the user interface of our web application (Refer to Section 4 for Liberty Dash User Interface mockups). By interacting with the View, the User can generate various events and requests that will be sent to the Controller to handle. For example, if a user interacts with the View by logging in, the user credentials are hashed by the client and sent to the server. The server will then authenticate with Active Directory and give an appropriate response to the client.



### **2.2.4 Controller**

The Controller separates the frontend of our system from the backend. When a User takes an action, the Controller receives instructions and either updates the View or passes on the instructions to the Model. When the Controller receives data from the Model, the Controller updates the View as necessary.

## **2.3 Libraries and Software Packages**

The section will give a specification of all the various libraries and different software packages we will be using for the development of the Liberty Dash system. For more context, refer to Figure 2.1

### **2.3.1 Node.Js**

Node.js (Node) is a simple, commonly used JavaScript development platform. Node.js is an industry standard for web application development and, because of its ubiquity, will work on almost all web browsers. Node's package manager (NPM/package.json) also provides easy access to many JavaScript packages to aid with development. By using Node.js, development is expedited. ReactJS provides a set of patched javascript libraries for cross-platform compatibility and frontend-backend abstraction. (Refer to React DOM model figure 2.1 within React.JS server)

### **2.3.2 Bootstrap**

Bootstrap is a webkit, providing many css classes and static UI elements. The core benefit of utilizing Bootstrap is the ability for the developer to easily specify the static orientation of webpages in development.

### **2.3.3 Log4JS**

Log4Js is a reliable, fast, and flexible logging framework. This framework is highly configurable through external configuration files at runtime. It views the logging process in terms of levels of priorities and offers a flexible architecture where the concept of an “appender” has been abstracted, such that custom appenders (For Example: Oracle JDBC database appender) can be easily created by a developer and loaded into the framework. Log4JS will be used as a logging abstraction layer between the system and internal MySQL database.

### **2.3.4 MySQL**

The system will implement an MySQL database, and utilize the JDBC library to interface with the database.

#### **2.3.4.1 MySQL Database**

MySQL Database is a relational database management system (DBMS) that will manage the Internal Liberty Dash Database and the External Liberty Mutual Database for the dashboard. MySQL Database is a cross-platform software that is available on most major operating systems, including but not limited to Linux, Windows, and Mac OSX. MySQL Database provides an efficient database management system which has the ability to run queries time efficiently. This allows the use for concurrency on the database with a fail safe known as row locking. Row locking will prevent multiple users from accessing and changing the same row entity at the same time. MySQL Database creates a very efficient DBMS capable of handling multiple requests and also reduces the risk of data corruption with built-in safeguards.

#### **2.3.4.2 Oracle JDBC**

The Oracle JDBC library will be used by Liberty Dash to interface with both the Internal Liberty Dash Database and the External Liberty Mutual Database. The Oracle JDBC is an interface library designed by Oracle, which will guarantee compatibility with our MySQL Databases.

### **2.3.5 Azure Active Directory Authentication**

The Microsoft Azure Active Directory Authentication Library will be used to interface with the Liberty Mutual Active Directory servers. This library supports a variety of different programming languages, including javascript and Node.JS.

#### **2.3.5.1 ADAL-Node**

Since Node.JS is the server module that will be used, the adal-node package available through NPM will be used instead of the adal-js javascript library. The benefit of using the NPM package over the native javascript library is that NPM allows for version control over the exact version of the library that will have been confirmed to work in the system. This allows for standardization amongst developers and for less native library code to reside in the project git repository.

### 3 - High Level Design

This section provides an overview of the Liberty Dash's high level design. This includes an explanation of what our client does, as well as the design of the Internal Liberty Dash Database.

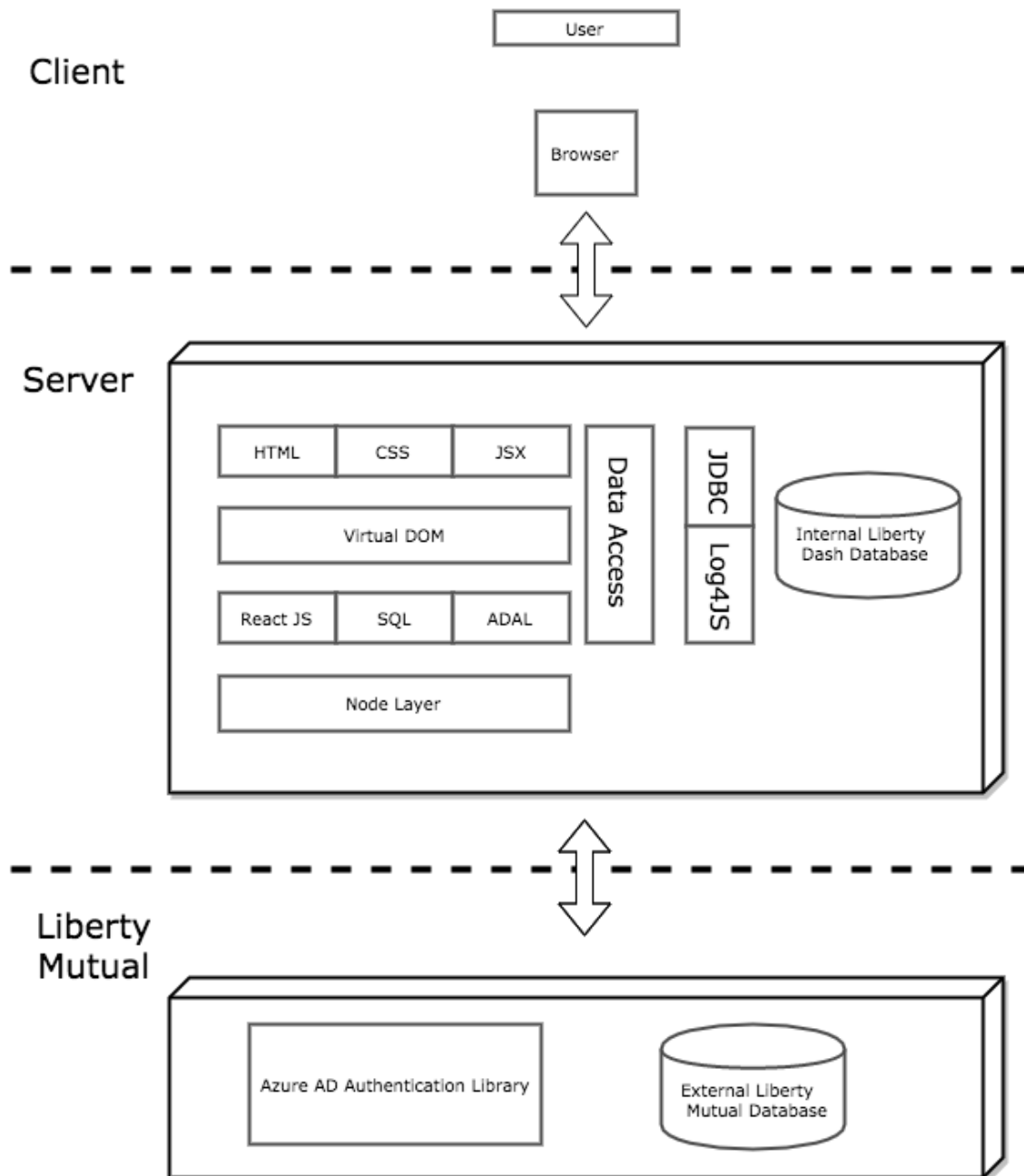
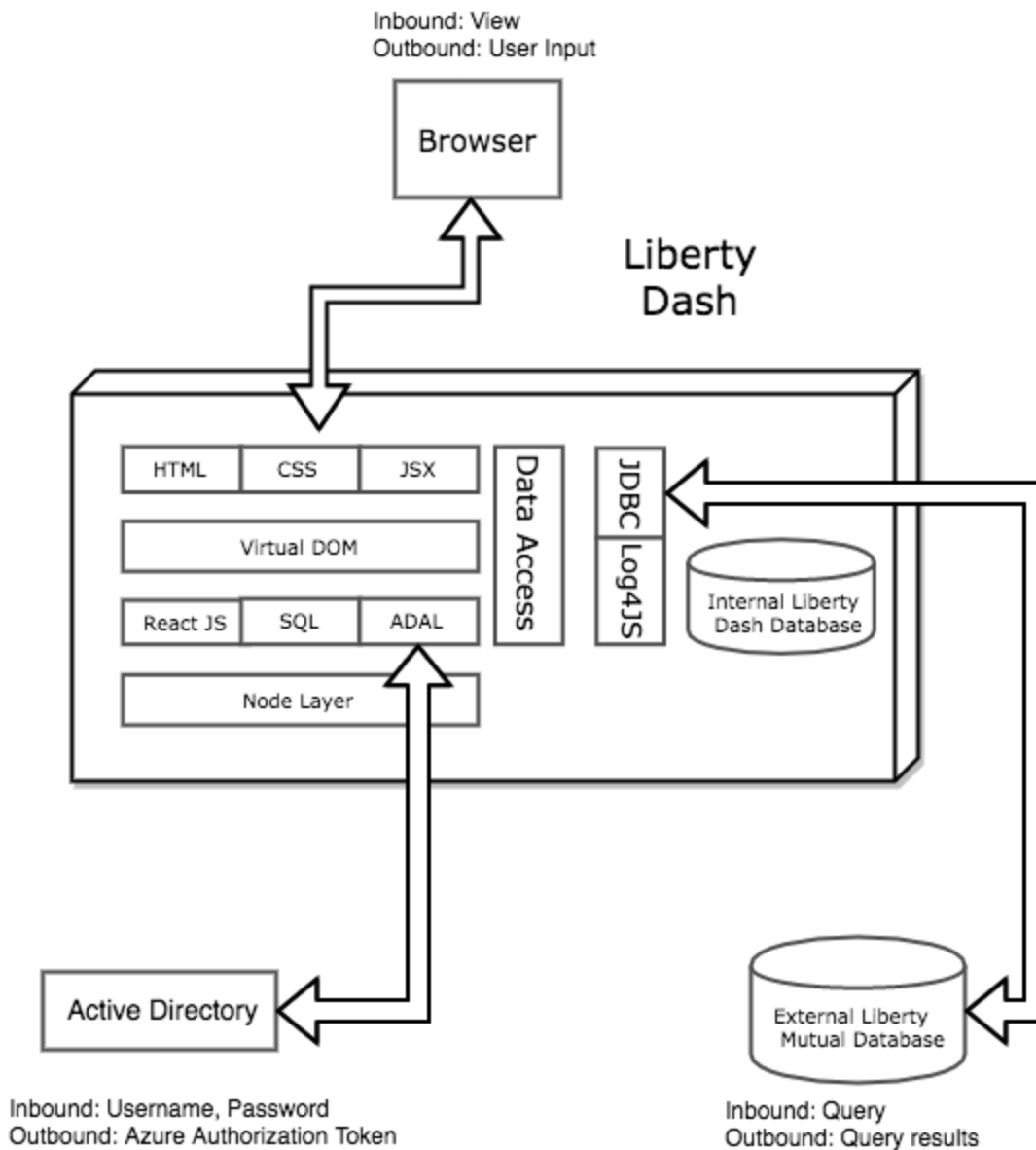


Figure 3.0 - Architecture Layers



**Figure 3.1 - External Communication**

### 3.1 Client

The Liberty Dash system provides a simple interface for the execution of macros that manipulate the data stored in the External Liberty Mutual Database. All users will be able to schedule future macro execution against the External Liberty Mutual Database after being approved by a peer user through the Peer Review process. Users will be sent Peer Review requests, which they can then choose to either approve or decline.

## 3.2 Databases

The External Liberty Mutual Database will be a relational database consisting of multiple tables. The External Liberty Mutual Database's tables will consist of the data being manipulated by our dashboard. This database will only interface with the server through SQL queries. NodeJs provides the correct drivers for MySQL, which will be used to establish a connection with the database. The MySQL Database queries will be called through the following database macros:

1. PM\_EDW\_META\_D.M\_DL\_DR\_SCHED\_RN
2. PM\_EDW\_META\_D.M\_UD\_DR\_SCHED\_START\_RN\_AID
3. PM\_EDW\_META\_D.M\_UD\_DR\_SCHED\_STTS\_RN\_AID
4. PM\_EDW\_META\_D.M\_UD\_DR\_SCHED\_VAL\_END\_RN\_AID
5. PM\_EDW\_META\_D.M\_UD\_DR\_SCHED\_VAL\_START\_RN\_AID
6. PM\_EDW\_META\_D.M\_UD\_DR\_SCHED\_SLA\_AID
7. PM\_EDW\_META\_D.M\_UD\_DR\_SCHED\_SLA\_RN
8. PM\_EDW\_META\_D.M\_UD\_DR\_SCHED\_H\_SLA\_RN
9. PM\_EDW\_META\_D.M\_DL\_DR\_STEP\_RN
10. PM\_EDW\_META\_D.M\_DL\_DR\_STEP\_RN\_GN
11. PM\_EDW\_META\_D.M\_DL\_DR\_STEP\_RN\_SID
12. PM\_EDW\_META\_D.M\_DL\_DR\_STEP\_DTL\_RN
13. PM\_EDW\_META\_D.M\_UD\_DR\_STEP\_DTL\_RN\_GN
14. PM\_EDW\_META\_D.M\_UD\_DR\_STEP\_DTL\_RN\_STPDTLID
15. PM\_EDW\_META\_D.M\_UD\_DR\_STEP\_ASI\_SID
16. PM\_EDW\_META\_D.M\_UD\_DR\_STEP\_ASI\_RN\_SID
17. PM\_EDW\_META\_D.M\_UD\_DR\_STEP\_ASI\_RN
18. PM\_EDW\_META\_D.M\_UD\_DR\_STEP\_ASI\_RN\_GN

The Internal Liberty Dash Database is a database located inside our system that stores the Changelog table, Peer Review table, and information on processes that have run and indication of whether the process completed or encountered errors. It interfaces with the client using SQL queries. When a macro is invoked, the Internal Liberty Dash Database stores information regarding the accompanying Peer Review request in the Peer Review table. This information includes the time of the request, the owner (who requested the peer review), the reviewer (who is being asked to review the request), the parameters, and (if applicable) the time of approval. Upon a macro's completion, an entry recording its ID and time of completion is stored in the Changelog. The information associated with the process's exit status (completed/failed) is stored in the process table.

### 3.2.1 Internal Liberty Dash Database Schema

Figure 3.2.1 below shows the relationships of the entities stored in the Internal Liberty Dash Database.

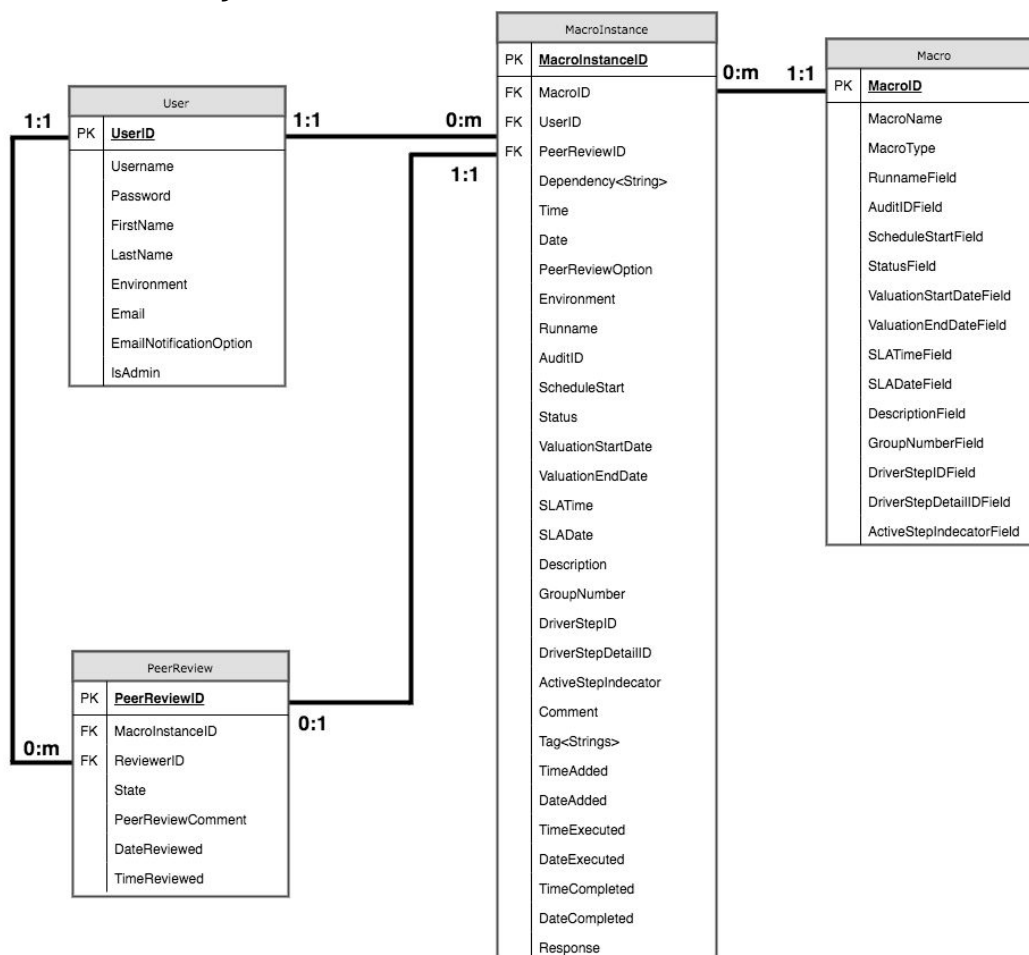


Figure 3.2.1: Internal Liberty Dash Database ER Diagram

The structure of the Internal Liberty Dash Database consists of multiple relational database tables. The data dictionary for these tables are outlined below.

Entity Name	Entity Description	Column Name	Column Description	Data Type	Length
User	A user with access to all basic functionalities of the system.	UserID	The unique identification number for each individual User	Integer	25
		UserName	The unique account name of the User used to log in	Varchar	25
		Password	The password for the User to log in	Varchar	25
		FirstName	First name of the User	Varchar	25
		LastName	Last name of the User	Varchar	25
		Environment	The Environment that the User is working in (D: Development, T: Test, Q: Quality Assurance, P: Procudtion)	Character	1
		Email	The email address of the User used for Notifications	Varchar	50
		EmailNotificationOption	Whether or not the User wishes to receive email notifications	Boolean	1
		IsAdmin	Whether or not the User is an Admin User	Boolean	1

**Table 3.2.1.1: User Entity**

Entity Name	Entity Description	Column Name	Column Description	Data Type	Length
PeerReview	The review of process by which submitted tasks are reviewed and either approved or rejected by other user in the system.	PeerReviewID	For the unique identification of Peer Review Information	Integer	10
		MacroInstanceID	The ID of the Macro Instance the Peer Review is related to	Integer	10
		ReviewerID	The UserID of the User that reviews the Peer Review	Integer	25
		State	The state of the Peer Review (P: Pending, A: Accepted, R: Rejected)	Character	1
		PeerReviewComment	The comment for the Peer Review (reason for rejection)	Varchar	512
		DateReviewed	The date the Macro is reviewed	Date	
		TimeReviewed	The time the Macro is reviewed	Time	

**Table 3.2.1.2: Peer Review Entity**



Entity Name	Entity Description	Column Name	Column Description	Data Type	Length
MacroInstance	A Macro Instance is created by an User with a value for each parameter	MacroInstanceID	For the unique identification of Macro Instance information	Integer	10
		MacroID	The ID of the Macro	Integer	10
		UserID	The UserID of the owner of the Macro Instance	Varchar	25
		PeerReviewID	The ID of the Peer Review the Macro Instance is related to	Integer	10
		Dependency<String>	The list of dependencies that the Macro Instance is related to	List<String>	
		Time	The time the Macro Instance is submitted	Time	
		Date	The date the Macro Instance is submitted	Date	
		PeerReviewOption	Whether or not the Macro Instance needs to be Peer Reviewed	Boolean	1
		Environment	The environment that the Macro Instance is working in (D:Development, T: Test, Q: Quality Assurance, P: Production)	Character	1
		Runname	The runname parameter for the Macro Instance	Varchar	52
		AuditID	Audit ID parameter for the Macro Instance	Varchar	10
		ScheduleStart	Schedule start parameter for the Macro Instance	Date	
		Status	Status parameter for the Macro Instance	Varchar	26
		ValuationStartDate	Valuation start date parameter for the Macro Instance	Date	
		ValuationEndDate	Valuation end date parameter for the Macro Instance	Date	
		SLATime	SLA time parameter for the Macro Instance	Time	
		SLADate	SLA date parameter for the Macro Instance	Date	
		Description	Description parameter for the Macro Instance	Varchar	256
		GroupNumber	Group number parameter for the Macro Instance	Varchar	5
		DriverStepID	Driver step ID parameter for the Macro Instance	Varchar	20
		DriverStepDetailID	Driver step detail ID parameter for the Macro Instance	Varchar	26
		ActiveStepIndicator	Active step indicator parameter for the Macro Instance	Varchar	26
		Comment	The comment of the change	Varchar	512
		Tag<Strings>	A list of the Tags the User added to associate with the process.	List<String>	
		TimeAdded	The time the process is added	Time	
		DateAdded	The date the process is added	Date	
		TimeExecuted	The time the process is executed	Time	
		DateExecuted	The date the process is executed	Date	
		TimeCompleted	The time the process is completed	Time	
		DateCompleted	The date the process is completed	Date	
		Response	The response from the External Liberty Mutual Database (Accepted, Rejected or Pending)	Varchar	10

**Table 3.2.1.3: MacroInstance Entity**

Entity Name	Entity Description	Column Name	Column Description	Data Type	Length
Macro	Macro Types that are supported and require parameters fields	MacroID	For the unique identification of Macro Information	Integer	10
		MacroName	The name of the Macro	Varchar	255
		MacroType	The type of the Macro (D:Delete, E:Edit, V:View)	Character	1
		RunnameField	Runname field is required for the Macro	Boolean	1
		AuditIDField	Audit ID parameter field is required for the Macro	Boolean	1
		ScheduleStartField	Schedule start parameter field is required for the Macro	Boolean	1
		StatusField	Status parameter field is required for the Macro	Boolean	1
		ValuationStartDateField	Valuation start date parameter field is required for the Macro	Boolean	1
		ValuationEndDateField	Valuation end date parameter field is required for the Macro	Boolean	1
		SLATimeField	SLA time parameter field is required for the Macro	Boolean	1
		SLADateField	SLA date parameter field is required for the Macro	Boolean	1
		DescriptionField	Description parameter field is required for the Macro	Boolean	1
		GroupNumberField	Group Number parameter field is required for the Macro	Boolean	1
		DriverStepIDField	Driver step ID parameter field is required for the Macro	Boolean	1
		DriverStepDetailIDField	Driver step detail ID parameter field is required for the Macro	Boolean	1
		ActiveStepIndicatorField	Active step indicator parameter field is required for the Macro	Boolean	1

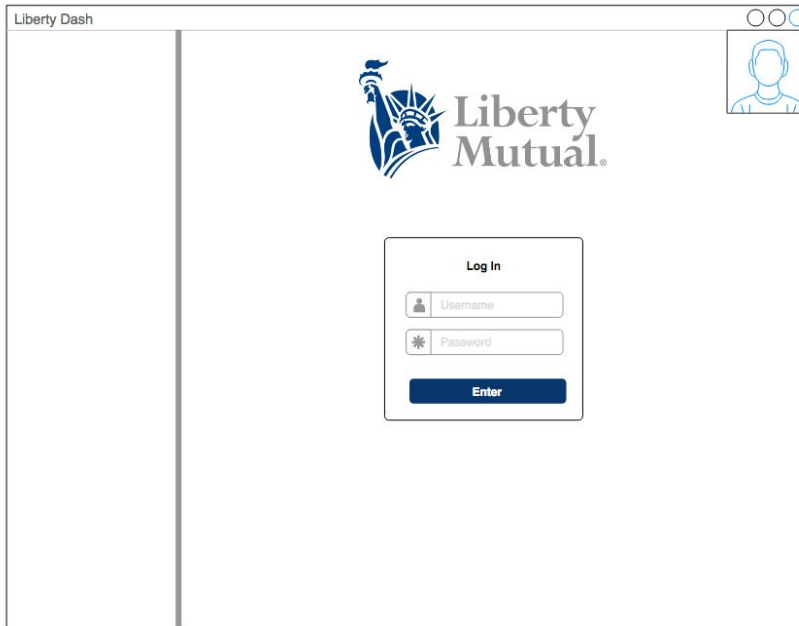
**Table 3.2.1.4: Macro Entity**

## 4 - GUI Design Mockups

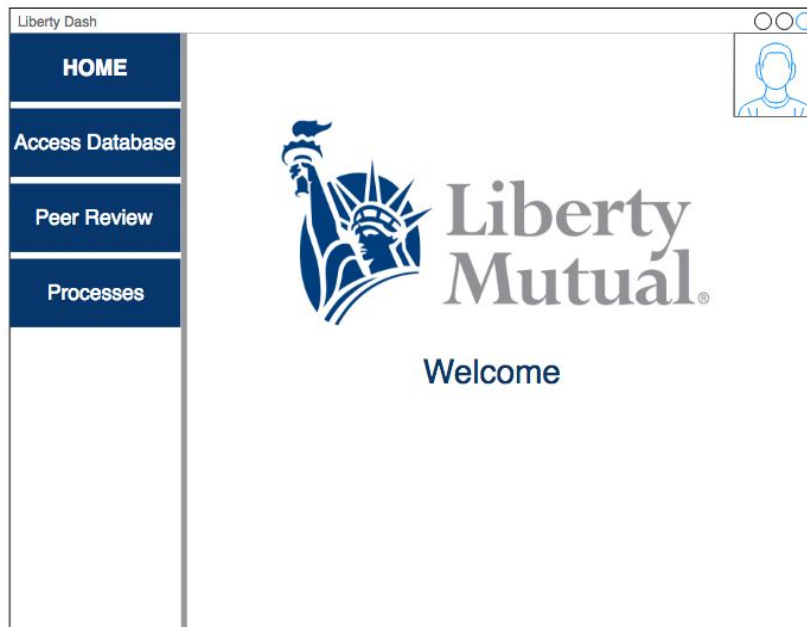
This section shows the mockups for the User Interface of the Liberty Dash system. Table 4.0 labels the use case(s) in our Software Requirements Specification (SRS) document that reference the below mockups.

Figure	Use Case(s)
4.01 - "Login" Page	3.1.14
4.02 - "Home" Page	3.1.1
4.03 - "Invoke Macro" Page	3.1.2
4.04 - "Changelog" Page	3.1.3
4.05 - "Pending Peer Reviews" Page	3.1.6
4.06 - "Accept or Reject Pending Peer Reviews" Page	3.1.5
4.07 - "Submitted Peer Reviews" Page	3.1.4
4.08 - "Resubmit Invoke Macro" Page	3.1.2, 3.1.11
4.09 - "Pending Processes" Page	3.1.6
4.10 - "Processes on Hold" Page	3.1.7
4.11 - "Running Processes" Page	3.1.8
4.12 - "Successful Processes" Page	3.1.9
4.13 - "Failed Processes" Page	3.1.10
4.14 - Processes Failure Error Message	3.1.16
4.15 - "Search Processes" Page	3.1.12
4.16 - Account Dropdown	3.1.17
4.17 - "Notifications" Page	3.1.16
4.18 - "Settings" Page	3.1.17
4.19 - Log Out Message	3.1.15
4.20: "Are you sure?" Prompt	3.1.2, 3.1.5, 3.1.11, 3.1.13, 3.1.15

**Table 4.0: SRS Use Case References**



**Figure 4.01 - “Login” Page**



**Figure 4.02 - “Home” Page**

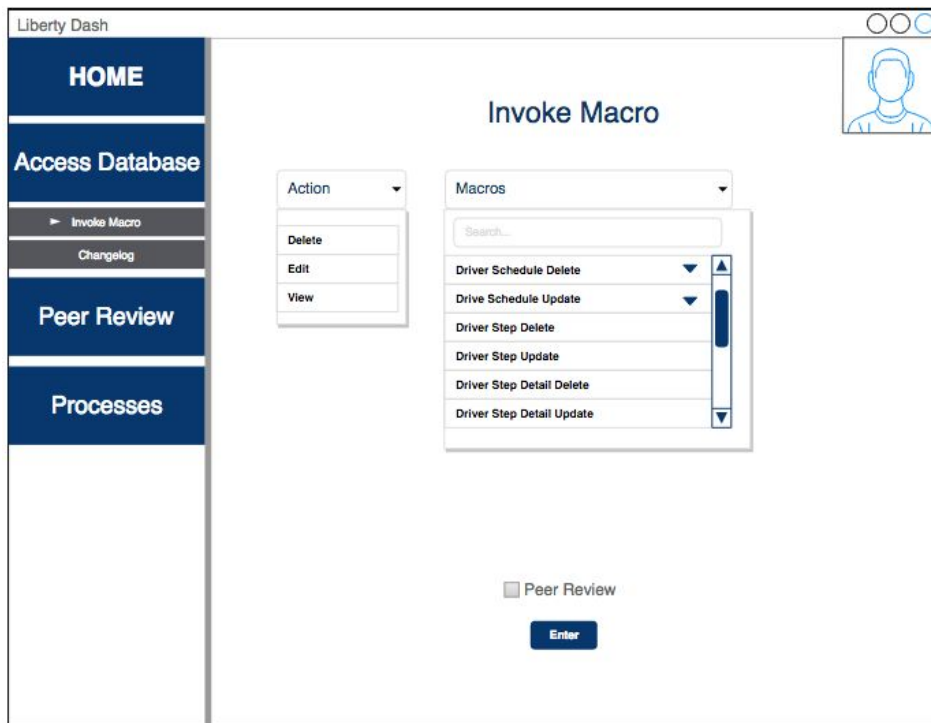


Figure 4.03 -“Invoke Macro” Page

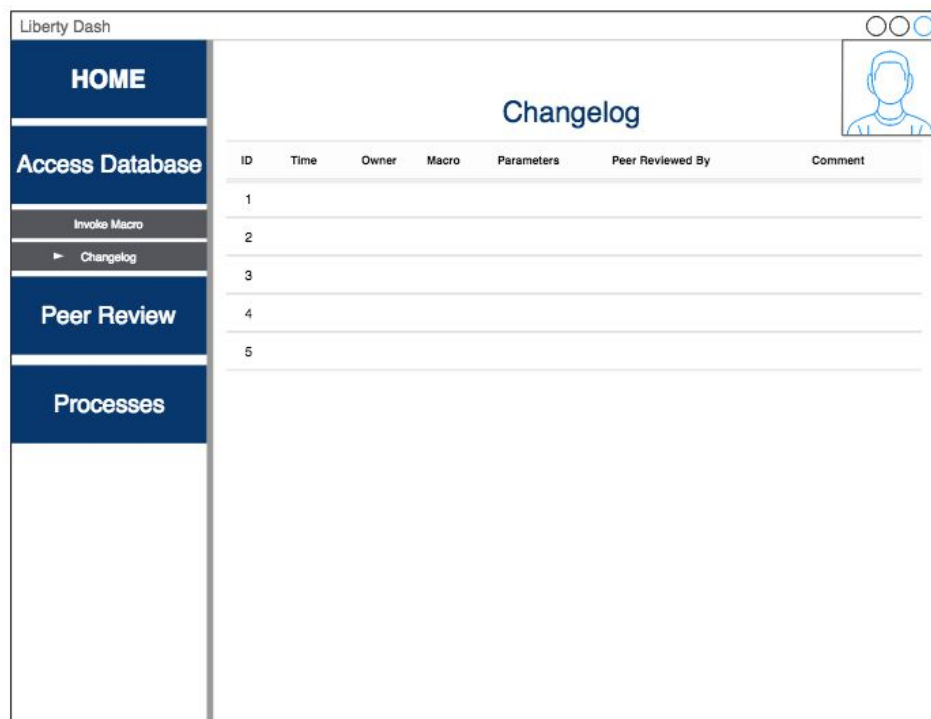


Figure 4.04 - “Changelog” Page

Liberty Dash

HOME

Access Database

Peer Review

► Pending Peer Reviews

Submitted Peer Reviews

Processes

○

○

○

Pending Peer Reviews

ID	Owner Name	Reviewer
1		
2		
3		

**Figure 4.05 - “Pending Peer Reviews” Page**

Liberty Dash

HOME

Access Database


Peer Review

➤ Pending Peer Reviews

Submitted Peer Reviews

Processes

○ ○ ○ ○



Accept or Reject Peer Review

Parameters

☐ Accept
 ☐ Reject

Notes

Submit

**Figure 4.06 - “Accept or Reject Pending Peer Reviews” Page**

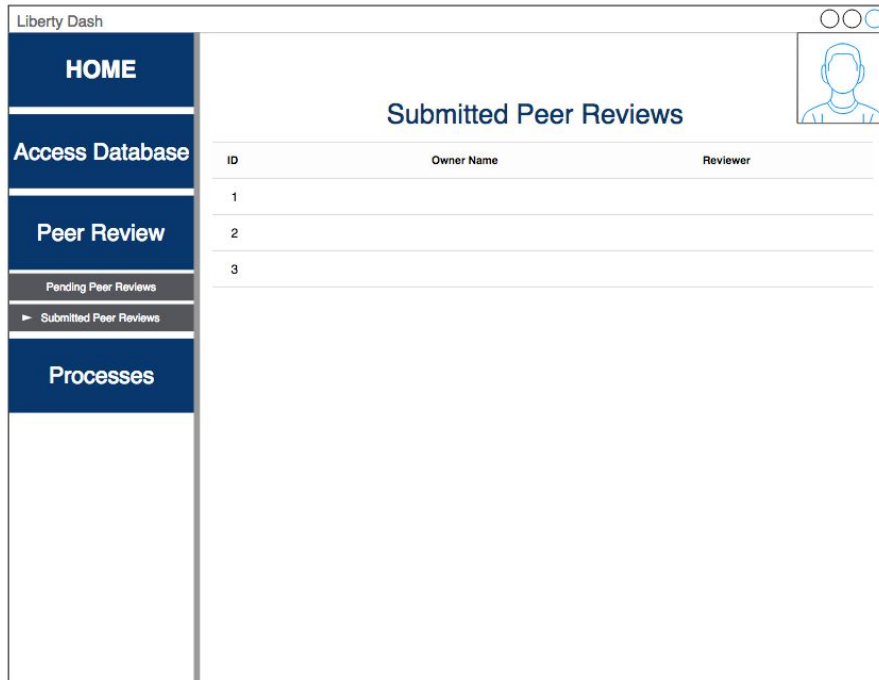


Figure 4.07 - "Submitted Peer Reviews" Page

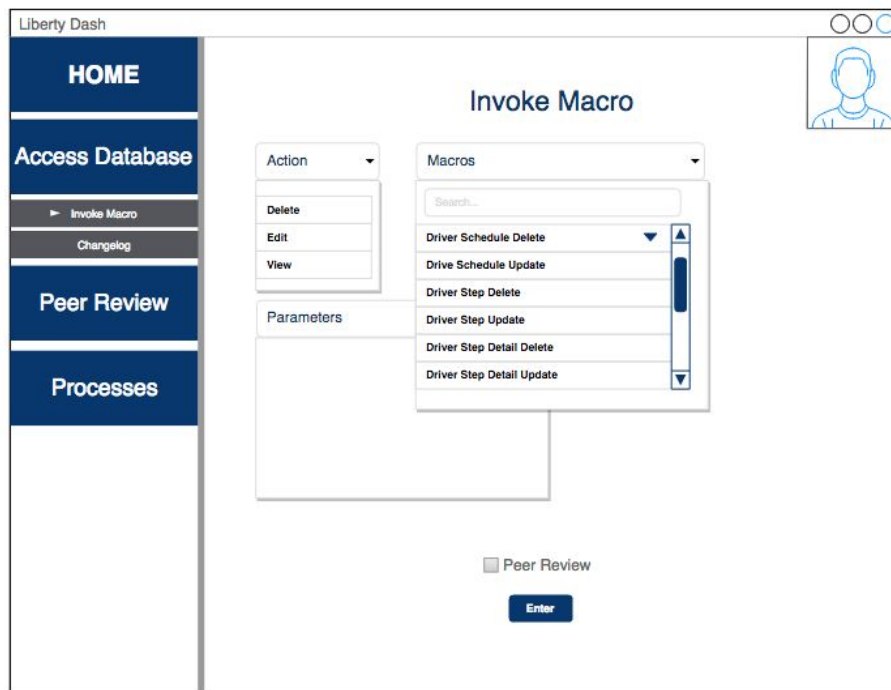


Figure 4.08 - "Resubmit Invoke Macro" Page

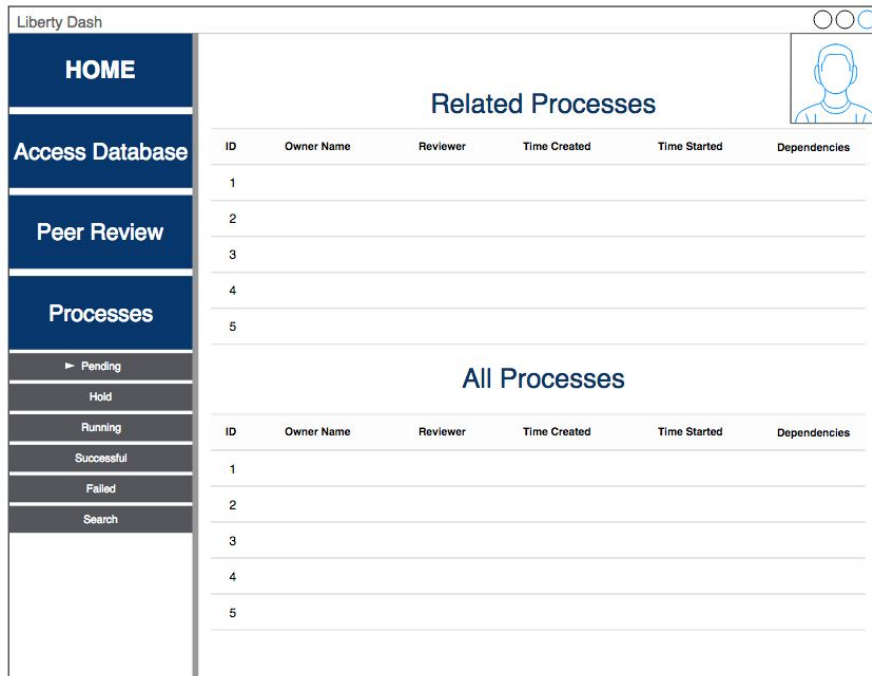


Figure 4.09 - “Pending Processes” Page

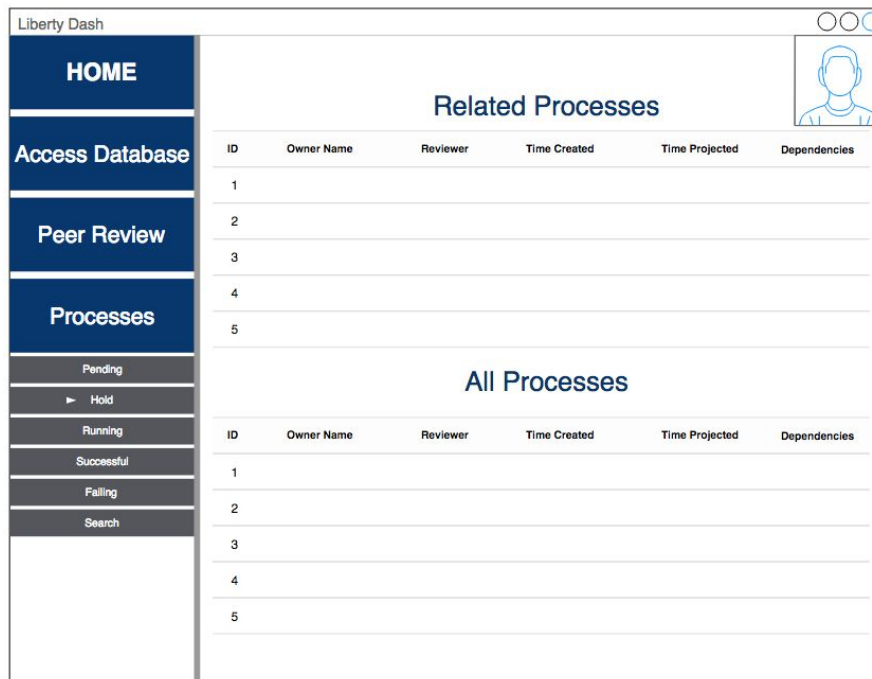


Figure 4.10 - “Processes on Hold” Page




Liberty Dash						
<div>HOME</div> <div>Access Database</div> <div>Peer Review</div> <div>Processes</div> <div>Pending</div> <div>Hold</div> <div>▶ Running</div> <div>Successful</div> <div>Falling</div> <div>Search</div>	Related Processes					
	ID	Owner Name	Reviewer	Time Created	Time Started	Dependencies
	1					
	2					
	3					
	4					
	5					
	All Processes					
	ID	Owner Name	Reviewer	Time Created	Time Started	Dependencies
	1					
	2					
	3					
	4					
	5					

Figure 4.11 - “Running Processes” Page


Liberty Dash						
<div>HOME</div> <div>Access Database</div> <div>Peer Review</div> <div>Processes</div> <div>Pending</div> <div>Hold</div> <div>Running</div> <div>▶ Successful</div> <div>Failed</div> <div>Search</div>	Related Processes					
	ID	Owner Name	Reviewer	Time Created	Time Finished	Dependencies
	1					
	2					
	3					
	4					
	5					
	All Processes					
	ID	Owner Name	Reviewer	Time Created	Time Finished	Dependencies
	1					
	2					
	3					
	4					
	5					

Figure 4.12 - “Successful Processes” Page

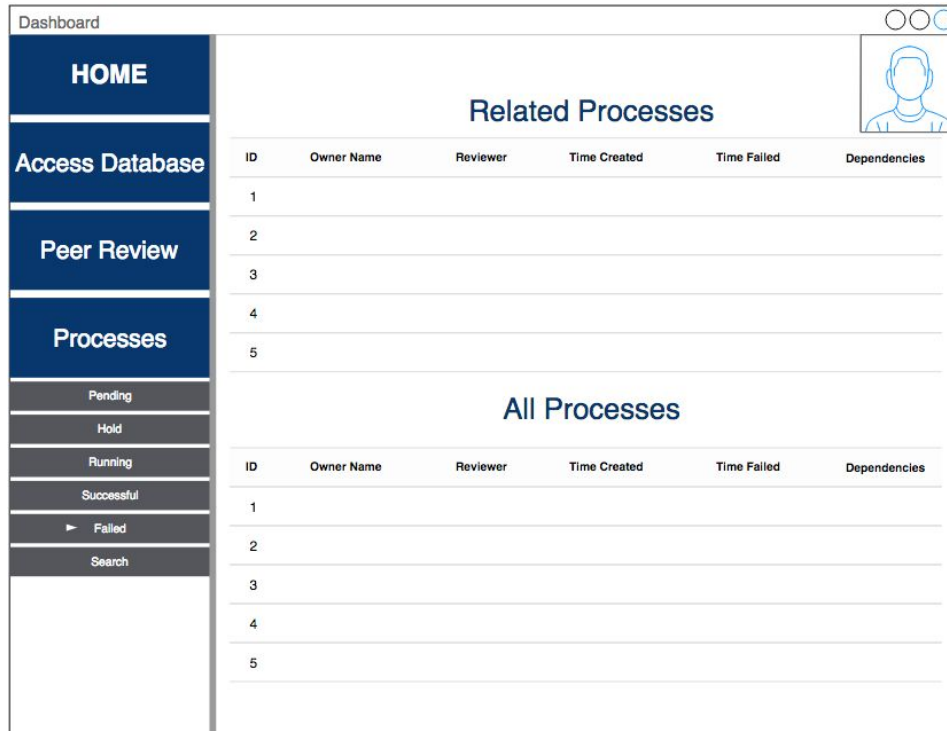


Figure 4.13 - “Failed Processes” Page



Figure 4.14 - Processes Failure Error Message

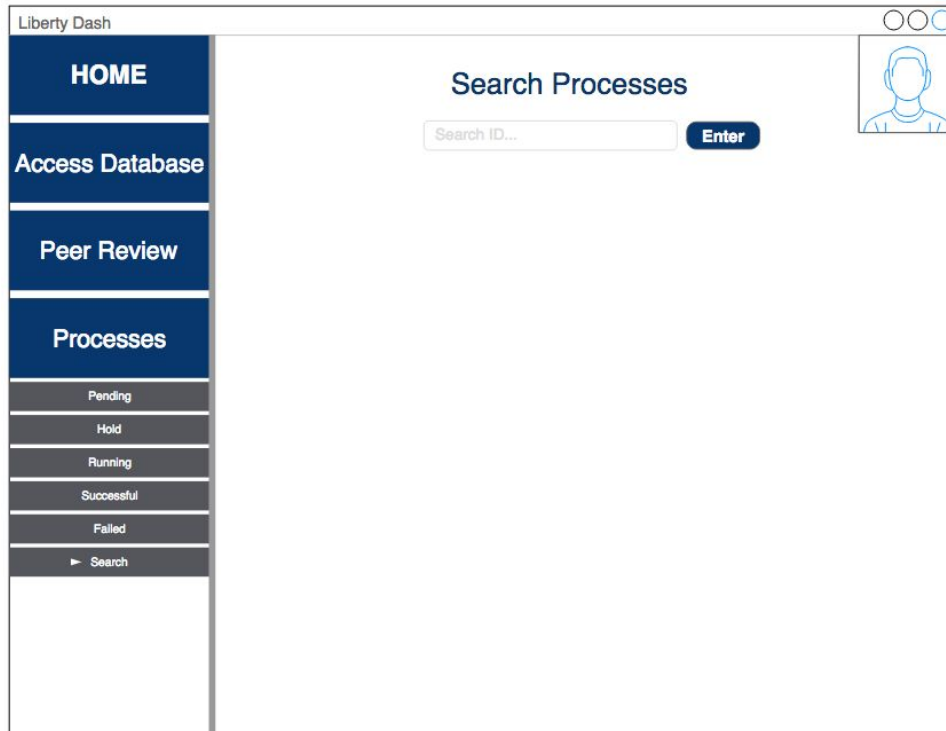


Figure 4.15 - "Search Processes" Page

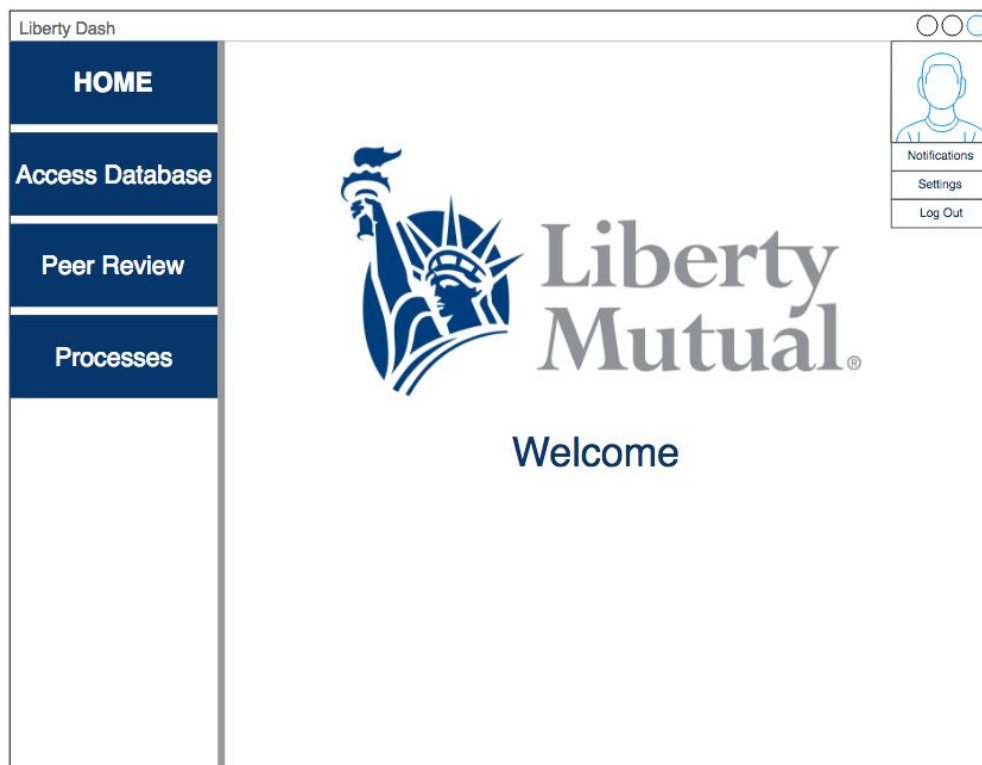


Figure 4.16 - Account Dropdown

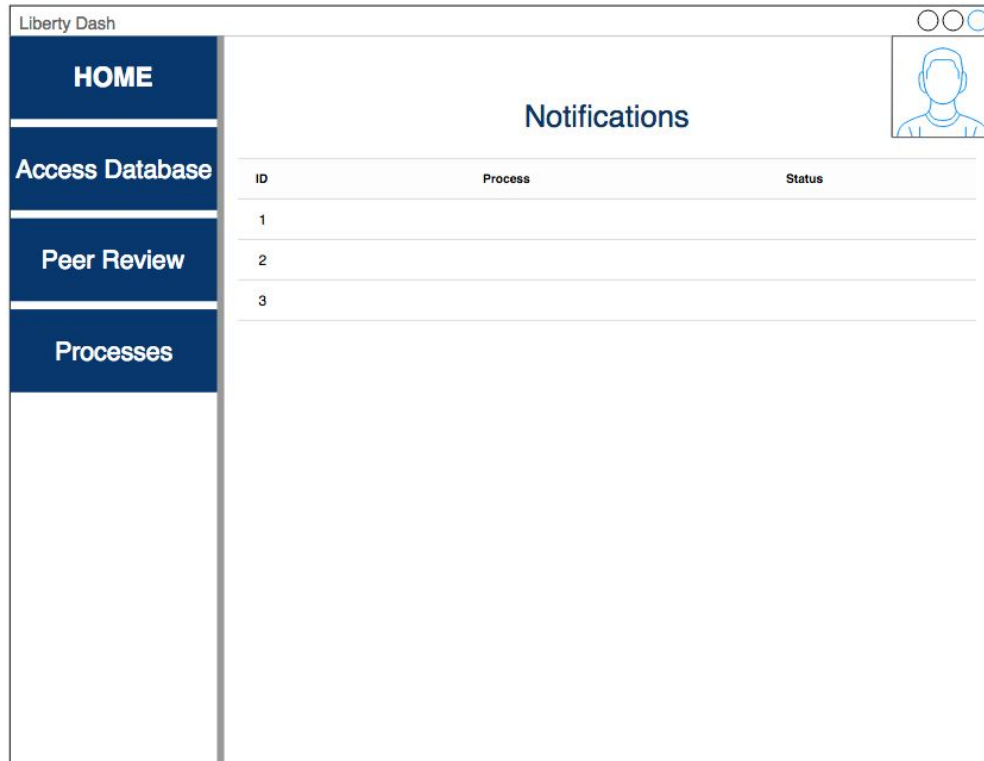


Figure 4.17 - "Notifications" Page

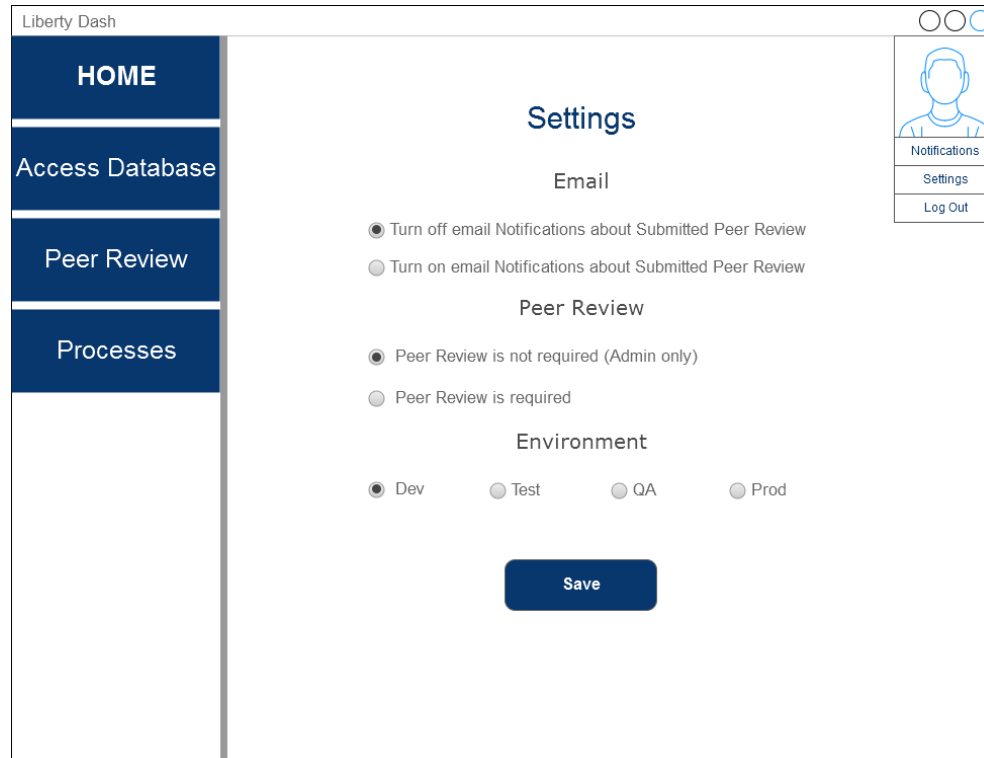
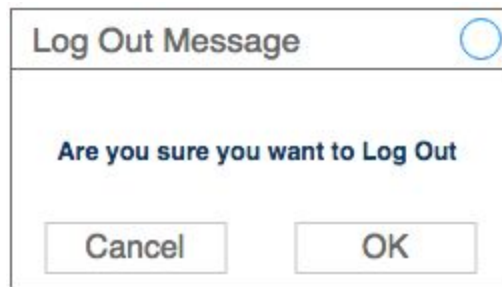


Figure 4.18 - "Settings" Page



**Figure 4.19 - Log Out Message**



**Figure 4.20: “Are you sure?” Prompt**

## 5 - Glossary

**Active Directory:** A database that stores all of an organization's user login credentials.

**Admin User:** A user with access to the same functionality as a General User, with additional permissions.

**Changelog:** A table stored in the Internal Liberty Dash Database. This table documents the changes made through the system on the External Liberty Mutual Database. The Changelog table includes information on: what was run (macro and parameters), who peer reviewed (if applicable), and a comment regarding why this macro was invoked.

**External Liberty Mutual Database:** The database from Liberty Mutual that holds Liberty Mutual's metadata. The Liberty Dash system will interact with this database, as it is external to our system.

**General User:** A user with access to all basic functionalities of the system.

**Internal Liberty Dash Database:** The database internal to our system that stores the Changelog, processes information, and Peer Reviews that have been requested.

**NPM:** Node Package Manager

**Peer Review:** The process by which submitted tasks are reviewed and either approved or rejected by other user in the system. By default for all macros Peer Review will be selected. However, if an administrator allows modifies the settings to allow developers to toggle it, when attempting to invoke a macro the developers will be unable to uncheck the "Peer Review" button to bypass this system in emergencies.

**User:** A General User or Admin User who accesses our system by logging in through the Microsoft Azure Active Directory library.

