

Data Dash



Team Blue Moose Requirements Specification

Team Manager

Ethan Miller

Team Members:

Corwin Burdick

David Dzgoev

Ajibola Famuyibo

Ethan Fine

Harry Juda

Sean Morris

Abdurrahman Munir

Dakota Orion

Chris Raff

Jonathan Shum

Table of Contents

1. Introduction	3
2. Overall System Description	4
3. Functional Requirements	6
3.1: Run A Macro	6
3.2: Peer Review Macro	8
3.3: View Past Macro History	9
3.4: Login	10
3.5: View Failures	11
3.6: View Dependencies	12
3.7: Add Driver Step Row	13
3.8: Add Driver Schedule Row	14
3.9 View Run Names	15
3.10: Display Historical Trending	16
3.11: View Job Service-level Agreements	17
3.12: View Pending Service-level Agreements Jobs	18
4. Non-Functional Requirements	19
5. Future requirements	20
6. User Interface	22
7. Glossary of terms	29

1. Introduction

For a brief overview of Data Dash and this document continue reading the introduction. For information on the system's purpose and structure please refer to the Overall System Description and the Functional Requirements sections of this document. For additional information about the system go to the Non-functional requirements and Future requirements section of this document. To view potential Wireframes/GUI options the User Interface section of this document provides images of our current vision of the potential future interface. The Data Dash system allows for the editing of information in the Liberty Mutual Environment and provides easy to access information about currently running processes and past processes in order to help the Admins and Developers more easily view and edit metadata.

The goal of this document is to provide a description of the Data Dash's software requirements and additional features. All the requirements and features that our team will be developing are expressed in this document. The Functional Requirements section will cover all possible use cases for every possible actor in the system. Non-Functional Requirements section will highlight this system's goals regarding usability, security, performance, and reliability.

When the standards of the client have been met, he/she will sign this document as an agreement to the given requirements for the new system.

2. Overall System Description

The universal driver is a common driver process that is used to manage the running of multiple applications. It uses a set of tables which specify the order that processes will be run, and the requirements necessary for a process to run. We are proposing a GUI based system that will allow a user to easily view, delete, add, and update specific rows in these tables. The manipulation of the tables is done through predefined macros that package the SQL queries necessary to manipulate the table values. The execution of these macros will be run through our system. Additionally, whenever a macro is run it will pass through a peer review process to verify the validity of the macro. Users will be able to see which processes are currently running, which have finished, and which are still waiting. Active Directory is used to handle the process of logging in and determining the status of a user. Based off of this distinction, we will be able to define the capabilities of each user; giving administrators and developers full access to the system's functionality including running macros and adding new macros. The system we are proposing will be separate from Liberty Mutual's system. It will allow users to direct the running of various processes on Liberty Mutual's database through a user-friendly GUI interface.

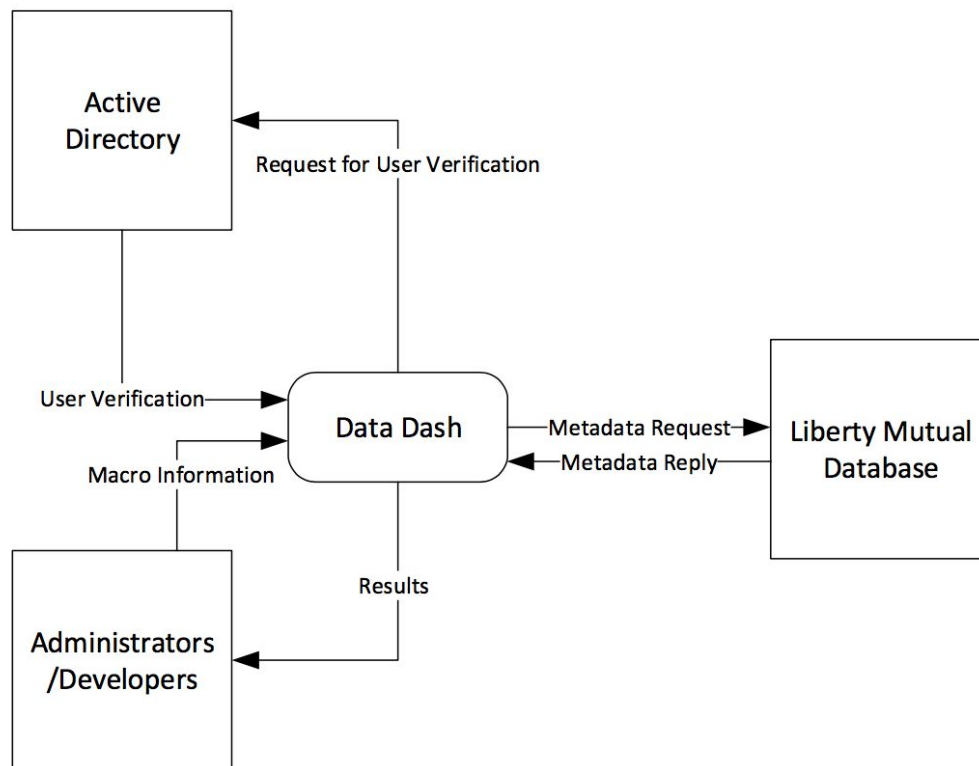


Figure 2.1: Context Diagram for the System

User Category	Capabilities
1. Developer	View Table Contents, Run Macros, Add Macros
2. Administrator	View Table Contents, Run Macros, Add Macros

Figure 2.2: Table of user types

3. Functional Requirements

Use Case ID:	3.1
Use Case Title:	Run A Macro
Description:	Executes a macro against the database with parameters filled in by the actor.
Actors:	Developer Admin
Pre-Conditions:	Actor is logged in
Post-Conditions:	Database is modified OR Macro is posted for peer review
Trigger:	Actor presses “Run Macro” button
Normal Flow:	<ol style="list-style-type: none">1. Actor selects macro from drop down menu. During this process they are provided descriptions of the macro.2. Actor selects parameters for macro one by one. Only valid parameters are shown when possible. Actor is given basic description of parameter on a best effort basis. If parameters are not valid, move to exception 2.3. Actor may check “disable peer review”. If so go to alternate flow A. If Macro always bypasses peer review, also go to flow A. If Macro always requires peer view, the “disable peer review” check box will be grayed out and so move on to step 4.4. Actor confirms that they wanted to submit the macro for peer review. Go to flow B if they cancel the submission instead.5. Actor is informed macro was submitted for peer review.
Alternate Flows:	<ol style="list-style-type: none">A. Actor submits macro for execution. Actor is asked if they are sure they want make this change, once they agree they are informed that the macro has begun execution. Go to flow B if they cancel the submission instead.B. Actor presses cancel submission and is brought back to step 2.

Exceptions:	2. Some of the parameters were not valid. The actor is displayed a message informing them of which parameters were invalid, and brought back to step 2.
--------------------	---

Created By:	Ethan Fine	Last Update By:	Corwin Burdick
Date Created:	10/8/2016	Last Revision Date:	11/17/2016
Status:	Requires Customer approval		

Use Case ID:	3.2
Use Case Title:	Peer Review Macro
Description:	Approves a macro posted for peer review for execution.
Actors:	Developer Admin
Pre-Conditions:	<ol style="list-style-type: none"> 1. Actor receives a notification within the system that a macro is available for peer review. 2. Actor is logged in.
Post-Conditions:	Macro is run against the database, and removed from peer review page or is journaled as rejected at peer review stage.
Trigger:	Actor selects peer review.
Normal Flow:	<ol style="list-style-type: none"> 1. Actor is displayed macros that need peer review and all chosen parameters for each macro. 2. Actor is given the opportunity to change any of the parameters. 3. Actor clicks approve execution button. Go to flow A If they click reject macro instead. 4. Actor is informed the macro has begun execution.
Alternate Flows:	A. Actor is informed that macro was rejected
Exceptions:	<ol style="list-style-type: none"> 4. Some of the parameters were not valid. The actor is displayed a message informing them of which parameters were invalid, and brought back to step 4.

Created By:	Ethan Fine	Last Update By:	Sean morris
Date Created:	10/10/2016	Last Revision Date:	11/17/2016
Status:	Requires Customer approval		

Use Case ID:	3.3
Use Case Title:	View Past Macro History
Description:	View the run parameters of a macro run in the past.
Actors:	Developer Admin
Pre-Conditions:	Actor is logged in.
Post-Conditions:	Actor is displayed details about a macro run or rejected at peer review stage in the past, including macro type, parameters, and the name of the actor who created and approved or rejected the macro if applicable.
Trigger:	Actor clicks history button.
Normal Flow:	<ol style="list-style-type: none"> 1. Actor selects a time period. 2. Actor is displayed a list of macros run or rejected at peer review stage in that time period with basic info: macro type, name of actor who created, name of actor who peer reviewed if applicable and run date. 3. Actor selects a macro of interest to view in further detail.
Alternate Flows:	N/A
Exceptions:	N/A

Created By:	Ethan Fine	Last Update By:	Ethan Fine
Date Created:	10/11/2016	Last Revision Date:	11/1/2016
Status:	Requires Customer approval		

Use Case ID:	3.4
Use Case Title:	Login
Description:	Actor logs into the system to gain access to all other functionality.
Actors:	Developer Admin
Pre-Conditions:	Actor had an account created with active directory, outside of our system. Actor is not logged in.
Post-Conditions:	Actor is logged in.
Trigger:	Actor visits website.
Normal Flow:	<ol style="list-style-type: none"> 1. Actor is displayed login form. 2. Actor fills in username and password. 3. Actor submits information. 4. Actor is informed that login was successful.
Alternate Flows:	N/A
Exceptions:	<ol style="list-style-type: none"> 3. Actor does not submit valid information. The system informs them that login was unsuccessful, Go to step 1.

Created By:	Ethan Fine	Last Update By:	Ethan Fine
Date Created:	10/13/2016	Last Revision Date:	10/13/2016
Status:	Requires Customer approval		

Use Case ID:	3.5
Use Case Title:	View Failures
Description:	View macro failures.
Actors:	Developer Admin
Pre-Conditions:	A macro has failed. The actor is logged in.
Post-Conditions:	Failure information is displayed to the developer or admin.
Trigger:	Actor clicks history tab.
Normal Flow:	<ol style="list-style-type: none"> 1. Actor selects the failures subsection. 2. List of failed macros are displayed . 3. Actor selects macro to see failure information including date run, user who initiated, and parameters used.
Alternate Flows:	N/A
Exceptions:	N/A

Created By:	Sean Morris	Last Update By:	Sean Morris
Date Created:	10/17/2016	Last Revision Date:	10/20/2016
Status:	Requires Customer approval		

Use Case ID:	3.6
Use Case Title:	View Dependencies
Description:	View run dependencies of each application and run name in which predecessors are defined..
Actors:	Developer Admin
Pre-Conditions:	Actor is logged in.
Post-Conditions:	Run dependencies of each application are shown to the actor.
Trigger:	Actor clicks dependencies tab.
Normal Flow:	<ol style="list-style-type: none"> 1. Actor selects app name and run name to display dependencies. 2. Dependencies are displayed. Go to flow A if run name does not have dependencies.
Alternate Flows:	A. Actor will get a message that it does not have any dependencies.
Exceptions:	N/A

Created By:	Sean Morris	Last Update By:	Sean Morris
Date Created:	10/17/2016	Last Revision Date:	10/20/2016
Status:	Requires Customer approval		

Use Case ID:	3.7
Use Case Title:	Add Driver Step Row
Description:	Add driver step row to the C_DRIVER_STEP table using the same process as executing a macro.
Actors:	Developer Admin
Pre-Conditions:	Actor is logged in.
Post-Conditions:	New driver step row is added to the C_DRIVER_STEP table.
Trigger:	Actor follows use case 3.1, selecting the “Add Driver Step Row” macro in step 1.
Normal Flow:	<ol style="list-style-type: none"> 1. Actor selects parameters as in use case 3.1 step 2. Those parameters are: APP Name, Run Name, Group Number, Group Name, Run Order Number, Path Text, Command Text, Parameter Text, Group Concurrency Indicator, Step Concurrency Indicator, Notify Text, Step Type Code, Step Name, Error Process Number 2. Actor follows the remaining steps of use case 3.1 starting with step 3.
Alternate Flows:	N/A
Exceptions:	See exceptions for use case 3.1

Created By:	Sean Morris	Last Update By:	Ethan Fine
Date Created:	10/17/2016	Last Revision Date:	11/16/2016
Status:	Requires customer approval.		

Use Case ID:	3.8
Use Case Title:	Add Driver Schedule Row
Description:	Add driver schedule row to the C DRIVER SCHEDULE table using the same process as executing a macro.
Actors:	Developer Admin
Pre-Conditions:	Actor is logged in.
Post-Conditions:	New driver schedule row is added to the C_DRIVER_SCHEDULE table.
Trigger:	Actor follows use case 3.1, selecting the “Add Driver Schedule Row” macro in step 1.
Normal Flow:	<ol style="list-style-type: none"> 1. Actor selects parameters as in use case 3.1 step 2. Those parameters are: App Name, Run Name, Run Number, Re Run Number, Scheduled Start Date Time, Status Code, Valuation Start Date Time, Valuation End Date Time 2. Actor follows the remaining steps of use case 3.1 starting with step 3.
Alternate Flows:	N/A
Exceptions:	See exceptions for use case 3.1

Created By:	Sean Morris	Last Update By:	Ethan Fine
Date Created:	10/17/2016	Last Revision Date:	11/16/2016
Status:	Requires customer clarification		

Use Case ID:	3.9
Use Case Title:	View Run Names
Description:	View currently running or past run names.
Actors:	Developer Admin
Pre-Conditions:	Actor is logged in
Post-Conditions:	N/A
Trigger:	Actor selects run names tab
Normal Flow:	<ol style="list-style-type: none"> 1. Actor selects a time period, either current or historical. 2. Actor is displayed all run names that were or are running in that time period.
Alternate Flows:	N/A
Exceptions:	N/A

Created By:	Ethan Fine	Last Update By:	Ethan Fine
Date Created:	10/18/2016	Last Revision Date:	10/20/2016
Status:	Requires Customer approval		

Use Case ID:	3.10
Use Case Title:	Display Historical Trending
Description:	Display historical trending at all levels (Batch, Job, Step)
Actors:	Developer Admin
Pre-Conditions:	Actor is logged in.
Post-Conditions:	N/A
Trigger:	Actor clicks on Trends tab.
Normal Flow:	<ol style="list-style-type: none"> 1. Actor selects level (Batch, Job, or Step). 2. Actor is given a list of Batches, Jobs, or Steps. 3. Actor can select specific Batch, Job, or Step and they are displayed information such as historical runtime averages by day.
Alternate Flows:	N/A
Exceptions:	N/A

Created By:	Ethan Fine	Last Update By:	David Dzgoev
Date Created:	10/18/2016	Last Revision Date:	10/20/2016
Status:	Requires customer clarification		

Use Case ID:	3.11
Use Case Title:	View Job Service-level Agreements
Description:	For any job you will have the ability to view the service-level agreement for it
Actors:	Developer Admin
Pre-Conditions:	Actor is logged in
Post-Conditions:	Actor is able to view SLA for a particular job.
Trigger:	Actor selects SLA tab
Normal Flow:	<ol style="list-style-type: none"> 1. Actor selects specific job 2. Actor is displayed SLA for selected Job
Alternate Flows:	N/A
Exceptions:	N/A

Created By:	David Dzgoev	Last Update By:	David Dzgoev
Date Created:	10/20/2016	Last Revision Date:	10/20/2016
Status:	Requires customer clarification.		

Use Case ID:	3.12
Use Case Title:	View Pending Service-level Agreements Jobs
Description:	View all pending service-level agreement jobs.
Actors:	Developer Admin
Pre-Conditions:	Actor is logged in
Post-Conditions:	Actor is able to view pending SLA jobs.
Trigger:	Actor selects SLA tab
Normal Flow:	<ol style="list-style-type: none"> 1. Actor selects pending tab. 2. Actor is displayed pending SLA jobs.
Alternate Flows:	N/A
Exceptions:	N/A

Created By:	Sean Morris	Last Update By:	Sean Morris
Date Created:	10/20/2016	Last Revision Date:	10/20/2016
Status:	Requires customer clarification.		

4. Non-Functional Requirements

4.1 Security

- i. Users will be able to log into the system through Active Directory.
- ii. Active Directory will dictate security groups and permissions.
- iii. Can be configured to point at different Active Directory groups.

4.2 Performance

- i. The system should not take significantly longer than the current system's implementation.

4.3 Provides Feedback

- i. The system will provide feedback when something is completed, successfully or unsuccessfully.

4.4 Usability

- i. The system should be easy to use.
- ii. The system should limit the risk of human error when executing macros as well as limit the possibility for data corruption.

4.5 Reporting

- i. Keep journal of changes made to the database when the changes are made through the system. Configurable as to which database it points at.

4.6 Concurrency

- i. Multiple users should be able to run macros at the same time without interference or error.

5. Future requirements

5.1 Historical Data

In addition to providing the historical runtime of all the run names in the system, we can use this data to extrapolate into the future and perhaps predict the performance of a run name well into the future.

5.2 Better UI

Right now, we intend to build a very basic UI for the system. However once the main aspects of the system are completed, we could revisit the UI and design/implement something more feature rich.

5.3 Macro Creation

The planned system does not support the creation of new macros through the UI. Macro creation could be supported in future iterations of the system.

5.4 Auditing

Auditing could also be implemented in the system. Details of the auditing system are not revealed in the charter.

5.5 System Diagnostics

Using historical runtime analytics information, we could also implement an automatic diagnostic tool that could provide detailed insight/information on the reason(s) why a macro is beginning to slow down.

5.6 Root Configuration

Implement the ability to configure the root directory of the project.

5.7 Peer Review Email Feature

In the future, we would include the ability for users to receive email notifications for peer review when a macro is posted by another user.

5.8 Interaction with the Unix directory

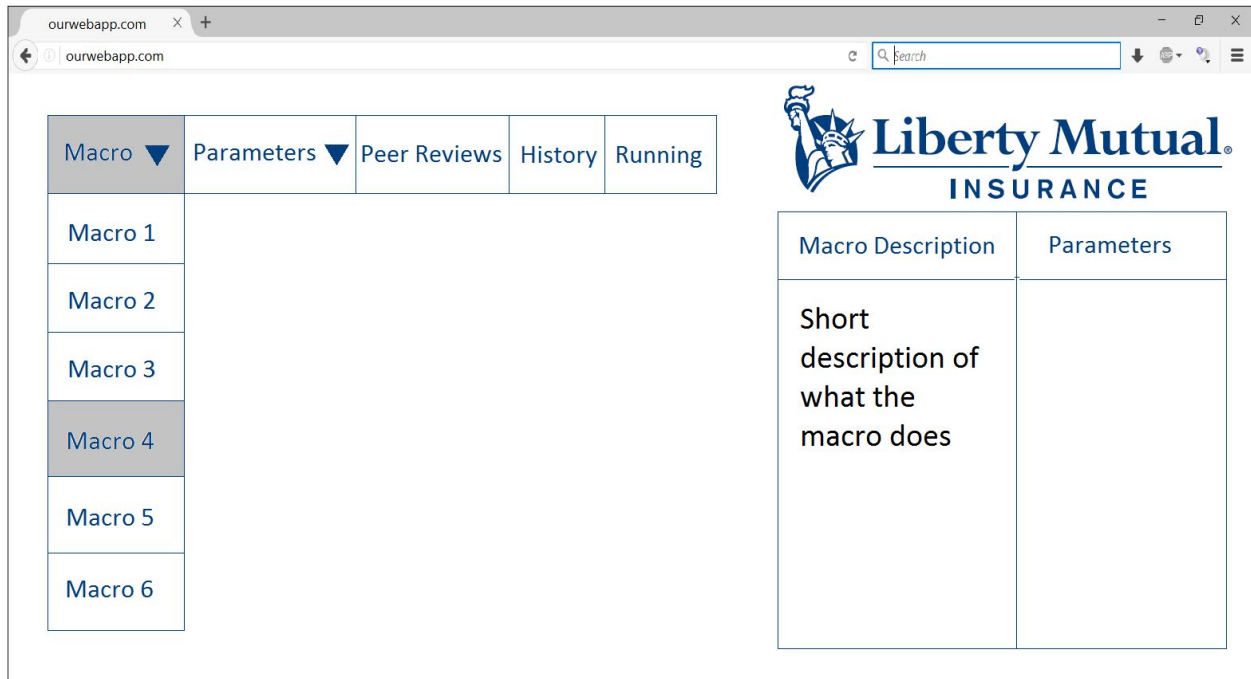
5.8.1 Runname Creation

In the future we would implement the ability to execute run name creation.

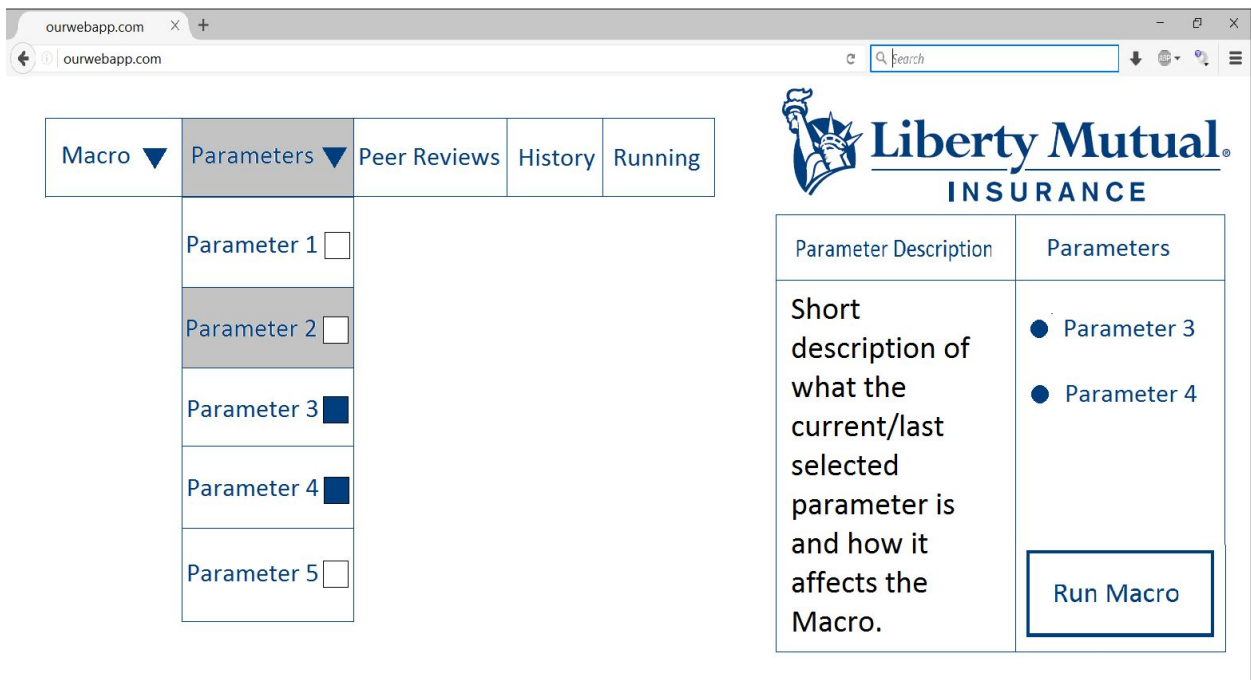
5.8.2 Scanning the Existing Database

We would also add the ability for the system to scan the unix directory for existing runnames and allow these existing runnames to be run as parameters in queries.

6. User Interface



Use Case 3.1 step 1



Use Case 3.1 step 2

ourwebapp.com

ourwebapp.com

Search

Macro ▼

Parameters ▼

Peer Reviews

History

Running

Parameter Description

Parameters

Short description of what the current/last selected parameter is and how it affects the Macro.

● Parameter 3

● Parameter 4

Run Macro

Disable Peer Review

Use Case 3.1 step 4

ourwebapp.com

ourwebapp.com

Search

Macro ▼

Parameters ▼

Peer Reviews

History

Running

ion

Parameters

Macro has been submitted for peer approval. Check its status in the "Peer Reviews" tab.

OK

Parameters

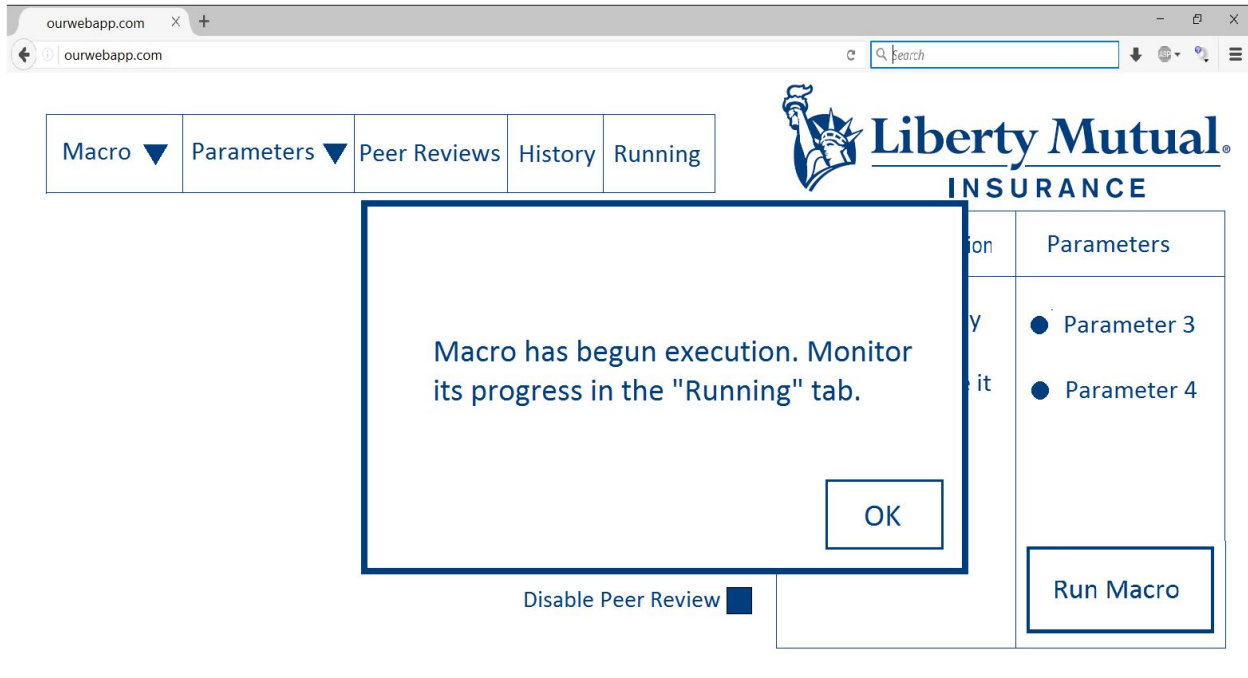
● Parameter 3

● Parameter 4

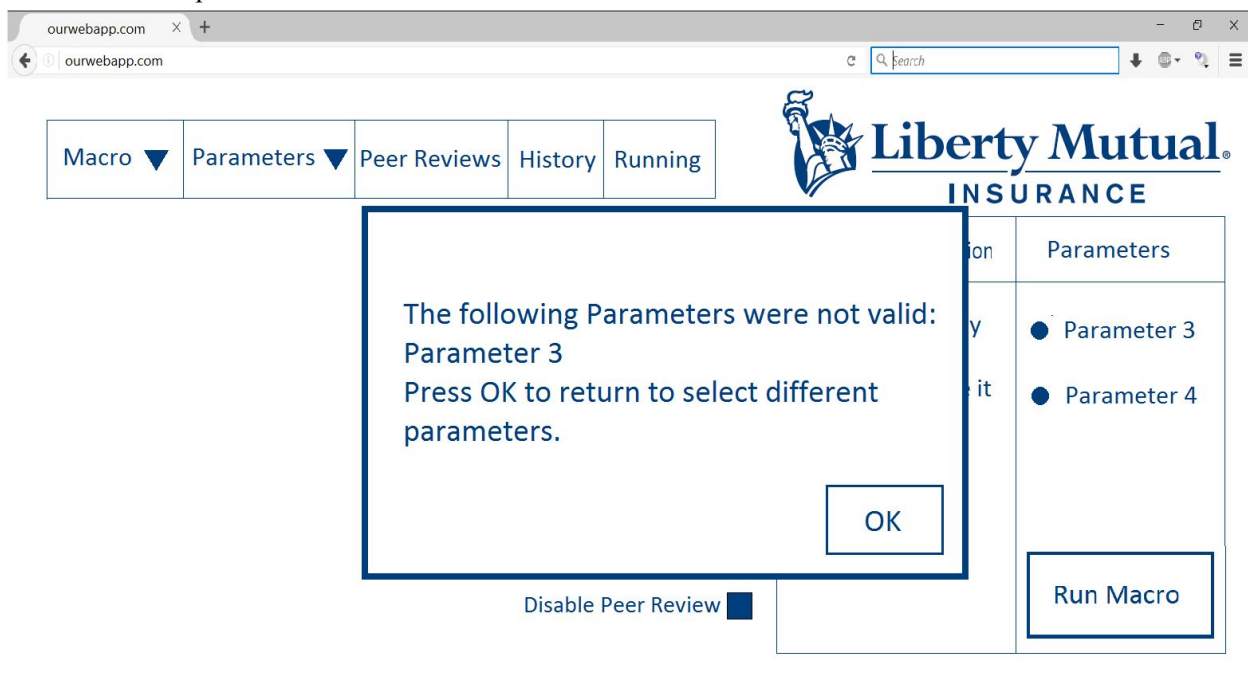
Run Macro

Disable Peer Review

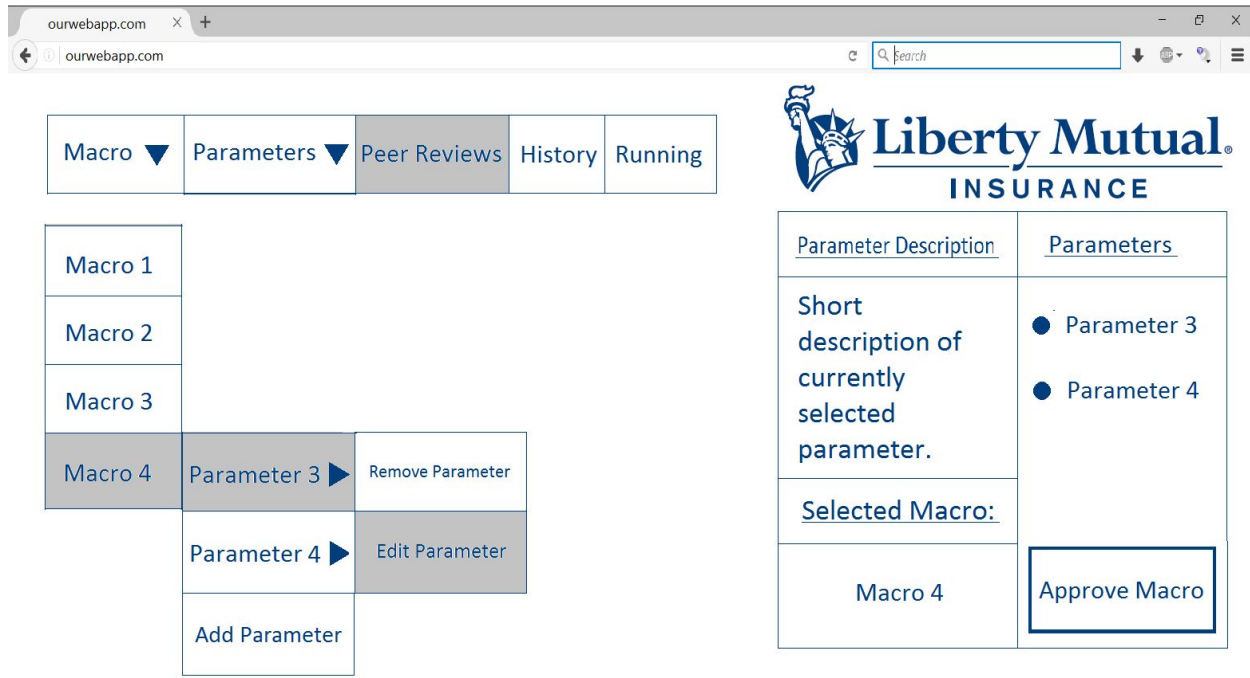
Use Case 3.1 step 5



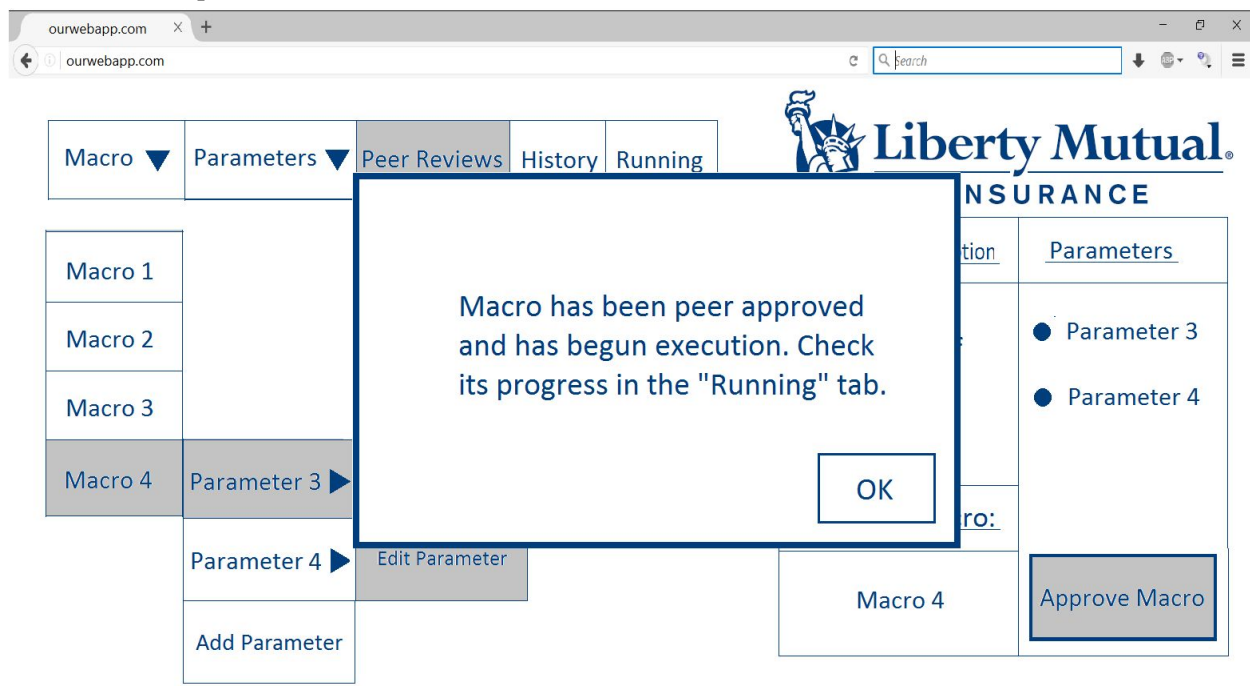
Use Case 3.1 step 3 Alternate Flow A



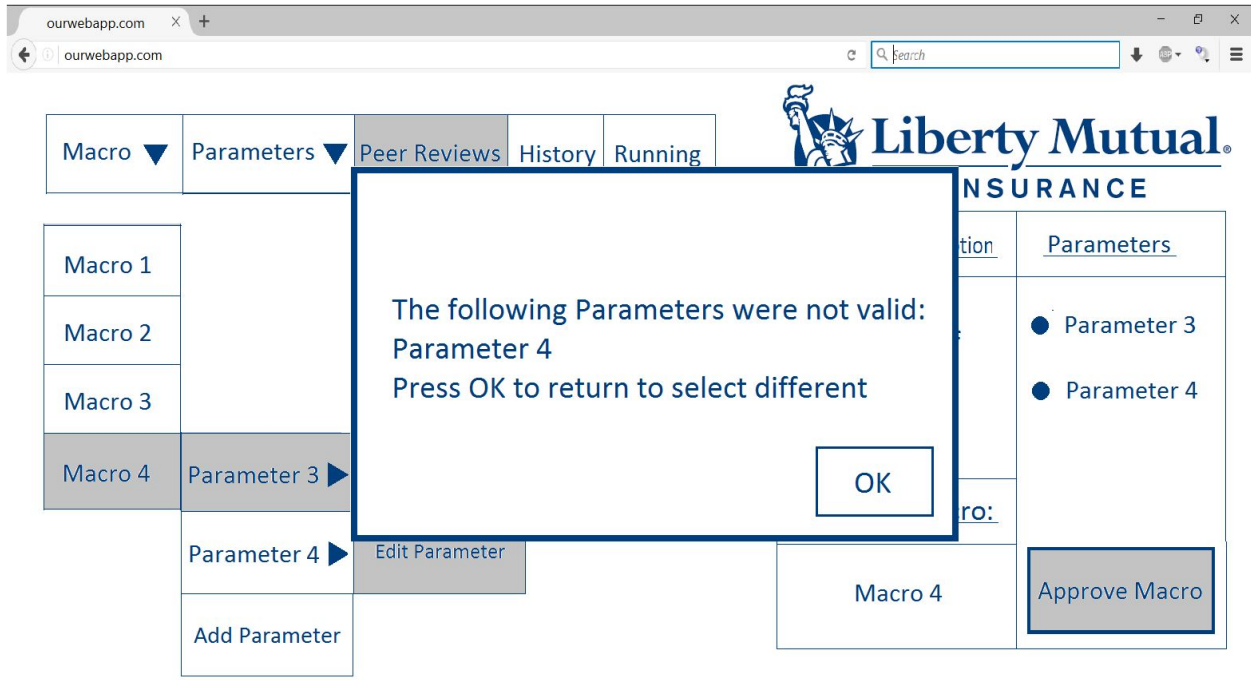
Use Case 3.1 step 4 exception 4



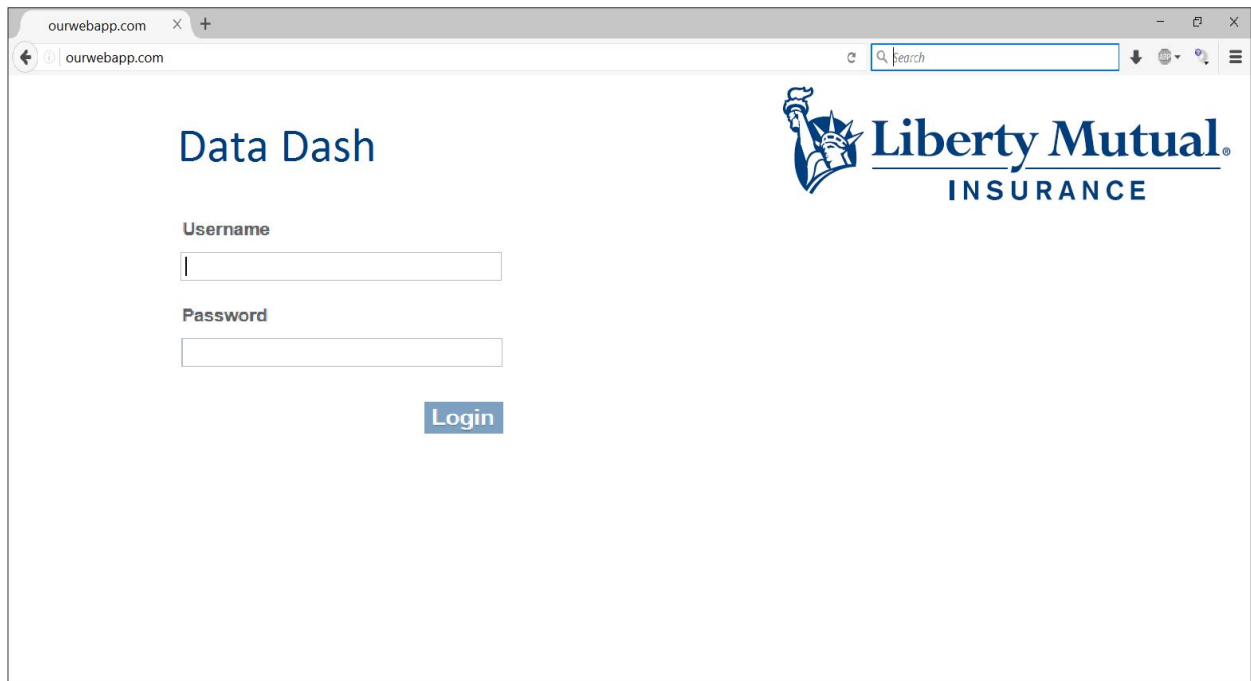
Use Case 3.2 step 4 and 5



Use Case 3.2 step 7



Use Case 3.2 step 4 exception 4




Use Case 3.4 step 1

ourwebapp.com x +

ourwebapp.com search

Data Dash



Username

Password

Login

Use Case 3.4 step 2

ourwebapp.com x +

ourwebapp.com search

Data Dash



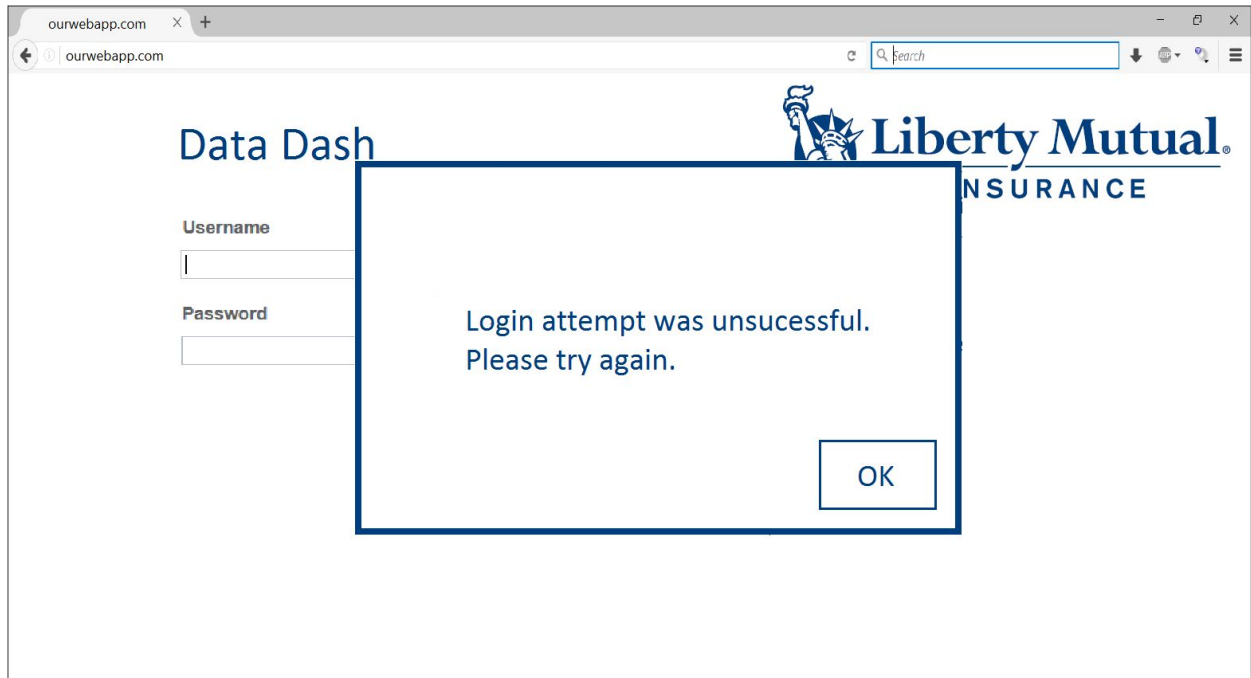
Username

Password

You have been signed in.

OK

Use Case 3.4 step 4



Use Case 3.4 step 3 exception 3.

7. Glossary of terms

Batch	A batch is similar to a run name in that it references a specific set of processes within an application. Batches have a unique Audit ID (AUDT_ID) that can be used to control the scheduling of an APP_NME, RUN_NME processing row.
C App Run Dependency Table (C_APP_RUN_DEPENDENCY)	This table contains the run dependency of each application and run name in which predecessors are defined. Each run name can have one or more predecessors or none. When a process - as defined in app name and run name - starts, it would check this table. If a predecessor is defined for the process in this table, the predecessor has to be successful as defined as “s” in column STTS_CD of table C_DRIVER_SCHEDULE before it can execute each step as defined in table C_DRIVER_STEP .
C Driver Schedule Table (C_DRIVER_SCHEDULE)	The information in this table identifies what is to run and when.
C Driver Step Table (C_DRIVER_STEP)	This table defines the high level metadata needed to define the driver detail for each app_name and run_name process.
C Driver Step Detail Table (C_DRIVER_STEP_DETAIL)	This is the table containing all active processing. This table is populated during each run with the detailed information for the driver process, pulled from C_DRIVER_STEP and C_DRIVER_SCHEDULE . As the table is populated, it is initialized with a status of ‘P’ for a pending process status. As the driver loops through this table, executing and completing each command, the status in this table will change. When the process has completed successfully, it is removed from this table, leaving only actively running processes or processes that have failed and are being recovered.
Developer Environment (Dev Environment)	The Dev Environment is where developers perform initial coding and self unit testing. Upon completion, code is moved to the Test Environment for further testing. The only environment where runname creation is allowed.

Environments	<p>See:</p> <ul style="list-style-type: none"> • Developer Environment • Test Environment • QA Environment • Production Environment
Job Step	A job step is a subset of processes within a batch. The job steps for a batch are located in C_DRIVER_STEP_T table and joined by the AUDT_ID, APP_NME, RUN_NME.
Macros	Processes to manipulate a database
Production Environment (Prod Environment)	Environment for finished code to run on daily/weekly basis.
QA Environment	Environment for User Acceptance Testing (UAT) and Regression Testing. Upon completion, code is move to Production Environment for use.
Queries	See Macros
Runnames	Name to identify a specific set of processes running within an application. An application can have multiple run names. Run names can break out processes under an application without defining a second application.
Runtime	The length of a process
Tables	<p>There are four tables used for the UD processing, two of which contain the application specific information needed for the process to run. The third table is a table used for the active running processes only. The fourth table is used to define the dependency of each process.</p> <p>See:</p> <ol style="list-style-type: none"> 1. C Driver Schedule Table 2. C Driver Step Table 3. C Driver Step Detail Table 4. C App Run Dependency Table
Test Environment	Environment for more robust testing. Upon completion, code is moved to QA Environment for further testing.