HIGH LEVEL DESIGN (HLD) FOR

Kenny's Krew

Digital Dash

Version 2.0
December 4th, 2016

Prepared by:
Craig Norton, David Ter-Ovanesyan, Dylan Gelinas, Elvis Chen, Hoang Nguyen, Ian Torres, Joseph Geneva, Kavya Krishna, Siddharth Parasnis, Thanh Pham

Managed by: Kenneth Tsui

Table of Contents

1 Introduction

1.1 Outlines

1. Introduction

This section briefly introduces the basic contents of this document, including all the features of Digital Dash, the structure of the internal software system and how it interacts with external entities and third party services.

2. Architectural and High-level Design

This section contains an overview of our system software which includes a UML diagram (explaining the purpose of each class and function), the first level Data Flow Diagram or DFD (generalizing our applications process execution states), the data dictionary (data relations along with any involved functional variables), class definitions, and function descriptions.

3. Technical Details

This section describes the frameworks and libraries used to create our application, including AngularJS, ExpressJS, NodeJS, and SQLJS, as well as other software such as: SQLite, Microsoft Azure Active Directory (AAD), and third party services such as SendGrid. We describe the frameworks that allow us to create dynamic layouts as well as to effectively communicate between the User and Liberty Mutual's database systems. We describe the technologies used to effectively mirror preexisting Liberty Mutual systems as closely as possible.

4. Architectural Rationale

This section explains the advantages of using the chosen software, framework, and external libraries. It will explain the modular nature and functional advantages of our architecture, specifically how AngularJS and JSON give us the ability to have dynamic layouts and customizable code.

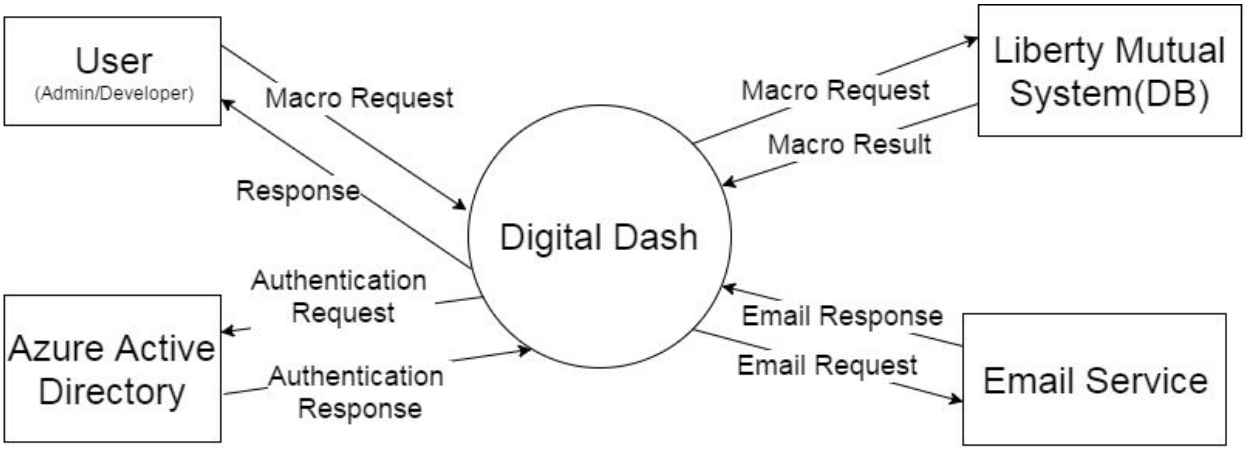
1.2 System Overview

The system will be accessed by the User through an AngularJS web application. This web interface will allow the User to communicate with the central Liberty Mutual server through the ExpressJS framework. Our server will be the main component that manipulates the Liberty Mutual metadata tables using SQLJS and backend functions used to run predefined macros. This system will greatly simplify the process of manually executing macros that manipulate Liberty Mutual's metadata tables used for scheduling and running system drivers.

Features:

1. Login and Logout: Using Azure Active Directory authentication features the user will be able to securely access the Digital Dash web application. (SRS sections 2.1 and 2.7)
2. Update Entry in Driver Tables: The application allows the User to select any currently defined macros and change any entry field to different values corresponding to the macro's definition. (SRS section 2.2)
3. Delete Entry in Driver Tables: A delete function which supports the ability to remove any current driver entries in Driver Tables. (SRS section 2.3)
4. Add Entry to Driver Tables: The application allows the User to add new driver entries into the Driver Tables. (SRS section 2.4)
5. View Log: A log allows the User to view actions and tasks that have been completed by a particular User on the system at a given time. (SRS section 2.5)
6. Peer Review: A Peer Review allows an Admin to review a request to run a macro by an User. (SRS section 2.6)

1.3 Context Level Diagram



Entity Descriptions:

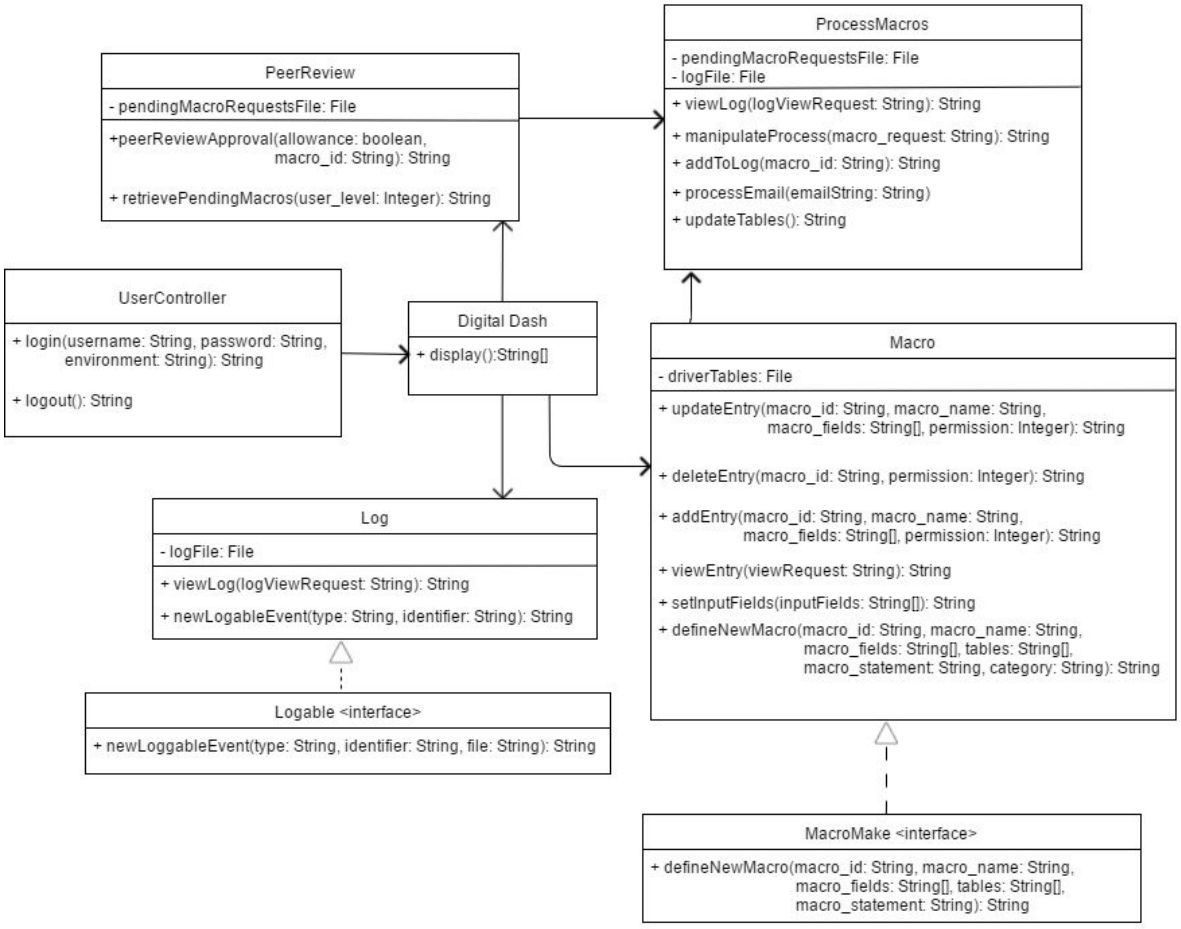
1. *User:* Either an Admin or Developer (SRS section 1.4) that is using the web application. They cannot be signed in as both an Admin and a Developer simultaneously. Each User will have permission levels granted by the Liberty Mutual IT Systems Admins, in order to ensure that all developers who need emergency execution of macros have the capabilities of doing so. Otherwise, developers must go through a peer review process in which the Admins must check to see that the macros have been implemented correctly. Admins automatically have emergency execution powers, but they will still have the option to execute through a peer review process if they wish.
2. *Liberty Mutual Systems (DB):* The User's will be authenticated through AD authentication. We will simulate AD authentication services through Azure Active Directory. Through this cloud based system our team will be able to more accurately simulate Liberty Mutual's Database systems (i.e. authentication and data handling). For our intents and purposes, Azure Active Directory will house the SQLite database specified by the example driver tables sent by Liberty Mutual. All modifications to the example driver tables will be handled via ExpressJS coupled with AngularJS directives/services that have been designed to take User input and route the information accordingly.

3. *Email Service:* The software will send an email through an email service such as SendGrid to notify Admin's of a macro execution request. Using this cloud email service will allow Users to make macro requests through the application with the side effect of an email being sent to Liberty Mutual Administrators for further peer review. The User will then have the option to send customized text along with the email to specify any important details coupled with the macro. All Admins that are associated with the application will receive an email notification that a macro request was sent through the application.

2 Architectural and High-Level Design

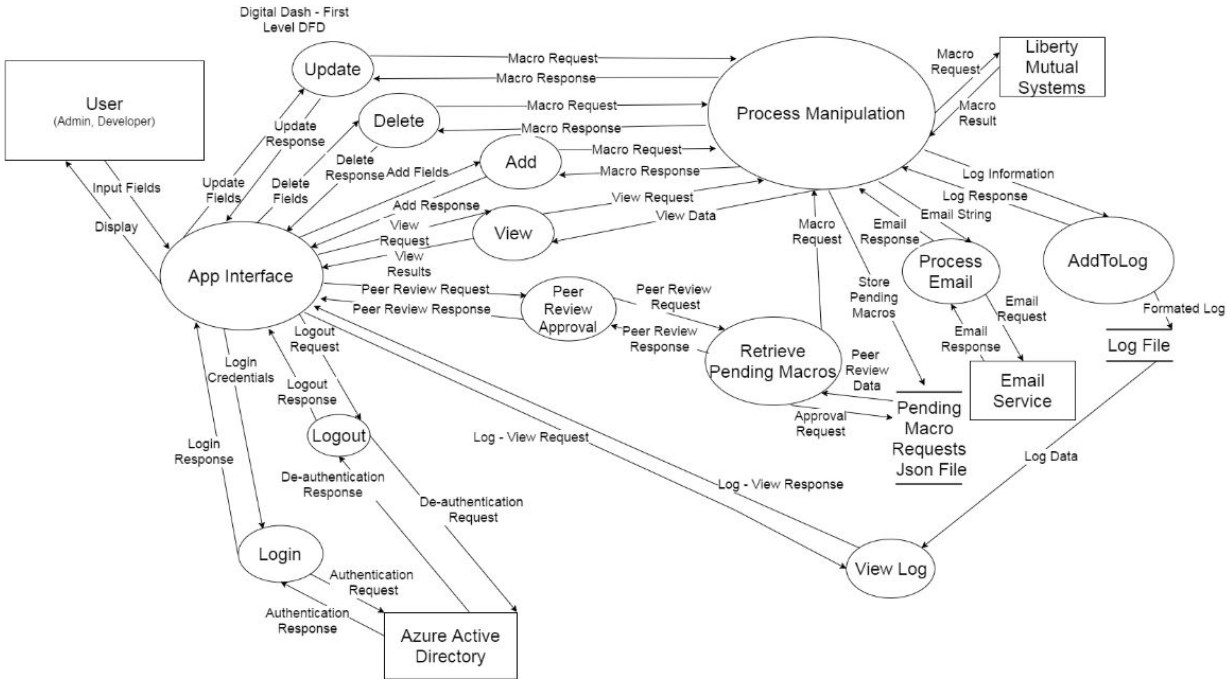
2.1 System Structure

1. UML Diagram



2.2 System Implementation

1. First Level DFD



2. Data Dictionary

Active field (active_field): A variable that indicates whether a particular part of the relation will be transmitted via a macro request.

1. Update Fields
    1. (p_run_nme, active_field): (String, boolean)
    2. (p_grp_nbr, active_field): (String, boolean)
    3. (p_actv_step_ind, active_field): (String, boolean)
    4. (p_audt_id, active_field): (String, boolean)
    5. (p_status, active_field): (String, boolean)
    6. (p_drvr_step_dtl_id, active_field): (String boolean)
    7. (p_drvr_step_id, active_field): (String, boolean)
    8. (p_sched_start, active_field): (String, boolean)

9. (p_val_end, active_field): (String, boolean)
10. (p_val_start, active_field): (String, boolean)
11. (sla_dt, active_field): (String, boolean)
12. (sla_time, active_field): (String, boolean)
13. (audt_id, active_field): (String, boolean)
14. (perf_dtl_desc, active_field): (String, boolean)

2. Delete Fields
    1. (p_run_nme,active_field): (String, boolean)
    2. (p_grp_nbr,active_field): (String, boolean)
    3. (p_drvr_step_id, active_field): (String, boolean)
3. Add Fields
    1. Driver Schedule
        1. (audt_id, active_field): (String, boolean)
        2. (app_nme, active_field): (String, boolean)
        3. (run_nbr, active_field): (String, boolean)
        4. (schedl_start_dtm, active_field): (String, boolean)
        5. (vlutn_start_dtm, active_field): (String, boolean)
        6. (run_start_dtm, active_field): (String, boolean)
        7. (crt_dtm, active_field): (String, boolean)
        8. (app_run, active_field): (String, boolean)
        9. (run_nme, active_field): (String, boolean)
        10. (re_run_nbr, active_field): (String, boolean)
        11. (stts_cd, active_field): (String, boolean)
        12. (vlutn_end_dtm, active_field): (String, boolean)
        13. (run_end_dtm, active_field): (String, boolean)
        14. (lst_mdfd_dtm, active_field): (String, boolean)
        15. (sla_date, active_field): (String, boolean)
        16. (sla_time, active_field): (String, boolean)
    2. Driver Step
        1. (drvr_step_id, active_field): (String, boolean)
        2. (app_nme, active_field): (String, boolean)
        3. (grp_nbr, active_field): (String, boolean)
        4. (run_order_nbr, active_field): (String, boolean)

5. (cmd_txt, active_field): (String, boolean)
6. (grp_cncrrncy_ind, active_field): (String, boolean)
7. (notify_txt, active_field): (String, boolean)
8. (step_nme, active_field): (String, boolean)
9. (crt_dtm, active_field): (String, boolean)
10. (app_run, active_field): (String, boolean)
11. (technology, active_field): (String, boolean)
12. (category, active_field): (String, boolean)
13. (use case, active_field): (String boolean)
14. (run_nme, active_field): (String, boolean)
15. (group_nme, active_field): (String, boolean)
16. (path_txt, active_field): (String, boolean)
17. (prmtr_txt, active_field): (String, boolean)
18. (step_cncrrncy, active_field): (String, boolean)
19. (step_typ_cd, active_field): (String, boolean)
20. (err_prc_nbr, active_field): (String, boolean)
21. (lst_mdfd_dtm, active_field): (String, boolean)
22. (actv_step_ind, active_field): (String, boolean)

4. Login Credentials
    1. user_name: String
    2. password: String
5. Login Response
    1. valid_tokens: boolean
    2. authentication_token: Hash<md5>
    3. connection_credential: Link
6. Logout Request
    1. deauthenticate: boolean
7. Logout Response
    1. logout_message: String
    2. logged_out: boolean
8. Peer Review Request
    1. user_name: String
    2. allowance: boolean

9. Peer Review Response
    1. approved: boolean
10. Approval Request
    1. approval_token: boolean
    2. macro_id: String
11. Peer Review Data
    1. peer_reviews: json<macro_id: String, macro: String, date_submitted: String, user_id: String>
12. Log - View Request
    1. date: Date
13. Log - View Response
    1. logs_from_date: String
14. Log Information
    1. macro_fields: String[]
    2. allowance: boolean
15. Log Response
    1. confirmation_message: String
16. Log Data
    1. user_name: String
    2. macro_run: String
    3. macro_fields: String[]
17. Store Pending Macros
    1. macro_name: String
    2. macro_fields: String[]
18. Formatted Log
    1. run_macro_details: String
19. Macro Request
    1. macro_fields: String[]
    2. allowance: boolean
20. Macro Response
    1. confirmation_message: String
21. Email String
    1. user_name_and_pending_macro: String

22. Email Request
    1. email_address: String
    2. message: String
23. Email Response
    1. confirmation_message: String
24. Input Fields
    1. input_variables: String[]
25. Display
    1. display_fields: String[]
26. Authentication Request
    1. user_name: String
    2. password: String
27. Authentication Response
    1. user_name: String
    2. user_level: String
28. Deauthentication Request
    1. logout_option: boolean
29. Deauthentication Response
    1. logout_response: boolean

3. Class Definitions

1. Macro - Manages all macro functions such as adding, deleting, updating and creating rows in the database
2. UserController- Handles beginning and ending user sessions
3. Log - Handles viewing the log as well as adding to the log file
4. PeerReview - Handles the peer review process, it sends out peer review requests, and returns the responses to the original sender
5. ProcessManipulation - Handles all incoming queries, this includes handling update, delete, and add requests that are sent from the dashboard to the peer review requests json object

4. Function Descriptions

1. public addEntry(macro_id: String, macro_name: String,
   macro_fields: String[], permission: Integer): String
   Adds a driver entry to one or many of the *driver tables* and sends an add response back to the user consisting of the macro that has been requested to be added, along with any macro fields
2. public addToLog(macro_id: String): String
   Creates a record of the driver task initiated by the user, and writes it into a log file. This method returns a confirmation message to indicate whether the macro requests or approvals have been successfully added to the log file.
3. public deleteEntry(macro_id: String, permission: Integer): String
   Deletes a driver entry from one or many of the *driver tables* and sends a delete response back to the user.
4. private login(username: String, password: String,
   environment: String): String
   Logs the user into their account, if the username and password are correct. If it they are incorrect it will throws an error message otherwise.
5. private logout(): String
   Logs out the user that is currently signed in and returns a confirmation message when a user successfully logs out of the application.
6. public peerReviewApproval(allowance: boolean, macro_id: String): String
   Initiated once an admin accepts, or declines a pending macro. This results is the pending JSON being updated and re displayed to the user. This also communicates with the log to update its contents.
7. public processEmail(emailString: String): String
   Takes a string containing the body text of a confirmation email as a parameter, and sends it along to the email service. A response

from the service (indicating whether the email went through or not) is then received, and returned, by the function.

8. public retrievePendingMacros(user_level: Integer): String

   Takes the information from the Pending Macro JSON File and returns a formatted view of the log including, who submitted the macro, when it was submitted, and the macro that is to be run. The user will then approve or disapprove the macro via peerReviewApproval. The following request is communicated through retrievePendingMacros and is then reflected in the JSON file (i.e. the macro is removed from the file).

9. public updateTables(): String

   Updates the driver tables and sends an update response back to the user via Process Manipulation.

10. public updateEntry(macro_id: String, macro_name: String, macro_fields: String[], permission: Integer): String

    Updates a driver entry to one or many of the driver tables and sends an update response back to the user.

11. public viewEntry(viewRequest: String): String

    Takes the viewRequest as a parameter, and gets the desired entry data from one of the driver tables. Returns view results containing the requested data.

12. public viewLog(logViewRequest: String): String

    Takes a log view request as a parameter and gets the desired data from the log file. A log view response, containing the requested data, is then sent back to the user who sent the initial request.

13. public setInputFields(inputFields: String[]): String

    Sets the input parameters for all processes that communicate with the main app interface. Once execution of the method is complete it will notify the user of process completion through the various macro, view, or peer review responses.

14. public newLoggableEvent(type: String, identifier: String, file: String):String
    Defines a new loggable instance (e.g. extending functionality to log runtimes of commonly executed macros to a log file). (SRS section 4.6)
15. public defineNewMacro(macro_id: String, macro_name: String, macro_fields: String[], tables: String[], macro_statement: String): String
    Allows for the creation of new macro definitions to be used in the system. Once defined the user will be able to actively use this macro in the system. (SRS section 4.6)
16. public manipulateProcess(macro_request: String): String
    This method takes incoming macros and adds them either to the pending macro requests json file for further peer review or overrides the peer review process based on the user's set permission and action taken upon macro execution.

3 Technical Details

3.1 Third Party Software

We will be using the following third party softwares to build our app:

1. AngularJS: a dynamic front end framework
2. NodeJS: an interactive javascript test/run environment
3. ExpressJS: a server side routing framework
4. SQLJS: a database connection and querying framework
5. SQLite: For the database
6. Microsoft Azure: Intermediary between our app and Active Directory
7. SendGrid: Send email notifications of pending macro requests to Liberty Mutual Admins

3.2 Protocols

1. TCP/IP: SQLite and Microsoft Active Directory
   Transmission control protocol and Internet protocol are standard protocols that shared files across a network using a network file system.
2. LDAP: Microsoft Active Directory
   Lightweight directory access protocol is a Industry standard application protocol for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network.
3. SMTP: SendGrid
   SendGrid can be used to work with standardized email services using SMTP. Since it is available as a cloud service app, it will seamlessly integrate with our web application as a back end framework.

4 Architectural Rationale

4.1 Rationale

The architecture Model-View-Control (MVC) was chosen because we want modularized components. This allows us to switch to a different component if needed, such as a command line interface for the view. It would still be able to work with the Model, using the same API. This also allows us to develop each component individually. We are using AngularJS as our view, a commonly used, well tested framework for creating a frontend. Utilizing ExpressJS as the controller, a middleware server running on NodeJS that help us communicate with the database. The Azure cloud service is our model, which will hold our database. Our view will send requests from the client to the controller. These controllers will then connect to our domain and send the macro inputs as query parameters to the database.This proposed model then makes any necessary database changes, and provides feedback to the view, which will update the client.

4.2 Framework and library

The application will be using NodeJS as an environment for developing our application, since it will allow us to use Javascript across the stack. We will only need to use one language, (thereby) lowering the learning curve. For the front end, we decided to use AngularJS, a frontend framework for web applications, because it will enable us to construct a system that follows the MVC pattern. For the backend, we will be using ExpressJS to handle HTTPS requests from our frontend services as well as connect to the database cloud service. SQLJS will be used to query the given mock database file. The Azure cloud service will be running Active Directory, allowing us to use its built in security. Overall each framework and library is chosen specifically for its specialization for each dependency requirement we need to accomplish.