## Table of Contents

* 1.0. Project Overview
* 1.1. File and Folder Structure
* 1.2. Database Structure
* 2.0. How to Get Set up
* 2.1. Creating the Database
* 2.2. Changing the HTTP Requester Domain for the Proxies
* 2.3. Configure Database Credentials
* 2.4. Change config.php
* 3.0. Aditional Information
* 3.1. Known Bugs
* 3.2. Security Concerns
* 3.2.1. API Security Concerns
* 3.2.2. Login System Security Concerns

## 1.0 Project Overview

Title: Symposium Scheduler Web Application Dates Worked On: Jan 2019 - June 2019 Team Members: TJ Breitenfeldt, Andrew Bosco, Chester Southwood, Tatyana Hubbard

This project is a web-based application that allows users to manage their own personal schedule for symposiums/conferences. It also allows administrators to create/manage/edit symposiums/conferences that they are in charge of. The purpose of this application is to replace the application EWU uses currently (Sched) with an application that is accessible.

#### Technologies:

Front-End:

HTML SASS (CSS extension) Javascript w/ JQuery

Back-End:

PHP MySQL

Section 1 Overview:

1.1 File and Folder Structure 1.2 Database Structure

### 1.1. File and Folder Structure

This section of our documentation is a small description of what each folder and file is used for within this project.

#### Root (application) folder

Within the application folder, it contains these folders:

* admin folder: This folder contains almost copies of the php files within the root, except slightly changed for specific admin use.
* conferenceAPI folder: Contains the files
* css folder: This folder contains files for our project’s css and SASS.
* databaseUtil folder: This folder contains the PDO in php to our database.
* javascriptLoads folder: This folder contains HTML / PHP files that will be loaded into the index.php’s innerContent div when a javascript action occurs.
* js folder: This folder contains all of our javascript code for the project.
* loginAPI folder:
* phpIncludes folder:
* proxies folder: This folder contains our proxy code that accesses the database with either a userID or adminID.

This folder contains these files (which are mainly HTML files wrapped as a php file):

* authenticateUser.php: A php file that contains a function to test if the user is registered or not for login purposes.
* config.php: A php file that defines and holds constants for the project.
* error.php: A php (HTMl) file that is shown if an error has occurred.
* forgotPassword.php: A php (HTML) form that can send an email to the user to reset the password.
* index.php: Our main php / HTML file that contains sidebars, the menu, references to the userJS folder, and a “content” div for javascript use.
* login.php: A php (HTML) form to login to the website.
* logout.php: A php file to log the user out of the website and go back to the login page.
* register.php: A php (HTML) form to register as a user for the website.
* resetForgotPassword.php: A php (HTML) form to reset the password after going through the forgotPassword.php file.

#### Admin folder

This folder contains these files (which are mainly HTML files wrapped as a php file):

* authenticateUser.php: A php file that contains a function to test if the login was an admin.
* config.php: A php file that defines and holds constants for the project on only the admin side.
* error.php: A php (HTMl) file that is shown if an error has occurred.
* forgotPassword.php: A php (HTML) form that can send an email to the admin to reset the password.
* index.php: Our main php / HTML file that contains admin specific accessibility and does not contain much css.
* login.php: A php (HTML) form to login to the website.
* logout.php: A php file to log the admin out of the website and go back to the login page.
* register.php: A php (HTML) form to register as an admin for the website (Can only be done through another admin).
* resetForgotPassword.php: A php (HTML) form to reset the password after going through the forgotPassword.php file.
* resetPassword.php: A php (HTML) form the reset the password.

#### conferenceAPI folder

This folder contains these files:

* delete.php: The handler for the http delete request for the API
* get.php: the handler for the http get request for the API
* index.php: The main file for the API, all API requests are directed through this file
* post.php: the handler for the http post request for the API
* put.php: the handler for the http put request for the API

#### databaseUtil folder

This folder contains these files:

* creds.php: The credentials and database settings defined as global constants for the database utility
* pdoUtil.php: The database utility used by all php API code in the project

#### javascriptLoads folder

This folder contains these files: \* aboutConference.php: This file is loaded into the root’s index.php (in place of the current loaded InnerContent tag) and shows information about a conference on load or when the home page button is chosen. \* conferenceChooser.php: This file is loaded into the roots’s index.php file when a User is not registered for a conference or when a user wants to change their conference \* conferenceSchedule.php: This file is loaded into the root’s index.php file when a user want to look at the full conference table \* editSchedule.php: This file is loaded into the roots’s index.php file when the user wants to add or delete an event from their schedule \* resetPassword.php: This file is loaded as a form onto the page when the user chooses the “Reset Password” button. \* showSchedule.php: This file is loaded into the index.php’s innerContent div and shows the user’s conference schedule \* userSettings.php: This file is loaded into the index.php’s innerContent div and shows the user’s current settings, which can be changed in this form.

#### js folder

Within the js folder, it contains these folders:

* adminJs: A folder that holds javascript functionality for the admin side only.
* conferenceAPIJs: contains a single file that is a wrapper for ajax calls to the conference API
* loginSystemJs: contains a single file that manages form collection and posts using ajax to the login API
* userJs: A folder that holds the javascript for frontend user functions.
* utilityJs: A folder that holds a useful javascript file to assist in other javascript modules.

##### js/adminJs

This folder contains these files:

* conferenceManager.js: a javascript file that contains the administrator dashboard functions for constructing and dynamically changing the admin UI.
* generateHTML.js: A helper javascript file for the admin dashboard used to construct and return strings of html.

##### js/conferenceAPIJs

This folder contains this file:

* databaseFunctions.js: A javascript file that uses either the conferenceAPI folder or proxies folder to easily access the database for the other javascript modules.

##### js/loginSystemJs

This folder contains this file:

* loginAJAX.js: The javascript file used to collect and post data to the login API, used for: login.php, register.php, forgotPassword.php, resetForgotPassword.php, and resetPassword.php html files for both the user and admin sides.

##### js/userJs

This folder contains these files:

* mainSchedule.js: A javascript file that is used to get the main conference’s information, have the functionality for the conference, and to check whether the user is even registered to a conference.
* menu.js: A javascript file that contains most of the frontend’s functionality; which is the sidebar’s use, the click of every option in the side bar, and the creation of cookies.
* userAccountRegistration.js: A javascript file that only has a function to toggle the phone aspect of the register.php file.
* userSchedule.js: A javascript file that creates the user’s schedule table and contains the table’s functionality.
* userSettings.js: A javascript file to update and add information to the userSettings.php file in javascriptLoads.

##### js/utilityJs

This folder contains this file:

* util.js: A javascript file that can be used anywhere to assist in use. The main functions it currently does is notify the screen reader and parse dates.

#### loginAPI folder

Within the loginAPI folder, it contains one folder:

* PHPMailer: This folder is a third party library that we use to send mail.

This folder contains these files:

* dataValidation.php: A php file that contains a collection of functions to currently only validate the registration form for the login system.
* forgotPasswordFunctions.php: A php file for using a provided email to look up the user account tied to that email address, and send a reset forgot password link to that email address.
* includeConfig.php: A php wrapper for the config.php file found in the root directory and the admin directory, used to identify which type of user is logged in, weather a user or admin, and will include the correct config.php file.
* loginFunctions.php: A php file used for handling the provided username and password, and checking against the database to see if the username and password match the username and password in the database.
* logoutFunctions.php: A small php file to log a user out by destroying the current session and redirecting the user to the login page.
* registerFunctions.php: A php file used to handle the provided registration data, validation and scrubbing of the data, hashing of the password, and insertion into the database.
* resetForgotPasswordFunctions.php: The php file used to handle the resetting of a forgot password. Depends on the forgotPassword.php file to be run first to generate a token. These functions will then check to make sure the provided token and email address are correct, then reset the users password.
* resetPasswordFunctions.php: A php file used to reset a users password, depends on the user already knowing his or her existing password. This file will check to make sure the existing password is valid based on the current user name that is logged in, then reset the password for that account.

#### phpIncludes folder

This folder contains these files:

* accesibilityMenuOnly.php: A stand alone html file that is included into all pages except for the user control panel to allow users to change the font or page color from anywhere.
* adminHeader.php: The stand alone html file that is included in all of the html admin files in the admin folder at the top inside a
* tag.
* footer.php: The stand alone html file that is included at the bottom of all of our html files to show the footer information.
* userHeader.php: The stand alone html file that is included at the top in all of the html user files in the root directory inside a
* tag.

#### proxies folder

This folder contains these files:

* deleteProxy.php: deleteProxy is used by both users and admins to delete data from the database.
* getProxy.php: getProxy is used to access the database using either the admin id or user id to get information.
* httpRequester.php: HttpRequester is referenced within 2.2. This file is used to set up how the proxies work. The only thing that will potentially need to be changed is the domain.
* postProxy.php: postProxy is used by both users and admins. Admins use this more, but it is used to add to tables for the user.
* putProxy.php: putProxy is used by only the admin side. This is used to edit information stored within the database.

### 1.2. Database Structure

Database Name: Db\_a444c6\_senior Tables:

* admin\_accounts - admin account information
* user\_accounts - user account information
* conference - conference information
* event - event information
* user\_schedule - user event schedule information
* user\_conference - user conference information

admin\_accounts: Primary Key - admin\_id

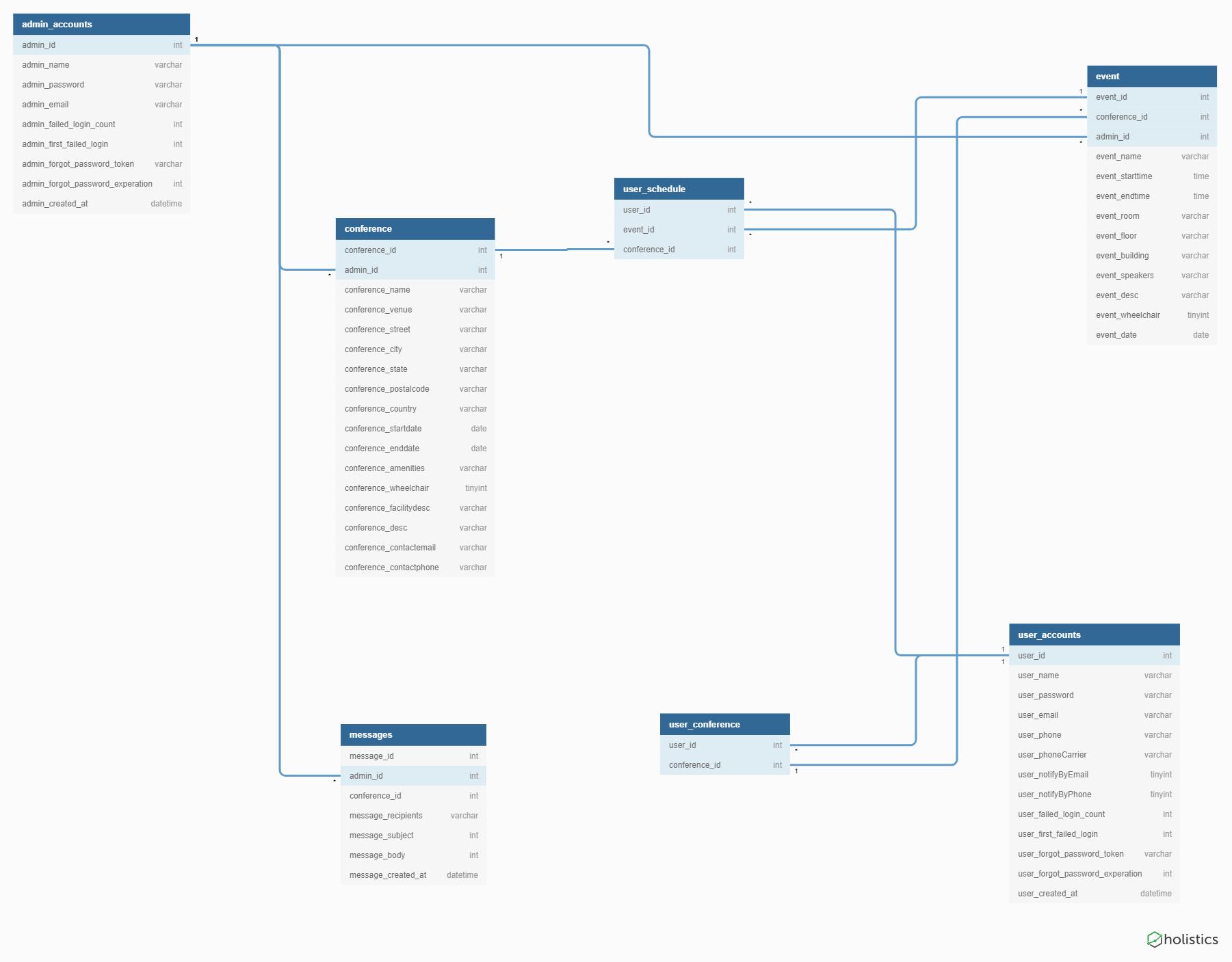
user\_accounts: Primary Key - user\_id

conference: Primary Key - conference\_id Foreign Key - admin\_id

event: Primary Key - event\_id Foreign Key - admin\_id, conference\_id

user\_schedule: Foreign Keys - user\_id, event\_id, conference\_id

user\_conference: Foreign Keys - user\_id, conference\_id



alt text

## 2.0. How to Get Set up

1. Install XAMPP on your computer.
2. Run XAMPP and start the apache and SQL server.
3. Download the project zip or pull down from git hub.
4. The root location of this project should be in the htdocs folder located in your XAMPP folder.
5. In your internet browser, type ‘localhost/phpmyadmin’.
6. In PhpMyAdmin, click Import, then click choose a file. From the file explorer, go to the ‘symposium-management-webapp’ folder and choose the ‘conference\_manager.sql’ file.
7. Continue through the import process to create the database structure on your localhost server. After it’s done, you should have an empty database.
8. Now you will need to create a user through PhpMyAdmin that has access to the database.
9. After you create that user, go to ‘symposium-management-webapp/application/databaseUtil/creds.php.’
10. In creds.php, change the user variables there to match the credentials of your newly created user. Look at section 2.3 for more info.
11. Find and open ‘symposium-management-webapp/application/proxies/httpRequester.php’.
12. Confirm that the variable ‘DOMAIN’ is the absolute path to the application folder. If the application is ever having an issue where it won’t load up a new page, it is more than likely an issue with your DOMAIN variable in this file. The commented out domains are different domains we’ve used between the four of us in different environments we’ve worked in.
13. Go to your browser and type in ‘localhost/symposium-management-webapp/application’ and now the application should be ready to use.
14. For getting this project set up on a hosting service such as Bluehost or SmarterASP, you can do steps 5-13 doing certain parts through the hosting service, specifically getting the database set up through your hosting service rather than through PhpMyAdmin.

Section 2 covers:

* 2.1. Creating the Database
* 2.2. Changing the HTTP Requester Domain for the Proxies
* 2.3. Configure Database Credentials
* 2.4. Change config.php

### 2.1. Creating the Database

For creating the database, you will need to use our SQL file. The SQL file is called “conference\_manager.sql” and is located in the root folder of this project. After it is downloaded into phpmyadmin or another service the database will be ready to use.

### 2.2. Changing the HTTP Requester Domain for the Proxies

The HTTP Requester’s domain is used to give the Proxies the main link to the root folder for the application. You will find the HTTP Requester file within the ‘application/proxies’ folder. If this project is not being used or tested through localhost or is being renamed, the domain will have to be changed. Enter in your own link instead of the current one to allow for the proxies to work.

### 2.3. Configure Database Credentials

In the creds.php file, you have the following variables:

DB\_NAME - Name of the database you wish to connect to. DB\_HOST - Website of the database. If locally running the database, this would be localhost. DB\_USERNAME - User that is accessing the database. DB\_PASSWORD - Password of that user. DB\_CHARSET - Charset of the database.

### 2.4. Change config.php

The config.php file located in ‘application/config.php’, and ‘application/admin/config.php’ is used for defining login system settings for the user and admin portals. The config.php file contains several php constants defining:

* the database table names refered to by the login system, such as user\_accounts, and admin\_accounts,
* the field names in the specified database user account tables, these constants must be included as fields in the specified tables sinse these constants are referenced in the SQL queries, such as username, password, email, etc…,
* the constants defined for locking out users after x number of attempts
* Aditional user information that is not included in the defined constants for the login system defined as a array,
* the constants for the forgot password system,
* the constants for the email system, the email address, password, and other information for sending an email using (the third party library php mailer)

## 3.0. Additional Information

### 3.1. Known Bugs

* On occasion, when logging into both the user and admin portals in two different tabs, there is an issue where the admin redirects to the user landing page, or the user may be redirected to the admin landing page. We believe that this is tied to the design decision of using the same session for all account session variables.
* If provided single or double quotes in the input for the admin dashboard it will break the functionallity of the buttons for the user schedule and conference schedule on the user application side. We believe this is due to a lack of data validation, and the quotes are actually being read as javascript code. See the login system security section for more about this concern.
* We have strange problems that come up every now and then, our guess is that any inconsistent issues that crop up are related to asynchronus ajax calls, where code is being run before the ajax call has finished.

### 3.2. Security Concerns

The major Security concern for all user input is that no data is scrubbed or validated except in the register form for the login system. This is an issue that should be addressed because it makes this application susceptible to SQL Injections in places we are not using prepared statements, and cross site scripting. In adition, a known bug is tied with this security concern, where if you input single or double quotes into any of the admin input fields they are read as javascript closing quotes sinse they are not being escaped, and cause various issues on the front end application.

### 3.2.1. API Security Concerns

In the folder ‘conferenceAPI’, we have a basic REST Api set up that can be configured for any database. In each of the files, there are restrictions built in specifically for our database structure. These restrictions are meant to not allow malicious users to do certain actions such as access other users/admins account info, edit/delete conference data, etc.

The known security issue with this api is that in each of these php files, roughly half of the parameters entered by the user are prepared, meaning that those parameters will be checked for SQL injection. The reason only half of them are prepared were due to a couple of reasons. The api dynamically creates a SQL command from user input parameters, so there are more parameters to prepare compared to other apis. The second reason was due to time constraints on the project, we opted to not spend more time on this concern and to put that time toward getting a working product.

If someone were to send a GET request to the API right now, it would dynamically create a SQL command that would look like the following:

**SELECT \* FROM event WHERE conference\_id = ? AND event\_id = ? ORDER BY event\_starttime;**

The ‘?’ here is a parameter that will be prepared by the pdoUtil. The ideal SQL command would look like the following:

**SELECT ? FROM ? WHERE ? = ? AND ? = ? ORDER BY ?;**

Any time a parameter is entered by a user, it should be replace by a ‘?’ and sent to the pdoUtil to be prepared. The reason this is a more time intense solution to implement is that there would be a lot of array manipulation involved while dynamically creating the SQL command. Additionally, due to the nature of SQL syntax, each of the four API php files would require their own implementation independant of each other to make sure that each parameter is prepared in the right order.

Each parameter that is prepared is stored in the $values array which is sent to the pdoUtil alongside the dynamically created SQL command.

### 3.2.2. Login System Security Concerns

Most of the security concerns mentioned here about the login system can be avoided if we had used a third party login API, such as the Google API.

As mentioned, data validation and scrubbing is performed on registration data, however, minimal data validation is performed on usernames and passwords when logging in, which could make us vulnerable to cross site scripting on the login pages. Prepared statements are used on all data, so sql injection should be protected against. Our site is vulnerable to cross site request forgery though, and no Precautions were taken to protect against this attack due to time constraints. It is also important to note that the login system locks users out after x number of attempts for y minutes. These values for lockout are stored in the database, so someone could start making password attempts on individuals accounts and lock them out from anywhere, the lockout is not spacific to IP address. In adition, session jacking was never an attack that was considered, and could be another vulnerability. It is important to note as well that the user and admin share the same session for their account variables, so this could be a security risk, but we are unsure of the ramifications for this design decision.