Teen Court Database

Software Development Document | Current Version 1.1.1

Prepared By:

Andrew Thompson

Robert Reilly

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Author** | **Version** | **Comments** |
| 9/26/12 | Andrew Thompson | 1.0.0 | Initial version |
| 10/30/12 | Andrew Thompson | 1.0.1 | Updated User Access and Defendant |
| 11/20/12 | Andrew Thompson | 1.0.2 | Added Court Program and Volunteer |
| 12/2/12 | Andrew Thompson | 1.0.3 | Added SQL schema information |
| 12/5/12 | Andrew Thompson | 1.0.4 | Added database table for each section |
| 12/6/12 | Robert Reilly | 1.0.5 | Added CONOPS |
| 4/18/13 | Robert Reilly | 1.0.6 | Added class documentation |
| 4/21/13 | Andrew Thompson | 1.1.0 | Renamed document, new schema, cleanup |
| 4/23/13 | Andrew Thompson | 1.1.1 | Component Design Documentation |

Table of Contents

[1.0 Overview 7](#_Toc354531454)

[1.1 Scope 7](#_Toc354531455)

[1.2 Purpose 7](#_Toc354531456)

[1.2.1 Domain Name and Web Space 7](#_Toc354531457)

[1.2.2 Server Environment and Database 7](#_Toc354531458)

[1.2.3 Web Application 7](#_Toc354531459)

[1.2.4 Security 7](#_Toc354531460)

[1.3 Account Credentials 8](#_Toc354531461)

[1.3.1 Web Server 8](#_Toc354531462)

[1.3.2 CPanel Hosting Access 8](#_Toc354531463)

[1.3.3 Application Administrator Login 8](#_Toc354531464)

[1.3.4 Google Analytics Login 8](#_Toc354531465)

[1.3.5 MySQL Web User 9](#_Toc354531466)

[1.4 Systems Goals 9](#_Toc354531467)

[1.5 Concept of Operations (CONOPS) 9](#_Toc354531468)

[1.5.1 Application User 9](#_Toc354531469)

[1.5.2 Login 9](#_Toc354531470)

[1.5.3 Register Account 9](#_Toc354531471)

[1.5.4 Home 9](#_Toc354531472)

[1.5.5 Defendant 9](#_Toc354531473)

[1.5.6 Edit Defendant 10](#_Toc354531474)

[1.5.7 New Defendant 10](#_Toc354531475)

[1.5.8 Search Defendant 10](#_Toc354531476)

[1.6 System Overview 10](#_Toc354531477)

[1.7 Technologies Overview 11](#_Toc354531478)

[2.0 Project Overview 13](#_Toc354531479)

[2.1 Team Members and Roles 13](#_Toc354531480)

[2.2 Project Management Approach 13](#_Toc354531481)

[2.3 Phase Overview 13](#_Toc354531482)

[2.3.1 User Stories and Requirements Gathering 13](#_Toc354531483)

[2.3.2 Database Schema and Class Design 13](#_Toc354531484)

[2.3.3 Prototype 13](#_Toc354531485)

[2.3.4 Application Development 14](#_Toc354531486)

[2.3.5 Testing 14](#_Toc354531487)

[2.3.6 Delivery 14](#_Toc354531488)

[2.4 Terminology and Acronyms 14](#_Toc354531489)

[3.0 Requirements 15](#_Toc354531490)

[3.1.1 User Access 15](#_Toc354531491)

[3.1.2 Teen/Youth Program & Data Access 15](#_Toc354531492)

[3.1.3 Defendant 15](#_Toc354531493)

[3.1.4 Volunteer 15](#_Toc354531494)

[3.1.5 Court 15](#_Toc354531495)

[3.1.6 Workshop 15](#_Toc354531496)

[3.1.7 Reports 16](#_Toc354531497)

[3.1.8 Statistics 16](#_Toc354531498)

[4.0 Design and Implementation 17](#_Toc354531499)

[4.1 Program 17](#_Toc354531500)

[4.1.1 Component Overview 17](#_Toc354531501)

[4.1.2 Application Pages 18](#_Toc354531502)

[4.1.3 Database Tables 18](#_Toc354531503)

[4.1.4 Classes 18](#_Toc354531504)

[4.1.5 Architecture Overview 19](#_Toc354531505)

[4.1.6 Design Overview 19](#_Toc354531506)

[4.2 User Account and Access 19](#_Toc354531507)

[4.2.1 Component Overview 19](#_Toc354531508)

[4.2.2 Application Pages 20](#_Toc354531509)

[4.2.3 Database Tables 20](#_Toc354531510)

[4.2.4 Classes 20](#_Toc354531511)

[4.2.5 Architecture Overview 20](#_Toc354531512)

[4.2.6 Design Overview 22](#_Toc354531513)

[4.3 Defendant 22](#_Toc354531514)

[4.3.1 Component Overview 22](#_Toc354531515)

[4.3.2 Application Pages 24](#_Toc354531516)

[4.3.3 Database Tables 25](#_Toc354531517)

[4.3.4 Classes 25](#_Toc354531518)

[4.3.5 Architecture Overview 25](#_Toc354531519)

[4.3.6 Design Overview 25](#_Toc354531520)

[4.4 Volunteer 26](#_Toc354531521)

[4.4.1 Component Overview 26](#_Toc354531522)

[4.4.2 Application Pages 26](#_Toc354531523)

[4.4.3 Database Tables 26](#_Toc354531524)

[4.4.4 Classes 27](#_Toc354531525)

[4.4.5 Architecture Overview 27](#_Toc354531526)

[4.4.6 Design Overview 27](#_Toc354531527)

[4.5 Courts 27](#_Toc354531528)

[4.5.1 Component Overview 27](#_Toc354531529)

[4.5.2 Application Pages 28](#_Toc354531530)

[4.5.3 Database Tables 28](#_Toc354531531)

[4.5.4 Classes 28](#_Toc354531532)

[4.5.5 Architecture Overview 28](#_Toc354531533)

[4.5.6 Design Overview 28](#_Toc354531534)

[4.6 Workshops 29](#_Toc354531535)

[4.6.1 Component Overview 29](#_Toc354531536)

[4.6.2 Application Pages 29](#_Toc354531537)

[4.6.3 Database Tables 29](#_Toc354531538)

[4.6.4 Classes 29](#_Toc354531539)

[4.6.5 Architecture Overview 30](#_Toc354531540)

[4.6.6 Design Overview 30](#_Toc354531541)

[4.7 Statistics 30](#_Toc354531542)

[4.8 Surveys 30](#_Toc354531543)

[4.9 Program Administration Options 30](#_Toc354531544)

[4.10 Application Administration Options 30](#_Toc354531545)

[5.0 Testing 30](#_Toc354531546)

[5.1 Overview 30](#_Toc354531547)

[5.2 Dependencies 30](#_Toc354531548)

[5.3 Test Setup and Execution 30](#_Toc354531549)

[6.0 Development Environment 30](#_Toc354531550)

[6.1 Development IDE and Tools 31](#_Toc354531551)

[6.2 Source Control 31](#_Toc354531552)

[6.3 Dependencies 31](#_Toc354531553)

[6.4 Build Environment 31](#_Toc354531554)

[6.5 Development Machine Setup 31](#_Toc354531555)

[7.0 Release | Setup | Deployment 32](#_Toc354531556)

[7.1 Setup Information 32](#_Toc354531557)

[7.2 System Versioning Information 32](#_Toc354531558)

[8.0 End User Documentation 32](#_Toc354531559)

[Appendix I: List of Figures 33](#_Toc354531560)

[Appendix II: Supporting Information and Details 34](#_Toc354531561)

[II.1 Database Schema 34](#_Toc354531562)

[Appendix III: Sprint Reports 45](#_Toc354531563)

[III.1 Sprint 1 Progress Report 45](#_Toc354531564)

[III.2 Sprint 2 Progress Report 45](#_Toc354531565)

[III.3 Sprint 3 Progress Report 45](#_Toc354531566)

[III.4 Sprint 4 Progress Report 45](#_Toc354531567)

[III.5 Sprint 5 Progress Report 45](#_Toc354531568)

[III.6 Sprint 6 Progress Report 45](#_Toc354531569)

[Appendix IV: Class Documentation 45](#_Toc354531570)

[IV.1 Citation 45](#_Toc354531571)

[IV.2 Core 46](#_Toc354531572)

[IV.3 Court Location 47](#_Toc354531573)

[IV.4 Court 47](#_Toc354531574)

[IV.5 Data 48](#_Toc354531575)

[IV.6 Defendant 50](#_Toc354531576)

[IV.7 Guardian 52](#_Toc354531577)

[IV.8 Location 53](#_Toc354531578)

[IV.9 Class Program 53](#_Toc354531579)

[IV.10 School 55](#_Toc354531580)

[IV.11 Sentence 55](#_Toc354531581)

[IV.12 User 56](#_Toc354531582)

[IV.13 Volunteer 57](#_Toc354531583)

[IV.14 Class Workshop Location 58](#_Toc354531584)

[IV.15 Workshop 59](#_Toc354531585)

# Overview

The Teen Court Database is a web application used to track the progress of juvenile defendants within the local teen, or youth court program. Court administrators will have the ability to generate reports, view statistics and track volunteer information. Since this is intended to be web application, access will be obtained with any internet capable computer or mobile device.

This document contains the necessary components for the design and implementation of the Teen Court Database application. [AT]

## Scope

The scope of this project is to develop a web based system that allows court administrators the ability to track information for juvenile defendants within the teen court programs. [AT]

## Purpose

The purpose of this document is to describe what the various components of the system are and how they will be developed. The major components of the application are described in various subsections:

* *Technologies Used:*  Lists the specific languages, plugins, or tools used to develop the component.
* *Component Overview:* Describes the data and database tables within the component.
* *Phase Overview:* Describes what work is done during specific phases of development.
* *Architecture Overview:* Describes how the component will function.
* *Design Overview:* Describes the look and feel of the component.

Any possible development issues or constraints will also be described. [AT]

### Domain Name and Web Space

This application will be developed under the domain name [teencourtdb.com](http://teencourtdb.com/). Web and database server hosting will be provided by the Baltimore based website design firm, eCoastStudios. Michael Roth is the main contact for this project.

### Server Environment and Database

The development server and production server will be on Linux servers with the latest version of PHP and MySQL database. A hosting control solution, cPanel, will allow the development team access to web space management.

### Web Application

The main application will provide all the requirements gathered from stakeholders and listed in the Software Requirements Document.

### Security

In addition to individual user access based upon email addresses, a security certificate/key has been provided for Secure Socket Layer (SSL) data encryption between the client and server. Also, once the application is live, the defendant table should be encrypted in the database.

## Account Credentials

These are the various accounts used to access the web hosting and database servers. [AT]

### Web Server

|  |  |
| --- | --- |
| Address: | [teencourtdb.com](http://teencourtdb.com) |
| Username: | teencour |
| Password: | 1Vz6S(k{o1rx=5>: |

### CPanel Hosting Access

This login is for the web hosting control panel. A vast array of information can be manipulated here.

|  |  |
| --- | --- |
| Address: | [teencourtdb.com/cpanel](http://teencourtdb.com/cpanel) |
| Username: | teencour |
| Password: | 1Vz6S(k{o1rx=5>: |

### Application Administrator Login

Use this account to have just application administrator access – can only add programs and users.

|  |  |
| --- | --- |
| Address: | [teencourtdb.com/index.php](http://teencourtdb.com/index.php) |
| Username: | admin@teencourtdb.com |
| Password: | [T33ncourt] |

### Google Analytics Login

Google Analytics tracks visitor information.

|  |  |
| --- | --- |
| Address: | <http://www.google.com/analytics/> |
| Username: | admin@teencourtdb.com |
| Password: | 1Vz6S(k{o1rx=5>: |

### MySQL Web User

This is used for the PHP database connections. The user has limited rights and can be viewed in Cpanel.

|  |  |
| --- | --- |
| Host: | localhost |
| Database Name: | teencour\_web |
| Database User: | 1Vz6S(k{o1rx=5>: |
| Database Password: | t33nc0urtw3b12 |

## Systems Goals

This project’s goals are to give teen/youth court program administrators and their employees an accessible and easy to use solution to track participants in youth court programs. By tracking defendant data, statistics can be generated to prove these types of alternate judicial processes work. This may lead to more funding and more courts taking this approach. [AT]

## Concept of Operations (CONOPS)

This section will cover how the program is to be used from the view point of individuals who will be using it. [RR]

### Application User

This will cover components that a standard court user will interact with.

### Login

The user will type in their user name and password. If the user name and password match what is in the database, the user will be moved to the home screen. If one or both are wrong, the program will display an error message and ask the user to log in again.

### Register Account

The user will type in the court program code, their email address, choose a secure password and confirm it, then enter the captcha. If the program code and captcha is valid, the user will be informed that the court administrator for that court has to approve them. If the program code or captcha is invalid, the user will be informed of the error and be asked to enter it again.

### Home

After successfully logging in to the court system, the user will be presented with the home screen. The home screen will show all upcoming courts. Clicking the Edit option on a court will allow the user to modify that court’s information in the Court tab.

### Defendant

Clicking on the Defendant tab or List Defendants will take the user to the Defendants list. All active defendants with that program will be displayed. Defendant information can be edited by clicking on the Edit option.

### Edit Defendant

Upon opening the page, the user will have all information of the defendant displayed. Users will be able to modify the defendant’s primary information, personal information, parent information, citation information, intake information, court information, sentence information, workshop information, have the option to expunge the defendant according to the expunge level set by the program administrator, print forms relating to the defendant, and record case notes. Recording any changes made to a defendant will require hitting the Update button at top. Previous and Next can be used to go to the next defendant according to ID, but will not save any changes. Delete can be used to return to the Defendant List without saving any changes, but will not delete the defendant after they have been entered.

### New Defendant

Clicking on the New Defendant button at the top of the page will allow the user to create a new defendant. The user can enter the defendant’s last name, first name, date of birth, home phone, court case number, and agency case number. After the information has been entered, the user can click the Add Defendant button at the top of the page. The user will then be moved to the Edit Defendant page to enter other information relating to the defendant.

### Search Defendant

Clicking on the Search Defendant button will allow the user to do a detailed search of the database. Options for how to search are: first name, last name, city, date of birth, home phone, expunged level, court case number, agency case number, and if the defendant is active, closed, or all. Searching will return all defendants that qualify under that information. The user can use the search bar included with the table to further search.

## System Overview

Users will access this application through qualified web browsers offered on most major internet capable devices, such as personal computers, tablets and smart phones. The application will offer cross-platform and cross-browser support. When a user’s browser makes a connection to the website, the pages and data will be served to the client. The diagram in figure 1 shows the system’s web architecture overview.

There are five major components to the actual web application. The top level object, Program, contains information pertaining to an individual teen/youth court program. Users belong to a specific program and are able to add defendants and volunteers. The defendants and volunteers are used to build workshop and court objects. Figure 2 shows the relationship between these components. The design and implementation of each component is described in full detail in section [4.0 Design and Implementation](#_Design_and_Implementation). [AT]

|  |  |
| --- | --- |
| C:\Users\1064529\Dropbox\School\Senior Design 2\graphics\ArchDiagram.tif  Figure 1: System Diagram | C:\Users\1064529\Dropbox\School\Senior Design 2\graphics\datadesign1.tif  Figure 2: Major Component Diagram |

## Technologies Overview

This application will be developed using the latest version of PHP and take an Object-Oriented approach. PHP is popular and widely used HTML-embedded scripting language that allows developers to quickly build dynamically generated web pages. More information: can be found at <http://us3.php.net/>.

A data access layer will be implemented to help secure information between the web and database server. This will most likely be PHP’s built in PHP Data Object (PDO). For more information, visit <http://www.php.net/manual/en/book.pdo.php>.

For data storage, the latest version of MySQL will be used with this application. This is a popular open-source relational database management system (RDBMS). More information can be found at <http://www.mysql.com/>.

JSON, JavaScript Object Notation, will be used for formatting the data of SQL results. <http://www.json.org/>

A JavaScript Framework, JQuery, will be used to ensure cross-browser JavaScript support. To manipulate client webpages without making additional requests to the server, AJAX, or Auto-synchronous JavaScript and XML, will be used and is built into the framework. More information can be found at <http://www.jquery.com>.

JQueryUI , a JavaScript Framework that generates client-side widgets will be used to provide a more functional user interface. More information about this framework is available at <http://www.jqueryui.com>.

Several JQuery plugins were used, these include:

DataTables, a plugin that reads JSON formatted data generated from the database and displays highly functional listing of tabular data. Sorting, searching and paging make data easy to view and manipulate. <http://www.datatables.net/>

JQuery Validate, a plugin to allow easy client-side validation and error reporting for web forms. <http://docs.jquery.com/Plugins/Validation>

JQuery Mask, a plugin to allow form field input masks. Useful to force phone number styles, dates, etc. <http://plugins.jquery.com/mask/>

PwStrength, a plugin to check user password strengths based on weights of characters, numbers, symbols, etc. Not a password policy enforcement, just provides a general strength. <http://plugins.jquery.com/pwstrength/>

JQuery ptTimeSelect, a plugin that mimics JQueryUI’s DatePicker but does it for time.

<http://pttimeselect.sourceforge.net/example/index.html>

Google RECAPTCHA, this is used on the publically accessible registration page to prove human interactions. [AT]

# Project Overview

This section will describe team member roles and how the project will be managed. [AT]

## Team Members and Roles

Andrew Thompson will be the Team Leader and Lead Developer. Robert Reilly will be Lead Tester and Developer. Author notation will be identified by [AT] for Andrew and [RR] for Robert at the start of major sections (*ex: after heading 2.0 above*). Subsections within the main sections assumed to be by the same author.

## Project Management Approach

The Scrum Team is scheduled to have Sprint Meetings every Tuesday and Thursday at 10:00 AM. These meetings will not last longer than 15 minutes. Additional Meetings will be scheduled as needed. The Project Lead, Andrew Thompson, is the primary contact with the Product Owner, Marlene Todd.

The project backlog will be tracked within Trello – an online collaboration and organization tool. Scrum Masters and Scrum Team Members are required access to this site. The Product Owner will not need to access this site. The Scrum Team will manage Sprint Backlog priorities based upon the Product Owners input. Sprints will last between two and three weeks. Sprint Reports will be sent to the Scrum Master and Product Owner at the end of scheduled Sprints.

The online repository, GitHub, will provide source control, source-code browser, and a project wiki. Scrum Team Members and the Scrum Master are required to have access to this site. [AT]

## Phase Overview

This system will be developed in phases, they are as follow: [AT]

### User Stories and Requirements Gathering

During this phase, the requirements for the system will be gathered from user stories. The product owner informs the team of any requirements, limitations or constraints. This information can be found in the Software Requirements Document and [3.0 Requirements](#_Requirements).

### Database Schema and Class Design

The database schema, or design, will be developed during the second phase. The team needs to mimic the existing Access Database currently being used by Lawrence County Teen Court, while making sure the design is as efficient as possible. The major class objects will also be designed during this phase; this includes defendants, volunteers, court listings, workshops and any other objects that can be derived from the requirements.

### Prototype

The prototype will provide limited functionality of the final application and act as a demonstration for the client. This will allow the team to make changes without major code rework. All major areas of the application should be built for the client review.

### Application Development

This phase will be the actual development phase. All aspects of the application will be implemented to the client’s specifications and requirements. Some unit testing will take place during this phase.

### Testing

Some aspects of testing will take place during the Application Development phase. This is to ensure proper data entry for the application. Individual domain constraints will be tested along with data accuracy and program functionality.

### Delivery

This phase is where the application will be turned over to the client. The web files and database will already be on the production server. User manuals for each user level will also be delivered in PDF format and hard copy. The user accounts used to access the website and database will also be turned over.

## Terminology and Acronyms

Listed here is common terms used throughout the document. [AT]

* AJAX: Asynchronous JavaScript and XML is a set of development tools to provide the client side to communicate with the server without refreshing the page.
* CAPTCHA: A CAPTCHA is a program that protects websites against bots by generating and grading tests that humans can pass but current computer programs cannot.
* Client: An application or system that accesses a service provided by a server.
* Court Program: A local city, county or community youth or teen court that uses this web application.
* Database: An organized set of data and associated data structures.
* DBMS: A Database Management System allows an interface to manipulate a database.
* Domain Name: An identification string that identifies represents an Internet Protocol (IP) resource. A Domain Name Server (DNS) handles the translation from Domain Name to IP address.
* JSON: JavaScript Object Notation is a lightweight data-interchange format.
* JQuery: JavaScript Framework that allows client manipulation.
* JQueryUI: JavaScript Framework that allows various client widgets to provide a more functional user interface.
* MySQL: A popular and reliable Database Management System (DBMS).
* PHP: PHP Hypertext Preprocessor is an HTML-embedded scripting language capable of Object Orientated Programming (OOP) principles.
* Server: A computer hardware system and software that serves as a dedicated host for one or more services.
* SSL: Secure Socket Layer is a cryptographic protocol that provides communication security over the Internet.
* Web Application: A software application that is developed in a browser-supported programming language and is accessed by users over an Intranet or Internet.
* Web Hosting: A hosting service on the Internet that allows website owners to access their site and typically provides web space management tools or control panels.

# Requirements

The purpose of this web application is to allow youth and teen courts to track participants within their programs. In order to achieve this, several requirements are specified from the stakeholders. [AT]

### User Access

Only individuals working for a particular court program may have access to the application. Defendants, their families, and volunteers will **not** be allowed to view **any** information contained in the application. The different levels are described in section [4.1 User Account and Access](#_User_Account_and).

### Teen/Youth Program & Data Access

The authorized users shall only have access to information pertaining to their particular court program. Court program administrators are able to adjust options pertaining to their individual programs. These options are covered in further detail in section [4.2 Court Program](#_Court_Program). Application Administrators will have the additional access to site-wide statistics and options to add new court programs. These levels of access are described in section [4.1 User Account and Access](#_User_Access).

### Defendant

The defendant is the primary data gathering point for this application. A defendant can be added, updated, deleted, and expunged. Each court program and their users can only access defendant information they enter. The different information fields are described in section [4.3 Defendant](#_Defendant).

#### Defendant Expunging

Each court program will have one of four options to expunge defendant information: none, sealed, partial expunge and full expunge. The sealed option retains defendant data within the database but restricts the data to be displayed in results other than demographic statistics. Partial expunging removes the defendant’s personal information from the database but leaves citation information for statistics gathering. Full expunge removes all traces of the defendant within the system.

### Volunteer

Volunteers are those individuals who assist with the teen court program. Generally they are assigned as court members or jury participants. The ability view status and volunteer history will be implemented. More information can be found in [4.4 Volunteer](#_Volunteer).

### Court

A user is able to create a court session at a certain time and location and attach defendant cases, court members and jury pool. Volunteer and defendant hours will be updateable. More information can be found in [4.5 Court](#_Courts).

### Workshop

Various workshops can be created and populated from open defendants. More information can be found in [4.6 Workshops](#_Workshops).

### Reports

All necessary and legal documents will be generated for the defendants.

### Statistics

Statistics reporting will include discoverable data, demographics data, and both local and regional statistics. Statistics have not been included at this time, rather they need to be produced once data has been gathered and area of statistics narrowed. They will not be implemented during this phase.

# Design and Implementation

The web application will be developed as several different areas or components. Components may interact with others depending upon their type or usage. Each component is described in detail below. The five major areas are the teen/youth program, user accounts and access, defendant, volunteers, courts and workshops.

Throughout the site, several areas will remember previously entered data and allow the user to select this data instead of re-entering it. These are displayed as small buttons next to the input field and looks like a window icon. Some of these areas are locations (city, state, zip), school information, statutes, sentencing requirements and common locations.

Whenever a date or time field is required, JQueryUI allows a stylized calendar or time selection window to appear. These fields also have client-side masking to ensure the date is formatted properly. The current format is MM/DD/YYYY HH:MM A. Dates get converted from user time zones to Central Standard Time (function in the core class) prior to database insertion. When the date is retrieved, it is then converted from Central Standard Time to the time zone set for the user. This ensures that proper times are displayed for users.

Phone number fields are also provided with a client-side validation mask. This ensures all phone numbers are entered in the same format. The current format is (123) 456-7890.

Some areas require a large input of data or have data that can be segmented to make the user interface less cluttered. When this is the case, the JQuery Tab elements are used. A client cookie is used to keep track of the current tab in case the user submits a form and gets redirected to the last page.

In some areas, such as defendant, the JQuery functions have been implemented with their own file. These are named jquery.js and control any JQuery elements on the component. If a jquery.js file is absent, the functions are at the top of the application pages themselves.

Almost all areas have a process.php page. This is where form POST and GET data gets sent, database interaction handled and redirection occurs. [AT]

## Program

Every youth or teen court program or association who uses this application will be assigned their own data space. This is the highest level component and almost all data is reliant upon it. Once all the necessary data has been entered, users can be added to it and they can start using the application. [AT]

### Component Overview

Only Application Administrators and the hybrid Application Administrator / Program Administrator can add or edit programs. The Program Administrators access level can only update their information.

Every program will have the following information available:

* Court Name
* Code
* Phone Number
* Time Zone
* Expunge Type
* Physical Address
* Mailing Address

Once the locations (city, state, zip) get entered, it can be selected throughout the site wherever a location is prompted. This is done to reduce data entry and possible user error.

Time zone is used as the default for users added to this program. This is used to adjust date and timestamp data from the database.

When a new program is added, a list of court positions is automatically added to the *court\_position* table where the program id is a foreign key. This is done by calling the stored procedure *addCourtPositions.* Program administrators can add their own custom positions in the My Program menu item. Volunteers use these for availability as court members. View the courts section for default member values.

### Application Pages

Not all of these pages are used at the same time. These are mainly areas that use some aspect of the program component.

* /admin/programs.php
* /admin/program.php
* /admin/view\_program.php
* /admin/process.php – all form POST/GET processing
* /includes/header\_internal.php
* /includes/secure.php
* /data/programs.php – for JSON data
* /data/programs\_common\_locations.php – for JSON data
* /data/programs\_locations.php – for JSON data
* /data/programs\_schools.php – for JSON data
* /data/programs\_sentences.php – for JSON data
* /data/programs\_statutes.php – for JSON data

### Database Tables

The following MySQL tables are primarily used for this component with program id used throughout the application as foreign key for many tables:

* teencour.program
* teencour.program\_common\_location
* teencour.program\_locations
* teencour.program\_officers
* teencour.program\_schools
* teencour.program\_sentences
* teencour.program\_statutes

### Classes

The following classes are primarily used or dependent for this component:

* /includes/class\_data.php
* /includes/class\_program.php
* /includes/class\_location.php
* /includes/class\_school.php
* /includes/class\_sentence.php

### Architecture Overview

Being the primary component, program data is used in a variety of ways in almost every section of the application. Mostly the program id is used as foreign key constraints on may tables.

### Design Overview

The only area where the program information is actually accessible is by the Application/Program Administrator or the My Program area where the Program Administrator can update or add various parts that rely on the program. These are the collected data such as locations, schools, etc.

## User Account and Access

This section describes the User Account and User Access development process. Users are validated by email address and a password. There are five different levels of access and each provides different functionality. Refer to figure 2 below. [AT]

C:\Users\1064529\Dropbox\School\Senior Design 2\graphics\UserDiagram.tif

Figure 3: User Access Levels

### Component Overview

The following areas are for the users can be accessed from the user menu option.

#### User Listing

There are two places to view users. One is the Application Administrator access level which can only view program and user information site-wide. The other is the Program Administrator which only has access to their specific users.

#### User View

Every user accessing the system will be required to have the following information in the database:

* Active or not
* ID of the Court Program they belong to
* Name (First and Last)
* Email Address
* Password (not actually stored in the database)
* Phone numbers
* Account Type

There are five account types: Application Administrator, a hybrid of both Program Administrator and Application Administrator (ex: Marlene’s positions), Program Administrator, Program Manager and Program User.

### Application Pages

This component is used widely throughout the site because of user access, but the following pages are where the primary user information is entered or accessed.

* /index.php – the main login page
* /register.php
* /profile.php
* /admin/users.php
* /admin/view\_users.php
* /admin/process.php – all form POST/GET processing
* /includes/secure.php
* /data/users.php – for JSON data
* /data/user\_history.php – for JSON data

### Database Tables

The following MySQL tables are primarily used for this component:

* teencour.program
* teencour.user
* teencour.user\_log
* teencour.user\_phone

### Classes

The following classes are associated with this component:

* /includes/class\_program.php
* /includes/class\_user.php
* /includes/class\_data.php – for JSON data

### Architecture Overview

The primary concern with user access is ensuring the user passwords cannot be accessed through unauthorized means. Prevention against hacking and brute force access scripts must be high priority. *~~There will be a maximum of 3 login attempts before the user is locked out of the system. They must then wait a certain amount of time before trying again. This prevents brute force scripted hacking attempts.~~ (Not Yet Implemented)*

The user’s password is not actually stored in the database; rather it is used in conjunction with a random value, a salt string and the user’s email address. This is hashed using the SHA-256 cryptographic hash function to create a 128-character string which is stored in the user table. To gain access, the first 64-characters from the previously hashed string are used as the salt string along with the password entered to create a new SHA-256 hashed string. The existing string is compared to the generated string to ensure user authentication. A new hash is then generated upon successful login.

New users will register their account from the registration page. The users will be given a court access code by their individual administrators and will need this to register with the application. The access code will be matched with the database to ensure it is a valid code. As the user enters a password, it will be checked against a strength algorithm, using AJAX, to make sure they are using a somewhat secure.

To avoid bots from spamming the registration page, a CAPTCHA system will be implemented to prove a human is registering for the application.

#### User Registration

Users are allowed to register for the application by using a code given to them by their program administrator. This code is generated when the program is created. After the user enters their credentials and passes the Google RECAPTCHA, the program administrator must approve or deny and assign a level to the user.

Figure 2 shows the architecture diagram of the registration process.



Figure 4: User Registration Architecture Diagram

#### User Access

A user gains access to the web application by entering their email address and password. If they are validated, their access level is set, login is recorded and session gets approved. A session will stay valid up to one hour of inactivity at which point the user will be logged out and the session destroyed.

Figure 3 is the architecture diagram of the user access implementation.



Figure 5: User Access Architecture Diagram

### Design Overview

The user profile and registration page use the JQuery PwStrength plugin to access the users password strength. The user is available to change their password, email, time zone or phone numbers at any time by clicking on their name in the upper right.

## Defendant

This section describes the Defendant development process. The Defendant is the main data object for this application. Several fields will be propagated via dropdown boxes or modal selection windows that the court administrator has access to. [AT]

### Component Overview

The following information is available for all defendants and available from the defendant menu option.



Figure 6: Defendant Tab Diagram

#### Defendant Listing

This section uses DataTables for data display. JSON Source file is located at: /data/defendants.php

#### Defendant View

Primary Defendant Information

* Name
  + First Name
  + Middle Initial
  + Last Name
* Date of Birth
* Home phone number
* Added
* Court Case number
* Agency Case number
* Closed date
* Expunged date

The added field is an automatic timestamp.

Personal Information (Tab)

* Physical Address & Location
* Mailing Address & Location
* School Data
  + School Name & Location
  + Grade level
  + Contact Name
  + Contact Phone Number
* Driver's License Data
  + Number
  + State
* Physical Description
  + Height
  + Weight
  + Sex
  + Eye Color
  + Hair Color
  + Ethnicity

Once a location or school is added, that information can be selected using the select button. This is done to avoid duplication and to prevent user data errors.

Parent/Guardian Information (Tab)

* Relationship
* First Name
* Last Name
* Home Number
* Work Number
* Employer
* Email Address
* Physical Address & Location
* Mailing Address & Location
* Living with that parent/guardian

There is no limit on the number of guardian or parents.

Citation Information (Tab)

* Citation Date
* Citation Time
* Location
* Common Place
* Citing Officer Data
* Mirandized Checkbox
* Drugs or Alcohol Involved
* Offense Data
  + Statue
  + Title
  + Type
* Stolen/Vandalized Data
  + Description of items
  + Value or Amount
* Vehicle Data
  + Year
  + Make
  + Model
  + Color
  + License Number
  + License State
  + Comments

Intake Information (Tab)

* Intake Date
* Intake Time
* Reschedule Date
* Reschedule Time
* Intake Interviewer
* Referred to Juvenile – Not Qualified
* Dismissed / No Complaint

Court Information (Tab)

* Court Assigned
  + Type
  + Venue
  + Location
  + Closed
* Jury Assigned
  + Venue
  + Location
  + Closed
  + Hours Assigned

Sentence Information (Tab)

* Type
* Requirement
* Complete

Multiple sentence information can be entered and once a sentence is added, it can be selected and used again.

Workshop (Tab)

* Workshops Attending
  + Date
  + Name
  + Venue
  + Location
  + Completed

Expunge (Tab)

* Order Signed
* Letter Completed
* Case Completed
* Defendant Evaluation Completed
* Parent Evaluation Completed
* Workshop Completed
* Referred to Juvenile Date

### Application Pages

The application pages containing this component are:

* /defendant/index.php
* /defendant/view.php

These files are part of view.php by using php’s include function:

* + /defendant/tab\_personal.php
  + /defendant/tab\_guardian.php
  + /defendant/tab\_citation.php
  + /defendant/tab\_intake.php
  + /defendant/tab\_court.php
  + /defendant/tab\_sentence.php
  + /defendant/tab\_workshop.php
  + /defendant/tab\_expunge.php
  + /defendant/tab\_forms.php
  + /defendant/tab\_notes.php
* /defendant/process.php – all form POST/GET processing
* /defendant/search.php
* /defendant/jquery.js – JQuery functions

### Database Tables

Since the defendant is one of the primary components, it is used in a majority of areas via foreign key relations. These are the primary tables that defendant information is contained:

* teencour.defendant
* teencour.guardian
* teencour.citation
* teencour.citation\_offense
* teencour.citation\_stolen\_items
* teencour.citation\_vehicle
* teencour.defendant\_sentence
* teencour.intake\_information

### Classes

The following classes are associated with this component:

* /includes/class\_defendant.php
* /includes/class\_location.php
* /includes/class\_school.php
* /includes/class\_guardian.php
* /includes/class\_citation.php
* /includes/class\_workshop.php
* /includes/class\_workshop\_location.php
* /includes/class\_court.php
* /includes/class\_court\_location.php
* /includes/class\_sentence.php
* /includes/class\_data.php – for JSON data

### Architecture Overview

Again, once some specific areas have been added, like location, schools, common locations, etc. – they will be available from the modal selection windows throughout the section.

Under the citation tab, the user can select existing offenses or immediately add a new one. Also, stolen items or vehicles involved can be added. There is no limit on how many can be added.

The court and workshop tabs just list what the defendant is currently enrolled in or assigned to. Adding the defendant needs to be done in those particular components.

When a defendant is expunged, it is done so based on the programs expunge type. Depending on this value, different data will be removed from the defendant record or not at all.

### Design Overview

This section will be compromised of two main areas, the top defendant information and a multiple-tab area containing additional information described above. Once the primary information is added, the user will have access to the additional tabbed areas.

## Volunteer

Volunteers are used to fulfill court positions or be a member of a jury. They need to be assigned to a court and have their hours tracked. [AT]

### Component Overview

The following areas are accessible from the volunteer menu option.

#### Volunteer Listing

This section uses DataTables for data display. JSON Source file is located at: /data/volunteers.php

#### Volunteer View

Primary Information

* Active or not
* First Name
* Last Name
* Phone #
* Email

Tabbed Information

* Court Positions
* Hours volunteered

The court positions are generated with a default list when a program is added. This is done by the stored procedure call *addCourtPositions.* Program Administrators can add custom positions. View the court section for default member values. The hours volunteered display a list of courts they have been a member of or a part of a jury for.

### Application Pages

The following application pages are where the main information is for the volunteer component:

* /volunteer/index.php
* /volunteer/view.php
* /volunteer/search.php
* /volunteer/process.php (all form actions go here)

### Database Tables

The following MySQL tables are primarily used for this component:

* volunteer
* volunteer\_position

### Classes

The following classes are associated with this component:

* /includes/class\_volunteer.php
* /includes/class\_data.php – for JSON data

### Architecture Overview

The volunteer is a pretty straight forward component. A person gets added to the program and court positions are selected. They can then be added to a court.

### Design Overview

Once primary volunteer is added, the tabbed area comes available to select court positions. This is just a list of positions and checkboxes.

## Courts

Courts are where the defendants will have their cases judged by a jury of their peers and/or volunteers. Volunteers can act as various court positions. Court positions are automatically generated when a program is created, but the Program Administrator has the ability to add more. The initial positions available are:

* Judge
* Prosecuting Attorney
* Defense Attorney
* Clerk
* Bailiff
* Exit Interviewer
* Advisor
* Jury

Volunteer hours are also recorded in this component. [AT]

### Component Overview

The following areas are accessible from the court menu option or the initial page upon login.

#### Court Listing

This section uses DataTables for data display. JSON Source file is located at: /data/courts.php. These data tables are under the court listing area in the menu and the home menu item. The initial age upon login, or /main.php page contains a list of upcoming court dates.

#### Court View

* Court Date
* Court Time
* Court Type
* Contract Signed
* Closed
* Court Address
  + Name
  + Location
* Court Members
* Jury Members

The court name and location are recorded and can be selected from a list by the modal select windows.

### Application Pages

The application pages containing this component are:

* /court/index.php
* /court/view.php

These files are part of view.php by using php’s include function:

* + /court/tab\_members.php
  + /court/tab\_jury.php
* /court/hour\_entry.php
* /court/hours.php
* /court/process.php – all form POST/GET processing
* /court/search.php
* /court/jquery.js – JQuery functions

### Database Tables

The following MySQL tables are primarily used for this component:

* court
* court\_guardian
* court\_defendant
* court\_jury\_defendant
* court\_jury\_volunteer
* court\_location
* court\_member
* court\_position

### Classes

The following classes are associated with this component:

* /includes/class\_court.php
* /includes/c lass\_court\_location.php

### Architecture Overview

### Design Overview

## Workshops

Workshops are primarily used as sentencing requirements for defendants. Once a workshop location is entered once, it can be selected with the modal selection button. Once a workshop is created, defendants can be added to it. If the defendant attended the workshop, they can be marked as having it completed. [AT]

### Component Overview

The following areas are accessible from the workshop menu option.

#### Workshop Listing

This section uses DataTables for data display. JSON Source file is located at: /data/workshops.php.

#### Workshop View

* Title
* Date
* Time
* Instructor
* Officer
* Description
* Workshop Address
  + Venue Name
  + Address
  + Location

### Application Pages

The application pages containing this component are:

* /workshop/index.php
* /workshop/view.php
* /workshop/search.php
* /workshop/process.php – all form POST/GET processing
* /workshop/jquery.js – JQuery functions

### Database Tables

The following MySQL tables are primarily used for this component:

* teencour.workshop
* teencour.workshop\_location
* teencour.workshop\_roster

### Classes

The following classes are associated with this component:

* /includes/class\_workshop.php
* /includes/class\_workshop\_location.php

### Architecture Overview

### Design Overview

## Statistics

Statistics will not be implemented in this version.

## Surveys

Surveys will not be implemented in this version.

## Program Administration Options

Not yet implemented – will be developed during summer 2013.

## Application Administration Options

Not yet implemented – will be developed during summer 2013.

# Testing

## Overview

## Dependencies

## Test Setup and Execution

# Development Environment

This section describes programs used to develop the application, where it will be hosted and what kind of environment the server is.

## Development IDE and Tools

For MySQL schema model and database management, MySQL Workbench will be used: <http://dev.mysql.com/downloads/workbench/>. Aptana Studio 3 will be used as the IDE for development of the web application and the prototype: [http://www.aptana.com/products/studio3](http://www.aptana.com/products/studio3%20). The Aptana IDE has integrated GitHub support.

## Source Control

GitHub will be used for the primary file repository and source control. Builds will be pushed to GitHub after every editing session. The link is: <https://github.com/SDSMT-CSC/TCD>.

## Dependencies

PHP 5.2.17, MySQL 5.5, MySQL PDO, JQuery and JQueryUI using Google Hosted Library.

## Build Environment

The build environment is a Linux server running Apache HTTP Web Server and MySQL Database Management System. The servers are located in Dallas, TX. The login credentials are located in [Account Credentials](#_Account_Credentials). The dedicated IP address: 50.22.71.90

## Development Machine Setup

The application will be built on the existing web space. In the event of any hosting or connection issues that may arise during development, the team has access to a private, temporary server with the same server environment. This way progress on the application will not be hindered while the issue is resolved. [AT]

# Release | Setup | Deployment

The site is already released and live on <http://teencourtdb.com>. [AT]

## Setup Information

An initial program with id set to 0 is the default program needed to add others. It is listed as Teen Court DB.

## System Versioning Information

Prototypes were developed and numbered 1 to 4 and are available on GitHub. The most current version is always located at <http://teencourtdb.com>.

# End User Documentation

A user guide will be made available in PDF format on GitHub and delivered to the client. [RR]

* + - 1. List of Figures

[Figure 1: System Diagram 10](#_Toc354528347)

[Figure 2: Major Component Diagram 10](#_Toc354528348)

[Figure 3: User Access Levels 17](#_Toc354528349)

[Figure 4: User Registration Architecture Diagram 19](#_Toc354528350)

[Figure 5: User Access Architecture Diagram 20](#_Toc354528351)

[Figure 6: Defendant Tab Diagram 20](#_Toc354528352)

* + - 1. Supporting Information and Details
         1. Database Schema

CREATE TABLE IF NOT EXISTS `citation`   
  (   
     `citationid`     *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `defendantid`    *INT*(10) UNSIGNED NOT NULL,   
     `officerid`      *INT*(10) UNSIGNED DEFAULT NULL,   
     `date`           *TIMESTAMP* NULL DEFAULT NULL,   
     `address`        *VARCHAR*(45) DEFAULT NULL,   
     `locationid`     *INT*(11) DEFAULT NULL,   
     `mirandized`     *TINYINT*(1) DEFAULT NULL,   
     `drugsoralcohol` *TINYINT*(1) DEFAULT NULL,   
     `commonplaceid`  *INT*(10) UNSIGNED DEFAULT NULL,   
     `added`          *TIMESTAMP* NOT NULL DEFAULT CURRENT\_TIMESTAMP,   
     PRIMARY KEY (`citationid`),   
     KEY `fk\_citation\_court\_common\_place1\_idx` (`commonplaceid`),   
     KEY `fk\_citation\_defendant1\_idx` (`defendantid`),   
     KEY `fk\_citation\_citation\_officer1\_idx` (`officerid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `citation\_offense`   
  (   
     `offenseid`   *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `defendantid` *INT*(10) UNSIGNED NOT NULL,   
     `statuteid`   *INT*(10) UNSIGNED NOT NULL,   
     PRIMARY KEY (`offenseid`)   
  ) engine= innodb;   
  
CREATE TABLE IF NOT EXISTS `citation\_stolen\_items`   
  (   
     `itemid`     *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `citationid` *INT*(10) UNSIGNED NOT NULL,   
     `name`       *VARCHAR*(45) DEFAULT NULL,   
     `value`      *DECIMAL*(18, 2) DEFAULT NULL,   
     PRIMARY KEY (`itemid`),   
     KEY `fk\_citation\_stolen\_items\_citation1\_idx` (`citationid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `citation\_vehicle`   
  (   
     `vehicleid`     *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `citationid`    *INT*(10) UNSIGNED NOT NULL,   
     `licensenumber` *VARCHAR*(50) DEFAULT NULL,   
     `licensestate`  *VARCHAR*(50) DEFAULT NULL,   
     `make`          *VARCHAR*(50) DEFAULT NULL,   
     `model`         *VARCHAR*(50) DEFAULT NULL,   
     `year`          *VARCHAR*(50) DEFAULT NULL,   
     `color`         *VARCHAR*(50) DEFAULT NULL,   
     `comment`       *TEXT*,   
     PRIMARY KEY (`vehicleid`),   
     KEY `fk\_citation\_vehicle\_citation1\_idx` (`citationid`)   
  ) engine=innodb; 

CREATE TABLE IF NOT EXISTS `court`   
  (   
     `courtid`         *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `programid`       *INT*(10) UNSIGNED NOT NULL,   
     `defendantid`     *INT*(10) UNSIGNED NOT NULL,   
     `courtlocationid` *INT*(10) UNSIGNED DEFAULT NULL,   
     `type`            *VARCHAR*(45) NOT NULL,   
     `contract`        *TINYINT*(1) DEFAULT NULL,   
     `date`            *DATETIME* NOT NULL,   
     `timeentered`     *TINYINT*(1) NOT NULL DEFAULT '0',   
     `closed`          *DATETIME* DEFAULT NULL,   
     PRIMARY KEY (`courtid`, `programid`),   
     KEY `fk\_jury\_court1\_idx` (`programid`),   
     KEY `fk\_jury\_defendant1\_idx` (`defendantid`),   
     KEY `fk\_trial\_trial\_location1\_idx` (`courtlocationid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `court\_guardian`   
  (   
     `courtid`    *INT*(10) UNSIGNED NOT NULL,   
     `guardianid` *INT*(10) UNSIGNED NOT NULL,   
     PRIMARY KEY (`courtid`, `guardianid`),   
     KEY `fk\_trial\_guardian\_guardian1\_idx` (`guardianid`),   
     KEY `fk\_trial\_guardian\_trial1\_idx` (`courtid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `court\_jury\_defendant`   
  (   
     `courtid`     *INT*(10) UNSIGNED NOT NULL,   
     `defendantid` *INT*(10) UNSIGNED NOT NULL,   
     `hours`       *DECIMAL*(8, 2) DEFAULT NULL,   
     PRIMARY KEY (`courtid`, `defendantid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `court\_jury\_volunteer`   
  (   
     `courtid`     *INT*(10) UNSIGNED NOT NULL,   
     `volunteerid` *INT*(10) UNSIGNED NOT NULL,   
     `hours`       *DECIMAL*(8, 2) DEFAULT NULL,   
     PRIMARY KEY (`courtid`, `volunteerid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `court\_location`   
  (   
     `courtlocationid` *INT*(11) NOT NULL auto\_increment,   
     `locationid`      *INT*(10) UNSIGNED NOT NULL,   
     `programid`       *INT*(10) UNSIGNED NOT NULL,   
     `name`            *VARCHAR*(45) DEFAULT NULL,   
     `address`         *VARCHAR*(45) DEFAULT NULL,   
     PRIMARY KEY (`courtlocationid`),   
     KEY `locationid\_unique` (`locationid`)   
  ) engine=innodb; 

CREATE TABLE IF NOT EXISTS `court\_member`   
  (   
     `courtid`     *INT*(10) UNSIGNED NOT NULL,   
     `volunteerid` *INT*(10) UNSIGNED NOT NULL,   
     `positionid`  *INT*(10) UNSIGNED NOT NULL,   
     `hours`       *DECIMAL*(8, 2) DEFAULT NULL,   
     PRIMARY KEY (`courtid`, `positionid`, `volunteerid`),   
     KEY `fk\_jury\_pool\_jury1\_idx` (`courtid`),   
     KEY `fk\_trial\_members\_trial\_position1\_idx` (`positionid`),   
     KEY `fk\_jury\_pool\_volunteer1` (`volunteerid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `court\_position`   
  (   
     `positionid` *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `programid`  *INT*(10) UNSIGNED NOT NULL,   
     `position`   *VARCHAR*(45) NOT NULL,   
     PRIMARY KEY (`positionid`),   
     KEY `fk\_jury\_position\_court1` (`programid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `custom\_data`   
  (   
     `customid`    *INT*(10) UNSIGNED NOT NULL,   
     `defendantid` *INT*(10) UNSIGNED NOT NULL,   
     `value`       *VARCHAR*(45) DEFAULT NULL,   
     PRIMARY KEY (`customid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `custom\_fields`   
  (   
     `fieldid`   *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `programid` *INT*(10) UNSIGNED NOT NULL,   
     `fieldname` *VARCHAR*(45) DEFAULT NULL,   
     `required`  *TINYINT*(1) DEFAULT NULL,   
     PRIMARY KEY (`fieldid`, `programid`),   
     UNIQUE KEY `fieldid\_unique` (`fieldid`),   
     KEY `fk\_custom\_fields\_court1\_idx` (`programid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `defendant`   
  (   
     `defendantid`        *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `programid`          *INT*(10) UNSIGNED NOT NULL,   
     `firstname`          *VARCHAR*(45) NOT NULL,   
     `lastname`           *VARCHAR*(45) NOT NULL,   
     `middlename`         *VARCHAR*(45) DEFAULT NULL,   
     `homephone`          *VARCHAR*(45) DEFAULT NULL,   
     `dob`                *VARCHAR*(25) NOT NULL,   
     `paddress`           *VARCHAR*(150) DEFAULT NULL,   
     `plocationid`        *INT*(11) DEFAULT NULL,   
     `maddress`           *VARCHAR*(150) DEFAULT NULL,   
     `mlocationid`        *INT*(11) DEFAULT NULL,   
     `schoolid`           *INT*(11) DEFAULT NULL,   
     `schoolcontactname`  *VARCHAR*(150) DEFAULT NULL,   
     `schoolcontactphone` *VARCHAR*(50) DEFAULT NULL,   
     `schoolgrade`        *VARCHAR*(15) DEFAULT NULL,   
     `height`             *VARCHAR*(25) DEFAULT NULL,   
     `weight`             *VARCHAR*(25) DEFAULT NULL,   
     `eyecolor`           *VARCHAR*(25) DEFAULT NULL,   
     `haircolor`          *VARCHAR*(25) DEFAULT NULL,   
     `sex`                *VARCHAR*(25) DEFAULT NULL,   
     `ethnicity`          *VARCHAR*(25) DEFAULT NULL,   
     `licensenum`         *VARCHAR*(25) DEFAULT NULL,   
     `licensestate`       *VARCHAR*(25) DEFAULT NULL,   
     `notes`              *TEXT*,   
     `courtcasenumber`    *VARCHAR*(45) DEFAULT NULL,   
     `agencycasenumber`   *VARCHAR*(45) DEFAULT NULL,   
     `expungedate`        *DATETIME* DEFAULT NULL,   
     `closedate`          *DATE* DEFAULT NULL,   
     `added`              *TIMESTAMP* NOT NULL DEFAULT CURRENT\_TIMESTAMP,   
     PRIMARY KEY (`defendantid`, `programid`),   
     KEY `fk\_defendant\_program1\_idx` (`programid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `defendant\_sentence`   
  (   
     `defsentid`   *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `sentenceid`  *INT*(11) NOT NULL,   
     `defendantid` *INT*(11) NOT NULL,   
     `value`       *VARCHAR*(255) DEFAULT NULL,   
     `additional`  *VARCHAR*(255) DEFAULT NULL,   
     `complete`    *DATE* DEFAULT NULL,   
     PRIMARY KEY (`defsentid`),   
     KEY `index` (`sentenceid`),   
     KEY `defendant` (`defendantid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `guardian`   
  (   
     `guardianid`  *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `defendantid` *INT*(10) UNSIGNED NOT NULL,   
     `relation`    *VARCHAR*(25) DEFAULT NULL,   
     `firstname`   *VARCHAR*(45) DEFAULT NULL,   
     `lastname`    *VARCHAR*(45) DEFAULT NULL,   
     `paddress`    *VARCHAR*(150) DEFAULT NULL,   
     `plocationid` *INT*(11) DEFAULT NULL,   
     `maddress`    *VARCHAR*(45) DEFAULT NULL,   
     `mlocationid` *INT*(11) DEFAULT NULL,   
     `homephone`   *VARCHAR*(45) DEFAULT NULL,   
     `employer`    *VARCHAR*(45) DEFAULT NULL,   
     `workphone`   *VARCHAR*(45) DEFAULT NULL,   
     `email`       *VARCHAR*(100) DEFAULT NULL,   
     `liveswith`   *TINYINT*(1) DEFAULT NULL,   
     PRIMARY KEY (`guardianid`),   
     KEY `fk\_parents\_defendant1\_idx` (`defendantid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `intake\_information`   
  (   
     `defendantid` *INT*(11) NOT NULL,   
     `intake`      *DATETIME* NOT NULL,   
     `reschedule`  *DATETIME* DEFAULT NULL,   
     `inteviewer`  *INT*(11) NOT NULL,   
     `referred`    *DATETIME* DEFAULT NULL,   
     `dismissed`   *DATETIME* DEFAULT NULL,   
     PRIMARY KEY (`defendantid`)   
  ) engine=innodb;   
CREATE TABLE IF NOT EXISTS `program`   
  (   
     `programid`  *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `code`       *VARCHAR*(75) NOT NULL,   
     `name`       *VARCHAR*(150) NOT NULL,   
     `paddress`   *VARCHAR*(150) DEFAULT NULL,   
     `pcity`      *VARCHAR*(45) DEFAULT NULL,   
     `pstate`     *CHAR*(2) DEFAULT NULL,   
     `pzip`       *VARCHAR*(25) DEFAULT NULL,   
     `maddress`   *VARCHAR*(150) DEFAULT NULL,   
     `mcity`      *VARCHAR*(45) DEFAULT NULL,   
     `mstate`     *CHAR*(2) DEFAULT NULL,   
     `mzip`       *VARCHAR*(25) DEFAULT NULL,   
     `phone`      *VARCHAR*(25) NOT NULL,   
     `expunge`    *TINYINT*(4) NOT NULL,   
     `timezoneid` *INT*(10) NOT NULL,   
     `active`     *TINYINT*(1) NOT NULL DEFAULT '1',   
     `added`      *TIMESTAMP* NOT NULL DEFAULT CURRENT\_TIMESTAMP,   
     PRIMARY KEY (`programid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `program\_common\_location`   
  (   
     `commonplaceid` *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `programid`     *INT*(10) UNSIGNED NOT NULL,   
     `commonplace`   *VARCHAR*(45) DEFAULT NULL,   
     PRIMARY KEY (`commonplaceid`, `programid`),   
     KEY `fk\_court\_common\_place\_court1\_idx` (`programid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `program\_locations`   
  (   
     `locationid` *INT*(11) NOT NULL auto\_increment,   
     `programid`  *INT*(11) NOT NULL,   
     `city`       *VARCHAR*(50) NOT NULL,   
     `state`      *CHAR*(2) NOT NULL,   
     `zip`        *VARCHAR*(25) NOT NULL,   
     PRIMARY KEY (`locationid`),   
     KEY `fk\_programid\_program` (`programid`)   
  ) engine=myisam;   
  
CREATE TABLE IF NOT EXISTS `program\_officers`   
  (   
     `officerid` *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `programid` *INT*(10) UNSIGNED NOT NULL,   
     `firstname` *VARCHAR*(45) DEFAULT NULL,   
     `lastname`  *VARCHAR*(50) DEFAULT NULL,   
     `idnumber`  *VARCHAR*(50) NOT NULL,   
     `phone`     *VARCHAR*(45) DEFAULT NULL,   
     PRIMARY KEY (`officerid`),   
     KEY `fk\_citation\_officer\_court1\_idx` (`programid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `program\_schools`   
  (   
     `schoolid`   *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `programid`  *INT*(10) UNSIGNED NOT NULL,   
     `schoolname` *VARCHAR*(45) DEFAULT NULL,   
     `address`    *VARCHAR*(45) DEFAULT NULL,   
     `city`       *VARCHAR*(45) DEFAULT NULL,   
     `state`      *VARCHAR*(45) DEFAULT NULL,   
     `zip`        *VARCHAR*(45) DEFAULT NULL,   
     PRIMARY KEY (`schoolid`, `programid`),   
     KEY `fk\_school\_information\_program1\_idx` (`programid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `program\_sentences`   
  (   
     `sentenceid`  *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `programid`   *INT*(10) UNSIGNED NOT NULL,   
     `name`        *VARCHAR*(150) DEFAULT NULL,   
     `description` *VARCHAR*(150) DEFAULT NULL,   
     `additional`  *VARCHAR*(255) DEFAULT NULL,   
     PRIMARY KEY (`sentenceid`),   
     KEY `programid` (`programid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `program\_statutes`   
  (   
     `statuteid`   *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `programid`   *INT*(10) UNSIGNED NOT NULL,   
     `statute`     *VARCHAR*(50) DEFAULT NULL,   
     `title`       *VARCHAR*(50) DEFAULT NULL,   
     `description` *TEXT*,   
     PRIMARY KEY (`statuteid`, `programid`),   
     UNIQUE KEY `statuteid\_unique` (`statuteid`),   
     KEY `fk\_citation\_court1\_idx` (`programid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `survey`   
  (   
     `surveyid`    *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `description` *MEDIUMTEXT*,   
     `created`     *TIMESTAMP* NOT NULL DEFAULT CURRENT\_TIMESTAMP,   
     `sent`        *DATETIME* DEFAULT NULL,   
     PRIMARY KEY (`surveyid`),   
     UNIQUE KEY `surveyid\_unique` (`surveyid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `survey\_answers`   
  (   
     `surveyid`   *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `userid`     *INT*(10) UNSIGNED NOT NULL,   
     `questionid` *INT*(10) UNSIGNED NOT NULL,   
     `answer`     *VARCHAR*(45) DEFAULT NULL,   
     PRIMARY KEY (`surveyid`, `userid`, `questionid`),   
     KEY `fk\_survey\_answers\_survey1\_idx` (`surveyid`),   
     KEY `fk\_survey\_answers\_survey\_questions1\_idx` (`questionid`),   
     KEY `fk\_survey\_answers\_user1\_idx` (`userid`)   
  ) engine=innodb; 

CREATE TABLE IF NOT EXISTS `survey\_options`   
  (   
     `optionid`   *VARCHAR*(45) NOT NULL,   
     `questionid` *INT*(10) UNSIGNED NOT NULL,   
     `option`     *VARCHAR*(45) DEFAULT NULL,   
     PRIMARY KEY (`optionid`),   
     KEY `fk\_survey\_options\_survey\_questions1\_idx` (`questionid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `survey\_questions`   
  (   
     `questionid` *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `surveyid`   *INT*(10) UNSIGNED NOT NULL,   
     `question`   *VARCHAR*(255) DEFAULT NULL,   
     `type`       *INT*(10) UNSIGNED DEFAULT NULL,   
     PRIMARY KEY (`questionid`, `surveyid`),   
     UNIQUE KEY `questionid\_unique` (`questionid`),   
     KEY `fk\_survey\_questions\_survey1\_idx` (`surveyid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `timezone`   
  (   
     `timezoneid` *INT*(11) NOT NULL auto\_increment,   
     `display`    *VARCHAR*(100) NOT NULL,   
     `timezone`   *VARCHAR*(100) NOT NULL,   
     PRIMARY KEY (`timezoneid`)   
  ) engine= innodb;   
  
CREATE TABLE IF NOT EXISTS `user`   
  (   
     `userid`     *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `programid`  *INT*(10) UNSIGNED NOT NULL,   
     `typeid`     *INT*(10) UNSIGNED NOT NULL,   
     `firstname`  *VARCHAR*(75) NOT NULL,   
     `lastname`   *VARCHAR*(75) NOT NULL,   
     `email`      *VARCHAR*(255) NOT NULL,   
     `timezoneid` *INT*(10) NOT NULL,   
     `hash`       *CHAR*(128) NOT NULL,   
     `createdate` *TIMESTAMP* NOT NULL DEFAULT CURRENT\_TIMESTAMP,   
     `lastlogin`  *DATETIME* DEFAULT NULL,   
     `active`     *TINYINT*(1) NOT NULL DEFAULT '0',   
     `deleted`    *TINYINT*(1) NOT NULL DEFAULT '0',   
     PRIMARY KEY (`userid`),   
     KEY `programid` (`programid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `user\_log`   
  (   
     `logid`      *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `userid`     *INT*(10) UNSIGNED NOT NULL,   
     `action`     *VARCHAR*(45) NOT NULL,   
     `recordid`   *INT*(11) DEFAULT NULL,   
     `ip\_address` *VARCHAR*(45) NOT NULL,   
     `date`       *TIMESTAMP* NOT NULL DEFAULT CURRENT\_TIMESTAMP,   
     PRIMARY KEY (`logid`),   
     KEY `fk\_user\_log\_user1` (`userid`)   
  ) engine=innodb; 

CREATE TABLE IF NOT EXISTS `user\_phone`   
  (   
     `phoneid`  *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `userid`   *INT*(10) UNSIGNED NOT NULL,   
     `phonenum` *VARCHAR*(45) NOT NULL,   
     `ext`      *VARCHAR*(5) DEFAULT NULL,   
     `type`     *VARCHAR*(25) DEFAULT NULL,   
     PRIMARY KEY (`phoneid`),   
     KEY `fk\_user\_phone\_user1` (`userid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `user\_type`   
  (   
     `typeid` *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `type`   *VARCHAR*(50) NOT NULL,   
     `active` *TINYINT*(1) NOT NULL,   
     PRIMARY KEY (`typeid`)   
  ) engine=myisam;   
  
CREATE TABLE IF NOT EXISTS `volunteer`   
  (   
     `volunteerid` *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `programid`   *INT*(10) UNSIGNED NOT NULL,   
     `firstname`   *VARCHAR*(45) DEFAULT NULL,   
     `lastname`    *VARCHAR*(45) DEFAULT NULL,   
     `phone`       *VARCHAR*(45) DEFAULT NULL,   
     `email`       *VARCHAR*(255) DEFAULT NULL,   
     `active`      *TINYINT*(1) DEFAULT '1',   
     PRIMARY KEY (`volunteerid`, `programid`),   
     UNIQUE KEY `volid\_unique` (`volunteerid`),   
     KEY `fk\_volunteer\_court1\_idx` (`programid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `volunteer\_position`   
  (   
     `volunteerid` *INT*(10) UNSIGNED NOT NULL,   
     `positionid`  *INT*(10) UNSIGNED NOT NULL,   
     PRIMARY KEY (`volunteerid`, `positionid`),   
     KEY `fk\_volunteer\_position\_court\_position1\_idx` (`positionid`),   
     KEY `fk\_volunteer\_position\_volunteer1\_idx` (`volunteerid`)   
  ) engine=innodb;   
  
CREATE TABLE IF NOT EXISTS `workshop`   
  (   
     `workshopid`         *INT*(10) UNSIGNED NOT NULL auto\_increment,   
     `programid`          *INT*(10) UNSIGNED NOT NULL,   
     `date`               *TIMESTAMP* NULL DEFAULT NULL,   
     `title`              *VARCHAR*(45) DEFAULT NULL,   
     `description`        *TEXT*,   
     `instructor`         *VARCHAR*(45) DEFAULT NULL,   
     `officerid`          *INT*(11) DEFAULT NULL,   
     `workshoplocationid` *INT*(10) DEFAULT NULL,   
     PRIMARY KEY (`workshopid`, `programid`)   
  ) engine=innodb; 

CREATE TABLE IF NOT EXISTS `workshop\_location`   
  (   
     `workshoplocationid` *INT*(11) NOT NULL auto\_increment,   
     `locationid`         *INT*(11) NOT NULL,   
     `programid`          *INT*(11) NOT NULL,   
     `name`               *VARCHAR*(45) DEFAULT NULL,   
     `address`            *VARCHAR*(45) DEFAULT NULL,   
     PRIMARY KEY (`workshoplocationid`)   
  ) engine= innodb;   
  
CREATE TABLE IF NOT EXISTS `workshop\_roster`   
  (   
     `workshopid`  *INT*(10) UNSIGNED NOT NULL,   
     `defendantid` *INT*(10) UNSIGNED NOT NULL,   
     `completed`   *DATETIME* DEFAULT NULL,   
     PRIMARY KEY (`workshopid`, `defendantid`),   
     KEY `fk\_workshop\_roster\_workshop1\_idx` (`workshopid`),   
     KEY `fk\_workshop\_roster\_defendant1\_idx` (`defendantid`)   
  ) engine=innodb;   
  
ALTER TABLE `citation`   
  ADD CONSTRAINT `fk\_citation\_court\_common\_place1` FOREIGN KEY (`commonplaceid`)   
  REFERENCES `program\_common\_location` (`commonplaceid`) ON DELETE no action ON   
  UPDATE no action,   
  ADD CONSTRAINT `fk\_citation\_defendant1` FOREIGN KEY (`defendantid`) REFERENCES   
  `defendant` (`defendantid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `citation\_stolen\_items`   
  ADD CONSTRAINT `fk\_citation\_stolen\_items\_citation1` FOREIGN KEY (`citationid`)   
  REFERENCES `citation` (`citationid`) ON DELETE no action ON UPDATE no action,   
  ADD CONSTRAINT `fk\_stolen\_items\_citation` FOREIGN KEY (`citationid`)   
  REFERENCES `citation` (`citationid`);   
  
ALTER TABLE `citation\_vehicle`   
  ADD CONSTRAINT `fk\_citation\_vehicle\_citation1` FOREIGN KEY (`citationid`)   
  REFERENCES `citation` (`citationid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `court`   
  ADD CONSTRAINT `fk\_jury\_court1` FOREIGN KEY (`programid`) REFERENCES `program`   
  (`programid`) ON DELETE no action ON UPDATE no action,   
  ADD CONSTRAINT `fk\_jury\_defendant1` FOREIGN KEY (`defendantid`) REFERENCES   
  `defendant` (`defendantid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `court\_guardian`   
  ADD CONSTRAINT `fk\_trial\_guardian\_guardian1` FOREIGN KEY (`guardianid`)   
  REFERENCES `guardian` (`guardianid`) ON DELETE no action ON UPDATE no action,   
  ADD CONSTRAINT `fk\_trial\_guardian\_trial1` FOREIGN KEY (`courtid`) REFERENCES   
  `court` (`courtid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `court\_jury\_defendant`   
  ADD CONSTRAINT `fk\_jury\_member\_def\_trial1` FOREIGN KEY (`courtid`) REFERENCES   
  `court` (`courtid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `court\_member`   
  ADD CONSTRAINT `fk\_jury\_pool\_jury1` FOREIGN KEY (`courtid`) REFERENCES `court`   
  (`courtid`) ON DELETE no action ON UPDATE no action,   
  ADD CONSTRAINT `fk\_jury\_pool\_volunteer1` FOREIGN KEY (`volunteerid`)   
  REFERENCES `volunteer` (`volunteerid`) ON DELETE no action ON UPDATE no action   
,   
  ADD CONSTRAINT `fk\_trial\_members\_trial\_position1` FOREIGN KEY (`positionid`)   
  REFERENCES `court\_position` (`positionid`) ON DELETE no action ON UPDATE no   
  action;   
  
ALTER TABLE `court\_position`   
  ADD CONSTRAINT `fk\_jury\_position\_court1` FOREIGN KEY (`programid`) REFERENCES   
  `program` (`programid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `custom\_data`   
  ADD CONSTRAINT `fk\_custom\_data\_custom\_fields1` FOREIGN KEY (`customid`)   
  REFERENCES `custom\_fields` (`fieldid`) ON DELETE no action ON UPDATE no action   
;   
  
ALTER TABLE `custom\_fields`   
  ADD CONSTRAINT `fk\_custom\_fields\_court1` FOREIGN KEY (`programid`) REFERENCES   
  `program` (`programid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `defendant`   
  ADD CONSTRAINT `fk\_defendant\_program1` FOREIGN KEY (`programid`) REFERENCES   
  `program` (`programid`);   
  
ALTER TABLE `guardian`   
  ADD CONSTRAINT `fk\_parents\_defendant1` FOREIGN KEY (`defendantid`) REFERENCES   
  `defendant` (`defendantid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `program\_common\_location`   
  ADD CONSTRAINT `fk\_court\_common\_place\_court1` FOREIGN KEY (`programid`)   
  REFERENCES `program` (`programid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `program\_officers`   
  ADD CONSTRAINT `fk\_citation\_officer\_court1` FOREIGN KEY (`programid`)   
  REFERENCES `program` (`programid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `program\_sentences`   
  ADD CONSTRAINT `program\_sentences\_ibfk\_1` FOREIGN KEY (`programid`) REFERENCES   
  `program` (`programid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `program\_statutes`   
  ADD CONSTRAINT `fk\_citation\_court1` FOREIGN KEY (`programid`) REFERENCES   
  `program` (`programid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `survey\_answers`   
  ADD CONSTRAINT `fk\_survey\_answers\_survey1` FOREIGN KEY (`surveyid`) REFERENCES   
  `survey` (`surveyid`) ON DELETE no action ON UPDATE no action,   
  ADD CONSTRAINT `fk\_survey\_answers\_survey\_questions1` FOREIGN KEY (`questionid`   
  ) REFERENCES `survey\_questions` (`questionid`) ON DELETE no action ON UPDATE   
  no action,   
  ADD CONSTRAINT `fk\_survey\_answers\_user1` FOREIGN KEY (`userid`) REFERENCES   
  `user` (`userid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `survey\_options`   
  ADD CONSTRAINT `fk\_survey\_options\_survey\_questions1` FOREIGN KEY (`questionid`   
  ) REFERENCES `survey\_questions` (`questionid`) ON DELETE no action ON UPDATE   
  no action;   
  
ALTER TABLE `survey\_questions`   
  ADD CONSTRAINT `fk\_survey\_questions\_survey1` FOREIGN KEY (`surveyid`)   
  REFERENCES `survey` (`surveyid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `user`   
  ADD CONSTRAINT `fk\_user\_court` FOREIGN KEY (`programid`) REFERENCES `program`   
  (`programid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `user\_log`   
  ADD CONSTRAINT `fk\_user\_log\_user1` FOREIGN KEY (`userid`) REFERENCES `user` (   
  `userid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `user\_phone`   
  ADD CONSTRAINT `fk\_user\_phone\_user1` FOREIGN KEY (`userid`) REFERENCES `user`   
  (`userid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `volunteer`   
  ADD CONSTRAINT `fk\_volunteer\_court1` FOREIGN KEY (`programid`) REFERENCES   
  `program` (`programid`) ON DELETE no action ON UPDATE no action;   
  
ALTER TABLE `volunteer\_position`   
  ADD CONSTRAINT `fk\_volunteer\_position\_court\_position1` FOREIGN KEY (   
  `positionid`) REFERENCES `court\_position` (`positionid`) ON DELETE no action   
  ON UPDATE no action,   
  ADD CONSTRAINT `fk\_volunteer\_position\_volunteer1` FOREIGN KEY (`volunteerid`)   
  REFERENCES `volunteer` (`volunteerid`) ON DELETE no action ON UPDATE no action   
;   
  
ALTER TABLE `workshop\_roster`   
  ADD CONSTRAINT `fk\_workshop\_roster\_defendant1` FOREIGN KEY (`defendantid`)   
  REFERENCES `defendant` (`defendantid`) ON DELETE no action ON UPDATE no action   
,   
  ADD CONSTRAINT `fk\_workshop\_roster\_workshop1` FOREIGN KEY (`workshopid`)   
  REFERENCES `workshop` (`workshopid`) ON DELETE no action ON UPDATE no action;

DELIMITER $$

CREATE DEFINER=`teencour`@`localhost` PROCEDURE `addCourtPositions`( IN pID INT )

BEGIN

INSERT INTO court\_position (programid,position) VALUES ( pid,'Judge');   
INSERT INTO court\_position (programid,position) VALUES ( pid,'Prosecuting Attorney');   
INSERT INTO court\_position (programid,position) VALUES ( pid,'Defense Attorney');   
INSERT INTO court\_position (programid,position) VALUES ( pid,'Clerk');   
INSERT INTO court\_position (programid,position) VALUES ( pid,'Bailiff');   
INSERT INTO court\_position (programid,position) VALUES ( pid,'Exit Interviewer');   
INSERT INTO court\_position (programid,position) VALUES ( pid,'Advisor');   
INSERT INTO court\_position (programid,position) VALUES ( pid,'Jury');

END$$

DELIMITER;

* + - 1. Sprint Reports

This section will contain a complete list of all of the period progress and/or sprint reports which are deliverables for the phases and versions of the system.

* + - * 1. Sprint 1 Progress Report

Download the complete Sprint 1 report from GitHub: [Sprint Report 1.pdf](https://github.com/SDSMT-CSC/TCD/blob/master/Documents/Sprint%20Report%201.pdf)

* + - * 1. Sprint 2 Progress Report

Download the complete Sprint 2 report from GitHub: [Sprint Report 2.pdf](https://github.com/SDSMT-CSC/TCD/blob/master/Documents/Sprint%20Report%202.pdf)

* + - * 1. Sprint 3 Progress Report

Download the complete Sprint 3 report from GitHub: [Sprint Report 3.pdf](https://github.com/SDSMT-CSC/TCD/blob/master/Documents/Sprint%20Report%203.pdf)

* + - * 1. Sprint 4 Progress Report

Download the complete Sprint 4 report from GitHub: [Sprint Report 4.pdf](https://github.com/SDSMT-CSC/TCD/blob/master/Documents/Sprint%20Report%204.pdf)

* + - * 1. Sprint 5 Progress Report

Download the complete Sprint 5 report from GitHub: [Sprint Report 5.pdf](https://github.com/SDSMT-CSC/TCD/blob/master/Documents/Sprint%20Report%205.pdf)

* + - * 1. Sprint 6 Progress Report

Download the complete Sprint 6 report from GitHub: [Sprint Report 6.pdf](https://github.com/SDSMT-CSC/TCD/blob/master/Documents/Sprint%20Report%206.pdf)

* + - 1. Class Documentation

This section will contain a complete list of all PHP classes used covering class purpose, class members, and class methods.

* + - * 1. Citation

Description: The citation class stores information about a defendant’s citation information, what offenses they are charged with, what items were stolen, and if vehicles were used. [RR]

Class Members

citationID: Private, Unique identifier for this citation.

defendantID: Private, Unique identifier for the defendant.

officerID: Public, Unique identifier for the arresting officer.

citationDate: Public, Date of citation.

address: Public, Address of citation location.

locationID: Public, Unique identifier for location. Holds city, state, and zip fields.

commonLocationID: Public, Unique identifier for common places. Holds a short description of the location.

mirandized: Public, Whether the defendant was mirandized or not.

drugsOrAlcohol: Public, Whether drugs or alcohol was present at the time of citation.

Class Methods

\_\_construct($ defendantID ): Public, Loads citation information for the given defendantID into citation object, otherwise returns with an empty object.

updateCitation(): Public, Updates the citation information in the database if citationID is filled, otherwise it creates a new row and inserts the information.

getOffenseList( $user\_type ): Public, Gets the offenses for the defendant based off of defendantID and citationID. Option to remove offense is visible depending on user type.

addOffense( $statuteID ): Public, Adds new offense to record.

removeOffense( $offenseID ): Public, Removes offense from record.

getStolenItems(): Public, Gets stolen items based off of citationID.

addStolenItem( $name, $value ): Public, Records the name and value of an item and adds to record.

removeStolenItem( $itemID ): Public, Removes an item from record.

getVehicles():Public, Gets vehicles used in offense.

addVehicle( $year, $make, $model, $color, $license, $state, $comment ): Public, Adds a vehicle to record.

removeVehicle( $vehicleID ): Public, Removes a vehicle from record.

getDefendantID():Public, Returns DefendantID.

getCitationID():Public, Returns CitationID.

* + - * 1. Core

Description: Core class used to handle connections to the database. [RR]

Class Members

dbh: Public, PDO for connecting to the database.

instance: Private static, object holding instance of open database connection

Class Methods

\_\_construct(): Private, Sets up PDO object.

dbOpen(): Public static, Create a new instance of a database connection.

dbClose(): Public static, Closes the database connection.

covertToServerDate( $originalTS, $userTimeZone): Public, Converts time stamps to Chicago time zone to correctly store within the database.

* + - * 1. Court Location

Description: The court location handles information for where courts will be held. [RR]

Class Members

courtLocationID: Private, Unique identifier for this court location

programID: Private, Unique identifier for the program.

locationID: Public, Unique identifier for location. Holds city, state, and zip fields.

name: Public, Location name.

address: Public, Location address.

city: Public, Location city, brought in with locationID.

state: Public, Location state, brought in with locationID.

zip: Public, Location zip, brought in with locationID.

Class Methods

\_\_construct():Public, create empty court location object.

updateCourtLocation(): Public, adds a new court location to database if it doesn’t exist in the database, otherwise updates the location if they don’t match.

getCourtLocation( $workshopLocID ): Public, loads court location object with data based on courtLocID.

getCourtLocationID(): Public, returns courtLocationID.

getProgramID(): Public, returns programID.

setLocationID( $val ): Public, sets locationID.

setProgramID($val ): Public, sets programID.

* + - * 1. Court

Description: The court class is used to assign defendants to court and manage court positions filled by volunteers.

Class Members

courtID: Private, unique identifier for court.

programID: Private, unique identifier for program.

defendantID: Private, unique identifier for defendant.

courtDate: Public, date and time court is to be held.

type: Public, what kind of court it is (hearing, trial)

contractSigned: Public, if the defendant signed the contract to go through the Teen Court program.

closed: Public, if the court was completed.

courtLocationID: Public, unique identifier for court location.

timeEntered: Public, mark if the time has been set for members of this court

Class Methods

\_\_construct( $user\_programID ): Public, create an empty court object.

updateCourt(): Public, adds court if courtID is 0, otherwise updates court.

getFromID( $id ): Public, gets court information from courtID.

compareProgramID( $id, $user\_program ): Public, returns true if user’s programID matches court’s programID.

deleteCourt(): Public, removes everything from this court.

getCourtMembers(): Public, get court members for this program.

updateCourtMembers( $members ): Public, update all assigned court members.

existingCourtMembers(): Public, Gets a list of existing court members for a particular court, used to make volunteer active in the court member dropdown lists.

getJuryMembers(): Public, Gets a list of existing court members for a particular court, used to make volunteer active in the court member dropdown lists.

updateJuryMember(): Public, update all assigned jury members.

deleteJuryMember( $id, $type): Public, deletes a jury member from the assigned jury pool.

updateCourtGuardians( $guardians ): Public, updates guardians attending a particular court.

checkGuardianAttending(): Public, returns an array of guardians attending a particular court, used to check dropdowns.

getMembers for Time( $type ): Public, returns an array of members for a particular court and their time spent.

setMembersTime( $globalHrs, $members, $jury ): Public, sets time spent for members/jury on a particular court.

setDefendantID( $val ): Public, sets defendantID.

setCourtID( $val ): Public, sets courtID.

getDefendantID(): Public, return defendantID.

getCourtID(): Public, return courtID.

* + - * 1. Data

Description: The Data class is used to fetch information that is displayed with the dataTables and dropdown menu options. [RR]

Class Members

None

Class Methods

fetchUserListing( $user\_programID, $user\_type ): Public, fetches all users or only users in the program depending on user type into JSON encoding.

fetchProgramListing():Public, fetches all programs into JSON encoding.

fetchProgramLocations( $user\_programID ): Public, fetches locations for the given program into a JSON object.

fetchProgramSchools( $user\_programID ): Public, fetches schools for the given program into a JSON object.

fetchProgramStatutes( $user\_programID ): Public, fetches statutes for the given program into a JSON object.

fetchProgramSentences( $user\_programID ): Public, fetches sentence for the given program into a JSON object.

fetchProgramDropdown( $id ): Public, generates a dropdown list of programs that are active, marks a program as selected depending on $id.

fetchUserTypeDropdown( $id, $utype ): Public, fetches the possible user types that a user could be assigned into a JSON object.

fetchTimezoneDropdown( $id ): Public, generates a dropdown list of possible timezones.

fetchCourtListing( $user\_programID ): Public, fetches all courts for the given program into a JSON object.

fetchCourtLocation( $user\_programID ): Public, fetches court locations for the given program into a JSON object.

fetchCourtJuryPool( $user\_programID ): Public, fetches possible jury pool members for the given program into a JSON object.

checkJuror( $courtID, $jurorID, $type ): Public, checks a juror to see if they are available for the court listing or not.

fetchDefendantListing( $user\_programID ): Public, fetches defendants for the given program who are still going through the Teen Court program into a JSON object.

fetchVolunteerListing( $user\_programID ): Public, fetches active volunteers for the given program into a JSON object.

fetchWorkshopListing( $user\_programID ): Public, fetches workshops for the given program into a JSON object.

fetchWorkshopDefendantsListing( $user\_programID ): Public, fetches defendants from the given program to be workshop participants into a JSON object.

fetchWorkshopLocation( $user\_programID ): Public, fetches workshop locations for the given program into a JSON object.

fetchProgramCommonLocation( $user\_programID ): Public, fetches common places for the given program into a JSON object.

* + - * 1. Defendant

Description: The defendant class is used to handle and protect information about defendants within the Teen Court program.

Class Members

defendantID: Private, unique identifier for the defendant.

programID: Private, unique identifier for the program.

firstName: Private, defendant’s first name.

lastName: Private, defendant’s last name.

middleName: Private, defendant’s middle name.

phoneNumber: Private, defendant’s phone number.

dateOfBirth: Private, defendant’s date of birth.

courtCaseNumber: Private, court case defendant is assigned to.

agencyNumber: Private, agency defendant is assigned to.

expungeDate: Private, date defendant was expunged from Teen Court program.

closeDate: Private, date defendant’s trial was closed.

pID: Public, unique identifier for physical city, state, and zip.

pAddress: Public, address defendant lives at.

pCity: Public, city defendant lives in, brought in through pID.

pState: Public, state defendant lives in, brought in through pID.

pZip: Public, zip code defendant lives in, brought in through pID.

mID: Public, unique identifier for mailing city, state, and zip.

mAddress: Public, address defendant’s mail goes to.

mCity: Public, city defendant’s mail goes to, brought in through mID.

mState: Public, state defendant’s mail goes to, brought in through mID.

mZip: Public, zip code defendant’s mail goes to, brought in through mID.

schoolID: Public, unique identifier for defendant’s school.

schoolContactName: Public, staff member to contact at school.

schoolContactPhone: Public, staff member’s phone.

schoolGrade: Public, defendant’s grade level.

height: Public, defendant’s height.

weight: Public, defendant’s weight.

eyecolor: Public, defendant’s eye color.

haircolor: Public, defendant’s hair color.

sex: Public, defendant’s gender.

ethnicity: Public, defendant’s ethnicity.

licenseNum: Public, defendant’s driver’s license number.

licenseState: Public, state that issued defendant’s driver’s license.

notes: Public, notes about the defendant.

intake: Public, date and time for intake interview.

reschedule: Public, date and time for rescheduled interview.

interviewer: Public, unique identifier for person interviewing defendant.

referred: Public, date and time defendant was referred to juvenile system.

dismissed; Public, date and time defendant was dismissed from Teen Court.

added: Public, date and time defendant was added to program.

Class Methods

\_\_construct(): Public, create empty defendant object.

getFromID( $id ): Public, gets defendant information from a defendant id.

updateDefendant(): Public, adds the defendant if userid is 0, otherwise updates the defendant record.

updatePersonal(): Public, updates the defendant's personal information. This is done after initially adding a defendant to the database or when editing one.

updateIntake(): Public, updates the defendant’s intake information. This is done after initially adding a defendant to the database.

getGuardianList(): Public, returns an array of guardianIDs for the defendant.

totalGuardians(): Public, returns a count of guardians for the defendant.

checkWorkshop(): Public, returns if the defendant is in a workshop and if they have completed it.

checkCourt(): Public, returns if the defendant has been assigned to a court and if they have completed it.

checkJury(): Public, returns if the defendant has been assigned to any courts as a jury member and hours assigned so far.

checkSentence(): Public, returns list of sentences that have been assigned to the defendant .

updateNotes(): Public, updates defendant notes.

setDefendantID( $str ): Public, sets defendantID.

setProgramID( $str ): Public, sets programID.

setFirstName( $str ): Public, sets firstName.

setLastName( $str ): Public, sets lastName.

setMiddleName( $str ): Public, sets middleName.

setPhoneNumber( $str ): Public, sets phoneNumber.

setDateOfBirth( $str ): Public, sets dateOfBirth.

setCourtCaseNumber( $str ): Public, sets courtCaseNumber.

setAgencyNumber( $str ): Public, sets agencyNumber.

getDefendantID(): Public, returns defendantID.

getProgramID(): Public, returns programID.

getFirstName(): Public, returns firstName.

getLastName(): Public, returns lastName.

getMiddleName(): Public, returns middleName.

getPhoneNumber(): Public, returns phoneNumber.

getDateOfBirth(): Public, returns dateOfBirth.

getCourtCaseNumber(): Public, returns courtCaseNumber.

getAgencyNumber(): Public, returns agencyNumber.

getExpungeDate(): Public, returns expungeDate or N/A if expungeDate is null.

getCloseDate(): Public, returns closeDate or N/A if closeDate is null.

* + - * 1. Guardian

Description: The guardian class is used to handle information about a defendant’s parent or guardian. [RR]

Class Members

defendantID: Private, Unique identifier for the defendant.

guardianID: Private, Unique identifier for the guardian.

relation: Public, Guardian’s relation to the defendant.

firstName: Public, Guardian’s first name.

lastName: Public, Guardian’s last name.

homePhone: Public, Guardian’s home phone number.

workPhone: Public, Guardian’s work phone number.

employer: Public, Guardian’s employer.

email: Public, Guardian’s email.

pAddress: Public, Guardian’s physical address.

pID: Public, Unique identifier for guardian’s physical city, state, and zip.

mAddress: Public, Guardian’s mailing address.

mID: Public, Unique identifier for guardian’s mailing city, state, and zip.

liveswith: Public, If the defendant lives with the guardian.

Class Methods

\_\_construct( $defendantID ) Public, creates a guardian object with the given defendantID.

getFromID( $id ): Public, loads the guardian object with information based off the given ID.

updateGuardian(): Public, inserts or updates the guardian object into the database depending on if guardianID is set.

removeGuardian():Public, deletes guardian from database.

setGuardianID( $str ): Public, set guardianID.

getGuardianID():Public, return guardianID.

getDefendantID():Public, return defendantID.

* + - * 1. Location

Description: The Location class stores city, state, and zip information for use in other classes without needing to store the same information in different places. [RR]

Class Members

programID: Private, Unique identifier for the program.

locationID: Public, Unique identifier for the location.

city: Public, City name.

state: Public, State city is in.

zip: Public, City’s zip code.

Class Methods

\_\_construct( $programID ): Public, create an empty location object with the given programID.

getFromID( $id ): Public, load the location object with information based off the given ID.

findLocation( $city, $state, $zip ): Public, return location object based off programID, city, state, and zip.

addLocation( $city, $state, $zip ): Public, if the location city, state, and zip is not in the database, adds the location to the database. Otherwise retrieves the locationID.

* + - * 1. Class Program

Description: The Program class is used to set up and manage the individual Teen Court programs. [RR]

Class Members

programID: Private, Unique identifier for the program.

code: Private, Used to allow other users to register to the program.

name: Private, Program name.

phys\_address: Public, Program’s physical address.

phys\_city: Public, Program’s physical city.

phys\_state: Public, Program’s physical state.

phys\_zip: Public, Program’s physical zip code.

mail\_address: Public, Program’s mailing address.

mail\_city: Public, Program’s mailing city.

mail\_state: Public, Program’s mailing state.

mail\_zip: Public, Program’s mailing zip code.

phone: Public, Program’s phone number.

expunge: Public, Program’s expunge method.

timezoneID: Public, Program’s timezone.

active: Public, If the program is active or not.

Class Methods

\_\_construct(): Public, creates an empty program object.

programExists( $code ): Public, checks the database to see if a program exists.

getFromCode( $code ): Public, gets program information from an existing program code.

getFromID( $id ): Public, gets program information from id.

updateProgram(): Public, updates or adds the program depending on if programID is set.

fetchUserDropdown( $userID ): Public, returns dropdown options of users in the program. Will return with a user selected if userID matches.

fetchOfficerDropdown( $officerID): Public, returns dropdown options of officers in the program. Will return with an officer selected if officerID matches.

addCommonLocation( $location ): Public, inserts a new common place if location is new and returns the commonplaceID, otherwise just returns the commonplaceID.

getCommonLocation( $location ): Public, returns the common place name.

addOfficer( $firstname, $lastname, $idNumber, $phone ): Public, inserts the officer into the database.

addStatute( $programID, $code, $title, $description ): Public, inserts the statute into the database.

getProgramPositions(): Public, returns key and ID of court positions.

addSentence( $name, $description, $additional ): Public, inserts the sentence into the database.

getProgramID(): Public, gets ProgramID.

getName(): Public, gets program name.

getCode(): Public, Gets program code.

getFullAddress(): Public, Gets the program’s physical address.

setProgramID( $str ): Public, Sets ProgramID.

setName( $str ): Public, Sets program name.

setCode( $str ): Public, Sets program code.

* + - * 1. School

Description: The School class is used to manage information about the schools that defendants attend. [RR]

Class Members

programID: Private, unique identifier for the program.

schoolID: Public, unique identifier for the school.

name: Public, school’s name.

address: Public, school’s address.

city: Public, school’s city.

state: Public, school’s state.

zip: Public, school’s zip code.

Class Methods

\_\_construct( $programID ): Public, creates an empty school object with the given programID.

getFromID( $id ): Public, loads the school information that matches.

findSchool( $name, $address, $city, $state, $zip ): Public, return a school object that matches all fields.

addSchool( $name, $address, $city, $state, $zip ): Public, this function looks up a school based on name, address, city, state and zip code. if it is found, the id is returned. If not, it is added to the school table. This prevents any duplicate information in the database.

* + - * 1. Sentence

Description: The Sentence class is used to connect sentences a court gives to a defendant. [RR]

Class Members

defsentID: Private, unique identifier for the sentence that a defendant was given.

sentenceID: Private, unique identifier for the sentence that a program has.

defendantID: Private, unique identifier for the defendant.

name: Public, name of sentence.

description: Public, description of what the sentence will require to be completed.

additional: Public, optional extra field for sentence.

additionalValue: Public, value of additional field.

completeDate: Public, date for when sentence was completed.

Class Methods

\_\_construct( $defendantID ): Public, create empty sentence object with the given ID.

addSentenceFunction( $defendantID, $sentenceID ): Private, assigns a sentence to a defendant.

addSentence( $sentences ): Public, add new sentence for defendant.

getFromID( $defsentID ): Public, gets sentence information.

removeSentence( $defsentID ): Public, removes sentence requirement.

updateSentence(): Public, updates sentence requirements

setDefendantID( $var ): Public, sets defendantID.

getDefendantID(): Public, returns defendantID.

getSentenceID(): Public, returns sentenceID.

* + - * 1. User

Description: The user class is used to add and edit users, verify and update user password, and ensure only one email is used within the system.

Class Members

userID: Private, unique identifier for the user.

programID: Private, unique identifier for the program.

typeID: Private, determines user access levels.

firstName: Private, user first name.

lastName: Private, user last name.

email: Private, user email.

password: Private, hashed and salted version of user password.

lastLogin: Private, last time user last logged into account.

timezoneID: Private, user local timezoneID.

timezone: Private, user local timezone as string.

active: Private, if user is active or ‘deleted’.

Class Methods

\_\_construct(): Public, create empty user object.

getFromLogin( $email, $password ): Public, checks the email and password the user entered at login with record in the database. If the email address exists, the password is checked with the current hash. If the hash is the same, proceed with user access and login.

getFromID ($id ): Public, gets user information from a user id.

checkHash( $hash ): Private, checks a current hash against the algorithm to verify integrity.

newHash(): Private, generate a new hash based on email and password.

updateUser(): Public, adds the user if userid is 0, otherwise updates the user record.

removeUser( $id ): Public, marks the user as deleted and inactive in the database, doesn't actually remove the user

emailExists( $email ): Public, checks the database to see if an email address exists for a user.

fetchPhoneNumbers(): Public, gets the users phone numbers.

addPhone( $type, $number, $ext ): Public, to add a phone number to the user.

removePhone( $id ): Public, to remove a phone number from the user.

addEvent( $event, $id = NULL ): Public, logs a user action.

fetchHistory( $id ): gets a list of user’s actions.

getUserID(): Public, return userID.

getFirstName(): Public, return firstName.

getLastName(): Public, return lastName.

getName(): Public, returns firstName and lastName.

getProgramID(): Public, returns programID.

getType(): Public, returns typeID.

getTimezone(): Public, returns timezone.

getTimezoneID(): Public, returns timezoneID.

getLastLogin(): Public, returns lastLogin.

getEmail(): Public, returns email.

isActive(): Public, returns active.

setUserID($val ): Public, sets userID.

setProgramID($val ): Public, sets programID.

setType($val ): Public, sets typeID.

setFirstName($val ): Public, sets firstName.

setLastName($val ): Public, sets lastName.

setEmail($val ): Public, sets email.

setPassword($val ): Public, sets password.

setTimezoneID($val ): Public, sets timezoneID.

setActive($val ): Public, sets active.

display(): Public, for testing, shows all fields for user.

* + - * 1. Volunteer

Description: The volunteer class is for handling volunteer information, positions, and hours assisting the court. [RR]

Class Members

volunteerID: Private, unique identifier for the volunteer.

programID: Private, unique identifier for the program.

firstName: Private, volunteer’s first name.

lastName: Private, volunteer’s last name.

phone: Private, volunteer’s phone number.

email: Private, volunteer’s email.

positions: Private, array of volunteer positions.

active: Private, whether volunteer is active or not.

Class Methods

\_\_construct( $programID ): Public, creates empty volunteer object.

getVolunteer( $id ): Public, returns volunteer information based on id.

updateVolunteer(): Public, if volunteerID is empty, inserts volunteer into database. Otherwise updates volunteer information. After insert/update, clears all positions the volunteer holds in the database and inserts new positions.

clearPositions(): Public, deletes all positions that the volunteer is currently assigned.

deleteVolunteer(): Public, sets volunteer to inactive.

editVolunteerHours(): Public,

printVolunteerHours(): Public,

getVolunteerID(): Public, returns volunteerID.

getProgramID(): Public, returns programID.

getFirstName(): Public, returns firstName.

getLastName(): Public, returns lastName.

getPhone(): Public, returns phone.

getEmail(): Public, returns email.

getPositions(): Public, returns positions.

getActive(): Public, returns active.

setVolunteerID($val ): Public, sets volunteerID.

setProgramID($val ): Public, sets programID.

setFirstName($val ): Public, sets firstName.

setLastname($val ): Public, sets lastName.

setPhone($val ): Public, sets phone.

setEmail($val ): Public, sets email.

setPositions($val ): Public, sets positions.

setActive($val ): Public, sets active.

* + - * 1. Class Workshop Location

Description: The workshop location class handles information for where workshops will be held. [RR]

Class Members

workshopLocationID: Public, unique identifier for this court location

locationID: Public, unique identifier for location. Holds city, state, and zip fields.

programID: Public, unique identifier for the program.

name: Public, location name.

address: Public, location address.

city: Public, location city, brought in with locationID.

state: Public, location state, brought in with locationID.

zip: Public, location zip, brought in with locationID.

Class Methods

\_\_construct(): Public, reates an empty workshop location object.

updateWorkshopLocation(): Public, adds a new workshop or edits an existing workshop depending on workshopLocationID.

getWorkshopLocation( $workshopLocID ): Public, loads workshop location object with data based on workshopLocID.

getWorkshopLocationID():Public, return locationID.

getProgramID():Public, returns programID.

setLocationID($val ): Public, set locationID.

setProgramID($val ): Public, set programID.

* + - * 1. Workshop

Description: The workshop class is used to create, modify and remove workshops and workshop participants. [RR]

Class Members

workshopID: Private, Unique identifier for the workshop.

programID: Private, Unique identifier for the program.

date: Private, date and time of the workshop.

title: Private, title of workshop.

description: Private, description of workshop.

instructor: Private, workshop’s instructor.

officerID: Private, unique identifier for the officer if one is assisting the workshop.

workshopLocationID: Private, unique identifier for the workshop location.

Class Methods

\_\_construct( $user\_programID ): Public, creates a new workshop object.

updateWorkshop(): Public, inserts a new workshop into the database if workshopID is empty, otherwise updates the information for that workshop.

deleteWorkshop(): Public, deletes workshop roster and workshop.

getWorkshop( $id ): Public, returns workshop information retrieved based on id.

addWorkshopParticipant( $workshopID, $defendantID ): Public, adds defendant and program to roster.

removeWorkshopParticipant( $workshopID, $defendantID ): Public, deletes defendant from workshop.

completedWorkshopParticipant( $workshopID, $defendantID ): Public, marks defendant as having completed the workshop at current time.

listWorkshopParticipants( $id ): Public, returns list of workshop participants for the given workshop.

getWorkshopID(): Public, returns workshopID.

getProgramID(): Public, returns programID.

getDate(): Public, returns date.

getTitle(): Public, returns title.

getDescription(): Public, returns description.

getInstructor(): Public, returns instructor.

getOfficerID(): Public, returns officerID.

getworkshopLocationID(): Public, returns workshopLocationID.

setWorkshopID($val ): Public, sets workshopID.

setProgramID($val ): Public, sets programID.

setDate($val ): Public, sets date.

setTitle($val ): Public, sets title.

setDescription($val ): Public, sets description.

setInstructor($val ): Public, sets instructor.

setOfficerID($val ): Public, sets officerID.

setworkshopLocationID($val ): Public, sets workshopLocationID.