Teen Court Database

Software Design Document | Current Version 1.0.4

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 |
| 4/18/13 | Bobby Reilly | 1.0.5 | Added class documentation |

Table of Contents

[1.0 Overview 8](#_Toc352958819)

[1.1 Scope 8](#_Toc352958820)

[1.2 Purpose 8](#_Toc352958821)

[1.2.1 Domain Name and Web Space 8](#_Toc352958822)

[1.2.2 Server Environment and Database 8](#_Toc352958823)

[1.2.3 Web Application 8](#_Toc352958824)

[1.2.4 Security 8](#_Toc352958825)

[1.3 Systems Goals 9](#_Toc352958826)

[1.4 System Overview and Diagram 9](#_Toc352958827)

[1.5 Technologies Overview 9](#_Toc352958828)

[2.0 Project Overview 10](#_Toc352958829)

[2.1 Team Members and Roles 10](#_Toc352958830)

[2.2 Project Management Approach 10](#_Toc352958831)

[2.3 Phase Overview 10](#_Toc352958832)

[2.3.1 User Stories and Requirements Gathering 10](#_Toc352958833)

[2.3.2 Database Schema and Class Design 11](#_Toc352958834)

[2.3.3 Prototype 11](#_Toc352958835)

[2.3.4 Application Development 11](#_Toc352958836)

[2.3.5 Testing 11](#_Toc352958837)

[2.3.6 Delivery 11](#_Toc352958838)

[2.4 Terminology and Acronyms 11](#_Toc352958839)

[3.0 Requirements 12](#_Toc352958840)

[3.1.1 User Access 12](#_Toc352958841)

[3.1.2 Court Program & Data Access 12](#_Toc352958842)

[3.1.3 Defendant 12](#_Toc352958843)

[3.1.4 Volunteer 13](#_Toc352958844)

[3.1.5 Workshop 13](#_Toc352958845)

[3.1.6 Reports and Statistics 13](#_Toc352958846)

[4.0 Design and Implementation 13](#_Toc352958847)

[4.1 User Account and Access 13](#_Toc352958848)

[4.1.1 Technologies Used 13](#_Toc352958849)

[4.1.2 Component Overview 13](#_Toc352958850)

[4.1.3 Phase Overview 14](#_Toc352958851)

[4.1.4 Architecture Overview 14](#_Toc352958852)

[4.1.5 Design Details 16](#_Toc352958853)

[4.2 Court Program 16](#_Toc352958854)

[4.2.1 Technologies Used 16](#_Toc352958855)

[4.2.2 Component Overview 16](#_Toc352958856)

[4.2.3 Phase Overview 16](#_Toc352958857)

[4.2.4 Architecture Overview 16](#_Toc352958858)

[4.2.5 Design Details 16](#_Toc352958859)

[4.3 Defendant 16](#_Toc352958860)

[4.3.1 Technologies Used 16](#_Toc352958861)

[4.3.2 Component Overview 16](#_Toc352958862)

[4.3.3 Phase Overview 19](#_Toc352958863)

[4.3.4 Architecture Overview 19](#_Toc352958864)

[4.3.5 Design Details 19](#_Toc352958865)

[4.4 Volunteer 19](#_Toc352958866)

[4.4.1 Technologies Used 19](#_Toc352958867)

[4.4.2 Component Overview 19](#_Toc352958868)

[4.4.3 Phase Overview 19](#_Toc352958869)

[4.4.4 Architecture Diagram 19](#_Toc352958870)

[4.4.5 Data Flow Diagram 19](#_Toc352958871)

[4.4.6 Design Details 19](#_Toc352958872)

[4.5 Courts 19](#_Toc352958873)

[4.5.1 Technologies Used 19](#_Toc352958874)

[4.5.2 Component Overview 20](#_Toc352958875)

[4.5.3 Phase Overview 20](#_Toc352958876)

[4.5.4 Architecture Overview 20](#_Toc352958877)

[4.5.5 Design Details 20](#_Toc352958878)

[4.6 Workshops 20](#_Toc352958879)

[4.6.1 Technologies Used 20](#_Toc352958880)

[4.6.2 Component Overview 20](#_Toc352958881)

[4.6.3 Phase Overview 20](#_Toc352958882)

[4.6.4 Architecture Overview 20](#_Toc352958883)

[4.6.5 Design Details 20](#_Toc352958884)

[4.7 Reports and Statistics 20](#_Toc352958885)

[4.7.1 Technologies Used 20](#_Toc352958886)

[4.7.2 Component Overview 21](#_Toc352958887)

[4.7.3 Phase Overview 21](#_Toc352958888)

[4.7.4 Architecture Overview 21](#_Toc352958889)

[4.7.5 Design Details 21](#_Toc352958890)

[4.8 Surveys 21](#_Toc352958891)

[4.8.1 Technologies Used 21](#_Toc352958892)

[4.8.2 Database Table Schemas 21](#_Toc352958893)

[4.8.3 Component Overview 21](#_Toc352958894)

[4.8.4 Phase Overview 21](#_Toc352958895)

[4.8.5 Architecture Overview 21](#_Toc352958896)

[4.8.6 Design Details 21](#_Toc352958897)

[4.9 Program Administration Options 21](#_Toc352958898)

[4.9.1 Technologies Used 21](#_Toc352958899)

[4.9.2 Component Overview 22](#_Toc352958900)

[4.9.3 Phase Overview 22](#_Toc352958901)

[4.9.4 Architecture Overview 22](#_Toc352958902)

[4.9.5 Design Details 22](#_Toc352958903)

[4.10 Application Administration Options 22](#_Toc352958904)

[4.10.1 Technologies Used 22](#_Toc352958905)

[4.10.2 Component Overview 22](#_Toc352958906)

[4.10.3 Phase Overview 22](#_Toc352958907)

[4.10.4 Architecture Overview 22](#_Toc352958908)

[4.10.5 Design Details 22](#_Toc352958909)

[4.11 Miscellaneous 22](#_Toc352958910)

[5.0 System and Unit Testing 23](#_Toc352958911)

[5.1 Overview 23](#_Toc352958912)

[5.2 Dependencies 23](#_Toc352958913)

[5.3 Test Setup and Execution 23](#_Toc352958914)

[6.0 Development Environment 23](#_Toc352958915)

[6.1 Development IDE and Tools 23](#_Toc352958916)

[6.2 Source Control 23](#_Toc352958917)

[6.3 Dependencies 23](#_Toc352958918)

[6.4 Build Environment 24](#_Toc352958919)

[6.5 Development Machine Setup 24](#_Toc352958920)

[7.0 Release | Setup | Deployment 24](#_Toc352958921)

[7.1 Deployment Information and Dependencies 24](#_Toc352958922)

[7.2 Setup Information 24](#_Toc352958923)

[7.3 System Versioning Information 24](#_Toc352958924)

[8.0 End User Documentation 24](#_Toc352958925)

[Appendix I: List of Figures 25](#_Toc352958926)

[Appendix II: Supporting Information and Details 26](#_Toc352958927)

[II.1 Database Schema 26](#_Toc352958928)

[Appendix III: Progress | Sprint Reports 38](#_Toc352958929)

[III.1 Sprint 1 Progress Report 38](#_Toc352958930)

[III.2 Sprint 2 Progress Report 38](#_Toc352958931)

[III.3 Sprint 2 Progress Report 38](#_Toc352958932)

[Appendix IV: Class Documentation 38](#_Toc352958933)

[IV.1 Class Citation 38](#_Toc352958934)

[IV.1.1 Class Methods 38](#_Toc352958935)

[IV.2 Class Core 39](#_Toc352958936)

[IV.2.1 Class Methods 39](#_Toc352958937)

[IV.3 Class Court Location 39](#_Toc352958938)

[IV.3.1 Class Methods 40](#_Toc352958939)

[IV.4 Class Court 40](#_Toc352958940)

[IV.4.1 Class Methods 40](#_Toc352958941)

[IV.5 Class Data 40](#_Toc352958942)

[IV.5.1 Class Methods 41](#_Toc352958943)

[IV.6 Class Defendant 41](#_Toc352958944)

[IV.6.1 Class Methods 42](#_Toc352958945)

[IV.7 Class Guardian 42](#_Toc352958946)

[IV.7.1 Class Methods 42](#_Toc352958947)

[IV.8 Class Location 42](#_Toc352958948)

[IV.8.1 Class Methods 43](#_Toc352958949)

[IV.9 Class Program 43](#_Toc352958950)

[IV.9.1 Class Methods 43](#_Toc352958951)

[IV.10 Class School 44](#_Toc352958952)

[IV.10.1 Class Methods 44](#_Toc352958953)

[IV.11 Class Sentence 44](#_Toc352958954)

[IV.11.1 Class Methods 44](#_Toc352958955)

[IV.12 Class Trial 44](#_Toc352958956)

[IV.12.1 Class Methods 45](#_Toc352958957)

[IV.13 Class User 45](#_Toc352958958)

[IV.13.1 Class Methods 45](#_Toc352958959)

[IV.14 Class Volunteer 45](#_Toc352958960)

[IV.14.1 Class Methods 45](#_Toc352958961)

[IV.15 Class Workshop Location 45](#_Toc352958962)

[IV.15.1 Class Methods 45](#_Toc352958963)

[IV.16 Class Workshop 46](#_Toc352958964)

[IV.16.1 Class Methods 46](#_Toc352958965)

# 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.

## 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.

## 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.

### 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, it is recommended to obtain a security certificate/key to provide a Secure Socket Layer (SSL) for data encryption between the client and server. Also, various sensitive data will be encrypted in the database.

## 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.

## System Overview and Diagram

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 major components.



Figure : System 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>.

# Project Overview

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

## Team Members and Roles

Andrew Thompson will be the Team Leader and Lead Developer. Robert Reilly will be Lead Tester and Developer.

## 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.

## Phase Overview

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

### 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

* 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).
* NAYC: National Association of Youth Courts provides resources for over 1,050 youth and teen court programs throughout the country.
* 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.

### 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).

### Court 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 three options to expunge defendant information: 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 to track hours, status and volunteer history will be implemented. More information can be found in [4.4 Volunteer](#_Volunteer).

### Workshop

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

### Reports and Statistics

All necessary and legal documents will be generated for the defendants. Statistics reporting will include discoverable data, demographics data, and both local and regional statistics.

# 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.

## User Account and Access

This section describes the User Account and User Access development process.

### Technologies Used

PHP, MySQL, AJAX, JQuery and JQuery UI. Registration page uses an additional Captcha plugin.

### Component Overview

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

* ID of the Court Program they belong to
* Name (First and Last)
* Email Address (encrypted in the database)
* Password (encrypted in the database)
* Phone numbers
* Account Type
  + Application Administrator
  + Program Administrator
  + Program Manager
  + Program User

#### Database Tables Used

The following MySQL tables are used: user, user\_log, user\_phone, program

### Phase Overview

This phase will be the design and implementation of the different levels of users able to access the application. The tables for the database will be designed and the various web pages and scripting will be implemented.

### Architecture Overview

This phase will implement the *index.php* and *main.php* pages that will handle the user access and *registration.php* that will be used for new user account registration.

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 scripted hacking attempts.

The email and password will be protected by using the SHA-256 cryptographic hash function with a random salt, or additional string value. In addition to the encrypted password, there will also be a limited number of login attempts to prevent brute force access using automated scripts.

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

Figure 2 shows the architecture diagram of the registration process.



Figure : User Registration Architecture Diagram

#### User Access

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



Figure : User Access Architecture Diagram

### Design Details

These pages use CSS and JQuery UI elements, such as buttons and a custom theme, to provide unique styling.

## Court Program

### Technologies Used

### Component Overview

#### Database Table Schemas

### Phase Overview

### Architecture Overview

### Design Details

## 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 that the court administrator has access to.

### Technologies Used

PHP, MySQL, AJAX, JSON, JQuery.

### Component Overview

The following information is required for all Defendants:

* Primary Defendant Information
  + Name
    - First Name
    - Middle Initial
    - Last Name
  + Citation Number (from citation information)
  + Citation Date (from citation information)
  + Citation Time (from citation information)
  + Home phone number
  + Date of Birth
  + Court Case number
  + Agency Case number
  + Expunged date
  + Closed date
* Personal Information
  + Address
    - Physical Address
    - Mailing Address
  + School Data
    - School Name
    - Grade level
  + Physical Description
    - Height
    - Weight
    - Eye Color
    - Hair Color
    - Race
  + Driver's License Data
    - Number
    - Issue State
* Parental Information
  + Parental Relationship
  + Name
    - First Name
    - Middle Initial
    - Last Name
    - Suffix
  + Phone Number
    - Home Number
    - Work Number
  + Address
    - Physical Address
    - Mailing Address
  + Field to specify if living with that parent or not
* Citation Information
  + Offense Date
  + Offense Time
  + Day of Week
  + (Race/Age/Sex is auto filled from Personal Data)
  + Location
    - Address Number
    - City
    - State
    - Zip Code
    - Common Place
    - Cross Street
  + Citing Officer Data
    - Name
    - Identification Number
  + Mirandized checkbox
  + Offense Data
    - Statue
    - Title
    - Type
  + Vehicle Data
    - License Number
    - License State
    - Make
    - Type
  + Drugs or Alcohol Involved
  + Stolen/Vandalized Data
    - Description of items stolen or vandalized
    - Value or Amount
* Intake Information
  + Intake Date
  + Intake Time
  + Reschedule Date
  + Reschedule Time
  + Intake Interviewer
  + Referred to Juvenile – Not Qualified (Date)
  + Dismissed / No Complaint (Date)
* Court Information
  + Court Information
    - Court Date
    - Court Time
    - Court Type
    - Court Location
  + Judge
  + Court Officers
    - Defense Attorney
    - Prosecuting Attorney
    - Bailiff
    - Court Clerk
    - Exit Interviewer
  + Advisor (Multiple)
  + Jury (Multiple)
  + Parent Present (Multiple)

#### Database Table Schemas

Defendant, school, intake,

### Phase Overview

The initial layout will be developed during the design of the prototype. Actual implantation takes place during the Application Development phase.

### Architecture Overview

### Design Details

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

### Technologies Used

### Component Overview

#### Database Table Schemas

### Phase Overview

### Architecture Diagram

### Data Flow Diagram

### Design Details

## Courts

### Technologies Used

### Component Overview

#### Database Table Schemas

### Phase Overview

### Architecture Overview

### Design Details

## Workshops

### Technologies Used

### Component Overview

#### Database Table Schemas

### Phase Overview

### Architecture Overview

### Design Details

## Reports and Statistics

### Technologies Used

### Component Overview

#### Database Table Schemas

### Phase Overview

### Architecture Overview

### Design Details

## Surveys

### Technologies Used

### Database Table Schemas

### Component Overview

### Phase Overview

### Architecture Overview

### Design Details

## Program Administration Options

### Technologies Used

### Component Overview

#### Database Table Schemas

### Phase Overview

### Architecture Overview

### Design Details

## Application Administration Options

### Technologies Used

### Component Overview

#### Database Table Schemas

### Phase Overview

### Architecture Overview

### Design Details

## Miscellaneous

# System and Unit 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

## 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:

* Domain Name:
  + <http://teencourtdb.com>
* CPanel Access for hosting management:
  + URL: <http://teencourtdb.com/cpanel>
  + Username: teencour
  + Password: ifHQS14zvcN2
* FTP Access for file upload/download:
  + Username: teencour
  + Password: ifHQS14zvcN2
* 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.

# Release | Setup | Deployment

## Deployment Information and Dependencies

## Setup Information

## System Versioning Information

# End User Documentation

* + - 1. List of Figures

[Figure 1: System Diagram 8](#_Toc342611802)

[Figure 2: User Registration Architecture Diagram 14](#_Toc342611803)

[Figure 3: User Access Architecture Diagram 14](#_Toc342611804)

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

USE `teencour\_data` ;   
  
-- -----------------------------------------------------   
-- Table `teencour\_data`.`program`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`program` (   
    `programID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `name` VARCHAR(150) NOT NULL ,   
    `PAddress` VARCHAR(150) NULL ,   
    `pCity` VARCHAR(45) NULL ,   
    `pState` CHAR(2) NULL ,   
    `pZip` VARCHAR(25) NULL ,   
    `mAddress` VARCHAR(150) NULL ,   
    `mCity` VARCHAR(45) NULL ,   
    `mState` CHAR(2) NULL ,   
    `mZip` VARCHAR(25) NULL COMMENT 'Mailing address zip code' ,   
    `added` TIMESTAMP NOT NULL DEFAULT now() ,   
    PRIMARY KEY (`programID`) )   
ENGINE = InnoDB;

-- -----------------------------------------------------   
-- Table `teencour\_data`.`user`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`user` (   
    `userID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `programID` INT UNSIGNED NOT NULL ,   
    `type` INT UNSIGNED NOT NULL ,   
    `firstName` VARCHAR(75) NOT NULL ,   
    `lastName` VARCHAR(75) NOT NULL ,   
    `email` VARCHAR(255) NOT NULL ,   
    `password` VARCHAR(255) NOT NULL ,   
    `createDate` DATETIME NULL ,   
    `loginDate` TIMESTAMP NOT NULL DEFAULT now() ,   
    PRIMARY KEY (`userID`) ,   
    INDEX `programID` (`programID` ASC) ,   
    CONSTRAINT `fk\_user\_court`   
        FOREIGN KEY (`programID` )   
        REFERENCES `teencour\_data`.`program` (`programID` ) )   
ENGINE = InnoDB;

-- -----------------------------------------------------   
-- Table `teencour\_data`.`user\_phone`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`user\_phone` (   
    `phoneID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `userID` INT UNSIGNED NOT NULL ,   
    `phoneNum` VARCHAR(45) NOT NULL ,   
    `ext` VARCHAR(5) NULL ,   
    `type` VARCHAR(25) NULL ,   
    PRIMARY KEY (`phoneID`) ,   
    UNIQUE INDEX `phoneID\_UNIQUE` (`phoneID` ASC) ,   
    CONSTRAINT `fk\_user\_phone\_user1`   
        FOREIGN KEY (`userID` )   
        REFERENCES `teencour\_data`.`user` (`userID` ) )   
ENGINE = InnoDB;

-- -----------------------------------------------------   
-- Table `teencour\_data`.`school`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`school` (   
    `schoolID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `programID` INT UNSIGNED NOT NULL ,   
    `schoolName` VARCHAR(45) NULL ,   
    `address` VARCHAR(45) NULL ,   
    `city` VARCHAR(45) NULL ,   
    `state` VARCHAR(45) NULL ,   
    `zip` VARCHAR(45) NULL ,   
    `phone` VARCHAR(45) NULL ,   
    `contact` VARCHAR(45) NULL ,   
    `schoolcol` VARCHAR(45) NULL ,   
    PRIMARY KEY (`schoolID`, `programID`) ,   
    INDEX `fk\_school\_information\_court1\_idx` (`programID` ASC) ,   
    CONSTRAINT `fk\_school\_information\_court1`   
        FOREIGN KEY (`programID` )   
        REFERENCES `teencour\_data`.`program` (`programID` ) )   
ENGINE = InnoDB;

-- -----------------------------------------------------   
-- Table `teencour\_data`.`defendant`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`defendant` (   
    `defendantID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `programID` INT UNSIGNED NOT NULL ,   
    `firstName` VARCHAR(45) NOT NULL ,   
    `lastName` VARCHAR(45) NOT NULL ,   
    `middleName` VARCHAR(45) NULL ,   
    `homePhone` VARCHAR(45) NULL ,   
    `dob` INT NOT NULL ,   
    `courtCaseNumber` VARCHAR(45) NULL ,   
    `agencyCaseNumber` VARCHAR(45) NULL ,   
    `expungeDate` DATETIME NULL ,   
    `closeDate` DATETIME NULL ,   
    `schoolID` INT UNSIGNED NULL ,   
    `grade` VARCHAR(45) NULL ,   
    `height` VARCHAR(45) NULL ,   
    `weight` VARCHAR(45) NULL ,   
    `eyeColor` VARCHAR(45) NULL ,   
    `hairColor` VARCHAR(45) NULL ,   
    `sex` CHAR(1) NULL ,   
    `race` VARCHAR(45) NULL ,   
    `licenseNum` VARCHAR(45) NULL ,   
    `licenseState` VARCHAR(45) NULL ,   
    `added` TIMESTAMP NOT NULL DEFAULT now() ,   
    PRIMARY KEY (`defendantID`, `programID`) ,   
    INDEX `fk\_defendant\_court1\_idx` (`programID` ASC) ,   
    INDEX `fk\_defendant\_school1\_idx` (`schoolID` ASC) ,   
    CONSTRAINT `fk\_defendant\_court1`   
        FOREIGN KEY (`programID` )   
        REFERENCES `teencour\_data`.`program` (`programID` ) ,   
    CONSTRAINT `fk\_defendant\_school1`   
        FOREIGN KEY (`schoolID` )   
        REFERENCES `teencour\_data`.`school` (`schoolID` ) )   
ENGINE = InnoDB;

-- -----------------------------------------------------   
-- Table `teencour\_data`.`volunteer`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`volunteer` (   
    `volunteerID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `programID` INT UNSIGNED NOT NULL ,   
    `firstName` VARCHAR(45) NULL ,   
    `lastName` VARCHAR(45) NULL ,   
    `phone` VARCHAR(45) NULL ,   
    `email` VARCHAR(255) NULL ,   
    PRIMARY KEY (`volunteerID`, `programID`) ,   
    INDEX `fk\_volunteer\_court1\_idx` (`programID` ASC) ,   
    UNIQUE INDEX `volID\_UNIQUE` (`volunteerID` ASC) ,   
    CONSTRAINT `fk\_volunteer\_court1`   
        FOREIGN KEY (`programID` )   
        REFERENCES `teencour\_data`.`program` (`programID` ) )   
ENGINE = InnoDB;

-- -----------------------------------------------------   
-- Table `teencour\_data`.`court\_location`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`court\_location` (   
    `locationID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `programID` INT UNSIGNED NOT NULL ,   
    `name` VARCHAR(45) NULL ,   
    `address` VARCHAR(45) NULL ,   
    `city` VARCHAR(45) NULL ,   
    `state` CHAR(2) NULL ,   
    `zip` VARCHAR(45) NULL ,   
    PRIMARY KEY (`locationID`, `programID`) ,   
    INDEX `fk\_trial\_location\_program1\_idx` (`programID` ASC) ,   
    UNIQUE INDEX `locationID\_UNIQUE` (`locationID` ASC) ,   
    CONSTRAINT `fk\_trial\_location\_program1`   
        FOREIGN KEY (`programID` )   
        REFERENCES `teencour\_data`.`program` (`programID` ) )   
ENGINE = InnoDB; 

-- -----------------------------------------------------   
-- Table `teencour\_data`.`court`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`court` (   
    `courtID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `programID` INT UNSIGNED NOT NULL ,   
    `defendantID` INT UNSIGNED NOT NULL ,   
    `locationID` INT UNSIGNED NOT NULL ,   
    `type` VARCHAR(45) NOT NULL ,   
    `date` DATETIME NOT NULL ,   
    `closed` TINYINT(1) NULL ,   
    PRIMARY KEY (`courtID`, `programID`) ,   
    INDEX `fk\_jury\_court1\_idx` (`programID` ASC) ,   
    INDEX `fk\_jury\_defendant1\_idx` (`defendantID` ASC) ,   
    INDEX `fk\_trial\_trial\_location1\_idx` (`locationID` ASC) ,   
    CONSTRAINT `fk\_jury\_court1`   
        FOREIGN KEY (`programID` )   
        REFERENCES `teencour\_data`.`program` (`programID` ) ,   
    CONSTRAINT `fk\_jury\_defendant1`   
        FOREIGN KEY (`defendantID` )   
        REFERENCES `teencour\_data`.`defendant` (`defendantID` ) ,   
    CONSTRAINT `fk\_trial\_trial\_location1`   
        FOREIGN KEY (`locationID` )   
        REFERENCES `teencour\_data`.`court\_location` (`locationID` ) )   
ENGINE = InnoDB;

-- -----------------------------------------------------   
-- Table `teencour\_data`.`court\_position`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`court\_position` (   
    `positionID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `programID` INT UNSIGNED NOT NULL ,   
    `position` VARCHAR(45) NOT NULL ,   
    PRIMARY KEY (`positionID`) ,   
    CONSTRAINT `fk\_jury\_position\_court1`   
        FOREIGN KEY (`programID` )   
        REFERENCES `teencour\_data`.`program` (`programID` ) )   
ENGINE = InnoDB;

-- -----------------------------------------------------   
-- Table `teencour\_data`.`court\_member`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`court\_member` (   
    `courtID` INT UNSIGNED NOT NULL ,   
    `volunteerID` INT UNSIGNED NOT NULL ,   
    `positionID` INT UNSIGNED NOT NULL ,   
    `hours` DECIMAL(8,2) NULL ,   
    PRIMARY KEY (`courtID`, `positionID`, `volunteerID`) ,   
    INDEX `fk\_jury\_pool\_jury1\_idx` (`courtID` ASC) ,   
    INDEX `fk\_trial\_members\_trial\_position1\_idx` (`positionID` ASC) ,   
    CONSTRAINT `fk\_jury\_pool\_volunteer1`   
        FOREIGN KEY (`volunteerID` )   
        REFERENCES `teencour\_data`.`volunteer` (`volunteerID` ) ,   
    CONSTRAINT `fk\_jury\_pool\_jury1`   
        FOREIGN KEY (`courtID` )   
        REFERENCES `teencour\_data`.`court` (`courtID` ) ,   
    CONSTRAINT `fk\_trial\_members\_trial\_position1`   
        FOREIGN KEY (`positionID` )   
        REFERENCES `teencour\_data`.`court\_position` (`positionID` ) )   
ENGINE = InnoDB;   
  
-- -----------------------------------------------------   
-- Table `teencour\_data`.`user\_log`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`user\_log` (   
    `logID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `userID` INT UNSIGNED NOT NULL ,   
    `action` VARCHAR(45) NOT NULL ,   
    `ip\_address` VARCHAR(45) NOT NULL ,   
    `date` TIMESTAMP NOT NULL DEFAULT now() ,   
    PRIMARY KEY (`logID`) ,   
    UNIQUE INDEX `logID\_UNIQUE` (`logID` ASC) ,   
    CONSTRAINT `fk\_user\_log\_user1`   
        FOREIGN KEY (`userID` )   
        REFERENCES `teencour\_data`.`user` (`userID` ) )   
ENGINE = InnoDB; 

-- -----------------------------------------------------   
-- Table `teencour\_data`.`guardian`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`guardian` (   
    `guardianID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `defendantID` INT UNSIGNED NOT NULL ,   
    `firstName` VARCHAR(45) NULL ,   
    `lastName` VARCHAR(45) NULL ,   
    `address` VARCHAR(150) NULL ,   
    `city` VARCHAR(45) NULL ,   
    `state` VARCHAR(45) NULL ,   
    `zip` VARCHAR(45) NULL ,   
    `homePhone` VARCHAR(45) NULL ,   
    `emailAddress` VARCHAR(45) NULL ,   
    `employer` VARCHAR(45) NULL ,   
    `workPhone` VARCHAR(45) NULL ,   
    `livesWith` TINYINT(1) NULL ,   
    PRIMARY KEY (`guardianID`) ,   
    UNIQUE INDEX `parentID\_UNIQUE` (`guardianID` ASC) ,   
    INDEX `fk\_parents\_defendant1\_idx` (`defendantID` ASC) ,   
    CONSTRAINT `fk\_parents\_defendant1`   
        FOREIGN KEY (`defendantID` )   
        REFERENCES `teencour\_data`.`defendant` (`defendantID` ) )   
ENGINE = InnoDB;   
  
-- -----------------------------------------------------   
-- Table `teencour\_data`.`custom\_fields`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`custom\_fields` (   
    `fieldID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `programID` INT UNSIGNED NOT NULL ,   
    `fieldName` VARCHAR(45) NULL ,   
    `required` TINYINT(1) NULL ,   
    PRIMARY KEY (`fieldID`, `programID`) ,   
    UNIQUE INDEX `fieldID\_UNIQUE` (`fieldID` ASC) ,   
    INDEX `fk\_custom\_fields\_court1\_idx` (`programID` ASC) ,   
    CONSTRAINT `fk\_custom\_fields\_court1`   
        FOREIGN KEY (`programID` )   
        REFERENCES `teencour\_data`.`program` (`programID` ) )   
ENGINE = InnoDB;

-- -----------------------------------------------------   
-- Table `teencour\_data`.`custom\_data`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`custom\_data` (   
    `customID` INT UNSIGNED NOT NULL ,   
    `defendantID` INT UNSIGNED NOT NULL ,   
    `value` VARCHAR(45) NULL ,   
    PRIMARY KEY (`customID`) ,   
    INDEX `fk\_custom\_data\_custom\_fields1\_idx` (`customID` ASC) ,   
    CONSTRAINT `fk\_custom\_data\_custom\_fields1`   
        FOREIGN KEY (`customID` )   
        REFERENCES `teencour\_data`.`custom\_fields` (`fieldID` ) )   
ENGINE = InnoDB; 

-- -----------------------------------------------------   
-- Table `teencour\_data`.`survey`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`survey` (   
    `surveyID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `description` MEDIUMTEXT NULL ,   
    `created` TIMESTAMP NOT NULL DEFAULT now() ,   
    `sent` DATETIME NULL ,   
    PRIMARY KEY (`surveyID`) ,   
    UNIQUE INDEX `surveyID\_UNIQUE` (`surveyID` ASC) )   
ENGINE = InnoDB;   
  
-- -----------------------------------------------------   
-- Table `teencour\_data`.`survey\_questions`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`survey\_questions` (   
    `questionID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `surveyID` INT UNSIGNED NOT NULL ,   
    `question` VARCHAR(255) NULL ,   
    `type` INT UNSIGNED NULL ,   
    PRIMARY KEY (`questionID`, `surveyID`) ,   
    UNIQUE INDEX `questionID\_UNIQUE` (`questionID` ASC) ,   
    INDEX `fk\_survey\_questions\_survey1\_idx` (`surveyID` ASC) ,   
    CONSTRAINT `fk\_survey\_questions\_survey1`   
        FOREIGN KEY (`surveyID` )   
        REFERENCES `teencour\_data`.`survey` (`surveyID` ) )   
ENGINE = InnoDB;

-- -----------------------------------------------------   
-- Table `teencour\_data`.`survey\_answers`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`survey\_answers` (   
    `surveyID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `userID` INT UNSIGNED NOT NULL ,   
    `questionID` INT UNSIGNED NOT NULL ,   
    `answer` VARCHAR(45) NULL ,   
    PRIMARY KEY (`surveyID`, `userID`, `questionID`) ,   
    INDEX `fk\_survey\_answers\_survey1\_idx` (`surveyID` ASC) ,   
    INDEX `fk\_survey\_answers\_survey\_questions1\_idx` (`questionID` ASC) ,   
    INDEX `fk\_survey\_answers\_user1\_idx` (`userID` ASC) ,   
    CONSTRAINT `fk\_survey\_answers\_survey1`   
        FOREIGN KEY (`surveyID` )   
        REFERENCES `teencour\_data`.`survey` (`surveyID` ) ,   
    CONSTRAINT `fk\_survey\_answers\_survey\_questions1`   
        FOREIGN KEY (`questionID` )   
        REFERENCES `teencour\_data`.`survey\_questions` (`questionID` ) ,   
    CONSTRAINT `fk\_survey\_answers\_user1`   
        FOREIGN KEY (`userID` )   
        REFERENCES `teencour\_data`.`user` (`userID` ) )   
ENGINE = InnoDB; 

-- -----------------------------------------------------   
-- Table `teencour\_data`.`survey\_options`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`survey\_options` (   
    `optionID` VARCHAR(45) NOT NULL ,   
    `questionID` INT UNSIGNED NOT NULL ,   
    `option` VARCHAR(45) NULL ,   
    PRIMARY KEY (`optionID`) ,   
    INDEX `fk\_survey\_options\_survey\_questions1\_idx` (`questionID` ASC) ,   
    CONSTRAINT `fk\_survey\_options\_survey\_questions1`   
        FOREIGN KEY (`questionID` )   
        REFERENCES `teencour\_data`.`survey\_questions` (`questionID` ) )   
ENGINE = InnoDB;

-- -----------------------------------------------------   
-- Table `teencour\_data`.`court\_defendant\_jury`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`court\_defendant\_jury` (   
    `courtID` INT UNSIGNED NOT NULL ,   
    `defendantID` INT UNSIGNED NOT NULL ,   
    `positionID` INT UNSIGNED NOT NULL ,   
    `hours` DECIMAL(8,2) NULL ,   
    PRIMARY KEY (`courtID`, `defendantID`, `positionID`) ,   
    INDEX `fk\_jury\_member\_def\_trial1\_idx` (`courtID` ASC) ,   
    INDEX `fk\_jury\_member\_def\_defendant1\_idx` (`defendantID` ASC) ,   
    INDEX `fk\_court\_jury\_court\_position1\_idx` (`positionID` ASC) ,   
    CONSTRAINT `fk\_jury\_member\_def\_trial1`   
        FOREIGN KEY (`courtID` )   
        REFERENCES `teencour\_data`.`court` (`courtID` ) ,   
    CONSTRAINT `fk\_jury\_member\_def\_defendant1`   
        FOREIGN KEY (`defendantID` )   
        REFERENCES `teencour\_data`.`defendant` (`defendantID` ) ,   
    CONSTRAINT `fk\_court\_jury\_court\_position1`   
        FOREIGN KEY (`positionID` )   
        REFERENCES `teencour\_data`.`court\_position` (`positionID` ) )   
ENGINE = InnoDB;   
  
-- -----------------------------------------------------   
-- Table `teencour\_data`.`statute`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`statute` (   
    `statuteID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `programID` INT UNSIGNED NOT NULL ,   
    `code` VARCHAR(50) NULL ,   
    `title` VARCHAR(50) NULL ,   
    `description` TEXT NULL ,   
    PRIMARY KEY (`statuteID`, `programID`) ,   
    INDEX `fk\_citation\_court1\_idx` (`programID` ASC) ,   
    UNIQUE INDEX `statuteID\_UNIQUE` (`statuteID` ASC) ,   
    CONSTRAINT `fk\_citation\_court1`   
        FOREIGN KEY (`programID` )   
        REFERENCES `teencour\_data`.`program` (`programID` ) )   
ENGINE = InnoDB;

-- -----------------------------------------------------   
-- Table `teencour\_data`.`court\_common\_place`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`court\_common\_place` (   
    `commonPlaceID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `programID` INT UNSIGNED NOT NULL ,   
    `commonPlace` VARCHAR(45) NULL ,   
    `address` VARCHAR(45) NULL ,   
    `city` VARCHAR(45) NULL ,   
    `state` CHAR(2) NULL ,   
    `zip` VARCHAR(45) NULL ,   
    `closeTo` VARCHAR(45) NULL ,   
    PRIMARY KEY (`commonPlaceID`, `programID`) ,   
    INDEX `fk\_court\_common\_place\_court1\_idx` (`programID` ASC) ,   
    CONSTRAINT `fk\_court\_common\_place\_court1`   
        FOREIGN KEY (`programID` )   
        REFERENCES `teencour\_data`.`program` (`programID` ) )   
ENGINE = InnoDB;   
  
-- -----------------------------------------------------   
-- Table `teencour\_data`.`citation\_officer`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`citation\_officer` (   
    `officerID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `programID` INT UNSIGNED NOT NULL ,   
    `firstname` VARCHAR(45) NULL ,   
    `lastname` VARCHAR(50) NULL ,   
    `idNumber` VARCHAR(50) NOT NULL ,   
    `phone` VARCHAR(45) NULL ,   
    PRIMARY KEY (`officerID`, `programID`) ,   
    INDEX `fk\_citation\_officer\_court1\_idx` (`programID` ASC) ,   
    UNIQUE INDEX `officerID\_UNIQUE` (`officerID` ASC) ,   
    CONSTRAINT `fk\_citation\_officer\_court1`   
        FOREIGN KEY (`programID` )   
        REFERENCES `teencour\_data`.`program` (`programID` ) )   
ENGINE = InnoDB;

-- -----------------------------------------------------   
-- Table `teencour\_data`.`citation`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`citation` (   
    `citationID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `defendantID` INT UNSIGNED NOT NULL ,   
    `statueID` INT UNSIGNED NOT NULL ,   
    `officerID` INT UNSIGNED NOT NULL ,   
    `date` TIMESTAMP NULL ,   
    `address` VARCHAR(45) NULL ,   
    `city` VARCHAR(45) NULL ,   
    `state` CHAR(2) NULL ,   
    `zip` VARCHAR(45) NULL ,   
    `mirandized` TINYINT(1) NULL ,   
    `drugsOrAlcohol` TINYINT(1) NULL ,   
    `commonPlaceID` INT UNSIGNED NULL ,   
    `entered` TIMESTAMP NOT NULL DEFAULT now() ,   
    PRIMARY KEY (`citationID`, `defendantID`) ,   
    INDEX `fk\_citation\_court\_common\_place1\_idx` (`commonPlaceID` ASC) ,   
    INDEX `fk\_citation\_defendant1\_idx` (`defendantID` ASC) ,   
    INDEX `fk\_citation\_statute1\_idx` (`statueID` ASC) ,   
    INDEX `fk\_citation\_citation\_officer1\_idx` (`officerID` ASC) ,   
    UNIQUE INDEX `citationID\_UNIQUE` (`citationID` ASC) ,   
    CONSTRAINT `fk\_citation\_court\_common\_place1`   
        FOREIGN KEY (`commonPlaceID` )   
        REFERENCES `teencour\_data`.`court\_common\_place` (`commonPlaceID` ) ,   
    CONSTRAINT `fk\_citation\_defendant1`   
        FOREIGN KEY (`defendantID` )   
        REFERENCES `teencour\_data`.`defendant` (`defendantID` ) ,   
    CONSTRAINT `fk\_citation\_statute1`   
        FOREIGN KEY (`statueID` )   
        REFERENCES `teencour\_data`.`statute` (`statuteID` ) ,   
    CONSTRAINT `fk\_citation\_citation\_officer1`   
        FOREIGN KEY (`officerID` )   
        REFERENCES `teencour\_data`.`citation\_officer` (`officerID` ) )   
ENGINE = InnoDB;

-- -----------------------------------------------------   
-- Table `teencour\_data`.`citation\_vehicle`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`citation\_vehicle` (   
    `vehicleID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `citationID` INT UNSIGNED NOT NULL ,   
    `licenseNumber` VARCHAR(45) NULL ,   
    `licenseState` CHAR(2) NULL ,   
    `make` VARCHAR(45) NULL ,   
    `model` VARCHAR(45) NULL ,   
    `color` VARCHAR(45) NULL ,   
    PRIMARY KEY (`vehicleID`, `citationID`) ,   
    INDEX `fk\_citation\_vehicle\_citation1\_idx` (`citationID` ASC) ,   
    UNIQUE INDEX `vehicleID\_UNIQUE` (`vehicleID` ASC) ,   
    CONSTRAINT `fk\_citation\_vehicle\_citation1`   
        FOREIGN KEY (`citationID` )   
        REFERENCES `teencour\_data`.`citation` (`citationID` ) )   
ENGINE = InnoDB; 

-- -----------------------------------------------------   
-- Table `teencour\_data`.`citation\_stolen\_items`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`citation\_stolen\_items` (   
    `itemID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `citationID` INT UNSIGNED NOT NULL ,   
    `name` VARCHAR(45) NULL ,   
    `value` DECIMAL(18,2) NULL ,   
    PRIMARY KEY (`itemID`) ,   
    INDEX `fk\_citation\_stolen\_items\_citation1\_idx` (`citationID` ASC) ,   
    CONSTRAINT `fk\_citation\_stolen\_items\_citation1`   
        FOREIGN KEY (`citationID` )   
        REFERENCES `teencour\_data`.`citation` (`citationID` ) )   
ENGINE = InnoDB;

-- -----------------------------------------------------   
-- Table `teencour\_data`.`court\_guardian`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`court\_guardian` (   
    `courtID` INT UNSIGNED NOT NULL ,   
    `guardianID` INT UNSIGNED NOT NULL ,   
    PRIMARY KEY (`courtID`, `guardianID`) ,   
    INDEX `fk\_trial\_guardian\_guardian1\_idx` (`guardianID` ASC) ,   
    INDEX `fk\_trial\_guardian\_trial1\_idx` (`courtID` ASC) ,   
    CONSTRAINT `fk\_trial\_guardian\_guardian1`   
        FOREIGN KEY (`guardianID` )   
        REFERENCES `teencour\_data`.`guardian` (`guardianID` ),   
    CONSTRAINT `fk\_trial\_guardian\_trial1`   
        FOREIGN KEY (`courtID` )   
        REFERENCES `teencour\_data`.`court` (`courtID` ) )   
ENGINE = InnoDB;   
  
-- -----------------------------------------------------   
-- Table `teencour\_data`.`intake\_information`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`intake\_information` (   
    `defendantID` INT NOT NULL ,   
    `description` VARCHAR(50) NOT NULL ,   
    `date` TIMESTAMP NOT NULL ,   
    `inteviewer` VARCHAR(45) NULL ,   
    `referred` TINYINT(1) NULL ,   
    `dismissed` TINYINT(1) NULL ,   
    PRIMARY KEY (`defendantID`) )   
ENGINE = InnoDB; 

-- -----------------------------------------------------   
-- Table `teencour\_data`.`workshop`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`workshop` (   
    `workshopID` INT UNSIGNED NOT NULL AUTO\_INCREMENT ,   
    `programID` INT UNSIGNED NOT NULL ,   
    `date` DATETIME NULL ,   
    `title` VARCHAR(45) NULL ,   
    `description` TEXT NULL ,   
    `instructor` VARCHAR(45) NULL ,   
    `officerID` INT NULL ,   
    PRIMARY KEY (`workshopID`, `programID`) ,   
    INDEX `fk\_workshop\_program1\_idx` (`programID` ASC) ,   
    CONSTRAINT `fk\_workshop\_program1`   
        FOREIGN KEY (`programID` )   
        REFERENCES `teencour\_data`.`program` (`programID` ) )   
ENGINE = InnoDB;   
  
-- -----------------------------------------------------   
-- Table `teencour\_data`.`workshop\_roster`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`workshop\_roster` (   
    `workshopID` INT UNSIGNED NOT NULL ,   
    `defendantID` INT UNSIGNED NOT NULL ,   
    `completed` DATETIME NULL ,   
    INDEX `fk\_workshop\_roster\_workshop1\_idx` (`workshopID` ASC) ,   
    INDEX `fk\_workshop\_roster\_defendant1\_idx` (`defendantID` ASC) ,   
    PRIMARY KEY (`workshopID`, `defendantID`) ,   
    CONSTRAINT `fk\_workshop\_roster\_workshop1`   
        FOREIGN KEY (`workshopID` )   
        REFERENCES `teencour\_data`.`workshop` (`workshopID` ),   
    CONSTRAINT `fk\_workshop\_roster\_defendant1`   
        FOREIGN KEY (`defendantID` )   
        REFERENCES `teencour\_data`.`defendant` (`defendantID` ) )   
ENGINE = InnoDB;   
  
-- -----------------------------------------------------   
-- Table `teencour\_data`.`volunteer\_position`   
-- -----------------------------------------------------   
CREATE    TABLE IF NOT EXISTS `teencour\_data`.`volunteer\_position` (   
    `volunteerID` INT UNSIGNED NOT NULL ,   
    `positionID` INT UNSIGNED NOT NULL ,   
    PRIMARY KEY (`volunteerID`, `positionID`) ,   
    INDEX `fk\_volunteer\_position\_court\_position1\_idx` (`positionID` ASC) ,   
    INDEX `fk\_volunteer\_position\_volunteer1\_idx` (`volunteerID` ASC) ,   
    CONSTRAINT `fk\_volunteer\_position\_court\_position1`   
        FOREIGN KEY (`positionID` )   
        REFERENCES `teencour\_data`.`court\_position` (`positionID` ),   
    CONSTRAINT `fk\_volunteer\_position\_volunteer1`   
        FOREIGN KEY (`volunteerID` )   
        REFERENCES `teencour\_data`.`volunteer` (`volunteerID` ) )   
ENGINE = InnoDB;

* + - 1. Progress | 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 2 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. Class Documentation

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

* + - * 1. Class 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. Class 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. Class 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. Class 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. Class 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. Class 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. Class 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. Class 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. Class 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. Class 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. Class 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. Class 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. Class 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.