Developer Guide

V3.0

5/7/2015

JUDGE FROG

Revision History

Version	Changes	Edited
1.0	• Initial Draft	March 15, 2015
2.0	• Completed Arvixe, CakePHP sections • Added Model section	May 3, 2015
2.1	• Completed Controller overview • Added AppController, PagesController, AdminPanelController, CaseEditsController, UsersController sections	May 3, 2015
2.2	• Added MySQL section content	May 3, 2015
2.3	• Revised Introduction section • Revised System Overview section	May 3, 2015
2.4	• Finish controller sections	May 3, 2015
2.5	• Update Glossary	May 5, 2015
2.6	• Fixed Table of Contents • Fixed section headers	May 5, 2015
3.0	• Fixed Table of Contents • Fixed page numbering	May 7, 2015

Revision Sign-Off

By signing the following, the team member is stating that he has read the entire document and has verified that the information contained within this document is accurate, relevant to the project, and void of errors.

Name	Signature	Date Signed
Brice Boula		
Collin Duncan		
David Tomlinson		
Landon Westrom		

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to provide you with the tools necessary to assume and continue development of the Human Trafficking Data (HTD) project. This document contains a detailed breakdown of the structure and interactions of our project. For more specific information on code, please check the source code itself for comments.

1.2 Overview

This document includes the following sections.

Section 2: System Overview – This section describes a brief overview of all system components.

Section 3: Arvixe – This section contains detailed information about the hosting setup for Human Trafficking Data.

Section 4: MySQL Database – This section contains a detailed description of the database schema and configuration.

Section 5: CakePHP – This section contains a detailed description of the PHP framework used for the project.

Section 6: Models – This section contains information about the data models used in the HTD application.

Section 7: Controllers – This section contains a detailed breakdown of all controllers in the HTD project, their functions, interactions, and views.

Section 8 - Glossary of Terms: Includes a list of abbreviations and their technical terms and their associated definitions.

2. System Overview

2.1 System Components

Judge Frog is comprised of a CakePHP web application, a MySQL database, and a web host.

2.1.1 CakePHP Web Application

The web application is written primarily in PHP, using a framework called CakePHP. This framework provides excellent functionality when interacting with a MySQL database.

2.1.2 MySQL Database

The MySQL database contains all the case information and user login information. The database is stored on the web host's server.

2.1.3 Web Host

The web host the project relies on is Arvixe. Three domains are purchased from Arvixe including; humantraffickingdata.org, humantraffickingdata.com, humantraffickingdata.net. The web application and database are stored on a Linux server using Apache for the web server.

3. Arvixe

3.1 Overview

The Judge Frog project contracted with a web host called Arvixe (http://www.arvixe.com/) to provide an Apache server for the project. Please see Dr. Bouche for the credentials necessary to access Arvixe. Three domains are registered with Arvixe: www.humantraffickingdata.org, www.humantraffickingdata.com, and www.humantraffickingdata.net.

3.2 Oyster

3.2.1 cPanel & Apache

The Human Trafficking Data website is hosted on an Apache server named Oyster. To access the server, the URL is www.oyster.arvixe.com. Use the cPanel login option and enter the admin credentials given to you by Dr. Bouche. As of the writing of this document, the Apache version running is **2.2.27**.

3.2.2 PHP

The PHP version running on the server is listed as **5.3.28** on the cPanel stats menu, but you can choose to run a different version of PHP for a given directory on the server. We have done so, and the PHP version running on HTD is **5.5**. To customize the PHP version, use the **PHP Selector** utility in the Software/Services section of the cPanel.

3.2.3 MySQL

The current MySQL version as of this writing is **5.5.42-37.1**. There are several utilities for administrating the MySQL database on Oyster. First, to modify, check, or repair a database, use the **MySQL Databases** tool in the Databases section. A list of current databases is displayed. Here you can also control the accounts that our application uses in order to access the database. For interfacing with the database directly, **phpMyAdmin** is available.

3.2.4 FTP

For managing files on the server, a standard FTP utility is available in the Files section, called **File Manager**. Here you will see a listing of all directories available to us on Oyster. All files related to HTD are stored in the **public_html** directory.

3.2.5 Subdomain Management

For management of subdomains on Oyster, there is a utility called **Subdomains** in the Domains section. Here you can set custom subdomains and direct them to look at specific directories.

4. MySQL Database

4.1 Overview

HTD utilizes a **MySQL v5.5.42-37**.1 database to store all of the records regarding case and user information. There are 3 individual tables which hold different records; the first is the **User** table, the second being the **DataInProgress** table, and the third being the **Data** table. The **User** table holds all of the information necessary for user accounts and permissions. The **DataInProgress** table contains all of the information regarding cases which are currently in progress. In progress refers to cases which have been flagged to be reviewed by administrators. This flagging is done when: a case gets submitted for review from case creation or a case gets edited. The **Data** table contains the same exact information as **DataInProgress** but holds information in a different state to separate finalized from in-progress data. These variables will be explained more in **Section 4.4** of this document. The application we used to administer and model the database was **MySQL Workbench** and a **MySQL Workbench** file containing a model of the database will be included in our project directory.

4.2 MySQL Workbench

MySQL Workbench is a utility designed for DBAs and developers and allows them to design and model their database, develop SQL, administrate the database, and perform database migration. For our development process, we chose the most up-to-date version of **MySQL Workbench** which is **v6.3.3**.

4.2.1 Installation

To install **MySQL Workbench** on your machine, please follow the instructions listed at http://dev.mysql.com/downloads/workbench/.

4.2.2 Useful References

MySQL provides very extensive documentation on how to use **MySQL Workbench**. If there are any questions on how **MySQL Workbench** performs please visit their documentation at the following URL: http://dev.mysql.com/doc/workbench/en/index.html.

4.3 Accessing the Database

To access the database we used for this project there are 2 methods, one for accessing the database through **Arvixe's cPanel** and one using **MySQL Workbench**. Following these two methods there will be general information that will allow the developer to view the settings usually necessary to connect to a **MySQL database**.

4.3.1 Arvixe

One can access the database by using **Arvixe's** pre-installed **cPanel** and navigating to the utilities which connect to the database. Specifically, the modules listed in **Section 3.2.3** of this document.

4.3.2 MySQL Workbench

To configure **MySQL Workbench** to remotely connect to the **MySQL database**, we must complete a few simple steps.

1. In the toolbar navigate to: Database → Manage Connections. A popup with **MySQL Connections** on the left-hand side should open.
2. Click the **'New' button** in the bottom left-hand corner.
3. Rename the **Connection Name** to a meaningful name for the project.
4. Under the **'Connection'** tab
    a. Set the **Hostname** to **'oyster.arvixe.com'**.
    b. Set the **Port** to **3306**.
    c. Set the **Username** to the username given by the project administrator.
    d. Set the **Password** to the password given by the project administrator.
5. **All other fields** should remain at **default** values.
6. Hit the **'Test Connection' button** to verify the connection is working.

If the connection test returned positive, then you have successfully connected to the **Arvixe database**. If they have not, please refer to **Section 4.2.2** to find **references** which will aid you in solving the issue.

4.3.3 Other

This information is intended to be used if the developer wishes to connect to the **MySQL database** hosted by **Arvixe**.

- **Hostname:** oyster.arvixe.com
- **Port:** 3306
- **Username:** Contact project administrator
- **Password:** Contact project administrator

4.4 Database Schema

The **schema** for the **MySQL database** used in the current version of this project will be represented below in a table showing the: **name**, **type**, **attributes**, **default**, and **description** of each **column** for each **table** in the **database**. The **attributes** column will contain abbreviations which correspond to **MySQL Workbench** abbreviations: **PK** is **primary key**; **NN** is **not null**, **UQ** is **unique**, **BIN** is **binary**, **UN** is **unsigned**, **ZF** is **zero-fill**, **AI** is **auto-increment** (http://dev.mysql.com/doc/workbench/en/wb-table-editor-columns-tab.html).

Users table:

Name	Type	Attributes	Default	Description
Id	INTEGER	PK, NN, UQ, AI		ID given to each user to uniquely identify them.
Username	VARCHAR(45)	NN, UQ		Username for the given user.
Email	VARCHAR(45)	NN		Email address for the given user.
Role	VARCHAR(16)	NN	‘ra’	The role of the given user. The value ‘admin’ gives the user administrative permissions and any other value gives them research assistant permissions.
Password	VARCHAR(255)	NN		Salted and hashed password for the given user.
Created	DATETIME			Timestamp of when the user was created.
Modified	DATETIME		Null	Timestamp of when the user was modified.
First_name	VARCHAR(45)			First name of the user.
Last_name	VARCHAR(45)			Last name of the user.

Data table:

The value '[statute]' in the **Data** table represents a value for one of the recognized statutes within our application. These statutes are: 1961 – 1968, 1028, 1351, 1425, 1512, 1546, 1581 – 1588, 1589, 1590, 1591, 1592, 2252, 2260, 2421 – 2424, 1324, and 1328. They are also listed in more detail on our website: http://humantraffickingdata.org/description (under Statutes tab).

Name	Type	Attributes	Description
Id	INTEGER	PK, NN, AI	ID given to each record to uniquely identify it.
Author	VARCHAR(45)	NN	User who published this record.
Created	DATETIME		Timestamp of when the record was published.
Modified	DATETIME		Timestamp of when the record was last modified.
CaseDefId	INTEGER		
CaseNam	VARCHAR(45)		The official name of the case.
CaseNum	VARCHAR(45)		The official number of the case.
NumDef	INTEGER		The total number of defendants in this case.
State	VARCHAR(2)		The state in which the case was held.
FedDistrictLoc	VARCHAR(45)		The location within the federal district (North, Central, South, etc.)
FedDistrictNum	INTEGER		The number of the federal district where this case was located (1-13).
JudgeName	VARCHAR(45)		The name of the last judge involved in the case.
JudgeRace	INTEGER		The race of the last judge involved in the case. The values are as defined: 0 is white; 1 is black; 2 is Hispanic; 3 is Asian; 4 is Indian.
JudgeGen	BOOLEAN		The gender of the last judge involved in the case. True corresponds with female and false corresponds with male.
JudgeTenure	YEAR		The tenure of the last judge involved in the case.
JudgeApptBy	BOOLEAN		The political party who appointed the judge. True corresponds with Democrat and false corresponds with Republican.
CaseSummary	LONGTEXT		A summary of the case.
DefLast	VARCHAR(45)	NN	The last name of this defendant.

DefFirst	VARCHAR(45)	NN	The first name of this defendant.
Alias	TEXT		A list of aliases of the defendant separated by ';'.
DefGender	BOOLEAN		The gender of the defender. True designates female and false designates male.
DefRace	INTEGER		The race of the defendant. 0 represents white; 1 represents black; 2 represents Hispanic; 3 represents Asian; 4 represents other.
DefBirthdate	YEAR	NN	The year of the defendant's birth.
DefArrestAge	INTEGER		The age at arrest of the defendant.
ChargeDate	DATE		The date the defendant was charged.
ArrestDate	DATE		The date the defendant was arrested.
Detained	BOOLEAN		Value representing whether defendant was detained.
BailType	INTEGER		The type of bail given to defendant. 0 represents none, 1 represents surety, 2 represents non-surety.
BailAmount	DOUBLE		The amount of bail to be paid by the defendant.
LaborTraf	BOOLEAN		Value representing whether this case contained labor trafficking charges.
AdultSexTraf	BOOLEAN		Value representing whether this case contained adult sex trafficking charges.
MinorSexTraf	BOOLEAN		Value representing whether this case contained minor sex trafficking charges.
FelCharged	INTEGER		The total number of felonies charged against the defendant.
FelSentenced	INTEGER		The total number of felonies sentenced against the defendants.
S[statute]	BOOLEAN		Value representing whether this particular statute, [statute], was charged in the given case for this defendant.
Counts[statute]	INTEGER		Number of counts charged against defendant for this particular statute.
CountsNP[statute]	INTEGER		Number of counts nolle prossed for this defendant for this particular statute.
PleaDismissed[statute]	INTEGER		Number of pleas dismissed for this defendant for this particular statute.

PleaGuilty[statute]	INTEGER		Number of times plead guilty for this defendant for this particular statute.
TrialGuilty[statute]	INTEGER		Number of trials found guilty for this defendant for this particular statute.
TrialNG[statute]	INTEGER		Number of trials found not guilty for this defendant for this particular statute.
Fines[statute]	INTEGER		Amount of fines levied against defendant for this particular statute.
Sent[statute]	INTEGER		Number of months sentenced against this defendant for this particular statute.
Prob[statute]	INTEGER		Number of months of probation required for this defendant for this particular statute.
DateTerm	DATE		The date when the sentence was terminated.
SentDate	DATE		The date when the sentencing began.
TotalSentence	INTEGER		The total number of months sentenced for this defendant.
Restitution	DOUBLE		The amount of restitution required to pay by the defendant.
AssetForfeit	BOOLEAN		Value representing whether the defendant had to forfeit various assets.
SupRelease	BOOLEAN		Value representing whether the defendant had supervised release.
Probation	INTEGER		The total number of months of probation required by this defendant.
NumVic	INTEGER		Total number of victims involved in this case.
NumVicMinor	INTEGER		Total number of minor victims involved in this case.
NumVicForeign	INTEGER		Total number of foreign victims involved in this case.
NumVicFemale	INTEGER		Total number of female victims involved in this case.
OCName1	VARCHAR(45)		The name of the first organized crime group that this defendant was involved in.
OCType1	INTEGER		The size of the first organized crime group that this defendant was involved in. 1 represents ‘Mom & Pop’, 2 represents ‘Street Gang’, 3 represents ‘Cartel/Syndicate/Mafia’,

			5 represents ‘Prison Gang’, 6 represents ‘Other’.
OCRace1	INTEGER		The race of the first organized crime group that this defendant was involved in. 0 represents ‘None’, 1 represents ‘Black’, 2 represents ‘White’, 3 represents ‘Hispanic’, 4 represents ‘Asian’, 5 represents ‘Other’.
OCScope1	INTEGER		The scope of the first organized crime group that this defendant was involved in. 0 represents ‘Other’, 1 represents ‘Local Only’, 2 represents ‘Trans-State’, 3 represents ‘Trans-National’.
OCName2	VARCHAR(45)		The size of the second organized crime group that this defendant was involved in. 1 represents ‘Mom & Pop’, 2 represents ‘Street Gang’, 3 represents ‘Cartel/Syndicate/Mafia’, 5 represents ‘Prison Gang’, 6 represents ‘Other’.
OCType2	INTEGER		The race of the second organized crime group that this defendant was involved in. 0 represents ‘None’, 1 represents ‘Black’, 2 represents ‘White’, 3 represents ‘Hispanic’, 4 represents ‘Asian’, 5 represents ‘Other’.
OCRace2	INTEGER		The scope of the second organized crime group that this defendant was involved in. 0 represents ‘Other’, 1 represents ‘Local Only’, 2 represents ‘Trans-State’, 3 represents ‘Trans-National’.
OCScope2	INTEGER		The size of the second organized crime group that this defendant was involved in. 1 represents ‘Mom & Pop’, 2 represents ‘Street Gang’, 3 represents ‘Cartel/Syndicate/Mafia’, 5 represents ‘Prison Gang’, 6 represents ‘Other’.

DataInProgress table:

The **DataInProgress** table features exactly the same information as the **Data** table. However, our client wanted to separate the data into two tables with one being for finalized information and one being for information still in progress. The only additional field in **DataInProgress** is:

Name	Type	Description
SubmittedForReview	BOOLEAN	If true, this case will be displayed for administrators to publish to **Data**. If false, the case is still being revised.

4.5 Case Data Model

The following is a JSON representation of how the case data is structured in various parts of the application to provide some amount of structure to the denormalized case information. Controllers that utilize **$case_info** or something similar will use a structure like this.

```
Case = {
        CaseName,
        CaseNumber,
        ChargeDate,
        LaborTraf,
        AdultSexTraf,
        MinorSexTraf,
        NumDef,
        State,
        FedDistrictLoc,
        FedDistrictNum,
        CaseSummary,
        JudgeName,
        JudgeRace,
        JudgeGen,
        JudgeTenure,
        JudgeApptBy
        NumVic,
        NumVicMinor,
        NumVicForeign,
        NumVicFemale,
        Defendants : [
                {
                        DefLast,
                        DefFirst,
                        Alias,
                        Gender,
                        Race,
                        Birthdate,
                        ArrestAge,
```

```
                    Detained,
                    BailType,
                    BailAmount,
                    FelCharged,
                    FelSentenced,
                    DateTerm,
                    SentDate,
                    TotalSentence,
                    Restitution,
                    AssetForfeit,
                    SupRelease,
                    Probation,
                    Charges: [
                        {
                            Statute,
                            Counts,
                            CountsNP,
                            PleaDismissed,
                            PleaGuilty,
                            TrialGuilty,
                            TrialNG,
                            Fines,
                            Sent,
                            Prob
                        }
                    ],
                    OCName1,
                    OCType1,
                    OCRace1,
                    OCScope1,
                    OCName2,
                    OCType2,
                    OCRace2,
                    OCScope2
                }
            ]
};
```

5. CakePHP

5.1 Overview

HTD is written in primarily PHP, using a framework called CakePHP (www.cakephp.org). The CakePHP version used in developing HTD was 2.6.1, due to limitations at the time. CakePHP is a Model-View-Controller (MVC) based framework. Documentation for CakePHP is available in the form of the CakePHP Cookbook. The Cookbook contains a detailed description of coding conventions and the structure of MVC interactions in CakePHP.

5.2 Configuring CakePHP

Configuring CakePHP is relatively straight-forward. To download a clean version of CakePHP, go to the GitHub repository of CakePHP, available at www.github.com/cakephp/cakephp/tags. Every version of CakePHP is available via the release tags on this page, so choose the 2.6.1 tag. You can download an archive of the release in various forms (zip/tar.gz). For development, you can simply add this CakePHP application directory to your own Apache server config. To see more detailed setup and configuration instructions, see the Installation page in the Cookbook.

There are several specific files that must be setup properly, conveniently located in the Config directory in the CakePHP application. The **database.php** file contains the configuration for the database that CakePHP will use. The **core.php** file contains core configurations for the CakePHP application, including the security salts and seeds used later in password hashing. This file also contains the ability to set the debug level of the application; the various levels are noted in the code comments. Here also the **routes.php** file contains routing information for accessing controller methods, etc. This is where most of the URL routing for the HTD application is configured.

Our CakePHP application **requires** PHP 5.5 or higher, due to an error in the implementation of Blowfish/bcrypt in lower versions of PHP. Hashes created with lower versions of PHP will result in errors when trying to log a user in.

5.3 Useful References

The following links are a few of the more important articles available in the CakePHP Cookbook. **This is not a substitute for reading the Cookbook on your own!**

CakePHP Cookbook: http://book.cakephp.org/2.0/en/index.html
Installation Instructions: http://book.cakephp.org/2.0/en/installation.html
Conventions: http://book.cakephp.org/2.0/en/getting-started/cakephp-conventions.html
CakePHP Structure: http://book.cakephp.org/2.0/en/getting-started/cakephp-structure.html
CakePHP Folder Structure: http://book.cakephp.org/2.0/en/getting-started/cakephp-folder-structure.html
CakePHP Models: http://book.cakephp.org/2.0/en/models.html

CakePHP Controllers: http://book.cakephp.org/2.0/en/controllers.html
CakePHP Views: http://book.cakephp.org/2.0/en/views.html

6. Models

6.1 Overview

CakePHP models are the primary method for interacting with the database. By default, CakePHP models have several functions for retrieving data from and saving data to the database. When interacting with models, please be careful to follow CakePHP conventions; a lot of functionality depends on these conventions. **If you choose not to follow conventions, much of CakePHP's automagic will fail.** For a detailed list of these functions and their parameters, please see the following links:

Retrieving Your Data: http://book.cakephp.org/2.0/en/models/retrieving-your-data.html
Saving Your Data: http://book.cakephp.org/2.0/en/models/saving-your-data.html
Validating Your Data: http://book.cakephp.org/2.0/en/models/data-validation.html

We have set our own rules for data validation within our models. Validation must be successful when data is saved to the database; if the validation fails, no data will be saved to the database by default. Validation errors can be accessed by calling the **$this->ModelName->validationErrors** array.

6.2 Datum

The **Datum** model is associated with the **Data** table in the database schema. This is the model used by controllers which interact with the **Data** table. The **$validate** array contains the validation rules associated with the **Datum** model. Each rule in the **$validate** array is associated with a column in the **Data** table. Validation rules can be complex or relatively simple; we have chosen to use relatively simple validation rules. For more information on validation rules, please see the **Validating Your Data** link given above.

6.3 DataInProgress

The **DataInProgress** model is associated with the **DataInProgress** table in the database schema. This is the model most heavily used by the controllers which interact with the DataInProgress table, usually related to editing case information. The **$validate** array contains the validation rules associated with the **DataInProgress** model. Each rule is associated with a column in the **DataInProgress** table. For more information on validation rules, please see the **Validating Your Data** link given above.

6.4 User

The **User** model is associated with the **users** table in the database schema. This model is used by the **UsersController** when interacting with user logins, logouts, creation, and deletion.

7. Controllers

7.1 Overview

CakePHP controllers contain much of the logic required for processing user requests and rendering views to a user's browser. We have several controllers, one for each major section of the application. Each controller has several views associated with it in order to display the information handled by that controller.

7.2 AppController

The **AppController** is the parent controller to all other controllers. In the **AppController**, we can create methods, attributes, and components that will be available in all other controllers. Note that even if a controller has no interactions in the Interactions section, it still interacts with the AppController because of inheritance. The interaction is instead implied.

7.2.1 Interactions

This controller contains attributes and functions inherited by all other controllers.

7.2.2 Views Associated

No views are associated with the AppController.

7.2.3 Functions

beforeFilter() – This method currently specifies three actions which do not require authentication in order to access. For more information on the **beforeFilter()** function, please see the associated CakePHP documentation.

isAuthorized($user) – This method is the parent method for all user authorization requests. The method checks the passed user's permissions.

7.3 PagesController

The **PagesController** serves the views not specifically associated with other controllers. These are simply a few of the pages on the front end of HTD. This is a default controller supplied with CakePHP. Normally it dynamically uses the current URL to redirect the request to the proper controller, but our application instead relies heavily on the **routes.php** file for proper routing, thereby bypassing the PagesController generally.

7.3.1 Interactions

No interactions with other controllers.

7.3.2 Views Associated

About.ctp
Contact.ctp

Description.ctp
Home.ctp

7.3.3 Functions

display() – This method is a default CakePHP method for redirecting requests to generate the associated view.

home() – This method is called when a request for the HTD homepage is received.

about() – This method is called when a request for the About page is received.

description() – This method is called when a request for the Description page is received.

contact() – This method is called when a request for the Contact page is received.

7.4 AdminPanelController

The **AdminPanelController** serves views for the admin panel not associated with other controllers. Much of this controller's former functionality was moved to other controllers as development proceeded.

7.4.1 Interactions

No interactions with other controllers.

7.4.2 Views Associated

Index.ctp

7.4.3 Functions

isAuthorized($user) – This method allows user authorization to the index method in the controller. The method checks the passed user's permissions.

index() – This method renders the admin panel index view.

7.5 UploadsController

The **UploadsController** handles the **batch upload** functionality. This is the only thing that this controller does. To do this, the **UploadsController** uses the **Datum** model and all of its associated validation rules.

7.5.1 Interactions

No interactions with other controllers.

7.5.2 Views Associated

No views associated with this controller.

7.5.3 Functions

Add() – this function begins by checking to make sure a post request has been made and a file chosen. After this, it stores a copy of the uploaded file **./webroot/files/** directory with its specified name. Then the file is read in line by line until the entire contents of the file have been parsed. The data is then saved using the **Datum** model with **validation** required on all fields and all rows, else the insert fails. After the method succeeds or fails, a receipt is returned specifying how many records got inserted or where the errors in the .CSV are found.

7.6 DownloadController

The **DownloadController** contains the functionality to download a .CSV file of the **Data** table in the same format as it was uploaded. To do this, **DownloadController** uses the **Datum** model.

7.6.1 Interactions

No interactions with other controllers.

7.6.2 Views Associated

No views associated with this controller.

7.6.3 Functions

Download() – this function gathers all of the information currently inside the **Data** table of the database and parses through the data to create a .CSV matching the format and specifications of our client's demands.

7.7 CaseEditsController

The **CaseEditsController** contains the functionality necessary for editing case information present in the database. This is one of the major controllers. There are five primary views associated with the controller, though the add and edit views are largely imitations of each other, with the edit views having an added value to every form input in order to populate existing data. Changes made to the structure of an add view should probably be mirrored to the corresponding edit view.

7.7.1 Interactions

This controller has no direct interactions with other controllers. However, data operated on by this controller is used in the **CaseReviewsController**.

7.7.2 Views Associated

Add_case.ctp
Add_defendant.ctp
Edit.ctp
Edit_defendant.ctp
Index.ctp

7.7.3 Functions

isAuthorized($user) – This method checks user authorization for the controller's other methods. The method checks the passed user's permissions. Methods allowed for RAs to access are specified here; all other methods are thrown back to the parent isAuthorized() method in the AppController. This means methods not specified here require administrator permissions.

index() – This method gives the default view for the case edit/creation screen. It also retrieves all cases in the **Data** and **DataInProgress** tables and passes them to the **index.ctp** view for display.

edit($num) – This method renders the **edit.ctp** view. It takes the case number passed to it and retrieves all information related to that case. When a POST request is received from the **edit.ctp** view, data relevant to the case only (not defendant-specific information) is extracted and validated. This data is then validated and saved to the **DataInProgress** table on successful validation.

editDefendant($defArray) – This method renders the **edit_defendant.ctp** view. It takes the defendant name and case number passed to it, and renders a view with defendant-specific information based on the defendant name and case number. When a POST request is received from the **edit.ctp** view, the defendant's updated information is then saved to the **DataInProgress** table on successful validation.

addCase() – This method renders the **add_case.ctp** view. It allows the creation of a new case to be inserted into the database. When a POST request is received, the new case is added to the **DataInProgress** table and redirects the user to the **edit.ctp** view with the new case's information.

addDefendant($caseNumber) – This method adds a defendant to a case based on the case number passed to the method. The method pulls case-specific information from other rows in the case and adds it to the submitted defendant-specific information when a POST request is received.

migrateFromDataToDataInProgress($num) – This method behaves exactly as the name suggests: it moves an entire case from the **Data** table to the **DataInProgress** table. This method is only called when a case in the **Data** table needs to be edited. This method does not have a view associated with it.

getCase($caseNumber) – This method retrieves the case information for a case based on the case number passed to the method. Used in other functions. This method does not have a view associated with it.

getRowId() – This method retrieves the current max ID primary key in both the **Data** table and the **DataInProgress table**. This is necessary for smooth migration between the tables.

deleteEmptyCases($caseNum) – This method deletes rows in the database where the DefFirst and DefLast columns are null, and the CaseNum column matches the $caseNum value passed to the method. This is necessary due to the way adding and editing a defendant within a case behaves.

delete_case($num) – This method deletes all rows in the **Data** table associated with the case number passed to the method.

delete_incomplete_case($num) – This method deletes all rows in the **DataInProgress** table associated with the case number passed to the method.

delete_def($defArray) – This method deletes a defendant from a specific case based on the DefLast, DefFirst, and CaseNum values passed in the array.

7.8 CaseReviewsController

7.8.1 Interactions

This controller has no interactions with other controllers. However, data submitted from **CaseEditsController** is utilized in this controller and data published from this controller is used in **SearchController**.

7.8.2 Views Associated

Review.ctp – this view contains a dynamic DataTable with information regarding cases that are in need of administrator review.

7.8.3 Functions

isAuthorized($user) – checked to make sure that the role of the user accessing these functions is an **administrator** only.

Review() – the review function generates the variables needed in the view and Session for the other functions and views to behave properly. The variables it sets are **$p_cases** and **$case_info**. **$p_cases** contains an array of the information that will be displayed in the DataTable in **review.ctp**. **$case_info** is an array of objects which contains all of the information for each of the cases in the same order as **$p_cases**. The structure of the object is documented at… idk.

GenerateTable($index) – generateTable is performed whenever an individual case is clicked in the DataTable in the view, **review.ctp**. **$index** is the index within the DataTable of the case that got clicked and is used to access all of the case information found in **$case_info[$index]**. This function then prints a section of **HTML code** which creates the dynamic view of all of the case information at the right-hand side of the view.

PublishCase($index) – this function publishes the case at the given **$index** to the **Data** table and removes the instances from the **DataInProgress** table.

Delete_case($CaseNum) – this deletes all instances of cases with the given **$CaseNum**.

7.9 UsersController

The **UsersController** handles all requests related to the user login and management system.

7.9.1 Interactions

No interactions with other controllers.

7.9.2 Views Associated

Create.ctp
Login.ctp

7.9.3 Functions

isAuthorized($user) – This method allows access to the login and logout methods, but restricts all others to administrator level.

login() – This method attempts to log a user in with the username and password that it receives via POST and CakePHP automagic.

logout() – This method logs the current user out.

create() – This method creates a user using data submitted via POST.

delete_user() – This method deletes a user from the **users** table.

7.10 SearchController

7.10.1 Interactions

This controller does not interact with other controllers. However, data stored in user's browser session by this controller is used in **AnalyzeController**.

7.10.2 Views Associated

Home.ctp – this view contains a large **form element** along with a **Google Table Chart** and a **modal** to search and display all relevant information obtained from the functions of **SearchController**.

7.10.3 Functions

Home() – this function is called whenever the **SearchController** is accessed and controls which information is displayed and whether an **update()** to the table should be performed.

Update() – this function takes in all of the **search parameters** listed in the form element and parses the **parameters** to ensure no erroneous searches are made. These **parameters** are fed into a **$conditions** array which is used to specify the **WHERE** clause for **CakePHP's MySQL**

query using the **Datum** model. **Update()** also performs a **filter** which corresponds to what is displayed by default in the **modal** in **home.ctp**. After the filter, an object is created to store all of the case information, the previous search information, and the different categories searched by.

AutoComplete() – this is used for **Case Name, Judge Name,** and **Defendant Name** and searches through the **Data** table and returns back any instance which contains the **substring** in whichever field is being focused.

7.11 AnalyzeController

7.11.1 Interactions

This controller does not interact with other controllers. However, it directly uses Session data created by **SearchController**.

7.11.2 Views Associated

Index.ctp – this view contains a few small forms and one list which allows the user to choose the type of graph to generate along with the previous search criteria which provided them with the result set that they will be generating graphs on. This view also utilizes **Google Chart** libraries and performs an **AJAX** call on each request.

7.11.3 Functions

Index() – all this function accomplishes is reading in **$case_info** and **$query** which contain the **case information** and the **search parameters** received from **SearchController**.

GenerateGraph($type, $yIndex, $xIndex) – this function determines which type of graph needs to be created and sets up variables to perform the necessary graph. This function then calls one of the following functions: **barLinePie(), hst(), geo()**. Each of these functions generates an array of data which is returned back to the **AJAX** call in the view.

BarLinePie($cs, $opts, $yIndex, $xIndex) – is a function used for creating a bar, line, or pie chart. **$cs** is the case information, **$opts** are the options allowed for use in creating a title, and **$yIndex/$xIndex** contain the selections made by the user in the 2 (or one) dropdown menus in the view. All of the three share the same variables so all are created using the same function. The exception here is pie as it only needs one variable but **generateGraph()** determines the other variable based off of the first variable and then treats it as if it were a bar or line chart. Bar and line charts, however, behave normally. The general layout of the function starts by generating an array of keys needed for the chart and then searching and finding all of the data needed for the specified keys and creating a new data array with the final results of the graph. This data array is then returned.

Hst($cs, $opts, $var) – is a function which generates a histogram using the data given. **$cs** is the case information, **$opts** contains an array of the various options allowed for a histogram for the

purpose of creating a title for the chart, and **$var** is the variable the user selected to generate the histogram on. This function then loops through the case data, appropriately grouping and summing the data based on the **$var** and returns the data gathered.

Geo($cs, $opts) – creates a geographic chart for the total number of cases in the subset of data returned from the search controller. **$cs** is the case information and **$opts** are the allowed options for the use of creating a title.

8. Glossary of Terms

Administrator – privileged user capable of performing major changes to database.

Apache – An open-source Web server software package.

Application – Group of programs designed to supply an end-user with expected functionality.

CakePHP – A free, open-source development framework for PHP.

Controller – A PHP class that handles Web requests and serves views and responses.

Control Panel – interface specifically designed to allow administrators to easily perform their tasks.

Database – A structured set of data held in a computer, accessible in various ways.

Database Schema – The definition of a database's structure.

Foreign Key – A field in one table that uniquely identifies a row of another table.

Function – Do we really need to define a function for you?

GitHub – A Web service for software version control.

Human Trafficking Data – The name of the website created by this project.

Model – A PHP class that manages data flow between the application and the database.

MySQL – An open-source relational database management system.

MySQL Workbench – A graphical utility for MySQL databases designed to make creation and management of database schemas more intuitive.

PHP – A general-purpose scripting language that is especially suited to server-side web development.

Primary Key – Uniquely identifies each record in the table.

Prototype – simulates only a few aspects of, and may be completely different from, the final product.

Schema – See **Database Schema**.

TCU – Texas Christian University

UML – Unified Modeling Language; a modeling language designed to provide a standard way to visualize the design of a system.

View – A file that generates output for a specific Web request, called by a controller.

Walk-through – Points during the project where the team describes significant project components with clients and individuals within the team.

Web Application – Application that is accessed by visiting a specific URL.