PSAP DATABASE

Developer's Documentation

Akhilesh Mantripragada
COMS W4995 Internet Tech Econ Policy
Project Fall 2014

Contents

1 Introduction

PSAP Database is a web application that allows its users to access PSAP registry entries. A primary PSAP is defined as a PSAP to which 9-1-1 calls are routed directly from the 9-1-1 Control Office, such as, a selective router of 9-1-1 tandems. The PSAP Database serves as a tool to aid the Commission in evaluating the state of PSAP readiness and E9-1-1 deployment.

The PSAP Database has three types of users including Public Access Users, Private Access Users and Admin Users.

- Public Access Users: These users are allowed to access all PSAP entries. These users can search for PSAP entries and filter by PSAP ID, PSAP Name, State, County, or City. Public Access Users also have access to the PSAP Database API, which accepts requests and returns JSON objects based on the requests. More information about API can be found in the API Documentation File.
- Admin Access Users: Usually members of the controlling body, Admin type users have access to all PSAP entries (including those owned by Private Access Members). These users can create new entries, modify and delete all existing entries. Admin users can also add/remove Admin/Private Access users.
- Private Access Users: An Admin or a Private Access user either owns each PSAP entry. Private Access users have access all entries, but can modify or delete only entries owned by them. These users can also create new entries.

This project deals with the design, implementation, testing and deployment of the above-mentioned tasks for each type of user. PSAP Database has been implemented using the classic client-server architecture details about which are provided under the implementation section. The PSAP Database is currently hosted on Digital Ocean's cloud server.

The purpose of this document is to provide an in-depth description of the architecture design of PSAP Database. The document is an extensive review of the architectural design, as well as the key design decisions that were implemented during the development of PSAP Database. This document also provides detailed description of the database structure, implementation details and testing.

2 Implementation

PSAP Database has been implemented using the client server architecture. This architecture is separated into three parts: a front-end or client application tier, a back-end or server tier and the database tier. The front-end or client application focuses on input requests from the user through the graphical user interface. All requests are sent to the server for permission and user authentication where the server interacts with eh database to analyze, retrieve, and present the requested data through the graphical user interface.

The following technologies were used at each layer of the architecture.

Front End Technology

- HTML 5
- CSS 3
- JavaScript
- JQuery
- Twitter Bootstrap

Back End Technology

- PHP
- Laravel (PHP Framework)

Database Technology

- MySQL

Other Technology

- Version Control: GitHub
- IDE: PHP Storm
- Hosting: Digital Ocean
- Testing: Codeception

Next section of the document provides a detailed description of each layer and design decisions that were taken during the development process.

2.0 Build and Dependencies

- Composer: Download and install Composer from getcomposer.org. Composer can be used to handle all PHP dependencies.
- Laravel: Download and install from laravel.com or use composer to download laravel.
- Code: The code of the project is hosted at https://github.com/amantrip/psap. Please pull code from Github.
- PHP: The system must have PHP version 5.4 or greater.

Once the above-mentioned are installed:

- cd into the project directory and run `$ composer update`. This will install required dependencies required for the project. All required dependencies are listed in composer.json. For new dependencies, please update composer.json and run a composer update.
- MySQL is the default database for this project, please refer to 2.2.2 for database setup.
- Run `$ php artisan serve` to serve the project on the local host.

Running the project on the inbuilt server makes it easy to deploy the project on the localhost.

Alternatively, one could download Vagrant, Virtual box and Homestead and serve the application on a Vagrant box. Homestead Vagrant box comes with the required PHP version and MySQL server; it becomes easier to create a visualized, isolated development environment.

Please follow the link below for more information about Homestead Vagrant Box:

http://laravel.com/docs/4.2/quick#local-development-environment

For convenience, PSAP Database is currently hosted on Digital Ocean's Cloud platform at: http://104.131.71.236/

2.1 Back End or Server Tier

Laravel was used to implement the back end of the PSAP Database. Laravel is a web application framework with expressive, elegant syntax. Laravel makes it relatively easy to perform tasks like routing, sessions and caching. Laravel helps make the development process a pleasant one for the developer without sacrificing application functionality. Laravel is accessible, yet powerful, providing tools needed for a large, robust application. With Laravel MVC or Model View Controller and ORM or Object Relational Model come for free, with all the business logic placed in the Controllers, Model essentially acting as knowledge representation of the database and Views displaying the output to the end-user.

In a Laravel project, there are three main folders:

- /app: consists of all the back-end logic, models, views, configuration files, database migrations, authentication, routes and controllers
- /public: consists of the all the CSS, JavaScript and JQuery code that is visible to the end-user and code that Views use to display information parsed by the controllers
- /tests: All tests are in this folder. This folder consists of: acceptance, functional and unit tests. For the purposes of this project, only functional tests are relevant. More information about tests folder is provided in the Testing section of this document.

In this section we will examine the functionality of each of the controllers found in the /app/controllers folder. There are five controllers and a base controller, which handle all the business logic of the application. Controllers feed View with information to display to the end-user and accept POST/GET input from the end-user.

- Public Controller: This controller handles the “Public Users”. This controller uses its functions to display all PSAP entries to the end-user. This controller also handles API GET requests and returns the resulting JSON object.
- Admin Controller: This controller handles all the business logic of the Admin user. Most functions of this controller are access controlled by Authentication class of Laravel. So a user (any: private access user or admin) who is not authenticated is redirected to the Login page to authenticate. Additionally, only “admin” type users are allowed to access the Admin Controller functions, hence even if a user is authenticated, this controller checks the type of the user to ensure that they have rights to access Admin views. If not, that user is redirected to the Private Controller. Admin Controller consists of multiple

functions that render admin views, accept POST requests (update admin information, reset password etc.) and update the database.

- Private Controller: This controller handles all the business logic of the Private Access user. Most functions of this controller are access controlled by Authentication class of Laravel. So a user (any: private access user or admin) who is not authenticated is redirected to the Login page to authenticate. Additionally, only “private” type users are allowed to access the Private Controller functions, hence even if a user is authenticated, this controller checks the type of the user to ensure that they have rights to access Private views. If not, that user is redirected to the Admin Controller. Private Controller consists of multiple functions that render private views, accept POST requests (update admin information, reset password etc.) and update the database.
- Home Controller: This controller handles the miscellaneous functions like rendering the home page view, and multiple documentation views. Also, since Login and Registration are functions common to both private and admin users, the Home controller handles both login process and registration process. This controller also handles the logout process, which un-authenticates the logged in user.
- Registry Controller: Creation, update, and deletion of PSAP entry are common to both Private and Admin users. Registry controller handles all three functions and the corresponding views.

Please refer to the corresponding PHP files under the /app/controller folder for detailed explanation of each function and Code Inspection.

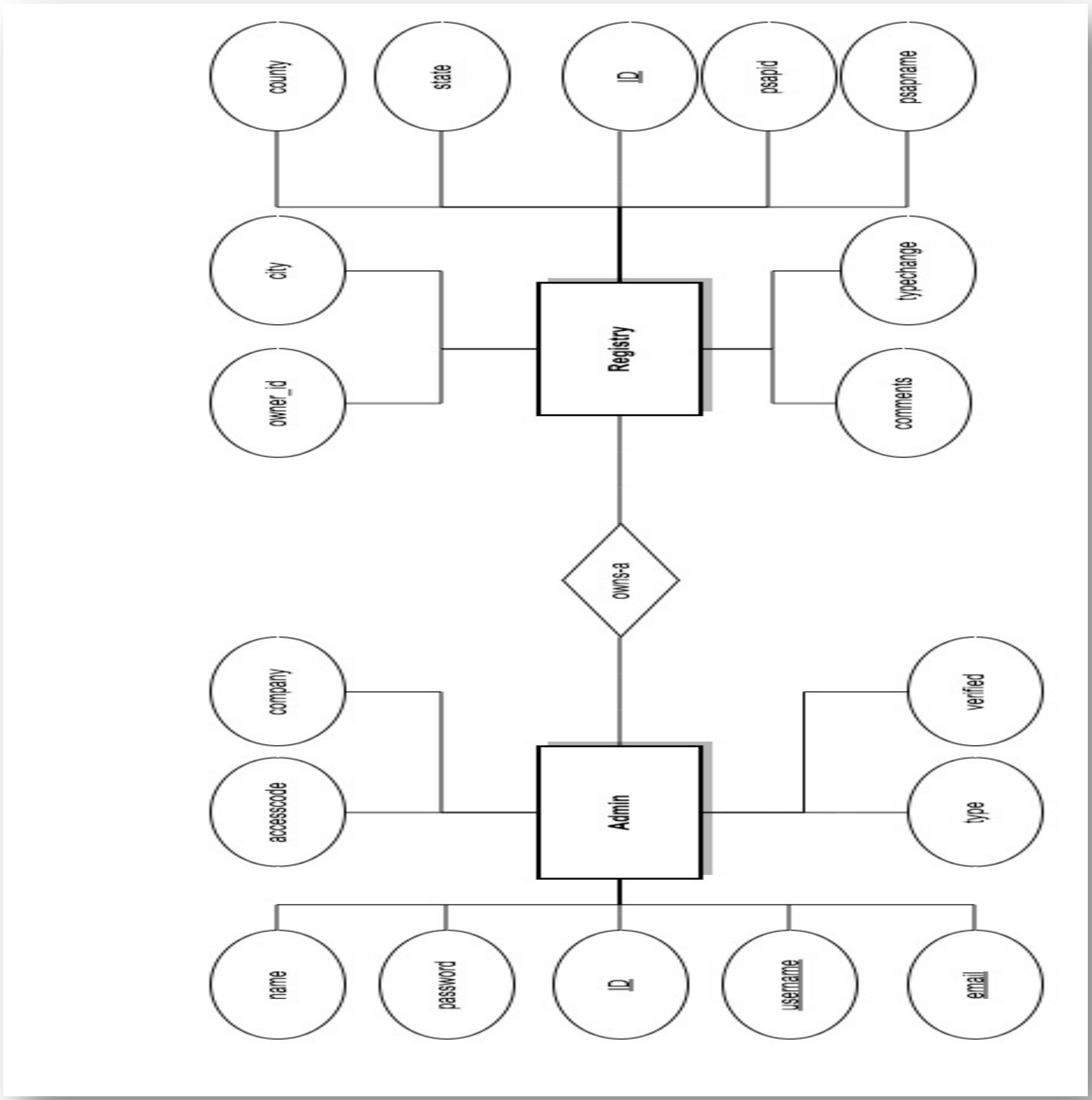
Note: All routes are declared in routes.php

2.2 Database Tier

PSAP Database has been implemented using MySQL on the database tier. This section of the document illustrates the database design and how to access the database.

There are three models for this application namely: Admin, Registry and Owners. Models in Laravel can be thought of as knowledge representation for database tables. Each model refers to one table; in this case Admin.php refers to admin table, Owners refers to owners table and Registry.php refers to registry table.

Entity Relation Diagram:



2.2.1 Database Schema

Although the backend implementation consists of two separate controllers for Admin and Private Access users, for the purposes of the database both such users are considered admins. This helps the developers make many simplifications including common login and registration process. Since Laravel provides an in build Authentication class, developers can refer to 'admin' table in auth.php to perform all authentications on one table instead of two tables. Additionally, since most information about Admin and Private Access users is similar, it makes sense to store them as one table.

In order to different between the two users, we add a column to admin called 'type' which can be of two types: admin, and private.

Here is the actual database schema:

Admin

- ID: int, primary key
- Email: varchar, unique
- Password: varchar (Note: All passwords are stored after salting and hashing using laravel's Hash::make($password) function)
- Name: varchar
- Company: varchar
- Accesscode: varchar
- Type: enum['admin', 'private']
- Verified: enum['yes', 'no']

Registry

- ID: int, primary key
- Psapid: int, unique
- Psapname: varchar
- State: varchar
- County: varchar
- City: varchar
- Typechange: enum {NC, O, A, M, S}
- Text-911: enum{no, yes}
- Comments: varchar
- Changes_Owner_id: int (Keeps track of the last user who updated the entry)

Owners

- ID: int primary key
- PSAPID: int,
- Owner_ID: int, ID of owner of the PSAP Entry. This facilitates multiple admin owning one entry. (Note: Front-end for this feature has not been implemented)

2.2.2 Database setup

Please find database.php under /app/config to setup database (HOST, USERNAME, PASSWORD, DATABASE and PORT). Alternatively, developers can create `.env.php` and put all required credentials in an array.
Example of a `.env.php` file would be:

```
<?php
return [
    'DB_HOST' => 'localhost',
    'DB_NAME' => 'psap',
    'DB_USERNAME' => 'homestead',
    'DB_PASSWORD' => '*****',
    'DB_PORT' => '33060'
];
?>
```

Once configured, developers can run the following command to migrate all database tables to their local database:
$php artisan migrate
Note: Create a 'psap' database in your local server before running the migrate command.

Alternatively, developers can find psap_2014-11-14.sql under /app/database and manually import the database to their local database incase the migration fails. Ideally, php artisan migrate is a better choice because it keeps track of the migrations and developers can rollback to older versions.

To create a new MySQL table developers an run the following command:

```
$ php artisan migrate:make create_dummy_table –table=”dummy” --
create
```

This create a php file under /app/database/migrations. Admin migration looks like this:

```
public function up()
{
    Schema::create('admin', function(Blueprint $table)
    {
        $table->increments('id');
        $table->String('username')->unique();
        $table->String('email')->unique();
        $table->String('password'); #stored as hash
        $table->String('name');
        $table->String('company');
        $table->String('accesscode');
        $table->enum('type', ['admin', 'private']);
        $table->enum('verified', ['no', 'yes'])->default('no');
        $table->timestamps();
        $table->rememberToken();
    });
}
public function down()
{
    Schema::drop('admin');
}
```

Developers can use this template to create new tables and run a “php artisan migrate” to create the table.

3 Testing

Codeception was used as a testing tool for PSAP Database project. For more information about Codeception please visit www.codeception.com.

3.1 Testing Coverage

All files relevant to testing can be found under /tests folder of the project. Codeception inherently creates three folders functional, acceptance and unit tests. For the purpose of this project only functional tests were created with a goal to have over 85% line coverage. About 4 functional tests have been created to test the functionality of the system.

3.2 Important Codeception Commands

To run all functional tests developers can use the following command:

```
$ vendor/bin/codecept run functional
```

This runs all functional tests that have been created and return how many tests were successful and how many failed.

To run a specific test developer can use the following command:

```
$ vendor/bin/codecept run functional LoginCept.php
```

where LoginCept.php is the file that contains the test.

To create a new test developers can use the following command:

```
$ vendor/bin/codecept generate:cept functional NAME_OF_TEST
```

This creates a new php file with name NAME_OF_TESTCept in the /tests/functional folder.

Finally, to find the coverage of the tests, developers can run the following command:

```
$ vendor/bin/codecept run functional --coverage-html
```

Please note that *xdebug* is required to run coverage of tests. Vagrant box provides it by default.

This command creates an html file under /tests/_output/coverage which consists of all coverage statistics.

3.3 Configuration

To add new configurations to each testing, developers can edit the functional.yml, acceptance.yml and unit.yml. These files can be found under /tests folder.

To add and remove folders from testing developers can add folders to “include” and “exclude” list in codeception.yml file that can be found in the root directory of the project. Currently all /app/* files have been included for testing.

4 Conclusion

This document provides an overview of the implementation of PSAP Database project.
For the source code of the project please visit the Github repository at https://github.com/amantrip/psap.
For more information about the API Documentation please refer to the API Documentation document (Appendix A).
For more information about How To's please refer to How To document (Appendix B).
This documents provides the end users an overview of how to use the website with screenshots.
For any other information please contact me at am4227@columbia.edu.

An

akhilesh
mantripragada

Presentation

COMS W4995 Internet Tech Econ Policy
Term Project
Fall 2014