ECE 435 Fall 2013

NM HIDTA DATABASE

PROGRAMMER'S GUIDE

Bhim Subedi, Amanda Gonzales, Julian Lucero, Kurt Hollowell,
Jose Fragoso, Shu-Jie Chen, Dominic Quintana, Thomas Mondragon, Elijah Wolfe

Front Matter

This guide is intended to explain the basic structure of our NM HIDTA database. Inevitably, our design will need to be modified; whether to increase functionality, improve the data security, or to improve user experience. This document is intended to detail the configuration and organization or the database and interface software. However, this document is not intended to serve as a "blue print" of our design.

Overview of System and Data Flow

To begin, it was the decision of the collaborators to design the system using rails version 3.2.8. This was determined easily by the lack of experience with any other tools utilized for web design and database creation,. Additional gems used include, sqlite3, simple-form, and bcrypt-ruby. This guide assumes that the programmer has some prior experience with Ruby on Rails.

The Rails structure divides the application into three subdirectories: Model, View, and Controller. This shaped the way we approached the assignment, as it helped to divide the software application into database logic and interface logic.

Controller

The implementation of our NM HIDTA database utilized the same notion of division. Hence, we utilized 9 different controllers to split up the code and functionality. Controller modules were made for the main interface, the administrator functionality, and the database design. A short description of each is detailed below.

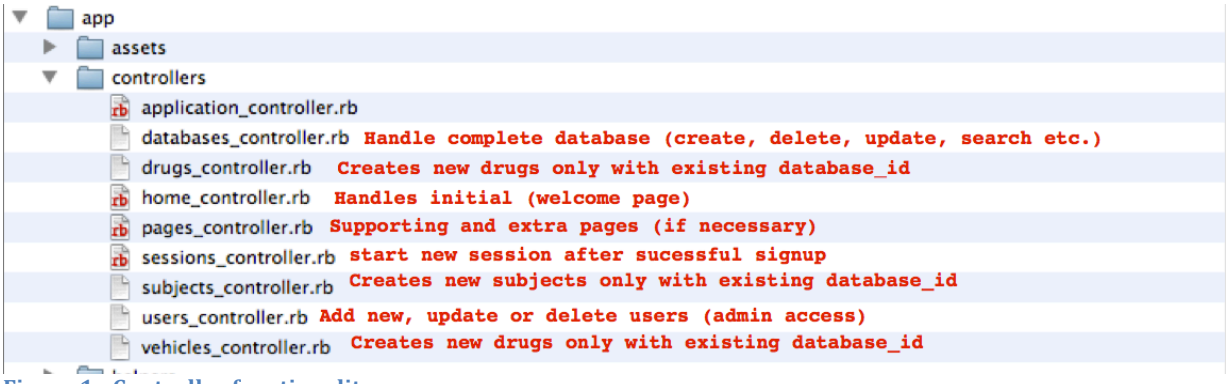

Figure 1 - Controller functionality

Database Design

To organize our database we went with a hierarchical approach. At a higher level of abstraction, there is only one main database that handles all of the seizure form entry and storage. Initially, this was the entirety of our database design. We evolved our

approach for the database structure in order to add the capability to enter multiple entities and link them together in a more organized manner. At the top level, there is a main database that acts as the "parent." Within this database, three other database structures exist for storage of vehicles, drugs, or subjects. The different levels are tied together with the database entry ID that is automatically created upon a new entry. Theoretically, the programmer could create each part of the database separately, but the system is designed such that the entry in the children databases may not create a new database ID number.

A controller is implemented for each of them and the basic database functions are included within. In the future, in order to expand on the database or the types of entries possible, this is where changes would be made.

Models

The models integrate the database entities. For instance, the database must interact with the models in order to render a view. In addition, search entities are defined in the models. The figure below shows the relationship between the different levels of our database, as well as some their basic functions.

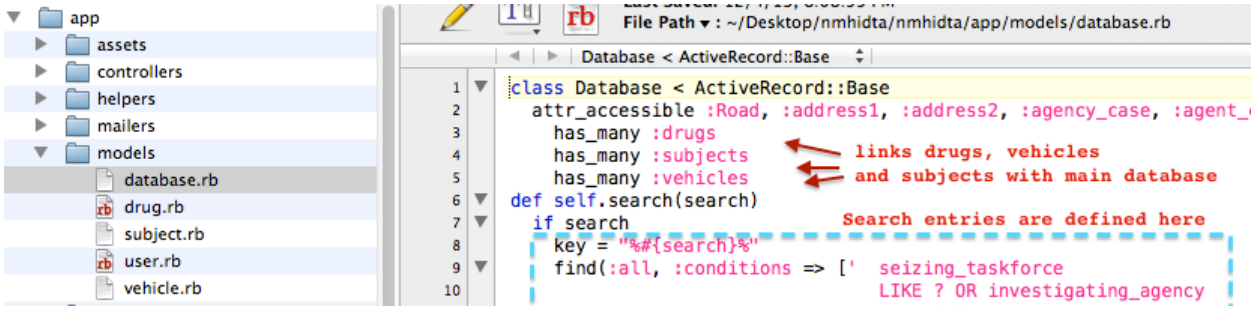

Figure 2 - Model details

Views

The views of the controllers we want to display are rendered. Perhaps the most crucial part of the user interface, the seizure form, was created inside the form view. In addition, partials were created so that the whole webpage didn't need to be loaded if only a portion of it changed. The audit trail in development is obtained from the information in the seizure view. A short description of the content in each view is depicted below.

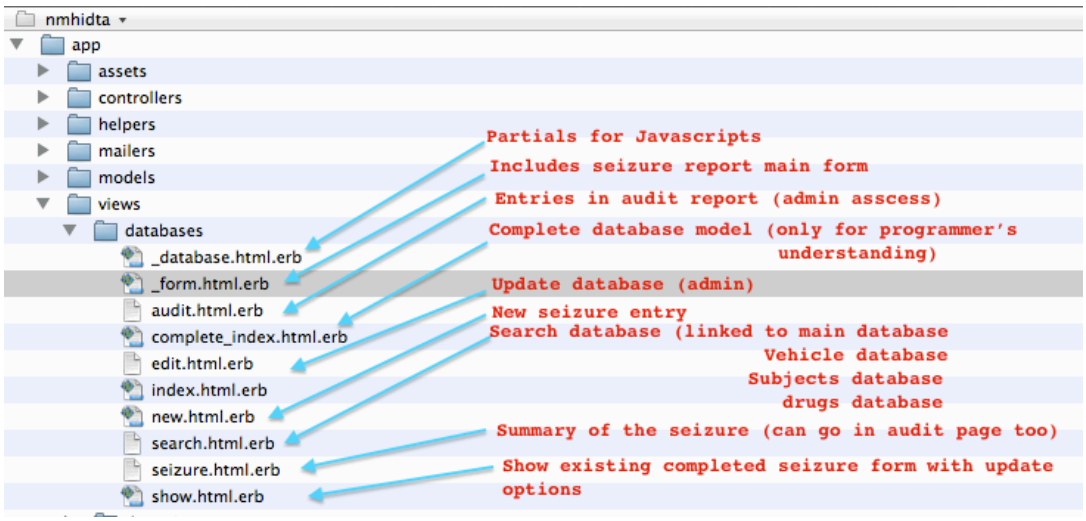

Figure 3 - Describes the content of each view

Assets

Ruby on Rails creates an assets folder when the program is generated. Because of our lack of expertise, we did not want to edit what was generated. A separate folder was created for the cascading style sheets and JavaScript used to render the page.

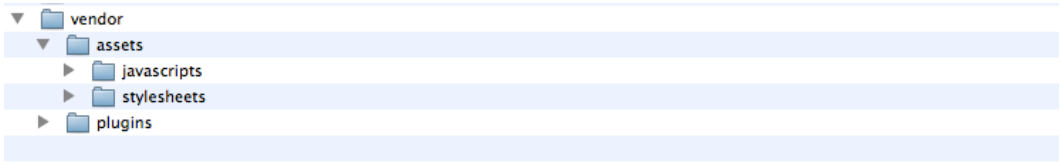

Figure 4 - Interface Assets

Here we utilized an open source CSS template to begin our interface design, and customized it to best suit our need. We created "helpers" to enhance the usability of the form entry in the form of dropdowns, as well as remove the need for some verification of valid entries. We used our best judgment to create helpers that would be most effective for the users, but did not create them for every field, as we did not know exactly how the data would be entered.

Figure 5 - Helpers. Not all have content.

Code:

All of the most current Ruby on Rails code is available on the Green Team's github here.