School of Library and Information Studies
University of the Philippines Diliman
Second Semester, A.Y. 2017-2018

Content Management System (CMS)

Submitted to
Mr. Janny S. Surmieda
In Partial Fulfilment of the Requirements for LIS 161

Submitted by
LIS 161 THY1

June 1, 2018

I. Making the CMS

There were seven identified functionalities that must be addressed in order to make the CMS:

1. Roles and Permissions Management
2. Navigations and Navigation Types
3. User Interface
4. Content Interface
5. Content Type Management
6. Revisions Management
7. Content Navigation

Roles and Permissions are managed through Role ID assignment during registration. Actions can be assigned to a certain role via this function. Role IDs are a useful tool to manage the access to certain functionalities. *Middlewares* are used to manage the access for the designated user roles. *Route::group* for each user type is defined in *web.php*.

Navigations and Navigation Types will dictate which of the available navigation links will be accessible to the specified user.

Users Management The scaffold for login and register was done by utilizing the *php artisan make:auth* command in laravel.

Interface involves the whole look and feel of the CMS. In this project, it has been decided to keep it simple and clean. A library theme is applied to both *welcome.blade.php* and to *app.blade.php*.

Content Interface is where users of the CMS manage their posts. These, again, are kept simple and features only the essentials. Text boxes and buttons for editing and deleting posts. Content creation is managed through the *ContentController.php* which contains the functionality for Creating, Editing, and Deleting contents

Content Type Management classifies which type of post has been done. In this project, we limited ourselves to text posts to not further complicate things. This simple limitation also serves as a stepping stone to what can be described as a steep learning curve for CMS development.

Revisions Management include saving all edits made to a post. Posts are not overwritten but are all kept in a table called *Revisions*. All edits made are still assigned to their title and kept track of via an ID saying which number of revision the current iteration is. This is integrated to the *ContentController.php*.

Content Navigation is solely involved in specifying which content goes into which navigation link.

II. Functionalities of the CMS

• To initiate the database `lis161cms`, the command `php artisan migrate` is used. To populate database with values, the command `php artisan db:seed` is used.

• The CMS includes a registration system where users are able to input their credentials into the system. Users can also specify whether they will take on the role of either a *User* or an *Admin*.

• A user who is registered as an *Admin* is able to manage other users. They can edit their designation and can also delete users.

• The CMS also includes facilities to add contents as they wish. Aside from that, there is also a function to edit existing posts. These edits are saved as different versions under a table called *Revisions*.

• Also, there is a functionality which enables user to delete posts. It should be noted that these deletes are just *soft deletes* and can be reversed via *PHPmyAdmin*.

• Registered users are able to filter viewing all the contents or just what they authored themselves. In line with this function, users are also allowed to delete and edit posts which they made.

• Registered users with the role of *Admin* can change the links found in the users' navigation bar. The options are presented in checkboxes with each option attached to a predetermined link for easier routing. Deselecting the options would render them inactive. The navactivated column found in the navigations table of the database is used to determine whether an option has been selected or deselected.

• An *Admin* can delete and edit any post regardless of who authored it.

III. Contribution

The class did what they could to reach the extent of the current state of the project. In the beginning, the class was divided into teams to accomplish different aspects of the development project to maximize the efficiency of the class. Shown below are the designated work distributions.

Class Leader: Jeanne Torres
Roles and Permissions: Gynard Alfonso & Joshua Tade
Navigations and Navigation Types: Jastine Ducut & Ella Abela
User Interface: Princess de Vera & Angelyn Sager
Content Type Management: Kyle Jemino & Arvin Aquino

Revisions: Issa Topacio & Eunice Elazegui
Content: Winston Isaac & Kevin Gabaldon-Co
Users: Jeanne Torres & Nicole Tacuboy

The students most comfortable with programming led the project flow and debugged as necessary. They also helped fix mistakes in commits made by the more inexperienced students, and handled their pull requests and branches. The project was integrated, unified, and then finalized with each incremental change. Details for each contribution can be found in the GitHub repository.