4-19-2019

The Standards Project

Dustin Robbins
Western Michigan University, dustinprobbins@gmail.com

The Standards Project Documentation

Mitchell Hobner, Dustin Robbins, Shahbaaz Singh

CS-4910

April 29, 2019

Table of Contents

Introduction

The Standards Project is a web app that is intended to assist United States K-12 students in meeting the academic standards each state has set out for their students. The app is planned to allow instructors to see how proficient incoming students are in standards set for the prior grade (e.g. 6th grade students' 5th grade math skills would be shown) and launch "interventions"—be these online modules with educational content and quiz questions, after school activities, or some other form of instruction—in order to help students in problem areas while spending a minimum of class time on old material.

Our team has been tasked with creating the barest foundations of the functional app; our focuses were database organization and presenting test data to various web pages. This documentation while explain how to install, set up, and modify the project as it currently exists, and give as much grounding in the design of the app as possible for the benefit of future developers.

Setup

The Standards Project was developed using a pre-packaged PHP development environment called Laravel Homestead with a preference for Google Chrome as a browser. The Laravel Homestead environment runs on a Vagrant environment and includes the Ubuntu 18.04 operating system, MySQL, Nginx server technology, and the Laravel framework itself. Multiple virtualization programs are supported by Vagrant, including VirtualBox and VMware. Detailed instructions for setting up the development environment are given on Laravel's own website at the following URL:

https://laravel.com/docs/5.8/homestead.

Homestead works by setting up a virtual Ubuntu environment which will contain everything needed for the project, including database, server, and packages that make development easier. One such package includes the Artisan package which can be used for creating migrations and models. The files of the project are always in sync between the local and virtual machines, meaning any changes made to the local files will change the virtual files. The syncing is managed in the Homestead.yaml file. Once this has been set up, simply run '`vagrant up`.' This will activate the virtual machine, and the site can be accessed on any browser by navigating to Homestead.test (or whatever URL is specified in the .yaml file).

If errors occur on startup, consult Laravel's documentation for assistance.

Version Control

Laravel Homestead comes with support for Git for the purposes of version control. The project can be found here:

https://github.com/ShahbaazSingh/thestandardsproject

It is important to note that no Laravel project can successfully be run immediately after being pulled from a repository, as some files are altered on the initial pull. One example would be the .env file. For this reason, there are some steps that must be followed after the first pull of a Laravel project, as detailed at the following article:

https://devmarketer.io/learn/setup-laravel-project-cloned-github-com/

As the article explains, all Composer dependencies will need to be reinstalled, the env.example file will need to be renamed, an app encryption must be generated, and a new database must be created, populated, and seeded on any local virtual environment. Once all of these steps have been completed, it won't be necessary to repeat them unless sweeping changes have been made to the app.

Testing and Site Navigation

The site can be navigated using three user types: teacher, student, and principal. The first page of the site for an unauthenticated (signed out) user will be a login page. Registering a new user will give them the role of "student" in the database by default. A user can log in with a username or email address, as well as a password.

A number of mock users currently exist in the database that can be used for testing when the database has been seeded. There are twenty student logins named 'studentone' to 'studenttwenty,' and the password for each is 'passwordone'. Email logins are the same same as the username with '@email.com' appended.

All teacher logins use either 'teacherone', 'teachertwo', or 'teacherthree'. 'teacherone' is the only one that currently has classes and students assigned to them. The password for 'teachone' is 'passwordone'.

The only mock principal login is 'principalone' with the password 'passwordone'.

Note that all twenty mock students belong to class id 1, which belongs to teacherone.

There are currently two classes assigned to teacherone. Currently, only the first class (class id = 1) has students in it. The only extra option on the bottom that is functional is the module tracking page—these are modules sent by teacherone. Clicking on class id 1 will lead to a page that shows how proficient the students in the class are in each subject. By the first release of the app, no teacher should be able to assign modules to students outside of their own subject.

Clicking on 'Math' will lead to a detailed page of students and how proficient they are in that subject. Clicking on a specific proficiency will show a page that shows what units the students scored "not proficient" on. The buttons, as of writing this documentation, are not currently functional. They buttons are planned to be able to select the checkboxes of all the

students that marked an X under it as a quality-of-life feature. A teacher can still deselect any student. After selecting the students, the teacher can choose modules to send them. The module will be added to the database table for student assigned modules, with the current date as the assigned date, and a week after the assigned date as the due date.

The student login will lead to a student page listing the classes they are currently enrolled in. All a student is supposed to see so far is their class list that and the modules assigned to them. As things are, all modules the student has ever been assigned will be visible and displayed in the order that they were assigned. All incomplete modules will have an alert sign stating that new modules have been assigned. The student may click on any module and complete them. Answering questions incorrectly will give a hint, and the student may be prompted to answer all questions until all answers are correct. Once a module is done, its entry will update on the main student page. The principal login will simply show all students enrolled in a class in the school "PS101."

Code and Functions

Laravel follows the Model-View-Controller (MVC) architecture, meaning Models, Views, and Controllers are used as separate components that communicate with each other to allow the site to function. Models are packages that describe an object from a database table. Currently the project has User, Classes, Proficiency, and Modules models. The code within them simply describes a few features such as what a primary key would be if it is not the default key. Each model comes with a '`$fillable`' variable which must be declared with any columns entered into it. Currently there is no use for `$fillable`, however, it must contain at least one field in order to access Eloquent relationships. In models that require "pivot relationships" or

relationships with other models with the use of foreign keys and/or intermediate tables it is possible to declare a class that's the plural of what one wishes to access.

Models are made alongside migrations through the use of artisan commands. Migrations should follow a similar title format as a lot of functions rely on these naming conventions. Migrations are the way a model is set up in a SQL database. Every time a migrate command is run, it will take the layout of the migrations and create a table in that structure. They also act as version control for databases as they can be rolled back or refreshed. When the project is first pulled to a local environment and it has been given the proper database tables with empty fields in them, the migration will create the structure of those tables. In all cases, please refer to the official Laravel documentation for more information:

https://laravel.com/docs/5.8/migrations

Creating a migration involves specifying what value a table column holds, what the column name is, and other information of that manner. Each model has a corresponding migration, however, not all migrations need to have models, in the case of intermediate or "pivot" tables. It is necessary to drop existing tables in every migration, as a database should be clean before it is repopulated.

Database seeders are an essential tool for populating databases with rows in tables. There are currently seeders for every table in the database. The seeder's main function must first use the `delete()` function to delete all rows in the table. It does not delete the table itself. After, it is possible to either use the '`DB::insert`' facade to directly insert data into a database using arrays, or to use the '`Model::create`' facade to create models before they are inserted into the database. The latter allows for eloquent relationships, while the former only allows DB queries. There must be a 'Database Seeder' class which calls all the corresponding database

seeder classes that have been made in the desired order. It is imperative that, every time a change is made to either a migration or database seeder, '`php artisan migrate:refresh --seed`' is run to perform a fresh install of the database with the default mock values. If this returns an error, use the '`fresh`' command instead of '`refresh`' to do a completely blank installation. If errors persist, it may be necessary to `drop` and re-create the "thestandardsproject" database in MySQL, then re-seed.

Laravel's Controller functions are separated into main controller and middleware classes. Middleware are controllers that can be called in a controller's constructor, usually in the case of authentication. No middleware has been written for the project to date, however some middleware classes were automatically created along with the initial project. These middleware classes should be left alone unless changing the login function itself. In order to develop complex websites, Controllers must be used for managing data, and views for showing data. When a link in a view is accessed with information to send to another view, it must call the controller and its corresponding function within the web.php file. These are referred to as routes. A route will send data to a controller and the controller will decide what to do with the data.

It should be mentioned that it is simple to do data management within views, but it is not the best practice for them to be the main form of data management within the app. Controller functions are called from routes and can run a series of data management code. Once that is done, data is sent back to the views for use using the '`->with()`' function. Some data management has been implemented in some views, specifically the home page, but this has been left incomplete on other pages.

One existing controller called the ProfileController simply returns the homepage, managed by the HomeController. The other controllers were created automatically through the

default login page. The ProfileController has functions called by all users when navigating from view to view. An important function within that controller is the '`getProficiencyOverview`' function which returns data containing the proficiency of students within a class. This function is heavily relied upon by all teacher views.

The other functions are straight-forward, with comments left for further clarification. Most functions make sure that a user accessing a page has the right to access the page. For example, a teacher should not be able to access a class they do not teach, so the functions check whether that class on the URL is within their class list. Currently only GET forms are used for sending data through URLs. The views are rather simple, documented with in-line comments. The existing code demonstrates the recommended way of gathering information from the database.

Future Considerations

One of the most important points of development is expanding on the login and register functions of the site. It would be ideal to add a verification system, or an admin page which can change the roles certain users have. Another consideration is to find a way to gather information from modules after they are completed and to find a way to compare the progress made to what the proficiency unit for that student shows.

Modules themselves still need a good deal of additional support; there is currently no way to add or construct modules, and thus there is no designated file type for them to be built as.

Thought should be put into creating a parent view that would allow parents to see their children's progress with a similar view to a student's, sans the ability to take modules.

Emphasis should also be put on adaptive visual design, as ideally the app will someday work for mobile as well as desktop instances of Google Chrome.

Database

The following is an ER diagram of the relationships between the tables used in the "thestandardsproject" database.