Project Documentation

Server & File Installation Process:

As the project is developed using the following technologies with their versions

1. PHP (8.1)
2. Laravel (9.2)
3. Vue JS (2)
4. MySQL
5. Apache

Server Setup:

As the project is based on LAMP stack therefore the server should be linux preferably the Ubuntu version greater than 20.

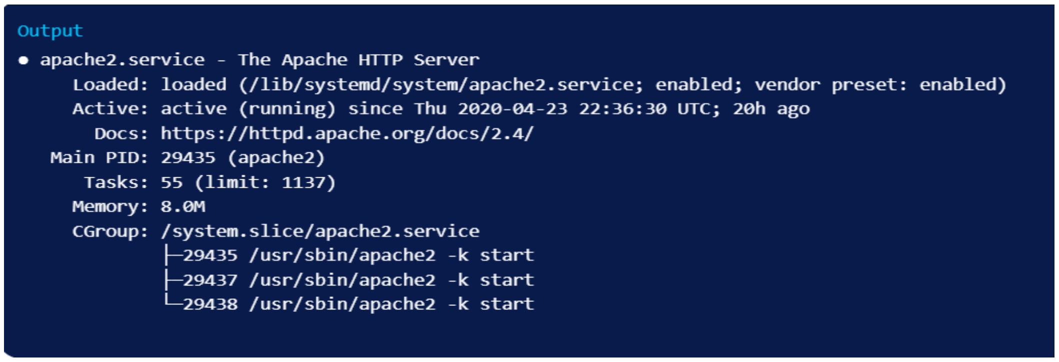
Following steps should be followed for the server setup:

1. **Apache Install**
    a. To install the apache on the server, run the following command for updating the local package index to reflect the latest upstream changes:
        i. sudo apt update
    b. Then, install the `apache2` package:
        i. sudo apt install apache2

c. Now the apache has been installed and to check whether it is installed or not, run the following command:

   i. sudo systemctl status apache2

   You will see the following output



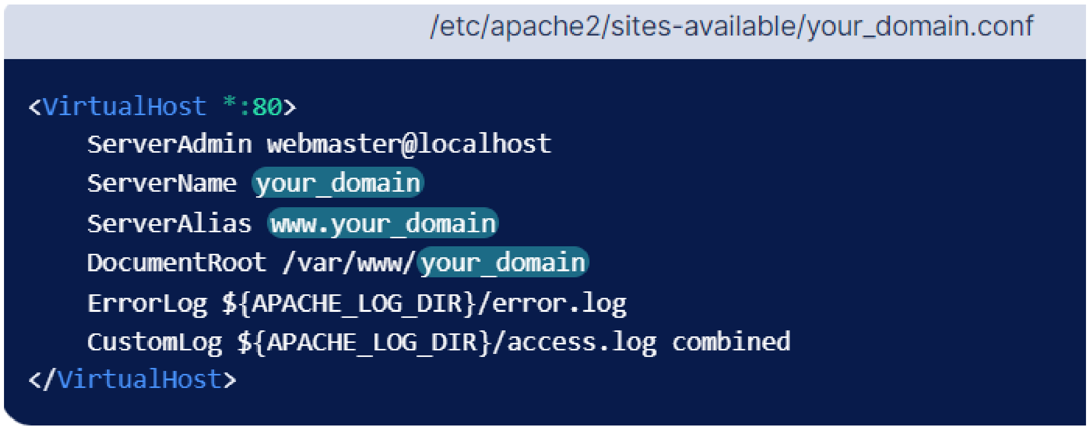
```
Output
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2020-04-23 22:36:30 UTC; 20h ago
       Docs: https://httpd.apache.org/docs/2.4/
   Main PID: 29435 (apache2)
      Tasks: 55 (limit: 1137)
     Memory: 8.0M
     CGroup: /system.slice/apache2.service
             ├─29435 /usr/sbin/apache2 -k start
             ├─29437 /usr/sbin/apache2 -k start
             └─29438 /usr/sbin/apache2 -k start
```

   As confirmed by this output, the service has started successfully.

2. Virtual hosts configuration

a. Create the directory for your_domain as follows:
   i. sudo mkdir /var/www/your_domain
b. Next, assign ownership of the directory with the `$USER` environment variable:
   i. sudo chown -R $USER:$USER /var/www/your_domain
c. To ensure that your permissions are correct and allow the owner to read, write, and execute the files while granting only read and execute permissions to groups and others, you can input the following command:
   i. sudo chmod -R 755 /var/www/your_domain
d. Next, create a sample `index.html` page for testing using `nano` or your favorite editor:
   i. sudo nano /var/www/your_domain/index.html
e. In order for Apache to serve this content, it's necessary to create a virtual host file with the correct directives. Instead of modifying the default configuration file located at `/etc/apache2/sites-available/000-default.conf` directly, let's make a new one at `/etc/apache2/sites-available/your_domain.conf`:
   i. sudo nano /etc/apache2/sites-available/your_domain.conf

f. Paste in the following configuration block, which is similar to the default, but updated for our new directory and domain name:

i. 

/etc/apache2/sites-available/your_domain.conf

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    ServerName your_domain
    ServerAlias www.your_domain
    DocumentRoot /var/www/your_domain
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

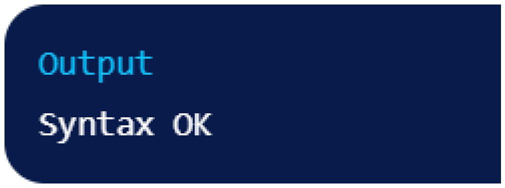
g. Notice that we've updated the `DocumentRoot` to our new directory and `ServerAdmin` to an email that the your_domain site administrator can access. We've also added two directives: `ServerName`, which establishes the base domain that should match for this virtual host definition, and `ServerAlias`, which defines further names that should match as if they were the base name.

h. Save and close the file when you are finished.

i. Let's enable the file with the `a2ensite` tool:
   i. sudo a2ensite your_domain.conf

j. Disable the default site defined in `000-default.conf`:
   i. sudo a2dissite 000-default.conf

k. Next, let's test for configuration errors:
   i. sudo apache2ctl configtest

l. You should receive the following output:
   i. 

```
Output
Syntax OK
```

m. Restart Apache with the following command to implement your changes:
   i. sudo systemctl restart apache2

3. **Laravel project configuration on Server:**

    a. Put the laravel project inside /var/www/ folder, make sure to sepcify the public folder inside the laravel project the root directory in the above file we created your_domain.conf
    b. Now when you will hit the domain, index.php file inside the public folder will be hit, and then from index.php, laravel will bootstrap the entire application, this is how the laravel request life cycle works. To learn more about the laravel architecture you can read the official docs of laravel: https://laravel.com/docs/10.x/lifecycle
    c. Now comes the database part, you have create a database and get the credentials. After you have the credentials create an .env file and paste the credentials there. Laravel has .env file which we used to store the keys, password of application.
    d. Now the database has been connected, the next part is to run the migrations which you can do by running the following commands inside the laravel project directory: php artisan migrate
    e. This will create all the tables in out database
    f. Now our application backend part is ready, lets move to setup the frontend part.

4. **Frontend**
    a. To setup the server, as we have used VUE in out application, we have to setup the vue as well.
    b. You have to run the following commands to setup the vue and all other things which are necassary for laravel ecosystem: npm install
    c. After this to compile to vue js component into the javascript we have to run the following command: npm run prod
    d. After this our front end is ready
    e. Now the application is ready to surf

Laravel PHP System Structure:

Lifecycle Overview:

The entry point for all requests to a Laravel application is the public/index.php file. All requests are directed to this file by your web server (Apache) configuration. The index.php file doesn't contain much code. Rather, it is a starting point for loading the rest of the framework.

The index.php file loads the Composer generated autoloader definition, and then retrieves an instance of the Laravel application from bootstrap/app.php. The first action taken by Laravel itself is to create an instance of the application / service container.

Http Kernel:

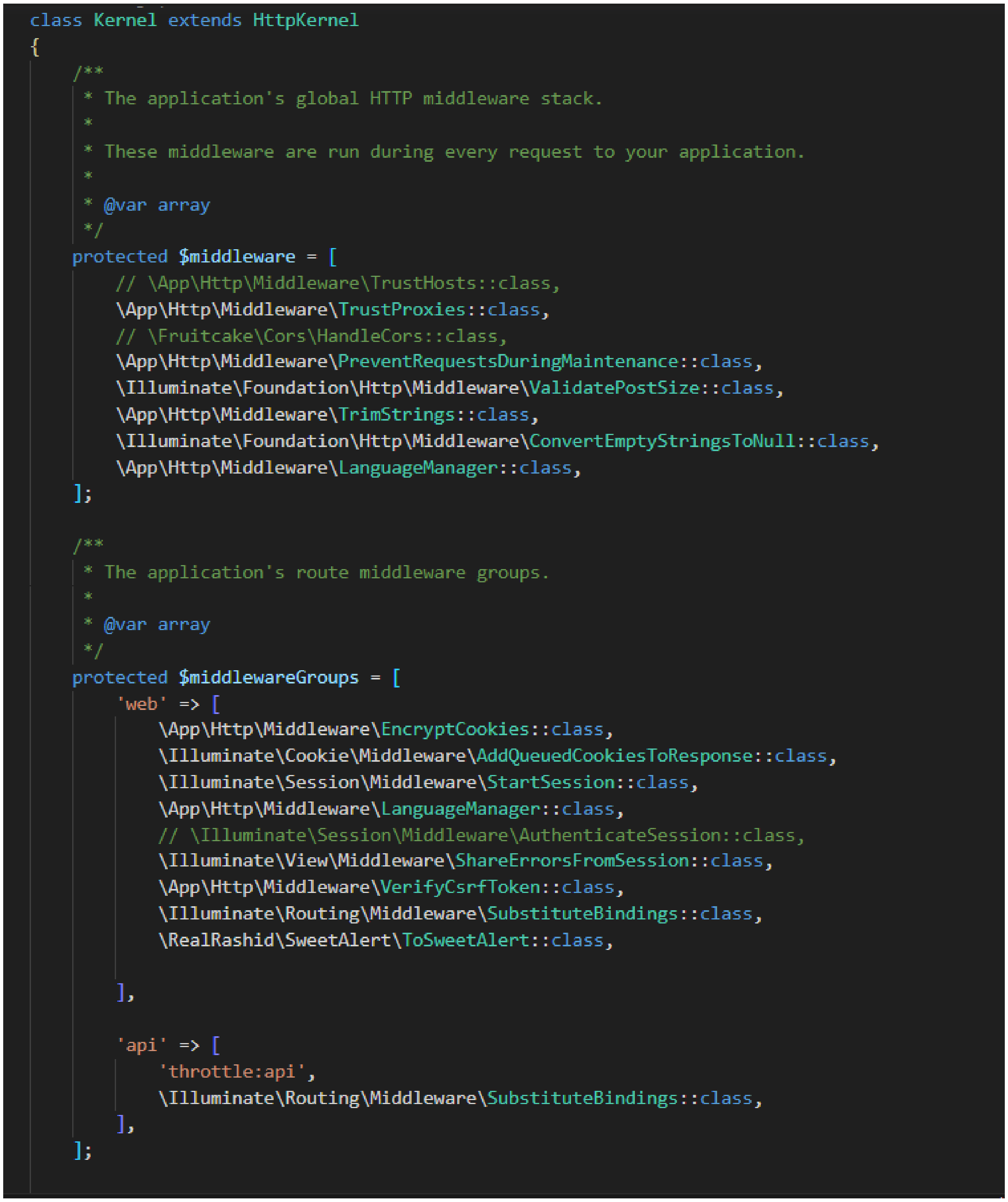
Next, the incoming request is sent to either the HTTP kernel or the console kernel, depending on the type of request that is entering the application. These two kernels serve as the central location that all requests flow through. For now, let's just focus on the HTTP kernel, which is located in app/Http/Kernel.php.

The HTTP kernel extends the Illuminate\Foundation\Http\Kernel class, which defines an array of bootstrappers that will be run before the request is executed. These bootstrappers configure error handling, configure logging, detect the application environment, and perform other tasks that need to be done before the request is actually handled. Typically, these classes handle internal Laravel configuration that you do not need to worry about.

The HTTP kernel also defines a list of HTTP middleware that all requests must pass through before being handled by the application. These middleware handle reading and writing the HTTP session, determining if the application is in maintenance mode, verifying the CSRF token, and more. We'll talk more about these soon.

The method signature for the HTTP kernel's handle method is quite simple: it receives a Request and returns a Response. Think of the kernel as being a big black box that represents your entire application. Feed it HTTP requests and it will return HTTP responses.

Here is the screenshot of kernel class, with the route middlewares and middleware groups which we are using in out application



```php
class Kernel extends HttpKernel
{
    /**
     * The application's global HTTP middleware stack.
     *
     * These middleware are run during every request to your application.
     *
     * @var array
     */
    protected $middleware = [
        // \App\Http\Middleware\TrustHosts::class,
        \App\Http\Middleware\TrustProxies::class,
        // \Fruitcake\Cors\HandleCors::class,
        \App\Http\Middleware\PreventRequestsDuringMaintenance::class,
        \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,
        \App\Http\Middleware\TrimStrings::class,
        \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,
        \App\Http\Middleware\LanguageManager::class,
    ];

    /**
     * The application's route middleware groups.
     *
     * @var array
     */
    protected $middlewareGroups = [
        'web' => [
            \App\Http\Middleware\EncryptCookies::class,
            \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
            \Illuminate\Session\Middleware\StartSession::class,
            \App\Http\Middleware\LanguageManager::class,
            // \Illuminate\Session\Middleware\AuthenticateSession::class,
            \Illuminate\View\Middleware\ShareErrorsFromSession::class,
            \App\Http\Middleware\VerifyCsrfToken::class,
            \Illuminate\Routing\Middleware\SubstituteBindings::class,
            \RealRashid\SweetAlert\ToSweetAlert::class,

        ],

        'api' => [
            'throttle:api',
            \Illuminate\Routing\Middleware\SubstituteBindings::class,
        ],
    ];
```

Service Providers:

One of the most important kernel bootstrapping actions is loading the service providers for your application. Service providers are responsible for bootstrapping all of the framework's various components, such as the database, queue, validation, and routing components. All of the service providers for the application are configured in the config/app.php configuration file's providers array.

Laravel will iterate through this list of providers and instantiate each of them. After instantiating the providers, the register method will be called on all of the providers. Then, once all of the providers have been registered, the boot method will be called on each provider. This is so service providers may depend on every container binding being registered and available by the time their boot method is executed.

Essentially every major feature offered by Laravel is bootstrapped and configured by a service provider. Since they bootstrap and configure so many features offered by the framework, service providers are the most important aspect of the entire Laravel bootstrap process.

Routing:

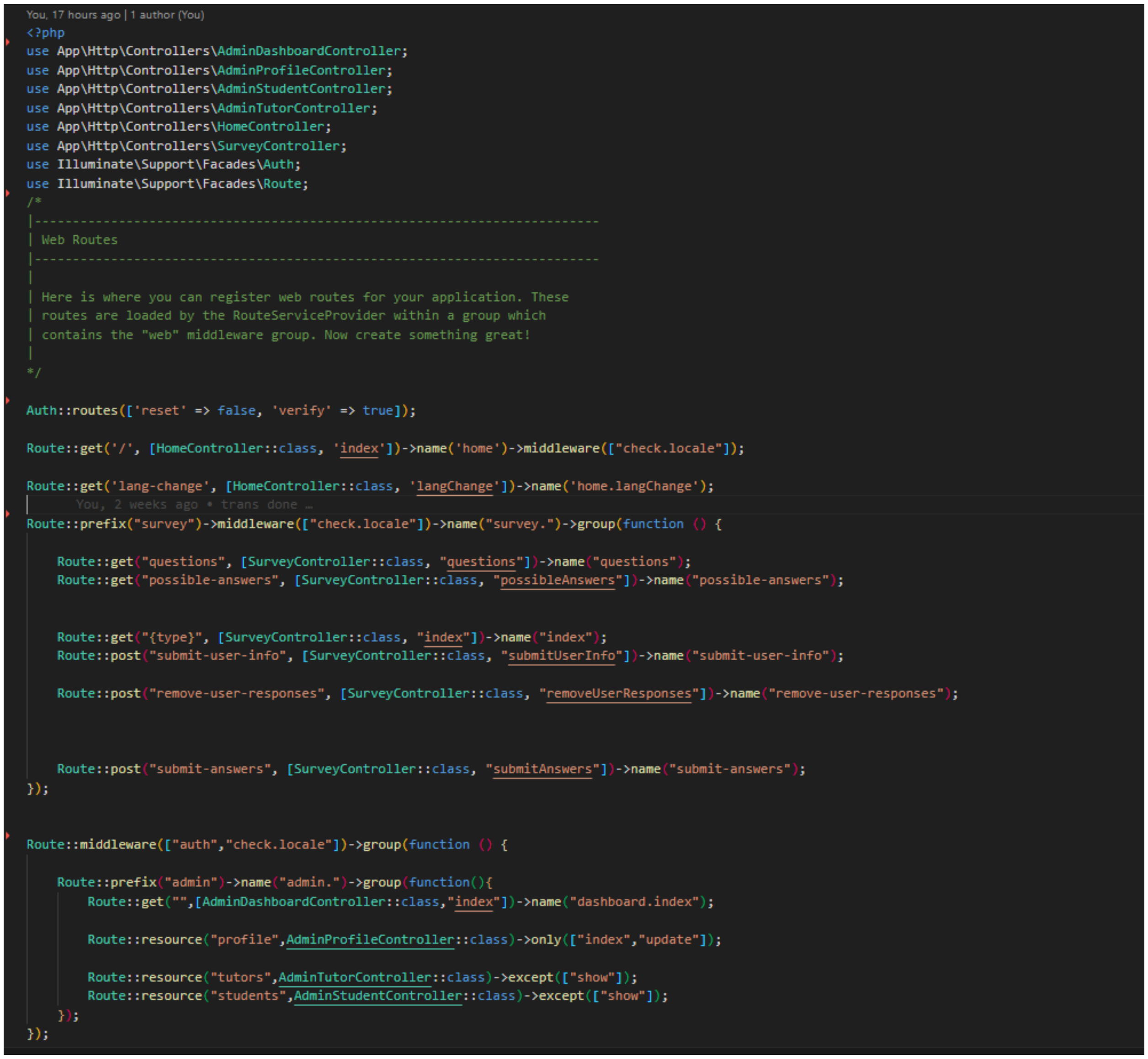
One of the most important service providers in your application is the App\Providers\RouteServiceProvider. This service provider loads the route files contained within your application's routes directory.

Once the application has been bootstrapped and all service providers have been registered, the Request will be handed off to the router for dispatching. The router will dispatch the request to a route or controller, as well as run any route specific middleware.

As discussed above middleware provide a convenient mechanism for filtering or examining HTTP requests entering your application. For example In our application, for admin panel we are using middleware that verifies if the user is authenticated. If the user is not authenticated, the middleware will redirect the user to the login screen. However, if the user is authenticated, the middleware will allow the request to proceed further into the application and redirect the user to the admin panel. Some middleware are assigned to all routes within the application, like those defined in the $middleware property of your HTTP kernel, while some are only assigned to specific routes or route groups. You can learn more about middleware by reading the complete middleware documentation.

If the request passes through all of the matched route's assigned middleware, the route or controller method will be executed and the response returned by the route or controller method will be sent back through the route's chain of middleware.

Here is the screenshot of the routes which we are using in out web application, as we are developing a webs application, therefore you can find the routes in routes/web.php file.



```
You, 17 hours ago | 1 author (You)
<?php
use App\Http\Controllers\AdminDashboardController;
use App\Http\Controllers\AdminProfileController;
use App\Http\Controllers\AdminStudentController;
use App\Http\Controllers\AdminTutorController;
use App\Http\Controllers\HomeController;
use App\Http\Controllers\SurveyController;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Route;
/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Auth::routes(['reset' => false, 'verify' => true]);

Route::get('/', [HomeController::class, 'index'])->name('home')->middleware(["check.locale"]);

Route::get('lang-change', [HomeController::class, 'langChange'])->name('home.langChange');
       You, 2 weeks ago • trans done …
Route::prefix("survey")->middleware(["check.locale"])->name("survey.")->group(function () {

    Route::get("questions", [SurveyController::class, "questions"])->name("questions");
    Route::get("possible-answers", [SurveyController::class, "possibleAnswers"])->name("possible-answers");


    Route::get("{type}", [SurveyController::class, "index"])->name("index");
    Route::post("submit-user-info", [SurveyController::class, "submitUserInfo"])->name("submit-user-info");

    Route::post("remove-user-responses", [SurveyController::class, "removeUserResponses"])->name("remove-user-responses");



    Route::post("submit-answers", [SurveyController::class, "submitAnswers"])->name("submit-answers");
});


Route::middleware(["auth","check.locale"])->group(function () {

    Route::prefix("admin")->name("admin.")->group(function(){
        Route::get("",[AdminDashboardController::class,"index"])->name("dashboard.index");

        Route::resource("profile",AdminProfileController::class)->only(["index","update"]);

        Route::resource("tutors",AdminTutorController::class)->except(["show"]);
        Route::resource("students",AdminStudentController::class)->except(["show"]);
    });
});
```

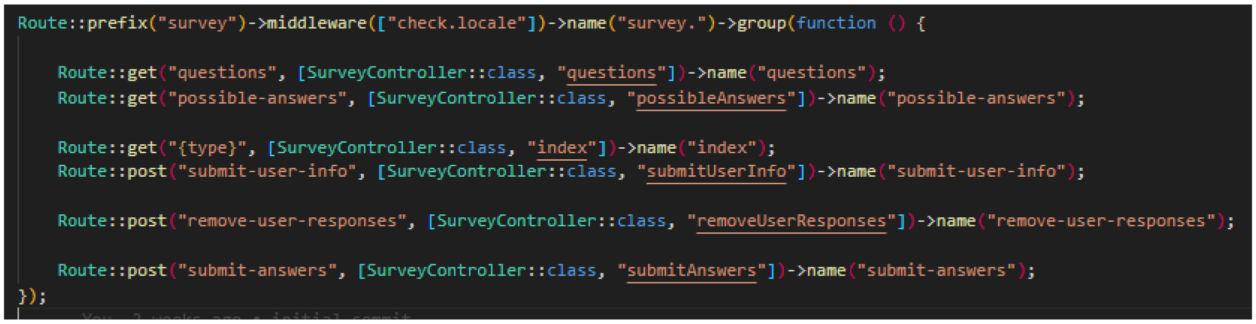
Let me explain the each route separately:

```
Auth::routes(['reset' => false, 'verify' => true]);
```

This route is basically a group of routes for authentication, which includes login, register, email verification, forgot password routes.

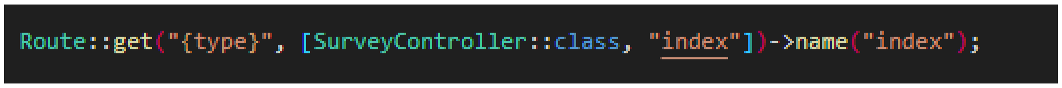
```
Route::get('/', [HomeController::class,
'index'])->name('home')->middleware(["check.locale"]);
```

This route is basically for the landing page, whenever the user will enter only the domain name, the user will be redirected to the home page



```
Route::prefix("survey")->middleware(["check.locale"])->name("survey.")->group(function () {

    Route::get("questions", [SurveyController::class, "questions"])->name("questions");
    Route::get("possible-answers", [SurveyController::class, "possibleAnswers"])->name("possible-answers");

    Route::get("{type}", [SurveyController::class, "index"])->name("index");
    Route::post("submit-user-info", [SurveyController::class, "submitUserInfo"])->name("submit-user-info");

    Route::post("remove-user-responses", [SurveyController::class, "removeUserResponses"])->name("remove-user-responses");

    Route::post("submit-answers", [SurveyController::class, "submitAnswers"])->name("submit-answers");
});
```

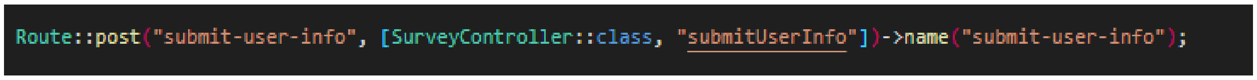
Now when the user will start the questionnaires, the aforementioned routes will be used to surve the users.
The user will either start the questionnaire as a student or as a tutor, for that purpose I have defined the following route:



```
Route::get("{type}", [SurveyController::class, "index"])->name("index");
```

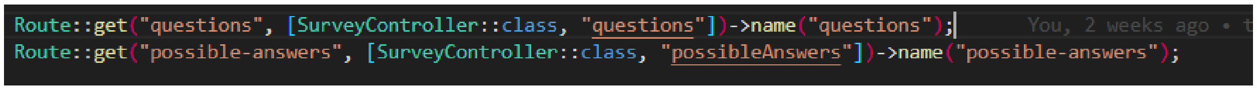
Here the type is a parameter, which will receive either student or a student.

Then when a user submits the information to start the questionnaire, the following route will be used:



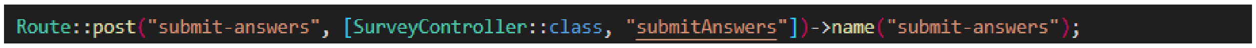
```
Route::post("submit-user-info", [SurveyController::class, "submitUserInfo"])->name("submit-user-info");
```

Then to get the questions with the answers, the following routes are being used:



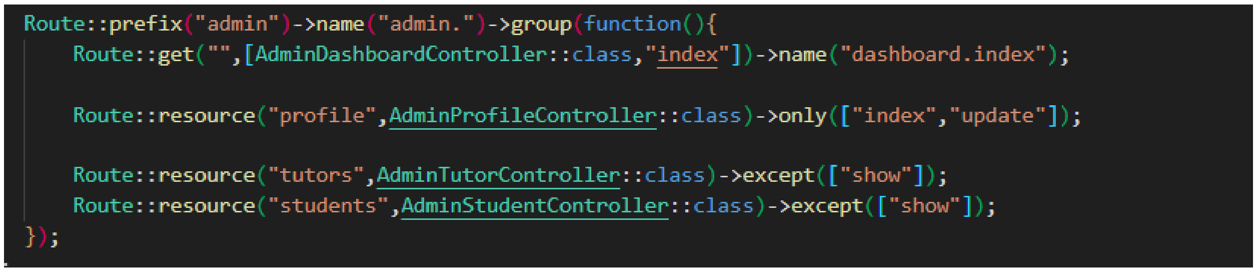
```
Route::get("questions", [SurveyController::class, "questions"])->name("questions");      You, 2 weeks ago • t
Route::get("possible-answers", [SurveyController::class, "possibleAnswers"])->name("possible-answers");
```

And when user submit the questions with the answers, we are utilizing the following route:



```
Route::post("submit-answers", [SurveyController::class, "submitAnswers"])->name("submit-answers");
```

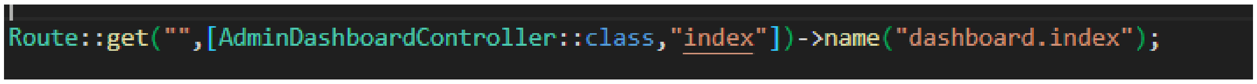
Now lets move to the admin panel routes:



```
Route::prefix("admin")->name("admin.")->group(function(){
    Route::get("",[AdminDashboardController::class,"index"])->name("dashboard.index");

    Route::resource("profile",AdminProfileController::class)->only(["index","update"]);

    Route::resource("tutors",AdminTutorController::class)->except(["show"]);
    Route::resource("students",AdminStudentController::class)->except(["show"]);
});
```

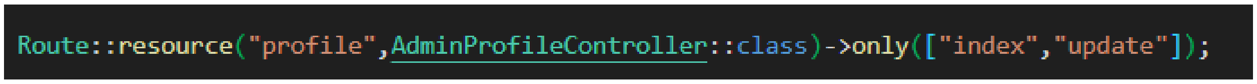
The above routes are being used for the admin panel, prefixes by the admin.

To view the admin dashboard, we have:



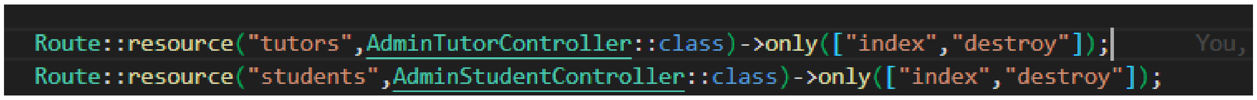
```
Route::get("",[AdminDashboardController::class,"index"])->name("dashboard.index");
```

To view, or update the admin profile information, we have:



```
Route::resource("profile",AdminProfileController::class)->only(["index","update"]);
```

To view, delete the students or tutors questionnaires we have the following routes:



```
Route::resource("tutors",AdminTutorController::class)->only(["index","destroy"]);      You,
Route::resource("students",AdminStudentController::class)->only(["index","destroy"]);
```

Finishing Up:

Once the route or controller method returns a response, the response will travel back outward through the route's middleware, giving the application a chance to modify or examine the outgoing response.

Finally, once the response travels back through the middleware, the HTTP kernel's handle method returns the response object and the index.php file calls the send method on the returned response. The send method sends the response content to the user's web browser. We've finished our journey through the entire Laravel request lifecycle!

Database Scheme:

The detail descriptions of whole database will be sent in a separate PDF file. To view the database you can use the phpmyadmin with the following link and credentials:

http://167.71.242.208/phpmyadmin/
User: root
Password: `grasha`

Directory Structure:

The root Directories:

The App Directory:

The app directory contains the core code of your application. We'll explore this directory in more detail soon; however, almost all of the classes in your application will be in this directory.

The Bootstrap Directory:

The bootstrap directory contains the app.php file which bootstraps the framework. This directory also houses a cache directory which contains framework generated files for performance optimization such as the route and services cache files. You should not typically need to modify any files within this directory.

The Config Directory:

The config directory, as the name implies, contains all of your application's configuration files. For example, in config/app.php we have configure the application default language, application name, timezone e.t.c.

The Database Directory:

The database directory contains your database migrations, model factories, and seeds.

The Lang Directory:

The lang directory houses all of your application's language files. Here I have defined the english and spanish translations of the application.

The Public Directory:

The public directory contains the index.php file, which is the entry point for all requests entering your application and configures autoloading. This directory also houses your assets such as images, JavaScript, and CSS.

The Resources Directory:

The resources directory contains your views (V in MVC) as well as your raw, un-compiled assets such as CSS or JavaScript.

The Routes Direcotory:

The routes directory contains all of the route definitions for your application. By default, several route files are included with Laravel: web.php, api.php, console.php, and channels.php.

The Storage Direcory:

The storage directory contains your logs, compiled Blade templates, file based sessions, file caches, and other files generated by the framework. This directory is segregated into app, framework, and logs directories. The app directory may be used to store any files generated by your application. The framework directory is used to store framework generated files and caches. Finally, the logs directory contains your application's log files.

The storage/app/public directory may be used to store user-generated files, such as profile avatars, that should be publicly accessible. You should create a symbolic link at public/storage which points to this directory. You may create the link using the php artisan storage:link Artisan command.

The Test Directory:

The tests directory contains your automated tests. Example PHPUnit unit tests and feature tests are provided out of the box. Each test class should be suffixed with the word Test. You may run your tests using the phpunit or php vendor/bin/phpunit commands. Or, if you would like a more detailed and beautiful representation of your test results, you may run your tests using the php artisan test Artisan command.

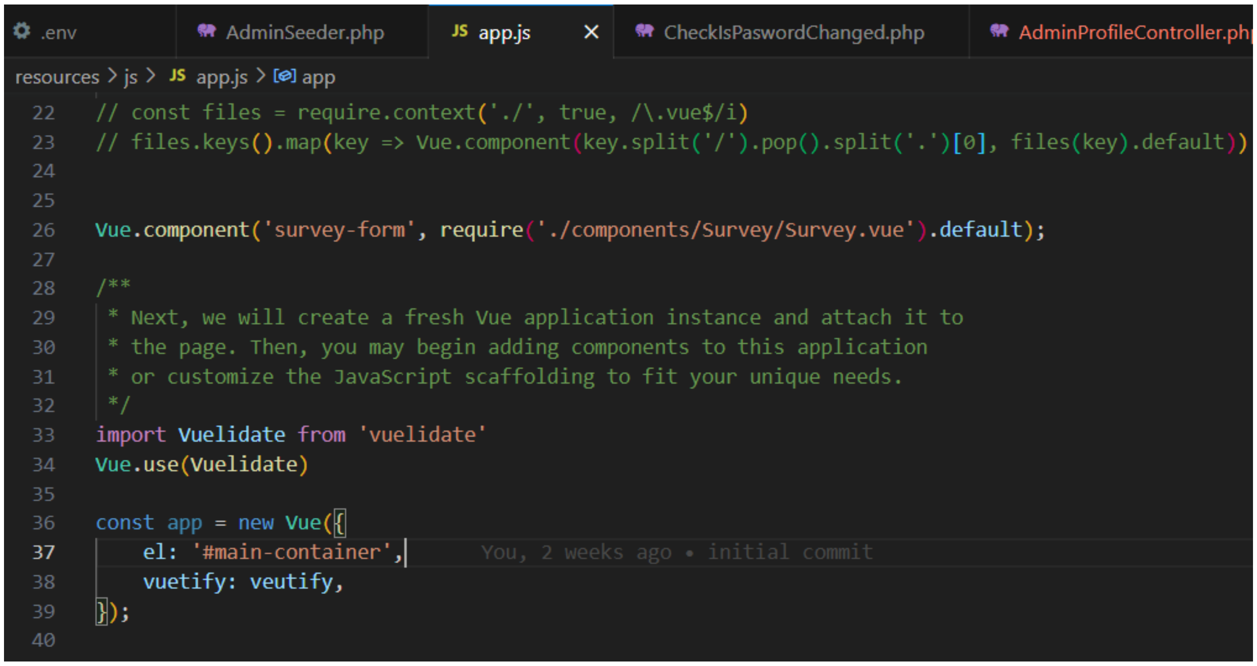
The Vendor Directory:

The vendor directory contains your Composer dependencies.

Laravel Vue Integration:

Laravel comes with default vue integration therefore we dont have to do anything to setup VUE in the application, we just to run the npm install command to install the vue and npm run dev to compile the vue assets.

In resources/js/app.js, we initilize the vue and register our components.


.env
AdminSeeder.php
app.js
CheckIsPaswordChanged.php
AdminProfileController.ph
resources > js > app.js > app

```
// const files = require.context('./', true, /\.vue$/i)
// files.keys().map(key => Vue.component(key.split('/').pop().split('.')[0], files(key).default))


Vue.component('survey-form', require('./components/Survey/Survey.vue').default);

/**
 * Next, we will create a fresh Vue application instance and attach it to
 * the page. Then, you may begin adding components to this application
 * or customize the JavaScript scaffolding to fit your unique needs.
 */
import Vuelidate from 'vuelidate'
Vue.use(Vuelidate)

const app = new Vue({
    el: '#main-container',        You, 2 weeks ago • initial commit
    vuetify: veutify,
});
```

The vue components are defined in resources/js/components folder. In our application I have defined the following two components which are being used in the questionnaire form.

Survey.vue & Question.vue.

Survey.vue is the main component while Question.vue is a child component. Survey.vue contains the user info form where the user is asked to enter their their details to start the questionnaire, once the user submits the information, then the Question.vue component is being called and then rest of the process is done through Question.vue Component.

We are using VUE JS in only the questionnaire form, therefore we have only these two components. Rest of the pages are in blade which is a default template engine of Laravel.