ASHESI UNIVERSITY COLLEGE

BUILDING A WEB-APP SOLUTION FOR A WASTE DISPOSAL SYSTEM IN BAATSONA

APPLIED PROJECT

B.Sc. Computer Science

Kwaku Ofosu-Agyeman

2022

ASHESI UNIVERSITY COLLEGE

BUILDING A WEB-APP SOLUTION FOR A WASTE DISPOSAL SYSTEM IN BAATSONA

APPLIED PROJECT

Applied Project submitted to the Department of Computer Science, Ashesi University College in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Science.

Kwaku Ofosu-Agyeman

2022

DECLARATION

I hereby declare that this Applied Project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

K.O

Candidate's Name:

Kwaku Ofosu-Agyeman

Date:

4/4/2022

I hereby declare that the preparation and presentation of this Applied Project were supervised in accordance with the guidelines on supervision of Applied Projects laid down by Ashesi University.

Supervisor's Signature:

……………………………………………………………………………………….

Supervisor's Name:

Todd Warren

Date:

4/4/2022

Acknowledgements

To my supervisor, whose encouragement and academic advice helped me undertake this Project.

Abstract

Waste management in Ghana is a huge problem that the government faces in stopping. This is evident in the fact that the country was once ranked in the top ten dirtiest countries in the world. This was a huge embarrassment for a country that has one of the fastest-growing economies among developing countries. This problem started with the inadequate supply of bins. It later grew into an inability to process most of the waste. Now it has just become a habit of the citizens to do so whenever they do not see a bin around to dispose of the waste. Current waste disposal solutions implemented in the country involve the use of the *Zoomlion* Company which provides waste disposal services for homes among others to help fight this problem. However, judging from the fact that the country is still known to be one of the dirtiest countries in the world, shows that the company is not able to do enough to help curb this problem. The other solutions involve the use of *aboboya*. A name given to small-time waste collectors that go from house to house collecting waste in areas that require their services. The solution that I propose puts them in some form of umbrella that brings them together under one name and helps them to gain more clients. One of the main reasons for this solution is that some of them would be willing to go out of their way into some areas that they usually do not collect from. This report speaks about the development of a web app that would function as an uber for trash collectors. It would allow the trash collectors to know when and where their services are required as well as let the regular users schedule or call for a pick-up at a time of their convenience. The solution also offers an option through a messaging or USSD platform that would make it easier for users that may not find the web app easy to use.

Table of Content

Contents

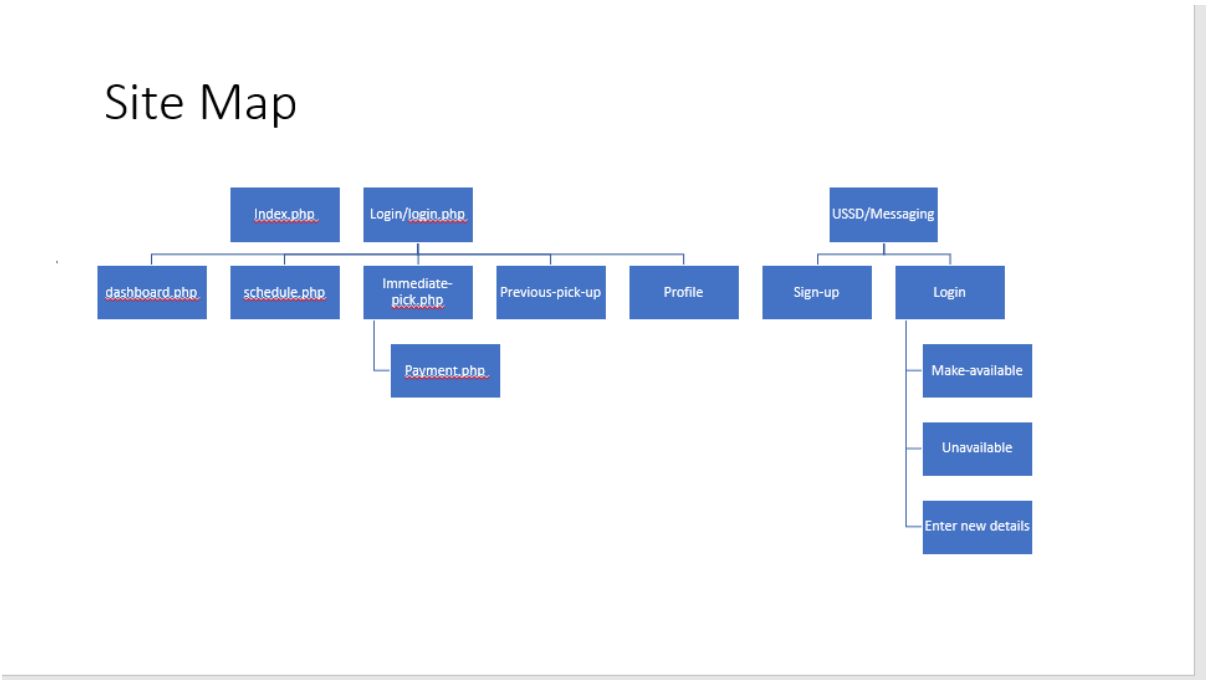
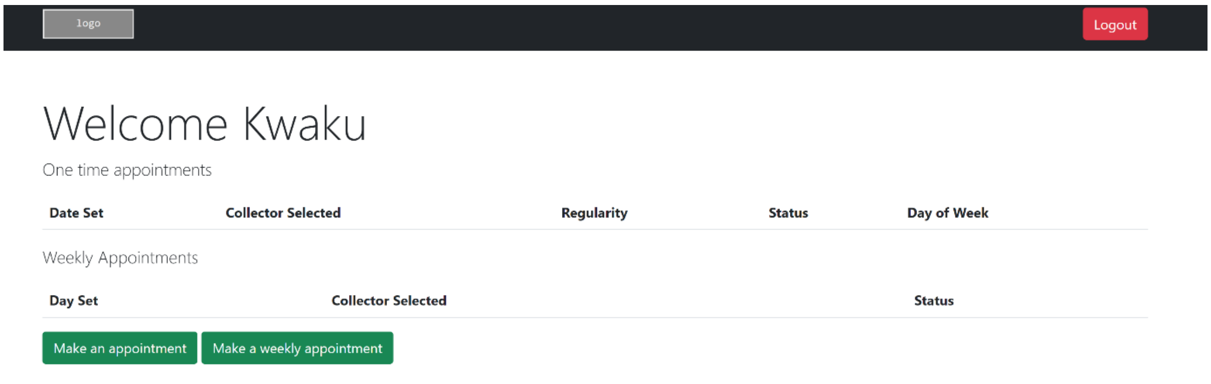


Figure 3.2



Welcome Kwaku

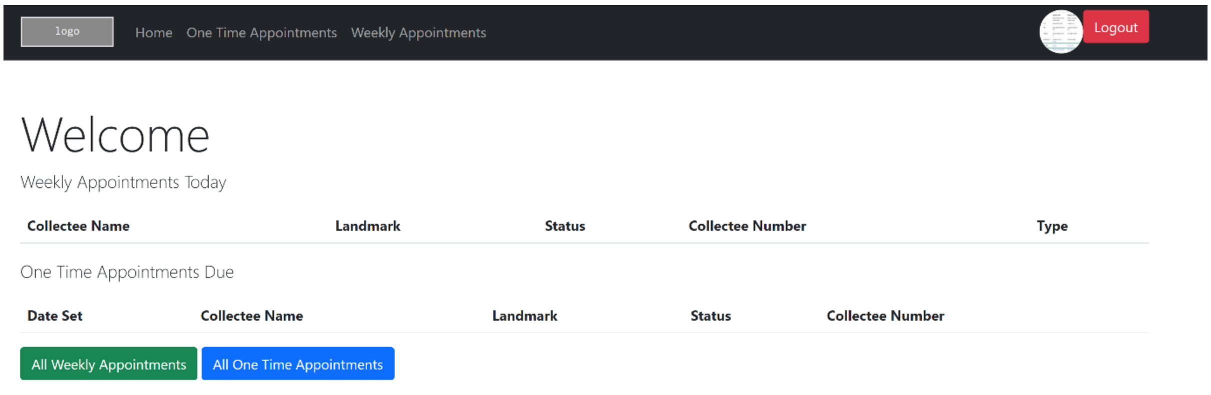
One time appointments

Date Set	Collector Selected	Regularity	Status	Day of Week

Weekly Appointments

Day Set	Collector Selected	Status

Figure 4.2.1



Welcome

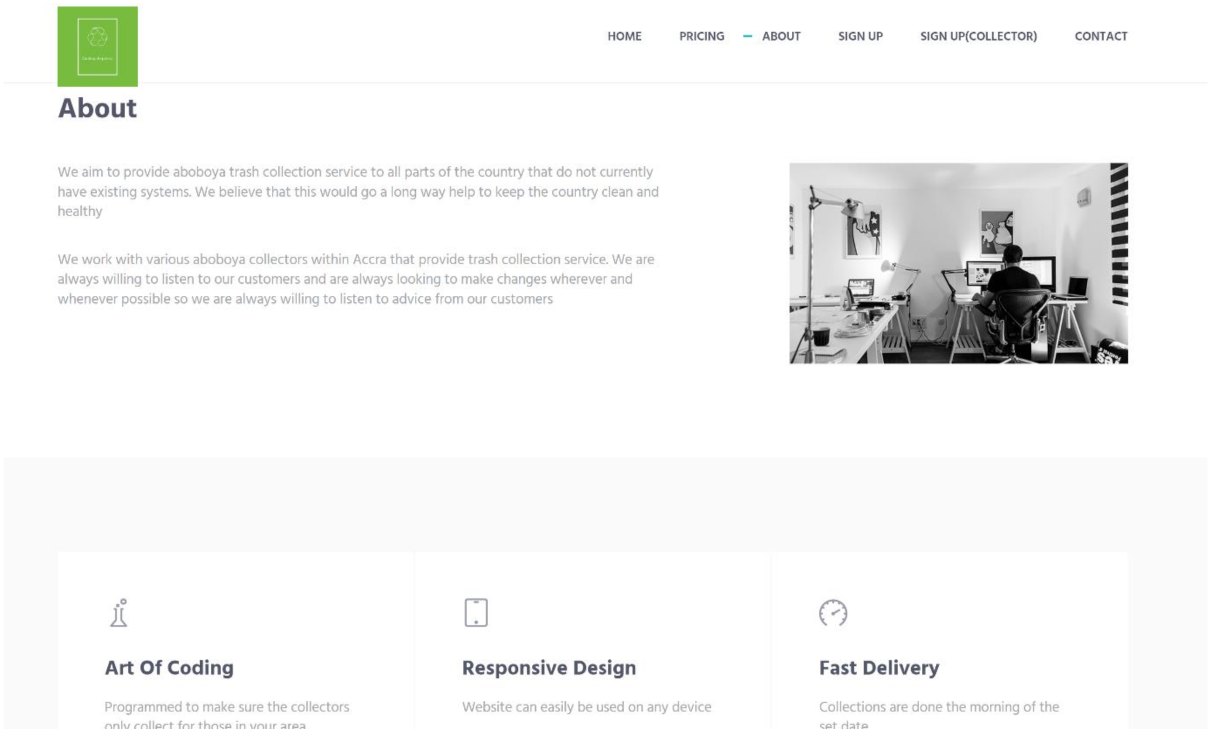
Weekly Appointments Today

Collectee Name	Landmark	Status	Collectee Number	Type

One Time Appointments Due

Date Set	Collectee Name	Landmark	Status	Collectee Number

Figure 4.2.2


HOME PRICING ABOUT SIGN UP SIGN UP(COLLECTOR) CONTACT

About

We aim to provide aboboya trash collection service to all parts of the country that do not currently have existing systems. We believe that this would go a long way help to keep the country clean and healthy

We work with various aboboya collectors within Accra that provide trash collection service. We are always willing to listen to our customers and are always looking to make changes wherever and whenever possible so we are always willing to listen to advice from our customers

Art Of Coding

Programmed to make sure the collectors only collect for those in your area

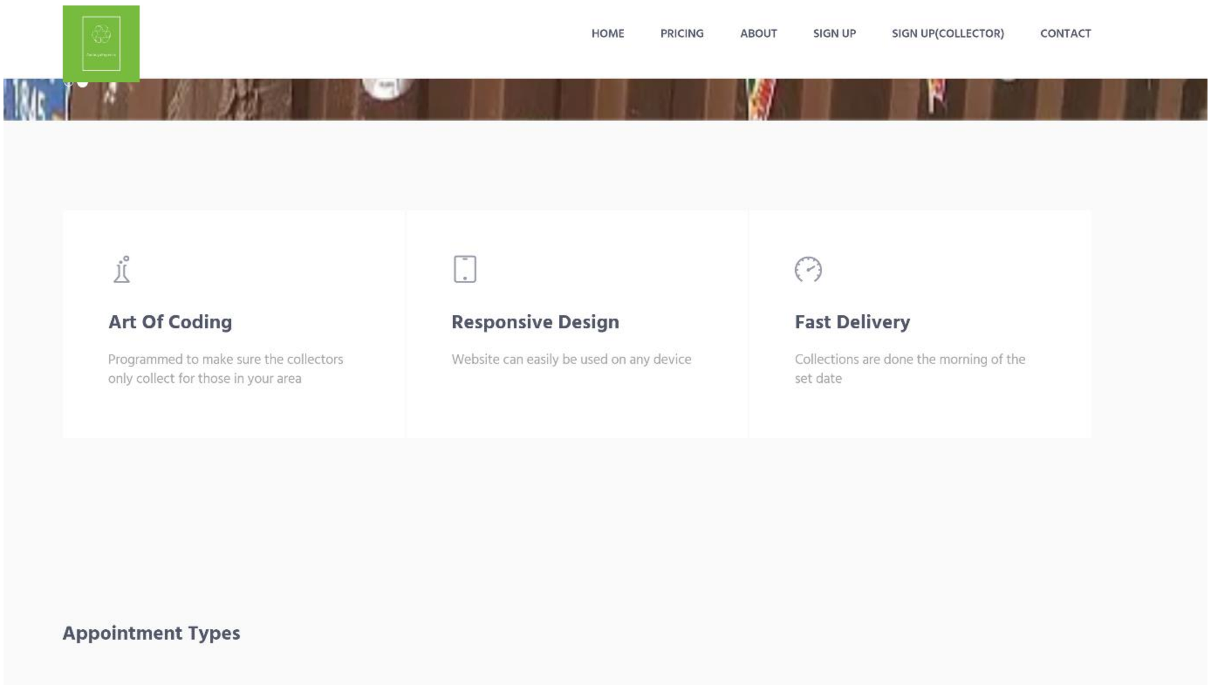
Responsive Design

Website can easily be used on any device

Fast Delivery

Collections are done the morning of the set date

Figure 4.2.3


HOME PRICING ABOUT SIGN UP SIGN UP(COLLECTOR) CONTACT

Art Of Coding

Programmed to make sure the collectors only collect for those in your area

Responsive Design

Website can easily be used on any device

Fast Delivery

Collections are done the morning of the set date

Appointment Types

Figure 4.2.4

Chapter 1: Introduction

The sheer volume of solid waste generated in Ghana is increasing because of the continuous population density in urban areas where solid waste is generated the most in the country. “Provision of sanitation and garbage collection services is an important and yet challenging issue in the rapidly growing cities of developing countries, with significant human health and environmental sustainability implications.” [5] It is common knowledge that cities around the world would generate the highest volumes of solid waste in their countries however the amount produced in cities in Ghana is much more than the government’s current waste management plan can handle. This is evident with Ghana being ranked seventh of the dirtiest countries in the world. [Graphic] Over 10,000 tons of solid waste is generated in Ghana every day, however, only about 10% of this amount is properly disposed of at dumping sites [3, 4]. Despite the amount being so low, the amount spent by the government to have this is in no way small. It is no secret that the government of Ghana spends millions of cedis every year on waste disposal and management in the country however the fact that it is still a problem shows just how poorly planned the systems that they spend so much money on and how little of an effect it has in solving the solution.

This is especially true for waste collection in Ghana. *Zoom lion* Ghana is the largest waste collection, management, and disposal company in the country. The government pays the company enormous amounts to help in keeping the country clean. The company is the largest collector of solid waste in the country and is one of the leading companies in the country that help the government to keep the country clean. The company handles all types of waste. They also happen to have certain parts of the country where they do door-to-door collections of domestic waste. They pick up the trash once a week at the doorstep of the customers within the area. This makes it easy for the customers since they do not have to always call them before they come to pick it up. Despite this, coupled with the voluntary work the company already does to help keep the country cleaner, it still made the list of one of the dirtiest countries in the world. The citizens of the country also make the situation worse with the constant littering in their surroundings. People that live in cities in the country are known to have developed the habit of littering. This makes it even worse since the government ends up paying companies such as ‘Zoom lion’ to handle these situations that could have easily been avoided if they properly disposed of their waste.

This paper proposes a solution to helping with the collection of waste from the homes of people in urban areas. The solution would allow people in the community to book a date and time for their domestic waste to be collected at least once every week. It also allows the users to call for a pick-up on any other day that they may accumulate substantial amounts of waste due to large gatherings. Since the collectors may not be conversant with using current technology, the alert to the collectors would be sent via a text message which can be received by anyone if they have a mobile phone no matter how old.

Chapter 2: Requirements Analysis

Functional Requirement	Description
1	The system will allow the user(customer) to call for an immediate trash pick-up
2	The system will allow the user(customer) to schedule a trash pick-up to be done as many times a week as they would like.
3	The system will allow the user(customer) to view their previous pick-ups and current package plans for pick-up as well as any data about them
4	The system will inform the collectors when a user(customer) wants to pick-up up the required details they would need for easy service delivery
5	The system will allow registration through USSD or a messaging platform for collectors
6	The system will allow users(customers) to make reports on the improper provision of service.

Non-functional Requirements	Description
1	The system will be secure. The system will ensure some form of authentication to help protect user data
2	The system will be recoverable. The system will save the user database in a backup database to ensure the user data is kept if the original ever gets corrupted or damaged.

3	The system will be useable. The system will be easy-to-use and understand by all users of it.
4	The system will be reliable. The system will be consistent in its performance.

The majority of the functional requirements were obtained from interviews conducted with a few potential customers as well as the collector that was interviewed. One of the requests made by the collectors was allowing collectors to register for the system through USSD platforms since not all the collectors would be able to use the website easily. Security and reliability non-functional requirements were the two main things that all the customers required that the system would provide. "The customer is the primary focus class for the proposed MobApp where the main goal of this MobApp is to make customers able to send and receive shipments by submitting online requests" [1] Similarly, the focus of this paper is the customer being able to book appointments and receive the appointment service through making an online request. The customers also requested that they be allowed to report collectors for malicious practices so that they may be warned or kicked off the app. The system being easy to use was a recommendation made by a lecturer because not all the users of the system would be conversant with the current technology so it would be best for it to be as simplistic as possible whiles offering as many functionalities to those who would like to use them as well.

In creating this solution, several steps must be taken with careful consideration into the design of the system since it must be simple enough for anyone to be able to use. The prototype of the system will be made in Figma and tested with customers of different age ranges as well as collectors. The major purpose of this is to test to make sure that the customers and collectors will be able to easily interact with the solution as well as make sure it is easy to understand and navigate so that they will be able to gain the full benefit of the solution.

In designing the solution, color schemes and flashes would be considered a top priority to make sure that they do not affect any potential users, who may have diseases or disabilities relating to this, and be affected by it anyway. The text should also be easy to read and simplistic to make sure it is understandable and easy to read since the main users of the app would be those in the older generation

Chapter 3: System Architecture and Design

The solution for this project is a web app as well as a USSD or messaging platform. The web app would be implemented using the Laravel PHP framework for the backend with HTML and CSS being used for the frontend. The database would be structured using MySQL.

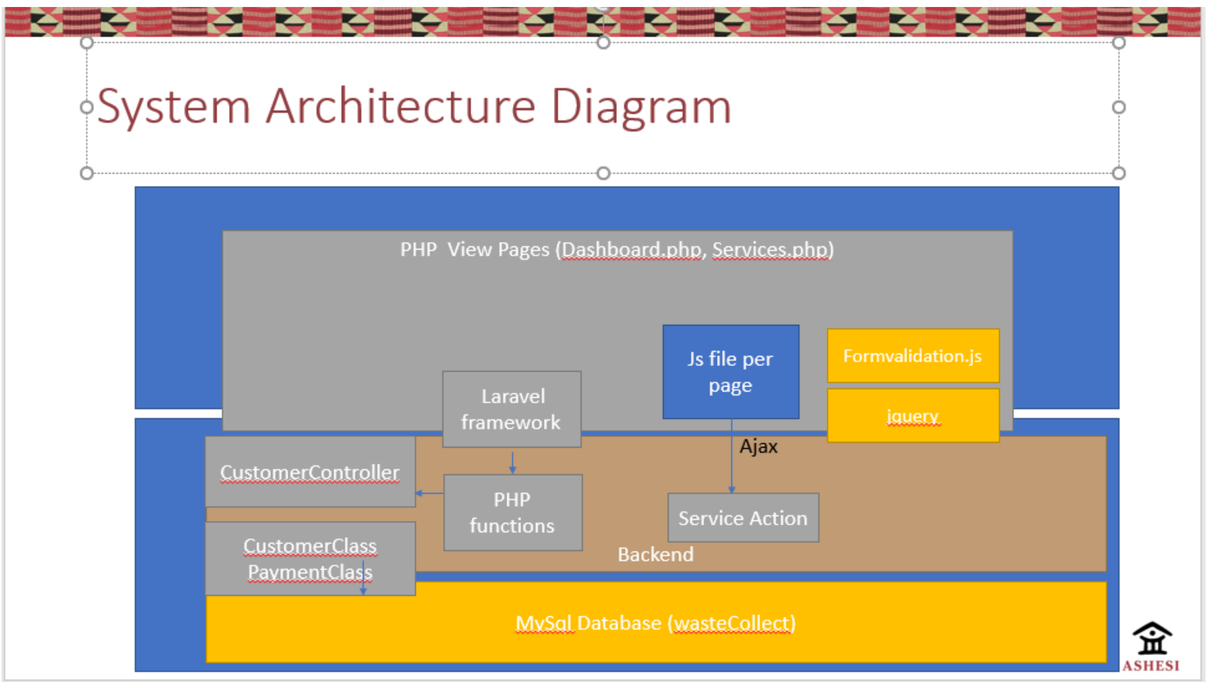


Figure 3.1

The image (figure 3.1) above shows a rough sketch of the system structure of the web app. The JavaScript files are there for extra validation of any form details the user may enter to prevent any malicious data from being added to the database.

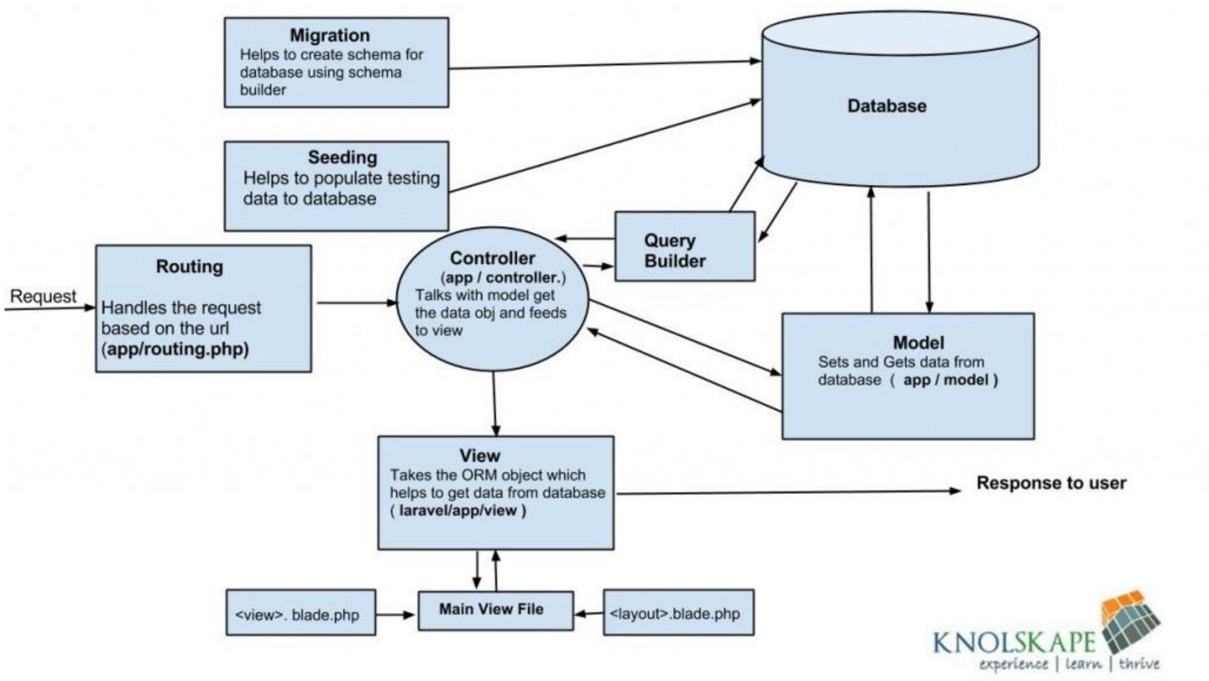

Migration
Helps to create schema for database using schema builder
Database
Seeding
Helps to populate testing data to database
Query Builder
Routing
Handles the request based on the url (app/routing.php)
Request
Controller (app / controller.)
Talks with model get the data obj and feeds to view
Model
Sets and Gets data from database (app / model)
View
Takes the ORM object which helps to get data from database (laravel/app/view)
Response to user
<view>. blade.php
Main View File
<layout>.blade.php
KNOLSKAPE
experience | learn | thrive

Figure 3.3

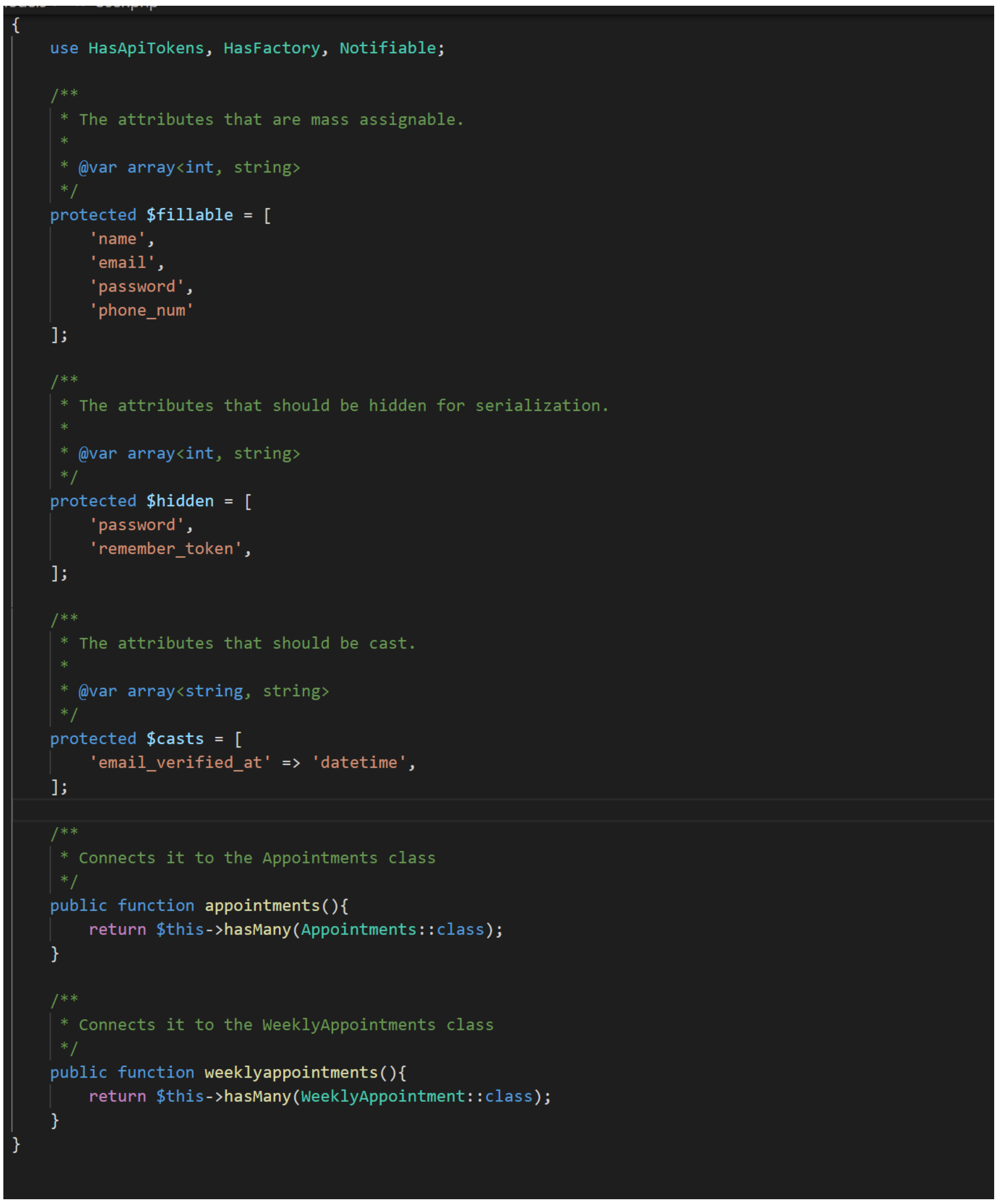
The web app also uses the MVC structure which makes it easy for unit testing to be used on it. The MVC structure contains classes such as the customer class and appointment class. There is also a class for the collectors and a double authentication method would be used in Laravel to make ensure security. This allows the system to distinguish between a regular user, a collector, and an administrator. The USSD version is added using an API that would provide such services. Figure 3.3 above shows the MVC structure used by Laravel web apps. The routing allows the developer to define functions that make the website much smoother and easier to handle. The database would be structured to cater to the needs of the customers that would be making requests. There is a table for the customer details, a table for the services to help keep track of the available services offered, a table tracking each customer's order, to allow a customer to view their previous pick-up orders as well as a table for the report details. Some tables keep records of the areas that we have collectors for as well as areas that ensure that the collectors would be able to pick the areas within which they would collect. Many to one and one to many relationships exist between these tables which allow the use of two or more tables to get data to be displayed to the users for an improved experience.

After this, a prototype of the web app and USSD interface was built and tested on the collectors and customers to find their responses and how the design can be edited. This is later followed by the implementation of the solution. The USSD or messaging platform would then be built first to have the user interact with it before the web app is built which would also be linked to the USSD or messaging platform

Chapter 4: Implementation

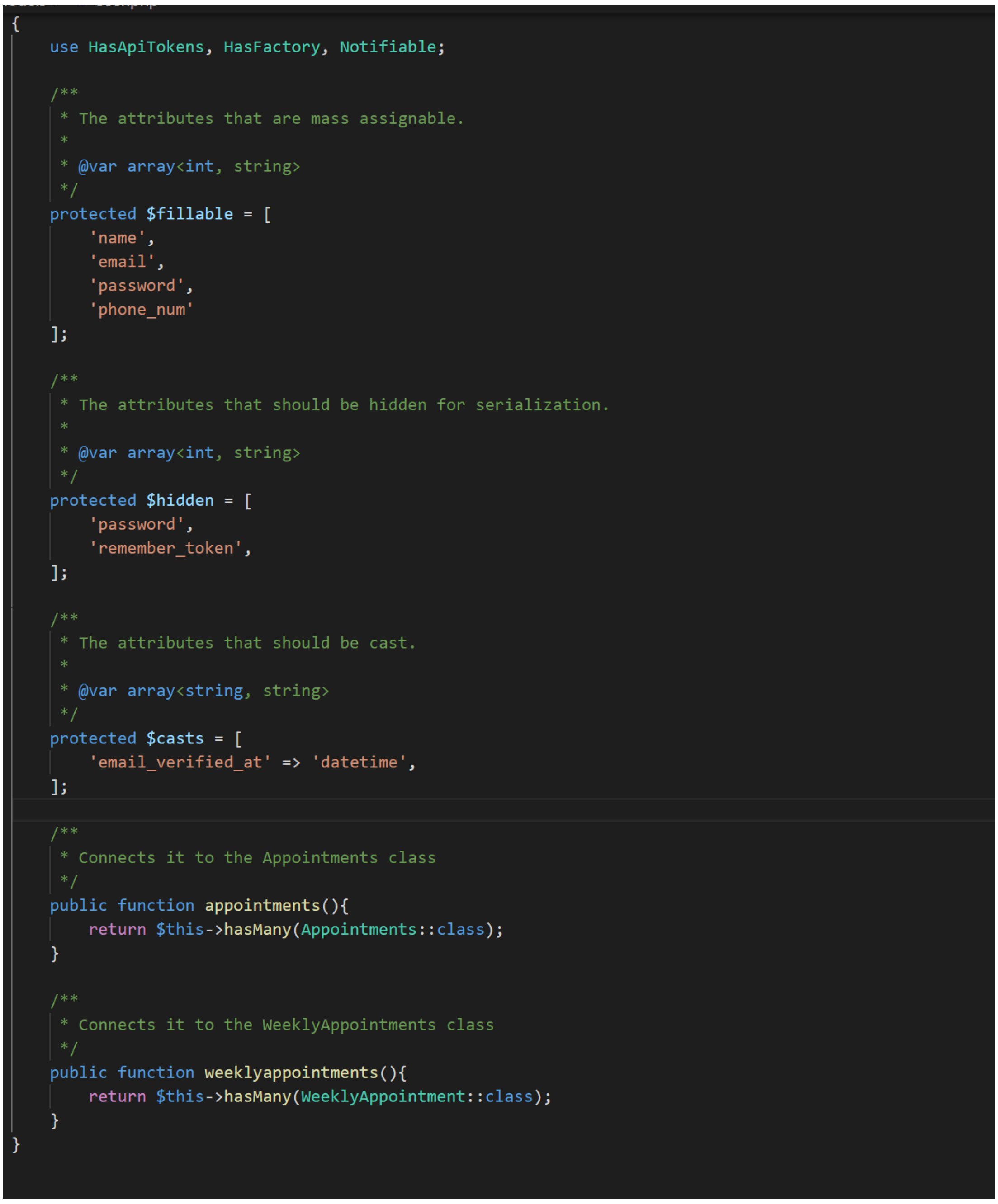
4.1 Back-end Development

A Laravel PHP web app has been built. Laravel is a PHP framework used for web development in modern-day. It contains several features that make it easier for back-end developers especially when building the system. The website created allows the user to sign in and make either a weekly appointment or a one-time appointment. The user is required to provide information on the location of the house for each appointment they make to make it easier for the collectors to find their way to the house.



```php
{
    use HasApiTokens, HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'name',
        'email',
        'password',
        'phone_num'
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

    /**
     * Connects it to the Appointments class
     */
    public function appointments(){
        return $this->hasMany(Appointments::class);
    }

    /**
     * Connects it to the WeeklyAppointments class
     */
    public function weeklyappointments(){
        return $this->hasMany(WeeklyAppointment::class);
    }
}
```

Figure 4.1.1

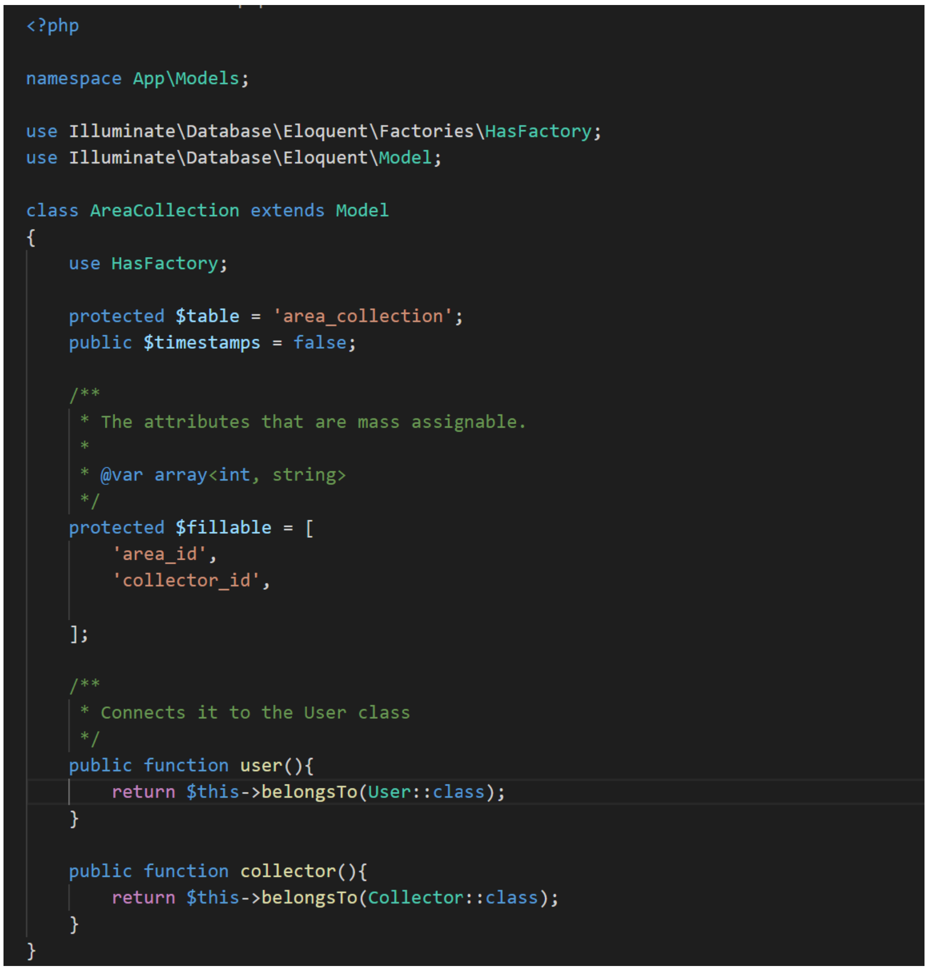


```php
{
    use HasApiTokens, HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'name',
        'email',
        'password',
        'phone_num'
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

    /**
     * Connects it to the Appointments class
     */
    public function appointments(){
        return $this->hasMany(Appointments::class);
    }

    /**
     * Connects it to the WeeklyAppointments class
     */
    public function weeklyappointments(){
        return $this->hasMany(WeeklyAppointment::class);
    }
}
```

Figure 4.1.2

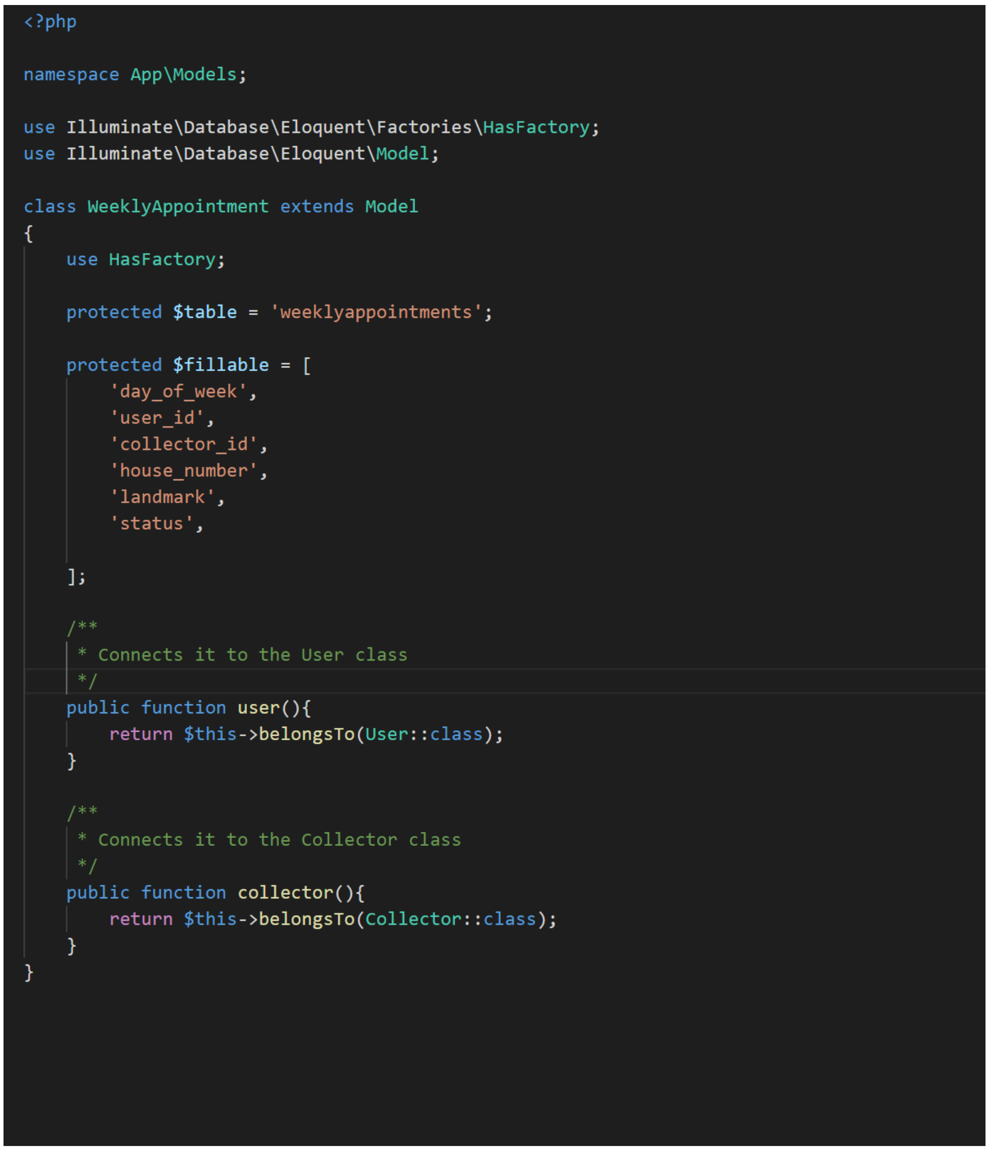
Figures 4.1.1 and 4.1.2 show the user and collector classes. The protected variable *fillable* holds the column names of all the columns in the database table for the user and collector. The *hidden* variable makes sure the password attribute as well as the remember token is kept hidden from the user. This class contains the user's details. The functions *appointments* and *weekly appointments* use Laravel functions that define a one-to-many relationship with the Appointments and *Weekly Appointments* classes. This relationship allows for certain conventions when getting data from the database. This is shown in the front-end sub-chapter.

The *casts* variable assigns the *email_verified_at* to a *DateTime* attribute.



```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class AreaCollection extends Model
{
    use HasFactory;

    protected $table = 'area_collection';
    public $timestamps = false;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'area_id',
        'collector_id',

    ];

    /**
     * Connects it to the User class
     */
    public function user(){
        return $this->belongsTo(User::class);
    }

    public function collector(){
        return $this->belongsTo(Collector::class);
    }
}
```

Figure 4.1.3

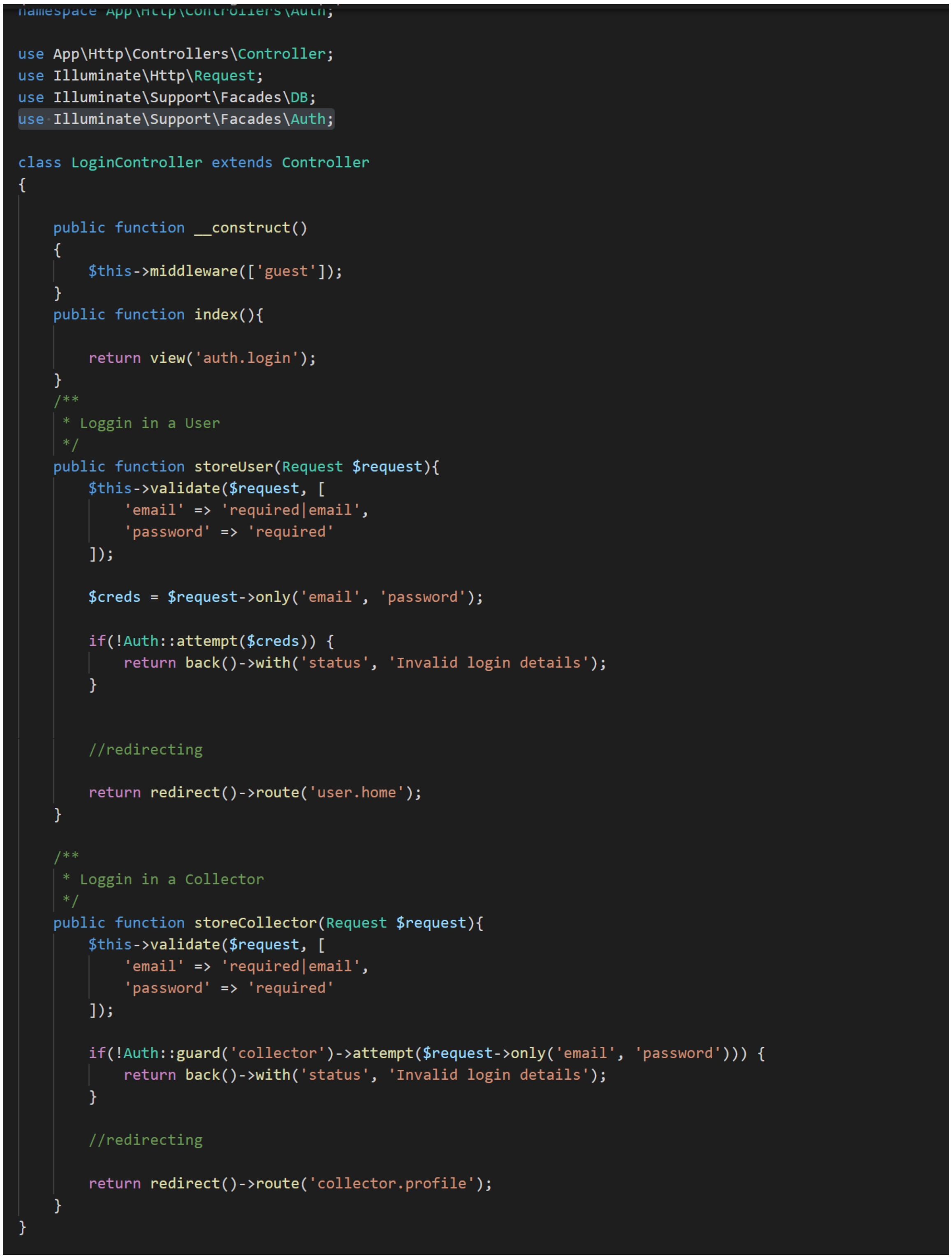


```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class WeeklyAppointment extends Model
{
    use HasFactory;

    protected $table = 'weeklyappointments';

    protected $fillable = [
        'day_of_week',
        'user_id',
        'collector_id',
        'house_number',
        'landmark',
        'status',

    ];

    /**
     * Connects it to the User class
     */
    public function user(){
        return $this->belongsTo(User::class);
    }

    /**
     * Connects it to the Collector class
     */
    public function collector(){
        return $this->belongsTo(Collector::class);
    }
}
```

Figure 4.1.4

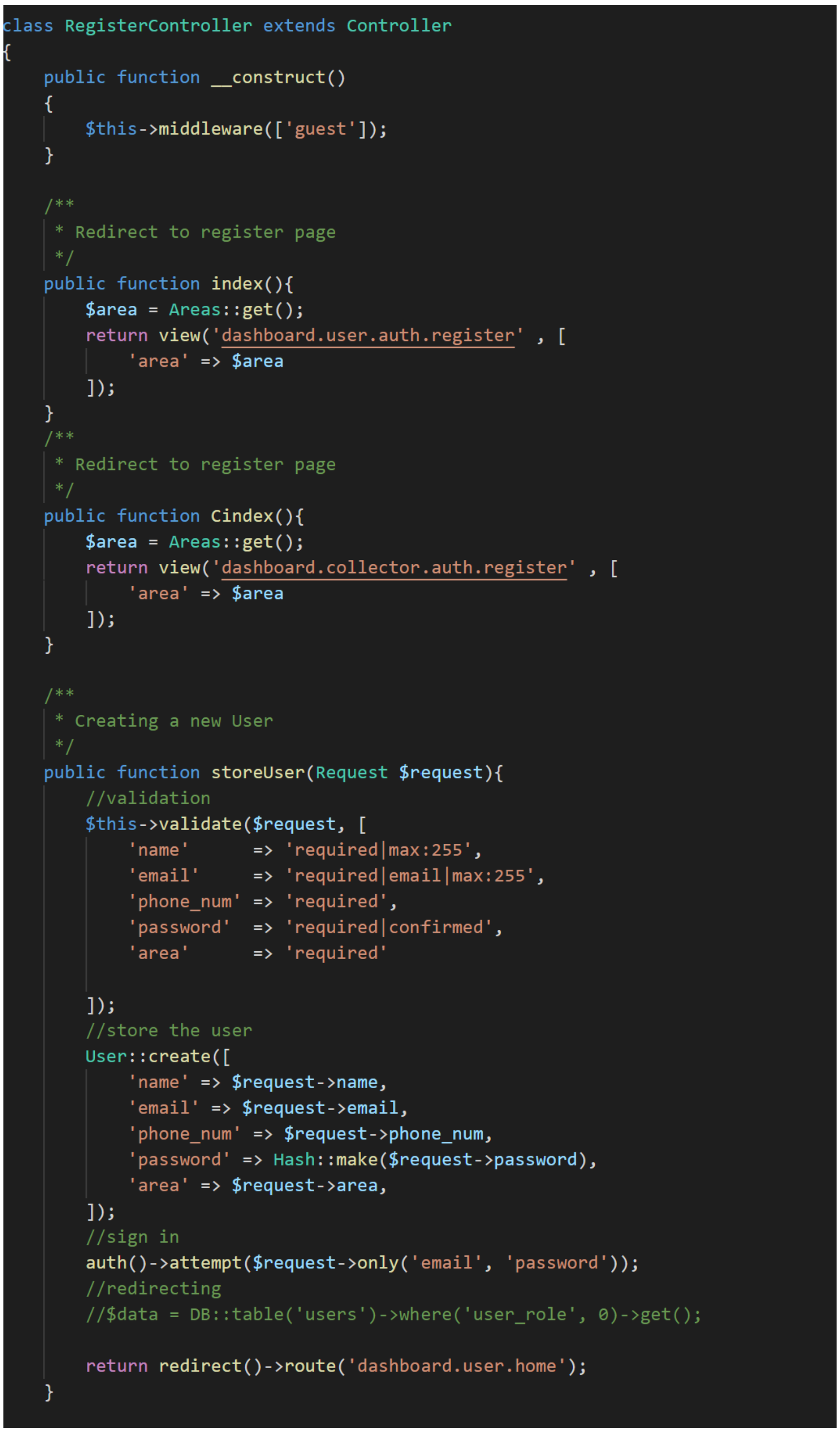
Figures 4.1.3 and 4.1.4 show the *appointment* and *weeklyappointment* class. The user and collector functions in these classes establish many-to-one relationships with the user and collector classes respectively.

The user is also able to pick the collector that they prefer to collect for them based on their names. For convenience, the user is only able to choose a collector that works within the area that they live in. On the collector side, they would be able to see the appointments they have due that day as well as the ones coming up. They can also view their previous one-time appointments for confirmation on anything that they need. The collector can see the details provided by the user when making the appointment as well as their phone number in case there are any problems or a need for enquiries.



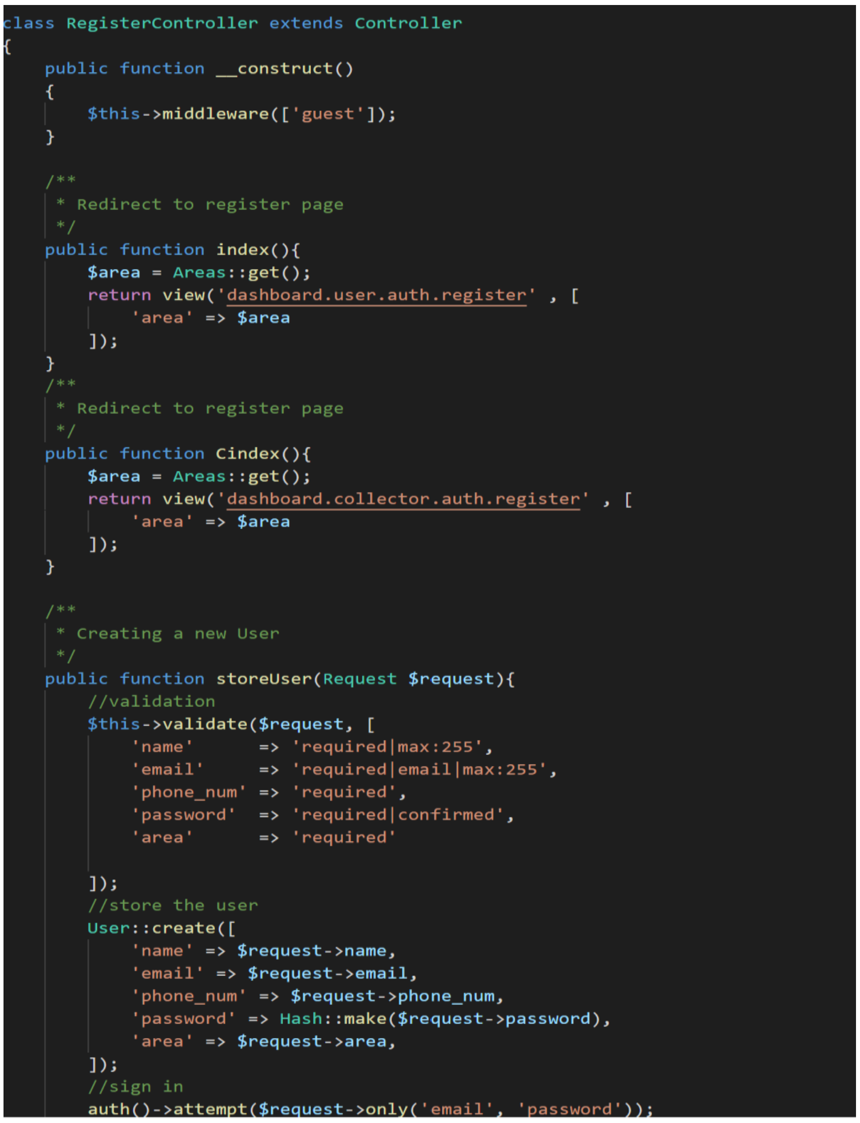
```php
namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Auth;

class LoginController extends Controller
{

    public function __construct()
    {
        $this->middleware(['guest']);
    }
    public function index(){

        return view('auth.login');
    }
    /**
     * Loggin in a User
     */
    public function storeUser(Request $request){
        $this->validate($request, [
            'email' => 'required|email',
            'password' => 'required'
        ]);

        $creds = $request->only('email', 'password');

        if(!Auth::attempt($creds)) {
            return back()->with('status', 'Invalid login details');
        }


        //redirecting

        return redirect()->route('user.home');
    }

    /**
     * Loggin in a Collector
     */
    public function storeCollector(Request $request){
        $this->validate($request, [
            'email' => 'required|email',
            'password' => 'required'
        ]);

        if(!Auth::guard('collector')->attempt($request->only('email', 'password'))) {
            return back()->with('status', 'Invalid login details');
        }

        //redirecting

        return redirect()->route('collector.profile');
    }
}
```

Figure 4.1.5



```
class RegisterController extends Controller
{
    public function __construct()
    {
        $this->middleware(['guest']);
    }

    /**
     * Redirect to register page
     */
    public function index(){
        $area = Areas::get();
        return view('dashboard.user.auth.register' , [
            'area' => $area
        ]);
    }
    /**
     * Redirect to register page
     */
    public function Cindex(){
        $area = Areas::get();
        return view('dashboard.collector.auth.register' , [
            'area' => $area
        ]);
    }

    /**
     * Creating a new User
     */
    public function storeUser(Request $request){
        //validation
        $this->validate($request, [
            'name'      => 'required|max:255',
            'email'     => 'required|email|max:255',
            'phone_num' => 'required',
            'password'  => 'required|confirmed',
            'area'      => 'required'

        ]);
        //store the user
        User::create([
            'name' => $request->name,
            'email' => $request->email,
            'phone_num' => $request->phone_num,
            'password' => Hash::make($request->password),
            'area' => $request->area,
        ]);
        //sign in
        auth()->attempt($request->only('email', 'password'));
        //redirecting
        //$data = DB::table('users')->where('user_role', 0)->get();

        return redirect()->route('dashboard.user.home');
    }
```

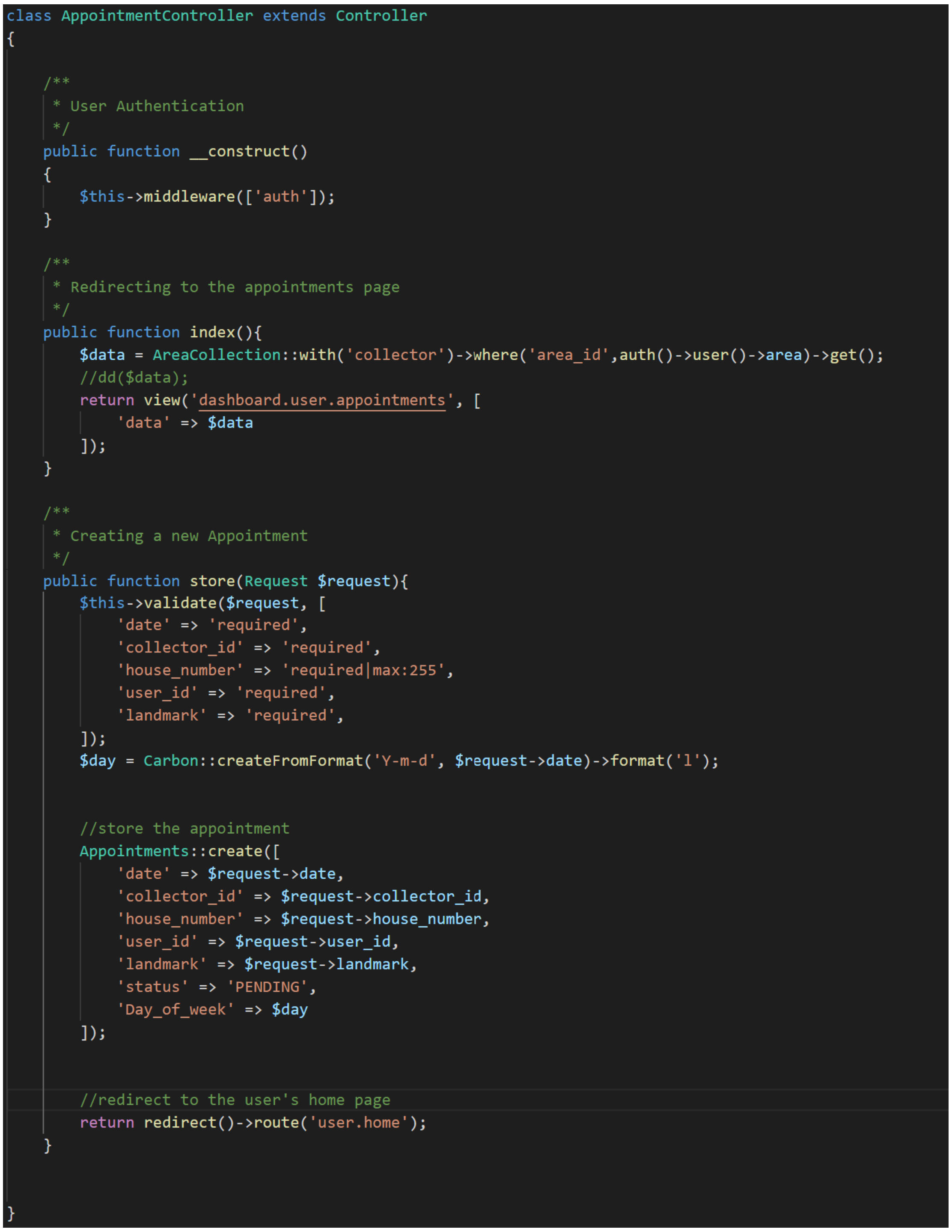
Figure 4.1.6



```php
class RegisterController extends Controller
{
    public function __construct()
    {
        $this->middleware(['guest']);
    }

    /**
     * Redirect to register page
     */
    public function index(){
        $area = Areas::get();
        return view('dashboard.user.auth.register' , [
            'area' => $area
        ]);
    }
    /**
     * Redirect to register page
     */
    public function Cindex(){
        $area = Areas::get();
        return view('dashboard.collector.auth.register' , [
            'area' => $area
        ]);
    }

    /**
     * Creating a new User
     */
    public function storeUser(Request $request){
        //validation
        $this->validate($request, [
            'name'      => 'required|max:255',
            'email'     => 'required|email|max:255',
            'phone_num' => 'required',
            'password'  => 'required|confirmed',
            'area'      => 'required'

        ]);
        //store the user
        User::create([
            'name' => $request->name,
            'email' => $request->email,
            'phone_num' => $request->phone_num,
            'password' => Hash::make($request->password),
            'area' => $request->area,
        ]);
        //sign in
        auth()->attempt($request->only('email', 'password'));
```

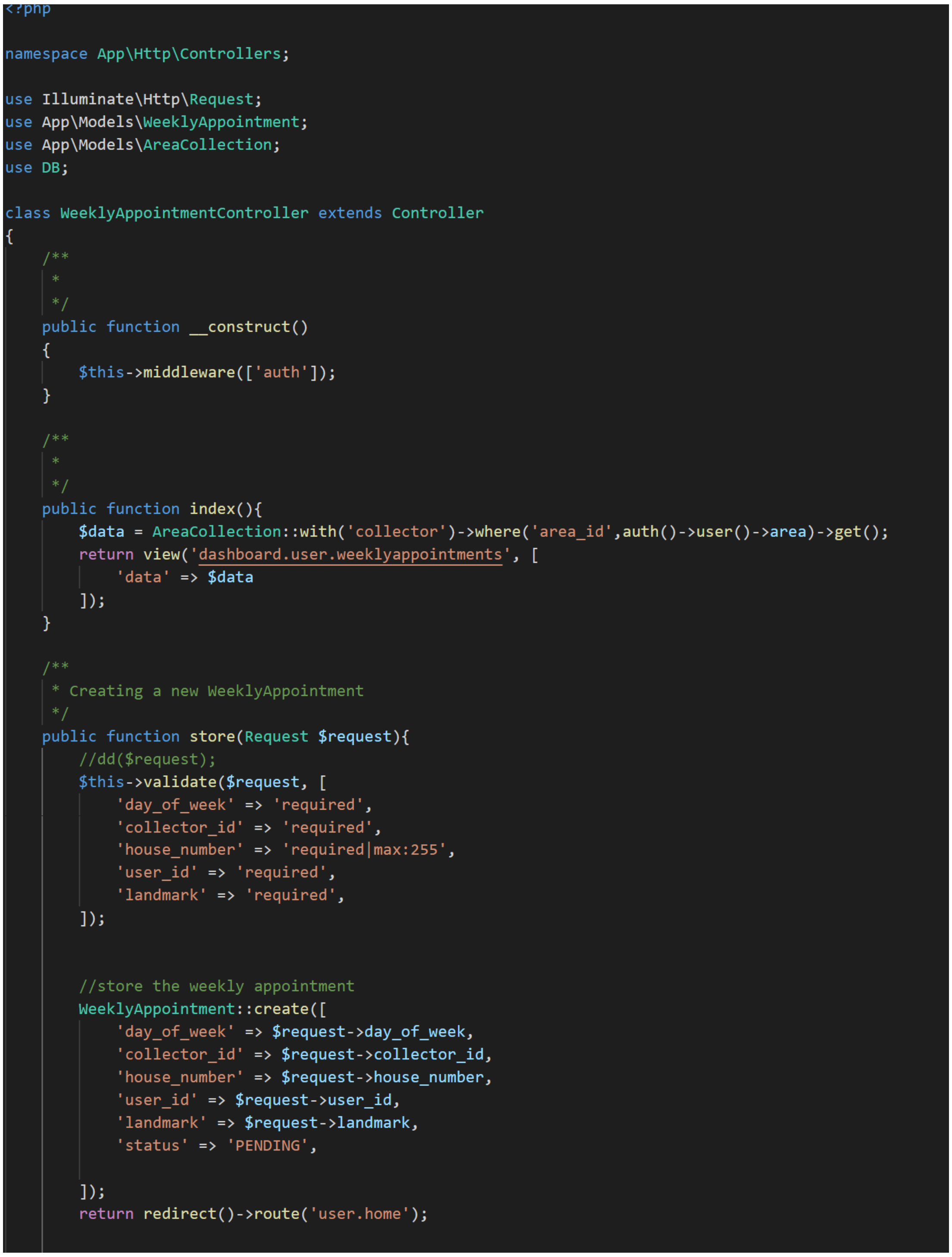
Figure 4.1.7

The images above (Figures 4.1.5, 4.1.6 and 4.1.7) show the controllers that help with the user and collector authentication. The register controller does backend validation of the form's details. If the form's details pass this test, the user or collector's details are added to the database. The login controller checks to see if the user's email and password are within the database before redirecting them to the home page. The index function sends the user to the registration page with the areas from the database which would allow them to choose an area that would have a collector.



```php
class AppointmentController extends Controller
{

    /**
     * User Authentication
     */
    public function __construct()
    {
        $this->middleware(['auth']);
    }

    /**
     * Redirecting to the appointments page
     */
    public function index(){
        $data = AreaCollection::with('collector')->where('area_id',auth()->user()->area)->get();
        //dd($data);
        return view('dashboard.user.appointments', [
            'data' => $data
        ]);
    }

    /**
     * Creating a new Appointment
     */
    public function store(Request $request){
        $this->validate($request, [
            'date' => 'required',
            'collector_id' => 'required',
            'house_number' => 'required|max:255',
            'user_id' => 'required',
            'landmark' => 'required',
        ]);
        $day = Carbon::createFromFormat('Y-m-d', $request->date)->format('l');


        //store the appointment
        Appointments::create([
            'date' => $request->date,
            'collector_id' => $request->collector_id,
            'house_number' => $request->house_number,
            'user_id' => $request->user_id,
            'landmark' => $request->landmark,
            'status' => 'PENDING',
            'Day_of_week' => $day
        ]);


        //redirect to the user's home page
        return redirect()->route('user.home');
    }


}
```

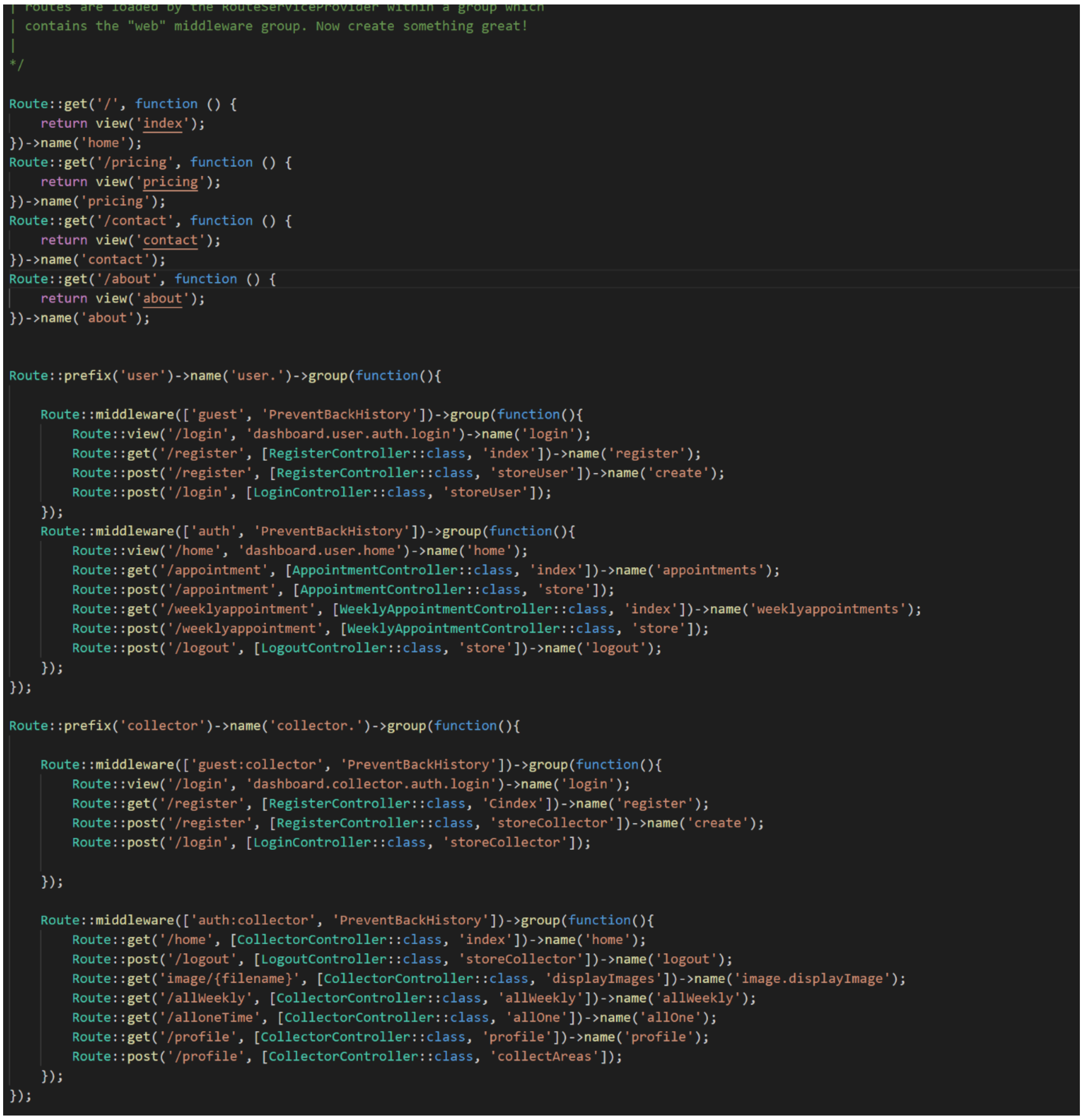
Figure 4.1.7



```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\WeeklyAppointment;
use App\Models\AreaCollection;
use DB;

class WeeklyAppointmentController extends Controller
{
    /**
     *
     */
    public function __construct()
    {
        $this->middleware(['auth']);
    }

    /**
     *
     */
    public function index(){
        $data = AreaCollection::with('collector')->where('area_id',auth()->user()->area)->get();
        return view('dashboard.user.weeklyappointments', [
            'data' => $data
        ]);
    }

    /**
     * Creating a new WeeklyAppointment
     */
    public function store(Request $request){
        //dd($request);
        $this->validate($request, [
            'day_of_week' => 'required',
            'collector_id' => 'required',
            'house_number' => 'required|max:255',
            'user_id' => 'required',
            'landmark' => 'required',
        ]);


        //store the weekly appointment
        WeeklyAppointment::create([
            'day_of_week' => $request->day_of_week,
            'collector_id' => $request->collector_id,
            'house_number' => $request->house_number,
            'user_id' => $request->user_id,
            'landmark' => $request->landmark,
            'status' => 'PENDING',

        ]);
        return redirect()->route('user.home');
```

Figure 4.1.8

Figures 4.1.7 and 4.1.8 show the *appointmentcontroller* and *weeklyappointmentcontroller*. These controller classes store the appointments once the request is made by the user for their creation. The *__contstruct()* function prevents guests from using functions on this page.



```
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::get('/', function () {
    return view('index');
})->name('home');
Route::get('/pricing', function () {
    return view('pricing');
})->name('pricing');
Route::get('/contact', function () {
    return view('contact');
})->name('contact');
Route::get('/about', function () {
    return view('about');
})->name('about');


Route::prefix('user')->name('user.')->group(function(){

    Route::middleware(['guest', 'PreventBackHistory'])->group(function(){
        Route::view('/login', 'dashboard.user.auth.login')->name('login');
        Route::get('/register', [RegisterController::class, 'index'])->name('register');
        Route::post('/register', [RegisterController::class, 'storeUser'])->name('create');
        Route::post('/login', [LoginController::class, 'storeUser']);
    });
    Route::middleware(['auth', 'PreventBackHistory'])->group(function(){
        Route::view('/home', 'dashboard.user.home')->name('home');
        Route::get('/appointment', [AppointmentController::class, 'index'])->name('appointments');
        Route::post('/appointment', [AppointmentController::class, 'store']);
        Route::get('/weeklyappointment', [WeeklyAppointmentController::class, 'index'])->name('weeklyappointments');
        Route::post('/weeklyappointment', [WeeklyAppointmentController::class, 'store']);
        Route::post('/logout', [LogoutController::class, 'store'])->name('logout');
    });
});

Route::prefix('collector')->name('collector.')->group(function(){

    Route::middleware(['guest:collector', 'PreventBackHistory'])->group(function(){
        Route::view('/login', 'dashboard.collector.auth.login')->name('login');
        Route::get('/register', [RegisterController::class, 'Cindex'])->name('register');
        Route::post('/register', [RegisterController::class, 'storeCollector'])->name('create');
        Route::post('/login', [LoginController::class, 'storeCollector']);

    });

    Route::middleware(['auth:collector', 'PreventBackHistory'])->group(function(){
        Route::get('/home', [CollectorController::class, 'index'])->name('home');
        Route::post('/logout', [LogoutController::class, 'storeCollector'])->name('logout');
        Route::get('image/{filename}', [CollectorController::class, 'displayImages'])->name('image.displayImage');
        Route::get('/allWeekly', [CollectorController::class, 'allWeekly'])->name('allWeekly');
        Route::get('/alloneTime', [CollectorController::class, 'allOne'])->name('allOne');
        Route::get('/profile', [CollectorController::class, 'profile'])->name('profile');
        Route::post('/profile', [CollectorController::class, 'collectAreas']);
    });
});
```

Figure 4.1.9

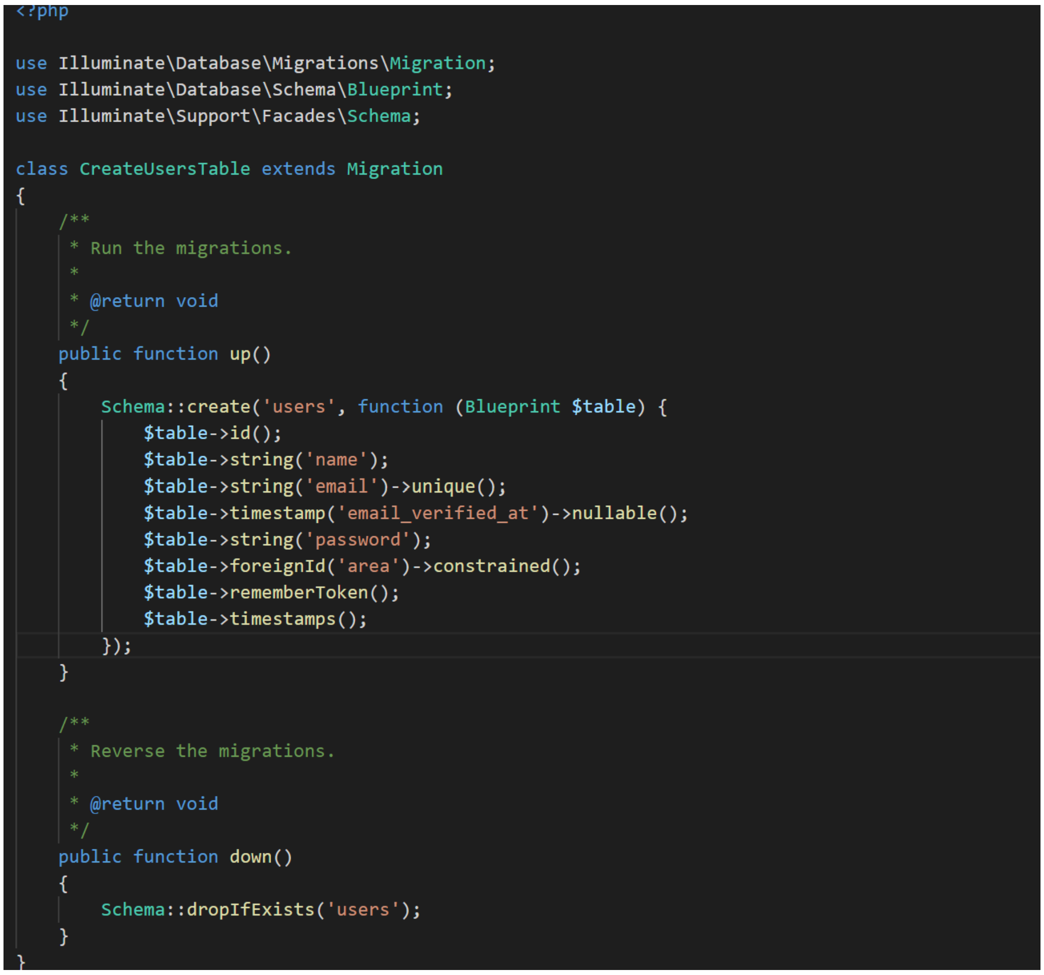
Figure 4.1.9 shows the routing methods used by the website for navigating through the pages. In this case, the prefix is used since I implemented a multiple authentication method that allows me to have different *auth* users. The *auth* model is a Laravel eloquent model that gets the user's data and allows you the check if the user is authenticated. These are the major classes and functions that the system uses in the backend.

4.2 Front-end development

For the front-end of the website, the bootstrap framework is used. Bootstrap is a front-end framework that helps with the styling of HTML attributes much easier than using regular CSS. It allows the user to enter keywords in classes that represent already created versions of what the user wants. JavaScript is also used in the system for form validation and for displaying images in certain situations. Several self-styled pages use CSS since bootstrap did not contain the preferred styling that would be used for the page. Figure 4.2.1 shows the user side of the system. The user can view the appointments they currently have and whether it is due that day. They can also book either a one-time or weekly appointment. Figure 4.2.2 shows the collector side of the website where they can view all past appointments as well as those due that day, be it a one-time appointment or a weekly appointment. Figure 4.2.1 shows a depiction of a user that has logged in. The user can view their collections due that day as well as allow the user to make one-time or weekly appointments. There is also a logout button that allows the user to log out from the account. "…shows the schedule of garbage collector interface that presents the day and the time of the garbage collector duty." [2] Figure 4.2.2 shows the home page of a collector. The collector can view previous records of their collections. They can also view all their collections due that day and those that are soon to be completed.

4.3 Database development

The database is developed using Laravel on the MySQL database. Laravel has a feature in the system called migrations. Figure 3.4 shows the entity-relationship diagram that was used to build the database.



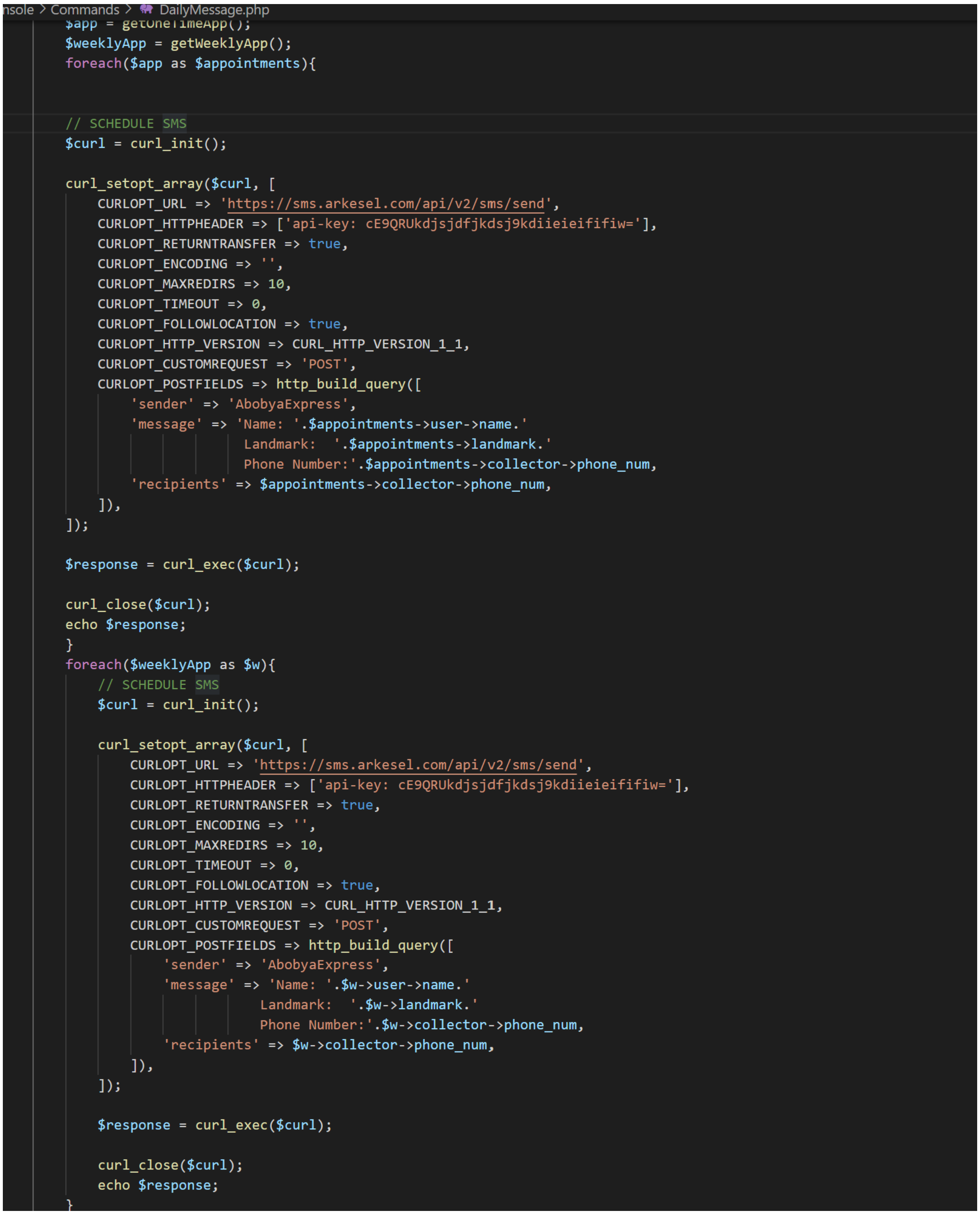
```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->foreignId('area')->constrained();
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

Figure 4.3.1

The migrations allow the developer to create migrations files that when migrated would become the tables in the database. Figure 4.3.1 shows a migration class for the user table. The class contains functions that create and delete the table. It also has conventional functions that allow the developer to define the attributes of the columns in the table. There are tables for appointments, weekly appointments, and tables for each type of user on the system. The table system would allow for easy access to data on the back-end side of the system.

4.4 Messaging platform

The messaging system would send messages to all collectors every day each morning to alert them of the people they have to collect for on that day. The messaging platform would be developed using Arkesel API. The API allows the user to develop a messaging system of their own that would send messages whenever they want.



```php
$app = getOneTimeApp();
$weeklyApp = getWeeklyApp();
foreach($app as $appointments){


// SCHEDULE SMS
$curl = curl_init();

curl_setopt_array($curl, [
    CURLOPT_URL => 'https://sms.arkesel.com/api/v2/sms/send',
    CURLOPT_HTTPHEADER => ['api-key: cE9QRUkdjsjdfjkdsj9kdiieieififiw='],
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_ENCODING => '',
    CURLOPT_MAXREDIRS => 10,
    CURLOPT_TIMEOUT => 0,
    CURLOPT_FOLLOWLOCATION => true,
    CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
    CURLOPT_CUSTOMREQUEST => 'POST',
    CURLOPT_POSTFIELDS => http_build_query([
        'sender' => 'AbobyaExpress',
        'message' => 'Name: '.$appointments->user->name.'
                      Landmark:  '.$appointments->landmark.'
                      Phone Number:'.$appointments->collector->phone_num,
        'recipients' => $appointments->collector->phone_num,
    ]),
]);

$response = curl_exec($curl);

curl_close($curl);
echo $response;
}
foreach($weeklyApp as $w){
    // SCHEDULE SMS
    $curl = curl_init();

    curl_setopt_array($curl, [
        CURLOPT_URL => 'https://sms.arkesel.com/api/v2/sms/send',
        CURLOPT_HTTPHEADER => ['api-key: cE9QRUkdjsjdfjkdsj9kdiieieififiw='],
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_ENCODING => '',
        CURLOPT_MAXREDIRS => 10,
        CURLOPT_TIMEOUT => 0,
        CURLOPT_FOLLOWLOCATION => true,
        CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
        CURLOPT_CUSTOMREQUEST => 'POST',
        CURLOPT_POSTFIELDS => http_build_query([
            'sender' => 'AbobyaExpress',
            'message' => 'Name: '.$w->user->name.'
                        Landmark:  '.$w->landmark.'
                        Phone Number:'.$w->collector->phone_num,
            'recipients' => $w->collector->phone_num,
        ]),
    ]);

    $response = curl_exec($curl);

    curl_close($curl);
    echo $response;
}
```

Figure 4.4.1

Figure 4.4.1 shows the *Arkesel* function that uses the API to schedule messages to be sent to the collectors. The messages are sent every morning using Laravel’s scheduler that allows the user to schedule at which they would like a particular command to be run.

Chapter 5: Testing

5.1 User Testing:

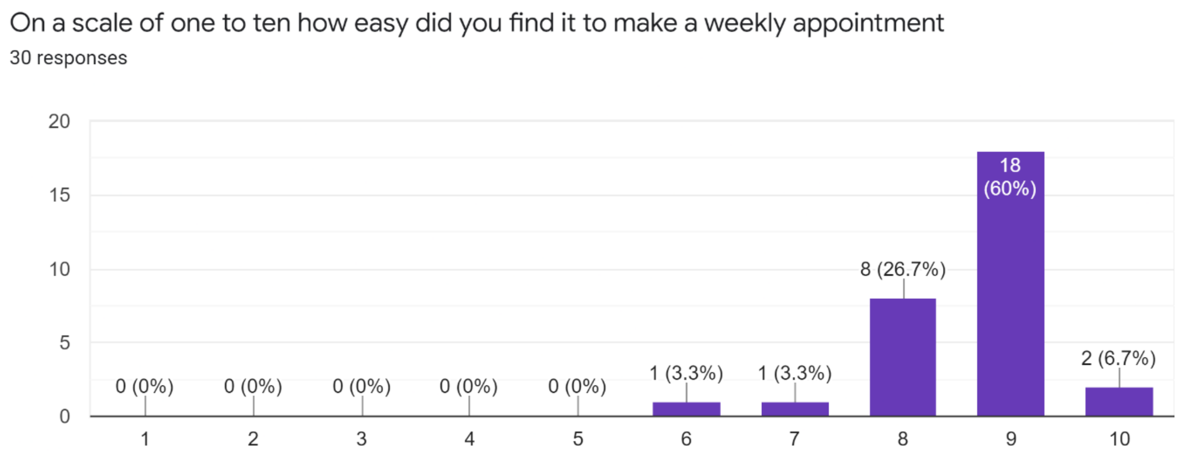
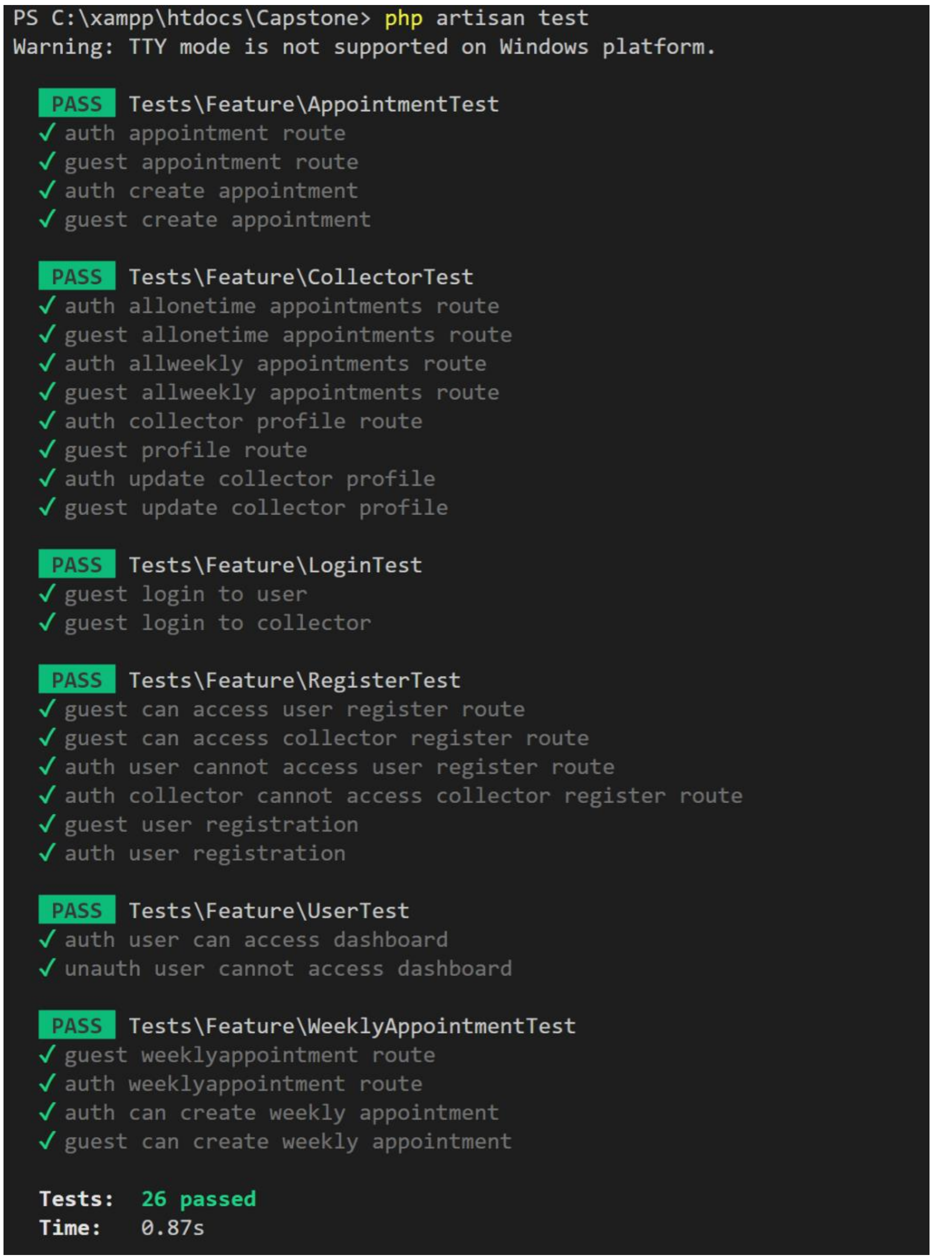


Figure 5.1.1

A form was given to every user that was shown a sample of the website for them to fill to determine whether the system built would be useable by the thirty potential users. Though there were very few users that claimed that it was perfect, the majority of them acknowledged that it was near perfect. There was also one that claimed that the system was good but could be made better and their views would be taken into consideration when developing the system in the future.

5.2 Unit Testing

To test the system, Laravel's in-built unit testing tool was used to test the various features of the website. Most of the features that were tested for were the functional requirements however the non-functional requirements were tested for as well. A connection to Apache and MySQL must be established on your system to run this test. This can be easily done using XAMPP. To view the test results, run the command *php artisan test*.



```
PS C:\xampp\htdocs\Capstone> php artisan test
Warning: TTY mode is not supported on Windows platform.

   PASS  Tests\Feature\AppointmentTest
  ✓ auth appointment route
  ✓ guest appointment route
  ✓ auth create appointment
  ✓ guest create appointment

   PASS  Tests\Feature\CollectorTest
  ✓ auth allonetime appointments route
  ✓ guest allonetime appointments route
  ✓ auth allweekly appointments route
  ✓ guest allweekly appointments route
  ✓ auth collector profile route
  ✓ guest profile route
  ✓ auth update collector profile
  ✓ guest update collector profile

   PASS  Tests\Feature\LoginTest
  ✓ guest login to user
  ✓ guest login to collector

   PASS  Tests\Feature\RegisterTest
  ✓ guest can access user register route
  ✓ guest can access collector register route
  ✓ auth user cannot access user register route
  ✓ auth collector cannot access collector register route
  ✓ guest user registration
  ✓ auth user registration

   PASS  Tests\Feature\UserTest
  ✓ auth user can access dashboard
  ✓ unauth user cannot access dashboard

   PASS  Tests\Feature\WeeklyAppointmentTest
  ✓ guest weeklyappointment route
  ✓ auth weeklyappointment route
  ✓ auth can create weekly appointment
  ✓ guest can create weekly appointment

  Tests:    26 passed
  Time:     0.87s
```

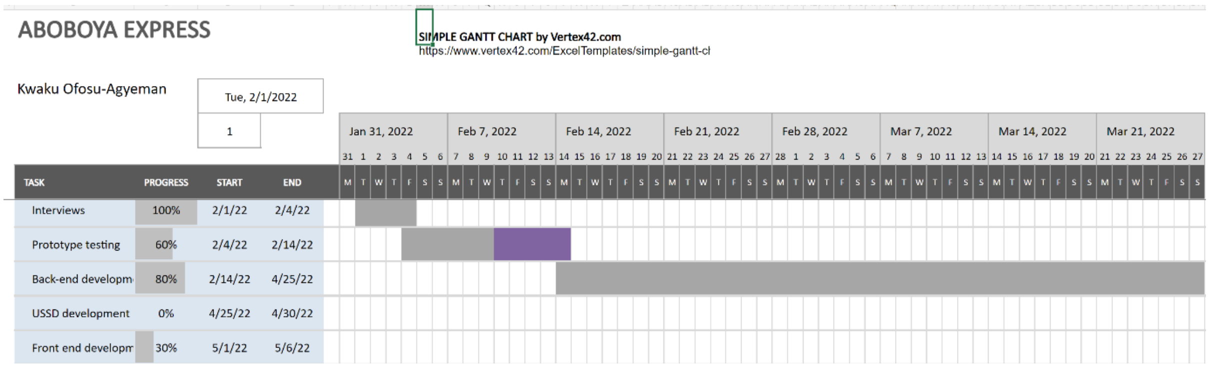
Figure 5.2.1

Figure 5.2.1 above shows the results of the conducted tests. The description of each file's test features is also shown. The tests conducted involved user authentication tests (login, logout, and register) as well as tests to make sure that the appointments could only be made by an authenticated user. The tests also involved testing for routes to make sure that users and guests are not allowed to access certain pages.

Chapter 6: Summary

The project is supposed to help people in Ghanaian homes that do not have any proper form of waste disposal to gain an acceptable system that would work as efficiently as already existing ones like *Zoomlion*. This project is not to make a profit and any money made from it is to be used to pay for hosting privileges and the use of tools for development. The project meets at least 90% of the functional requirements that were stated above. There is however one thing that I was unable to develop for the project. This is the USSD platform to allow collectors to register for the system. The USSD platform was to make it easier for collectors that are more familiar with using USSD and not websites to easily register for the service. Since they would receive messages each day, informing them of the users that made an appointment with them, they would not have to check the website to see if they do have to collect for a user on any particular day.

Appendices



ABOBOYA EXPRESS

SIMPLE GANTT CHART by Vertex42.com
https://www.vertex42.com/ExcelTemplates/simple-gantt-cl

Kwaku Ofosu-Agyeman

Tue, 2/1/2022

1

TASK	PROGRESS	START	END
Interviews	100%	2/1/22	2/4/22
Prototype testing	60%	2/4/22	2/14/22
Back-end developm	80%	2/14/22	4/25/22
USSD development	0%	4/25/22	4/30/22
Front end developm	30%	5/1/22	5/6/22

The Gantt chart above shows the progress I have made in the system as well as the schedule I used to develop the system. The beginning was used for conducting interviews to understand the perspective of all prospective users of the system. A system prototype was then later developed to test how they would react to the system and help with designing the full system. After this, the development of the complete system began. The PHP Laravel backend of the system was first completed to ensure all functionalities of the system were functioning appropriately before the front-end development began. The front-end development was finished within two weeks of the completion of the backend.

REFERENCES

[1] Atawneh, S., Al-Kasasbeh, B. and Ben Rshed, M., 2019. Android-based Mobile Application for Door-to-Door Product Delivery. *International Journal of Interactive Mobile Technologies (iJIM)*, 13(03), p.125.

[2] Ghazali, M., Kassim, M., Ya'acob, N., Idris, A. and Saaidin, S., 2019. Mobile Application on Garbage Collector Tracker Using Google Maps. *2019 IEEE 9th International Conference on System Engineering and Technology (ICSET)*.

[3] Miezah K, Obiri-Danso K, Kádár Z, Fei-Baffoe B, Mensah MY. Municipal solid waste characterization and quantification as a measure towards effective waste management in Ghana. Waste Manag. 2015. pmid:26421480

[4] Mensah, A. and Larbi E. solid waste disposal in ghana. WELL FACTSHEET uk—Regional Annex. 2005.

[5] Zhang, S., Zhang, J., Zhao, Z. and Xin, C., 2021. Robust Optimization of Municipal Solid Waste Collection and Transportation with Uncertain Waste Output: A Case Study. *Journal of Systems Science and Systems Engineering*, 31(2), pp.204-225.