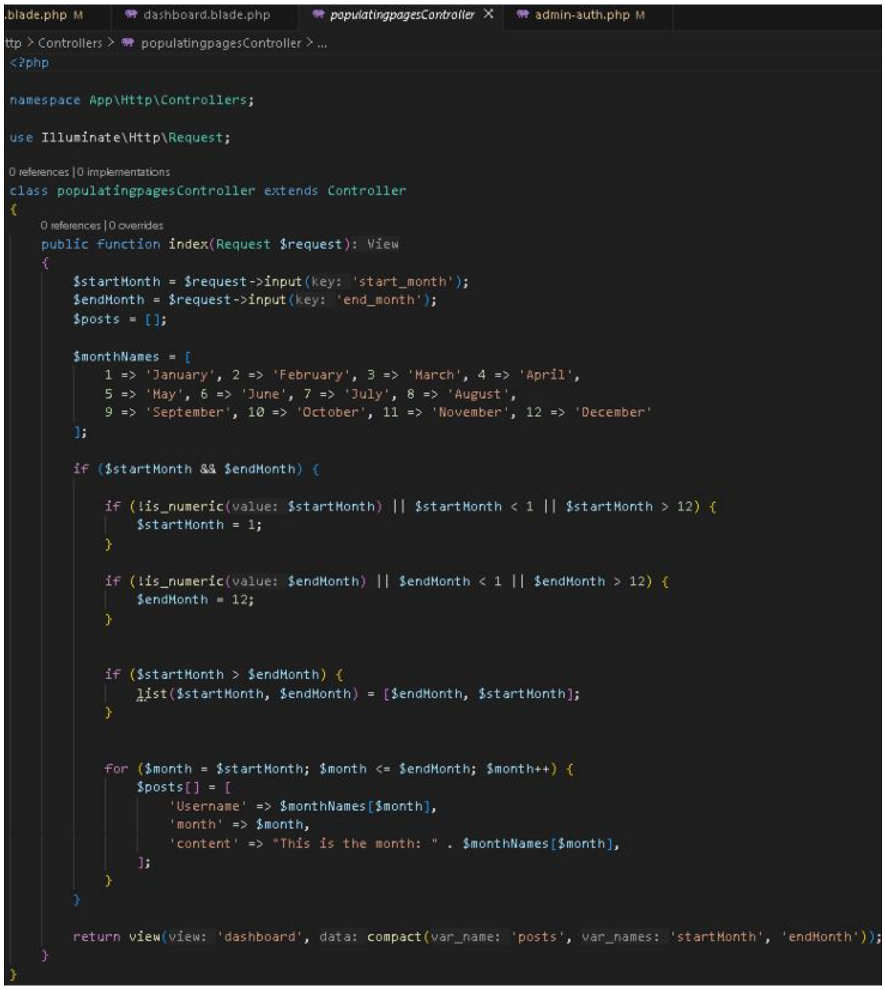
Casimiro, Althea M.

BSIT – 3C

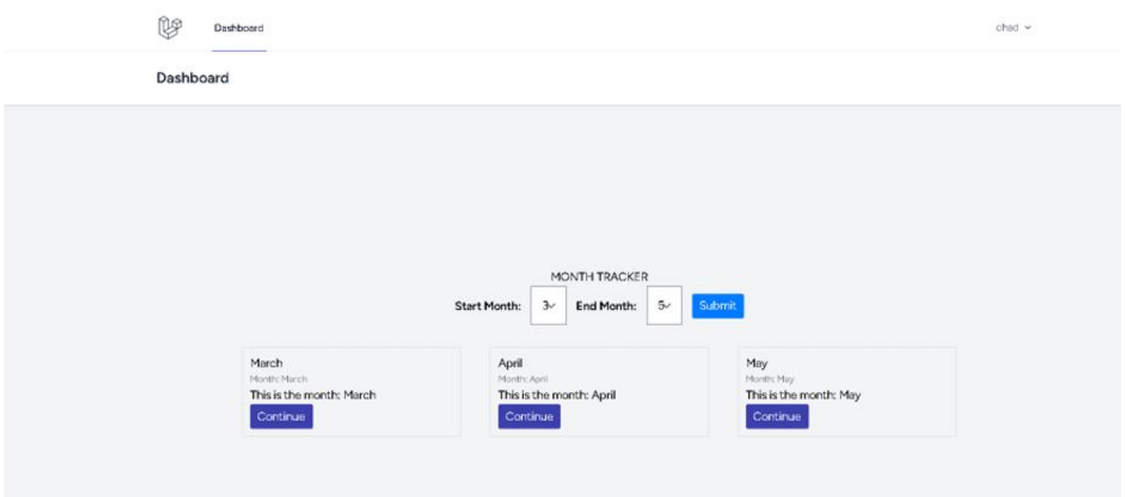
Web Development

Lab Activity: Populating from a Database


.blade.php M
dashboard.blade.php
populatingpagesController
admin-auth.php M
ttp > Controllers > populatingpagesController > ...

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

0 references | 0 implementations
class populatingpagesController extends Controller
{
    0 references | 0 overrides
    public function index(Request $request): View
    {
        $startMonth = $request->input(key: 'start_month');
        $endMonth = $request->input(key: 'end_month');
        $posts = [];

        $monthNames = [
            1 => 'January', 2 => 'February', 3 => 'March', 4 => 'April',
            5 => 'May', 6 => 'June', 7 => 'July', 8 => 'August',
            9 => 'September', 10 => 'October', 11 => 'November', 12 => 'December'
        ];

        if ($startMonth && $endMonth) {

            if (!is_numeric(value: $startMonth) || $startMonth < 1 || $startMonth > 12) {
                $startMonth = 1;
            }

            if (!is_numeric(value: $endMonth) || $endMonth < 1 || $endMonth > 12) {
                $endMonth = 12;
            }


            if ($startMonth > $endMonth) {
                list($startMonth, $endMonth) = [$endMonth, $startMonth];
            }


            for ($month = $startMonth; $month <= $endMonth; $month++) {
                $posts[] = [
                    'Username' => $monthNames[$month],
                    'month' => $month,
                    'content' => "This is the month: " . $monthNames[$month],
                ];
            }
        }

        return view(view: 'dashboard', data: compact(var_name: 'posts', var_names: 'startMonth', 'endMonth'));
    }
}
```

- This controller is designed to handle the functionality of populating the dashboard by allowing users to specify the number of months they wish to track and monitor their expenses. It ensures that the user's preferred time frame is used to retrieve and display the relevant expense data.

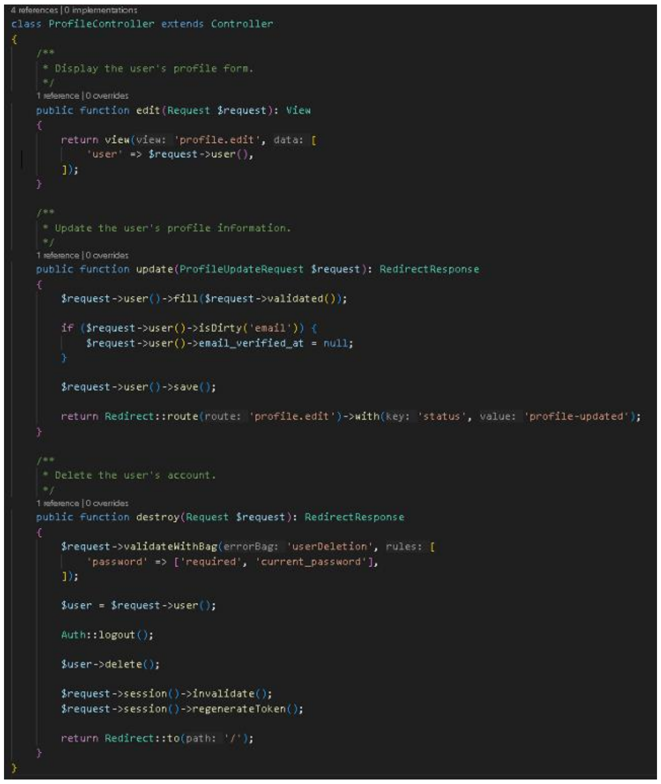


March
Month: March
This is the month: March
Continue

April
Month: April
This is the month: April
Continue

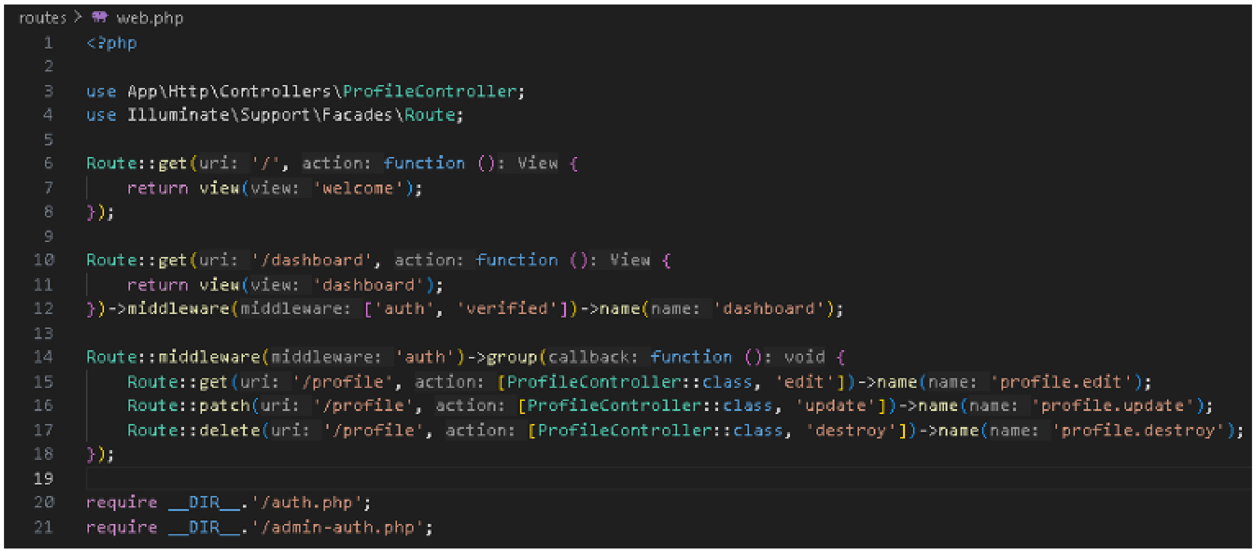
May
Month: May
This is the month: May
Continue

- The generated output



```
4 references | 0 implementations
class ProfileController extends Controller
{
    /**
     * Display the user's profile form.
     */
    1 reference | 0 overrides
    public function edit(Request $request): View
    {
        return view(view: 'profile.edit', data: [
            'user' => $request->user(),
        ]);
    }

    /**
     * Update the user's profile information.
     */
    1 reference | 0 overrides
    public function update(ProfileUpdateRequest $request): RedirectResponse
    {
        $request->user()->fill($request->validated());

        if ($request->user()->isDirty('email')) {
            $request->user()->email_verified_at = null;
        }

        $request->user()->save();

        return Redirect::route(route: 'profile.edit')->with(key: 'status', value: 'profile-updated');
    }

    /**
     * Delete the user's account.
     */
    1 reference | 0 overrides
    public function destroy(Request $request): RedirectResponse
    {
        $request->validateWithBag(errorBag: 'userDeletion', rules: [
            'password' => ['required', 'current_password'],
        ]);

        $user = $request->user();

        Auth::logout();

        $user->delete();

        $request->session()->invalidate();
        $request->session()->regenerateToken();

        return Redirect::to(path: '/');
    }
}
```

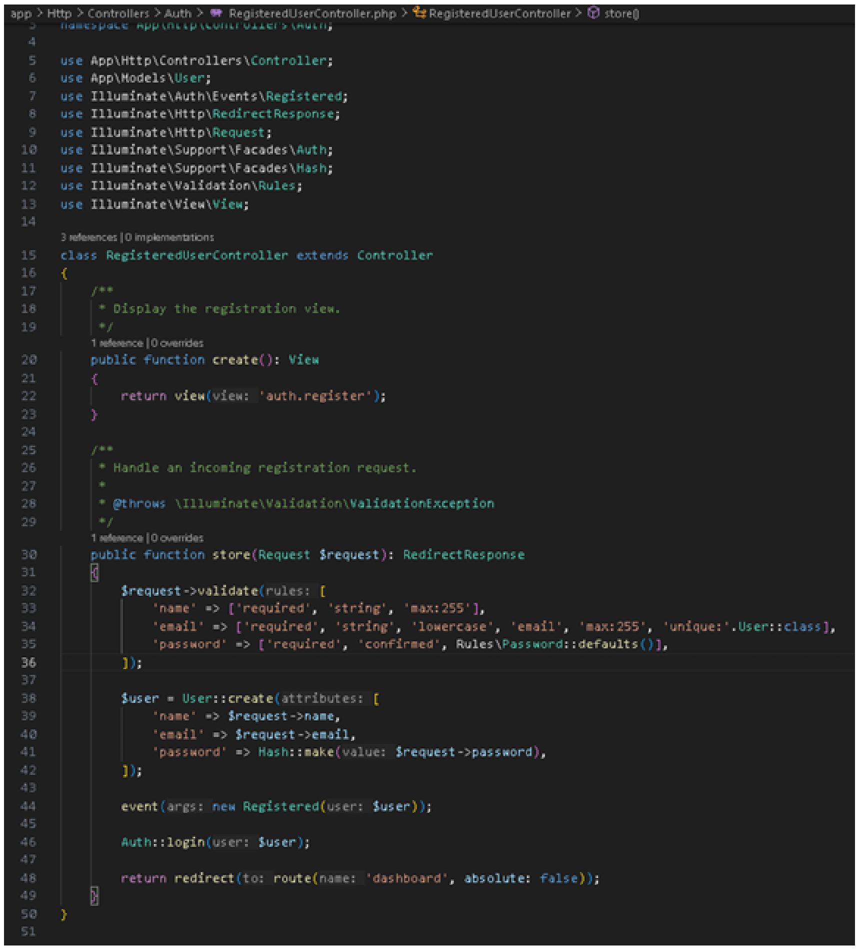
- This controller, which is registered in the application's routes, is responsible for managing the functionality related to user profile editing.



routes > web.php

```php
<?php

use App\Http\Controllers\ProfileController;
use Illuminate\Support\Facades\Route;

Route::get(uri: '/', action: function (): View {
    return view(view: 'welcome');
});

Route::get(uri: '/dashboard', action: function (): View {
    return view(view: 'dashboard');
})->middleware(middleware: ['auth', 'verified'])->name(name: 'dashboard');

Route::middleware(middleware: 'auth')->group(callback: function (): void {
    Route::get(uri: '/profile', action: [ProfileController::class, 'edit'])->name(name: 'profile.edit');
    Route::patch(uri: '/profile', action: [ProfileController::class, 'update'])->name(name: 'profile.update');
    Route::delete(uri: '/profile', action: [ProfileController::class, 'destroy'])->name(name: 'profile.destroy');
});

require __DIR__.'/auth.php';
require __DIR__.'/admin-auth.php';
```

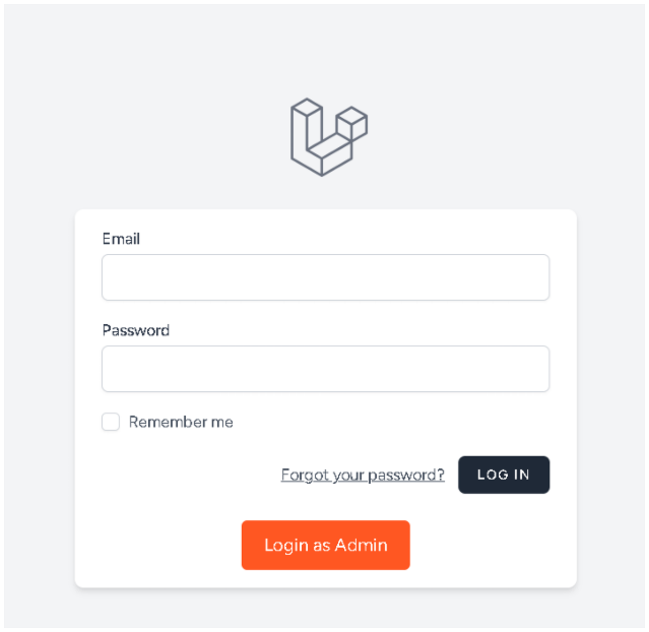
- These are the application routes, and within them, the ProfileController is registered to handle all requests related to user profiles.



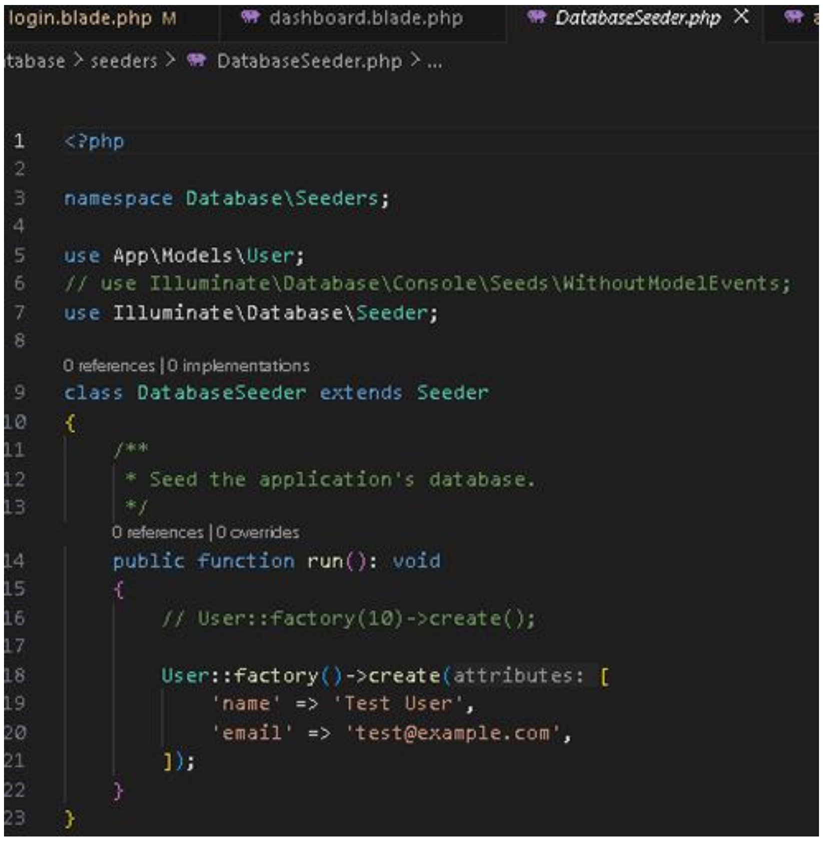
app > Http > Controllers > Auth > RegisteredUserController.php > RegisteredUserController > store()

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

use App\Http\Controllers\Controller;
use App\Models\User;
use Illuminate\Auth\Events\Registered;
use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use Illuminate\Validation\Rules;
use Illuminate\View\View;

3 references | 0 implementations
class RegisteredUserController extends Controller
{
    /**
     * Display the registration view.
     */
    1 reference | 0 overrides
    public function create(): View
    {
        return view(view: 'auth.register');
    }

    /**
     * Handle an incoming registration request.
     *
     * @throws \Illuminate\Validation\ValidationException
     */
    1 reference | 0 overrides
    public function store(Request $request): RedirectResponse
    {
        $request->validate(rules: [
            'name' => ['required', 'string', 'max:255'],
            'email' => ['required', 'string', 'lowercase', 'email', 'max:255', 'unique:'.User::class],
            'password' => ['required', 'confirmed', Rules\Password::defaults()],
        ]);

        $user = User::create(attributes: [
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make(value: $request->password),
        ]);

        event(args: new Registered(user: $user));

        Auth::login(user: $user);

        return redirect(to: route(name: 'dashboard', absolute: false));
    }
}
```

- This controller is responsible for handling the entire process of account creation. It manages the logic behind registering new users.

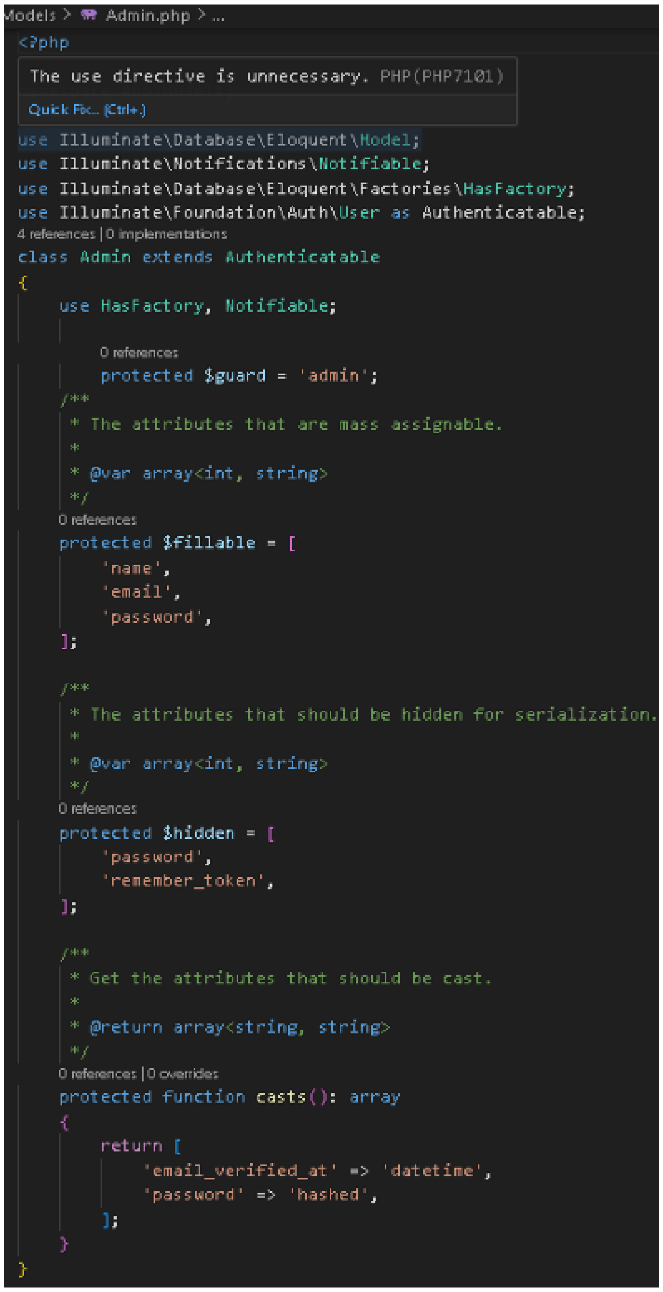


- This feature is visible on the login page, allowing users to begin the process of logging into their accounts.



```php
<?php

namespace Database\Seeders;

use App\Models\User;
// use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;

0 references | 0 implementations
class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     */
    0 references | 0 overrides
    public function run(): void
    {
        // User::factory(10)->create();

        User::factory()->create(attributes: [
            'name' => 'Test User',
            'email' => 'test@example.com',
        ]);
    }
}
```

- This is the seeding process, which runs and generates a test email specifically created for testing purposes. This seeded email can be used to log in as a user, allowing developers or testers to simulate a user login and verify the functionality of the application.


Models > Admin.php > ...
The use directive is unnecessary. PHP(PHP7101)
Quick Fix... (Ctrl+.)
4 references | 0 implementations
0 references
0 references
0 references
0 references | 0 overrides

```php
<?php
use Illuminate\Database\Eloquent\Model;
use Illuminate\Notifications\Notifiable;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
class Admin extends Authenticatable
{
    use HasFactory, Notifiable;

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

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

    /**
     * Get the attributes that should be cast.
     *
     * @return array<string, string>
     */
    protected function casts(): array
    {
        return [
            'email_verified_at' => 'datetime',
            'password' => 'hashed',
        ];
    }
}
```

- This model represents the admin functionality within the application and is responsible for performing essential administrative tasks, such as updating and deleting user accounts.