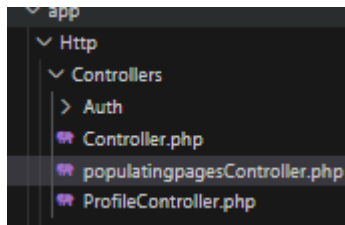


Naje, Niño Miguel L.  
BS IT – 3 C  
Web Dev Populating the webpages

These are controllers that we made:



Here is the populating the web pages controller:

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class PopulatingPagesController extends Controller
8 {
9     public function index(Request $request)
10     {
11         $startMonth = $request->input('start_month');
12         $endMonth = $request->input('end_month');
13         $posts = [];
14
15         $monthNames = [
16             1 => 'January', 2 => 'February', 3 => 'March', 4 => 'April',
17             5 => 'May', 6 => 'June', 7 => 'July', 8 => 'August',
18             9 => 'September', 10 => 'October', 11 => 'November', 12 => 'December'
19         ];
20
21         if ($startMonth && $endMonth) {
22             if (!is_numeric($startMonth) || $startMonth < 1 || $startMonth > 12) {
23                 $startMonth = 1;
24             }
25
26             if (!is_numeric($endMonth) || $endMonth < 1 || $endMonth > 12) {
27                 $endMonth = 12;
28             }
29
30             if ($startMonth > $endMonth) {
31                 list($startMonth, $endMonth) = [$endMonth, $startMonth];
32             }
33
34             for ($month = $startMonth; $month <= $endMonth; $month++) {
35                 $posts[] = [
36                     'username' => $monthNames[$month],
37                     'month' => $month,
38                     'content' => "This is the month: " . $monthNames[$month],
39                 ];
40             }
41
42             return view('dashboard', compact('posts', 'startMonth', 'endMonth'));
43         }
44     }
45 }
```

We created the dashboard to allow users to easily track their expenses by specifying a range of months they want to monitor. To do this, the user needs to input both a starting month and an ending month, which will determine the period for which the dashboard displays data.

For example, the months are represented as numbers, where January is 1, February is 2, and so on, up to December, which is represented as 12. If a user decides to track their expenses starting from the 1st month (January) and ending at the 3rd month (March), the dashboard will automatically generate and display data covering the months of January, February, and March. This system makes it straightforward for users to customize the period they wish to analyze, ensuring flexibility and ease of use.

The next feature is the **Profile Controller**, which allows users to manage and update their profile information easily. With this functionality, users can perform various actions such as editing their email address, updating their password, and making other necessary changes to their profile details.

This controller implements **CRUD operations** (Create, Read, Update, and Delete), ensuring that users can not only view and modify their profile data but also manage it comprehensively as needed. This feature is designed to provide a seamless and user-friendly experience for maintaining up-to-date and accurate account information.

Here's the code:

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Http\Requests\ProfileUpdateRequest;
6 use Illuminate\Http\RedirectResponse;
7 use Illuminate\Http\Request;
8 use Illuminate\Support\Facades\Auth;
9 use Illuminate\Support\Facades\Redirect;
10 use Illuminate\View\View;
11
12 class ProfileController extends Controller
13 {
14     /**
15      * Display the user's profile form.
16      */
17     public function edit(Request $request): View
18     {
19         return view('profile.edit', [
20             'user' => $request->user(),
21         ]);
22     }
23
24     /**
25      * Update the user's profile information.
26      */
27     public function update(ProfileUpdateRequest $request): RedirectResponse
28     {
29         $request->user()->fill($request->validated());
30
31         if ($request->user()->isDirty('email')) {
32             $request->user()->email_verified_at = null;
33         }
34
35         $request->user()->save();
36
37         return Redirect::route('profile.edit')->with('status', 'profile-updated');
38     }
39
40     /**
41      * Delete the user's account.
42      */
43     public function destroy(Request $request): RedirectResponse
44     {
45         $request->validateWithBag('userDeletion', [
46             'password' => ['required', 'current_password'],
47         ]);
48
49         $user = $request->user();
50
51         Auth::logout();
52
53         $user->delete();
54
55         $request->session()->invalidate();
56         $request->session()->regenerateToken();
57
58         return Redirect::to('/');
59     }
60 }
```

Settings

Activity Log

Log Out

SAVE

Update Password

Ensure your account is using a long, random password to stay secure.

Current Password

New Password

Confirm Password

SAVE

Delete Account

Once your account is deleted, all of its resources and data will be permanently deleted. Before deleting your account, please download any data or information that you wish to retain.

DELETE ACCOUNT

And here's the output:

## Feed

Start

Month:

1

End

Month:

3

Submit

### January

Month: January

This is the month: January

continue

### February

Month: February

This is the month: February

continue

### March

Month: March

This is the month: March

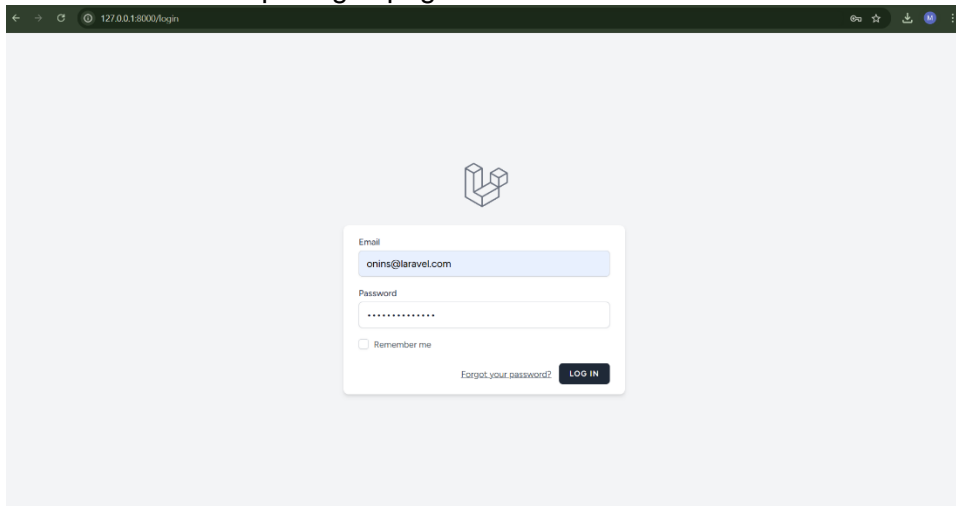
continue

Here is the output when you click the continue button:

127.0.0.1:8000 says  
Continue/January!

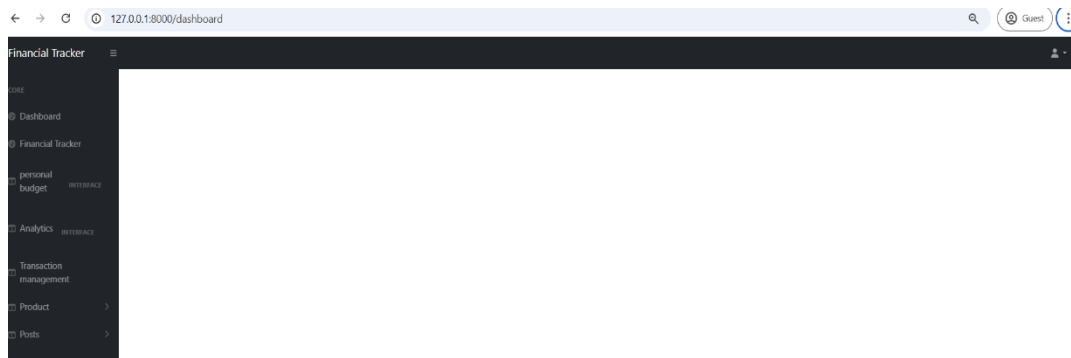
OK

Here is the sample log in page.



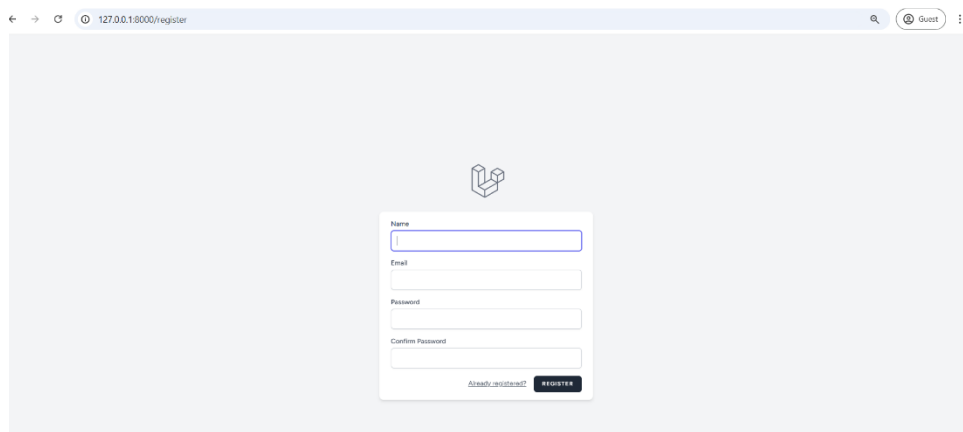
The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000/login'. The page has a light gray background and a central white login form. At the top of the form is a blue cube icon. Below the icon are two input fields: 'Email' with the value 'onins@laravel.com' and 'Password' with masked characters. There is a 'Remember me' checkbox and a 'Forgot your password?' link. A dark blue 'LOG IN' button is at the bottom right of the form.

After logging in here is the dashboard

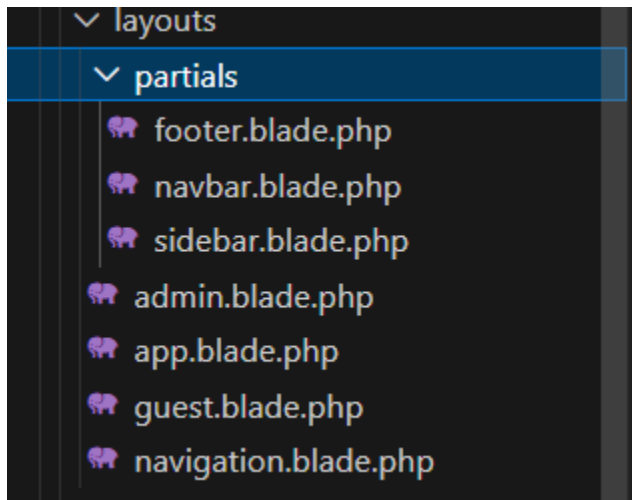


The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000/dashboard'. The page has a dark sidebar on the left with the title 'Financial Tracker' and a list of menu items: 'Dashboard', 'Financial Tracker', 'personal budget', 'Analytics', 'Transaction management', 'Product', and 'Posts'. The main content area is white and currently empty.

The user can then choose to register an account



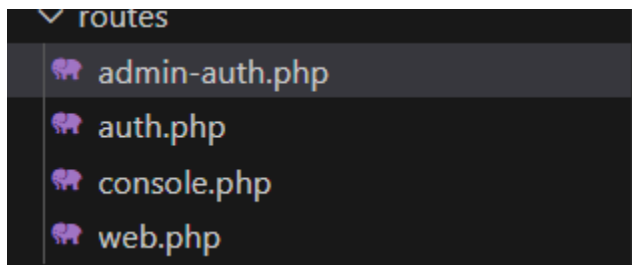
The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000/register'. The page has a light gray background and a central white registration form. At the top of the form is a blue cube icon. Below the icon are four input fields: 'Name', 'Email', 'Password', and 'Confirm Password'. There is a link 'Already registered?' and a dark blue 'REGISTER' button at the bottom right of the form.



The Layout serves as the foundation of the web application's design and structure. It includes key components such as the footer, navigation bar, sidebar, and other essential elements that define the overall look and feel of the application.

The footer typically contains important links, copyright information, or contact details, while the navigation bar helps users move easily between different sections of the application. The sidebar provides quick access to various features or tools, enhancing the user experience.

These are the other routes:



For the **admin routes**, we created a dedicated setup to ensure that all admin-related functionalities are organized and separate from the user routes. This separation improves security and makes the application easier to manage and maintain. The admin routes include features specific to administrative tasks, such as managing user data, monitoring system activity, and overseeing other critical operations.

We also implemented an **admin/login** page where users can choose whether to log in as a regular user or an admin. This login interface provides a clear distinction between the two roles, ensuring that admins have access to advanced features while regular users are limited to the functionalities appropriate to their accounts.

```
<?php
use App\Http\Controllers\Auth\Admin\Auth\LoginController;
use App\Http\Controllers\Auth\Admin\Auth\RegisteredUserController;
use Illuminate\Support\Facades\Route;

Route::prefix('admin')->middleware('guest:admin')->group(function () {
    Route::get('register', [RegisteredUserController::class, 'create'])->name('admin.register');
    Route::post('register', [RegisteredUserController::class, 'store']);

    Route::get('login', [LoginController::class, 'create'])->name('admin.login');
    Route::post('login', [LoginController::class, 'store']);
});

Route::prefix('admin')->middleware('auth:admin')->group(function () {
    Route::get('/dashboard', function () {
        return view('admin.dashboard');
    })->name('admin.dashboard');

    Route::post('logout', [LoginController::class, 'destroy'])->name('admin.logout');
});
```

```

use App\Http\Controllers\Auth\AuthenticatedSessionController;
use App\Http\Controllers\Auth\ConfirmablePasswordController;
use App\Http\Controllers\Auth>EmailVerificationNotificationController;
use App\Http\Controllers\Auth>EmailVerificationPromptController;
use App\Http\Controllers\Auth\NewPasswordController;
use App\Http\Controllers\Auth>PasswordController;
use App\Http\Controllers\Auth>PasswordResetLinkController;
use App\Http\Controllers\Auth\RegisteredUserController;
use App\Http\Controllers\Auth\VerifyEmailController;
use Illuminate\Support\Facades\Route;

Route::middleware('guest')->group(function () {
    Route::get('register', [RegisteredUserController::class, 'create'])
        ->name('register');

    Route::post('register', [RegisteredUserController::class, 'store']);

    Route::get('login', [AuthenticatedSessionController::class, 'create'])
        ->name('login');

    Route::post('login', [AuthenticatedSessionController::class, 'store']);

    Route::get('forgot-password', [PasswordResetLinkController::class, 'create'])
        ->name('password.request');

    Route::post('forgot-password', [PasswordResetLinkController::class, 'store'])
        ->name('password.email');

    Route::get('reset-password/{token}', [NewPasswordController::class, 'create'])
        ->name('password.reset');

    Route::post('reset-password', [NewPasswordController::class, 'store'])
        ->name('password.store');
});

Route::middleware('auth')->group(function () {
    Route::get('verify-email', EmailVerificationPromptController::class)
        ->name('verification.notice');

    Route::get('verify-email/{id}/{hash}', VerifyEmailController::class)
        ->middleware(['signed', 'throttle:6,1'])
        ->name('verification.verify');

    Route::post('email/verification-notification', [EmailVerificationNotificationController::class, 'store'])
        ->middleware('throttle:6,1')
        ->name('verification.send');

    Route::get('confirm-password', [ConfirmablePasswordController::class, 'show'])
        ->name('password.confirm');

    Route::post('confirm-password', [ConfirmablePasswordController::class, 'store']);

    Route::put('password', [PasswordController::class, 'update'])->name('password.update');

    Route::post('logout', [AuthenticatedSessionController::class, 'destroy'])
        ->name('logout');
});

```

The system includes a complete CRUD functionality within the user registration process, allowing users to manage their accounts effectively. Users can create a new account by providing necessary details such as email and password, view their account information, update their profile by editing details like email or password, and delete their account if they no longer wish to use the platform. Additionally, the system features a log-out option to ensure users can securely exit their accounts after use, providing a seamless and secure account management experience.

We also worked on connecting the database to ensure that it functions properly and supports the application's features. The database will store information for both registered users and admins, securely managing their account details. Additionally, we plan to use the database to store and manage the user's data, such as budget trackers, expense records, and other relevant information. This setup ensures that all data is organized, accessible, and ready to support the application's core functionalities.

Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> admins	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
<input type="checkbox"/> migrations	★ Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
<input type="checkbox"/> password_reset_tokens	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
<input type="checkbox"/> sessions	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_unicode_ci	48.0 KiB	-
<input type="checkbox"/> users	★ Browse Structure Search Insert Empty Drop	8	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
10 tables	Sum	35	InnoDB	utf8mb4_general_ci	256.0 KiB	0 B