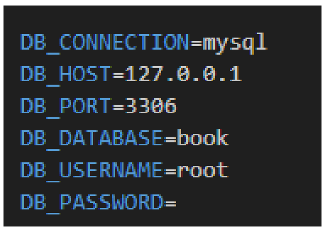
IT Elec: WEB DEVELOPMENT
POPULATING FROM A DATABASE

WENDEE DIANE FLORES LLONA
BSIT-3C

DATABASE:

We created a table named ‘book’ in our chosen MySql server, Xampp.

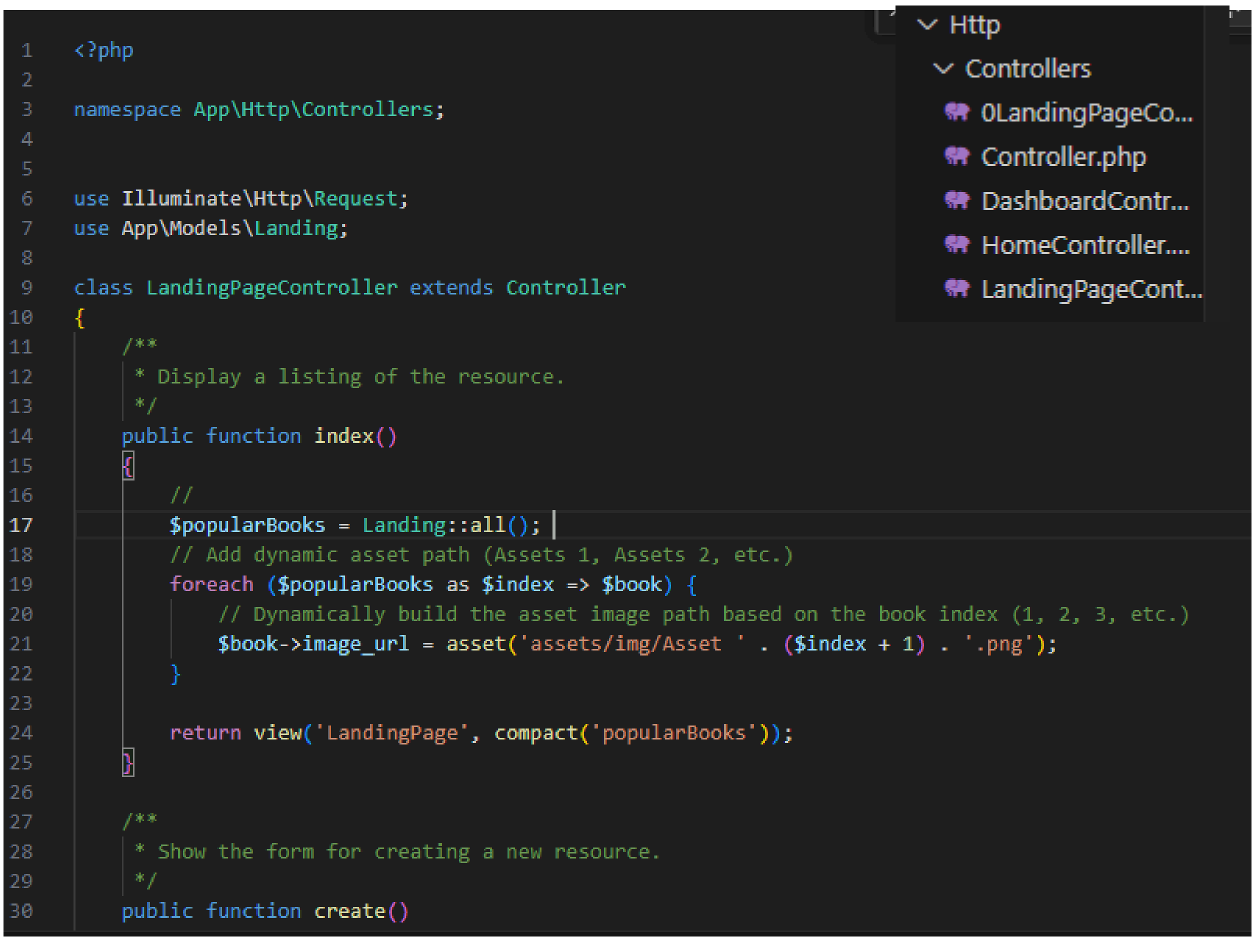
.env:



```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=book
DB_USERNAME=root
DB_PASSWORD=
```

We modified the .env file to update the necessary configuration settings for our application.

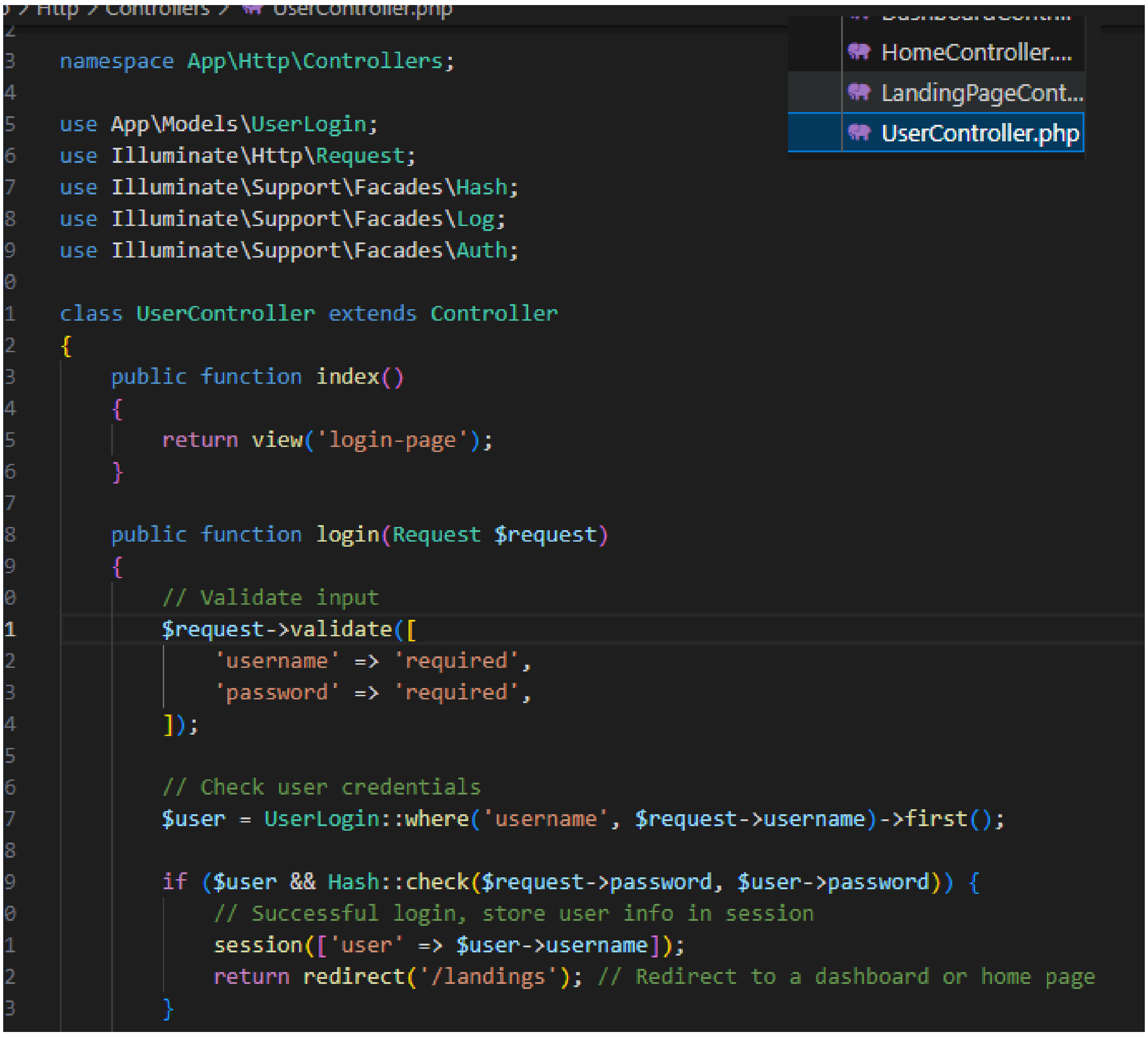
CONTROLLERS: LandingPageController.php



```php
<?php

namespace App\Http\Controllers;


use Illuminate\Http\Request;
use App\Models\Landing;

class LandingPageController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        //
        $popularBooks = Landing::all();
        // Add dynamic asset path (Assets 1, Assets 2, etc.)
        foreach ($popularBooks as $index => $book) {
            // Dynamically build the asset image path based on the book index (1, 2, 3, etc.)
            $book->image_url = asset('assets/img/Asset ' . ($index + 1) . '.png');
        }

        return view('LandingPage', compact('popularBooks'));
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
```

This LandingPageController is a resource controller that manages CRUD operations for Landing model resources. The index method retrieves all Landing model records and stores them in ‘popularBooks’. It uses the Landing::all() function to retrieve all entries from the corresponding database table. Additionally, It dynamically assigns an image_url to each book by iterating through the records and generating an asset path based on the book's index (e.g., Asset 1.png, Asset 2.png). This ensures each book is linked to a specific image for display in the view.

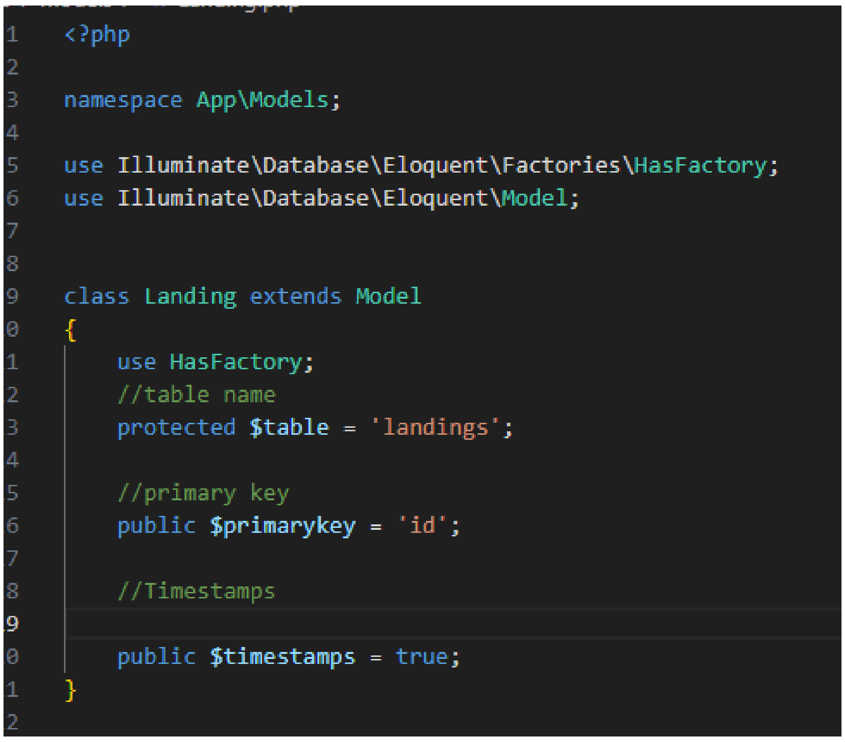
CONTROLLERS: UserController.php



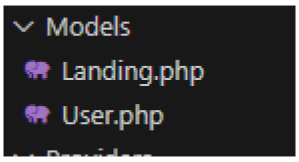
```php
namespace App\Http\Controllers;

use App\Models\UserLogin;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Log;
use Illuminate\Support\Facades\Auth;

class UserController extends Controller
{
    public function index()
    {
        return view('login-page');
    }

    public function login(Request $request)
    {
        // Validate input
        $request->validate([
            'username' => 'required',
            'password' => 'required',
        ]);

        // Check user credentials
        $user = UserLogin::where('username', $request->username)->first();

        if ($user && Hash::check($request->password, $user->password)) {
            // Successful login, store user info in session
            session(['user' => $user->username]);
            return redirect('/landings'); // Redirect to a dashboard or home page
        }
```

This UserController handles user authentication and registration. The index method displays the login page. The login method validates the input, checks if the username and password match a record in the UserLogin model, and redirects users to the landing page upon success or back with errors if the credentials are invalid. The register method validates user input, ensures the username is unique, and securely stores the user's information in the database using Hash for the password. After successful registration, it redirects to the login page with a success message.

MODELS: Landing.php

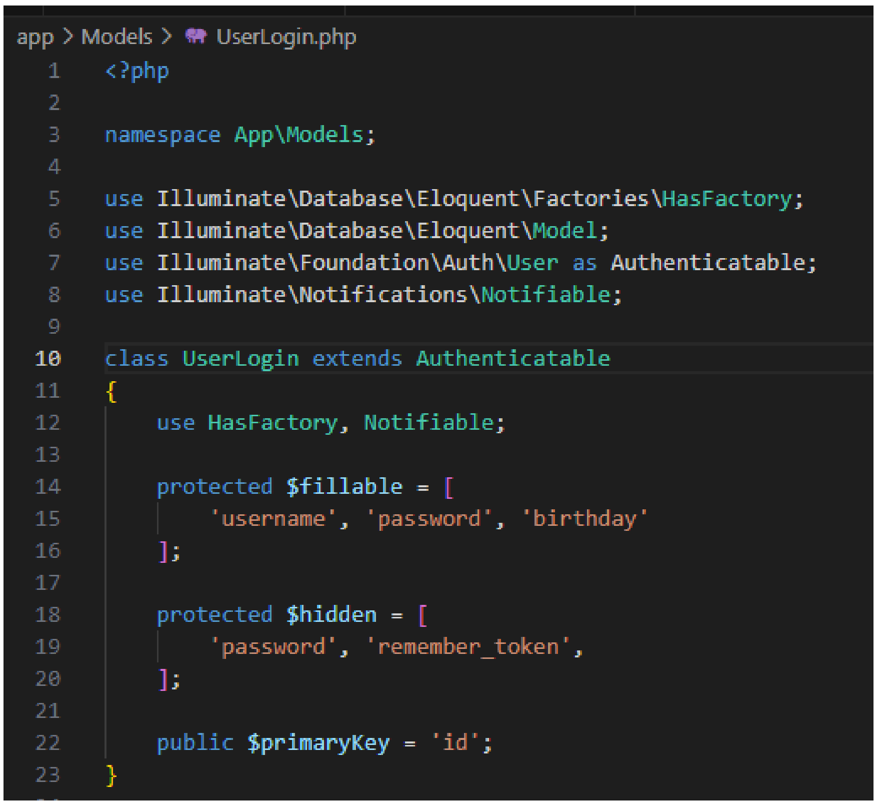


```php
<?php

namespace App\Models;

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


class Landing extends Model
{
    use HasFactory;
    //table name
    protected $table = 'landings';

    //primary key
    public $primarykey = 'id';

    //Timestamps

    public $timestamps = true;
}
```

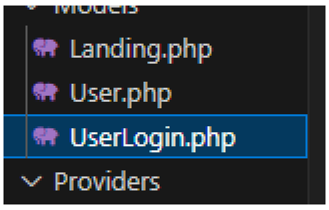


The Landing model maps to the ‘landings table’, uses id as its primary key, and manages timestamps (created_at and updated_at) automatically.

MODELS: UserLogin.php

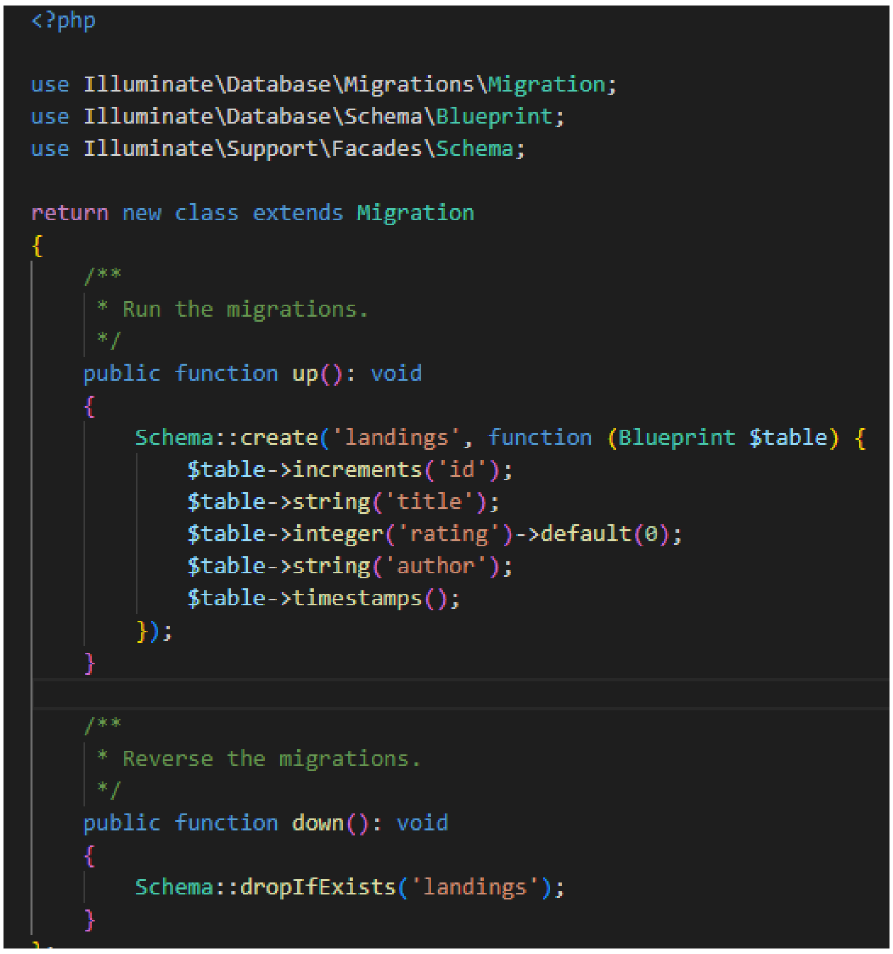


```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class UserLogin extends Authenticatable
{
    use HasFactory, Notifiable;

    protected $fillable = [
        'username', 'password', 'birthday'
    ];

    protected $hidden = [
        'password', 'remember_token',
    ];

    public $primaryKey = 'id';
}
```

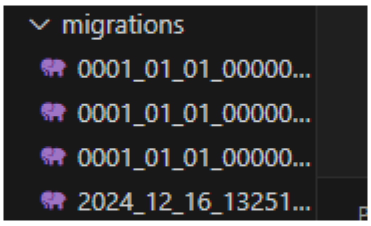


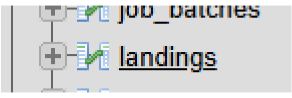
This UserLogin model extends Laravel's Authenticatable class, enabling user authentication functionality. It represents a database table with id as the primary key and specifies username, password, and birthday as mass-assignable attributes. Sensitive fields like password and remember_token are hidden from arrays for security purposes.

MIGRATIONS: landings



```php
<?php

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

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('landings', function (Blueprint $table) {
            $table->increments('id');
            $table->string('title');
            $table->integer('rating')->default(0);
            $table->string('author');
            $table->timestamps();
        });
    }

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

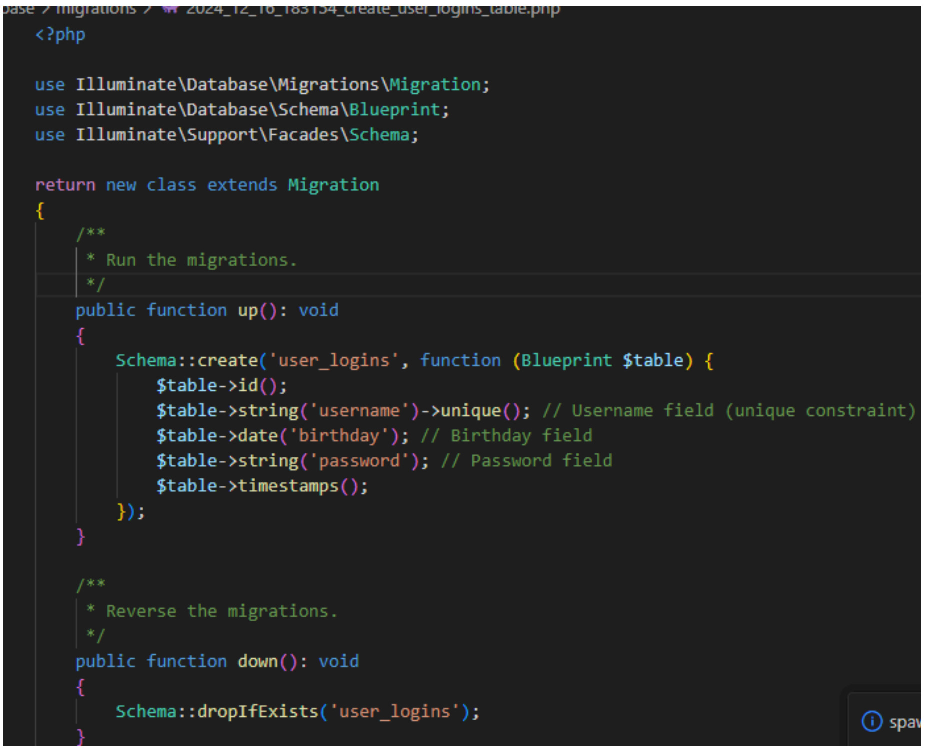




This migration creates the landings table with columns for id (primary key), title (book title), rating (integer with default value 0), author (author's name), and timestamps for tracking creation and updates. The down method ensures that the table is dropped if the migration is reversed.

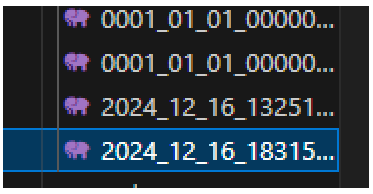
When ‘php artisan migrate’ is run, it will add the landings table with the specified columns to the database.

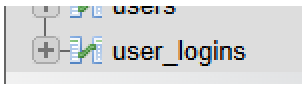
MIGRATIONS: user_logins



base > migrations > 2024_12_16_183134_create_user_logins_table.php

```php
<?php

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

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('user_logins', function (Blueprint $table) {
            $table->id();
            $table->string('username')->unique(); // Username field (unique constraint)
            $table->date('birthday'); // Birthday field
            $table->string('password'); // Password field
            $table->timestamps();
        });
    }

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

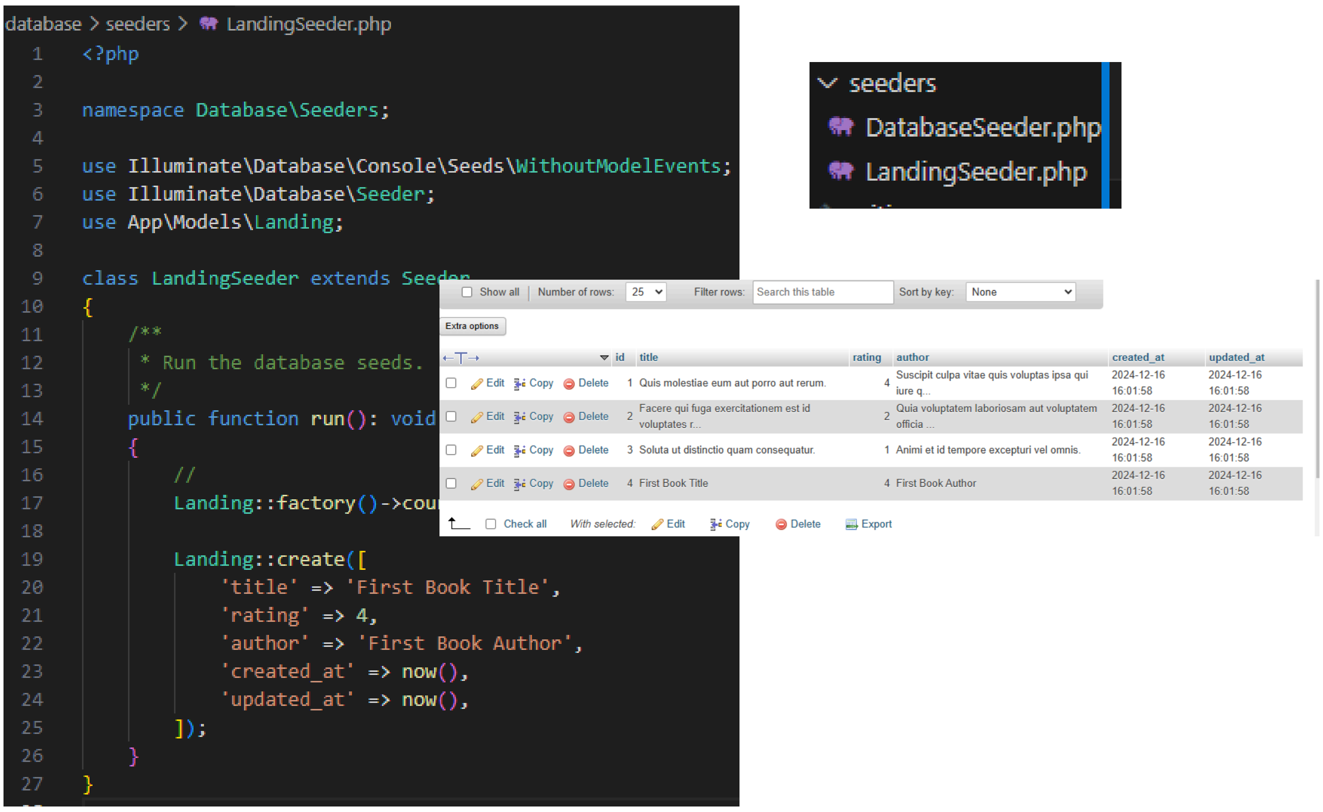




This migration creates a user_logins table with an auto-incrementing id, a unique username, a birthday field, and a password field. It also includes timestamps for tracking when records are created and updated. The down method ensures the table is dropped if the migration is rolled back.

When ‘php artisan migrate’ is run, it will create the user_logins table with the specified columns in the database.

SEEDING: LandingSeeder.php



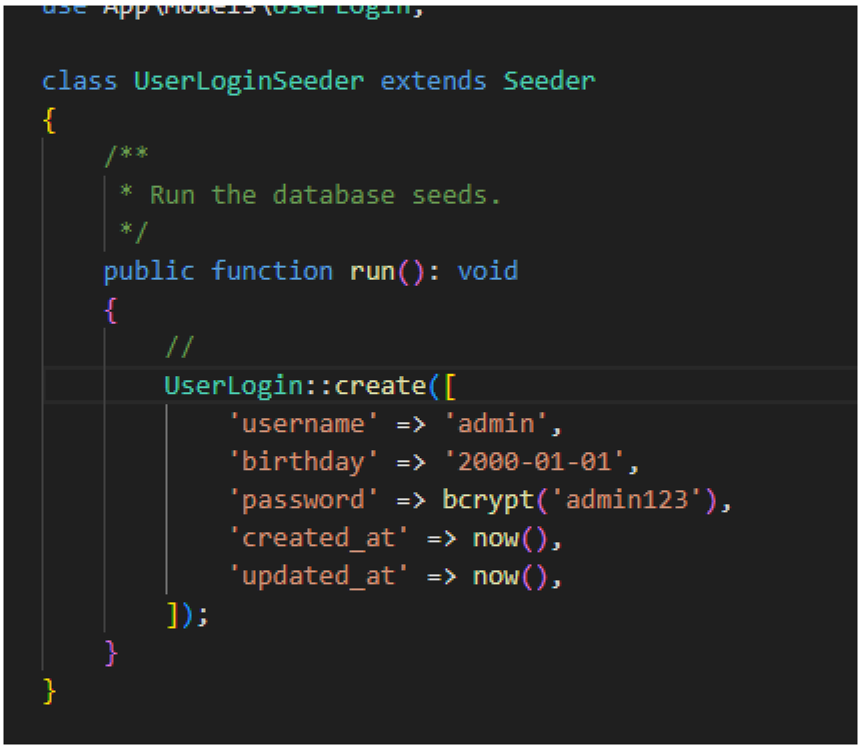
database > seeders > LandingSeeder.php

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use App\Models\Landing;

class LandingSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        //
        Landing::factory()->cou

        Landing::create([
            'title' => 'First Book Title',
            'rating' => 4,
            'author' => 'First Book Author',
            'created_at' => now(),
            'updated_at' => now(),
        ]);
    }
}
```

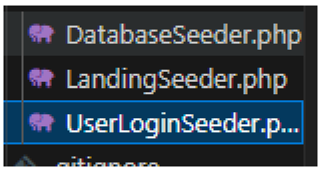
	id	title	rating	author	created_at	updated_at
□ Edit Copy Delete	1	Quis molestiae eum aut porro aut rerum.	4	Suscipit culpa vitae quis voluptas ipsa qui iure q...	2024-12-16 16:01:58	2024-12-16 16:01:58
□ Edit Copy Delete	2	Facere qui fuga exercitationem est id voluptates r...	2	Quia voluptatem laboriosam aut voluptatem officia ...	2024-12-16 16:01:58	2024-12-16 16:01:58
□ Edit Copy Delete	3	Soluta ut distinctio quam consequatur.	1	Animi et id tempore excepturi vel omnis.	2024-12-16 16:01:58	2024-12-16 16:01:58
□ Edit Copy Delete	4	First Book Title	4	First Book Author	2024-12-16 16:01:58	2024-12-16 16:01:58

This seeder populates the landings table with data. It uses a factory to create 3 random records for the table, then manually adds one specific entry with predefined values for title, rating, author, and timestamps (created_at and updated_at). The run method is executed when the seeder is run via ‘php artisan db:seed --class=LandingSeeder’.

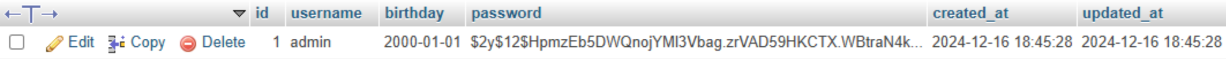
SEEDING: UserLoginSeeder.php



```php
use App\Models\UserLogin;

class UserLoginSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        //
        UserLogin::create([
            'username' => 'admin',
            'birthday' => '2000-01-01',
            'password' => bcrypt('admin123'),
            'created_at' => now(),
            'updated_at' => now(),
        ]);
    }
}
```



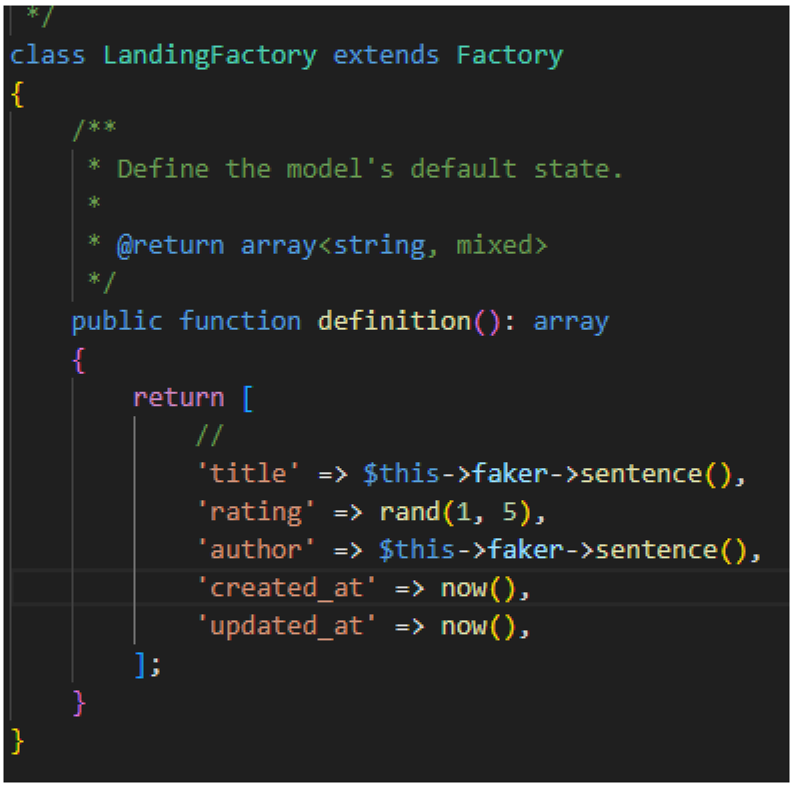
DatabaseSeeder.php
LandingSeeder.php
UserLoginSeeder.p...
.gitignore



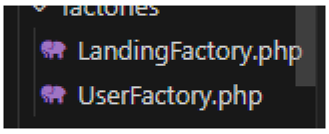
	id	username	birthday	password	created_at	updated_at
□ Edit Copy Delete	1	admin	2000-01-01	$2y$12$HpmzEb5DWQnojYMI3Vbag.zrVAD59HKCTX.WBtraN4k...	2024-12-16 18:45:28	2024-12-16 18:45:28

This seeder inserts a new user record into the user_logins table. It creates a user with the username set to 'admin', a birthday of '2000-01-01', and a hashed password of 'admin123' using the bcrypt function. The created_at and updated_at fields are set to the current timestamp. The run method is executed when the seeder is run with ‘php artisan db:seed --class=UserLoginSeeder’.

FACTORY: LandingFactory.php



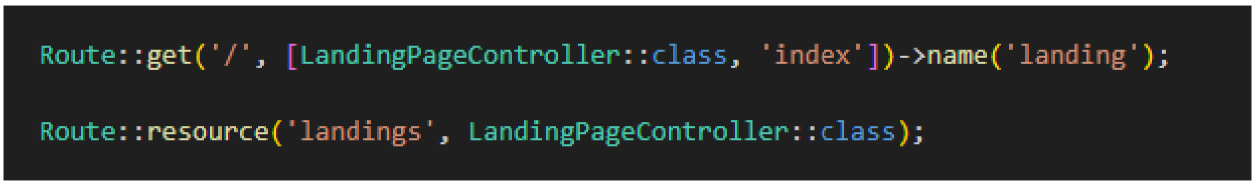
```php
 */
class LandingFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    public function definition(): array
    {
        return [
            //
            'title' => $this->faker->sentence(),
            'rating' => rand(1, 5),
            'author' => $this->faker->sentence(),
            'created_at' => now(),
            'updated_at' => now(),
        ];
    }
}
```



factories
LandingFactory.php
UserFactory.php

This factory generates fake data for the Landing model, including a random title, rating, author, and timestamps.

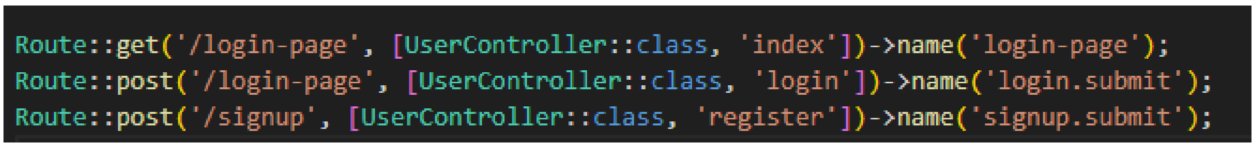
ROUTES: LandingPage



```
Route::get('/', [LandingPageController::class, 'index'])->name('landing');

Route::resource('landings', LandingPageController::class);
```

The first route maps the root URL (/) to the index method of LandingPageController with the name landing. The second route sets up resourceful routes for landings, linking CRUD actions to the LandingPageController.

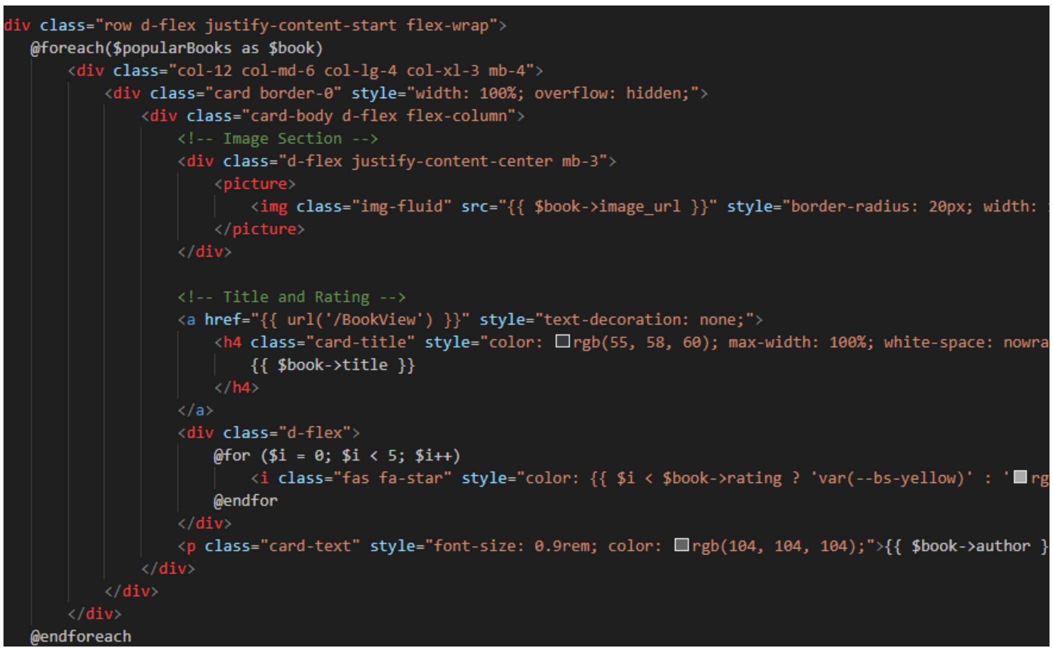
ROUTES: login-page



```
Route::get('/login-page', [UserController::class, 'index'])->name('login-page');
Route::post('/login-page', [UserController::class, 'login'])->name('login.submit');
Route::post('/signup', [UserController::class, 'register'])->name('signup.submit');
```

The first route maps the /login-page URL to the index method of UserController with the name login-page. The second route handles a POST request to /login-page, triggering the login method for user authentication. The third route maps a POST request to /signup, calling the register method to create a new user.

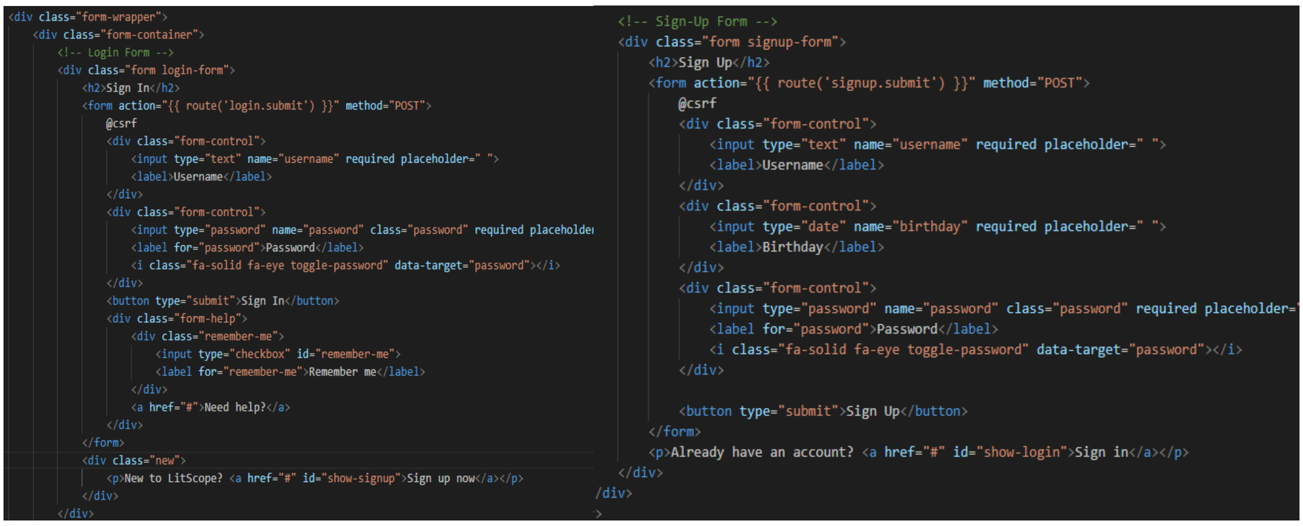
BLADE FILES: LandingPage.blade.php



```
div class="row d-flex justify-content-start flex-wrap">
   @foreach($popularBooks as $book)
       <div class="col-12 col-md-6 col-lg-4 col-xl-3 mb-4">
           <div class="card border-0" style="width: 100%; overflow: hidden;">
               <div class="card-body d-flex flex-column">
                   <!-- Image Section -->
                   <div class="d-flex justify-content-center mb-3">
                       <picture>
                           <img class="img-fluid" src="{{ $book->image_url }}" style="border-radius: 20px; width:
                       </picture>
                   </div>

                   <!-- Title and Rating -->
                   <a href="{{ url('/BookView') }}" style="text-decoration: none;">
                       <h4 class="card-title" style="color: □rgb(55, 58, 60); max-width: 100%; white-space: nowra
                           {{ $book->title }}
                       </h4>
                   </a>
                   <div class="d-flex">
                       @for ($i = 0; $i < 5; $i++)
                           <i class="fas fa-star" style="color: {{ $i < $book->rating ? 'var(--bs-yellow)' : '■rg
                       @endfor
                   </div>
                   <p class="card-text" style="font-size: 0.9rem; color: ■rgb(104, 104, 104);">{{ $book->author }
               </div>
           </div>
       </div>
   @endforeach
```

This code generates a responsive grid of book cards, iterating over the $popularBooks collection. Each card displays the book's image, title, rating (as stars), and author. The title links to a detailed book view page.

BLADE FILES: login-page.blade.php

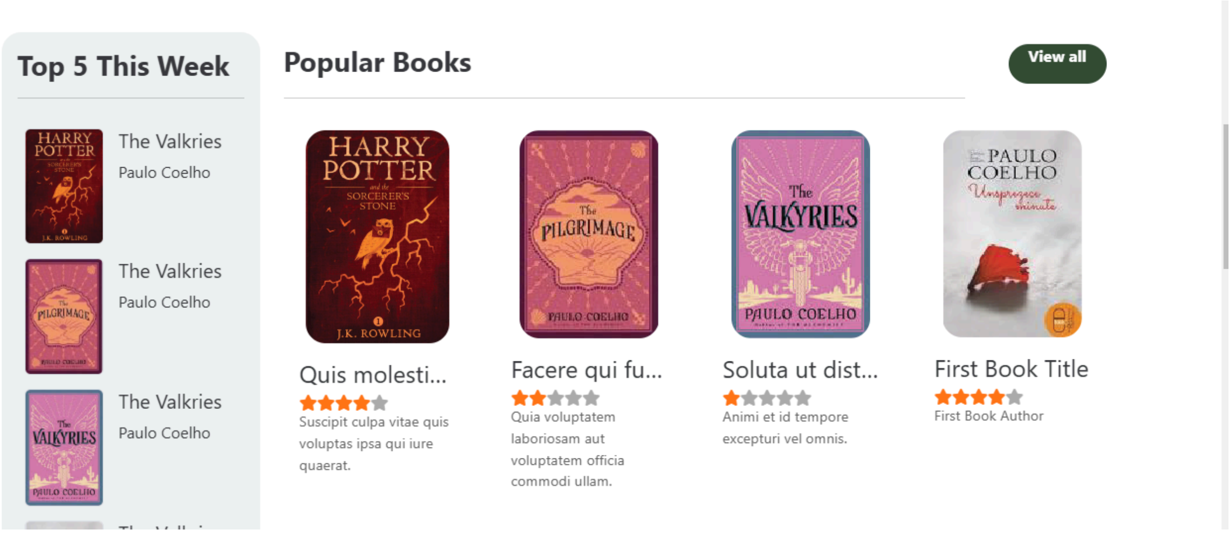


```
<div class="form-wrapper">
    <div class="form-container">
        <!-- Login Form -->
        <div class="form login-form">
            <h2>Sign In</h2>
            <form action="{{ route('login.submit') }}" method="POST">
                @csrf
                <div class="form-control">
                    <input type="text" name="username" required placeholder=" ">
                    <label>Username</label>
                </div>
                <div class="form-control">
                    <input type="password" name="password" class="password" required placeholde
                    <label for="password">Password</label>
                    <i class="fa-solid fa-eye toggle-password" data-target="password"></i>
                </div>
                <button type="submit">Sign In</button>
                <div class="form-help">
                    <div class="remember-me">
                        <input type="checkbox" id="remember-me">
                        <label for="remember-me">Remember me</label>
                    </div>
                    <a href="#">Need help?</a>
                </div>
            </form>
            <div class="new">
                <p>New to LitScope? <a href="#" id="show-signup">Sign up now</a></p>
            </div>
        </div>
```

```
        <!-- Sign-Up Form -->
        <div class="form signup-form">
            <h2>Sign Up</h2>
            <form action="{{ route('signup.submit') }}" method="POST">
                @csrf
                <div class="form-control">
                    <input type="text" name="username" required placeholder=" ">
                    <label>Username</label>
                </div>
                <div class="form-control">
                    <input type="date" name="birthday" required placeholder=" ">
                    <label>Birthday</label>
                </div>
                <div class="form-control">
                    <input type="password" name="password" class="password" required placeholder="
                    <label for="password">Password</label>
                    <i class="fa-solid fa-eye toggle-password" data-target="password"></i>
                </div>

                <button type="submit">Sign Up</button>
            </form>
            <p>Already have an account? <a href="#" id="show-login">Sign in</a></p>
        </div>
    /div>
>
```

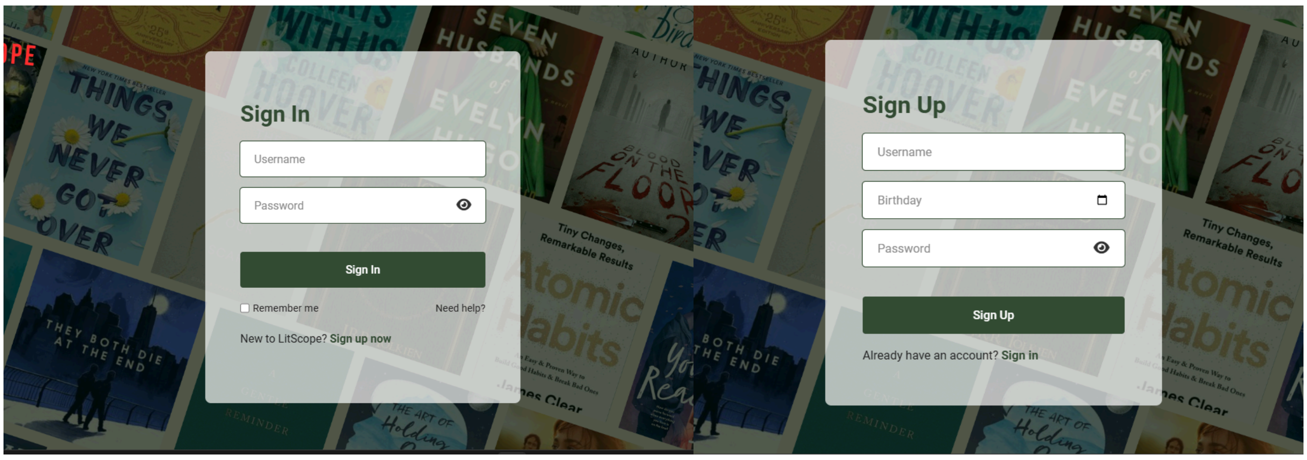
This code contains two forms: one for signing in and one for signing up. The login form requires a username and password, while the sign-up form also asks for a username, birthday, and password. Both forms include CSRF protection and options for remembering the user or needing help. There's a toggle for showing the password in both forms and a link to switch between sign-in and sign-up forms.

RENDERED WEB PAGES: LandingPage.blade.php



Three books are randomly added using the factory, and one specific entry is manually added using the seeder with predefined values.

RENDERED WEB PAGES: login-page.blade.php



This is the sign-in and sign-up form. When the user selects sign-in, their credentials are checked against the database. If the user is verified, they are redirected to the ‘LandingPage’. If they choose to sign up, their data is stored in the database for future use when they sign in.