Task Periodic Manager

**Solution**

**1. User Interface (UI):**

Design Decision: Aimed for a clean, user-friendly interface that places important tasks at the forefront. Leveraged a responsive design to ensure accessibility on both mobile and desktop platforms.

Technologies: Tailwind CSS for styling. It's utility-first and allows rapid UI development with fewer custom styles.

**3. Reactive Components:**

Design Decision: To provide users with a dynamic experience without full-page reloads, reactive components were utilized for task creation, updating, and marking tasks as completed.

Technologies: Laravel Livewire. It integrates seamlessly with Laravel and offers a great developer experience for building dynamic interfaces.

**4. Data Modeling:**

Design Decision: Created normalized database tables for users, task\_groups, and tasks to minimize data redundancy. Established relationships to easily retrieve related data, such as a user's tasks or tasks within a specific group.

Technologies: Laravel's Eloquent ORM for data manipulation and relationships.

**5. Authentication & Authorization:**

Design Decision: Used the laravel Jetstream starter package because of it’s Tailwind CSS based components and authentication and authorization . Additionally, it has role-based access control (RBAC) to ensure tasks can only be managed by their creators.

Technologies: Laravel's built-in authentication system and middleware.

**6. Task Scheduling & Grouping:**

Design Decision: Allowed flexibility in task scheduling with daily, weekly, monthly intervals and specific weekdays. Grouping tasks help in categorizing them and decluttering the user interface.

Technologies: Custom Laravel services and controllers to handle the logic.

**7. Testing:**

Design Decision: Adopted Test-Driven Development (TDD) for critical components, ensuring stability and reduced bugs.

Technologies: PHPUnit integrated with Laravel for unit and feature tests.

**8. Version Control & Deployment:**

Design Decision: Used a version control system for collaboration, tracking changes, and deployment.

Technologies: Git for version control, hosted on GitHub.

**Technologies Used:**

Laravel: The PHP framework provided the core structure for the application, including routing, controllers, models, and views. Laravel also offers powerful features like database migrations and seeding.

Livewire: Livewire components were used to create dynamic and interactive elements within the application, such as task creation forms and real-time updates.

Tailwind CSS: Tailwind CSS classes were employed for styling the user interface. It allowed for the rapid development of a clean and consistent design.

MySQL: MySQL served as the relational database management system to store task and user data. Laravel's Eloquent ORM facilitated database operations.

Authentication: Laravel's built-in authentication system was used for user registration, login, and access control.

Version Control: Git was used for version control, and the application code was hosted on GitHub for collaborative development and code management.

Testing: PHPUnit and Laravel's testing features were used to write unit tests for the application, ensuring code quality and stability.

**Instructions**

Prerequisites

Before you begin, make sure you have the following installed:

PHP (>=7.3)

Composer

Node.js

NPM

MySQL or another database of your choice

**Installation**

Clone the repository:

git clone <https://github.com/Kashan-amd/task-manager.git>

Navigate to the project directory:

cd task-manager

Install PHP dependencies using Composer:

composer install

Install JavaScript dependencies using NPM:

npm install

Create a .env file by copying .env.example and update it with your database configuration:

cp .env.example .env

Generate a new application key:

php artisan key:generate

Migrate the database:

php artisan migrate

Start the development server:

php artisan serve

Visit http://localhost:8000 in your web browser to access the application.

**Tests**

**Test to create a Task**

public function test\_create\_task()

{

// Arrange: Create a task group

$taskGroup = TaskGroup::factory()->create();

// Act: Create a new task

$taskData = [

'user\_id' => User::factory()->create()->id,

'name' => 'Sample Task',

'description' => 'Sample Task Description',

'frequency' => 'daily',

'duration' => 30,

'start\_date' => now(),

'due\_date' => now()->addDays(30),

'completed' => false,

'task\_group\_id' => $taskGroup->id,

];

$task = TaskService::create($taskData);

// Assert: Check if the task was stored in the database correctly

$this->assertDatabaseHas('tasks', $taskData);

// Assert: Check specific attributes of the created task

$this->assertEquals($taskData['name'], $task->name);

$this->assertEquals($taskData['description'], $task->description);

$this->assertEquals($taskData['frequency'], $task->frequency);

$this->assertEquals($taskData['duration'], $task->duration);

$this->assertEquals($taskData['task\_group\_id'], $task->task\_group\_id);

}

**Test to find a specific Task by ID**

public function test\_find\_task\_by\_id()

{

// Create a task using the factory and retrieve its ID

$task = Task::factory()->create();

// Use the built-in `find` method to retrieve the task by its ID

$foundTask = Task::find($task->id);

// Assert that the found task is an instance of the Task model

$this->assertInstanceOf(Task::class, $foundTask);

// Assert that the IDs of the original and found tasks match

$this->assertEquals($task->id, $foundTask->id);

}

**Test to see if a Task is completed and again created**

public function test\_marked\_completed\_and\_recreated()

{

// Arrange: Create a task with a weekly frequency and due date in the past

$task = Task::factory()->create([

'frequency' => 'weekly',

'due\_date' => now()->subWeek(),

]);

// Act: Mark the task as completed

TaskService::markAsCompleted($task->id);

// Assert: Verify that the task is marked as completed and recreated

$this->assertTrue($task->fresh()->completed);

}

**Test to update a Task**

public function test\_update\_task()

{

// Create a task with default data

$task = Task::factory()->create();

// New data for the task

$taskData = [

'name' => 'New Task Name',

'description' => 'New Description for New Task Name',

];

// Update the task

TaskService::update($taskData, $task);

// Assert that the task in the database matches the updated data

$this->assertDatabaseHas('tasks', $taskData);

// Retrieve the updated task from the database

$updatedTask = TaskService::find($task->id);

// Assert that the task name and description have been updated

$this->assertEquals($taskData['name'], $updatedTask->name);

$this->assertEquals($taskData['description'], $updatedTask->description);

}

**Database Diagram**

