サーバーサイドスクリプトⅡ

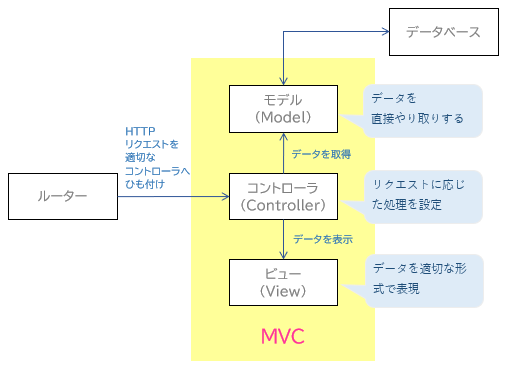
７．DB処理（条件付きSELECT、INSERT）

第7回目では、条件付き抽出とデータ挿入を行っていきます。

下記③⑤の手順を行っていきます。（①②は第5回、④は第6回で作成および設定済みのものを使用）

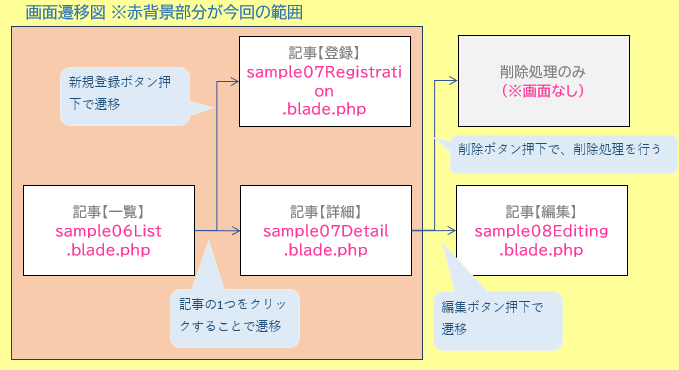
1. **マイグレーション**
2. **モデルの作成**
3. **コントローラの作成**
4. **ルーティングの設定**
5. **ビューの作成**

■図：　MVC



今回の処理範囲は下図の通りです。

　＊記事一覧（sample06List.blade.php）は6回目で作成済み。



７－１． 条件付きSELECT

７－１－１． テーブルのIDを使った条件抽出

第6回目でCRUDに対応したリソースコントローラ（Ｓａｍｐｌｅ０６Controller）を作成しました。今回はこのリソースコントローラの「指定されたリソースの詳細情報を表示」する役割を担う「show」メソッドに処理を追記していきます。

＊リソースコントローラについては、第6回目の資料「■リソースコントローラ　各メソッドの役割」を参照。（下記はshowのみ抜粋）

　■リソースコントローラshowメソッド

|  |  |  |  |
| --- | --- | --- | --- |
| HTTPメソッド | パス | メソッド | 役割 |
| GET | /sample06/{sample06} | **show** | 指定されたリソースの詳細情報を表示 |

■Sample06Controller.php

　＊showメソッド部分のみ記載。（デフォルトコード）

    public function show(string $id)

    {

    }

showメソッドはテーブルの主キーを指定する引数（$id）を持ちます。

テーブルのカラム名がIDの場合は、Eloquentの「 find 」メソッドを使うことで簡単に単一のレコードを取得することが出来ます。

今回は取得したデータを「sample07Detail」ビューに渡します。

■Sample06Controller.php

　＊showメソッド部分のみ記載。

public function show(string $id)

{

$sample\_data = Sample::find($id);

    return view("sample07Detail", compact("sample\_data"));

}

上記により、指定idのデータを取得してビューへ渡すことはできました。

リソースコントローラは、ルーティングの設定もresourceメソッドで行っているため、

「[http://localhost:8000/sample06/{sample06}](http://localhost:8000/sample06/%7bsample06%7d)」でアクセスすることが可能です。

＊{sample06}にはid番号が入る。（例：http://localhost:8000/sample06/1）

ただし、今回はサンプル記事一覧画面から詳細画面へ画面遷移する仕様なので、**サンプル記事一覧画面で各記事のリンクが押下されたら、記事のidを引数に渡し、showメソッドを呼び出す**処理が必要となります。

７－１－２．一覧画面からshowメソッドを呼び出す

6回目で作成した記事一覧画面（sample06List.blade.php）の記事をクリックした場合のリンク先を、ルーティングで設定されたshowメソッドのパスにします。

今回のサンプル環境では、下記のようなルーティング設定となっています。**※６回目で指定済み。（復習）**

■web.php

// Sample06

Route::resource('sample06', Sample06Controller::class);

■ルーティングリストを確認　＊ターミナルで下記コマンドを実行すると、パスの一覧が表示される。

　　php artisan route:list –path=sample06

■パスの一覧　＊サンプル環境（sample06）の場合

|  |  |  |  |
| --- | --- | --- | --- |
| HTTP  メソッド | パス | ルーティング名 | メソッド |
| GET | /sample06 | **sample06.index** | Sample06Controller@index |
| GET | /sample06/create | **sample06.create** | Sample06Controller@create |
| POST | /sample06 | **sample06.store** | Sample06Controller@store |
| GET | /sample06/{sample06} | **sample06.show** | Sample06Controller@show |
| GET | /sample06/{sample06}/edit | **sample06.edit** | Sample06Controller@edit |
| PUT/  PATCH | /sample06/{sample06} | **sample06.update** | Sample06Controller@update |
| DELETE | /sample06/{sample06} | **sample06.delete** | Sample06Controller@delete |

サンプル環境の記事一覧のリンクにshowメソッドのパスを指定します。ただし、今回はパスの文字列ではなく、上記表のルーティング名「sample06.show」を使用します。

■routeヘルパー



　＊第2引数は省略可。メソッドがルートパラメータを受け付ける場合のみ。

サンプル環境では、ルーティング名「sample06.show」とし、パラメータには記事の「id」を指定します。

■sample06List.blade.php

　＊記事の表示部分のみ抜粋。

　＊赤枠部を編集。

@foreach ($list\_data as $data)

    <!-- ▼▼記事1件分の表示▼▼ -->

    <article

        class="row-span-2 bg-white hover:bg-white rounded-md shadow-md hover:shadow-lg transition-shadow overflow-hidden">

        <a href="{{ route('sample06.show', $data->id) }}"

class="block w-full h-full">

            <h3 class="font-bold mt-5 mb-2 px-2">{{ $data->title }}</h3>

            <p class="text-gray-400 text-xs mb-5 px-2"><time datetime="">{{ $data->created\_at }}</time></p>

        </a>

    </article>

    <!-- ▲▲記事1件分の表示▲▲ -->

@endforeach

今回は名前付きルート（ルーティング名）を使用したので、routeヘルパーを使用しましたが、従来のパスで指定する方法も使用可能です。ちなみに、パスを使用する場合は下記のような記述になります。

　＊hrefの値にはrouteヘルパーを使用せずに、「パス＋変数の値」も使用可。

    <a href="sample06/{{ $data->id }}" class="block w-full h-full">

７－１－３． 詳細画面にデータを表示する

サンプル環境では、Sample06Controllerのshowメソッドで、idをキーに1件のデータを抽出し、詳細画面に結果を渡しています。

詳細画面（sample07Detail.blade.php）を作成し、受け取った結果を表示していきます。

詳細画面のデザインは、第3回目の課題articleDetail.blade.phpと同じデザインとしています。

コードは次ページに記載。

■sample07Detail.blade.php

　＊テンプレート部分などは省略。

　＊カラム 「title」 「created\_at」 「body」のデータを表示。

　＊戻るボタン押下時は、サンプル一覧画面に遷移。

<section>

    <div class="bg-white hover:bg-white p-5 mb-10 rounded-md shadow-md">

        <h3 class="text-2xl font-bold leading-10 my-5 px-5 py-2 border-b">{{ $sample\_data->title }}</h3>

        <p class="text-gray-400 text-sm text-right px-3"><time datetime="">{{ $sample\_data->created\_at }}</time></p>

        <div class="flex justify-between py-3">

            <p class="grow w-8/12 text-lg leading-loose px-3 py-5">{{ $sample\_data->body }}</p>

        </div>

    </div>

    <div class="flex justify-end">

        <a href="{{ route('sample06.index') }}"

            class="block w-16 text-white text-center bg-gray-500 hover:bg-gray-400 mr-5 px-3 py-2 rounded-md">戻る</a>

        <form action="" method="">

            <button type="submit"

                class="block w-16 text-white text-center bg-red-600 hover:bg-red-500 mr-5 px-3 py-2 rounded-md">削除</button>

        </form>

        <a href=""

            class="block w-20 text-white text-center bg-pink-600 hover:bg-pink-500 px-3 py-2 rounded-md">編集</a>

    </div>

</section>

ビューが完成したら、実行します。

■実行結果

**＊必ず、一覧画面から遷移すること。**

一覧画面　　　　　　　　　　　　　　　　　　　　　　　詳細画面　＊一覧画面で選択したデータが表示されること。

グラフィカル ユーザー インターフェイス, Web サイト

自動的に生成された説明　　　グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

自動的に生成された説明

詳細画面を表示した際のURLの末尾には、一覧画面で選択したIDが正しく付加されていることを確認する。

■詳細画面URL

＊一覧画面で「ID：２」のデータを選択した場合



７－２． INSERT

一覧画面の「新規投稿」ボタン押下時、記事登録画面に遷移します。

記事登録画面でデータを入力し、「投稿」ボタンを押下すると、DBにデータを登録する処理を作成します。

　■使用メソッド＆ルーティング名

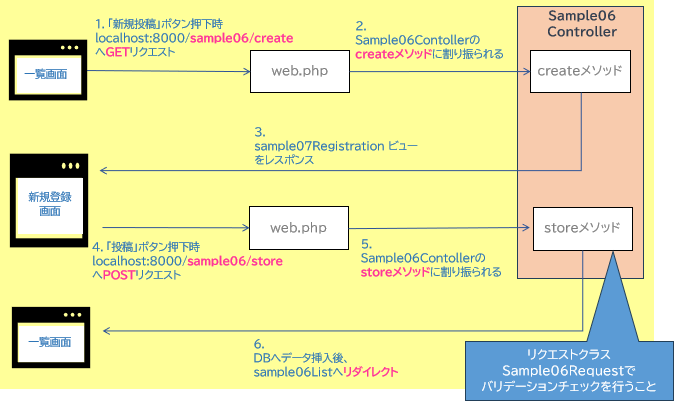
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| HTTPメソッド | パス | ルーティング名 | メソッド | 役割 |
| GET | /sample06/create | **sample06.create** | **create** | リソースを新規作成するためのフォームを表示 |
| POST | /sample06 | **sample06.store** | **store** | リソースを新規作成 |

記事登録画面を表示する、すなわち一覧画面で「新規投稿」ボタンが押下された時の処理は、createメソッドが担います。

記事登録画面で「投稿」ボタンが押された時、POSTデータを受け取りDBへ登録する処理はstoreメソッドが担います。

処理の流れは下図の通りです。

■一覧画面～新規投稿画面　処理の流れイメージ



次項より、それぞれの処理を作成していきます。

７－２－１． 記事登録画面を表示する

記事一覧画面の「新規投稿」ボタン押下時の設定を追加します。

■sample06List.blade.php

　＊新規投稿ボタン部分のみ抜粋。

　＊赤枠部を編集。

<div class="flex justify-end mb-10">

    <a href="{{ route('sample06.create') }}"

        class="block w-24 text-white text-center bg-sky-600 hover:bg-sky-500 px-3 py-2 rounded-md">新規投稿</a>

</div>

リンクの設定が完了したので、処理を担当するSample06Controllerのcreateメソッドを実装していきます。

■Sample06Controller.php

　＊createメソッドのみ記載。

public function create()

{

    //

    return view('sample07Registration');

}

＊viewメソッドを使用して、新規登録画面（sample07Registration.balde.php）を表示します。

最後に、新規登録画面（sample07Registration.blade.php）を作成します。

■sample07Registration.blade.php

　＊テンプレート部分などは省略。＊赤枠部分…「戻る」ボタン押下時は一覧画面に遷移。

<section>

    <form action="" method="" enctype="multipart/form-data">

@csrf

        <div class="bg-white hover:bg-white p-5 mb-10 rounded-md shadow-md">

            <div class="my-5 px-5 py-2 border-b">

                <label class="block text-gray-500 text-sm uppercase" for="title">title</label>

                <input type="text" name="title" id="title"

                    class="w-full text-2xl font-bold leading-10 border border-gray-300 rounded-md" value="">

            </div>

            <div class="flex justify-between py-3">

                <div class="w-4/12 mr-5">

                    <label class="block text-gray-500 text-sm uppercase" for="image">image file</label>

                    <input type="file" name="image" id="image"

                        class="w-full h-80 text-xs px-3 py-2 border border-gray-300 rounded-md" value="">

                </div>

                <div class="grow">

                    <label class="block text-gray-500 text-sm uppercase" for="body">body</label>

                    <textarea name="body" class="w-full h-80 text-lg px-3 py-2 border border-gray-300 rounded-md resize-none"></textarea>

                </div>

            </div>

        </div>

        <div class="flex justify-end">

            <a href="{{ route('sample06.index') }}"

                class="block w-16 text-white text-center bg-gray-500 hover:bg-gray-400 mr-5 px-3 py-2 rounded-md">戻る</a>

            <button type="submit"

                class="block w-20 text-white text-center bg-sky-600 hover:bg-sky-500 px-3 py-2 rounded-md">投稿</button>

        </div>

    </form>

</section>

新規登録画面のビューが完成したら、実行します。

一覧画面から、「新規登録」ボタン押下時、新規登録画面が表示されることを確認してください。

■サンプル記事一覧画面　　　　　　　　　　　　　　　　　　　　■サンプル新規登録画面

**新規登録ボタン押下**

グラフィカル ユーザー インターフェイス, Web サイト

自動的に生成された説明　　　　グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

＊サンプル新規登録画面のURLが「http://localhost:8000/**sample06/create**」となっていることを確認する。

７－２－２． 記事をDBに登録する

サンプル新規登録画面の「投稿」ボタン押下時、入力値のエラーチェックを行います。

■サンプル新規登録画面

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

**投稿ボタン押下**

まずは「投稿」ボタン押下時、「http://localhost:8000/sample06/store」へPOSTリクエストするようにします。

新規登録画面はフォーム入力なので、フォームのaction属性とmethod属性に値を設定しましょう。

■sample07Registration.blade.php

　＊form開始タグの箇所のみ記載。その他省略。

    <form action="{{ route('sample06.store') }}" method="POST" enctype="multipart/form-data">

        @csrf

　　　　　　～～省略～～

    </form>

ルーティングの設定では、「sample06.store」が指定されたとき、Sample06Controllerのstoreメソッドが処理を行うように設定されています。また、storeでは、デフォルトで引数としてリクエストデータ受け取るようになっています。

■storeメソッド＆ルーティング名 ＊その他パスの一覧は本資料「■パスの一覧」を参照。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| HTTPメソッド | パス | ルーティング名 | メソッド | 役割 |
| POST | /sample06 | **sample06.store** | **store** | リソースを新規作成 |

■Sample06Controller.php　＊storeメソッドのみ抜粋

public function store(Request $request)

{

    //

}

storeメソッドでリクエストデータを取得し、DBのsamplesテーブルにデータ登録します。

メソッド個別の説明は後述しますが、まずは処理の流れを確認しましょう。

■Sample06Controller.php

　＊storeメソッドのみ記載。

　＊リクエストデータのバリデーションチェックは未実装。（後の項で追加実装）

登録が完了したら、一覧画面へ遷移したいので、

redirectメソッドを使用して、一覧画面のルーティングを指定。

モデルのインスタンスを作成。

インスタンスの各プロパティ（テーブルの列に相当）へリクエストデータをセット。

save()　メソッドを使用。

＊インスタンスの内容をDBへ保存。

トランザクション処理

public function store(Request $request)

{

    // トークン再生成（二重送信の防止）

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

    // モデルクラスのインスタンスを作成

    $sampleDao = new Sample();

    // リクエストデータの取得

    $sampleDao->title = $request->title;

    $sampleDao->body = $request->body;

    // DBにデータの保存（トランザクションメソッド使用）

    DB::transaction(function () use ($sampleDao) {

        $sampleDao->save();

    });

    // 一覧画面にリダイレクト

    return redirect()->route("sample06.index");

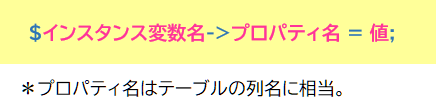
}

storeメソッドでは、モデルのインスタンスに、受け取ったリクエストデータをセットし、そのインスタンスの内容をDBへ保存します。

■モデルのインスタンス化



■インスタンスのプロパティへ値をセット



今回リクエストデータをインスタンス変数のプロパティにセットします。

続いて、今回はDB「登録」を行いますので、トランザクション処理を用います。

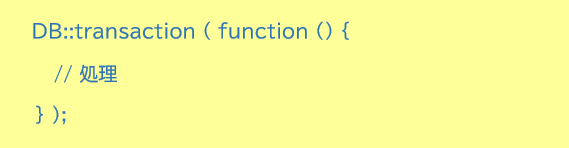
Laravelでトランザクション処理を行う方法は2通りあります。どちらもDBファサード（＊）を使用します。前述のサンプルコードは①の方法を採用しています。

|  |  |
| --- | --- |
| 警告 単色塗りつぶし | **（＊） ファサードとは**  クラスをインスタンス化しなくてもstaticメソッドのようにメソッドを実行できるようにしてくれる機能のこと。 |

【方法 １】 transactionメソッドを利用する

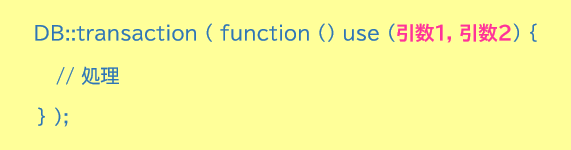
＊引数あり、引数なし　2パターンの構文あり。

■transactionメソッド（引数なし）



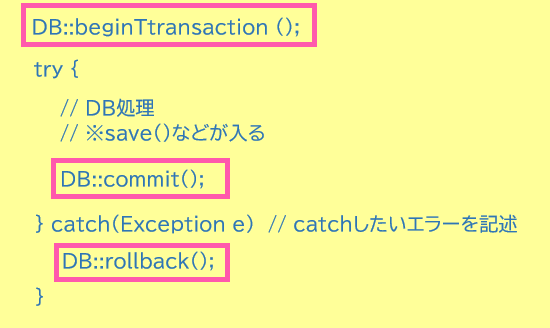
　　■transactionメソッド（引数あり）

　　　＊サンプルコードは、こちらを使用。



【方法 ２】 手動でトランザクションを行う

■beginTransaction（）、commit()、rollback()メソッドを使用する



DBへの保存には、saveメソッドを使用します。

　■saveメソッド

|  |  |
| --- | --- |
| メソッド名 | 説明 |
| save() | モデルに属性セットした後、データベースに保存する |

データの保存が完了したら、記事一覧画面へ遷移します。

ただし、viewメソッドでの指定だと、画面は表示できますが、DBからのデータ取得ができません。

また、storeメソッド内にDBからデータ取得の処理を実装することも効率的ではありません。なぜなら、データ一覧の取得は一覧画面の表示を担っているindexメソッドにて既に実装しています。

よって、今回はルート**「sample06.index」にリダイレクト**することにより、indexメソッドの処理により一覧画面が表示されるようにします。

リダイレクトには、redirectメソッドを使用します。

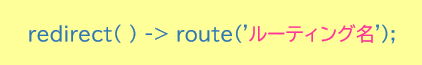
■redirectメソッド（URL指定）



■redirectメソッド（名前付きルート指定）

　＊サンプルではこちらを使用。

　＊redirectメソッドの引数はナシで呼び出し、routeメソッドでルーティング名を指定。



実装が完了したら、記事登録画面で「タイトル」と「本文」を入力し、「投稿」ボタンを押します。

DBへの登録が完了し、記事一覧ページに遷移します。また、登録したデータが記事一覧の最終ページ（末尾）に表示されていることを確認します。

■新規登録画面　　　　　　　　　　　　　　　　　　　　■記事一覧画面（最終ページ）

グラフィカル ユーザー インターフェイス

自動的に生成された説明　　　　　　グラフィカル ユーザー インターフェイス

自動的に生成された説明

７－２－３． DB登録前に入力値のチェックを行う

DB登録までは実装しましたが、現段階では入力値のチェックを行っていません。

よって、フォームに何も入力せずに登録しようとすると、SQLエラーが発生します。これは、テーブル定義の際に、「body」列のみNULL許可としていたためです。「title」列はLaravel初期値のNOT NULLのままとなっています。

■記事登録画面

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

フォームに何も入力せずに

「投稿」ボタンを押す。

■エラー画面（エラーメッセージ）



同じくエラー画面には、エラー行も記載されています。

■エラー画面（エラー行）



エラー行

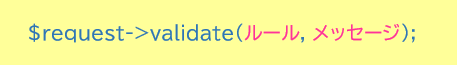
DBへのデータ保存時にエラーが発生していることが分かる

ファイル名（フルパス）　と行数

このアプリケーションエラーを解消するため、DB登録を行う前、更にはインスタンス変数にリクエストデータをセットする前に、バリデーションチェックを行います。validateメソッドは第4回目で使用しました。ただし、第4回目では第1引数のルールのみ指定していました。今回は第2引数のメッセージも指定して使用します。

■validateメソッド（第2引数の指定あり）

＊$requestに対してバリデーションチェックを行う場合



ルールもメッセージも配列で指定可能ですが、メソッド内に記載すると可読性が失われること、およびモデルに対するルールとメッセージであることから、今回はモデルクラス内に記述します。

サンプルでは「タイトル」「本文」ともに、必須の仕様とします。

※「本文」はDBのテーブル定義ではNULL許容としていますが、画面入力時に未入力の場合は入力を促す仕様とします。

■Sample.php

　＊赤枠部を追加。

　＊**モデルクラス内にstaticプロパティとして定義**。

class Sample extends Model

{

    use HasFactory;

    use SoftDeletes;                // 論理削除の場合

    // モデル名をテーブル名の単数形とした場合は省略可↓

    protected $table = 'samples';  // テーブル名

    // ルール用のプロパティ

    public static $rules = [

        "title" => ["required"],

        "body"  => ["required"],

    ];

    // メッセージ用のプロパティ

    public static $messages = [

        "title.required" => "タイトルが入力されていません",

        "body.required"  => "本文が入力されていません",

    ];

}

モデルクラスで定義したプロパティを利用し、コントローラでバリデーションチェックを行います。

■Sample06Controller.php

　＊storeメソッド部分のみ記載。

　＊赤枠部を追加。

モデルクラスはインスタンス化しているので、$sampleDaoを使用。

staticプロパティを使用するので、

「インスタンス変数::プロパティ名」で記述。

public function store(Request $request)

{

    // トークン再生成（二重送信の防止）

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

    // モデルクラスのインスタンス化

    $sampleDao = new Sample();

    //バリデーションチェック

    $request->validate($sampleDao::$rules, $sampleDao::$messages);

    // リクエストデータの取得

    $sampleDao->title = $request->title;

    $sampleDao->body = $request->body;

    // DBにデータの保存

    // 手動トランザクション

    DB::transaction(function () use ($sampleDao) {

        $sampleDao->save();

    });

    // 一覧画面にリダイレクト

    return redirect()->route("sample06.index");

}

バリデーションチェックが完了したので、エラーがあった場合はエラーメッセージを画面（ビュー）に表示します。

■sample07Registration.blade.php

　＊<form>タグ内の該当箇所のみ抜粋。

　＊赤枠部を追加。

<div class="my-5 px-5 py-2 border-b">

    <label class="block text-gray-500 text-sm uppercase" for="title">title</label>

    <input type="text" name="title" id="title"

        class="w-full text-2xl font-bold leading-10 border border-gray-300 rounded-md" value="">

    @error('title')

        <p class="text-sm text-red-600 my-2">

            {{ $message }}

        </p>

    @enderror

</div>

<div class="flex justify-between py-3">

    <div class="w-4/12 mr-5">

        <label class="block text-gray-500 text-sm uppercase" for="image">image file</label>

        <input type="file" name="image" id="image"

            class="w-full h-80 text-xs px-3 py-2 border border-gray-300 rounded-md" value="">

    </div>

    <div class="grow">

        <label class="block text-gray-500 text-sm uppercase" for="body">body</label>

        <textarea name="body" class="w-full h-80 text-lg px-3 py-2 border border-gray-300 rounded-md resize-none"></textarea>

        @error('body')

            <p class="text-sm text-red-600 my-2">

                {{ $message }}

            </p>

        @enderror

    </div>

</div>

追加が完了したら、実行します。

記事登録画面に入力せず「投稿」ボタンを押すと、エラーメッセージが表示されることを確認してください。

■記事登録画面

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

フォームに何も入力せずに

「投稿」ボタンを押す。

■記事登録画面

**＊エラーメッセージが表示されることを確認。**

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

最後に、「タイトル」「本文」のどちらかが空白の場合、下記の処理となるようにします。

・空白・・・エラー表示

・入力アリ・・・**入力値を保持　＊入力値の保持にはoldヘルパーを使用します。**

■oldヘルパー



bladeテンプレート上で{{ old('name属性の値') }}とするだけでnameの入力値が取れます。

■sample07Registration.blade.php

　＊タグを一部抜粋。

　＊赤枠部を追加。

<input type="text" name="title" id="title"

    class="w-full text-2xl font-bold leading-10 border border-gray-300 rounded-md"

    value="{{ old('title') }}">

～～中略～～

<textarea name="body" class="w-full h-80 text-lg px-3 py-2 border border-gray-300 rounded-md resize-none">{{ old('body') }}</textarea>

「タイトル」「本文」の片方のみ入力して、動作確認をします。

■記事登録画面

　＊「タイトル」のみ入力して「投稿」ボタン押下した場合

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

「本文」は未入力だったため、エラーメッセージが表示される。

「タイトル」には入力値が保持されている。

■記事登録画面

　＊「本文」のみ入力して「投稿」ボタン押下した場合

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

「本文」には入力値が保持されている。

「タイトル」は未入力だったため、エラーメッセージが表示される。

入力値保持の動作確認が完了したら、再度、正しく値が入力された場合の動作確認も行います。

＊コードを変更したら変更した箇所のみではなく、変更前に正しく動いていた動作も再確認すること。

＊正しく入力画面で入力した場合は、DBに記事が登録され一覧画面にも表示されること。

7回目は以上です。