サーバーサイドスクリプトⅡ

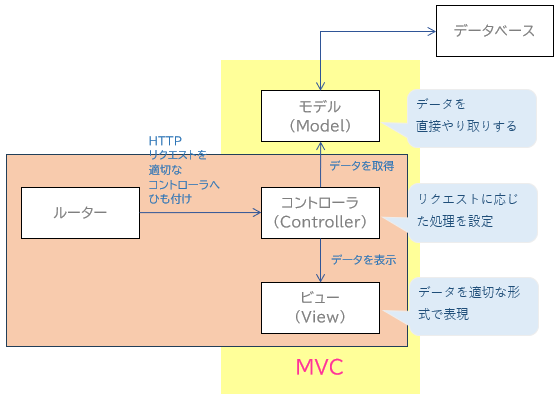
４．リクエストとバリデーション

４－１． フォームリクエスト

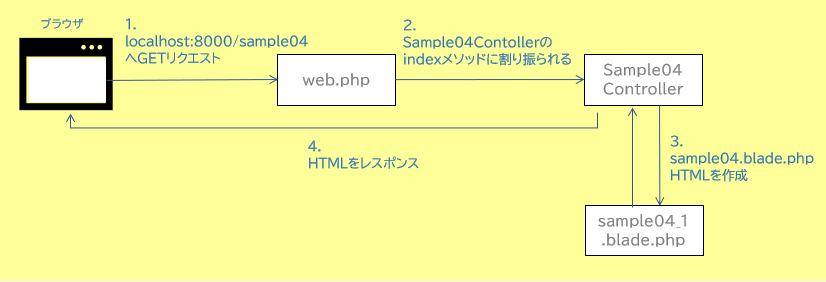
1. マイグレーション
2. モデルの作成
3. **コントローラの作成**
4. **ルーティングの設定**
5. **ビューの作成**

リクエストを送信するビューを作成して、実際にパラメータを取得してみましょう。

■図：　今回の範囲（赤枠部分）　MVCのVC



■サンプル４　入力画面表示処理の流れ



■Sample０４Controller

class Sample04Controller extends Controller

{

    //

    public function index()

    {

        return view('sample04\_1');

    }

}

フォームの作成を行います。sample04\_1.blade.phpを作成し、フォームを配置します。FormのactionはURI（<http://localhost:8000/>以降）を指定。データ形式はPOSTとしてください。

■sample04\_1.blade.php

<form action="sample04" method="POST">

    @csrf

    <div class="flex justify-between flex-col lg:flex-row items-stretch mb-5">

        <div class="flex flex-col w-full lg:w-6/12 mr-5">

            <!-- ▼▼氏名▼▼ -->

            <div class="flex flex-col w-full mb-5">

                <label class="text-gray-400 text-sm">氏名</label>

                <input type="text" name="name" value=""

                    class="w-full h-10 px-3 text-lg border-2 border-gray-200 rounded-md outline-none">

            </div>

            <!-- ▲▲氏名▲▲ -->

            <!-- ▼▼メールアドレス▼▼ -->

            <div class="flex flex-col w-full mb-5">

                <label class="text-gray-400 text-sm">メールアドレス</label>

                <input type="text" name="email" value=""

                    class="w-full h-10 px-3 text-lg border-2 border-gray-200 rounded-md outline-none">

            </div>

            <!-- ▲▲メールアドレス▲▲ -->

        </div>

    </div>

    <div class="flex justify-end">

        <button type="submit"

            class="text-white text-center leading-10 bg-pink-600 px-10 hover:bg-pink-500 rounded-md">確認</button>

    </div>

</form>

４－１－１． CSRFトークンチェック

Laravelでは初期設定としてすべてのリクエストに対し、CSRFトークンチェックを行います。CSRFはWebサイトにおける脆弱性の1つであり、Laravelではこの脆弱性の対策としてトークンチェックを行うことでその攻撃から防ぐ仕組みを持っています。

Laravelではセッションを利用して、アプリケーション固有のトークンを生成し、そのトークンが有効かどうかを判定して自身のアプリケーションから送信されたものであることを認識します。

この判定は「app/Http/Middleware/VerityCsrfToken」で行われます。

**CSRF対策で重要なことは、クライアントからトークンを送って貰うことが必要である**ということです。

Laravelでは**Bladeテンプレートのディレクティブに@csrfを用意していますので、これを設定するだけ**で対策が完了できます。

＊ディレクティブの説明は第3回の資料にあります。

先ほど作成したbladeの赤枠部分に該当します。

■sample04\_1.blade.php から抜粋

<form action="sample04" method="POST">

    @csrf

    <div class="flex justify-between flex-col lg:flex-row items-stretch mb-5">

～中略～

</form>

ルーティングの設定をして、sample04を表示してみます。

■web.php ルーティングの追加

|  |  |
| --- | --- |
| リスエスト形式 | Get |
| URI | https://localhost:8000**/sample04** |
| 呼び出しクラス・メソッド | Sample04Controller の　index　メソッド  ※use文も忘れずに。 |

■sample04

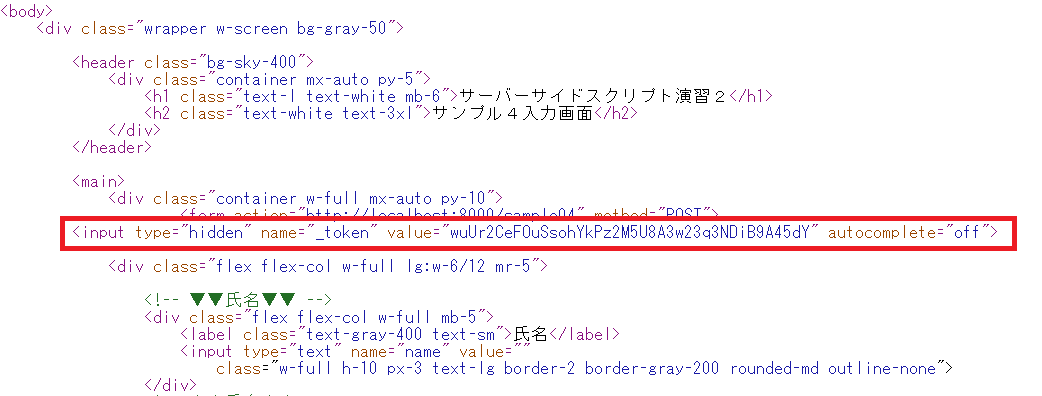
グラフィカル ユーザー インターフェイス

中程度の精度で自動的に生成された説明

この画面より、[右クリック]-[ページのソースを表示]としてみてください。

赤枠のようなCSRF対策用の「 Token（トークン） 」と呼ばれるランダムな文字列が非表示フィールドとして挿入されていることを確認します。

＊文字列はランダムなので、実行環境により異なります。



４－２． リクエストデータの取得

sample04\_1.blade.phpのフォームからデータが送信されたら、呼び出されるコントローラのメソッドを作成します。コントローラ（Sample04Controller）は既に存在するので、リクエストデータの処理を行うメソッドを追加します。

**＊メソッドの引数にRequestクラスを指定します。引数requestへのフォームデータの注入はLaravelにより自動的に注入されます。**

■Sample０４Controller

**ここの説明は次のページ**

use Illuminate\Http\Request;

class Sample04Controller extends Controller

{

    // indexメソッドは記載省略

    　　/\*\*

     \* 入力POSTデータ取得

     \*/

    public function post(Request $request)

    {

 //リクエストデータを変数に格納

        $result['name'] = $request->input('name');

        $result['email'] = $request->input('email');

// 入力値を渡して、sample04\_2確認画面へ

        return view('sample04\_2', compact('result'));

    }

}

４－２－１． Requestクラスのinputメソッド

**＊リクエストデータを取り出すには、「 Request 」クラスの「 input 」メソッドを使います。**

■Requestクラスのinputメソッド

（Requestクラスの変数が$requestの場合で記載）

**ロゴ

自動的に生成された説明**

４－２－２． 二重送信の防止

リクエストのたびにCSRFトークンを作り直すことで、コンピュータからの連続リクエストや誤操作による二重送信を防止することができます。

■CSRFトークンの破棄



Sample04Controllerのpostメソッドに追記します。

■Sample０４Controller

    public function post(Request $request)

    {

        // CSRFトークンの破棄

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

 //リクエストデータを変数に格納

        $result['name'] = $request->input('name');

        $result['email'] = $request->input('email');

// 入力値を渡して、sample04\_2確認画面へ

        return view('sample04\_2', compact('result'));

    }

ルーティングの設定をして、sample04を表示してみます。

■web.php ルーティングの追加

|  |  |
| --- | --- |
| リスエスト形式 | post |
| URI | https://localhost:8000**/sample04** |
| 呼び出しクラス・メソッド | Sample04Controller の　post　メソッド |

データ確認表示画面(sample04\_2.blade.php)を作成します。

＊コードが長いので、次ページに記載。

■sample04\_2.blade.php

　＊テンプレート使用。contentの内容のみ記載。

<div class="flex justify-between flex-col lg:flex-row items-stretch mb-5">

    <div class="flex flex-col w-full lg:w-6/12 mr-5">

        <!-- ▼▼氏名▼▼ -->

        <div class="flex flex-col w-full mb-5">

            <label class="text-gray-400 text-sm">氏名</label>

            <p name="name" class="w-full h-10 px-3 text-lg border-2 border-gray-200 rounded-md outline-none">{{ $result['name'] }}</p>

        </div>

        <!-- ▲▲氏名▲▲ -->

        <!-- ▼▼メールアドレス▼▼ -->

        <div class="flex flex-col w-full mb-5">

            <label class="text-gray-400 text-sm">メールアドレス</label>

            <p name="email" value=""

                class="w-full h-10 px-3 text-lg border-2 border-gray-200 rounded-md outline-none">{{ $result['email'] }}</p>

        </div>

        <!-- ▲▲メールアドレス▲▲ -->

    </div>

</div>

<div class="flex justify-end">

    <a href="sample04"

        class="text-white text-center leading-10 bg-gray-500 px-10 mr-10 hover:bg-gray-400 rounded-md">戻る</a>

    <button type="button"

        class="text-white text-center leading-10 bg-pink-600 px-10 hover:bg-pink-500 rounded-md">送信</button>

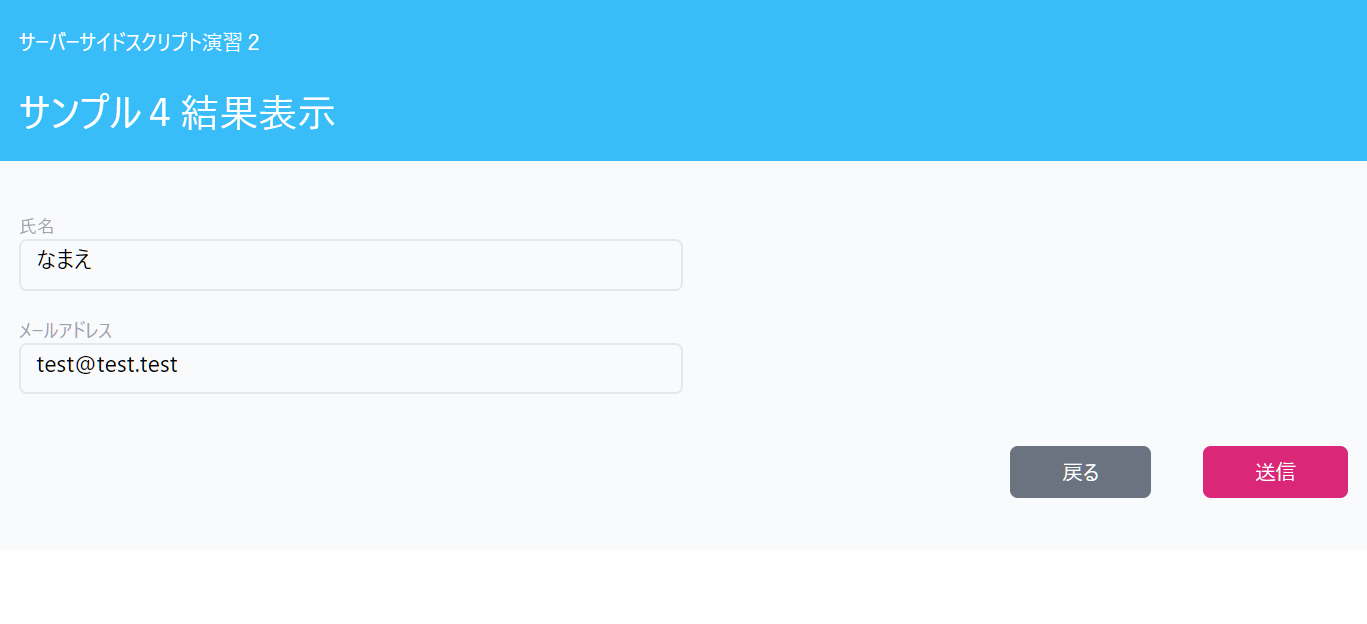
</div>

実行して、ブラウザからsample04にアクセスしてください。最初の表示ではルーティングにより、入力画面が表示されます。入力画面で値を入力し、[確認] ボタンを押下します。

データ確認表示画面(sample04\_2.blade.php)に入力画面で入力した値が表示されていることを確認してください。

■データ確認表示画面(sample04\_2.blade.php)

　＊入力画面からの遷移時のみ（POSTデータ送信があった場合のみ）表示される。



４－３． バリデーションチェック

入力から確認画面までの一連の処理は完了しました。ただし、入力値のチェックが行われていません。

続いて、リクエストデータの入力チェックを行います。

４－３－１． バリデーションルールの定義（コントローラ内で定義）

サンプルではそれぞれに、下記のルールを設定します。

＊名前…ルール「必須」

＊Eメール…ルール「必須」「Eメール形式」

構文などは次ページに記載があります。

■Sample０４Controller

    public function post(Request $request)

    {

        // CSRFトークンの破棄

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

        $request->validate(

            [

                'name'  => ['required'],

                'email' => [

                    'required', 'email'

                ],

            ]

        );

 //リクエストデータを変数に格納

        $result['name'] = $request->input('name');

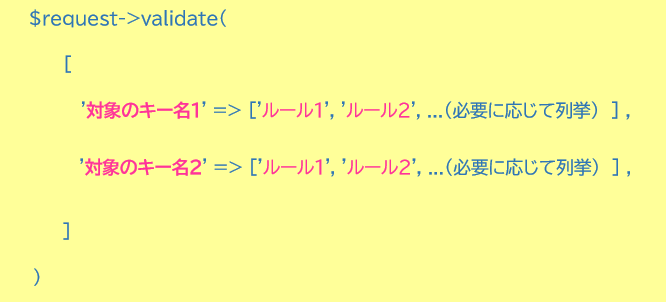
        $result['email'] = $request->input('email');

// 入力値を渡して、sample04\_2確認画面へ

        return view('sample04\_2', compact('result'));

    }

■validateメソッド



＊対象のキー名にはPOSTデータの名前を記載します。

　入力画面のinputタグのname属性を確認してください。

＊バリデーションエラーの場合、画面遷移は行われません。

■指定できるルール（抜粋）

|  |  |
| --- | --- |
| 指定値 | 内容 |
| integer | 整数かどうか |
| numeric | 数値かどうか |
| min : 値 | 指定値より小さいか |
| max : 値 | 指定値より大きいか |
| size : 値 | 指定の文字数かどうか |
| url | URL形式かどうか |
| email | Eメールアドレス形式かどうか |
| file | ファイル形式かどうか |
| Image | 画像ファイルかどうか |
| required | 値が存在するかどうか |
| regex : パターン | 正規表現で検証する |

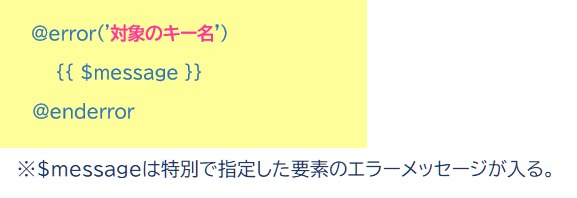
４－３－２． バリデーションエラーメッセージの表示

バリデーションエラー時のメッセージを表示します。

エラーが発生した場合、入力画面にエラーメッセージが表示されるように、入力画面へ追記を行います。

Bladeテンプレートのディレクティブに@errorが用意されていますので、これを設定します。

■@errorディレクティブ



サンプル入力画面にエラーメッセージを追記します。

＊コードは長いので次ページに記載。

■sample04\_1.blade.php

※必要箇所のみ抜粋。赤枠部分が追加部分。

サンプルでは、それぞれの項目の入力ボックスの下部にエラーメッセージが出るように追記。

<!-- ▼▼氏名▼▼ -->

<div class="flex flex-col w-full mb-5">

    <label class="text-gray-400 text-sm">氏名</label>

    <input type="text" name="name" value=""

        class="w-full h-10 px-3 text-lg border-2 border-gray-200 rounded-md outline-none">

    @error('name')

        <p class="text-xs text-pink-600">{{ $message }}</p>

    @enderror

</div>

<!-- ▲▲氏名▲▲ -->

<!-- ▼▼メールアドレス▼▼ -->

<div class="flex flex-col w-full mb-5">

    <label class="text-gray-400 text-sm">メールアドレス</label>

    <input type="text" name="email" value=""

        class="w-full h-10 px-3 text-lg border-2 border-gray-200 rounded-md outline-none">

    @error('email')

        <p class="text-xs text-pink-600">{{ $message }}</p>

    @enderror

</div>

<!-- ▲▲メールアドレス▲▲ -->

コントローラの追記、およびエラーメッセージの表示が完了したら、ブラウザで実行してみてください。

このとき、誤った値を入力し、[確認]ボタンを押下してみます。

サンプルでは、名前を未入力、メールアドレスをメール形式以外とします。

■実行結果（入力エラーを発生させる）

　＊赤枠部分のエラーメッセージが表示される。

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

それぞれの項目に応じたエラーメッセージが表示されました。

ただし、メッセージ内容は指定していないため、デフォルトのエラーメッセージ（英語）が表示されています。メッセージの編集は後の項で行います。

４－３－３． バリデーションルールの定義（フォームリクエスト）

バリデーションをコントローラのvalidateメソッドからフォームリクエストでの処理に変更します。

リクエストクラスを作成します。リクエストクラスはコマンド実行により作成してください。

■リクエストクラス作成コマンド



サンプルでは、Sample04Requestという名前で作成します。

下図のように「～～successfully」と表示されればOK。

■実行コマンド



作成したファイルは「プロジェクトフォルダ\app\Http\Request」の配下に作成されています。続いて作成したファイルを編集しますので、Sample04Requestファイルを開いてください。

「authorize」と「rules」というメソッドが用意されています。

■Sample０４Request

　＊コメント部分などは省略

class Sample04Request extends FormRequest

{

    public function authorize(): bool

    {

        return false;

    }

    public function rules(): array

    {

        return [

            //

        ];

    }

}

authorizeメソッドはフォーム処理へのアクセス権がある場合はtrue、ない場合はfalseを返します。デフォルトではfalseとなっているので、サンプルではリクエストパスが「sample04」の場合のみアクセス許可とします。

■Sample０４Request

＊authorizeメソッド修正

public function authorize(): bool

{

    if ($this->path() == 'sample04') {

        return true;

    } else {

        return false;

    }

}

続いて、バリデーションルールの指定をrulesメソッドで行います。returnにルールの配列を指定します。サンプルでは、先ほどのコントローラクラスのvalidateメソッドで使用した配列と同様にします。

■Sample０４Request

＊rulesメソッド追記　[ ]の数に注意。今回はメソッド内の引数ではなく、returnで配列を返す。

public function rules(): array

{

    return [

        //

        'name'  => ['required'],

        'email' => [

            'required', 'email'

        ],

    ];

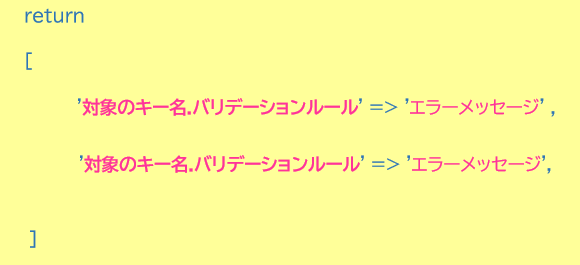
}

続いて、エラーメッセージのカスタマイズを行います。Sample04Requestクラス内のmessagesメソッドで指定します。returnに配列形式で、キー名.バリデーションルールごとにエラーメッセージを指定します。

＊デフォルトではmessagesメソッドの記載がないので、追記します。エラーメッセージを指定するメソッド名はmessagesです。メソッド名が違っていると正しく呼び出されません。

■returnの配列

　＊下記のように必要に応じて列挙する。



■Sample０４Request

＊messagesメソッドごと追記。

public function messages()

{

    return [

        'name.required' => '名前が入力されていません',

        'email.required'   => 'メールアドレスが入力されていません',

        'email.email'   => 'メールアドレスの形式が間違っています',

    ];

}

サンプルでは以下のメッセージを指定しています。

＊氏名（必須）・・・「名前が入力されていません」

＊Eメール（必須）…「メールアドレスが入力されていません」

＊Eメール（Eメール形式）…「メールアドレスの形式が間違っています」

４－３－４． 作成したリクエストクラスの利用

リクエストクラスの設定が完了したので、コントローラクラスから使用します。

リクエストデータの取得時に使用できるように書き換えます。

サンプルではpostメソッドを使用していますので、postメソッドの引数をRequestからSample04Requestに変更します。また、validateメソッドは不要になるのでコメントアウトします。

■Sample０４Controller

　＊赤枠部分が変更箇所（関連メソッド以外は記載省略）

use App\Http\Requests\Sample04Request;

class Sample04Controller extends Controller

{

// 中略

    public function post　(　Sample04Request 　$request)

    {

        // CSRFトークンの破棄

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

        // $request->validate(-中略- )メソッドは不要なのでコメントアウト

        //リクエストデータを変数に格納

        $result['name'] = $request->input('name');

        $result['email'] = $request->input('email');

        // 入力値を渡して、sample04\_2確認画面へ

        return view('sample04\_2', compact('result'));

    }

}

修正が完了したら実行してみてください。

■実行画面

　＊名前・メールアドレスともに未入力の場合。

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

■実行結果

＊メールアドレスのみ、入力しているがメールアドレス形式ではない場合

　メールアドレス未入力時とメッセージが変わっていることを確認。

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