サーバーサイドスクリプトⅡ

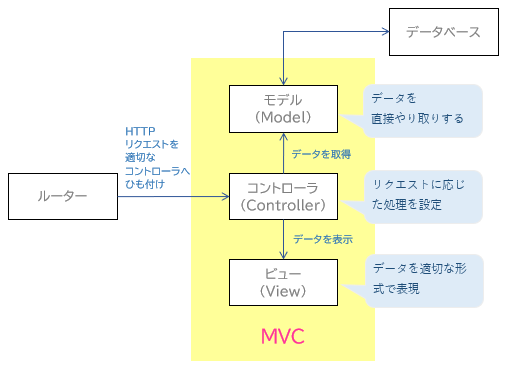
８．DB処理（UPDATE）

第８回目では、データ更新を行っていきます。

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**

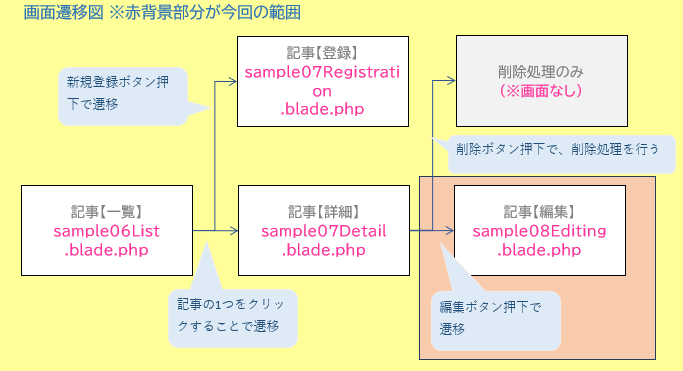
■図：　MVC



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、

記事詳細（sample07Detail.blade.php）は7回目で作成済み。



８－１． UPDATE

記事詳細の「編集」ボタン押下時、記事編集画面に遷移します。

記事編集画面でデータを入力し、「更新」ボタンを押下すると、DBの該当データを更新する処理を作成します。

■使用メソッド＆ルーティング名

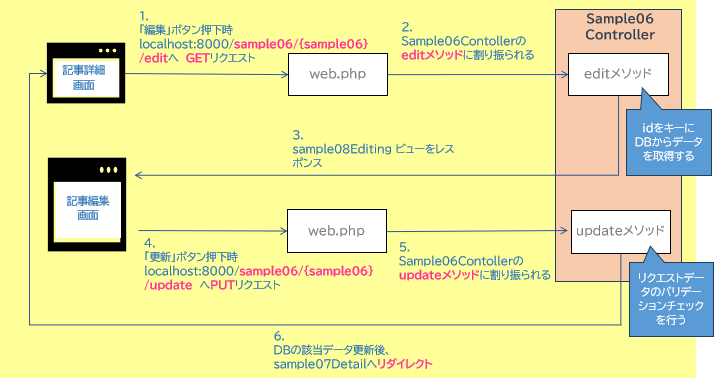
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| HTTP  メソッド | パス | ルーティング名 | メソッド | 役割 |
| GET | /sample06/{sample06}/edit | **sample06.edit** | **edit** | 指定されたリソースの編集フォームを表示 |
| PUT  /PATCH | /sample06/{sample06} | **sample06.update** | **update** | 指定されたリソースを更新  PUT：指定された情報を更新する  PATCH：指定された情報を部分的に更新する |

記事編集画面を表示する、すなわち記事詳細画面で「編集」ボタンが押下された時の処理は、editメソッドが担います。

記事編集画面で「更新」ボタンが押された時、PUTデータを受け取り、DBを更新する処理はupdateメソッドが担います。

処理の流れは下図の通りです。

■記事詳細画面～記事編集画面　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。

８－１－１． 記事編集画面を表示する

記事詳細画面の「編集」ボタン押下時の設定を追加します。

■sample07Detail.blade.php

　＊編集ボタン部分のみ抜粋。

　＊赤枠部を編集。

<a href="{{ route('sample06.edit', $sample\_data->id) }}"

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

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

■Sample06Controller.php

　＊editメソッドのみ記載。

public function edit(string $id)

{

    // 引数の$idをキーにサンプルテーブルより、データを取得

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

    // 取得データを渡し、sample08Editingビューをレスポンス

    return view("sample08Editing", compact("sample\_data"));

}

＊viewメソッドを使用して、編集画面（sample08Editing.blade.php）を表示します。

最後に、編集画面（sample08Editing.blade.php）を作成します。

編集画面のデザインは課題３の記事編集画面articleEditing.blade.php同様とする。（画像部分は除外）

■sample08Editing.blade.php

　＊テンプレート部分などは省略。

　＊赤枠部分…コントローラから受け取った値を表示。

　＊「編集」「戻る」ボタンやPUTメソッドについては、別途説明。

<section>

    <form action="{{ route('sample06.update', $sample\_data->id) }}" method="POST" enctype="multipart/form-data">

        @method('PUT')

        @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="{{ $sample\_data->title }}">

            </div>

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

                <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">

{{ $sample\_data->body }}

</textarea>

                </div>

            </div>

        </div>

        <div class="flex justify-end">

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

                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-green-600 hover:bg-green-500 px-3 py-2 rounded-md">更新</button>

        </div>

    </form>

</section>

記事編集画面のビューが完成したら、実行します。

記事詳細画面から、「編集」ボタン押下時、記事編集画面が表示されることを確認してください。また、記事詳細画面で選択した記事の内容が正しく表示されていること。

■サンプル記事詳細画面　　　　　　　　　　　　　　　　　　　　■サンプル新規編集画面

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　　　　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

**「編集」ボタン押下**

＊サンプル新規編集画面のURLが「http://localhost:8000/**sample06/{記事のid}/edit**」となっていることを確認する。

８－１－２． リクエスト送信の設定をする

記事編集画面で必要事項を入力後、「更新」ボタンを押したときのリクエスト送信設定を行います。

更新ボタンを押すと、入力したデータを送信します。ただし、この時、PUT（＊）リクエストを送信するものとする。

|  |  |
| --- | --- |
| 警告 単色塗りつぶし | （＊）PUTとは  POSTと同様にリソースの作成や更新を意味しますが、POSTとは違い、**冪等（べきとう）である**ことを表します。冪等とは、「ある操作を1回行っても、複数回行っても結果が同じになる」という概念です。編集リクエストは何度送られても同じ結果になるので、PUTを使用します。  つまり、PUTはPOSTより限定的な用途となり、主に既存リソースを置き換えるために使用します。 |

■前提条件（ルーティングの再確認）

今回は、既にルーティングでupdateではPUTメソッドを使用する設定となっている。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| HTTP  メソッド | パス | ルーティング名 | メソッド | 役割 |
| **PUT**  /PATCH | /sample06/{sample06} | **sample06.update** | **update** | 指定されたリソースを更新  PUT：指定された情報を更新する |

記事編集画面に以下の設定を行います。

■フォーム設定①　<form>タグの属性指定。

|  |  |
| --- | --- |
| 属性 | 値 |
| action | コントローラのデータ更新用のメソッドをルーティング名にて指定。記事のidを引数として渡す。  （サンプル環境では、「sample06.update」を指定） |
| method | POST  ＊**HTMLのformタグではGETとPOSTにしか対応していない**ため、今回はPOSTを指定。 |

■@methodディレクティブ



■フォーム設定②　formタグ内に追記

@method('PUT')

　＊ルーティングでPUTメソッドを指定しているため、formタグ内に正しいメソッドを指定。

以上をビューに反映します。

■sample08Editing.blade.php

　＊フォームの開始タグ部分のみ抜粋。

　＊赤枠部分を設定。

<form action="{{ route('sample06.update', $sample\_data->id) }}" method="POST" enctype="multipart/form-data">

     @method('PUT')

     @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>

**～～　以下、省略　～～**

編集画面で「更新」ボタンを押すと、PUTリクエストが送信され、updateメソッドが呼び出される動作となります。updateメソッドは引数として記事idとリクエストデータを受け取ります。

８－１－３． DBを更新する

updateメソッドの処理を記述していきます。updateメソッドは、記事idとリクエストデータを引数として受け取ります。

処理の流れは以下の通りです。

【更新処理】　＊赤字部分が新規登録と大きく違う箇所。

1. リクエストデータの入力値チェック
2. **記事idから該当データを取得**
3. **②で取得したデータを、リクエストデータで上書き**
4. DBに変更を保存する

■Sample06Controller.php

＊updateメソッドのみ抜粋

＊transaction、saveなどは第7回目に記載済。

更新なので、まずは更新したいデータを取得する。

その上で、取得データをリクエストデータで上書きし、その後saveする。

public function update(Request $request, string $id)

{

    // 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

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

    // トランザクション処理

    DB::transaction(function () use ($id, $sampleDao, $request) {

        // idをキーにデータを取得

        $sample\_data = $sampleDao->find($id);

        // 取得データをリクエストデータで上書き

        $sample\_data->title = $request->input("title");

        $sample\_data->body  = $request->input("body");

        // データを保存

        $sample\_data->save();

    });

    // 記事詳細画面へリダイレクト

    return redirect()->route("sample06.show", $id);

}

実装したら、データ更新処理が正しく行われることを確認します。

８－１－４． 画面にエラーメッセージを表示する

画面にエラーメッセージを表示するようにしてください。

また、入力値が保持されるようにしてください。挙動とヒントは以下の通りです。

■エラー時の挙動例

　＊例では本文だけだが、タイトルの場合も同じ挙動となること。

|  |  |
| --- | --- |
| ■記事編集画面  　例では、「本文」の内容を削除し、  空白のまま「更新」ボタン押下 | 【NGパターン】 ・エラーメッセージ表示  ・古いデータ表示（←NG） |
| 【OKパターン】  ・エラーメッセージ表示  ・入力値を保持（←OK） |

【ヒント】

■oldヘルパー　＊7回目の資料より抜粋

ロゴ

自動的に生成された説明

　または

テキスト

低い精度で自動的に生成された説明

　＊第2引数は、値がない時の初期値。（省略可）

８－１－５． 戻るボタン押下時の処理を追加する

戻るボタン押下時、記事詳細画面へ遷移します。

８－１－６． すべての挙動を確認する

完成したら、必ず、様々なパターンのテストを行うこと。（単体テスト）

８回目は以上です。