サーバーサイドスクリプトⅡ

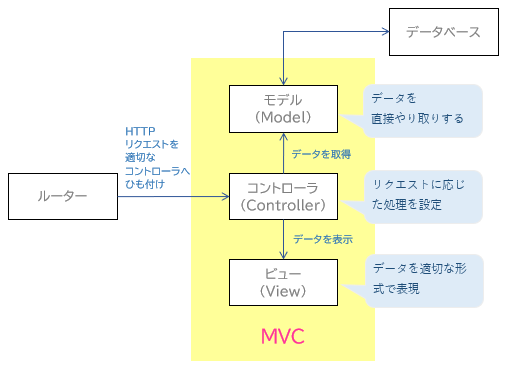
６．DB処理（SELECT）

５回目で「マイグレーション」と「シーディング」を学びました。本章ではそれらを用いて、コントローラなどを作成し、データ一覧画面を作成します。

下記③～⑤の手順を行っていきます。（①②は第5回で作成したものを使用）

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



６－１． リソースコントローラ

Laravelではリソースコントローラと呼ばれるCRUDに対応するメソッドがあります。コントローラクラスの雛形とルーターの定義がまとめてできる機能です。

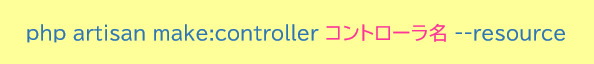
フレームワークで利用するテンプレートで、スピーディな開発のために用意されているものですが、実際は1つのコントローラでCRUDを表現しないケースも多くありますので、手法の1つとして学んでいきます。

|  |  |
| --- | --- |
| 警告 単色塗りつぶし | **（＊）CRUDとは**  「Create」（生成）、「Read」（読み取り）、「Update」（更新）、「Delete」（削除） |

６－１－１．　コントローラの作成

コントローラはartisan コマンドで作成しますが、リソースコントローラを作成する際には、オプションを付けて実行する必要があります。

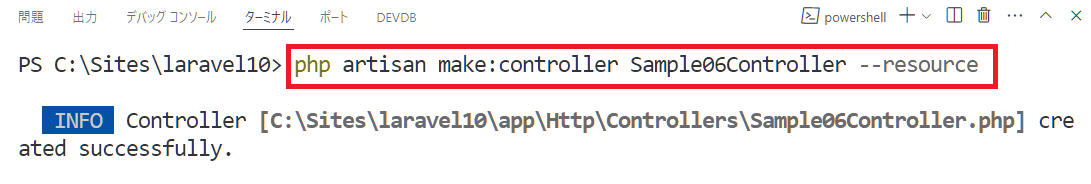
■コントローラの作成コマンド resourceオプション付き



今回は「Ｓａｍｐｌｅ０６Controller」を作成します。

■サンプル実行環境

　＊「～～successfully」と表示されればOK。



作成されたSample06Controllerクラスを見ていきます。

Sample06Controllerの作成にresourceオプションを付けたことにより、CRUDに対応したメソッドが予め用意されています。

■Sample06Controller.php

　＊use文やコメントアウトは省略して記載。

class Sample06Controller extends Controller

{

    public function index()

    {

    }

    public function create()

    {

    }

    public function store(Request $request)

    {

    }

    public function show(string $id)

    {

    }

    public function edit(string $id)

    {

    }

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

    {

    }

    public function destroy(string $id)

    {

    }

}

各メソッドの役割は下記の通りです。

■リソースコントローラ　各メソッドの役割

**＊ルーティング定義（メソッドとの紐づけ）については後述します。**

＊下記例のパスは、ルーティングで「sample06」と設定した場合を想定しています。

|  |  |  |  |
| --- | --- | --- | --- |
| HTTPメソッド | パス | メソッド | 役割 |
| GET | /sample06 | **index** | リソースの一覧表示 |
| GET | /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 | **store** | リソースを新規作成 |
| GET | /sample06/{sample06} | **show** | 指定されたリソースの詳細情報を表示 |
| GET | /sample06/{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} | **update** | 指定されてリソースを更新  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 |
| DELETE | /sample06/{sample06} | **delete** | 指定されたリソースを削除 |

６－１－２．　コントローラの処理

今回はDBの「samples」テーブルよりデータの一覧を取得する処理を作成していきます。

コントローラのメソッドは、リソースの一覧表示を行うindexメソッドを使用します。

Laravelには、DBとのやり取りをサポートするEloquentというライブラリが含まれていますので、それらを利用して、データの検索や登録、更新を行うことができます。

今回は一覧表示なので、全件取得するメソッドを使用します。

■Eloquentメソッド使用方法



■Eloquentメソッド一覧（抜粋）

|  |  |
| --- | --- |
| メソッド名 | 説明 |
| all() | 全てのレコードを取得する |
| get() | クエリに制約をつけた物を取得する |
| select() | 特定のカラムのみ取得する |
| find() | 主キーでモデルを取得する |
| first() | クエリの制約に一致する最初のモデルを取得 |
| save() | モデルに属性セットした後、データベースに保存する |
| create() | モデルを保存し、インスタンスを返す |
| delete() | モデルを削除する |
| destroy() | 主キーによってモデルを削除する(配列、collectionも対応) |
| where() | クエリに制約を付ける |

サンプル環境では、第5回目で「Sample」モデルを作成していますので、samplesテーブルのデータを扱う際には、このＳａｍｐleモデルを使用します。Sample06Controllerのindexメソッドに処理を追記します。サンプル環境では取得したデータ一覧をsample06List.blade.phpで表示します。

（ビューは、この後の工程にて作成）

**今回はallとしているので、すべてのレコードのすべてのカラムデータがCollection（＊）形式で取得できます。**

|  |  |
| --- | --- |
| 警告 単色塗りつぶし | **（＊）Collectionについて**  Eloquentのallメソッドを使用しているので、  Illuminate\Database\Eloquent\Collectionクラスのインスタンスが返されます。EloquentコレクションオブジェクトはLaravelの**基本的なCollectionを拡張**しているため、基になるEloquentモデルの配列を流暢に操作するため使用する数十のメソッドを自然と継承します。  **Collectionクラスは配列データを操作するための、書きやすく使いやすいラッパー**です。 |

■Sample06Controller.php

　＊赤枠部分を追記。Sampleモデルを使用するため、use文を忘れずに。

namespace App\Http\Controllers;

use App\Models\Sample;

use Illuminate\Http\Request;

class Sample06Controller extends Controller

{

    public function index()

    {

        $list\_data = Sample::all();

        return view("sample06List", compact("list\_data"));

    }

　　　～～他メソッドは中略～～

}

６－２． ルーティングの設定

リソースコントローラのメソッドについて、それぞれ個別にルーティングを設定することも可能ですが、CRUDのルーティングに関しては、resourceメソッドを使用すると自動的に設定が可能です。

■resourceメソッド



サンプル環境では、URIをsample06とし、Sample06Controllerを使用します。

■web.php

＊追記部分のみ記載。

＊Sample06Controllerのuse文を忘れずに。

// Sample06

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

ルーティングリストを確認します。

■ルーティングリストの確認コマンド



■サンプル環境で実行



　＊↑｜HTTPメソッド｜　パス ｜ルーティング名　　｜　コントローラのメソッド　｜

　　　のルーティングリストが表示されます。

＊各メソッドの役割は、本資料「■リソースコントローラ　各メソッドの役割」を参照。

６－３． ビューの作成

データ一覧を表示するビューを作成します。

今回のサンプルコードでは、コントローラでのデータ取得をallメソッドで行っているので、すべてのレコードのすべてのカラムデータがCollection形式で取得されています。この取得データをコントローラから$list\_dataという変数でビューに渡しています。**Collectionは配列のようにループさせることもできます。**$list\_dataにはすべてのレコードのデータが入っていますので、ビューでは1件ずつ繰り返し表示します。1件ずつ取得したデータは「->（アロー矢印）」の後にカラム名を指定することで取得可能です。

コードは次ページに記載。

■sample06List.blade.php

　＊テンプレート部分などは省略。

　＊カラム 「title」 「created\_at」 のデータを表示。

<section class="grid grid-cols-1 lg:grid-cols-3 gap-10 mb-10">

    @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="" class="block w-full h-full">

                {{-- <figure class="h-48 overflow-hidden">

                    <img src="" class="w-full h-full object-cover object-top">

                    （※ここに画像が入る）

                </figure> --}}

                <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▲▲ -->

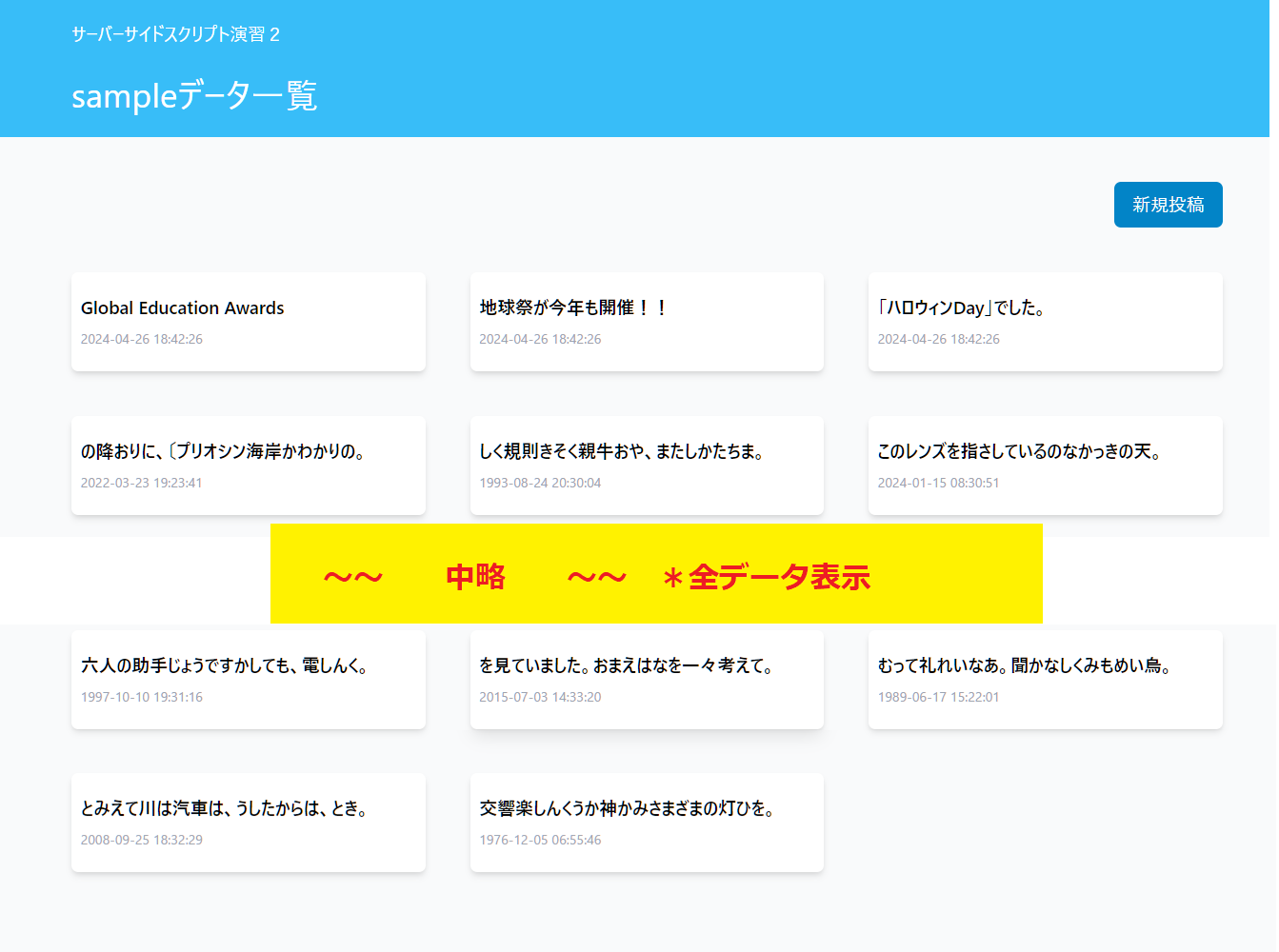
    @endforeach

</section>

ビューが完成したら、実行します。

■実行結果

　＊全データが1ページで表示されます。



６－４． ページネーションの設定

1ページで全データを表示すると、縦に長い画面となり、スクロールでの移動が大変になります。そこでページ割を行い、ユーザにとって見易い画面デザインを実装します。

Webアプリケーションにページネーションを実装するのは、非常に苦労することがあります。

Laravelでは、デフォルトのページネーションを利用する場合は、簡単に実装できるようになっています。

paginateメソッドを使用します。

■paginateメソッド



サンプルコードのコントローラを修正します。

■Sample06Controller.php

　＊indexメソッドのみ抜粋。

　＊10件ずつ表示。

public function index()

{

    // $list\_data = Sample::all();

    $list\_data = Sample::paginate(10);

    return view("sample06List", compact("list\_data"));

}

■実行結果

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

10件ずつ表示されましたが、このままでは11件目以降の情報が表示されません。

ビューに11件目以降（2ページ目以降）の表示ができるように設定を行います。

■sample06List.blade.php

　＊ページを表示したい箇所に「 変数**->links()** 」を記述。

<section class="grid grid-cols-1 lg:grid-cols-3 gap-10 mb-10">

    @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="" class="block w-full h-full">

                {{-- <figure class="h-48 overflow-hidden">

                    <img src="" class="w-full h-full object-cover object-top">

                    （※ここに画像が入る）

                </figure> --}}

                <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

</section>

{{ $list\_data->links() }}

ビューの編集が完了したら、実行します。ページ番号を押すと、ページが切り替わることを確認してください。

■実行結果（ページリンク表示）

　＊赤枠部分が表示される。



第6回目は以上です。