サーバーサイドスクリプトⅡ

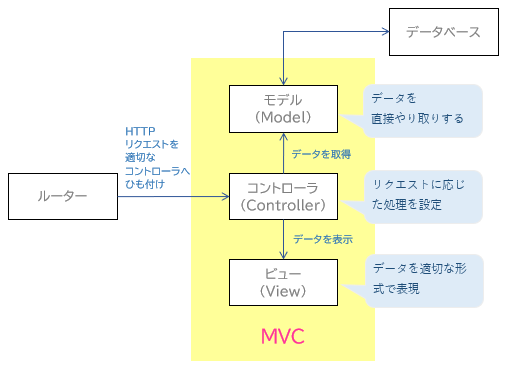
１１．ファイルアップロード （総合課題①）

第１１回目では、ファイルアップロードを行っていきます。

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　＊今回は処理を追加**
4. **ルーティングの設定**
5. **ビューの作成　＊今回は画像に関する表示を追加**

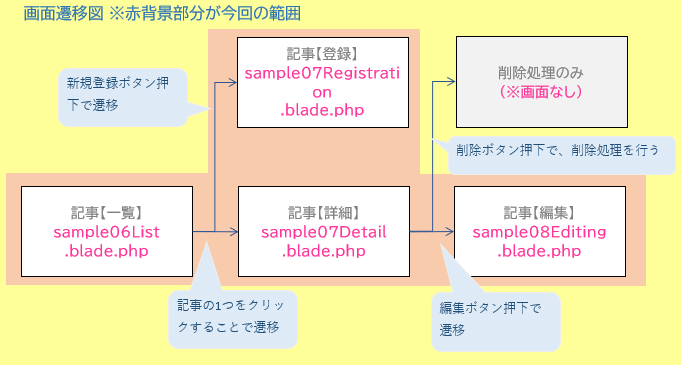
■図：　MVC



今回の処理範囲は下図の通りです。

　＊各画面と画像以外の処理は9回目までで完了しています。

　＊各画面において、画像ファイルに関する処理を追加します。（サンプルでは登録と一覧のみ行います）



１１－１． 前準備samplesテーブルに列追加

第5回の例題では、samplesテーブルのマイグレーションファイルを下記のようにしていました。

画像ファイルのパスを格納するようの列を作成していませんでしたので、追加で作成します。

＊「img\_path」列を作成済みの場合は、操作は不要ですが、列追加の方法は読み進めてください。

■create\_samples\_table.php　＊第5回目で作成。

　＊upメソッド部分のみ記載。

**「img\_path」列を定義していない。**

public function up(): void

{

    Schema::create('samples', function (Blueprint $table) {

        $table->id()->unsigned();

        $table->string('title', 255);

        $table->text('body')->nullable();

        $table->timestamp('created\_at')->useCurrent();

        $table->timestamp('updated\_at')->useCurrent()->useCurrentOnUpdate();

        $table->softDeletes();

    });

}

既にsamplesテーブルを作成し、テストデータも投入済みのため、テーブルを作り直すのではなく、列のみ追加していきます。

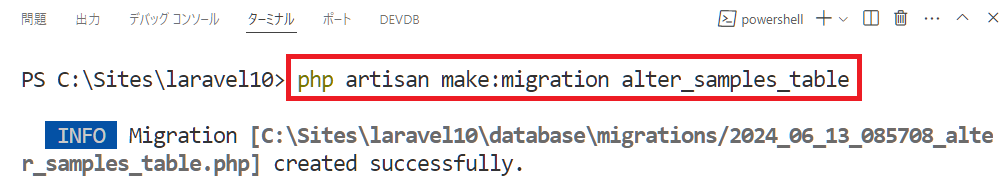
■samplesテーブルへの列追加　手順

1. 列追加用のマイグレーションファイルを作成します。
2. マイグレーションファイルに追加列の定義を記述します。
3. マイグレーションを実行します。

それぞれの手順について記述します。

1. 列追加用のマイグレーションファイルを作成。

コマンドでマイグレーションファイルを作成します。名前は「alter\_samples\_table」とします。



1. マイグレーションファイルに列定義を記述します。

■alter\_samples\_table.php

　＊upメソッド記載。

　＊Schema::create()ではなく、Schema::table()を使用します。

　＊追加する列のみ記述。列の追加のみなので、null許容にしておくこと。

public function up(): void

{

    Schema::table('samples', function (Blueprint $table) {

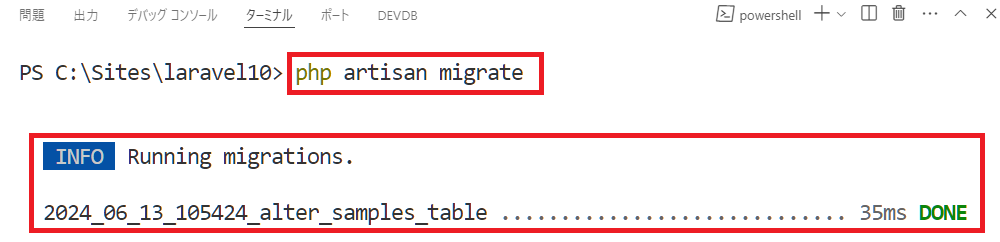
        $table->string('img\_path', '255')->nullable();

    });

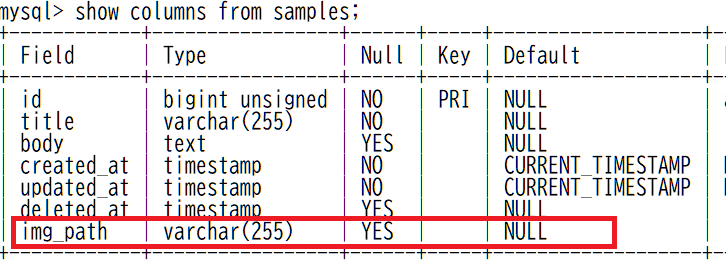
}

1. マイグレーションを実行

追加したファイルのみ実行されていることを確認する。



　　DBのテーブルの末尾に列が追加されていることも確認する。



１１－２． ファイルシステム

ファイルの保存場所について、設定を行います。

DBにはファイルの保存場所（パス）を保存しますが、ファイルそのものは指定したフォルダに保存（アップロード）します。

ファイルの操作にはStorageクラスを利用します。

１１－２－１．Flysystem

Laravel には Flysystem PHP パッケージを統合したファイルシステムが提供されています。

Laravel Flysystemでは、ストレージを操作するための以下のドライバーが提供されています。

- ローカルファイルシステム

- FTP

- SFTP

- AmazonS3

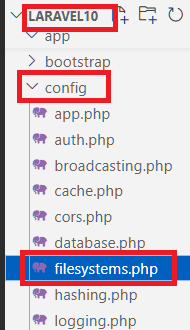
１１－２－２．ファイルストレージ

ファイルシステム設定ファイルは、「config/filesystems.php」にあります。

filesystems.php内で、すべてのディスク（ファイルシステム）を設定することができます。

サポートしている各ドライバーの設定例を設定ファイルに用意してくれているので、すぐにファイルシステムを利用することができるようになっています。

■filesystems.php場所



filesystems.phpを編集します。

（コードは次のページに記載。）

■filesystems.php

　＊コメント部分や、今回使用しない部分は省略して記載。

　＊赤枠部分を追加。

**デフォルト設定。（※最初から記述あり）**

**この後のサンプルコードではこちらを使用。**

**学内サーバ（nt24）への接続情報を追加**

<?php

return [

    'default' => env('FILESYSTEM\_DISK', 'local'),

    'disks' => [

        'ftp' => [

            'driver' => 'ftp',

            'host' => '10.201.10.38', //nt24のIP（学内サーバ）

            'username' => 'se2a', //クラス名

            'password' => 'ecc',　 //全員共通

        ],

        'local' => [

            'driver' => 'local',

            'root' => storage\_path('app'),

            'throw' => false,

        ],

        'public' => [

            'driver' => 'local',

            'root' => storage\_path('app/public'),

            'url' => env('APP\_URL').'/storage',

            'visibility' => 'public',

            'throw' => false,

        ],

**～～　省　略　～～**

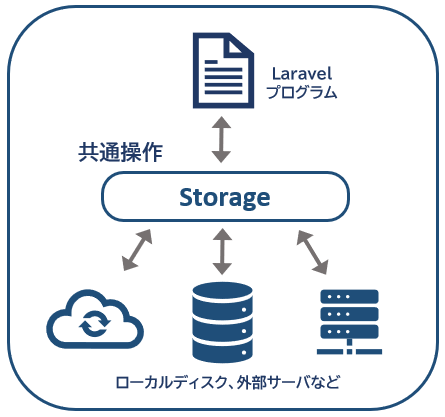
];

ファイルシステムの設定ファイルに記述することにより、Storageクラスを用いて指定することができます。

１１－２－３．Storageクラス

LaravelにはStorage機能が備わっています。

Storageを使えばローカルのファイルもクラウドのファイルも同じ方法で操作することができます。



たとえば、先ほどfilesystems.phpで「public」のストレージを指定します。

**disksの連想配列に定義したKey名での指定が可能となる。**

**今回はこの後のサンプルコードで「public」を使用する。**

<?php

return [

    'disks' => [

        'public' => [

            'driver' => 'local',

            'root' => storage\_path('app/public'),

            'url' => env('APP\_URL').'/storage',

            'visibility' => 'public',

            'throw' => false,

        ],

];

■diskメソッド



戻り値は　FileSystemAdapter でディスク情報が返ってきます。

■publicを指定したい場合

Storage::disk("public")

上記のように指定します。ここで指示される場所は、「Storage\app\public\」です。

「public」の「root」に記述されている場所になります。

次の項でファイルの保存場所とシンボリックリンクについて説明します。

<?php

return [

    'disks' => [

        'public' => [

            'driver' => 'local',

            'root' => storage\_path('app/public'),

            'url' => env('APP\_URL').'/storage',

            'visibility' => 'public',

            'throw' => false,

        ],

];

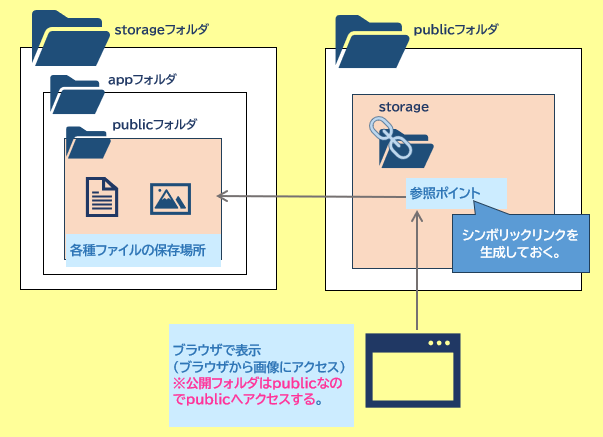
１１－２－４．　シンボリックリンク

アップロードしたファイル（今回は画像）は、「publicフォルダの外」に保存します。

publicフォルダはWebサーバを通じて外部に公開するため、画像などのファイルを保存するとインターネット上で直接アクセス可能となり、セキュリティ上のリスクが高まります。したがって、保存はstorageフォルダ（正確にはstrage\app\publicフォルダ配下）に行います。その上で、画像表示などでブラウザからpublicフォルダを指定しても、storege\app\publicのフォルダにアクセスできるようにシンボリックリンク（＊）というものを設定します。

|  |  |
| --- | --- |
| 警告 単色塗りつぶし | **（＊）シンボリックリンクとは**  OSのファイルシステムの機能の一つで、特定のファイルやディレクトリを指し示す別のファイルを作成し、それを通じて本体を参照できるようにする仕組みのこと。 |

■シンボリックリンクのイメージ

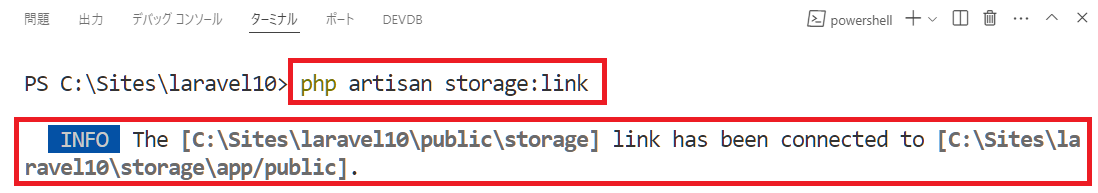


■シンボリックリンクの生成



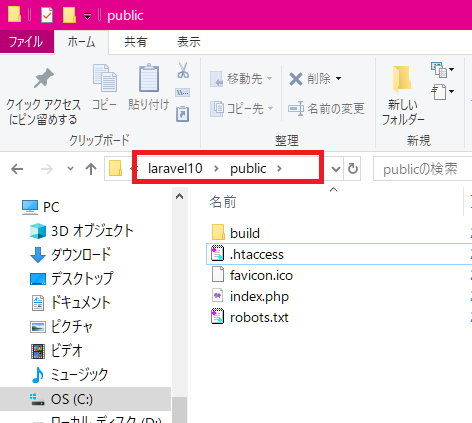
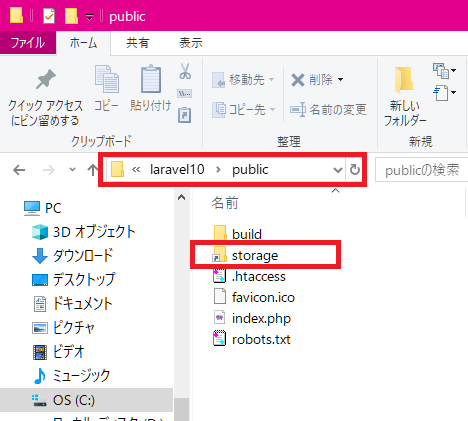
■サンプル実行例

　＊INFOの欄にシンボリックリンクの詳細も表示される。



エクスプローラで確認すると、コマンド実行前後でシンボリックリンクが生成されていることが確認できます。

■シンボリックリンク生成前　　　　　　　　　　　　　　　　　　　■シンボリックリンク生成後

シンボリックリンクを作成したら、assetヘルパを使用してstorage/app/public内のファイルへのURLが作成できます。

■assetヘルパ



assetヘルパはpublicディレクトリのパスを返すので、引数のファイルパスにはpublicディレクトリ以降のパスを指定します。

■使用例

<img src="{{ asset("storage/images/sample.png") }}"

上記を指定すると

http://localhost:8000/storage/images/sample.pngという指定になる。

１１－３． 記事登録　ファイルアップロード

記事登録処理において、画像ファイルアップロードの処理を追加していきます。

画像登録については、下記２件の処理を行います。

1. 画像ファイルをWebサーバにアップロード
2. 画像ファイルのファイル名をsamplesテーブルの「img\_path」列に登録

＊画像ファイルそのものの保管と、画像の存在する場所の保管の2点が必要となります。

１１－３－１．　画像ファイルの処理

記事登録画面で「投稿」ボタン押下時の処理を追加します。「投稿」ボタン押下時の処理は、Ｓａｍｐｌｅ０６Ｃontrollerのstoreメソッドで行っていますので、画像ファイルの処理を追加します。

コードは次ページに記載。

■Sample06Controller.php　＊コントローラクラスは第６回目で作成。

　＊storeメソッド部分のみ記載。

**リクエストデータに「image」が存在したら**

1. **データを取得**
2. **ストレージに保存**
3. **サンプルクラスにファイル名を格納**

**※各メソッドの説明は後述。**

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);

    if ($request->has("image")) {

        $file = $request->file("image");

        $image = Storage::disk("public")->put("images", $file);

        $sampleDao->img\_path = basename($image);

    }

    // 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");

}

【Requestクラスで使用できるメソッド】

Request $request 変数を用いる前提で記載。

■hasメソッド



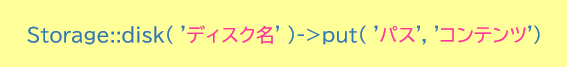
　戻り値：　パラメータ名に指定したパラメータが存在すればtrue、しなければfalse

■fileメソッド



　戻り値：　パラメータ名に指定したパラメータのファイル情報

■putメソッド



　＊Storage::diskの説明は「11-2-3.Storageクラスに記述」

　＊パス…「Storage\app\public」以降を記載する。

　＊コンテンツ…ファイル情報。

１１－３－３．画像のバリデーションチェック

バリデーションチェック用のルールとメッセージの定義はモデルクラスで行っていましたので、Ｓａｍｐｌｅ.phpクラスに追記します。

■Sample.php

＊抜粋。赤枠部分のみ追加。

// ルール用の配列

public static $rules = [

    "title" => ["required"],

    "body"  => ["required"],

    "image" => ["image"]

];

// メッセージ用の配列

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",

    "image.image" => "画像をアップロードしてください"

];

１１－３－４．画面表示

処理の記述が完了したら、以下を追加してください。

＊１－１０までの復習、かつ今回は複合問題なので、以下の解答は記載しません。

1. 記事登録画面のエラーメッセージ表示

＊今回はfileなので、エラーがあり再入力となった場合の値保持はしません。

（ファイルは毎回選択する必要がある）

1. 記事一覧画面に画像を表示

＊記事登録後、登録した記事の正しい画像が表示されていること。

＊ファイルパスについては、「１１－２－４．シンボリックリンク」を参考にすること。

１１－３－５．動作確認

完成したら、動作確認してください。

1. 一連の画面遷移と処理が正しく動いている。

サンプルサイトで動作確認すること。

＊サンプルサイトの画像選択時のサムネイルは実装しなくても良い。

実装する場合はPHP１の講義⑤を参考にできます。

1. 画像ファイルがStorage以下の正しい場所に保存されている

シンボリックリンクのpublic配下でも確認できること。

1. DBのimg\_pathにファイル名が正しく保存されている。

11回目は以上です。