

はじめての Laravel6 入門

AWS Cloud9 で学ぶ

<<本書3つのポイント>>

1. 開発環境をインストールする必要がない「AWS Cloud9」で解説
2. 使用するコマンドは黄色いマーカーで表記
3. 手を動かしながらステップバイステップで学ぶ



山崎大助 著

PHP 初心者から次のステップへ

G'sACADEMY Lab10

はじめに

概念的内容からステップバイステップの形式でコードを打ちながら学ぶ内容で構成されています。Laravel 6.0 (LTS) のバージョンで動作確認ができていることを示しています。新機能解説書ではありませんのでご了承ください。

また、「Laravel 入門 5.5」のアップデート版になります。「AWS Educate」「AWS Cloud9」を使用しての内容に変更しております。

※Cloud9 が 2019 年末に AWS Cloud9 に統合するためアップデートしました。

読者ターゲット

「PHP 初心者本が終わって、次に何を学ぶか？」を考えている人、「フレームワークを学んでみたい人」、「公開サーバーにアップロードする」など、PHP の初心者～中級者の人向けに書いています。特に「フレームワークを勉強しようとしたけど難しかった」という人、勉強の仕方がわからない人にはぴったりかと思います。「フレームワーク」を勉強したい！そんな人にはぜひ「読みながら＝作って」ほしいと思います。レッスンブックですので、作りながら学ぶ内容です。特に「コマンド」は何回も出ていますが、慣れていくましょう。

開発環境

「AWS Cloud9」というクラウド開発環境を使います。「AWS Cloud9」サービスに登録していただければ、自分のパソコンに環境をインストールする必要なく、クラウド上の管理画面ボタンを押していくだけで「クラウド上に開発環境」が作れるからです。失敗したらすぐに削除し、新規で「プロジェクト」を作り直せる（複数同時進行での作成も可能です）ので、フレームワーク初心者には失敗しても打撃がない、丁度よい環境です。開発では、環境準備で「あきらめる」人が多いので、本書では「AWS Cloud9」で進めることにしています。

「はじめてのフレームワーク体験 → Web 公開まで」の

学ぶことにフォーカス、「慣れる」ことを目指します。

G'sACADEMY Lab10

◇フレームワークを学ぶ

サービスを作ることになると、必ず「バリデーション、LOGIN、セキュリティ、記述コードの統一性」などコード量や、考える事も増え、バグや心配が多くなります。

そこで、フレームワークを使うことで多くのことを解消できます。多くのプロダクトでは、フレームワークを使って構築するケースが多いです。PHP のフレームワークはいくつもありますが、Google ブラウザで「Laravel」を検索するとダントツに右肩上がりです。これから学ぶには最良のフレームワークの一つでしょう。フレームワークを覚えると便利で開発も圧倒的に早くなります。

◇学習コスト

最初の学習コストが高いため、そこを乗り越えられないケースが多いようです。筆者もフレームワークの最初の学習コストに悩まされた一人ですが、我慢しながらやり続けたら、少しずつ慣れて「こういうものか」と理解しました。スッと頭で理解できるくらい簡単ではありません。ただし、3~5回繰り返し、簡単で同じものでも作ってみると、何故か理解してきます。慣れ、「こういうものなんだ」という感覚がでてきます。そこまでなれば、ゴールは近いです。あとは応用の知識を作りながら、沢山身についていけば大丈夫です。

◇本書

laravel.com や世界、日本のドキュメントサイトを見て学びました。シンプルなアプリを筆者なりに解釈し、フレームワーク初心者に向けて解説を書いてあります。本書では「最初からシンプルなアプリを作っていく、最後に Web サーバーにアップロードするところまで」をフレームワークの使い方を学んでもらう書籍となっています。フレームワークは難しいですが、サービスを作る上では必要なのでがんばって習得しましょう。

◇おすすめ！最短コマンド集

<https://github.com/yamazakidaisuke/GsCodeSample>

ファイル名：[Laravel6_start_aws.txt](#)

こちらの Github に載せてるコマンドを参照し、記載されてる順番にコマンドをコピー＆ペーストで貼っていくだけで最小のアプリが完成します。とりあえず「一通りの流れ」を体験したい人にはとても良いので、是非使ってみてください。

G'sACADEMY Lab10

目次

.....	1
はじめての LARAVEL6 入門 AWS CLOUD9 で学ぶ.....	1
はじめに.....	2
CHAPTER1	14
PHP 初めてのフレームワーク	14
LARAVEL フレームワーク	15
そもそもフレームワークって？	15
LARAVEL の特徴.....	16
LARAVEL ヘルパー関数	16
MVC パターンとは(一般的なフレームワーク)	17
MVC	18
Controller/コントローラ:	18
Model/モデル:	18
View/ビュー:	18
Laravel は MVC では無い?	18
CHAPTER2	19
開発環境準備	19
LARAVEL 開発環境(インストール型)	20
主な環境環境.....	20
AWS CLOUD9 統合開発環境(クラウド型)	21
本書では「AWS Cloud9」を利用ていきます。	23
1. 「AWS マネジメントコンソール」画面から Cloud9 を検索.....	23
2. 「Create environment」クリック.....	23
3. "2箇所"を入力して「Next step」クリック.....	24
5. 「Create environment」クリック.....	25
6. 「Cloud9」がロードされるので少し待ちます。	25
7. 「Cloud9」画面がでれば完了です。	26
CLOUD9 の画面構成(重要:必ず見る)	27

G'sACADEMY Lab10

1. メニューバー	27
2. <i>Workspace</i> (プロジェクト)	27
3. エディター領域	27
4. コマンド領域	27
CHAPTER3	28
LARAVEL インストール	28
CLOUD9 の環境確認	29
CLOUD9 の PHP を「 PHP7.2」にバージョンアップ	30
LARAVEL フレームワークの準備	31
次に LARAVEL インストーラーを準備します。	31
LARAVEL プロジェクトを作成します(各バージョン)	32
インストールとは?	32
LARAVEL の「インストール確認」と「PREVIEW 設定」	33
1. <i>cms</i> フォルダに移動し、 <i>Built-in web</i> サーバーを起動します。	33
2. 「Preview」をクリックし「Preview Running Application」を選択。	33
3. 動作確認は「1画面表示」で行う必要があります！ 以下赤枠のアイコンをクリックすると1画面で表示することができます。	34
4. 別タブ(1画面表示)で Web ページが表示されます。	34
5. エディタ画面を表示して以下の手順で操作します。	35
6. 別タブで表示されるか確認。	35
WEB サーバーを STOP する方法	35
超重要:毎回使うので覚えておくこと	36
DB 作成	37
DB を起動して、データベースを作成	38
1. MySQL 起動する(毎回ログイン後に必要なコマンド!)	38
2. MySQL に root ユーザーで入る	38
3. 「c9」データベースを作成	38
4. 一覧表示コマンドで「c9」データベースを確認。	38
5. 「c9」データベースを選択	38
6. テーブルはまだありません。	39
7. root ユーザーのパスワードを「root」に設定変更	39

G'sACADEMY Lab10

8. 設定変更を反映させる.....	39
9. MySQL から抜けます.....	39
MySQL を使用しない場合 (SQLITE を選択する場合)	40
SQLite 使用設定.....	40
DB ファイルを作成.....	40
CHAPTER4	41
LARAVEL ルートディレクトリ フォルダ構成	41
LARAVEL ルートディレクトリ:フォルダ構成.....	42
app.....	42
bootstrap.....	42
config.....	42
database.....	42
public.....	42
resources.....	42
routes.....	42
storage.....	43
tests.....	43
vendor.....	43
最初は下記 5 つのディレクトリ & ファイルをよく覚えておきましょう。	43
「JAVASCRIPT&CSS」ファイルの配置場所は「PUBLIC」.....	44
JavaScript	44
CSS	44
* https 対応	44
CHAPTER5	45
練習アプリ:本管理アプリを作っていきましょう。	45
練習アプリ:本管理アプリを作っていきましょう。	46
完成形画面	46
今回のアプリ制作の流れ.....	47
データベースの設定	48
1. 「.env ファイル」を探します。	48
2. 「.env ファイル」は Laravel の設定ファイルです。DB の設定などをこのファイルでおこないます	

G'sACADEMY Lab10

ので、最初に表示されているか確認しましょう。	48
3. 「.env ファイル」が見つからない場合（Cloud9、Mac など非表示になっているケースがあります）	48
4. ワークスペースのファイルツリー右上ギアアイコンから..... [Show Hidden Files] にチェックで隠しファイルが表示されます.....	49 49
MySQL 選択する場合	49
CHAPTER6	50
練習アプリ 本管理アプリを作っていきましょう。	50
データベースマイグレーション	51
解説)コマンド書式:マイグレーション	51
練習)BOOKS テーブルを作成.....	52
1. 「books テーブルを作成する」マイグレーションファイル作成.....	52
2. マイグレーションファイルの「場所とファイル名」.....	53
3. マイグレーションファイルの構造を確認.....	53
カラム定義.....	54
4. 作成されたマイグレーションファイルに追加記述.....	55
5. 「up メソッド」に 4 カラム追加した状態.....	56
修正が必要.....	57
6. マイグレーションを実行(テーブル作成).....	58
7. MySQL にて実行確認.....	58
<<migrate コマンド一覧>>	59
<<コラム>>	60
CHAPTER7	61
練習アプリ:本管理アプリを作っていきましょう。	61
モデルを作成しましょう！	61
1. ELOQUENT モデル(エロケント モデル)	62
マイグレーション コマンド書式:.....	63
2. 練習: BOOK モデルを作成.....	63
作成されるモデル.....	63
作成されたモデル.....	64

G'sACADEMY Lab10

他のモデル作成コマンド書式.....	64
CHAPTER8	66
練習アプリ:本管理アプリを作っていきましょう。ルート定義を作成しましょう	66
ルート定義.....	67
ルート定義.....	68
ルート定義のスケルトンを作成.....	68
1. 今回作成するルート定義は、先にスケルトン(処理が空)のルート定義をしていきます。	69
2. 以下は「use」を使って参照する2クラス.....	70
CHAPTER9	71
練習アプリ ログイン認証機能と JS/CSS を追加	71
サインイン・ログイン認証.....	72
1. <i>laravel/ui</i> パッケージをインストール	73
2. <i>artisan</i> コマンドを実行	73
3. <i>npm</i> パッケージをインストール	73
4. パッケージをビルド	73
5. ルート定義に「2行追加」されています。確認しましょう。	73
6. ブラウザレビューで確認しましょう。	74
CHAPTER10	75
練習アプリ:本管理アプリを作っていきましょう。	75
レイアウトとビューを作成しましょう！	75
ビュー(レイアウト・テンプレート)	76
レイアウトとビューの作成	77
1. テンプレートの定義.....	78
2. テンプレート・レイアウトの関係.....	79
3. 子テンプレートの定義(パーツ).....	81
4. <i>@extends</i> ディレクティブ (親テンプレート読み込み)	83
<i>@extends('layouts.app')</i> で親テンプレートを読み込みます。	83
5. <i>@section</i> ディレクティブ (名前をつけて一括りにする)	83
<i>@section('content')</i> から <i>@endsection</i> の間の文字/HTMLなどを <i>content</i> に括る。 親テンプレート <i>@yield('content')</i> の箇所に <i>content</i> を挿入します。	83

G'sACADEMY Lab10

6. @include ディレクティブ.....	83
7. エラー表示テンプレートを作りましょう.....	84
8. ルート定義に <i>view</i> ヘルパー関数を定義します。.....	85
9. ビューの確認(<i>Web</i> サーバーを起動させましょう).....	85
• <i>Blade</i> テンプレートでの変数表示.....	86
変数のデータ表示(変数表示の切り分け).....	87
• <i>Blade</i> と <i>JavaScript</i> フレームワーク.....	87
CHAPTER11	88
練習アプリ:本管理アプリを作っていきましょう。.....	88
本の追加登録処理を作成しましょう	88
本の追加登録処理を作成.....	89
1. バリデーションを作成(入力チェック)	89
2. バリデーション設定	90
3. 本の登録処理を作成.....	91
CHAPTER12.....	92
練習アプリ:本管理アプリを作っていきましょう。	92
本の一覧表示を作成しましょう	92
登録している「本」の一覧表示を作成	93
1. 本の一覧データを取得し、 <i>view</i> へ変数を渡す	93
2. @foreach を使い \$books の値を \$book へ代入して表示	94
解説: <i>BLADE</i> テンプレートの制御構文	96
分歧処理.....	96
多分歧処理.....	96
反復処理(インデックス).....	96
反復処理(配列//連想配列//オブジェクト)	97
反復処理(配列//連想配列//オブジェクト)	97
条件式が偽(<i>False</i>)の場合の処理.....	97
条件式が偽(<i>False</i> // <i>True</i>)の場合の処理.....	97
CHAPTER13	98
練習アプリ:本管理アプリを作っていきましょう。	98

G'sACADEMY Lab10

本の削除機能を作成しましょう！	98
本の削除機能を作成	99
1. 削除ボタンを追加.....	99
2. 削除処理を追加.....	101
課題 1-3	102
課題1	103
<input type="checkbox"/> 入力フォームを増やす&データ登録.....	103
<input type="checkbox"/> バリデーションの設定.....	103
課題2.....	103
<input type="checkbox"/> 更新画面&更新処理も調べて作ってみましょう。.....	103
課題＊	103
<input type="checkbox"/> 同じアプリを複数作ってフレームワークに慣れましょう。.....	103
課題解説 1-2	104
画面完成例:VIEW サンプル・ダウンロード	105
HTTP://WWW.VENEZIA-WORKS.COM/AWS/KADA1.ZIP	105
課題 1と2:画面完成例	105
課題解説 1:ルート定義とビュー作成	106
1. ルート定義に更新画面のスタブを追加.....	106
2. スタブに表示処理を追加.....	107
3. 3つの「入力項目」追加(ビューを変更).....	108
4. 更新画面へのリンクを追加(ビューを変更).....	110
5. 更新画面のビューを新規作成.....	111
6. 表示と動作を確認.....	114
課題解説 1-2:登録と更新の処理作成	115
1. 課題のバリデーションを定義.....	115
2. 追加した入力項目を登録プロパティに追加.....	116
3. 更新処理に必要な「ルート定義と更新処理」を作成する.....	117
4. スケルトンに更新処理を追加 スケルトンに更新処理を追加しますが、「登録処理のルート定義」をコピーして使いましょう。重複する処理が多いからです。	118
5. 更新表示と更新処理の動作確認.....	120
6. 次に更新画面の更新処理を確認.....	121

G'sACADEMY Lab10

CHAPTER14	122
ルート定義からコントローラに処理コードを移行	123
コントローラ	125
ルーティング・コントローラ・ビューの流れ	126
BOOKSCONTROLLER の作成(更新処理)	127
1. 練習:BooksController を作成します。	127
2. 新規作成したコントローラファイルのコードを確認	128
3. コントローラに <i>update</i> メソッド(スタブ)を追加	129
4. ルート定義の「 更新 」処理をコピーします。	130
6. ルート定義をクロージャーから「コントローラ」に変更します。	132
BOOKSCONTROLLER へ登録処理の追加	133
1. コントローラに <i>store</i> メソッド(スタブ)を追加します。	133
2. ルート定義の「 登録 」処理をコピーします。	134
ルート定義している「登録処理」をコピーで持ってきます。	134
4. ルート定義をクロージャーから「コントローラ」に変更	136
課題3	137
課題3: ルート定義「3箇所」をコントローラに移行	138
1. 最初に表示されるページへのルート定義	138
2. 更新画面へのルート定義	138
3. 削除処理へのルート定義	139
<課題3:内容の確認>	139
<課題全体の捉え方>	139
課題解答3.1.....	140
課題1. 最初に表示されるページへのルートを定義	140
課題解答3.2.....	141
課題2. 更新画面へのルート定義	141
課題解答3.3.....	142
課題3. 削除処理へのルート定義	142
ルート定義:課題完了後のコード	143
CHAPTER15	144
ページネーション	144

G'sACADEMY Lab10

ページネーション	145
1. ページネーションを使う.....	145
2. ページネーションをテンプレートに埋め込む.....	146
3. <i>bootstrap/app.php</i> の末尾「 <i>return</i> 」上に追記	146
4. ブラウザで表示確認.....	146
CHAPTER16	147
サインイン・ログイン認証	147
1. ルート定義に以下認証用の2行が追加されています.....	148
2. コントローラに「 <i>HomeController.php</i> 」が記述されています。.....	148
3. URL に「/home」を追加して確認します。.....	149
4. 他のページもログイン認証後にだけ表示するように変更.....	150
5. 全てのページがログイン後に表示されるか確認.....	151
6. ビューに追加された「ログイン認証」ファイルを確認.....	152
ログイン認証.....	153
ユーザー情報取得 & MIDDLEWARE 認証	153
ログイン認証したユーザーの情報を取得方法.....	154
MIDDLEWARE を使ったログイン認証チェック.....	155
サインイン・ログイン認証 1ユーザー × 1サービス	156
ユーザーがログインしたらユーザーが登録した本のみ表示.....	157
ユーザー毎にデータを別々で保持することができる.....	158
1. ユーザー <i>id</i> を登録できるように <i>books</i> テーブルを変更.....	159
2. データテーブルをリセット(削除).....	159
3. データテーブルを再構築.....	159
4. MySQL コマンドを使用し、テーブルができているか確認	160
5. 表示処理:コントローラ「 <i>BooksController@index</i> 」を修正.....	161
6. コントローラ「 <i>BooksController@store</i> 」の表示処理を修正.....	162
7. コントローラ「 <i>BooksController@update</i> 」の更新処理を修正.....	163
8. コントローラ「 <i>BooksController@edit</i> 」の更新表示を修正.....	164
CHAPTER17	165
ファイルアップロード	165

G'sACADEMY Lab10

1. books テーブルにファイル名保存用 カラムを追加.....	167
[database/migrations/*****create_books_table.php].....	167
item_img カラムを string (文字列型) で追加記述	167
2. books テーブルを再構築する.....	167
3. テンプレートに「ファイル選択」ボタンを表示。	168
books.blade.php.....	168
以下、form 要素内に追加してください。	168
books.blade.php.....	168
4. テンプレートに「サムネイル表示」を追加.....	169
books.blade.php.....	169
5. ファイルアップロード処理を追加する	170
6. public (公開) フォルダ内に upload フォルダを作成.....	171

Chapter1

PHP 初めてのフレームワーク ～フレームワーク概要～

G'sACADEMY Lab10

Laravel フレームワーク

日本で最も広く使われている WordPress (CMS) や EC-CUBE (EC サイト) などは Web プログラミング言語 PHP で開発しています。このように Web サービスではよく使われている PHP ですが、Web アプリケーションを高速に開発するための Web アプリケーションフレームワークと呼ばれているものが多数あります。今回学ぶ「 Laravel 」は、PHP の世界的にもっとも人気のあるフレームワークです。もともと PHP には「 Symfony 」という人気の Web アプリケーションフレームワークが存在しますが、それを土台に作成されたのが「 Laravel 」です。

そもそもフレームワークって？

フレームワークを使わない PHP を使うプログラマーは、基本的にほとんど 0 の状態からプログラミングをして作っていきます。プログラミングの内容は、1 ファイルに長く記述していく人、モジュール化（関数）する人、オブジェクト指向（クラス）を取り入れる人など、人によっていろんな作り方ができるのが PHP の良さでもあります。そのいろんな作り方ができることから初心者から上級者まで幅広く使われています。

しかし、複数人でチームを組んで開発する場合には、この PHP の自由さが「規則性のないコード」を生産していきます。チームでの規則を決めても PHP はコードを書けば動いてしまうので、人によっては作りやすい方法で規則を無視して作成してしまう人も稀にでてきます。このようなことを防ぐにはどうするか？もっと規則が最初から決まっていて、そのとおりに組めれば楽なのに…。そこで「フレームワーク」の出番です。

最初から便利な枠組み、関数、クラスなど必要な処理が用意されています。「フレームワーク」は、その用意されているものを使うだけで実装できるので、使い方（実装の仕方）だけ覚えれば誰でも作れるのが良さです。ただし、関数、クラス、メソッドの名前など、「フレームワーク」の規則に従わないと動作しません。

「フレームワーク」は実装の負荷軽減、保守性、セキュリティ、規則性あるコード、可読性が確保できる特徴があります。

G'sACADEMY Lab10

Laravel の特徴

- 他のフレームワークに比べると学習コストが低い
- コードが読みやすく記述量が少ない
- 強力で簡単に記述できる入力チェック
- ログイン認証（ユーザー登録・認証など）
- ページネーションの簡単設置（2箇所の記述で可能）
- 本番環境へのアップロードが他のフレームワークに比べて簡単

何と言っても一番は「学習コストの低さ」です。他の言語（Python, Ruby, Java...）のフレームワークで Web アプリケーションを作成したとします、本番環境へのデプロイを考慮するとプログラミング言語以外のサーバー等の知識・技術が必要になります。PHP の場合は HTML/CSS/JavaScript をアップロードする感覚で本番環境に PHP ファイルをアップロードすることで動作させることができます。

※さくらインターネット（スタンダードプラン）、Microsoft Azure...

Laravel ヘルパー関数

ヘルパー関数とは、Laravel フレームワークで用意されている PHP ネイティブの関数とは別の関数です。

参考：laravel.com （ヘルパー関数）

<https://laravel.com/docs/6.x/helpers>

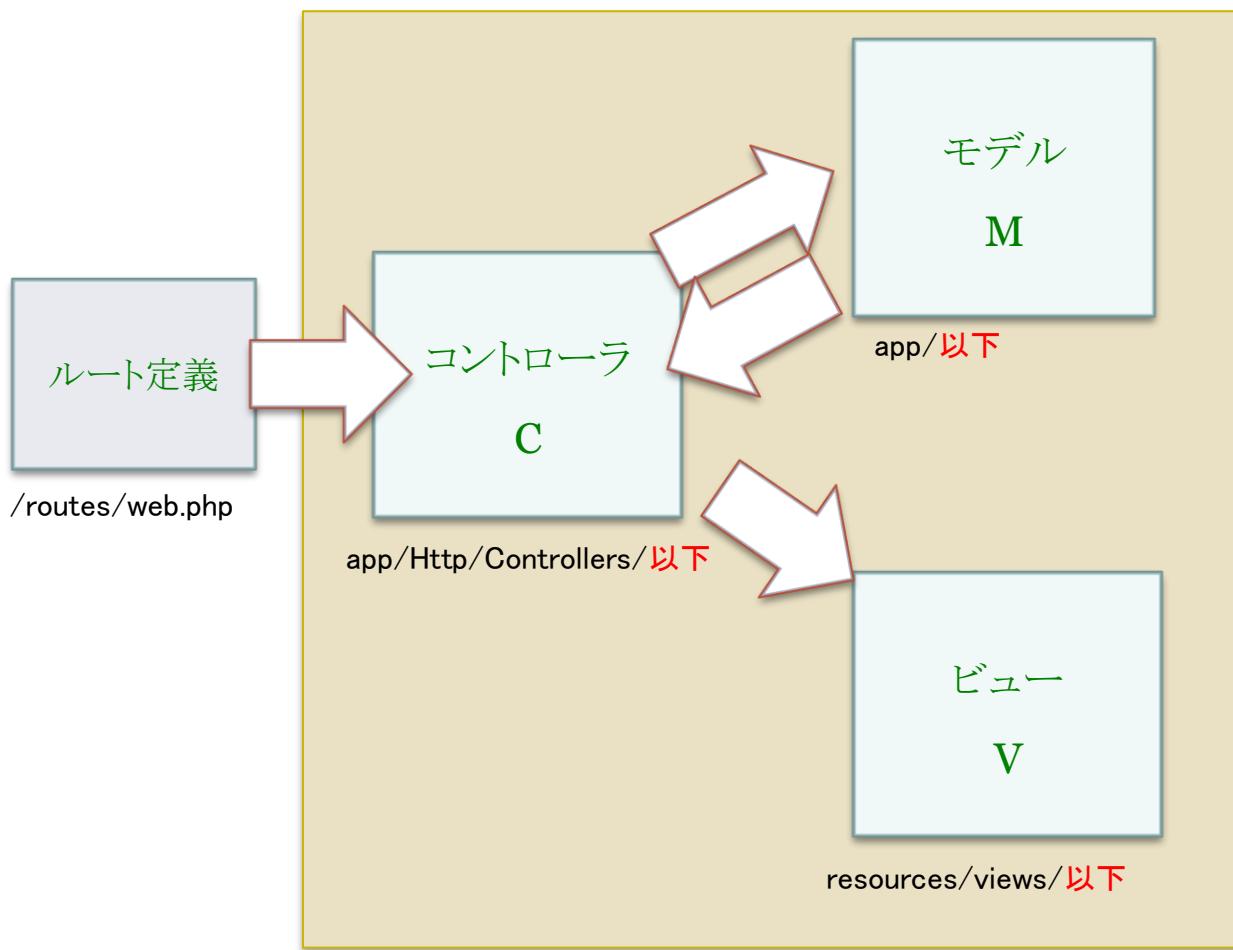
<https://laravel.com/docs/5.5/helpers>

G'sACADEMY Lab10

MVC パターンとは（一般的なフレームワーク）

PHP フレームワークである「 Laravel 」は他のフレームワークと同様 MVC パターンという「 Model ・ View ・ Controller 」の 3 つの主要パーティで構成されています。 MVC パターンは以下の 3 種類のプログラムで構成しています。

とは言っても、



初心者に MVC を理解しろと言うのは難しい話です。

初心者の段階では「 表示・データ・制御 」と別れているんだなという程度の理解で進んでいきましょう。フレームワークの概念を理解しようと思うなら、実際にコードを打って進んだほうが、明らかにスキルアップして、知識はあとで紐付いてスキルアップできます。頭の片隅に上記図を入れておいて、ステップバイステップで進めて、「 表示・データ・制御 」を紐付けていきます。

G'sACADEMY Lab10

MVC

Controller/コントローラ :

コントローラの役割は、モデルとビューの制御です。リクエストされた URL や フォームの入力値にしたがい処理を実行します。

コントローラは、基本的には「app/Http/Controllers」内に作成します。

Model/モデル :

Web システムの中でデータ連携処理などをモデルが扱います。Web システムのデータを扱う本体部分になります。データベースから必要な情報を抽出、登録、更新、削除などをおこすのが主な処理内容です。

View/ビュー :

コントローラ・モデルの実行結果を View で受け取る。ビューが受け取ったデータをもとに HTML を生成し、ブラウザで表示する HTML を出力します。

表示領域を担当するのがビューになります。

ビューは、基本的に resources/views 内に作成します。

※ルート定義 :

ルーティングと呼ばれており、URL と MVC の紐付けする役割をしています。

Laravel は MVC では無い？

Laravel 開発者「Taylor Otwell」氏は「Laravel は MVC」ではないとコメントしている。

M, C という概念に囚われず自由に書けるように作られているのが Laravel の特徴とも言える。

Chapter2

開発環境準備

G'sACADEMY Lab10

Laravel 開発環境（インストール型）

Laravel (PHP フレームワーク) を使用する場合、Web サーバー (Apache...)、PHP、データベース (MySQL, SQLite...) の環境を準備する必要があります。
※ Laravel 5.5 バージョンの場合 : PHP 7.0 以上を使用

主な環境環境

XAMPP (Apache, PHP, MySQL)



The screenshot shows the official XAMPP website. At the top, there is a dark blue header bar with the Apache Friends logo, a search bar, and a language selection dropdown set to 'JP'. Below the header, the main title 'XAMPP Apache + MariaDB + PHP + Perl' is displayed in a large, bold font next to the XAMPP logo (an orange square with a white 'X'). A section titled 'XAMPP とは？' (What is XAMPP?) provides a brief overview of the software. To the right of this text is a large orange button with a play icon and the word 'XAMPP'. Below the title, there are download links for Windows, Linux, and OS X. A green arrow-shaped button on the left points to the 'ダウンロード' (Download) link. At the bottom, there is a call-to-action button for 'Interested in XAMPP Docker Container?'

XAMPP は Apache, PHP, MySQL がセットになっているアプリケーションです。Windows/Mac どちらの OS にも手軽にインストールでき、自身のパソコン上で直ぐに PHP や MySQL を実行確認できるのが良さです。Wifi 環境が無い状態でも、自身のパソコンの中に Web サーバーやデータベースをインストールするのでパソコン単体で開発できるのも特徴です。

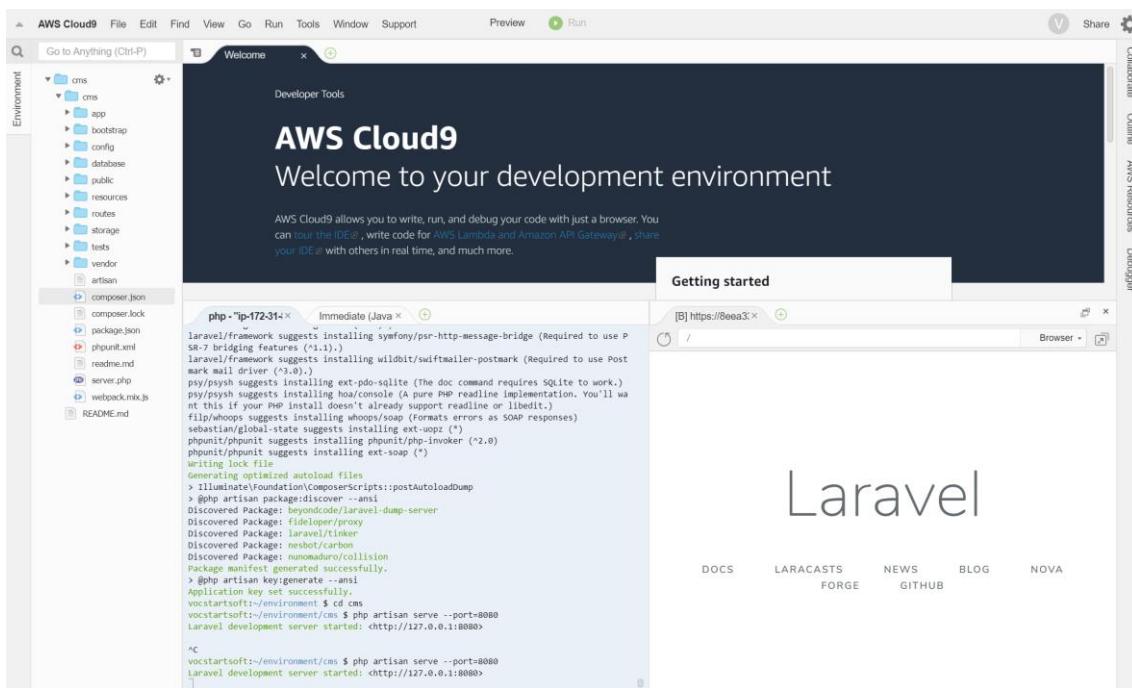
G'sACADEMY Lab10

AWS Cloud9 統合開発環境（クラウド型）

The screenshot shows the AWS Cloud9 landing page. At the top, there's a navigation bar with the AWS logo, 'サービス' (Services), 'リソースグループ' (Resource Groups), and a search bar. Below the header, it says 'Developer Tools' and 'AWS Cloud9'. The main title 'AWS Cloud9' is followed by the subtitle 'a cloud IDE for writing, running, and debugging code'. A brief description explains that AWS Cloud9 allows you to write, run, and debug your code with just a browser, providing immediate access to a rich code editor, integrated debugger, and built-in terminal with preconfigured AWS CLI. Below this, there's a large orange 'Create environment' button. To the right, there's a 'Getting started' sidebar with three sections: 'Before you start' (2 min read), 'Create a environment' (3 min read), and 'Working with environments' (15 min read).

AWS Cloud9 はクラウド上でプログラミングがおこなえる Web サービスです。Web ブラウザ上で操作するため、自身の OS に依存することなく開発できるのが特徴の一つです。プログラミング言語毎の開発環境構築に時間をかけなくて良いため、初心者には特におすすめの Web サービスです。プログラミングを習得する上で、問題となるのが開発環境構築でプログラミングを諦める人が多いですが、Cloud9 はクラウド上に各言語の開発環境を準備してくれるので、初心者には力強い味方と言えます。「PHP、Python、Ruby、Node.js ...」など多くの言語に対応しているクラウド IDE と呼ばれる「クラウド統合開発環境」です。開発環境だけでなくプログラムを記述するエディターも Web ブラウザで利用します。このエディターは使い勝手が素晴らしい、他によく使われている高機能エディター「VisualStudioCode」や「Atom」のような機能を備えています。また、コマンドラインも Amazon Linux/Ubuntu 上で動作しているため、Linux コマンドが使えるのも魅力です。

G'sACADEMY Lab10



利用したい言語が開発環境に最初から入っているので、クラウドサーバ上に自分で開発環境を整える必要が最小限ですみます。

クラウド統合開発環境の特徴は、自身のパソコンに依存することなくクラウド上に作成されるため、「AWS Cloud9」のサインインアカウント（ログインするID, Password）を持っていればWindows/Mac/LinuxなどOSに依存することなく、ブラウザさえあればどこからでもアクセスして開発を進めることができます。例えば、仕事先でWindows/Chromeで開発していたコードを途中でセーブ(保存)しておけば、自宅に帰ってから自宅のMac/FireFoxでAWS Cloud9にアクセスして開発を継続して進めることができます。このときに追加で何かを設定する必要は一切ありません。

AWS Cloud9を使えば、面倒な環境設定も少なく、どこにいてもどのパソコンからでも利用できるのです。

ソースコードはGitのホスティングサービスであるGitHubやBitbucketとの連携も可能ですがAWS Cloud9でTeam内のコード共有が可能なので、そちらを使ったほうが便利です。

G'sACADEMY Lab10

本書では「AWS Cloud9」を利用していきます。

<< 重要 >>

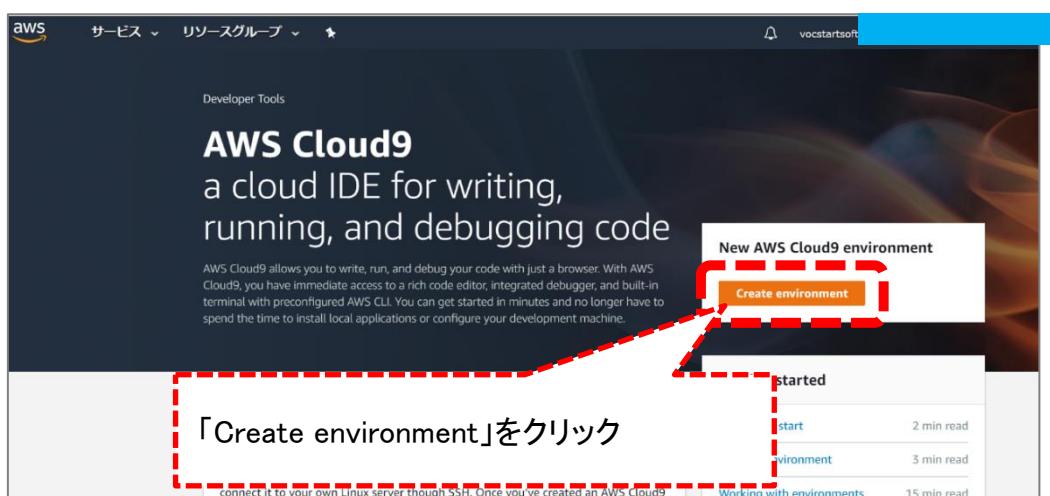
まずは AWS のアカウント登録して利用できる準備までおこなってください。

注意) AWS の登録にはクレジット登録が必用になります。

1. 「AWS マネジメントコンソール」画面から Cloud9 を検索

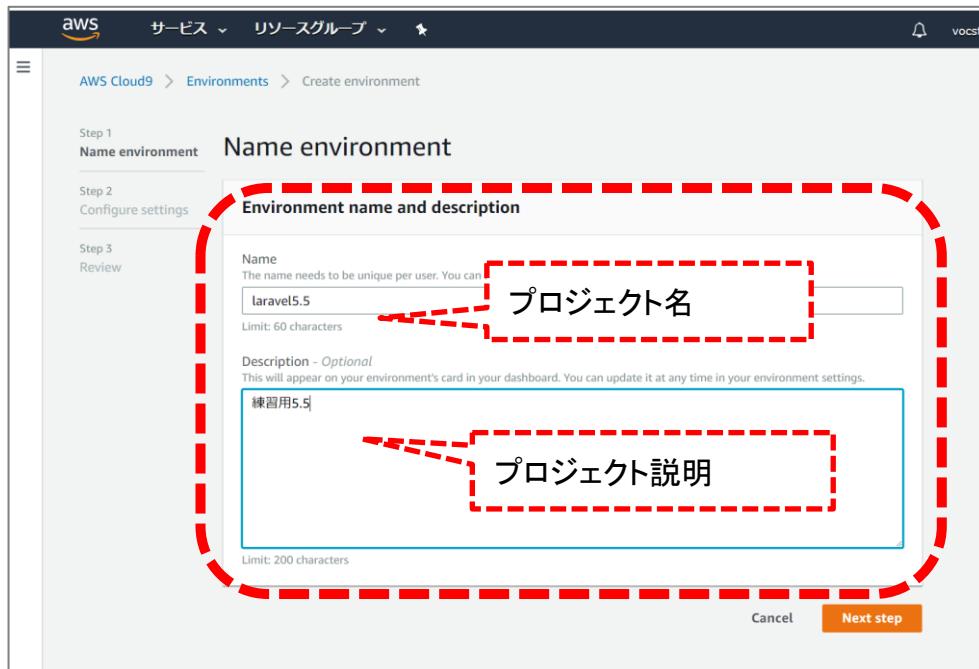


2. 「Create environment」クリック

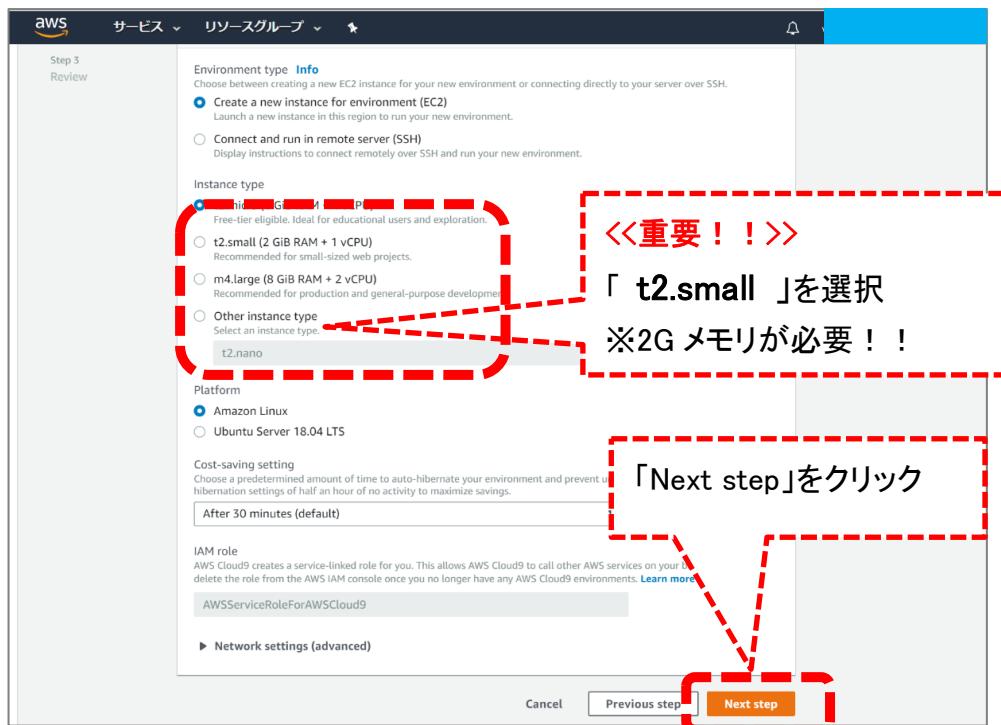


G'sACADEMY Lab10

3. “2箇所”を入力して「Next step」クリック



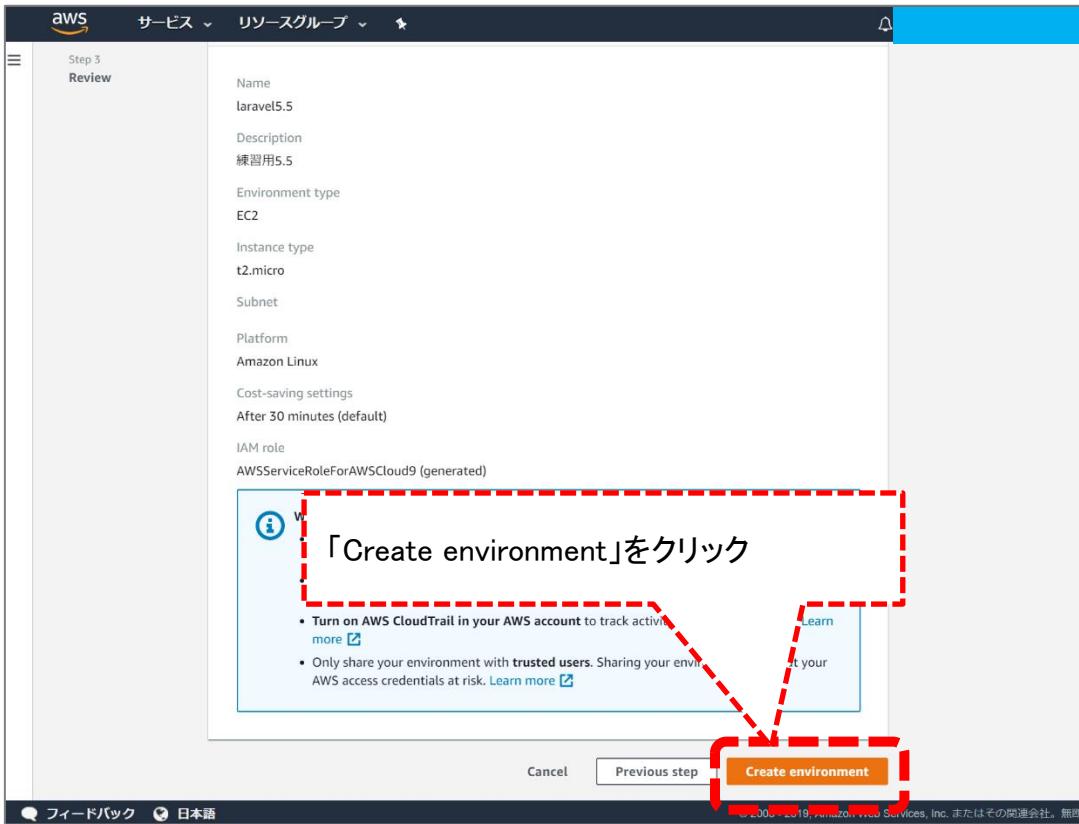
4. デフォルトの最小設定のまま「Next step」クリック



＜重要＞「Cost-saving setting」項目は重要で、初期設定は30分で強制的にログアウトし、作業中でも30分で一度再ログインが必要になります。時間を長く設定すると料金が高くなる可能性があります。最初は面倒でも初期設定にしておきましょう。

G'sACADEMY Lab10

5. 「Create environment」クリック

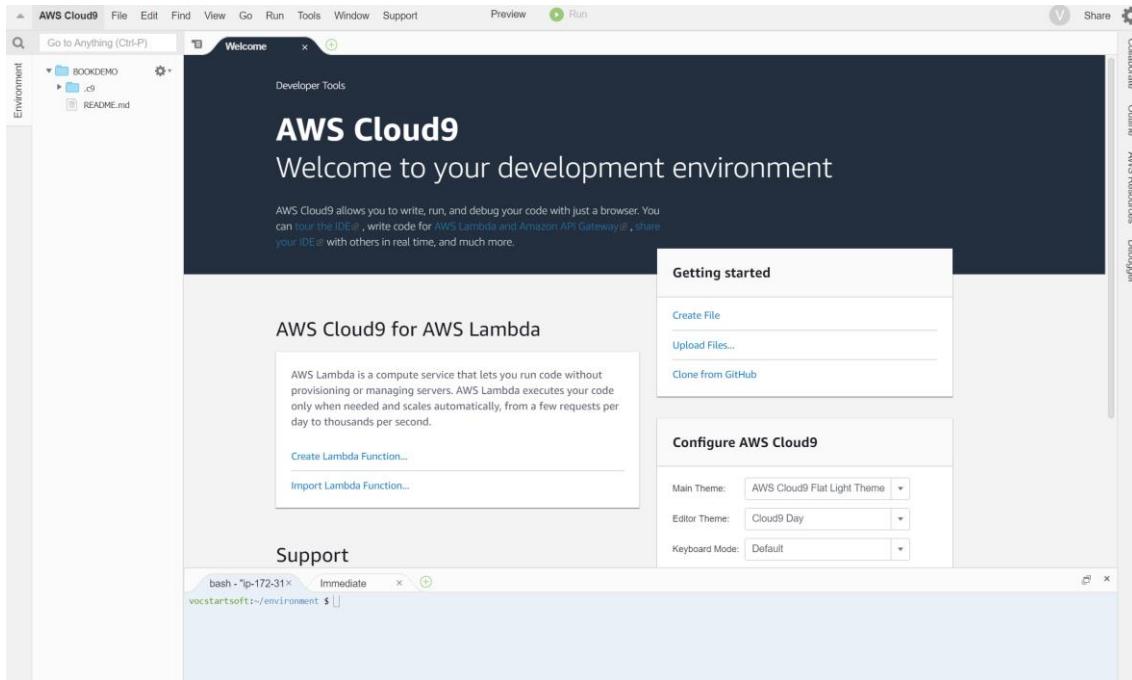


6. 「Cloud9」がロードされるので少し待ちます。



G'sACADEMY Lab10

7. 「Cloud9」画面がでれば完了です。

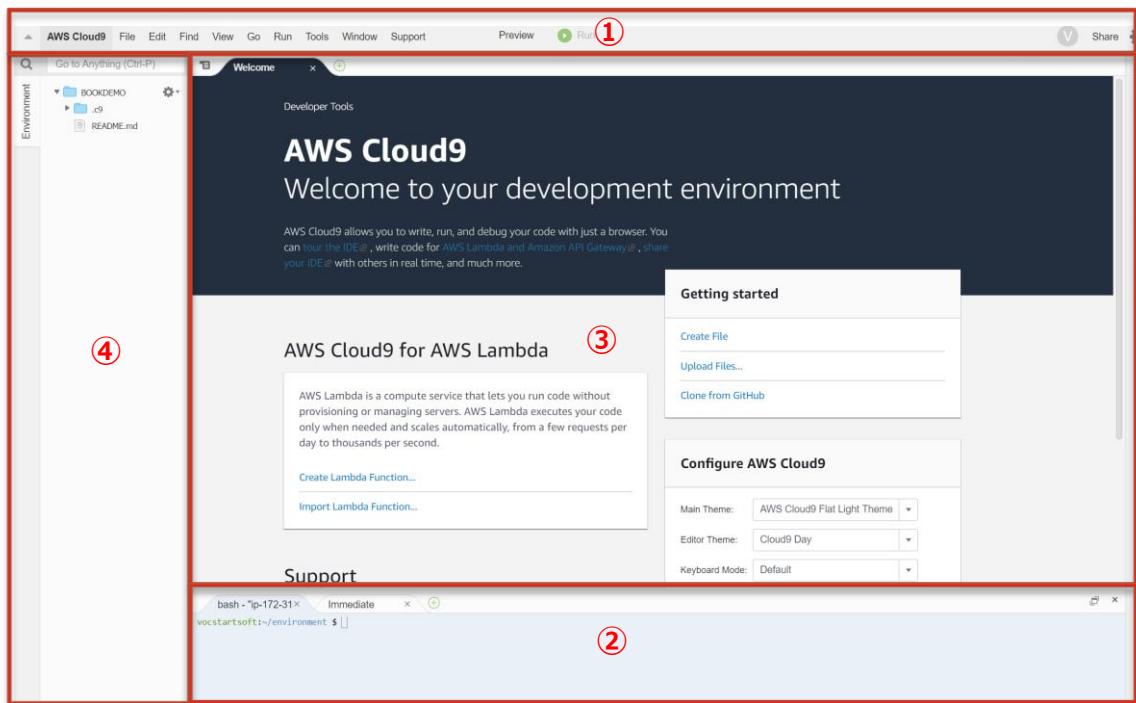


ここでCloud9の最初の設定が完了です。

G'sACADEMY Lab10

Cloud9 の画面構成（重要：必ず見る）

Cloud9 の画面構成を確認しましょう。



1. メニューバー

Cloud9 で使えるメニューが全て用意されています。

2. Workspace (プロジェクト)

ディレクトリとファイルを表示します。アプリのファイル構造を一覧できます。

3. エディター領域

HTML/CSS/JavaScript/PHP... などプログラミングを入力する画面領域になります。高度なエディターですので、要素・属性・変数・関数などの色分けをしてくれます。Emmet も使えます。

4. コマンド領域

Mac でいうとターミナル、Windows ではコマンドプロンプトの画面両域です。

本書ででてくるコマンドはここに入力して実行します。例) `php -v`

G'sACADEMY Lab10

Chapter3

Laravel インストール

ここからは手を動かしながら進めていきましょう。

◇おすすめ！最短コマンド集

<https://github.com/yamazakidaisuke/GsCodeSample>

ファイル名 : Laravel6_start_aws.txt

こちらの Github に載せてるコマンドを参照し、記載されてる順番にコマンドをコピー＆ペーストで貼っていくだけで最小のアプリが完成します。
とりあえず「一通りの流れ」を体験したい人にはオススメです！

G'sACADEMY Lab10

Cloud9 の環境確認

次に PHP のバージョンと Composer のバージョンを確認しつつ、両アプリとも動作しているのかを確認します。

1. PHP バージョン確認

php -v

上記「`php -v`」のコマンドで PHP のバージョンを確認します。

```
php - "ip-172-31-× Immediate × +  
vocstartsoft:~/environment $ php -v  
PHP 5.6.40 (cli) (built: Mar  8 2019 18:17:39)  
Copyright (c) 1997-2016 The PHP Group  
Zend Engine v2.6.0, Copyright (c) 1998-2016 Zend Technologies  
    with Xdebug v2.5.5, Copyright (c) 2002-2017, by Derick Rethans  
vocstartsoft:~/environment $
```

※執筆時: PHP バージョン 5.6.40 が表示される

2. composer のインストール

composer は PHP ライブライリの使用を管理してくれる便利ツール

◇以下コマンドでインストールと確認

```
curl -sS https://getcomposer.org/installer | php
```

```
sudo mv composer.phar /usr/bin/composer
```

composer

上記のように「Composer」とコマンド一覧が表示されればOKです。

※少し上に隠れるのでスクロールして確認してください。

G'sACADEMY Lab10

Cloud9 の PHP を「 PHP7.2 」にバージョンアップ

PHP のバージョンをアップデートします。

Laravel5.6 以降では「PHP 7.1」以上が必須となります。以下のコマンド手順で PHP を 7.2 までアップデートしましょう。

以下の順番でコマンドを実行します。

```
sudo yum -y install php72 php72-cli php72-common php72-devel php72-
mysqlnd php72-pdo php72-xml php72-gd php72-intl php72-mbstring php72-
mcrypt php72-opcache php72-pecl-apcu php72-pecl-imagick php72-pecl-
memcached php72-pecl-redis php72-pecl-xdebug
```

```
sudo alternatives --set php /usr/bin/php-7.2
```

※エラーなどでなければ問題なくインストールできています。

以下コマンドで PHP のバージョンを確認しましょう。

```
php -v
```

以下のようにバージョンが上がったことが確認できます。

```
vocstartsoft:~/environment $ php -v
PHP 7.2.16 (cli) (built: Mar  8 2019 18:56:55) ( NTS )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.2.0, Copyright (c) 1998-2018 Zend Technologies
    with Zend OPcache v7.2.16, Copyright (c) 1999-2018, by Zend Technologies
    with Xdebug v2.6.1, Copyright (c) 2002-2018, by Derick Rethans
vocstartsoft:~/environment $
```

G'sACADEMY Lab10

Laravel フレームワークの準備

次に Laravel インストーラーを準備します。

```
composer global require "laravel/installer"
```

上記コマンドを入力しましょう。

```
- Installing ralouphie/getallheaders (2.0.5): Downloading (100%)
- Installing psr/http-message (1.0.1): Downloading (100%)
- Installing guzzlehttp/psr7 (1.5.2): Downloading (100%)
- Installing guzzlehttp/guzzle (6.3.3): Downloading (100%)
- Installing laravel/installer (v2.0.1): Downloading (100%)
symfony/contracts suggests installing psr/cache (When using the Cache contracts)
symfony/contracts suggests installing psr/container (When using the Service contracts)
symfony/contracts suggests installing symfony/cache-contracts-implementation
symfony/contracts suggests installing symfony/service-contracts-implementation
symfony/contracts suggests installing symfony/translation-contracts-implementation
symfony/console suggests installing symfony/event-dispatcher
symfony/console suggests installing symfony/lock
symfony/console suggests installing psr/log (For using the console logger)
guzzlehttp/guzzle suggests installing psr/log (Required for using the Log middleware)
Writing lock file
Generating autoload files
vocstartsoft:~/environment $
```

上記のように終了していれば問題ありません。

これで Laravel を使用する前準備ができました。

<composer のメリット>

composer コマンドを使用することで、必要な他のライブラリも一緒にダウンロードして配置してくれます。composer が無いと、全て自分で探し、ダウンロードし、配置まで自分でする必要があります。

G'sACADEMY Lab10

Laravel プロジェクトを作成します(各バージョン)

以下コマンドを入力し Laravel フレームワーク一式をインストールします。

//6 の最新バージョンを選択する場合(今回はこちら!)

```
composer create-project laravel/laravel cms 6.* --prefer-dist
```

//5.5 バージョンを選択する場合

```
composer create-project laravel/laravel cms 5.5.* --prefer-dist
```

//最新のバージョンを選択したい場合

```
composer create-project laravel/laravel cms
```

※ PHP バージョンに合わせて Laravel の使える中での最新バージョンを選択してくれます。

上記の「**cms**」はプロジェクト名です。プロジェクト名とは今回作成するアプリ名（まとめて管理するフォルダ名のようなもの）です。

※Workspaces で入力したアプリ名フォルダの中に「**cms**」が作成されます。

```
pnunit/pnunit suggests installing pnunit/pnp-invoker (^1.1)
Package phpunit/phpunit-mock-objects is abandoned, you should avoid using it. No replacement was suggested.
Writing lock file
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover
Discovered Package: fideloper/proxy
Discovered Package: laravel/tinker
Discovered Package: nesbot/carbon
Package manifest generated successfully.
> @php artisan key:generate
Application key [base64:4QzgtbxrTHqVQTvWxosj/wMVFACPcVWd5yEYlPbMKxY=] set successfully.
vocstartsoft:~/environment $
```

インストール完了

インストールとは？

Laravel フレームワークをダウンロードして配置することを「インストール」と呼んでいます。一般的に MicrosoftOffice などをパソコンにインストールするなどという言葉で「インストール」を使いますが、Laravel フレームワークではファイル一式をダウンロードして配置することをそう呼んでいます。

G'sACADEMY Lab10

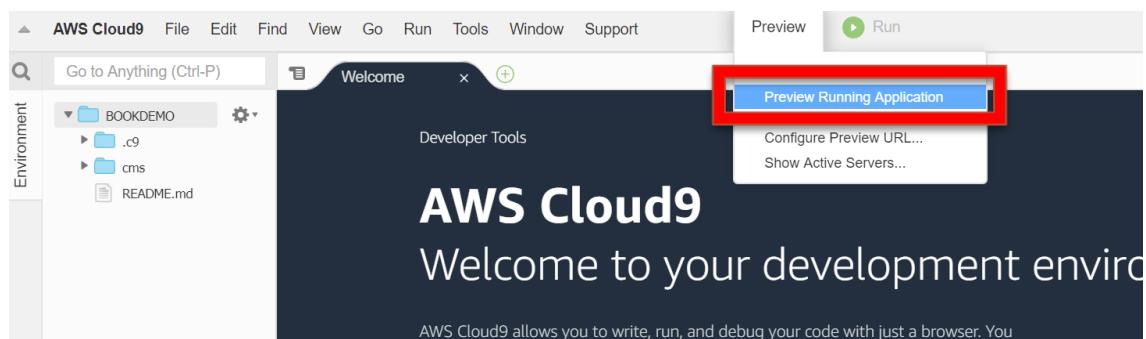
Laravel の「インストール確認」と「Preview 設定」

Laravel インストールの確認をするためには、Web サーバーを起動して動作確認する必要があります。AWS Cloud9 にはワークスペースごとに使用できる Web サーバーが搭載されています。Web ブラウザを使いページを表示することで確認を行います。

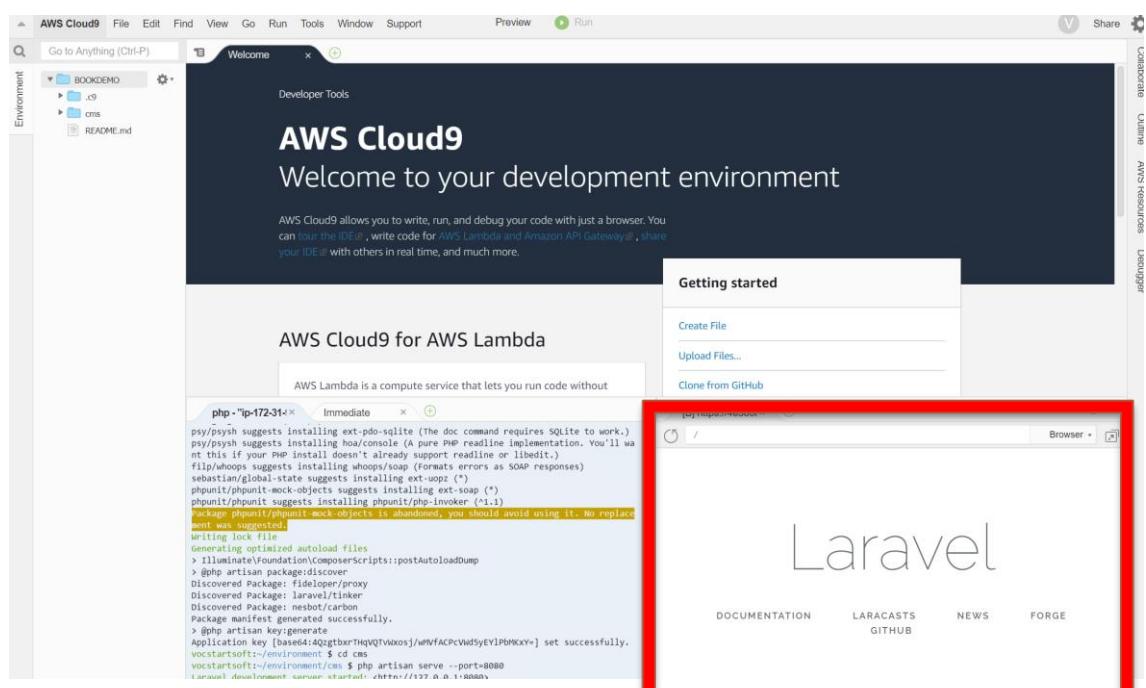
1. cms フォルダに移動し、Built-in web サーバーを起動します。

```
cd cms  
php artisan serve --port=8080
```

2. 「Preview」をクリックし「Preview Running Application」を選択。



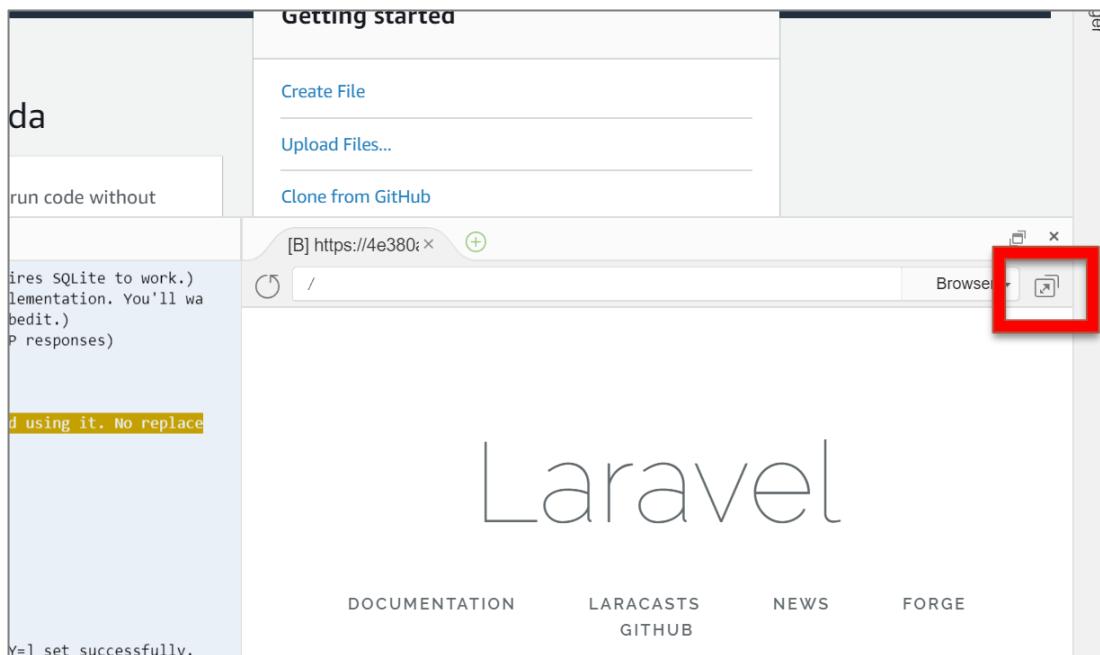
Laravel のページが見れていれば OK です。



G'sACADEMY Lab10

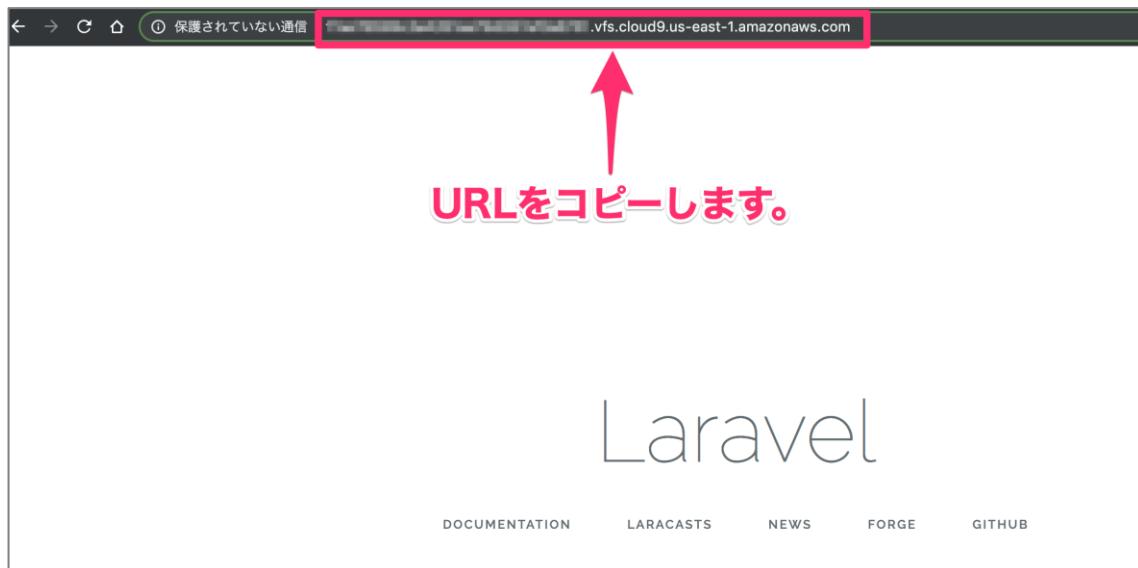
3. 動作確認は「1画面表示」で行う必要があります！

以下赤枠のアイコンをクリックすると1画面で表示することができます。



4. 別タブ（1画面表示）でWebページが表示されます。

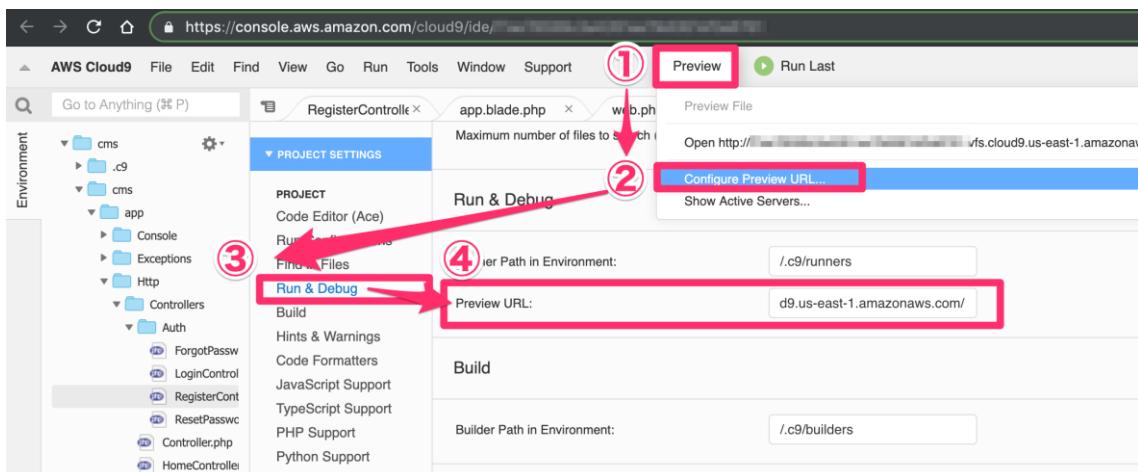
* 「URL」をコピーします。



G'sACADEMY Lab10

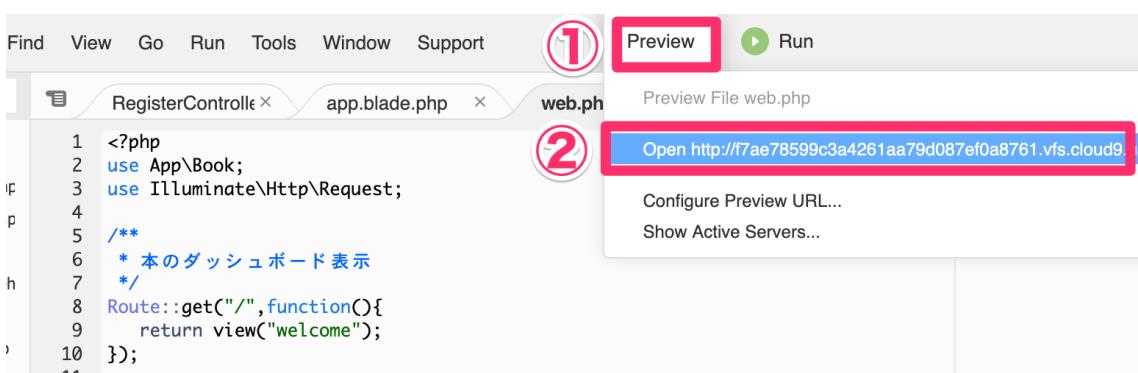
5. エディタ画面を表示して以下の手順で操作します。

1. 「Preview」をクリック
2. 「Configure Preview URL...」をクリック
3. 「Run & Debug」をクリック
4. 「Preview URL:」の入力欄にコピーしたURLを貼り付け。



これで別タブで表示する設定が完了しました。

6. 別タブで表示されるか確認。



※設定が反映されても別タブで表示されないことが偶にあります。

Web サーバーを STOP する方法

コマンド領域をクリックし、

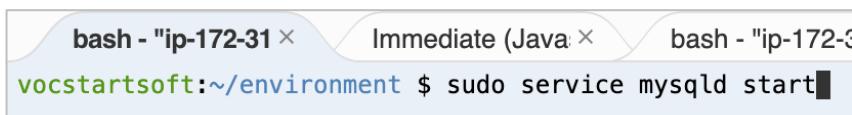
windows	Ctrl + C
Mac	Cmd + C

G'sACADEMY Lab10

超重要：毎回使うので覚えておくこと

以下コマンドはよく使います。PC のデスクトップなどにメモして置きましょう

1. MySQL を起動



```
bash - "ip-172-31" × Immediate (Java) × bash - "ip-172-31" ×  
vocstartsoft:~/environment $ sudo service mysqld start
```

sudo service mysqld start

2. Web サーバーを起動（cms フォルダ内で実行）

1. タブ「New Terminal」をもう1つ起動させます。



2. 新しい「Terminal」タブでWeb サーバーを起動すると、もう一つのターミナルがコマンド専用で使えるので便利です。



③もう一つのタブでWebサーバ起動

```
bash - "ip-172-31" × Immediate (Java) × bash - "ip-172-31" ×  
vocstartsoft:~/environment $ cd cms  
vocstartsoft:~/environment/cms $ php artisan serve --port=8080
```

タブを複数立ち上げ上手に使うのがコツ

php artisan serve --port=8080

ログアウトすると、Web サーバーも MySQL もストップします。そのため、毎回上記コマンドで動作させる必要があります。ここが面倒です。コマンドをテキストファイル等に写しておき、いつでもコピペができるようしておきましょう。

DB 作成

先に Database の設定をしましょう。

G'sACADEMY Lab10

DB を起動して、データベースを作成

1. MySQL 起動する (毎回ログイン後に必要なコマンド！)

```
sudo service mysqld start
```

2. MySQL に root ユーザーで入る

```
mysql -u root -p  
[Enter] ※パスワードなし
```

3. 「c9」 データベースを作成

```
create database c9;
```

4. 一覧表示コマンドで「c9」 データベースを確認。

```
show databases;
```

```
mysql> show databases;  
+-----+  
| Database |  
+-----+  
| information_schema |  
| c9 |  
| mysql |  
| performance_schema |  
| phpmyadmin |  
+-----+  
5 rows in set (0.00 sec)
```

5. 「c9」 データベースを選択

```
use c9;
```

```
bash - "yamazaki" × bash - "ubuntu@" × Immedia  
mysql> use c9  
Database changed
```

これで「c9」 データベースが存在していることがわかります。

G'sACADEMY Lab10

6. テーブルはまだありません。

```
show tables;
```

```
mysql> show tables;  
Empty set (0.00 sec)
```

```
mysql> █
```

「Empty」が表示される。テーブルが無いという意味。

7. root ユーザーのパスワードを「root」に設定変更

```
update mysql.user set password=password('root') where user='root';
```

8. 設定変更を反映させる

```
flush privileges;
```

9. MySQL から抜けます

```
exit;
```

<< パスワード変更が反映されない場合 >>

確認のため、一度「exit;」した後に再度 MySQL に入ってパスワードの確認をしたほうが良いでしょう。パスワード変更が反映されていなければ、MySQL の再起動を行います。

```
sudo service mysqld restart
```

※exit; した後に実行です！

G'sACADEMY Lab10

MySQL を使用しない場合 (SQLite を選択する場合)

※本書では MySQL を使用しますので、SQLite は使用しません。

標準では、「 .env ファイル」で mysql が指定されているので「 sqlite 」を使用するように設定変更する必要があります。

SQLite 使用設定

DB_DATABASE の行をコメントアウトします。

DB_CONNECTION=sqlite (初期値は” mysql ” になっている)
// DB_DATABASE=homestead (コメントアウト)

DB ファイルを作成

「database」ディレクトリ内に database.sqlite ファイルを作成します。拡張子は「sqlite」です。

touch database/database.sqlite

POINT: 「 database.sqlite 」と DB ファイル名は固定です。

Chapter4

Laravel ルートディレクトリ フォルダ構成

G'sACADEMY Lab10

Laravel ルートディレクトリ：フォルダ構成

様々なディレクトリが用意されています。

app

アプリケーションのコアコードを配置します。コントローラ・モデルも基本的にこのフォルダに配置します。

bootstrap

フレームワークの初期起動やオートローディングの設定を行う起動コードファイルを含んでいます。その中の `cache` ディレクトリは初期処理のパフォーマンスを最適化するため、フレームワークが生成するいくつかのファイルが保存されるフォルダです。

config

アプリケーションの全設定ファイルが設置されています。

database

データベースのマイグレーションと初期値設定(シーディング)を配置します。ご希望であれば、このファイルを SQLite データベースの設置場所としても利用できます。

public

フロントコントローラとアセット(画像、JavaScript、CSS など)を配置します。

※DocumentRoot になるディレクトリ(URL から見れる場所)

resources

ビューやアセットの元ファイル(LESS、SASS、CoffeeScript)、「言語」ファイルを配置します。

routes

ルート定義をおこなうファイルを置きます。Web.php と api.php があり api.php は API 利用する場合との使い分けとなります。

G'sACADEMY Lab10

storage

app、framework、logs ディレクトリの3つのフォルダがあります。app ディレクトリはアプリケーションにより使用されるファイルを保存するために利用します。framework ディレクトリはフレームワークが生成するファイルやキャッシュに利用されます。最後の logs ディレクトリはアプリケーションのログファイルが保存されます。

tests

自動テストを配置します。サンプルの PHPUnit テストが最初に含まれています。

vendor

Composer でインストールしたパッケージが配置されます。

<POINT>

最初は下記 5 つのディレクトリ & ファイルをよく覚えておきましょう。

この 5 つを最初に覚えれば、基本的な部分は抑えられます。

1. ./env (環境設定)
2. /config/app.php (アプリケーション基本設定)
3. /routes/web.php (ルート定義)
4. /resources/views/以下 (ビュー・HTML などの表示要素を設定)
5. /app/Http/Controllers/以下 (コントロール/モデルを設定)

G'sACADEMY Lab10

「JavaScript & CSS」ファイルの配置場所は「public」

読み込む JavaScript/CSS ファイルは public フォルダ内に配置します。

デフォルトでは `asset()` 関数を使用します。`asset()` 関数は「/public」以下を参照するので、以下のサンプルコードのように記述して外部ファイルを読み込みましょう。

以下は「/public/assets」フォルダを作成し、「assets/js/」以下や「assets/css/」以下に配置した場合の記述例となります。

JavaScript

```
public > assets > js > *****.js
```

```
<script src="{{asset('/assets/js/jquery.min.js')}}"></script>
<script src="{{asset('/assets/js/bootstrap.min.js')}}"></script>
```

CSS

```
public > assets > css > *****.css
```

```
<link href="{{asset('/assets/css/bootstrap.min.css')}}" rel="stylesheet">
<link href="{{asset('/assets/css/style.css')}}" rel="stylesheet">
```

* https 対応

`asset` 関数の”第2引数”に `true` を渡すと `https://...` が付きます。

※第2引数を指定しない場合 `false` となり、`http://...` となります。

```
<script src="{{asset('/assets/js/jquery.min.js', true)}}></script>
<link href="{{asset('/assets/css/style.css', true)}}" rel="stylesheet">
```

G'sACADEMY Lab10

Chapter5

練習アプリ：本管理アプリを作っていきましょう。

G'sACADEMY Lab10

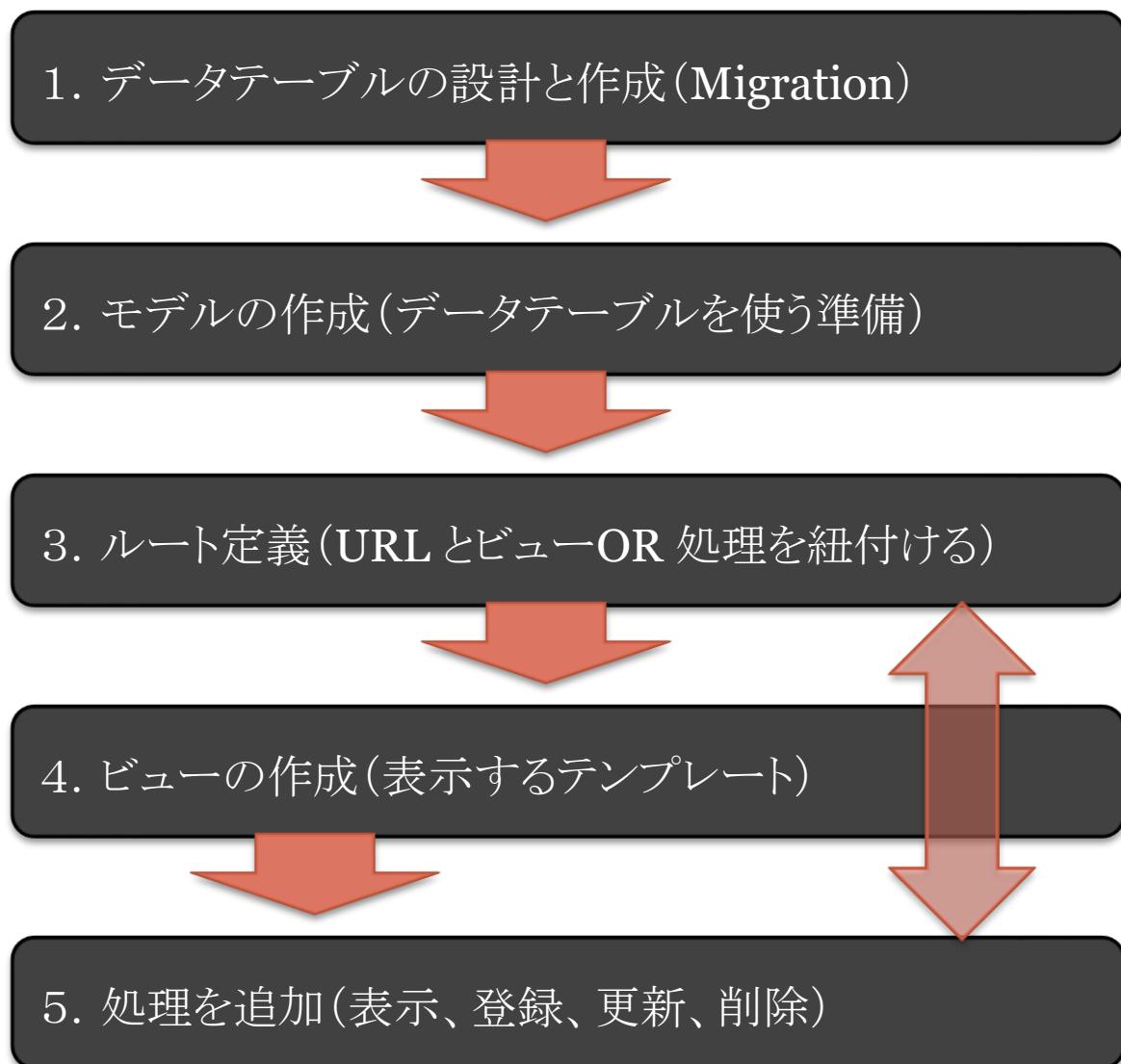
練習アプリ：本管理アプリを作っていくましょう。

完成形画面

The screenshot shows a user interface for managing books. At the top, there is a text input field labeled "本のタイトル" (Book Title) with a placeholder "本のタイトル" and a blue "Save" button below it. Below this, the text "現在の本" (Current Books) is displayed. Underneath, a section titled "本一覧" (List of Books) contains five entries, each with a red "削除" (Delete) button to its right:

- PHP基礎
- PHP応用
- Laravel基礎
- Laravel基礎 2
- Laravel応用 1

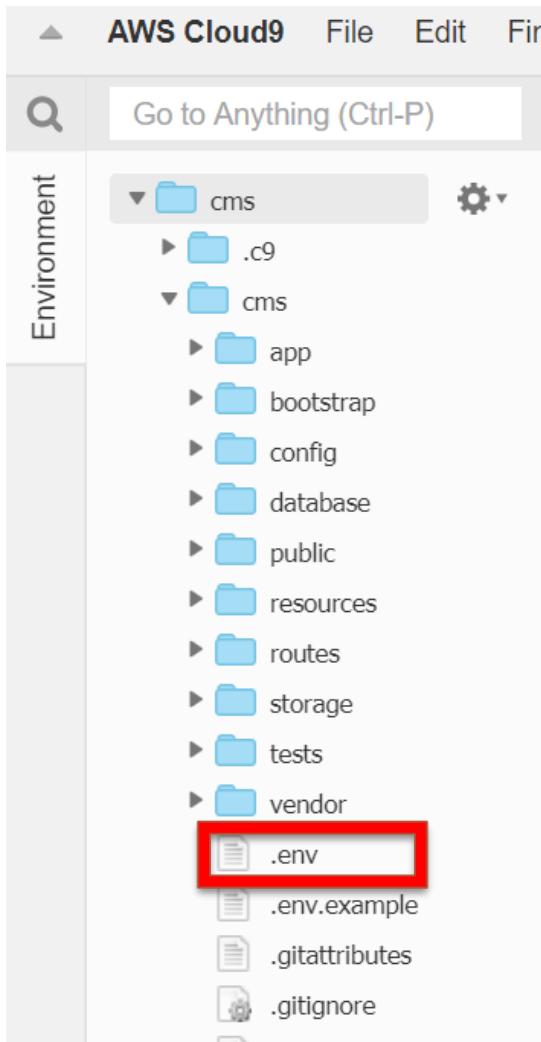
今回のアプリ制作の流れ



G'sACADEMY Lab10

データベースの設定

1. 「.env ファイル」を探します。

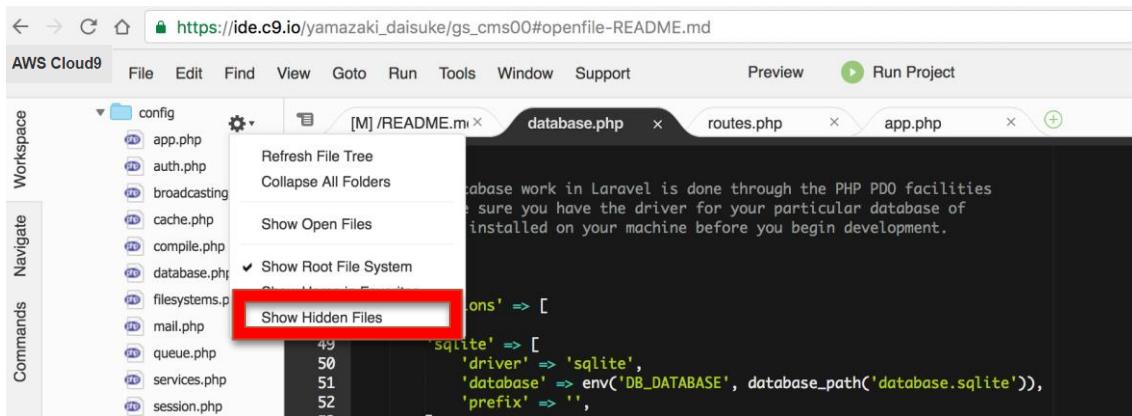


2. 「.env ファイル」は Laravel の設定ファイルです。DB の設定などをこのファイルでおこないますので、最初に表示されているか確認しましょう。

3. 「.env ファイル」が見つからない場合
(Cloud9、Mac など非表示になっているケースがあります)

G'sACADEMY Lab10

4. ワークスペースのファイルツリー右上ギアアイコンから [Show Hidden Files]にチェックで隠しファイルが表示されます



MySQL 選択する場合

先に、プロジェクトフォルダ内の「.env」ファイルを開き、
「.env」ファイル内のデータベース設定をアカウントに合わせた MySQL 用に変更します。

The screenshot shows the .env file in the AWS Cloud9 IDE. The MySQL connection settings (DB_CONNECTION=mysql, DB_HOST=127.0.0.1, DB_PORT=3306, DB_DATABASE=c9, DB_USERNAME=[REDACTED], DB_PASSWORD=[REDACTED]) are highlighted with a large red box. Below the file content, a red arrow points down to the corresponding configuration in the database.php file.

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=base64:hF1pyKpFB8qFsxNqynyrsMEFFfKk4Lx3W1VdNyDCrfI=
APP_URL=http://localhost

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=c9
DB_USERNAME=[REDACTED]
DB_PASSWORD=[REDACTED]

VER=file
RTVER=file
CLK=sync
```

DB_HOST=localhost
DB_DATABASE=c9
DB_USERNAME=root
DB_PASSWORD=root
上記に書き換えます。

G'sACADEMY Lab10

Chapter6

練習アプリ

本管理アプリを作っていきましょう。

データベースにテーブルを作成しましょう！

サンプルコード

http://www.venezia-works.com/aws/laravel6_01.zip

G'sACADEMY Lab10

データベースマイグレーション

最初にデータベースのテーブルを作成するための構造を PHP ファイルで定義します。そして artisan コマンドの「make:migration」と「make:migrate」を順番に組み合わせて使いテーブルを作成します。MySQL(SQLite)や phpMyAdmin 管理画面などを使用せず構築することが可能です。(DB の設定は.env ファイル。)

これをひとまとめにデータベースマイグレーションと呼んでいます。

Laravel のデータベースマイグレーションは PHP コードからデータベーステーブルの構造を定義、更新、削除するための簡単な仕組みが備わっています。

解説) コマンド書式 : マイグレーション

```
php artisan make:migration ファイル名 --create=テーブル名
```

上記コマンドを実行すると

以下のような Migration ファイルが作成されます。

例) [年]_[月]_[日]_[時分秒]_[ファイル名].php

```
.....
public function up()
{
    Schema::create('テーブル名', function (Blueprint $table) {
        $table->increments('id');
        //ここに追加のカラムを記述していきます。
        $table->timestamps();
    });
}
.....
```

「/database/migrations/以下」に作成されます。

G'sACADEMY Lab10

練習) books テーブルを作成

今回は

- ・ファイル名「`create_books_table`」
- ・テーブル名「`books`」※テーブル名は最後に「s」を付ける（重要）

の名前でマイグレーションファイルとテーブルを作成していきます。

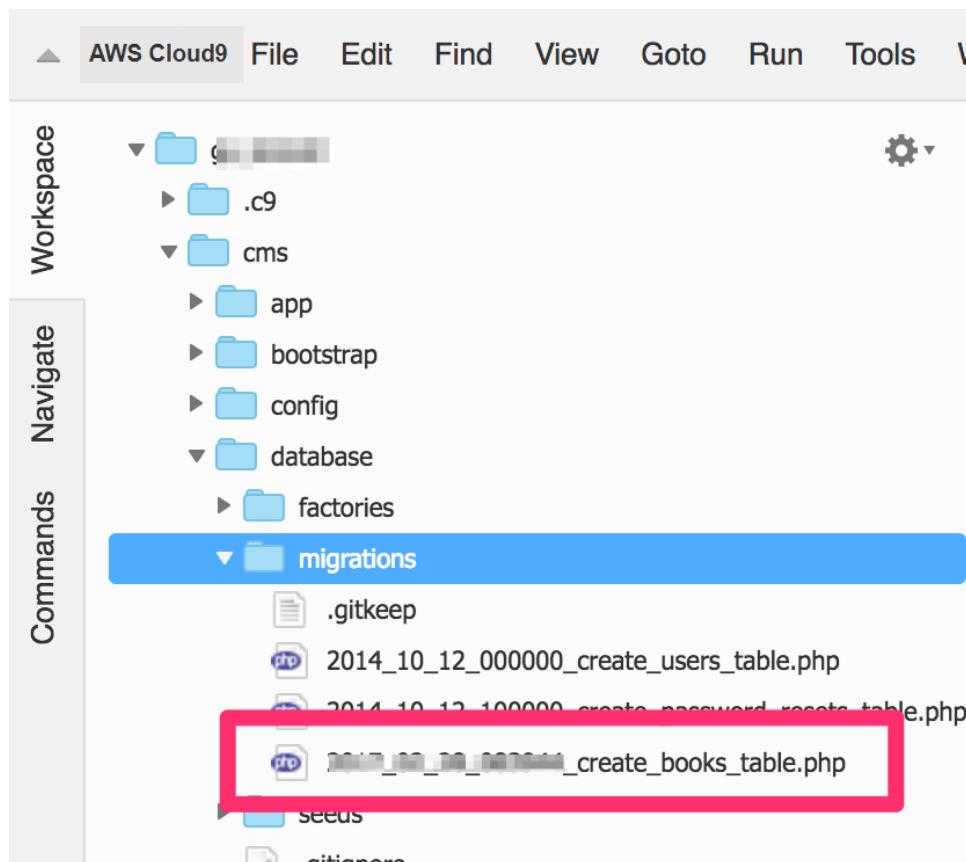
1. 「books テーブルを作成する」マイグレーションファイル作成

```
php artisan make:migration create_books_table --create=books
```

※コマンド「`create_books_table`」は作成されるファイル名になります。

何を保存するテーブルなのか意味のわかる名前をつけましょう。

※コマンド「`--create=books`」はテーブル名になります。



G'sACADEMY Lab10

2. マイグレーションファイルの「場所とファイル名」

/database/migrations/[年]_[月]_[日]_[時分秒]_[作成ファイル名].php

「名前にタイムスタンプが含まれている」のは、マイグレーション実行順(時間)をフレームワークに知らせるための記録のようなものです。

[作成ファイル名]で識別しますので、ファイル名の付け方は大事です。

3. マイグレーションファイルの構造を確認

マイグレーションコマンドを実行するとマイグレーションファイルには「up」と「down」の2つのメソッドが作成されています。

```
// up メソッド
// 新しいテーブル/カラム/インデックスをデータベース追加するために使用
public function up()
{
    Schema::create('テーブル名', function (Blueprint $table) {
        $table->increments('id'); //カラム定義
        $table->timestamps(); //カラム定義
    });
}

// down メソッド
// up メソッドで作成したテーブルを削除します。
public function down()
{
    Schema::drop('テーブル名');
}
```

※上記「テーブル名」には、今回は「books」が記述されます。

G'sACADEMY Lab10

カラム定義

新規テーブル作成では「create」メソッドを使用しカラムも定義します。
create メソッドには2つの引数が必要で、「第1引数には、テーブル名」、「第2引数には、function(Bluepring \$table) {....}」を記述します。
※ Bluepring はテーブル定義に使うオブジェクトです。

カラム定義の記述

```
$table->カラム型('カラム名');
```

必要なカラムの数だけ追加していきます。

カラムタイプ一覧

```
$table->increments('id');           //ID 自動採番(主キー)  
$table->string('email');          //VARCHAR カラム  
$table->string('name', 100);        //VARCHAR, 長さ指定カラム  
$table->integer('price');          //INTEGER カラム  
$table->text('description');      //TEXT カラム  
$table->dateTime('created_at');    //DATETIME カラム  
$table->timestamps();             //created_at と update_at カラムの追加  
$table->boolean('confirmed');     //true, false カラム  
$table->json('meta');              //JSON カラム
```

カラム修飾子

```
$table->string('email')->nullable(); //カラムに NULL を許可する
```

インデックス作成

```
$table->string('email')->unique(); //指定したカラムの値を一意にする
```

上記はリファレンスサイトから抜粋して紹介しています。

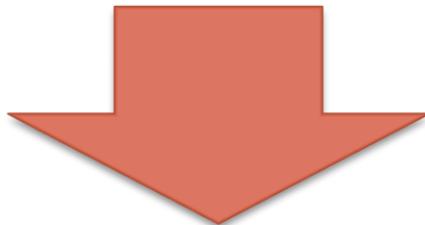
G'sACADEMY Lab10

4. 作成されたマイグレーションファイルに追加記述

[年]_[月]_[日]_[時分秒]_**create_books_table.php**

```
public function up()
{
    Schema::create('books', function (Blueprint $table) {
        $table->increments('id');
        $table->timestamps();
    });
}
```

※ファイルを開くと上記コードが最初から記述されています。



「books」テーブルには最低限のテーブルのカラムは設定されています。
次に「books」に必要なカラムを追加していきます。
今回の練習用テーブルには以下の「4カラムを追加」しましょう。

4カラム追加

- ・ item_name = 本の名前
- ・ item_number = 何冊
- ・ item_amount = 金額
- ・ published = 本公開日時

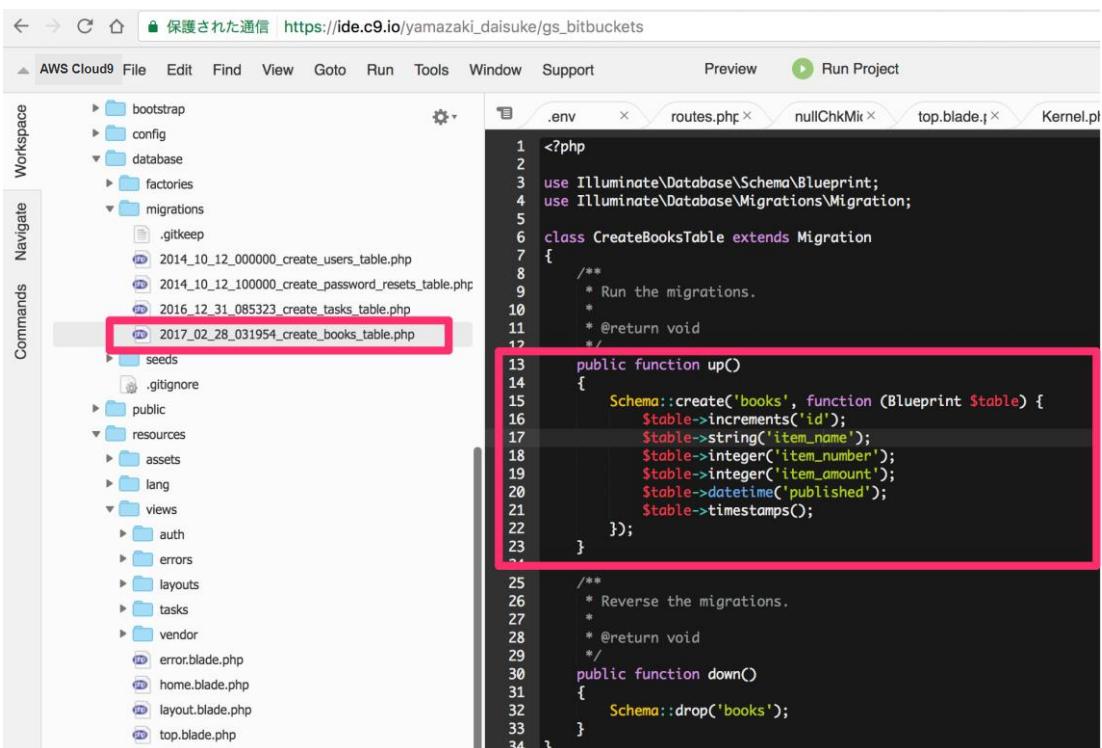
をファイルに追加記述します。

G'sACADEMY Lab10

5. 「up メソッド」に4カラム追加した状態

[年]_[月]_[日]_[時分秒]_create_books_table.php

```
public function up()
{
    Schema::create('books', function (Blueprint $table) {
        $table->increments('id');
        $table->string('item_name');
        $table->integer('item_number');
        $table->integer('item_amount');
        $table->datetime('published');
        $table->timestamps();
    });
}
```



The screenshot shows the AWS Cloud9 IDE interface. On the left, the workspace sidebar lists project files: bootstrap, config, factories (with migrations and .gitkeep), seeds, public, resources, assets, lang, views (auth, errors), layouts, tasks, vendor, error.blade.php, home.blade.php, layout.blade.php, and top.blade.php. A specific migration file, 2017_02_28_031954_create_books_table.php, is selected and highlighted with a red box. The main editor area displays the PHP code for this migration, specifically the 'up' method which creates the 'books' table with four new columns: item_name, item_number, item_amount, and published.

```
<?php
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateBooksTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('books', function (Blueprint $table) {
            $table->increments('id');
            $table->string('item_name');
            $table->integer('item_number');
            $table->integer('item_amount');
            $table->datetime('published');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('books');
    }
}
```

G'sACADEMY Lab10

修正が必要

「 /app/Providers/AppServiceProvider.php」を修正します。

```
// boot メソッド
use Illuminate\Support\Facades\Schema; //この行を追加 -----+
use Illuminate\Support\Facades\URL; //この行を追加
public function boot()
{
    Schema::defaultStringLength(191); //この行を追加 -----+
    URL::forceScheme('https'); //この行を追加
}
...
-----+
```

以下追加後のスクリーンショット

```
1 <?php
2
3 namespace App\Providers;
4
5 use Illuminate\Support\ServiceProvider;
6 use Illuminate\Support\Facades\Schema; //この行を追加
7 use Illuminate\Support\Facades\URL; //この行を追加
8 class AppServiceProvider extends ServiceProvider
9 {
10     /**
11      * Bootstrap any application services.
12      *
13      * @return void
14      */
15     public function boot()
16     {
17         //
18         Schema::defaultStringLength(191); //この行を追加
19         URL::forceScheme('https'); //この行を追加
20     }
21 }
```

修正解説 : Laravel 5.4 以上から標準 charset が utf8mb4 に変更され、標準の varchar (255) 設定が実行されると Cloud9 の MySQL バージョンではエラーになります。上記のコードで、varchar (191)に変更することで解消しています。また 「URL::forceScheme('https');」 は AWS/Cloud9 を使用する際に https でのブラウザ確認が必要なため設定しています (2020/05 時点)。

G'sACADEMY Lab10

6. マイグレーションを実行（テーブル作成）

マイグレーションを実行するには、artisan migrate コマンドを使います。

```
php artisan migrate
```

このコマンドは作成したマイグレーションファイル全てを実行し、「全データベーステーブル」を生成します。ただし、「1度実行したマイグレーションファイル」はテーブル migrations に登録され、「2回目は実行されません」。

データベーステーブルを調べると、マイグレーションで定義した「books テーブル」を確認できます。

7. MySQL にて実行確認

1. mysql -u root -p
2. root ※パスワード入力
3. use c9;
4. show tables;
5. desc books;

```
mysql> desc books;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id   | int(10) unsigned | NO  | PRI | NULL    | auto_increment |
| item_name | varchar(255) | NO  |     | NULL    |                |
| item_number | int(11) | NO  |     | NULL    |                |
| item_amount | int(11) | NO  |     | NULL    |                |
| published | datetime | NO  |     | NULL    |                |
| created_at | timestamp | YES |     | NULL    |                |
| updated_at | timestamp | YES |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

6. exit;

テーブルが作成されていることを確認しましょう。

※id, created_at, updated_at のカラムは自動で追加作成されます。

id カラムは1から順に番号を自動で登録してくれます（ユニーク値）。

created_at, updated_at のカラムは、日時を自動でデータ登録します。

G'sACADEMY Lab10

<<migrate コマンド一覧>>

migrate	Migration ファイルを実行
migrate:install	migrations テーブルを作成します
migrate:refresh	Migration を再実行してテーブルを再構築します テーブルを初期化。データも初期化。
migrate:reset	全ての Migration 操作を元に戻す（全削除）
migrate:rollback	一つ前の Migration 操作した情報に戻す
migrate:status	Migration ファイルと実行状態を確認できる

・ テーブル定義変更

Migration を使用して定義変更する。

一度実行した Migration ファイルは「`php artisan migrate`」で再度実行したくても、migrations テーブルで状態を管理されているため実行されません。定義変更用に新しく「Migration ファイル」を作りましょう。

・ テーブル定義を初期化したい

「`php artisan migrate`」や「`php artisan migrate:reset`」ではなく、

`php artisan migrate:refresh`

を実行します。

`migrate:reset` はロールバックさせ元に戻すため再構築に手間がかかるてしまいます。一度初期化したい場合には、「`php artisan migrate:refresh`」が良いでしょう。注意も必要で初期化して「データが無くなる」ことがわかっている前提で使いましょう。

また、`migrate:refresh` でも初期化できない場合には、`migrate:reset` でロールバックさせ元に戻す方法になります。

G'sACADEMY Lab10

<<コラム>>

チーム開発でのメリット

データベースマイグレーションを使用することで、チーム開発など複数のメンバーで開発しているときに便利です。チームメンバーは各自に開発環境を持っていることが多いです。その場合、データベースマイグレーションを使用することで、データベースの構造を各チームメンバーが追加できるようになります。勝手に追加されることが無く、artisan 「make:migrate」 コマンドを実行しない限り、データテーブルの変更はありません。

参考 : laravel.com (マイグレーション、テーブル・カラム作成)

<https://laravel.com/docs/6.x/migrations>

<https://laravel.com/docs/5.5/migrations>

G'sACADEMY Lab10

Chapter7

練習アプリ：本管理アプリを作っていきましょう。

モデルを作成しましょう！

サンプルコード

http://www.venezia-works.com/aws/laravel6_02.zip

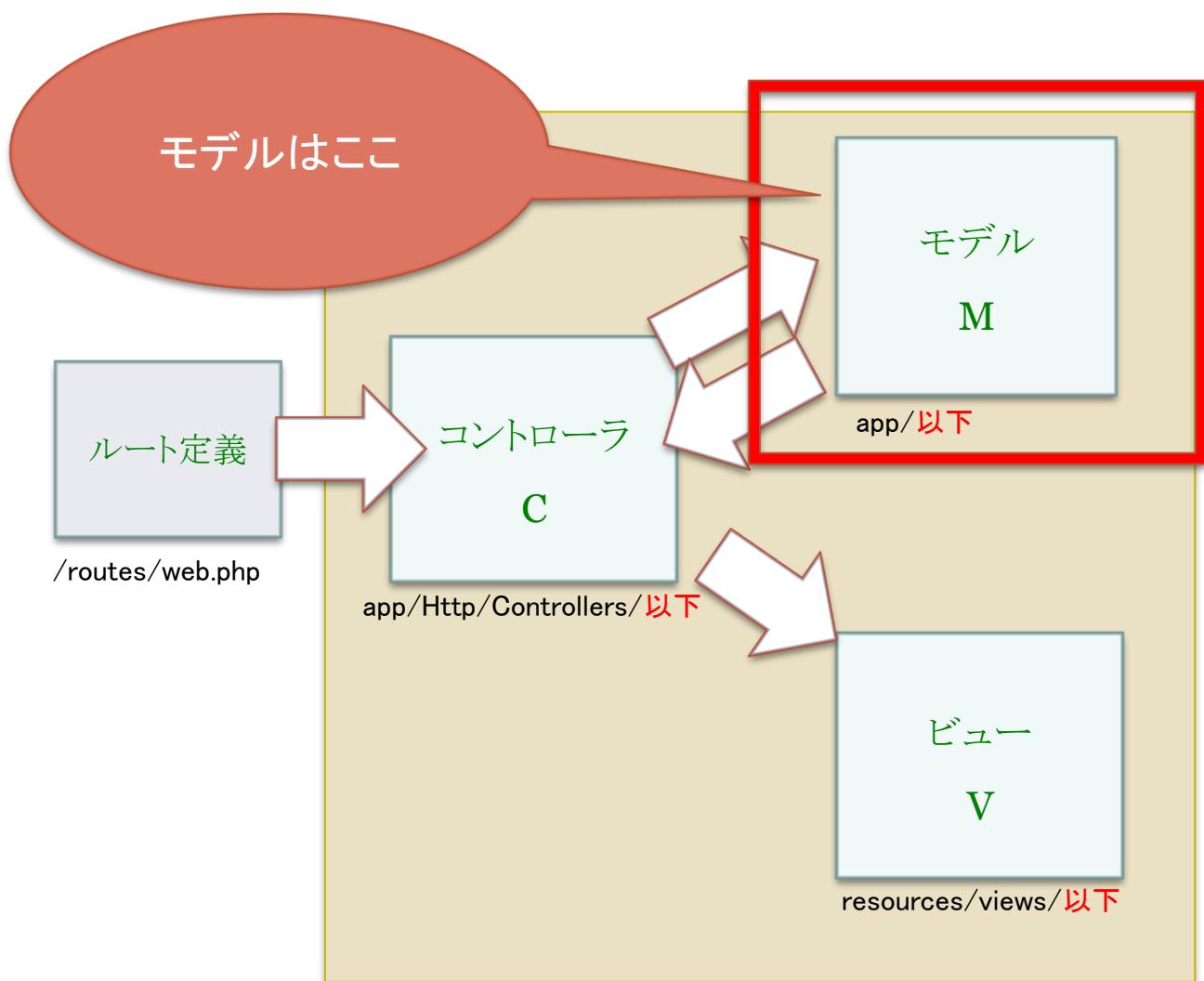
G'sACADEMY Lab10

1. Eloquent モデル（エロクイント モデル）

Eloquent モデルは Laravel のデフォルト ORM(object-relational mapper)です。

ORM とはデータベースのレコードを、オブジェクトとして直感的に扱えるようにし、SQL を記述せず意識することなくプログラムが書けます。

Eloquent モデルは明確に定義された「モデル」を用いることで、簡単にデータベースへのデータ保存／取得をおこなえます。Eloquent モデルは、一つのデータベーステーブルに対応します。



G'sACADEMY Lab10

マイグレーション コマンド書式 :

```
php artisan make:model モデル名
```

練習テーブル「books」テーブルに対応する Book モデルを定義してみましょう。このモデルを生成するために、artisan コマンドを使用します。

2. 練習： Book モデルを作成

```
php artisan make:model Book
```

<解説：命名ルール（一般的な命名ルール）>

テーブル名	books
モデル名	Book

テーブル名は最後に「s」をつける。

モデル名は頭1文字を大文字にし「s」を取る。

作成されるモデル

/app/Book.php

このモデルは「app」ディレクトリに設置されます。

デフォルトでは、このクラスは「空」で作成します。データベーステーブルはモデルの複数形の名前だと想定されているため、Eloquent モデルがどのテーブルに対応するかを明確に宣言する必要はありません。ですから、この場合 Book モデルは books データベースのテーブルと連携していると仮定しましょう。

空のモデルは次のようになっています。

G'sACADEMY Lab10

作成されたモデル

```
/app/Book.php
```

```
<?php  
namespace App;  
use Illuminate\Database\Eloquent\Model;  
  
class Book extends Model  
{  
    //  
}
```

他のモデル作成コマンド書式

モデル作成時にデータベースマイグレーションを生成する方法1

```
php artisan make:model モデル名 -m
```

モデル作成時にデータベースマイグレーションを生成する方法2

```
php artisan make:model モデル名 --migration
```

参考 : laravel.com (Eloquent モデル)

<https://laravel.com/docs/6.x/eloquent>

<https://laravel.com/docs/5.5/eloquent>

G'sACADEMY Lab10

Eloquent モデルの集計関数

<https://laravel.com/docs/6.x/eloquent#retrieving-aggregates>

<https://laravel.com/docs/5.5/eloquent#retrieving-aggregates>

Eloquent モデルのリレーション

<https://laravel.com/docs/6.x/eloquent-relationships>

<https://laravel.com/docs/5.5/eloquent-relationships>

QueryBuilder からクエリを発行

<https://laravel.com/docs/6.x/queries#introduction>

<https://laravel.com/docs/5.5/queries#introduction>

QueryBuilder の集計関数

<https://laravel.com/docs/6.x/queries#aggregates>

<https://laravel.com/docs/5.5/queries#aggregates>

QueryBuilder JOIN

<https://laravel.com/docs/6.x/queries#joins>

<https://laravel.com/docs/5.5/queries#joins>

G'sACADEMY Lab10

Chapter8

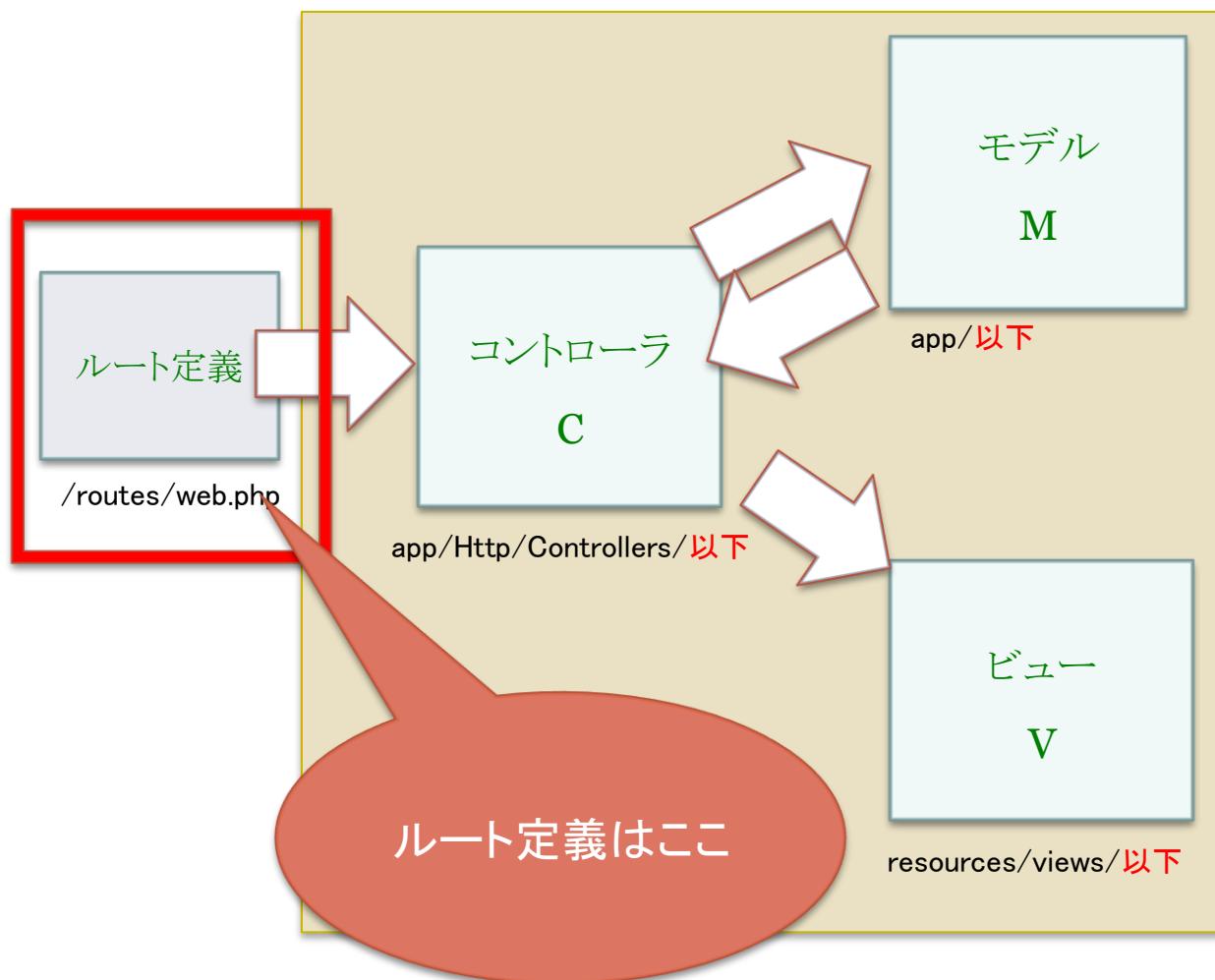
練習アプリ：本管理アプリを作っていきましょう。

ルート定義を作成しましよう

ルート定義

- ルート定義ファイル

/routes/web.php



G'sACADEMY Lab10

ルート定義

ルートとは？

`https://*****.amazonaws.com/`

URL ドメインの最後の '`/`' スラッシュがルートと言います。

ルート定義文字列

例)

`https://*****.amazonaws.com/`

`https://*****.amazonaws.com/home`

`https://*****.amazonaws.com/login`

上記、赤文字の「`/`」と「`/***`」がルート定義で指定できる箇所です。

ルート定義は '`/`' スラッシュ以降の文字を使い、アプリケーション処理と紐付けます。

ルート定義のスケルトンを作成

今回のアプリケーションに必要な 3 つのルート定義を追加していきます。

ルート定義は「ユーザーがページアクセスする URL」と「アプリケーション処理」を紐付かせるための定義となります。

<<今回作成する 3 つのルート定義>>

-
- ・新しい本を追加する処理
 - ・全ての本を一覧表示する処理
 - ・既存の本を削除する処理
-

上記 3 つの処理とユーザーが「ページアクセスする URL」を紐付かせます。

G'sACADEMY Lab10

1. 今回作成するルート定義は、先にスケルトン（処理が空）のルート定義をしていきます。

Laravel のルート定義ファイル

/routes/web.php

```
use App\Book;
use Illuminate\Http\Request;

/**
 * 本のダッシュボード表示
 */
Route::get('/', function () {
    return view("welcome");
});

/**
 * 新「本」を追加
 */
Route::post('/books', function (Request $request) {
    //
});

/**
 * 本を削除
 */
Route::delete('/book/{book}', function (Book $book) {
    //
});
```

G'sACADEMY Lab10

2. 以下は「use」を使って参照する2クラス

```
use App\Book;
```

Eloquent モデル「/app/Book.php」を参照できるようにするため。

```
use Illuminate\Http\Request;
```

HTTP リクエストを扱うための様々なメソッドを参照できるようにします。

参考 : laravel.com (リクエスト)

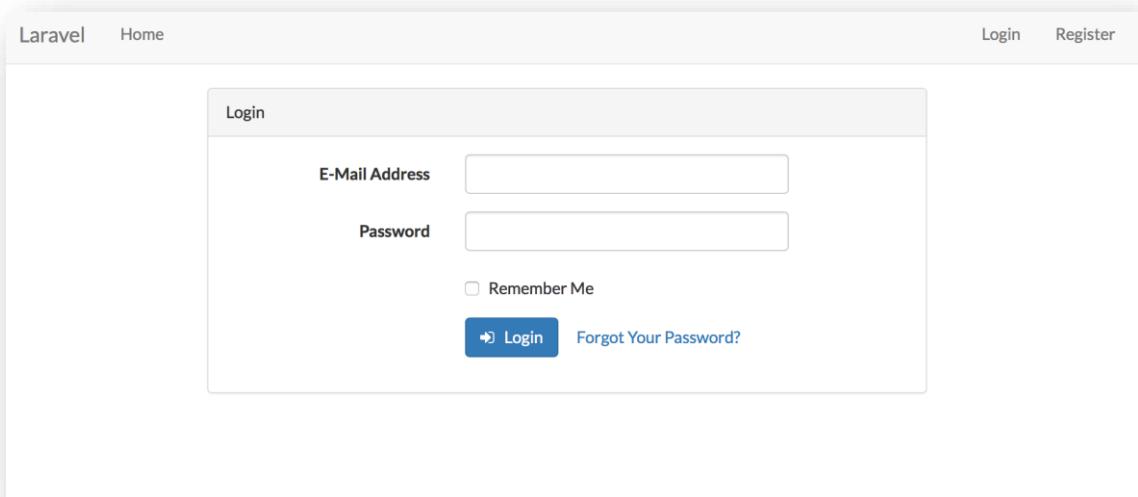
<https://laravel.com/docs/6.x/requests>

<https://laravel.com/docs/5.5/requests>

G'sACADEMY Lab10

Chapter9

練習アプリ ログイン認証機能と JS/CSS を追加



G'sACADEMY Lab10

サインイン・ログイン認証

ユーザーのサインイン・サインアップ機能を「設置」する

ユーザーログイン画面

The screenshot shows a web browser window with a light gray header bar containing the text "Laravel" and "Home" on the left, and "Login" and "Register" on the right. Below the header is a white rectangular form with a thin gray border. The form has a title "Login" at the top. It contains two input fields: one labeled "E-Mail Address" and another labeled "Password". Below these fields is a small checkbox labeled "Remember Me". At the bottom of the form are two buttons: a blue "Login" button with a white arrow icon, and a link "Forgot Your Password?".

ユーザー登録画面

The screenshot shows a web browser window with a light gray header bar containing the text "Laravel" and "Home" on the left, and "Login" and "Register" on the right. Below the header is a white rectangular form with a thin gray border. The form has a title "Register" at the top. It contains four input fields: "Name", "E-Mail Address", "Password", and "Confirm Password". Below these fields is a blue "Register" button with a white person icon.

G'sACADEMY Lab10

1. laravel/ui パッケージをインストール

Laravel6.* の場合（2020年5月時点）

```
composer require laravel/ui 1.*
```

2. artisan コマンドを実行

```
php artisan ui vue --auth
```

3. npm パッケージをインストール

```
npm install
```

4. パッケージをビルド

```
npm run dev
```

5. ルート定義に「2行追加」されています。確認しましょう。

/routes/web.php

```
Auth::routes(); //認証機能を使用する  
Route::get('/home', 'HomeController@index')->name('home');
```

※' /home' のページは、ログイン後に表示可能な初期設定となっています。

G'sACADEMY Lab10

6. ブラウザレビューで確認しましょう。



G'sACADEMY Lab10

Chapter10

練習アプリ：本管理アプリを作っていくましょう。

レイアウトとビューを作成しましょう！

サンプルコード

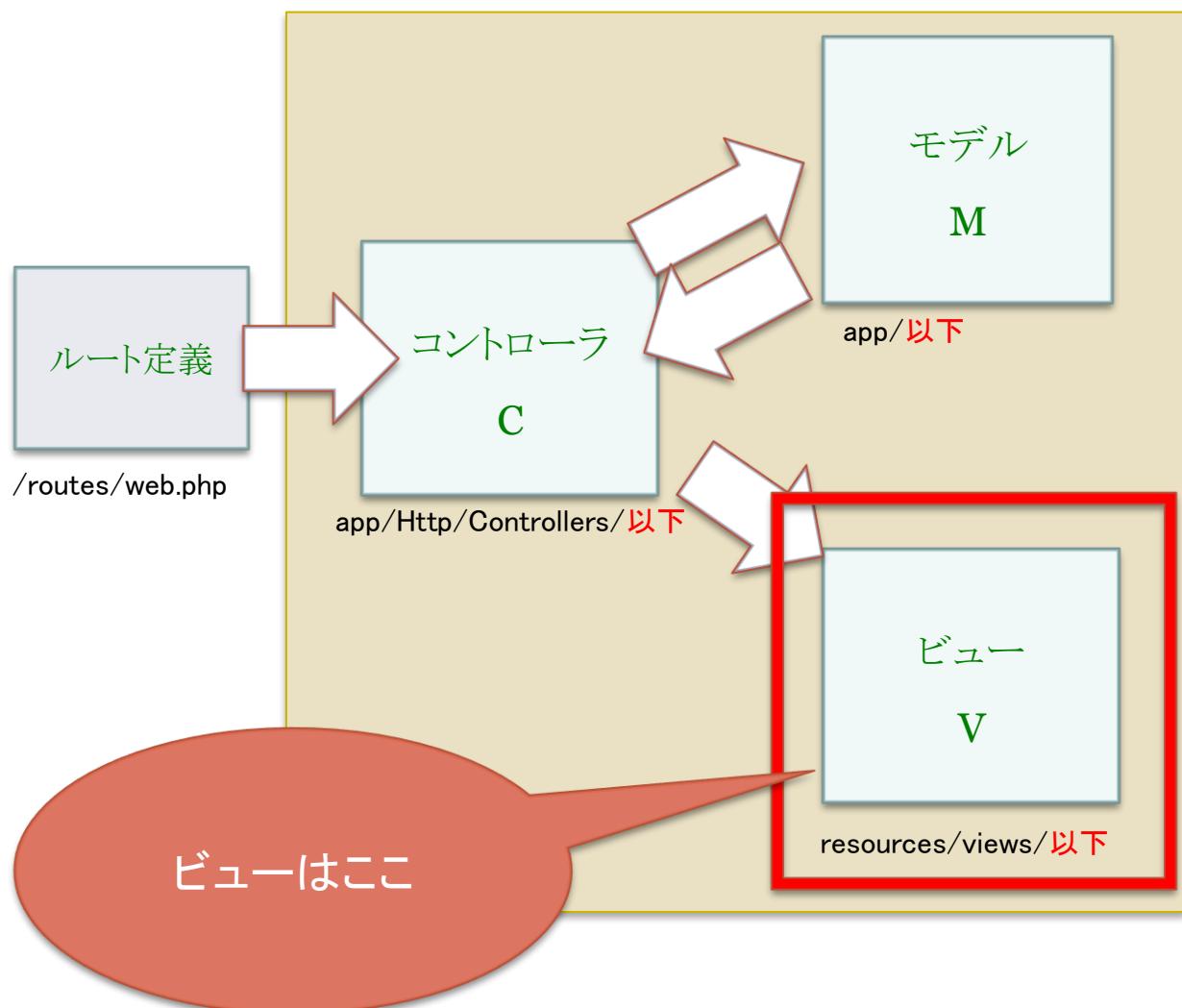
http://www.venezia-works.com/aws/laravel6_03.zip

G'sACADEMY Lab10

ビュー（レイアウト・テンプレート）

ビュー（view） ディレクトリ

resources/views/以下に配置

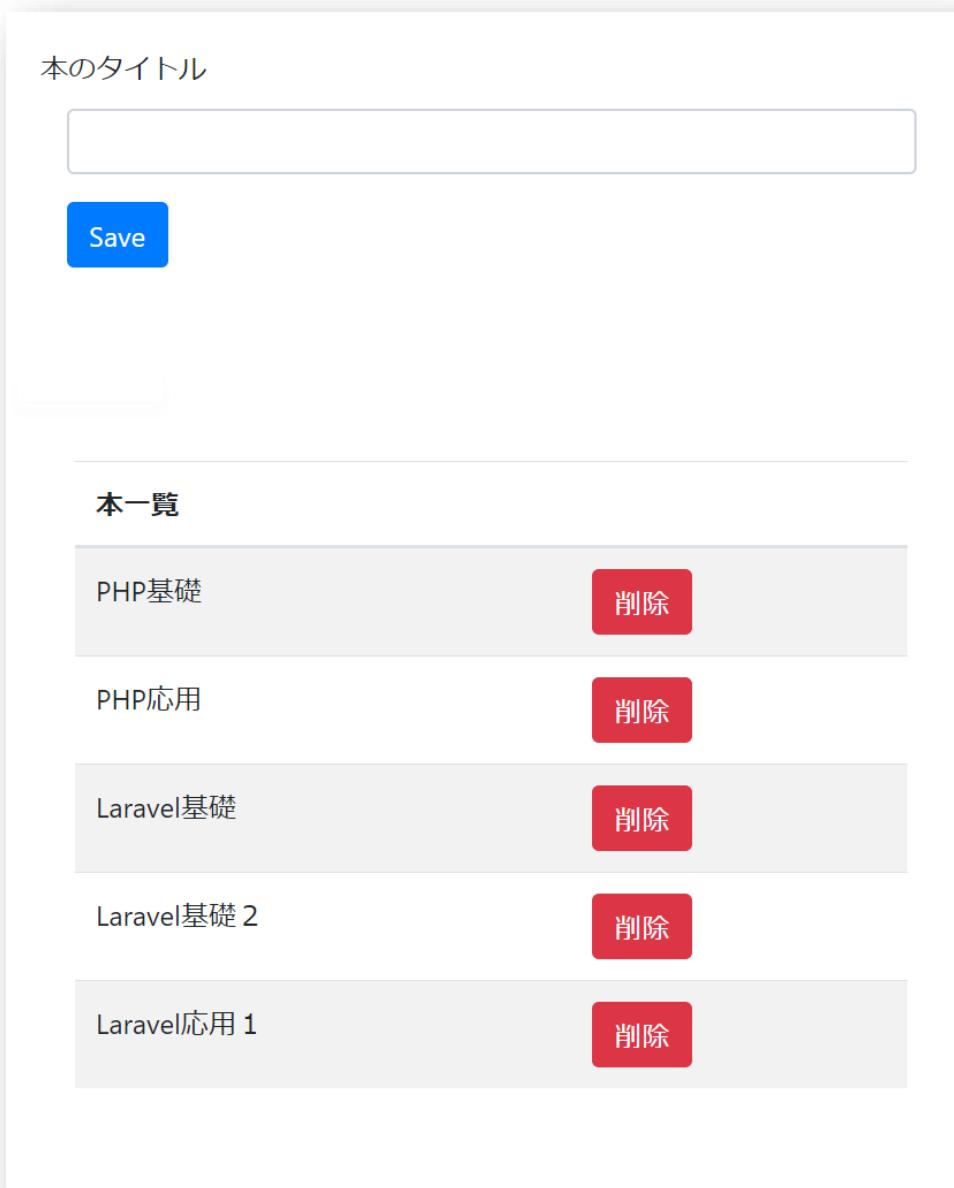


G'sACADEMY Lab10

レイアウトとビューの作成

練習で作成するアプリケーションは「本（書籍）」を追加するためのフォームを設置し、同時に現在の「本（書籍）」を一覧表示するビューを作成します。 Laravel では最初から Bootstrap が使用可能です、デザインレイアウトは Bootstrap の CSS スタイルを使います。

以下、画面作成の例です。



G'sACADEMY Lab10

1. テンプレートの定義

Laravel ではテンプレートエンジン「Blade テンプレート」を使用しています。

Web アプリケーションの全ページのレイアウトは、「ヘッダー、フッター、メニュー」など、ほとんど同じレイアウトになります。そのため「ヘッダー、フッター、メニュー」をパート化し、そのパートを各ページで読み込み、共有利用すると効率良く作成できます。そうすることで、パートを修正するだけで他のページの「ヘッダー、フッター、メニュー」も同時に更新できるようになります。これは「運用開始」後のページ変更や更新にとても便利です。パート化せずに使うことも可能です。

Blade テンプレートはベースのテンプレートとパートを組み合わせ表示し、テンプレート上で if 分岐表示することも可能です。そのような制御構文も用意してあり、高機能なテンプレートです。

「*****.blade.php」拡張子

ビュー表示するときにフレームワークへ「Blade テンプレートエンジン」を使用して表示することを知らせる拡張子です。必ずファイル名の後に「.blade.php」をつけます。これでフレームワークがテンプレートだと認識するようになります。

ビュー 設置箇所

Laravel の全てのビューは、「resources/views」配下に設置します。ですから新しいレイアウトビューも「resources/views」の中に新しく「layouts」フォルダを作成し、その中に「app.blade.php」を作成します。階層を含めたファイルで表すと、

/resources/views/layouts/app.blade.php

になります。

G'sACADEMY Lab10

2. テンプレート・レイアウトの関係

テンプレートは、親と子に別けられます(分けなくてもよい)。

- 最初に指定して呼び出されるのが「子テンプレート」
- ***.blade.php から読み込むテンプレートは「親テンプレート」

① コントローラ : /routes/web.php

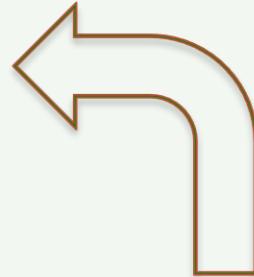
```
Route::get('/', function () {
    return view('books');
});
```

②

/resources/views/books.blade.php

【パート・テンプレート】

```
@extends('layouts.app')
@section('title')
    トップリスト
@endsection
```



③ /resources/views/layouts/app.blade.php

【ベース・テンプレート】

```
<!DOCTYPE html>
<html lang="ja">
    <head>
        <meta charset="utf-8">
        <title>@yield('title')</title>
```

G'sACADEMY Lab10

/resources/views/layouts/app.blade.php を開きます。

```
<!-- resources/views/layouts/app.blade.php -->
<!DOCTYPE html>
<html lang="ja">
  <head>
    ...
  </head>
  <body>
    <div class="container">
      <nav class="navbar navbar-default">
        ...
      </nav>
    </div>
    @yield('content')
  </body>
</html>
```

レイアウトの@yield('content') の部分がパート配置キーワード（Blade ディレクティブ）です。

共有パートの「子テンプレート」を「親テンプレート(app.blade.php)」へ配置指定するために '@yield('*****')' を使います。

次に、このレイアウトを使用しメインコンテンツを表示する、子のビューを定義しましょう。

G'sACADEMY Lab10

3. 子テンプレートの定義（パート）

新しい「本」を登録するためのフォーム表示と、既に登録してある「本一覧」表示のビューを作成しましょう。

「resources/views/books.blade.php」を作成しましょう。

```
<!-- resources/views/books.blade.php -->
@extends('layouts.app')
@section('content')

<!-- Bootstrap の定形コード… -->
<div class="card-body">
    <div class="card-title">
        本のタイトル
    </div>

    <!-- バリデーションエラーの表示に使用-->
    @include('common.errors')
    <!-- バリデーションエラーの表示に使用-->

    <!-- 本登録フォーム -->
    <form action="{{ url('books') }}" method="POST" class="form-
horizontal">
        {{ csrf_field() }}

        <!-- 本のタイトル -->
        <div class="form-group">
            <div class="col-sm-6">
                <input type="text" name="item_name" class="form-control">
            </div>
        </div>
    </form>
```

G'sACADEMY Lab10

```
<!-- 本 登録ボタン -->
<div class="form-group">
    <div class="col-sm-offset-3 col-sm-6">
        <button type="submit" class="btn btn-primary">
            Save
        </button>
    </div>
</div>
</form>
</div>
<!-- Book: 既に登録されてる本のリスト -->
```

@endsection

先に進む前に、上記「子テンプレート」について次のページで説明します。

G'sACADEMY Lab10

4. @extends ディレクティブ（親テンプレート読み込み）

`@extends('layouts.app')` で親テンプレートを読み込みます。
layouts.app は layouts/app と同意で、"."は"/"に解釈され、
layouts/app.blade.php を読み込むという意味です。

※ 親テンプレート : resources/views/layouts/app.blade.php

5. @section ディレクティブ（名前をつけて一括りにする）

`@section('content')` から `@endsection` の間の文字/HTMLなどを content に括る。
親テンプレート `@yield('content')` の箇所に content を挿入します。

※ 今回は content と命名していますが、任意で名前をつけられます。

6. @include ディレクティブ

`@include` は PHP で使用する include と同じ使い方ができます。
`@include('ディレクトリ名.ファイル名')` を記述した箇所に挿入します。

今回は、フォーム・バリデーションエラーの表示テンプレートとして読み込みに使用しています。

`@include('common.errors')` は、 common/errors.blade.php を読み込みます。
※ `@include('common.errors')` を全ページに記述することで簡単に配置できます。

common/errors.blade.php は次に作ります。

G'sACADEMY Lab10

7. エラー表示テンプレートを作りましょう

resources/views/common/errors.blade.php

```
<!-- resources/views/common/errors.blade.php -->

@if (count($errors) > 0)
    <!-- Form Error List -->
    <div class="alert alert-danger">
        <div><strong>入力した文字を修正してください。</strong></div>
        <div>
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
    </div>
@endif
```

注意：\$errors 変数は全ての Laravel テンプレートビューの中で参照できます。

これでテンプレートのレイアウトビューが作成できました。

G'sACADEMY Lab10

8. ルート定義に view ヘルパー関数を定義します。

```
/routes/web.php  
ルート定義 (books.blade.php を表示)
```

```
Route::get('/', function () {  
    return view('books');  
});
```

view 関数に引数として渡した「 books 」がファイル名です。
テンプレート「books.blade.php」を表示するという指定です。

9. ビューの確認 (Web サーバーを起動させましょう)

```
php artisan serve --port=8080
```

※cms フォルダ内で上記コマンドを実行

```
※MySQL が起動していない場合 「 sudo service mysqld start 」
```

ブラウザで表示確認しましょう。

作成したシンプルなテンプレートの表示ができるようになりました。

The screenshot shows a web browser displaying a Laravel application. The header has 'Laravel' on the left and 'Login Register' on the right. Below the header is a form with a single input field labeled '本のタイトル' (Book Title) and a blue 'Save' button below it.

登録フォームを作成しました。まだ登録処理を作っていません。

Chapter10 で登録処理を作成しますので、このまま進んでください。

G'sACADEMY Lab10

解説: Blade テンプレート

Blade は「シンプルでルールが少ない Laravel テンプレートエンジン」です。テンプレートの中には、PHP を直接記述することができ、他のテンプレートではルールが多く自由度を抑えた仕様となっているため、Blade テンプレートと他のテンプレートエンジンでは違いがあります。Blade テンプレートは、一度 PHP へコンパイルされます。変更があるまでキャッシュに持つことでアプリケーションオーバーヘッドは基本的に 0 です。

・変数表示

テンプレート内では

```
<?php echo $変数名; ?>
```

ではなく、

```
{{ $変数名 }}
```

で記述することができます。

・Blade テンプレートでの変数表示

{{ \$変数名 }}	エスケープ有り
{{!! \$変数名 !!}}	エスケープ無し

Blade テンプレートは {{ \$変数名 }} を使用した場合、XSS 攻撃を防ぐために、PHP のエスケープ関数を自動的に通して表示します。理由がありエスケープしたくない場合には {{!! \$変数名 !!}} を使用することができます。しかし、セキュリティを考慮すると基本は {{変数名}} で使っていきましょう。

G'sACADEMY Lab10

変数のデータ表示（変数表示の切り分け）

```
{{ $name or 'Default' }}
```

この例の場合、\$name 変数が存在すれば値が表示されます。しかし存在していないければ Default という言葉が表示されます。

・Blade と JavaScript フレームワーク

Blade テンプレートで {{変数名}} を使用していますが、JavaScript フレームワークでも {{変数名}} を利用する JavaScript フレームワークがあります。その場合の対処方法として、「@」を {{変数}} の頭につけると「@」の後ろが変数として処理せずに {{変数名}} を残してくれます。つまりは JavaScript の変数としてその記述を変換せずに処理します。以下例文を参照。

```
<h1>Laravel</h1>
```

こんにちは @{{ name }}.

参考 : laravel.com (Blade テンプレート)

<https://laravel.com/docs/6.x/blade>

<https://laravel.com/docs/5.5/blade>

参考 : laravel.com (ビュー)

<https://laravel.com/docs/6.x/views>

<https://laravel.com/docs/5.5/views>

G'sACADEMY Lab10

Chapter11

練習アプリ：本管理アプリを作っていきましょう。

本の追加登録処理を作成しましょう

サンプルコード

http://www.venezia-works.com/aws/laravel6_04.zip

本の追加登録処理を作成

これでビュー・テンプレートにフォームが用意できました。

1. バリデーションを作成（入力チェック）

フォームに入力された値が、正常な値かを確認（バリデーション）します。
新しい「本」を登録する。ルーティングの記述で、入力バリデーション処理を記述します。

「POST /books」ルートのコードを/routes/web.phpへ追加しましょう

```
Route::post('/books', function (Request $request) {
```

```
    //バリデーション
    $validator = Validator::make($request->all(), [
        'item_name' => 'required|max:255',
    ]);
```

追加

```
    //バリデーション: エラー
    if ($validator->fails()) {
        return redirect('/')
            ->withInput()
            ->withErrors($validator);
    }
```

```
    // 本を作成処理...
});
```

G'sACADEMY Lab10

2. バリデーション設定

◇ item_name (必須) 255 文字以内

※ 「 | 」パイプを間に挟むことでバリデーション条件の追加が可能です。例えば「 required|min:3|max:255」のように「min:3」最小値3と設定も可能です。

◇ バリデーション失敗の場合 (if(\$validator->fails()){...})

セッションに値を保存し、ユーザーを「 / 」ルート URL ヘリダイレクトする。

```
redirect('/') //「/」ルートヘリダイレクト  
withInput() //セッションに値を保存
```

前ページからの入力値とエラーをセッションへフラッシュデータとして保存。フラッシュデータとしてセッション変数に入力した値を保存することで、バリデーションエラー時に「ユーザーが入力した値」を消すことなく、再表示できます。方法は簡単、入力フォームの

```
<input name="name名" ... value="{{ old('name名') }}" ...>
```

のように value 属性に {{ old('name 属性値') }} を埋め込むだけです。

◇ \$errors 変数

この例の->withErrors(\$validator) の部分について解説しましょう。-

>withErrors(\$validator) の呼び出しは、指定されたバリデータ (validator) インスタンスの「エラー」をフラッシュデータとしてセッションへ保存し、ビューの中で「\$errors」変数としてアクセスできるようにします。
フォームのバリデーションエラーを表示するため、今回は @include('common.errors') ディレクティブを使い、「errors.blade.php」テンプレートを読み込み使用します。

参考 : laravel.com (リクエスト)

<https://laravel.com/docs/6.x/requests>

参考 : laravel.com (バリデーション)

<https://laravel.com/docs/6.x/validation#available-validation-rules>

G'sACADEMY Lab10

3. 本の登録処理を作成

入力のバリデーションは作成できました。新しい「本」を登録するためにルート処理の定義を続けていきましょう。新しい「本」を登録するには、Eloquent モデル「Book」に対し、値を代入した後に「save」メソッドで保存します。処理が終わったら `return redirect('/');` でルートに遷移します。

/routes/web.php

```
Route::post('/books', function (Request $request) {
    //バリデーション
    $validator = Validator::make($request->all(), [
        'item_name' => 'required |min:3 | max:255',
    ]);
    //バリデーション: エラー
    if ($validator->fails()) {
        return redirect('/')
            ->withInput()
            ->withErrors($validator);
    }
    // Eloquent モデル
    $books = new Book;
    $books->item_name = $request->item_name;
    $books->item_number = '1';
    $books->item_amount = '1000';
    $books->published = '2017-03-07 00:00:00';
    $books->save(); //「/」ルートにリダイレクト
    return redirect('/');
});
```

追加

これで本を登録できるようになりました。次に本一覧を表示する機能を追加しましょう。

G'sACADEMY Lab10

Chapter 1 2

練習アプリ：本管理アプリを作っていきましょう。

本の一覧表示を作成しましょう

サンプルコード

http://www.venezia-works.com/aws/laravel6_05.zip

G'sACADEMY Lab10

登録している「本」の一覧表示を作成

1. 本の一覧データを取得し、view へ変数を渡す

最初に「 / 」ルート定義内を編集し、既存の「本」全てをビューに渡します。

view 関数は第 2 引数に、ビューで使用するデータを配列で受け付けます。配列のキーはビューの中で変数となります。

/routes/web.php (既存コード編集)

```
Route::get('/', function () {
    $books = Book::orderBy('created_at', 'asc')->get();
    return view('books', [
        'books' => $books
    ]);
});
```

view 関数に配列データを渡したら、「books.blade.php」ビューの中で反復処理をおこない HTML テーブル要素を作成して表示します。

<解説>view 側での処理

@foreach は \$books 変数内のデータレコード数だけ処理を繰り返します。

\$books はコントローラ 'books' から渡された変数を保持しています。

```
@foreach ($books as $book)
    <div>{{ $book->item_name }}</div>
    <div>{{ $book->item_amount }}</div>
@endforeach
```

※ \$book は foreach 文中で使用する変数です。

G'sACADEMY Lab10

2. @foreach を使い \$books の値を \$book へ代入して表示

resources/views/books.blade.php (</form> の下行に以下コードを追記)

```
<!-- 現在の本 -->
@if (count($books) > 0)
<div class="card-body">
    <div class="card-title">
        現在の本
    </div>
    <div class="card-body">
        <table class="table table-striped task-table">
            <!-- テーブルヘッダ -->
            <thead>
                <th>本一覧</th>
                <th>&nbsp;</th>
            </thead>
            <!-- テーブル本体 -->
            <tbody>
                @foreach ($books as $book)
                    <tr>
                        <!-- 本タイトル -->
                        <td class="table-text">
                            <div> {{ $book->item_name }} </div>
                        </td>
                        <!-- 本: 削除ボタン -->
                        <td>
                            </td>
                    </tr>
                @endforeach
            </tbody>
        </table>
    </div>
</div>
```

G'sACADEMY Lab10

```
</tbody>
</table>
</div>
</div>
@endif
<!-- Book: 既に登録されてる本のリスト -->
```

「本」アプリケーションは、ほぼ完成です。

いらなくなつた本を削除する機能がないので次に作成します。

G'sACADEMY Lab10

解説 : Blade テンプレートの制御構文

Blade テンプレートは@extends、@section、@include 以外の制御構文を用意しています。@を頭に付けた短縮形の記述で波括弧 {...} を使用しません。そのため、制御構文はシンプルで視認性が良いのが特徴です。

分岐処理

```
@if (count($books) === 1)  
    本が一つある！  
@endif
```

多分岐処理

```
@if (count($books) === 1)  
    本が一つある！  
@elseif (count($books) > 1)  
    本が2冊以上ある！  
@else  
    本が全くない！  
@endif
```

反復処理（インデックス）

```
@for ($i=0; $i<10; $i++)  
    現在の COUNT : {{ $i }}  
@endfor
```

G'sACADEMY Lab10

反復処理（配列||連想配列||オブジェクト）

```
@foreach ($users as $user)
    <p>ユーザー名 {{ $user->name }} </p>
@endforeach
```

反復処理（配列||連想配列||オブジェクト）

```
@forelse ($users as $user)
    <li>ユーザー名 {{ $user->name }}</li>
@empty
    <li>ユーザーなし</li>
@endforelse
```

条件式が偽(False)の場合の処理

```
@unless (Auth::check())
    ログインしていません。
@endunless
```

条件式が偽(False||True)の場合の処理

```
@unless (Auth::check())
    ログインしていません。
@else
    ログインしています。
@endunless
```

参考 : laravel.com (制御構文)

<https://laravel.com/docs/6.x/blade#control-structures>

G'sACADEMY Lab10

Chapter13

練習アプリ：本管理アプリを作っていきましょう。

本の削除機能を作成しましょう！

サンプルコード

http://www.venezia-works.com/aws/laravel6_06.zip

本の削除機能を作成

1. 削除ボタンを追加

コード中、削除ボタンを設置する場所に”`<!-- 本: 削除ボタン -->`”を残してあります。

「books.blade.php」ビューの「本一覧」の表示リストに、`<削除ボタン>`を追加しましょう。ボタンがクリックされると、`DELETE /books` リクエストがアプリケーションに送信されます。

```
<tr>
    <!-- 本タイトル -->
    <td class="table-text">
        <div>{{ $book->item_name }}</div>
    </td>

    <!-- 本: 削除ボタン -->
    <td>
        <form action="{{ url('book/' . $book->id) }}" method="POST">
            {{ csrf_field() }}
            {{ method_field('DELETE') }}

            <button type="submit" class="btn btn-danger">
                削除
            </button>
        </form>
    </td>
</tr>
```

```
        <button type="submit" class="btn btn-danger">
            削除
        </button>
    </form>
```

追加

G'sACADEMY Lab10

解説：擬似的メソッド

```
 {{ method_field('DELETE') }}
```

削除ボタンフォームの method が POST を使用しているのにかかわらず、定義しているルートは Route::delete である点に注視してください。

HTML フォームは GET と POST のみが許可されています。そのため、フォームの DELETE リクエストを使う場合は、擬似的に使える方法があります。

フォームの中で method_field('DELETE') 関数の結果を出力することにより、DELETE リクエストへ見せかけることができます。

Laravel はこれを認識し、実際の HTTP リクエストメソッドをオーバーライドします。生成されるフィールドは次の内容です。

```
<input type="hidden" name="_method" value="DELETE">
```

{{ method_field('DELETE') }} を使うだけで使えるようになる。覚えておきましょう。ルート定義側は「 Route::delete 」の記述で紐付けられます。

解説：CSRF からの保護

```
 {{ csrf_field() }}
```

「CSRF」はユーザーになり代わり悪いことをするハッキング手法の一つです。 Laravel には「CSRF」からアプリケーションを簡単に守るためのヘルパー関数が用意されています。基本的にはログイン認証後などのフォームには必須の記述のため「常に記述する」と覚えておけばいいでしょう。

※削除処理が表示されている画面を「ユーザーになり代わられたら...」、勝手に削除されてしまいますが、学習（練習）のため記述しておきましょう。

参考：laravel.com（擬似的メソッド）

<https://laravel.com/docs/5.5/routing#form-method-spoofing>

<https://laravel.com/docs/6.x/routing#form-method-spoofing>

G'sACADEMY Lab10

2. 削除処理を追加

クリックした本を削除するロジックをルートへ追加しましょう。

{book} ルートパラメータに対応する Book モデルを自動的に取得するため、暗黙のモデル結合が使えます。

レコードを削除するために `delete` メソッドを使います。

/routes/web.php (既存コードを編集)

```
Route::delete('/book/{book}', function (Book $book) {
    $book->delete();
    return redirect('/');
});
```

レコードが削除された後は、ユーザーを「 / 」ルートへ、リダイレクトします。

参考 : laravel.com (モデル結合ルート)

<https://laravel.com/docs/5.5/routing#route-model-binding>

<https://laravel.com/docs/6.x/routing#route-model-binding>

課題 1-3

G'sACADEMY Lab10

課題 1

入力フォームを増やす & データ登録

item_name	本の名前 //ここは既にできています。以下の 3 項目を追加。
item_number	何冊
item_amount	金額
published	本公開日時 //（年月日時分秒 or 年月日）

バリデーションの設定

item_name	(必須) 最小 3 文字～255 文字以内
item_number	(必須) 最小 1 文字～3 文字以内
item_amount	(必須) 6 文字以内
published	(必須) ※年月日だけでも OK

※練習のための課題なので、自分で変更してもかまいません。

自分で考えながら打って確かめることが大事です。

課題 2

更新画面 & 更新処理も調べて作ってみましょう。

※わからない場合でも解答例があるので安心してください。

課題 *

同じアプリを複数作ってフレームワークに慣れましょう。

本書では「本」を管理するアプリを作成しましたが、

別のアプリを 2～3 個作りフレームワークに慣れましょう。

※フレームワークは「枠組み」がカッチリと決まっているので、カッチリしたルールに慣れることができがスキルアップの早道です。

参考 : laravel.com

<https://laravel.com/docs/5.5>

課題解説 1 – 2

G'sACADEMY Lab10

画面完成例：view サンプル・ダウンロード

<http://www.venezia-works.com/aws/kadai1.zip>

解答例のサンプルコードをダウンロードし、参考にして進めてください。

課題 1 と 2：画面完成例

画面完成例)PC 画面

The screenshot displays two main sections of a web application:

登録項目 (Registration Item)

This section contains fields for inputting item details:

- 本のタイトル: Book
- 金額: (empty input field)
- 数: (empty input field)
- 公開日: 年 /月 /日 (empty input field)

A blue "Save" button is located at the bottom left of this section.

更新・削除 (Update/Delete)

This section shows a list of items with their details and update/delete buttons:

本一覧	更新	削除
PHP基礎	更新	削除
PHP応用	更新	削除
Laravel基礎	更新	削除
Laravel基礎 2	更新	削除
Laravel応用 1	更新	削除

G'sACADEMY Lab10

課題解説1：ルート定義とビュー作成

解答例のサンプルコードを参考にしてください。

1. ルート定義に更新画面のタブを追加

1.1 「画面を表示する URL」を先に決めましょう。

/booksedit/{books}

上記アドレスでルート定義します。

1.2 ルート定義のタブを作成

//更新画面
Route::post('/booksedit/{books}', function(Book \$books) {
 // {books} id 値を取得 => Book \$books id 値の1レコード取得
});

1.3 上記ルーティングの解説

/booksedit/{books} の /booksedit/ はアドレスです。
{books} は id 値パラメータを取得します。
「 Book \$books 」は Book モデルを指定して引数に「 \$books 」を指定することで、id 値が等しい「1レコードを取得」します。

G'sACADEMY Lab10

2. スタブに表示処理を追加

2.1 view ヘルパー関数の解説

このルーティング処理では、

```
return view('読み込むビューの指定', [ビューに渡す値]);
```

上記コードでは、「ビューの選択」と「変数をビューに渡す」記述となります。

ビューで使用する「変数」の渡し方の記述は以下になります。

記述

```
[ 'book' => $books ]
```

解説

```
[ 'ビュー側使用する変数名' => 値 (変数/配列/オブジェクト) ]
```

2.2 ルート定義に view ヘルパー関数を記述

```
//更新画面
```

```
Route::post('/booksedit/{books}', function(Book $books) {
    //{$books}id 値を取得 => Book $books id 値の1レコード取得
    return view('booksedit', ['book' => $books]);
});
```

今回のルーティング定義では引数に「Book \$books」を指定しているため
「id 値が等しい1レコードを取得した状態」を保持している\$booksをビュー
に渡します。ビュー側のテンプレートでは「\$book」に値が保存されるので
\$book->id
\$book->item_name
など「\$book->プロパティ名」の使い方でテンプレートでは使用可能です。

G'sACADEMY Lab10

3. 3つの「入力項目」追加（ビューを変更）

「`books.blade.php`」ビューファイルに記述追記します。

以下コードは以前の入力項目を含んで変更した完成内容です。

`/resources/views/books.blade.php`

```
....  
><!-- 本のタイトル --&gt;<br/><form action="{{ url('books') }}" method="POST" class="form-horizontal">  
    {{ csrf_field() }}  
  
    <div class="form-row" style="border: 1px solid red; padding: 10px; margin-bottom: 10px;">  
        <div class="form-group col-md-6" style="border: 1px solid red; padding: 5px; margin-bottom: 5px;">  
            <label for="book" class="col-sm-3 control-label" style="border: 1px solid red; padding: 2px;">Book            <input type="text" name="item_name" class="form-control" style="border: 1px solid red; width: 100%; height: 30px; border-radius: 5px; padding: 5px; margin-top: 5px;">  
        </div>  
  
        <div class="form-group col-md-6" style="border: 1px solid red; padding: 5px; margin-bottom: 5px;">  
            <label for="amount" class="col-sm-3 control-label" style="border: 1px solid red; padding: 2px;">金額            <input type="text" name="item_amount" class="form-control" style="border: 1px solid red; width: 100%; height: 30px; border-radius: 5px; padding: 5px; margin-top: 5px;">  
        </div>  
    </div>  
    <div class="form-row" style="margin-bottom: 10px;">  
        <div class="form-group col-md-6" style="border: 1px solid red; padding: 5px; margin-bottom: 5px;">  
            <label for="number" class="col-sm-3 control-label" style="border: 1px solid red; padding: 2px;">数            <input type="text" name="item_number" class="form-control" style="border: 1px solid red; width: 100%; height: 30px; border-radius: 5px; padding: 5px; margin-top: 5px;">  
        </div>  
  
        <div class="form-group col-md-6" style="border: 1px solid red; padding: 5px; margin-bottom: 5px;">  
            <label for="published" class="col-sm-3 control-label" style="border: 1px solid red; padding: 2px;">公開日            <input type="date" name="published" class="form-control" style="border: 1px solid red; width: 100%; height: 30px; border-radius: 5px; padding: 5px; margin-top: 5px;">  
        </div>  
    </div>  
</div>
```

G'sACADEMY Lab10

```
<!-- 本 登録ボタン -->                                form 内を変更
<div class="form-row">
    <div class="col-sm-offset-3 col-sm-6">
        <button type="submit" class="btn btn-primary">
            Save
        </button>
    </div>
</div>

</form>
```

表示確認すると以下のように表示されます。

本のタイトル
Book

金額

数

公開日
年/月/日

Save

本一覧

PHP基礎	削除
PHP応用	削除
Laravel基礎	削除
Laravel基礎 2	削除
Laravel応用 1	削除

G'sACADEMY Lab10

4. 更新画面へのリンクを追加（ビューを変更）

「本のタイトル」と「削除ボタン」のコード記述の間くらいに「更新ボタン」のコードを記述するといいでしよう。

/resources/views/books.blade.php

```
...  
<!-- 本タイトル -->  
<td class="table-text">  
    <div>{{ $book->item_name }}</div>  
</td>
```

```
<!-- 本: 更新ボタン -->  
<td>  
    <form action="{{ url('booksedit/' . $book->id) }}" method="POST">  
        {{ csrf_field() }}  
        <button type="submit" class="btn btn-primary">  
            更新  
        </button>  
    </form>  
</td>
```

追加

```
<!-- 本: 削除ボタン -->  
...  
-----
```

ビューURL 定義：

```
    {{ url('booksedit/' . $book->id) }}
```

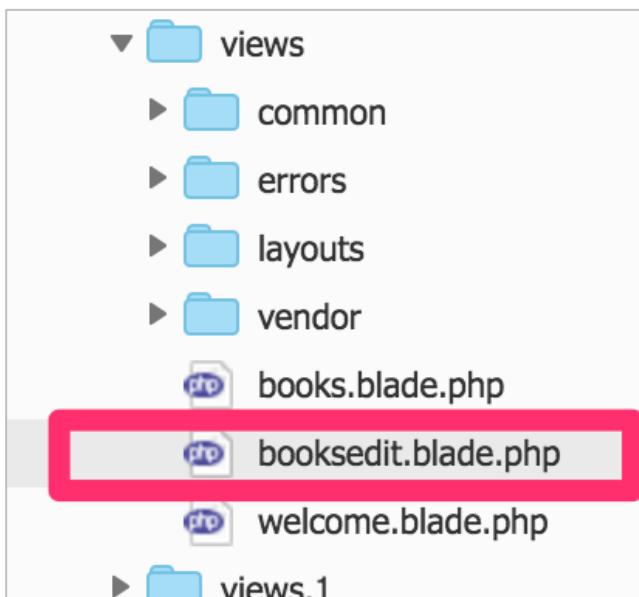
url 関数内で「booksedit/」の URL 文字列に id 値のパラメータ「\$book->id」を付与することで、ルーティング定義「Route::post('/booksedit/{books}」との関係を作成します。

G'sACADEMY Lab10

5. 更新画面のビューを新規作成

views フォルダの中に「booksedit.blade.php」を新規で作成。

/resources/views/booksedit.blade.php



「booksedit.blade.php」のコードを記述。

resources/views/booksedit.blade.php (新規追加)

```
@extends('layouts.app')

@section('content')
<div class="row">
    <div class="col-md-12">
        @include('common.errors')
        <form action="{{ url('books/update') }}" method="POST">

            <!-- item_name -->
            <div class="form-group">
                <label for="item_name">Title</label>
```

G'sACADEMY Lab10

```
<input type="text" name="item_name" class="form-control"
value="{{ $book->item_name}}">
</div>
<!-- item_name -->

<!-- item_number -->
<div class="form-group">
    <label for="item_number">Number</label>
    <input type="text" name="item_number" class="form-control"
value="{{ $book->item_number}}">
</div>
<!-- item_number -->

<!-- item_amount -->
<div class="form-group">
    <label for="item_amount">Amount</label>
    <input type="text" name="item_amount" class="form-control"
value="{{ $book->item_amount}}">
</div>
<!-- item_amount -->

<!-- published -->
<div class="form-group">
    <label for="published">published</label>
    <input type="datetime" name="published" class="form-control"
value="{{ $book->published}}"/>
</div>
<!-- published -->

<!-- Save ボタン/Back ボタン -->
<div class="well well-sm">
    <button type="submit" class="btn btn-primary">Save</button>
```

G'sACADEMY Lab10

```
<a class="btn btn-link pull-right" href="{{ url('/')}}">
    Back
</a>
</div>
<!-- Save ボタン/Back ボタン -->

<!-- id 値を送信 -->
<input type="hidden" name="id" value="{{ $book->id}}" />
<!--/ id 値を送信 -->

<!-- CSRF -->
{{ csrf_field() }}
<!--/ CSRF -->

</form>
</div>
</div>
@endsection
```

G'sACADEMY Lab10

6. 表示と動作を確認

以下、表示確認

- ・入力項目：4項目が表示されているか確認
- ・更新ボタン：更新ボタンが表示されているか確認

本のタイトル		登録項目
Book	金額	
<input type="text"/>	<input type="text"/>	
数	公開日	
<input type="text"/>	<input type="text"/>	
<input type="button" value="Save"/>		

更新・削除

本一覧		
PHP基礎	<input type="button" value="更新"/>	<input type="button" value="削除"/>
PHP応用	<input type="button" value="更新"/>	<input type="button" value="削除"/>
Laravel基礎	<input type="button" value="更新"/>	<input type="button" value="削除"/>
Laravel基礎 2	<input type="button" value="更新"/>	<input type="button" value="削除"/>
Laravel応用 1	<input type="button" value="更新"/>	<input type="button" value="削除"/>

課題解説 1-2：登録と更新の処理作成

1. 課題のバリデーションを定義

/routes/web.php (既存コードに3行追加)

```
=====
/*
 * 新本 追加
 */
Route::post('/books', function (Request $request) {
    //バリデーション
    $validator = Validator::make($request->all(), [
        'item_name' => 'required | min:3 | max:255',
        'item_number' => 'required | min:1 | max:3',           追加
        'item_amount' => 'required | max:6',
        'published' => 'required',
    ]);
    ....
```

<上記、「バリデーション」に以下3行のコードを追加>

```
'item_number' => 'required | min:1 | max:3',
'item_amount' => 'required | max:6',
'published' => 'required',
```

G'sACADEMY Lab10

2. 追加した入力項目を登録プロパティに追加

/routes/web.php (既存コードに3行追加)

```
// 本作成処理…  
$books = new Book;  
$books->item_name = $request->item_name;  
$books->item_number = $request->item_number; 追加  
$books->item_amount = $request->item_amount;  
$books->published = $request->published;  
$books->save();  
return redirect('/');  
....
```

<上記、「登録処理」に以下3行のコードを追加>

```
$books->item_number = $request->item_number;  
$books->item_amount = $request->item_amount;  
$books->published = $request->published;
```

G'sACADEMY Lab10

3. 更新処理に必要な「ルート定義と更新処理」を作成する

3.1 「ルート定義の URL」を先に決めましょう。

/books/update

上記アドレスをルート定義します。

3.2 ルート定義のスタブを作成

```
//更新処理
Route::post('/books/update', function(Request $request) {
    //バリデーション
});
```

3.3 ルート定義解説

/books/update はアドレスです。

URL に「ドメイン/books/update」と入力されていれば、このルート定義を処理します。

POST パラメータは「Request \$request」で取得、ルート定義内にて「\$request」で取得可能です。

参考 : laravel.com (Eloquent 利用の開始)

<https://laravel.com/docs/6.x/eloquent>

<https://laravel.com/docs/5.5/eloquent>

G'sACADEMY Lab10

4. スケルトンに更新処理を追加

スケルトンに更新処理を追加しますが、「登録処理のルート定義」をコピーして使いましょう。重複する処理が多いからです。

/routes/web.php

```
....  
//更新処理  
Route::post('/books/update', function(Request $request) {  
    //バリデーション  
    $validator = Validator::make($request->all(), [  
        'id' => 'required',  
        'item_name' => 'required|min:3|max:255',  
        'item_number' => 'required|min:1|max:3',  
        'item_amount' => 'required|max:6',  
        'published' => 'required',  
    ]);  
    //バリデーション：エラー  
    if ($validator->fails()) {  
        return redirect('/')->  
            ->withInput()  
            ->withErrors($validator);  
    }  
    //データ更新  
    $books = Book::find($request->id);  
    $books->item_name = $request->item_name;  
    $books->item_number = $request->item_number;  
    $books->item_amount = $request->item_amount;  
    $books->published = $request->published;  
    $books->save();  
    return redirect('/');  
});
```

G'sACADEMY Lab10

4.1 「バリデーション」に以下1行のコードを追加

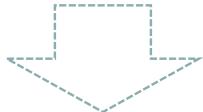
```
'id' => 'required',
```

※更新処理にはターゲットとして id 値が必須。id 値が無いと処理ができないため、必須としています。

4.2 「データ更新」の処理のため、以下1行コードを変更

変更前)

```
$books = new Book;
```



変更後)

```
$books = Book::find($request->id);
```

4.3 構文 :

モデル::find('id値');

Book モデル (Book::) と find メソッドを組み合わせて使用することで、books テーブルの「id 値」を検索して、「テーブルの id 値」と「find で指定した id 値」の等しいレコードを取得します。

4.4 「データ更新」の以下3行コードを変更

変更前)

```
$books->item_number = $request->item_number;  
$books->item_amount = $request->item_amount;  
$books->published = $request->published;
```

固定文字を設定していた箇所に「`$request->プロパティ名`」の送信されてきたデータを代入するように記述変更。

G'sACADEMY Lab10

5. 更新表示と更新処理の動作確認

以下確認項目

本のタイトル

Book	金額
<input type="text"/>	<input type="text"/>

数

<input type="text"/>	公開日
<input type="text"/>	年/月/日

Save

本一覧

PHP基礎	<button>更新</button>	<button>削除</button>
PHP応用	<button>更新</button>	<button>削除</button>
Laravel基礎	<button>更新</button>	<button>削除</button>
Laravel基礎 2	<button>更新</button>	<button>削除</button>
Laravel応用 1	<button>更新</button>	<button>削除</button>

- ・ 更新ボタンをクリックし、更新画面が表示されていることを確認
- ・ エラーは表示されていないこと
- ・ 登録されているデータが表示されているか確認

G'sACADEMY Lab10

- ・「更新」ボタンをクリックし、入力欄にデータが表示されていれば OK

The screenshot shows a mobile application interface for updating a record. The fields and their current values are:

- Title: PHP基礎
- Number: 1
- Amount: 1000
- published: 2017-03-07 00:00:00

At the bottom, there are two buttons: a blue "Save" button and a white "Back" button.

6. 次に更新画面の更新処理を確認

- ・「title」 入力欄の文字列を変更する
- ・「Number」 入力欄の文字列を変更する
- ・「Amount」 入力欄の文字列を変更する
- ・「published」 入力欄の文字列を変更する

上記入力した値の変更だけでなく、バリデーションの文字数（最小・最大）もチェックしておくといいでしょう。

変更した値を確認するには、トップページに戻り、変更したデータを「更新」ボタンをクリックして再度更新画面を開き、データ変更されていることを確認。これで動作確認は完了です。

G'sACADEMY Lab10

Chapter14

練習アプリ

本管理アプリを作っていきましょう。

ルート定義以外のコードをコントローラに移行する

※ルート定義とコントローラの役割の違いを理解する

サンプルコード

http://www.venezia-works.com/aws/laravel6_07.zip

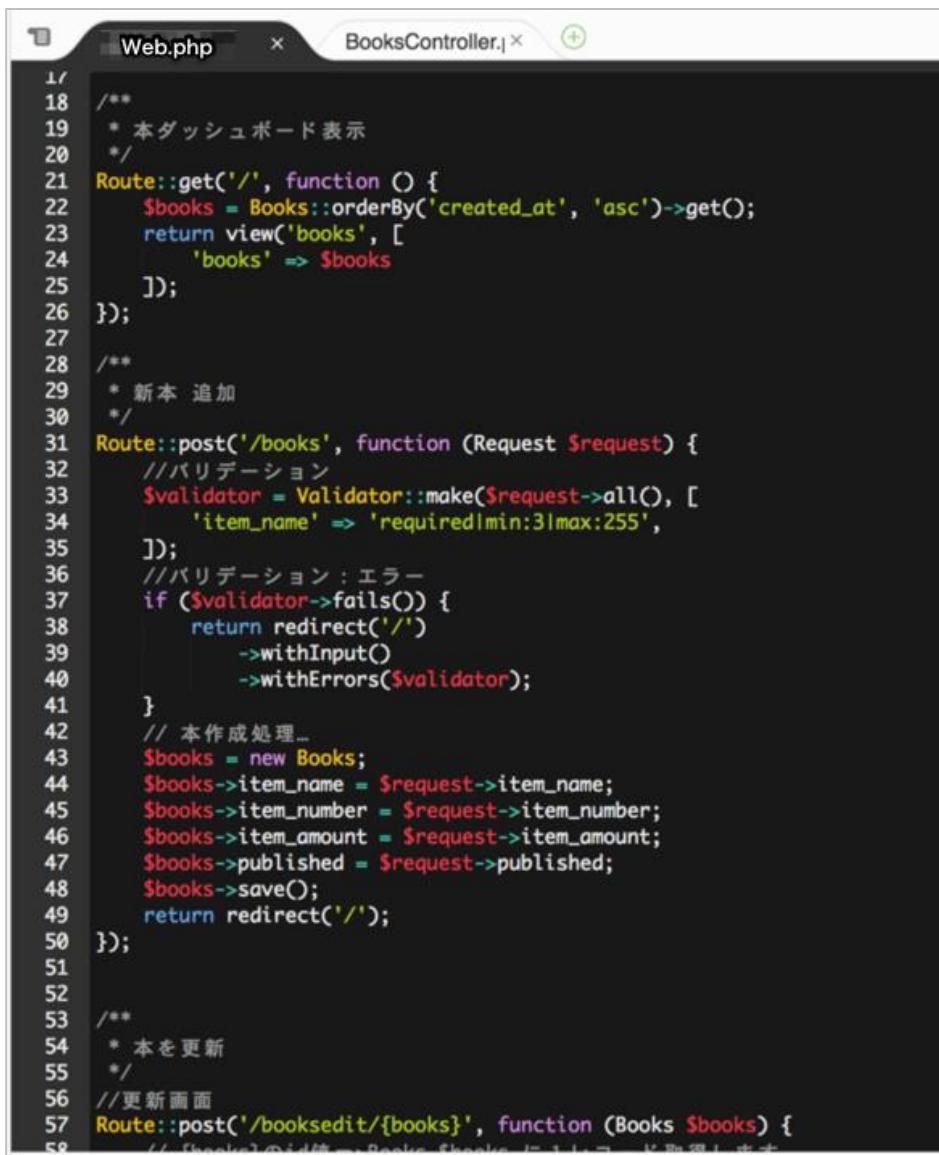
G'sACADEMY Lab10

ルート定義からコントローラに処理コードを移行

簡単なプログラムであれば、コントローラを使用せずにルート定義に記述して完結することも可能です。しかし、ルーティングが増えた場合にはコードの親認性が悪くなります。MVCの観点からもルート定義以外の「各処理」のコードは、コントローラに移行しましょう。

「各処理」とは、特定の「本」一冊を取得したり、保存したりする処理です。新しく作るコントローラは「app/Http/Controllers」ディレクトリに新規作成・配置する artisan コマンドを使います。今回は「BooksController」を作成します。

例) ルーティングに「ルート定義以外の処理」が記述されている。



```
Web.php x BooksController.php +  
17  
18     /**  
19      * 本ダッシュボード表示  
20     */  
21     Route::get('/', function () {  
22         $books = Books::orderBy('created_at', 'asc')->get();  
23         return view('books', [  
24             'books' => $books  
25         ]);  
26     });  
27  
28     /**  
29      * 新本 追加  
30     */  
31     Route::post('/books', function (Request $request) {  
32         //バリデーション  
33         $validator = Validator::make($request->all(), [  
34             'item_name' => 'required|min:3|max:255',  
35         ]);  
36         //バリデーション: エラー  
37         if ($validator->fails()) {  
38             return redirect('/')39                 ->withInput()  
40                 ->withErrors($validator);  
41         }  
42         // 本作成処理...  
43         $books = new Books;  
44         $books->item_name = $request->item_name;  
45         $books->item_number = $request->item_number;  
46         $books->item_amount = $request->item_amount;  
47         $books->published = $request->published;  
48         $books->save();  
49         return redirect('/');  
50     });  
51  
52  
53     /**  
54      * 本を更新  
55     */  
56     //更新画面  
57     Route::post('/booksedit/{books}', function (Books $books) {  
58         // 該当の本情報を BooksCreate に渡す  
59     });
```

G'sACADEMY Lab10

例) ルーティングに「ルート定義」だけ記述している。

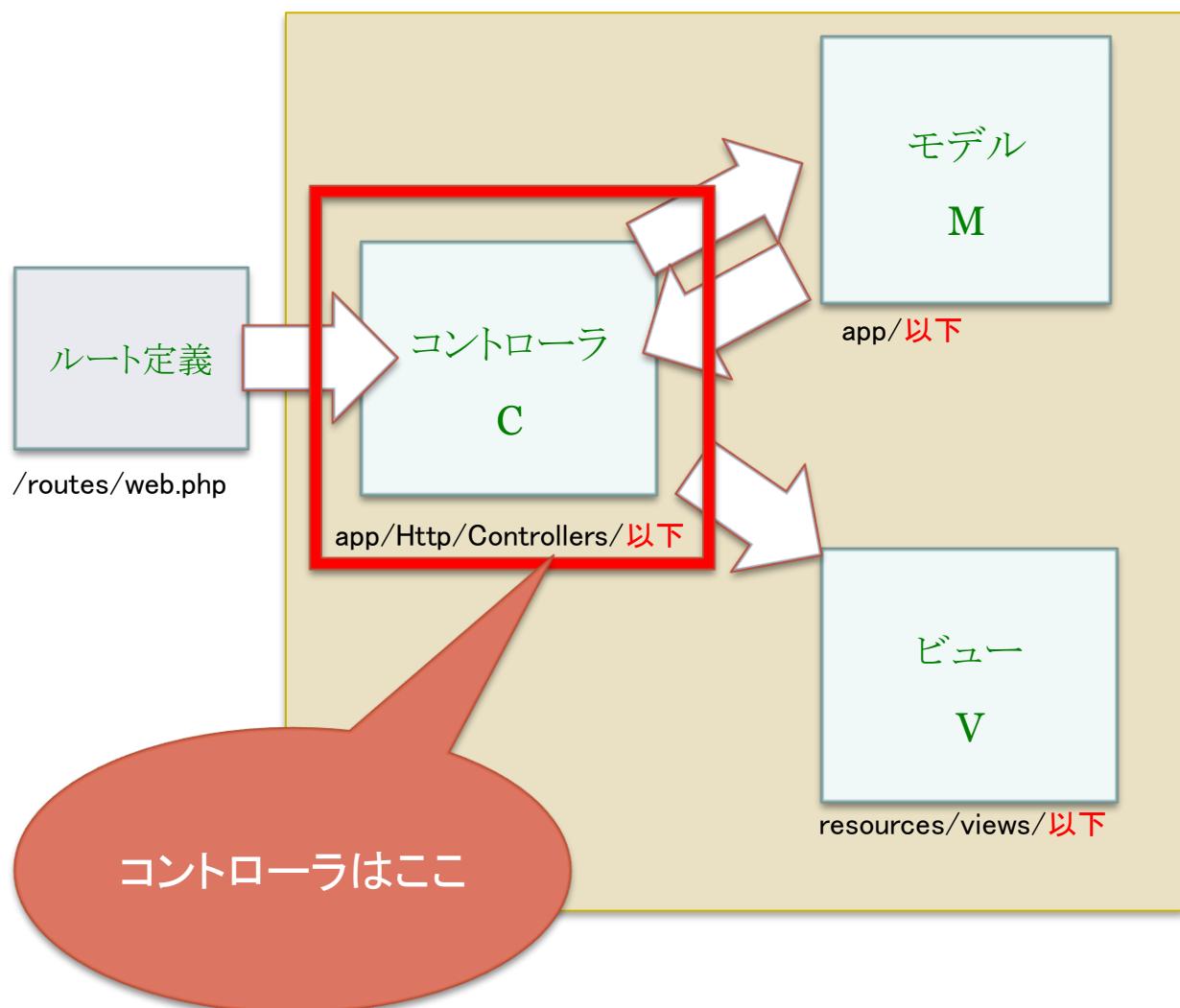
```
1 <?php
2 /*
3 |-----
4 | Application Routes
5 |-----
6 |
7 | Here is where you can register all of the routes for an application.
8 | It's a breeze. Simply tell Laravel the URIs it should respond to
9 | and give it the controller to call when that URI is requested.
10 |
11 */
12 use App\Books;
13 use Illuminate\Http\Request;
14
15 //本ダッシュボード表示
16 Route::get('/', 'BooksController@index');
17
18 //新本 追加
19 Route::post('/books', 'BooksController@store');
20
21 //更新画面
22 Route::post('/booksedit/{books}', 'BooksController@edit');
23
24 //更新処理
25 Route::post('/update', 'BooksController@update');
26
27 //本を削除
28 Route::delete('/book/{book}', 'BooksController@destroy');
```

ルーティングだけなので、視認性・メンテナンス性も良くなります。

コントローラ

- ・コントローラ (Controllers) ディレクトリ

/App/Http/Controllers/**以下に配置**



G'sACADEMY Lab10

ルーティング・コントローラ・ビューの流れ

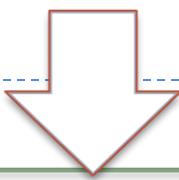
① ルーティング: /routes/web.php

```
Route::get('/books', 'BooksController@index');
```



② コントローラ: /App/Http/Controllers/BooksController.php

```
public function index(){
    return view('books');
}
```



③ ビュー: /resources/views/books.blade.php

【パート・テンプレート】

```
@extends('layouts.app')  
@section('title')
```

トップリスト

```
@endsection
```

【ベース・テンプレート】

```
<!DOCTYPE html>  
<html lang="ja">  
<head>  
    <meta charset="utf-8">  
    <title>@yield('title')</title>
```



G'sACADEMY Lab10

BooksController の作成（更新処理）

ここでは「 BooksController 」というコントローラファイルを作成します。

コントローラ作成 コマンド書式 :

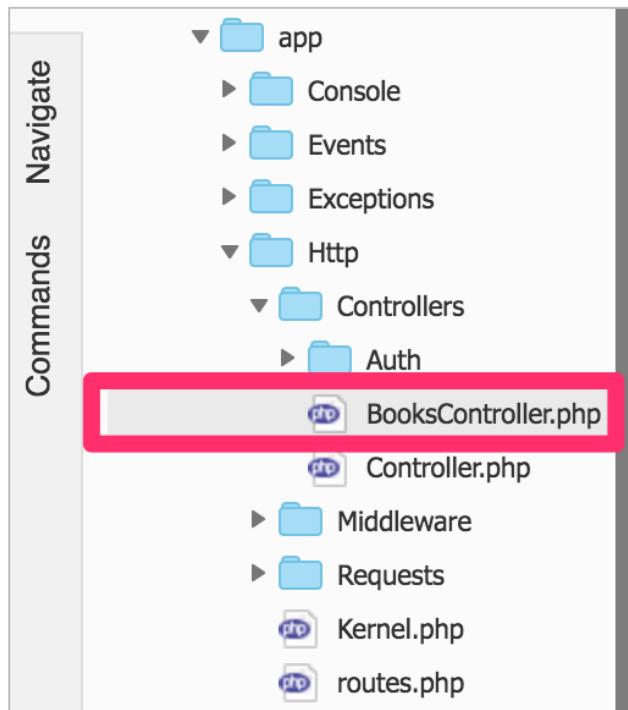
```
php artisan make:controller コントローラ名
```

全 Laravel コントローラは Laravel に含まれている基本コントローラクラスを継承して使用します。

1. 練習 : BooksController を作成します。

```
php artisan make:controller BooksController
```

コントローラ作成コマンドを実行し、`app/Http/Controllers/`以下に「`BooksController.php`」が作成されていることを確認しましょう。



G'sACADEMY Lab10

2. 新規作成したコントローラファイルのコードを確認

app/Http/Controllers/BooksController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class BooksController extends Controller
{
    //
```

上記ファイル「BooksController.php」で使用するための「他のクラス」を2つ読み込みましょう。

- `use App\Book;` //Book モデルを使えるようにする
- `use Validator;` //バリデーションを使えるようにする
- `use Auth;` //認証モデルを使用する

下記は「他のクラス」2つを読み込んだコードです。

app/Http/Controllers/BooksController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
```

```
use App\Book;
use Validator;
use Auth;
```

追加

...

G'sACADEMY Lab10

3. コントローラに update メソッド（スタブ）を追加

update メソッドの引数では「Request パラメータ」を取得し、メソッド内で Request パラメータを処理します。また「更新処理」は後ほど追加記述します。

```
<?php

namespace App\Http\Controllers;
use Illuminate\Http\Request;

use App\Book;
use Validator;

class BooksController extends Controller
{
    //更新
    public function update(Request $request)
    {
        //
    }
}
```

追加

上記は update メソッドを追加したファイル内容になります。

G'sACADEMY Lab10

4. ルート定義の「更新」処理をコピーします。

ルート定義している「更新処理」をコピーで持ってきます。

/routes/web.php (ルート定義)

//更新処理

```
Route::post('/books/update', function(Request $request) {
```

//バリデーション

```
$validator = Validator::make($request->all(), [  
    'id' => 'required',  
    'item_name' => 'required|min:3|max:255',  
    'item_number' => 'required|min:1|max:3',  
    'item_amount' => 'required|max:6',  
    'published' => 'required',  
]);
```

コピー

//バリデーション: エラー

```
if ($validator->fails()) {  
    return redirect('/)  
        ->withInput()  
        ->withErrors($validator);  
}
```

```
$books = Book::find($request->id);
```

```
$books->item_name = $request->item_name;  
$books->item_number = $request->item_number;  
$books->item_amount = $request->item_amount;  
$books->published = $request->published;  
$books->save();  
return redirect('/');
```

```
});
```

G'sACADEMY Lab10

5. ルート定義の「更新」処理をコントローラにペーストします。

app/Http/Controllers/BooksController.php (コントローラ)

```
class BooksController extends Controller
{
    //更新
    public function update(Request $request)
    {
        //バリデーション
        $validator = Validator::make($request->all(), [
            'id' => 'required',
            'item_name' => 'required|min:3|max:255',
            'item_number' => 'required|min:1|max:3',
            'item_amount' => 'required|max:6',
            'published' => 'required',
        ]);
        //バリデーション：エラー
        if ($validator->fails()) {
            return redirect('/')
                ->withInput()
                ->withErrors($validator);
        }
        $books = Book::find($request->id);
        $books->item_name = $request->item_name;
        $books->item_number = $request->item_number;
        $books->item_amount = $request->item_amount;
        $books->published = $request->published;
        $books->save();
        return redirect('/');
    }
}
```

ペースト

上記は update メソッドが完成した状態になります。

G'sACADEMY Lab10

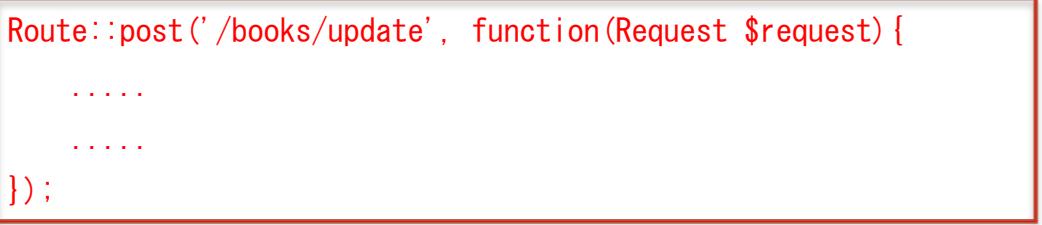
6. ルート定義をクロージャーから「コントローラ」に変更します。

ルート定義内に複数行に渡って処理を記述していましたが、処理内容をコントローラ「BooksController」に移行しました。

次にルート定義の「更新処理」を削除し、新たにルート定義します。

6.1 変更前：/routes/web.php（ルート定義・更新処理）

```
....  
//更新処理  
Route::post('/books/update', function(Request $request) {  
    ....  
    ....  
});  
....
```



削除

6.2 ルート定義：コントローラを利用した場合

```
Route::post('URL', 'コントローラ名@メソッド名');
```



追加

6.3 変更後：/routes/web.php（ルート定義・更新処理）

```
....  
//更新処理  
Route::post('/books/update', 'BooksController@update');  
....
```

G'sACADEMY Lab10

BooksController へ登録処理の追加

1. コントローラに store メソッド（スタブ）を追加します。

store メソッドの引数では「Request パラメータ」の値を取得し、メソッド内で Request パラメータを処理します。また「登録処理」も追加記述します。

```
<?php

namespace App\Http\Controllers;
use Illuminate\Http\Request;

use App\Book;
use Validator;

class BooksController extends Controller
{
    //更新
    public function update(Request $request)
    {
        //

    }

    //登録
    public function store(Request $request)
    {
        //
    }
}
```

追加

上記は store メソッドを追加したファイル内容になります。

G'sACADEMY Lab10

2. ルート定義の「登録」処理をコピーします。

ルート定義している「登録処理」をコピーで持ってきます。

/routes/web.php (ルート定義)

```
/*
 * 新本 追加
 */

Route::post('/books', function (Request $request) {
    //バリデーション
    $validator = Validator::make($request->all(), [
        'item_name' => 'required|min:3|max:255',
        'item_number' => 'required|min:1|max:3',
        'item_amount' => 'required|max:6',
        'published' => 'required',
    ]);
    //バリデーション：エラー
    if ($validator->fails()) {
        return redirect('/')
            ->withInput()
            ->withErrors($validator);
    }
    // 本作成処理…
    $books = new Book;
    $books->item_name = $request->item_name;
    $books->item_number = $request->item_number;
    $books->item_amount = $request->item_amount;
    $books->published = $request->published;
    $books->save();
    return redirect('/');
});
```

コピー

G'sACADEMY Lab10

3. ルート定義の「登録」処理をコントローラにペーストします。

app/Http/Controllers/BooksController.php (コントローラ)

```
//登録
public function store(Request $request)
{
    //バリデーション
    $validator = Validator::make($request->all(), [
        'item_name' => 'required|min:3|max:255',
        'item_number' => 'required|min:1|max:3',
        'item_amount' => 'required|max:6',
        'published' => 'required',
    ]);
    //バリデーション：エラー
    if ($validator->fails()) {
        return redirect('/')
            ->withInput()
            ->withErrors($validator);
    }
    // 本作成処理...
    $books = new Book;
    $books->item_name = $request->item_name;
    $books->item_number = $request->item_number;
    $books->item_amount = $request->item_amount;
    $books->published = $request->published;
    $books->save();
    return redirect('/');
}

....
```

上記は store メソッドが完成した状態になります。

G'sACADEMY Lab10

4. ルート定義をクロージャーから「コントローラ」に変更

4.1 変更前： /routes/web.php（ルート定義・更新処理）

```
....  
/**  
 * 新本 追加  
 */  
Route::post('/books', function (Request $request) {  
    ....  
    ....  
});  
....
```

削除



4.2 ルート定義：コントローラを利用した場合

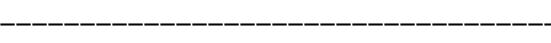
```
Route::post('URL', 'コントローラ名@メソッド名');
```



4.3 変更後： /routes/web.php（ルート定義・登録処理）

```
....  
//登録処理  
Route::post('/books', 'BooksController@store');
```

追加



課題 3

G'sACADEMY Lab10

課題3：

ルート定義「3箇所」をコントローラに移行

1. 最初に表示されるページへのルート定義

```
/*
 * 本ダッシュボード表示
 */
Route::get('/', function () {
    $books = Book::orderBy('created_at', 'asc')->get();
    return view('books', [
        'books' => $books
    ]);
});
```

※ルート定義も変更する必要があります。

2. 更新画面へのルート定義

```
//更新画面
Route::post('/booksedit/{books}', function (Book $books) {
    // {books} の id 値=>Book $books に1レコード取得します。
    return view('booksedit', [
        'book' => $books
    ]);
});
```

※ルート定義も変更する必要があります。

G'sACADEMY Lab10

3. 削除処理へのルート定義

```
/**  
 * 本を削除  
 */  
Route::delete('/book/{book}', function (Book $book) {  
    $book->delete();  
    return redirect('/');  
});
```

※ルート定義も変更する必要があります。

<課題3：内容の確認>

全てのルート定義に記述している「処理」を
「app/Http/Controllers/BooksController.php」
コントローラに移行します。

<課題全体の捉え方>

練習のための課題です。自分で考えた変更でもかまいません。
自分で考え、知識を消化できるよう打って確かめながら進めることが大事です。

※わからない場合でも解答例があるので安心してください。

参考 : laravel.com

<https://laravel.com/docs/6.x/>
<https://laravel.com/docs/5.5/>

課題解答 3.1

課題 1. 最初に表示されるページへのルートを定義

コントローラに index メソッドを追加しましょう。

index メソッドの引数は必要ありません。

```
class BooksController extends Controller
{
    //本ダッシュボード表示
    public function index()
    {
        $books = Book::orderBy('created_at', 'asc')->get();
        return view('books', [
            'books' => $books
        ]);
    }
}
```

追加

```
//更新
public function update(Request $request)
{
    //
}
...
}
```

変更後ルート定義 : /routes/web.php

```
//本ダッシュボード表示
Route::get('/', 'BooksController@index');
```

課題解答 3.2

課題 2. 更新画面へのルート定義

コントローラに edit メソッドを追加しましょう。

edit メソッドの引数には「Book \$books」を記述。Book モデルを通して
「\$books」に 1 レコードの値を受け取ります。※ルート定義の記述と同意。

```
//本ダッシュボード表示
public function index()
{
    ...
}
```

```
//更新画面
public function edit(Book $books)
{
    return view('booksedit', [
        'book' => $books
    ]);
}
```

追加

変更後ルート定義 : /routes/web.php

```
//更新画面
Route::post('/booksedit/{books}', 'BooksController@edit');
```

課題解答 3.3

課題 3. 削除処理へのルート定義

コントローラに destroy メソッドを追加

destroy メソッドの引数には「Book \$books」を記述。Book モデルを通して
「\$books」に 1 レコードの値を受け取ります。※ルート定義の記述と同意。

```
//本ダッシュボード表示
public function index()
{
    ...
}
```

```
//削除処理
public function destroy(Book $book)
{
    $book->delete();
    return redirect('/');
}
```

追加

変更後ルート定義 : /routes/web.php

```
//本を削除
Route::delete('/book/{book}', 'BooksController@destroy');
```

G'sACADEMY Lab10

ルート定義：課題完了後のコード

ルート定義 web.php にはルーティングだけを記述。

```
use App\Book;  
use Illuminate\Http\Request;  
  
//本 ダッシュボード表示  
Route::get('/', 'BooksController@index');  
  
//登録処理  
Route::post('/books', 'BooksController@store');  
  
//更新画面  
Route::post('/booksedit/{books}', 'BooksController@edit');  
  
//更新処理  
Route::post('/books/update', 'BooksController@update');  
  
//本を削除  
Route::delete('/book/{book}', 'BooksController@destroy');
```

ルート定義とコントローラを分けたことで、コードはスッキリしました。
コントローラには、処理とデータ操作を記述することでコードが見やすくなり
ました。そしてコントローラとルート定義の役割も理解が進んだはずです。

課題サンプルコード

<http://www.venezia-works.com/aws/kadai3.zip>

G'sACADEMY Lab10

Chapter15

ページネーション

サンプルコード

<http://www.venezia-works.com/aws/paginate.zip>

ページネーション

1. ページネーションを使う

1ページに表示する制限数を定めることができます。表示制限を設定した場合に「表示できないページ」がでてきますが、「2ページ」「3ページ」とリンクが自動で生成されます。この機能を「ページネーション」と呼んでいます。

記述式)

モデル名::paginate(1ページあたりの表示数);

記述例)

\$books = Book::paginate(3);

※ get(); all(); のデータ取得メソッドと代えて使います。

変更前

```
class BooksController extends Controller
{
    //本ダッシュボード表示
    public function index()
    {
        $books = Book::orderBy('created_at', 'asc')->get();
    }
}
```

変更後

```
class BooksController extends Controller
{
    //本ダッシュボード表示
    public function index()
    {
        $books = Book::orderBy('created_at', 'asc')->paginate(3);
    }
}
```

G'sACADEMY Lab10

2. ページネーションをテンプレートに埋め込む

ページネーションの機能を使用した場合には、ビューの Blade テンプレートにも、`{{モデル名->links()}}` の 1 構文だけ HTML に記述する必要があります。

記述式)

```
 {{ モデル名->links() }}
```

記述例)

```
...
<div class="row">
    <div class="col-md-4 offset-md-4">
        {{ $books->links() }}
    </div>
</div>
...

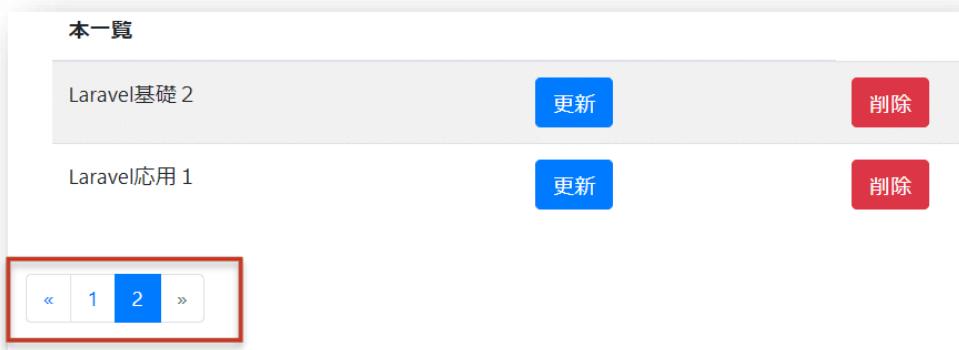
```

3. bootstrap/app.php の末尾「return」上に追記

```
Illuminate\Pagination\AbstractPaginator::defaultView("pagination::bootstrap-4");
```

※Bootstrap4 対応

4. ブラウザで表示確認



G'sACADEMY Lab10

Chapter16

サインイン・ログイン認証

The screenshot shows a web application interface for logging in. At the top, there are navigation links: 'Laravel' and 'Home' on the left, and 'Login' and 'Register' on the right. Below this is a 'Login' form. The form has two input fields: 'E-Mail Address' and 'Password'. There is also a 'Remember Me' checkbox and two buttons at the bottom: a blue 'Login' button with a key icon and a link 'Forgot Your Password?'. The background of the page is white.

「 Chapter9 」にて認証機能を既に「設置」しています。

G'sACADEMY Lab10

1. ルート定義に以下認証用の2行が追加されています

```
/routes/web.php
```

```
-----  
Auth::routes(); //認証機能を使用する  
Route::get('/home', 'HomeController@index')->name('home');
```

’/home’ のページは、ログイン後に表示可能な初期設定となっています。

2. コントローラに「HomeController.php」が記述されています。

```
/app/Http/Controller/HomeController.php
```

```
-----  
public function __construct()  
{  
    $this->middleware('auth');  
}
```

「`__construct()`」はクラスが呼ばれたら自動で最初に実行されます。
「`$this->middleware('auth');`」は、「認証していたら表示する」という処理です。ログイン認証してなければ非表示になります。また、この1行をコメントアウトすると「ログイン認証」してなくても表示されます。

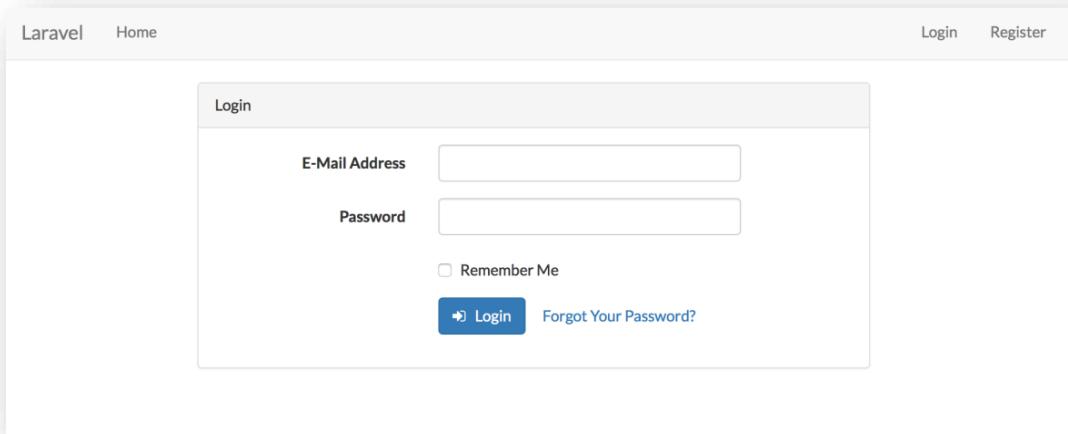
G'sACADEMY Lab10

3. URL に「/home」を追加して確認します。

<表示確認 URL>

https://*****自身のURL*****.amazonaws.com/home

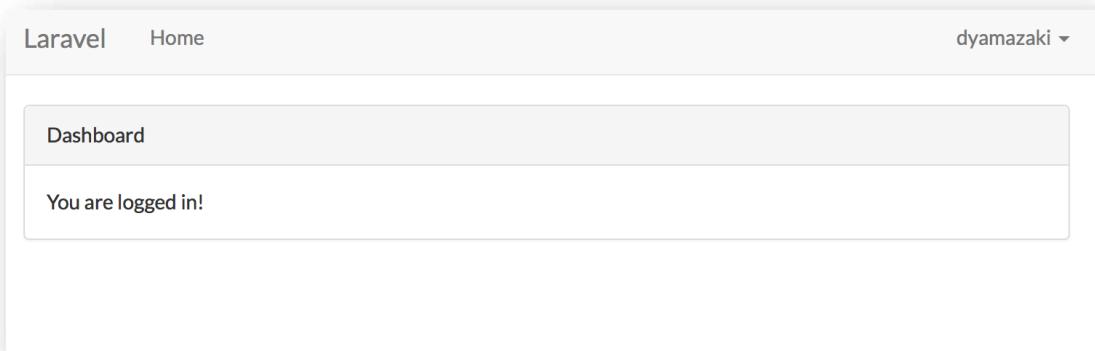
ログイン画面が表示されれば正常な動作です。



ログインしていない状態で、/home を表示すると、「ログインください」と催促し、ログイン画面が表示されます。

また、他のページにはログインの制御処理を記述していないので、今までどおり表示できていることを確認しましょう。

ログイン後に '/home' を表示すると以下のようないいな画面が表示されます。



G'sACADEMY Lab10

4. 他のページもログイン認証後にだけ表示するように変更

コントローラ「BooksController.php」に、「HomeController.php」の
「__construct()」メソッドを追加記述します。

app/Http/Controllers/BooksController.php

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Book;
use Validator;

class BooksController extends Controller
{
    //コンストラクタ（このクラスが呼ばれたら最初に処理をする） 追加
    public function __construct()
    {
        $this->middleware('auth');
    }
}
```

上記は「__construct()」メソッドを追加したファイル内容になります。

11. 「Register」新規ユーザー登録後に表示するページを変更

Register 後に「登録/一覧画面」を表示するように変更します。

- ・変更前： Route::get('/home', 'HomeController@index')->name('home');
- ・変更後： Route::get('/home', 'BooksController@index')->name('home');

G'sACADEMY Lab10

5. 全てのページがログイン後に表示されるか確認

<表示確認 URL>

http://* * * * *自身の URL* * * * *.amazonaws.com

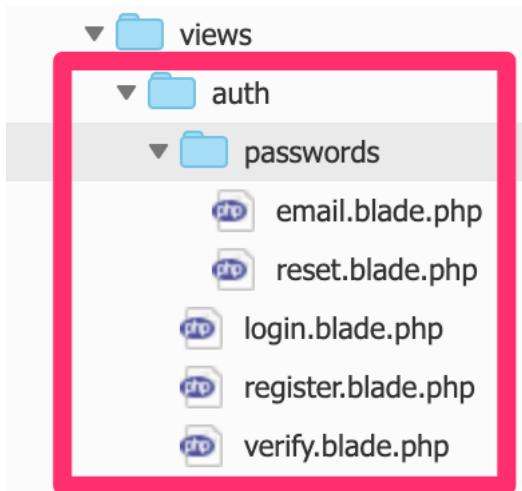


※他の URL(ページ)も確認しましょう。全てログイン認証が必要になります。

G'sACADEMY Lab10

6. ビューに追加された「ログイン認証」ファイルを確認

ビューに追加された「auth」フォルダ一式は「ログイン認証」用です。



ログイン認証、サインインなど必要なテンプレートが追加されました。
項目名やHTMLの構造などは、このフォルダ内のテンプレートで指定可能で
す。

-
- `login.blade.php`
 - `register.blade.php`
-

の項目名を日本語に変えてみましょう。

※ Laravel 5.4 以上では `emails` フォルダはありません。

参考 : laravel.com (認証機能)

<https://laravel.com/docs/6.x/authentication>

<https://laravel.com/docs/5.5/authentication>

ログイン認証

ユーザー情報取得 & Middleware 認証

G'sACADEMY Lab10

ログイン認証したユーザーの情報を取得方法

ログインしたユーザーの情報を使用することはアプリではよくあります。例えば「本（Book）を登録するにはログイン認証が必要で、登録したデータには誰が登録したかをデータテーブルに保持しておきたい」このようなケースは往々にあります。

記述式：ログインユーザーの情報取得

Auth::user ()->プロパティ名；

記述例)

テンプレート側で値を取得する方法

```
<td> id : {{ Auth::user ()->id }} </td>
<td> {{ Auth::user ()->name }} さん </td>
<td> 登録日時 : {{ Auth::user ()->created_at }} </td>
```

記述例)

コントローラ内で取得して利用する方法

```
use Auth; //認証モデルを使用する(他の use を記述している箇所下に記述)
```

...

//使用例 1

```
$auth = Auth::user ()->id; // $auth に「id」値のみ代入
```

//使用例 2

```
$auths = Auth::user (); // $auths に「id, name, email...」の複数値を代入
```

```
// テンプレート側で「 $auths->プロパティ名 」で値取得が可能
```

```
return view( 'books' , [ 'books' =>$books, 'auths' =>$auths ] );
```

コントローラで取得して使用するのはデータの登録や更新に使用することが多くなると思いますので、覚えておきましょう。

G'sACADEMY Lab10

Middleware を使ったログイン認証チェック

ログイン認証のチェックは前章で解説した以下 1 の方法だけではなく、middleware を利用した方法があります。

1. 前章の解説例)

```
//コンストラクタ（このクラスが呼ばれたら最初に処理をする）
public function __construct()
{
    $this->middleware('auth');
}
```

...

2. Middleware を使用した例)

```
/routes/web.php
```

```
Route::group(['middleware' => 'auth'], function () {
    //welcome ページを表示
    Route::get("/", function () {
        return view("welcome");
    });
});
```

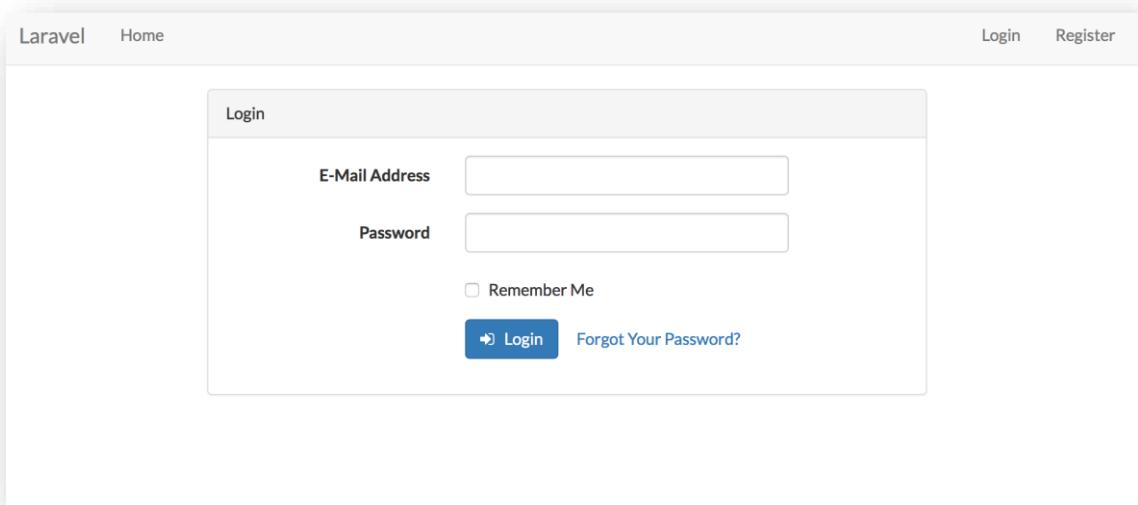
...

上記のように `Route::group(['middleware' => 'auth'], ...)` を使用することで、簡単にログインしていないと見れないページにする事ができます。

コントローラー側かルーティング側かは、場合に応じて使い分けましょう。

G'sACADEMY Lab10

サインイン・ログイン認証 1ユーザー × 1サービス



G'sACADEMY Lab10

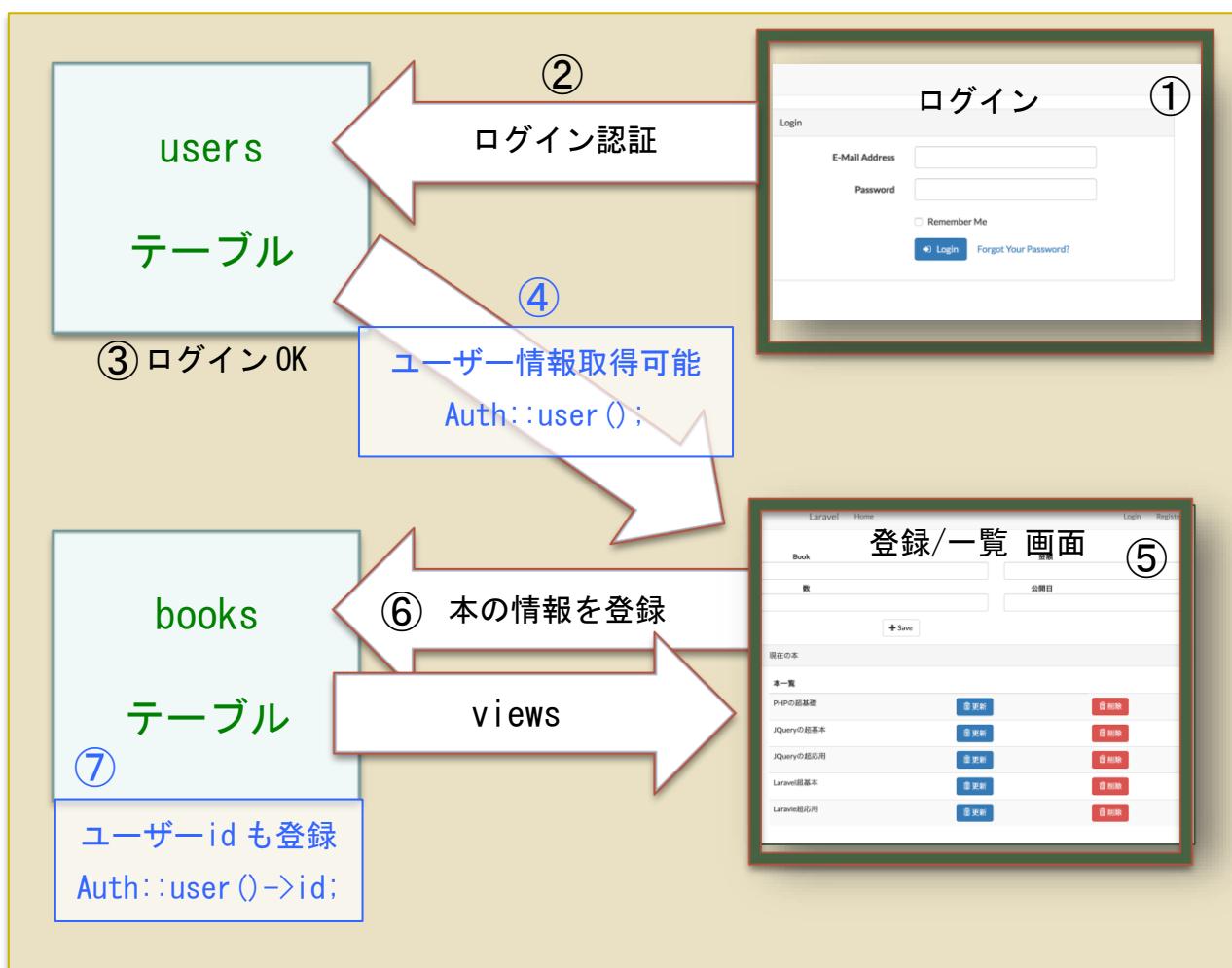
ユーザーがログインしたらユーザーが登録した本のみ表示

ここ迄は、ログイン認証追加、ユーザー情報取得を知りました。

現状の仕様は、ログイン認証後「登録した全てのデータを一覧表示」する仕様になっています。一つのサービスを全員で共有する仕様です。

(現状の books テーブルのデータは、誰が登録したかわからない状態です。)

ここでは、誰が登録したかも books テーブルに保持できるようにしましょう。



上記図を参照、

④の段階で、ユーザー情報全てを「`Auth::user()`」にオブジェクトで保持。

⑦で「`Auth::user()>id;`」でユーザーidを取得、本データと一緒に登録。

G'sACADEMY Lab10

ユーザー毎にデータを別々で保持することができる

- ・赤枠「yamazaki01」としてログインしたデータ表示例

Laravel

yamazaki01 ▾

Book	金額
数	公開日

Save

本一覧

Laravel02	更新	削除
Laravel01	更新	削除

- ・赤枠「yamazaki02」としてログインしたデータ表示例

Laravel

yamazaki02 ▾

Book	金額
数	公開日

Save

本一覧

jquey昨日入門2	更新	削除
JQuery今日入門3	更新	削除

G'sACADEMY Lab10

1. ユーザーidを登録できるようにbooksテーブルを変更

最初にテーブルの構造を変更する必要があるので、booksテーブルのmigrationファイルを修正します。

修正するファイル：

[yyyy_mm_dd]_[hhii:ss]_create_books_table.php

```
14     public function up()
15     {
16         Schema::create('books', function (Blueprint $table) {
17             $table->increments('id');
18             $table->integer('user_id'); //Add:user_id
19             $table->string('item_name');
20             $table->integer('item_number');
21             $table->integer('item_amount');
22             $table->datetime('published');
23             $table->timestamps();
24         });
25     }
26 }
```

2. データテーブルをリセット（削除）

ファイルを修正したら以下コマンドでリセットをしましょう。

※データが削除して無くなることを認識しておきましょう。

```
php artisan migrate:reset
```

※mysqlが起動しないとErrorになります。「`sudo service mysqld start`」コマンドでMysqlが起動しましょう。

3. データテーブルを再構築

ファイルを修正したら以下コマンドでデータテーブルの再構築をしましょう。

```
php artisan migrate
```

G'sACADEMY Lab10

4. MySQL コマンドを使用し、テーブルができているか確認
順番にコマンドを打って確認していきましょう。

1. MySQL コマンド開始

```
mysql -u root -p  
root #パスワードを入力し Enter (パスワードは入力しても表示されません)
```

2. データベースを選択

```
use c9;
```

3. テーブルを一覧表示

```
show tables;  
mysql> show tables;  
+-----+  
| Tables_in_c9 |  
+-----+  
| books      |  
| migrations |  
| password_resets |  
| tasks      |  
| users      |  
+-----+  
5 rows in set (0.00 sec)
```

4. books テーブルデータを表示

```
select * from books;  
  
mysql> select * from books;  
Empty set (0.00 sec)
```

※books テーブルのデータはリセットされ消去しました。

5. users テーブルデータを表示

```
select * from users;  
  
mysql> select * from users;  
Empty set (0.00 sec)
```

※users テーブルのデータはリセットされ消去しました。

G'sACADEMY Lab10

6. books テーブルデータの構造を表示

```
desc books;
```

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_increment
user_id	int(11)	NO		NULL	
item_name	varchar(255)	NO		NULL	
item_number	int(11)	NO		NULL	
item_amount	int(11)	NO		NULL	
published	datetime	NO		NULL	
created_at	timestamp	YES		NULL	
updated_at	timestamp	YES		NULL	

books テーブルに 1 カラム「user_id」が追加されました。

これで「books テーブル」、「users テーブル」のデータが空になっていることを確認できました。また、books テーブルに「user_id」カラムが追加されていることも確認できました。

5. 表示処理：コントローラ「BooksController@index」を修正

次に「表示処理」の一部が修正必要です。修正していきましょう。

BooksController.php [@index]

```
19 //本ダッシュボード表示
20
21 public function index()
22 {
23     $books = Book::where('user_id',Auth::user()->id)->orderBy('created_at', 'desc')->paginate(3);
24     $auths = Auth::user();
25     return view('books', [
26         'books' => $books,
27         'auths' => $auths
28     ]);
29 }
```

BooksController.php に以下を追加を追加しておきましょう。

use Auth; //Auth を使えるように読み込む

変更前 : \$books = Book::orderBy('created_at', 'asc')->paginate(3);

変更後 : \$books = Book::where('user_id',Auth::user()->id)
->orderBy('created_at', 'desc')
->paginate(3);

G'sACADEMY Lab10

6. コントローラ「BooksController@store」の表示処理を修正

次に「登録処理」の一部が修正必要です。修正していきましょう。

BooksController.php [@store]

```
68     //登録
69     public function store(Request $request){
70         //バリデーション
71         $validator = Validator::make($request->all(), [
72             'item_name' => 'required|min:3|max:255',
73             'item_number' => 'required|min:1|max:3',
74             'item_amount' => 'required|max:6',
75             'published' => 'required',
76         ]);
77         //バリデーション：エラー
78         if ($validator->fails()) {
79             return redirect('/')
80                 ->withInput()
81                 ->withErrors($validator);
82         }
83         // 本作成処理...
84         $books = new Book;
85         $books->user_id      = Auth::user()->id; //追加のコード
86         $books->item_name    = $request->item_name;
87         $books->item_number   = $request->item_number;
88         $books->item_amount   = $request->item_amount;
89         $books->published    = $request->published;
90         $books->save();
91         return redirect('/');
92     }
93 }
```

追加するコード 1 行

```
$books = new Book; //既存のコード
$books->user_id  = Auth::user()->id; //追加のコード
....
```

G'sACADEMY Lab10

7. コントローラ「BooksController@update」の更新処理を修正

次に「更新処理」の一部が修正必要です。修正していきましょう。

BooksController.php [@update]

```
41     //更新
42     public function update(Request $request)
43     {
44
45         ///バリデーション
46         $validator = Validator::make($request->all(), [
47             'id' => 'required',
48             'item_name' => 'required|min:3|max:255',
49             'item_number' => 'required|min:1|max:3',
50             'item_amount' => 'required|max:6',
51             'published' => 'required',
52         ]);
53         ///バリデーション：エラー
54         if ($validator->fails()) {
55             return redirect('/')
56                 ->withInput()
57                 ->withErrors($validator);
58         }
59         $books = Book::where('user_id',Auth::user()->id)->find($request->id);
60         $books->item_name = $request->item_name;
61         $books->item_number = $request->item_number;
62         $books->item_amount = $request->item_amount;
63         $books->published = $request->published;
64         $books->save();
65         return redirect('/');
66     }
```

追加するコード1行

変更前：

\$books = Book::find(\$request->id);

変更後：

\$books = Book::where('user_id',Auth::user()->id)->find(\$request->id);

G'sACADEMY Lab10

8. コントローラ「BooksController@edit」の更新表示を修正

次に「更新表示処理」の一部が修正必要です。修正していきましょう。

こここの変更は大きいので、全て変更くらいの感覚で書き直しましょう。

BooksController.php [@edit]

```
33     public function edit($book_id)
34     {
35         $books = Book::where('user_id',Auth::user()->id)->find($book_id);
36         return view('booksedit', [
37             'book' => $books
38         ]);
39     }
```

変更後：変更箇所を赤文字にしてあります。

変更前：

```
public function edit(Book $books)
{
    return view('booksedit', [
        'book' => $books
    ]);
}
```

変更後：

```
public function edit($book_id)
{
    $books = Book::where('user_id',Auth::user()->id)->find($book_id);
    return view('booksedit', [
        'book' => $books
    ]);
}
```

以上になります。

ブラウザ画面で各ユーザーの画面で登録データを確認してください。

Chapter17

ファイルアップロード

G'sACADEMY Lab10

Laravel のファイルアップロードの「基本的」な方法を知りましょう。
(Laravel File Manager を使わない方法)

<完成例>

The screenshot shows a web application interface for managing files. At the top, there is a form with fields for '本のタイトル' (Book Title), '金額' (Amount), '数' (Quantity), and '公開日' (Publication Date). Below this is a section labeled '①画像を選択して登録' (① Select an image and register) with a red box around it. It includes a file input field with the placeholder 'ファイルを選択' (Select file) and the message '選択されていません' (No file selected). A blue 'Save' button is located to the right. Below the form is a table titled '本一覧' (List of books) showing two entries:

書名	著者名	状態
Laravel基礎	著者名	登録済み
PHP基礎	著者名	未登録

A red box highlights the first row under 'Laravel基礎'. To the right of the table is a blue '更新' (Update) button. Overlaid on the bottom right of the table area is the text '②登録された画像が表示される' (② The registered image is displayed).

<以下、追加内容>

- ・ テーブルにファイル保存用のカラムを追加
- ・ テンプレートにファイル選択ボタンを作成
- ・ ファイルアップロード処理を追加
- ・ public(公開)フォルダに upload フォルダを作成(ファイルアップロード用)

G'sACADEMY Lab10

1. books テーブルにファイル名保存用 カラムを追加

[database/migrations/*****create_books_table.php]

```
14     public function up()
15     {
16         Schema::create('books', function (Blueprint $table) {
17             $table->increments('id');
18             $table->string('item_name');
19             $table->integer('item_number');
20             $table->integer('item_amount');
21             $table->string('item_img'); -----
22             $table->datetime('published');
23             $table->timestamps();
24         });
25     }
```

item_img カラムを string (文字列型) で追加記述

```
$table->string('item_img');
```

2. books テーブルを再構築する

artisan コマンドで、上記内容にテーブルを再定義

```
php artisan migrate:refresh
```

※既に登録されているデータは他のテーブル含めて、全て初期化されます。

mysql 画面で確認してみましょう。

```
mysql> desc books;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra       |
+-----+-----+-----+-----+-----+-----+
| id    | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| item_name | varchar(191) | NO   |     | NULL    |               |
| item_number | int(11) | NO   |     | NULL    |               |
| item_amount | int(11) | NO   |     | NULL    |               |
| item_img | varchar(191) | NO   |     | NULL    |               -----
| published | datetime | NO   |     | NULL    |               |
| created_at | timestamp | YES  |     | NULL    |               |
| updated_at | timestamp | YES  |     | NULL    |               |
+-----+-----+-----+-----+-----+-----+
8 rows in set (0.00 sec)
```

G'sACADEMY Lab10

3. テンプレートに「ファイル選択」ボタンを表示。

重要: <form ...> に「`enctype="multipart/form-data"`」を追加！

books.blade.php

```
...  
<form enctype="multipart/form-data" action="{{ url('books') }}" met...  
...
```

「`enctype="multipart/form-data"`」はファイル送信に必要な属性です。

以下、form 要素内に追加してください。

books.blade.php

```
...  
<!-- file 追加 -->  
<div class="col-sm-6">  
  <label>画像</label>  
  <input type="file" name="item_img">  
</div>  
...
```



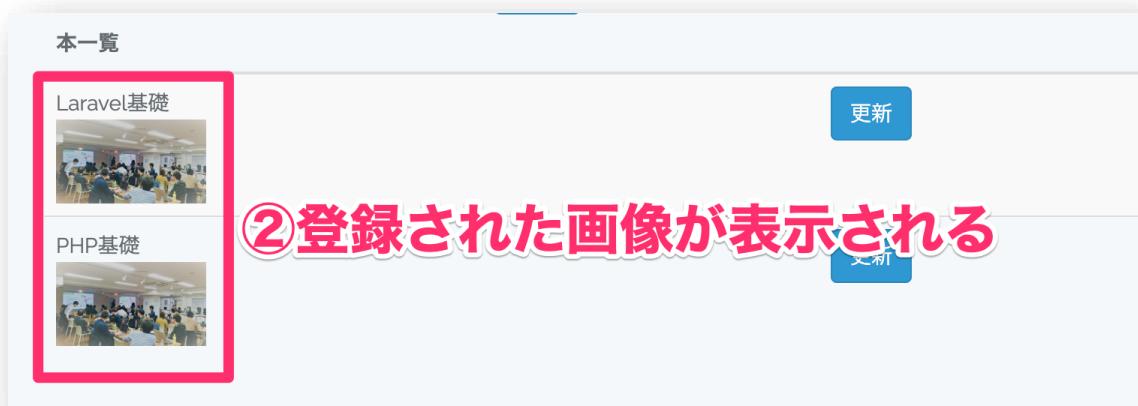
G'sACADEMY Lab10

4. テンプレートに「サムネイル表示」を追加

本タイトルのコメントのある div 要素内に以下赤い記述を追加しましょう。

books.blade.php

```
...
<!-- 本タイトル -->
<td class="table-text">
    <div> {{ $book->item_name }} </div>
    <div> </div>
</td>
...
```



G'sACADEMY Lab10

5. ファイルアップロード処理を追加する

POST/GET の文字列を受け取る方法と同様に「 Request \$request 」を使いファイルを受取ります。ファイルは upload フォルダに保存します。

BooksController.php [@store]

```
...
//file 取得
$file = $request->file('item_img');
//file が空かチェック
if( !empty($file) ){
    //ファイル名を取得
    $filename = $file->getClientOriginalName();
    //AWSの場合どちらかになる事がある"../upload/" or "./upload/"
    $move = $file->move('./upload/' , $filename); //public/upload/...
} else{
    $filename = "";
}
```

```
// Eloquent モデル
$books = new Book;
$books->item_name = $request->item_name;
$books->item_number = $request->item_number;
$books->item_amount = $request->item_amount;
$books->item_img = $filename;
$books->published = $request->published;
```

「 \$filename 」にアップロードしたファイル名が代入されます。

「 \$books->item_img = \$filename; 」の記述を追加

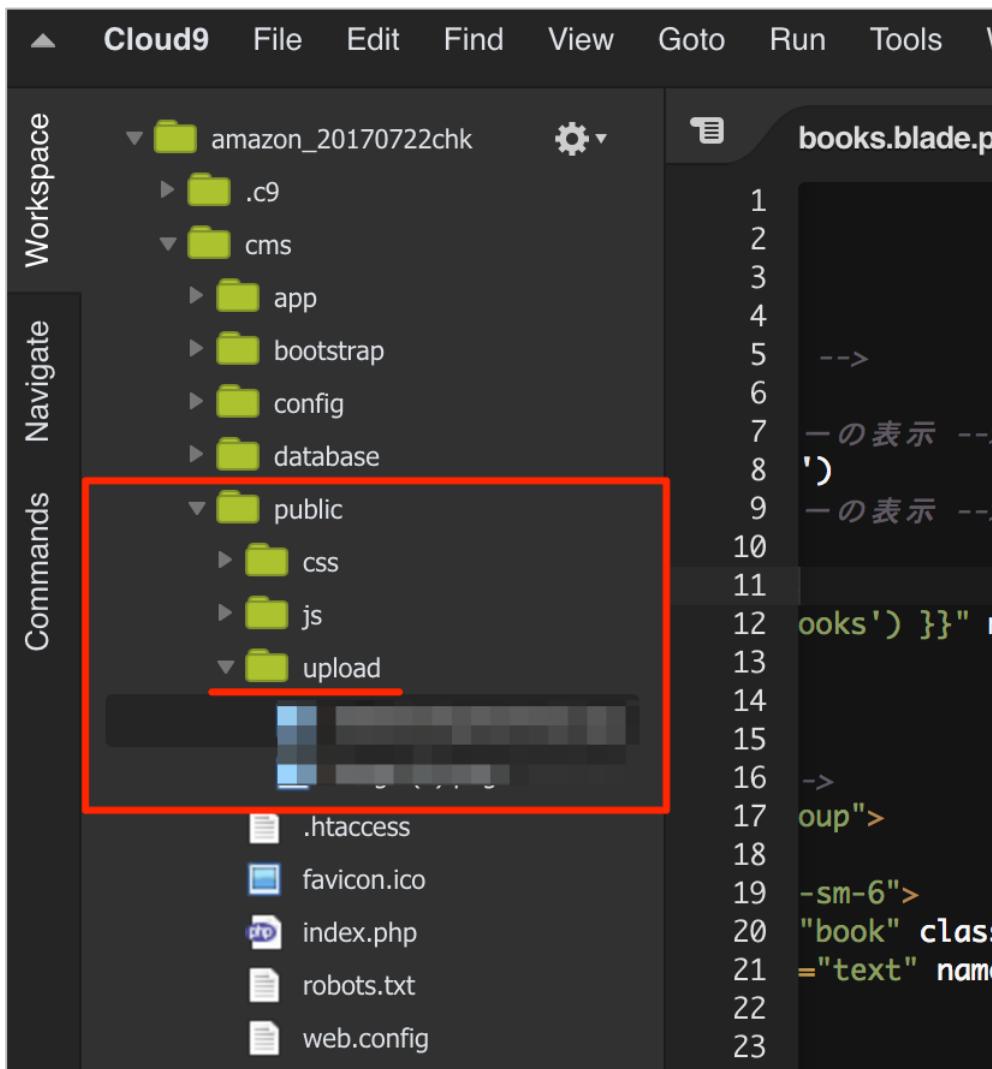
テーブルにファイル名を保存する際に「 \$filename 」の値を使います。

G'sACADEMY Lab10

6. public (公開) フォルダ内に upload フォルダを作成

「upload」フォルダはファイルアップロードしたファイルを保存する場所です。他のフォルダに作ってしまうとブラウザから見れないので「public」に作ります。

public に upload フォルダを追加した状態



The screenshot shows the Cloud9 IDE interface. On the left, there are three vertical panels: 'Workspace' (containing a project named 'amazon_20170722chk' with sub-folders '.c9', 'cms', 'app', 'bootstrap', 'config', and 'database'), 'Navigate' (listing files like '.htaccess', 'favicon.ico', 'index.php', 'robots.txt', and 'web.config'), and 'Commands' (listing files 'css', 'js', and 'upload' under the 'public' folder). A red box highlights the 'public' folder and its contents. On the right, the code editor displays a PHP file named 'books.blade.php' with some code. The 'upload' folder is currently selected in the 'Commands' panel.

作業は以上で完了です。

ブラウザからファイルを選択して画像が表示されるか確認しましょう。

CODE 書く！