

1, 下準備 - 1

1, PHPのインストール

<https://www.php.net/manual/ja/install.php>

2, PHPのバージョン確認

```
$ php --version
```

3, composerのインストール

<https://getcomposer.org/download/>

4, composerのバージョン確認

```
$ composer -V
```

5, laravel のインストール (バージョン指定しなければ最新版)

```
$ composer create-project laravel/laravel app_name "6.0.*"
```

6, laravel バージョン確認 (cd app_name でフォルダ移動)

```
php artisan --version
```

7, ビルトインサーバーで確認

```
php artisan serve
```

(8, フロントエンド開発のベースツール)

```
composer require laravel/ui (もしくは) composer require laravel/ui "1.x" --dev
```

(9, vue のベースコード作成)

```
php artisan ui vue
```

(10, フロントエンドパッケージインストール)

```
npm install
```

(11, Vue でルーティング処理を行う際、必要となる)

```
npm install --save vue-router
```

(12, ソースコードをビルド。今後動かない時、これを打ち忘れて売ることが多い)

```
npm run dev
```

↓上手く行かない時↓

<https://qiita.com/minato-naka/items/2d2def4d66ec88dc3ca2>

2, 下準備 - 2

1, MAMP の MySQL の設定に laravel の dev を合わせる

2, DBの作成

3/config/app の設定の "timezone", "locale", "faker_locale" の変更

```
'Asia/Tokyo', 'ja', 'ja_JP'
```

3, マイグレーションとモデル

1, モデル (DBと連携するクラス) とマイグレーションファイル (テーブルの設計図) の作成。大文字開始、単数形の名前が一般的モデルは app/モデル名.php として作成、マイグレーションファイルは database/migration/ 下に作成される

```
$ php artisan make:model -m Test
```

2, ↓を参考にマイグレーションファイルに定義などを書き込む。sql 構文とビミョーに違う

<https://readouble.com/laravel/8.x/ja/migrations.html#collection-method-list>

3, 失敗しても、追加も変更も削除も可能。ただデータが飛ぶ点に注意。イケると思ったらマイグレーションの実行

```
$ php artisan migrate
```

4, テーブルにダミーデータを登録

1, モデルにフィラブルを設定する、フィラブルは変更する可能性のあるカラム名を指定する

```
$fillable = ['column1', 'column4', 'column5', ];
```

2, factory を作成する。factory は database/factory/ 下に作られる

```
$ php artisan make:factory Test
```

3, factory 内でダミーデータの定義を初期サンプルや ↓を参考に更新する

<https://qiita.com/tosite0345/items/1d47961947a6770053af>

4, ↓コマンドで seeder ファイルを作成する

```
$ php artisan make:seeder TestSeeder
```

5, seed の実行

```
$ php artisan db:seed
```

6, やり直し

```
$ php artisan migrate:refresh --seed
```

7, マイグレーションファイルを指定して実行 (元のマイグレーションファイルが残っている場合)

```
artisan migrate:refresh --step=1 --path=/database/migrations/マイグレーションファイル名.php
```

※Factory でダミーデータの定義をし、seeder ファイルのメソッドを用いて、migrate したテーブルにダミーデータを登録する

※上手く行かなかったら、factory、migration、model の関連性をイメージする

※ factory はデフォルトで ID、created_at、updated_at に値を渡すようになっているので、設定を変えない限りはテーブルに最低この3つのカラムを用意する必要がある

5 , ログイン機能を作る

1 , ルーティング

```
-- routes/web.php --
use App\Http\Controllers\Auth\AuthController
Routes::get('/', [AuthController::class,
'login'])->name('login')
```

2 , コントローラーの作成

```
$ php artisan make:controller Auth/AuthController
```

3 , 作成したコントローラーにページ (ここでは login ページ) を返してもらう

```
-- Auth/AuthController --
return view('login.login_form')
```

4 , resources/views フォルダ下の view ファイルをカスタマイズする

5 , bootstrap を利用するのであれば "npm run dev" と打つ

6 , view ファイル内の {{ asset('js/app.js') }} / {{ asset('css/app.css') }} でベースを読み込み

7 , ↓のbootstrap のサンプルを使い効率化を狙うのもアリ

<https://getbootstrap.jp/docs/4.3/examples/>

8 , 「bootstrap-〇〇/site/docs/〇〇/examples/」にサンプルデータがある

※リンクのパスの設定に注意する

バリデーション

1 , フォームリクエスト作成

```
$ php artisan make:request LoginFormRequest
```

2 , \Http\Request\ に作成されたファイルにバリデーションのルールを決める

```
'title' => 'required|max:255',
```

3 , 先程作成したコントローラーにて、use する

```
-- AuthController.php --
use App\Http\Requests\LoginFormRequest
```

4 , バリデーションのエラーメッセージを日本語に変更する

<https://readouble.com/laravel/8.x/ja/validation-php.html>

バリデーション (通して良いデータか否かの判断) を行うためには、Illuminate\Http\Request オブジェクトによって提供される validate メソッドを使用する必要がある

