参考

https://www.hypertextcandy.com/laravel-tutorial-introduction

環境設定

docker で環境作成のサンプル

基本 (Lavel9)

ルーティング

routes ディレクトリ配下にファイルを置く。
URI と挙動を関連付ける。コントローラ以外にもビューや文字列をそのまま返すこともできる

```
Route::get('/', 'App\Http\Controllers\HomeController@index')
Route::post('login', [AuthenticatedSessionController::class, 'store']);
Route::view('/hoge', 'dashboard');
Route::get('/hoge2', function () {
    return "hoge2";
});
```

URI から ID 等の引数を受け取るときは以下のようにかける

```
Route::get('/folders/{id}/tasks/{task_id}/edit', 'App\Http\Controllers\TaskController@showEditForm');
```

またルートに名前をつけて、そのルートへの URI の生成を容易にすることができる。

```
Route::get('/folders/{id}/tasks/{task_id}/edit', 'App\Http\Controllers\TaskController@showEditForm')->name('tasks.edit');
```

ビューで URI を生成するときは

```
<a href="{{ route('tasks.edit', ['id' => $task->folder_id, 'task_id' => $task->id]) }}">編集</a>
```

みたいな感じ。

コントローラ

ルートの引数の順番で引数が渡される。
$request については、どこに定義しても正しく渡される。

```
public function edit(int $id, int $task_id, EditTask $request)
{
    $task = Task::find($task_id);


    $task->title = $request->title;
    $task->status = $request->status;
    $task->due_date = $request->due_date;
    $task->save();


    return redirect()->route('tasks.index', [
    'id' => $task->folder_id,
    ]);
}
```

■ Requests

入力チェックに関して、いくつか方法があり、以下のようにコントローラで直接バリデーションを指定できる

```
$request->validate([
  'name' => ['required', 'string', 'max:255'],
  'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
  'password' => ['required', 'confirmed', Rules\Password::defaults()],
  ], [
  'required' => ':attribute は必須。メッセージ変更可能。',
  ],[
  'name' => 'ユーザー名',
  'email' => 'メールアドレス',
  'password' => 'パスワード',
  ]);
```

また、Request を作成しコントローラの引数として取得することもできる。
コントローラの引数

```
public function edit(int $id, int $task_id, EditTask $request)
```

EditTask

```
<?php
  namespace App\Http\Requests;

  use App\Models\Task;
  use Illuminate\Validation\Rule;

class EditTask extends CreateTask
{
    public function rules()
    {
        $rule = parent::rules();

        $status_rule = Rule::in(array_keys(Task::STATUS));

        return $rule + [
        'status' => 'required|' . $status_rule,
        ];
    }

    public function attributes()
    {
        $attributes = parent::attributes();

        return $attributes + [
        'status' => '状態',
        ];
    }

    public function messages()
    {
        $messages = parent::messages();

        $status_labels = array_map(function ($item) {
            return $item['label'];
        }, Task::STATUS);

        $status_labels = implode('、', $status_labels);

        return $messages + [
        'status.in' => ':attribute には ' . $status_labels. ' のいずれかを指定してください。',
        ];
    }
}
```

ビュー

■サンプル

```
<!DOCTYPE html>
<html lang="ja">

  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>ToDo App</title>
    @yield('styles')
    <link rel="stylesheet" href="/css/styles.css">
  </head>

  <body>
    <header>
      <nav class="my-navbar">
        <a class="my-navbar-brand" href="/">ToDo App</a>
        <div class="my-navbar-control">
          @if (Auth::check())
            <span class="my-navbar-item">ようこそ , {{ Auth::user()->name }} さん </span>
            |
            <a href="#" id="logout" class="my-navbar-item">ログアウト </a>
              <form id="logout-form" action="{{ route('logout') }}" method="POST" style="display:
none;">
                @csrf
              </form>
          @else
            <a class="my-navbar-item" href="{{ route('my-login') }}">ログイン </a>
            |
            <a class="my-navbar-item" href="{{ route('my-register') }}">会員登録 </a>
          @endif
        </div>
      </nav>
    </header>
    <main>
      @yield('content')
    </main>
    @if (Auth::check())
      <script>
        document.getElementById('logout').addEventListener('click', function(event) {
            event.preventDefault();
            document.getElementById('logout-form').submit();
        });
      </script>
    @endif
    @yield('scripts')
  </body>

</html>
```

■ディレクティブ

PHP の変数を表示したり、条件分岐したり、レイアウト定義したり、インポートしたり、必要なことはできる。

モデル

■ getXXXXXAttribute

DB の日付をフォーマットして表示したい等でモデルの項目を編集して表示できる

```
public function getFormattedDueDateAttribute()
  {
      return Carbon::createFromFormat('Y-m-d', $this->attributes['due_date'])
          ->format('Y/m/d');
  }
```

■リレーション

```
class Folder extends Model
{
```

```
    use HasFactory;
    public function tasks()
    {
        return $this->hasMany('App\Models\Task');
    }
}
```

とすると関連するテーブルを取得できる。N+1 問題に注意

ミドルウェア

認証済のユーザのみ表示とか、ログインしていないユーザのみ表示とか、リクエストの前後に任意の処理を入れることができる。

```
Kernel.php
```

でミドルウェアの登録し、ルートを指定するときにミドルウェアを指定する。

```
Route::middleware('guest')->group(function () {
  Route::get('register', [RegisteredUserController::class, 'create'])
            ->name('register');
  (省略)
});
Route::middleware('auth')->group(function () {
  Route::get('verify-email', [EmailVerificationPromptController::class, '__invoke'])
            ->name('verification.notice');
  (省略)
});
```

ミドルウェアの指定方法はいくつか書き方がある。

イベント

イベント発生させて、イベントリスナーにイベントを通知することで、処理のつながりを粗結合にする。
例えば、ユーザ登録したときにメールを送信する等。
イベントとリスナーを作成し、EventServiceProvider にイベントとリスナーを関連付ける。

```
protected $listen = [
      Registered::class => [
          SendEmailVerificationNotification::class,
          TestListener::class,
      ],
  ];
```

以下のようにイベントを発生させると、上記の場合は

```
SendEmailVerificationNotification::class
TestListener::class
```

にイベントが通知される。

その他

ログ

■出力先変更

.env の LOG_CHANNEL で変更できる。
標準エラーに出力するには

```
LOG_CHANNEL=stderr
```

とする。

出力方法

```
$message = ['a' => 1, 'b' => 2, 'c' => 3];
logger($message);
info($message);
```

ルーティング

■ CakePHP のようなルーティング

https://stackoverflow.com/questions/35624907/laravel-5-2-routing-like-cakephp
CakePHP のように URL とコントローラを命名規則で結びつける方法。

web.php に追記

```
Route::any('{anyRoute}', function($anyRoute){
    $call = "";
    $parts = explode("/", $anyRoute);
    $size = sizeof($parts);

    if($size > 0){
        $controller = ucfirst(strtolower(trim($parts[0])));
        $action = trim(array_get($parts, 1));
        $params = [];

        if(empty($controller)){
            return view("welcome");
        }
        else{
            if(empty($action)){
                $action = "index";
            }
        }

        if($size > 2){
            unset($parts[0], $parts[1]);
            $params = array_merge($params, $parts);
        }

        $object = app('App\\Http\\Controllers\\'.$controller.'Controller');
        return call_user_func_array([$object, $action], $params);
    }

})->where('anyRoute', '(.*)');
```

http://hgoehoge/test/index にアクセスすると TestController の index が実行される。

vite.config.js

■ vite server のホスト名を変更する

vite server を使って開発する場合、 public/host が自動的に生成され、記載されている vite server のホスト名、ポートで通信が行われる。
もし、このホスト名を変更したい場合は以下のように vite.config.js を記載する

```
import { defineConfig } from 'vite';
import laravel from 'laravel-vite-plugin';

export default defineConfig({
    plugins: [
      laravel({
          input: [
            'resources/css/app.css',
            'resources/js/app.js',
          ],
          refresh: true,
      }),
    ],
    server: {
      host: true,
      hmr: {
        host: 'hogehoge', <- ここに変更したいホスト名を指定する
      }
    },
});
```