



Laravel9 その1

An abstract graphic on the left side of the slide, featuring vibrant, overlapping brushstrokes in shades of yellow, orange, red, pink, and purple, creating a dynamic and artistic background.

Laravelインストール

ターミナルの使い方

コマンドプロンプト(win) / ターミナル(mac)

cd で移動 cd ../ で上のフォルダに移動

lsでフォルダ内確認(mac) winなら dir

pwdで現在地確認(mac) winなら cd

Laravelインストール

```
cd /Applications/MAMP/htdocs/laravel
```

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

```
laravel:^9 task_test —prefer-dist
```

Laravel起動確認

```
cd task_list
```

```
php artisan serve ・ ・ 簡易サーバー
```

```
Ctrl +C / Cmd + C でサーバー停止
```



Laravel初期設定

Laravel初期設定



タイムゾーン

言語設定

デバググバー

データベース設定

エラーメッセージの日本語化->後で

タイムゾーン、言語設定

タイムゾーン / 言語設定

config/app.php

‘timezone’ => ‘Asia/Tokyo’;

‘locale’ => ‘ja’;

デバグバー



デバグバーのインストール

```
composer require barryvdh/laravel-debugbar:^3.7
```

インストールすると composer.jsonに追記される

php artisan serve で確認

.envのAPP_DEBUGで表示切り替えできる


もし表示が消えなかったら

```
php artisan cache:clear
```

```
php artisan config:clear
```

これらのコマンドでキャッシュを消して
再度試してみてください。

データベース設定



phpMyAdminからデータベースとユーザーを作成

DB ・ ・ laravel_task

ユーザー laravel_user

パスワード ・ ・ password123

.env にも追記する

DB_DATABASE, DB_USERNAME, DB_PASSWORD

php artisan migrateでテーブルが作成されたらOK

もしエラーがでたら



原因は様々あり得るので、
下記個人ブログも参考に
原因確認をしてみてください。

https://coinbaby8.com/access_denied.html



Laravelの概要

MVCモデル

Model ・ ・ データベースとやりとり

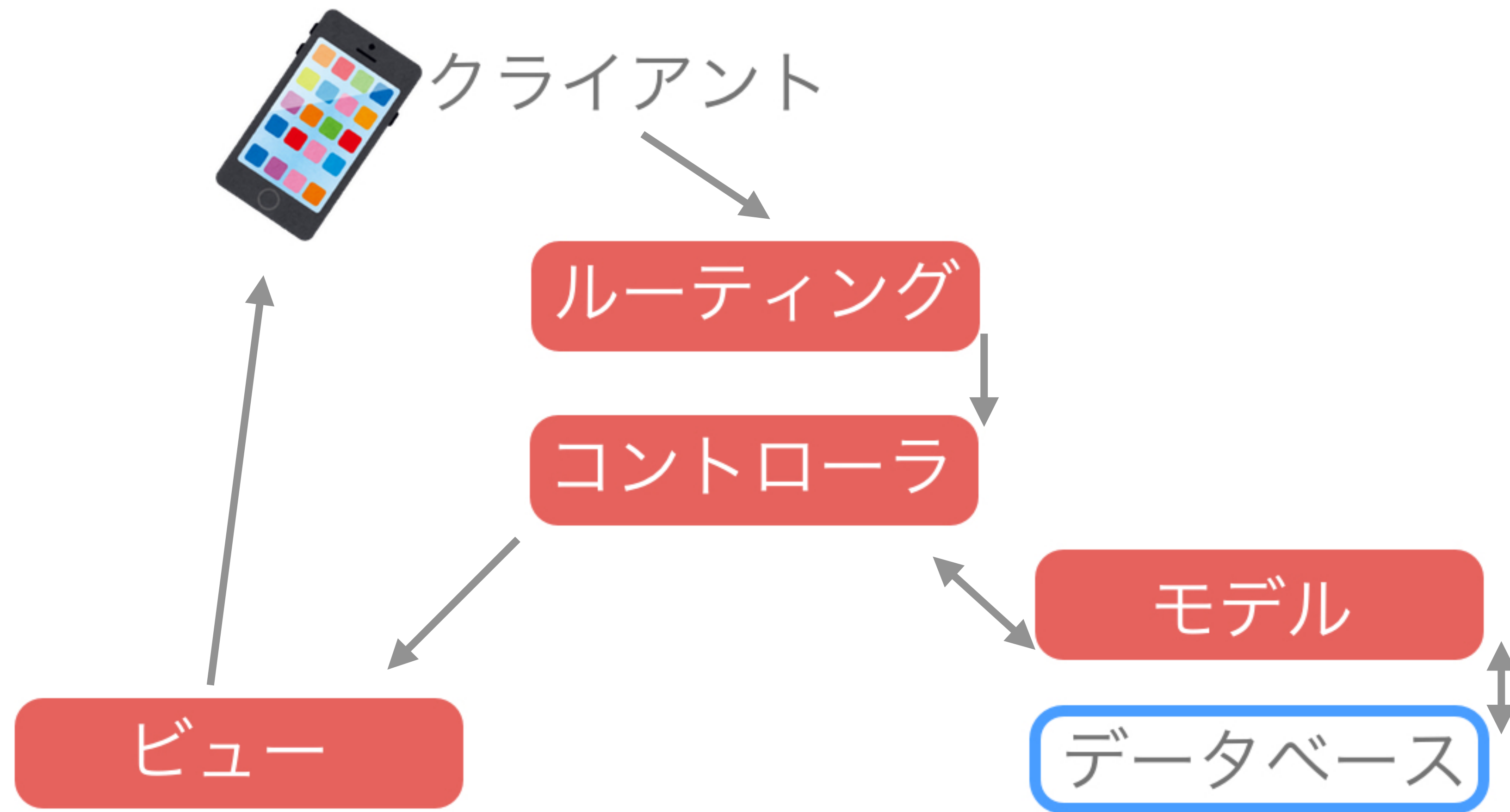
View ・ ・ 見た目

Controller ・ ・ 処理

Routing ・ ・ アクセスの振り分け

Migration ・ ・ DBテーブルの履歴管理

Laravelの構成(ざっくり版)



ルーティング (Routing)

routes/web.php

// Route機能を読み込んでいる

use Illuminate\Support\Facades\Route;

// /にアクセスしたら welcome というビューファイルを表示する

Route::get('/', function () {

return view('welcome');

});

ビュー (View) 見たい目

resources/views/

xxx.blade.php

blade.php とつけるのはお約束

Artisan (職人)



Laravelでよく使う操作やファイル生成を担当

artisanコマンドをリスト表示

```
php artisan list
```


モデル (Model) DBとやりとり

Eloquent(エロクアント)

ORM/ORマッパー

Object-Relational Mapping

オブジェクト関係マッピング

特徴: DBとのやりとりをPHPで書ける

モデル (Model) DBとやりとり

モデルファイルの作成

```
php artisan make:model Test
```

App/Models フォルダ内に生成される

オプションをつけると

コントローラとマイグレーションも同時に生成できる

```
php artisan make:model Test -mc
```


マイグレーション (Migration)

DBテーブルの履歴管理 (テーブルの設定をPHPで書ける)

ファイル場所は databases/migrations

モデルは単数形 マイグレーションは複数形で書くと

Laravel側で自動判定してくれる ex) Person -> people

ファイル作成

```
php artisan make:migration create_test_s_table
```

```
php artisan migrate // DBに反映
```

```
php artisan migrate:fresh // テーブルを全て削除し再生成
```

```
php artisan migrate:refresh // ロールバックして再生成
```

マイグレーション (Migration)

作成できる列の形式はマニュアルを参照ください。

データベース -> マイグレーション

<https://readouble.com/laravel/9.x/ja/migrations.html#column-method-string>

```
public function up()
{
    Schema::create('tests', function (Blueprint $table) {
        $table->id(); // 初期値
        $table->string('text'); // 追記
        $table->timestamps(); // 初期値、created_atとupdated_at
    });
}
```


tinker (DB簡易接続)

対話型 コマンド入力でデータ保存・閲覧できる

```
php artisan tinker
```

```
$test = new App\Models\Test; // インスタンス化
```

```
$test->text = "aaa"; // 設定
```

```
$test->save(); // 保存
```

```
App\Models\Test::all(); // 全件表示
```

exit でコマンド入力画面から抜けられる

コントローラー (処理)

php artisan make:controller

TestController

app/Http/Controllers/

配下に生成される

ルーティング->コントローラ->ビュー

routes/web.php

```
use App\Http\Controllers\TestController; // ファイル内で使えるようにする
Route::get('tests/test', [TestController::class, 'index']); // 配列で書く
```

App/Http/Controllers/TestController.php

```
public function index()
{
    return view('tests.test'); // viewはLaravelのヘルパ関数 フォルダ名.ファイル名
}
```

resources/views/tests/test.blade.php // ファイル名.blade.phpと書く

test

コントローラ内でモデルを取得

App/Http/Controllers/TestController.php

```
use App\Models\Test; // Testモデルを使えるように読み込む

public function index()
{
    $values = Test::all(); // 全件取得

    // dd($values); // die + var_dump 処理を止めて内容を確認できる

    //compact関数でView側に変数を渡すと楽
    // 変数が複数あってもコンマでつなげば複数渡せる
    return view('tests.test', compact('values'));
}
```


コントローラからビューへ

resources/views/tests/test.blade.php

// 複数形 as 単数系 と書くとわかりやすい


```
@foreach($values as $value)
```

```
    {{ $value->id }} <br>
```

```
    {{ $value->text }} <br>
```

```
@endforeach
```

PHPのおさらい



namespace(名前空間)

use構文

class 中の関数・・・メソッド


class中の変数(定数)・・・プロパティ

Laravelのデータ型

配列(Array) ・ ・ 複数の値


コレクション型(Collection)

->配列のラッパー。Laravelオリジナル



ヘルパ関数

Laravelが用意している便利関数



マニュアル: より深く知る->ヘルパ

<https://readouble.com/laravel/9.x/ja/helpers.html>

その中でも特によく使う

route, to_route, url, auth, app, bcrypt

collect, dd, env, config, redirect,

factory, old, view, withなど



エロクアントと コレクション

DBからデータを取得する

DBから情報を取得する方法は大きく2つ

1. Eloquent(エロクアント)

```
use App\Models\Test;
```

```
$tests = Test::all(); // モデル名::メソッド
```

```
dd($tests); // コレクション型(配列を拡張した型)
```

2. クエリビルダ

```
DB::table('tests')->get(); // DBファサード
```


コレクション型

コレクション型専用のメソッド多数
メソッドチェーンで記述可能

より深く知る->コレクション

<https://readouble.com/laravel/9.x/ja/collections.html>

よく使うのはall (全件), chunk (小分け), count (集計), each (全件に処理), first, firstOrFail(最初の一件がなければエラー表示),
get(取得), where系(検索) など

データ型のあれこれ

allやgetを使う事でコレクション型になる

(getをつけないとQueryBuilder途中(確定していない))

```
$values = Test::all(); // Eloquent/Collection
```

```
$count = Test::count(); // 数字
```

```
$first = Test::findOrFail(1); // インスタンス
```

```
$whereBBB = Test::where('text', '=', 'bbb'); // Eloquent/Builder
```

```
$whereBBB = Test::where('text', '=', 'bbb')->get(); // Collection
```

```
dd($values, $count, $first, $whereBBB);
```



クエリビルダ

クエリをPHPでかける

Select, where, groupby, orderByなどSQLに近い構文
rawで生のSQLもかける(その場合はSQLインジェクションに注意)
getやfirstで確定、確定しなければQueryBuilder型
<https://readouble.com/laravel/9.x/ja/queries.html>

```
use Illuminate\Support\Facades\DB;
```


```
DB::table('tests')->where('text', '=', 'bbb')->select('id', 'text')->get(); // コレク  
ション型
```

```
DB::table('tests')->where('text', '=', 'bbb')->select('id', 'text'); // QueryBuilder
```


エロクアント vs クエリビルダ

速度的には多少クエリビルダが早いけれど
基本的にエロクアントを優先的に使う方が
メリットが多い

リレーション(複数テーブルの連携)、
スコープ(クエリの分割)などの機能を使えるため



ファサード

Facade フランス語で正面入り口

デザインパターンの用語にも使われている

複雑な関連を持つクラス群を簡単に使うための窓口

<https://shimooka.hateblo.jp/entry/20141215/1418620292>

よく使うファサード Auth(認証), DB(クエリビルダ)、Hash(暗号化), Gate(認可),
Log(ログ)、Mail(メール)、Route(ルーティング)、
Storage(ストレージ)

ファサードの設定は config/app.phpのalias

vendor/laravel/framework/src/Illuminate/Support/Facades/Facades.php の
defaultAliases



Laravel起動処理

DIと

サービスコンテナ

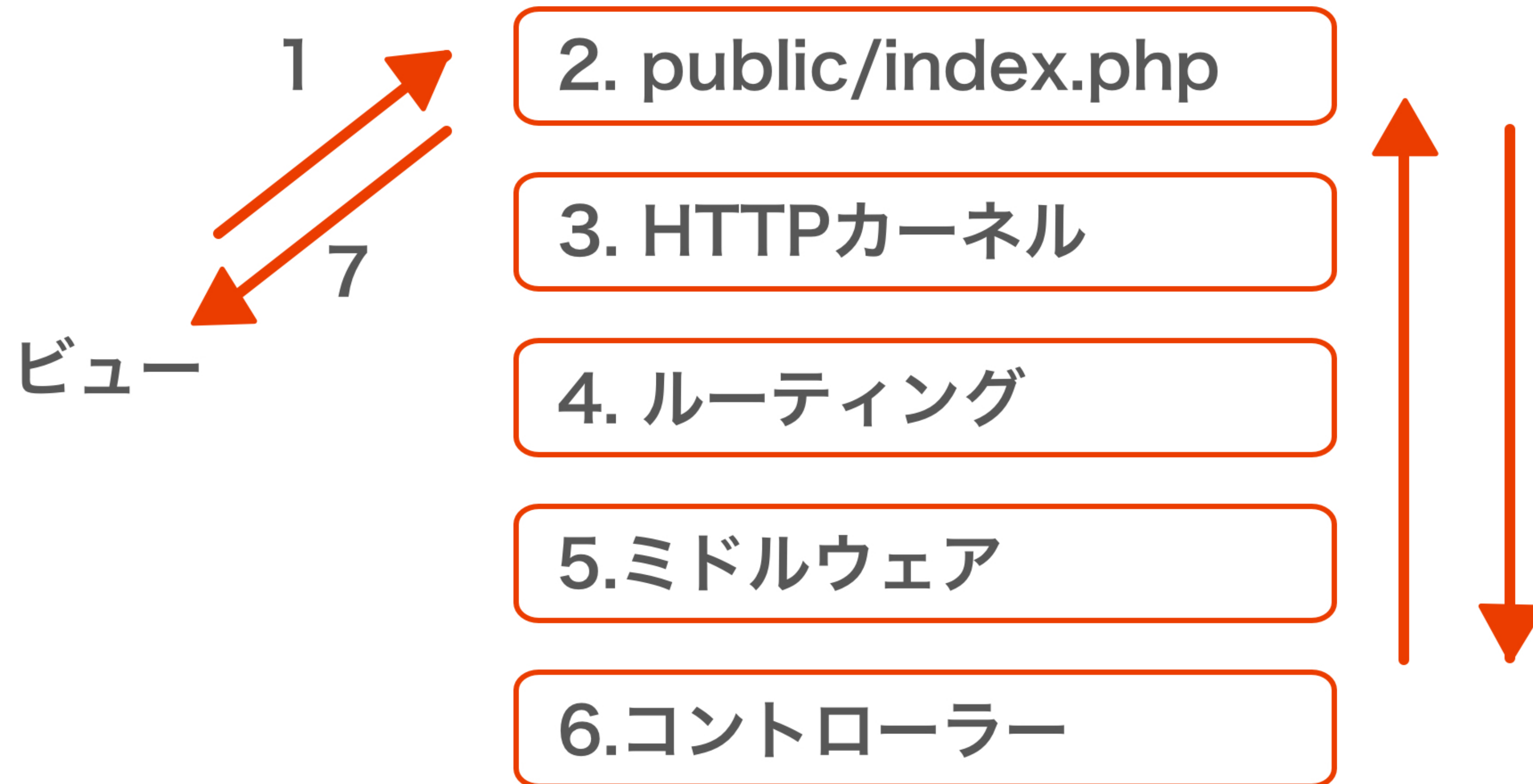
DIとサービスコンテナ



DI・・・外部でインスタンス化して注入
サービスコンテナ・・・DIをまとめて担う

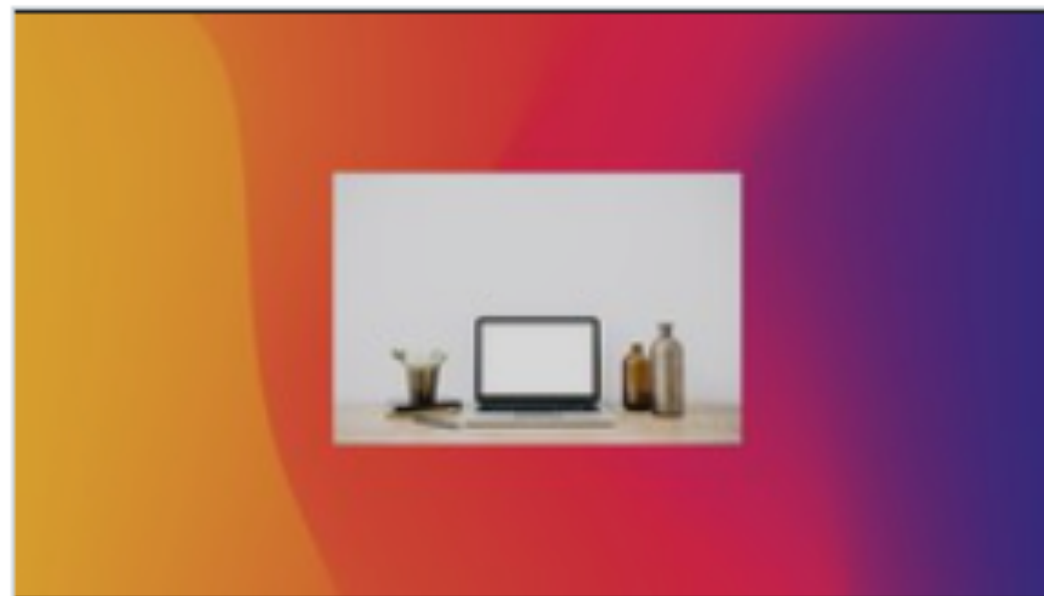
[https://qiita.com/namizato/items/
801da1d03dc322fad70c](https://qiita.com/namizato/items/801da1d03dc322fad70c)

Laravel起動・表示の流れ



ライフサイクル・サービスコンテナ

より詳細に関しては
第2弾の講座も参考にしてください。



【Laravel】マルチログイン機能を構築し本格的なECサイトをつくってみよう 【Breeze/tailwindcss】

Laravelが搭載している認証機能を活用し、管理者、オーナー、ユーザーと3つのログイン情報を持たせ、本格的なECサイトをつくってみよう。Bladeコンポーネント, Stripe決済, 画像アップロードなど実戦形式でたっぷり解説してます。

世界のアオキ (Akihiro Aoki)

4.6 ★★★★★ (372)

合計21.5時間・レクチャーの数: 186・中級



Blade

Blade テンプレートエンジン

<https://readouble.com/laravel/9.x/ja/blade.html>

resources/viewsの中に .blade.php として保存する

{{ }} でエスケープ処理して表示

よく使う構文

@csrf (CSRF対策) @foreach (配列表示)

@if(条件判定) @auth(認証) @isset(設定されているか)

@for(繰り返し) @php (生のphp) @error

@section @yieldなどでテンプレート継承

View側はいろんなパターン

View側はいろんなパターンがあります



【Laravel】イベント予約システムをつくってみよう【Jetstream x...

ライブ ¥10,000 - 公開

Livewire ・ ・ PHPで動的に書ける第3弾

Inertial (Vue.js) ・ ・ 第4弾



【Laravel】【Vue.js3】で【CRM(顧客管理システム)】をつくってみよう...


ライブ ¥11,000 - 公開

Blade(Component) 第2弾



【Laravel】マルチログイン機能を構築し本格的なECサイトをつくってみよう...

ライブ ¥13,000 - 公開



フロントエンド (FrontEnd)

フロントエンドとバックエンド

クライアントサイドとサーバーサイド



クライアント



サーバー



ニーズを求めた結果・・・

クライアント・・・使いやすい・表示が早い

開発側・・・作りやすい・管理しやすい

どんどんカオスな状態に



クライアント



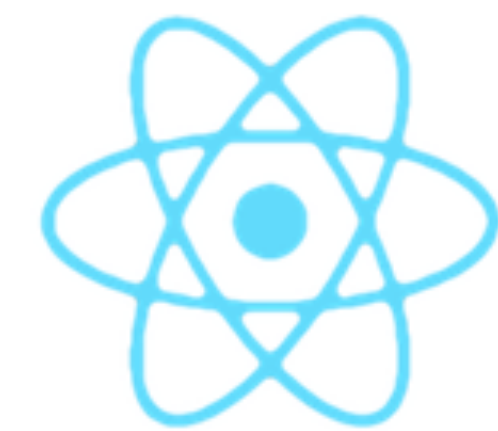
pug



Sass



Vue.js



React

フロントエンド必須ツール



Node.js <https://nodejs.org/ja/>

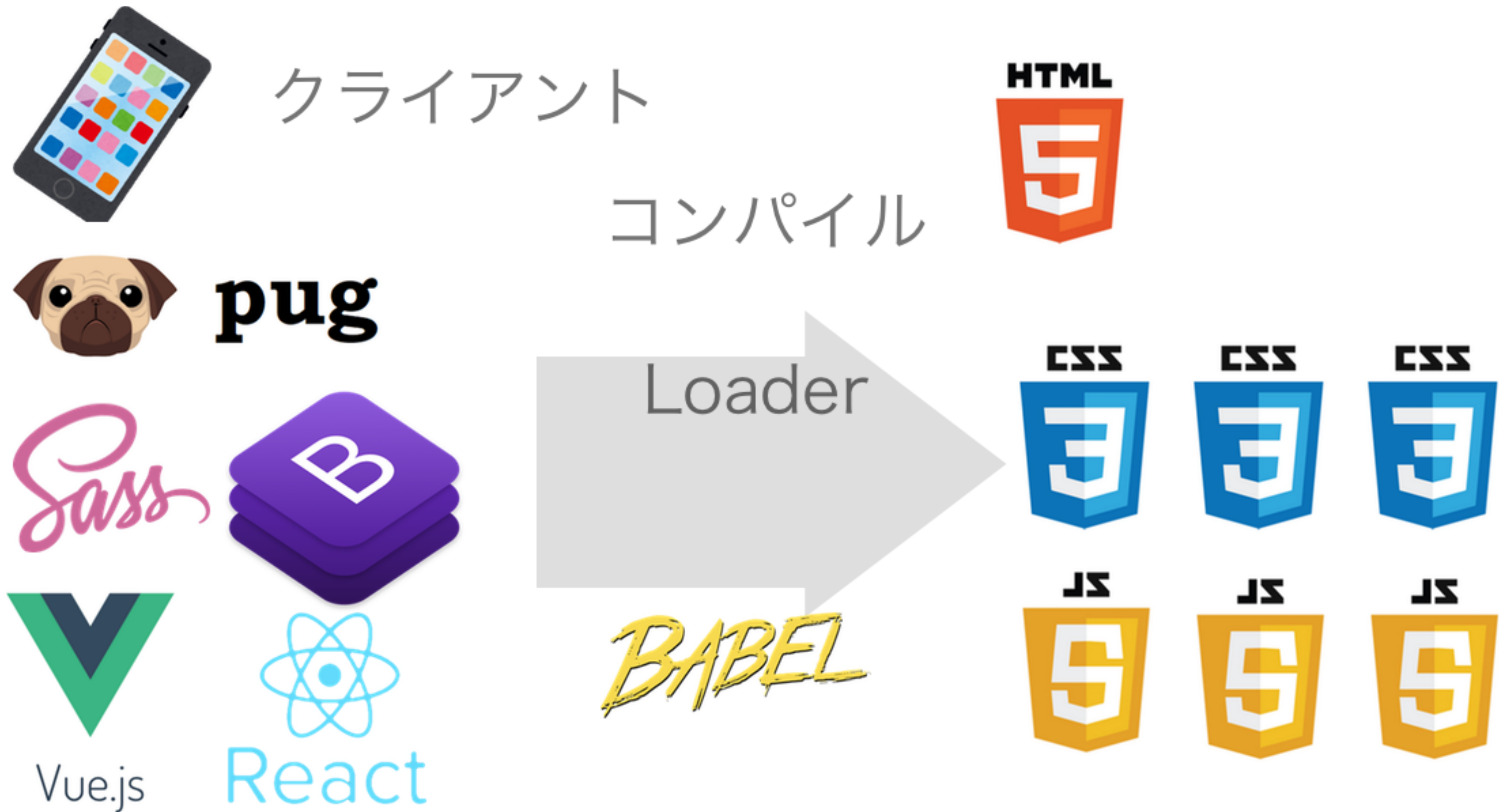
npm (node package manager) パッケージ管理ツール

Vite <https://ja.vitejs.dev/>

webpack <https://webpack.js.org/>

Babel <https://babeljs.io/>

コンパイル(compile) (新->旧)



バンドル(bundle) (複数->1つ)



クライアント

バンドル(bundle)



Vite



webpack



2022/6より Viteが搭載

Laravel搭載時期

2022/6/28 Laravel 9.18 Viteに変更

2017/1 Laravel 5.4 Laravel Mix (webpack)

2015/2 Laravel 5.0 Laravel Elixir (Gulp)

Node.js/npm 別途インストール必要

package.json / package-lock.json 設定管理ファイル

vite.config.js Vite設定ファイル



Laravel Breeze

認証

認証ライブラリ比較

	Laravel / ui	Laravel Breeze	Fortify	Jetstream
Version	6.x～	8.x～	8.x～	8.x～
View (PHP)	Blade	Blade	-	Livewire + Blade
JS	Vue.js / React.js	Alpine.js	-	Inertia.js + Vue.js
CSS	Bootstrap	Tailwindcss	-	Tailwindcss
追加ファイル	View/Controller/Route	View/Controller/Route	-	View/Controller/Route
機能1	ログイン、ユーザー登録、パスワードのリセット、 メール検証、パスワード確認			-
機能2	-	-	2要素認証、 プロフィール管理、チーム 管理	APIサポート (Sanctum)

Laravel Breezeインストール



マニュアル -> スターターキット

<https://readouble.com/laravel/9.x/ja/starter-kits.html>

```
composer require laravel/breeze:^1.13 --dev
```

```
php artisan breeze:install
```

```
php artisan migrate
```

```
npm install
```

```
npm run dev // 開発サーバー起動
```

```
npm run build // 本番用にファイル出力
```


追加・追記ファイル群(抜粋)

composer.json 追記 package.json 追記

App\Models\User.php 追記

resources/css/app.css 追記

resources/js/app.js 追記

下記が追加される

App/Http/Controllers/Auth

App/View/Components

resources/views/auth


resources/views/components

resources/views/layouts

resources/views/dashboard.blade.php

routes/auth.php

ルート情報を表示



php artisan route:list で

現在設定されている全てのルート情報を表示

routes/web.phpの中に

require __DIR__ . '/auth.php'; が追記

auth.phpから

App/Http/Controllers/Auth/配下の
コントローラに振り分けている

エラーメッセージの日本語化

1. マニュアルの言語ファイルをlang/ja/配下にそれぞれ配置する

lang/ja/auth.php, pagination.php, passwords.php, validation.php

2. validation.php の 'attributes' に追記

'attributes' => ['password' => 'パスワード']

3. ja.jsonファイルを作成

lang/ja.json

{

"Whoops! Something went wrong.":"何か問題が発生しました。"

}