Université Paris Nanterre

Université Paris Nanterre

Projet établissement avec programmation web

L3 MIASHS

M. NAFI

m.nafi@parisnanterre.fr

Année: 2023/2024

Objectifs du cours

- Création de sites web dynamiques à l'aide du Framework **Laravel**

- **Prérequis:**
  - Fonctionnement du web
  - Langage **HTML**
  - Langage **CSS**
  - Langage **PHP**

Plan du cours

1. Fonctionnement du web
2. Langage HTML
3. Langage CSS
4. Langage PHP
5. Architecture MVC
6. Framework Laravel

Plan du cours

1. Fonctionnement du web
2. Langage HTML
3. Langage CSS
4. Langage PHP
5. **Architecture MVC**
6. Framework Laravel

Non MCV vs MVC

MVC (Modèle Vue contrôleur)

❑ Non MVC

- Mélange de code HTML, PHP et les données

❑ MVC

- **02** couches: Présentation (HTML ce que l'utilisateur voit) et le code PHP (business logic)

=> Cette séparation permet la *maintenabilité* du code

Modèle MVC

- Le modèle MVC (Modèle-Vue-Contrôleur) est un modèle de **conception** logicielle (application=03 composants)

- Il est souvent utilisé pour le développement d'applications **Web**

- Il permet la séparation de la **présentation**, la **logique** et les **données** de l'application

- Il permet de mieux **organiser**, **faciliter** la maintenance et **améliorer** la réutilisabilité du code

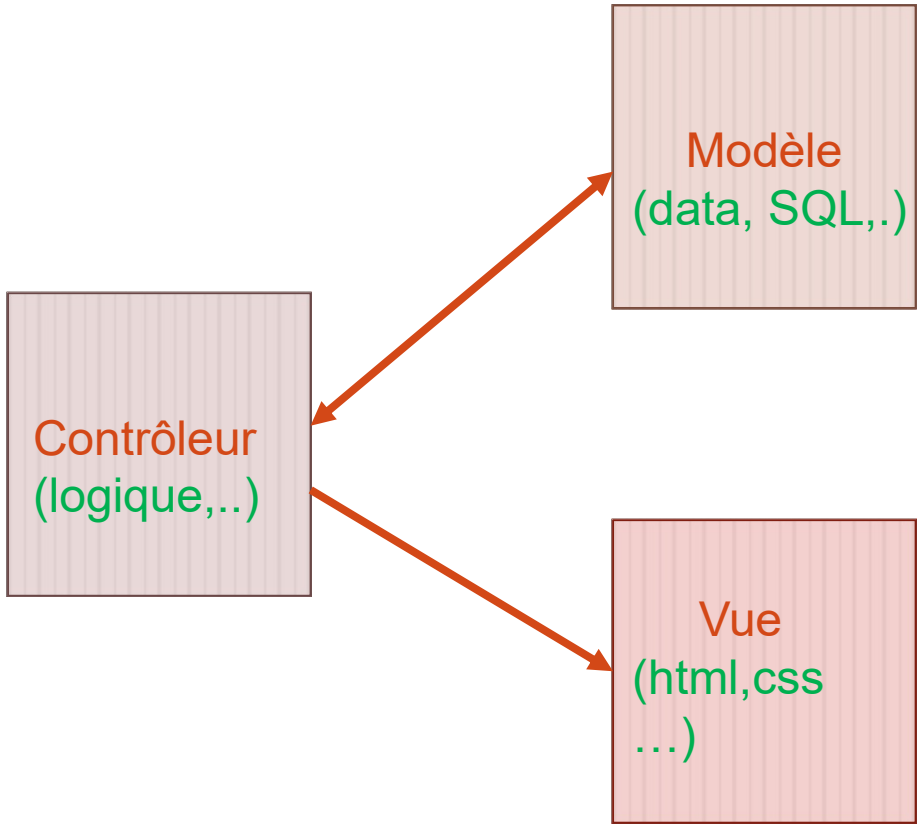
Modèle MVC

- Ce modèle se compose de **trois** parties : Modèle, Vue et Contrôleur
- **Modèle** (Model) gère les **données** et les opérations qui y sont associées. Il peut s'agir d'une *base de données*, d'un *système de fichiers*, d'un *service web* ou de toute autre source de données. On y trouve des requêtes SQL

- **Vue** (View) est responsable de **l'affichage** des données au format HTML, XML ou autre. Elle n'a pas de connaissance directe de la source de données. On y trouve du HTML et un peu de PHP

- **Contrôleur** (Controller) : gère les **interactions** de l'utilisateur avec l'application. Il sert d'intermédiaire entre le modèle et la vue. Il gère la **logique** du code et prend des **décisions**. Il *reçoit* les entrées de l'utilisateur, *effectue* des opérations sur le modèle et *renvoie* une réponse à la vue.

Fig. Architecture MVC



Processus d'échange d'information

1.Le contrôleur demande les données au modèle

2. Le modèle traduit la demande en une requête SQL,
- Récupère les données et les renvoie au contrôleur

3. Le contrôleur transmet les données récupérées à la vue

4. La vue affiche ces données

Exemple site e-commerce

- **Modèle**: une **base de données** contenant les produits, les clients et les commandes, ainsi qu'un ensemble de fonctions pour gérer les interactions avec ces données
- **Vue**: les pages **HTML** générées dynamiquement à partir des données du modèle. Par exemple, une page qui affiche la liste des produits avec leur nom, leur prix et un bouton "Commander"
- **Contrôleur**: Par exemple, lorsqu'un utilisateur clique sur le bouton "Commander ", le contrôleur reçoit la demande, ajoute le produit au panier du client dans le modèle, puis renvoie une réponse à la vue pour afficher la mise à jour du panier

Plan du cours

1. Fonctionnement du web
2. Langage HTML
3. Langage CSS
4. Langage PHP
5. Architecture MVC
6. Framework Laravel

Plan du cours

1. Fonctionnement du web
2. Langage HTML
3. Langage CSS
4. Langage PHP
5. Architecture MVC
6. **Framework Laravel**

Framework Laravel

- Présentation générale
- Routage
- Vue
- Contrôleur
- Modèle

Présentation générale

- Un framework est un ensemble de **bibliothèques**
- Un ensemble cohérent de *composants logiciels* ou *outils* structurels
  - **Pourquoi un framework**
- Eviter de développer à nouveau des composants déjà fait par des gens plus compétents=> gain de temps
- Se concentrer sur la *valeur ajoutée* du produit
- ça donne un **cadre**
- **simplifier** le développement d'applications
- **accélérer** le processus de développement

Frameworks PHP

- Symfony
- Zend framework
- Slim
- Cake PHP
- CoddeIgniter
- **Laravel**

Pourquoi Laravel

- Le concepteur de laravel a **réutilisé** et amélioré le système de routage (gère les requêtes HTTP) de Symfony pour obtenir un système plus *efficace*
- Laravel n'est pas seulement le regroupement des bibliothèques existantes. C'est aussi un ensemble de composants originaux
- Dans laravel, on y trouve entre autres un système de:
  - routage perfectionné
  - template efficace appelé blade
  - Migration de base de données
  - Authentification
  - Gestion de sessions
  - etc

Laravel

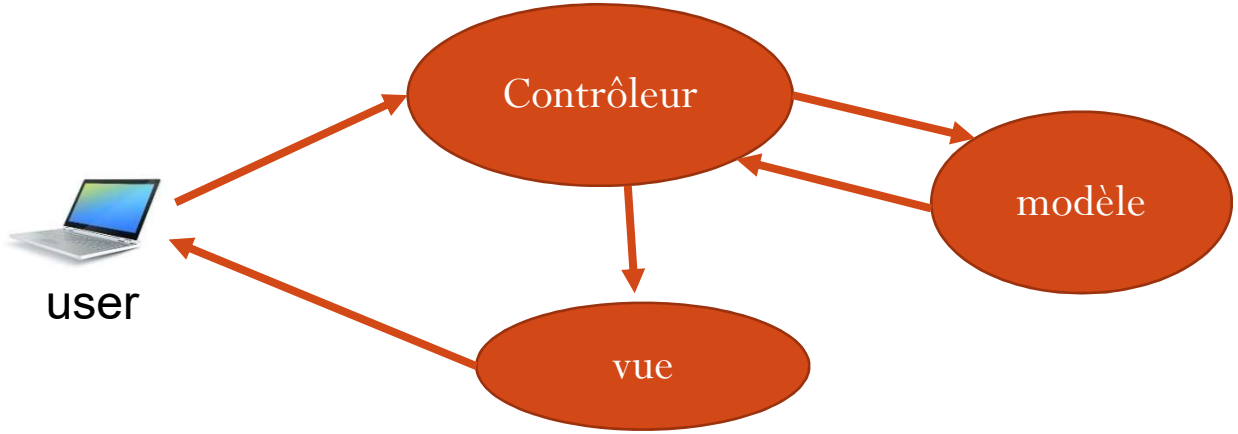
- Laravel a été créé en **2011** par Taylor Otwell
- C'est un framework web, basé sur PHP, et doté d'une syntaxe expressive et élégante
- C'est une collection de codes PHP
- La version actuelle de Laravel est 10.x qui nécessite au minimum la version 8.1 de PHP
- Il fournit une **structure** et un **point de départ** pour le développement d'applications web
- Laravel suit le modèle Modèle-Vue-Contrôleur (MVC) et fournit une variété d'outils et de fonctionnalités qui simplifient et accélèrent le processus de développement

Caractéristiques clés

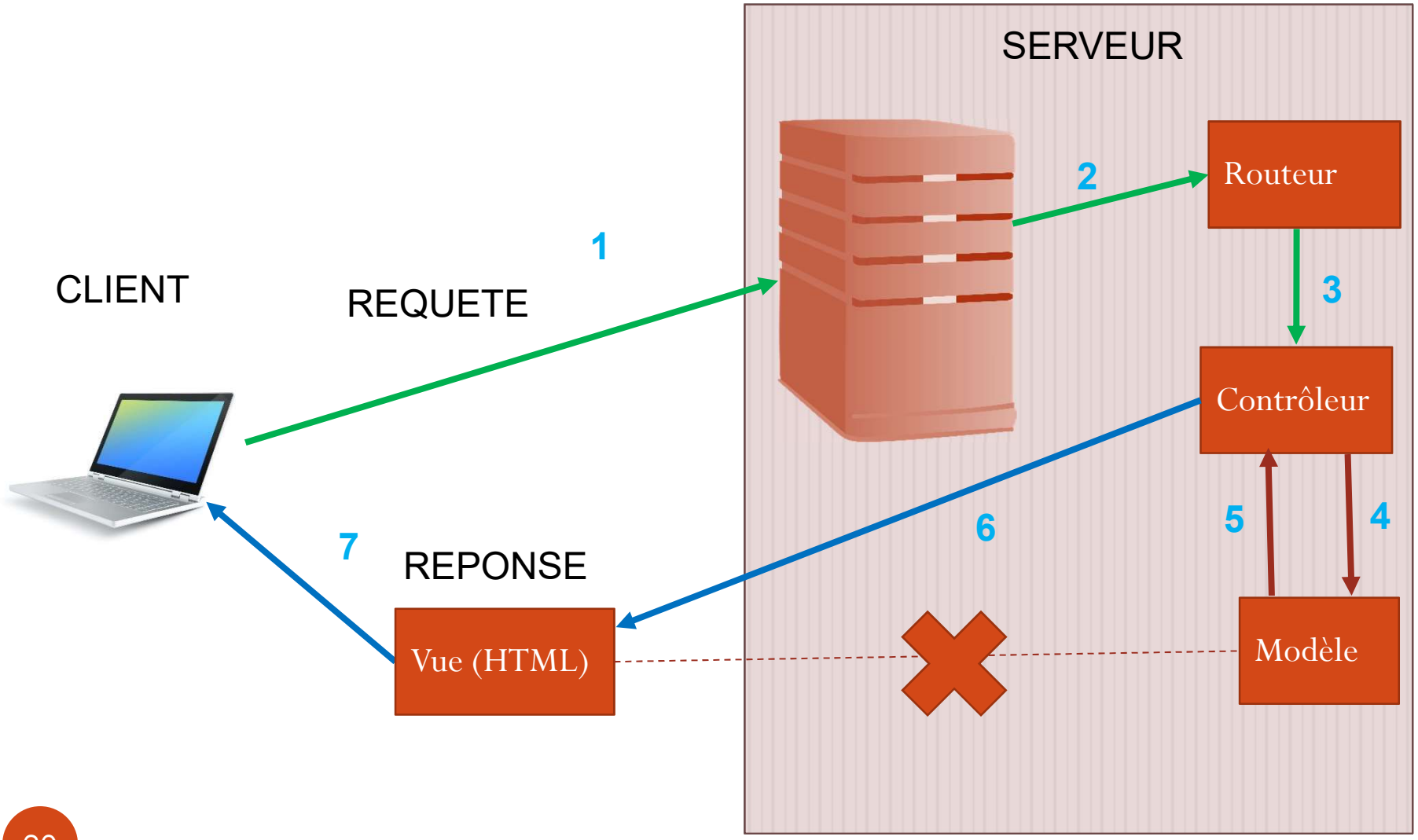
- Open source et gratuit
- Syntaxe simple
- Sécurisé, léger et rapide
- Scalable
- Large communauté active
- Riche documentation
- Certains de ses composants sont issus de Symfony (**30%** environ scr:wikipedia)
- Suit le modèle MVC

Dans laravel,

- **Modèle**: représente une **table** de la base de données (ex.étudiant). C'est une classe qui étend la classe **Model**
- **Vue**: **Template** qui affiche des données aux utilisateurs. fichier avec du code HTML, ou utilisant le système de template **Blade**
- **Contrôleur**: comme un **agent de circulation (intermédiaire)**, prend les requêtes HTTP à partir du navigateur, valide les entrées, récupère les données de la base (modèle), et renvoie des réponses aux utilisateurs (vues).



Modèle Vue Contrôleur



Installation (1/3)

Laravel a besoin d'un environnement complet: Php, Serveur web, MySql, Composer, Nodejs, etc

Il peut être installé de différentes manières: Docker, composer, etc.

1. Environnement de développement(IDE): Il est recommandé d'installer les plateformes Laragon, Wampserver ou XAMPP qui contiennent: l'interpréteur PHP, serveur web Apache, et serveur de base de données MYSQL, etc

- Laragon: https://laragon.org/
- Wampserver: wampserver.org
- XAMPP: apachefriends.org

Installation (2/3)

2. Composer: C'est un outil de gestion de paquets ou gestionnaire des dépendances pour PHP

- Laravel utilise des composants d'autres sources
- composer est utilisé pour installer de façon automatique les bibliothèques ou composants dont un projet a besoin
- Laravel a besoin de 'composer' pour s'installer et gérer ses bibliothèques et ses dépendances
- On peut le télécharger et l'installer depuis le site officiel https://getcomposer.org

Installation (3/3)

3. Editeurs de code: Visual Studio Code propose de très nombreux plugins pour laravel (Laravel Blade Snippets, etc), PhpStorm qui est complet mais pas gratuit, etc.

4. Laravel: une fois 'composer' installé, on peut installer laravel avec la commande:

composer global require laravel/installer

5. Nodejs

- *Il est aussi recommandé d'installer nodejs et npm*

Artisan

- Outil ou une interface en lignes de commandes (CLI) fourni avec Laravel. Il permet d'exécuter des commandes

- La commande php artisan serve permet de lancer le serveur web sur le port **8000**

- Pour se connecter, il suffit de se rendre sur l'adresse:

http://localhost:8000 ou http://127.0.0.1:8000

Création d'un projet Laravel

Il existe plusieurs façons de créer une application avec laravel

- Installer globalement l'installeur de Laravel:

  composer global require laravel/installer

  - créer une application:

  laravel new nom-projet

- Ou avec Composer:(pas besoin d'installer laravel)

  composer create-project laravel/laravel nom-projet

- Démarrer le serveur local en utilisant **artisan**

  - Aller dans le répertoire de votre projet :

    cd nom-projet

  - Démarrer le serveur :

    php artisan serve

  - l'application sera accessible à l'adresse: http://localhost:8000 ou http://127.0.0.1:8000.

Structure d'une application

- Une application contient plusieurs dossiers et fichiers:
  - **App**: tous les fichiers PHP et classes. Il contient des éléments essentiels qui sont regroupés dans les sous dossiers suivants:
    - Console: toutes les commandes
    - Exceptions: Erreurs d'exécution
    - http: communication: contrôleurs, middlewares, kernel
    - Models: modèles
    - Providers: tous les fournisseurs de services. Initialiser les composants.

- **Bootstrap**: scripts lancés par laraval pour s'initialiser (chargement des classes, lancement de l'application)
- **Config**: toutes les configurations : application, authentification, cache, base de données, espaces de noms, etc. (ex l'acces à la BDD sera diffèrent sur l' environnement de développement et l'environnement de production)
- **Database**: migrations et populations
- **Public**: c'est le point d'entrée de l'application. Seul à être accessible pour le client
- **Resources**: ensemble des vues, fichiers CSS et JavaScript
- **Routes**: liste de routes ou URL de l'application (web.php)

Structure d'une application

- **Storage**: données temporaires : logs, le cache, les versions
- **Tests**: fichiers de tests unitaires et tests d'intégration
- **Vendor**: composants et dépendances de Laravel. l'endroit où sont installées les dépendances dont le projet a besoin.

❑ Fichiers de la racine:

- **.env** : spécifier l'environnement d'exécution (local ou de production, nom d'application, etc)
- **artisan** : outil en ligne de Laravel pour des tâches de gestion
- **composer.json** : fichier de référence de composer
- etc

Configuration

- Il existe deux types de configurations:
  - Propre à l'environnement (poste): fichier .env
  - Partagée: dossier config
- Fichier .env : Exemples d'éléments à configurer
  - Nom d'application
  - URL de base
  - Choix d'une BDD (MySql, Sqlite, PostreSql,etc)
  - etc

Routage

- Laravel offre un système de route simple
- Un site web utilise des adresses ou URL
- Les routes web sont définies dans le fichier routes/web.php et les routes api sont définies dans routes/api.php
- Les **routes web** sont celles qui sont visitées par les utilisateurs finaux
- Définir une route revient à lier une URI à un code à exécuter

URI est la partie d'une adresse qui suit le nom du domaine

Ex URL: www.monsite.fr/accueil

URI: /accueil

- Pour créer une route, il faut appeler la classe Route avec la méthode souhaitée (get, post, put, delete, etc)

qui prend deux paramètres: l'URI et retour ou fonction à exécuter

Ex. déclarer une route sur la racine du domaine se fait:

classe | méthode | URI

```
Route :: get  ('/',  function(){        Fonction anonyme(rappel)

        return view('bonjour')          code PHP

        }
```

NB: La fonction doit retourner un résultat ie la dernière instruction est **return**

❑ Définition de routes

Il en existe 02 façons:

❖ Fonctions **anonymes**

- La manière la plus simple pour définir une route est de faire correspondre un **chemin (path)** à une **fonction anonyme (closure)**
- **Closure** est une fonction **anonyme** qui peut être affectée à une variable, passée comme paramètre à une autre fonction, etc.

- Ex. Route::get('/', function(){
  return 'Bonjour';});

Si un visiteur se rend sur la racine du domaine (/), le routeur de laravel exécute la fonction de rappel et retourne la chaine de caractères 'Bonjour'.

- Ex. Route::get('about', function(){
  return view('about');
  });

NB: **return** est utilisé au lieu de **echo** ou **print** ?

À la fin de l'exécution de la fonction, il n'est pas encore temps de retourner le résultat au navigateur, mais il continue son processus de requête/réponse

❖ Contrôleurs:

- Passer le nom du contrôleur et le nom de la méthode au lieu de la fonction anonyme comme une chaine de caractères.
- Ex. Route::get('/','WelcomeControleur@Index');
- Route::get('/', ['WelcomeControleur::class'], 'Index');
- Les requêtes vers ce chemin sont passées à la méthode index du contrôleur WelcomeControleur qui se trouve dans App/Http/Controllers

Méthodes HTTP

Requêtes auxquelles le serveur pourra répondre:

- GET: demande une ressource qui ne change jamais.

  Elle récupère des données à partir du serveur
- POST: crée ou modifie une ressource.

  Elle envoie des données aux serveur
- PUT: ajoute ou écrase (remplace complètement) une ressource existante

- PATCH: met à jour partiellement une ressource sur le serveur
- DELETE: supprime une ressource
- HEAD: Similaire à GET, mais elle récupère seulement l'entête de la réponse.
- ANY: répond à n'importe quelle méthode HTTP
- MATCH: définit plusieurs méthodes pour une même route

```
Ex.Route::match(['get','post'],'/',function(){}
```

Paramètres de routes

- Il est parfois nécessaire de capturer certains **paramètres** de l'URI pour les traiter plus tard
- Déclarer des paramètres **variables** dans l'URI d'une route permet de ne pas créer des routes **spécifiques** une par une
- Pour traiter la variable capturée, il faut la passer en paramètres à la fonction anonyme en lui donnant le même nom.

Ex. Route::get('etudiant/{id}', function ($id){ … }

- Possible de déclarer plusieurs paramètres à la fois

Ex. Route::get('user/{id1}/admin{id2}', function (){ … }

- **Paramètres optionnels** sont introduits avec le symbole **?** à la suite du nom du paramètre
- Nécessité de donner une **valeur par défaut** à la variable passée à la fonction anonyme. Cette valeur peut être: null, false, valeur définie

Ex. Route::get('accueil/{nom?}, function($nom=null){

```
If ($nom===null){
return 'bonjour à tous';}
else{
Return 'bonjour $nom';});
```

Que fait ce code?

Réponse: Dire bonjour à un user si son nom est fourni et saluer tout le monde dans le cas contraire

- **Question**: les noms de paramètres de route et les noms de paramètres de méthode ou fonction doivent-ils être les mêmes ou identiques?
- Réponse: Non
- Ce qui importe c'est leur ordre

Ex. Route::get('user/{id1}/comment{id2}',
function($UserId, $CommentId){
//});

NB: Il est recommandé de garder les mêmes noms

Contrainte de route (Expression régulière)

- Contraindre un paramètre à n'accepter que certaines valeurs
- Elles sont utilisées si une route ne doit correspondre que si un paramètre répond à une exigence particulière

Ex. Route::get('users/{id}', function($id){ //

})->where('id', '[0-9]+');

Ex. Route::get('users/{nom}', function($nom){ //

})->where('nom', '[A-Za-z]+');

- Si l'utilisateur visite un chemin qui correspond à une route mais le paramètre ne respecte pas l'expression régulière, alors ça retournera une erreur 404 (Not Found).

Exemple

- Routes spécifiques:
  - Route::get('1', function(){return 'le visiteur N° 1'});
  - Route::get('2', function(){return 'le visiteur N° 2'});
  - Route::get('3', function(){return 'le visiteur N° 3'});
- Problème: Que va-t-il se passer si on tape l'url: www.exemple.fr/4?
- Réponse: 404 Not Found
- Solution: définir une route paramétrée
  - Route::get('{n}', function($n){return 'le visiteur N°' .$n});
- Que va-t-il se passer si on tape l'url: www.exemple.fr/4?
- Réponse: 'le visiteur N° 4'

- Problème: Que va-t-il se passer si on tape l'url: www.exemple.fr/Jean?
- Réponse: 'le visiteur N° Jean'=> chaine de caractères
- Solution: Contrainte de route (expressions régulières)
- Définir une route paramétrée avec contrainte:
  - Route::get('{n}', function($n){return 'le visiteur N°' .$n})->where('n', '[0-9]+');

Nommage de route

- Laravel permet de nommer les routes pour leur donner des alias court afin de les référencer n'importe où
- Ceci est utile car on affecte un nom simple à une route complexe. De plus, on a pas à réécrire nos liens si le chemin change
- On définit les routes avec name() dans routes/web.php
- Ex. Route::get('users/{id}', UserController@afficher')->name('users.afficher');
- On lie la route dans une vue en utilisant route() helper
- <a href='<?php echo route('user.afficher', ['id'=>1])?;>'>

- Convention de nommage: pluriel du nom de la ressource, période puis l'action
- Ex. users.créer, users.modifier, users.supprimer
- une route est nommée avec la méthode name()
- Ex. Route::get('/', **function**() {

  return 'Je suis sur la page d\'accueil !';

  })->name('home')
- pour générer l'url qui correspond à une route, on peut utiliser le helper route()

  route('home')

=> Ceci va générer l'url de base du site web

Ordre des routes

- Les routes sont analysées dans leur ordre dans le fichier des routes
- Ex. Route::get('{n}', **function**($n) {

```
    return 'Je suis sur la page ' . $n . ' !';
});
Route::get('contact', function() {
return "Voici mon contact.";
});
```

- Que va-t-il se passer si on tape: http://exemple.fr/contact ?
- Réponse:

Groupage des routes

- Parfois, un ensemble de routes partagent les même caractéristiques (authentification pour MAJ un user, préfixe de chemin, etc)
- Eviter de redéfinir ses caractéristiques plusieurs fois ie dans chaque route
- Regrouper plusieurs routes et leur appliquer en une seule fois une même configuration. Ceci évitera la redondance

Ex. Route::**group**(function(){

```
Route::get('hello', function(){
    return 'hello';});
Route::get('world', function(){
    return 'world';});
```

});

- **Exemple** d'utilisation de groupe de routes: appliquer un middleware au groupe de routes

```
Ex. Route::middleware('auth')->group(function(
    Route::get('compte', function(
        return view('compte'); });
    Route::get('admin', function(
        return view('administrateur'); });
});
```

- Restreindre un ensemble de routes aux utilisateurs authentifiés

Préfixe de chemin

- Un ensemble de routes qui partagent un même segment de leur chemin

Ex. Route::prefix('site')->groupe(function(

```
route::get('images', function(
//    /site.images
route::get('videos', function(
//    /site.videos
});
```

Réponses

- Automatiques:

Ex. Route::get('test', **function** () {

return 'un test';

});

- Personnalisées:

- utiliser la classe response de Laravel pour construire une réponse

Ex. Route::get('test', **function** () {

return response('un test', 206)->header('Content-Type', 'text/plain');

});

- Vues