

Introduction à



Objectif général

Prendre en main l'un des Framework PHP les plus utilisés.

Objectifs spécifiques

- Faire correspondre une URL donnée à un traitement précis grâce au routage
- Regrouper des traitements connexes grâce aux contrôleurs
- Récupérer les données d'une requête http grâce à Request
- Retourner des contenus aux formats texte, HTML, JSON, etc. grâce à Response
- Intégrer des données dans des templates grâce à Blade
- Interagir avec l'utilisateur grâce aux formulaires
- Faciliter la communication avec une base de données grâce à Eloquent
- Filtrer les requêtes grâce aux middlewares

Sommaire

- 1) Préliminaires
- 2) Routage
- 3) Contrôleurs
- 4) Requêtes
- 5) Réponses
- 6) Vues
- 7) BDD
- 8) Middlewares

Préliminaires

sommaire

- 1) Qu'est ce qu'un Framework ?
- 2) Qu'est ce que Laravel ?
- 3) Prérequis pour installer Laravel
- 4) Installation de Laravel sous Windows avec Composer
- 5) Installation de Laravel sous Mac et Linux avec Composer

Préliminaires

Qu'est ce qu'un Framework ?

- **Problématiques :**

Comment développer efficacement (rapidité, bonnes pratiques, ...) une application ?

- **Solution :**

Ensemble de composants et de préconisations « prêt à l'emploi » = Framework

- **Avantages d'un Framework**

- Structuration du code (modèle MVC)
- Abstraction de la base de données
- Réutilisation de composants éprouvés et approuvés (Email, Users, ...)
- Instauration de bonnes pratiques de codage
- Facilitation de la maintenance et de l'évolution du code
- Facilitation du travail en équipe
- Forte communauté (support et mises à jour)

Préliminaires

Qu'est ce que Laravel ?

- Laravel est un des Framework PHP les plus utilisés
- Créé par Taylor Otwell en juin 2011
- Dernière version : 9 (8 Février 2022)
- Projet PHP le mieux noté de GitHub en 2016
- Laravel reste basé sur Symfony pour au moins 30 % de ses lignes
- Concurrencé par Symfony, CodeIgniter, Zend Framework, CakePHP, ...

Préliminaires

prérequis pour installer Laravel

- Version de PHP (CLI) ≥ 8
- Activer les extensions PHP suivantes
 - BCMath
 - Ctype
 - JSON
 - Mbstring
 - OpenSSL
 - PDO
 - Tokenizer
 - XML

Préliminaires

Installation de Laravel sous Windows avec Composer

- S'assurer qu'un exécutable PHP est accessible globalement : chemin d'accès présent dans le PATH
- Télécharger (<https://getcomposer.org/Composer-Setup.exe>) et exécuter Composer-Setup.exe
- Installer avec Laravel Installer
`composer global require laravel/installer`
- Créer un nouveau projet Laravel
`laravel new nomDuProjet`
Exemple : créez le projet nommé « `exemples-laravel` »
- Démarrer le projet avec les commandes :
`cd nomDuProjet/` puis `php artisan serve`
- Accéder à la page d'accueil de Laravel à l'adresse <http://localhost:8000/>

Préliminaires

Installation de Laravel sous Mac et Linux

- S'assurer qu'un exécutable PHP est accessible globalement
- Télécharger la dernière version du composer.phar

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

- Rendre globale puis exécutable la commande composer

```
cp chemin/vers/composer.phar /usr/local/bin/composer sudo chmod +x /usr/local/bin/composer
```

- Installer avec Laravel Installer
`composer global require laravel/installer`

- Créer un nouveau projet Laravel
`laravel new nomDuProjet`

Exemple : créez le projet nommé « `exemples-laravel` »

- Démarrer le projet avec les commandes :
`cd nomDuProjet/ puis php artisan serve`

- Accéder à la page d'accueil de Laravel à l'adresse <http://localhost:8000/>

Routage

sommaire

- 1) Présentation
- 2) Définition
- 3) Paramètres de route
- 4) Paramètres optionnels de route
- 5) Route nommée
- 6) Génération d'URL et redirection

Routage

présentation

- Routage = faire correspondre une URL donnée à un traitement précis
- Intérêt : avoir de belles URL pour un bon référencement Web et un confort des visiteurs
Ex : /read/intro-to-laravel au lieu de index.php?article_id=57
- Définition d'une route = motif d'un path d'URL + traitement

Routage

définition

- Emplacement
fichier « route/web.php »
- Syntaxe
`Route::methodHTTP('chemin', closure);`
- Exemple
`Route::get('/', function () { return view('welcome'); });`

Routage

paramètres de route

- Paramètre de route = segment variable du path
- Syntaxes d'ajout du paramètre
 - 1) Suffixer {nomParametre} au path
 - 2) Passer \$nomParametre en paramètre à la fonction de rappel
- Exemple

```
Route::get('/hello/{prenom}', function ($prenom) {return 'Hello '$prenom; });
```
- Tester le chemin /hello/votrePrenom

Routage

paramètres optionnels de route

- Syntaxes d'ajout du paramètre optionnel

1) Suffixer {nomParametre?} au path

2) Passer \$nomParametre=valeurParDefaut en paramètre à la fonction de rappel

- Exemple

```
Route::get('/hello/{prenom?}', function( $prenom = 'world') {return 'Hello '$prenom; });
```

- Tester les chemins

- /hello/VotrePrenom
- /hello

Routage

route nommée

- Les routes nommées permettent la génération d'URL et la redirection
- Pour nommer une route, il suffit de chaîner la méthode `name()` à sa définition
- Exemple

```
Route::get('/hello/{prenom}', function ($prenom) { return 'Bonjour ' . $prenom; }->name('hello');
```


Routage

génération d'URL et redirection

- Syntaxe de la génération d'une URL

`$url = route('nomDeLaRoute');`

- Syntaxes d'une redirection

- `redirect-> route('nomDeLaRoute');`
- `redirect-> route('nomDeLaRoute', ['param'=>valeur, ...]);`
- Dans une vue avec attribut = href ou action:
`attribut="{{ route('nomDeLaRoute',['param' => valeur, ...]) }}"`

- Exemple

1) On rappelle que « hello » est le nom de la route de chemin

« /hello/{prenom} »

2) Dans la route de chemin « / »,

remplacez « `view('welcome')` » par `redirect()->route('hello',['prenom'=>'Redirected User'])`

3) Testez le chemin /

Contrôleurs

sommaire

- 1) Présentation
- 2) Définition
- 3) Exemple

Contrôleurs

présentation

- Contrôleur : classe qui regroupe des méthodes qui agissent sur la même ressource
- Action : fonction ou méthode qui reçoit une requête, la traite et retourne une réponse (texte, HTML, XML, JSON, image, redirection, erreur 404, ...)
- Pour simplifier certain traitement, le contrôleur peut hériter de la classe Controller

Contrôleurs

définition

- Emplacement app/Http/Controllers
- Commande de création
php artisan make:controller nomContrôleur
- Syntaxe de la route
Route::methodHTTP('chemin', 'nomContrôleur@nomAction');

Contrôleurs

exemple

1) Lancez la commande
`php artisan make:controller UserController`

2) À la classe `UserController` ajouter les méthodes

```
public function hello($prenom) {  
    return 'Hello '.$prenom;  
}  
  
public function index() {  
    return redirect()->route('hello',['prenom'=>'Redirected User']);  
}
```

3) Remplacez toutes les routes par

```
Route::get('/hello/{prenom}', 'App\Http\Controllers\UserController@hello')->name('hello');  
Route::get('/', 'App\Http\Controllers\UserController@index');
```

4) Testez les routes

Requêtes

sommaire

- 1) Accéder à la requête
- 2) Récupérer les données de la requête
- 3) Exemple

Requêtes

accéder à la requête

- Une requête HTTP est très souvent accompagnée de données comme les propriétés d'une ressource à ajouter ou à modifier.
- le serveur a besoin d'y accéder pour effectuer le traitement adéquat
- Pour accéder à la requête via une fonction (closure ou méthode), il suffit de passer en paramètre une instance de Request
- Syntaxe : ... (Request \$request)
- Tout éventuel autre paramètre de la fonction doit être mis après l'instance de Request
- NB : importer la classe Request use Illuminate\Http\Request;

Requêtes

récupérer les données de la requête

- `$request->all()` et `$request->input()` : récupèrent toutes les données dans un tableau associatif
- `$request->input('nomDonnée')` : récupère une donnée
- `$request->query('nomParamètre')` : récupère une donnée de la query string
- `$request->query()` : récupère toutes les données de la query string dans un tableau associatif
- `$request->nomDonnée` : récupère une donnée dans le corps, dans la query string ou dans le chemin
- `$request->input('nomObjet.nomChamp')` : récupère une donnée JSON

Requêtes

exemple

- 1) Modifiez l'action hello() comme suit :

```
public function hello(Request $request, $prenom){  
    $nom = $request->query('nom');  
    $age = $request->input('age');  
    return "Hello $prenom $nom ! Are you $age old ?";  
}
```

- 2) Ajoutez la route
Route::post('/hello/{prenom}', 'UserController@hello');
- 3) Dans app/Http/Middleware/VerifyCsrfToken.php, ajoutez au tableau \$except, l'URL
'http://127.0.0.1:8000/hello/Abdou?nom=Diop'
- 4) Avec un client REST faites la requête telle que :
 - Méthode = POST
 - URL = <http://localhost:8000/hello/Abdou?nom=Diop>
 - Paramètre de formulaire : age=23

Réponses

sommaire

- 1) Retourner du texte
- 2) Retourner du JSON
 - a) via un tableau
 - b) via la méthode json()
- 3) Retourner une instance de Response
- 4) Retourner une vue
- 5) Rediriger
 - a) Vers une action d'un contrôleur
 - b) Vers un chemin relatif
- 6) Exemples

Réponses

retourner ou rediriger

- `return 'texte à retourner'`
retourne du texte au format text/html
- `return tableau` ou `return response()->json(tableau)` retourne du contenu au format JSON
- `response($contenu, $statusCode)`
 - >`header('nomEntête1', $valeur)`
 - >`header('nomEntête2', $valeur)`retourner une instance de Response avec contenu, code et entêtes
- `return view('nomVue')` retourne une vue
- `redirect()->action('nomController@nomAction', tabAssParam)`
redirige vers une action avec d'éventuels paramètres
- `redirect('chemin/de/la/page.html')` redirige vers une page

Réponses

exemples

Exemple 1

- 1) Dans l'action hello(), retournez alternativement
 - compact('prenom','nom','age')
 - response()->json(compact('prenom','nom','age'))
 - response("Hello \$prenom \$nom ! Are you \$age old ?", 200)
->header('Content-Type', 'text/plain')
- 2) Testez avec un client REST et vérifiez l'entête « content-type »

Exemple 2

- 1) Dans l'action index(), retournez redirect()->action([UserController::class, 'hello'], ['prenom'=>'Redirected User', 'nom'=>'XX', 'age' => 34])
- 2) Testez le chemin '/'

Vue

sommaire

- 1) Présentation
- 2) Créer une vue
- 3) Retourner une vue via une route
- 4) Retourner une vue via une fonction
- 5) Passer des données à une vue
- 6) Blade
 - a) Présentation
 - b) Expressions
 - c) Structures conditionnelles
 - d) Structures itératives
 - e) Héritage d'un template
 - f) Inclusion d'une vue
 - g) Formulaire

Vue présentation

- Une vue gère la disposition générale des éléments de l'interface ainsi que l'aspect visuel de cette interface
- Quelques intérêts
 - Séparation du traitement et de la présentation le traitement ne « sais pas » comment les données seront affichées
 - la vue ne « sais pas » comment les données sont obtenues
 - Facilitation du travail en équipe (développeurs & designers)
- Emplacement des vues

resources/views

- Une vue peut contenir du HTML, du CSS, du JavaScript et du PHP (gestion du contenu dynamique)

Vue

retourner une vue

- Retourner une vue à partir d'une fonction (closure ou action)
 - `return view('nomVue')`
 - `return view('nomVue')->with(variable)`
 - `return view('nomVue', tabAss)`
retourne une vue en lui passant des données
- Retourner une vue via une route
 - `Route::view('chemin', 'nomVue')`
 - `Route::view('chemin', 'nomVue', tabAss)`
- Exemple
 - Éditer `greeting.blade.php`

```
<p>  
    <?php echo "Hello $prenom $nom ! Are you $age old ?"; ?>  
</p>
```
 - Dans l'action `hello()`, retournez
`view('greeting', compact('prenom', 'nom', 'age'))`

Vue

Blade - présentation

- Blade est un moteur de template
- Moteur de template : un programme qui
 - facilite la gestion du contenu dynamique d'un template
 - utilise un pseudo-code plus accessible au designer que le PHP
 - utilise un cache
- Blade interprète un pseudo-code pour
 - afficher la valeur d'une expression PHP avec `{{ }}`
 - effectuer plusieurs tâches avec des directives comme `@if`,
`@else`, `@elseif`, `@switch`, `@for`, `@while`, `@foreach`, `@forelse`,
`@isset`, `@empty`, `@csrf`, `@method`, `@include`, `@yield`,
`@section`, `@extends`, `@php`, ...

Vue

Blade - expression

- Syntaxe d'affichage d'une expression

`{{ expression }}`

- Exemples

- Variable simple : `{{ $age }}`
- Élément d'un tableau : `{{ $apprenant['prenom'] }}`
- Attribut d'un objet : `{{ $apprenant -> prenom }}`
- Dans `greeting.blade.php`, commenter juste avant `echo` et ajouter, en dessous, la ligne :

Hello `{{ $prenom.' '.$nom }}` ! Are you `{{ $age }}` old ?

Vue

Blade – structures conditionnelles

- @if (condition)

```
    traitement  
    [@else  
    traitement ]
```

@endif

- @if (condition)

```
        Traitement  
    @elseif (condition)  
        traitement  
    @else  
        traitement
```

@endif

- @isset(\$sensDeDonnées)

```
    // $sensDeDonnées est défini et  
    non vide...
```

@endisset

- @empty(\$sensDeDonnées)

```
    // $sensDeDonnées est vide...
```

@endempty

Vue

Blade – structures itératives

- @for (init; condition; modif)
 traitement
@endfor
- @foreach (\$sensDeDonnées as \$donnée)
 traitement sur \$donnée
@endforeach
- @forelse (\$sensDeDonnées as \$donnée)
 traitement sur \$donnée
@empty
 <p> Aucune donnée </p>
@endforelse
- @while (condition)
 Traitement
@endwhile

Vue

Blade – exemple sur structures contrôles

1) Ajoutez à UserController

```
public $notes = [2,13,5,6,12,8,10,9];  
public function notes()  
{  
    return view('notes',['notes' => $this->notes]);  
}
```

2) Editez la vue notes.blade.php

```
<ul>  
    @foreach ($notes as $note)  
        <li>  
            {{ $note }} est  
            @if($note >= 10)  
                supérieure ou égale  
            @else  
                inférieure  
            @endif  
            à 10  
        </li>  
    @empty  
        <li>Loading...</li>  
    @endforeach  
</ul>
```

3) Ajoutez la route

```
Route::get('/notes', [UserController::class,'notes'])->name('notes');
```

Vue

Blade – héritage d'un template - parent

- Le template parent définit
 - du contenu commun à tous les templates enfants
 - des sections à remplir ou à compléter par des templates enfants
- Syntaxes de définition d'une section
 - `@yield('nomSection' [, 'contenu par défaut'])`
 - `@section('nomSection')`
`<p>Contenu par défaut plus élaboré</p>`
...
`@show`

Vue

Blade – héritage d'un template - enfant

- Le template enfant
 - doit d'abord étendre le template de base
 - `@extends('chemin.du.parent')`
 - (re)définit toute section qu'il souhaite remplir en
 - `@section('nomSection', 'Contenu de la section')`
 - `@section('nomSection')`
 - Contenu de la section
 - `@endsection`
 - `@parent` rajoutant du contenu avec `@parent`
 - conserve le contenu par défaut de toute section non
 - (re)défini

Vue

Blade – héritage d'un template - exemple

Éditez resources/views/layouts/parent.blade.php

```
<title>
    @section('title')
        MyApp -
    @show
</title>
</head>
<body>
    <div>
        <a href="{{ route('hello') }}">Hello</a> |
        <a href="{{ route('notes') }}">Notes</a> |
        <a href="#">Update</a>
    </div>
    @yield('content')
</body>
```

Vue

Blade – héritage d'un template - exemple

Éditez resources

Éditez resources/views/notes.blade.php

Rajoutez juste les lignes 1 à 6 et 22. Requêter la route « notes »

```
1  @extends('layouts.parent')
2  @section('title')
3      @parent
4      Liste des notes
5  @endsection
6  @section('content')
7      <ul>
8  >      @foreach ($notes as $note) ...
18 >      @empty ...
20      @endforeach
21  </ul>
22  @endsection
```


Vue

Blade – inclusion d'une vue

- `@include('chemin.de.la.vue')` inclut une vue. La vue a accès à toute variable du template enveloppant.
- `@include('chemin.de.la.vue', ['nom' => val, ...])` inclusion avec passation de données
- `@includeIf('chemin.de.la.vue', ...)` inclut si la vue existe
- `@includeWhen(condition, 'chemin.de.la.vue', ...)` inclut si une condition est vraie

Vue

Blade – exemple sur formulaires

```
public $apprenants = [
    ['id' => 1 , 'nom'=> 'Fatou Dia' , 'age'=> 23 ],
    ['id' => 2 , 'nom'=> 'Ngor Diouf' , 'age'=> 18 ]
];

public function updateForm($id){
    foreach ($this->apprenants as $apprenant) {
        if ($apprenant['id']==$id) {
            $apprenantToUpdate = $apprenant;
            return view('/update')->with('apprenantToUpdate',$apprenantToUpdate);
        }
    }
}

public function update(Request $request, $id){
    foreach ($this->apprenants as &$apprenant) {
        if ($apprenant['id']==$id) {
            $apprenant['nom'] = $request->input('nom');
            $apprenant['age'] = $request->input('age');
            return $this->apprenants;
        }
    }
}
```





BDD

sommaire

- 1) Présentation
- 2) Configurer la base de données
- 3) Migration
- 4) Eloquent

BDD

présentation

- Actuellement (avril 2021), Laravel supporte quatre SGBD
-  MySQL 5.7+
-  PostgreSQL 9.6+
-  SQLite 3.8.8+
-  SQL Server 2017+
- • Dans ce cours, nous travaillerons avec MySQL
- • Exemple
- dans MySQL, créer la base de données
- laravel-classroom

BDD

configurer la base de données

- Dans /.env, assigner des valeurs aux variables
- d'environnement : DB_CONNECTION, DB_HOST,
- DB_PORT, DB_DATABASE, DB_USERNAME et
- DB_PASSWORD
- • Exemple
- éditez /.env avec les paramètres suivants :
- DB_CONNECTION=mysql
- DB_HOST=127.0.0.1
- DB_PORT= 3306
- DB_DATABASE= laravel-classroom
- DB_USERNAME= root
- DB_PASSWORD=

BDD migration

- Une migration permet de créer, de mettre à jour et de suivre
- les évolutions d'un schéma de base de données.
- • Commande pour créer une migration
- `php artisan make:migration create_nomTable_table`
- `nomTable` doit être au pluriel et en minuscules
- • La migration (au format `date_time_create_nomTable_table.php`)
- sera créée dans le dossier `database/migrations` et contiendra
- deux méthodes
- `up()` : exécutée lorsque la migration est lancée
- `down()` : exécutée lorsque la migration est annulée
- • Commande pour lancer les migrations
- `php artisan migrate`
- • Commande pour annuler les migrations
- `php artisan migrate:rollback`

BDD

migration-exemple

- 1) Créer la migration apprenants
- 2) Rajouter à apprenants les attributs
nom (string) et age (integer)
- 3) Lancer les migrations

BDD

Eloquent - présentation

- Eloquent = ORM (Object-Relation Mapping ou correspondance objet-relationnel en fr) permettant de faciliter la communication avec une base de données
- Modèle = classe PHP correspondant à une table de la bdd
- Instance (objet) du modèle ? enregistrement de la table

Classe
Apprenant
id
prenom
age

Objet
Apprenant
id : 1
prenom : Ali
age : 23

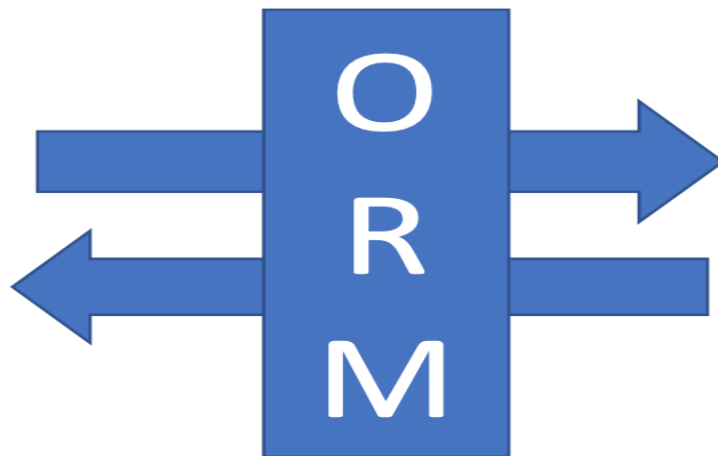


Table apprenants		
id	prenom	age
1	Ali	23

BDD

Eloquent – définition d'un modèle

- Commande de création d'un modèle
- `php artisan make:model NomModele`
- NomModele en UpperCamelCase
- • Commande de création d'un modèle avec génération d'une migration
- `php artisan make:model NomModel -m`
- • Avec la dernière version, tout modèle est créé par défaut dans le dossier `app/Models`
- • Ds certaine version, tout modèle est créé par défaut dans le dossier `app`
- • Pour créer un modèle dans un sous-dossier de `app`, il suffit de le préfixer au nom du modèle
- • Exemple
- créez le modèle `Apprenant` dans `app/Models`

BDD

Eloquent – persister un nouvel objet

- Dans un contrôleur, importer la classe du modèle
- `use App\Models\NomModèle`
- • Dans une action d'un contrôleur
- 1) créer une nouvelle instance de modèle
- `$nomObjet = new NomModele`
- 2) renseigner les attributs de cet objet
- `$nomObjet->nomAttr = valeur`
- 3) invoquer la méthode `save` sur l'objet
- `$nomObjet->save()`
- • L'objet inséré reçoit un id récupérable par `$nomObjet->id`

BDD

Eloquent – exemple pour persister un nouvel objet

Dans UserController

```
use App\Models\Apprenant;

public function add(Request $request){
    $newApprenant = new Apprenant;
    $newApprenant->nom = $request->nom;
    $newApprenant->age = $request->age;
    $newApprenant->save();
    return "L'apprenant a été persisté avec l'identifiant $newApprenant->id";
}
```

Dans web.php ajouter la route :

```
Route::get('/add', 'App\Http\Controllers\UserController@add')->name('add');
```

Requêtez avec un client REST

BDD

Eloquent – récupérer des objets

- Dans un contrôleur, importer la classe du modèle
- `use App\Models\NomModèle`
- • Dans une action d'un contrôleur, récupérer un objet via sa clé primaire
- `$nomObjet = NomModèle::find($id);`
- • Autres méthodes
- `firstWhere('nomAttr', valeur)` : récupérer le 1er objet à partir d'un attribut
- `where('attr', 'op', val)->firstOr(tabAssAttr?, callback)` : récupérer des attributs du 1er objet ou bien appeler une callback
- `all()` : récupérer tous les objets

BDD

Eloquent – exemple pour récupérer un objet par son id

Dans UserController : ajouter cette fonction

```
public function getOne($id){  
    $apprenant = Apprenant::find($id);  
    if(!$apprenant){  
        return response("Aucun apprenant trouvé avec  
l'identifiant $id", 404);  
    }  
    return $apprenant;  
}
```

Dans web.php ajouter la route :

```
Route::get('/getOne/{id}', 'App\Http\Controllers\UserController@add')->name('getOne');
```

Requêtez les chemins « getOne/1 » et « getOne/0 »

BDD

Eloquent – modifier des objets

- Pour modifier un objet
 - 1) Récupérer l'objet
 - 2) Modifier tout attribut souhaité de l'objet
 - 3) Invoquer la méthode `save()` sur l'objet
- • Pour modifier plusieurs objets
 - ☐ Invoquer `where()` et `update()` sur le modèle
 - ☐ Exemple
- `App\Flight::where('active', 1)`
 - >`where('destination', 'San Diego')`
 - >`update(['delayed' => 1]);`

BDD

Eloquent – exemple sur modifier un objet

Modifier les actions updateForm() et update()

```
public function updateForm($id){  
    $apprenantToUpdate = Apprenant::find($id);  
    return view('/update')->with('apprenantToUpdate',$apprenantToUpdate);  
}  
  
public function update(Request $request, $id){  
    $apprenant = Apprenant::find($id);  
    $apprenant['nom'] = $request->input('nom');  
    $apprenant['age'] = $request->input('age');  
    $apprenant->save();  
    return "L'apprenant d'$id a maintenant $apprenant->age ans";  
}
```

Requêter le chemin « update/1 »

BDD

Eloquent – supprimer des objets

- Pour supprimer un objet
 - 1) Récupérer l'objet
 - 2) Invoquer la méthode delete() sur l'objet
- • Pour supprimer un (ou +sieurs) objet(s) via un (ou des) identifiant(s),
 - `?` invoquer `destroy(listId)` sur le modèle
- `?` Exemples
 - `?` `Apprenant::destroy(1);`
 - `?` `Apprenant::destroy(1, 2, 3);`
- • Pour supprimer plusieurs objets
 - • Invoquer `where()` et `delete()` sur le modèle
 - • Ex : `App\Flight::where('active', 0)->delete();`

Référence

- <https://laravel.com/docs/8.x>
- https://apcpedagogie.com/cours-et-tutoriels/les_cours/cours-de-programmation/laravel/
- <https://walkerspider.com/cours/laravel>