INSTITUT SUPÉRIEUR DES ETUDES TECHNOLOGIQUES DE BIZERTE
المعهد العالي للدراسات التكنولوجية ببنزرت

Atelier Framework coté serveur

Enseignant: Nizar MAATOUG

Groupe: DSI22 ISET Bizerte

AU: 2021-2022

Aperçu du contenu (1/2)

- Architecture des applications web
  - Notions de Framework, serveur web
  - Architecture de notre application
- Framework Laravel
  - Cycle de vie d'une requête
  - Architecture MVC
  - Installation des outils, première application, anatomie
  - Modèle: entités, Mapping objet/relationnel, base de données: migration, factory, seeder
  - Design pattern Repository
  - Contrôleur: implémentation
  - Les routes: web et API REST (REpresentational State Transfer)

Aperçu du contenu (2/2)

- Les vues: Moteur de template Blade
- Autres concepts:
  - Middleware
  - Session
  - Validation
  - Cache
  - FileStorage
  - Events
  - Mail
  - Job
  - ...

Objectifs

- À la fin de ce module, vous serez en mesure de:
  - Maitriser le développement d’une application web avec le framework Laravel

Architecture des applications web

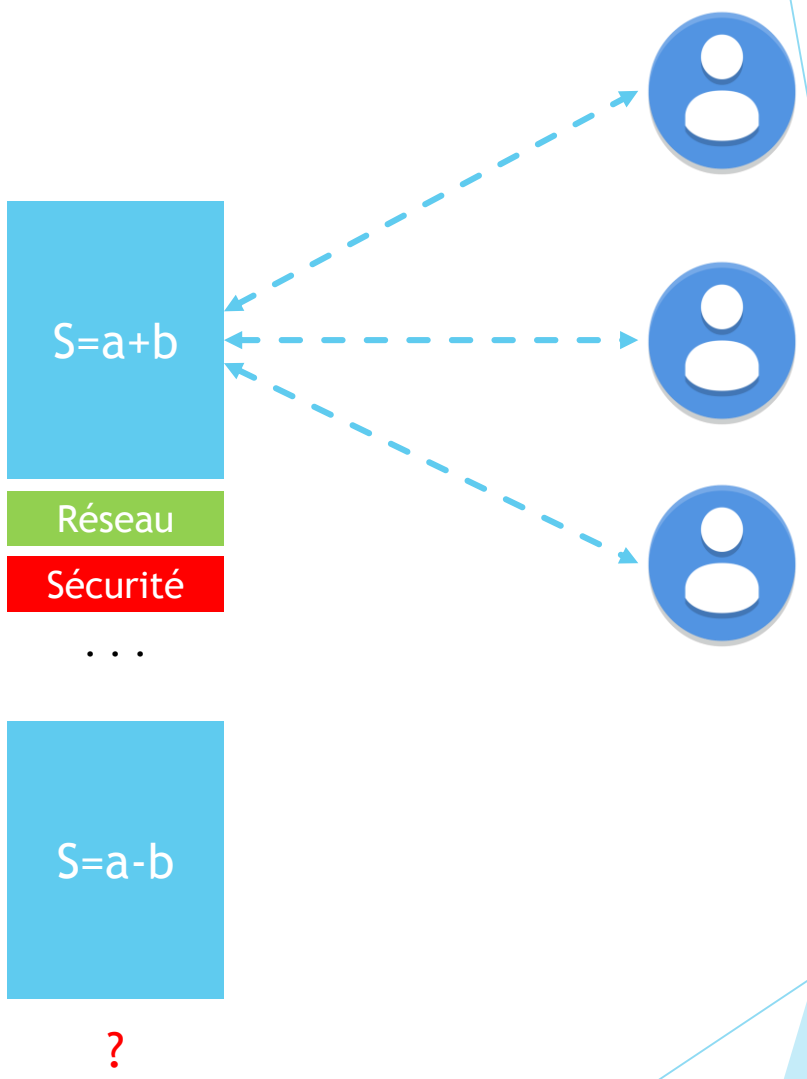
Résultats

À la fin de cette séance, vous serez en mesure de:

- Définir le concept architecture d'une application informatique
- Définir le concept Framework
- Démontrer le besoin en terme d'architecture logicielle
- Maitriser l'architecture d'une application web

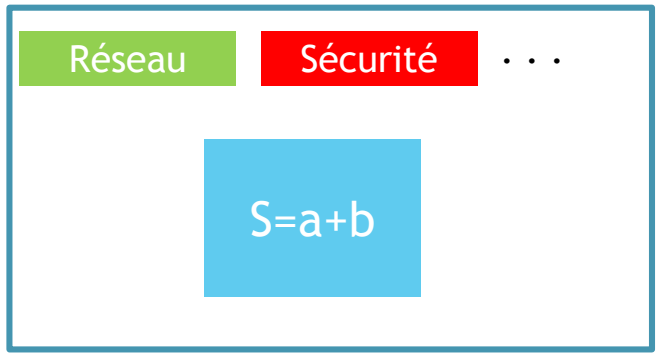
Problématique:

- Il vous ait demandé de coder un programme:
  - Partagé entre plusieurs utilisateurs
  - Sécurisé
  - …
- Que faire pour un deuxième programme avec les mêmes spécifications techniques ?



Solution

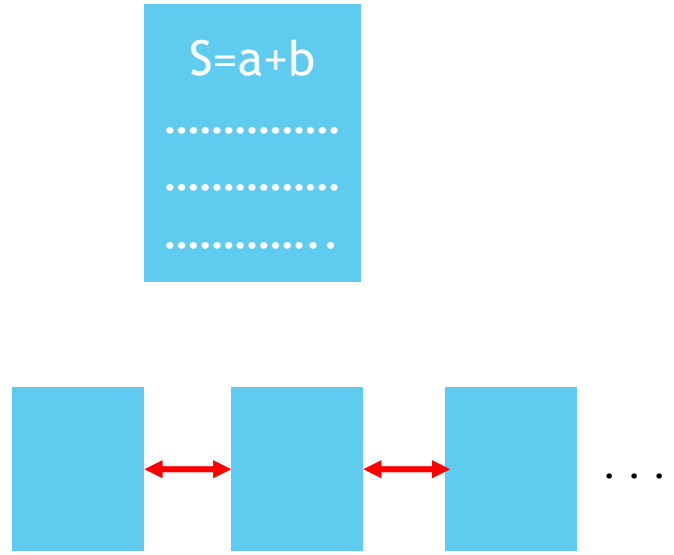
- Réutiliser les services transversaux
- Les assembler dans un seul programme appelé serveur d'application
- Se focaliser sur la logique métier de votre application



Serveur d'application

Problématique

- Il vous ait demandé d'ajouter d'autres fonctionnalités métiers
  - fichier volumineux
  - Doit être décomposé en plusieurs fichiers
  - Quelle stratégie suivre ? Quel modèle ?
  - Quel type de communication ?

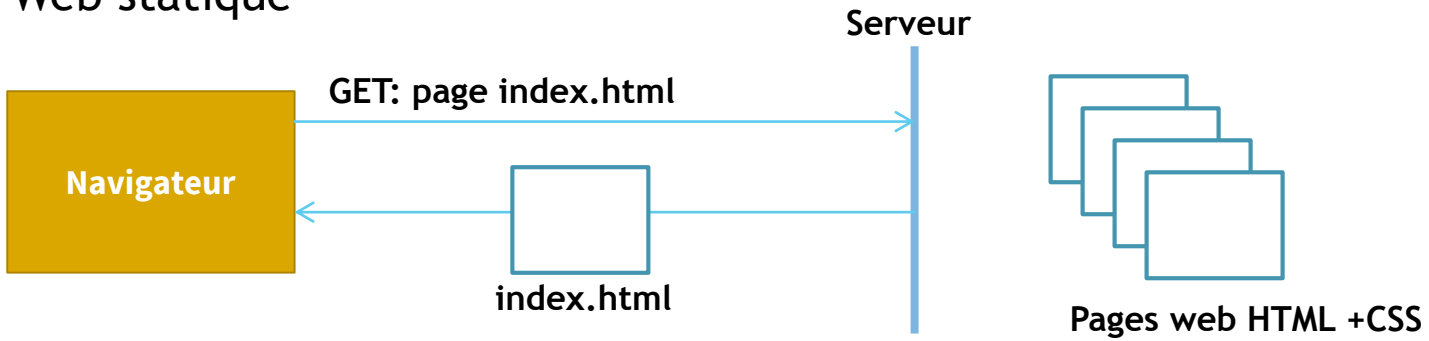


Solution

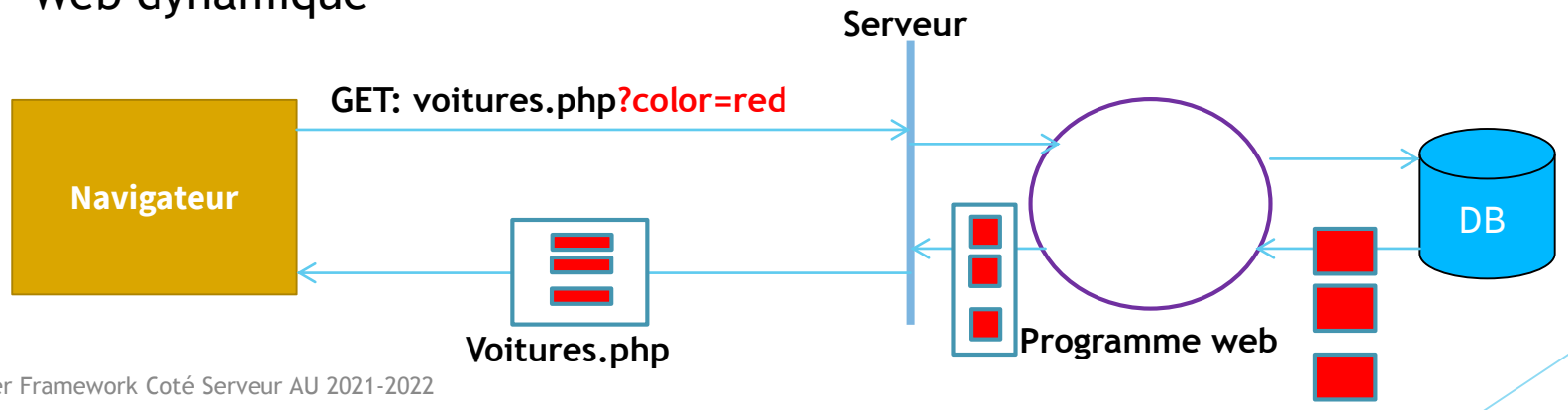
- Définir une architecture ou utiliser un modèle d'architecture déjà existant
- Une architecture définit:
  - les types de composants de votre application:
    - Exemple: composant d'accès aux données, composant de traitement métier, composant d'affichage,…
  - La communication entre ces types de composants
- **Un Framework** est une implémentation technique d'une architecture
  - Facilite le développement en respectant l'architecture associée
  - Offre des services transversaux
  - Exploite aussi les services du serveur d'application (serveur web)

Architecture web classique

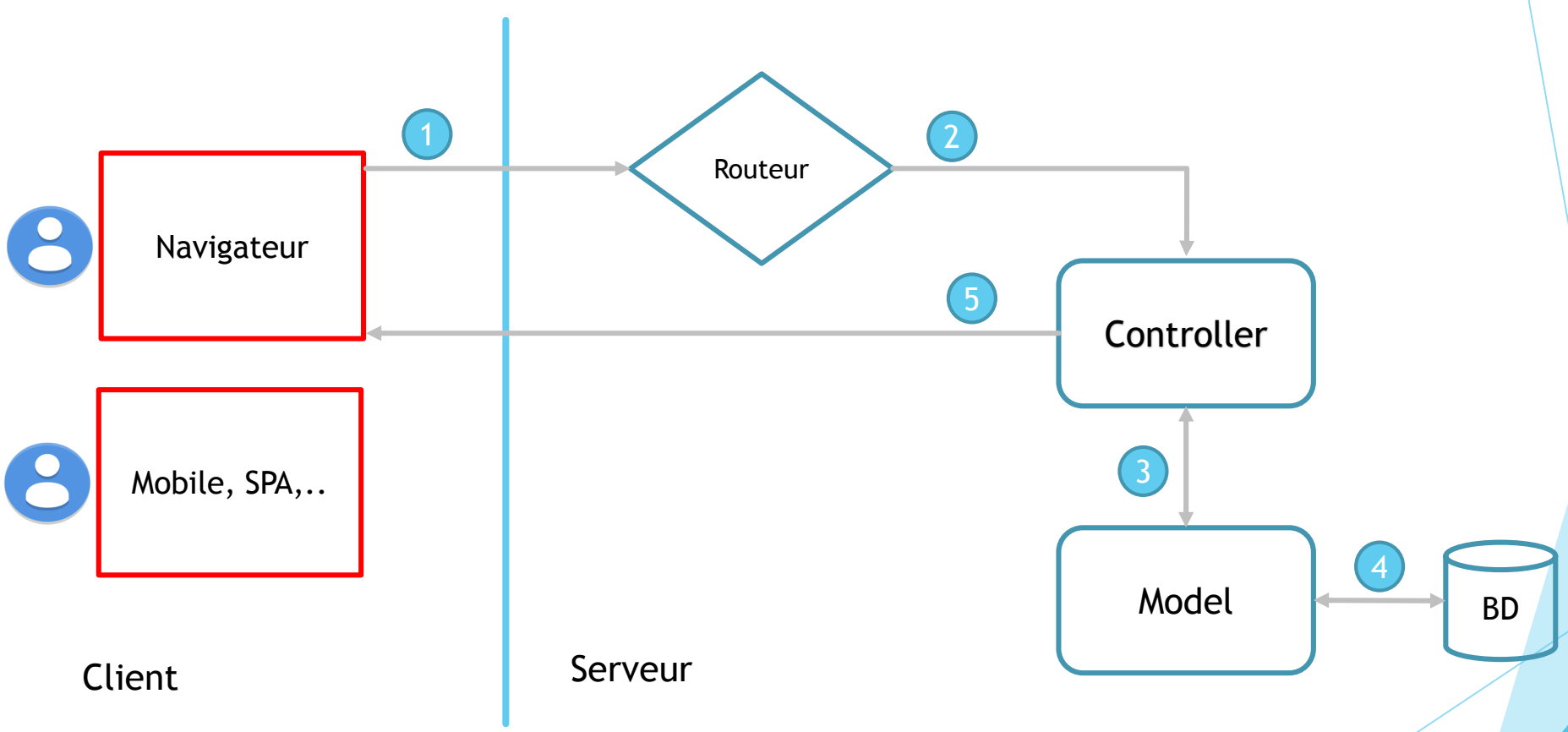
Web statique



Web dynamique



Laravel: Architecture MVC



Framework Laravel

Résultats

À la fin de cette partie, vous serez en mesure de:

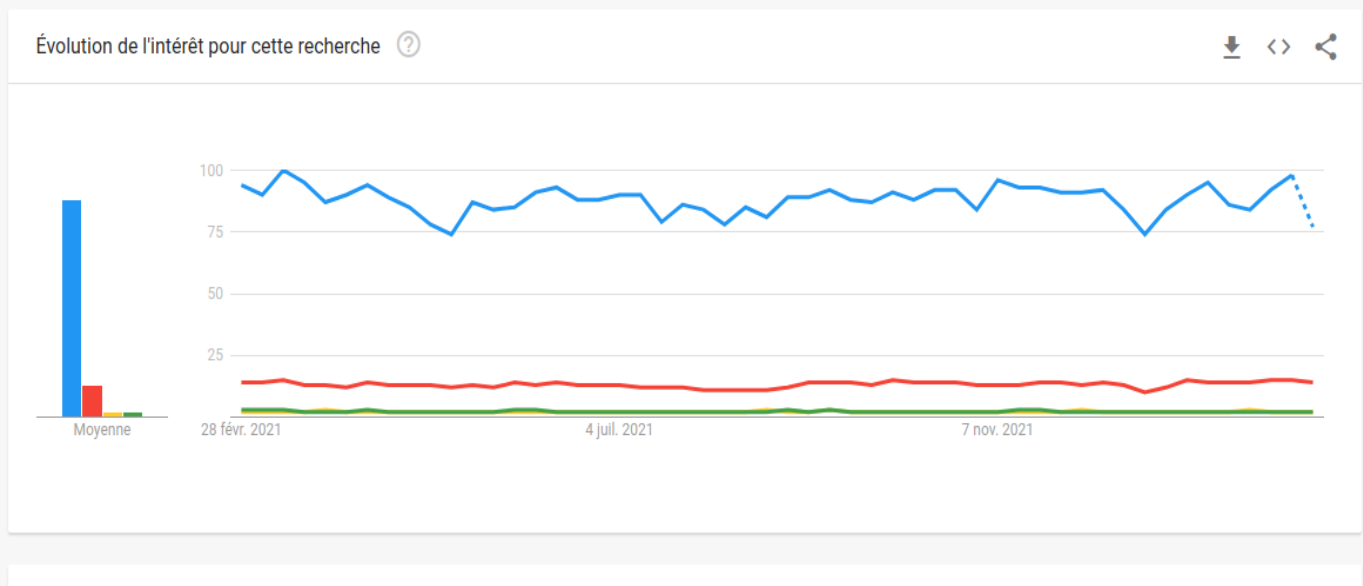
- Maitriser l’architecture d’une application Laravel
- Développer une application Laravel

Laravel

- Un Framework d'application web créé par Taylor Otwel en 2011
- Framework open source
- Basé sur le langage PHP orienté objet:
  - Classe, interface, namespace, trait, fonction anonyme, closure,..
- Implémente une architecture MVC (Modèle Vue Controller)
- Implémente les designs patterns facilitant le développement:
  - Inversion de contrôle et Injection de dépendances
  - Façade
- Permet de développer:
  - Des applications web
  - Des API REST

Laravel

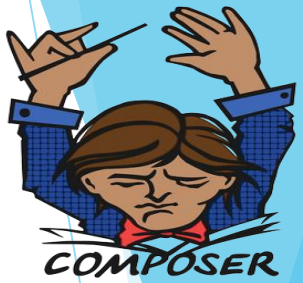
- Pourquoi Laravel ?
  - Facile à maitriser
  - Permet de créer des applications robustes et fiables
  - Le framework PHP le plus utilisé



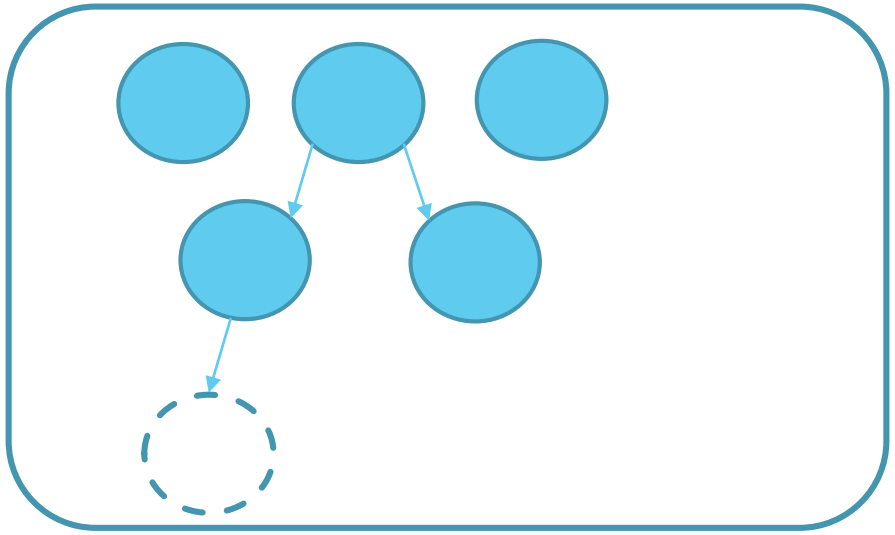
Laravel: installation des outils

- XAMPP : regroupe le langage php, le serveur apache et le SGBD MySql
  - @: https://www.apachefriends.org/fr/download.html
- Composer: gestionnaire de dépendances PHP
  - @: https://getcomposer.org/download/
- VS code: éditeur de code
  - @: https://code.visualstudio.com/download
  - @: https://marketplace.visualstudio.com/items?itemName=onecentlin.laravel-extension-pack
- Postman: pour tester l'API REST
  - @: https://www.postman.com/downloads/
- Node.js NPM (Node Package Manager)
  - @: https://nodejs.org/en/download/

Composer: gestionnaire de dépendances

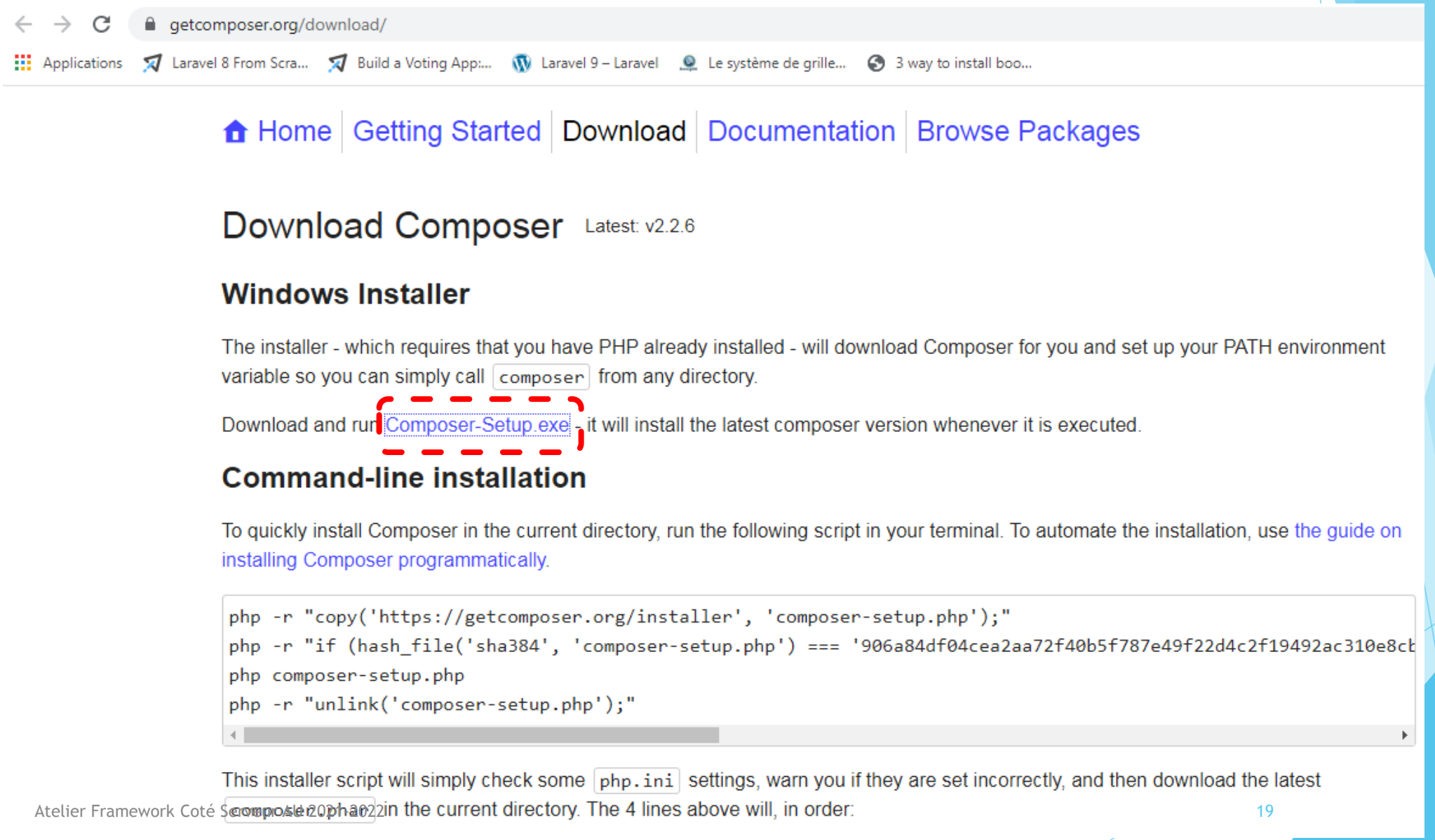


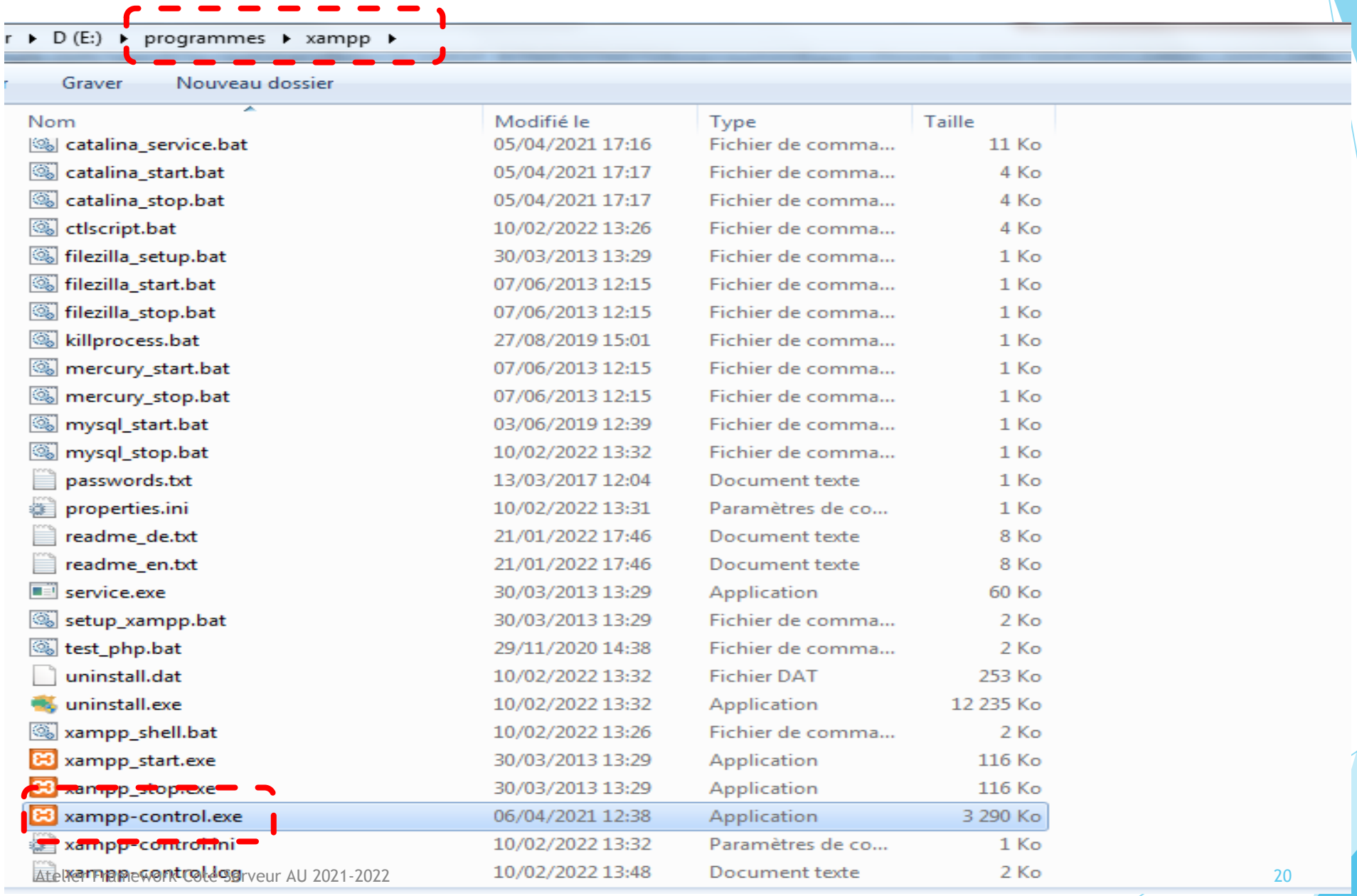
- Un outil permettant la gestion des dépendances d'un projet PHP

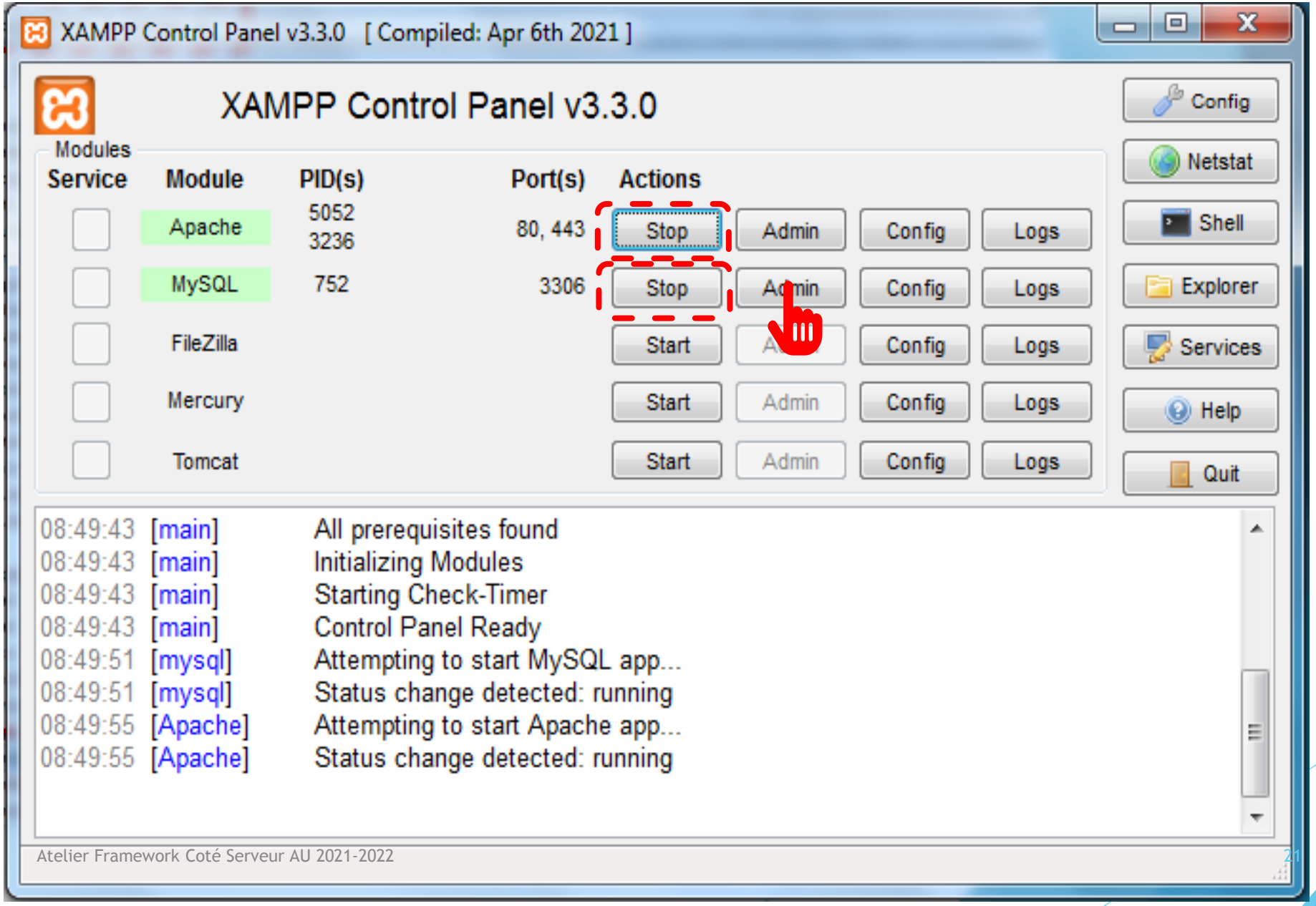


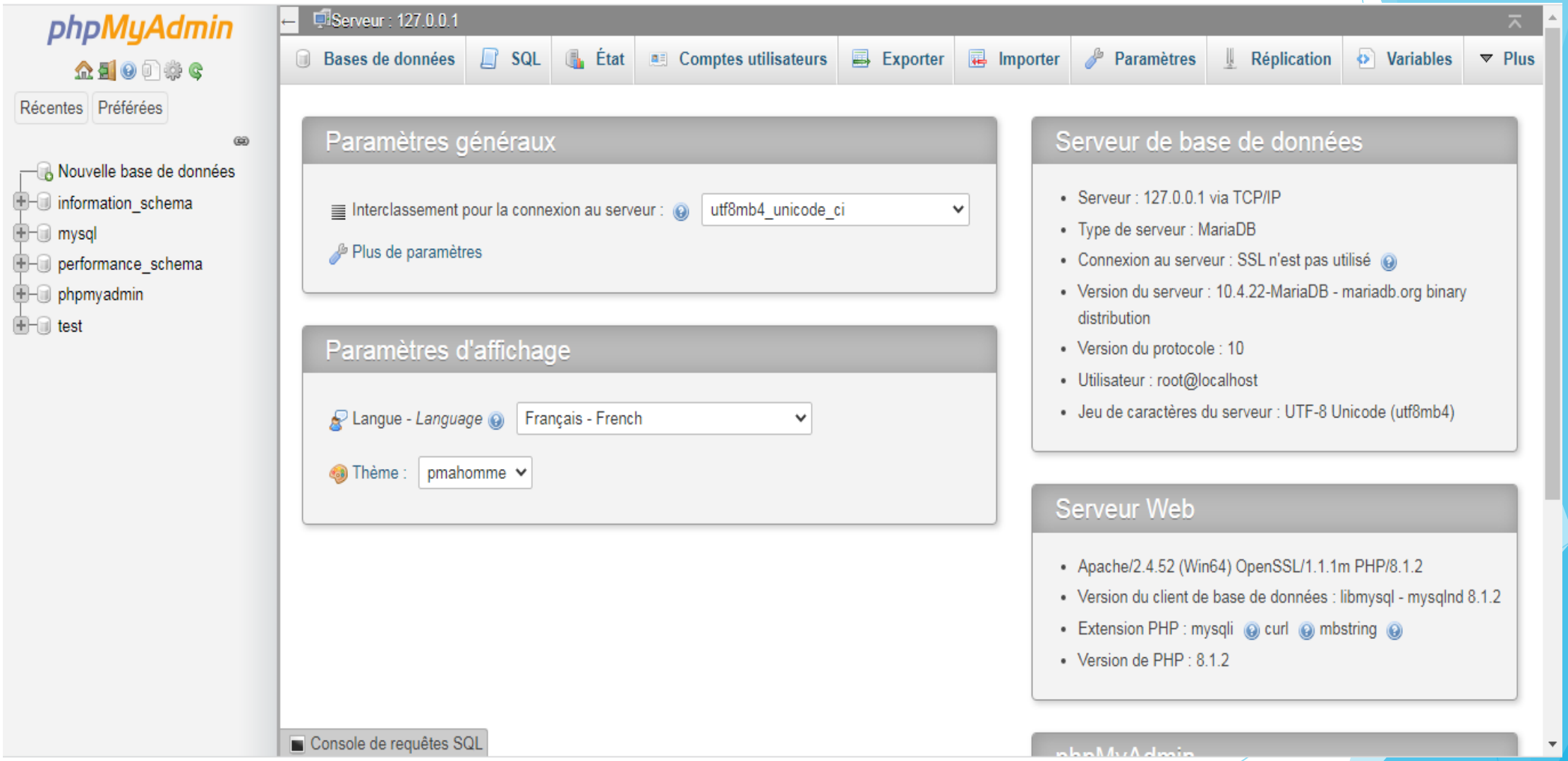
Projet laravel

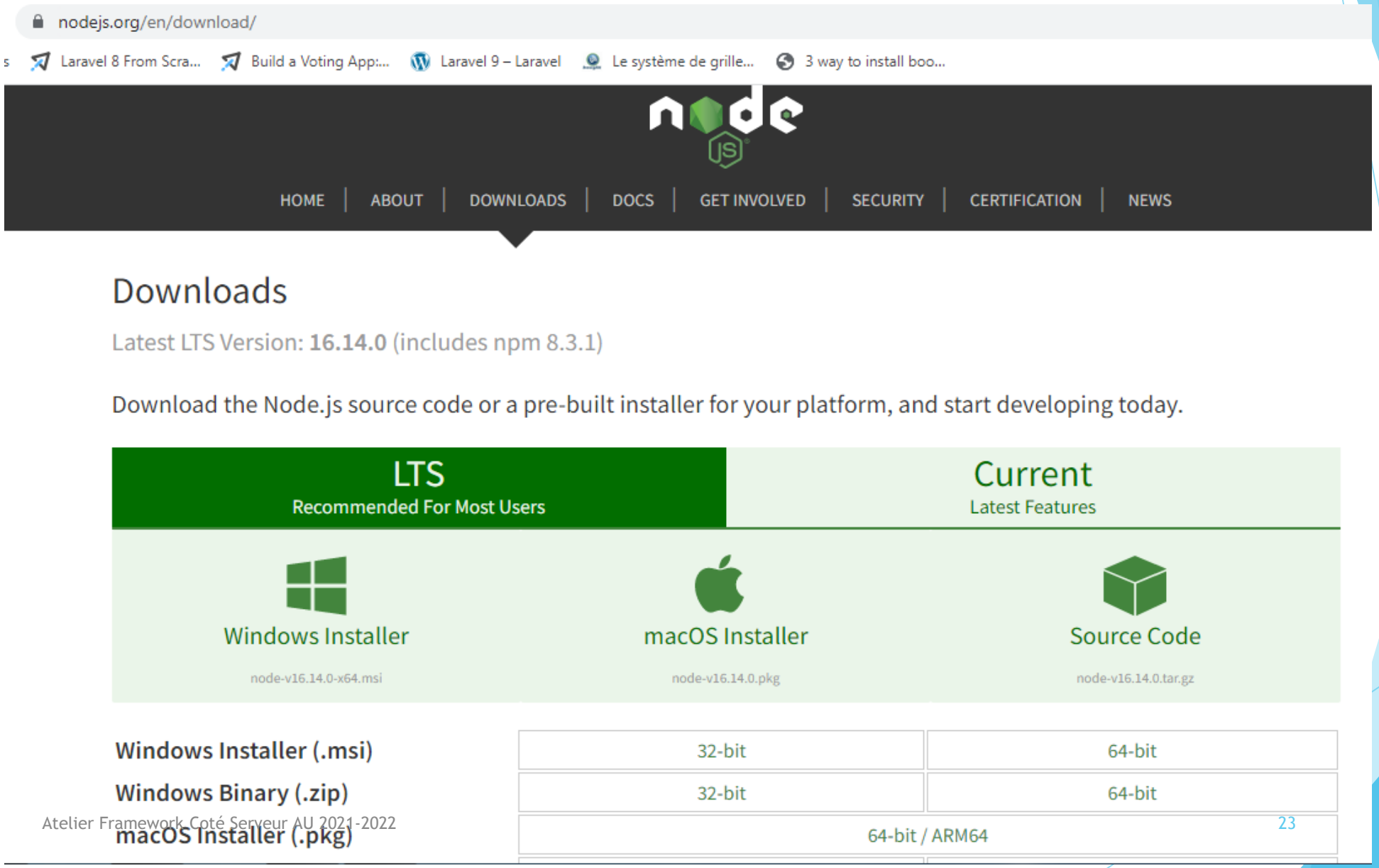
- La gestion de dépendances est difficile à gérer manuellement.
- **Composer** permet d'automatiser la gestion des dépendances en se basant sur:
  - Un fichier de configuration
  - Des dépôts distants de dépendances


getcomposer.org/download/
Applications
Laravel 8 From Scra...
Build a Voting App:...
Laravel 9 – Laravel
Le système de grille...
3 way to install boo...
Home
Getting Started
Download
Documentation
Browse Packages
Download Composer
Latest: v2.2.6
Windows Installer
The installer - which requires that you have PHP already installed - will download Composer for you and set up your PATH environment variable so you can simply call composer from any directory.
Download and run Composer-Setup.exe - it will install the latest composer version whenever it is executed.
Command-line installation
To quickly install Composer in the current directory, run the following script in your terminal. To automate the installation, use the guide on installing Composer programmatically.
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') === '906a84df04cea2aa72f40b5f787e49f22d4c2f19492ac310e8cb
php composer-setup.php
php -r "unlink('composer-setup.php');"
This installer script will simply check some php.ini settings, warn you if they are set incorrectly, and then download the latest composer.phar in the current directory. The 4 lines above will, in order:


D (E:) ▸ programmes ▸ xampp ▸
Graver
Nouveau dossier
Nom
Modifié le
Type
Taille
catalina_service.bat 05/04/2021 17:16 Fichier de comma... 11 Ko
catalina_start.bat 05/04/2021 17:17 Fichier de comma... 4 Ko
catalina_stop.bat 05/04/2021 17:17 Fichier de comma... 4 Ko
ctlscript.bat 10/02/2022 13:26 Fichier de comma... 4 Ko
filezilla_setup.bat 30/03/2013 13:29 Fichier de comma... 1 Ko
filezilla_start.bat 07/06/2013 12:15 Fichier de comma... 1 Ko
filezilla_stop.bat 07/06/2013 12:15 Fichier de comma... 1 Ko
killprocess.bat 27/08/2019 15:01 Fichier de comma... 1 Ko
mercury_start.bat 07/06/2013 12:15 Fichier de comma... 1 Ko
mercury_stop.bat 07/06/2013 12:15 Fichier de comma... 1 Ko
mysql_start.bat 03/06/2019 12:39 Fichier de comma... 1 Ko
mysql_stop.bat 10/02/2022 13:32 Fichier de comma... 1 Ko
passwords.txt 13/03/2017 12:04 Document texte 1 Ko
properties.ini 10/02/2022 13:31 Paramètres de co... 1 Ko
readme_de.txt 21/01/2022 17:46 Document texte 8 Ko
readme_en.txt 21/01/2022 17:46 Document texte 8 Ko
service.exe 30/03/2013 13:29 Application 60 Ko
setup_xampp.bat 30/03/2013 13:29 Fichier de comma... 2 Ko
test_php.bat 29/11/2020 14:38 Fichier de comma... 2 Ko
uninstall.dat 10/02/2022 13:32 Fichier DAT 253 Ko
uninstall.exe 10/02/2022 13:32 Application 12 235 Ko
xampp_shell.bat 10/02/2022 13:26 Fichier de comma... 2 Ko
xampp_start.exe 30/03/2013 13:29 Application 116 Ko
xampp_stop.exe 30/03/2013 13:29 Application 116 Ko
xampp-control.exe 06/04/2021 12:38 Application 3 290 Ko
xampp-control.ini 10/02/2022 13:32 Paramètres de co... 1 Ko
xampp_control.log 10/02/2022 13:48 Document texte 2 Ko


XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]
XAMPP Control Panel v3.3.0
Modules
Service
Module
PID(s)
Port(s)
Actions
Apache
5052
3236
80, 443
Stop
Admin
Config
Logs
MySQL
752
3306
Stop
Admin
Config
Logs
FileZilla
Start
Admin
Config
Logs
Mercury
Start
Admin
Config
Logs
Tomcat
Start
Admin
Config
Logs
Config
Netstat
Shell
Explorer
Services
Help
Quit
08:49:43 [main] All prerequisites found
08:49:43 [main] Initializing Modules
08:49:43 [main] Starting Check-Timer
08:49:43 [main] Control Panel Ready
08:49:51 [mysql] Attempting to start MySQL app...
08:49:51 [mysql] Status change detected: running
08:49:55 [Apache] Attempting to start Apache app...
08:49:55 [Apache] Status change detected: running


phpMyAdmin
Récentes
Préférées
Nouvelle base de données
information_schema
mysql
performance_schema
phpmyadmin
test
Serveur : 127.0.0.1
Bases de données
SQL
État
Comptes utilisateurs
Exporter
Importer
Paramètres
Réplication
Variables
Plus
Paramètres généraux
Interclassement pour la connexion au serveur :
utf8mb4_unicode_ci
Plus de paramètres
Paramètres d'affichage
Langue - Language
Français - French
Thème :
pmahomme
Serveur de base de données
Serveur : 127.0.0.1 via TCP/IP
Type de serveur : MariaDB
Connexion au serveur : SSL n'est pas utilisé
Version du serveur : 10.4.22-MariaDB - mariadb.org binary distribution
Version du protocole : 10
Utilisateur : root@localhost
Jeu de caractères du serveur : UTF-8 Unicode (utf8mb4)
Serveur Web
Apache/2.4.52 (Win64) OpenSSL/1.1.1m PHP/8.1.2
Version du client de base de données : libmysql - mysqlnd 8.1.2
Extension PHP : mysqli curl mbstring
Version de PHP : 8.1.2
Console de requêtes SQL


nodejs.org/en/download/
Laravel 8 From Scra...
Build a Voting App:...
Laravel 9 – Laravel
Le système de grille...
3 way to install boo...
HOME
ABOUT
DOWNLOADS
DOCS
GET INVOLVED
SECURITY
CERTIFICATION
NEWS
Downloads
Latest LTS Version: 16.14.0 (includes npm 8.3.1)
Download the Node.js source code or a pre-built installer for your platform, and start developing today.
LTS
Recommended For Most Users
Current
Latest Features
Windows Installer
node-v16.14.0-x64.msi
macOS Installer
node-v16.14.0.pkg
Source Code
node-v16.14.0.tar.gz
Windows Installer (.msi)
32-bit
64-bit
Windows Binary (.zip)
32-bit
64-bit
macOS Installer (.pkg)
64-bit / ARM64

Laravel: première application

1) Avec composer

> composer create-project laravel/laravel first-app

> cd first-app

> php artisan serve

Laravel: première application

2) Avec laravel installer

> composer global require laravel/installer

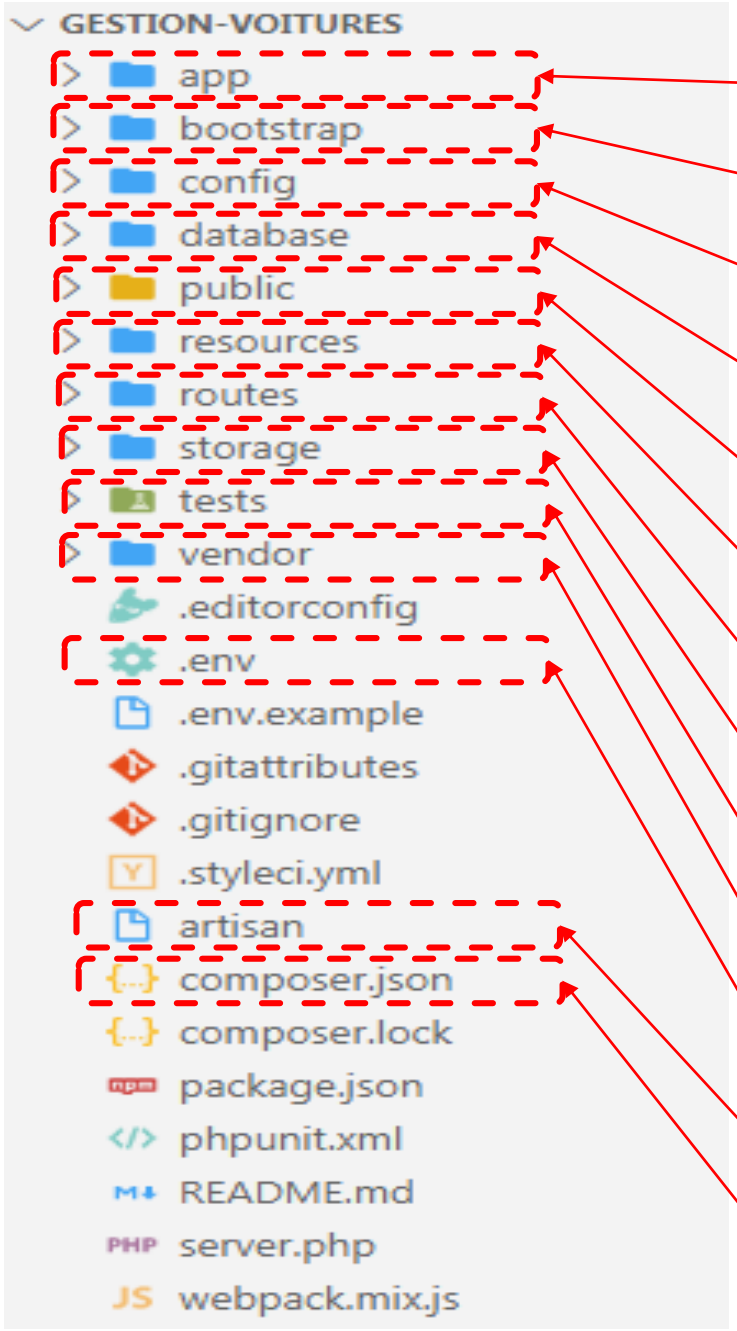
> laravel new first-app

> cd first-app

> php artisan serve

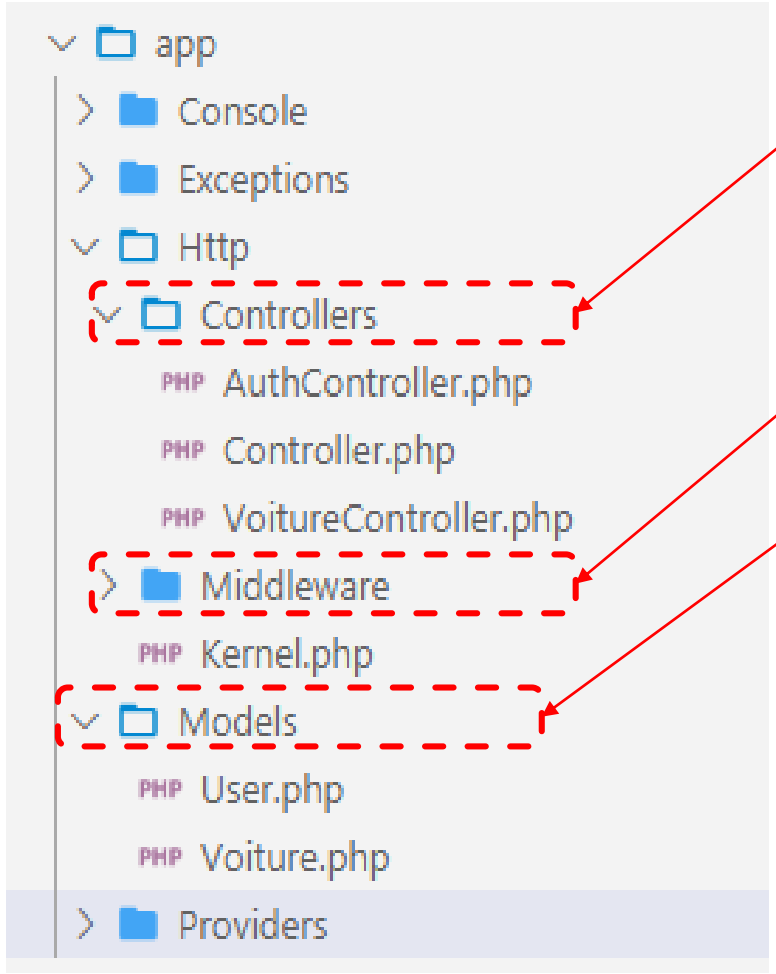
Ajouter le chemin d'installation au PATH %USERPROFILE%\AppData\Roaming\Composer\vendor\bin

Anatomie



- Contient l'essentiel du code de l'application (Modèles, Controlleurs,…)
- Contient le fichier ***app.php*** qui amorce l'exécution de l'application
- Contient les fichiers de configuration de l'application
- Contient les fichiers de migrations du DB, les factories et seeders
- Le seul rep accessible depuis l'extérieur, contient ***index.php***: point d'entrée de toutes les requêtes peut cotenir aussi des fichiers images, javascripts et css publiques
- Contient les vues, les images, les scripts et les fichiers de traduction
- Contient les fichiers définissant les routes de l'application (web, API, console, channels)
- contient tous les fichiers générés par l'application, par exemple des factures PDF
- Contient les fichiers de tests fonctionnels et unitaires
- Contient les dépendances composer de l'application
- environnement (connexion BD, mots de passes,..)
- Outil en ligne de commande pour la gestion du projet
- Décrit les dépendances de l'application

Anatomie: Dossier App

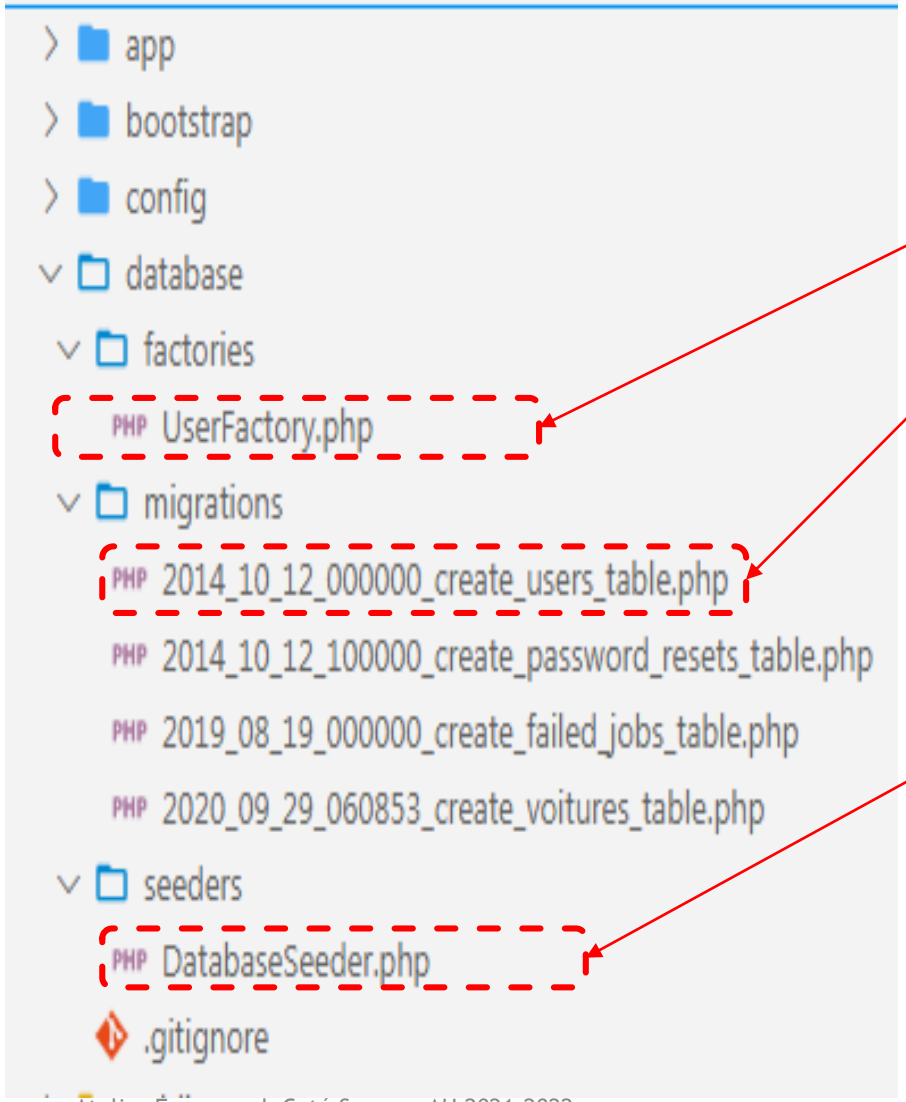


Contient les classes contrôleurs

Contient les middlewares (des contrôles exécutés avant le traitement des requêtes)

Contient les classes modèles (entités)

Anatomie: Dossier database

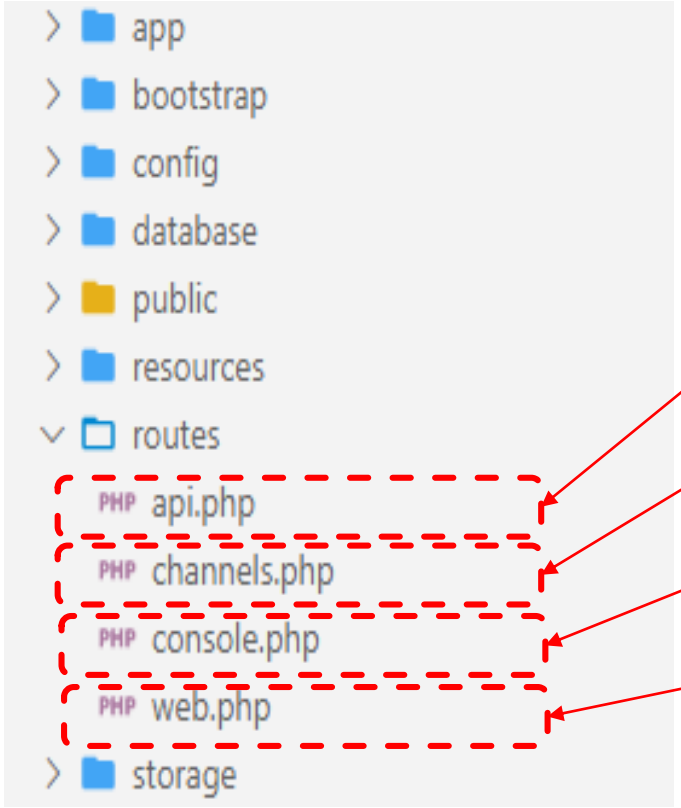


Fabrique d'entités User (données de test)

Fichier de création du schéma de la table ***users***

Fait appel aux factories afin de créer des données de test

Anatomie: Dossier routes



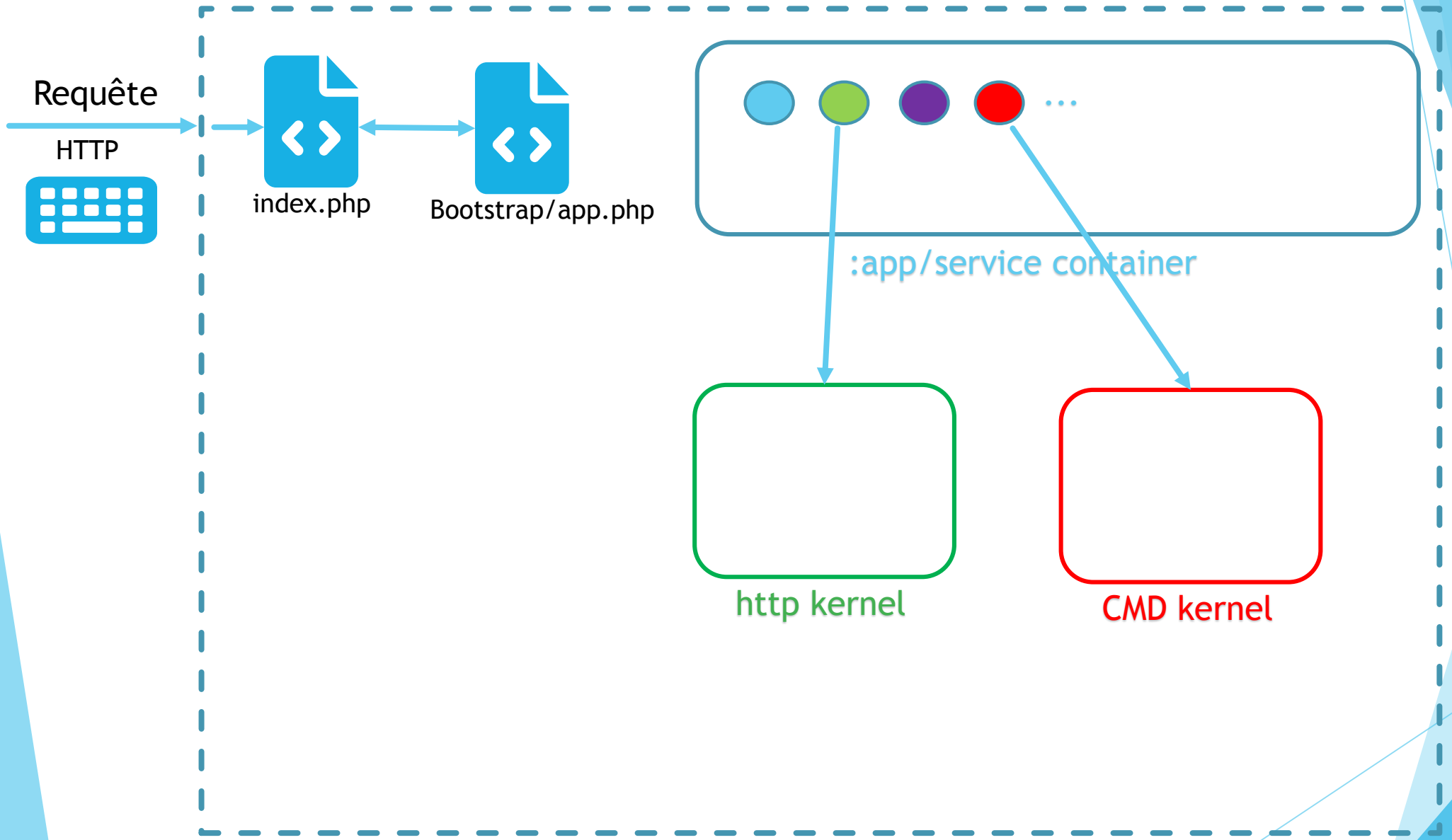
Définit l'API REST

Définit les chaines de diffusion (websockets)

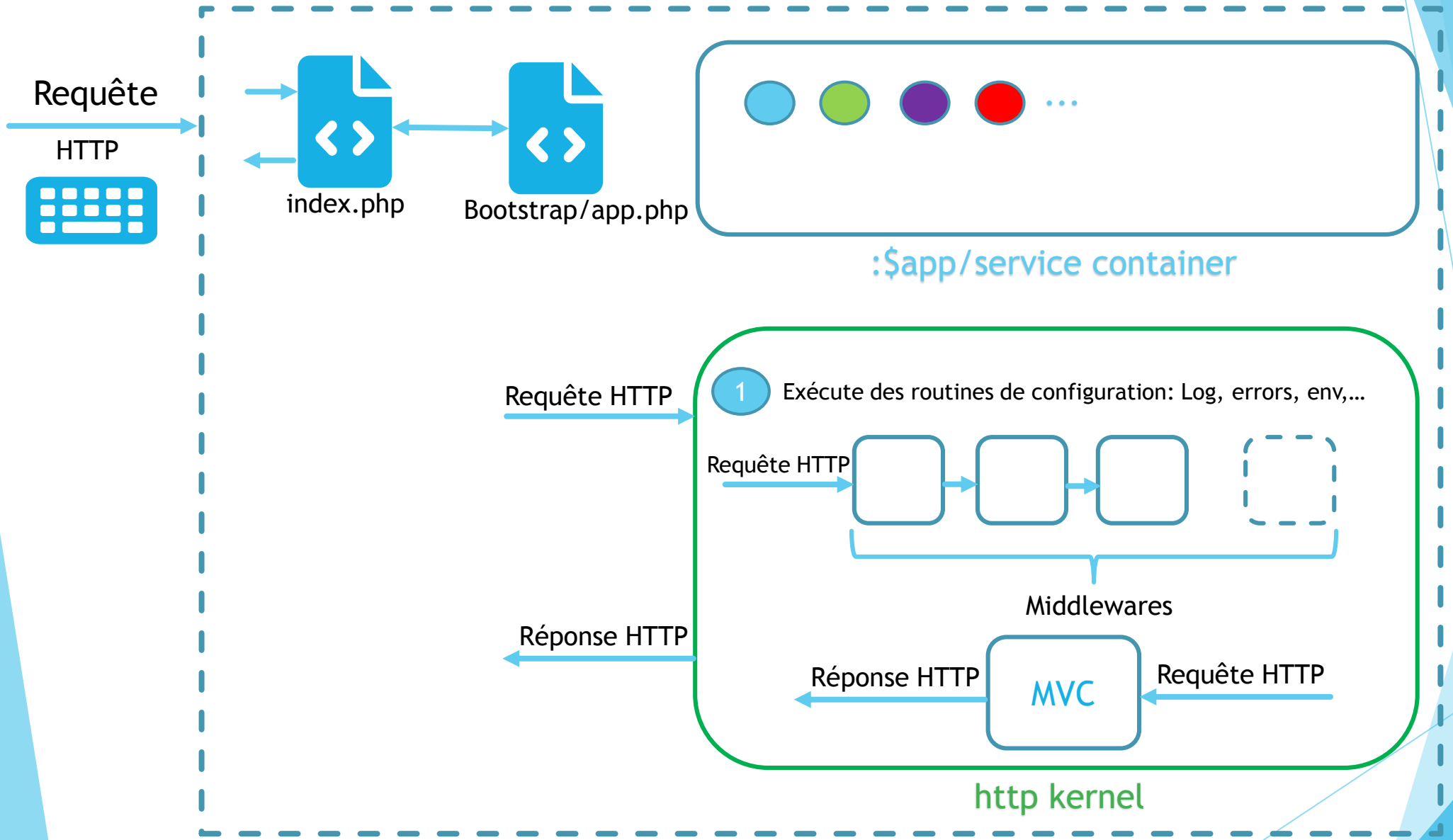
Définit l'interaction en mode commande

Définit le routage web

Cycle de vie d’une requête



Application **Laravel** dans le contexte d'un serveur web



Application **Laravel** dans le contexte d'un serveur web

Modèle

Résultats

- À la fin de cette séance, vous serez en mesure de:
  - Bien identifier les entités persistantes de votre application
  - Démontrer l'avantage d'un ORM (Object/Relationel Mapping)
  - Maitriser le processus de gestion des classes modèles avec l'ORM Eloquent
  - Améliorer l'architecture de votre application avec le design pattern Repository

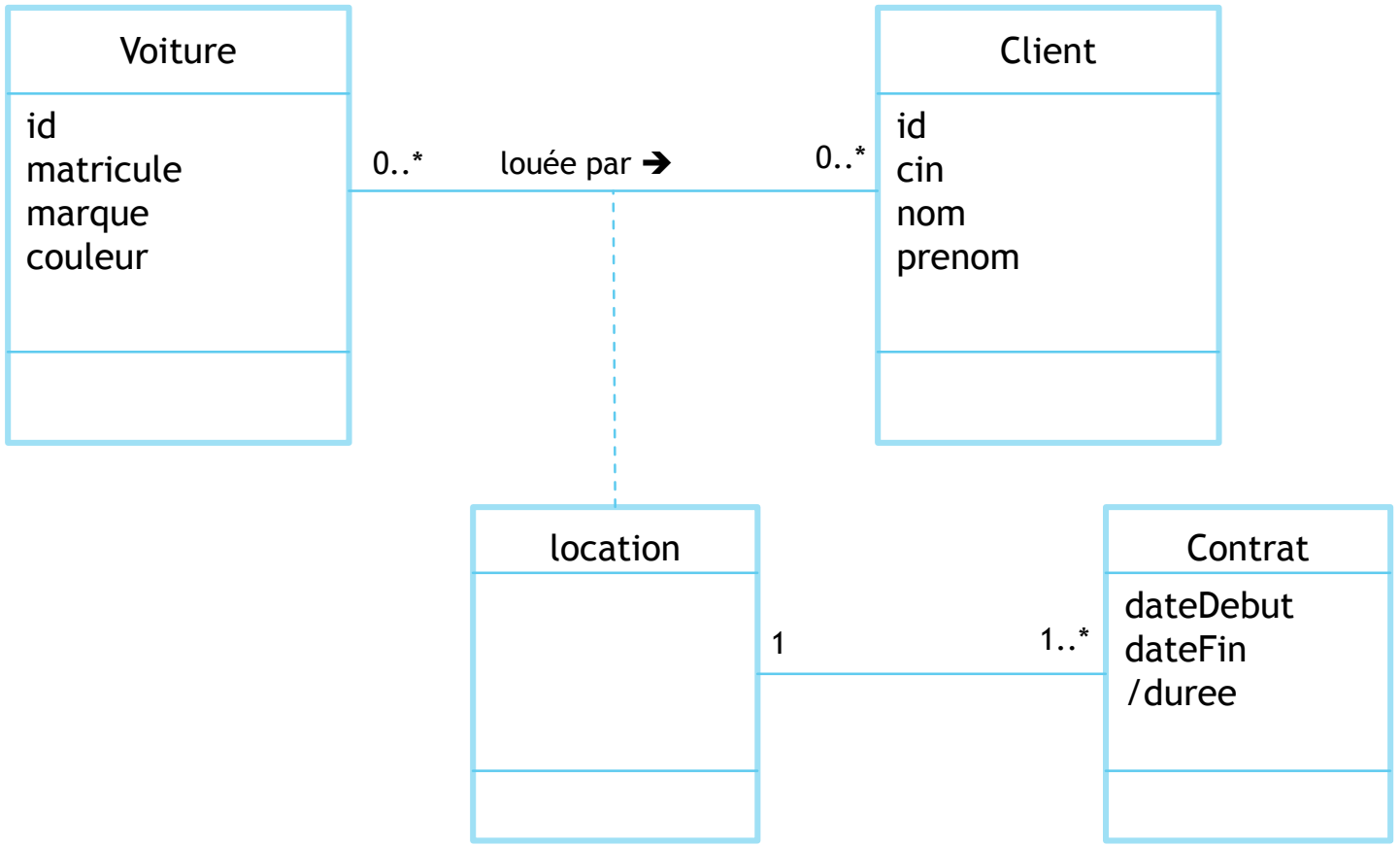
Entités persistantes

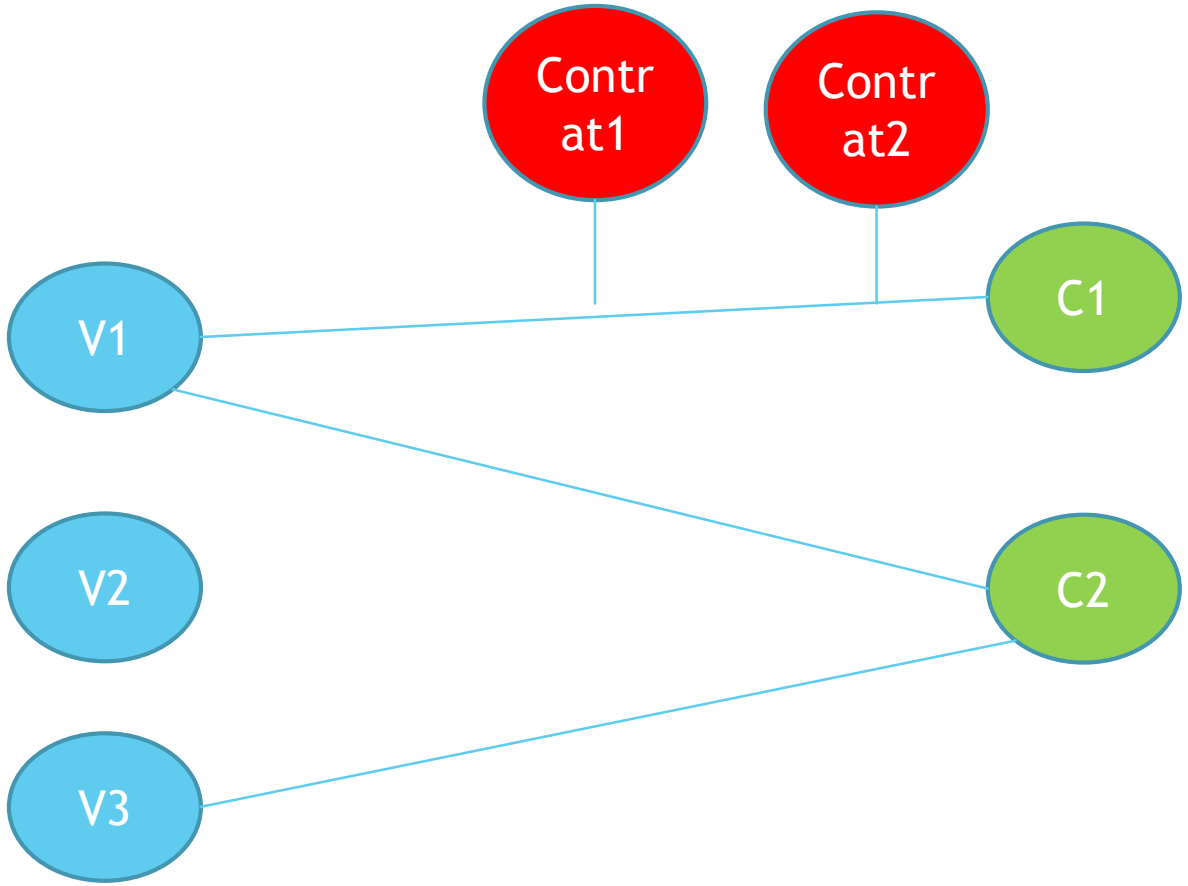
- Comment identifier les entités persistantes de votre application ?
  - Définit par plusieurs attributs (Exemple: une voiture: marque, modèle, couleur,…)
  - Manipulée par votre application
  - Vous juger qu'elle doit être persister dans une base de données
- Chercher la nature des associations entre les entités persistantes:
  - Relation père fils
  - Agrégation, composition
  - Plusieurs à plusieurs, relation porteuse d'information

Entités persistantes (Exemple)

- On souhaite développer une application de gestion de location des voitures. Une voiture est caractérisée par une matricule, marque et couleur.
- Un client, identifier par son CIN, nom et prénom peut louer des voitures, dans ce cas un contrat de location sera établit entre l'agence et ce dernier.
- Un contrat de location possède une durée de validité comprise entre deux dates

Entités persistantes (Solution)

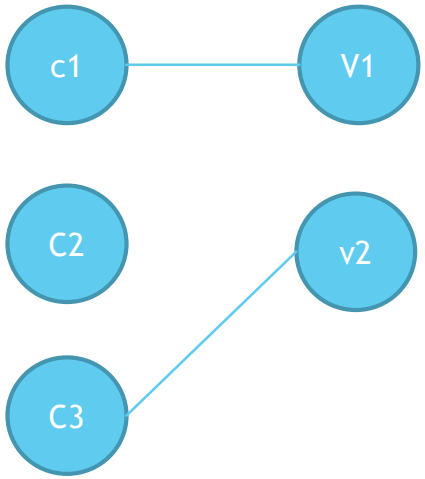


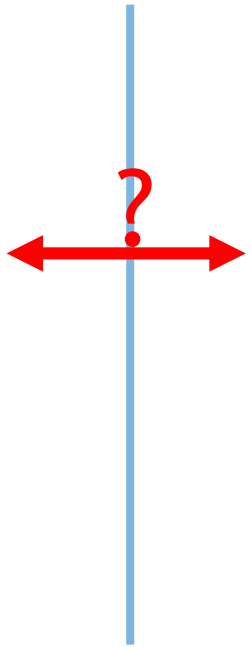

Contr
at1
Contr
at2
C1
V1
V2
C2
V3

Objet/Relational Mapping (ORM)

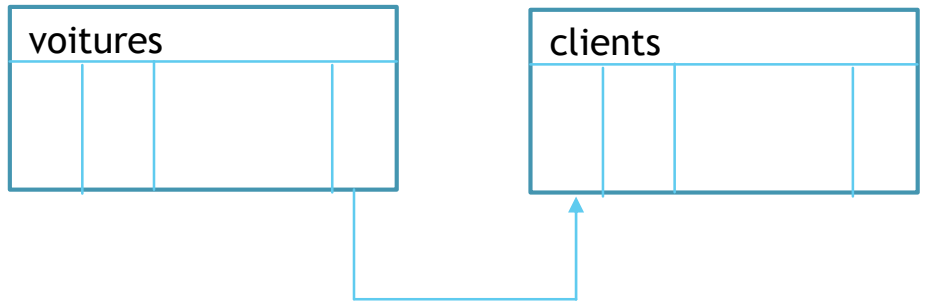
Problématique

Orienté objet





Relationnel



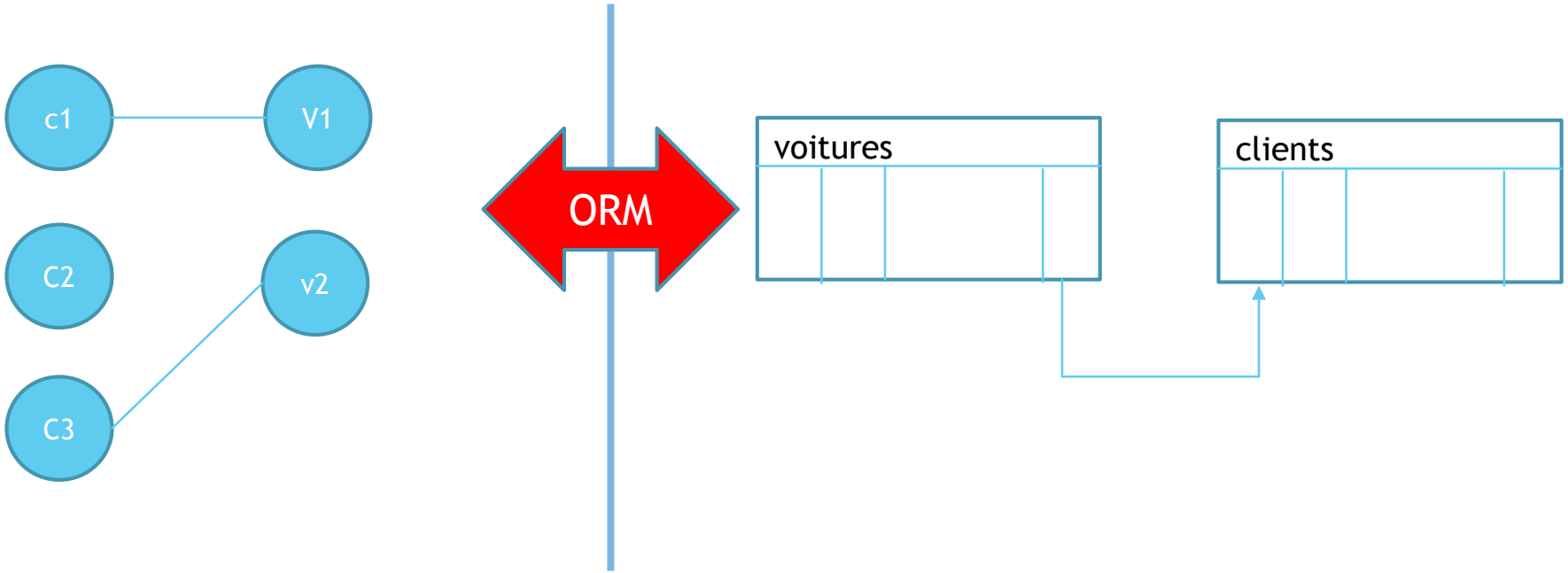
Solutions

- La conversion objet/Relationel suit des règles bien déterminées:
- Conversion manuelle entre objet/Relationel:
  - beaucoup de code,
  - Risque d'erreurs d'exécution
  - …
- Utiliser un ORM: composant logiciel qui s'interface entre une base de données relationnel et une application Orientée Objet
- Permet de simuler une base de données OO
- simplifie énormément le développement de la couche accès aux données

Solution

Orienté objet

Relationnel



ORM: implémentation

- Java: spécification JPA ➔ implémentation: Hibernate, TopLink,…
- .NET: Entity Framework, Nhibernate,…
- PHP5: Doctrine, Eloquent,…
- …

Eloquent

- Eloquent est un ORM inclut au framework Laravel
- Découvrons cet ORM par la pratique
- On souhaite ajouter l'entité voiture à notre application gestion-location-voitures
- Une voiture est caractérisée par
  - Un identifiant
  - Une marque
  - Un modèle
  - Une couleur

Démarche

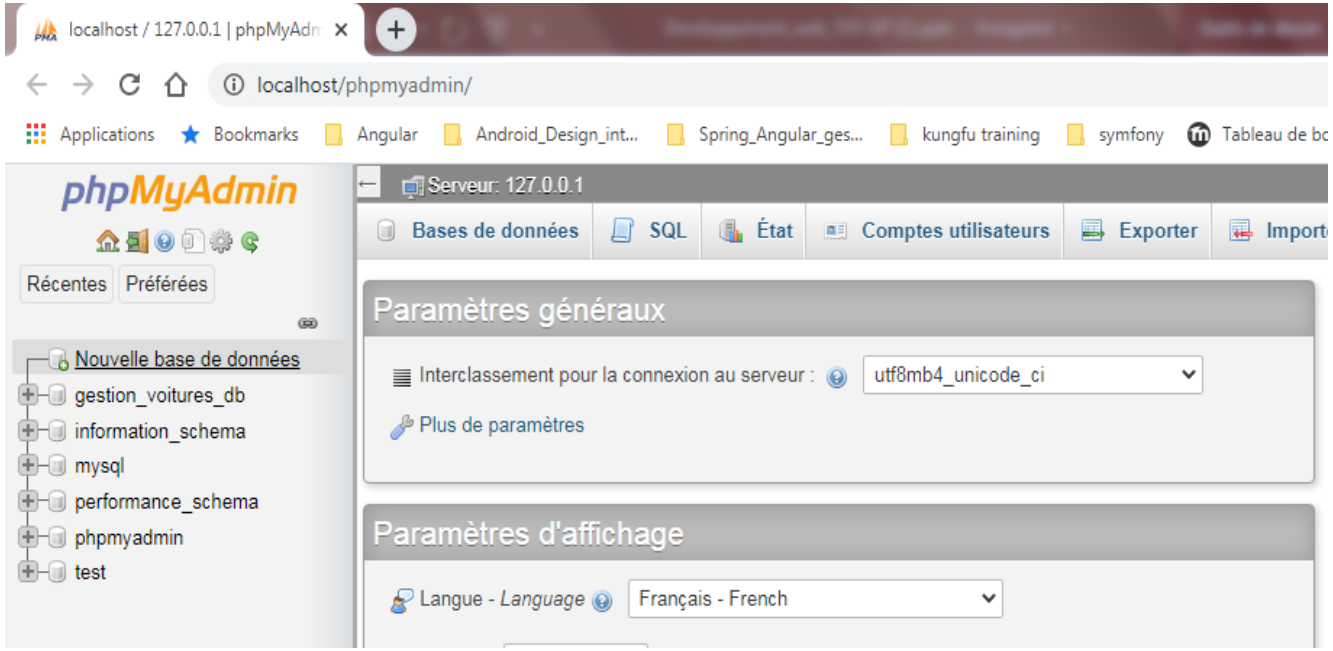
Étape par Étape

1) Créer une base de données **gestion_voitures_db** avec l'outil phpMyAdmin
2) Configurer la connexion à la base de données
3) Créer le modèle Voiture avec les fichiers de migration, factory associés
4) ajouter les attributs de l'entité voiture dans le fichier de migration
5) Créer la table voitures
6) Tester le modèle avec l'outil Tinker

Création de la base de données

Étape par Étape

1) À partir de l'interface de contrôle XAMPP, Démarrer le service MySQL
2) Cliquer sur le bouton Admin pour ouvrir l'interface d'administration phpMyAdmin



Configurer la connexion à la BD

Étape par Étape

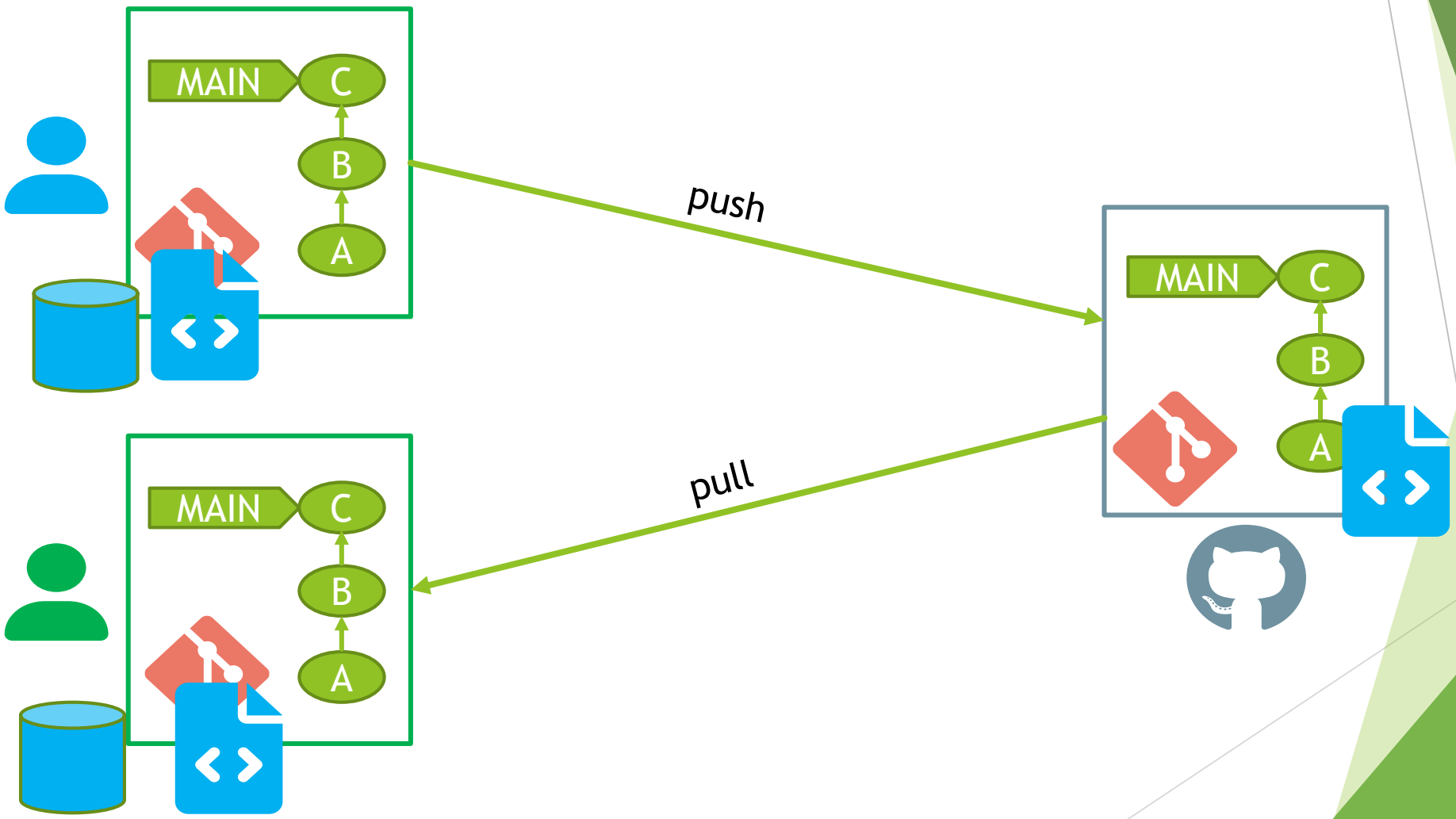
1) Ouvrir le fichier **.env** de l'application gestion-voitures
2) Compléter la configuration de la connexion :

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=gestion_voitures_db
DB_USERNAME=root
DB_PASSWORD=
```

3) Tester la connexion (exécuter une migration):

```
>php artisan migrate
```

Git: gestion de versions distribuée



Modèle Voiture

Étape par Étape

1) Créer la classe Voiture avec les fichiers de migration, factory associés:

```
>php artisan make:model Voiture -mfs
```

2) Définir les attributs de la classe Voiture dans le fichier de migration:

```
public function up()
    {
        Schema::create('voitures', function (Blueprint $table)
         {
            $table->id();
            $table->string('marque');
            $table->string('modele');
            $table->string('couleur');
            $table->string('photo');
            $table->timestamps();
        });
    }
```

Table voitures

Étape par Étape

1) Créer la table voitures:

```
>php artisan migrate
```

Astuces:

```
>php artisan migrate:rollback //annule la dernière migration

>php artisan migrate:rollback --step=3//annule les 3 dernières migrations

>php artisan migrate:rollback reset //annule toutes les migrations

>php artisan migrate --force //force la migration même si elle détruit les données existantes
```

Tester le modèle Voiture avec tinker

Étape par Étape

```
>php artisan tinker
>>>use App\Models\Voiture
>>>$voiture=new Voiture([
. . .'marque' => 'peugeot',
. . .'modele' => '206',
. . .'couleur' =>'bleu',
. . .'photo' => 'chemin_photo'
. . .]);
>>>$voiture->save();
=>true
>>>$voiture->toArray();
…
```

Affectation en masse (Mass Assignment)

- Pour des raisons de sécurité, Laravel interdit l'affectation de plusieurs attributs à une entité.

```
>>>$voiture=new Voiture([
. . .'marque' => 'peugeot',
. . .'modele' => '206',
. . .'couleur' =>'bleu',
. . .'photo' => 'chemin_photo'
. . .]);
>>>$voiture->save();
```

Sauvegarde interdite

Possibilité de modifier un attribut sensible: droit d'accès par exemple

- Il faut déclarer explicitement:
  - Les attributs autorisés dans le tableau $fillable
  - Ou bien, Les attributs interdits dans le tableau $guarded

Affectation en masse (Mass Assignment)

```
class Voiture extends Model
{
    use HasFactory;

    protected $guarded=[];

    /**Ou bien  */

    /* protected $fillable=[
        'marque',
        'modele',
        'couleur',
        'photo'
    ]; */

}
```

Tester la base de données: Factory et Seeder

- Laravel offre plusieurs outils pour tester la base de données:
- Les Fabriques de modèles (Model Factories) et les seeders permettent d'alimenter la base de données avec des données fictives.
- Exemple: modèle user

Exemple: Modèle User

```php
class UserFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    public function definition()
    {
        return [
            'name' => $this->faker->name(),
            'email' => $this->faker->unique()->safeEmail(),
            'email_verified_at' => now(),
            'password' => '$2y$10$92IXUNpkjO0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', // password
            'remember_token' => Str::random(10),
        ];
    }
```

Exemple: Modèle User

```
class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        // \App\Models\User::factory(10)->create();
    }
}
```

Modèle Voiture: implémentation du Factory

```php
class VoitureFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    public function definition()
    {
        return [
            'marque' => $this->faker->company(),
            'modele' => $this->faker->companySuffix(),
            'couleur' => $this->faker->colorName(),
            'photo' => $this->faker->imageUrl(640, 480, 'cars', true)
        ];
    }
}
```

Modèle Voiture: Seeder

```php
class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        Voiture::factory(10)->create();
    }
}
```

Modèle Voiture: exécuter le Seeder

>php artisan migrate:refresh

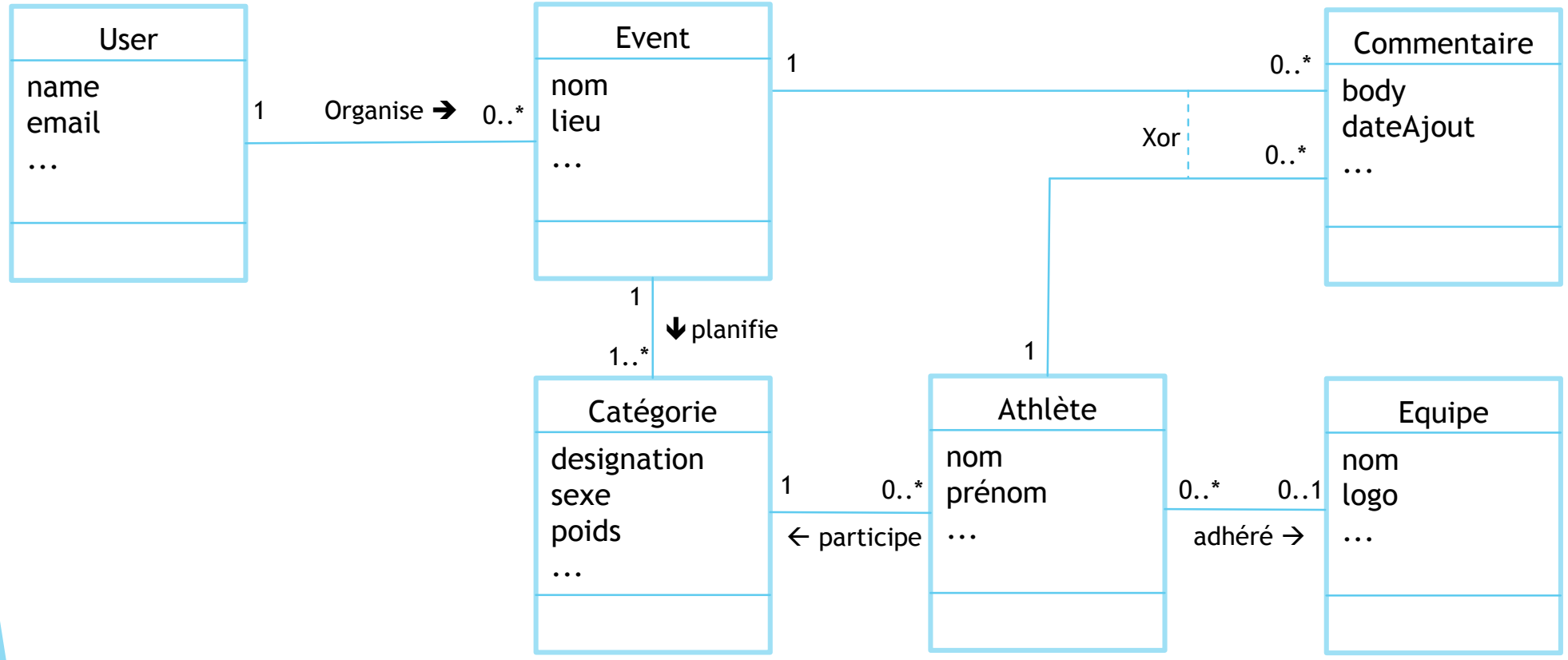
>php artisan db:seed

Vérifier la table voitures

Etude de cas: Gestion des événements sportifs

- On souhaite développer une application web de gestion des événements sportif (Sport de combat: TKD, Karaté,…).
- Un événement sportif est caractérisé par un nom, lieu de déroulement, un poster, une date de début et une date de fin.
- Les athlètes seront classés par catégories (Minim, Cadet,…poids, sexe)
- Un athlète peut être adhéré à une équipe sportive.
- Un athlète est caractérisé par un nom, prénom, sexe, photo,..
- Une équipe est caractérisée par un nom, logo,…
- un internaute peut ajouter un commentaire à un événement ou dans le profil d'un athlète.

Analyse: diagramme de classe



Implémentation de la couche modèle

Modèle, Association , Migration, Factory et Seeder

Démarche

Étape par Étape

- Création d'un nouveau projet laravel **gestion-events-sportif**
- Synchronisation du projet avec un dépôt distant sur **github**
- Création et configuration de l'accès à la base de données **events-sportif-db**
- Création des entités (EvenementSportif, Categorie,..)
- Implémentation du schéma de la base de données (sans associations)
- Première migration de la base de données
- Implémentation des associations coté Relationnel (ajout des contraintes d'intégrité)
- Implémentation des associations coté Modèle (ajout des méthodes d'accès aux attributs)
- Deuxième migration (modèle et associations)
- Test du modèle:
  - Implémentation des fabriques (Factories)
  - Implémentation des seeders pour alimenter la base de données avec des données fictives

Création d’un nouveau projet laravel

Étape par Étape

> composer create-project laravel/laravel gestion-events-sportif

- git init
- git add .
- git commit –m "Projet initial"
- git remote add origin https://token@gitub.com/...
- git push –u origin main

Création et config de la BD

Étape par Étape

Création des entités

Étape par Étape

- php artisan make:model EvenementSportif –mfs
- php artisan make:model Categorie –mfs
- php artisan make:model Equipe –mfs
- php artisan make:model Athlete –mfs
- php artisan make:model Commentaire –mfs

Fichiers de migration (Sans Relationship)

Étape par Étape

```
public function up()
    {
        Schema::create('evennement_sportifs', function (Blueprint $table) {
            $table->id();
            $table->string('nom', 100);
            $table->text('description');
            $table->string('lieu', 100);
            $table->string('poster');
            $table->date('dateDebut');
            $table->date('dateFin');
            $table->timestamps();
        });
    }
```

Fichiers de migration (Sans Relationship)

Étape par Étape

```
public function up()
    {
        Schema::create('categories', function (Blueprint $table) {
            $table->id();
            $table->string('nom',100);
            $table->enum('sexe',['HOMME','FEMME']);
            $table->string('poids',20);
            $table->timestamps();
        });
    }
```

Fichiers de migration (Sans Relationship)

Étape par Étape

```
public function up()
    {
        Schema::create('equipes', function (Blueprint $table) {
            $table->id();
            $table->string('nom',100);
            $table->string('logo')->nullable();
            $table->timestamps();
        });
    }
```

Fichiers de migration (Sans Relationship)

Étape par Étape

```
public function up()
    {
        Schema::create('athletes', function (Blueprint $table) {
            $table->id();
            $table->string('nom',60);
            $table->string('prenom',60);
            $table->enum('sexe',['HOMME','FEMME']);
            $table->string('photo');
            $table->integer('score')->default(0)->unsigned();
            $table->timestamps();
        });
    }
```

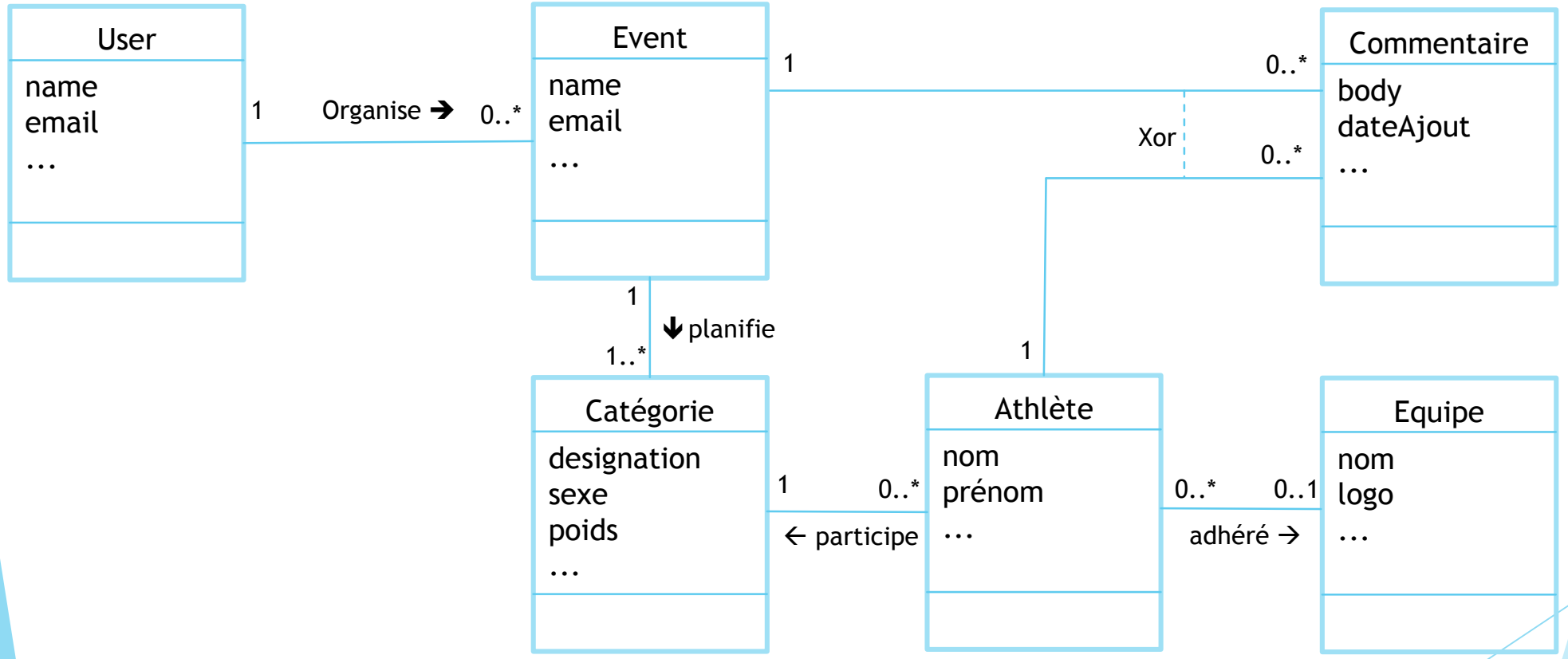
Fichiers de migration (Sans Relationship)

Étape par Étape

```
public function up()
    {
        Schema::create('commentaires', function (Blueprint $table)
{
            $table->id();
            $table->text('body');
            $table->timestamp('dateAjout');
            $table->timestamps();
        });
    }
```

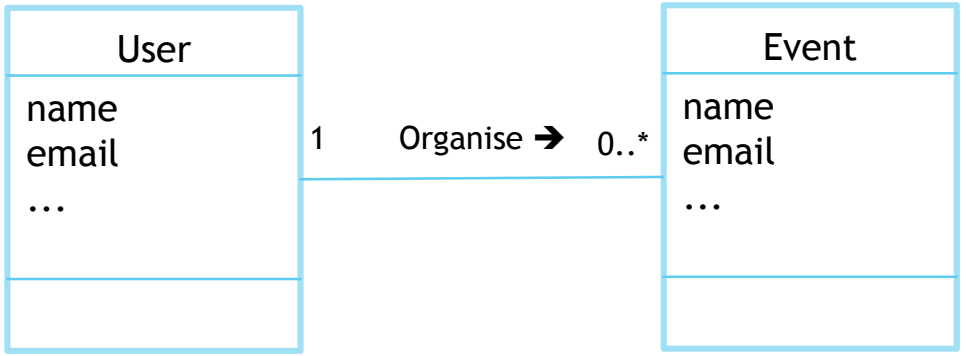
Implémentation des associations

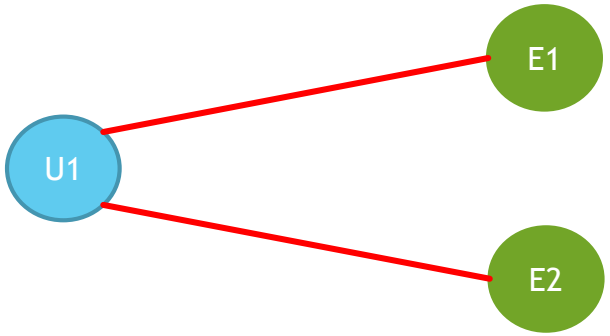
Étape par Étape

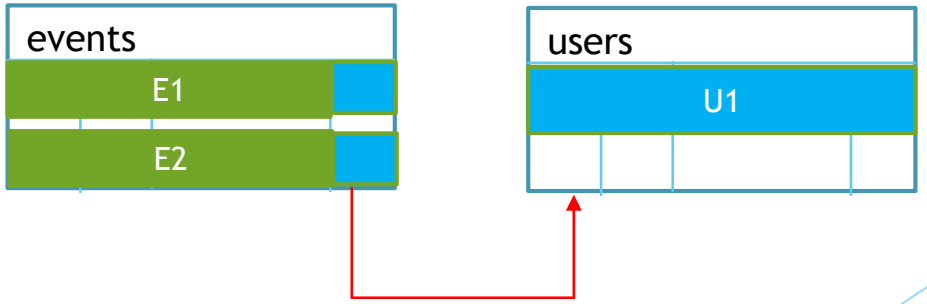


User<->Event: One-To-Many

Étape par Étape







User<->Event: One-To-Many

Étape par Étape

```
public function up()
    {
        Schema::create('evennement_sportifs', function (Blueprint $table) {
            $table->id();
            $table->string('nom', 100);
            $table->text('description');
            $table->string('lieu', 100);
            $table->string('poster');
            $table->date('dateDebut');
            $table->date('dateFin');
            $table->foreignId('user_id')->constrained()->onDelete('cascade');
            $table->timestamps();
        });
    }
```

User<->Event: One-To-Many

Étape par Étape

```
class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;
    //...


    public function evennementSportifs(){

        return $this->hasMany(EvennementSportif::class);

    }
}
```

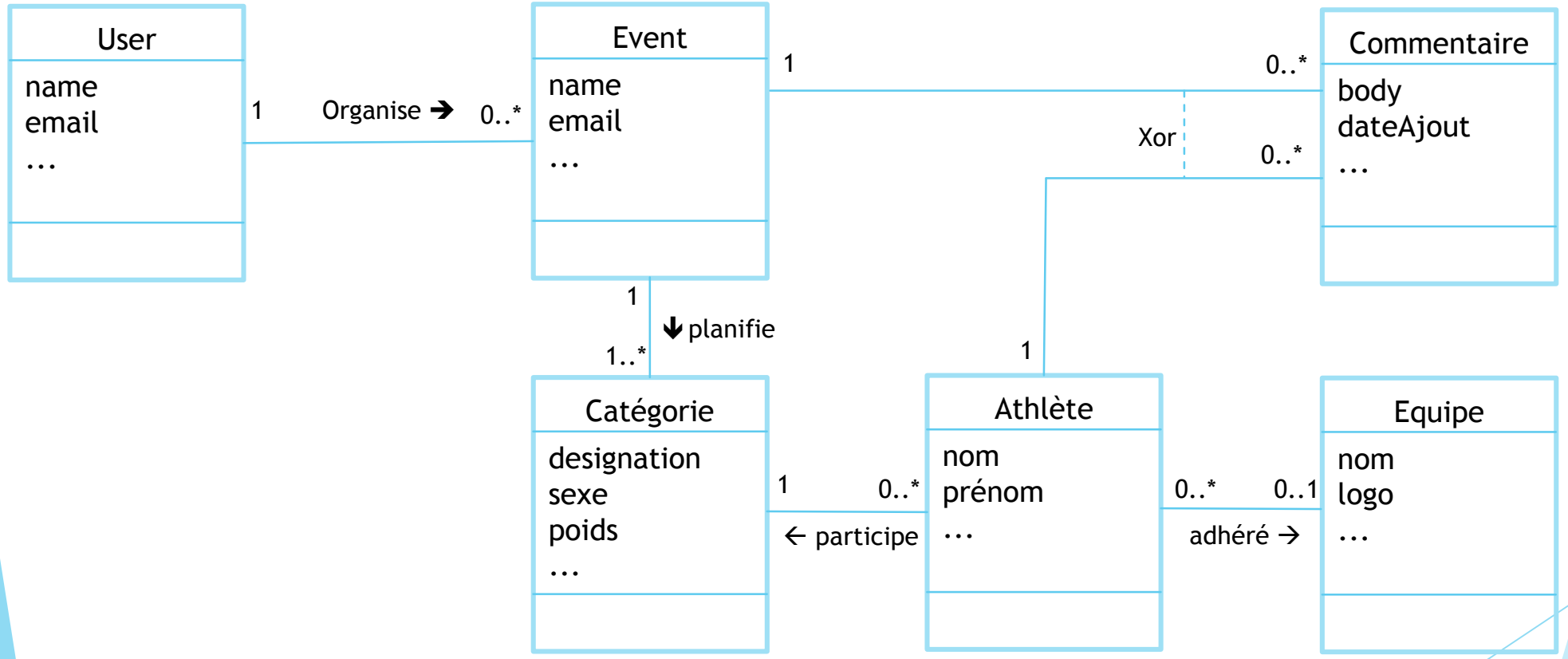
User<->Event: One-To-Many

Étape par Étape

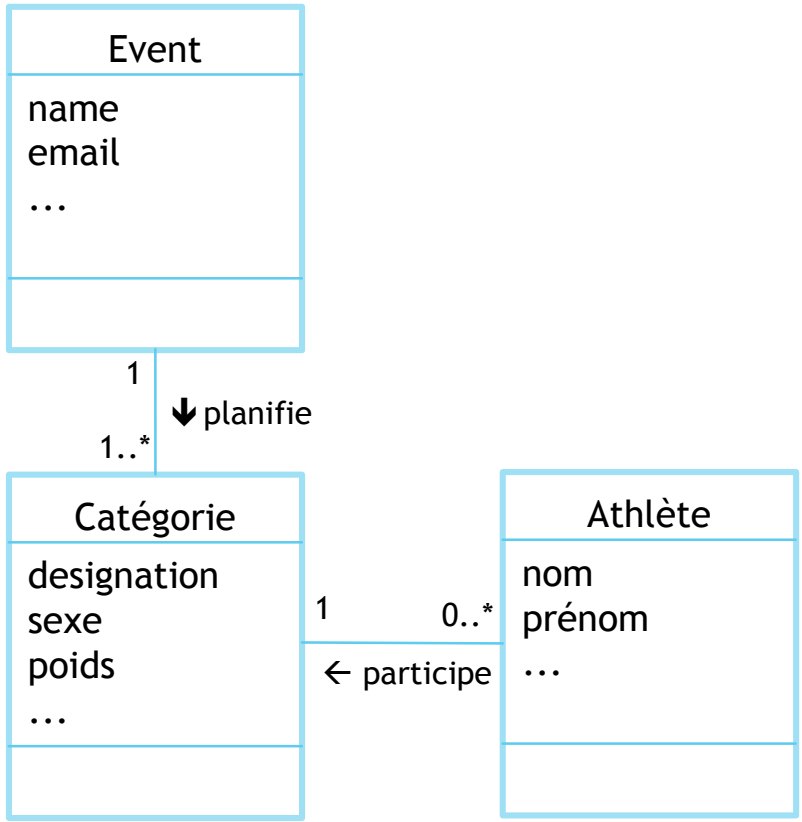
```
class EvennementSportif extends Model
{
    use HasFactory;

    public function organisateur(){

        return $this->belongsTo(User::class);
    }
}
```

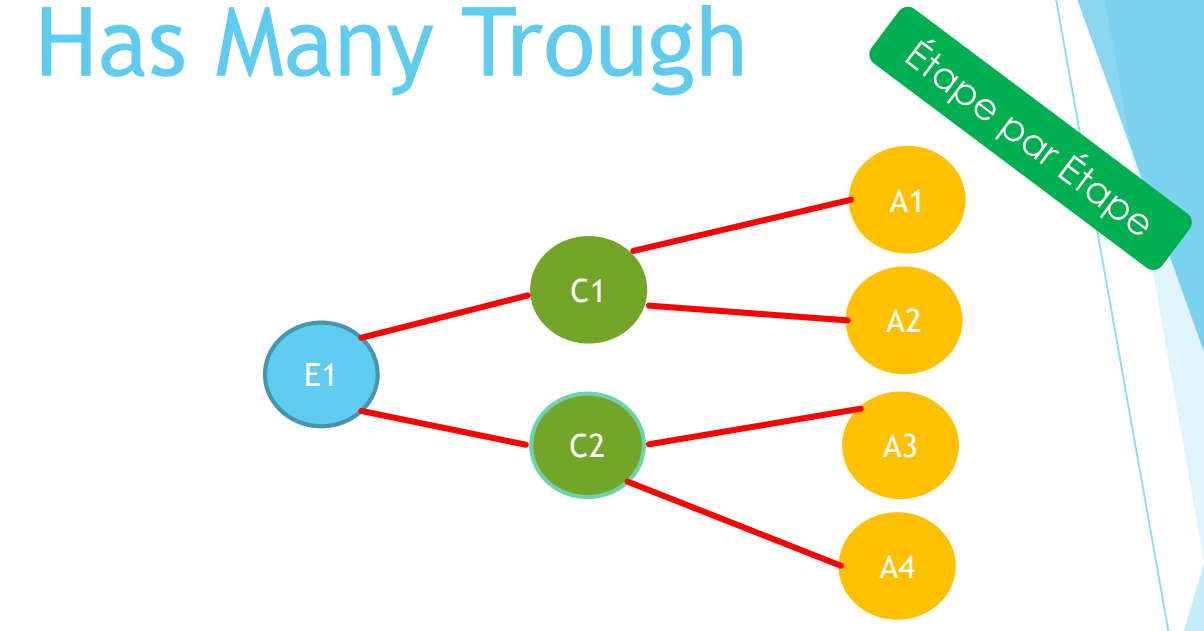
Implémentation des associations

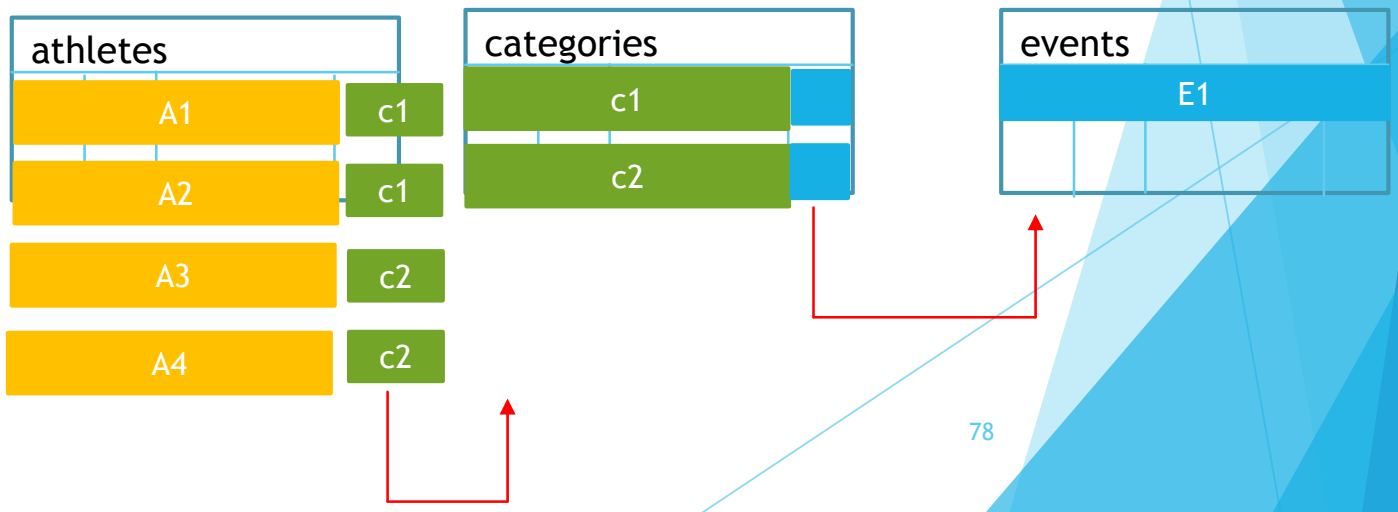
Étape par Étape



Event <-> Athlete: Has Many Trough







Event <-> Athlete: Has Many Trough

Étape par Étape

```
public function up()
    {
        Schema::create('categories', function (Blueprint $table) {
            $table->id();
            $table->string('nom',100);
            $table->enum('sexe',['HOMME','FEMME']);
            $table->string('poids',20);
            $table->foreignId('evennement_sportif_id')->constrained()->cascadeOnDelete();
            $table->timestamps();
        });
    }
```

Event <-> Athlete: Has Many Trough

Étape par Étape

```
public function up()
    {
        Schema::create('athletes', function (Blueprint $table) {
            $table->id();
            $table->string('nom',60);
            $table->string('prenom',60);
            $table->enum('sexe',['HOMME','FEMME']);
            $table->string('photo');
            $table->integer('score')->default(0)->unsigned();
            $table->foreignId('categorie_id')->nullable()->constrained()->cascadeOnDelete();
            $table->foreignId('equipe_id')->nullable()->constrained()->nullOnDelete();
            $table->timestamps();
        });
    }
```

Event <-> Athlete: Has Many Trough

Étape par Étape

```
class EvennementSportif extends Model
{
    use HasFactory;

    public function categories(){

        return $this->hasMany(Categorie::class);

    }

    public function athletes(){

        return $this->hasManyThrough(Athlete::class,Categorie::class);

    }
}
```

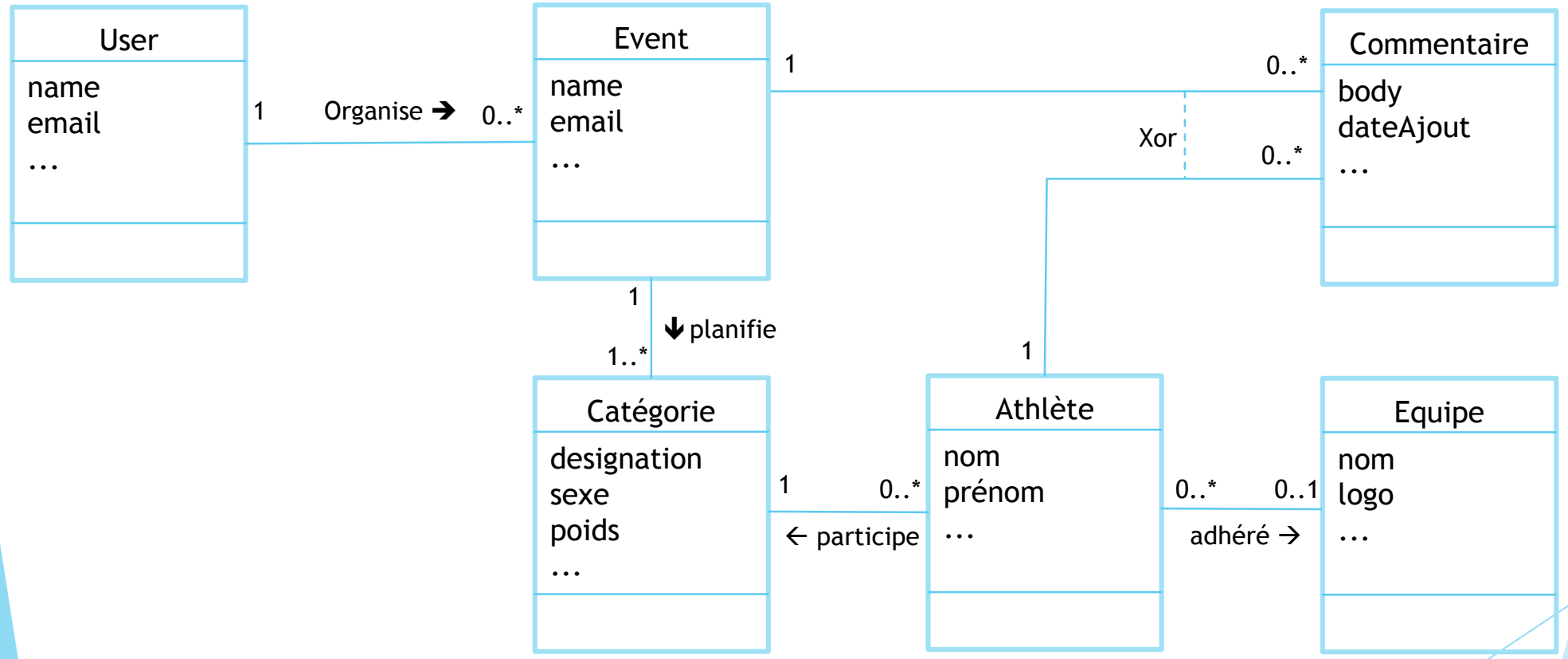
Event <-> Athlete: Has Many Trough

Étape par Étape

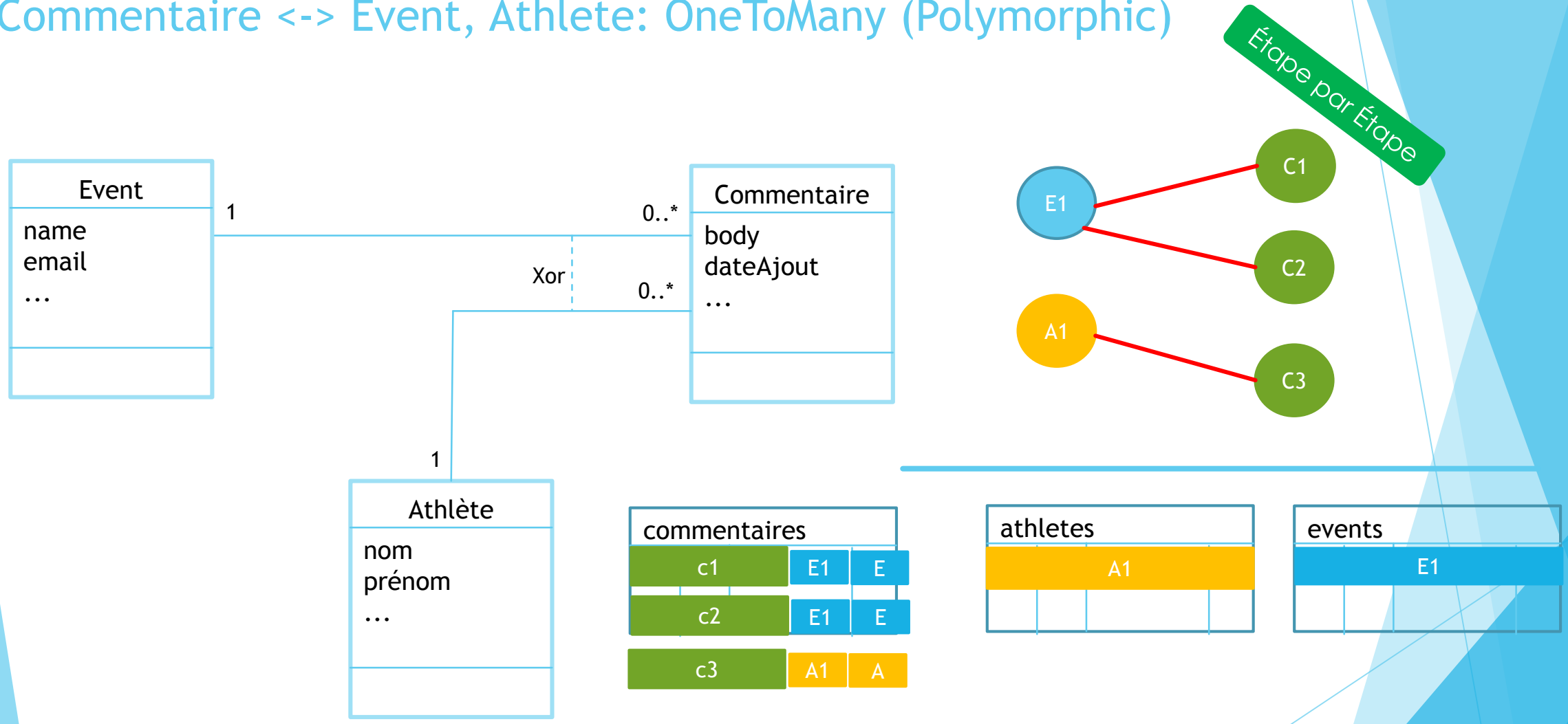
```php
class Categorie extends Model
{
    use HasFactory;

    public function evennementSportif(){

        return $this->belongsTo(EvennementSportif::class);
    }

    public function athletes(){
        return $this->hasMany(Athlete::class);
    }
}
```

Implémentation des associations

Étape par Étape



Commentaire <-> Event, Athlete: OneToMany (Polymorphic)



Commentaire <-> Event, Athlete: OneToMany (Polymorphic)

Étape par Étape

```
public function up()
    {
        Schema::create('commentaires', function (Blueprint $table) {
            $table->id();
            $table->text('body');
            $table->timestamp('dateAjout');
            $table->unsignedBigInteger('commentable_id');
            $table->String('commentable_type');
            $table->timestamps();
        });
    }
```

Commentaire <-> Event, Athlete: OneToMany (Polymorphic)

Étape par Étape

```
class Commentaire extends Model
{
    use HasFactory;

    public function commentable(){
        return $this->morphTo();
    }
}
```

Commentaire <-> Event, Athlete: OneToMany (Polymorphic)

Étape par Étape

```php
class EvennementSportif extends Model
{
    use HasFactory;



    public function commentaires(){
        return $this->morphMany(Commentaire::class,'commentable');
    }
}
```

Commentaire <-> Event, Athlete: OneToMany (Polymorphic)

Étape par Étape

```php
class athlete extends Model
{
    use HasFactory;

    public function commentaires(){
        return $this->morphMany(Commentaire::class, 'commentable');
    }
}
```

Deuxième migration (modèle et associations)

Étape par Étape

- php artisan migrate:refresh

Test du modèle: factory

Étape par Étape

```
class EvennementSportifFactory extends Factory
{
        public function definition()
    {
         return [
             'nom' => $this->faker->sentence(),
             'description' => $this->faker->words(2,true),
             'lieu'=> $this->faker->state(),
             'poster' => $this->faker->imageUrl(360, 360, true),
             'dateDebut' => $this->faker->date(),
             'dateFin' => $this->faker->date(),
         ];
    }
}
```

Test du modèle: factory

Étape par Étape

```php
class EquipeFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    public function definition()
    {
        return [
            'nom' =>$this->faker->company(),
            'logo' => $this->faker->imageUrl(360, 360, true),
        ];
    }
}
```

Test du modèle: factory

Étape par Étape

```
class AthleteFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    public function definition()
    {
        return [
            'nom' => $this->faker->firstName(),
            'prenom' => $this->faker->lastName(),
            'photo' => $this->faker->imageUrl(360,360,true)
        ];
    }
}
```

Test du modèle: factory

Étape par Étape

```php
class CommentaireFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    public function definition()
    {
        return [
            'body' => $this->faker->words(3,true),
            'dateAjout' =>now()
        ];
    }
}
```

Test du modèle: Seeder

Étape par Étape

```
class UserSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        User::factory(2)
        ->has(EvennementSportif::factory()
                ->count(1)
```

Test du modèle: Seeder

Étape par Étape

```
->has(EvennementSportif::factory()
                ->count(1)
                ->has(Categorie::factory()
                        ->count(3)
                        ->state(new Sequence(
                            ['nom' => 'Minim'],
                            ['nom' => 'Cadet'],
                            ['nom' => 'Senior'],
                        ))
                        ->state(new Sequence(
                            ['sexe' => 'HOMME'],
                            ['sexe' => 'FEMME'],
                        ))
                        ->state(new Sequence(
                            ['poids' => '-40 KG'],
                            ['poids' => '-50 KG'],
                            ['poids' => '+50 KG'],
                        ))
```

Test du modèle: Seeder

Étape par Étape

```
->has(athlete::factory()
        ->count(2)
        ->state(function (array $attributes, Categorie $categorie) {
                        return ['sexe' => $categorie->sexe];
                })
                ->for(Equipe::factory())
                ->hasCommentaires(2)

    )

 )
 ->hasCommentaires(2)

 )
 ->create();
    }
}
```

Test du modèle: Seeder

Étape par Étape

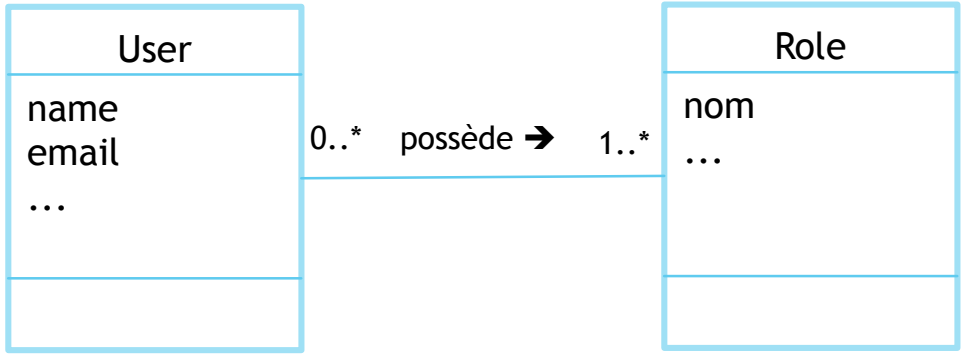
```php
class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        $this->call([
            UserSeeder::class,
            //EvennementSportifSeeder::class,
            //CategorieSeeder::class,
            //EquipeSeeder::class,
            //AthleteSeeder::class,
            //CommentaireSeeder::class
        ]);
    }
}
```

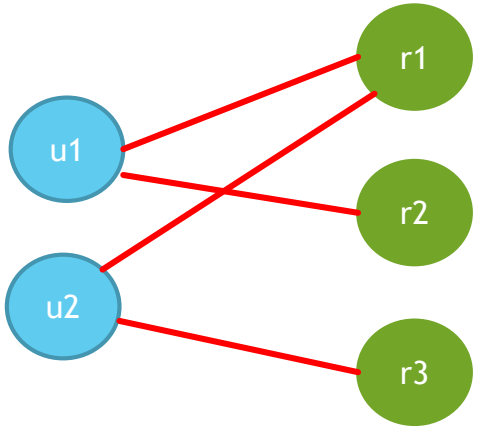
Test du modèle: Seeder

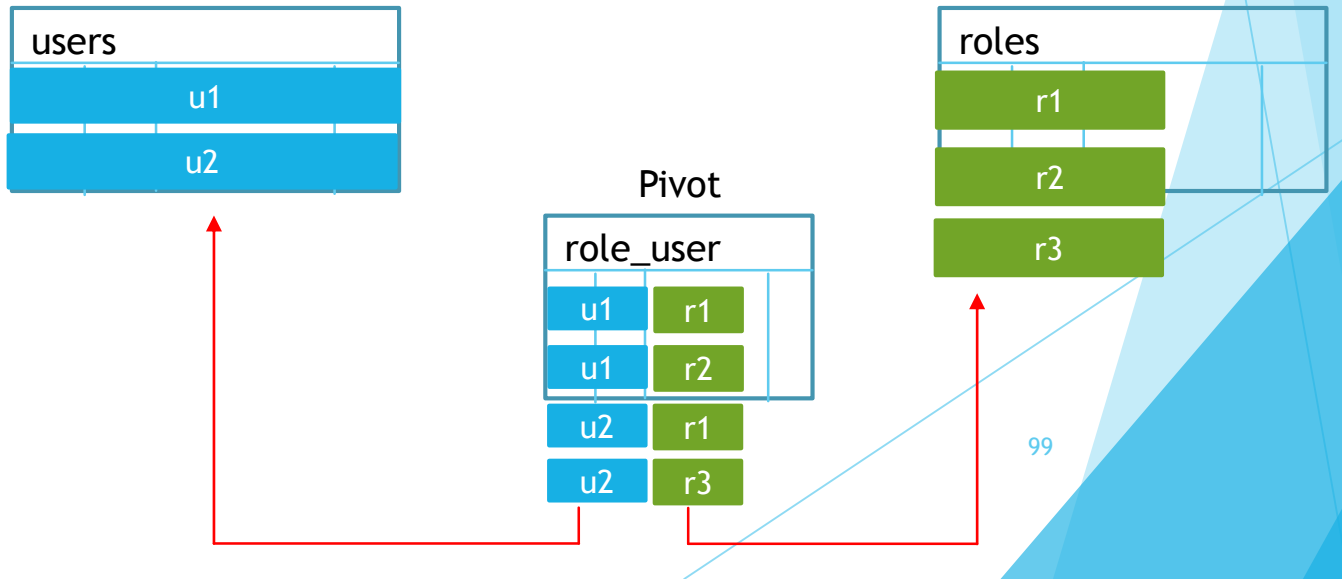
Étape par Étape

- php artisan migrate:refresh
- Php artisan db:seed

Association ManyToMany







Démarche

Étape par Étape

- Créer une nouvelle application web Laravel RelationShips-test
- Configurer la base de données
- Créer le modèle **Role**
- Créer une migration pour la table pivot **role_user**
- Définir le schéma de la table pivot **role_user**
- Définir les associations dans les modèles **User** et **Role**
- Définir les **factories** et les **seeders**
- Alimenter la base de données en exécutant le seeder

Créer le modèle Role

Étape par Étape

- php artisan make:model Role -mfs

```
Schema::create('roles', function (Blueprint $table) {
            $table->id();
            $table->string('nom');
            $table->timestamps();
        });
```

Créer la migration role_user

Étape par Étape

- php artisan make:migration create_role_user

```
Schema::create('role_user', function (Blueprint $table) {
            $table->id();
            $table->foreignId('user_id')->constrained();
            $table->foreignId('role_id')->constrained();
            $table->timestamps();
        });
```

Définir les associations dans les modèles

Étape par Étape

```php
class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;
    //...


    public function roles(){
        return $this->belongsToMany(Role::class, 'role_user','user_id','role_id')
        ->withTimestamps();
    }
}

class Role extends Model
{
    use HasFactory;

    public function users(){
        return $this->belongsToMany(User::class);
    }
}
```

Définir le Seeder: façon 1

Étape par Étape

```
User::factory(2)->has(Role::factory()->count(2)
                                ->state(new Sequence(
                                    [
                                        'nom' => 'Admin',
                                        'nom' => 'User',
                                        'nom' => 'Manager'

                                    ]
                                ))
                            )
                        ->create();
```

Définir le Seeder: façon 2

Étape par Étape

```
$roles=Role::factory(2)->state(new Sequence(
            [
                'nom' => 'Admin',
                'nom' => 'User',
                'nom' => 'Manager'

            ]))
            ->create();
$users=User::factory()->count(3)->hasAttached($roles)->create();
```

Définir le Seeder: façon 3

Étape par Étape

```
$roles=Role::factory(2)->state(new Sequence(
            [
                'nom' => 'Admin',
                'nom' => 'User',
                'nom' => 'Manager'

            ]))
            ->create();
 $users=User::factory()->count(3)->hasAttached(Role::inRandomOrder()->limit(2)->get())
                    ->create();
```

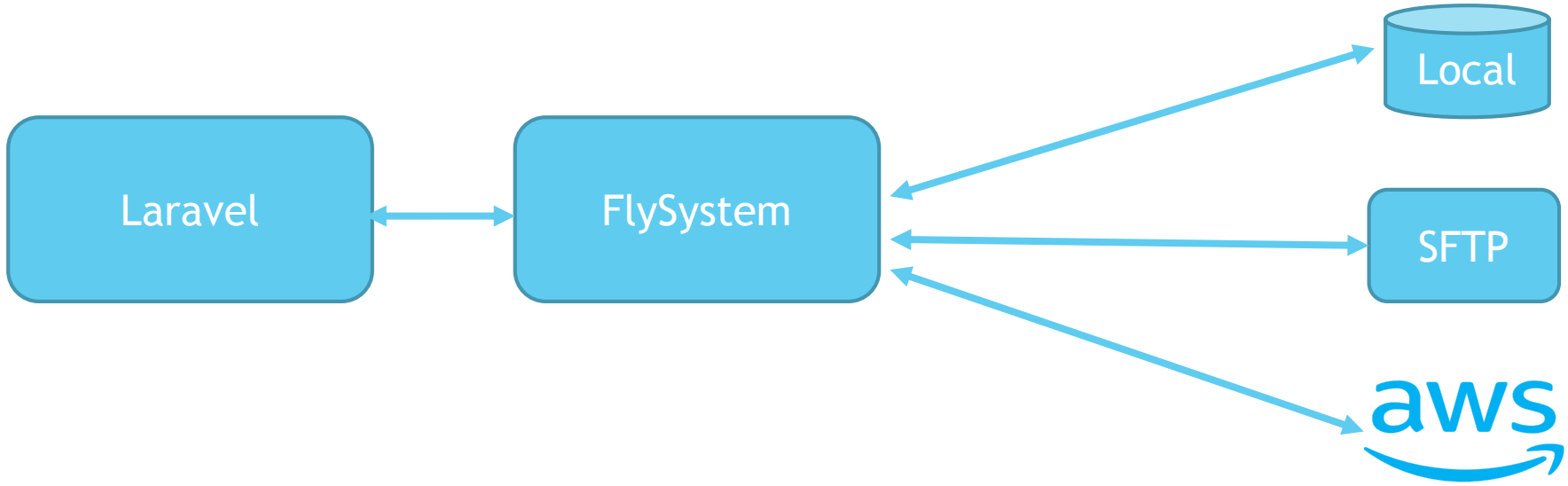
Appliquer le Seeder

Étape par Étape

- php artisan db:seed

File Storage

- Laravel offre une API simple et puissant pour la gestion des fichiers



Configuration: config/filesystems.php

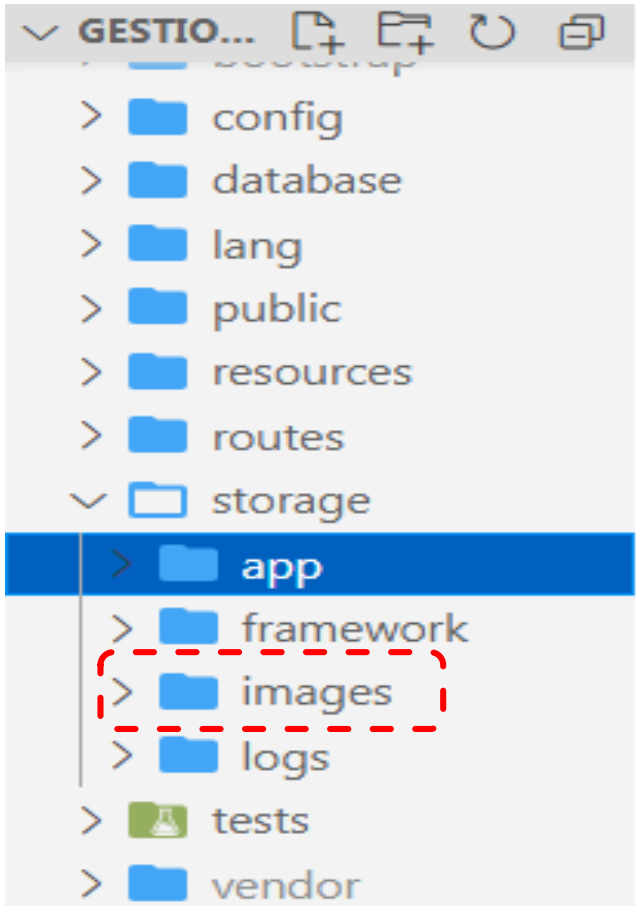
Générer les images: Démarche

Étape par Étape

- Créer le dossier storage/images
- Utiliser la bibliothèque **faker** pour générer et enregistrer les images
- Appliquer de nouveau le Seeder

Créer dossier storage/images

Étape par Étape



Athlète: Générer les images avec faker

Étape par Étape

```
public function definition()
    {
        $width=200;
        $height=200;
        $path=$this->faker->image('storage/images',$width,$height,'person',true,true,'person',false);
        return [
            'nom' => $this->faker->firstName(),
            'prenom' => $this->faker->lastName(),
            'photo' => $path,//$this->faker->imageUrl(360,360,true)
        ];
    }
```

Event: Générer les images avec faker

Étape par Étape

```php
public function definition()
    {
        $width=640;
        $height=800;
        $path=$this->faker->image('storage/images',$width,$height,'sport',true,true,'sport',false);
        return [
            'nom' => $this->faker->sentence(),
            'description' => $this->faker->words(2,true),
            'lieu'=> $this->faker->state(),
            'poster' =>$path,// $this->faker->imageUrl(360, 360, true),
            'dateDebut' => $this->faker->date(),
            'dateFin' => $this->faker->date(),
        ];
    }
```