Base de données

- Toute application web moderne utilise une base de données

- L'interaction avec la base se fait en utilisant la façade DB (requêtes SQL), constructeur de requêtes ou le modèle éloquent

- Laravel supporte 05 bases de données: MariaDB, MySQL, PostgreSQL, SQLServer, SQlite

Base de données

Configuration

- La configuration d'une base de données s'effectue dans les fichiers config/database.php et .env

- Dans le premier fichier, il est possible de définir **toutes** les connexions et spécifier une connexion par **défaut**

❑ Requêtes SQL

- L'exécution des requêtes peut se faire avec la façade DB
- Possède des méthodes pour chaque type de requête tels que select, insert, update, delete, and statement

Méthode SELECT

- Permet la récupération des enregistrements de la base

Ex. $res = DB::select('SELECT * FROM users WHERE ville = ? AND age > ?', [‘Paris', 25]);

- Cette méthode retourne un tableau de valeurs

- La méthode select est remplacée par la méthode scalar lorsque le résultat retourné est une **valeur scalaire**

Ex. $res=DB::scalar(“select count(*) from users”);

Méthode INSERT

- Elle permet d'insérer des enregistrements dans la table de la base

Ex. DB::insert('insert into users (ville, age) values (?, ?)', ['Tour', 30]);

Méthode UPDATE

- Elle permet de mettre à jour des enregistrements d'une table de la base

Ex. $res=DB::update("update users set ville= 'Lyon' where ville= ?", ['Tour']);

- $res reçoit le nombre de lignes affectées par cette mise à jour

- Méthode DELETE

- Elle permet de supprimer des enregistrements d'une table de la base

Ex. $res= DB::delete('delete from users')

- Méthode STATEMENT

- Certaines instructions SQL ne retournent aucune valeur

Ex. $res= DB::statement(drop table users‘)

- Méthode CONNECTION

- Elle permet l'accès à une connexion **spécifique** lorsque **plusieurs** connexions sont définies

Ex. $res= DB::connection('mysql')->select('select * from users where id = ?', [10]);

Quelques commandes sur les BD

- php artisan bd [nomconnextion]: connexion à une base de données

Ex. php artisan db sqlite

- php artisan db:show: affichage d'un aperçu de la base de données

- php artisan bd:table nomtable: affichage de la structure d'une table donnée

- etc

Migration

- Gère la structure de la base de données
- facilite la collaboration avec d'autres développeurs
- La création d'une migration se fait avec la commande artisan make:migration

  Ex. php artisan make:migration create_étudiants_table

  => crée la table étudiant

- Le fichier généré sera placé dans le répertoire database/migrations

- Chaque **nom** de fichier contient une estampille qui permet à Laravel de déterminer l'ordre des migrations

- Editer le fichier créé pour définir la structure de la base. Les migrations utilisent Schema pour créer et modifier les tables et les colonnes de la base de données
- Exécuter la migration et créer la table dans la base de données avec la commande

  php artisan migrate

- Laravel utilise le nom de la migration pour tenter de deviner le nom de la table
- La structure de la migration contient deux méthode:

- up() : ajouter de nouvelles tables, colonnes et index à la base de données

- down(): assurer les opérations inverses de up()

Ex.
```
public function up(): void  {
    Schema::create('taches', function (Blueprint $table) {
      $table->id();
      $table->string('titre');
      $table->timestamps();
      });   }
   public function down(): void  {
    Schema::dropIfExists('taches');  }
```

- Pour connaitre l'ensemble des migrations qui ont été exécutées, on utilise la commande

  php artisan migrate:status
- L'annulation de la dernière migration se fait avec

  php artisan rollback
- Pour annuler N migrations à la fois, on exécute

  php artisan migrate:rollback --step=N
- L'annulation de toutes les migrations se fait avec

  php artisan migrate:reset

- La commande php artisan migrate:refresh permet de réinitialiser la base de données en **annulant** toutes les migrations et en les **exécutant** à nouveau.

- La commande php artisan migrate:refresh --step=N annule et exécute à nouveau les N dernières migrations

- La commande php artisan migrate:fresh supprime toutes les tables de la base de données et exécute la commande migrate

Schéma de la base de données

❑ Table

- **Création :** Pour créer une nouvelle table, on utilise la méthode create de la façade Schema qui prend deux arguments: le nom de la table et une closure
- Ex. Schema::create('users', function (Blueprint $table) {…}

- **Mise à jour** se fait avec la méthode table qui prend aussi deux arguments
- Ex. Schema::table('users', function (Blueprint $table) {…}

- **Suppression d'une table** se fait avec la méthode drop ou dropIfExists

  Ex. Schema::drop('users');

  Schema::dropIfExists('users');

- **Renommage d'une table** se fait avec la méthode rename qui prend deux arguments: l'ancien et le nouveau nom

Ex. Schema::rename($ancien, $nouveau);

❑ Colonne

- **Création d'une colonne s**e fait comme suit:

  $table->typeCol(nomCol'); où typeCol, nomCol désigne le type, le nom de la colonne

- Ex. Types de colonne: integer, string, float, increments, id, etc

- **Maj des colonnes** se fait avec la méthode change
- Ex. $table->string('adresse', 50)->change();

- **Renommage** se fait avec renameColumn
- Ex. $table->renameColumn('ancien', 'nouveau');

- **Suppression** des colonnes se fait avec dropColumn
- Ex. $table->dropColumn('adresse');

Index

- **Création d'index**
- $table->primary('Id_produit');
- $table->primary('Id_produit', 'Id_commande');
- $table->unique('matricule');
- $table->index('matricule');
- $table->foreign('refClient')->references('id')->on('clients'); // clé étrangère

- NB: primary() n'est pas nécessaire si increments() ou bigIncrements() sont utilisées.

- **Suppression d'index**

- $table->dropPrimary('Id_produit');
- $table->dropUnique('matricule');
- $table->dropIndex('matricule');
- $table->dropForeign('[refClient'])

Modèle éloquent

- Eloquent ORM est un ensemble d'outils permettant d'interagir avec la base de données
- Un modèle interagit avec la base de données
- Chaque table possède un modèle (classe)

- Un modèle joue un rôle pivot pour construire des relations entre les tables

- Cette classe peut à la fois interagir avec **toute** la table et avec **chacun** de ses enregistrements

- Laravel permet de construire des modèles puissants avec un minimum de code

- La création d'un modèle se fait avec la commande artisan make:model

  Ex. php artisan make:model NomModèle

- Ceci génère le code suivant dans le fichier app/models/NomModèle.php

```
...
use Illuminate\Database\Eloquent\Model;
class NomModèle extends Model{...}
```

- Il est possible de créer une migration lors de la création d'un modèle avec les options –m ou –migration

Ex. php artisan make:model NomModèle –m

=> Génère à la fois un **modèle** et une **migration**

- Laravel donne à une table le nom du modèle correspondant au pluriel avec la première lettre au minuscule

  Ex. Nom du **modèle** : Produit

  Nom de la **table**: produits

- Possible de personnaliser le nom de la table dans le modèle avec la propriété $table

  Ex. protected $table='articles'

- Par défaut, à chaque table est affectée une clé primaire sous forme d'un entier nommé id qui s'auto incrémente

- La clé primaire peut être changée avec la propriété $primarykey

  Ex. protected $primarykey='id_article'

- La propriété d'auto-incrémentation peut être désactivée avec

  public $incrementing=false;

- Eloquent ajoute une estampille à chaque table. Cette fonctionnalité peut être désactivée avec

```
public $timestamps=false;
```

ou personnaliser son format avec:

```
protected $dateformat='U'; // date en secondes (estampille d'Unix)
```

Récupération des données

- **Tous** les enregistrements d'une table avec la méthode all() du modèle éloquent

  Ex. $article=Article::all()

- Un **seul** enregistrement avec les méthodes first(), firstOrFail(), find() et findOrFail()

  Ex. $article=Article::findOrFail(1)

- Plusieurs enregistrements avec la méthode get()

Insertion des données

Il existe deux façons pour insérer un enregistrement:

- Créer une instance de la classe éloquent
- Affecter manuellement les valeurs aux propriétés
- Appeler la méthode save()

Ex.

```
$client=new Client;
$client->prénom='Jean';
$client->save();
```

- Avec la méthode create du modèle éloquent

Ex. $client=Client::create([

'prénom'=>'Jean']);

MAJ des données

- Il en existe deux approches:

❑ Récupérer l'enregistrement à mettre à jour, le modifier et sauvegarder les modifications

Ex. $client=Client::find(10);

$client->adresse='Paris';

$client->save();

NB: le champ updated_at prendra une nouvelle valeur

❑ avec la méthode update qui prend en paramètre un tableau

$client=Client::find(10);

$client->update(['adresse'=>'Paris']);

Suppression des données (1/2)

- **Suppression physique**

```
Ex. $client=Client::find(10);
    $client->delete();
```

- ou avec l'ID

```
Ex. Client::destroy(1);
    Client::destroy(['1','2']);
```

Suppression des données (2/2)

❑ Suppression logique

- Marque seulement l'enregistrement comme supprimé sans le faire réellement dans la table

  => possibilité de restauration

- **Etapes**:
  - Ajouter softDeletes à la **migration**: $table->softDeletes();
  - Importer softDeletes dans le **modèle**: use softDeletes;
  - Ajouter **deleted_at** à la **$date**: $date=['deleted_at'];

- La méthode trashed() est utilisée pour savoir si un enregistrement a été supprimé

- La restauration d'un enregistrement supprimé se fait avec la méthode restore()

  Ex. $client->restore();

- La suppression définitive se fait avec a méthode forceDelete()

Relations

- Une **relation** est définie comme une méthode dans un modèle éloquent
- Dans un modèle relationnel, on trouve des tables reliées entre elles
- Eloquent fournit des outils simples et puissants pour relier les tables
- Il existe plusieurs types de relations :
  - One to one
  - One to many
  - Many to many

Relations

- One to one:

Chaque enregistrement dans la première table est associé à un seul enregistrement dans la seconde table, et vice versa

Ex. Un client possède un seul numéro de téléphone (hasOne())

```
class Client extends Model{
        public function numTelephone() {
                return this-> hasOne(NumTelephone::class);}}
```

Ex. Un numéro de téléphone est associé à un seul client (belongsTo())

```
class NumTelephone extends Model{
        public function client() {
        return this-> belongsTo(Client::class);}}
```

Exemples ??

Relations

- One to many:

- Chaque enregistrement dans la première table est associé à plusieurs enregistrements dans la seconde table
- Chaque enregistrement dans la seconde table est associé à un **seul** enregistrement dans la première table

Ex. Un client peut passer plusieurs commandes (hasMany())

```
class Client extends Model{
        public function commande() {
                return this-> hasMany(Commande::class);}}
```

Ex. Chaque commande est associée à un seul client (belongsTo())

```
class Commande extends Model{
        public function client() {
        return this-> belongsTo(Client::class);}}
```

Exemples ??

Relations

- Many to many:

- Chaque enregistrement dans la première table peut être associé à plusieurs enregistrements dans la seconde table, et vice versa.
- => table pivot relie les deux tables

Ex. un produit peut être inclus dans plusieurs commandes (belongsToMany())

```
class Produit extends Model{
        public function commande() {
            return this-> belongsToMany(Commande::class);}}
```

Ex. une commande peut contenir plusieurs produits (belongsToMany())

```
class Commande extends Model{
        public function produit() {
        return this-> belongsToMany(Produit::class);}}
```

Exemples ??

Table pivot

- Table **intermédiaire** utilisée pour représenter une relation plusieurs-à-plusieurs entre deux tables

- Elle possède **deux clés étrangères** qui font référence aux clés primaires des deux tables

Sécurité

- Dans laravel, la sécurité est assurée par divers mécanismes tels que:
  - Chiffrement
  - Hachage
  - Authentification
  - Autorisation

Chiffrement

- Laravel permet de crypter et décrypter des données en utilisant les chiffrements AES-256 et AES-128

- Le texte crypté est ensuite signé à l'aide d'un code d'authentification de message (MAC)

=> un attaquant ne peut pas le **modifier**

- Pour utiliser ce service, la configuration d'une clé dans le fichier config/app.php est **requise**

- Utiliser la commande php artisan key:generate pour générer une valeur à la variable d'environnement APP_KEY

- Le chiffrement des données se fait avec la méthode encryptString fournie avec la façade Crypt

  Ex. $textChiffré = Crypt::encryptString($textClair);

- Le déchiffrement des données se fait avec la méthode decryptString

  Ex. $textClair = Crypt::decryptString($textChiffré);

Hachage

- La façade Hash fournit un hachage Bcrypt et Argon2 sécurisé pour stocker les mots de passe des utilisateurs

- La configuration d'un système de hachage par défaut se fait dans le fichier de configuration config/hashing.php

- Ces systèmes sont utilisés pour protéger les mots de passes

- Le hachage se fait avec la méthode make de la façade Hash

  Ex. $mdp = Hash::make('motdepasse');

- Le nombre de rounds peut être ajusté si le système Bcrypt est utilisé
- Le nombre de round élevé augmente la sécurité des mots de passe, mais augmente aussi le temps de calcul
- Les options mémoire, temps et threads peuvent être ajustés si le système Argon2 est utilisé
- La méthode check da la façade Hash permet de vérifier si un mot de passe correspond à un hash donné

```
Ex. if (Hash::check('$mdp', $mdpHaché)){
        echo "Mot de passe valide."; }
    else {
        echo "Mot de passe non valide.";
        }
```

Authentification

- Laravel propose **plusieurs** méthodes d'authentification des utilisateurs telle que:

  - Middleware
  - Helper auth
  - Facade Auth

❑ Middleware

- Lier un middleware d'authentification à une route

Exemples

- auth.basic (authentification de base HTTP)

il fournit un moyen **rapide** d'authentifier les utilisateurs sans avoir à créer une page de connexion dédiée

- auth: restreindre l'accès aux utilisateurs authentifiés
- guest: restreindre l'accès aux utilisateurs **non** authentifiés
- etc

Ex. Route::get('admin', function(){…}

->middleware('auth.basic')

=>Le middleware invite l'utilisateur à introduire un **nom** et un **mot de passe** corrects pour accéder à cette URL

Ex. Route::get('admin', function(){…}

->middleware('auth')

Ex. Route::get('login', function(){…}

->middlware('guest');

- En cas **d'échec** d'authentification, l'utilisateur est de nouveau **redirigé** vers la page d'authentification

Ex. protected function redirectTo(Request $request): string{

```
    return route('login');
}
```

=> Lorsque le middleware d'authentification détecte un utilisateur non authentifié, il le redirige vers la route nommée le "login"

❑ helper global auth()

- auth()->check retourne vrai si l'utilisateur courant est connecté
- auth()->guest retoune vrai si l'utilisateur n'est pas connecté
- auth()->user() et auth()->id() retournent null si aucun utilisateur n'est connecté
- auth()->logout() déconnecte l'user actuel

❑ Façade Auth

- Auth::check() **vérifie** si l'utilisateur courant est **authentifié**
- Auth::user(): retourne l'utilisateur actuellement authentifié
- Auth::id(): retourne l'ID de l'utilisateur actuellement authentifié
- Auth::logout(): déconnexion de l'utilisateur courant

❑ Authentification manuelle

- auth::attempt(): traite les tentatives d'authentification
- Si les informations sont valides, une **session** est générée et l'utilisateur sera connecté, sinon, la méthode retourne **false**

Ex. $cred = $request->only('login', 'password');

```
if (Auth::attempt($cred)) {
    $request->session()->regenerate();
}
```

- Pour sauvegarder les informations d'authentification d'un utilisateur, la variable booléenne $remember est utilisé
- La table des utilisateurs doit contenir la colonne remember_token

Ex. $cred = $request->only('login', 'password');

```
if (Auth::attempt($cred, $remember)) {
    $request->session()->regenerate();}
```

=> Si $remember=**true** les informations de connexion de l'utilisateur sont sauvegardées jusqu'à ce qu'il se déconnecte

D'autres méthodes

- Authentification avec l'ID avec la méthode loginUsingId()

(Clé primaire de la table utilisateur)

Ex. $user = User::find(10);

```
if ($user) {
    Auth::loginUsingId(10);
}
```

Ex. $user = User::find(10);

```
if ($user) {
    Auth::loginUsingId(10, $remember = true);
}
```

- Connexion pour une seule requête avec la méthode once

```
Ex. if (Auth::once($cred)) {
        // L'authentification a réussi
    } else {
        // L'authentification a échoué pour cette requête
    }
```

Protection contre CSRF

- CSRF (cross-site request forgery)
- Le middleware VerifyCsrfToken est responsable de la vérification du jeton CSRF pour protéger l'application

- VerifyCsrfToken est activé sur **toutes** les routes web

- Une route peut être **exclue** de la vérification CSRF en ajoutant son URI à la propriété $except du middleware

- Un jeton CSRF est généré automatiquement dans chaque formulaire avec la directive blade @csrf

- Ce jeton est sauvegardé dans la **session** courante de l'utilisateur et accessible avec la méthode csrf_token()

- L'authentification se fait par le middleware qui vérifie si le jeton **soumis** correspond au jeton **généré** lors de la création du formulaire