CONTROLE N° 3

Au titre de l'année : 2022/2023

INSTITUT SPÉCIALISÉ DE TECHNOLOGIE APPLIQUÉE HAY AL ADARISSA FÈS	
Année de formation : 2A	**Groupe :** DEVOWFS206
Niveau de formation : TS	**Durée :** 02h00
Filière : Développement Digital – Option web full stack	**Barème :** / 20 pts
Intitulé du module : M205 Développement back end	

Soit le schéma relationnel suivant :

users (**id**, nomPrenom, email, date_embauche, role, password)
responsables (**id**, titre, date_mise_en_service, user_id)
chantiers (**id**, propriétaire, type, date_demarrage, responsable_id)
chantier_user (**chantier_id, user_id**, date_mission, description, etat)

- Les clés primaires sont en gras et les secondaires sont soulignées.
- La table « users » représente le compte d'un employé ; celui-ci peut être un responsable ;
- Quand on désire affecter une mission à un employé pour un chantier donné, on les enregistre dans la table « chantier_user » ;
- Le champ « role » peut être : employe, responsable ou admin
- Le champ « etat » prendra les valeurs : en cours ou terminée
- Le champ « date_mission » représente la date où un employé a commencé le travail sur le chantier

1. Donner la commande de création du modèle « Chantier » **(1pt)**

   php artisan make:model Chantier

2. On suppose que tous les modèles ont été créés,
   a. Dans le modèle « Responsable », définir sa relation avec « User » ; **(1pts)**

```
class Responsable extends Model
{
    use HasFactory;

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

   b. Dans le modèle « Chantier », donner la définition de ses relations avec : « Responsable » et avec « User » ; **(2pts)**

```
class Chantier extends Model
{
    use HasFactory;
    protected $guarded = [];

   public function responsable(){
```

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

    public function users(){
        return $this->belongsToMany(User::class)
        ->withPivot( "date_mission", "description", "etat");
    }
}
```

3. On suppose que toutes les relations ont été définies, exprimer les requêtes suivantes :
   a. Liste des chantiers qui ont démarré cette année (l'année en cours) **(2pts)**

```
Chantier::whereYear("date_demarrage", date("Y"))
->get();
```

   b. Le nombre de chantiers par type. **(2pts)**

```
Chantier::selectRaw("type, count(*) as nombre_chantiers")
->groupBy("type")
->get();
```

   c. Les employés qui ont été responsables sur plus de 2 chantiers de type « construction » **(2pts)**

```
Responsable::whereHas("chantiers", function(Builder $query){
    $query->where("type","construction");
},">=", 1)->get();
```

   d. Pour un chantier donné ($chantier), lister les employés qui ont terminé leur mission. **(2pts)**

```
$chantier=Chantier::find(1);
$chantier->users()->wherePivot("etat", "terminée")->get();
```

4. Soient une méthode ***filtrer(Request $request)*** dans un contrôleur « ***ChantierController*** » et une vue « ***index.blade.php*** » placée sous « ***views/chantiers***»,
   La méthode « filtrer » permet de retourner la liste des chantiers selon le type passé dans la requête
   Sachant qu'on souhaite afficher 5 chantiers par page, donner le code de cette méthode. **(2pts)**

```
    $type=$request->type;
    $chantiers= Chantier::where("type", "like", "%".$type."%")
    ->Paginate(2);
    return view('chantier.index', compact('chantiers'));
```

Méthode de pagination sur le résultat de recherche :

```
public function filtrer(Request $request){
    $type=$request->type;
    $chantiers= Chantier::where("type", "like", "%".$type."%")
    ->Paginate(2)
    ->appends(["type"=>$type]);
    return view('chantier.index', compact('chantiers', 'type'));
}
```

Sur index.blade.php

```
@if(isset($type))
```

```
            {{$chantiers->appends(["type"=>$type])->links()}}
        @else
            {{$chantiers->links()}}
            @endif
```

5. Soient les deux routes suivantes :

```
Route::get('mission/create', [MissionController::class, 'create'])->name('mission.create')
Route::post('mission', [MissionController ::class, 'store'])->name('mission.store')
```

- La première route affiche une vue qui contient un formulaire où l'utilisateur peut fournir les informations pour affecter une mission à un employé.
- La deuxième route traite les données du formulaire et enregistre la nouvelle ligne dans la table « chantier_user »
- On désire ajouter la règle suivante : La date de mission de l'employé ne doit pas être postérieure à la date de démarrage du chantier.
- On propose de valider cette règle avec un middleware « *DateMissionValidation* »

a. Donner la commande permettant de créer le middleware « ***DateMissionValidation*** » ; **(1pt)**

Php artisan make:middleware ***DateMissionValidation***

b. Sachant que si la règle n'est pas respectée, on redirige l'utilisateur vers la route (mission.create) avec un message d'erreur et que ***$request*** contient les informations suivantes : chantier_id, user_id, date_mission, description et etat, donner le code de la fonction ***handle*** : **(2pts)**

```
public function handle($request, Closure $next)
{
        $Chantier=Chantier::find($request->chantier_id);
        $date_mission= $request->date_mission;
        if($date_mission>=$Chantier->date_demarrage)
        {
            return $next($request);
        }

        return redirect()->route("mission.create")->with("error", "La date de la
mission ne peut pas être postérieure à la date de démarrage du chantier!");
    }
```

6. Donner la méthode permettant de vérifier si un utilisateur est authentifié ou non. **(1pt)**

Auth ::check()

auth()->check()

7. Soit la règle suivante : un responsable ne peut afficher que les chantiers qui ont déjà démarré et dont il est responsable ;

On a utilisé un Gate pour valider cette règle ; voici son utilisation :

```
public function show(Chantier $chantier)
    {
        Gate::authorize('afficher_chantier', $chantier);

        return view("chantier.show", compact("chantier"));
    }
```

Définir le « Gate » permettant de faire ces vérifications. **(2pts)**

```
Gate::define("afficher_chantier", function(User $user, Chantier $chantier){
```

```
            if( $user->role!="responsable")
                return Response::deny("Vous n'êtes pas responsable", "403");

            if(  $chantier->responsable->user_id!=$user->id)
                return Response::deny("Vous n'êtes pas responsable sur le chantier
numéro ".$chantier->id, "403");

            if($chantier->date_demarrage > Carbon::now())
                return Response::deny("Ce chantier n'a pas encore démarré!", "403");

            return Response::allow();

        });
```