RAPPORT TECHNIQUE FINAL

DU:

Responsables de back-end

rédigé par:
ABBAD ABDELOUAHED
EL_BATTEOUI OUSSAMA

Encadré par:
Mr.GHAILANI MOHAMED

Sommaire

Introduction

Cette documentation technique présente une vue d'ensemble des différents contrôleurs utilisés dans notre application. Chaque contrôleur a des responsabilités spécifiques pour assurer le bon fonctionnement du système. Le AdminActiviteeController gère les activités, le AdminOffreController traite les offres, et le AdminUserController s'occupe des utilisateurs administratifs. D'autres contrôleurs, tels que le NotificationController et le PDFController, assurent respectivement la gestion des notifications et la génération de fichiers PDF. Nous avons également des contrôleurs spécifiques pour la gestion des horaires (AdminHoraireController), des parents (FatherController), des animateurs (AnimatorController), et des affichages (ShowController). Ce document détaille les différentes méthodes implémentées dans ces contrôleurs, en expliquant leurs rôles et leurs interactions au sein de l'application.

Commandes utilisées

php artisan serve: Lancement du serveur de développement

php artisan migrate: migration des tables dans la base de donnés

php artisan migrate:rollback : Retour en arrière des dernières migrations
php artisan migrate:refresh :réinitialiser la base de données en retournant en arrière toutes les migrations et en les réexécutant.

php artisan db:seed: Exécution des seeders

php artisan migrate:refresh --seed: réinitialisation + Exécution des seeders
php artisan migrate:reset:Réinitialisation de toutes les migrations

php artisan make:migration create_users_table : créer une nouvelle migration

php artisan make:model ModelName: créer un model

php artisan make:model ModelName -m: Création d'un modèle avec une migration associée

php artisan make:controller ControllerName: Création d'un nouveau contrôleur

php artisan make:controller ControllerName --resource: Création d'un contrôleur de ressources

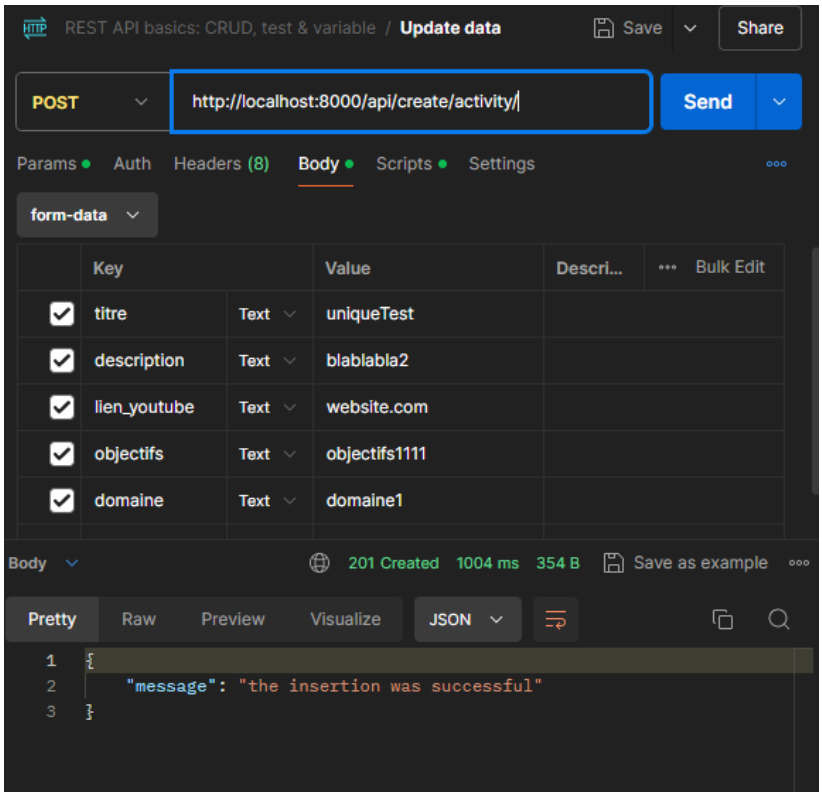
php artisan route:list: afficher une liste de toutes les routes définies

AdminActiviteeController

Le contrôleur `AdminActiviteeController` est responsable de trois fonctionnalités principales : la création, la mise à jour et la suppression des activités. Ces fonctionnalités sont spécifiquement réservées à l'administrateur, et pour le moment, ces activités n'ont aucune relation avec les offres.

createActivity

```php
1 usage  ABDELOUAHED ABBAD
public function createActivity(Request $request){
    $formFields = $request->validate([
        'titre' => ['required',Rule::unique( table: 'activites', column: 'titre')],
        'description'=>'required',
        'lien_youtube' => 'required',
        'objectifs' => 'required',
        'domaine' =>'required'
    ]);
    if($request->hasFile( key: 'IMAGE_PUB')){
        $formFields['IMAGE_PUB'] = $request->file( key: 'IMAGE_PUB')->store( path: 'IMAGE_PUBs', options: 'public');
    }
    Activite::create($formFields);
    return response()->json(['message'=>'the insertion was successful'], status: 201);
}
```

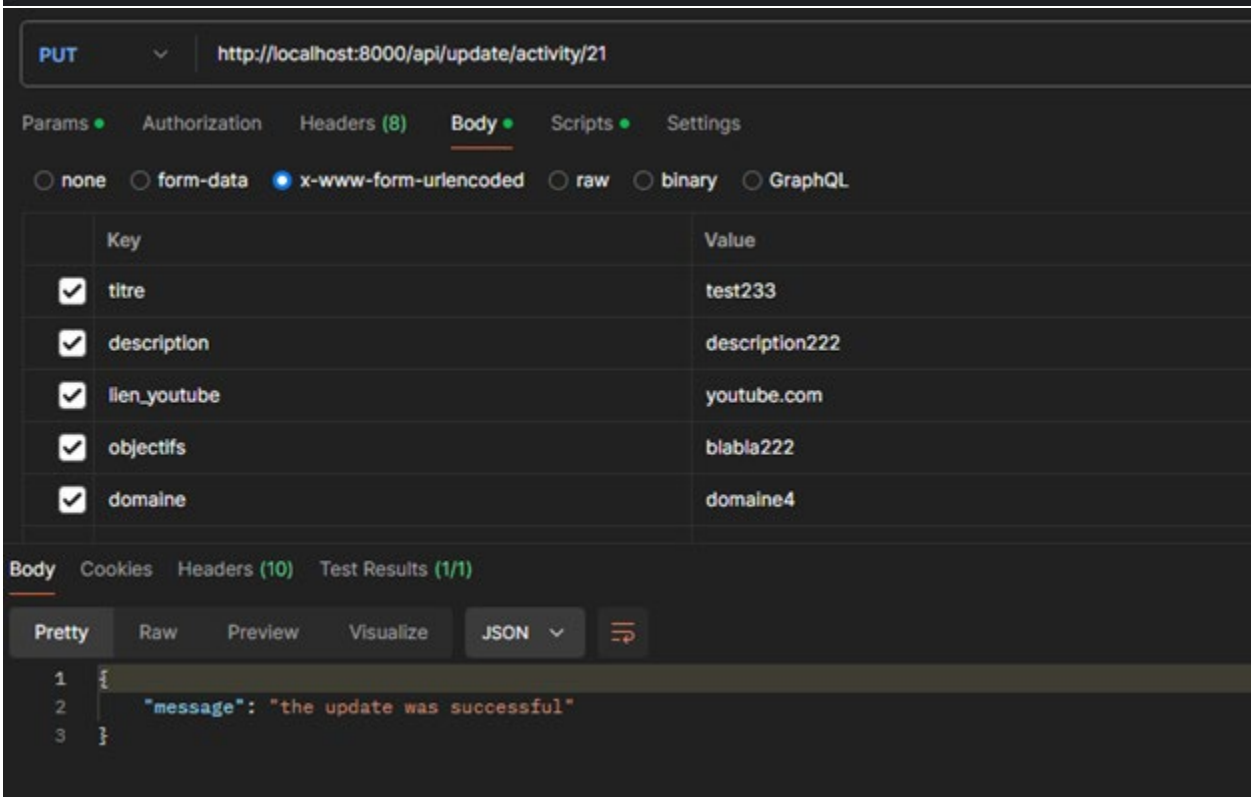


La fonction `createActivite` est utilisée pour créer une nouvelle activité dans l'application. Elle valide les données de la requête, traite l'upload de l'image, et sauvegarde l'activité dans la base de données.

```
ROUTE:
Route::post('/create/activity/',[AdminActiviteeController::class,'createActivity']);
```

updateActivity

```php
1 usage  ABDELOUAHED ABBAD *
public function updateActivity(Request $request,Activite $activity){
    $formFields = $request->validate([
        'titre' => ['required',Rule::unique( table: 'activites', column: 'titre')],
        'description'=>'required',
        'lien_youtube' => 'required',
        'objectifs' => 'required',
        'domaine' =>'required'
    ]);
    if($request->hasFile( key: 'IMAGE_PUB')){
        $formFields['IMAGE_PUB'] = $request->file( key: 'IMAGE_PUB')->store( path: 'IMAGE_PUBs', options: 'public');
    }
    $activity->update($formFields);
    return response()->json(['message'=>'the update was successful'], status: 200);
}
```



Le code met à jour l'activité existante dans la base de données avec les nouvelles informations fournies, ce qui permet de maintenir les informations à jour et de les modifier facilement en fonction des besoins.

ROUTE :

```php
Route::put('/update/activity/{activity}',[AdminActiviteController::class,'updateActivity']);
```

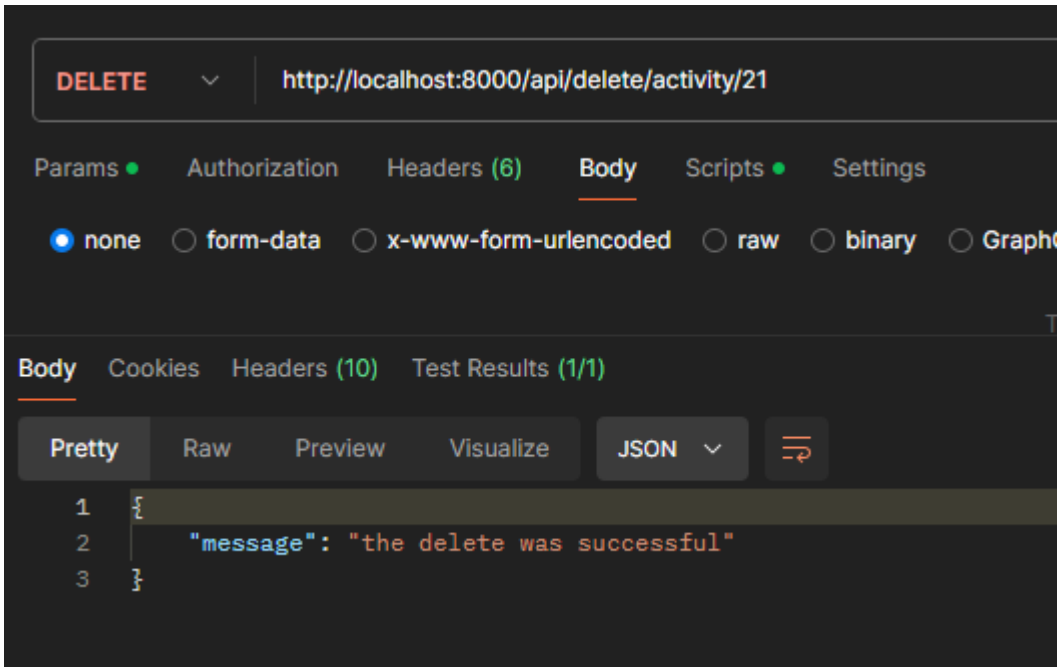
Validation et Traitement de l'Image

Dans les fonctions **createActivity** et **updateActivity**, la validation des champs (titre, description, lien_youtube, objectifs, et domaine) assure l'intégrité des données. Le code utilise la méthode validate pour vérifier que les champs requis sont présents et conformes aux contraintes. Si une image (IMAGE_PUB) est fournie, elle est stockée dans le répertoire public/image et son chemin est enregistré. Ce traitement garantit que les données et les fichiers sont correctement gérés et sécurisés.

---

deleteActivity

```
1 usage   ABDELOUAHED ABBAD
public function destroyActivity(Activite $activity)
{
    $activity->delete();
    return response()->json(['message'=>'the delete was successful'], status: 200);
}
```



La fonction **destroyActivity** est utilisée pour supprimer une activité existante de l'application. Elle supprime l'entrée de l'activité dans la base de données et retourne un message de confirmation de la suppression.

ROUTE :

```
Route::delete('/delete/activity/{activity}',[AdminActiv
iteeController::class,'destroyActivity']);
```

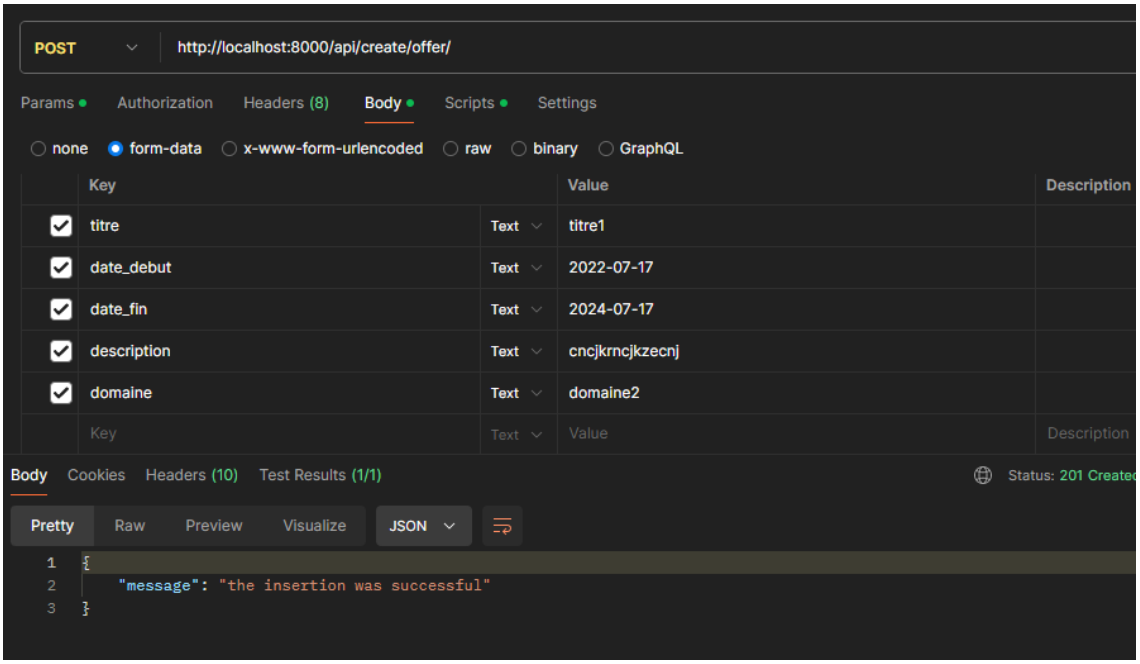
AdminOffreController

Le contrôleur `AdminOffreController` sera responsable de trois fonctionnalités principales : la création, la mise à jour et la suppression des offres. Ces fonctionnalités sont spécifiquement réservées à l'administrateur pour gérer les différentes offres proposées sur la plateforme.

createOffer

```php
1 usage  ABDELOUAHED ABBAD
public function createOffer(Request $request){
    $formFields = $request->validate([
        'titre' => 'required',
        'date_debut' => 'required|date',
        'date_fin' => 'required|date|after:date_debut',
        'description' => 'required',
        'domaine' => 'required'
    ]);
    $user_id = auth()->id();
    $admin = Administrateur::where('user_id',$user_id)->first();
    $formFields['admin_id'] = $admin->id;

    if($request->has( key: 'remise')) $formFields['remise'] = $request->remise;
    else $formFields['remise'] = 0;
    Offre::create($formFields);
    return response()->json(['message'=>'the insertion was successful'], status: 201);
}
```



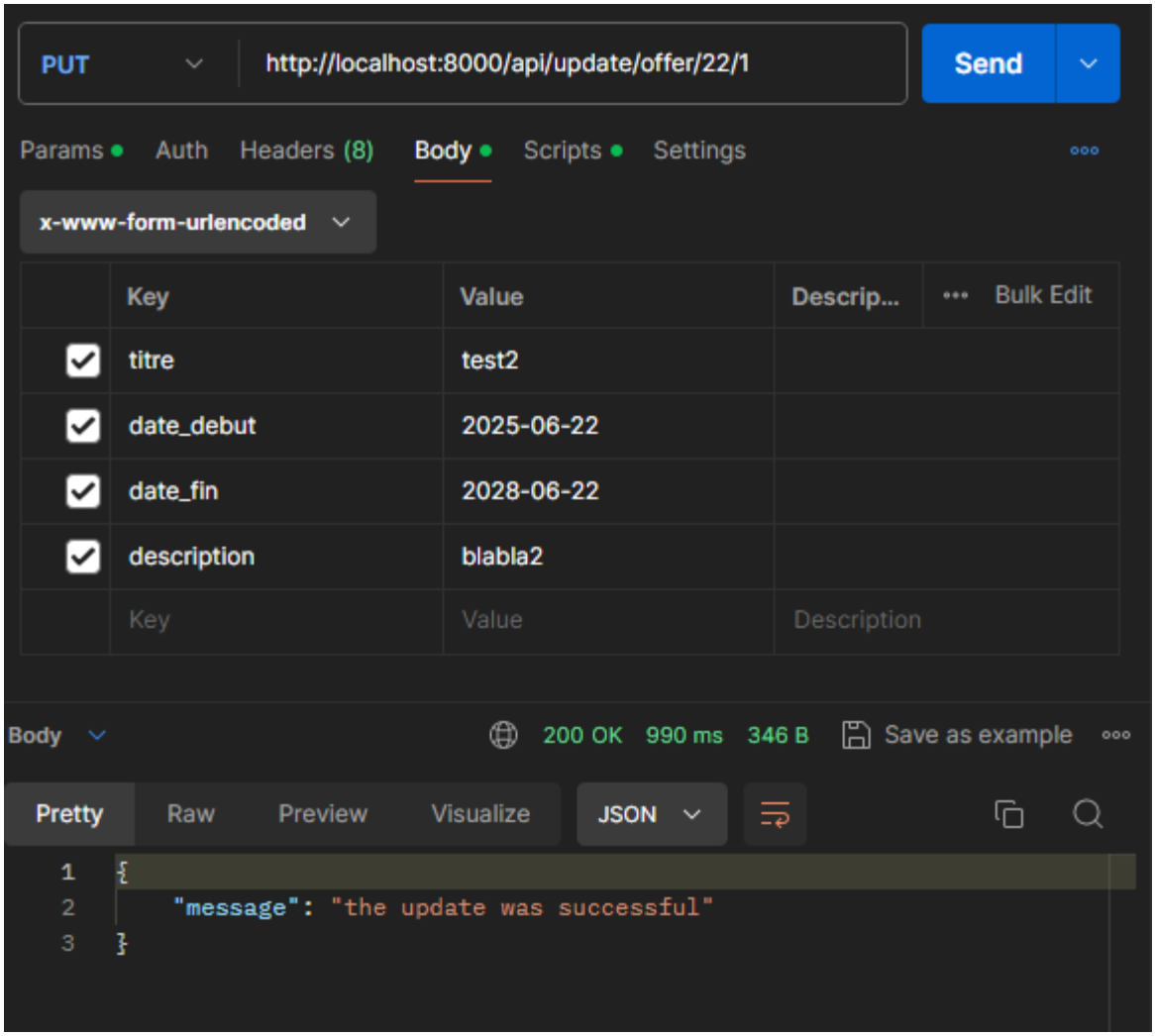
La fonction `createOffer` est utilisée pour créer une nouvelle offre. Elle commence par valider les données de la requête, incluant des champs comme `titre`, `date_debut`, `date_fin`, `description`, et `domaine`, pour s'assurer qu'ils sont présents et conformes aux contraintes spécifiées. Ensuite, elle associe l'offre à un administrateur en récupérant l'ID de l'utilisateur authentifié via `auth()->id()` pour les tests, cet ID est remplacé par un ID d'administrateur de la base de données. Si une remise est fournie dans la requête, elle est également traitée et ajoutée aux champs de formulaire, en s'assurant qu'elle est positive. Enfin, l'offre est créée dans la base de données avec les champs validés, et une réponse JSON confirmant le succès de l'insertion est retournée avec un code de statut HTTP 201.

ROUTE :

```
Route::post('/create/offer/',[AdminOffreController::class,'createOffer']);
```

updateOffer

```php
1 usage   ABDELOUAHED ABBAD
public function updateOffer(Request $request,Offre $offer){
    $formFields = $request->validate([
        'titre' => 'required',
        'date_debut' => 'required|date',
        'date_fin' => 'required|date|after:date_debut',
        'description' => 'required',
        'domaine' => 'required'
    ]);
    if($request->has( key: 'remise')) $formFields['remise'] = $request->remise;
    $offer->update($formFields);
    return response()->json(['message'=>'the update was successful'], status: 200);
}
```



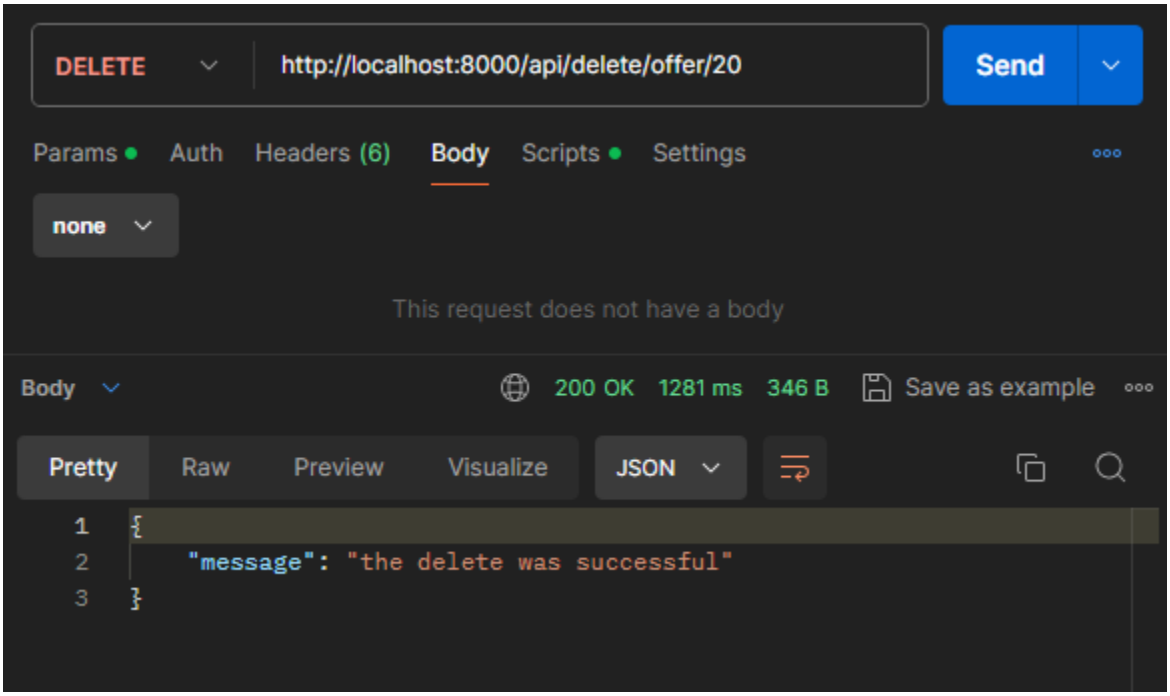
La fonction `updateOffer` fonctionne de manière similaire à la fonction `createOffer`, mais avec quelques différences notables. Elle valide également les champs `titre`, `date_debut`, `date_fin`, `description`, et `domaine`, et associe l'offre à un administrateur en utilisant l'ID de l'utilisateur authentifié. Cependant, au lieu de créer une nouvelle offre, cette fonction met à jour une offre existante en utilisant l'ID de l'offre passé dans l'URL de la requête PUT. De plus, si une remise est fournie dans la requête, elle est traitée et ajoutée aux champs de formulaire. L'offre est ensuite mise à jour dans la base de données, et une réponse JSON confirmant le succès de la mise à jour est retournée avec un code de statut HTTP 200.

ROUTE :

```
Route::put('/update/offer/{offer}/',[AdminOffreControll
er::class,'updateOffer']);
```

destroyOffer

```
ABDELOUAHED ABBAD
public function destroyOffer(Offre $offer)
{
    $offer->delete();
    return response()->json(['message'=>'the delete was successful'], status: 200);
}
```



La fonction `destroyOffer` est utilisée pour supprimer une offre existante de l'application. Elle supprime l'entrée de l'offre dans la base de données et retourne un message de confirmation de la suppression.

Route :

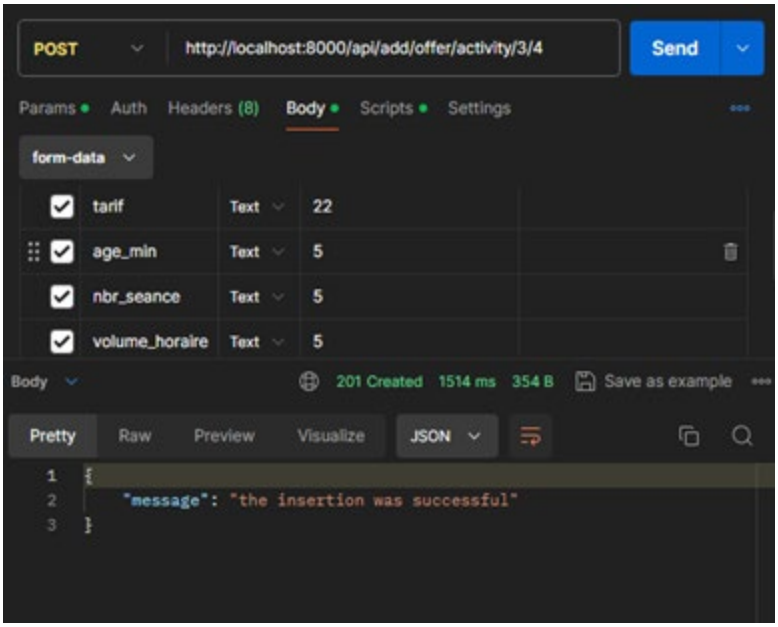
```
Route::delete('/delete/offer/{offer}',[AdminOffreController::class,'destroyOffer']);
```

AdminActiviteeOffreController

Le contrôleur adminActiviteeOffreController est chargé de la gestion des activités associées aux offres. Il permet de créer, mettre à jour et supprimer des activités dans les offres.

addActivityToOffer

```php
ABDELOUAHED ABBAD *
public function addActivityToOffer(Request $request,Offre $offer,Activite $activity){
    $formFields = $request->validate([
        'tarif' => 'required|numeric',
        'age_min' => 'required|integer|min:3',
        'nbr_seance' => 'required|integer|min:1',
        'volume_horaire' => 'required|integer|min:1',
        'option_paiement' => 'required',
        'age_max' => 'required|integer|min:3'
    ]);
    $formFields['offre_id'] = $offer->id;
    $formFields['activite_id'] = $activity->id;
    ActiviteOffre::create($formFields);
    return response()->json(['message'=>'the insertion was successful'], status: 201);
}
```



addActivityToOffer() permet d'ajouter une activité à une offre en insérant les données dans la table activityOffer. Elle valide les champs requis comme tarif, age_min, nbr_seance, volume_horaire, option_paiement, et age_max. Après validation, elle attribue les IDs de l'offre et de l'activité aux champs offre_id et activite_id. Ensuite, elle insère les données dans la base de données en utilisant ActivityOffer::create($formFields) et retourne une réponse JSON confirmant la réussite de l'insertion.

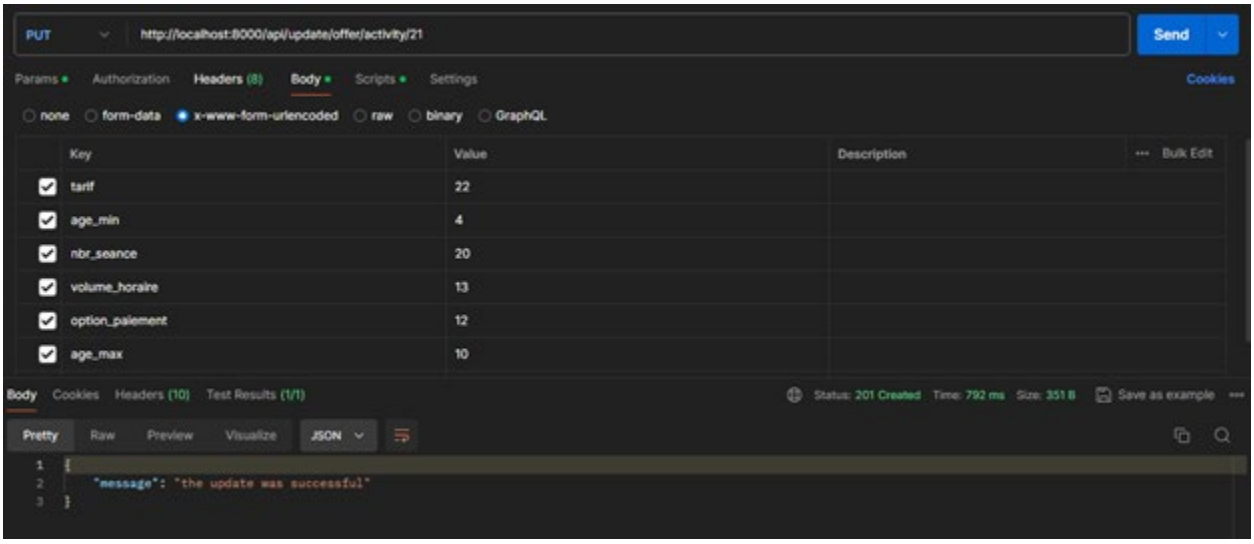
ROUTE :

```
Route::post('/add/offer/activity/{offer}/{activity}',[AdminActiviteeOffreController::class,'addActivityToOffer']);
```

updateActivityInOffer

```
1 usage   ABDELOUAHED ABBAD *
public function updateActivityInOffer(Request $request,ActiviteOffre $activityOffer){
    $formFields = $request->validate([
        'tarif' => 'required|numeric',
        'age_min' => 'required|integer|min:3',
        'nbr_seance' => 'required|integer|min:1',
        'volume_horaire' => 'required|integer|min:1',
        'option_paiement' => 'required',
        'age_max' => 'required|integer|min:3'
    ]);

    $activityOffer->update($formFields);
    return response()->json(['message'=>'the update was successful'], status: 201);
}
```



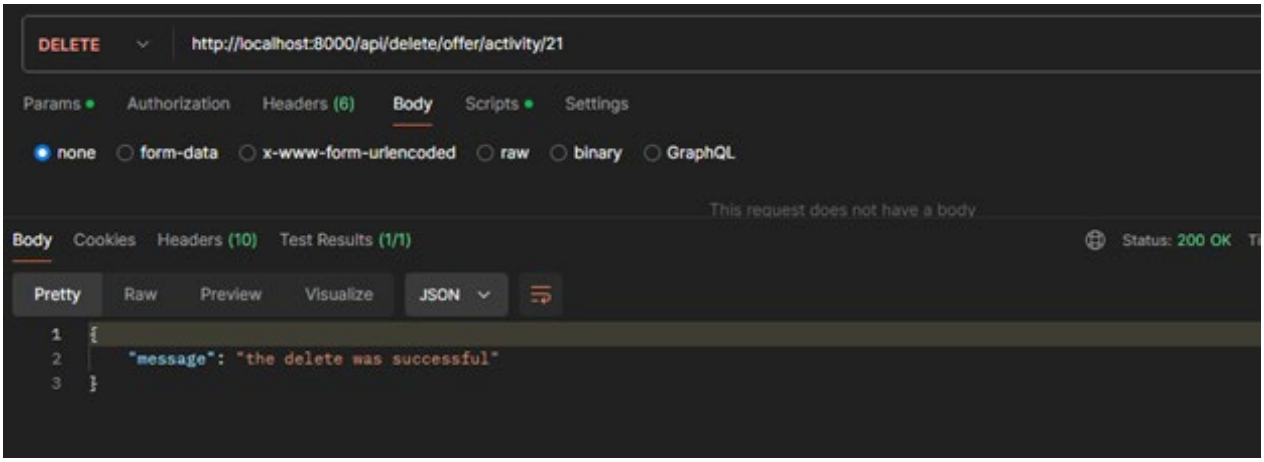
La fonction updateActivityInOffer permet de mettre à jour les détails d'une activité dans une offre. Elle vérifie et valide les champs requis comme tarif, age_min, nbr_seance, volume_horaire, option_paiement et age_max, puis effectue la mise à jour dans la table activityOffer avec ActivityOffer::update($formFields). Enfin, elle retourne une réponse JSON indiquant que la mise à jour a été effectuée avec succès.

ROUTE :

```
Route::put('/update/offer/activity/{activityOffer}',[AdminActiviteeOffreController::class,'updateActivityInOffer']);
```

destroyActivity

```
1 usage   ABDELOUAHED ABBAD
public function destroyActivity(ActiviteOffre $activityOffer)
{
    $activityOffer->delete();
    return response()->json(['message'=>'the delete was successful'], status: 200);
}
```



La fonction destroyActivity supprime une activité d'une offre dans la table activityOffer en utilisant la méthode delete() et renvoie une réponse JSON confirmant la suppression avec succès.

ROUTE :

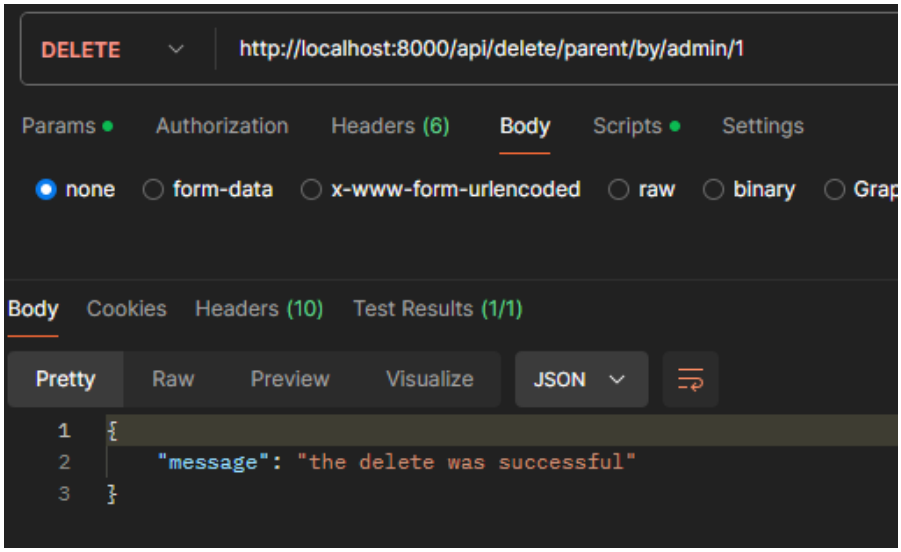
```
Route::delete('/delete/offer/activity/{activityOffer}',
[AdminActiviteeOffreController::class,'destroyActivity'
]);
```

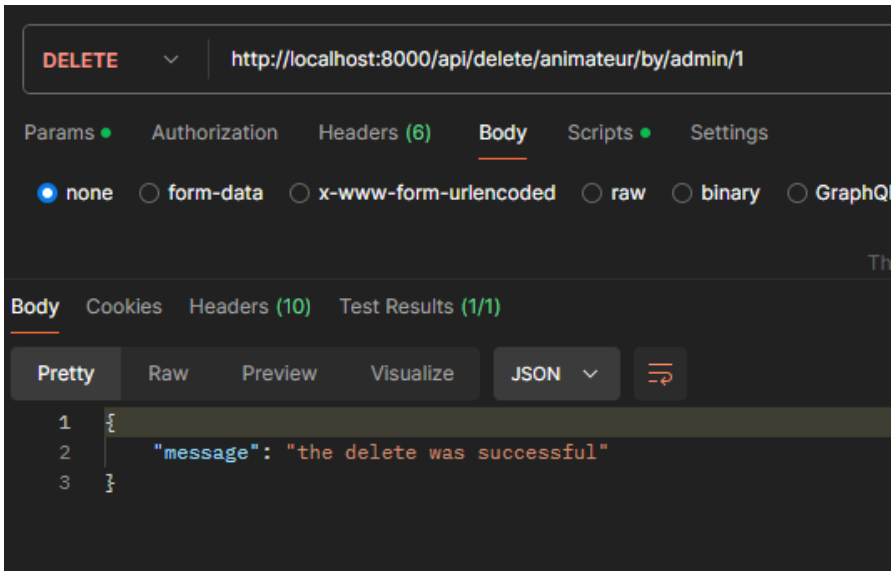
AdminUserController

Le `AdminUserController` est le contrôleur chargé de gérer les opérations de suppression des utilisateurs spécifiques, à savoir les animateurs et les parents, effectuées par l'administrateur. Ce contrôleur assure que les utilisateurs ciblés, ainsi que toutes leurs relations associées, sont correctement et complètement supprimés de l'application.

deleteParent & deleteAnimateur

```php
1 usage  ABDELOUAHED ABBAD
public function deleteParent(Father $parent){
    $user = User::find($parent->user_id);
    $user->delete();
    return response()->json(['message'=>'the delete was successful'], status: 200);
}
1 usage  ABDELOUAHED ABBAD
public function deleteAnimateur(Animateur $animateur){
    $user = User::find($animateur->user_id);
    $user->delete();
    return response()->json(['message'=>'the delete was successful'], status: 200);
}
```





Les fonctions `deleteParent` et `deleteAnimateur` dans le contrôleur `AdminController` sont utilisées pour supprimer respectivement un parent et un animateur de l'application. Elles commencent par trouver l'utilisateur associé au parent ou à l'animateur en utilisant `user_id`. Ensuite, elles suppriment l'utilisateur de la base de données. La suppression de l'utilisateur entraîne automatiquement la suppression du parent et de tous les enfants associés à lui, ainsi que la suppression de l'animateur. Enfin, elles retournent une réponse JSON confirmant le succès de la suppression avec un code de statut http 200.

ROUTE :

```
Route::delete('/delete/parent/by/admin/{parent}',[Admin
UserController::class,'deleteParent']);
Route::delete('/delete/animateur/by/admin/{animateur}',
[AdminUserController::class,'deleteAnimateur']);
```

AdminDemandeController

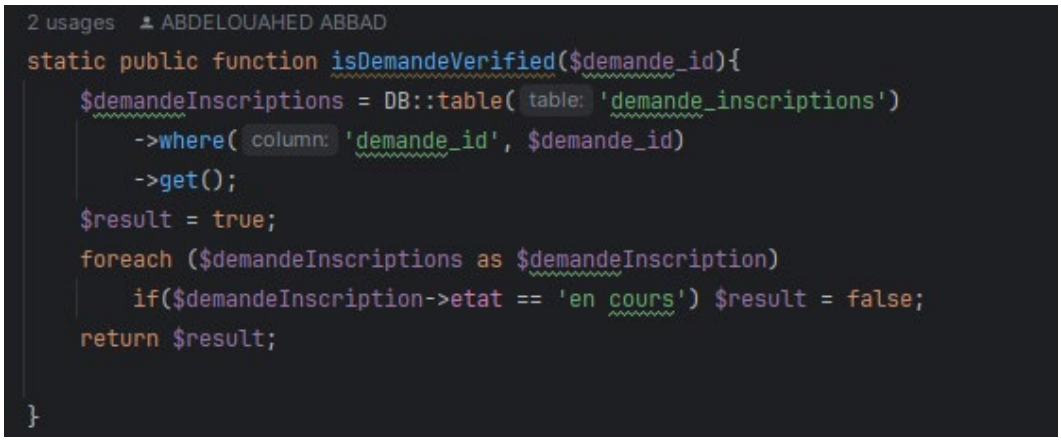
Le `AdminDemandeController` est responsable du traitement des demandes, incluant l'acceptation ou le refus des demandes de parents par un administrateur. Si toutes les activités de la demande ont été traitées, que ce soit par acceptation ou par refus, le contrôleur appelle le `DevisController` pour créer le devis pour les activités acceptées.

gererDemande & isDemandeVerified

```
1 usage   ABDELOUAHED ABBAD *
public function gererDemande(Request $request,$demande_id,$activite_offre_id,$enfant_id){
    $demande = DemandeInscription::where('enfant_id', $enfant_id)
        ->where('activite_offre_id', $activite_offre_id)
        ->where('demande_id', $demande_id)
        ->update([
            'etat' => $request->etat,
            'motif' => $request->motif,
            'updated_at' => now()
        ]);
    $demande = DemandeInscription::where('enfant_id', $enfant_id)
        ->where('activite_offre_id', $activite_offre_id)
        ->where('demande_id', $demande_id)->first();
    if($request->etat == 'accepte')$msg = "the request is well accepted";
    else {

        $horaire = explode( separator: ',',$demande->horaire);
        $horaire_id = Horaire::where('jour',$horaire[0])->where('heure_debut',$horaire[1])
            ->where('horaires.heure_fin',$horaire[2])->first()->id;
        $hda = Hda::find($horaire_id);
        $hda->update(['nbr_place_restant'=>$hda->nbr_place_restant+1]);
        $msg = "the request is well denied";
    }
```

```
$parent_id  = Enfant::find($enfant_id)->father_id;
$parent= Father::find($parent_id);
$user = User::find($parent->user_id);
if(AdminDemandeController::isDemandeVerified($demande_id)) {
    NotificationController::notifyDemandeHandled($demande_id);
    DevisController::createDevis($demande_id,$user);
}
return response()->json(['message'=>$msg], status: 200);
```



La fonction `gererDemande` gère la mise à jour des demandes d'inscription. Elle commence par valider et mettre à jour l'état (`etat`), le motif (`motif`) et la date de mise à jour (`updated_at`) de la demande spécifiée. Si la demande est acceptée, elle génère un message de confirmation. Si la demande est refusée, elle met à jour les horaires et le nombre de places restantes en conséquence. Pour cela, elle extrait les informations de l'horaire (`jour`, `heure_debut`, `heure_fin`) de la demande, trouve l'ID de l'horaire correspondant, et met à jour le nombre de places restantes dans la table `HDA`. Ensuite, elle vérifie si la demande est validée en appelant la fonction `isDemandeVerified`. Si la demande est vérifiée, elle envoie des notifications via le `NotificationController` et crée un devis via la fonction `createDevis` du contrôleur `AdminDemandeController`. La fonction retourne une réponse JSON confirmant le traitement de la demande avec un message approprié et un code de statut HTTP 200.

La fonction `isDemandeVerified` vérifie si une demande d'inscription est validée. Elle récupère toutes les demandes avec l'ID spécifié et vérifie leur état. Si une des demandes a un état `en cours`, la fonction retourne `false`, sinon elle retourne `true`.

ROUTE :

```
Route::put('/update/demande/inscription/{demande_id}/{activite_offre_id}/{enfant_id}',[AdminDemandeController::class,'gererDemande']);
```

Le test Postman sera affiché dans le contrôleur responsable de la génération des PDF.

DevisController

Le `DevisController` est responsable de la création d'un devis, qui sera appelé par le `AdminDemandeController`. Il est également responsable de la mise à jour des devis existants.

createDevis

```php
1 usage  ABDELOUAHED ABBAD *
public function createDevis($demande_id,$user){
    $activiteOffreIds = DB::table( table: 'demande_inscriptions')
        ->where( column: 'demande_id', $demande_id)
        ->where( column: 'etat', operator: 'accepte')
        ->pluck( column: 'activite_offre_id');
    $totale_ht = 0;
    foreach( $activiteOffreIds as $activiteOffreId){
        $activiteOffre = ActiviteOffre::find($activiteOffreId);
        $totale_ht+=$activiteOffre->tarif_remise;
    }
    $demande = Demande::find($demande_id);
    if($demande->pack_id) {
        $pack = Pack::find($demande->pack_id);
        $totale_ht = $totale_ht + ($totale_ht * ($pack->remise / 100));
    }
    $totale_ttc = $totale_ht + ($totale_ht*(11/100)); // (tva = 11%)
    $date = now();
    $pdf = PDFController::generatePDF($demande_id,$date,$totale_ht,$totale_ttc,$user);
```

```php
    $devis = Devis::create([
        'demande_id'=>$demande_id,
        'date'=>$date,
        'totale_ht'=>$totale_ht,
        'totale_ttc'=>$totale_ttc,
        'pdf'=>$pdf,
        'statut'=>'en cours',
        'etat'=>'non paye'
    ]);
        NotificationController::notifyDevisCreated($demande_id,$devis);
}
```

La methode `createDevis` génère un devis pour une demande d'inscription en calculant le total hors taxes (`totale_ht`) et le total toutes taxes comprises (`totale_ttc`). Elle commence par récupérer les IDs des offres d'activités acceptées associées à la demande, puis additionne les tarifs remisés de ces offres. Si la demande est associée à un pack, la remise du pack est appliquée au total HT. Ensuite, la fonction calcule le total TTC en ajoutant la TVA de 11% au total HT. Un

PDF du devis est généré via la méthode `generatePDF` du `PDFController`. Les informations du devis sont ensuite sauvegardées dans la base de données et une notification est envoyée via la méthode `notifyDevisCreated` du `NotificationController`, indiquant que le devis a été créé.

Il n'y a pas de route. La méthode sera appelée par `gererDemande` dans `AdminDemandeController`.

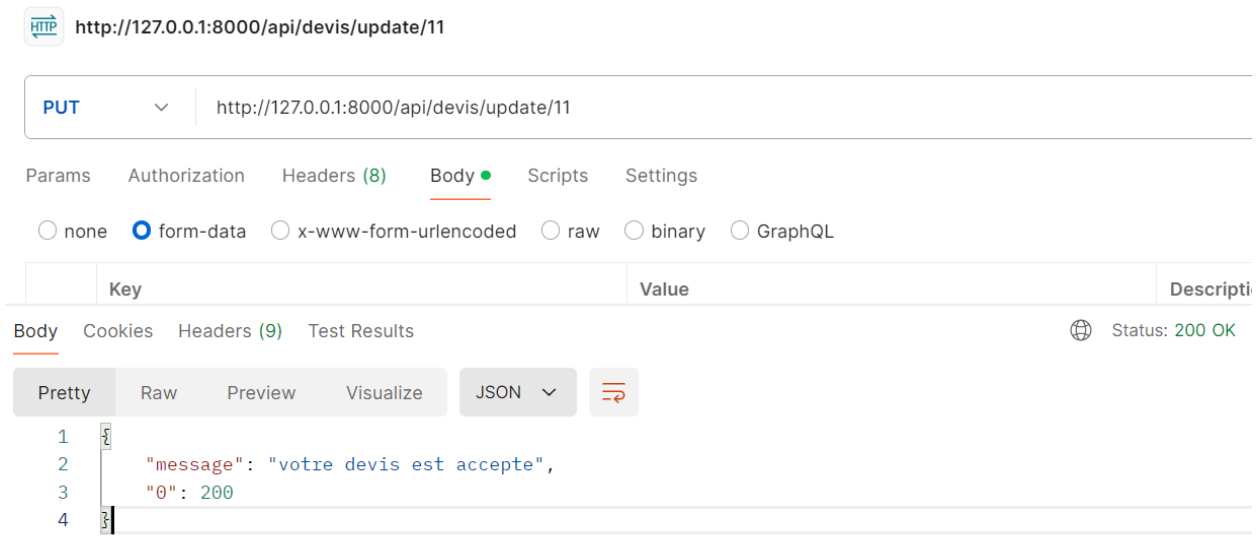
UpdateDevis

mettre à jour le statut du devis avec la valeur de statut provenant de la requête.

vérifier si la requête contient un champ motif si c'est le cas , le devis est refusée

retourne une réponse JSON contenant le message final ($msg) et un code de statut HTTP 200

```php
public function UpdateDevis(Request $request,Devis $devis){
    $msg = "votre devis est accepte";
    $devis->update([
        'statut'=>$request->statut
    ]);
    if($request->has('motif')){
        $msg = "votre devis est refuse";
        $devis->update([
            'motif'=>$request->motif
        ]);
    }
    return response()->json(['message'=>$msg,200]);
}
```



ROUTE :

```
Route::put('/devis/update/{id}',[
ParentDemandeController::class,'UpdateDevis']);
```

PDFController

Le `PDFController` est responsable de la génération des fichiers PDF.

generatePDF

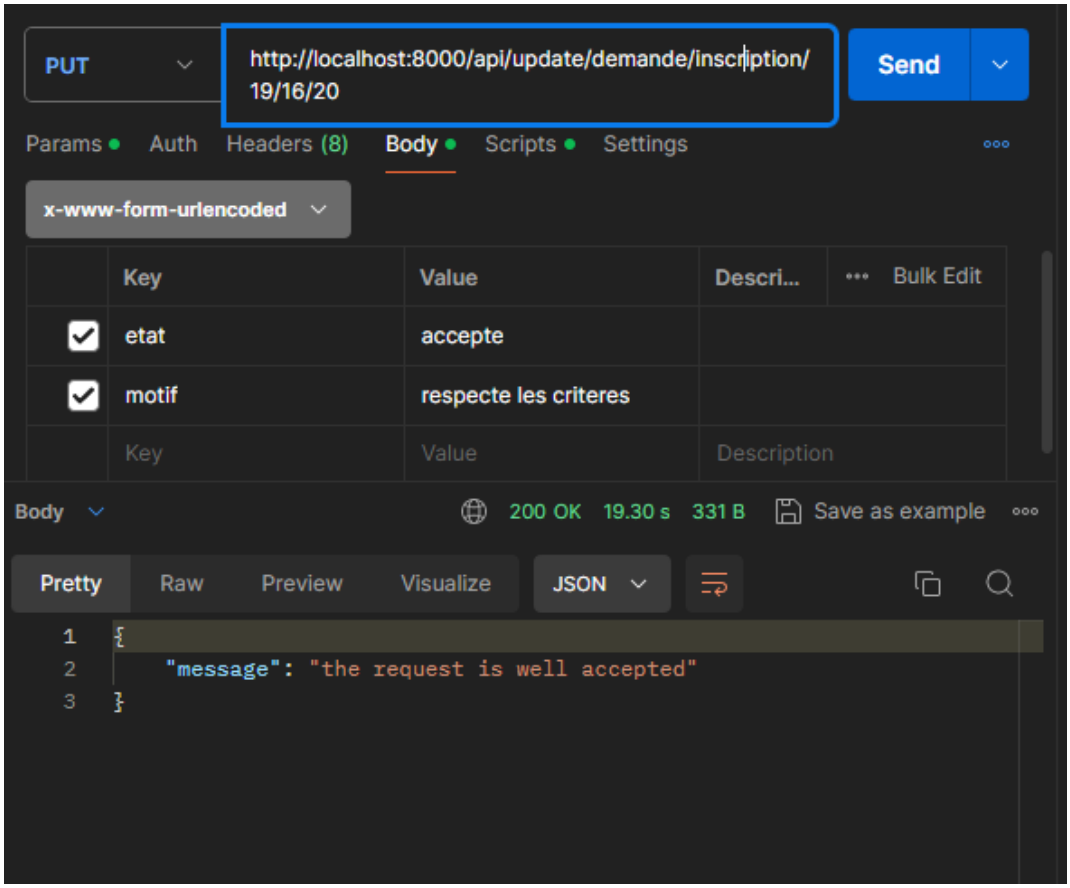
```
1 usage   ABDELOUAHED ABBAD *
static public function generatePDF($demande_id,$date,$totale_ht,$totale_ttc,$user){
    $dompdf = new Dompdf();
    $totale_ht = number_format($totale_ht, decimals: 2, decimal_separator: ',', thousands_separator: ' ');
    $totale_ttc = number_format($totale_ttc, decimals: 2, decimal_separator: ',', thousands_separator: ' ');
    $html = '
<style>
        <h2>Merci pour votre confiance !</h2>
    </div>
</div>';
    $dompdf->loadHtml($html);
    $dompdf->render();
    $pdfContent = $dompdf->output();
    $filePath = 'public/devis/' . $demande_id . '.pdf';
    Storage::put($filePath, $pdfContent);
    return $filePath;
}
```

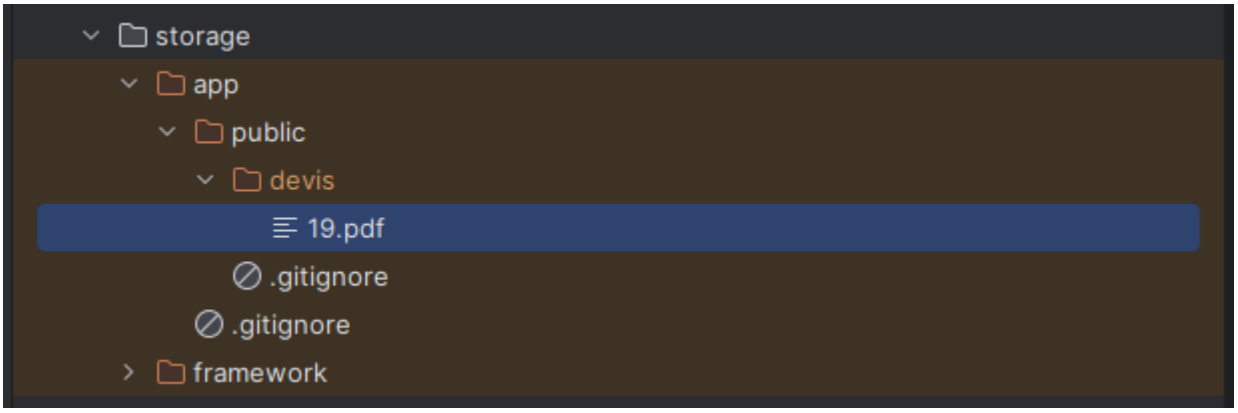
Pour utiliser Dompdf, j'ai dû exécuter la commande suivante pour l'importer :

composer require dompdf/dompdf

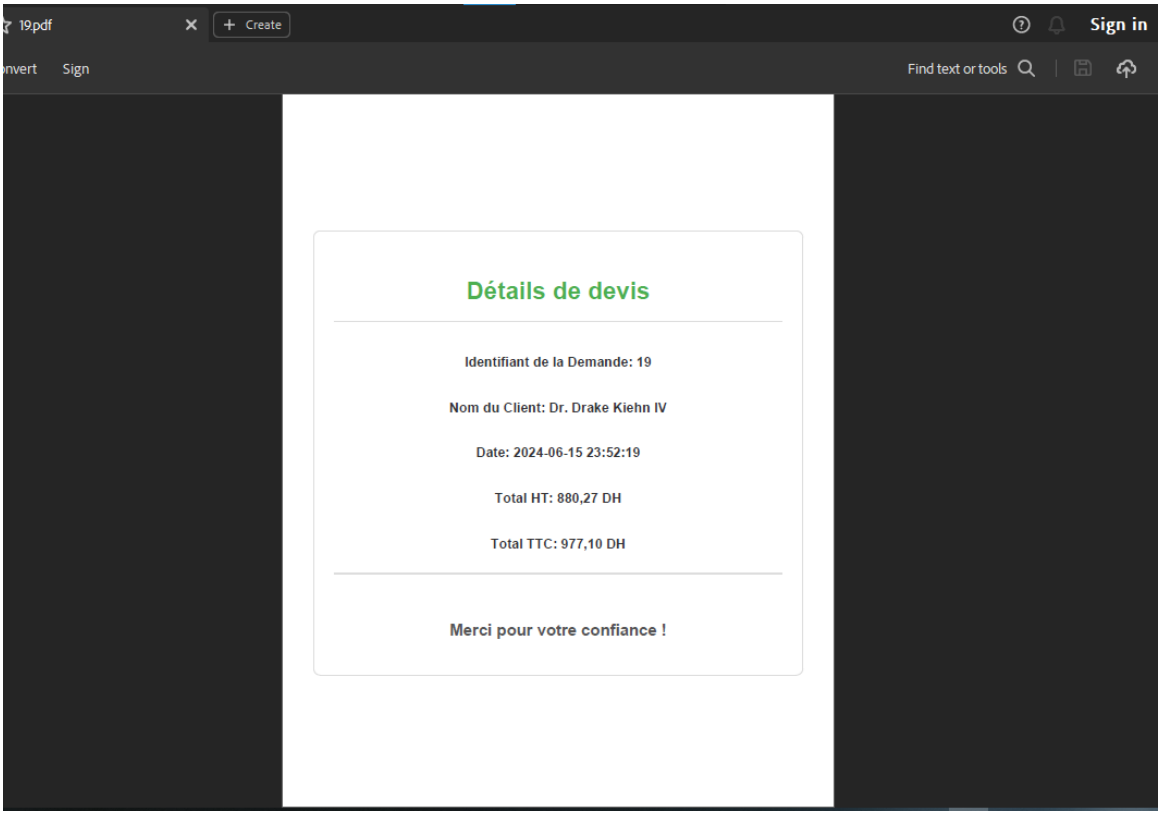
La methode `generatePDF` du contrôleur `PDFController` est utilisée pour générer un fichier PDF basé sur les détails d'une demande. Cette fonction prend en entrée l'ID de la demande (`demande_id`), la date (`date`), le total hors taxes (`totale_ht`), le total toutes taxes comprises (`totale_ttc`), et les informations de l'utilisateur (`user`). Elle commence par formater les montants `totale_ht` et `totale_ttc` pour qu'ils aient deux décimales. Ensuite, elle construit le contenu HTML du PDF, incluant des styles CSS pour la mise en page. Le contenu HTML est ensuite chargé dans une instance de Dompdf, qui le rend en PDF. Le fichier PDF généré est ensuite stocké dans le répertoire `public/devis/` avec un nom basé sur l'ID de la demande. La fonction retourne finalement le chemin du fichier PDF créé.

Test de traitement d'une demande pour que le PDF de devis soit généré.





Le PDF généré est ci-joint.



NotificationController

Le `NotificationController` est responsable de la gestion des notifications. Cela inclut l'envoi de notifications lorsque une demande est traitée et lorsqu'un devis est créé. De plus, ce contrôleur gère la suppression d'une notification individuelle ainsi que la suppression de toutes les notifications d'un utilisateur.

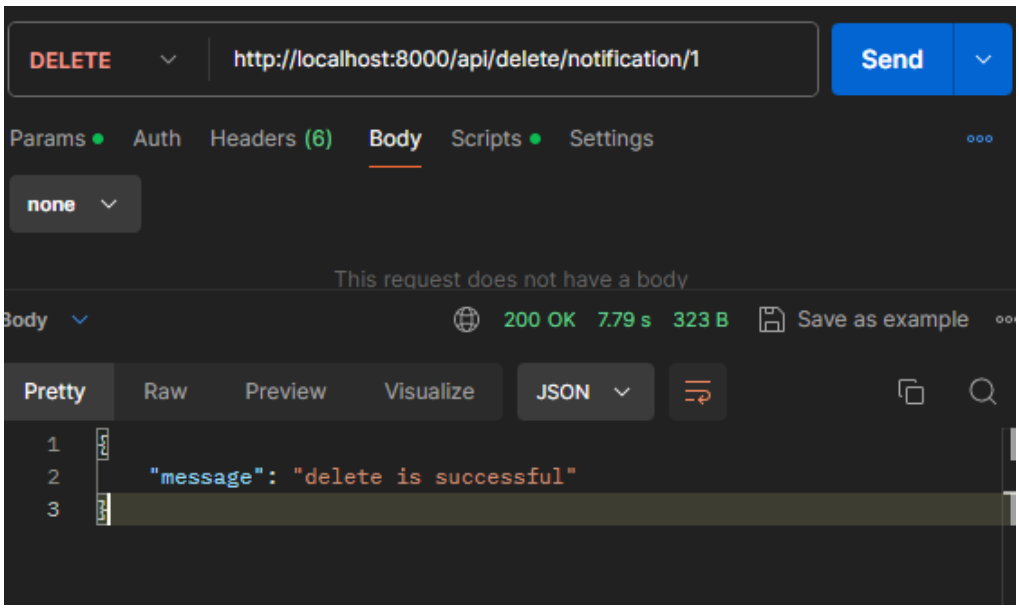
La methode `notifyDevisCreated` du `NotificationController` est utilisée pour créer et envoyer une notification lorsqu'un devis est créé. Elle commence par récupérer et formater le total toutes taxes comprises (TTC) du devis. Ensuite, elle récupère l'ID de l'enfant associé à la demande et, à partir de là, les informations du parent, y compris son ID utilisateur. Une fois toutes les informations nécessaires obtenues, la fonction crée une nouvelle notification dans la base de données, incluant l'ID utilisateur du parent, la date actuelle, le contenu du message indiquant la création du devis avec le montant TTC, et le type de notification.

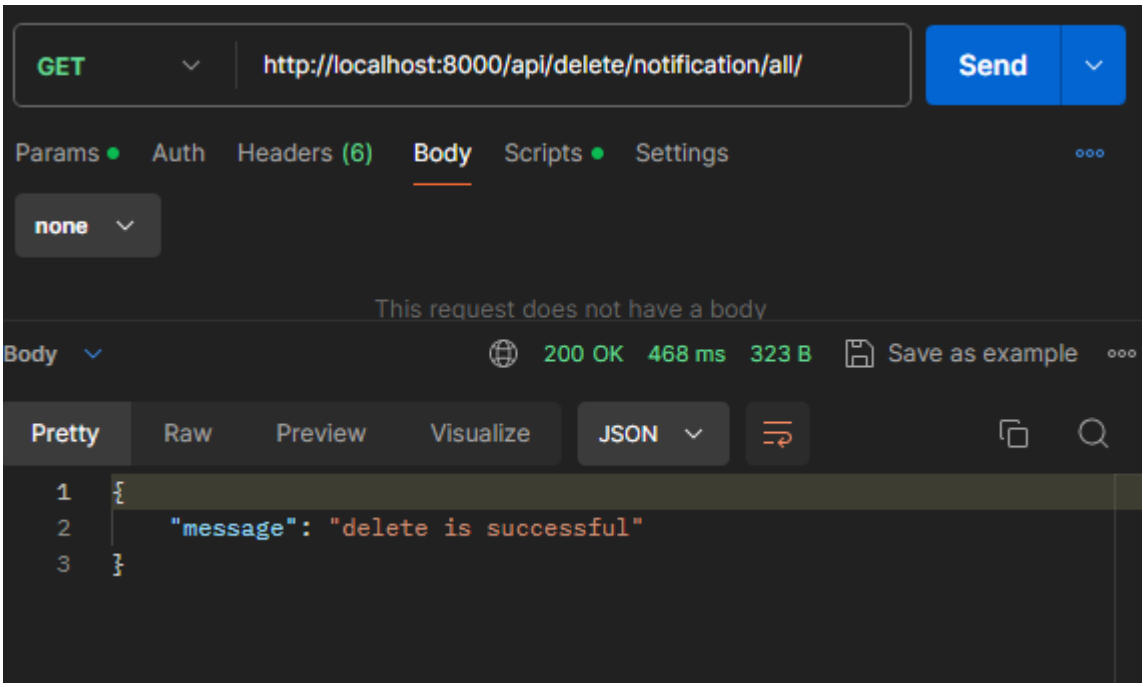
La methode `notifyDemandeHandled` du `NotificationController` est utilisée pour créer et envoyer une notification lorsqu'une demande a été traitée. Elle commence par récupérer l'ID de l'enfant associé à la demande en interrogeant la table `demande_inscriptions`. Ensuite, elle trouve l'enfant correspondant et récupère l'ID du parent. Avec l'ID du parent, elle trouve les informations de l'utilisateur associées. Enfin, elle crée une notification dans la base de données en incluant l'ID de l'utilisateur, la date actuelle, un message indiquant que la demande a été traitée avec succès, et le type de notification.

`notifyDevisCreated` et `notifyDemandeHandled`, n'ont pas de route dédiée. Elles sont appelées depuis le `AdminDemandeController` pour envoyer les notifications appropriées lorsqu'une demande est traitée ou qu'un devis est créé.

deleteNotification & deleteAllUserNotification

```php
1 usage  ABDELOUAHED ABBAD
public function deleteNotification(Notification $notification){
    $notification->delete();
    return response()->json(['message'=>'delete is successful'], status: 200);
}
1 usage  ABDELOUAHED ABBAD *
public function deleteAllUserNotifications(){
    $user_id = auth()->id();
    $notifications = Notification::where('user_id',$user_id)->get();
    foreach ($notifications as $notification) $notification->delete();
    return response()->json(['message'=>'delete is successful'], status: 200);
}
```





La fonction `deleteNotification` est utilisée pour supprimer une notification spécifique. Elle prend un objet `Notification` en paramètre, supprime cette notification de la base de données, puis retourne une réponse JSON confirmant la suppression réussie avec un code de statut HTTP 200.

La fonction `deleteAllUserNotifications` est utilisée pour supprimer toutes les notifications d'un utilisateur authentifié. Elle récupère l'ID de l'utilisateur actuellement authentifié, puis récupère et supprime toutes les notifications associées à cet ID utilisateur. Enfin, elle retourne une réponse JSON confirmant la suppression réussie avec un code de statut HTTP 200.

ROUTE :

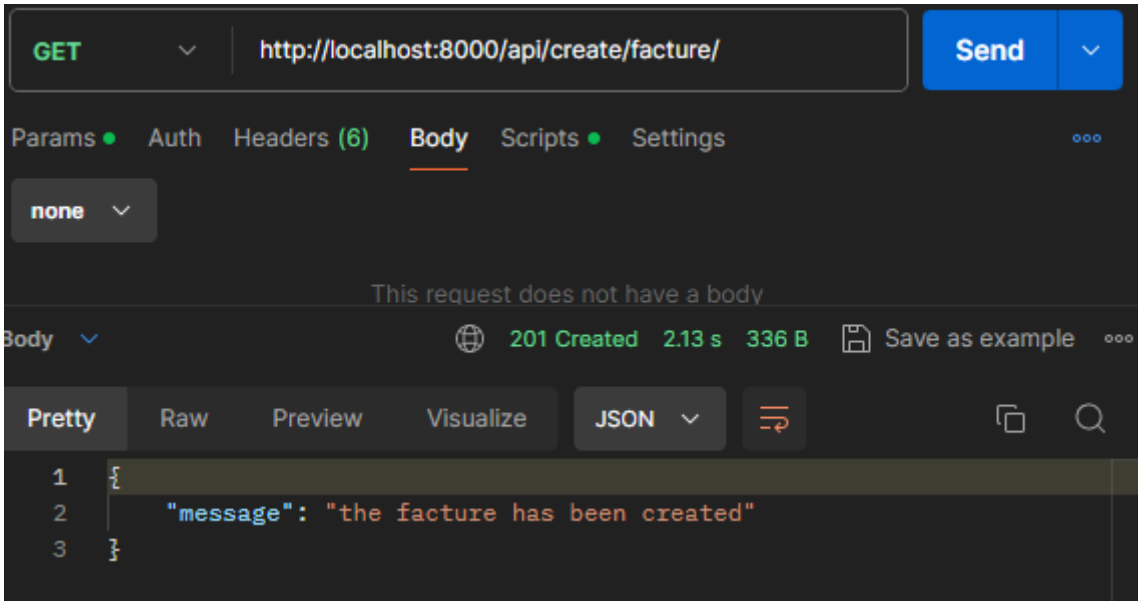
```
Route::delete('/delete/notification/{notification}',[NotificationController::class,'deleteNotification']);

Route::get('/delete/notification/all/',[NotificationController::class,'deleteAllUserNotifications']);
```

ParentFactureController

Le `ParentFactureController` est responsable de la génération des factures pour les devis acceptés par un parent lorsqu'il souhaite effectuer un paiement.

createFacture



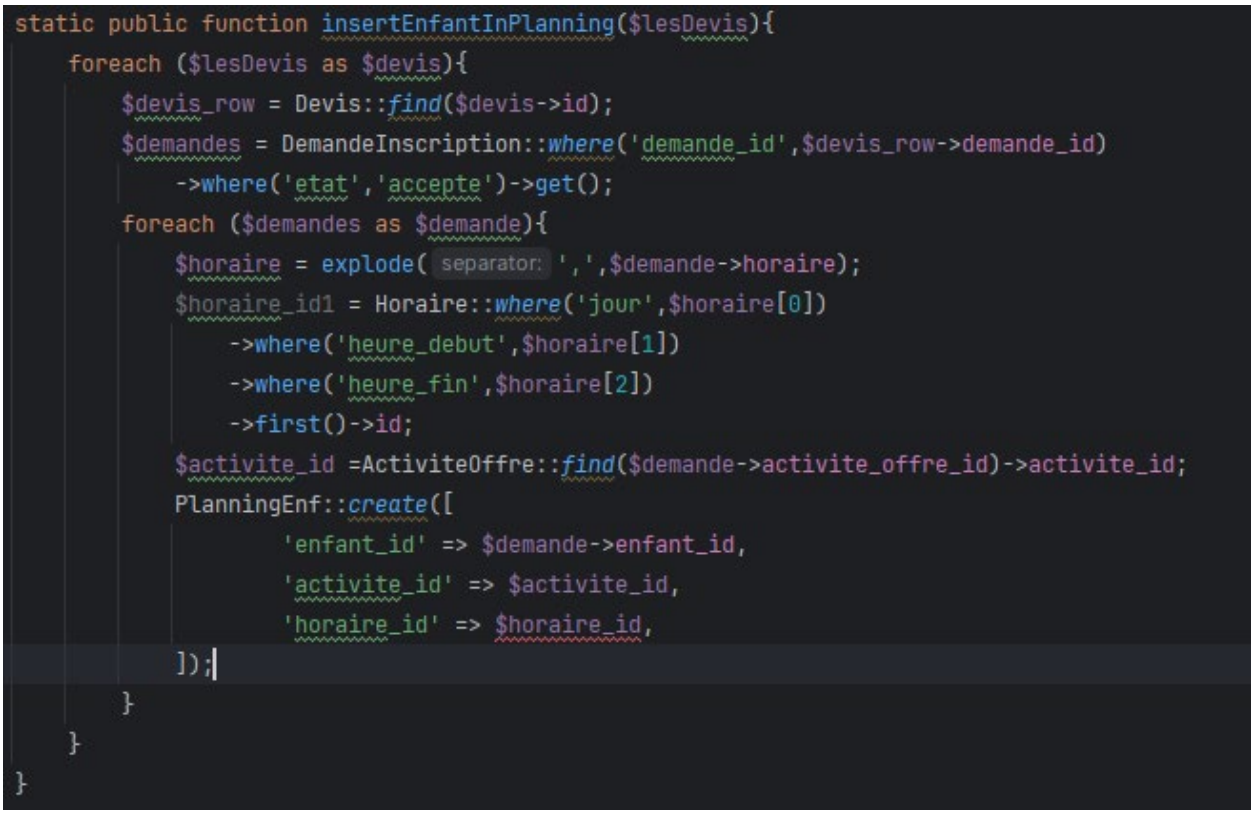
ROUTE :

```
Route::get('/create/facture/',[ParentFactureController:
:class,'createFacture']);
```

AdminPlanningController

Le AdminPlanningController est responsable de la planification, que ce soit pour les enfants ou pour les animateurs.

insertEnfantInPlanning



```php
static public function insertEnfantInPlanning($lesDevis){
    foreach ($lesDevis as $devis){
        $devis_row = Devis::find($devis->id);
        $demandes = DemandeInscription::where('demande_id',$devis_row->demande_id)
            ->where('etat','accepte')->get();
        foreach ($demandes as $demande){
            $horaire = explode( separator: ',',$demande->horaire);
            $horaire_id1 = Horaire::where('jour',$horaire[0])
                ->where('heure_debut',$horaire[1])
                ->where('heure_fin',$horaire[2])
                ->first()->id;
            $activite_id =ActiviteOffre::find($demande->activite_offre_id)->activite_id;
            PlanningEnf::create([
                    'enfant_id' => $demande->enfant_id,
                    'activite_id' => $activite_id,
                    'horaire_id' => $horaire_id,
            ]);
        }
    }
}
```

La methode `insertEnfantInPlanning` est utilisée pour insérer les enfants dans le planning en fonction des devis acceptés. Pour chaque devis, la fonction récupère les demandes associées qui ont été acceptées. Elle extrait les horaires de chaque demande, les sépare, et récupère les IDs des horaires correspondants en fonction du jour, de l'heure de début et de l'heure de fin. Ensuite, elle récupère l'ID de l'activité associée à la demande. Enfin, elle crée des entrées dans la table `PlanningEnf`, associant l'enfant, l'activité et l'horaire spécifique. Cette fonction assure que les enfants sont correctement planifiés pour les activités acceptées en fonction des devis.

La méthode `insertEnfantInPlanning` n'a pas de route dédiée ; elle est appelée depuis le `ParentFactureController`.

AdminHoraireController

Le `AdminHoraireController` est responsable de la création, de la mise à jour et de la suppression des horaires.

createHoraire & updateHoraire & deleteHoraire

```php
1 usage  ABDELOUAHED ABBAD
public function createHoraire(Request $request){
    $formFields = $request->validate([
        'jour' => ['required','in:Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday'],
        'heure_debut' => ['required', 'date_format:H:i'],
        'heure_fin' => ['required', 'date_format:H:i']
    ]);
    Horaire::create($formFields);
    return response()->json(['message'=>'the horaire has been created'], status: 201);
}
1 usage  ABDELOUAHED ABBAD
public function updateHoraire(Request $request,Horaire $horaire){
    $formFields = $request->validate([
        'jour' => ['required', 'in:Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday'],
        'heure_debut' => ['required', 'date_format:H:i'],
        'heure_fin' => ['required', 'date_format:H:i']
    ]);
    $horaire->update($formFields);
    return response()->json(['message'=>'the horaire has been updated'], status: 200);
}
1 usage  ABDELOUAHED ABBAD
public function deleteHoraire(Horaire $horaire){
    $horaire->delete();
    return response()->json(['message'=>'the horaire has been deleted'], status: 200);
}
```

La méthode `createHoraire` est utilisée pour créer un nouvel horaire. Elle valide les données de la requête, crée un nouvel enregistrement d'horaire dans la base de données, et retourne une réponse JSON confirmant la création avec un code de statut HTTP 201.

La méthode `updateHoraire` est utilisée pour mettre à jour un horaire existant. Elle valide les nouvelles données de la requête, met à jour l'enregistrement d'horaire correspondant dans la base de données, et retourne une réponse JSON confirmant la mise à jour avec un code de statut HTTP 200.

La méthode `deleteHoraire` est utilisée pour supprimer un horaire existant. Elle supprime l'enregistrement d'horaire correspondant dans la base de données et retourne une réponse JSON confirmant la suppression avec un code de statut HTTP 200.

ROUTE :

```
Route::post('/create/horaire/',[AdminHoraireController:
:class,'createHoraire']);
Route::put('/update/horaire/',[AdminHoraireController::
class,'updateHoraire']);
Route::put('/update/horaire/',[AdminHoraireController::
class,'deleteHoraire']);
```

FatherController

Le FatherController gère les opérations relatives aux parents.

UpdateFather

Cette méthode permet de mettre à jour le mot de passe et la fonction d'un parent.

```php
public function UpdateFather(Request $request)
 {
    $request->validate([
      'password' => 'required|string|min:6',
       'function' => 'nullable|string|max:255',
    ]);
    $id_user=auth()->id;
   $user= User::find($id_user);
   $user->update(['password' => $request->password]);
   $parent =Father::where('user_id',$id_user)->firstOrFail();
   $parent->update(['fonction' => $request->function]);
   return response()->json(['statut'=>'Modifié avec succès.',200]
```

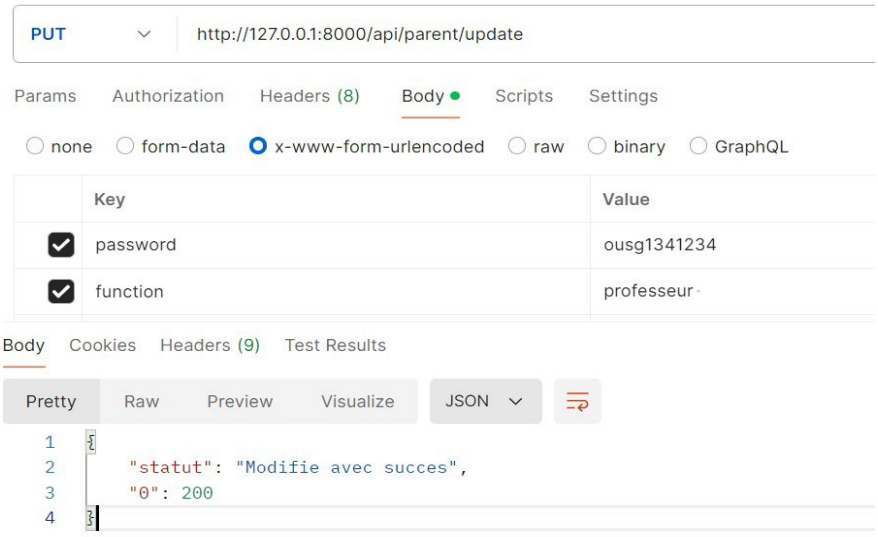
Validation des données du formulaire

Obtention de l'utilisateur actuellement authentifié

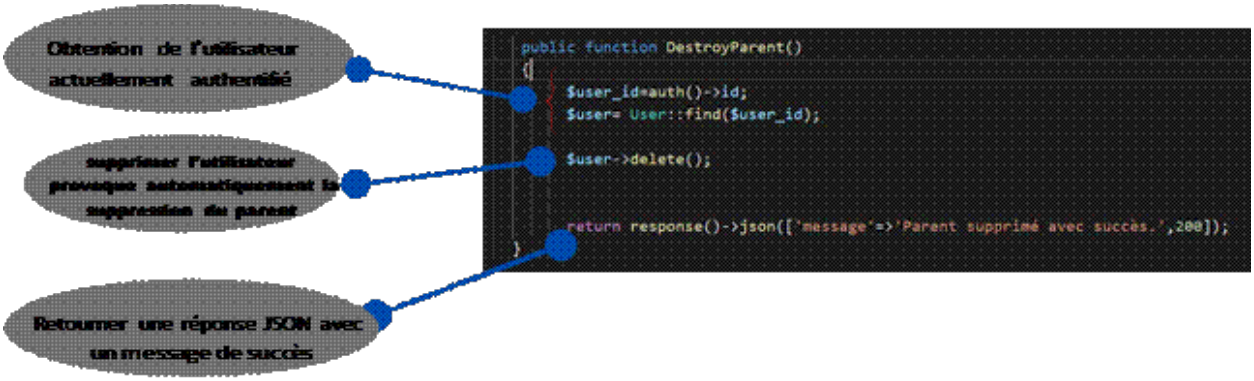
Mise à jour du mot de passe de l'utilisateur

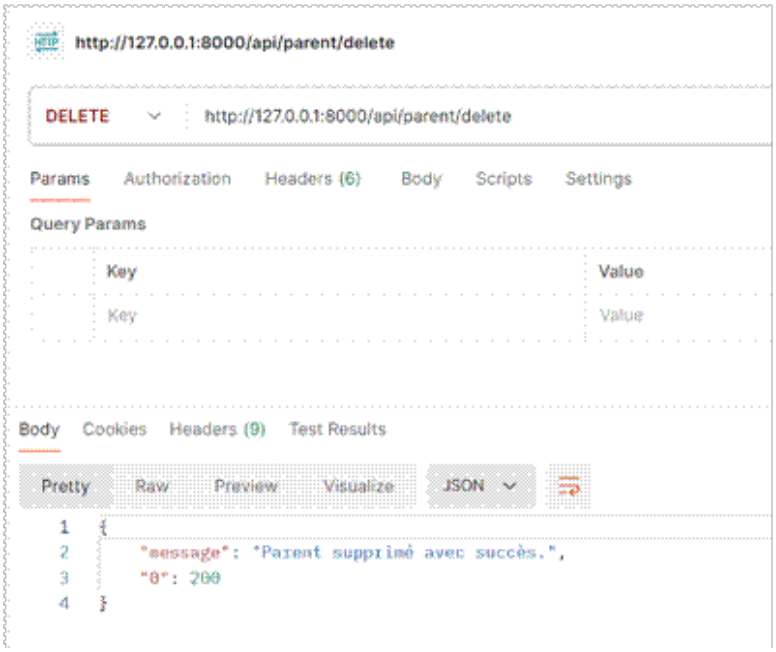
Recherche du parent associé à l'utilisateur et mise à jour de sa fonction

Retourner une réponse JSON avec un message de succès



DestroyParent





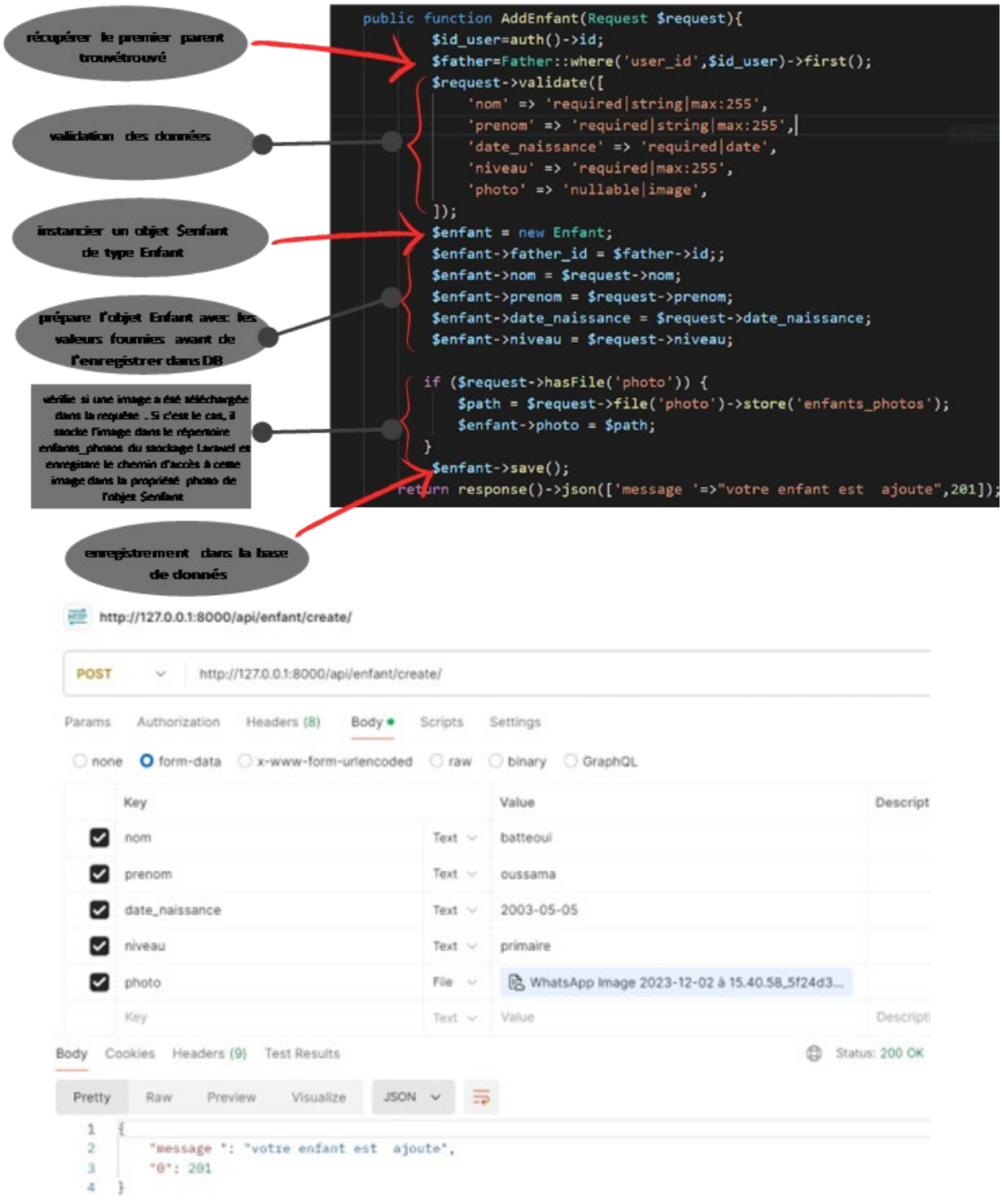
ROUTES :

```
Route::put('/parent/update',[FatherController::class, 'UpdateFather']);
Route::delete('/parent/delete',[FatherController::class, 'DestroyParent']);
```

EnfantController

Le EnfantController gère les opérations relatives aux enfants effectuées par un parent .

AddEnfant

- 
récupérer le premier parent trouvétrouvé
validation des données
instancier un objet $enfant de type Enfant
prépare l'objet Enfant avec les valeurs fournies avant de l'enregistrer dans DB
vérifie si une image a été téléchargée dans la requête . Si c'est le cas, il stocke l'image dans le répertoire enfants_photos du stockage Laravel et enregistre le chemin d'accès à cette image dans la propriété photo de l'objet $enfant
enregistrement dans la base de donnés
public function AddEnfant(Request $request){
$id_user=auth()->id;
$father=Father::where('user_id',$id_user)->first();
$request->validate([
'nom' => 'required|string|max:255',
'prenom' => 'required|string|max:255',
'date_naissance' => 'required|date',
'niveau' => 'required|max:255',
'photo' => 'nullable|image',
]);
$enfant = new Enfant;
$enfant->father_id = $father->id;;
$enfant->nom = $request->nom;
$enfant->prenom = $request->prenom;
$enfant->date_naissance = $request->date_naissance;
$enfant->niveau = $request->niveau;
if ($request->hasFile('photo')) {
$path = $request->file('photo')->store('enfants_photos');
$enfant->photo = $path;
}
$enfant->save();
return response()->json(['message '=>"votre enfant est ajoute",201]);
http://127.0.0.1:8000/api/enfant/create/
POST
http://127.0.0.1:8000/api/enfant/create/
Params
Authorization
Headers (8)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
Key
Value
Descript
nom
Text
batteoui
prenom
Text
oussama
date_naissance
Text
2003-05-05
niveau
Text
primaire
photo
File
WhatsApp Image 2023-12-02 à 15.40.58_5f24d3...
Body
Cookies
Headers (9)
Test Results
Status: 200 OK
Pretty
Raw
Preview
Visualize
JSON
{
"message ": "votre enfant est ajoute",
"0": 201
}

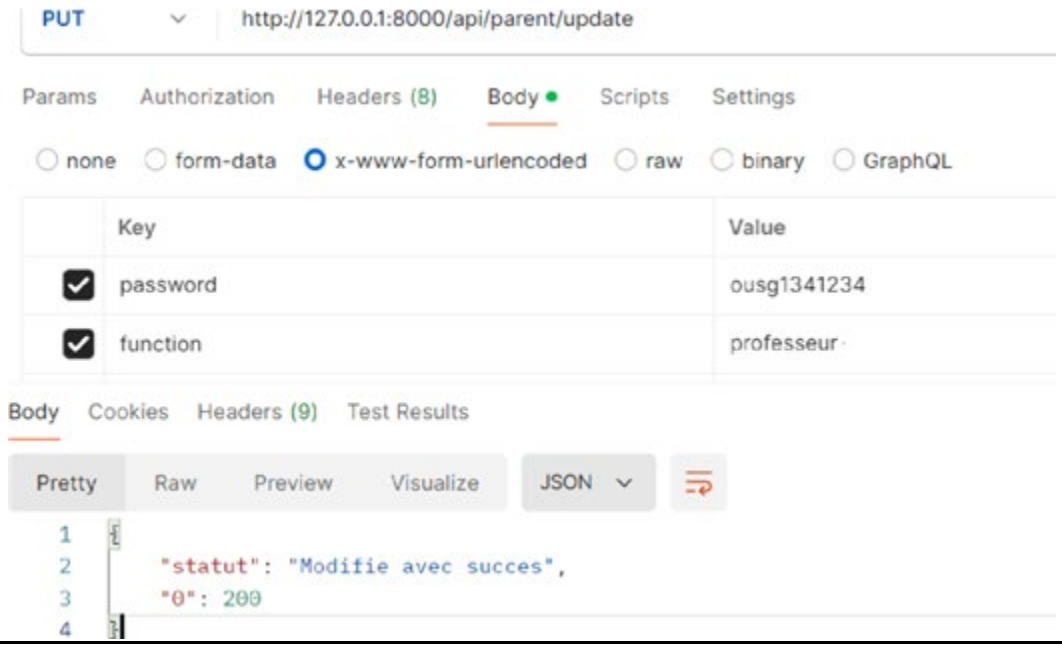
UpdateEnfant

```php
public function UpdateEnfant(Request $request,$id )
    {    $enfant=Enfant::findOrFail($id);
         $request->validate([
          'nom' => 'nullable|string|max:255',
          'prenom' => 'nullable|string|max:255',
          'date_naissance' => 'nullable|date',
          'niveau' => 'nullable|string|max:255',
          'photo' => 'nullable|image'
        ]);
        $data=[ 'nom'=>$request->nom,
                'prenom'=> $request->prenom,
                'date_naissance'=> $request->date_naissance,
                'niveau'=>$request->niveau ,];

        if ($request->hasFile('photo')) {
            $path = $request->file('photo')->store('enfants_photos');
            $data['photo']= $path;
        }
        $enfant->update($data);

        return response()->json(['message'=>'Données de l\'enfant mises à jour avec succes.',200]);
    }
```

utilisation de la methode update permet de modifier les valeurs des attributs de l'objet enfant par les nouveaux valeurs envoyées dans la requête



ROUTES :

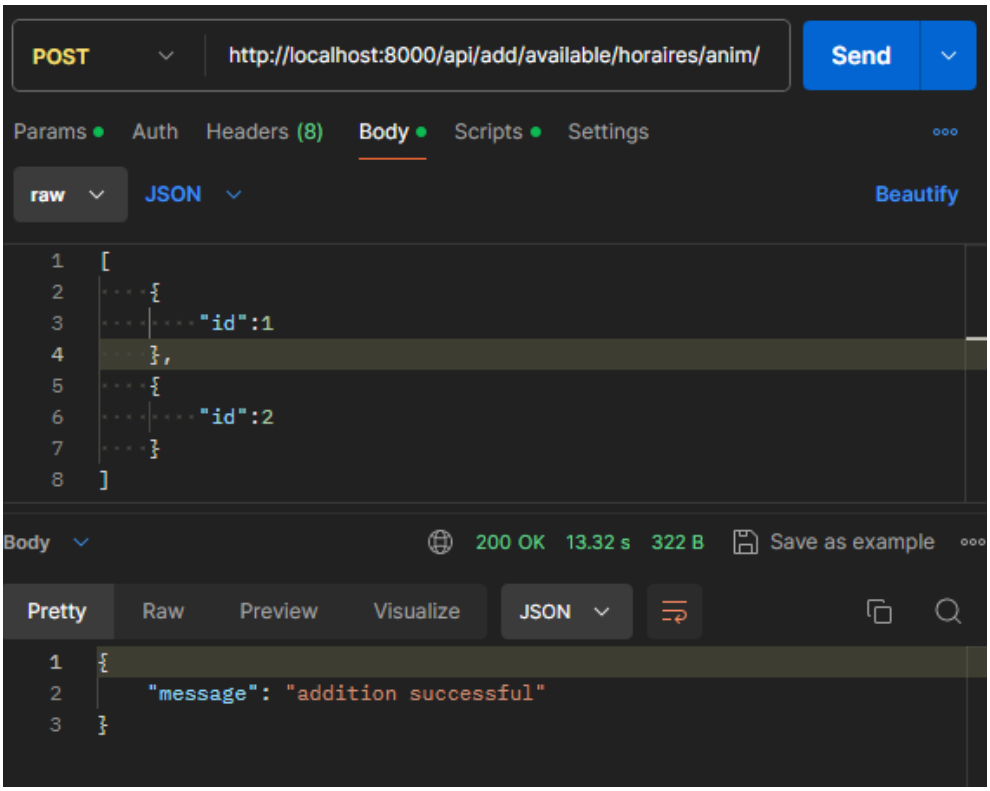
```php
Route::post('/enfant/create/',[EnfantController::class, 'AddEnfant']);
Route::put('/enfant/update/{id}',[EnfantController::class, 'UpdateEnfant']);
Route::delete('/enfant/delete/{id}',[EnfantController::class, 'DestroyEnfant']);
```

AnimatorController

Le AnimatorController est responsable de tout ce qui concerne les animateurs.

addAvailableHour

```php
1 usage   ABDELOUAHED ABBAD *
public function addAvailableHour(Request $request){
    $horaires = $request->json()->all();
    $user_id = auth()->id();
    $anim_id = Animateur::where('user_id',$user_id)->first()->id;
    foreach ($horaires as $horaire)
        HdAnim::create(['anim_id'=>$anim_id,'horaire_id'=>$horaire['id']]);
    return response()->json(['message'=>'addition successful'], status: 200);
}
```



La methode `addAvailableHour` permet d'ajouter des heures disponibles pour un animateur. Elle récupère les heures disponibles depuis la requête, trouve l'ID de l'animateur correspondant à l'utilisateur authentifié, puis crée des enregistrements dans la table `HdAnim` pour chaque horaire fourni.

ROUTE :

```php
Route::post('/add/available/horaires/anim/',[AnimatorController::class,'addAvailableHour']);
```

ShowController

Le ShowController est responsable de tout l'affichage, que ce soit pour les administrateurs, les animateurs ou les parents.

showOffers

```
1 usage   ABDELOUAHED ABBAD
public function showOffers(){
    $offers = Offre::latest()->get();
    foreach ($offers as $offer){
        $activitie = ActiviteOffre::join('activites', 'activites.id', 'activite_offres.activite_id')
            ->select('image_pub')
            ->where('offre_id', $offer->id)->first();
        $offer['image'] = $activitie->image_pub??null;
    }

    return response()->json($offers, status: 200);
}
```

Cette méthode récupère toutes les offres triées par date de création (de la plus récente à la plus ancienne). Pour chaque offre, elle récupère l'image associée à l'activité de l'offre via une jointure avec la table `activites`, puis elle ajoute cette image à l'offre. La réponse JSON renvoie la liste des offres avec leurs images.

showOffer

```
1 usage   ABDELOUAHED ABBAD
public function showOffer(Offre $offer){
    $activitie = ActiviteOffre::join('activites', 'activites.id', 'activite_offres.activite_id')
        ->select('image_pub')
        ->where('offre_id', $offer->id)->first();
    $offer['image'] = $activitie->image_pub??null;
    return response()->json($offer, status: 200);
}
```

Cette méthode prend une offre spécifique en paramètre, récupère l'image associée à l'activité de cette offre via une jointure avec la table `activites`, puis ajoute cette image à l'offre. La réponse JSON renvoie l'offre avec son image.

showDemandesOfAdmin

```
1 usage   ABDELOUAHED ABBAD
public function showDemandesOfAdmin(){
    $user_id = auth()->id();
    $admin = Administrateur::where('user_id', $user_id)->first();
    return response()->json(
        Demande::join('demande_inscriptions', 'demande_inscriptions.demande_id', '=', 'demandes.id')
            ->where('admin_id', $admin->id)
            ->where('statut', 'valide')
            ->where('etat', 'en cours')
            ->select('demandes.id', 'date')
            ->get(),
        status: 200
    );
}
```

Cette méthode récupère l'ID de l'administrateur authentifié et renvoie toutes les demandes validées et en cours associées à cet administrateur, en joignant les tables `demande_inscriptions` et `demandes`. La réponse JSON contient les ID et les dates des demandes.

showTopOffers

```
1 usage   ABDELOUAHED ABBAD
public function showTopOffers()
{
    $offers = Offre::orderBy('remise', 'desc')->limit(3)->get();
    foreach ($offers as $offer){
        $activitie = ActiviteOffre::join('activites', 'activites.id', 'activite_offres.activite_id')
            ->select('image_pub')
            ->where('offre_id', $offer->id)->first();
        $offer['image'] = $activitie->image_pub??null;
    }

    return response()->json($offers, status: 200);
}
```

Cette méthode récupère les trois meilleures offres triées par la remise la plus élevée. Pour chaque offre, elle récupère l'image associée à l'activité de l'offre via une jointure avec la table activites, puis elle ajoute cette image à l'offre. La réponse JSON renvoie les trois meilleures offres avec leurs images.

showRemainingOffers

```
1 usage   ABDELOUAHED ABBAD
public function showRemainingOffers()
{
    $offers = Offre::orderBy('remise', 'desc')->skip(3)->get();
    foreach ($offers as $offer){
        $activitie = ActiviteOffre::join('activites', 'activites.id', 'activite_offres.activite_id')
            ->select('image_pub')
            ->where('offre_id', $offer->id)->first();
        $offer['image'] = $activitie->image_pub??null;
    }
    return response()->json($offers, status: 200);
}
```

Cette méthode récupère toutes les offres sauf les trois premières triées par la remise la plus élevée. Pour chaque offre, elle récupère l'image associée à l'activité de l'offre via une jointure avec la table activites, puis elle ajoute cette image à l'offre. La réponse JSON renvoie les offres restantes avec leurs images.

showActivitiesOfferInOffer

```
1 usage   ABDELOUAHED ABBAD
public function showActivitiesOfferInOffer(Offre $offer)
{
    $activities = ActiviteOffre::where('offre_id', $offer->id)->latest()->get();
    return response()->json($activities, status: 200);
}
```

Cette méthode prend une offre spécifique en paramètre et récupère toutes les activités associées à cette offre, triées par date de création (de la plus récente à la plus ancienne). La réponse JSON renvoie la liste des activités associées à l'offre.

showActivitiesInOfferAllInfos

```
1 usage   ABDELOUAHED ABBAD
public function showActivitiesInOfferAllInfos(Offre $offer)
{
    $activities = ActiviteOffre::join('activites', 'activites.id', 'activite_offres.activite_id')
        ->select('activite_offres.id', 'titre', 'image_pub', 'description',
            'lien_youtube', 'objectifs', 'domaine',
            'tarif', 'tarif_remise', 'age_min', 'age_max', 'nbr_seance',
            'volume_horaire', 'option_paiement')
        ->where('offre_id', $offer->id)->get();
    return response()->json($activities,  status: 200);
}
```

Cette méthode prend une offre spécifique en paramètre et récupère toutes les activités associées à cette offre avec des informations détaillées (titre, image, description, lien YouTube, objectifs, domaine, tarif, tarif remisé, âge minimum et maximum, nombre de séances, volume horaire, et options de paiement). La réponse JSON renvoie la liste des activités avec toutes les informations détaillées.

showActivitiesInOffer

```
1 usage   ABDELOUAHED ABBAD
public function showActivitiesInOffer(Offre $offer)
{
    $activities = ActiviteOffre::join('activites', 'activites.id', 'activite_offres.activite_id')
        ->select('activite_offres.id', 'titre', 'image_pub', 'description',
            'lien_youtube', 'objectifs', 'domaine')
        ->where('offre_id', $offer->id)->get();
    return response()->json($activities,  status: 200);
}
```

Cette méthode prend une offre spécifique en paramètre et récupère toutes les activités associées à cette offre avec des informations de base (titre, image, description, lien YouTube, objectifs, et domaine). La réponse JSON renvoie la liste des activités avec les informations de base.

showHoraireInActivity

```
1 usage   ABDELOUAHED ABBAD
public function showHoraireInActivity($activite_id)
{
    $results = HDA::join('horaires', 'hdas.horaire_id', '=', 'horaires.id')
        ->where('activite_offre_id', $activite_id)
        ->where('nbr_place_restant', '>', 0)
        ->select('horaires.id', 'jour', 'heure_debut', 'heure_fin', 'nbr_place_restant', 'eff_max', 'eff_min')
        ->get();
    return response()->json($results, status: 200);
}
```

Cette méthode prend l'ID d'une activité en paramètre et récupère les horaires disponibles pour cette activité, incluant les jours, heures de début et de fin, nombre de places restantes, effectif maximum et minimum. La réponse JSON renvoie la liste des horaires disponibles pour l'activité spécifiée.

showEnfantInActivity

```
1 usage   ABDELOUAHED ABBAD
public function showEnfantInActivity($activite_id)
{
    $results = Enfant::join('planning_enfs', 'enfants.id', '=', 'planning_enfs.enfant_id')
        ->select('enfants.id', 'enfants.father_id', 'enfants.nom', 'enfants.prenom',
            'enfants.date_naissance', 'enfants.niveau', 'enfants.photo')
        ->where('planning_enfs.activite_id', '=', $activite_id)
        ->get();
    return response()->json($results, status: 200);
}
```

Cette méthode prend l'ID d'une activité en paramètre et récupère tous les enfants inscrits à cette activité, incluant leurs informations de base (ID, ID du père, nom, prénom, date de naissance, niveau, photo). La réponse JSON renvoie la liste des enfants inscrits à l'activité spécifiée

showEnfantOfParent

```
1 usage  ABDELOUAHED ABBAD
public function showEnfantOfParent()
{
    $user_id = auth()->id();
    $parent = Father::where('user_id', $user_id)->first();
    $enfants = Enfant::where('father_id', $parent->id)->get();
    return response()->json([
        $enfants
    ], status: 200);
}
```

Cette méthode récupère l'ID de l'utilisateur authentifié, trouve le parent correspondant, puis récupère tous les enfants associés à ce parent. La réponse JSON renvoie la liste des enfants du parent authentifié.

showTopParentNotifications

```
1 usage  ABDELOUAHED ABBAD
public function showTopParentNotifications()
{
    $user_id = auth()->id();
    $notifications = Notification::where('user_id', $user_id)
        ->orderBy('date', 'desc')
        ->limit(7)
        ->get();
    return response()->json($notifications, status: 200);
}
```

Cette méthode récupère l'ID de l'utilisateur authentifié et renvoie les sept dernières notifications de ce parent, triées par date décroissante. La réponse JSON renvoie la liste des sept dernières notifications du parent.

showRemainingParentNotifications

```php
1 usage  ABDELOUAHED ABBAD
public function showRemainingParentNotifications()
{
    $user_id = auth()->id();
    $notifications = Notification::where('user_id', $user_id)
        ->orderBy('date', 'desc')
        ->skip(7)
        ->get();
    return response()->json($notifications, status: 200);
}
```

Cette méthode récupère l'ID de l'utilisateur authentifié et renvoie toutes les notifications de ce parent au-delà des sept premières, triées par date décroissante. La réponse JSON renvoie la liste des notifications restantes du parent.

showDemandesOfParent

```php
1 usage  ABDELOUAHED ABBAD
public function showDemandesOfParent()
{
    $user_id = auth()->id();
    $parent = Father::where('user_id', $user_id)->first();
    return response()->json(
        Demande::join('demande_inscriptions', 'demande_inscriptions.demande_id', '=', 'demandes.id')
            ->join('enfants', 'demande_inscriptions.enfant_id', 'enfants.id')
            ->where('father_id', $parent->id)
            ->where('statut', 'valide')
            ->select('demandes.id', 'date')
            ->get(),
        status: 200
    );
}
```

Cette méthode récupère l'ID de l'utilisateur authentifié, trouve le parent correspondant, puis renvoie toutes les demandes validées associées à ce parent, en joignant les tables `demande_inscriptions`, `demandes`, et `enfants`. La réponse JSON renvoie la liste des demandes validées du parent avec les dates des demandes.

showActivitiesInDemandeOfParent

```php
1 usage  ABDELOUAHED ABBAD *
public function showActivitiesInDemandeOfParent($demande_id){
    $etat = AdminDemandeController::isDemandeVerified($demande_id);
    $demandes = DemandeInscription::where('demande_id', $demande_id)
        ->select('activite_offre_id')
        ->get();
    $i = 0;
    foreach ($demandes as $demande) {
        $activities[$i] = ActiviteOffre::join('activites', 'activites.id', 'activite_offres.activite_id')
            ->select('activite_offres.id', 'titre', 'image_pub', 'description',
                'lien_youtube', 'objectifs', 'domaine')
            ->where('activite_offres.id', $demande->activite_offre_id)->first();
        $activities[$i++]->etat = $etat;
    }
    return response()->json($activities, status: 200);
}
```

Cette méthode prend l'ID d'une demande en paramètre, vérifie son état, puis récupère les activités associées à cette demande, incluant des informations de base (titre, image, description, lien YouTube, objectifs, et domaine). La réponse JSON renvoie la liste des activités associées à la demande spécifiée avec leur état.

showEnfantInActivityInDemandeOfParent

```php
1 usage  ABDELOUAHED ABBAD
public function showEnfantInActivityInDemandeOfParent($demande_id, $activite_offre_id)
{
    $demandes = DemandeInscription::where('demande_id', $demande_id)
        ->where('activite_offre_id', $activite_offre_id)
        ->select('enfant_id', 'etat')
        ->get();
    $i = 0;
    foreach ($demandes as $demande) {
        $enfants[$i] = Enfant::find($demande->enfant_id);
        $enfants[$i++]->etat = $demande->etat;
    }
    return response()->json($enfants, status: 200);
}
```

Cette méthode prend l'ID d'une demande et l'ID d'une activité en paramètre, puis récupère les enfants associés à cette demande et à cette activité, incluant leur état. La réponse JSON renvoie la liste des enfants associés à la demande et à l'activité spécifiées avec leur état.

showPlaningEnfant

```
1 usage  ABDELOUAHED ABBAD
public function showPlaningEnfant($enfant_id)
{
    $plannings = Horaire::join('planning_enfs', 'horaires.id', '=', 'planning_enfs.horaire_id')
        ->join('activites', 'activites.id', '=', 'planning_enfs.activite_id')
        ->join('planning_anims', 'planning_anims.activite_id', '=', 'activites.id')
        ->join('animateurs', 'animateurs.id', '=', 'planning_anims.anim_id')
        ->join('users', 'users.id', '=', 'animateurs.user_id')
        ->where('enfant_id', $enfant_id)
        ->select('horaires.jour', 'horaires.heure_debut', 'horaires.heure_fin', 'activites.titre', 'users.name')
        ->get();
    return response()->json($plannings, status: 200);
}
```

Cette méthode prend l'ID d'un enfant en paramètre et récupère le planning de cet enfant, incluant les jours, heures de début et de fin, titre des activités, et nom des animateurs. La réponse JSON renvoie le planning complet de l'enfant spécifié.

showAnimateurs

```
1 usage  ABDELOUAHED ABBAD
public function showAnimateurs()
{
    return response()->json(Animateur::latest()->get(), status: 200);
}
```

Cette méthode renvoie la liste de tous les animateurs, triés par date de création (de la plus récente à la plus ancienne). La réponse JSON renvoie la liste complète des animateurs.

showAnimateur

```
1 usage  ABDELOUAHED ABBAD
public function showAnimateur(Animateur $animateur)
{
    return response()->json($animateur, status: 200);
}
```

Cette méthode prend un animateur spécifique en paramètre et renvoie les informations de cet animateur. La réponse JSON renvoie les détails de l'animateur spécifié.

showAllHoraireOfAnimateur

```
1 usage   ABDELOUAHED ABBAD
public function showAllHoraireOfAnimateur(Request $request)
{
    if ($request['anim_id']) $anim_id = $request->anim_id;
    else $anim_id = Animateur::where('user_id', auth()->id())->first()->id;
    $Horaires = Horaire::join('hd_anims', 'hd_anims.horaire_id', '=', 'horaires.id')
        ->where('anim_id', $anim_id)
        ->select('horaires.*')
        ->get();
    return response()->json($Horaires, status: 200);
}
```

Cette méthode peut prendre l'ID d'un animateur en paramètre (ou utilise l'ID de l'utilisateur authentifié si aucun ID n'est fourni), puis renvoie tous les horaires disponibles de cet animateur en joignant les tables `hd_anims` et `horaires`. La réponse JSON renvoie la liste complète des horaires de l'animateur spécifié.

showBusyHoraireOfAnimateurs

```
1 usage   ABDELOUAHED ABBAD
public function showBusyHoraireOfAnimateurs(Request $request)
{
    if ($request['anim_id']) $anim_id = $request->anim_id;
    else $anim_id = Animateur::where('user_id', auth()->id())->first()->id;
    $Horaires = PlanningAnim::join('horaires', 'planning_anims.horaire_id', '=', 'horaires.id')
        ->where('anim_id', $anim_id)
        ->select('horaires.*')
        ->get();
    return response()->json($Horaires, status: 200);
}
```

Cette méthode peut prendre l'ID d'un animateur en paramètre (ou utilise l'ID de l'utilisateur authentifié si aucun ID n'est fourni), puis renvoie tous les horaires occupés de cet animateur en joignant les tables `planning_anims` et `horaires`. La réponse JSON renvoie la liste des horaires occupés de l'animateur spécifié

showAvailableHoraireOfAnimateurs

```php
1 usage  ABDELOUAHED ABBAD
public function showAvailableHoraireOfAnimateurs(Request $request)
{
    if ($request['anim_id']) $anim_id = $request->anim_id;
    else $anim_id = Animateur::where('user_id', auth()->id())->first()->id;
    $horaires = Horaire::join('hd_anims', 'hd_anims.horaire_id', '=', 'horaires.id')
        ->leftJoin('planning_anims', 'planning_anims.horaire_id', '=', 'horaires.id')
        ->where('hd_anims.anim_id', $anim_id)
        ->whereNull('planning_anims.horaire_id')
        ->select('horaires.*')
        ->get();
    return response()->json($horaires, status: 200);
}
```

Cette méthode peut prendre l'ID d'un animateur en paramètre (ou utilise l'ID de l'utilisateur authentifié si aucun ID n'est fourni), puis renvoie tous les horaires disponibles de cet animateur, en s'assurant qu'ils ne sont pas déjà planifiés, en joignant les tables `hd_anims`, `planning_anims`, et `horaires`. La réponse JSON renvoie la liste des horaires disponibles de l'animateur spécifié.

showHoraireForAnimToAdd

```php
1 usage  ABDELOUAHED ABBAD
public function showHoraireForAnimToAdd($anim_id)
{
    $horaires = Horaire::leftJoin('hd_anims', function ($join) use ($anim_id) {
        $join->on('horaires.id', '=', 'hd_anims.horaire_id')
            ->where('hd_anims.anim_id', '=', $anim_id);
    })
        ->whereNull('hd_anims.horaire_id')
        ->select('horaires.*')
        ->get();

    return response()->json($horaires, status: 200);
}
```

Cette méthode prend l'ID d'un animateur en paramètre et renvoie tous les horaires disponibles pour être ajoutés à cet animateur en utilisant une jointure à gauche entre les tables `horaires` et `hd_anims`. La réponse JSON renvoie la liste des horaires disponibles pour l'animateur spécifié.

showOffersFiltered

```
no usages  ABDELOUAHED ABBAD
public function showOffersFiltered($domaine){
    return response()->json(Offre::where('domaine', $domaine)->get(), status: 200);
}
```

Cette méthode prend un domaine spécifique en paramètre et renvoie toutes les offres appartenant à ce domaine. La réponse JSON renvoie la liste des offres filtrées par domaine

showPacks

```
no usages  ABDELOUAHED ABBAD
public function showPacks(){
    return response()->json(Pack::latest()->get(), status: 200);
}
```

Cette méthode renvoie la liste de tous les packs, triés par date de création (de la plus récente à la plus ancienne). La réponse JSON renvoie la liste complète des packs.

showActivities

```
2 usages  ABDELOUAHED ABBAD
public function showActivities(){
    return response()->json(Activite::latest()->get(), status: 200);
}
```

Cette méthode renvoie la liste de toutes les activités, triées par date de création (de la plus récente à la plus ancienne). La réponse JSON renvoie la liste complète des activités.

showParentInfo

```
1 usage   ABDELOUAHED ABBAD *
public function showParentInfo(Request $request){
    if($request['father_id']) $user_id = Father::find($request['father_id'])->first()->user_id;
    else $user_id = auth()->id();
    $parent = User::find($user_id)->select('name','email')->first();
    $parent['fonction'] = Father::where('user_id',$user_id)->first()->fonction;
    return response()->json($parent, status: 200);
}
```

Cette méthode renvoie les informations d'un parent. Si un `father_id` est fourni dans la requête, elle utilise cet ID pour trouver l'ID utilisateur correspondant, sinon elle utilise l'ID de l'utilisateur authentifié. Elle récupère ensuite le nom et l'email de l'utilisateur et ajoute la fonction du parent à ces informations. La réponse JSON renvoie les informations du parent avec un code de statut HTTP 200.

showParents

```
1 usage   ABDELOUAHED ABBAD
public function showParents(){
    return response()->json(Father::latest()->get(), status: 200);
}
```

Cette méthode renvoie la liste de tous les parents, triés par date de création (de la plus récente à la plus ancienne). La réponse JSON renvoie la liste complète des parents avec un code de statut HTTP 200.

ROUTES (ShowController)

```
Route::get('/show/offers/', [ShowController::class,
'showOffers']);

Route::get('/show/offer/{offre}',
[ShowController::class, 'showOffer']);

Route::get('/show/demandes/admin/',
[ShowController::class, 'showDemandesOfAdmin']);

Route::get('/show/offers/top/', [ShowController::class,
'showTopOffers']);

Route::get('/show/offers/remaining/',
[ShowController::class, 'showRemainingOffers']);

Route::get('/show/offer/activities/{offer}',
[ShowController::class, 'showActivitiesInOffer']);

Route::get('/show/offer/activities/more/{offer}',
[ShowController::class, 'showActivitiesOfferInOffer']);

Route::get('/show/offer/activities/all/{offer}',
[ShowController::class,
'showActivitiesInOfferAllInfos']);

Route::get('/show/offer/activity/horaires/{activite_id}
', [ShowController::class, 'showHoraireInActivity']);

Route::get('/show/offer/activity/enfants/{activite_id}'
, [ShowController::class, 'showEnfantInActivity']);

Route::get('/show/parent/enfant/',
[ShowController::class, 'showEnfantOfParent']);

Route::get('/show/notification/parent/top/',
[ShowController::class, 'showTopParentNotifications']);

Route::get('/show/notification/parent/remaining/',
```

```
[ShowController::class,
'showRemainingParentNotifications']);

Route::get('/show/demandes/parent/',
[ShowController::class, 'showDemandesOfParent']);

Route::get('/show/parent/demande/activities/{demande_id
}', [ShowController::class,
'showActivitiesInDemandeOfParent']);

Route::get('/show/parent/demande/activity/enfants/{dema
nde_id}/{activite_offre_id}', [ShowController::class,
'showEnfantInActivityInDemandeOfParent']);

Route::get('/show/enfant/planning/{enfant_id}',
[ShowController::class, 'showPlaningEnfant']);

Route::get('/show/animateurs/', [ShowController::class,
'showAnimateurs']);

Route::get('/show/animateur/{animateur}',
[ShowController::class, 'showAnimateur']);

Route::get('/show/all/horaires/animateur/{anim_id?}',
[ShowController::class, 'showAllHoraireOfAnimateur']);

Route::get('/show/busy/horaires/animateur/{anim_id?}',
[ShowController::class,
'showBusyHoraireOfAnimateurs']);

Route::get('/show/available/horaires/animateur/{anim_id
?}', [ShowController::class,
'showAvailableHoraireOfAnimateurs']);

Route::get('/show/new/horaires/animateur/{anim_id}',
[ShowController::class, 'showHoraireForAnimToAdd']);

Route::get('/show/packs/', [ShowController::class,
'showPacks']);
```

```
Route::get('/show/activities/', [ShowController::class,
'showActivities']);
Route::get('/show/parent/infos/{father_id?}',
[ShowController::class, 'showParentInfo']);
Route::get('/show/parents/', [ShowController::class,
'showParents']);
```

Conclusion

En conclusion, cette documentation a fourni une description détaillée des contrôleurs et des méthodes qui composent notre application. Chaque contrôleur joue un rôle crucial dans le traitement des données et l'interaction avec les utilisateurs, que ce soit pour gérer les activités, les offres, les utilisateurs ou les notifications. La structure modulaire de ces contrôleurs permet une gestion efficace et évolutive de l'application. Les exemples de méthodes présentés illustrent la diversité des opérations supportées, allant de la création et la mise à jour de données à la génération de documents PDF et la gestion des notifications. Cette organisation assure non seulement une répartition claire des responsabilités mais aussi une facilité de maintenance et d'extension future de l'application.