Chemins actuels :

Informations d'accès valables uniquement sur la machine "**vmmathieu01**".
Chemin d'accès : **~/ProjetCoursPrives/express-react-projet**

Pour **build** les **images docker** (les Dockerfile sont dans les dossiers client & server) :

- docker build -t mathieu/coursprives-server-dev-01:1.0 .
- docker build -t mathieu/coursprives-client-dev-01:1.0 .

Images créées docker :
mathieu/coursprives-client-dev-01 1.1 : Image pour le front-end
mathieu/coursprives-server-dev-01 1.1 : Image pour le back-end

Containers déployés :
mathieu/coursprives-client-dev-01:1.1
mathieu/coursprives-server-dev-01:1.1

Pour **lancer** les **containers** :

- docker run -p 3001:3001 --name server-dev-01 -v $(pwd):/usr/src/app mathieu/coursprives-server-dev-01:1.X
- docker run -p 3000:3000 --name client-dev-01 -v $(pwd):/usr/src/app mathieu/coursprives-client-dev-01:1.X

Pour **accéder** à la **base de données postgresql** depuis la vmmathieu01 :

- docker exec -it e3324cef6b93 psql -U server_api -d coursprives_v1

Pour accéder aux **logs** du **serveur NodeJS** (conseillé lors de l'utilisation) :

- docker logs -f 2fa66c6a26c1

Pour **build/redémarrer** le **serveur NodeJS** (Typescript → Javascript) :
Chemin : ~/ProjetCoursPrives/express-react-projet/server

- npm run build

Pour **démarrer** le **serveur front-end** :

Chemin : ~/ProjetCoursPrives/express-react-projet/client

- npm start

Adresses web :
Front-end : http://192.168.1.208:3000/
Back-end : http://192.168.1.208:3001/

Back-End

Utilisation de NodeJs et plus spécifiquement ExpressJS.
L'aspect base de données est géré par ORM sous postgresql.

L'image Docker est une image dév et utilise nodemon qui actualise automatiquement le serveur à chaque changement. Pour build les codes, il faut taper dans le terminal avec le chemin parent du dossier "server" : npm run build. À terme, il faudra utiliser une image qui lance un simple serveur node.

Cette version est une version bêta et possède quelques problèmes d'erreur. Il se peut qu'une erreur non traitée survienne et fait planter le serveur. Pour le redémarrer, il faut exécuter la commande : "npm run build".

Chaque classe d'entité possède une interface commençant par un "I".
Exemple : User →IUser

Tous les dossiers/fichiers dans le dossier parent "Users" sont les entités principales du programme.

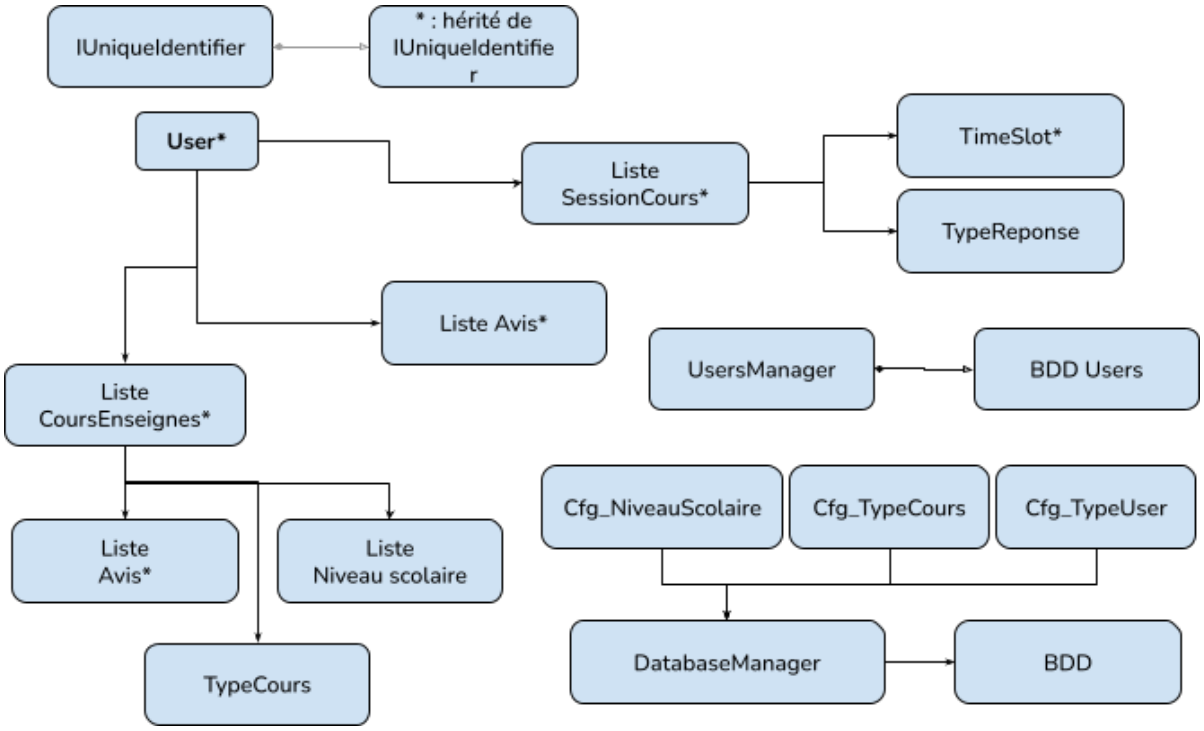
Dans le dossier "Entities", on y trouve les classes pour les configurations. C'est utilisé pour créer les bases de données via ORM vu que tout doit passer par ORM pour pouvoir rechercher facilement les données par la suite.

Dans le dossier "Database", on y retrouve DatabaseManager qui est toute la configuration de la base de données ainsi que la création et l'update des tables si besoin. Ce fichier me permet d'ajouter les lignes ou de commenter les lignes de synchronisation des tables. Attention : Si une table est synchronisée de force, les datas sont perdues.

La configuration de la base de données se trouve également dans le fichier DatabaseManager.

Le but est de laisser une totale liberté au front et laisser la possibilité de tout changer, même les informations privées (faire un front adapter pour ne pas pouvoir changer ce que l'on veut).

Les différentes entités :



User : Gère toutes les données utilisateurs. Toutes les données en lien avec les User et qui sont des entitées sont stockées sous forme de liste d'ID. Les ID sont uniques. Tous les objets dans les tables de la base de données possèdent un unique ID qui est un UUID.

Les différentes routes :

/api/login/:email/:hashPassword

/api/register/:email/:hashPassword
/api/user/get/:uuid
/api/user/get/:uuid/filter
/api/user/get_all
/api/user/update/:id/
/api/user/:uuid/changepassword/:currentPassword/:newPassword
/api/user/update/:id/
/api/coursEnseignes/search
/api/coursEnseignes/info/:idCours
/api/user/:idUser/coursEnseignes/get
/api/user/:idUser/coursEnseignes/remove/:idCours
/api/user/:idUser/coursEnseignes/add
/api/coursEnseignes/:idCours/avis/add
/api/coursEnseignes/:idCours/avis/get
/api/coursEnseignes/:idCours/avis/remove/:idAvis
/api/user/:idUser/avis/get
/api/user/:idUser/avis/add
/api/user/:idUser/session/add
/api/user/:idUser/session/get
/api/user/:idUser/session/pay/:idSession
/api/user/:idUser/session/setReponse/:idSession/:typeReponse
/api/config/listNiveauScolaire
/api/config/listTypeCours
/api/config/listTypeUser
/api/postForum/add
/api/postForum/get/:id
/api/postForum/get_amount/:nb
/api/postForum/remove/:id

Le détail :

/api/login/:email/:hashPassword
Desc : Permet de voir le mot de passe est correct
Arguments :

- ➔ email : email de l'utilisateur

➔ hashPassword : Mot de passe hashé avec la méthode adéquate

Retourne :

➔ Toutes les infos de l'utilisateur si l'identifiant est correct.

➔ Sinon, le statut 400 avec le message d'erreur.

/api/register/:email/:hashPassword

Desc : Permet d'enregistrer un nouveau compte avec l'email et le mot de passe envoyé

Arguments :

➔ email : email de l'utilisateur.

➔ hashPassword : Mot de passe hashé avec la méthode adéquate.

Retourne :

➔ Statut 500 si email déjà utilisé.

➔ Toutes les infos de l'utilisateur si l'identifiant est correct.

/api/user/get/:uuid

Desc : Permet d'obtenir toutes les informations stockées d'un utilisateur

Arguments :

➔ uuid : UUID de l'utilisateur

Retourne :

➔ Statut 400 si UUID inexistant

➔ Toutes les infos de l'utilisateur

/api/user/get/:uuid/filter

Desc : Permet d'obtenir seulement certaines informations d'un utilisateur

Exemple d'utilisation : "/api/user/get/:uuid/filter?Prenom&Nom"

Arguments :

➔ uuid : UUID de l'utilisateur

Retourne :

➔ Statut 400 si UUID inexistant

➔ Toutes les infos demandées de l'utilisateur

/api/user/get_all

Desc : Permet d'obtenir tous les utilisateurs de la base de données

Retourne :

➔ Un tableau avec tous les utilisateurs

/api/user/update/:id/

Desc : Permet de mettre à jour toutes les informations d'un utilisateur

Arguments :

- ➔ id : UUID de l'utilisateur
- ➔ body : L'ensemble des infos de l'utilisateur en json

Retourne :

- ➔ Statut 400 si UUID inexistant
- ➔ Statut 401 si aucune donnée dans le body de la requête
- ➔ Statut 200 si l'utilisateur s'est bien à jour correctement

/api/user/:uuid/changepassword/:currentPassword/:newPassword

Desc : Permet de changer son mot de passe

Arguments :

- ➔ uuid : UUID de l'utilisateur
- ➔ currentPassword : Mot de passe actuel de l'utilisateur
- ➔ newPassword : Nouveau mot de passe

Retourne :

- ➔ Statut 400 si UUID inexistant
- ➔ Statut 402 si mot de passe incorrect
- ➔ Statut 200 si mot de passe changé avec succès

/api/coursEnseignes/search

Desc : Permet de renvoyer tous les cours avec les critères demandés

Arguments :

- ➔ url : Mettre les champs demandés dans l'url (Exemple : /api/coursEnseignes/search?prixMin=20&prixMax=100). Si aucun filtre demandé, tous les cours sont renvoyés

Retourne :

- ➔ Renvoie la liste des cours disponibles selon les critères

/api/coursEnseignes/info/:idCours

Desc : Renvoie les infos de l'utilisateur qui a créé ce cours

Arguments :

- ➔ idCours : UUID du cours

Retourne :

- ➔ Statut 400 si cours introuvable
- ➔ Les informations de l'utilisateur ayant le cours

/api/user/:idUser/coursEnseignes/get

Desc : Renvoie toutes les informations de tous les cours enseignés par l'utilisateur.

Arguments :

- ➔ idUser : UUID de l'utilisateur

Retourne :

- ➔ Statut 400 si utilisateur introuvable
- ➔ Statut 500 si autre problème
- ➔ La liste de tous les cours enseignés avec leur détail

/api/user/:idUser/coursEnseignes/remove/:idCours

Desc : Supprime un cours d'un utilisateur

Arguments :

- ➔ idUser : UUID de l'utilisateur
- ➔ idCours : UUID du cours

Retourne :

- ➔ Statut 400 si utilisateur introuvable
- ➔ Statut 401 si cours introuvable
- ➔ Statut 200 si le cours a bien été supprimé

/api/user/:idUser/coursEnseignes/add

Desc : Créer un cours, le sauvegarde dans la BDD et l'ajoute à l'utilisateur

Arguments :

- ➔ idUser : UUID de l'utilisateur
- ➔ body : Le cours en JSON

Retourne :

- ➔ Statut 400 si utilisateur introuvable
- ➔ Statut 401 si requête invalide
- ➔ Statut 402 si autre erreur
- ➔ Statut 200 avec en json la liste de tous les cours de l'utilisateur

/api/coursEnseignes/:idCours/avis/add

Desc : Créer un avis, le sauvegarde dans la BDD et l'ajoute au cours

Arguments :

- ➔ idCours : UUID du cours
- ➔ body : L'avis en JSON

Retourne :

- ➔ Statut 400 si cours introuvable
- ➔ Statut 401 si requête invalide
- ➔ Statut 402 si autre erreur
- ➔ Statut 200 avec l'avis en json

/api/coursEnseignes/:idCours/avis/get

Desc : Renvoie la liste des avis d'un cours

Arguments :

- ➔ idCours : UUID du cours

Retourne :

- ➔ Statut 400 si cours introuvable
- ➔ Liste des avis en JSON

/api/coursEnseignes/:idCours/avis/remove/:idAvis

Desc : Supprime un avis d'un cours de la base de données

Arguments :

- ➔ idCours : UUID du cours
- ➔ idAvis : UUID de l'avis

Retourne :

- ➔ Statut 400 si cours introuvable
- ➔ Statut 401 si avis introuvable
- ➔ Statut 200 si l'avis a bien été supprimé

/api/user/:idUser/avis/get

Desc : Renvoie la liste des avis d'un utilisateur

Arguments :

- ➔ idUser : UUID de l'utilisateur

Retourne :

- ➔ Statut 400 si utilisateur introuvable
- ➔ Liste de tous les avis

/api/user/:idUser/avis/add

Desc : Créer un avis, le sauvegarde dans la BDD et l'ajoute à l'utilisateur

Arguments :

- ➔ idCours : UUID de l'utilisateur
- ➔ body : L'avis en JSON

Retourne :

- ➔ Statut 400 si cours introuvable
- ➔ Statut 401 si requête invalide
- ➔ Statut 402 si autre erreur
- ➔ Statut 200 avec l'avis en json

/api/user/:idUser/session/add

Desc : Créer une session de cours, la sauvegarde dans la BDD et l'ajoute à l'utilisateur ainsi qu'au professeur

Arguments :

- ➔ idUser : UUID de l'utilisateur élève
- ➔ body : L'avis en JSON

Retourne :

- ➔ Statut 400 si utilisateur élève introuvable
- ➔ Statut 401 si requête invalide
- ➔ Statut 402 si cours enseigné introuvable
- ➔ Statut 403 si date non défini
- ➔ Statut 404 si professeur introuvable
- ➔ Liste des toutes les sessions de l'utilisateur élève

/api/user/:idUser/session/get

Desc : Renvoie la liste des sessions de cours d'un utilisateur

Arguments :

- ➔ idUser : UUID de l'utilisateur

Retourne :

- ➔ Statut 400 si utilisateur introuvable
- ➔ Liste de tous les avis

/api/user/:idUser/session/pay/:idSession

Desc : Paye une session de cours

Arguments :

- ➔ idUser : UUID de l'utilisateur
- ➔ idSession : UUID de la session de cours

Retourne :

- ➔ Statut 400 si utilisateur introuvable
- ➔ Statut 401 si session introuvable
- ➔ Statut 300 si fonds manquants
- ➔ Statut 301 si session déjà payé

- ➔ Statut 200 si la session a bien été payé

/api/user/:idUser/session/setReponse/:idSession/:typeReponse

Desc : Actualise la réponse du professeur sur la demande de session de cours

Arguments :

- ➔ idUser : UUID de l'utilisateur
- ➔ idSession : UUID de la session de cours
- ➔ typeReponse : Doit être 0, 1 ou 2 pour respectivement Attente, Accepté, Refusé

Retourne :

- ➔ Statut 400 si utilisateur introuvable
- ➔ Statut 401 si session introuvable
- ➔ Statut 300 si le type de réponse n'est pas accepté
- ➔ Statut 301 si la session a déjà eu une réponse
- ➔ Statut 200 si la session a bien mis à jour

/api/config/listNiveauScolaire

Desc : Renvoie tous les niveaux scolaire de la base de données

Retourne :

- ➔ Un table avec tous les niveaux scolaire

/api/config/listTypeCours

Desc : Renvoie tous les types de cours proposés de la base de données

Retourne :

- ➔ Un table avec tous les types de cours proposés

/api/config/listTypeUser

Desc : Renvoie tous les types d'utilisateur

Retourne :

- ➔ Un table avec tous les types d'utilisateurs

/api/postForum/add

Desc : Permet de poster un nouveau post. Type = POST

Arguments :

- ➔ body : Le post en JSON

Retourne :

- ➔ Statut 401 si requête invalide

- ➔ Statut 402 si erreur non traitée
- ➔ Statut 200 si post bien posté et retourne le post en json

/api/postForum/get/:id

Desc : Récupère un post à partir de son UUID

Arguments :

- ➔ id : UUID du post

Retourne :

- ➔ Statut 400 si post introuvable
- ➔ Le post en json

/api/postForum/get_amount/:nb

Desc : Récupère X post de la base de données

Arguments :

- ➔ nb : Nombre de post voulu

Retourne :

- ➔ Statut 400 si post introuvable
- ➔ Liste de tous les posts

/api/postForum/remove/:id

Desc : Supprime un post de la base de données

Arguments :

- ➔ id : UUID du post

Retourne :

- ➔ Statut 400 si post introuvable
- ➔ Statut 200 si post supprimé

Front-End

Utilisation de ReactJS.

Réalisation d'un front d'exemple pour montrer les fonctionnalités principales.
Chaque composant à son fichier javascript associé avec toutes les méthodes utilisées.
Dans le dossier Api, nous y retrouvons les structures/classes réalisées dans le backend.
Actuellement, l'utilisateur