Projet SMART - GREENPLAY
2025

Documentation développeur de l'application web

1 tonne de bonnes pratiques de green IT

Réalisé par :
Matéo CHARLOTTE - Mohamed FAKRONI
Huu Thanh Tu HUYNH - Truong Son NGO - Adam SCHLEE - Jixiang SUN - Thi Tho VU

CRÉDITS ET REMERCIEMENTS

Ce document est une reprise de la documentation du projet SMART de l'an dernier, réalisé par :

Thibaut Chantrel, Lou Delcourt, Jade Le Roux, Grégoire Muller, Meije Pigeonnat, Sarah Pignol,

Nous sommes ravis d'avoir pris la suite de votre projet qui, sans travail de documentation, n'aurait pas été possible.

Nous remercions aussi :

Clément Lucas, Alexandre Schlepper, Yvan Bocandé, et Julien Cognet de CGI,

pour leur accompagnement tout au long du projet et leur accueil au sein des locaux de CGI

Enfin merci à :

Stéphane Bres et Lionel Brunie,

pour le suivi du projet et la confiance en celui-ci.

Table des matières

1. Architecture et technologies

a. NestJS avec Websocket socket.io

NestJS est un framework Node.js progressif pour construire des applications server-side efficaces, fiables et évolutives. Il utilise le langage TypeScript et est basé sur les principes de Angular. Nous avons choisi NestJS pour la construction du backend de notre jeu en raison de sa facilité de configuration, de son architecture modulaire et de son intégration transparente avec d'autres bibliothèques et frameworks.

Nous avons utilisé Websocket socket.io, une bibliothèque JavaScript pour les applications web temps réel, pour la communication en temps réel entre le client et le serveur. Cette technologie permet une communication bidirectionnelle entre le client et le serveur, ce qui est essentiel pour un jeu multijoueur interactif.

b. React pour le Frontend

React est une bibliothèque JavaScript open-source pour la construction d'interfaces utilisateur. Il est maintenu par Facebook et une communauté active de développeurs. Nous avons choisi React pour le développement du frontend de notre jeu en raison de sa facilité de création d'interfaces utilisateur dynamiques et interactives, ainsi que de sa gestion efficace de l'état de l'application.

c. PostgreSQL pour la Base de Données

PostgreSQL est un système de gestion de base de données relationnelle open-source et puissant. Nous avons utilisé PostgreSQL comme base de données pour stocker les données du jeu, y compris les informations sur les joueurs, les scores, les parties en cours, etc. PostgreSQL offre une grande robustesse, une grande fiabilité et une compatibilité élevée avec les normes SQL, ce qui en fait un choix idéal pour les applications nécessitant une gestion efficace des données.

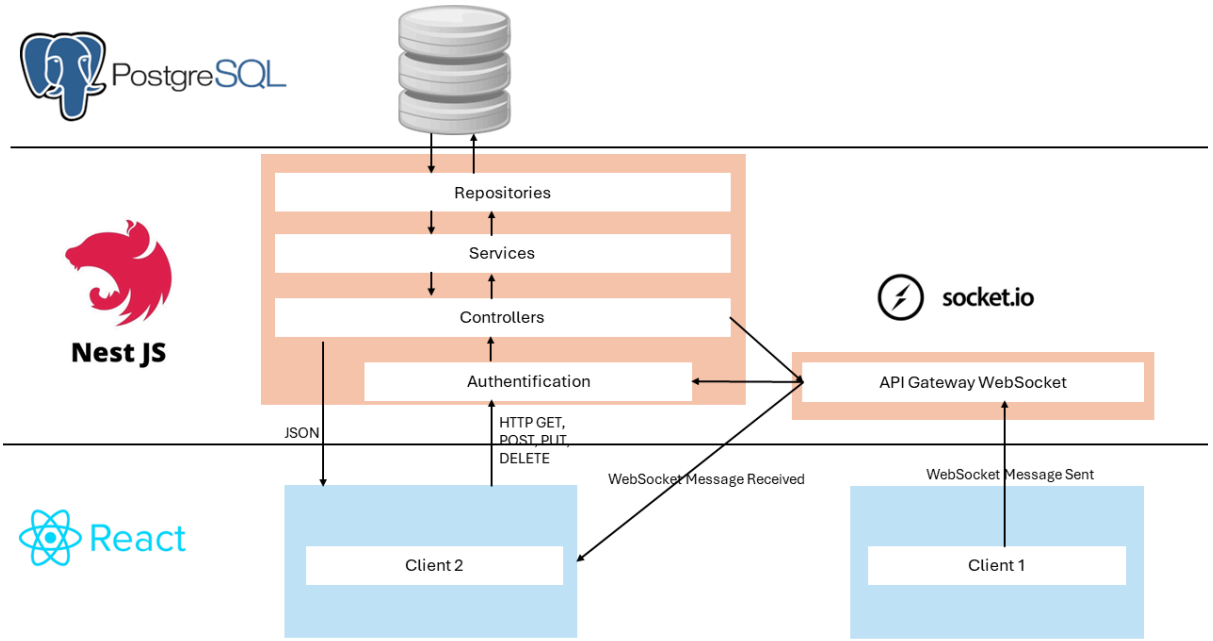


Schéma de l'architecture technique

2. Espace de travail :

Chaque espace de travail est organisé de manière à faciliter le développement, la maintenance et le déploiement du jeu.

a. API (Backend)

L'espace de travail `api` contient le code backend développé avec NestJS. Voici l'organisation générale du code dans cet espace de travail :

- **src/authentification** : Ce répertoire contient les fichiers relatifs à l'authentification des utilisateurs, tels que les stratégies d'authentification, les middleware d'authentification, etc.

- **src/booklet** : Ce répertoire contient les fichiers concernant la gestion des livrets de jeu, qui peuvent contenir des informations sur les règles du jeu, les objectifs, les succès, etc.

- **src/card** : Ce répertoire contient les fichiers liés à la gestion des cartes utilisées dans le jeu. Cela peut inclure la logique pour créer, distribuer et gérer les cartes.

- **src/entity** : Ce répertoire contient les entités de base de données qui représentent les objets principaux de l'application, tels que les utilisateurs, les jeux, les cartes, etc.

- **src/game** : Ce répertoire contient les fichiers liés à la logique métier du jeu.On y trouve la logique pour démarrer, arrêter, mettre en pause, reprendre le jeu, ainsi que pour gérer les tours, les phases, etc.

- **src/sensibilisation** : Ce répertoire contient les fichiers liés à la sensibilisation, tels que les questions de sensibilisation.

- **src/users** : Ce répertoire contient les fichiers concernant la gestion des utilisateurs, tels que les opérations CRUD (création, lecture, mise à jour, suppression) sur les utilisateurs etc.

- **src/websocket** : Ce répertoire contient les fichiers relatifs à la gestion des Websockets pour la communication en temps réel avec les clients. Cela peut inclure les gestionnaires d'événements, les middlewares, etc.

Cette structure permet d'organiser le code de manière logique et cohérente, en regroupant les fonctionnalités connexes dans des répertoires distincts. Cela facilite la navigation dans le code et la collaboration entre les membres de l'équipe de développement.

b. Front (Frontend)

L'espace de travail `front` contient le code frontend développé avec React. Voici l'organisation générale du code dans cet espace de travail :

- **src/components** : Ce répertoire contient les composants réutilisables de notre application React. Les composants peuvent être des éléments d'interface utilisateur tels que des boutons, des formulaires, des cartes, etc.
- **src/pages** : Ce répertoire contient les composants de page de l'application. Chaque fichier peut représenter une page distincte de l'application, par exemple la page d'accueil, la page de profil utilisateur, la page de jeu, etc.
- **src/js/components** : Ce répertoire contient les composants spécifiques TypeScript et CSS qui sont utilisés dans l'application.
- **src/js/hooks** : Ce répertoire contient les hooks personnalisés que l'on a créés pour réutiliser la logique dans nos composants fonctionnels.
- **src/js/pages** : Ce répertoire contient les pages de l'application. Ces composants peuvent être des classes ou des fonctions et sont utilisés pour afficher les différentes pages de l'application.

c. Shared

L'espace de travail `shared` contient le code partagé entre le backend et le frontend. Voici l'organisation générale du code dans cet espace de travail :

- **shared/common** : Ce répertoire contient les éléments communs utilisés à la fois par le client et le serveur. Cela inclut des constantes, des types de données partagés, des utilitaires, des fonctions de validation, etc.
- **shared/client** : Ce répertoire contient le code spécifique au client partagé entre les différentes parties de l'application. Cela inclut des événements clients, des fonctions de gestion des erreurs, des interfaces utilisateur partagées, etc.
- **shared/server** : Ce répertoire contient le code spécifique au serveur partagé entre les différentes parties de l'application. Cela inclut des middlewares, des utilitaires de validation côté serveur, des modèles de données, etc.

3. Gestion des langues

a. Traduction des pages (front-end)

Pour permettre un ajout plus simple de nouvelles langues, la gestion des langues pour le front-end se fait via des fichiers json. Ces fichiers sont rangés dans les répertoires correspondant à la langue

- **front/public/locales/en** (pour l'anglais par exemple)

Chaque json correspond à une page du jeu. Dans chaque fichier json les clés représentent des éléments de la page contenant du texte et les valeurs représentent le texte, dans la langue choisie.

Pour ajouter un élément sur une page (bouton, champ de texte, etc...) contenant du texte, il suffit d'ajouter une paire clé valeur dans les json de langue de cette page et d'utiliser cette même clé pour le texte dans la création de la page (fichier .tsx).

⚠️: En cas de modification d'un json de langue, attention à bien modifier les json de langue du même nom dans **chaque langue**. De même pour la création d'une nouvelle page qui s'accompagne d'un nouveau json de langue.

b. Traduction des cartes

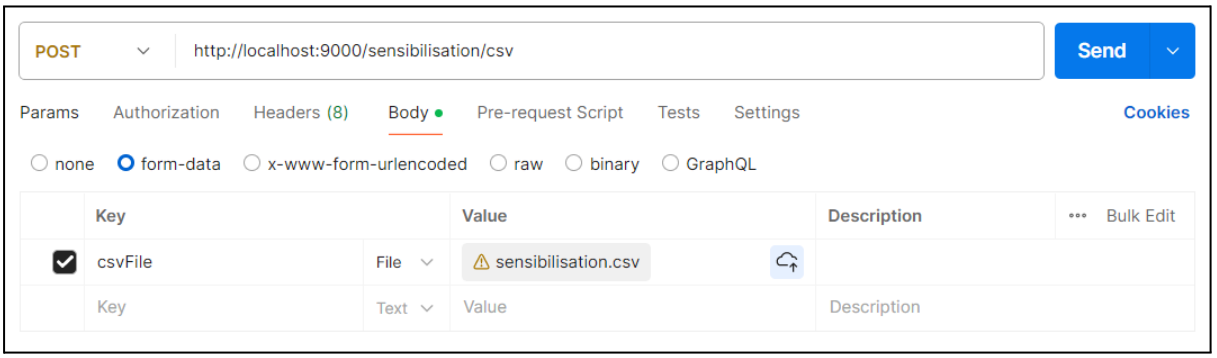
La traduction des cartes est réalisée de façon statique dans le fichier csv de configuration des cartes (/workspaces/api/src/dataCard.csv)
⚠️: l'ajout ou la modification des cartes et des traductions peut se faire dans ce csv, il faut faire attention à bien rajouter et/ou modifier toutes les langues.

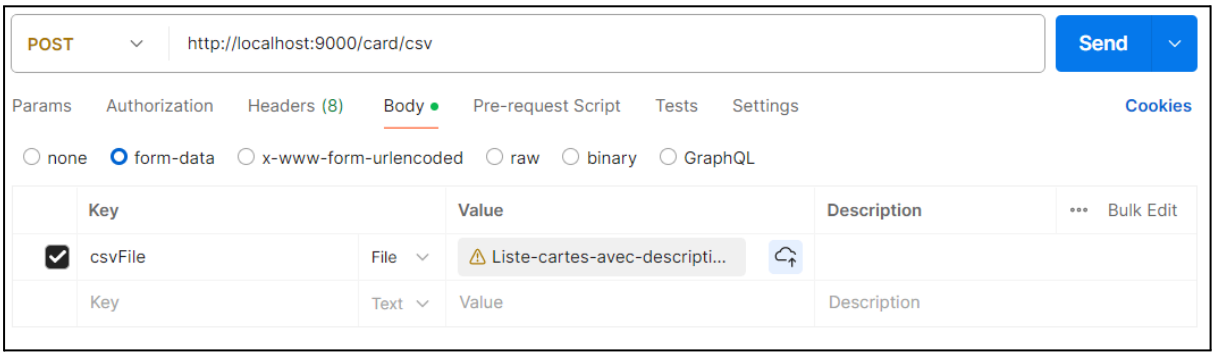
4.Les configurations

a. BD :

Lors de la première installation, il faut initialiser la base de données grâce à une requête.
Pour initialiser une BD vide, deux fichiers .csv sont à disposition pour respectivement charger les cartes à jouer et les quizz de sensibilisation.

Voici deux exemples de requête sur Postman :





b. Structure fichier workspaces/api/.env :

```
DATABASE_USER = <data_base_ name>
DATABASE_PASSWORD = <password_data_base>
DATABASE_HOST = localhost
DATABASE_PORT = 5432
DATABASE_URL = CGI-AGIR-INSA
CORS_ALLOW_ORIGIN = http://localhost:5173
```

c. Démarrage :

Depuis le terminal, dans le répertoire suivant /smartcgi :

- pour run le client : npm run client
- pour run le serveur : npm run server

4. Documentations des services back-end

a. Users

Fonctionnalité	Description	Input	Output
createUser	Créer un utilisateur dans la base de données avec ses attributs	mail: string, password: string, lastname: string, firstname: string	user_id : number
findOne	Trouve un utilisateur dans la base de données à partir de son mail	mail: string	user : User | undefined
getBooklet	Récupère le livret Green IT associé à l'utilisateur	access_token: string	booklet: Green_IT_Booklet
getNbGames	Compte le nombre de parties jouées par l'utilisateur	access_token: string	nb_games: number
getGamesJoined	Récupère l'historique des parties auxquelles l'utilisateur a participé	access_token: string	games: PlayerGameHistory[]
getUserId	Récupère l'ID de l'utilisateur à partir du token d'accès	access_token: string	number
getVictories	Compte le nombre de victoires de l'utilisateur	access_token: string	nb_victories: number
getTotalCO2Saved	Calcule la quantité totale de CO2 économisée par l'utilisateur	access_token: string	total_co2_saved: number
getNbGreenITPractices	Compte le nombre de bonnes pratiques Green IT dans le	access_token: string	nb_green_it_practices: number

	livret de l'utilisateur		
getNbMauvaisePratice	Compte le nombre de mauvaises pratiques dans le livret de l'utilisateur	access_token: string	nb_mauvaise_pratice: number

b. Card

Fonctionnalité	Description	Input	Output
parseCsv	Créer les cartes dans la base de données à partir d'un fichier csv	file: Express.Multer.File	cards: Card[]
getDeck	Créer la pioche initiale de 97 cartes mélangées		cards : Card[]
addCard	Créer une carte dans la base de données	card: Card	addedCard: Card
updateCard	Mettre à jour une carte dans la base de données	card: Card	updatedCard: Card
getCardByIdAndLanguage	Récupérer la carte par son id et la langue de contenu indiqué	id: string language: Language	card: Card
getAllContentsCardById	Récupère une carte par son ID avec tous ses contenus dans différentes langues	id: string	card: MultipleContentCard
getAllCards	Récupérer tous les cartes dans la base de données	language: Language	cards: Card[]
getBestPracticeCardDetails	Récupère les détails des cartes de bonnes pratiques	aucun	{ card_id: number; label: string; language: string }[]
getBadPracticeCa	Récupère les détails	aucun	{ card_id: number;

rdDetails	des cartes de mauvaises pratiques		label: string; language: string }[]

c. Authentification

Fonctionnalité	Description	Input	Output
signUp	Inscrit un utilisateur, appelle le service createUser	mail: string, password: string, lastname: string, firstname: string	success: boolean
signIn	Permet la connexion d'un utilisateur	mail: string, password: string	token: string role: UserRole
signOut	Permet la déconnexion d'un utilisateur	token: string	success: boolean
isConnected	Permet de tester si un user est actuellement connecté	token: string	connected: boolean role: UserRole
getUserByToken	Récupère l'ID de l'utilisateur associé à un token	token: string	userId: number

d. Sensibilisation

Fonctionnalité	Description	Input	Output
parseCsv	Analyse un fichier CSV pour extraire des données de questions	file: Express.Multer.File	questions: Question[]
getSensibilisationQuizz	Récupère une question de sensibilisation pour le quizz		question: SensibilisationQuestion
getSensibilisationQuizzById	Récupère la question de sensibilisation par	id: number	question: QuestionResponse

	son ID		
addQuestion	Créer une question dans la base de données	questionDto: QuestionDto	question: QuestionRespons e
updateQuestionB yId	Mettre à jour une question dans la base de données	id: number, questionDto: QuestionDto	question: QuestionRespons e

e. Game.instance

Fonctionnalité	Description	Input	Output
triggerStart	Démarre une partie en initialisant les decks de cartes et les joueurs	Aucun	Aucun
triggerFinish	Termine la partie et génère un rapport de jeu	ID du gagnant, nom du gagnant	Aucun
answerPracticeQu estion	Enregistre une réponse à une question de pratique et redirige selon le type de carte	ID du joueur, ID de carte, type de carte, réponse	Aucun
answerBestPracti ceQuestion	Enregistre la réponse à une question de meilleure pratique	ID du joueur, ID de carte, réponse	Aucun
answerBadPractic eQuestion	Enregistre la réponse à une question de mauvaise pratique	ID du joueur, ID de carte, réponse	Aucun
answerSensibilisa tionQuestion	Enregistre la réponse à une question de sensibilisation	ID du joueur, réponse	Aucun
discardCard	Jette une carte de la main du joueur	Carte, Socket authentifié	Aucun
playCard	Joue une carte	Carte, Socket	Aucun

	depuis la main du joueur	authentifié	
removeClient	Supprime un joueur de la partie et gère la transition du tour et de l'état du jeu	ID du joueur dans la partie	Aucun
triggerSetLanguage	Définit la langue du jeu pour un joueur et traduit ses cartes	ID du joueur, langue choisie	Aucun
ReceptDrawModeChoice	Enregistre le mode de pioche sélectionné et annonce la dernière carte défaussée	Mode de pioche sélectionné	Aucun
moveToNextState	Gère la transition entre les phases du jeu en fonction de l'état actuel	Aucun	Aucun

f. Game.gateway

Fonctionnalité	Description	Input	Output
onLobbyCreate	Gère la création d'un lobby	Client authentifié, données de création	Aucun
onLobbyJoin	Gère la connexion à un lobby	Client authentifié, données de connexion	Aucun
onLobbyLeave	Gère la déconnexion d'un lobby	Client authentifié	Aucun
onLobbyStartGame	Gère le démarrage d'une partie dans un lobby	Client authentifié, données de démarrage	Aucun
onClientReconnect	Gère la reconnexion d'un	Client authentifié, données de	Aucun

	client	reconnexion	
onPlayCard	Gère le jeu d'une carte dans une partie	Client authentifié, données de jeu	Aucun
onDiscardCard	Gère le jet d'une carte dans une partie	Client authentifié, données de jeu	Aucun
onSensibilisationQuestion	Gère la réponse à une question de sensibilisation	Client authentifié, données de réponse	Aucun
onPracticeQuestionAnswer	Gère la réponse d'un joueur à une question de pratique	Client authentifié, ID de carte, type de carte, réponse	Aucun
onDrawModeChoice	Gère le choix du mode de pioche par un joueur	Client authentifié, mode de pioche sélectionné	Aucun
onAnimationFinished	Gère la fin d'une animation et permet de passer à l'état de jeu suivant	Client authentifié	Aucun
onEndGame	Gère la demande de fin de partie	Client authentifié	Aucun
onChangeLanguage	Gère le changement de langue du joueur	Client authentifié, langue sélectionnée	Aucun

g. Booklet

Fonctionnalité	Description	Input	Output
createBooklet	Crée un nouveau livret Green IT pour un utilisateur	ID de l'utilisateur	Livret Green IT créé
getBooklet	Récupère le livret Green IT d'un utilisateur	ID de l'utilisateur	Livret Green IT récupéré
getAppliedPr	Récupère les bonnes	ID de l'utilisateur	Liste des

actices	pratiques appliquées dans un livret		pratiques appliquées
getBannedPractices	Récupère les mauvaises pratiques bannies dans un livret	ID de l'utilisateur	Liste des pratiques bannies
addBanToBooklet	Ajoute une mauvaise pratique bannie au livret	ID utilisateur, ID de la pratique, priorité	Livret mis à jour
removeBanFromBooklet	Supprime une pratique bannie du livret	ID utilisateur, ID de la pratique	Livret mis à jour
addApplyToBooklet	Ajoute une bonne pratique au livret avec une priorité donnée	ID utilisateur, ID de la pratique, priorité	Livret mis à jour
removeApplyFromBooklet	Supprime une bonne pratique du livret	ID utilisateur, ID de la pratique	Livret mis à jour
updatePriority	Met à jour la priorité d'une pratique (bonne ou mauvaise)	ID utilisateur, ID de la pratique, priorité, type	Livret mis à jour
exportBooklet	Exporte les pratiques (appliquées et bannies) avec leurs titres et descriptions	ID utilisateur	Liste enrichie des pratiques

5. Spécifications

a. Spécifications du back-end

Utilisation de jetons JWT : Le back-end utilise des jetons JWT (JSON Web Tokens) pour gérer l'authentification et l'autorisation des utilisateurs. Ces jetons sont générés lors de la connexion réussie d'un utilisateur et sont utilisés pour authentifier les requêtes ultérieures.

Hachage des mots de passe : Les mots de passe des utilisateurs sont hachés avant d'être stockés dans la base de données. Le hachage est effectué à l'aide de la bibliothèque bcrypt, ce qui garantit la sécurité des mots de passe en les rendant difficilement réversibles.

Gestion des sockets :

b. Spécifications du front-end

Fonctionnalité	Description	Input	Output
`findOne`	Recherche un utilisateur par son email	`mail: string`	`User | undefined`
`createUser`	Crée un nouvel utilisateur dans le système	`mail: string, password: string, lastname: string, firstname: string, role: UserRole`	`{ user_id: number }`
`getBooklet`	Récupère le livret Green IT associé à l'utilisateur	`access_token: string`	`{ booklet: Green_IT_Booklet }`
`getNbGames`	Compte le nombre de parties jouées par l'utilisateur	`access_token: string`	`{ nb_games: number }`
`getGamesJoined`	Récupère l'historique des parties auxquelles l'utilisateur a participé	`access_token: string`	`{ games: PlayerGameHistoryInterface[] }`
`getUserId`	Récupère l'ID de l'utilisateur à partir du token d'accès	`access_token: string`	`number`
`getVictories`	Compte le nombre de victoires de l'utilisateur	`access_token: string`	`{ nb_victories: number }`
`getTotalCO2Saved`	Calcule la quantité totale de CO2 économisée par l'utilisateur	`access_token: string`	`{ total_co2_saved: number }`

`getNbGreenITPractices`	Compte le nombre de bonnes pratiques Green IT dans le livret de l'utilisateur	`access_token: string`	`{ nb_green_it_practices: number }`
`getNbMauvaisePratice`	Compte le nombre de mauvaises pratiques dans le livret de l'utilisateur	`access_token: string`	`{ nb_mauvaise_pratice: number }`

Le service utilise plusieurs repositories pour interagir avec différentes tables de la base de données, notamment les utilisateurs, les livrets Green IT, les parties, et les relations entre ces entités.

Fonctionnalité	Description	Input	Output
`findOne`	Recherche un utilisateur par son email	`mail: string`	`User | undefined`
`createUser`	Crée un nouvel utilisateur dans le système	`mail: string, password: string, lastname: string, firstname: string, role: UserRole`	`{ user_id: number }`
`getBooklet`	Récupère le livret Green IT associé à l'utilisateur	`access_token: string`	`{ booklet: Green_IT_Booklet }`
`getNbGames`	Compte le nombre de parties jouées par l'utilisateur	`access_token: string`	`{ nb_games: number }`
`getGamesJoined`	Récupère l'historique des parties auxquelles l'utilisateur a participé	`access_token: string`	`{ games: PlayerGameHistoryInterface[] }`
`getUserId`	Récupère l'ID de l'utilisateur à partir du token d'accès	`access_token: string`	`number`

`getVictories`	Compte le nombre de victoires de l'utilisateur	`access_token: string`	`{ nb_victories: number }`
`getTotalCO2Saved`	Calcule la quantité totale de CO2 économisée par l'utilisateur	`access_token: string`	`{ total_co2_saved: number }`
`getNbGreenITPractices`	Compte le nombre de bonnes pratiques Green IT dans le livret de l'utilisateur	`access_token: string`	`{ nb_green_it_practices: number }`
`getNbMauvaisePratice`	Compte le nombre de mauvaises pratiques dans le livret de l'utilisateur	`access_token: string`	`{ nb_mauvaise_pratice: number }`

Fonctionnalité	Description	Input	Output
`findOne`	Recherche un utilisateur par son email	`mail: string`	`User | undefined`
`createUser`	Crée un nouvel utilisateur dans le système	`mail: string, password: string, lastname: string, firstname: string, role: UserRole`	`{ user_id: number }`
`getBooklet`	Récupère le livret Green IT associé à l'utilisateur	`access_token: string`	`{ booklet: Green_IT_Booklet }`
`getNbGames`	Compte le nombre de parties jouées par l'utilisateur	`access_token: string`	`{ nb_games: number }`
`getGamesJoined`	Récupère l'historique des parties auxquelles	`access_token: string`	`{ games: PlayerGameHistoryInterface[] }`

	l'utilisateur a participé		
`getUserId`	Récupère l'ID de l'utilisateur à partir du token d'accès	`access_token: string`	`number`
`getVictories`	Compte le nombre de victoires de l'utilisateur	`access_token: string`	`{ nb_victories: number }`
`getTotalCO2Saved`	Calcule la quantité totale de CO2 économisée par l'utilisateur	`access_token: string`	`{ total_co2_saved: number }`
`getNbGreenITPractices`	Compte le nombre de bonnes pratiques Green IT dans le livret de l'utilisateur	`access_token: string`	`{ nb_green_it_practices: number }`
`getNbMauvaisePratice`	Compte le nombre de mauvaises pratiques dans le livret de l'utilisateur	`access_token: string`	`{ nb_mauvaise_pratice: number }`