Documentation de l'API CESEAT

Introduction

Cette API utilise **GraphQL**, ici utilisé comme une alternative aux APIs REST, et qui a notamment la particularité d'être un **langage de requête**.

Cela signifie que notre API possède un seul endpoint (route), par laquelle passe toutes les requêtes:

Endpoint	Méthode
http://localhost:4000/graphql	POST

GraphQL permet de créer une infinité de requêtes, selon l'information que l'on souhaite obtenir. Il est donc impossible de toutes les détailler.

Seront donc présentés ici les différents Object Type, leurs Queries et Mutations possibles. Une liste exhaustive des requêtes possibles pour l'objet User sera également présentée en guise d'exemple.

Chaque requête doit être de type **POST** et doit contenir un body avec une **Query** et des **Variables**.

Format d'une requête

Il existe deux types de requêtes:

Nom	Description	Variables
Query	Permet de récupérer une donnée	Filtres (ex: Filtrer les utilisateurs selon la date de création du compte)
Mutation	Permet d'insérer / mettre à jour / supprimer une donnée	Record (pour l'ajout de données, contient les informations de cette dernière)

Glossaire

Resolver : La fonction qui connecte les schemas avec le backend, et permet donc transformer une requête GraphQL en données. Un resolver peut lire et écrire depuis n'importe où: BDD MySQL, NoSQL, API REST, ...

Object Type : Type d'objet (schéma) représentant une table dans la BDD. Il contient des "fields" qui correspondent à ses différentes propriétés.

User

L'object type USER contient toutes les requêtes CRUD relatives aux comptes utilisateurs.

Cet object type se connecte directement au microservice Users via une API REST. C'est ensuite le microservice qui transforme la requête GraphQL en requête MySQL.

POST [QUERY] Get Users

http://localhost:4000/graphql

Permet de récupérer la liste des utilisateurs inscrits.

Query

```
{
  users {
    ...
  }
}
```

Permissions

Utilisateur	Doit d'accès	Détails
Utilisateur	Non	
Restaurateur	Non	
Livreur	Non	
Service commercial	Oui	
Service technique	Oui	
Développeur tiers	Non	

Champs possibles

Nom	Type	Description
ID	String	Identifiant unique de l'utilisateur (généré par MySQL)
Firstname	String	Le prénom
Lastname	String	Le nom de famille
Password	String	Mot de passe haché avec BCRYPT
PhoneNumber	String	Numéro de téléphone
Avatar	String	Lien vers l'image de profil dans le CDN
SponsorCode	String	Code de parrainage unique à 5 lettres
HasAcceptedGDPR	Boolean	L'utilisateur a-t-il accepté l'utilisation de ses données par le site? (conformément au règlement UE 2016/679)
Role_ID	String	Identifiant du rôle de l'utilisateur*

Nom	Type	Description
CreatedAt	String	Date et heure de création du compte au format UTC suivant la norme ISO 8601

*Valeurs possibles:
1 = Utilisateur
2 = Restaurateur
3 = Livreur
4 = Service commercial
5 = Service technique
6 = Développeur tiers

Exemple de requête

```
{
  users {
    Avatar
    SponsorCode
    CreatedAt
  }
}
```

Exemple de réponse

```
{
    "users": [{
            "Avatar": "/image-1656517931010-972806316.png",
            "SponsorCode": "WDBCX",
            "CreatedAt": "2022-06-29T12:45:26.000Z"
    }]
}
```

POST [QUERY] Get User by ID

http://localhost:4000/graphql

Permet de récupérer un utilisateur spécifique via son ID.

Query

```
{
  userById(ID: ) {
     ...
  }
}
```

Permissions

Utilisateur	Doit d'accès	Détails
Utilisateur	Oui	Uniquement son propre utilisateur et son livreur
Restaurateur	Oui	Uniquement son propre utilisateur et le livreur
Livreur	Oui	Uniquement son propre utilisateur et son client
Service commercial	Oui	
Service technique	Oui	
Développeur tiers	Non	

Champs possibles

Pareils que pour la liste d'utilisateurs.

Paramètres

Nom	Type	Description
ID	String	Identifiant unique de l'utilisateur à chercher

Exemple de requête

```
{
  userById(ID: "1") {
    Firstname
    Lastname
  }
}
```

Exemple de réponse

```
{
    "userById": {
            "Firstname": "Baptiste",
            "Lastname": "Miquel"
    }
}
```

Exemple d'échec

```
{
    "errors": [{
            "message": "Cet utilisateur n'existe pas."
     }]
 }
```

POST [MUTATION] Create User

http://localhost:4000/graphql

Permet de créer un utilisateur.

Mutation

```
mutation USER($record: UserCreateInput) {
  userCreateOne(record: $record) {
    ...
  }
}
```

Permissions

Utilisateur	Doit d'accès	Détails
Utilisateur	Oui	
Restaurateur	Oui	
Livreur	Oui	
Service commercial	Oui	Uniquement des comptes client
Service technique	Non	
Développeur tiers	Oui	

Champs possibles

Nom	Type	Description
token	String	Token JWT signé par le serveur d'authentification
record	User	Utilisateur créé (voir format dans la liste d'utilisateurs)

Paramètres

Nom	Type	Requis?
Firstname	String	Oui
Lastname	String	Oui
Mail	String	Oui
Password	String	Oui

Nom	Type	Requis?
PhoneNumber	String	Non
Avatar	String	Non
Role_ID	String	Non. Valeur par défaut: "1" (Utilisateur)

Exemple de requête

Query

```
{
mutation USER($record: UserCreateInput) {
  userCreateOne(record: $record) {
    token
  }
}
```

Variables

```
{
  "record": {
    "Firstname": "Baptiste",
    "Lastname": "Miquel",
    "Mail": "user@test.com",
    "Password": "123"
  }
}
```

Exemple de réponse

```
{
    "userCreateOne": {
            "token": "..."
    }
}
```

Exemple d'échec

```
{
    "errors": [{
            "message": "Cet utilisateur existe déjà."
     }]
 }
```

POST [MUTATION] Delete User

http://localhost:4000/graphql

Permet de supprimer un utilisateur.

Mutation

```
mutation Mutation($id: String) {
  userDeleteById(ID: $id) {
    ...
  }
}
```

Permissions

Utilisateur	Doit d'accès	Détails
Utilisateur	Oui	Uniquement son propre utilisateur
Restaurateur	Oui	Uniquement son propre utilisateur
Livreur	Oui	Uniquement son propre utilisateur
Service commercial	Oui	
Service technique	Non	
Développeur tiers	Oui	Uniquement son propre utilisateur

Champs possibles

Nom	Type	Description
token	String	Token JWT signé par le serveur d'authentification
record	User	Utilisateur créé (voir format dans la liste d'utilisateurs)

Paramètres

Nom	Type	Requis?
ID	String	Oui

Exemple de requête

Query

```
mutation Mutation($id: String) {
  userDeleteById(ID: $id) {
    token
```

```
  }
}
```

Variables

```
{
  "id": "1"
}
```

Exemple de réponse

```
{
    "userCreateOne": {
            "token": "..."
    }
}
```

Exemple d'échec

```
{
    "errors": [{
            "message": "Cet utilisateur n'existe pas."
     }]
 }
```

POST [MUTATION] Update User

http://localhost:4000/graphql

Permet de mettre à jour un utilisateur.

Mutation

```
mutation($id: String, $record: UserCreateInput) {
  userUpdateById(ID: $id, record: $record) {
    ...
  }
}
```

Permissions

Utilisateur	Doit d'accès	Détails
Utilisateur	Oui	Uniquement son propre utilisateur

Utilisateur	Doit d'accès	Détails
Restaurateur	Oui	Uniquement son propre utilisateur
Livreur	Oui	Uniquement son propre utilisateur
Service commercial	Oui	
Service technique	Non	
Développeur tiers	Oui	Uniquement son propre utilisateur

Champs possibles

Nom	Type	Description
token	String	Token JWT signé par le serveur d'authentification
record	User	Utilisateur créé (voir format dans la liste d'utilisateurs)

Paramètres

Mêmes champs que pour la création d'utilisateur.

Exemple de requête

Query

```
mutation($id: String, $record: UserCreateInput) {
  userUpdateById(ID: $id, record: $record) {
    token
  }
}
```

Variables

```
{
  "id": "1",
  "record": {
    "Firstname": "Florian"
  }
}
```

Exemple de réponse

```
{
    "userUpdateById": {
            "token": "..."
    }
}
```

Exemple d'échec

```
{
    "errors": [{
            "message": "Cet utilisateur n'existe pas."
     }]
}
```

POST [MUTATION] Update User token

http://localhost:4000/graphql

Permet de se connecter et de générer un token JWT d'authentification.

Mutation

```
mutation($mail: String, $password: String) {
  userLogin(Mail: $mail, Password: $password) {
    ...
  }
}
```

Permissions

Utilisateur	Doit d'accès	Détails
Utilisateur	Oui	
Restaurateur	Oui	
Livreur	Oui	
Service commercial	Oui	
Service technique	Oui	
Développeur tiers	Oui	

Champs possibles

Nom	Type	Description
token	String	Token JWT signé par le serveur d'authentification
record	User	Utilisateur (voir format dans la liste d'utilisateurs)

Paramètres

Nom	Type	Requis?

Nom	Type	Requis?
mail	String	Oui
password	String	Oui

Exemple de requête

Query

```
mutation($id: String, $record: UserCreateInput) {
  userUpdateById(ID: $id, record: $record) {
    token
  }
}
```

Variables

```
{
  "mail": "user@test.com",
  "password": "123"
}
```

Exemple de réponse

```
{
    "userLogin": {
            "token": "..."
    }
}
```

Exemple d'échec

```
{
    "errors": [{
            "message": "Mot de passe incorrect."
     }]
 }
```

Autres

Les object types suivants peuvent être demandés à l'API:

Objet	Description

Objet	Description
User (voir ci-dessus)	Utilisateurs (Utilisateur normal, Livreur, Restaurateur, Service commercial, Service technique, Développeur tiers)
Address	Adresses des utilisateurs et des restaurants
Cart	Panier contenant des Produits et Menus
Category	Catégorie de Produits ou Menus
Friends	Lien entre deux utilisateur (parrain et parrainé)
Log	Logs de connexion
Menu	Menu contenant des Produits
Notification	Liste des notifications affichées aux utilisateurs
Order	Commande passée par un utilisateur
OrderStatus	Etat de la commande (En cours de préparation, En cours de livraison, ...)

Objet	Description
Product	Produit vendu par un Restaurant
Promotion	Promotion appliquée à un utilisateur
Restaurant	Un restaurant
RestaurantType	Type de restaurant (ex: Fast-food)
Role	Nom des rôles d'utilisateur et leur description
Ticket	Ticket d'aide envoyé par un restaurant au service technique
TicketStatus	Etat du ticket support (Envoyé, A traiter, En cours de traitement, ...)

Pour chacun de ces objets il existe un CRUD avec les requêtes suivantes:

Nom	Format	Exemple
Get	`ObjectType`s	restaurants
Get by ID	`ObjectType`byId	restaurantById
Create	`ObjectType`CreateOne	restaurantCreateOne
Delete	`ObjectType`DeleteById	restaurantDeleteById
Update	`ObjectType`UpdateById	restaurantUpdateById