

Documentation API  
Models Kit Database

## Table des matières

Révisions.....	3
Description et fonctionnement.....	4
Définitions.....	4
Définitions des routes.....	5
Country.....	5
chemin de base : country/.....	5
Rôle :.....	5
méthodes.....	5
Utilisateurs.....	6
chemin de base : users/.....	6
Rôle :.....	6
méthodes.....	6
Authentification.....	8
chemin de base : auth/.....	8
Rôle :.....	8
méthodes.....	8
Reload authentification.....	8
chemin de base : auth/reload/.....	8
Rôle :.....	8
méthodes.....	8
Refresh authentification.....	9
chemin de base : refresh/.....	9
Rôle :.....	9
méthodes.....	9
Déconnexion.....	9
chemin de base : logout/.....	9
Rôle :.....	9
méthodes.....	9
Category.....	10
chemin de base : category/.....	10
Rôle :.....	10
méthodes.....	10
Brand.....	11
chemin de base : brand/.....	11
Rôle :.....	11
méthodes.....	11
Scale.....	12
chemin de base : scale/.....	12
Rôle :.....	12
méthodes.....	12
Period.....	13
chemin de base : period/.....	13
Rôle :.....	13
méthodes.....	13

## Models Kit Database API

State.....	14
chemin de base : state/.....	14
Rôle :.....	14
méthodes.....	14
Builder.....	15
chemin de base : builder/.....	15
Rôle :.....	15
méthodes.....	15
Model.....	16
chemin de base : model/.....	16
Rôle :.....	16
méthodes.....	16
Routes en liaison avec l'utilisateur : user/.....	18
Routes en liaison avec les favoris : favorite/.....	19
Routes en liaison avec les infos générales : infos/.....	20
Order.....	23
chemin de base : order/.....	23
Rôle : Gère les commandes de l'utilisateur.....	23
méthodes.....	23
Supplier.....	26
chemin de base : supplier/.....	26
Rôle : Gestion des fournisseurs de l'utilisateur.....	26
méthodes.....	26
Statistiques en PDF.....	28
chemin de base : stats/id.....	28
Rôle : Génère un fichier PDF avec les données statistiques.....	28
méthodes.....	28
Administration.....	28
chemin de base : admin/.....	28
Rôle : Administration du site. Cette route est protégée au niveau globale avec accès admin uniquement,.....	28
méthodes.....	28
Nom de la route.....	29
chemin de base : /.....	29
Rôle :.....	29
méthodes.....	29

## Révisions

Version	Date	Auteur
1.0	03/02/2023	G. Crégut
1.0 RCA	10/05/2023	G.Crégut
1.1 RCA	10/06/2023	G.Crégut

## Description et fonctionnement

L'API fournit les informations et les méthodes nécessaires au fonctionnement du site Models Kit Database.

L'accès à certaines informations sont libre de connexion, alors que d'autres sont soumises à authentification, voir à autorisation.

L'authentification se base sur une double utilisation de tokens, un de courte vie et dynamique, l'autre permettant de rafraîchir le premier.

## Définitions

Toutes les routes définies ci dessous se réfèrent à l'URL de base de l'API, te que définie ci-dessous :

`http://url.du.serveur/api/v1/`

Les méthodes marquées d'un astérisque (\*) sont des routes protégées, uniquement accessibles à un utilisateur connecté.

Les méthodes marquées d'un double astérisque (\*\*\*) sont des méthodes soumises à autorisation suivant le niveau de l'utilisateur.

## Définitions des routes

### Country

chemin de base : country/

#### Rôle :

Permet la gestion et la récupération des pays dans la base de données

#### méthodes

##### get / :

données en sortie : Tableau d'objets JSON représentant la liste des pays. 404 si aucun résultat, 500 en cas de défaut serveur

```
[
  {
    id : integer
    name : string
  }
]
```

##### get /:id :

données en entrée : id du pays dans l'URL de la requête

données en sortie : Objet JSON représentant la liste des pays. 404 si aucun résultat, 500 en cas de défaut

```
{
  id : integer
  name : string
}
```

##### \*post / :

données en entrée : Objet JSON contenant name le nom du pays.

```
{
  Name : string
}
```

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur, 201 si créé avec le contenu.

```
{
  id : integer
  name : string
}
```

##### \*\*put /:id :

données en entrée : Objet JSON contenant name le nom du pays.

```
{
  Name : string
}
```

Id du pays à modifier dans l'URL de la requête

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur. 404 si le pays n'existe pas, 204 si la requête a aboutie

##### \*\*delete /:id :

données en entrée : id du pays à supprimer dans l'URL de la requête.

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur. 404 si le pays n'existe pas, 204 si la requête aboutit

## Utilisateurs

chemin de base : users/

### Rôle :

Permet l'ajout, la suppression la modification et la lecture des utilisateurs.

### méthodes

#### \*get / :

données en sortie : tableau d'objets JSON contenant les informations d'un utilisateur. 404 si aucun utilisateurs, et 500 en cas de défaut serveur

```
[
  {
    id: integer
    firstname : string
    Lastname : string
    email : email
    login : string
    rank : integer
  }
]
```

#### \*get / :id :

données en entrée : id de l'utilisateur à récupérer, dans l'URL de la requête.

données en sortie : objet JSON représentant les données d'un utilisateur. Si l'utilisateur n'existe pas erreur 404. En cas de défaut serveur code 500.

```
{
  id: integer
  firstname : string
  Lastname : string
  email : email
  login : string
  rank : integer
}
```

#### post /:

données en entrée : Objet JSON contenant :

firstname, lastname, password, rank (optionnel), email, login

```
{
  firstname : string
  Lastname : string
  password : string
  email : email
  login : string between 8-13
  rank : optionnal value, not implemented yet
}
```

données en sortie : Si une des données est absente ou erronée : 422

Si l'utilisateur existe déjà (login ou email déjà présents) : 409.

En cas de défaut serveur : 500

si la requête aboutit, un objet JSON représentant l'utilisateur ajouté avec son ID.

```
{
  id: integer
  firstname : string
  Lastname : string
  email : email
  login : string
  rank : integer
}
```

## Models Kit Database API

### **\*put /id :**

données en entrée : Objet JSON contenant :

firstname, lastname, password, rank, email, login

Ces valeurs sont optionnelles

id de l'utilisateur dans l'URL.

données en sortie : Si une des données est absente ou erronée : 422. Si l'utilisateur n'existe pas erreur 404, en cas d'erreur serveur 500, et si l'opération réussie, 204.

### **\*delete :**

données en entrée : id de l'utilisateur dans l'URL.

données en sortie : Si l'id n'est pas bonne 422, si l'utilisateur n'existe pas 404.

En cas de défaut serveur 500. Si l'opération réussie, 204.

### **\*post /model :** Ajoute un modèle au stock de l'utilisateur

données en entrée : Objet JSON contenant l'Id du modèle et l'Id de l'utilisateur

```
{
  user: integer
  model : integer
}
```

données en sorties : 422 si une erreur de validation de données, 404 si le modèle n'existe pas, 500 en cas d'erreur, 201 si l'ajout c'est bien réalisé avec objet JSON représentant le modèle ajouté :

```
{
  id: integer
  idModel : integer
  modelName: string
  state : integer
  pictures: string
  owner: integer
  stateName: string
  reference: string
  boxPicture: string
  builderName: string
  scaleName: string
  brandName: string
}
```

### **\*delete /model/:id :** Supprime un kit du stock utilisateur

données en entrées : id du kit à supprimer

données en sorties : 500 si erreur, 404 si non trouvé, 204 si OK

### **\*get /random :** Récupère un modèle au hasard dans le stock

données en sortie :

```
{
  id: integer,
  modelName: string,
  reference: string,
  boxPicture: string,
  builderName: string,
  scaleName: string,
  brandName: string
}
```



**\*\*patch admin/:id:**Change le niveau d'utilisateur

données en entrée : id de l'utilisateur par l'URL. Objet JSON représentant le nouveau niveau.

```
{  
  rank integer  
}
```

données en sortie : 422 si donnée erronée, 404 si utilisateur non trouvé, 500 en cas d'erreur serveur, 204 en cas de réussite.

## Authentification

**chemin de base :** auth/

**Rôle :**

Gère l'authentification d'un utilisateur auprès de l'API.

### méthodes

**get,put,delete :**

données en sortie : erreur 404

**post / :**

données en entrée : objet json avec username et password

```
{  
  login: string  
  password : string  
}
```

données en sortie : si les données en entrées ne sont pas bonnes, erreur 422. Si l'utilisateur n'existe pas, erreur 404. En cas de souci serveur : 500, si l'utilisateur a fourni de mauvaises informations, 401. En cas de succès, renvoie le token de refresh en cookie et le token d'accès en JSON

## Reload authentification

**chemin de base :** auth/reload/

**Rôle :**

Permet, grâce au token stocké dans le cookie et des paramètres (nom prénom) stocké dans le local storage de recréer un token d'accès

### méthodes

**get put, delete:**

données en sortie : erreur 404

**post :**

données en entrée : cookie stocké de refresh, firstname, lastname

```
{  
  firstname: string  
  lastname : string  
}
```

données en sortie : json contenant le nouveau token d'authentification

Si le cookie est absent : erreur 401. Si il y a un problème avec le token en cookie:403.

Erreur serveur : 500

## Refresh authentication

chemin de base : refresh/

Rôle :

Permet, grâce au token stocké dans le cookie de recréer un token d'accès

méthodes

**post put, delete:**

données en sortie : erreur 404

**get :**

données en entrée : cookie stocké de refresh

données en sortie : json contenant le nouveau token d'authentification

Si le cookie est absent : erreur 401. Si il y a un problème avec le token en cookie:403.

Erreur serveur : 500

## Déconnexion

chemin de base : logout/

Rôle :

Permet, grâce au token stocké dans le cookie de recréer un token d'accès

méthodes

**post put, delete:**

données en sortie : erreur 404

**get :**

données en entrée : cookie stocké de refresh

données en sortie : Si le token en cookie n'est pas trouvé dans le système :404.

Erreur serveur : 500. Si la requête aboutit, 204 avec le nouveau token.

## Category

**chemin de base :** category/

**Rôle :**

Permet la gestion et la récupération des catégories dans la base de données

### méthodes

**get /:**

données en sortie : Tableau d'objets JSON représentant la liste des catégories. 404 si aucun résultat, 500 en cas de défaut serveur

```
[
  {
    id : integer
    name : string
  }
]
```

**get /:id :**

données en entrée : id de la catégorie dans l'URL de la requête

données en sortie : Objet JSON représentant la liste des catégories. 404 si aucun résultat, 500 en cas de défaut serveur

```
{
  id : integer
  name : string
}
```

**\*post / :**

données en entrée : Objet JSON contenant name le nom de la catégorie.

```
{
  name : string
}
```

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur, 201 si créé avec la nouvelle valeur :

```
{
  id : integer
  name : string
}
```

**\*\*put / :id :**

données en entrée : Objet JSON contenant name le nom de la catégorie.

```
{
  name : string
}
```

Id de la catégorie à modifier dans l'URL de la requête

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur. 404 si la catégorie n'existe pas, 204 si la requête a aboutie

**\*\*delete / :id :**

données en entrée : id de la catégorie à supprimer dans l'URL de la requête.

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur. 404 si la catégorie n'existe pas, 204 si la requête aboutit

## Brand

**chemin de base :** brand/

### Rôle :

Permet la gestion et la récupération des marques dans la base de données

### méthodes

#### **get /:**

données en sortie : Tableau d'objets JSON représentant la liste des marques. 404 si aucun résultat, 500 en cas de défaut serveur

```
[
  {
    id : integer
    name : string
  }
]
```

#### **get /:id :**

données en entrée : id de la marque dans l'URL de la requête

données en sortie : Objet JSON représentant la marque. 404 si aucun résultat, 500 en cas de défaut serveur

```
{
  id : integer
  name : string
}
```

#### **\*post / :**

données en entrée : Objet JSON contenant name le nom de la marque.

```
{
  name : string
}
```

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur, 201 si créé avec les valeurs

```
{
  id : integer
  name : string
}
```

#### **\*\*put /:id :**

données en entrée : Objet JSON contenant name le nom de la marque.

Id de la marque à modifier dans l'URL de la requête

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur. 404 si la marque n'existe pas, 204 si la requête a aboutie

#### **\*\*delete /:id :**

données en entrée : id de la marque à supprimer dans l'URL de la requête.

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur. 404 si la marque n'existe pas, 204 si la requête aboutit

## Scale

**chemin de base : scale/**

**Rôle :**

Permet la gestion et la récupération des échelles dans la base de données

### méthodes

**get /:**

données en sortie : Tableau d'objets JSON représentant la liste des échelles. 404 si aucun résultat, 500 en cas de défaut serveur

```
[
  {
    id : integer
    name : string
  }
]
```

**get /:id :**

données en entrée : id de l'échelle dans l'URL de la requête

données en sortie : Objet JSON représentant une échelle. 404 si aucun résultat, 500 en cas de défaut serveur

```
{
  id : integer
  name : string
}
```

**\*post / :**

données en entrée : Objet JSON contenant name le nom de l'échelle.

```
{
  name : string
}
```

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur 201 en cas de succès avec les valeurs

```
{
  id : integer
  name : string
}
```

**\*\*put /:id :**

données en entrée : Objet JSON contenant name le nom de l'échelle.

```
{
  name : string
}
```

Id de l'échelle à modifier dans l'URL de la requête

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur. 404 si l'échelle n'existe pas, 204 si la requête a aboutie.

**\*\*delete /:id :**

données en entrée : id de l'échelle à supprimer dans l'URL de la requête.

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur. 404 si l'échelle n'existe pas, 204 si la requête aboutit

## Period

**chemin de base :** period/

**Rôle :**

Permet la gestion et la récupération des périodes dans la base de données

### méthodes

**get /:**

données en sortie : Tableau d'objets JSON représentant la liste des périodes. 404 si aucun résultat, 500 en cas de défaut serveur

```
[
  {
    id : integer
    name : string
  }
]
```

**get /:id :**

données en entrée : id de la période dans l'URL de la requête

données en sortie : Objet JSON représentant une période. 404 si aucun résultat, 500 en cas de défaut serveur

```
{
  id : integer
  name : string
}
```

**\*post /:id :**

données en entrée : Objet JSON contenant name le nom de la période.

```
{
  name : string
}
```

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur, 201 en cas de succès avec les valeurs

```
{
  id : integer
  name : string
}
```

**\*\*put /:id :**

données en entrée : Objet JSON contenant name le nom de la période.

Id de la périodes à modifier dans l'URL de la requête

```
{
  name : string
}
```

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur. 404 si la période n'existe pas, 204 en cas de succès

**\*\*delete /:id :**

données en entrée : id de la période à supprimer dans l'URL de la requête.

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur. 404 si la période n'existe pas, 204 si la requête aboutit

## State

**chemin de base : state/**

**Rôle :**

Permet la gestion et la récupération des états dans la base de données

### méthodes

**get /:**

données en sortie : Tableau d'objets JSON représentant la liste des états. 404 si aucun résultat, 500 en cas de défaut serveur

```
[  
  {  
    id : integer  
    name : string  
  }  
]
```

**get /:id :**

données en entrée : id de l'état dans l'URL de la requête

données en sortie : Objet JSON représentant une échelle. 404 si aucun résultat, 500 en cas de défaut serveur

```
{  
  id : integer  
  name : string  
}
```

**\*post / :**

données en entrée : Objet JSON contenant name le nom de l'état.

```
{  
  name : string  
}
```

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur  
201 si succès, avec les données

```
{  
  id : integer  
  name : string  
}
```

**\*\*put /:id :**

données en entrée : Objet JSON contenant name le nom de l'état.

```
{  
  name : string  
}
```

Id de l'état à modifier dans l'URL de la requête

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur.  
404 si l'état n'existe pas, 204 si la requête a aboutie.

**\*\*delete /:id :**

données en entrée : id de l'état à supprimer dans l'URL de la requête.

données en sortie : 422 si les données ne sont pas valides, 500 en cas d'erreur serveur.  
404 si l'état n'existe pas, 204 si la requête aboutit



## Builder

**chemin de base :** builder/

**Rôle :**

Permet de gérer les constructeurs dans la base de données.

### méthodes

**get / :**

données en sortie : un tableau d'objet JSON représentant les constructeurs

404 si non trouvé, 500 en cas d'erreur du serveur

```
[
  {
    id: integer
    countryId : integer
    name : string
    countryName : string
  }
]
```

**get /:id :**

données en entrée : id : l'ID du constructeur

données en sortie : un objet JSON représentant le constructeur. 404 si non trouvé, 500 en cas d'erreur du serveur

```
{
  id: integer
  countryId : integer
  name : string
  countryName : string
}
```

**\*post /:**

données en entrée : name, country.

données en sortie : un objet JSON représentant le constructeur code 201.

404 si non trouvé, 500 en cas d'erreur du serveur, 401 ou 403 en cas de défaut d'authentification

**\*\*put /:id :**

données en entrée :id, name, country

données en sortie : Un objet JSON représentant le pays modifié {id : integer, name : string}, 404 si non trouvé, 500 en cas d'erreur du serveur, 401 ou 403 en cas de défaut d'authentification

**\*\*delete /:id :**

données en entrée : id, id du constructeur.

données en sortie :204 si la requête a aboutie, 404 si non trouvé, 500 en cas d'erreur du serveur, 401 ou 403 en cas de défaut d'authentification

## Model

**chemin de base :** model/

**Rôle :**

Gère la liste des modèles et l'interaction avec le propriétaire (utilisateur).

### méthodes

**get / :**

données en entrée : Si l'utilisateur est connecté, son id dans le token pour connaître les modèles likés

données en sortie : tableau d'objet représentant les modèles. 404 si non trouvés, 500 si erreur serveur.

```
[
  {
    id: integer
    name: string
    category: integer
    period: integer
    reference: string
    scale: integer
    builder: integer
    brand: integer
    picture: string
    link: string
    categoryName: string
    periodName: string
    scaleName: string
    builderName: string
    brandName: string
    countryId: integer
    countryName: string
    isLiked : boolean
  }
]
```

**get /:id :**

données en entrée : id représentant l'ID du modèle.

données en sortie : Objet représentant le modèle. 404 si non trouvé, 500 si erreur serveur, 422 si l'ID n'est pas au format correct.

```
{
  id: integer
  name: string
  category: integer
  period: integer
  reference: string
  scale: integer
  builder: integer
  brand: integer
  picture: string
  link: string
  categoryName: string
  periodName: string
  scaleName: string
  builderName: string
  brandName: string
  countryId: integer
  countryName: string
}
```

## Models Kit Database API

### **\*post / :**

données en entrée : Un formulaire multipart représentant le modèle à ajouter

```
{
  name: string
  category: integer
  period: integer
  reference: string
  scale: integer
  builder: integer
  brand: integer
  scalemates: string
  file : file object with picture
}
```

données en sortie : 422 si une erreur en entrée, 500 en cas d'erreur serveur, 201 si succès avec données

```
{
  id: integer
  name: string
  category: integer
  period: integer
  reference: string
  scale: integer
  builder: integer
  brand: integer
  picture: string
  link: string
  categoryName: string
  periodName: string
  scaleName: string
  builderName: string
  brandName: string
  countryId: integer
  countryName: string
}
```

### **\*\*put /:id:**

données en entrée : id du modèle dans l'URL, au moins une des données suivantes, dans un formdata multipart :

```
{
  name: string
  category: integer
  period: integer
  reference: string
  scale: integer
  builder: integer
  brand: integer
  scalemates: string
  file : file object with picture
}
```

## Models Kit Database API

données en sortie : 422 en cas d'erreur de saisie, 500 en cas d'erreur, 404 en cas de modèle non trouvé, 204 en cas de modèle trouvé avec les données

```
{
  id: integer
  name: string
  category: integer
  period: integer
  reference: string
  scale: integer
  builder: integer
  brand: integer
  picture: string
  link: string
  categoryName: string
  periodName: string
  scaleName: string
  builderName: string
  brandName: string
  countryId: integer
  countryName: string
}
```

### **\*\*delete /:id:**

données en entrée : id du modèle dans l'url

données en sortie : 404 si le modèle n'existe pas, 500 en cas d'erreur serveur, 204 si succès

### **\*put /stock** : Gère l'état du modèle de l'utilisateur dans le stock

données en entrée : objet JSON représentant l'id du modèle de l'utilisateur, l'id utilisateur et le nouvel état

```
[
  {
    id: integer
    owner : integer
    newState : integer
  }
]
```

données en sorties : 422 si une valeur n'est pas correcte, 500 en cas d'erreur, 409 si le modèle est déjà en favori, 204 si OK

## Routes en liaison avec l'utilisateur : user/

### **\*get /:id** : Récupère la liste des modèles de l'utilisateur

données en entrées : id de l'utilisateur dans l'URL

données en sorties : 422 si l'id n'est pas une donnée correcte, 500 erreur serveur, 418 autrement. liste représentant la liste des modèles de l'utilisateur en cas de succès

```
[
  {
    id: integer
    idModel : integer
    modelName: string
    state : integer
    pictures: string
    owner: integer
    stateName: string
    reference: string
    boxPicture: string
    builderName: string
    scaleName: string
    brandName: string
  }
]
```

### **\*post /picture/:id** : ajoute des photos à la galerie utilisateur pour le modèle

## Models Kit Database API

données en entrées : Id de l'entité de liaison dans l'URL, formulaire multipart contenant :

```
{  
  file : array of file object (max 6)  
}
```

données en sorties : 422 en cas de données invalides, 500 en cas d'échec serveur, 418 autrement. 204 en cas de succès

**\*delete /picture/:id?file=filename** : supprime des photos à la galerie utilisateur pour le modèle

données en entrées : id : Id de l'entité représentant le modèle de l'utilisateur, file le nom du fichier à supprimer.

données en sorties : 422 si une valeur n'est pas correcte, 500 en cas d'erreur, 204 si OK

## Routes en liaison avec les favoris : favorite/

**\*get /:id** : Récupère les modèles favoris de l'utilisateur

données en entrées : id de l'utilisateur dans l'url

données en sortie : tableau listant les modèles favoris de l'utilisateur

```
[  
  {  
    id: integer  
    idModel : integer  
    modelName: string  
    builderName: string  
    scaleName: string  
    brandName: string  
  }  
]
```

**\*post /** : Ajoute ou retire le modèles des favoris

données en entrée : objet JSON contenant l'id du modèle à liker, l'id user et l'état du like

```
{  
  modelId: integer  
  owner : integer  
  like : boolean  
}
```

données en sortie : 422 si une données est incorrecte, 404 si le modèle n'est pas trouvé, 500 en cas d'erreur serveur, 204 en cas de succès.

## Routes en liaison avec les infos générales : infos/

**\*get /:id/user/:iduser** : Récupère toutes les informations détaillées d'un modèle de l'utilisateur

données en entrée : id du kit et id de l'utilisateur

données en sorties : 401 en cas de mauvais id utilisateur, 422 en cas d'erreur de saisie, 500 en cas d'erreur serveur, 418 dans les autres cas. En cas de succès, un objet JSON représentant les données

```
{
  id: integer
  model: integer
  provider: integer
  pictures: {
    baseFolder: string
    files: [
      array of string
    ]
  },
  price: float
  owner: integer
  state: integer
  modelName: string
  picture: string
  reference: string
  scalemates: string
  brandName: string
  periodName: string
  scaleName: string
  builderName: string
  categoryName: string
  providerName: string
}
```

## **get /user/:id** : Récupère les statistiques de l'utilisateur

données en entrée: id de l'utilisateur dans l'URL

données en sortie : 500 si erreur serveur, 422 si l'id n'est pas bon, objet JSON si tout est OK.

```
{
  state: [
    {
      count: integer
      name: Liste de souhaits
    },
    {
      count: integer
      name: En stock
    },
    {
      count: integer
      name: En cours
    },
    {
      count: integer
      name: Terminé
    },
    {
      count: integer
      name: Commandé
    }
  ],
  period: [
    {
      count: integer
      name: string
    },
    {
      count: integer
      name: string
    }
  ],
  category: [
    {
      count: integer
      name: string
    },
    {
      count: integer
      name: string
    },
    {
      count: integer
      name: string
    }
  ],
  provider: [
    {
      count: integer
      name: string
    },
    {
      count: integer
      name: string
    },
    {
      count: integer
      name: string
    }
  ],
}
```

## Models Kit Database API

```
scale: [  
  {  
  
    count: integer  
    r  
  
    name: string  
    },  
  {  
  
    count:  
    integer  
  
    name: string  
    },  
  {  
  
    count:  
    integer  
  
    name: string  
    }  
  ],  
  brand: [  
    {  
  
    count:  
    integer  
  
    name: string  
    },  
    {  
  
    count:  
    integer  
  
    name: string  
    }  
  ],  
  price:  
  float  
}
```



## Order

chemin de base : order/

Rôle : Gère les commandes de l'utilisateur

### méthodes

**\*\*get** /: Récupère toute la liste des commandes des utilisateurs

données en sortie : 500 en cas d'erreur, 200 et un tableau d'objets JSON représentant les commandes

```
[
  {
    providerId: integer
    providerName: string
    ownerId: integer
    reference: string
    models: [
      {
        id: integer
        qty: integer
        price: float
        name: string
      }
    ]
  }
]
```

**\*get** /:id :

données en entrée : La référence de la commande passée en paramètre

données en sortie : 404 si la commande n'est pas trouvée, 500 erreur serveur, 200 et objet JSON si OK

```
{
  providerId: integer
  providerName: string
  ownerId: integer
  reference: string
  models: [
    {
      id: integer
      qty: integer
      price: float
      name: string
    }
  ]
}
```

### **\*post /:**

données en entrée : un objet JSON représentant la commande

```
{
  supplier: integer
  owner: integer
  reference: string
  list: [
    {
      idModel: integer
      qty: integer
      price: float
    }
  ]
}
```

données en sortie : 409 si la commande existe, 422 si une donnée est erronée, 500 erreur serveur, 201 si OK avec objet JSON représentant la commande

```
{
  providerId: integer
  providerName: string
  ownerId: integer
  reference: string
  models: [
    {
      id: integer
      qty: integer
      price: float
      name: string
    }
  ]
}
```

### **put /:id:** Non implémentée

données en entrée :

données en sortie :

### **\*delete /:id:**

données en entrée : référence de la commande dans l'url

données en sortie : 404 si commande non trouvée, 500 si erreur serveur, 204 si OK

## Models Kit Database API

### **\*get /user/:id :**

données en entrée : id de l'utilisateur dans l'URL

données en sortie : 422 si donnée erronée, 500 si erreur serveur, 200 si OK avec tableau d'objets JSON représentant les commandes des utilisateurs

```
[
  {
    providerId: integer
    providerName: string
    ownerId: integer
    reference: string
    models: [
      {
        id: integer
        qty: integer
        price: float
        name: string
      }
    ]
  }
]
```

## Supplier

chemin de base : `supplier/`

Rôle : Gestion des fournisseurs de l'utilisateur

### méthodes

**\*\*get /:** Récupère la liste des fournisseurs

données en sortie : tableau d'objet représentant les fournisseurs

```
[
  {
    id : integer
    owner : integer
    name : string
  }
]
```

**\*get /:id :**

données en entrée : id du fournisseur en URL

données en sortie : Objet JSON représentant le fournisseur

```
{
  id : integer
  owner : integer
  name : string
}
```

**\*post /:**

données en entrée : Un objet JSON contenant les informations id utilisateur et nom du fournisseurs

```
{
  owner : integer
  name : string
}
```

données en sortie : 422 si une donnée est invalide, 500 en cas d'erreur serveur, 201 si OK avec les données

```
{
  id : integer
  owner : integer
  name : string
}
```

**\*put /:id:**

données en entrée : Id du fournisseur à changer en URL et objet JSON contenant les infos

```
{
  owner : integer
  name : string
}
```

données en sortie : 422 si une donnée est invalide, 500 en cas d'erreur serveur, 404 si le fournisseur n'existe pas. 204 si OK

**\*delete /:id:**

données en entrée : Id du fournisseur à changer en URL

## Models Kit Database API

données en sortie : 422 si une donnée est invalide, 500 en cas d'erreur serveur, 404 si le fournisseur n'existe pas. 204 si OK

**\*get /user/:id:** Récupère la liste des fournisseur d'un utilisateur

données en entrée : id de l'utilisateur en URL

données en sortie : 500 si erreur serveur, 422 si erreur sur id, 404 si user non trouvé. Tableau d'objets JSON représentant les fournisseurs si OK

```
[
  {
    id : integer
    owner : integer
    name : string
  }
]
```

## Statistiques en PDF

**chemin de base** : stats/id

**Rôle** : Génère un fichier PDF avec les données statistiques

### méthodes

**\*get /:id :**

données en entrée : id de l'utilisateur dans l'URL

données en sortie : un fichier PDF en stream , 500 en cas d'erreur

## Administration

**chemin de base** : admin/

**Rôle** : Administration du site. Cette route est protégée au niveau globale avec accès admin uniquement,

### méthodes

**\*\*get logs/:id :** récupère les logs serveurs

données en entrée : id du type de logs (1 : info, 2 warnings, 3 errors)

données en sortie : fichier de log en format txt en stream

template route :

## Nom de la route

chemin de base : /

Rôle :

### méthodes

**get /:**

données en sortie :

**get /:id :**

données en entrée :

données en sortie :

**post /:**

données en entrée :

données en sortie :

**put /:id:**

données en entrée :

données en sortie :

**delete /:id:**

données en entrée :

données en sortie :