DOCUMENTATION FONCTIONNELLE

1. Introduction

Cette documentation décrit en détail les fonctionnalités offertes, les étapes de mise en place, et les éventuels problèmes rencontrés.

2. Pré-requis et Installation

2.1 Outils nécessaires

Avant de commencer, assurez-vous d'avoir les outils suivants installés sur votre machine :

- **IntelliJ IDEA** : IDE pour développer et exécuter l'application
- **Docker** : Outil permettant de déployer la base de données
- **Postman** : Utilisé pour tester les requêtes API

2.2 Clonage du projet

Le projet est disponible sur GitHub et doit être cloné avec la commande suivante :

3. Récupérer le projet sur GitHub

3.1 Qu'est-ce que GitHub ?

GitHub est une plateforme où des projets de développement sont stockés et gérés avec **Git**, un outil de gestion de versions.

3.2 Étapes pour cloner le projet depuis GitHub

1. **Ouvrir un terminal** sur votre ordinateur.

2. **Se placer dans un dossier de travail** où vous souhaitez télécharger le projet. Par exemple, sur Windows :

```
cd C:\Users\VotreNom\Documents
```

Ou sur Linux/Mac :

```
cd ~/Documents
```

3. **Exécuter la commande de clonage** :

```
git clone https://github.com/ALEX67U/prod-logiciel
```

☑ Cela va télécharger tous les fichiers du projet dans un dossier nommé **prod-logiciel**.

4. **Se déplacer dans le dossier du projet** :

```
cd prod-logiciel
```

☑ Maintenant, vous avez récupéré le code source sur votre machine.

4. Ouvrir le projet dans IntelliJ IDEA

4.1 Qu'est-ce qu'IntelliJ IDEA ?

IntelliJ IDEA est un **IDE (Environnement de Développement Intégré)** qui permet d'écrire, exécuter et déboguer du code Java.

4.2 Étapes pour ouvrir le projet dans IntelliJ

1. **Ouvrir IntelliJ IDEA**.
2. **Cliquer sur** : **"Open"** (Ouvrir un projet).
3. **Sélectionner le dossier du projet** :
   📂 **prod-logiciel** (là où vous avez cloné le projet).
4. **Valider** en cliquant sur **OK**.
5. **Attendre que IntelliJ télécharge les dépendances** (cela peut prendre quelques minutes).

☑ Le projet est maintenant prêt à être modifié ou exécuté.

5. Créer la base de données avec Docker

5.1 Qu'est-ce que Docker ?

Docker est un outil qui permet de **créer et gérer des bases de données** et d'autres services facilement à l'aide de conteneurs.

5.2 Étapes pour lancer la base de données

1. **Vérifier que Docker est installé** et lancé sur votre machine.

- Sur Windows : Rechercher **Docker Desktop** et l'ouvrir.
- Sur Linux/Mac : Lancer Docker en exécutant :

```
docker --version
```

☑ S'il renvoie une version, Docker est installé.

2. **Ouvrir un terminal et se placer dans le dossier du projet** :

- Sur Windows

```
cd C:\Users\VotreNom\Documents\prod-logiciel
```

- sur Linux/Mac

```
cd ~/Documents/prod-logiciel
```

3. **Lancer la base de données en exécutant le fichier Docker** :

```
docker-compose -f resources/docker.yml up -d
```

- **Explication** :

  `docker-compose` → Commande pour exécuter plusieurs services Docker.

  `-f resources/docker.yml` → Indique le fichier de configuration pour créer la base de données.

  `up -d` → Démarre les services en arrière-plan.

4. **Vérifier que la base de données est bien créée** :

```
docker ps
```

- Cette commande affiche la liste des conteneurs en cours d'exécution. Vous devriez voir la base de données.

☑ La base de données est maintenant prête à être utilisée.

6. Lancer le projet (Démarrer l'application)

6.1 Étapes pour exécuter l'application

1. **Retourner dans IntelliJ IDEA**.
2. **Ouvrir la classe principale** :
   📂 **src/main/java/com/example/MySurveyApplication.java**.
3. **Exécuter le projet** :
   - Soit en cliquant sur le bouton ▶ **(Run)** en haut à droite d'IntelliJ.
   - Soit en appuyant sur **Shift + F10** sur Windows/Linux ou **Cmd + R** sur Mac.

6.2 Vérification

- Si tout se passe bien, IntelliJ affichera des logs indiquant que le serveur est démarré.

L'API sera disponible à l'adresse :

```
https://prod-logiciel.onrender.com/api/...
```

- (Les endpoints spécifiques sont à tester dans Postman).

☑ L'application est maintenant en cours d'exécution.

7. Tester les requêtes avec Postman

7.1 Qu'est-ce que Postman ?

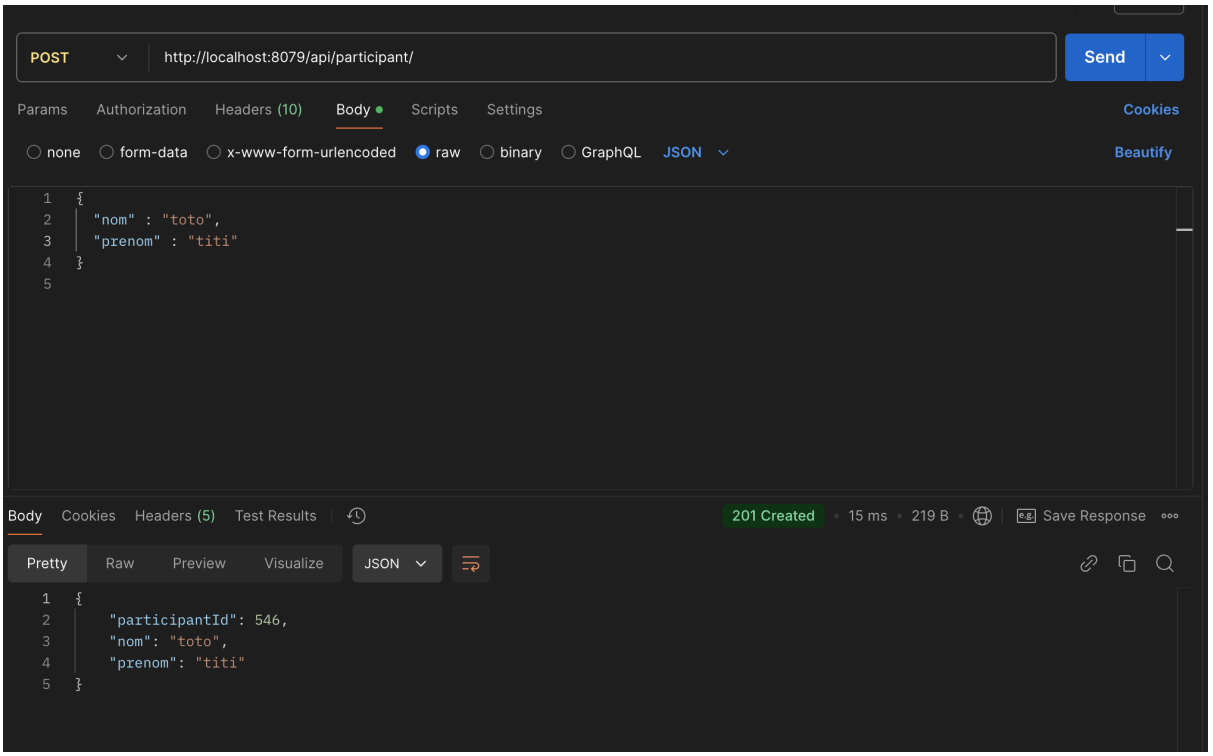
Postman est un outil permettant de **tester les API** sans écrire de code.

7.2 Étapes pour tester l'API

1. **Ouvrir Postman**.

2. Commencer les testes

A. Création d'un participant



Dans ce test, une requête **POST** est envoyée à l'API pour **ajouter un participant** à la base de données.

- **Méthode HTTP** : `POST`
- **URL** : `http://localhost:8079/api/participant/`
- **Headers** :
  Le contenu est envoyé en **JSON** (`Content-Type: application/json`).
- **Body** (Données envoyées) :
  json
  ```
  {
        "nom": "toto",
        "prenom": "titi"
  }
  ```
  Le **nom** du participant est `"toto"`.
  Le **prénom** est `"titi"`.

☑ **Réponse obtenue :**

- **Statut HTTP** : `201 Created` (Indique que la ressource a été créée avec succès).

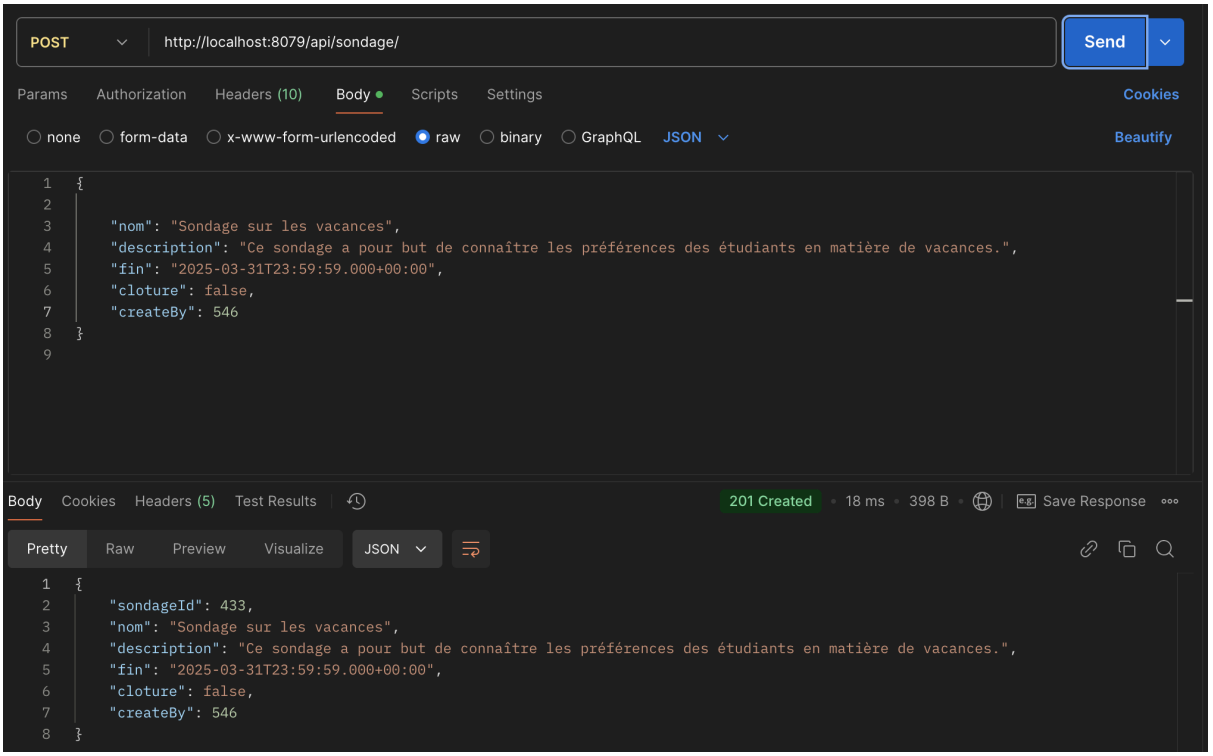
- **Corps de la réponse (JSON retourné)** :

```
{
 "participantId": 546,
 "nom": "toto",
 "prenom": "titi"
}
```

- **Explication :**
    - L'API a créé un nouveau participant avec l'**ID unique** `546`.
    - L'API renvoie les mêmes données que celles envoyées, avec l'ajout du `participantId`

B. Création d'un sondage

Dans ce test, une requête **POST** est envoyée à l'API pour **ajouter un sondage** à la base de données.



Détails de la requête :

- **Méthode HTTP** : `POST`
- **URL** : `http://localhost:8079/api/sondage/`
- **Headers** :

    Le contenu est envoyé en **JSON** (`Content-Type: application/json`).

- **Body** (Données envoyées) :

```
{
  "nom": "Sondage sur les vacances",
  "description": "Ce sondage a pour but de connaître les
   préférences des étudiants en matière de vacances.",
  "fin": "2025-03-31T23:59:59.000+00:00",
  "cloture": false,
  "createBy": 546
 }
```

- **Explication des champs** :
  - **nom** : Titre du sondage (**"Sondage sur les vacances"**).
  - **description** : Objectif du sondage.
  - **fin** : Date de clôture du sondage (**31 mars 2025 à 23:59 UTC**).
  - **cloture** : `false` (signifie que le sondage est encore actif).
  - **createBy** : Identifiant du participant ayant créé le sondage (`546`).

Résultat de la requête

☑ **Réponse obtenue :**

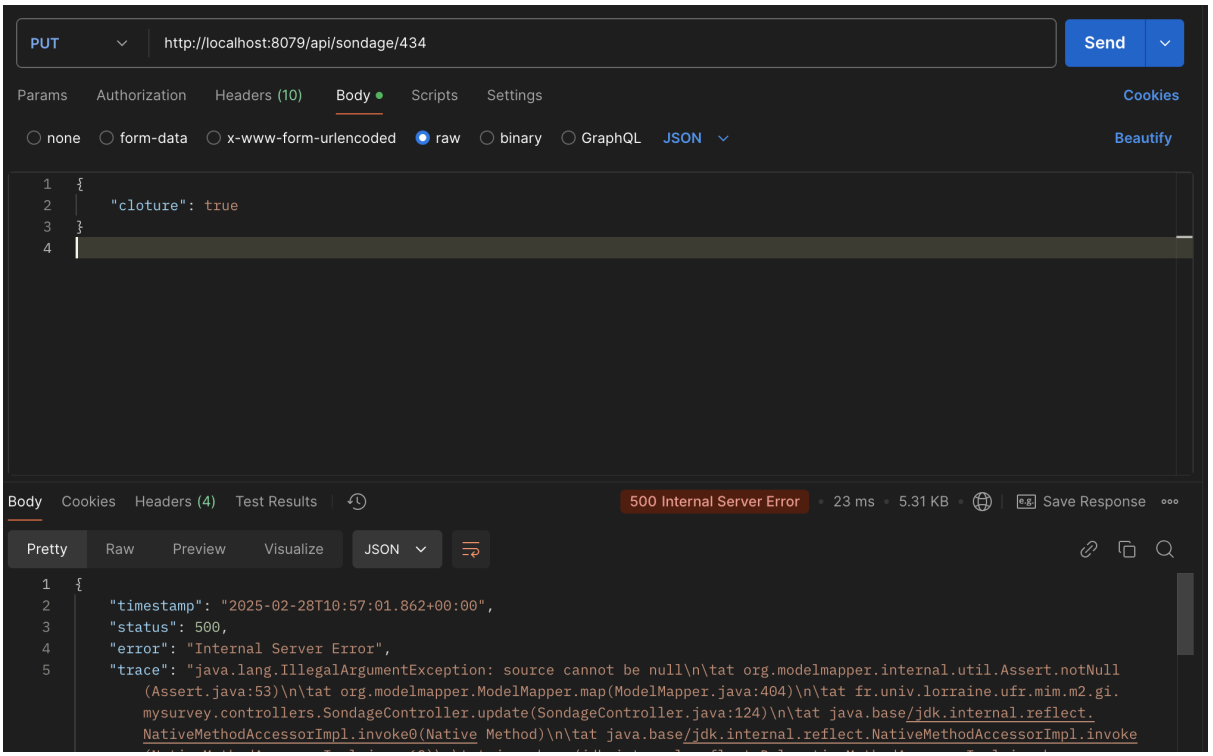
- **Statut HTTP** : `201 Created` (Indique que la ressource a été créée avec succès).
- **Corps de la réponse (JSON retourné)** :

```
{
  "sondageId": 433,
  "nom": "Sondage sur les vacances",
  "description": "Ce sondage a pour but de connaître les
préférences  des étudiants en matière de vacances.",
  "fin": "2025-03-31T23:59:59.000+00:00",
  "cloture": false,
  "createBy": 546
}
```

- **Explication :**
  - L'API a enregistré le sondage et lui a attribué un **ID unique (433)**.
  - L'API renvoie exactement les mêmes informations que celles envoyées.

C. Mise à jour d'un sondage

Dans ce test, une requête **PUT** est envoyée à l'API pour **mettre à jour** un sondage et modifier son état de clôture.



Détails de la requête :

- **Méthode HTTP** : `PUT`
- **URL** : `http://localhost:8079/api/sondage/434`
- **Headers** :
  Le contenu est envoyé en **JSON** (`Content-Type: application/json`).
- **Body** (Données envoyées) :

```
{
"cloture": true
}
```

☑ L'objectif est de **clôturer le sondage** (passer `cloture` à `true`).

Résultat de la requête

Réponse obtenue :

- **Statut HTTP** : `500 Internal Server Error` (Erreur interne du serveur).

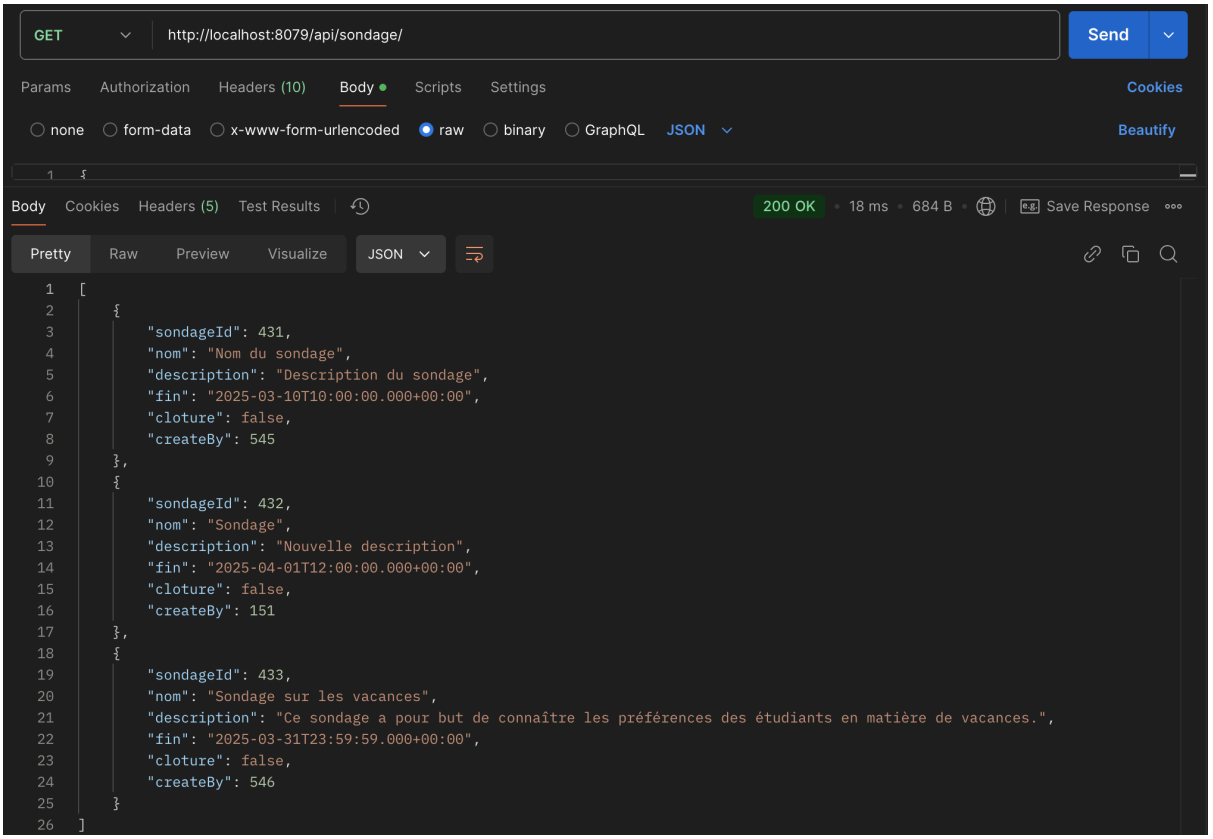
Message d'erreur retourné :

```
{
  "timestamp": "2025-02-28T10:57:01.862+00:00",
  "status": 500,
  "error": "Internal Server Error",
  "trace": "java.lang.IllegalArgumentException: source cannot    be
null
  at org.modelmapper.internal.util.Assert.notNull (Assert.java:53)
  at org.modelmapper.ModelMapper.map (ModelMapper.java:404)
  at mysurvey.controllers.SondageController.update
(SondageController.java:124)"
}
```

- **Explication de l'erreur :**
  - L'erreur **`java.lang.IllegalArgumentException: source cannot be null`** indique que l'**objet source** que l'API tente de convertir ou de mettre à jour est `null`.
  - La **méthode `map()` de ModelMapper** essaie de **mettre à jour un objet inexistant**.
  - L'erreur semble venir de la ligne `124` dans **SondageController.java**.

D. Récupération des sondages (GET /api/sondage/)

Dans ce test, une requête GET est envoyée à l'API pour récupérer la liste des sondages existants dans la base de données.



Détails de la requête :

- **Méthode HTTP** : `GET`
- **URL** : `http://localhost:8079/api/sondage/`
- **Headers** :Le contenu est demandé en **JSON** (`Accept: application/json`).

Résultat de la requête

☑ **Réponse obtenue :**

- **Statut HTTP** : `200 OK` (Indique que la requête a réussi et que les données ont été renvoyées).

Corps de la réponse (JSON retourné) :

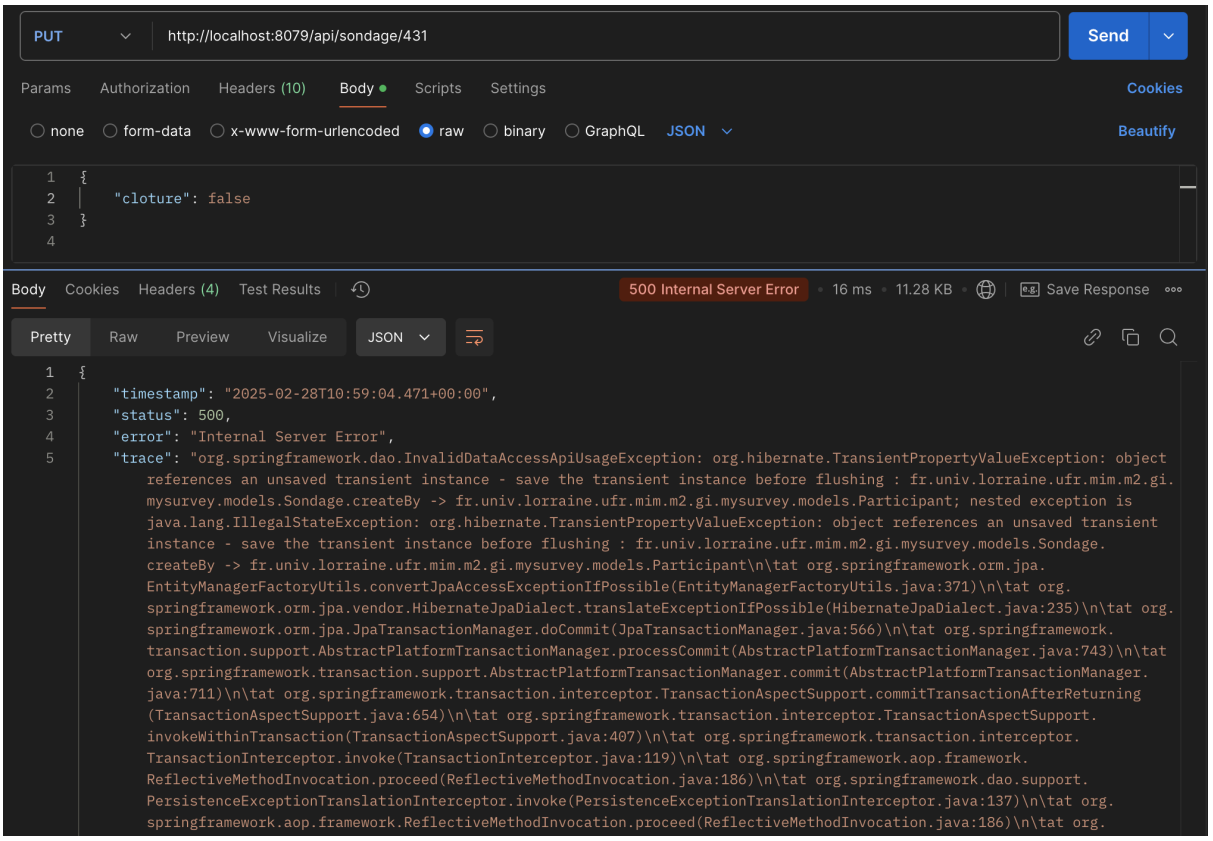

```
[
  {
    "sondageId": 431,
    "nom": "Nom du sondage",
    "description": "Description du sondage",
    "fin": "2025-03-10T00:00:00.000+00:00",
    "cloture": false,
```

```
    "createBy": 545
  },
  {
    "sondageId": 432,
    "nom": "Sondage",
    "description": "Nouvelle description",
    "fin": "2025-04-01T12:00:00.000+00:00",
    "cloture": false,
    "createBy": 151
  },
  {
    "sondageId": 433,
    "nom": "Sondage sur les vacances",
    "description": "Ce sondage a pour but de connaître les
préférences des étudiants en matière de vacances.",
    "fin": "2025-03-31T23:59:59.000+00:00",
    "cloture": false,
    "createBy": 546
  }
]
```


- **Explication des champs :**
  - **sondageId** : Identifiant unique du sondage.
  - **nom** : Titre du sondage.
  - **description** : Description détaillant l'objectif du sondage.
  - **fin** : Date limite pour voter.
  - **cloture** : Indique si le sondage est clôturé (`false` = ouvert, `true` = fermé).
  - **createBy** : Identifiant du créateur du sondage.

E. Clôturer un sondage

Dans ce test, une requête **PUT** est envoyée à l'API pour **clôturer un sondage**, c'est-à-dire empêcher les votes une fois que le sondage est fermé.



Détails de la requête

- **Méthode HTTP** : `PUT`
- **URL** : `http://localhost:8079/api/sondage/431`
- **Headers** :
  `Content-Type: application/json`
- **Body** (Données envoyées) :

```
{
"cloture": false
}
```

- L'objectif est de **s'assurer que le sondage reste ouvert** (`cloture: false`).

Résultat de la requête

☐ **Réponse obtenue :**

- **Statut HTTP** : `500 Internal Server Error`

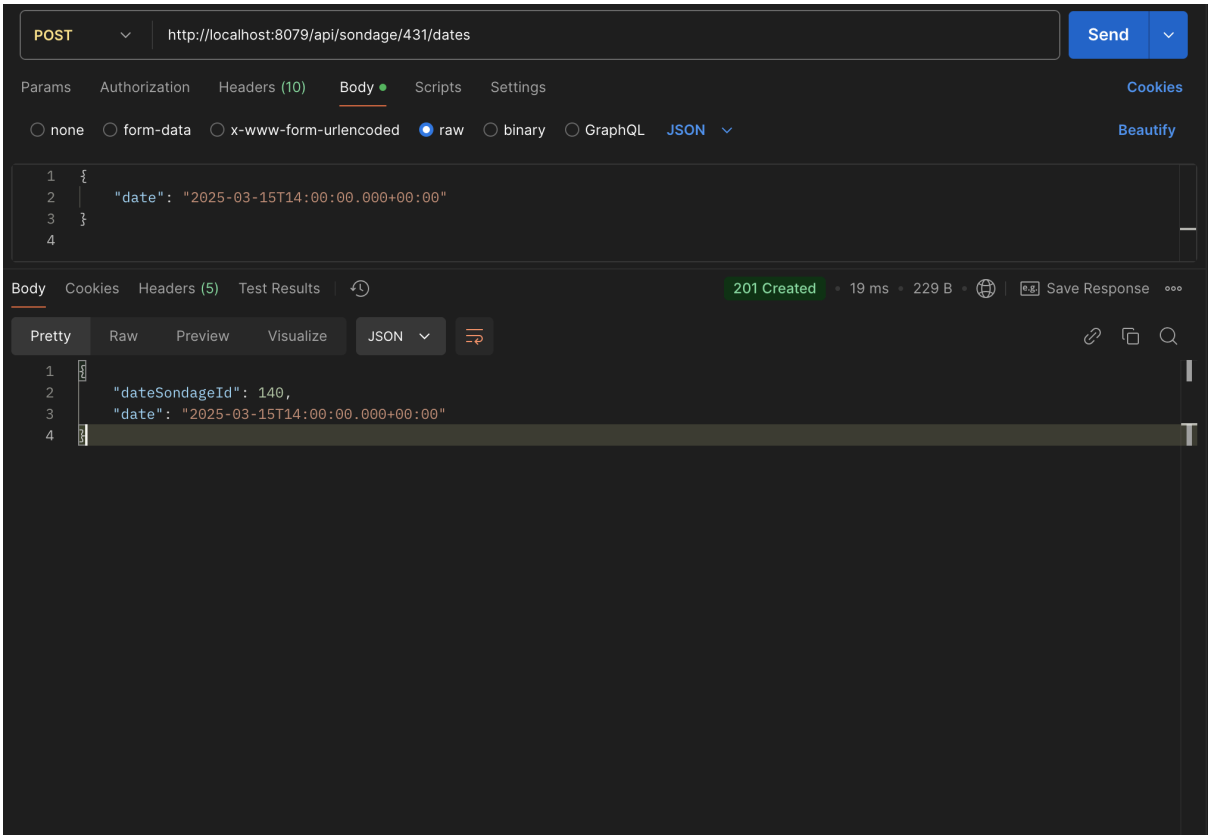
Message d’erreur retourné :

```
{
  "error": "Internal Server Error",
  "trace":
"org.springframework.dao.InvalidDataAccessApiUsageException:
  org.hibernate.TransientPropertyValueException:
  object references an unsaved transient instance -
  save the transient instance before flushing :
  fr.univ.lorraine.ufr.mim.m2.gi.mysurvey.models.Sondage.createBy
  -> fr.univ.lorraine.ufr.mim.m2.gi.mysurvey.models.Participant;"
}
```

- **Explication de l'erreur :**
  - L’erreur **`TransientPropertyValueException`** indique que **l’objet `Sondage` fait référence à un `Participant` qui n’a pas été enregistré en base de données**.
  - Plus précisément, l’attribut **`createBy`** (qui est censé contenir l’ID du créateur du sondage) **ne correspond pas à un participant existant en base**.
  - Hibernate empêche l’enregistrement du sondage car il tente de sauvegarder une relation (`createBy`) avec un objet inexistant en base.

F. Ajout d'une date à un sondage

Dans ce test, une requête **POST** est envoyée à l'API pour **ajouter une date** au sondage existant avec l'ID **431**.



Détails de la requête

- **Méthode HTTP** : `POST`
- **URL** : `http://localhost:8079/api/sondage/431/dates`
- **Headers** :
  `Content-Type: application/json`
- **Body** (Données envoyées) :

```
{

 "date": "2025-03-15T14:00:00.000+00:00"
}
```

☑ L'objectif est d'**ajouter une nouvelle date** (`15 mars 2025 à 14h00 UTC`) au sondage **431**.

Résultat de la requête

☑ **Réponse obtenue :**

- **Statut HTTP** : `201 Created` (Indique que la date a été ajoutée avec succès).

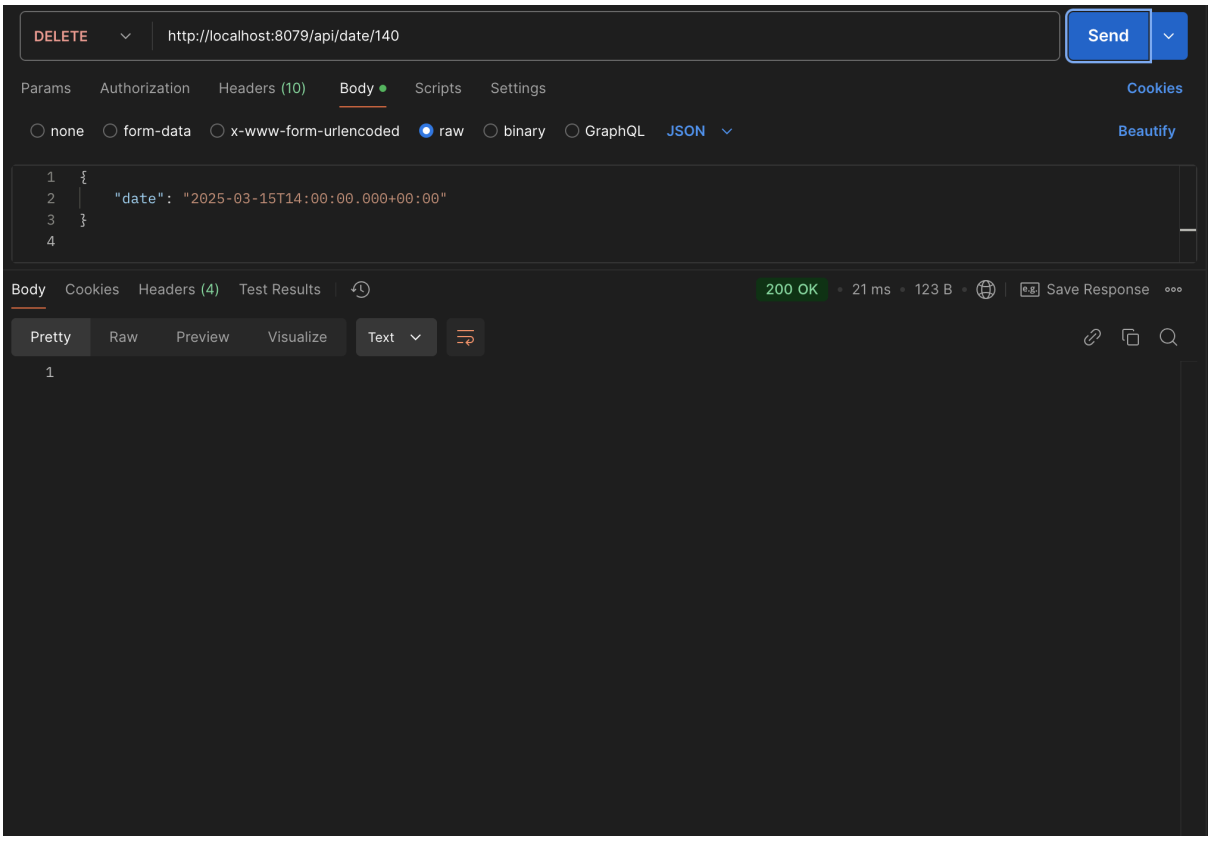
Corps de la réponse (JSON retourné) :

```
{
  "dateSondageId": 140,
  "date": "2025-03-15T14:00:00.000+00:00"
}
```

- **Explication :**
  - L'API a bien enregistré la nouvelle date et lui a attribué un **ID unique (`140`)**.
  - La date ajoutée est correctement renvoyée dans la réponse.

G. Suppression d'une date d'un sondage

Dans ce test, une requête **DELETE** est envoyée à l'API pour **supprimer une date spécifique** du sondage.



Détails de la requête

- **Méthode HTTP** : `DELETE`
- **URL** : `http://localhost:8079/api/date/140`
- **Headers** :
  `Content-Type: application/json`
- **Body** (Données envoyées) :

```
{
"date": "2025-03-15T14:00:00.000+00:00"
}
```

☑ L'objectif est de **supprimer la date** `2025-03-15T14:00:00.000+00:00`, qui avait été précédemment ajoutée au sondage.

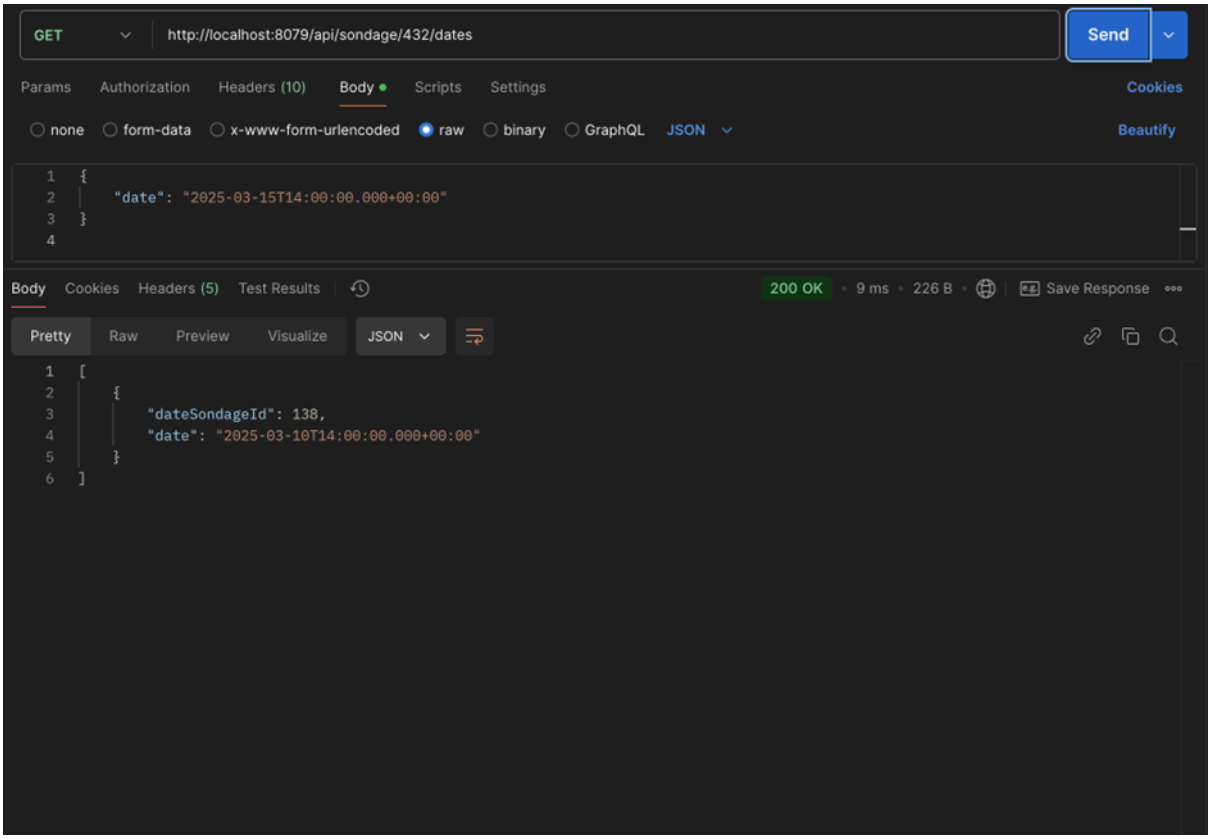
Résultat de la requête

☑ **Réponse obtenue :**

- **Statut HTTP** : `200 OK` (Indique que l'opération s'est bien déroulée).
- **Corps de la réponse** : `1`
  Explication :
  - L'API a bien traité la suppression.
  - Le retour `1` peut indiquer **le nombre d'enregistrements supprimés** en base de données.

H. Lister les dates d'un sondage;

Dans ce test, une requête **GET** est envoyée à l'API pour **récupérer la liste des dates associées** au sondage avec l'ID **432**.



Détails de la requête

- **Méthode HTTP** : `GET`
- **URL** : `http://localhost:8079/api/sondage/432/dates`
- **Headers** :
  - `Accept: application/json`

Résultat de la requête

☑ **Réponse obtenue :**

- **Statut HTTP** : `200 OK` (Indique que la requête a réussi et que les données ont été renvoyées).

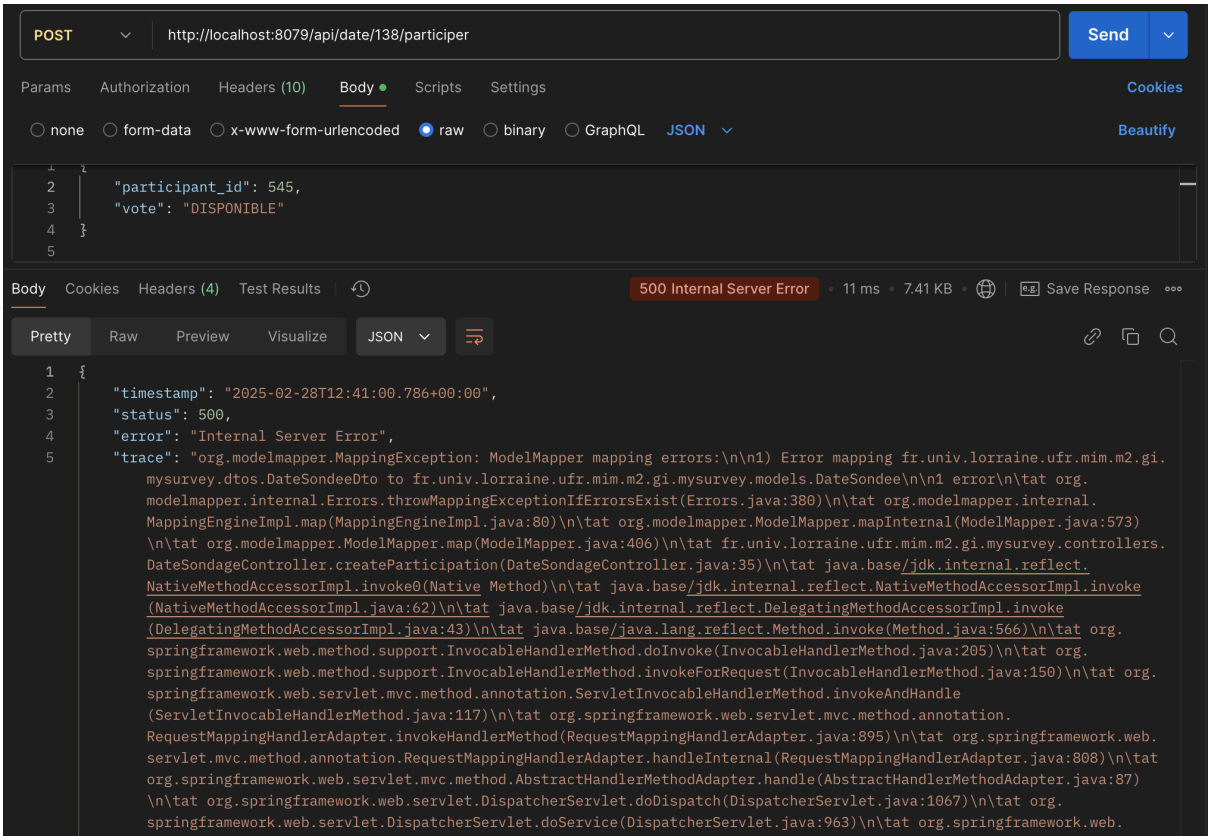
Corps de la réponse (JSON retourné) :

```
[
  {
    "dateSondageId": 138,
    "date": "2025-03-10T14:00:00.000+00:00"
  }
]
```

- **Explication des champs :**
  - **dateSondageId** : Identifiant unique de la date enregistrée pour le sondage.
  - **date** : Date enregistrée pour le sondage (ici, **10 mars 2025 à 14h00 UTC**).

I. Participation à un sondage

Dans ce test, une requête **POST** est envoyée à l'API pour **permettre à un participant de voter pour une date d'un sondage**.



Détails de la requête

- **Méthode HTTP** : `POST`
- **URL** : `http://localhost:8079/api/date/138/participer`
- **Headers** :

  `Content-Type: application/json`

- **Body** (Données envoyées) :

```
{
  "participant_id": 545,
  "vote": "DISPONIBLE"
}
```

☑ L'objectif est de **faire voter le participant `545`** pour la date `138` avec le choix **"DISPONIBLE"**.

Résultat de la requête

☐ **Réponse obtenue :**

- **Statut HTTP** : `500 Internal Server Error` (Erreur interne du serveur).

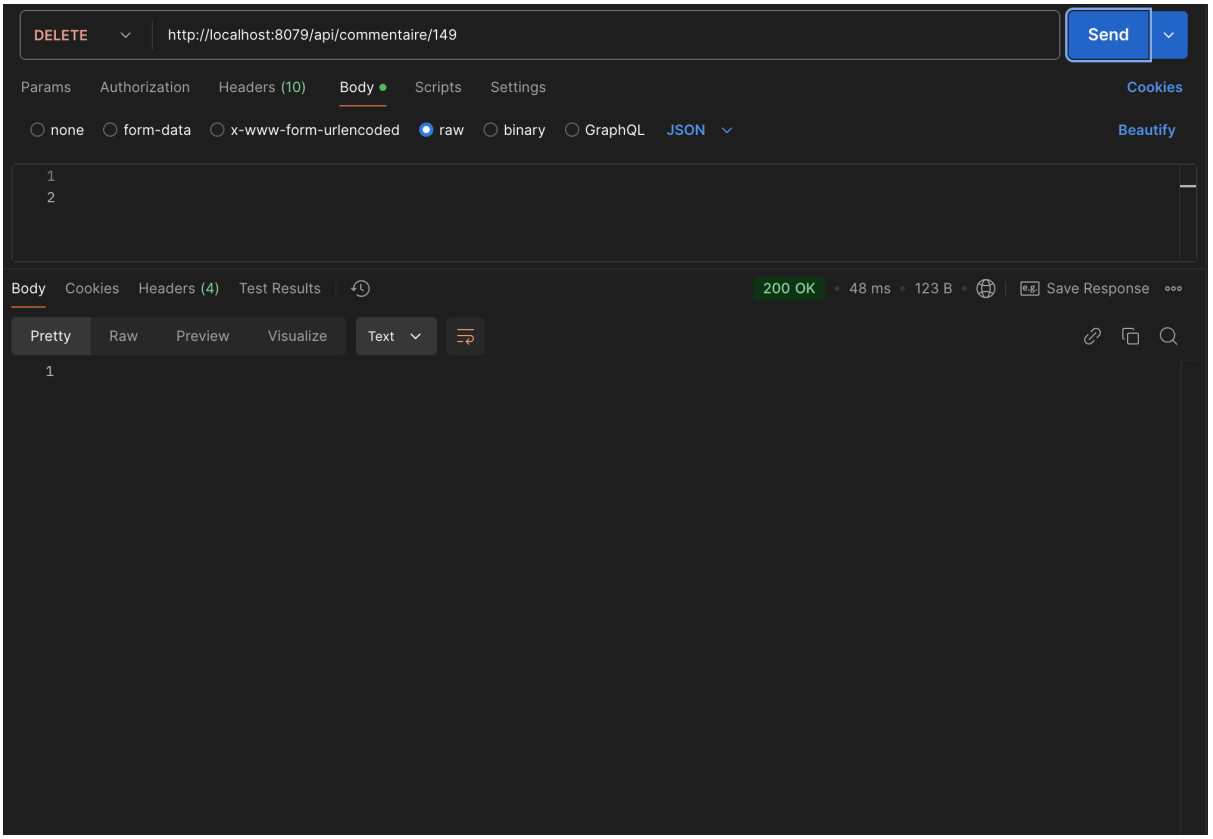
Message d'erreur retourné :

```
{
  "error": "Internal Server Error",
  "trace": "org.modelmapper.MappingException: ModelMapper mapping
errors:
  Error mapping
fr.univ.lorraine.ufr.mim.m2.gi.mysurvey.dtos.DateSondageDto
  to fr.univ.lorraine.ufr.mim.m2.gi.mysurvey.models.DateSondage"
}
```

- **Explication de l'erreur :**
  - L'erreur **`MappingException`** indique un problème de **conversion entre un DTO (`DateSondageDto`) et une entité (`DateSondage`)**.
  - L'API essaie d'exécuter un **mapping automatique** avec **ModelMapper**, mais un champ semble **mal configuré** ou **manquant**.

J. Suppression d’un commentaire

Dans ce test, une requête **DELETE** est envoyée à l’API pour **supprimer un commentaire** identifié par l’ID **149**.



Détails de la requête

- **Méthode HTTP** : `DELETE`
- **URL** : `http://localhost:8079/api/commentaire/149`
- **Headers** :
  - `Content-Type: application/json`

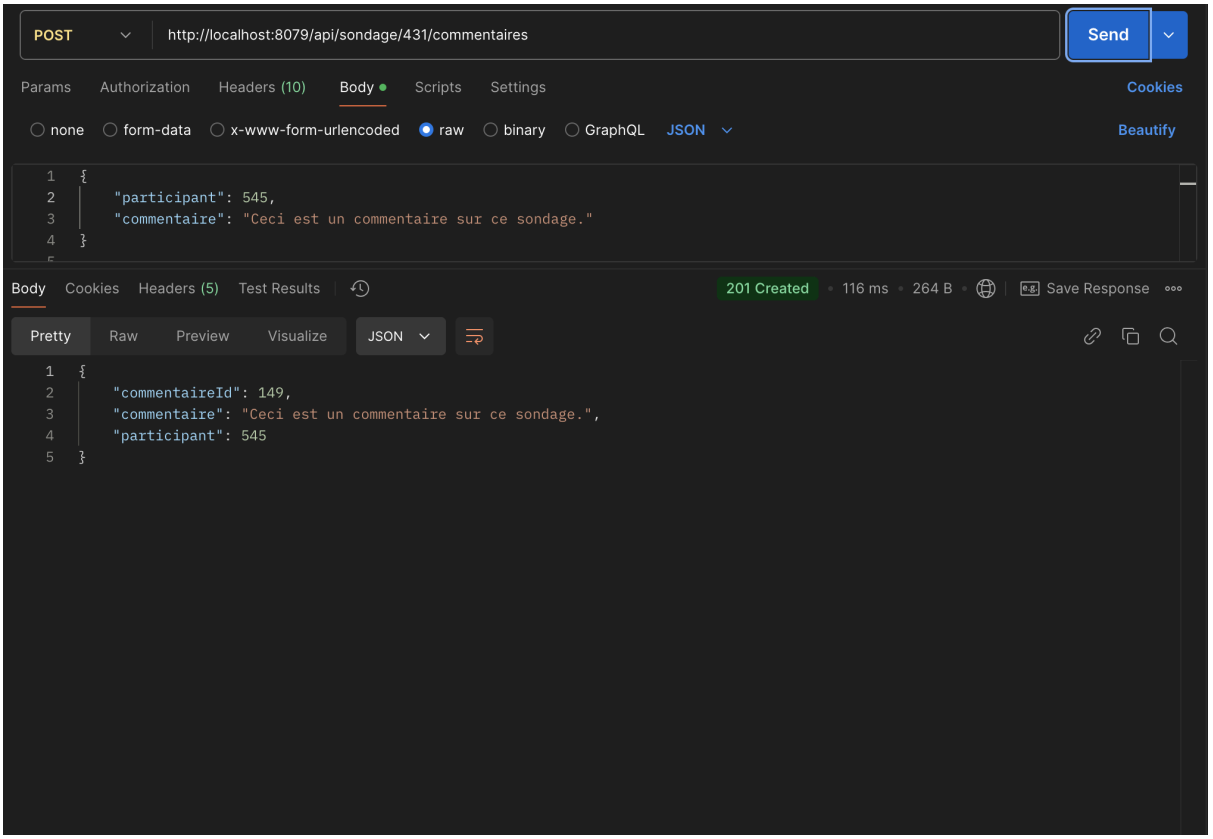
Résultat de la requête

☑ **Réponse obtenue :**

- **Statut HTTP** : `200 OK` (Indique que la suppression a été effectuée avec succès).
- **Corps de la réponse** : `1`
  Explication :
  - L’API a bien traité la suppression.
  - Le retour `1` peut signifier que **1 enregistrement a été supprimé en base de données**.

K. Ajout d’un commentaire à un sondage

Dans ce test, une requête **POST** est envoyée à l’API pour **ajouter un commentaire** à un sondage existant avec l’ID **431**.



Détails de la requête

- **Méthode HTTP** : `POST`
- **URL** : `http://localhost:8079/api/sondage/431/commentaires`
- **Headers** :
  ```
  Content-Type: application/json
  ```
- **Body** (Données envoyées) :
  ```
  {
    "participant": 545,
    "commentaire": "Ceci est un commentaire sur ce sondage."
  }
  ```
  - **participant** : `545` (L’ID du participant qui publie le commentaire).
  - **commentaire** : `"Ceci est un commentaire sur ce sondage."` (Le texte du commentaire).

Résultat de la requête

☑ **Réponse obtenue :**

- **Statut HTTP** : `201 Created` (Indique que le commentaire a été ajouté avec succès).

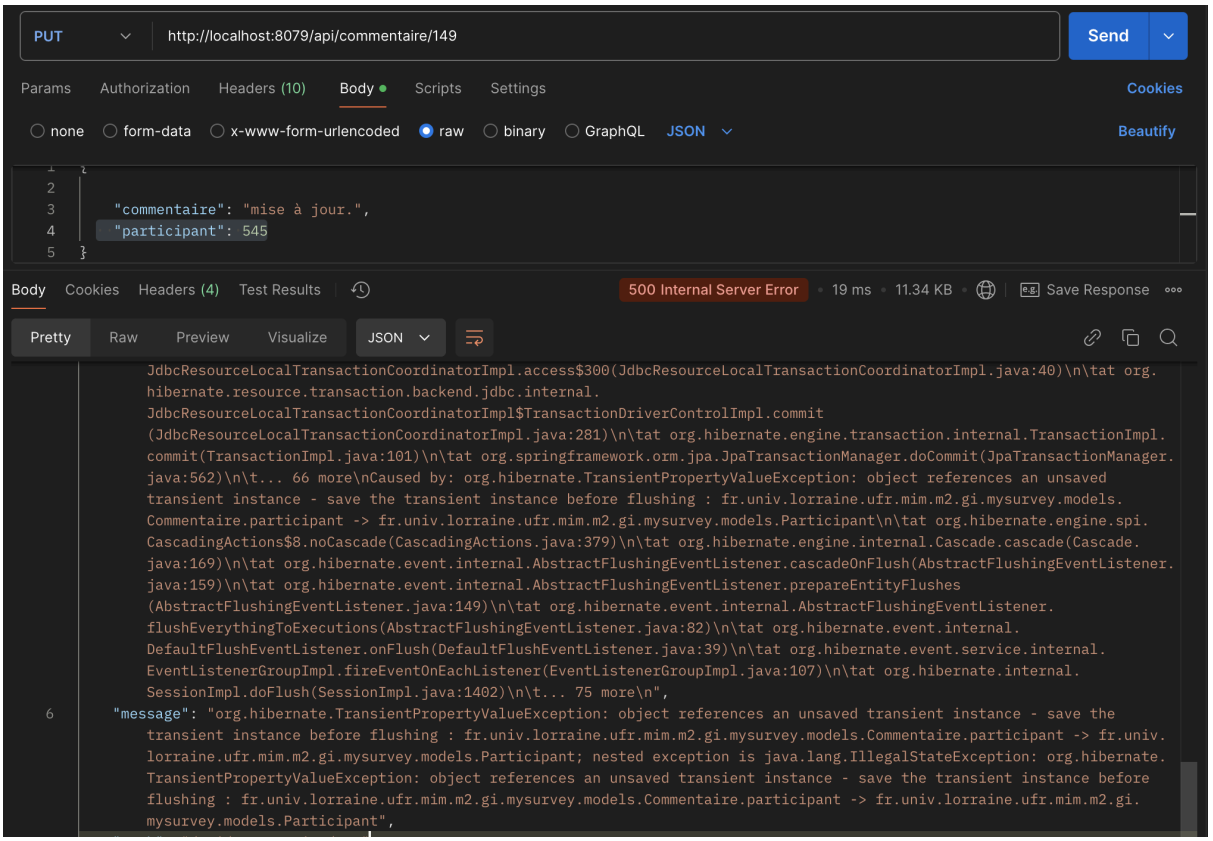
  Corps de la réponse (JSON retourné) :

  ```
  {
    "commentaireId": 149,
    "commentaire": "Ceci est un commentaire sur ce sondage.",
    "participant": 545
  }
  ```

- **Explication :**
  - L'API a bien enregistré le commentaire et lui a attribué un **ID unique (149)**.
  - Le commentaire est bien lié au participant 545 et au sondage 431.

L. Modification d'un commentaire

Dans ce test, une requête **PUT** est envoyée à l'API pour **modifier un commentaire existant** identifié par l'ID **149**.



Détails de la requête

- **Méthode HTTP** : `PUT`
- **URL** : `http://localhost:8079/api/commentaire/149`
- **Headers** : `Content-Type: application/json`

- **Body** (Données envoyées) :

```
{
  "commentaire": "mise à jour.",
  "participant": 545
}
```

  - **commentaire** : Mise à jour du texte du commentaire.
  - **participant** : `545` (L'ID du participant qui a posté le commentaire).

Résultat de la requête

☐ **Réponse obtenue :**

- **Statut HTTP** : `500 Internal Server Error` (Erreur interne du serveur).

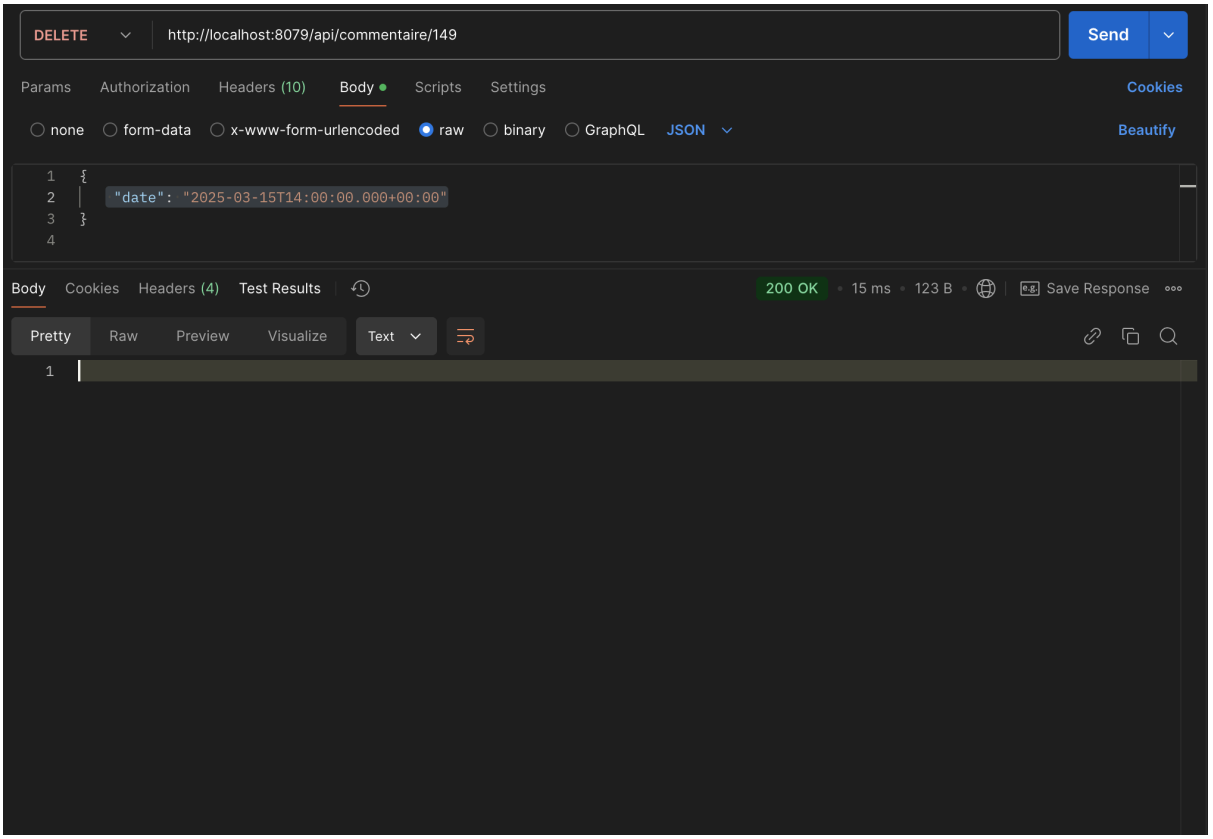
  Message d'erreur retourné :

```
{

  "message": "org.hibernate.TransientPropertyValueException:

  object references an unsaved transient instance - save the
transient instance before flushing :


fr.univ.lorraine.ufr.mim.m2.gi.mysurvey.models.Commentaire.par
ticipant ->
fr.univ.lorraine.ufr.mim.m2.gi.mysurvey.models.Participant"

}
```

- **Explication de l'erreur :**
  - L'erreur **`TransientPropertyValueException`** signifie que **l'API tente d'enregistrer un commentaire qui référence un participant qui n'est pas reconnu comme une entité persistante en base de données**.
  - Hibernate empêche la mise à jour car **le participant (`participant: 545`) n'est pas attaché à la session Hibernate**, ce qui cause une incohérence dans la gestion de la relation entre `Commentaire` et `Participant`.

M. Suppression d'un commentaire

Dans ce test, une requête **DELETE** est envoyée à l'API pour **supprimer un commentaire** identifié par l'ID **149**.



Détails de la requête

- **Méthode HTTP** : `DELETE`
- **URL** : `http://localhost:8079/api/commentaire/149`
- **Headers** : `Content-Type: application/json`
- **Body** (Données envoyées) :

```
{
 "date": "2025-03-15T14:00:00.000+00:00"
}
```

- **Remarque :**
  - Normalement, **aucun body n'est requis pour une requête DELETE** sauf si l'API l'exige explicitement.

Résultat de la requête

☑ **Réponse obtenue :**

- **Statut HTTP** : `200 OK` (Indique que la suppression a été effectuée avec succès).
- **Corps de la réponse** : **vide**
  Explication :
  - L'API a bien supprimé le commentaire.
  - Le **body vide** signifie que la suppression **n'a pas besoin de renvoyer de données**.

7.3 Precision

Dans Postman, les tests des requêtes ont mis en évidence un problème récurrent concernant toutes l**es requêtes PUT (mise à jour)**. En effet, chaque tentative de modification d'une ressource via PUT entraîne une erreur 500, non pas en raison d'une mauvaise utilisation de la méthode PUT dans Postman, mais plutôt à cause **d'un dysfonctionnement dans le code de l'API REST.** Ce problème affecte toutes les opérations de mise à jour, qu'il s'agisse de la modification d'un sondage, d'un commentaire ou d'un vote

8. Gestion du serveur et résolution de l'erreur 502

8.1 Accès au serveur

L'API REST du projet est déployée et accessible via l'URL suivante :
https://prod-logiciel.onrender.com/api/...
Toutes les requêtes doivent être envoyées à cette adresse pour interagir avec le backend.

8.2 Gestion des erreurs et redémarrage du serveur

Dans certains cas, l'API peut devenir inaccessible et renvoyer une **erreur 502 (Bad Gateway)**. Cela indique généralement que le serveur ne répond plus ou rencontre un problème temporaire.

a. Redémarrer l'API

Si une **erreur 502** est détectée, il est recommandé de **redémarrer l'API** en exécutant la commande suivante dans un terminal :

```
curl -X POST
https://api.render.com/v1/services/srv-cv0aoe0gph6c73c9k130/restart
\
     -H "Authorization: Bearer rnd_hwnqKnnmRAAjaW8bwcxXHA7yt9d4"
```

Explication :

- Cette commande envoie une requête POST à la plateforme **Render** pour redémarrer le service API.
- L'**Authorization Bearer Token** assure que l'opération est bien autorisée.

b. Redéployer l'API si l'erreur persiste

Si l'erreur **502** persiste après le redémarrage, cela signifie que l'API a besoin d'un **redéploiement complet**.
Dans ce cas, il faut exécuter la commande suivante :

```
curl -X POST
https://api.render.com/v1/services/srv-cv0aoe0gph6c73c9k130/deploys
\
     -H "Authorization: Bearer rnd_hwnqKnnmRAAjaW8bwcxXHA7yt9d4"
```

Explication :

- Cette commande déclenche un **nouveau déploiement de l'API** sur Render.
- **Temps estimé :** L'opération peut prendre **entre 7 et 11 minutes** pour être complètement effective.