HELHa
Haute École Louvain en Hainaut

API Contract

Bachelier en Informatique

Option Développement d'Applications

Bloc 3

Troc'App - API Contract

ASSADI Doha

DOGAN Hayriye

LIEBECQ Roxane

SOEGITO Jade

TAS Sudem

Année académique 2024 - 2025

ÉCONOMIQUE

TABLE DES MATIERES

SERVICE BACKEND : ADDRESS & GEOCODING

Une API RESTful permettant la gestion des adresses (CRUD) et leur géocodage via l'API Nominatim.

1. BASE URL

/addresses

2. ENDPOINTS

1. AJOUTER UNE ADRESSE

REQUÊTE:

Method: POST

URL: /addresses

Headers:

Content-Type: application/json

Body:

```
{
 "street": "string",
 "number": "string",
 "city": "string",
 "zipCode": "string"
}
```

Description : Géocodage de l'adresse (lat, lon) via Nominatim, suivi de l'enregistrement dans la base de données.

RÉPONSE:

- **Status Code**:
- 201 Created: Adresse ajoutée avec succès.
- 500 Internal Server Error: Une erreur s'est produite.

- **Body (201)**:

```
{
 "id": "int",
 "street": "string",
 "number": "string",
 "city": "string",
 "zipCode": "string",
 "latitude": "double",
```

```
 "longitude": "double"
}
```

2. RÉCUPÉRER UNE ADRESSE PAR ID

REQUÊTE:

Method: GET

URL: /addresses/{id}

Description : Récupère les détails d'une adresse via son ID.

RÉPONSE:

- **Status Code**:
  - 200 OK: Adresse trouvée.
  - 404 Not Found: Adresse non trouvée.
  - 500 Internal Server Error: Une erreur s'est produite.
- **Body (200)**:

```
{
 "id": "int",
 "street": "string",
 "number": "string",
 "city": "string",
 "zipCode": "string",
 "latitude": "double",
 "longitude": "double"
}
```

3. RÉCUPÉRER TOUTES LES ADRESSES

REQUÊTE:

- **Method:** GET
- **URL:** /addresses
- **Headers**:
  - Aucun requis.
- **Description:** Récupère une liste complète des adresses dans le système.

RÉPONSE:

- **Status Code**:
  - 200 OK: Liste retournée avec succès.
  - 500 Internal Server Error: Une erreur s'est produite.
- **Body (200)**:

```
[
 {
  "id": "int",
  "street": "string",
```

```
  "number": "string",
  "city": "string",
  "zipCode": "string",
  "latitude": "double",
  "longitude": "double"
 },
 {
  "id": "int",
  "street": "string",
  "number": "string",
  "city": "string",
  "zipCode": "string",
  "latitude": "double",
  "longitude": "double"
 }
]
```


4. SUPPRIMER UNE ADRESSE

REQUÊTE:

- **Method:** DELETE
- **URL:** /addresses/{id}
- **Headers:**
  - Aucun requis.
- **Description:** Supprime une adresse via son ID.

RÉPONSE:

- **Status Code:**
  - 204 No Content: Suppression réussie.
  - 404 Not Found: Adresse non trouvée.
  - 500 Internal Server Error: Une erreur s'est produite.

5. METTRE À JOUR UNE ADRESSE

REQUÊTE:

- **Method:** PUT
- **URL:** /addresses
- **Headers:**
  - Content-Type: application/json
- **Body:**


```
{
 "id": "int",
 "street": "string",
 "number": "string",
 "city": "string",
 "zipCode": "string"
}
```

- **Description**: Géocodage de l'adresse mise à jour (lat, lon) via Nominatim, suivi de sa sauvegarde dans la base de données.

RÉPONSE:

- **Status Code**:
  - 200 OK: Mise à jour réussie.
  - 500 Internal Server Error: Une erreur s'est produite.
- **Body (200)**:

```
{
 "id": "int",
 "street": "string",
 "number": "string",
 "city": "string",
 "zipCode": "string",
 "latitude": "double",
 "longitude": "double"
}
```

3. DÉPENDANCES TECHNIQUES

NOMINATIM API:

- **URL d'accès**: Configurable via la propriété nominatim.api.url.
- **Requête**:

  GET /?q={formattedAddress}&format=json

  - formattedAddress: Adresse formatée sous forme de chaîne.
- **Réponse**:

```
[
 {
  "lat": "double",
  "lon": "double"
 }
]
```

- **Codes de réponse**:

- 200 OK: Requête réussie.
- 4xx/5xx: Erreur avec description associée.

---

4. GESTION DES ERREURS

- Validation des champs obligatoires pour les requêtes entrantes.
- Gestion des exceptions internes avec des messages significatifs.
- Utilisation des codes HTTP standard pour refléter l'état de la requête.

SERVICE FRONTEND : ADDRESS

5. AJOUTER UNE ADRESSE (POST)

- **Méthode HTTP** : POST
- **URL** : /addresses
- **Description** : Envoie une requête au backend pour ajouter une adresse. Le frontend envoie l'adresse au backend, qui se charge de la géocodification (avec Nominatim) et de l'ajout dans la base de données.

EXEMPLE DE REQUÊTE :

```
// Angular Service (AddressService)

addAddress(address: Address): Observable<Address> {

 return this.httpClient.post<Address>(` ${this.baseUrl}` , address, { headers: this.getAuthHeaders()
}).pipe(

  catchError((error) => {

   console.error('Error adding address:', error);

   return throwError(() => new Error('Error adding address.'));

  })

 );

}
```

CORPS DE LA REQUÊTE :

```
{

 "street": "string",

 "number": "string",

 "city": "string",

 "zipCode": "string"

}
```

6. RÉCUPÉRER UNE ADRESSE PAR ID (GET)

- **Méthode HTTP** : GET
- **URL** : /addresses/{id}
- **Description** : Envoie une requête au backend pour récupérer une adresse spécifique à partir de son ID.

EXEMPLE DE REQUÊTE :

```
// Angular Service (AddressService)
```

```
getAddressById(id: number): Observable<Address> {
 return this.httpClient.get<Address>(`${this.baseUrl}/${id}`, { headers: this.getAuthHeaders() }).pipe(
  catchError((error) => {
   console.error(`Error fetching address with ID ${id}:`, error);
   return throwError(() => new Error(`Error fetching address with ID ${id}.`));
  })
 );
}
```

7. RÉCUPÉRER TOUTES LES ADRESSES (GET)

- **Méthode HTTP** : GET
- **URL** : /addresses
- **Description** : Envoie une requête au backend pour récupérer la liste complète des adresses.

EXEMPLE DE REQUÊTE :

```
// Angular Service (AddressService)
getAllAddresses(): Observable<Address[]> {
 return this.httpClient.get<Address[]>(`${this.baseUrl}`, { headers: this.getAuthHeaders() }).pipe(
  catchError((error) => {
   console.error('Error fetching all addresses:', error);
   return throwError(() => new Error('Error fetching all addresses.'));
  })
 );
}
```

8. SUPPRIMER UNE ADRESSE (DELETE)

- **Méthode HTTP** : DELETE
- **URL** : /addresses/{id}
- **Description** : Envoie une requête au backend pour supprimer une adresse spécifique par son ID.

EXEMPLE DE REQUÊTE :

```
// Angular Service (AddressService)
```

```
deleteAddress(id: number): Observable<void> {
 return this.httpClient.delete<void>(`${this.baseUrl}/${id}`, { headers: this.getAuthHeaders() }).pipe(
  catchError((error) => {
   console.error(`Error deleting address with ID ${id}:`, error);
   return throwError(() => new Error(`Error deleting address with ID ${id}.`));
  })
 );
}
```

9. METTRE À JOUR UNE ADRESSE (PUT)

- **Méthode HTTP** : PUT
- **URL** : /addresses
- **Description** : Envoie une requête pour mettre à jour une adresse existante. Le frontend envoie les nouvelles informations d'adresse au backend, qui effectue également la géocodification via Nominatim.

EXEMPLE DE REQUÊTE :

```
// Angular Service (AddressService)
updateAddress(address: Address): Observable<Address> {
 return this.httpClient.put<Address>(`${this.baseUrl}`, address, { headers: this.getAuthHeaders() }).pipe(
  catchError((error) => {
   console.error('Error updating address:', error);
   return throwError(() => new Error('Error updating address.'));
  })
 );
}
```

CORPS DE LA REQUÊTE :

```
{
 "id": "int",
 "street": "string",
 "number": "string",
 "city": "string",
```

"zipCode": "string"

}

SERVICE BACKEND: GDPR REQUEST

Une API RESTful pour gérer les demandes GDPR des utilisateurs, telles que la création de demandes, la consultation, le traitement, et la désactivation des utilisateurs.

Base URL :/gdpr-requests

Endpoints

1. Ajouter une demande GDPR

- Méthode: POST
- URL: /gdpr-requests
- Description: Crée une nouvelle demande GDPR pour un utilisateur spécifique.

REQUÊTE:

{

"requesttype": "string",

"user": {

"id": "int",

"firstName": "string",

"lastName": "string",

"email": "string"

},

"consent": true,

"justification": "string"

}

RÉPONSE:

CODE DE STATUT:

- 201 Created: La demande a été créée avec succès.
- 500 Internal Server Error: Une erreur s'est produite.

Body (201){

"id": "int",

"requesttype": "string",

"user": {

```
  "id": "int",
  "firstName": "string",
  "lastName": "string",
  "email": "string"
 },
 "consent": true,
 "justification": "string",
 "status": "Pending"
}
```


RÉCUPÉRER UNE DEMANDE GDPR PAR ID

- Méthode: GET
- URL: /gdpr-requests/{id}
- Description: Récupère une demande GDPR par son ID.

Réponse:

Code de statut:

200 OK: Demande trouvée.

404 Not Found: Demande non trouvée.

Body (200)


```
{
 "id": "int",
 "requesttype": "string",
 "user": {
  "id": "int",
  "firstName": "string",
  "lastName": "string",
  "email": "string"
 },
 "consent": true,
 "justification": "string",
 "status": "Pending"
```

}

RÉCUPÉRER TOUTES LES DEMANDES GDPR

- Méthode: GET
- URL: /gdpr-requests
- Description: Récupère une liste de toutes les demandes GDPR.

Réponse:

Code de statut:

200 OK: Liste retournée avec succès.

500 Internal Server Error: Une erreur s'est produite.

Body (200):[


```
 {
   "id": "int",
   "requesttype": "string",
   "user": {
     "id": "int",
     "firstName": "string",
     "lastName": "string",
     "email": "string"
   },
   "consent": true,
   "justification": "string",
   "status": "Pending"
 }
```


]

TRAITER UNE DEMANDE GDPR

- Méthode: POST
- URL: /gdpr-requests/{id}/process
- Description: Marque une demande comme traitée.

Réponse:

Code de statut:

200 OK: Demande traitée avec succès.

404 Not Found: Demande non trouvée.

Body (200):


```
{
 "id": "int",
 "requesttype": "string",
 "user": {
  "id": "int",
  "firstName": "string",
  "lastName": "string",
  "email": "string"
 },
 "consent": true,
 "justification": "string",
 "status": "Processed",
 "response": "string"
}
```


DÉSACTIVER UN UTILISATEUR SUITE À UNE DEMANDE GDPR

- Méthode: POST
- URL: /gdpr-requests/{id}/deactivate-user
- Description: Désactive un utilisateur en fonction de la demande GDPR traitée.

Réponse:

Code de statut:

200 OK: Utilisateur désactivé avec succès.

404 Not Found: Utilisateur non trouvé.

Dépendances techniques

UserRepository: Permet d'interagir avec la base de données pour effectuer des opérations CRUD sur l'entité User.

GdprRequestRepository: Gère les opérations de base de données concernant les demandes GDPR.

EmailService: Service de notification par email, utilisé pour envoyer des confirmations ou notifications après le traitement des demandes.

UserService: Fournit des opérations liées à la gestion des utilisateurs, comme la désactivation ou la modification de leurs informations.

Gestion des erreurs

Validation des champs: S'assurer que tous les champs nécessaires sont fournis et valides dans chaque requête.

Gestion des exceptions: Utilisation de messages d'erreur clairs pour les erreurs internes ou de validation.

Codes de réponse HTTP: L'API renvoie des codes de statut HTTP standards pour indiquer l'état de la requête.

Service Frontend : GDPR Request

AJOUTER UNE DEMANDE GDPR (POST)

- Méthode HTTP: POST
- URL: /gdpr-requests
- Description: Envoie une requête au backend pour créer une demande GDPR.

Exemple de requête:

```
addGdprRequest(gdprRequest: GdprRequest): Observable<GdprRequest> {

 return this.httpClient.post<GdprRequest>(` ${this.baseUrl}` , gdprRequest, { headers: this.getAuthHeaders() }).pipe(

  catchError((error) => {

   console.error('Error adding GDPR request:', error);

   return throwError(() => new Error('Error adding GDPR request.'));

  })

 );

}
```

Corps de la requête:

```
{

 "requesttype": "string",

 "user": {

  "id": "int",
```

```
  "firstName": "string",
  "lastName": "string",
  "email": "string"
 },
 "consent": true,
 "justification": "string"
}
```

RÉCUPÉRER UNE DEMANDE GDPR PAR ID (GET)

Méthode HTTP: GET

URL: /gdpr-requests/{id}

Exemple de requête:

```
getGdprRequestById(id: number): Observable<GdprRequest> {
 return this.httpClient.get<GdprRequest>(`${this.baseUrl}/${id}`, { headers: this.getAuthHeaders() }).pipe(
  catchError((error) => {
   console.error(`Error fetching GDPR request with ID ${id}:`, error);
   return throwError(() => new Error(`Error fetching GDPR request with ID ${id}.`));
  })
 );
}
```

RÉCUPÉRER TOUTES LES DEMANDES GDPR (GET)

- Méthode HTTP: GET
- URL: /gdpr-requests
- Exemple de requête:

```
getAllGdprRequests(): Observable<GdprRequest[]> {
 return this.httpClient.get<GdprRequest[]>(`${this.baseUrl}`, { headers: this.getAuthHeaders() }).pipe(
  catchError((error) => {
   console.error('Error fetching all GDPR requests:', error);
   return throwError(() => new Error('Error fetching all GDPR requests.'));
```

```
  })
 );
}
```

TRAITER UNE DEMANDE GDPR (POST)

- Méthode HTTP: POST
- URL: /gdpr-requests/{id}/process
- Exemple de requête:

```
processGdprRequest(id: number, responseMessage: string): Observable<GdprRequest> {
 return this.httpClient.post<GdprRequest>(`${this.baseUrl}/${id}/process`, { response: responseMessage }, { headers: this.getAuthHeaders() }).pipe(
  catchError((error) => {
   console.error(`Error processing GDPR request with ID ${id}:`, error);
   return throwError(() => new Error(`Error processing GDPR request with ID ${id}.`));
  })
 );
}
```

DÉSACTIVER UN UTILISATEUR SUITE À UNE DEMANDE GDPR (POST)

- Méthode HTTP: POST
- URL: /gdpr-requests/{id}/deactivate-user
- Exemple de requête:

```
deactivateUser(id: number): Observable<void> {
 return this.httpClient.post<void>(`${this.baseUrl}/${id}/deactivate-user`, {}, { headers: this.getAuthHeaders() }).pipe(
  catchError((error) => {
   console.error(`Error deactivating user with GDPR request ID ${id}:`, error);
   return throwError(() => new Error(`Error deactivating user with GDPR request ID ${id}.`));
  })
 );
}
```

SERVICE BACKEND : CATEGORY MANAGEMENT

Une API RESTful permettant la gestion des catégories (CRUD) pour une application.

1. BASE URL

/categories

2. ENDPOINTS

AJOUTER UNE CATÉGORIE

- **Méthode** : POST
- **URL** : /categories
- **Headers** :
  Content-Type: application/json
- **Body** :

{

"name": "string"

}

- **Description** : Crée une nouvelle catégorie dans la base de données.

RÉPONSE :

  - **Code de statut** :
    201 Created : Catégorie créée avec succès.
    400 Bad Request : Paramètres incorrects.
  - **Body (201)** :

{

"id": "int",

"name": "string"

}

RÉCUPÉRER UNE CATÉGORIE PAR ID

- **Méthode** : GET
- **URL** : /categories/{id}
- **Description** : Récupère les détails d'une catégorie via son ID.
- **Réponse** :
  - **Code de statut** :
    200 OK : Catégorie trouvée.
    404 Not Found : Catégorie non trouvée.

- **Body (200)** :


```
{
 "id": "int",
 "name": "string"
}
```


RÉCUPÉRER TOUTES LES CATÉGORIES

- **Méthode** : GET
- **URL** : /categories
- **Description** : Récupère une liste complète de toutes les catégories dans le système.

RÉPONSE :

- **Code de statut** :
  200 OK : Liste des catégories retournée avec succès.
- **Body (200)** :


```
[
 {
  "id": "int",
  "name": "string"
 },
 {
  "id": "int",
  "name": "string"
 }
]
```


METTRE À JOUR UNE CATÉGORIE

- **Méthode** : PUT
- **URL** : /categories/{id}
- **Headers** :
  Content-Type: application/json
- **Body** :


```
{
```

"name": "string"

}

- **Description** : Met à jour une catégorie existante.

RÉPONSE :

- **Code de statut** :
  200 OK : Mise à jour réussie. 404 Not Found : Catégorie non trouvée.
- **Body (200)** :

{

"id": "int",

"name": "string"

}

SUPPRIMER UNE CATÉGORIE

- **Méthode** : DELETE
- **URL** : /categories/{id}
- **Description** : Supprime une catégorie via son ID. La suppression est interdite si la catégorie contient des éléments.

RÉPONSE :

- **Code de statut** :
  204 No Content : Suppression réussie.
  400 Bad Request : La catégorie contient des éléments et ne peut pas être supprimée.
  404 Not Found : Catégorie non trouvée.

3. DÉPENDANCES TECHNIQUES

- **CategoryRepository** :
  Permet d'interagir avec la base de données pour effectuer les opérations CRUD sur les catégories.

4. GESTION DES ERREURS

- **Gestion des exceptions internes** :
  En cas d'erreur interne (par exemple, catégorie non trouvée), une exception est levée avec un message d'erreur significatif.
- **Codes de réponse HTTP** :
  Utilisation des codes de réponse HTTP standard (200 OK, 201 Created, 400 Bad Request, 404 Not Found, 204 No Content) pour refléter l'état de la requête.

SERVICE FRONTEND : CATEGORY

AJOUTER UNE CATÉGORIE (POST)

- **Méthode HTTP** : POST
- **URL** : /categories

EXEMPLE DE REQUÊTE :

```
addCategory(category: Category): Observable<Category> {
 return this.httpClient.post<Category>(`${this.baseUrl}`, category, { headers: this.getAuthHeaders() }).pipe(
  catchError((error) => {
   console.error('Error adding category:', error);
   return throwError(() => new Error('Error adding category.'));
  })
 );
}
```

CORPS DE LA REQUÊTE :

```
{
 "name": "string"
}
```

RÉCUPÉRER UNE CATÉGORIE PAR ID (GET)

- **Méthode HTTP** : GET
- **URL** : /categories/{id}

EXEMPLE DE REQUÊTE :

```
getCategoryById(id: number): Observable<Category> {
 return this.httpClient.get<Category>(`${this.baseUrl}/${id}`, { headers: this.getAuthHeaders() }).pipe(
  catchError((error) => {
   console.error(`Error fetching category with ID ${id}:`, error);
   return throwError(() => new Error(`Error fetching category with ID ${id}.`));
  })
 );
}
```

5. RÉCUPÉRER TOUTES LES CATÉGORIES (GET)

- **Méthode HTTP** : GET
- **URL** : /categories
- **Exemple de requête** :

```
getAllCategories(): Observable<Category[]> {
 return this.httpClient.get<Category[]>(`${this.baseUrl}`, { headers: this.getAuthHeaders() }).pipe(
  catchError((error) => {
   console.error('Error fetching all categories:', error);
   return throwError(() => new Error('Error fetching all categories.'));
  })
 );
}
```

METTRE À JOUR UNE CATÉGORIE (PUT)

- **Méthode HTTP** : PUT
- **URL** : /categories/{id}

EXEMPLE DE REQUÊTE :

```
updateCategory(id: number, category: Category): Observable<Category> {
 return this.httpClient.put<Category>(`${this.baseUrl}/${id}`, category, { headers: this.getAuthHeaders() }).pipe(
  catchError((error) => {
   console.error('Error updating category:', error);
   return throwError(() => new Error('Error updating category.'));
  })
 );
}
```

SUPPRIMER UNE CATÉGORIE (DELETE)

- **Méthode HTTP** : DELETE
- **URL** : /categories/{id}

EXEMPLE DE REQUÊTE :

```
deleteCategory(id: number): Observable<void> {
```

```
  return this.httpClient.delete<void>(` ${this.baseUrl}/${id}` , { headers: this.getAuthHeaders() }).pipe(
    catchError((error) => {
      console.error(` Error deleting category with ID ${id}:` , error);
      return throwError(() => new Error(` Error deleting category with ID ${id}.` ));
    })
  );
}
```

SERVICES BACKEND : USER ET RATING

Une API RESTful permettant la gestion des User (CRUD) et Rating (CRUD) via l'API rest.

6. BASE URL

/users

/ratings

7. ENDPOINTS

RÉCUPÉRER TOUS LES UTILISATEURS (GET)

- **Endpoint :** /users
- **Méthode :** GET
- **Headers :**
    - Authorization: Bearer {token}
- **Description :** Récupère une liste paginée des utilisateurs.

RÉPONSE :

- **200 OK :** Retourne la liste des utilisateurs avec les détails de la pagination.
- **500 Internal Server Error :** Une erreur s'est produite lors de la récupération des données.

RÉCUPÉRER UN UTILISATEUR PAR ID (GET)

- **Endpoint :** /users/{id}
- **Méthode :** GET
- **Headers :**
    - Authorization: Bearer {token}
- **Description :** Récupère les détails d'un utilisateur spécifique par son ID.

Réponse :

- **200 OK :** Utilisateur trouvé.
- **404 Not Found :** Utilisateur non trouvé.
- **500 Internal Server Error :** Une erreur s'est produite lors de la récupération des données.

SUPPRIMER UN UTILISATEUR (DELETE)

- **Endpoint :** /users/{id}
- **Méthode :** DELETE
- **Headers :**
    - Authorization: Bearer {token}
- **Description :** Supprime un utilisateur spécifique par son ID.

RÉPONSE :

- **204 No Content :** Utilisateur supprimé avec succès.
- **404 Not Found :** Utilisateur non trouvé.
- **500 Internal Server Error :** Une erreur s'est produite lors de la suppression.

METTRE À JOUR UN UTILISATEUR (PUT)

- **Endpoint :** /users/update-user
- **Méthode :** PUT
- **Headers :**
    - Authorization: Bearer {token}
    - Content-Type: application/json
- **Body :**


```
{
"id": "int",
 firstName": "string",
  "lastName": "string",
 "email": "string",
  "roles":  [
        {
          "id": 1,
```

```
            "name": "admin",
            "description": "Administrator with full privileges"
        }
    ]
]
```


- **Description :** Met à jour les détails d'un utilisateur existant.

Réponse :

- **200 OK :** Utilisateur mis à jour avec succès.
- **500 Internal Server Error :** Une erreur s'est produite lors de la mise à jour.

ATTRIBUER DES RÔLES À UN UTILISATEUR (PUT)

- **Endpoint :** /users/{userId}/roles
- **Méthode :** PUT
- **Headers :**
  - Authorization: Bearer {token}
  - Content-Type: application/json
- **Body :**


```
"roles": [
    {
        "id": 1,
        "name": "admin",
        "description": "Administrator with full privileges"
    }
]
```


- **Description :** Attribue des rôles à un utilisateur.

RÉPONSE :

- **200 OK :** Rôles mis à jour avec succès.

- **500 Internal Server Error :** Une erreur s'est produite lors de l'attribution des rôles.

RECHERCHER DES UTILISATEURS (GET)

- **Endpoint :** /users/search
- **Méthode :** GET
- **Headers :**
  - Authorization: Bearer {token}
- **Paramètres de requête :**
  - query: string (Terme de recherche)
- **Description :** Recherche des utilisateurs en fonction du terme fourni.

RÉPONSE :

- **200 OK :** Retourne une liste d'utilisateurs correspondant au terme de recherche.
- **500 Internal Server Error :** Une erreur s'est produite lors de la recherche.

AJOUTER UNE ÉVALUATION (POST)

- **Endpoint :** /ratings/add
- **Méthode :** POST
- **Headers :**
  - Authorization: Bearer {token}
  - Content-Type: application/json
- **Body :**

```
{
 "posterId": "int",
 "receiverId": "int",
 "score": "double",
 "comment": "string"
}
```

- **Description :** Ajoute une nouvelle évaluation d'un utilisateur à un autre.

RÉPONSE :

  - **201 Created :** Évaluation ajoutée avec succès.
  - **500 Internal Server Error :** Une erreur s'est produite lors de l'ajout de l'évaluation.

RÉCUPÉRER LES ÉVALUATIONS D'UN UTILISATEUR (GET)

- **Endpoint :** /ratings/received/{userId}
- **Méthode :** GET
- **Headers :**
  - Authorization: Bearer {token}
- **Description :** Récupère toutes les évaluations reçues par un utilisateur.

RÉPONSE :

  - **200 OK :** Retourne une liste d'évaluations.
  - **500 Internal Server Error :** Une erreur s'est produite lors de la récupération des évaluations.

SUPPRIMER UNE ÉVALUATION (DELETE)

- **Endpoint :** /ratings/{ratingId}
- **Méthode :** DELETE
- **Headers :**
  - Authorization: Bearer {token}
- **Description :** Supprime une évaluation spécifique.

Réponse :

  - **204 No Content :** Évaluation supprimée avec succès.
  - **500 Internal Server Error :** Une erreur s'est produite lors de la suppression.

METTRE À JOUR UNE ÉVALUATION (PUT)

- **Endpoint :** /ratings/{ratingId}
- **Méthode :** PUT
- **Headers :**
    - Authorization: Bearer {token}
    - Content-Type: application/json
- **Body :**

  {

  "numberStar": "double",

  "comment": "string"

  }
- **Description :** Met à jour une évaluation spécifique**.**

RÉPONSE :

- 200 OK : Évaluation mise à jour avec succès.
- 500 Internal Server Error : Une erreur s'est produite lors de la mise à jour.

SERVICE FRONTEND : USER ET RATING

Utilisation du backend dans le frontend via les services Angular.

1. AJOUTER UN UTILISATEUR

- **Méthode :** POST
- **URL :** /users
- **Description :** Envoie une requête pour ajouter un utilisateur via l'API.

CORPS DE LA REQUÊTE :

{

"firstName": "string",

"lastName": "string",

"email": "string",

"roles": ["int"]

}

RÉPONSE :

- **201 Created :** Utilisateur ajouté avec succès.
- **500 Internal Server Error :** Une erreur s'est produite lors de l'ajout de l'utilisateur.

2. RÉCUPÉRER TOUS LES UTILISATEURS

- **Méthode :** GET
- **URL :** /users
- **Description :** Récupère tous les utilisateurs via l'API.

RÉPONSE :

- **200 OK :** Retourne la liste des utilisateurs.
- **500 Internal Server Error :** Une erreur s'est produite lors de la récupération des utilisateurs.

3. SUPPRIMER UN UTILISATEUR

- **Méthode :** DELETE
- **URL :** /users/{id}
- **Description :** Supprime un utilisateur par son ID via l'API.

RÉPONSE :

- **204 No Content :** Utilisateur supprimé avec succès.
- **404 Not Found :** Utilisateur non trouvé.
- **500 Internal Server Error :** Une erreur s'est produite lors de la suppression.

AJOUTER UNE ÉVALUATION

- **Méthode :** POST
- **URL :** /ratings/add
- **Description :** Ajoute une évaluation d'un utilisateur à un autre.

CORPS DE LA REQUÊTE :

```
{
 "posterId": "int",
 "receiverId": "int",
```

```
"score": "double",
"comment": "string"
}
```


RÉPONSE :

- **201 Created :** Évaluation ajoutée avec succès.
- **500 Internal Server Error :** Une erreur s'est produite lors de l'ajout de l'évaluation.

4. RÉCUPÉRER LES ÉVALUATIONS D'UN UTILISATEUR

- **Méthode :** GET
- **URL :** /ratings/received/{userId}
- **Description :** Récupère toutes les évaluations reçues par un utilisateur.
- RÉPONSE **:**
  - **200 OK :** Retourne la liste des évaluations.
  - **500 Internal Server Error :** Une erreur s'est produite lors de la récupération des évaluations.

5. SUPPRIMER UNE ÉVALUATION

- **Méthode :** DELETE
- **URL :** /ratings/{ratingId}
- **Description :** Supprime une évaluation spécifique.
- RÉPONSE **:**
  - **204 No Content :** Évaluation supprimée avec succès.
  - **404 Not Found :** Évaluation non trouvée.
  - **500 Internal Server Error :** Une erreur s'est produite lors de la suppression.

SERVICE BACKEND : ITEM

Une API RESTful permettant la gestion des items (CRUD).

1. BASE URL

/items

2. ENDPOINTS

1. AJOUTER UN ITEM

REQUÊTE:

Method: POST

URL: /items

Headers:

Content-Type: application/json

Body:


```
{
  "id_item": number,
  "name": "string",
  "description": "string",
  "photo": « string »,
  "category": {
    "id_category":number
  },
  "owner": {
    "id": number
  },
  "available": false
}
```


Description : Géocodage de l'adresse (lat, lon) via Nominatim, suivi de l'enregistrement dans la base de données.

RÉPONSE:

- **Status Code**:
- 201 Created: Adresse ajoutée avec succès.
- 500 Internal Server Error: Une erreur s'est produite.
- **Body (201)**:


```
{
  "id": "int",
  "street": "string",
  "number": "string",
  "city": "string",
  "zipCode": "string",
  "latitude": "double",
  "longitude": "double"
```

}

2. RÉCUPÉRER UNE ADRESSE PAR ID

REQUÊTE:

Method: GET

URL: /addresses/{id}

Description : Récupère les détails d'une adresse via son ID.

RÉPONSE:

- **Status Code**:
  - 200 OK: Adresse trouvée.
  - 404 Not Found: Adresse non trouvée.
  - 500 Internal Server Error: Une erreur s'est produite.
- **Body (200)**:


```
{
 "id": "int",
 "street": "string",
 "number": "string",
 "city": "string",
 "zipCode": "string",
 "latitude": "double",
 "longitude": "double"
}
```


3. RÉCUPÉRER TOUTES LES ADRESSES

REQUÊTE:

- **Method:** GET
- **URL:** /addresses
- **Headers**:
  - Aucun requis.
- **Description**: Récupère une liste complète des adresses dans le système.

RÉPONSE:

- **Status Code**:
  - 200 OK: Liste retournée avec succès.
  - 500 Internal Server Error: Une erreur s'est produite.
- **Body (200)**:


```
[
 {
  "id": "int",
  "street": "string",
  "number": "string",
```

```
  "city": "string",
  "zipCode": "string",
  "latitude": "double",
  "longitude": "double"
 },
 {
  "id": "int",
  "street": "string",
  "number": "string",
  "city": "string",
  "zipCode": "string",
  "latitude": "double",
  "longitude": "double"
 }
]
```


4. SUPPRIMER UNE ADRESSE

REQUÊTE:

- **Method:** DELETE
- **URL:** /addresses/{id}
- **Headers:**
  - Aucun requis.
- **Description:** Supprime une adresse via son ID.

RÉPONSE:

- **Status Code**:
  - 204 No Content: Suppression réussie.
  - 404 Not Found: Adresse non trouvée.
  - 500 Internal Server Error: Une erreur s'est produite.

5. METTRE À JOUR UNE ADRESSE

REQUÊTE:

- **Method:** PUT
- **URL:** /addresses
- **Headers:**
  - Content-Type: application/json
- **Body:**


```
{
 "id": "int",
 "street": "string",
 "number": "string",
 "city": "string",
 "zipCode": "string"
}
```


- **Description**: Géocodage de l'adresse mise à jour (lat, lon) via Nominatim, suivi de sa sauvegarde dans la base de données.

RÉPONSE:

- **Status Code**:
  - 200 OK: Mise à jour réussie.
  - 500 Internal Server Error: Une erreur s'est produite.
- **Body (200)**:

```
{
 "id": "int",
 "street": "string",
 "number": "string",
 "city": "string",
 "zipCode": "string",
 "latitude": "double",
 "longitude": "double"
}
```

3. DÉPENDANCES TECHNIQUES

NOMINATIM API:

  - **URL d'accès**: Configurable via la propriété nominatim.api.url.
  - **Requête**:

    GET /?q={formattedAddress}&format=json

    - formattedAddress: Adresse formatée sous forme de chaîne.
  - **Réponse**:

```
[
 {
   "lat": "double",
   "lon": "double"
 }
]
```

  - **Codes de réponse**:

- 200 OK: Requête réussie.
- 4xx/5xx: Erreur avec description associée.

4. GESTION DES ERREURS

- Validation des champs obligatoires pour les requêtes entrantes.
- Gestion des exceptions internes avec des messages significatifs.
- Utilisation des codes HTTP standard pour refléter l'état de la requête.