**Routes Utilisateur**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CRUD | Point d'accès | Request body (si nécessaire) | Type de réponse attendue | fonction |
| POST | /api/account/signup | { userId: string,  password: string} | {message :string} | Hachage du mdp et ajout de l’utilisateur à la BDD |
| POST | /api/account/login | {userId: string,  password: string} | {userId :String  Role :string  Token :string} | Vérification des  Informations d'identification  de l'utilisateur,  renvoie l \_id de l'utilisateur  depuis la base de données  et un token web JSON signé  (contenant également l'\_id  de l'utilisateur). |
| DELETE | /api/account/ | {userId : string} | {message :string} | Supprime le compte loggé avec l’identification fournie.  Authentifié par token. |

**Routes Posts**

Les routes des posts doivent être identifiées avec un token

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CRUD | Point d'accès | Request body (si nécessaire) | Type de réponse attendue | Fonction |
| GET | /api/post | - | {posts :[Array of posts]} | Renvoie un tableau de tous les posts à afficher (à afficher du plus récent au plus ancien) |
| GET | /api/post/single/:postid | - | {posts :[Single post],  comments :[commentaires liés au Post]} | Renvoie le post avec l’ID fourni et les commentaires rattachés au post (à afficher du plus ancien au plus récent) |
| GET | /api/post/fromUser/ :userId | - | {posts :[Array of posts]} | Renvoie un tableau de tous les posts à afficher (à afficher du plus récent au plus ancien). Posts de l’utilisateur ou postés par l’utilisateur |
| PUT | /api/post/ flag  /:postId | - | {message :string} | Flag post : passe flag à true. Peut être fait par n’importe quel utilisateur |
| PUT | /api/post/unflag /:postId |  | {message :string} | unflag post : passe flag à false. Peut être fait uniquement par un admin |
| GET | /api/post/  flagged | - | {posts :[Array of posts]} | Fetch tous les posts signalés |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CRUD | Point d'accès | Request body (si nécessaire) | Type de réponse attendue | fonction |
| POST | /api/post | { post: String,  image: File } | {message :string} | Capture et enregistre  l'image, analyse le post  transformé en chaîne de caractères et l'enregistre  dans la base de données en définissant correctement  son imageUrl. Initialise les likes et dislikes de du post à 0 et les usersLiked et  usersDisliked avec des  tableaux vides. Remarquezque le corps de la demande  initiale est vide ; lorsque multer est ajouté, il renvoie une chaîne pour le corps de la demande en fonction des données soumises avec le fichier.  Ajoute l’ID du post au tableau des posts partagés par l’utilisateur sharedPosts, afin qu’il puisse être affiché sur le fil de l’utilisateur. |
| DELETE | /api/post/:postId | {userId : string} | { message: String } | Supprime le post avec l'\_id fourni. |
| POST | /api/post/:postid/like | {userId : string,  like: INT (1,0,-1}} | { message: String} | Définit le statut « Like » pour l' userId fourni. Si like = 1, l'utilisateur aime (= like) le post. Si like = 0, l'utilisateur annule son like ou son dislike. Si like = -1, l'utilisateur n'aime pas (= dislike) le post. L'ID de l'utilisateur doit être ajouté ou retiré du tableau approprié. Cela permet de garder une trace de leurs préférences et les empêche de liker ou de ne pas disliker le même post plusieurs fois : un utilisateur ne peut avoir qu'une seule valeur pour chaque post. Le nombre total de « Like » et de « Dislike » est mis à jour à chaque nouvelle notation. |

**Routes Commentaires**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CRUD | Point d'accès | Request body (si nécessaire) | Type de réponse attendue | fonction |
| POST | /api/comment | {  content: text  userId : string,  postId :INT  } | {message :string} | Créer Commentaire :  Créer un commentaire associé à un utilisateur et à un post. |
| DELETE | /api/comment/ :commentId | {userId :STRING} | {message :string} | Supprimer Commentaire :  Créer un commentaire associé à un utilisateur et à un post. |
| PUT | /api/comment/  : commentId | {  content: text  userId :STRING  } | {message :string} | Modifier Commentaire : Changer le texte contenu dans un commentaire |
| PUT | /api/comment/ flag / :commentId | - | {message :string} | flag Commentaire : passe flag à true. Peut être fait par n’importe quel utilisateur |
| PUT | /api/comment/ unflag / :commentId | - | {message :string} | unflag Commentaire : passe flag à false. Peut être fait uniquement par un admin |
| GET | /api/comment/getFrom/  :postId | - | Array of comments | Fetch tous les commentaires liés à un post |
| GET | /api/comment/  flagged | - | Array of comments | Fetch tous les commentaires signalés |
| GET | /api/comment/ flagged/ :commentId | - | comment | Fetch un commentaire pour vérification et modération. |

**Modèles de données Posts**

* postId : Int – Clé primaire, générée automatiquement par MySql
* userId : STRING –clé etrangère(Users)- required-- email de l’utilisateur qui a crée le post
* createdAt :date - required - les posts doivent-être affichés du plus ancien au plus récent. La date est crée automatiquement par Sequelize.
* title :string – required - titre défini par l’utilisateur
* imageUrl : string – required - l’URL de l’image ou du gif téléchargé par l’utilisateur.
* usersUpvotes : [ "String <userId>" ] — required- tableau des identifiants des utilisateurs qui ont aimé (= liked) le post
* usersDownvotes : [ "String <userId>" ] — required - tableau des identifiants des utilisateurs qui n'ont pas aimé (= disliked) le post
* flagged : boolean – default : false – passer à true quand signalé

**Modèles de données Users**

* userId : String —clé primaire-- required- adresse e-mail de l'utilisateur [unique]
* password : String — required- mot de passe de l'utilisateur haché
* rôle : String – required : standard ou admin. Standard par défaut. Le rôle ne peut être changé que par une intervention manuelle dans la Base de données.

**Modèles de données Commentaires**

* commentId : Int – Clé primaire, générée automatiquement par MySql
* userId : STRING – required-- clé etrangère(Users)- email de l’utilisateur qui a crée le commentaire
* postId : Int –required-- Clé étrangère (table posts), Identifiant du post commenté
* content : text (type sequelize)-- required – contenu du commentaire
* flagged : boolean – default : false – passer à true quand signalé

**Schéma relationnel des tables**

