

Cours MERN - Semaine 4

Finalisation des Opérations CRUD

Abdelweheb GUEDDDES & Mohamed Ben Jazia / Ecole Polytechnique Sousse

9 septembre 2025

Table des matières

1	Objectifs Pédagogiques Détaillés	2
2	Partie 1 : Concepts Techniques Approfondis (1h15)	2
2.1	Cibler une Ressource : Les Paramètres de Route	2
2.2	Méthodes HTTP et Status Codes Appropriés	2
2.3	Les Outils Mongoose pour le CRUD	3
3	Partie 2 : Atelier Pratique - Implémentation du CRUD (1h45)	4
3.1	Étape 1 : Lire un Article Spécifique (Read One)	4
3.2	Étape 2 : Mettre à Jour un Article (Update)	5
3.3	Étape 3 : Supprimer un Article (Delete)	6
3.4	Étape 4 : Cycle de Test Complet avec Postman	7
4	Conclusion et Vision pour la Suite	7
5	Travail Pratique Complémentaire (À faire par l'étudiant)	7

1 Objectifs Pédagogiques Détaillés

Cette semaine, nous allons rendre notre API entièrement fonctionnelle en complétant l'acronyme CRUD. À la fin de cette séance, vous serez capable de :

- **Comprendre et utiliser** les paramètres de route (ex : `/api/articles/:id`) pour cibler des ressources spécifiques.
- **Accéder** aux valeurs des paramètres via l'objet `req.params` dans les contrôleurs.
- **Implémenter** la logique pour lire un document unique depuis MongoDB (**Read One**).
- **Implémenter** la logique pour mettre à jour un document existant (**Update**).
- **Implémenter** la logique pour supprimer un document (**Delete**).
- **Utiliser** les méthodes Mongoose appropriées : `findById`, `findByIdAndUpdate`, `findByIdAndDelete`.
- **Gérer** les cas où une ressource n'est pas trouvée en renvoyant un statut HTTP 404 **Not Found**.
- **Maîtriser** le cycle de test complet d'une ressource API avec Postman.

2 Partie 1 : Concepts Techniques Approfondis (1h15)

2.1 Cibler une Ressource : Les Paramètres de Route

Jusqu'à présent, nos routes comme `/api/articles` s'appliquaient à l'ensemble de la collection. Comment faire pour lire, modifier ou supprimer *un seul article spécifique* ? La réponse est : les paramètres de route.

Un paramètre de route est un segment d'URL variable, que l'on définit en préfixant son nom par deux-points ('`:`').

Exemple : Dans la définition de route `router.get('/:id', ...)`, `:id` est un paramètre de route.

- Si un client fait une requête GET vers `/api/articles/60c72b2f9b1d8c001f8e4b2a`,
- Express intercepte cette requête, comprend que `60c72b2f9b1d8c001f8e4b2a` est la valeur du paramètre `id`,
- Et rend cette valeur disponible dans le contrôleur via l'objet `req.params`. On y accèdera avec `req.params.id`.

2.2 Méthodes HTTP et Status Codes Appropriés

Pour une API REST professionnelle, il est crucial d'utiliser les bons verbes HTTP et de renvoyer les bons codes de statut.

PUT : Utilisé pour **remplacer entièrement** une ressource existante. Le client doit fournir l'intégralité du nouvel objet.

DELETE : Utilisé pour **supprimer** une ressource.

PATCH : (Alternative à PUT) Utilisé pour une mise à jour **partielle**. Le client ne fournit que les champs à modifier. Nous utiliserons PUT dans ce cours pour sa simplicité.

Codes de Statut Essentiels pour aujourd'hui :

- **200 OK** : Succès standard. Parfait pour un GET ou un PUT réussi.
- **404 Not Found : Crucial.** Doit être renvoyé lorsque le client demande une ressource qui n'existe pas (ex : un article avec un ID incorrect).

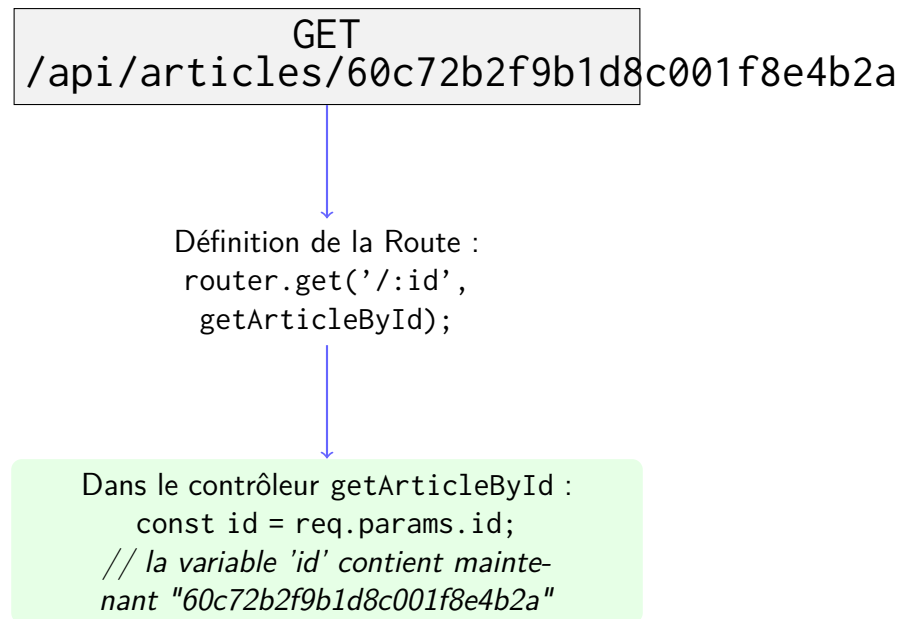


FIGURE 1 – Comment Express mappe un segment d’URL à `req.params`.

- **204 No Content** : Souvent utilisé pour un DELETE réussi, quand le serveur n’a rien de plus à dire et ne renvoie pas de corps de réponse. Par simplicité et pour donner un feedback clair, nous renverrons un **200 OK** avec un message JSON.

2.3 Les Outils Mongoose pour le CRUD

Mongoose nous fournit des méthodes directes pour chaque opération que nous voulons implémenter. Toutes ces méthodes retournent des promesses, elles sont donc parfaites pour `async/await`.

- `Model.findById(id)` : Recherche un unique document correspondant à l’ID fourni. Retourne le document s’il est trouvé, sinon `null`.
- `Model.findByIdAndUpdate(id, update, options)` : Trouve un document par son ID et le met à jour.
 - `id` : L’ID du document à mettre à jour.
 - `update` : Un objet contenant les champs à modifier (ex : `req.body`).
 - `options` : Un objet de configuration. L’option `{ new: true }` est très importante : elle force la méthode à retourner le document *après* la mise à jour, et non avant.
- `Model.findByIdAndDelete(id)` : Trouve un document par son ID et le supprime. Retourne le document qui a été supprimé.

3 Partie 2 : Atelier Pratique - Implémentation du CRUD (1h45)

Nous allons reprendre notre projet et compléter la gestion des articles.

3.1 Étape 1 : Lire un Article Spécifique (Read One)

1. **Mettre à jour le routeur** (routes/articleRoutes.js) : Ajoutez une nouvelle route qui accepte un paramètre id.

```
1 // ... (importations)
2 const { getAllArticles, createArticle, getArticleById } = require
  ('../controllers/articleController');
3
4 router.get('/', getAllArticles);
5 router.post('/', createArticle);
6
7 // NOUVELLE ROUTE : Doit correspondre à la nouvelle fonction du
  contrôleur
8 router.get('/:id', getArticleById);
9
10 module.exports = router;
11
```

Listing 1 – routes/articleRoutes.js - Ajout de la route GET by ID

2. **Ajouter la logique au contrôleur** (controllers/articleController.js) : Créez la fonction 'getArticleById'.

```
1 // ... (autres fonctions)
2
3 // @desc    Récupérer un article par son ID
4 // @route   GET /api/articles/:id
5 const getArticleById = async (req, res) => {
6   try {
7     const article = await Article.findById(req.params.id);
8
9     // Si l'article n'est pas trouvé, findById retourne null
10    if (!article) {
11      return res.status(404).json({ message: "Article non
12      trouvé." });
13    }
14
15    res.status(200).json(article);
16  } catch (err) {
17    res.status(500).json({ message: "Erreur serveur.", error:
18    err.message });
19  }
20 };
21
22 module.exports = {
23   getAllArticles,
24   createArticle,
25   getArticleById
26 };
```

```
23     getArticleById // Ne pas oublier d'exporter la nouvelle
    fonction
24 };
25
```

Listing 2 – controllers/articleController.js - Ajout de getArticleById

3.2 Étape 2 : Mettre à Jour un Article (Update)

1. Mettre à jour le routeur :

```
1 // ... (importations)
2 const { getAllArticles, createArticle, getArticleById,
    updateArticle } = require('../controllers/articleController');
3
4 // ... (routes GET et POST)
5 router.get('/:id', getArticleById);
6
7 // NOUVELLE ROUTE
8 router.put('/:id', updateArticle);
9
10 module.exports = router;
11
```

Listing 3 – routes/articleRoutes.js - Ajout de la route PUT

2. Ajouter la logique au contrôleur :

```
1 // ... (autres fonctions)
2
3 // @desc    Mettre à jour un article
4 // @route   PUT /api/articles/:id
5 const updateArticle = async (req, res) => {
6     try {
7         const updatedArticle = await Article.findByIdAndUpdate(
8             req.params.id, // L'ID de l'article à mettre à jour
9             req.body,      // Les nouvelles données de l'article
10            { new: true, runValidators: true } // Options
11        );
12
13        if (!updatedArticle) {
14            return res.status(404).json({ message: "Article non
15            trouvé." });
16        }
17
18        res.status(200).json(updatedArticle);
19    } catch (err) {
20        res.status(400).json({ message: "Erreur lors de la mise à
21        jour.", error: err.message });
22    }
23 };
24
25 module.exports = {
26     // ...
27 }
```

```
25   getArticleById ,
26   updateArticle // Exporter la fonction
27 };
28
```

Listing 4 – controllers/articleController.js - Ajout de updateArticle

3.3 Étape 3 : Supprimer un Article (Delete)

1. Mettre à jour le routeur :

```
1 // ... (importations)
2 const { getAllArticles , createArticle , getArticleById ,
   updateArticle , deleteArticle } = require('../controllers/
   articleController');
3
4 // ... (routes GET, POST, PUT)
5 router.put('/:id', updateArticle);
6
7 // NOUVELLE ROUTE
8 router.delete('/:id', deleteArticle);
9
10 module.exports = router;
11
```

Listing 5 – routes/articleRoutes.js - Ajout de la route DELETE

2. Ajouter la logique au contrôleur :

```
1 // ... (autres fonctions)
2
3 // @desc   Supprimer un article
4 // @route  DELETE /api/articles/:id
5 const deleteArticle = async (req, res) => {
6   try {
7     const deletedArticle = await Article.findByIdAndDelete(
7       req.params.id);
8
9     if (!deletedArticle) {
10       return res.status(404).json({ message: "Article non
11         trouvé." });
12
13       res.status(200).json({ message: "Article supprimé avec
14         succès.", id: req.params.id });
15     } catch (err) {
16       res.status(500).json({ message: "Erreur serveur.", error:
17         err.message });
18     }
19   }
20 };
21
22 module.exports = {
23   // ...
24   updateArticle ,
25
```

```
22   deleteArticle // Exporter la fonction
23 };
24
```

Listing 6 – controllers/articleController.js - Ajout de deleteArticle

3.4 Étape 4 : Cycle de Test Complet avec Postman

Suivez ce scénario pour valider que tout fonctionne.

1. **Créez un article** : Envoyez une requête POST à `/api/articles` avec un corps JSON. Copiez l'ID (`_id`) de l'article créé dans la réponse.
2. **Lisez tous les articles** : Envoyez une requête GET à `/api/articles` pour confirmer que votre nouvel article est bien dans la liste.
3. **Lisez l'article spécifique** : Envoyez une requête GET à `/api/articles/VOTRE_ID`. Vous devriez recevoir uniquement cet article.
4. **Testez le cas 404** : Envoyez une requête GET à `/api/articles/un_id_incorrect`. Vous devriez recevoir un statut **404 Not Found** et un message d'erreur.
5. **Mettez à jour l'article** : Envoyez une requête PUT à `/api/articles/VOTRE_ID`. Dans l'onglet "Body", mettez un JSON avec un titre ou un contenu différent. La réponse doit être l'article mis à jour.
6. **Supprimez l'article** : Envoyez une requête DELETE à `/api/articles/VOTRE_ID`. Vous devriez recevoir un message de succès.
7. **Confirmez la suppression** : Envoyez à nouveau une requête GET à `/api/articles/VOTRE_ID`. Vous devriez maintenant recevoir une erreur **404 Not Found**.

4 Conclusion et Vision pour la Suite

Félicitations! Vous disposez maintenant d'une API REST entièrement fonctionnelle pour une ressource. Vous maîtrisez le cycle CRUD complet, la gestion des paramètres de route et le renvoi des codes de statut HTTP appropriés. C'est le socle de la quasi-totalité des applications web.

La semaine prochaine, nous aborderons des sujets plus avancés pour rendre notre API encore plus robuste et professionnelle : la **validation des données entrantes** et une **gestion des erreurs centralisée**.

5 Travail Pratique Complémentaire (À faire par l'étudiant)

Appliquez le cycle CRUD complet à la ressource "Utilisateurs".

1. **Implémentez les routes** et les fonctions de contrôleur pour :
 - GET `/api/users/:id` (Récupérer un utilisateur par son ID)
 - PUT `/api/users/:id` (Mettre à jour un utilisateur)
 - DELETE `/api/users/:id` (Supprimer un utilisateur)
2. **Logique du contrôleur** : Utilisez `User.findById()`, `User.findByIdAndUpdate()`, et `User.findByIdAndDelete()`.
3. **Gestion des erreurs** : N'oubliez pas de gérer le cas où un utilisateur n'est

pas trouvé (erreur 404).

4. **Cycle de test complet** : Utilisez Postman pour tester rigoureusement chaque nouvelle route, y compris les cas d'erreur.

Note Importante : Travail à Rendre

Votre compte rendu devra inclure des captures d'écran de Postman pour chaque nouvelle route implémentée (GET by ID, PUT, DELETE), y compris le test du cas 404.

Expliquez le rôle des options `{ new: true }` et `{ runValidators: true }` dans `findByIdAndUpdate`. Échéance : La soumission doit se faire au plus tard la veille de la prochaine séance, à 23h59 précises. Ce compte rendu est obligatoire et noté.