Etude de cas
Création de l'API des articles

Marron Alexis

Le 21/07/2024

1.Rédigez un diagramme UML de la base de données

Les réponses doivent être répertoriées sur un compte GitHub ou GitLab. Assurez-vous que le lien envoyé soit public.

Lien github : https://github.com/cactusSpy/express_initiation

Etape Installation

1. Téléchargement de MongoDB Community Server via le site web : https://www.mongodb.com

2. Installation des packages(dépendances) avec la commande :

```
PS D:\projetPro\Nodejs-approfondissement> npm install -y
npm WARN old lockfile
npm WARN old lockfile The package-lock.json file was created with an old version of npm,
```

Dossier Node modules crée :

3. Résoudre les problèmes de vulnérabilités

```
46 packages are looking for funding
  run `npm fund` for details

20 vulnerabilities (7 moderate, 11 high, 2 critical)
```

Exécution de la commande **npm audit** dans le PowerShell pour le détails des vulnérabilités.

```
# npm audit report

@babel/traverse  <7.23.2
Severity: critical
Babel vulnerable to arbitrary code execution when compiling specifically crafted malicious code -
om/advisories/GHSA-67hx-6x53-jw92
fix available via `npm audit fix`
node_modules/@babel/traverse

braces  <3.0.3
Severity: high
Uncontrolled resource consumption in braces - https://github.com/advisories/GHSA-grv7-fg5c-xmjg
fix available via `npm audit fix`
node_modules/braces
```

Lancement de la commande npm audit fix , cela résout les vulnérabilités en mettant à jour les paquets vers des versions corrigées.

Pour ne pas casser l'application, on se contentera de cette commande afin de ne pas apporter de changements majeurs

Résultats des vulnérabilités après la commande :

```
4 vulnerabilities (3 moderate, 1 high)
```

4. Test et lancement du serveur sur le port 3000
   Lancement de l'application à travers un script et la commande npm run « script »
   npm run **start** :

```
"scripts": {
  "start": "nodemon www/app.js",
  "dev":   "node www/app.js",
  "test": "jest"
},
```

   Cela va lancer un utilitaire **nodemon** qui permet de **:**
   .Surveille le dossier du projet.
   .Relance le serveur à chaque modification des fichiers

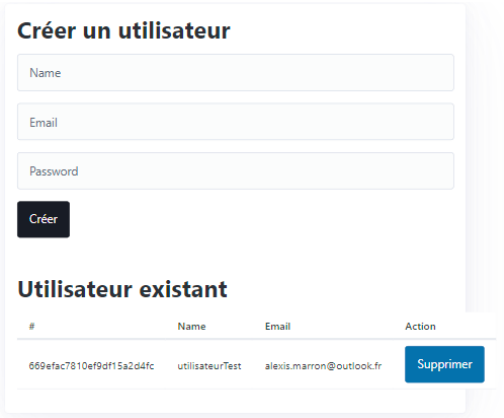
```
PS D:\projetPro\Nodejs-approfondissement> npm run start

> myapp@1.0.0 start
> nodemon www/app.js

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node www/app.js`
```

   Ou avec le script dev qui lance **node** et non nodemon :
   .Il faudra dans ce cas, arrêter et relancer manuellement le projet à chaque modifications.

   Lancement des tests

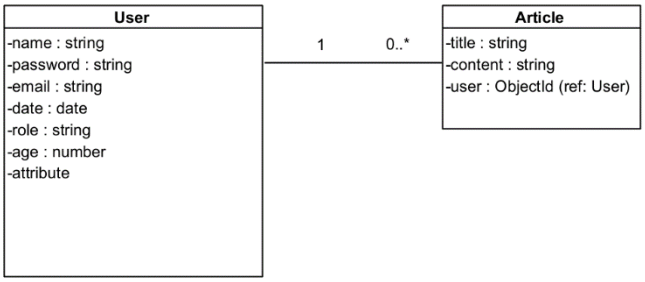
```
PS D:\projetPro\Nodejs-approfondissement> npm run test

> myapp@1.0.0 test
> jest

 PASS  tests/users.spec.js
  tester API users
    √ [Users] Get All (29 ms)
    √ [Users] Create User (72 ms)
    √ Est-ce userService.getAll (4 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
```

5. L’application sur le port 3000



Etape UML BDD

Sur visual paradigm avec un diagramme de classe UML voici la représentation obtenue grâce au fichiers du projet :

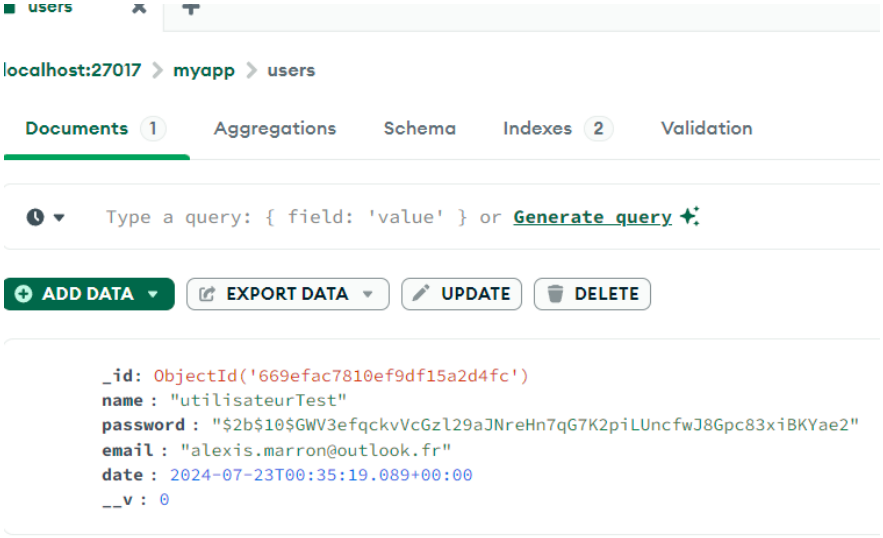


6. test de la BDD et utilisateurs

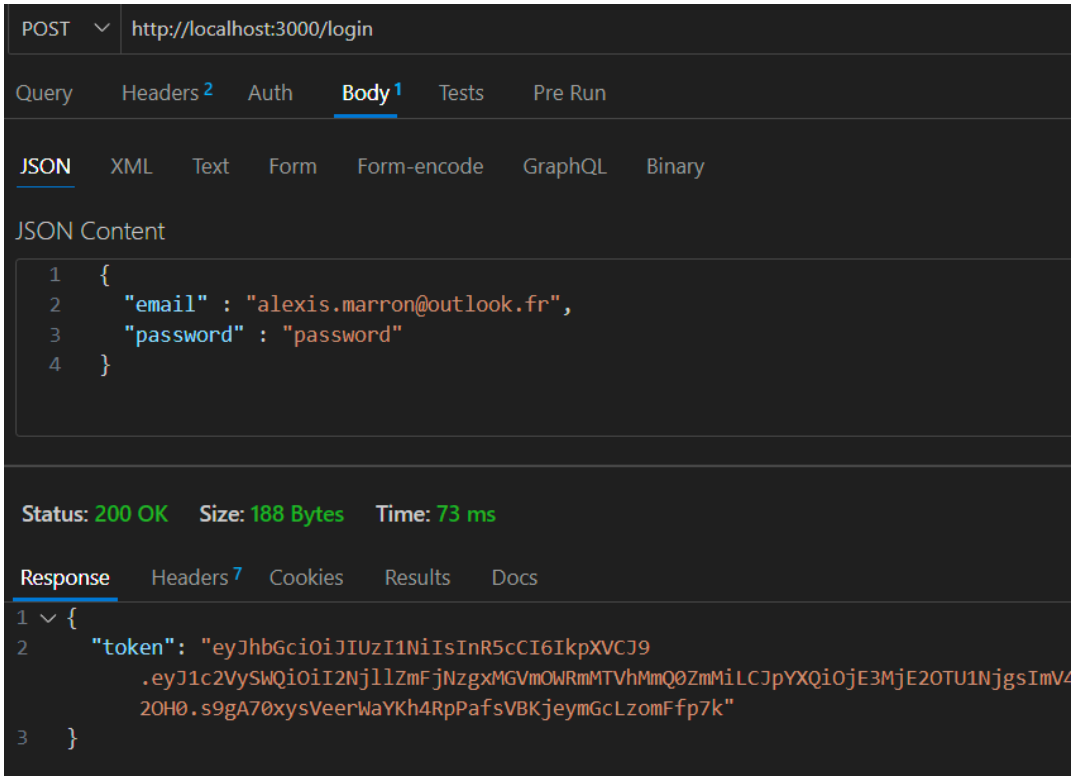
Ci-dessous la création d'un utilisateur pour effectuer des tests :

email : alexis.marron@outlook.fr
PWD : password

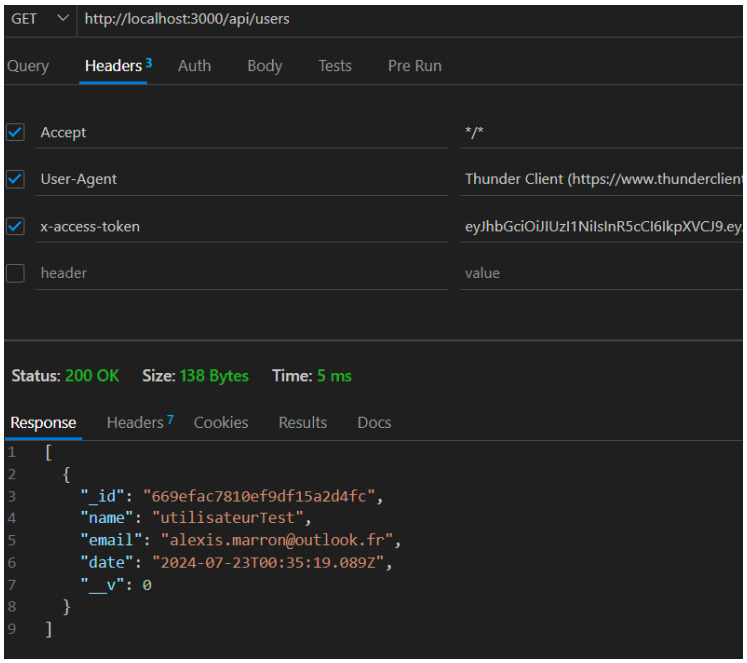
Dans la BDD :



Authentification via POST : http//localhost :3000/login



Et la récupération des utilisateurs avec le token précèdent



2. Ajouter l'énumération dans le schéma de article.schema.js avec deux valeurs : draft, published

```
const articleSchema = Schema({
  title: String,
  content: String,
  user: {
    type: Schema.Types.ObjectId,
    ref: "User",
  },
  phase: {
    type: String,
    enum: ['draft', 'published'],
  },
});
```

D'après la documentation https://mongoosejs.com/docs/schematypes.html#string-validators pour des string :

- `enum`: Array, creates a validator that checks if the value is in the given array.

On ajoute donc les deux valeurs **draft** et **published** dans l'énumération phase de type String.

3. Créer les Endpoint de création, mise à jour et suppression d'un article.
L'utilisateur doit être connecté pour effectuer la création

1. Rédigez le code dans /*api*/articles/articles.service.js avec les 3 méthodes (création, mise à jour, suppression)

```
const Article = require("./article.schema");

class ArticleService {
  create(data) {
    const article = new Article(data);
    return article.save();
  }

  update(id, data) {
    return Article.findByIdAndUpdate(id, data, { new: true });
  }

  delete(id) {
    return Article.deleteOne({ _id: id });
  }
}

module.exports = new ArticleService();
```

2. Rédigez le code dans *api/*articles/articles.controller.js avec les 3 méthodes correspondant au contrôleur

3. Rédigez les routes dans api/articles/articles.router.js. Pensez à l'ajouter dans votre application (server.js)

```
const express = require("express");
const articlesController = require("./articles.controller");
const router = express.Router();

router.post("/", articlesController.create);
router.put("/:id", articlesController.update);
router.delete("/:id", articlesController.delete);

module.exports = router;
```

Dans le fichier server.js :

```
app.use('/api/articles', articlesRouter);
```

Avec ceci articlesRouter gère toutes les requêtes liées aux articles .

4. Pensez à ajouter le temps réel dans le contrôleur
On utilise ici « req.io.emit » pour diffuser des événements à tous les clients connectés via WebSocket.

5. Modifiez le middleware d'authentification afin de faire récupérer toutes les informations d'un utilisateur et les faire passer dans « req » (et pas seulement l'id de l'utilisateur)

```
ojetPro > Nodejs-approfondissement > middlewares > JS auth.js > ...
const UnauthorizedError = require("../errors/unauthorized");
const jwt = require("jsonwebtoken");
const config = require("../config");
const User = require("../api/users/users.service");

module.exports = async (req, res, next) => {
  try {
    const token = req.headers["x-access-token"];
    if (!token) {
      throw new Error("No token provided");
    }
    const decoded = jwt.verify(token, config.secretJwtToken);
    console.log("Decoded token:", decoded);

    const user = await User.get(decoded.userId);
    console.log("Decoded user:", user);

    if (!user) {
      throw new Error("User not found");
    }

    req.user = user;
    next();
  } catch (error) {
    console.error("Authentication error:", error);
    next(new UnauthorizedError(error.message || error));
  }
};
```

```
Decoded token: {
  userId: '669efac7810ef9df15a2d4fc',
  iat: 1721819005,
  exp: 1722078205
}
Decoded user: {
  _id: new ObjectId("669efac7810ef9df15a2d4fc"),
  name: 'utilisateurTest',
  email: 'alexis.marron@outlook.fr',
  date: 2024-07-23T00:35:19.089Z,
  __v: 0
}
```

6. Faites en sorte que la modification et la suppression (dans le contrôleur) s'effectuent seulement si l'utilisateur est « admin » (propriété « role » sur l'utilisateur).

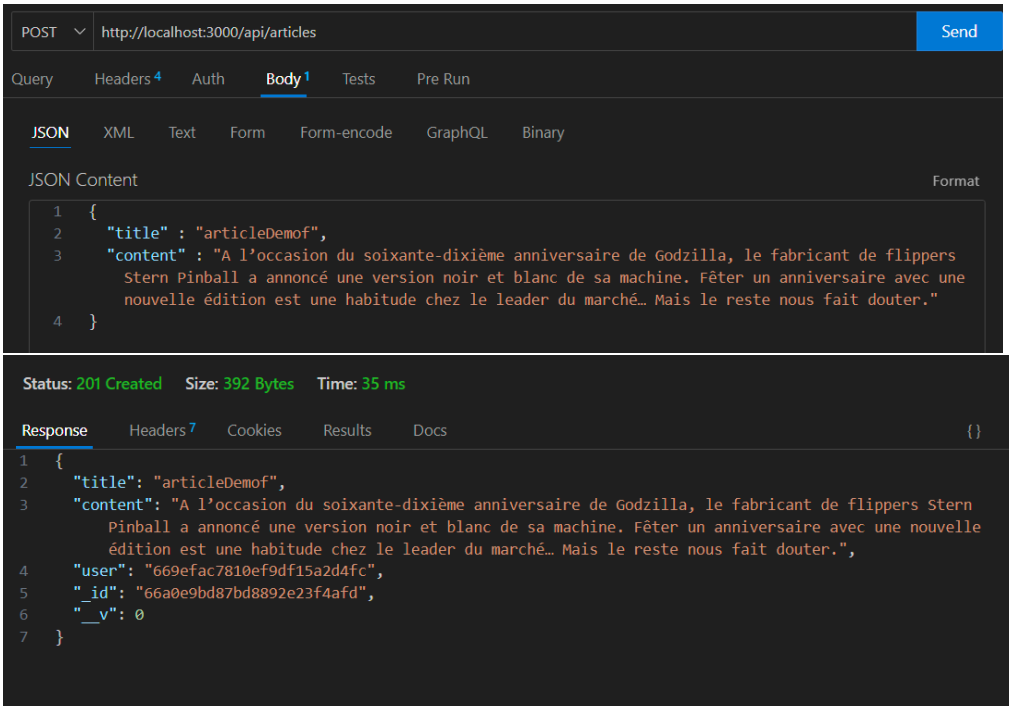
```
projetPro > Nodejs-approfondissement > api > articles > JS articles.router.js > ...
const express = require("express");
const articlesController = require("./articles.controller");
const router = express.Router();
const authMiddleware = require('../../middlewares/auth');

router.post('/', authMiddleware, articlesController.create);
router.put('/:id', authMiddleware, articlesController.update);
router.delete('/:id', authMiddleware, articlesController.delete);
```

```
try {
  if (req.user.role !== "admin") {
    throw new UnauthorizedError("yout cannot update with this role, please try with ad
  }
```

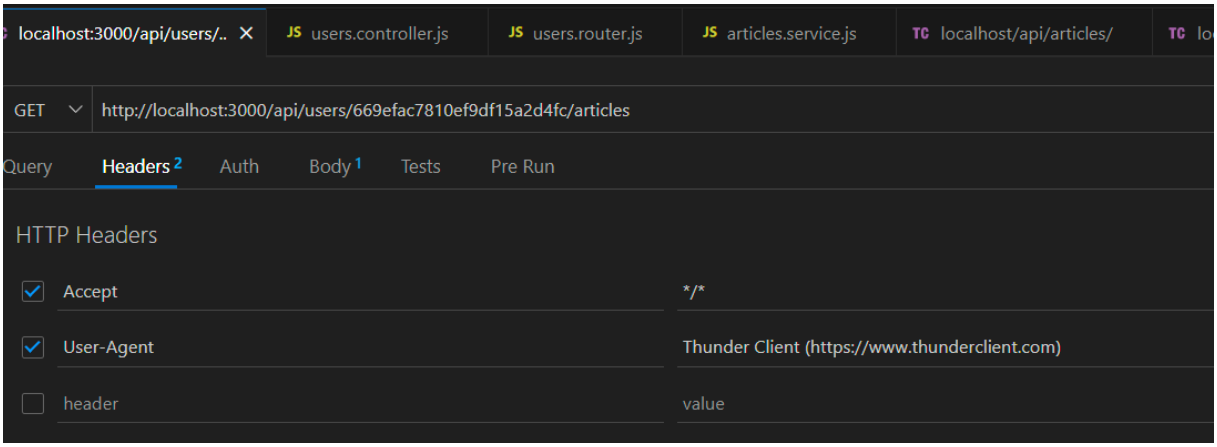
7. Lors de la création, faites un enregistrement en utilisant l'id de l'utilisateur connecté.

a) Modification du model article pour empêcher l'enregistrement sans le champ user

```
user: {
  type: Schema.Types.ObjectId,
  ref: "User",
  required: true,
},
```



4 .Créer le endpoint public pour afficher les articles d'un utilisateur. Le endpoint doit être sous la forme api/users/:userId/articles



Ajout de la route dans user.rooter

```
router.get('/:userId/articles', usersController.getUserArticles); // Route publique
```

Ajout dans user.controller

```
async getUserArticles(req, res, next) {
  try {
    const userId = req.params.userId;
    const articles = await articlesService.getArticlesByUserId(userId);
    req.io.emit('userArticlesFetched', { userId, articles });
    res.status(200).json(articles);
  } catch (error) {
    next(error);
  }
}
```

Ajout dans articles.service de la méthode

```
  getArticlesByUserId(userId) {
    return Article.find({ user: userId }).populate('user', '-password');
  }
}
```

5.Créer les tests unitaires du point n°2 (création, mise à jour , suppression)

```
const request = require('supertest');
const { app } = require('../server');
const jwt = require('jsonwebtoken');
const config = require('../config');
const mockingoose = require('mockingoose');
const Article = require('../api/articles/articles.schema');
const User = require('../api/users/users.model');

describe('Articles API', () => {
  let token;
  const USER_ID = 'fake';
  const ADMIN_ID = 'admin';
  const MOCK_USER = { _id: USER_ID, role: 'user' };
  const MOCK_ADMIN = { _id: ADMIN_ID, role: 'admin' };
  const MOCK_ARTICLE = { title: 'Test Article', content: 'Test Content',
userId: USER_ID };
  const MOCK_UPDATED_ARTICLE = { title: 'Updated Article', content: 'Updated
Content', userId: USER_ID };
  const MOCK_ARTICLES = [
    { title: 'Test Article', content: 'Test Content', userId: USER_ID },
    { title: 'Another Article', content: 'Another Content', userId: USER_ID }
  ];

  beforeEach(() => {
    token = jwt.sign({ userId: USER_ID }, config.secretJwtToken);
    mockingoose(User).toReturn(MOCK_USER, 'findOne');
    mockingoose(Article).toReturn(MOCK_ARTICLES, 'find');
  });

  test('Create Article', async () => {
    const res = await request(app)
      .post('/api/articles')
      .send(MOCK_ARTICLE)
      .set('x-access-token', token);
    expect(res.status).toBe(201);
    expect(res.body.title).toBe(MOCK_ARTICLE.title);
  });
```

```
  test('Update Article', async () => {
    token = jwt.sign({ userId: ADMIN_ID }, config.secretJwtToken);
    mockingoose(User).toReturn(MOCK_ADMIN, 'findOne');
    mockingoose(Article).toReturn(MOCK_UPDATED_ARTICLE, 'findOneAndUpdate');

    const res = await request(app)
      .put('/api/articles/1')
      .send(MOCK_UPDATED_ARTICLE)
      .set('x-access-token', token);
    expect(res.status).toBe(200);
    expect(res.body.title).toBe(MOCK_UPDATED_ARTICLE.title);
    expect(res.body.content).toBe(MOCK_UPDATED_ARTICLE.content);
  });

  test('Delete Article', async () => {
    token = jwt.sign({ userId: ADMIN_ID }, config.secretJwtToken);
    mockingoose(User).toReturn(MOCK_ADMIN, 'findOne');
    mockingoose(Article).toReturn(null, 'findOneAndDelete');

    const res = await request(app)
      .delete('/api/articles/1')
      .set('x-access-token', token);
    expect(res.status).toBe(204);


    const fetchRes = await request(app).get(`/api/articles/1`);
    expect(fetchRes.status).toBe(404);
  });

  afterEach(() => {
    jest.restoreAllMocks();
  });
});
```

6 Etablir une configuration de déploiement

Première étape installer pm2

npm install -g pm2

Ensuite configuration de ecosystem.config.js

```
module.exports = {
  apps: [
    {
      name: "app",
      script: "./www/app.js",
      env_production: {
        NODE_ENV: "production",
      },
      error_file: "./logs/err.log",
      max_memory_restart: "200M",
      instances: 3,
      exec_mode: "cluster",

    },
  ],
};
```

Et donner la commande la commande PM2 et lancer l'application avec 3 instances en parallèle

```
pm2 start ecosystem.config.js --env production -i 3
```