l'école d'ingénierie informatique

EPSI BORDEAUX - I1 EISI

Atelier - Développement et Service Cloud - Support Travaux Pratique - Projet

Code Module	Durée	Titre Diplôme	Bloc de Compétences	Promotion	Auteur
DEVE844	14h	EISI / RNCP 35584	Concevoir & Développer des solutions applicatives métiers	2023/2024	Julien COURAUD

1. Contexte du projet (en binôme)

Vous êtes une équipe d'architectes techniques et développeurs officiant au sein d'une entreprise de service en informatique.

Vous êtes affectés au projet de refonte du socle SERVEUR/BDD de l'application 'MFLIX' fournissant des informations cinématographiques en ligne (application fictive type 'Allociné').

Votre client stockait toutes ses données sur des serveurs physiques au sein de son centre de service. Pour des raisons de coût et de sécurité, il a décidé migrer ses bases de données dans le Cloud.

La partie Front-End de l'application 'MFLIX' reste inchangée et fonctionne actuellement selon un contrat d'API. Votre tâche sera donc de mettre en place un socle SERVEUR/BDD qui pourra communiquer avec son application et fournir des services via une API REST.

La base MongoDB migrée sur le cloud contient des informations sur les films, les utilisateurs de la plateforme ou encore les commentaires précédemment postés.

Le contrat d'API défini dans le cahier des charges du projet est le suivant:

- ***/movies*** - (GET) Récupérer tous les films
- ***/movie/:idMovie*** - (GET-POST-PUT-DELETE) Récupérer/Ajouter/Modifier/Supprimer un film via son ID
- ***/movie/comments*** - (GET) Récupérer la liste de tous les commentaires liés à un film
- ***/movie/comment/:idComment*** - (GET-POST-PUT-DELETE) Récupérer/Ajouter/Modifier/Supprimer un commentaire d'un film

Pour répondre à cette problématique votre équipe décide donc d'utiliser le framework Next.JS lié à la base de données MongoDB gérée depuis MongoDB Atlas.

2. Setup et architecture

2.1 Installation de NPM

Installez Node.js :

- Allez sur le site officiel de Node.js (https://nodejs.org/) et téléchargez la version correspondant à votre système d'exploitation.
- Lancez l'installeur et suivez les instructions à l'écran pour installer Node.js.

2.2 Setup du starter-kit Next.js "with-mongo"

Vous utiliserez le starter-kit "with-mongo" proposé par Vercel pour démarrer le projet avec une structure d'application toute faite pour connecter votre serveur à une base de données MongoDB: https://github.com/vercel/next.js/tree/canary/examples/with-mongodb

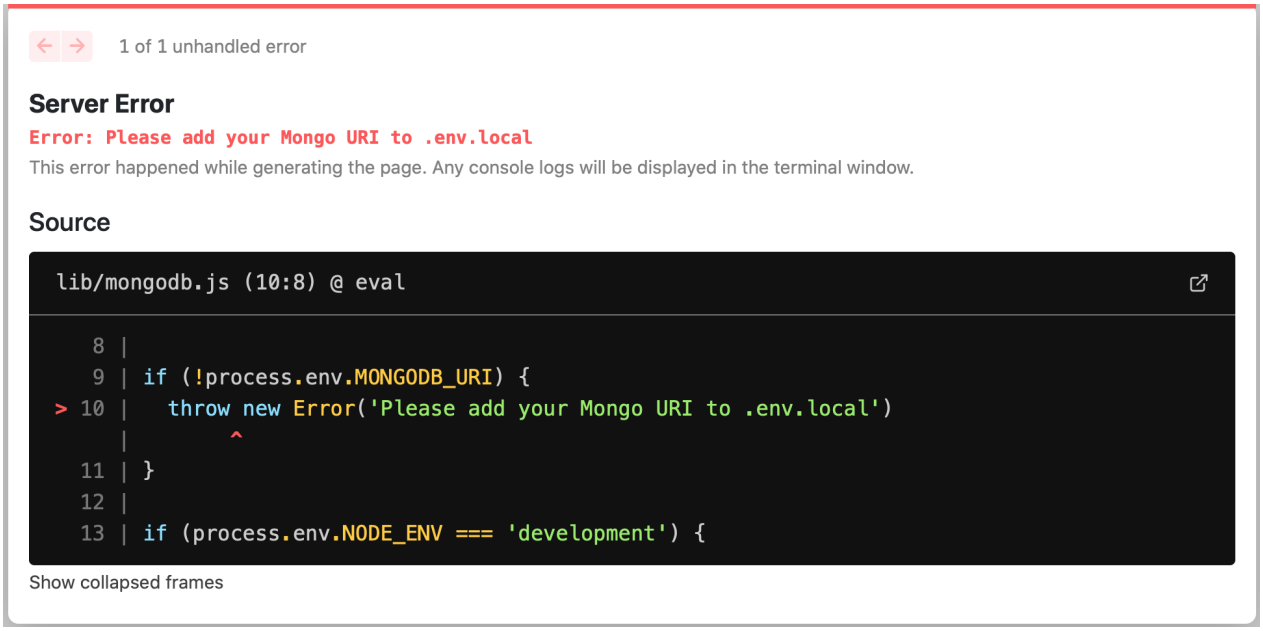
- Créer le projet:

```
npx create-next-app --example with-mongodb with-mongodb-app
```

- Installer et lancer le projet (http://localhost:3000):

```
npm install && npm run dev
```

Le projet se lance correctement, il reste cependant à configurer notre base de données MongoDB et la lier à la couche applicative côté serveur.



2.3 Setup MongoDB Atlas

Qu'est ce que MongoDB Atlas ?

MongoDB Atlas est un service de base de données cloud géré pour MongoDB, qui permet aux développeurs de déployer, de gérer et d'évoluer facilement leur base de données MongoDB dans le cloud. Cet outil est conçu pour être facile à utiliser et offre une gamme de fonctionnalités pour gérer efficacement votre base de données MongoDB dans le cloud, notamment :

- Déploiement facile : MongoDB Atlas simplifie le processus de déploiement de votre base de données MongoDB dans le cloud. Vous pouvez créer un cluster en quelques minutes à l'aide de l'interface utilisateur web ou de l'API.

- Gestion et surveillance de la base de données : MongoDB Atlas offre des fonctionnalités de gestion et de surveillance de votre base de données MongoDB, y compris la surveillance en temps réel, les alertes et les tableaux de bord de

performance.

- Sécurité avancée : MongoDB Atlas inclut des fonctionnalités de sécurité avancées, telles que l'authentification, l'autorisation, la mise en réseau et le chiffrement des données.
- Évolutivité horizontale : MongoDB Atlas facilite l'évolutivité horizontale de votre base de données MongoDB. Vous pouvez ajouter des nœuds à votre cluster pour gérer des charges de travail plus importantes et assurer une haute disponibilité.
- Disponibilité multi-région : MongoDB Atlas offre la possibilité de déployer votre base de données MongoDB dans plusieurs régions du monde, ce qui peut améliorer les performances et la disponibilité pour vos utilisateurs.

Setup de la base de données sur MongoDB Atlas

Sur l'outil Atlas, nous allons créer notre espace de stockage et le remplir avec des documents/collections/données automatiquement générés.

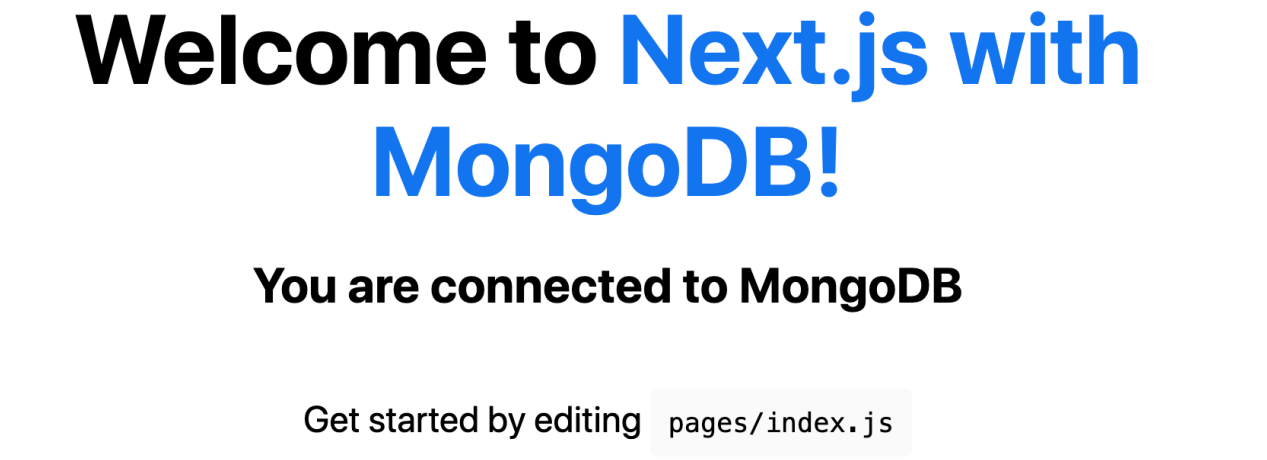
- Créer le cluster partagé (gratuit, 512Mb stockage)
- Génerer les databases/collections grâce à la fonctionnalité Atlas.
- Autoriser les adresses IP pour la connexion depuis la couche applicative.

2.4 Setup de la liaison entre l'application et la base de données

Fichier ".env.local" à la racine du projet Next.JS.

```
MONGODB_URI=mongodb+srv://<username>:<password>@<db-name>-<db-adress>/?retryWrites=true&w=majority
```

Vous utiliserez dans le cadre de ce projet la base de données générée: '***sample_mflix***'.



3. Mise en place de l'API interagissant avec la BDD Atlas Cloud

3.1 Ouvrir des endpoints REST

Premier endpoint

Afin de donner un accès aux données de votre base, vous allez ouvrir des 'endpoints' basés sur le pattern des API REST. Cela vous permettra d'avoir des points d'entrées sur votre serveur qui proposeront des actions spécifiques sur vos collections.

La base de données 'sample_mflix' propose plusieurs collections dont la collection 'movies' qui propose plus de 23000 documents (ressources) relatifs à des films.

Vous allez dans un premier temps ouvrir un endpoint '/api/movies' qui retourne l'ensemble de la collection au format JSON.

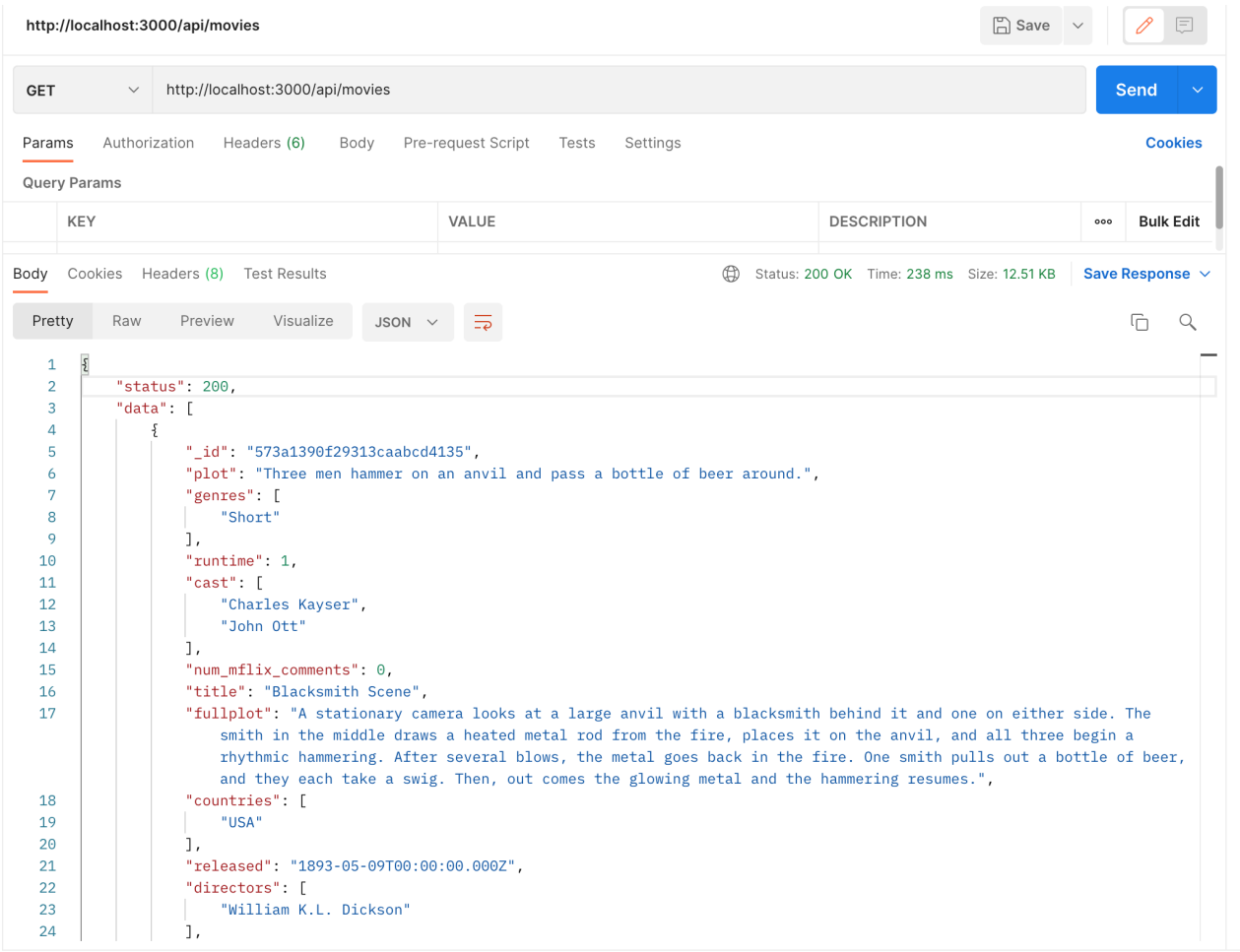
Avec le framework Next.JS, vous pouvez ouvrir des routes API en créant simplement un fichier dans un dossier 'api' dans le dossier 'pages'. Vous allez donc créer le fichier 'pages/api/movies.js' et ouvrir le endpoint avec le code suivant:

```
// pages/api/movies.js
import clientPromise from "../../lib/mongodb";

export default async function handler(req, res) {

  const client = await clientPromise;
  const db = client.db("sample_mflix");
  const movies = await db.collection("movies").find({}).limit(10).toArray();
  res.json({ status: 200, data: movies });
}
```

Vous pouvez alors tester votre endpoint, par exemple avec un outil de requêtage comme Postman, et observer les ressources renvoyées.



Exemple de filtrage en fonction de la méthode de requête

Une même route de endpoint peut être appelée avec une méthode de requête différente, on pourra notamment différencier ces appels à notre API avec les méthodes GET, POST, UPDATE, DELETE, etc..

Par exemple, vous devrez utiliser la route '/api/movie' avec la méthode GET pour récupérer les informations relative à un film mais également avec la méthode POST pour ajouter un film à la collection.

Pour cela vous pourrez utiliser un simple "switch/case":

```
switch (req.method) {

      case "POST":
      // .. Here the logic for POST case
      break;

    case "GET":
      // .. Here the logic for GET case
      break;

}
```

Dynamic API Routes

Il peut arriver que l'on ai besoin d'avoir des routes dynamiques, c'est à dire avec des paramètres d'URL. Prenons par exemple la récupération de la ressource relative à un film en particulier via son ID (qui pourrait être utilisé dans un contexte Front-end).

Pour cela il suffit de créer un fichier "[id].js" dans un dossier "movie" afin donc de générer la route "/api/movie/:id". Vous pouvez alors récupérer l'ID du film depuis l'URL du endpoint, ce qui vous permettra d'effectuer vos recherches dans la base de données et renvoyer la ressource.

```
// pages/api/movie/[id].js

export default function handler(req, res) {
  const { idMovie } = req.query
  const dbMovie = await db.collection("movies").findOne({ _id : idMovie }).toArray();
  res.json({ status: 200, data: {movie: dbMovie} });
}
```

Documentation

Toutes les méthodes Javascript d'accession à une base de données MongoDB sont détaillées dans cette documentation:

- https://www.mongodb.com/docs/manual/reference/method/js-collection/

3.2 Récapitulatif des endpoints à créer

- ***/movies*** - (GET) Récupérer tous les films
- ***/movie/:idMovie*** - (GET-POST-PUT-DELETE) Récupérer/Ajouter/Modifier/Supprimer un film via son ID
- ***/movie/comments*** - (GET) Récupérer la liste de tous les commentaires liés à un film
- ***/movie/comment/:idComment*** - (GET-POST-PUT-DELETE) Récupérer/Ajouter/Modifier/Supprimer un commentaire d'un film

NB: Les notions d'authentification à la plateforme sont volontairement laissés de côté pour le moment. Un ID utilisateur sera généré "en dur" dans le code pour les endpoints en ayant besoin.

4. Documentation Swagger

4.1 Qu'est ce qu'une documentation Swagger

La documentation Swagger est un ensemble de spécifications et d'outils qui permettent de décrire, de documenter et de tester les API RESTful. Elle peut être utilisée pour générer automatiquement des clients de service, des kits de développement logiciel et des tests de service.

Elle est devenue un standard pour la documentation d'API RESTful en raison de sa simplicité, de sa flexibilité et de sa prise en charge par une grande variété de langages de programmation et de frameworks.

4.2 SwaggerUI et React

Afin d'illustrer au mieux vos routes, vous proposerez une page type Swagger, servant de documentation en ligne de votre API REST.

Vous utiliserez les librairies "next-swagger-doc" et "swagger-ui-react".

```
// package.json
"next-swagger-doc": "^0.1.9",
"swagger-ui-react": "^3.52.3",
```

```
// pages/swagger/index.jsx

import Head from 'next/head';

import SwaggerUI from 'swagger-ui-react';
import 'swagger-ui-react/swagger-ui.css';

const Swagger = () => {

  return (
      <div>
        <Head>
          <title>BrowserStack Demo API</title>
          <meta name="description" content="BrowserStack Demo API Swagger" />
          <link rel="icon" href="/favicon.svg" />
        </Head>
        <SwaggerUI url="/api/doc" />
      </div>
  );
};

export default Swagger;
```

```
// pages/api/doc.js

import { withSwagger } from 'next-swagger-doc';

const swaggerHandler = withSwagger({
  openApiVersion: '3.0.0',
  title: 'BrowserStack Demo API',
  version: '1.0.0',
  apiFolder: 'pages/api',

});
export default swaggerHandler();
```

```
//pages/api/movies.js

/**
 * @swagger
 * /api/movies:
 *   get:
 *     description: Returns movies
```

```
 *     responses:
 *       200:
 *         description: Hello Movies
 */
export default async function handler(req, res) {
 //........
}
```

5. Bonus

Dans le cadre de ce projet, vous pouvez approfondir votre maitrise du framework Next.JS et aller plus loin dans la réalisation de votre application.
Vous pourrez par exemple:

- Créer de simples interfaces React permettant d'illustrer et consommer vos endpoints.
- Créer des endpoints autour de l'URL "/auth" afin de gérer l'authentification sur la plateforme et les sessions de connexion (lib "jsonwebtoken" et "bcryptjs")
- Proposer des tests unitaires afin d'améliorer la qualité globale de l'application.

6. Recapitulatif de ce qui est attendu

Vous proposerez comme rendu de ce projet, un repository Github public (ou privé avec invitation).

Il est attendu de vous au minimum dans ce projet:

- La définition et l'implémentation des endpoints demandés dans le cahier des charges (cf partie 3.2)
- Une interface SwaggerUI qui proposera une documentation de vos endpoints (cf partie 4.).