

Express JS



Les technologies

Express est un **framework** reposant sur **Node**, qui facilite la création et la gestion des serveurs Node. Coder des serveurs web en Node pur est possible, mais long et laborieux. En effet, cela exige d'analyser manuellement chaque demande entrante. L'utilisation du framework Express **simplifie ces tâches**, en nous permettant de déployer nos API beaucoup plus rapidement.

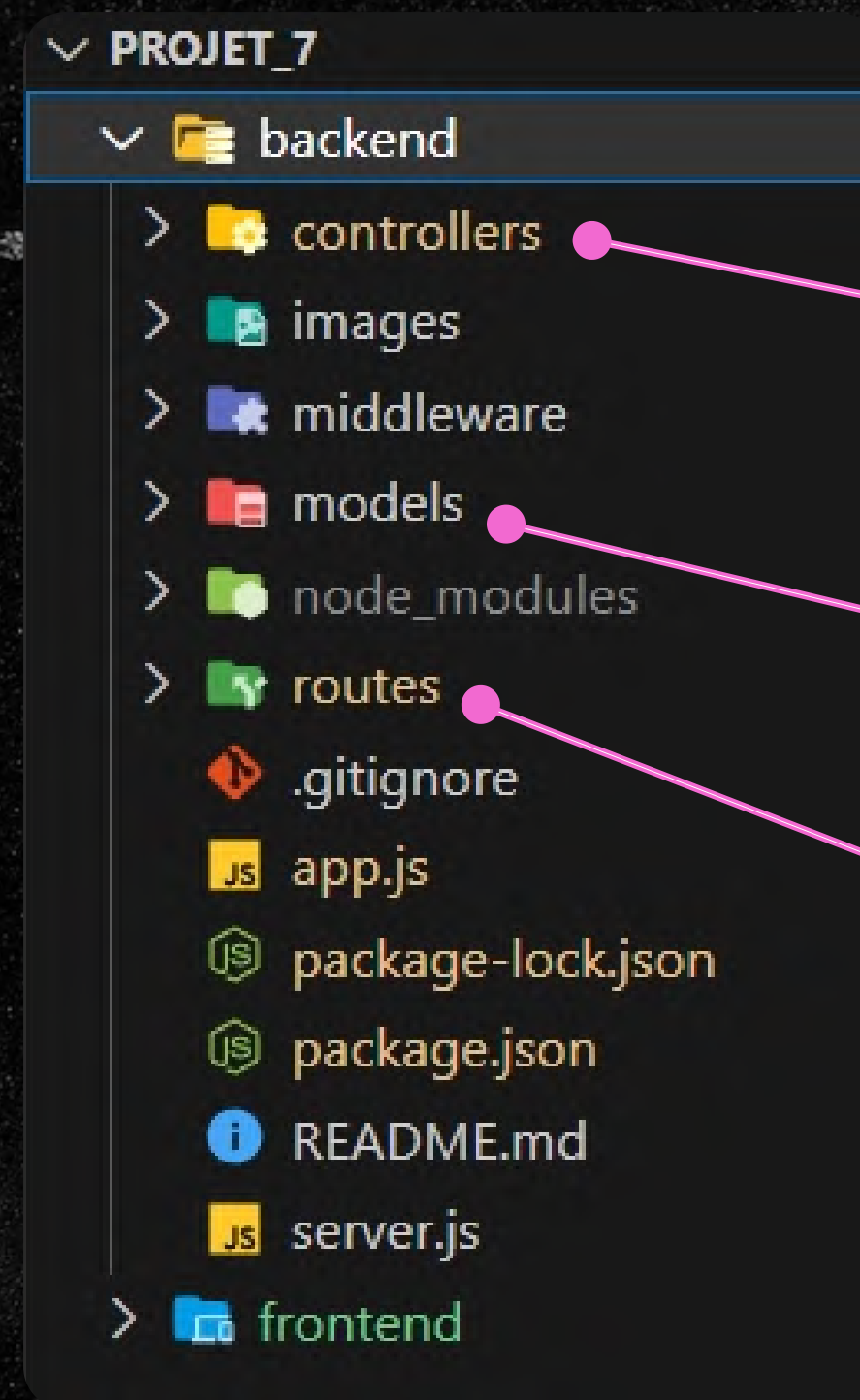
Node est le **runtime** qui permet d'écrire toutes nos tâches côté serveur, en JavaScript, telles que la logique métier, la persistance des données et la sécurité. Node ajoute également des fonctionnalités que le JavaScript du navigateur standard ne possède pas, comme par exemple l'accès au système de fichiers local.

Nodemon surveille les modifications de vos fichiers et redémarrera le serveur lorsqu'il aura besoin d'être mis à jour. Cela vous garantit d'avoir **toujours la dernière version de votre serveur dès que vous sauvegardez**, sans devoir relancer manuellement le serveur.

Mongoose est une bibliothèque de modélisation de données d'objet (ODM) basée sur js pour MongoDB. Il s'apparente à un ORM (Object Relational Mapper) tel que SQLAlchemy pour les bases de données SQL traditionnelles. Le problème que Mongoose vise à résoudre est de permettre aux développeurs d'appliquer **un schéma spécifique au niveau de la couche application**.

MongoDB (de l'anglais humongous qui peut être traduit par « énorme ») est un système de gestion de base de données orienté documents, réparti sur un nombre quelconque d'ordinateurs et ne nécessitant pas de schéma prédéfini des données. Il est écrit en C++. Le serveur et les outils sont distribués sous licence SSPL, les pilotes sous licence Apache et la documentation sous licence Creative Commons2. Il fait partie de la mouvance NoSQL.

Architecture du projet



MVC signifie Model, View, Controller est un modèle architectural qui sépare une application en trois composants logiques principaux : le modèle, la vue et le contrôleur. Chacun de ces composants est conçu pour gérer des aspects de développement spécifiques d'une application.

Le contrôleur est la partie qui s'occupe du traitement de la demande client qui gère la requête HTTP et renvoie une réponse. La réponse peut être soit un JSON si vous appelez un point de terminaison API, soit une page Web HTML standard.

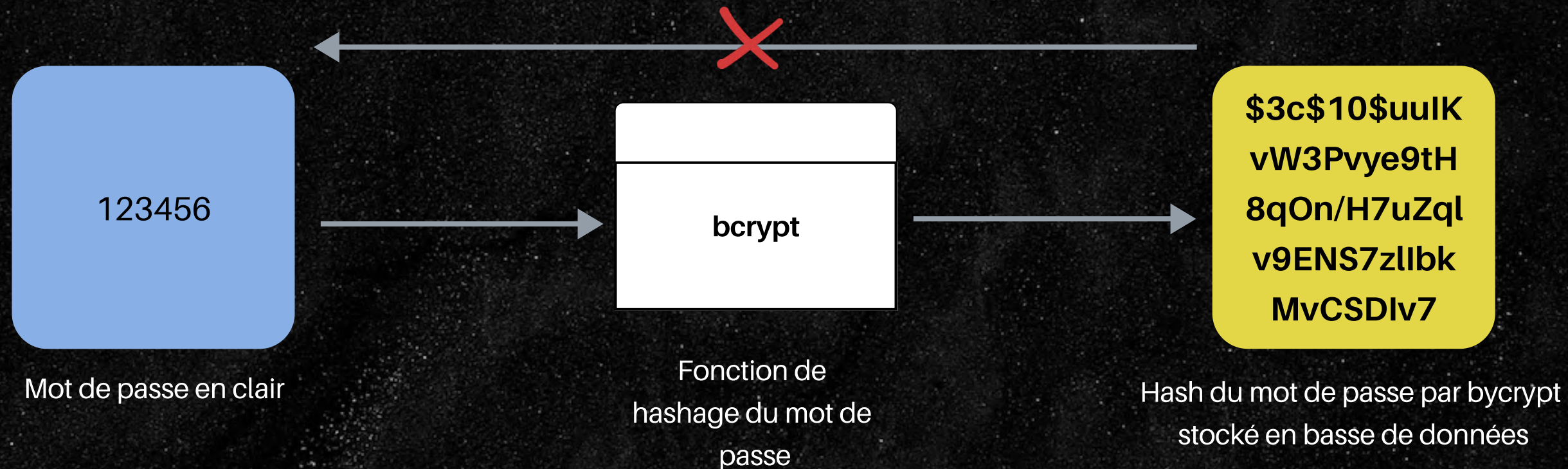
Le model est l'interface de base de données qui vous permet d'interagir avec l'API de base de données et de créer différents schémas d'entité de votre application sur la base de données (MySQL, MongoDB), il est appelé par le contrôleur en fonction de la demande du client.

La vue est ce que le client va dans la plupart des cas obtenir en réponse à ce qu'il a demandé (par exemple : pour afficher les détails de son profil). Ici côté backend la vue est représentée par nos routes car dans la cadre d'un API (mais "render()" des views est possible dans l'absolu avec un moteur de rendu -ou moteur de visualisation -eng view engine -tels Embedded Javascript (EJS), Pug, Mustache ...

Sécurité

1 Le mot de passe de l'utilisateur doit être haché : bcrypt

Le hachage d'un mot de passe est un processus **unidirectionnel** qui transforme un mot de passe en une chaîne de caractères **aléatoire**, généralement de longueur fixe. Cette opération est conçue de manière à ce qu'il soit extrêmement difficile, voire impossible, de reconstituer le mot de passe d'origine à partir du hachage.



Comment savoir si l'utilisateur a renseigné le bon mot de passe ?

On va comparer les deux mots de passe hashés (celui renseigné par l'utilisateur et celui en bdd) :

- Les deux hash seront différents mais le package bcrypt sait s'ils ont été produits par la même chaîne de caractère !



JWT

```
token: jwt.sign({ userId: user._id }, "RANDOM_TOKEN_SECRET", {  
  expiresIn: "24h",
```

Payload : Données que l'on veut encoder. Ici le userId
(on pourra l'utiliser par la suite par sécurité au lieu
d'utiliser ce que nous fournit le front)

Options: ici le temps de validité du token (mais il y en a
beaucoup d'autre)

Clef secrète pour l'encodage : ici une clef que l'on a
choisi nous-même et qui servira de base pour
l'encodage (et le décodage)

La méthode **sign()** de jwt créer un Jason WebToken pour l'utilisateur.
De nombreuses options peuvent être spécifiées comme le type d'algorithme :

- `algorithms` : List of strings with the names of the allowed algorithms. For instance, ["HS256" , "HS384"] .

If not specified a defaults will be used based on the type of key provided

- `secret` - ['HS256', 'HS384', 'HS512']
- `rsa` - ['RS256', 'RS384', 'RS512']
- `ec` - ['ES256', 'ES384', 'ES512']
- `default` - ['RS256', 'RS384', 'RS512']

- Les JSON web tokens sont des tokens chiffrés qui peuvent être utilisés pour l'autorisation.
- La méthode `sign()` du package `jsonwebtoken` utilise une clé secrète pour chiffrer un token qui peut contenir un payload personnalisé et avoir une validité limitée.

JWT

```
router.post("/:id/rating", auth, bookController.addBookRating);
```

backend > middleware > `auth.js` > ...

```
1  const jwt = require("jsonwebtoken");
2
3  module.exports = (req, res, next) => {
4    try {
5      const token = req.headers.authorization.split(" ")[1];
6      const decodedToken = jwt.verify(token, "RANDOM TOKEN SECRET");
7      const userId = decodedToken.userId;
8      req.auth = {
9        userId: userId,
10      };
11      next();
12    } catch (error) {
13      res.status(401).json({ error });
14    }
15  };
```

- La méthode `verify()` du package `jsonwebtoken` permet de vérifier la validité d'un token (sur une requête entrante, par exemple).

```
decodedToken {
  userId: '646e20d65177f64cf2c361db',
  iat: 1685276773,
  exp: 1685363173
}
```




Sécurité

```
const mongoose = require("mongoose");
const uniqueValidator = require("mongoose-unique-validator");

const userSchema = mongoose.Schema({
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
});

userSchema.plugin(uniqueValidator);

module.exports = mongoose.model("User", userSchema);
```

model User

bcrypt

Que fait ce plugin ?

mongoose-unique-validator est un plugin qui ajoute une validation de pré-enregistrement pour les champs uniques dans un schéma Mongoose. Cela facilite grandement la gestion des erreurs, car vous obtiendrez une erreur de validation Mongoose lorsque vous tenterez de violer une contrainte unique, plutôt qu'une erreur E11000 de MongoDB.

npmjs.com/package/mongoose-unique-validator

Pourquoi utiliser un plugin ? Notre schéma contient bien "unique : true" ?

"unique" est géré au niveau MongoDB pas au niveau de Mongoose.

Pour comprendre cela, vous devez comprendre ce qui se passerait si nous nous contentions de "unique : true" dans le schéma.

La manière dont Mongoose implémente cette validation consiste à ajouter un index unique dans mon attribut email dans le schéma utilisateur.

UNIQUEMENT APRÈS cela, MongoDB se comportera comme nous le souhaitons : lancer une erreur si nous essayons d'insérer deux utilisateurs avec des e-mails en double.

Ensuite, Mongoose détectera l'erreur et nous la renverra. Notez que la construction de l'index prendra un certain temps. Et comme il est asynchrone, avant que l'index ne soit créé avec succès, il ne vous empêchera **PAS** de créer des données en double. Bien sûr, il fonctionnera correctement après sa construction.

Source : [Do you know the `unique` option is not a validator in Mongoose?](https://stackoverflow.com/questions/47587537/mongoose-unique-validator-vs-mongoose-unique-validator)

Sécurité

1 Le mot de passe de l'utilisateur doit être haché : bcrypt

```
exports.signup = (req, res, next) => {  
  bcrypt  
    .hash(req.body.password, 10)  
    .then((hash) => {  
      const user = new User({  
        email: req.body.email,  
        password: hash,  
      });  
      user  
        .save()  
        .then(() => {  
          res  
            .status(201)  
            .json({ message: "L'utilisateur a bien été enregistré" })  
        })  
        .catch((error) => res.status(400).json({ error }));  
    })  
    .catch((error) => res.status(500).json({ error }));  
};
```

user controller / signup



Le cas d'utilisation typique de ce module Node.js haute vitesse consiste à :

- **Convertir des images volumineuses** dans des formats courants en images JPEG, PNG, WebP, GIF et AVIF plus petites et adaptées au Web, de dimensions variables.
- **Redimensionner une image.** Le redimensionnement d'une image est généralement 4 à 5 fois plus rapide que l'utilisation des paramètres ImageMagick et GraphicsMagick les plus rapides en raison de son utilisation de libvips
- La rotation, l'extraction, la composition et la correction gamma sont disponibles.

```
const sharp = require("sharp");

module.exports = async function process(req, res, next) {
  if (req.file) {
    const nameWithoutSpaces = req.file.originalname
      .split(" ")
      .join("_")
      .toString()
      .split(".");
    filename = nameWithoutSpaces[0] + Date.now() + ".webp";
    const path = `../backend/images/${filename}`;

    await sharp(req.file.buffer)
      .resize({
        width: 200,
        height: 200,
        fit: sharp.fit.contain,
        background: { r: 242, g: 227, b: 206, alpha: 0.5 },
      })
      .toFormat("webp")
      .toFile(path);
    res.locals.filename = filename;
    next();
  } else {
    next();
  }
};
```


Sécurité

1 Le mot de passe de l'utilisateur doit être haché : bcrypt

toto

Mot de passe renseigné par l'utilisateur

\$3c\$10\$uulKvW3Pvye9tH8qOn/H7uZqlv9ENS7zllbkMvCSDlv7

Hash du mot de passe par bcrypt
stocké en basse de données

\$3c\$10\$uulKvW3Pvye9tH8qOn/H7uZqlv9ENS7zllbkMvCSDlv7

Hash du mot de passe par bcrypt stocké en basse de données

\$3c\$10\$uulKvW3Pvye9tH8qOn/H7uZqlv9ENS7zllbkMvCSDlv7

Version de l'algorithme de bcrypt utilisée

\$3c\$10\$uulKvW3Pvye9tH8qOn/H7uZqlv9ENS7zllbkMvCSDlv7

Nombre d'itération ou *cost factor*

```
bcrypt  
.hash(req.body.password, 10)
```

\$3c\$10\$uulKvW3Pvye9tH8qOn/H7uZqlv9ENS7zllbkMvCSDlv7

Le sel de hashage

\$3c\$10\$uulKvW3Pvye9tH8qOn/H7uZqlv9ENS7zllbkMvCSDlv7

Le mot de passe hashé

Sécurité

1 Le mot de passe de l'utilisateur doit être haché : bcrypt

```
exports.login = (req, res, next) => {
  User.findOne({ email: req.body.email })
    .then((user) => {
      if (user === null) {
        res.status(401).json({ message: "Paire id/mdp invalide" });
      } else {
        bcrypt
          .compare(req.body.password, user.password)
          .then((valid) => {
            if (!valid) {
              res.status(401).json({ message: "Paire id/mdp invalide" });
            } else {
              res.status(200).json({
                userId: user._id,
                token: jwt.sign({ userId: user._id }, "RANDOM_TOKEN_SECRET", {
                  expiresIn: "24h",
                }),
              });
            }
          })
          .catch((error) => {
            res.status(500).json({ error });
          });
      }
    })
    .catch((error) => {
      res.status(500).json({ error });
    });
};
```

usercontroller/login

bcrypt

`.compare(req.body.password, user.password)`

toto

\$3c\$10\$uulKvW
3Pvye9tH8qOn/
H7uZqlv9ENS7zl
IbkMvCSDlv7

Sécurité

1 Le mot de passe de l'utilisateur doit être haché : bcrypt

1 Elle extrait le sel associé au hachage stocké

\$3c\$10\$uulKvW3Pvye9tH8qOn/H7uZqlv9ENS7zllbkMvCSDlv7

2 Elle applique le même processus de hachage au mot de passe fourni en utilisant le sel extrait et les paramètres de hachage associés (comme le facteur de coût et l'algorithme de hachage adaptatif de Bcrypt).

\$3c\$10\$uulKvW3Pvye9tH8qOn/H7uZqlv9ENS7zllbkMvCSDlv7

toto

```
EksBlowfishSetup(cost, salt, key)
state ← InitState()
state ← ExpandKey(state, salt, key)
repeat (2cost)
  state ← ExpandKey(state, 0, key)
  state ← ExpandKey(state, 0, salt)
return state
```

```
EksBlowfishSetup(cost, salt, key)
state ← InitState()
state ← ExpandKey(state, salt, key)
repeat (2cost)
  state ← ExpandKey(state, 0, key)
  state ← ExpandKey(state, 0, salt)
return state
```


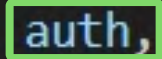
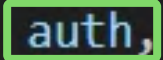
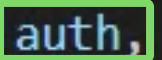
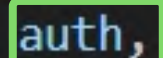
true

false



Sécurité

2 L'authentification doit être renforcée sur toutes les routes livre (book) requises

```
backend > routes >  book.js > ...  
1  const express = require("express");  
2  const auth = require("../middleware/auth");  
3  const bookController = require("../controllers/book");  
4  const multer = require("../middleware/multer-config");  
5  const sharp = require("../middleware/sharp-config");  
6  const router = express.Router();  
7  
8  router.get("/bestrating", bookController.getBooksWithBestRating);  
9  router.get("/", bookController.getAllBooks);  
10 router.post("/:id/rating", , bookController.addBookRating);  
11 router.get("/:id", bookController.getBookById);  
12 router.post("/", , multer, sharp, bookController.createNewBook);  
13 router.put("/:id", , multer, sharp, bookController.updateBookById);  
14 router.delete("/:id", , bookController.deleteBookById);  
15  
16 module.exports = router;  
17
```


Sécurité

2 L'authentification doit être renforcée sur toutes les routes livre (book) requises

```
exports.login = (req, res, next) => {
  User.findOne({ email: req.body.email })
    .then((user) => {
      if (user === null) {
        res.status(401).json({ message: "Paire id/mdp invalide" });
      } else {
        bcrypt
          .compare(req.body.password, user.password)
          .then((valid) => {
            if (!valid) {
              res.status(401).json({ message: "Paire id/mdp invalide" });
            } else {
              res.status(200).json({
                userId: user._id,
                token: jwt.sign({ userId: user._id }, "RANDOM_TOKEN_SECRET", {
                  expiresIn: "24h",
                }),
              });
            }
          })
      }
    })
    .catch((error) => {
      res.status(500).json({ error });
    });
}

.catch((error) => {
  res.status(500).json({ error });
});
};
```

user controller / login

Sécurité

2 L'authentification doit être renforcée sur toutes les routes livre (book) requises

```
token: jwt.sign({ userId: user._id }, "RANDOM_TOKEN_SECRET", {  
  expiresIn: "24h",
```

Payload : Données que l'on veut encoder. Ici le userId
(on pourra utiliser par la suite par sécurité au lieu
d'utiliser ce que nous fournit le front)

Options: ici le temps de validité du token (mais il y en a
beaucoup d'autre)

Clef secrète pour l'encodage : ici une clef que l'on a
choisi nous-même et qui servira de base pour
l'encodage (et le décodage)

La méthode **sign()** de jwt créer un Jason WebToken pour l'utilisateur.
De nombreuses options peuvent être spécifiées comme le type d'algorithme :

- `algorithms` : List of strings with the names of the allowed algorithms. For instance, `["HS256", "HS384"]`.

If not specified a defaults will be used based on the type of key provided

- `secret` - ['HS256', 'HS384', 'HS512']
- `rsa` - ['RS256', 'RS384', 'RS512']
- `ec` - ['ES256', 'ES384', 'ES512']
- `default` - ['RS256', 'RS384', 'RS512']

- Les JSON web tokens sont des tokens chiffrés qui peuvent être utilisés pour l'autorisation.
- La méthode `sign()` du package `jsonwebtoken` utilise une clé secrète pour chiffrer un token qui peut contenir un payload personnalisé et avoir une validité limitée.

Sécurité

2 L'authentification doit être renforcée sur toutes les routes livre (book) requises

```
router.post("/:id/rating", auth, bookController.addBookRating);
```

backend > middleware > `auth.js` > ...

```
1  const jwt = require("jsonwebtoken");
2
3  module.exports = (req, res, next) => {
4    try {
5      const token = req.headers.authorization.split(" ")[1];
6      const decodedToken = jwt.verify(token, "RANDOM TOKEN SECRET");
7      const userId = decodedToken.userId;
8      req.auth = {
9        userId: userId,
10      };
11      next();
12    } catch (error) {
13      res.status(401).json({ error });
14    }
15  };
```

- La méthode `verify()` du package `jsonwebtoken` permet de vérifier la validité d'un token (sur une requête entrante, par exemple).

```
decodedToken {
  userId: '646e20d65177f64cf2c361db',
  iat: 1685276773,
  exp: 1685363173
}
```




Sécurité

- 3 Les adresses électroniques dans la base de données sont uniques, et un plugin Mongoose approprié est utilisé pour garantir leur unicité et signaler les erreurs : Mongoose Unique Validator

```
const mongoose = require("mongoose");
const uniqueValidator = require("mongoose-unique-validator");

const userSchema = mongoose.Schema({
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
});

userSchema.plugin(uniqueValidator);

module.exports = mongoose.model("User", userSchema);
```

model User

bcrypt

Que fait ce plugin ?

mongoose-unique-validator est un plugin qui ajoute une validation de pré-enregistrement pour les champs uniques dans un schéma Mongoose. Cela facilite grandement la gestion des erreurs, car vous obtiendrez une erreur de validation Mongoose lorsque vous tenterez de violer une contrainte unique, plutôt qu'une erreur E11000 de MongoDB.

npmjs.com/package/mongoose-unique-validator

Pourquoi utiliser un plugin ? Notre schéma contient bien "unique : true" ?

"unique" est géré au niveau MongoDB pas au niveau de Mongoose.

Pour comprendre cela, vous devez comprendre ce qui se passerait si nous nous contentions de "unique : true" dans le schéma.

La manière dont Mongoose implémente cette validation consiste à ajouter un index unique dans mon attribut email dans le schéma utilisateur.

UNIQUEMENT APRÈS cela, MongoDB se comportera comme nous le souhaitons : lancer une erreur si nous essayons d'insérer deux utilisateurs avec des e-mails en double.

Ensuite, Mongoose détectera l'erreur et nous la renverra. Notez que la construction de l'index prendra un certain temps. Et comme il est asynchrone, avant que l'index ne soit créé avec succès, il ne vous empêchera **PAS** de créer des données en double. Bien sûr, il fonctionnera correctement après sa construction.

Source : [Do you know the `unique` option is not a validator in Mongoose?](https://stackoverflow.com/questions/47209087/mongoose-unique-validator-does-not-work)

Sécurité

- 4 La sécurité de la base de données MongoDB (à partir d'un service tel que MongoDB Atlas) ne doit pas empêcher l'application de se lancer sur la machine d'un utilisateur.

Vous devrez également accéder à l'onglet **Network Access**, cliquer sur *Add IP Adress* et autoriser l'accès depuis n'importe où (*Add access from Anywhere*).

Configuration de MonDB Atlas

- 5 Les erreurs issues de la base de données doivent être remontées.

(voir code directement pour la gestion des erreurs)

Green Coding

C'est quoi ?

Le "green coding" dans le développement web se réfère à l'adoption de pratiques de programmation et de conception écologiquement responsables. L'objectif est de réduire l'impact environnemental des sites web et des applications en optimisant leur efficacité énergétique et en minimisant leur empreinte carbone.

Quelques principes clés (back-end)

- **Optimisation de la gestion des ressources** : Assurez-vous d'optimiser l'utilisation des ressources serveur, telles que la mémoire et le processeur. Évitez les requêtes inutiles, optimisez les requêtes de base de données et utilisez des caches pour réduire les opérations coûteuses en énergie.
- **Éviter les opérations coûteuses en énergie** : Identifiez les parties du code qui nécessitent beaucoup de puissance de calcul et d'énergie, comme les boucles intensives ou les algorithmes inefficaces. Optimisez ces parties du code pour réduire leur impact énergétique.



- **Choix d'un hébergeur écologique** : Sélectionnez des services d'hébergement qui utilisent des sources d'énergie renouvelable ou qui compensent leur empreinte carbone. Optez pour des centres de données alimentés par des énergies propres afin de réduire l'impact environnemental global de votre infrastructure.



Le cas d'utilisation typique de ce module Node.js haute vitesse consiste à :

- **Convertir des images volumineuses** dans des formats courants en images JPEG, PNG, WebP, GIF et AVIF plus petites et adaptées au Web, de dimensions variables.
- **Redimensionner une image.** Le redimensionnement d'une image est généralement 4 à 5 fois plus rapide que l'utilisation des paramètres ImageMagick et GraphicsMagick les plus rapides en raison de son utilisation de libvips
- La rotation, l'extraction, la composition et la correction gamma sont disponibles.

```
const sharp = require("sharp");

module.exports = async function process(req, res, next) {
  if (req.file) {
    const nameWithoutSpaces = req.file.originalname
      .split(" ")
      .join("_")
      .toString()
      .split(".");
    filename = nameWithoutSpaces[0] + Date.now() + ".webp";
    const path = `../backend/images/${filename}`;

    await sharp(req.file.buffer)
      .resize({
        width: 200,
        height: 200,
        fit: sharp.fit.contain,
        background: { r: 242, g: 227, b: 206, alpha: 0.5 },
      })
      .toFormat("webp")
      .toFile(path);
    res.locals.filename = filename;
    next();
  } else {
    next();
  }
};
```


RESTful ?

- C'est quoi ?

REST signifie **Representational State Transfer** (ou *transfert d'état de représentation*, en français), et constitue un ensemble de **normes**, ou de lignes directrices **architecturales** qui structurent la façon de communiquer les données entre votre application et le reste du monde, ou entre différents composants de votre application.

- Règles à suivre

Il y a **six** lignes directrices architecturales clés pour les API REST :

#1 : Client-serveur separation

#2 : Stateless

- le serveur ne sauvegarde aucune des requêtes ou réponses précédentes.

#3 : Cacheable (ou *sauvegardable*, en français)

La réponse doit contenir l'information sur la capacité ou non du client de mettre les données **en cache**, ou de les sauvegarder. Si les données **peuvent être mises en cache**, la réponse doit être accompagnée d'un numéro de version. Ainsi, si votre utilisateur formule deux fois la même requête (c'est-à-dire s'il veut revoir une page) et que les informations n'ont pas changé, alors votre serveur n'a pas besoin de rechercher les informations une deuxième fois. À la place, le client peut simplement mettre en cache les données la première fois, puis charger à nouveau les mêmes données la seconde fois.

#4 : Uniforme Interface (interface uniforme)

Une API REST d'une application peut communiquer *de la même façon* avec une autre application entièrement différente.

#5 : Layered system (système de couches)

Chaque composant qui utilise REST n'a pas accès aux composants au-delà du composant précis avec lequel il interagit.

#6 : Code on demand (code à la demande)

Le code à la demande signifie que le serveur peut étendre sa fonctionnalité en envoyant le code au client pour téléchargement. C'est facultatif, car tous les clients ne seront pas capables de télécharger et d'exécuter le même code – donc ce n'est pas utilisé habituellement, mais au moins, vous savez que ça existe !

- Toutes les API ne sont pas RESTful et les API REST ont des lignes directrices architecturales spécifiques.
- Les avantages clés des API REST sont les suivants :
 - la séparation du client et du serveur, qui aide à scaler plus facilement les applications ;
 - le fait d'être stateless, ce qui rend les requêtes API très spécifiques et orientées vers le détail ;
 - la possibilité de mise en cache, qui permet aux clients de sauvegarder les données, et donc de ne pas devoir constamment faire des requêtes aux serveurs.
- SOAP est un autre type d'API, mais est plus utilisé dans les grandes entreprises.

CORS



CORS (Cross-Origin Resource Sharing) est un mécanisme de sécurité utilisé par les navigateurs web pour contrôler les demandes d'accès aux ressources d'un domaine (origine) à partir d'un autre domaine. Il s'agit d'une politique de sécurité du même origine qui empêche les requêtes entre différents domaines, sauf si ces requêtes sont explicitement autorisées.

Lorsqu'un navigateur effectue une requête HTTP vers un domaine, il envoie automatiquement l'origine de la requête en tant qu'en-tête HTTP "Origin". Le serveur du domaine peut alors décider d'autoriser ou de refuser la requête en fonction de cette origine.

Les requêtes qui ne sont pas du même domaine sont appelées requêtes cross-origin. Par défaut, les navigateurs bloquent les réponses des requêtes cross-origin pour des raisons de sécurité, afin d'éviter que des scripts malveillants sur un site ne récupèrent des informations sensibles à partir d'autres sites sans autorisation.

1. L'en-tête "Access-Control-Allow-Origin" spécifie les origines (domaines) autorisées à accéder à la ressource. Par exemple, si un serveur renvoie "Access-Control-Allow-Origin: <https://www.example.com>", cela signifie que seule cette origine est autorisée à accéder à la ressource.
2. L'en-tête "Access-Control-Allow-Methods" spécifie les méthodes HTTP autorisées pour la ressource.
3. L'en-tête "Access-Control-Allow-Headers" spécifie les en-têtes HTTP personnalisés autorisés dans la requête.
4. L'en-tête "Access-Control-Allow-Credentials" indique si les informations d'authentification (cookies, certificats client, etc.) peuvent être envoyées avec la requête.

Middleware

Dans le contexte de Node.js, un middleware est une fonction intermédiaire qui s'exécute entre la réception d'une requête HTTP et l'envoi d'une réponse correspondante. Les middlewares sont une composante fondamentale des frameworks web tels que Express.js, qui les utilise pour gérer les requêtes de manière modulaire et réutilisable.

Un middleware peut effectuer les actions suivantes :

1. Effectuer des opérations sur la requête : Le middleware peut analyser les données de la requête, valider les paramètres, extraire des informations spécifiques, etc. Cela permet de préparer les données pour le traitement ultérieur.
2. Modifier la réponse : Le middleware peut ajouter ou modifier des en-têtes de réponse, formater le corps de la réponse ou effectuer d'autres modifications avant que la réponse ne soit renvoyée au client.
3. Exécuter du code supplémentaire : Le middleware peut effectuer des opérations supplémentaires, telles que la gestion de sessions, l'authentification, la journalisation, la compression des réponses, la gestion des erreurs, etc. Il peut également appeler d'autres middlewares pour continuer le flux de traitement de la requête.

Les middlewares sont organisés en une pile (ou pipeline) dans l'ordre dans lequel ils sont déclarés. Lorsqu'une requête est reçue, elle passe à travers chaque middleware de la pile. Chaque middleware a accès à l'objet de requête (request object), à l'objet de réponse (response object) et à une fonction `next()` qui lui permet de passer la main au middleware suivant. Cela permet de décomposer la logique de traitement des requêtes en plusieurs étapes, simplifiant ainsi le développement et la gestion des applications web.

```
router.post("/", auth, multer, sharp, bookController.createNewBook);
```



Les middlewares offrent une grande flexibilité et permettent de construire des applications web modulaires et extensibles en ajoutant ou en supprimant facilement des fonctionnalités à la demande.



Pitch

Want to make a presentation like this one?

Start with a fully customizable template, create a beautiful deck in minutes, then easily share it with anyone.

Create a presentation (It's free)

