TP : API "DealExpress" - Plateforme de Bons Plans

Durée : 5 heures • Framework : Express.js • Mode : Individuel

Contexte

Vous allez développer DealExpress, une API REST complète inspirée de Dealabs permettant aux utilisateurs de partager des bons plans, de les voter (hot/cold) et de les commenter. Ce projet intègre un système de modération avec gestion de rôles.

Stack Technique

Express.js • MongoDB (Mongoose) • JWT • express-validator • bcryptjs • Postman

Modèles de Données

User : username, email, password (hashé), role (user/moderator/admin), createdAt

Deal : title, description, price, originalPrice, url, category, status (pending/approved/rejected), temperature (calculé), authorId, createdAt

Comment : content, dealId, authorId, createdAt

Vote : type (hot/cold), userId, dealId, createdAt

Fonctionnalités à Implémenter

Phase 1 - Authentification et Rôles (1h)

Créez les endpoints suivants pour gérer l'authentification des utilisateurs. Le système doit supporter trois niveaux d'accès : user (utilisateur standard), moderator (modérateur) et admin (administrateur).

Endpoints :

- POST /api/auth/register : inscription avec role "user" par défaut
- POST /api/auth/login : connexion et génération du token JWT (inclure le role dans le payload)
- GET /api/auth/me : récupération du profil de l'utilisateur connecté

Middlewares à créer :

- authenticate : extraction et vérification du token JWT, attachement de req.user
- requireRole(roles) : vérification que l'utilisateur possède le rôle requis

Validation :

- Email au format valide, mot de passe minimum 8 caractères
- Username unique, entre 3 et 30 caractères, alphanumérique uniquement

Phase 2 - CRUD Deals avec Permissions (1h30)

Les deals sont au cœur de l'application. Implémentez un système complet de gestion avec différents niveaux d'accès selon les rôles.

Endpoints :

- GET /api/deals : liste publique des deals approuvés avec pagination

- GET /api/deals/search : recherche de deals par mots-clés dans title et description
- GET /api/deals/:id : détails complets d'un deal avec auteur, commentaires et nombre de votes
- POST /api/deals : création d'un deal (authentification requise, status "pending")
- PUT /api/deals/:id : modification d'un deal (ownership requis, uniquement si status "pending")
- DELETE /api/deals/:id : suppression d'un deal (ownership requis ou admin)

Logique métier importante :

Les utilisateurs standards créent des deals avec le status "pending". Seuls les deals ayant le status "approved" apparaissent dans la route publique GET /api/deals. Un utilisateur peut modifier son deal uniquement si celui-ci est encore en attente de modération. Les administrateurs peuvent supprimer n'importe quel deal. Pensez à populate l'auteur lors des GET, mais excluez toujours le champ password.

Validation des données :

- Title : entre 5 et 100 caractères
- Description : entre 10 et 500 caractères
- Price : nombre positif ou zéro
- Category : valeurs autorisées ["High-Tech", "Maison", "Mode", "Loisirs", "Autre"]

Recherche (GET /api/deals/search) :

Implémentez une recherche qui accepte un paramètre query "q" et retourne les deals dont le titre ou la description contient le terme recherché. Utilisez une regex insensible à la casse. Exemple : /api/deals/search?q=iphone

Phase 3 - Système de Votes (45 min)

Le système de votes permet aux utilisateurs d'indiquer si un deal est intéressant (hot) ou non (cold). La température d'un deal correspond au nombre de votes hot moins le nombre de votes cold.

Endpoints :

- POST /api/deals/:id/vote : voter hot ou cold pour un deal
- DELETE /api/deals/:id/vote : retirer son vote d'un deal

Logique métier :

Un utilisateur ne peut voter qu'une seule fois par deal. Si un utilisateur vote à nouveau, son vote précédent doit être remplacé (pas de doublon dans la base). Calculez la température en comptant : (votes hot) - (votes cold). Retournez la température mise à jour après chaque action de vote.

Format du body pour POST /api/deals/:id/vote :

```
{
  "type": "hot"
}
```

ou

```
{
  "type": "cold"
}
```

Phase 4 - Commentaires (45 min)

Les utilisateurs peuvent commenter les deals pour donner leur avis ou poser des questions. Chaque utilisateur gère ses propres commentaires.

Endpoints :

- GET /api/deals/:dealId/comments : liste des commentaires d'un deal avec auteurs
- POST /api/deals/:dealId/comments : ajout d'un commentaire (authentification requise)
- PUT /api/comments/:id : modification de son propre commentaire (ownership requis)
- DELETE /api/comments/:id : suppression de son commentaire (ownership requis ou admin)

Logique métier :

Populate systématiquement l'auteur de chaque commentaire (username uniquement). Triez les commentaires par date de création, du plus récent au plus ancien. Les administrateurs peuvent supprimer n'importe quel commentaire.

Validation :

- Content : minimum 3 caractères, maximum 500 caractères

Phase 5 - Modération et Administration (1h)

Les modérateurs et administrateurs ont accès à des fonctionnalités spéciales pour gérer la plateforme et les utilisateurs.

Endpoints de modération :

- GET /api/admin/deals/pending : liste de tous les deals en attente de validation (moderator/admin)
- PATCH /api/admin/deals/:id/moderate : approuver ou rejeter un deal (moderator/admin)

Format du body pour modérer un deal :

```
{
  "status": "approved"
}
```

ou

```
{
  "status": "rejected"
}
```

Endpoints de gestion des utilisateurs (admin uniquement) :

- GET /api/admin/users : liste de tous les utilisateurs avec pagination
- PATCH /api/admin/users/:id/role : modification du rôle d'un utilisateur

Format du body pour changer le rôle :

```
{
  "role": "moderator"
}
```

Valeurs possibles : "user", "moderator", "admin"

Logique métier :

Seuls les modérateurs et administrateurs peuvent accéder aux routes de modération. Un deal approuvé devient visible dans GET /api/deals. Un deal rejeté reste invisible des utilisateurs standards mais reste accessible aux modérateurs. Seuls les administrateurs peuvent modifier les rôles des utilisateurs et accéder à la liste complète des utilisateurs.

Matrice de Permissions

Action	User	Moderator	Admin
Créer deal	Oui (pending)	Oui (pending)	Oui (pending)
Modifier deal	Son deal (si pending)	Son deal (si pending)	Tous les deals
Supprimer deal	Son deal	Son deal	Tous les deals
Approuver/Rejeter deal	Non	Oui	Oui
Supprimer commentaire	Son commentaire	Son commentaire	Tous les commentaires
Voter sur deal	Oui	Oui	Oui
Gérer utilisateurs	Non	Non	Oui

Aide au Développement

Structure de Projet

Organisez votre code selon le pattern MVC. Voici une structure recommandée qui sépare clairement les responsabilités :

```
dealexpress/
├── src/
│   ├── config/
│   │   └── database.js
│   ├── models/
│   │   ├── User.js
│   │   ├── Deal.js
│   │   ├── Comment.js
│   │   └── Vote.js
│   ├── routes/
│   │   ├── auth.routes.js
│   │   ├── deals.routes.js
│   │   ├── comments.routes.js
│   │   └── admin.routes.js
│   ├── controllers/
│   │   ├── auth.controller.js
│   │   ├── deal.controller.js
│   │   ├── comment.controller.js
│   │   └── admin.controller.js
│   ├── middlewares/
│   │   ├── auth.middleware.js
│   │   └── validate.middleware.js
│   ├── validators/
│   │   ├── auth.validator.js
│   │   └── deal.validator.js
│   └── app.js
├── .env
├── .gitignore
└── package.json
```

Indices Techniques

Pour calculer la température d'un deal :

Vous pouvez soit utiliser une méthode dans votre contrôleur qui compte les votes, soit créer un virtuel Mongoose. Pour compter, utilisez Vote.countDocuments avec des filtres sur dealId et type.

Pour vérifier l'ownership dans un middleware réutilisable :

Créez une fonction qui prend le modèle en paramètre, récupère la ressource, compare authorId avec req.user._id et vérifie également si l'utilisateur est admin. Si aucune des conditions n'est remplie, retournez un status 403.

Pour empêcher le double vote :

Avant de créer un vote, cherchez s'il existe déjà un vote avec findOne en filtrant sur userId et dealId. Si un vote existe, mettez à jour son type. Sinon, créez-en un nouveau.

Pour filtrer les deals selon le rôle :

Dans votre contrôleur GET /api/deals, construisez un objet filter. Si l'utilisateur n'est pas connecté ou a le role "user", ajoutez la condition status: "approved". Les modérateurs et admins verront tous les deals.

Pour la recherche :

Utilisez une regex MongoDB avec l'opérateur $or pour rechercher dans title et description. Exemple : { $or: [{ title: regex }, { description: regex }] }

Variables d'Environnement

Créez un fichier .env à la racine du projet :

```
PORT=3000
MONGODB_URI=mongodb://localhost:27017/dealexpress
JWT_SECRET=votre_secret_super_securise_a_changer
JWT_EXPIRE=7d
```

Packages à Installer

```
npm install express mongoose jsonwebtoken bcryptjs dotenv cors express-validator
npm install -D nodemon
```

Configuration package.json

Ajoutez ces scripts dans votre fichier package.json :

```
{
  "scripts": {
    "start": "node src/app.js",
    "dev": "nodemon src/app.js"
  }
}
```

Critères de Validation

Votre projet sera évalué selon les critères suivants. Assurez-vous de les respecter tous :

- Authentification : mots de passe hashés avec bcrypt, JWT fonctionnel, middleware authenticate opérationnel
- RBAC : trois rôles implémentés (user, moderator, admin), permissions respectées selon la matrice
- CRUD Deals : toutes les routes fonctionnelles avec filtrage par status selon le rôle de l'utilisateur
- Recherche : endpoint de recherche fonctionnel avec résultats pertinents
- Votes : système sans doublons, température calculée et mise à jour correctement
- Commentaires : CRUD complet avec vérification de l'ownership
- Modération : routes admin protégées, changement de status des deals opérationnel
- Gestion utilisateurs : liste des users et changement de rôles fonctionnels (admin uniquement)
- Validation : toutes les entrées utilisateur validées avec messages d'erreur explicites
- Relations : populate correct des auteurs, commentaires et votes

Livrables

Code source :

- Repository Git avec commits réguliers et messages explicites
- README.md incluant instructions d'installation, liste des endpoints et exemples d'utilisation
- Fichier .env.example avec les variables nécessaires (sans les valeurs sensibles)

Documentation :

- Collection Postman exportée contenant tous les scénarios de test
- Documentation des 3 comptes de test dans le README (user, moderator, admin) avec leurs credentials

Démonstration (facultatif) :

Préparez un scénario de démonstration montrant : création d'un deal par un user, vote et commentaire, puis modération par un moderator, et enfin gestion d'utilisateur par l'admin.

Barème d'Évaluation

Le projet sera noté sur 20 points répartis comme suit :

Architecture et organisation (3 points)

Structure de dossiers claire et logique, séparation routes/controllers/models respectée, code modulaire et maintenable.

Authentification (3 points)

Endpoints register et login fonctionnels, génération JWT avec role, middleware authenticate opérationnel.

Système RBAC (4 points)

Trois rôles implémentés correctement, permissions respectées selon la matrice, middleware requireRole fonctionnel et réutilisable.

CRUD Deals et Recherche (4 points)

Routes complètes et fonctionnelles, filtrage par status selon le rôle, vérification de l'ownership, endpoint de recherche opérationnel.

Votes (2 points)

Système de vote fonctionnel, gestion des doublons, calcul correct de la température.

Commentaires et Modération (2 points)

CRUD commentaires avec permissions, routes de modération protégées, gestion des utilisateurs (admin).

Qualité du code (2 points)

Validation des données sur tous les endpoints, gestion cohérente des erreurs, code propre et commenté aux endroits clés.

Conseils de Réalisation

Planning suggéré :

- Heure 1 : Configuration du projet, connexion MongoDB, modèle User, authentification de base
- Heure 2 : CRUD Deals sans permissions, puis ajout progressif des vérifications de rôle
- Heure 3 : Système de votes, gestion des doublons, calcul de température
- Heure 4 : Commentaires avec ownership, endpoint de recherche
- Heure 5 : Routes de modération, gestion utilisateurs, tests finaux et documentation

Par où commencer :

Démarrez par la configuration de base : installez les dépendances, configurez la connexion MongoDB, créez le modèle User. Ensuite, implémentez l'authentification (register et login) avec le middleware authenticate. Une fois l'auth fonctionnelle, créez le CRUD des deals sans les permissions. Ajoutez progressivement les vérifications de rôle. Terminez par les fonctionnalités avancées (votes, recherche, modération).

Pièges à éviter :

N'oubliez pas de hasher les mots de passe avant de les stocker. Attention à bien filtrer les deals selon le statut et le rôle de l'utilisateur. Gérez correctement les doublons de votes. N'oubliez jamais de populate les relations et d'exclure les champs sensibles (password). Validez toutes les données entrantes avant de les insérer en base. Testez systématiquement avec différents rôles pour vérifier les permissions.

Stratégie de test :

Créez trois comptes de test dès le début (user, moderator, admin). Exportez leurs tokens dans des variables d'environnement Postman. Testez chaque endpoint avec les trois tokens pour vérifier que les permissions fonctionnent correctement. Créez des scénarios complets : un user crée un deal, vote et commente, un moderator approuve le deal, un admin modifie un rôle.

Ressources utiles :

- Documentation Mongoose : mongoosejs.com/docs/populate.html
- Express Validator : express-validator.github.io/docs
- JWT : jwt.io pour décoder et vérifier vos tokens
- Postman : learning.postman.com pour apprendre à créer des collections de tests

Bon développement ! Ce projet couvre l'ensemble des compétences essentielles pour créer une API professionnelle avec Express.js. Prenez le temps de bien structurer votre code dès le départ, cela vous fera gagner du temps par la suite.