SuperHeroManager

I. 🎯 Objectif général

Développer une **application web complète** permettant de **gérer une base de super-héros** à partir d'un fichier JSON existant.
L'application comportera :

- une interface utilisateur en **React + TypeScript + Vite**,
- une **API REST** en **Node.js + Express + TypeScript**,
- une **base de données MongoDB** pour stocker les informations,
- et une **gestion des images** (upload, affichage, suppression).

Le projet vise à faire découvrir la logique d'une application **full-stack moderne** : front-end, back-end et base de données connectés.

Objectifs pédagogiques

Vous apprendrez à :

1. Structurer un projet full-stack complet.
2. Manipuler des données JSON et les importer dans MongoDB.
3. Créer et consommer une API REST sécurisée.
4. Gérer des formulaires complexes et des fichiers dans React.
5. Implémenter une authentification JWT.
6. Appliquer de bonnes pratiques de typage (TypeScript).

II. 📦 Architecture technique

Composant	Technologie	Rôle

Front-end	React + TypeScript + Vite	Interface utilisateur (SPA)
Back-end	Node.js + Express + TypeScript	API RESTful
Base de données	MongoDB + Mongoose	Stockage des héros et utilisateurs
Authentification	JWT (JSON Web Token)	Sécurisation des routes
Upload	Multer	Gestion des images (héros)
Données initiales	Fichier `SuperHerosComplet.json`	Import initial de la base

III. Fonctionnalités attendues

1. Authentification

- Inscription et connexion utilisateur (admin / éditeur)
- Stockage sécurisé des mots de passe (bcrypt)
- Authentification par token JWT
- Protection des routes sensibles selon le rôle

2. Gestion des héros (CRUD complet)

- **Lire** : lister tous les héros avec recherche et filtres
- **Créer** : ajouter un nouveau héros via un formulaire React
- **Mettre à jour** : modifier les informations d'un héros
- **Supprimer** : retirer un héros de la base
- **Afficher** : détails d'un héros (image, pouvoirs, univers, etc.)

3. Gestion des images

- Upload d'image lors de la création ou de la modification d'un héros
- Affichage des images sur la page d'accueil et la fiche détaillée
- Suppression automatique de l'image lors de la suppression du héros

4. Filtres et tri

- Filtrer par univers (Marvel, DC, Autre)
- Rechercher par nom ou alias
- Trier par date d'ajout ou ordre alphabétique

5. Rôles utilisateurs (bonus)

- **Admin** : accès total à toutes les fonctionnalités
- **Éditeur** : peut créer et modifier mais pas supprimer
- **Visiteur** : accès uniquement en lecture

6. Journalisation (bonus)

- Historique des actions (ajout, modification, suppression)
- Affichage dans un tableau "Logs" réservé à l'admin

IV. Base de données MongoDB

1. Collection `heroes`

Importée à partir du fichier `SuperHerosComplet.json`.
Champs typiques :

Champ	Type	Description
`_id`	ObjectId	Identifiant unique
`nom`	string	Nom du héros
`alias`	string	Surnom ou nom civil
`univers`	string	Marvel / DC / Autre
`pouvoirs`	[string]	Liste des pouvoirs
`description`	string	Brève biographie
`image`	string	URL ou chemin local
`origine`	string	Lieu ou contexte d'origine
`premiereApparition`	string	Année ou œuvre d'apparition

2. Collection `users`

Champ	Type	Description
`_id`	ObjectId	Identifiant unique
`username`	string	Nom d'utilisateur
`passwordHash`	string	Mot de passe chiffré
`role`	string	admin / editor
`createdAt`	date	Date d'inscription

V. Interface utilisateur

1. Pages principales

1. **Page de connexion**
   - Authentification utilisateur
   - Redirection vers le tableau de bord après connexion
2. **Tableau de bord (Dashboard)**
   - Liste de tous les héros (cartes ou tableau)
   - Recherche et filtres
   - Boutons "Modifier", "Supprimer", "Ajouter un héros"
3. **Formulaire d'ajout / modification**
   - Champs : nom, alias, univers, pouvoirs, description, image
   - Validation des champs (`Yup`)
   - Gestion d'upload avec aperçu d'image
4. **Page de détail**
   - Présentation complète du héros (image, texte, pouvoirs)
   - Boutons de modification et suppression selon rôle
5. **Page Administration**
   - Liste des utilisateurs
   - Historique des modifications (audit log)

2. Spécifications techniques

Back-end

- Serveur Express
- Routes REST (`/api/heroes`, `/api/auth`, `/api/users`)
- Middleware :
  - `authMiddleware` pour la vérification du JWT
  - `roleMiddleware` pour les permissions
  - `multerMiddleware` pour les uploads d'image
- Connexion MongoDB via Mongoose
- Fichier `.env` (variables : MONGO_URI, JWT_SECRET, PORT)

Front-end

- SPA (Single Page Application)
- React Router pour la navigation
- Axios pour les appels à l'API
- Formik + Yup pour les formulaires
- Design simple avec TailwindCSS ou MUI ou Boostrap
- Stockage du token JWT dans `localStorage`

VI. Arborescence complète du projet

```
superhero-manager/
│
├── README.md                    # Présentation du projet
├── .gitignore                   # Fichiers à ignorer dans Git
├── docker-compose.yml            # (Optionnel) Pour MongoDB + API
│
├── backend/                     # Serveur Node.js + Express
│   ├── package.json
│   ├── tsconfig.json
│   ├── .env                     # Variables d'environnement (PORT, JWT_SECRET,
MONGO_URI, etc.)
│   │
│   ├── src/
│   │   ├── index.ts             # Point d'entrée principal du serveur
│   │   │
│   │   ├── config/
│   │   │   └── db.ts            # Connexion MongoDB avec Mongoose
│   │   │
│   │   ├── models/              # Schémas de données (Mongoose)
│   │   │   ├── Hero.ts          # Schéma du super-héros
│   │   │   └── User.ts          # Schéma utilisateur (authentification)
│   │   │
│   │   ├── controllers/         # Logique métier de chaque ressource
│   │   │   ├── heroController.ts # CRUD héros (get, add, update, delete)
│   │   │   └── authController.ts # Connexion, inscription, vérification de token
│   │   │
│   │   ├── routes/              # Routes Express
│   │   │   ├── heroRoutes.ts    # Routes /api/heroes
```

```
│   │   │   └── authRoutes.ts      # Routes /api/auth
│   │   ├── middleware/            # Middlewares (auth, upload, logs)
│   │   │   ├── authMiddleware.ts  # Vérification du JWT
│   │   │   ├── roleMiddleware.ts  # Vérification des rôles (admin/editor)
│   │   │   └── uploadMiddleware.ts # Configuration de Multer (upload images)
│   │   ├── utils/                 # Fonctions utilitaires
│   │   │   ├── logger.ts          # Journalisation console/logs
│   │   │   └── seedDatabase.ts    # Import initial du JSON SuperHerosComplet.json
│   │   ├── uploads/               # Dossier où seront stockées les images des héros
│   │   └── SuperHerosComplet.json # Données d'import initiales
│   └── tests/                     # (Optionnel) Tests Jest ou supertest pour les routes
├── frontend/                      # Application React + TypeScript + Vite
│   ├── package.json
│   ├── tsconfig.json
│   ├── vite.config.ts             # Configuration Vite (proxy vers le backend)
│   ├── public/
│   │   └── logo.svg               # Fichiers statiques publics
│   ├── src/
│   │   ├── main.tsx               # Point d'entrée de l'application React
│   │   ├── App.tsx                # Routes principales
│   │   ├── api/                   # Fonctions pour appeler l'API (via Axios)
│   │   │   ├── heroApi.ts         # Appels CRUD pour /heroes
│   │   │   └── authApi.ts         # Appels pour /auth
│   │   ├── components/            # Composants réutilisables (UI)
│   │   │   ├── Navbar.tsx
│   │   │   ├── HeroCard.tsx
│   │   │   ├── HeroForm.tsx
│   │   │   ├── SearchBar.tsx
│   │   │   └── ProtectedRoute.tsx
│   │   ├── pages/                 # Pages principales
│   │   │   ├── LoginPage.tsx
│   │   │   ├── Dashboard.tsx
│   │   │   ├── HeroDetails.tsx
│   │   │   ├── AddHero.tsx
│   │   │   ├── EditHero.tsx
│   │   │   └── AdminPage.tsx
│   │   ├── context/               # Gestion de l'état global (authentification)
│   │   │   └── AuthContext.tsx
│   │   ├── hooks/                 # Custom hooks (API, formulaires, etc.)
│   │   │   └── useAuth.ts
│   │   ├── styles/                # Feuilles de style globales
│   │   │   └── index.css
│   │   └── types/                 # Interfaces TypeScript partagées
│   │       └── Hero.ts
│   │
```

```
    └── tests/                    # (Optionnel) Tests unitaires React
└── scripts/                      # (Optionnel) Scripts d'automatisation
    └── importData.ts             # Script d'import automatique du JSON vers MongoDB
```

Dossier	Apprentissage clé
`/backend/models`	Structuration de la donnée avec Mongoose
`/backend/controllers`	Séparation logique métier / route
`/backend/middleware`	Sécurité (auth, upload, rôles)
`/frontend/pages`	Architecture SPA (Single Page Application)
`/frontend/api`	Communication front ↔ back via Axios
`/frontend/context`	Gestion d'état global (authentification)
`/frontend/types`	Bonnes pratiques TypeScript et typage strict

VII. Livrables attendus

1. **Code source complet** (backend + frontend)
   - Projet propre, fonctionnel et versionné sur GitHub
2. **Documentation technique**
   - Liste des routes API
   - Schéma des collections MongoDB
   - Captures d'écran des interfaces
3. **Rapport de projet (5 pages max)**
   - Présentation générale
   - Architecture et technologies
   - Fonctionnalités réalisées
   - Difficultés rencontrées
   - Axes d'amélioration
4. **Vidéo de démonstration** (2 à 3 minutes)

VIII. Fiche – Technologies utilisées

1. Vue d'ensemble

L'application est composée de **trois couches principales** :

Couche	Technologie	Rôle
Front-end	React + TypeScript + Vite	Interface utilisateur (affichage, formulaires, navigation)
Back-end	Node.js + Express + TypeScript	API REST (traitement des données, sécurité, logique)
Base de données	MongoDB + Mongoose	Stockage des héros et utilisateurs

L'ensemble forme une **architecture client-serveur moderne** dite **"Full-Stack JavaScript"** : le même langage, **JavaScript / TypeScript**, est utilisé **partout**.

2. React – Front-end (interface utilisateur)

Description

React est une **bibliothèque JavaScript** développée par Facebook pour créer des **interfaces web dynamiques**. Elle repose sur un système de **composants réutilisables**.

Pourquoi l'utiliser

- Permet de construire une **SPA (Single Page Application)** : navigation fluide sans rechargement complet.
- Facilite la séparation du code (chaque élément visuel = composant).
- Compatible avec TypeScript (typage fort).
- Très utilisée dans l'industrie (LinkedIn, Netflix, GitHub…).

Rôle dans le projet

- Afficher la **liste des super-héros**.
- Gérer les **formulaires d'ajout / édition**.
- Implémenter la **connexion utilisateur** et la navigation.
- Appeler les routes du back-end avec **Axios**.

3. Vite – Outil de développement du front-end

Description

Vite est un **bundler** et **serveur de développement ultra-rapide** créé par Evan You (créateur de Vue.js).

Pourquoi l'utiliser

- Temps de démarrage quasi instantané.
- Rafraîchissement automatique du navigateur (HMR).
- Support natif de TypeScript.
- Plus simple et plus léger que Create React App.

Rôle dans le projet

- Démarrer le projet React (`npm create vite@latest frontend`).
- Compiler et lancer le code TypeScript en quelques secondes.

4. TypeScript – Langage commun front & back

Description

TypeScript est un **sur-ensemble de JavaScript** qui ajoute un **typage statique** et une **vérification à la compilation**.

Pourquoi l'utiliser

- Détecte les erreurs avant l'exécution.
- Améliore la lisibilité et la maintenabilité du code.
- Simplifie la collaboration entre développeurs.
- Compatible avec React, Node.js, et tous les outils modernes.

Rôle dans le projet

- Définir des **interfaces** pour les objets (`Hero`, `User`).
- Garantir que les échanges entre front et back soient cohérents.
- Faciliter la réutilisation du même type de données des deux côtés.

5. Node.js – Environnement serveur

Description

Node.js est un **environnement d'exécution JavaScript côté serveur**.
Il permet d'écrire du code backend (API, scripts, automatisations) en JavaScript.

Pourquoi l'utiliser

- Permet d'avoir un **langage unique** pour front et back.
- Excellent pour créer des **API REST rapides**.
- Gère facilement les opérations asynchrones (réseau, fichiers, BDD).

Rôle dans le projet

- Héberger le **serveur Express** qui gère les requêtes HTTP.
- Connecter l'application à MongoDB.
- Traiter les images et sécuriser les routes.

6. Express – Framework Node.js pour les API REST

Description

Express est un **framework minimaliste** pour créer des serveurs web et des **API REST**.

Pourquoi l'utiliser

- Simple et rapide à configurer.
- Routes et middlewares clairs.
- Grande communauté et nombreuses extensions (Multer, JWT…).

Rôle dans le projet

- Fournir les routes `/api/heroes`, `/api/auth`, etc.
- Traiter les opérations CRUD (Create, Read, Update, Delete).
- Appliquer les middlewares d'authentification et de validation.

7. MongoDB – Base de données NoSQL

Description

MongoDB est une **base de données orientée documents**.
Les données y sont stockées sous forme de **documents JSON**, très proches du format JavaScript.

Pourquoi l'utiliser

- Structure flexible (pas besoin de schéma fixe).
- Parfaite pour les applications JS (mêmes structures de données).
- Scalable et facile à héberger (MongoDB Atlas).

Rôle dans le proje

- Stocker les héros, utilisateurs et leurs informations.
- Être interrogée par le serveur Express via Mongoose.

8. Mongoose – ODM (Object Data Modeling)

Description

Mongoose est une **couche d'abstraction** entre Node.js et MongoDB.
Il permet de définir des **schémas de données** et de manipuler la base avec des méthodes simples.

Pourquoi l'utiliser

- Facilite la validation des données.
- Simplifie les requêtes vers MongoDB.
- Gère les relations entre collections si nécessaire.

Rôle dans le projet

- Définir les modèles `Hero` et `User`.
- Offrir des méthodes simples comme `Hero.find()` ou `Hero.create()`.

9. JWT (JSON Web Token) – Authentification

Description

JWT est un **standard pour l'authentification** entre un client et un serveur via un **token signé**.

Pourquoi l'utiliser

- Sécurise les routes protégées sans stocker de session sur le serveur.
- Léger, rapide et compatible avec toutes les plateformes.
- Facile à intégrer avec React (stockage dans `localStorage`).

Rôle dans le projet

- Générer un token à la connexion (`/auth/login`).
- Vérifier le token pour accéder aux routes `/api/heroes`.

10. *Multer – Upload de fichiers*

Description

Multer est un **middleware Express** pour gérer les **uploads de fichiers** (images, PDF…).

Pourquoi l'utiliser

- Supporte les formulaires multipart/form-data.
- Gère les noms de fichiers et les destinations.
- Simple à utiliser avec Node.js et React.

Rôle dans le projet

- Permet d'envoyer et stocker les **images des héros**.
- Gère l'enregistrement local des fichiers dans `backend/uploads`.

11. *Axios – Communication front / back*

Description

Axios est une **librairie HTTP** pour effectuer des appels réseau (API REST).

Pourquoi l'utiliser

- Syntaxe simple (`axios.get`, `axios.post`, etc.)
- Gestion facile des en-têtes (headers JWT)
- Promesses intégrées (async / await)

Rôle dans le projet

- Consommer les routes du back-end depuis React :
- `axios.get("http://localhost:5000/api/heroes")`

12. *Dotenv – Gestion des variables d'environnement*

Description

Dotenv charge les variables du fichier `.env` (non versionné dans Git).

Pourquoi l'utiliser

- Sépare la configuration sensible du code source (clés, mots de passe).
- Nécessaire pour MONGO_URI, JWT_SECRET, PORT.

Rôle dans le projet

- Fichier `.env` typique :
- `PORT=5000`
- `MONGO_URI=mongodb://localhost:27017/superheroes`
- `JWT_SECRET=supersecretkey`

13. CORS – Communication entre domaines

Description

CORS (Cross-Origin Resource Sharing) autorise les appels entre le front (port 5173) et le back (port 5000).

Pourquoi l'utiliser

- Par défaut, le navigateur bloque ces requêtes.
- Le middleware `cors()` dans Express permet de lever cette restriction.

14. Synthèse – Schéma global du projet