**Mern-todo Project**

mini projet MERN (MongoDB, Express, React, Node.js)

Création du projet backend (Node + Express).

Connexion à MongoDB.

Création de l’API REST.

Création du frontend React.

Connexion frontend ↔ backend.

Backend (Node + Express + MongoDB) :

API REST avec routes pour ajouter, lister, modifier, supprimer une tâche.

Frontend (React) :

Interface avec un champ texte et un bouton pour ajouter une tâche.

Liste des tâches avec cases à cocher pour marquer "terminée".

1. initialiser le projet
2. cd backend  
   npm init -y

npm install express mongoose cors dotenv

2. Créer un fichier server.js

3.Ajouter .env

Lance ton backend : node server.js

(bash)

Étape 3 : Frontend (React)

1. Créer le projet React

cd ../frontend

npx create-react-app .

npm install axios

**mern-todo/**

**├── backend/**

**│ ├── models/**

**│ │ ├── Task.js**

**│ │ └── Note.js**

**│ ├── routes/**

**│ │ ├── tasks.js**

**│ │ └── notes.js**

**│ ├── .env**

**│ ├── package.json**

**│ └── server.js**

**└── frontend/**

**├── src/**

**│ ├── components/**

**│ │ ├── Agenda.js**

**│ │ └── Notes.js**

**│ ├── App.js**

**│ └── ...**

**└── package.json**

**7. Installer les dépendances et démarrer**

bash

cd backend

npm install

npm start

Nous allons ajouter l'authentification JWT avec une inscription et une connexion.  
Étapes :

1. Créer un modèle User pour MongoDB.
2. Créer les routes pour l'inscription et la connexion.
3. Créer un middleware pour vérifier le token sur les routes protégées.
4. Modifier les routes existantes pour les protéger (tâches et notes ne seront accessibles qu'aux utilisateurs connectés).
5. Adapter le frontend pour gérer l'authentification (formulaires de login/register, envoi du token dans les requêtes, etc.).

Nous allons procéder par étapes, d'abord le backend puis le frontend.

Backend :

a) Modèle User (backend/models/User.js)  
b) Routes d'authentification (backend/routes/auth.js)  
c) Middleware d'authentification (backend/middleware/auth.js)  
d) Modification des routes de tâches et de notes pour les protéger.

Frontend :

a) Créer des composants pour Login et Register.  
b) Gérer l'état d'authentification (context ou état local).  
c) Stocker le token dans le localStorage et l'envoyer dans les entêtes des requêtes.  
d) Rediriger l'utilisateur après une connexion réussie.

Commençons par le backend.

Excellente idée ! Voici comment ajouter l'authentification complète à votre projet MERN :

## 1. Backend - Modèles et routes d'authentification

### Middleware d'authentification (backend/middleware/auth.js)

### d'authentification (backend/routes/auth.js)

javascript

### Modifier les modèles existants pour inclure les utilisateurs

**backend/models/Task.js**

**backend/models/Note.js**

### Modifier les routes pour les protéger

**backend/routes/tasks.js** (version modifiée)

Faire la même modification pour **backend/routes/notes.js**

## 2. Frontend - Authentification

### Context d'authentification (frontend/src/context/AuthContext.js)

### Composant Login (frontend/src/components/Login.js)

### Composant Register (frontend/src/components/Register.js)

### Modifier App.js pour gérer l'authentification

## 3. Installer les nouvelles dépendances

**Backend :**

bash

cd backend

npm install bcryptjs jsonwebtoken

**Frontend :** (pas de nouvelles dépendances nécessaires)

## 4. Mettre à jour les composants existants

Modifiez **Agenda.js** et **Notes.js** pour qu'ils utilisent automatiquement le token d'authentification. Les appels axios incluront maintenant automatiquement le header d'authentification grâce au contexte.

### Votre application est maintenant une **vraie application multi-utilisateurs** avec une authentification complète !

| **Caractéristique** | **Votre projet** | **Pro professionnel** |
| --- | --- | --- |
| ✅ Authentification | ✅ Implémentée | ✅ Requise |
| ✅ Base de données | ✅ MongoDB + relations | ✅ Oui |
| ✅ API REST | ✅ Routes protégées | ✅ Oui |
| ✅ Interface utilisateur | ✅ React moderne | ✅ Oui |
| ✅ Sécurité | ✅ JWT + bcrypt | ✅ Oui |
| ✅ Gestion d'état | ✅ Context API | ✅ Oui |
| ✅ Architecture | ✅ MVC séparée | ✅ Oui |
| 🔲 Tests | ❌ Manquant | ✅ Souhaitable |
| 🔲 Validation avancée | ❌ Basique | ✅ Recommandée |
| 🔲 Déploiement | ❌ Local | ✅ Nécessaire |

## 🚀 ****Plan DevOps complet pour votre projet****

### 1. **Conteneurisation avec Docker**

**docker-compose.yml** (à la racine du projet)

yaml

version: '3.8'

services:

mongodb:

image: mongo:latest

container\_name: mern-mongodb

environment:

MONGO\_INITDB\_ROOT\_USERNAME: admin

MONGO\_INITDB\_ROOT\_PASSWORD: password

MONGO\_INITDB\_DATABASE: merntodo

ports:

- "27017:27017"

volumes:

- mongodb\_data:/data/db

networks:

- mern-network

backend:

build: ./backend

container\_name: mern-backend

environment:

NODE\_ENV: production

MONGODB\_URI: mongodb://admin:password@mongodb:27017/merntodo?authSource=admin

JWT\_SECRET: votre\_super\_secret\_jwt\_production

PORT: 5000

ports:

- "5000:5000"

depends\_on:

- mongodb

networks:

- mern-network

restart: unless-stopped

frontend:

build: ./frontend

container\_name: mern-frontend

environment:

REACT\_APP\_API\_URL: http://localhost:5000/api

ports:

- "3000:80"

depends\_on:

- backend

networks:

- mern-network

restart: unless-stopped

volumes:

mongodb\_data:

networks:

mern-network:

driver: bridge

**backend/Dockerfile**

dockerfile

# Backend Dockerfile

FROM node:18-alpine

WORKDIR /app

# Copier package.json et installer les dépendances

COPY package\*.json ./

RUN npm ci --only=production

# Copier le code source

COPY . .

# Exposer le port

EXPOSE 5000

# Commande de démarrage

CMD ["node", "server.js"]

**frontend/Dockerfile**

dockerfile

# Frontend Dockerfile

FROM node:18-alpine as build

WORKDIR /app

# Copier package.json et installer les dépendances

COPY package\*.json ./

RUN npm ci

# Copier le code source et builder

COPY . .

RUN npm run build

# Stage de production avec nginx

FROM nginx:alpine

# Copier les fichiers buildés

COPY --from=build /app/build /usr/share/nginx/html

# Copier la configuration nginx

COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

**frontend/nginx.conf**

nginx

server {

listen 80;

server\_name localhost;

location / {

root /usr/share/nginx/html;

index index.html index.htm;

try\_files $uri $uri/ /index.html;

}

*# Proxy pour les API*

location /api {

proxy\_pass http://backend:5000;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

proxy\_cache\_bypass $http\_upgrade;

}

}

### 2. **CI/CD avec GitHub Actions**

**.github/workflows/ci-cd.yml**

yaml

name: MERN App CI/CD

on:

push:

branches: [ main, master ]

pull\_request:

branches: [ main, master ]

jobs:

test-backend:

runs-on: ubuntu-latest

defaults:

run:

working-directory: ./backend

steps:

- uses: actions/checkout@v3

- name: Setup Node.js

uses: actions/setup-node@v3

with:

node-version: '18'

cache: 'npm'

cache-directory: ./backend/node\_modules

- name: Install dependencies

run: npm ci

- name: Run tests

run: npm test

env:

MONGODB\_URI: mongodb://localhost:27017/test

JWT\_SECRET: test-secret

- name: Build check

run: npm run build --if-present

test-frontend:

runs-on: ubuntu-latest

defaults:

run:

working-directory: ./frontend

steps:

- uses: actions/checkout@v3

- name: Setup Node.js

uses: actions/setup-node@v3

with:

node-version: '18'

cache: 'npm'

cache-directory: ./frontend/node\_modules

- name: Install dependencies

run: npm ci

- name: Run tests

run: npm test

- name: Build project

run: npm run build

- name: Upload build artifacts

uses: actions/upload-artifact@v3

with:

name: frontend-build

path: frontend/build/

docker-build:

needs: [test-backend, test-frontend]

runs-on: ubuntu-latest

if: github.ref == 'refs/heads/main' || github.ref == 'refs/heads/master'

steps:

- uses: actions/checkout@v3

- name: Build backend Docker image

run: |

docker build -t ${{ secrets.DOCKER\_USERNAME }}/mern-backend:latest ./backend

- name: Build frontend Docker image

run: |

docker build -t ${{ secrets.DOCKER\_USERNAME }}/mern-frontend:latest ./frontend

- name: Login to Docker Hub

uses: docker/login-action@v2

with:

username: ${{ secrets.DOCKER\_USERNAME }}

password: ${{ secrets.DOCKER\_PASSWORD }}

- name: Push Docker images

run: |

docker push ${{ secrets.DOCKER\_USERNAME }}/mern-backend:latest

docker push ${{ secrets.DOCKER\_USERNAME }}/mern-frontend:latest

deploy:

needs: docker-build

runs-on: ubuntu-latest

if: github.ref == 'refs/heads/main' || github.ref == 'refs/heads/master'

steps:

- name: Deploy to production

run: |

echo "Deploying to production server..."

# Ici vous ajouteriez le déploiement sur votre serveur

# Exemple avec SSH :

# ssh deploy@${{ secrets.SERVER\_HOST }} 'docker-compose pull && docker-compose up -d'

### 3. **Configuration d'environnement**

**backend/.env.production**

env

NODE\_ENV=production

MONGODB\_URI=mongodb://admin:password@mongodb:27017/merntodo?authSource=admin

JWT\_SECRET=votre\_super\_secret\_jwt\_très\_long\_et\_sécurisé\_production

PORT=5000

CLIENT\_URL=https://votre-domaine.com

**frontend/.env.production**

env

REACT\_APP\_API\_URL=https://api.votre-domaine.com/api

REACT\_APP\_ENV=production

### 4. **Scripts package.json améliorés**

**backend/package.json**

json

{

"scripts": {

"start": "node server.js",

"dev": "nodemon server.js",

"test": "jest --verbose",

"test:watch": "jest --watch",

"test:coverage": "jest --coverage",

"build": "echo 'Backend build completed'",

"lint": "eslint .",

"lint:fix": "eslint . --fix",

"docker:build": "docker build -t mern-backend .",

"docker:run": "docker run -p 5000:5000 mern-backend"

}

}

**frontend/package.json**

json

{

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject",

"lint": "eslint src/",

"lint:fix": "eslint src/ --fix",

"docker:build": "docker build -t mern-frontend .",

"docker:run": "docker run -p 3000:80 mern-frontend"

}

}

### 5. **Monitoring et Logging**

**backend/utils/logger.js**

javascript

import winston from 'winston';

const logger = winston.createLogger({

level: 'info',

format: winston.format.combine(

winston.format.timestamp(),

winston.format.json()

),

transports: [

new winston.transports.File({ filename: 'error.log', level: 'error' }),

new winston.transports.File({ filename: 'combined.log' }),

new winston.transports.Console({

format: winston.format.simple()

})

]

});

export default logger;

**docker-compose.monitoring.yml**

yaml

version: '3.8'

services:

prometheus:

image: prom/prometheus

ports:

- "9090:9090"

volumes:

- ./monitoring/prometheus.yml:/etc/prometheus/prometheus.yml

networks:

- mern-network

grafana:

image: grafana/grafana

ports:

- "3001:3000"

environment:

GF\_SECURITY\_ADMIN\_PASSWORD: admin

volumes:

- grafana\_data:/var/lib/grafana

networks:

- mern-network

depends\_on:

- prometheus

node-exporter:

image: prom/node-exporter

ports:

- "9100:9100"

networks:

- mern-network

volumes:

grafana\_data:

### 6. **Tests automatisés**

**backend/tests/auth.test.js**

javascript

import request from 'supertest';

import app from '../server.js';

import mongoose from 'mongoose';

import User from '../models/User.js';

beforeAll(async () => {

await mongoose.connect(process.env.MONGODB\_URI);

});

afterAll(async () => {

await mongoose.connection.close();

});

describe('Auth API', () => {

it('should register a new user', async () => {

const res = await request(app)

.post('/api/auth/register')

.send({

username: 'testuser',

email: 'test@example.com',

password: 'password123'

});

expect(res.statusCode).toEqual(201);

expect(res.body).toHaveProperty('token');

});

});

### 7. **Health Checks**

**backend/routes/health.js**

javascript

import express from 'express';

import mongoose from 'mongoose';

const router = express.Router();

router.get('/', async (req, res) => {

const healthcheck = {

uptime: process.uptime(),

message: 'OK',

timestamp: Date.now(),

database: mongoose.connection.readyState === 1 ? 'Connected' : 'Disconnected'

};

try {

res.status(200).send(healthcheck);

} catch (error) {

healthcheck.message = error;

res.status(503).send();

}

});

export default router;

### 8. **Scripts de déploiement**

[**deploy.sh**](https://deploy.sh/)

bash

#!/bin/bash

echo "🚀 Starting deployment..."

*# Pull latest images*

docker-compose pull

*# Stop existing containers*

docker-compose down

*# Start new containers*

docker-compose up -d

*# Run database migrations (si nécessaire)*

*# docker-compose exec backend npm run migrate*

*# Run tests*

docker-compose exec backend npm test

echo "✅ Deployment completed!"

## 🛠 ****Mise en œuvre étape par étape :****

### **Phase 1 : Conteneurisation (Docker)**

1. Créer les Dockerfiles
2. Tester en local avec docker-compose
3. Builder et pousser les images

### **Phase 2 : CI/CD (GitHub Actions)**

1. Configurer les secrets dans GitHub
2. Tester le pipeline CI
3. Automatiser le déploiement

### **Phase 3 : Monitoring**

1. Configurer Prometheus/Grafana
2. Mettre en place les logs centralisés
3. Configurer les alertes

### **Phase 4 : Sécurité**

1. Scan des vulnérabilités Docker
2. Analyse de code sécurisé
3. Certificats SSL

## 🌐 ****Options de déploiement :****

### **Option 1 : VPS (DigitalOcean, OVH)**

bash

*# Sur votre serveur*

git clone votre-repo

docker-compose -f docker-compose.yml -f docker-compose.prod.yml up -d

### **Option 2 : Cloud (AWS, GCP, Azure)**

* **AWS** : ECS/EKS + RDS + S3
* **GCP** : GKE + Cloud SQL + Storage
* **Azure** : AKS + Azure SQL + Blob Storage

### **Option 3 : Platform as a Service**

* **Heroku** (plus simple)
* **Railway**
* **Render**

## 📊 ****Avantages DevOps pour votre projet :****

✅ **Déploiements reproductibles**  
✅ **Scalabilité horizontale**  
✅ **Surveillance en temps réel**  
✅ **Récupération automatique en cas d'échec**  
✅ **Tests automatisés à chaque changement**  
✅ **Environnements isolés (dev/staging/prod)**  
✅ **Sécurité renforcée**

Votre projet passe ainsi de **"application locale"** à **"application professionnelle déployable en production"** ! 🚀