Progetto Full Backend API “TaskManager”

Obiettivo generale

Realizzare un **backend completo e professionale** con Node.js, Express e MongoDB, capace di gestire utenti, autenticazione, risorse CRUD e upload di file, il tutto convalidato, sicuro e **pubblicato online**.

Descrizione generale del progetto

Creerai un’applicazione backend chiamata **TaskManager API** che consente a ogni utente registrato di:

- Creare un account e autenticarsi con **JWT**;
- Gestire una lista personale di **Task** (aggiungere, leggere, modificare, eliminare);
- Allegare un **file immagine** a un task (es. screenshot, documento, ecc.);
- Validare tutti i dati in ingresso con **Joi**;
- Pubblicare online l’app su **Render o Railway** e connetterla a **MongoDB Atlas**.

Requisiti tecnici

Stack

- Node.js + Express
- MongoDB + Mongoose
- Librerie suggerite:
    - `bcryptjs` → hash password
    - `jsonwebtoken` → JWT auth
    - `joi` → validazione dati
    - `multer` → upload file
    - `dotenv` → gestione variabili d’ambiente
    - `helmet` e `express-rate-limit` → sicurezza
    - `morgan` o `winston` → logging
    - `cors` → accesso cross-domain

Struttura del progetto

Organizza le cartelle in modo pulito e modulare:

```
src/
├─ server.js
├─ config/
│  ├─ db.js
│  └─ env.js
├─ models/
│  ├─ user.model.js
│  └─ task.model.js
├─ controllers/
│  ├─ auth.controller.js
│  └─ task.controller.js
├─ services/
│  ├─ auth.service.js
│  └─ task.service.js
├─ routes/
│  ├─ auth.routes.js
│  └─ task.routes.js
├─ middleware/
│  ├─ auth.js
│  ├─ validate.js
│  └─ errorHandler.js
├─ utils/
│  └─ logger.js
└─ uploads/
```

Parte 1 — Autenticazione utente (JWT)

Obiettivo

Permettere la registrazione e il login di un utente con protezione JWT.

Requisiti

1. **POST /api/auth/register** → crea nuovo utente (email + password + nome)
2. **POST /api/auth/login** → restituisce un token JWT valido
3. **Middleware auth** → protegge le rotte successive (controlla il token)
4. **User model** → password hashata con `bcrypt`

Validazione con Joi

- Email: obbligatoria e valida
- Password: minimo 6 caratteri
- Nome: opzionale o obbligatorio secondo scelta

Parte 2 — CRUD dei Task

Obiettivo

Ogni utente autenticato può gestire i propri task.

Rotte richieste

1. `GET /api/tasks` → lista task dell'utente loggato
2. `POST /api/tasks` → crea nuovo task
3. `PATCH /api/tasks/:id` → aggiorna titolo o stato
4. `DELETE /api/tasks/:id` → elimina task

Validazione

Ogni task deve avere almeno:

- `title`: stringa obbligatoria
- `description`: opzionale
- `completed`: booleano, default false

Usa **Joi** in un middleware `validateTask` per controllare i dati prima di arrivare al controller.

Parte 3 — Upload di file

Obiettivo

Consentire di allegare un file immagine a un task.

Requisiti

- Usa **multer** per gestire l'upload;
- Accetta solo immagini (`image/*`);
- Salva i file nella cartella `/uploads`;
- Salva nel DB il nome originale, il percorso e l'owner (utente).

Rotta

`POST /api/tasks/:id/upload` → allega un file al task specificato.

Parte 4 — Sicurezza e validazione

Implementa:

1. **Helmet** → aggiunge header di sicurezza automatici.
2. **express-rate-limit** → limita richieste su `/api/auth/` (es. max 100 ogni 15 min).

3. **CORS mirato** → solo il dominio del frontend o `localhost`.
4. **Error handling** → centralizzato in `errorHandler.js`.

Parte 5 — Logging

- Usa **morgan** o **winston** per registrare:
    - metodo e rotta della richiesta,
    - codice di stato,
    - tempo di risposta.

Suggerimento: crea un file `logger.js` che esporta una funzione `logRequest()` da usare come middleware.

Parte 6 — Deployment online

Obiettivo

Rendere il backend pubblico e funzionante online.

Step:

1. Crea un database su **MongoDB Atlas**;
2. Pubblica il progetto su **GitHub**;
3. Fai deploy su **Render** o **Railway**;
4. Imposta le **variabili d'ambiente**:
    - `MONGO_URI`
    - `JWT_SECRET`
    - `NODE_ENV=production`
5. Testa con Postman l'endpoint `/api/health`.

Verifica finale

Il progetto sarà considerato **completo** se:

- L'autenticazione funziona (JWT valido e protetto);
- Le rotte CRUD dei task rispondono correttamente;
- Gli upload funzionano e i file vengono salvati;
- Le validazioni Joi impediscono dati errati;
- La sicurezza base (helmet, rate-limit) è attiva;
- L'app è online e testabile da URL pubblico.

Consegna per gli studenti

Cosa consegnare:

1. **Link GitHub** del progetto.
2. **URL pubblico Render o Railway** del backend.
3. Breve **README.md** con:
    - Descrizione del progetto
    - Librerie principali usate
    - Istruzioni per avviare il progetto