Progetto Full Backend API “TaskManager”

Si analizza il progetto **TaskManager API** dal punto di vista **backend**, considerando struttura, flusso logico, modellazione dati, aspetti funzionali e operativi. Ti faccio un’analisi completa e approfondita.

Architettura generale del backend

Il progetto segue un **pattern modulare e MVC-light**:

```
src/
├─ server.js            → entry point del server
├─ config/              → configurazioni (DB, variabili ambiente)
├─ models/              → modelli Mongoose
├─ controllers/         → logica delle richieste
├─ services/            → logica di business e operazioni DB
├─ routes/              → definizione API e middleware
├─ middleware/          → autenticazione, validazioni, error
handling
├─ utils/               → utility generiche, logger
└─ uploads/             → storage locale dei file
```

Analisi:

- Mantiene separazione dei livelli: routing → controller → service → DB.
- Facilita testing e manutenzione.
- middleware centralizza cross-cutting concerns (auth, validate, error handling).

Modellazione dei dati

Utente (user.model.js)

Campi principali:

- name → string, opzionale/obbligatorio
- email → string, obbligatorio, unico
- password → string, hashata
- timestamps (createdAt, updatedAt)

Considerazioni:

- Password **hashata con bcrypt**.
- Email unica per identificare utenti.
- JWT basato su _id e email per autenticazione stateless.

Task (task.model.js)

Campi principali:

- title: string, obbligatorio
- description: string, opzionale
- completed: booleano, default false
- owner: ObjectId → riferimento a User
- file: opzionale, oggetto { originalName, path } se allegato file
- timestamps

Considerazioni:

- Associazione **uno-a-molti (1:N)** con utente (owner).
- File opzionale per task: utile per screenshot, documenti ecc.
- Default completed=false semplifica logica frontend.

Flusso logico delle funzionalità

Parte 1 — Autenticazione

1. **POST /api/auth/register**
    - Validazione input con Joi (email, password ≥6, nome).
    - Hash della password con bcrypt.
    - Salvataggio in MongoDB.
2. **POST /api/auth/login**
    - Verifica email e password.
    - Generazione JWT con jsonwebtoken.
3. **Middleware auth.js**
    - Legge Authorization: Bearer <token> header.
    - Decodifica JWT e aggiunge req.user.
    - Protegge tutte le rotte CRUD e upload.

Flow:

Client → Auth Route → Validation Middleware → Controller → Service → DB

Parte 2 — CRUD Task

- Tutte le rotte richiedono autenticazione (auth middleware).
- **GET /api/tasks** → lista solo i task dell'utente loggato.
- **POST /api/tasks** → crea task con validazione Joi.
- **PATCH /api/tasks/:id** → modifica titolo, stato, description.
- **DELETE /api/tasks/:id** → elimina task solo se proprietario.

Flow:

Client → Auth Middleware → Validate Task → Controller → Service → DB → Response

Nota funzionale: la separazione Controller vs Service permette di spostare logiche complesse fuori dal controller, mantenendo testabilità.

Parte 3 — Upload file

- **Middleware multer:**
  - Storage su cartella /uploads.
  - Limitazione a image/*.
- **Endpoint POST /api/tasks/:id/upload:**
  - Verifica proprietà owner (solo il proprietario può allegare file).
  - Salva file fisico + meta (nome originale, path) nel DB.

Flow:

Client → Auth → Task Owner Check → Multer → Controller → Service → DB → Response

Parte 4 — Sicurezza e validazione

- **Helmet** → header HTTP sicuri (XSS, MIME sniffing ecc.)
- **express-rate-limit** → previene brute force su auth (max 100/15min)
- **CORS** → permette solo frontend autorizzati (es. localhost o dominio)
- **Central Error Handler (errorHandler.js)** → unifica gestione errori JSON e log.

Parte 5 — Logging

- **morgan** o **winston** per:
  - Metodo, rotta, codice stato, tempo risposta.
- Logger centralizzato in utils/logger.js.
- Può evolvere in log persistenti o integrati con sistemi esterni.

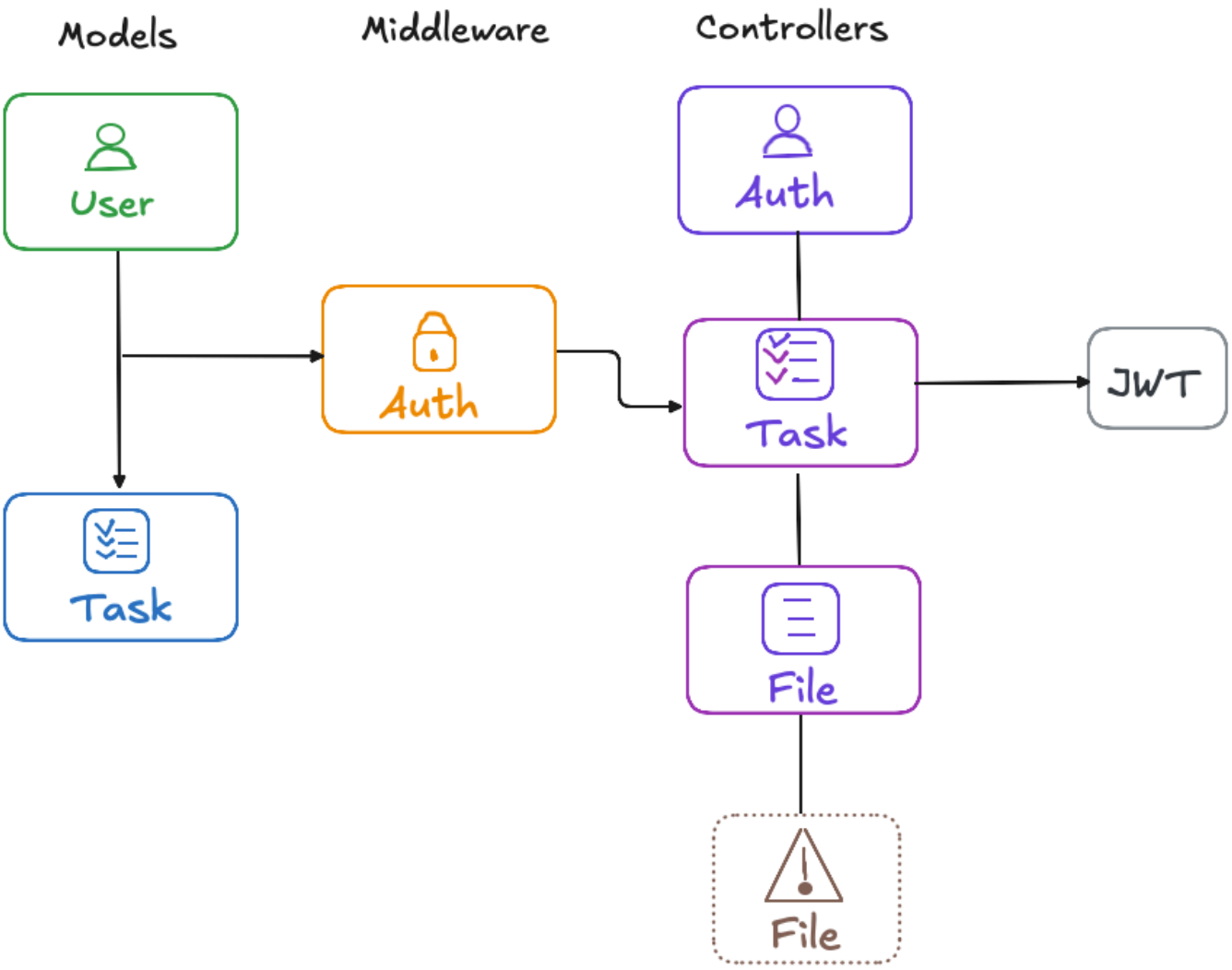
Considerazioni funzionali e qualità

Aspetto	Analisi
Validazione	Joi protegge endpoint da dati errati.
Autenticazione	JWT stateless sicuro, password hashate.
Autorizzazione	Task protetti da ownership, upload limitato.
Scalabilità	Modularità + services permette aggiunta funzionalità.
Sicurezza	Rate-limit, Helmet, CORS mirato.
Logging	Tracciamento completo richieste.
Deployment	Preparato per MongoDB Atlas + Render/Railway.

Flusso complessivo utente-task

1. Utente registra account
2. Utente logga → riceve token
3. Utente crea task → validazione + salvataggio DB
4. Utente legge task → filtro per owner
5. Utente aggiorna/elimina task → verifica owner + modifica DB
6. Utente allega file → multer salva file + DB update
7. Richieste passano per auth + rate-limit + helmet + cors

Questo flusso garantisce **funzionalità completa, sicurezza e separazione dei livelli**, pronto per essere esteso (es. notifiche, tag, categorie).


Models
Middleware
Controllers
User
Auth
Auth
Task
Task
JWT
File
File

Setup ambiente e struttura base

Inizializza il progetto

```
mkdir taskmanager-api && cd taskmanager-api
npm init -y
```

Installa le dipendenze base

```
npm install express mongoose dotenv
npm install bcryptjs jsonwebtoken joi multer
npm install helmet express-rate-limit cors morgan
npm install nodemon –save-dev (sono 2 trattini prima di save)
```

Il seguente è lo **Stack** suggerito**:**

- 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

Crea la struttura cartelle

```
mkdir -p src/
{config,models,controllers,services,routes,middleware,utils,upload
s}
touch src/server.js
```

Configura gli script package.json

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

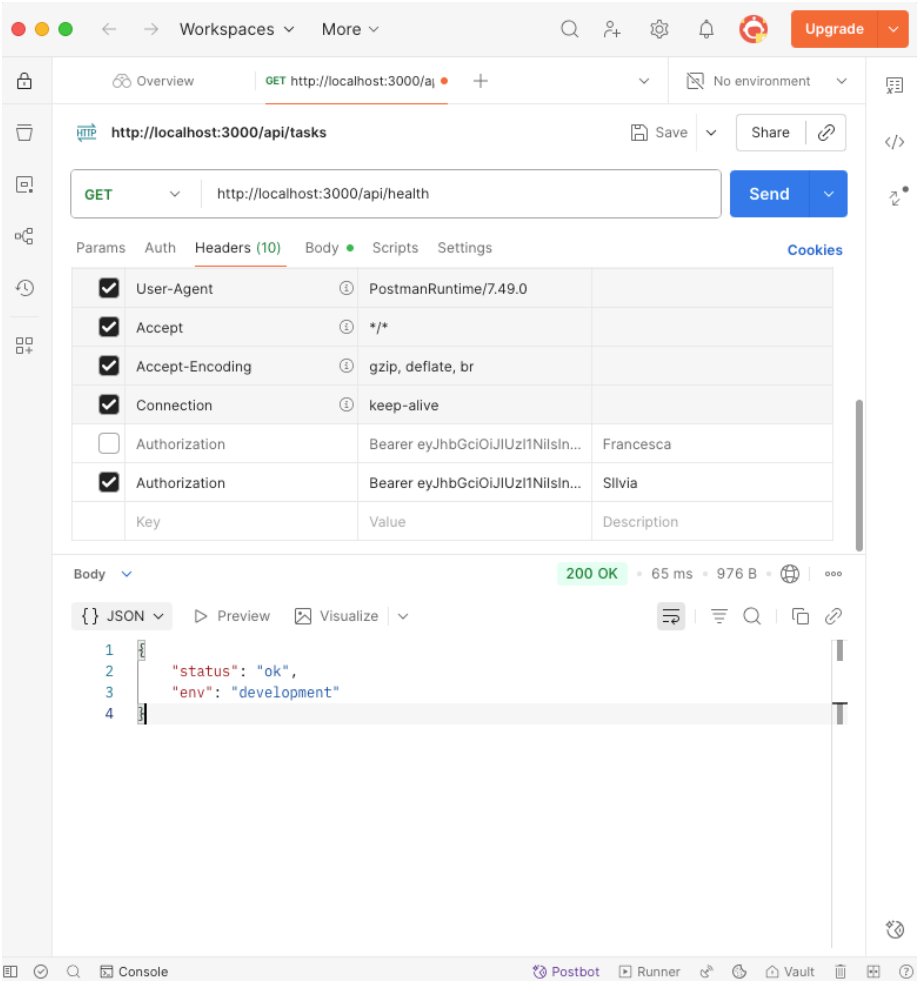
creare package.json (se non è presente) e installare le dipendenze:
`npm install`

per lo sviluppo:
`npm run dev`

Testare l'endpoint di health check

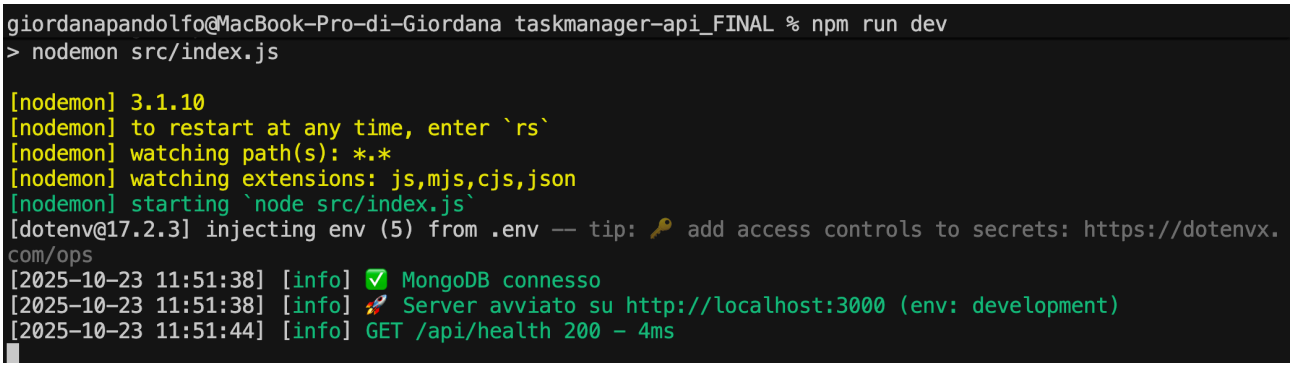
Aprire **Postman** o il browser:

http://localhost:3000/api/health



Se la risposta è ok:

- Il server Express funziona
- MongoDB è connesso
- Le middleware base (helmet, cors, rate-limit, morgan, logger) sono operative



Tutta la parte iniziale del backend è correttamente impostata.

Opzionale: Verifica i log

Durante l'esecuzione sia Morgan che Winston scrivono nel terminale (vedi immagine sopra), questo mostra che i log HTTP passano correttamente al sistema di logging centralizzato.

Testare cosa succede se il DB non è raggiungibile

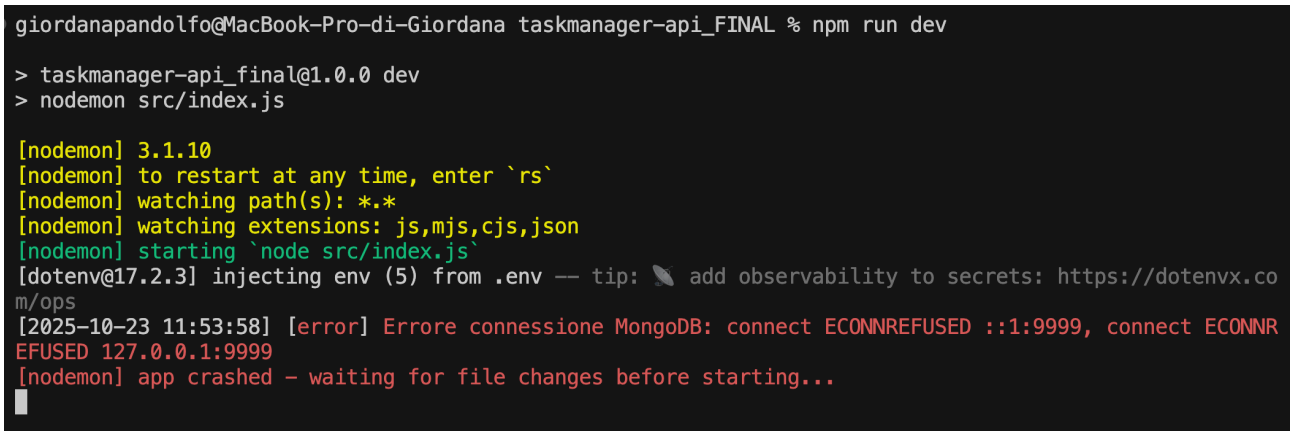
Per testare la gestione errori:

- Modificare temporaneamente la MONGO_URI nel .env con qualcosa di errato (es. mongodb://localhost:9999/error)
- Riavviare il server

Si vedrà in console:

`Errore avvio server: …`

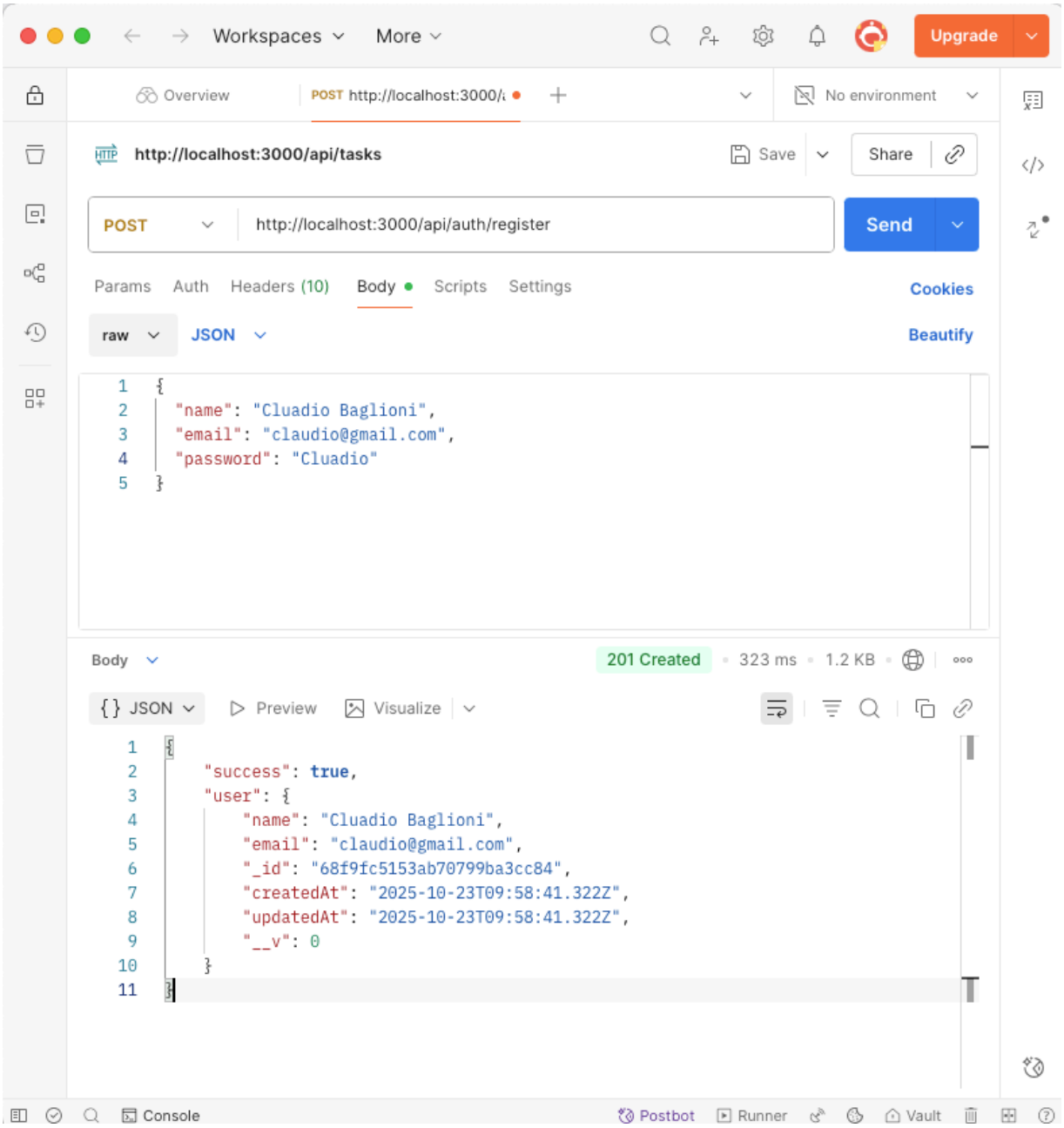
Questo conferma che l'error handling della connessione funziona correttamente.



Come testare (Postman / curl) — esempi pratici

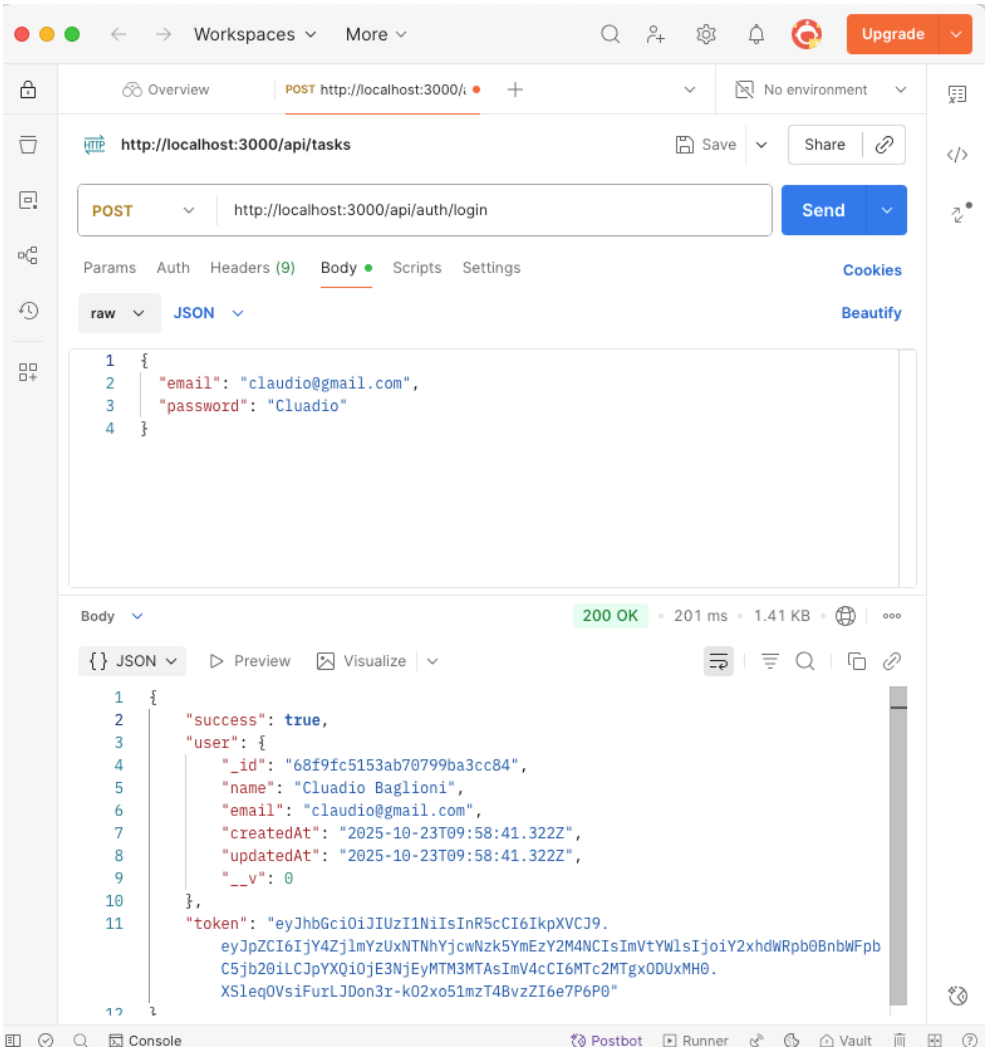
1) Registrazione

`POST http://localhost:3000/api/auth/register`



2)Login

`POST http://localhost:3000/api/auth/login`



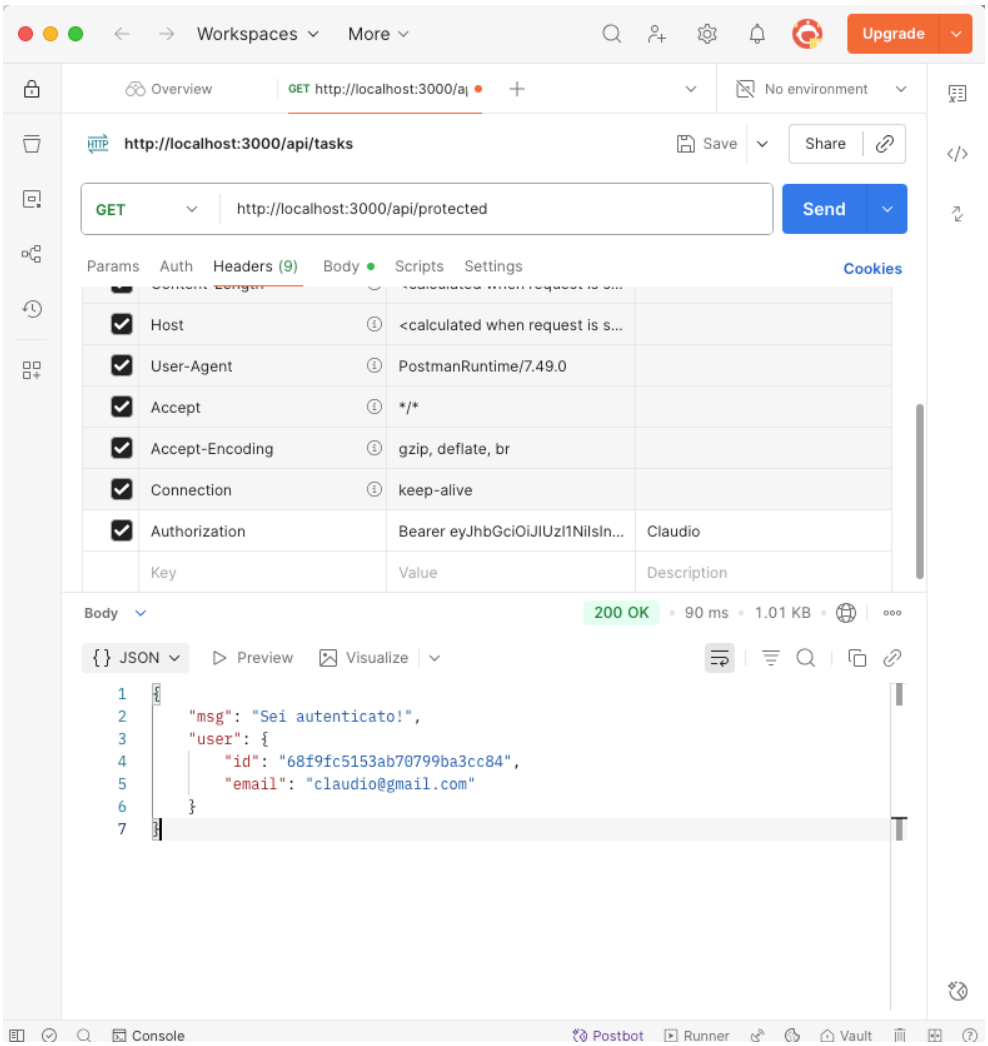
In MongoDB:

```
// Rotta protetta di test per verificare il middleware JWT
app.get('/api/protected', auth, (req, res) => {
  res.json({ msg: 'Sei autenticato!', user: req.user });
});
```

Questa è una **rotta di test** per verificare che:

- Il token JWT è valido
- Il middleware auth riesce a decodificarlo
- Viene popolato req.user

```
Get http://localhost:3000/api/protected
```



Obiettivi finali:

Registrazione utente (/api/auth/register)

- Crea un nuovo documento in MongoDB.
- Password hashata con bcrypt (pre-save hook).

Login (/api/auth/login)

- Verifica email e password.
- Genera token JWT valido 7 giorni.

Middleware auth

- Legge l'header Authorization: Bearer <token>.
- Verifica la firma con JWT_SECRET.
- Aggiunge req.user con { id, email }.

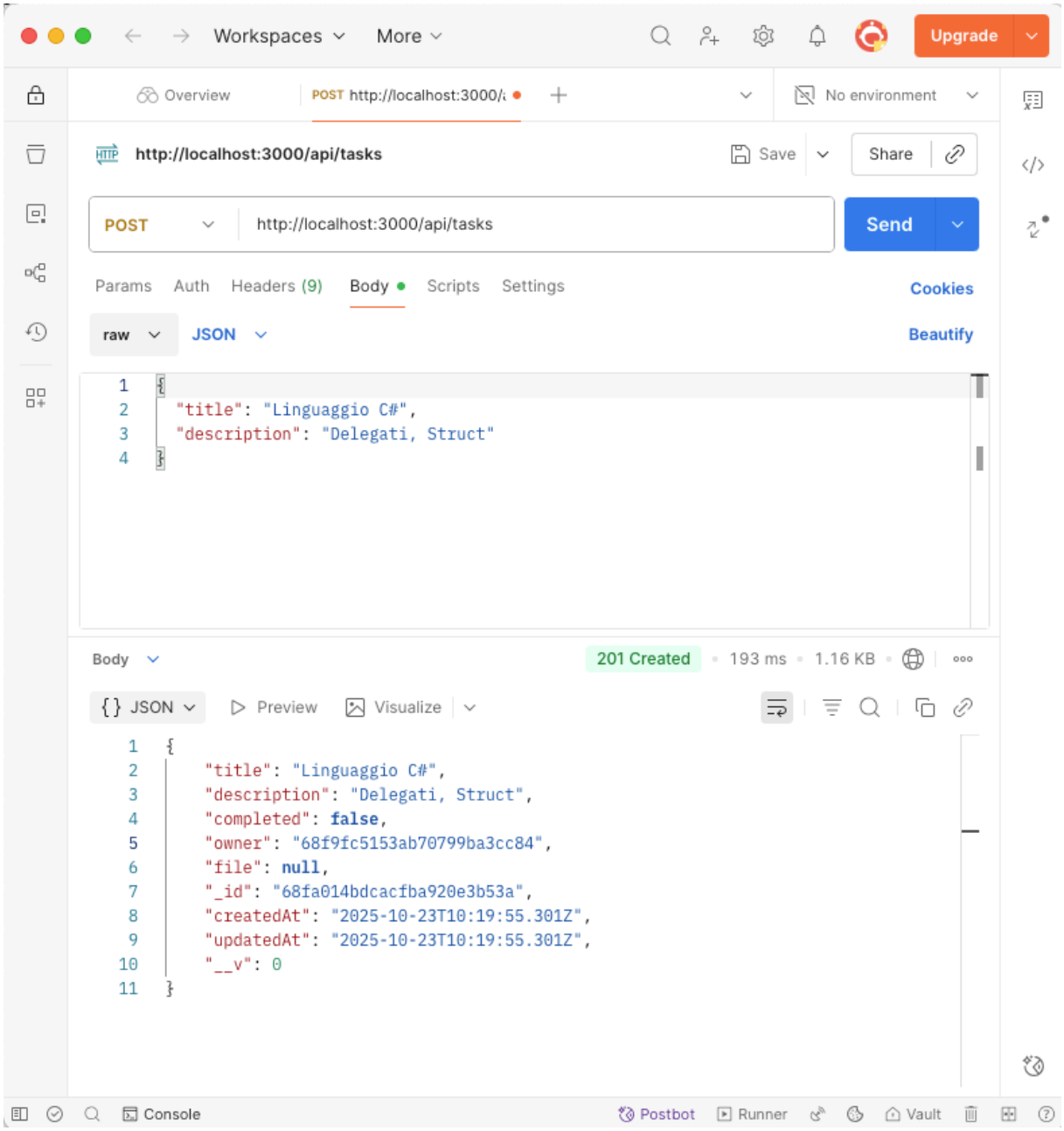
Test rotta protetta (/api/protected)

- Ti conferma che il token è valido e il middleware funziona.

Test con Postman

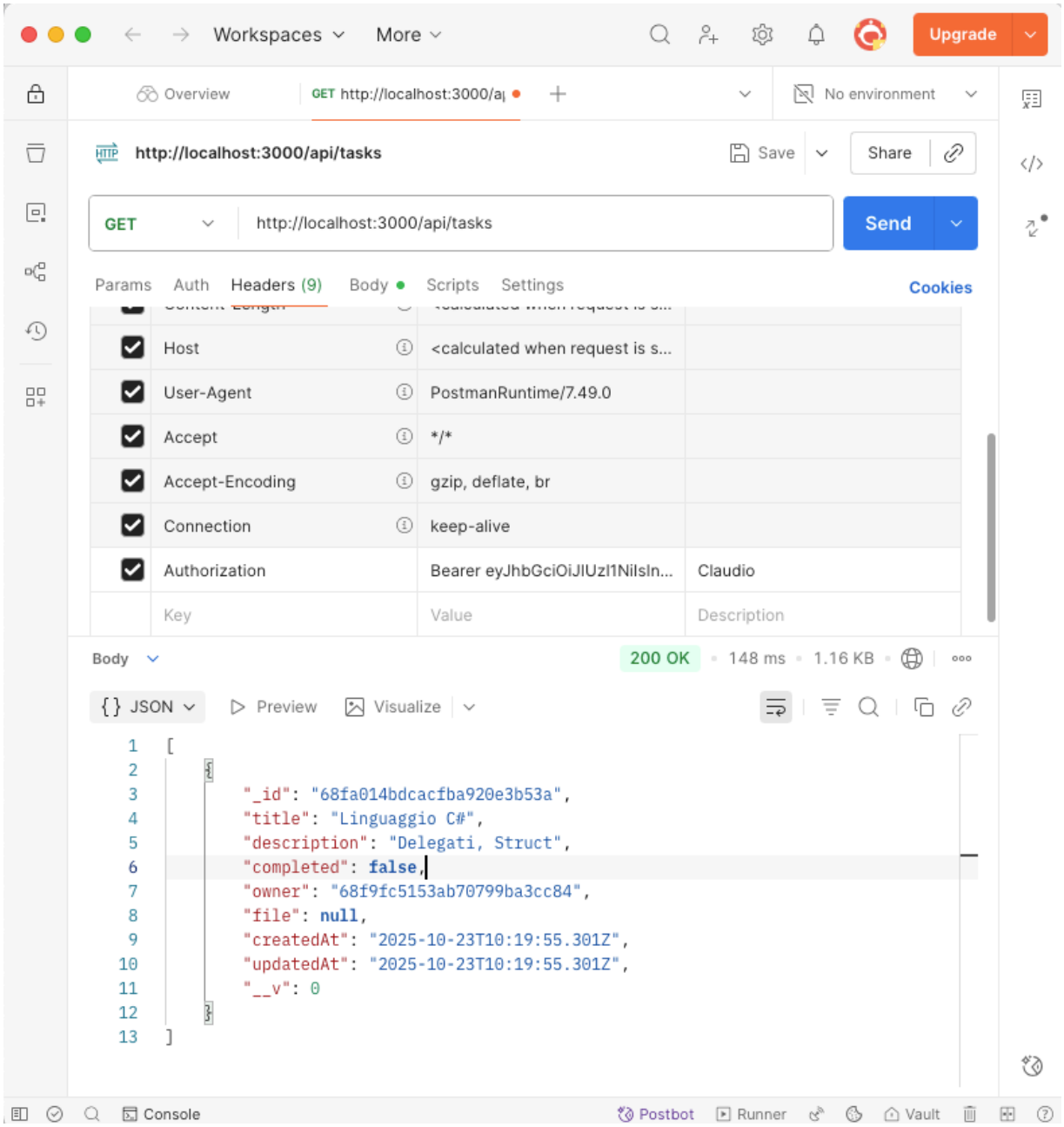
Crea un Task

POST http://localhost:3000/api/tasks



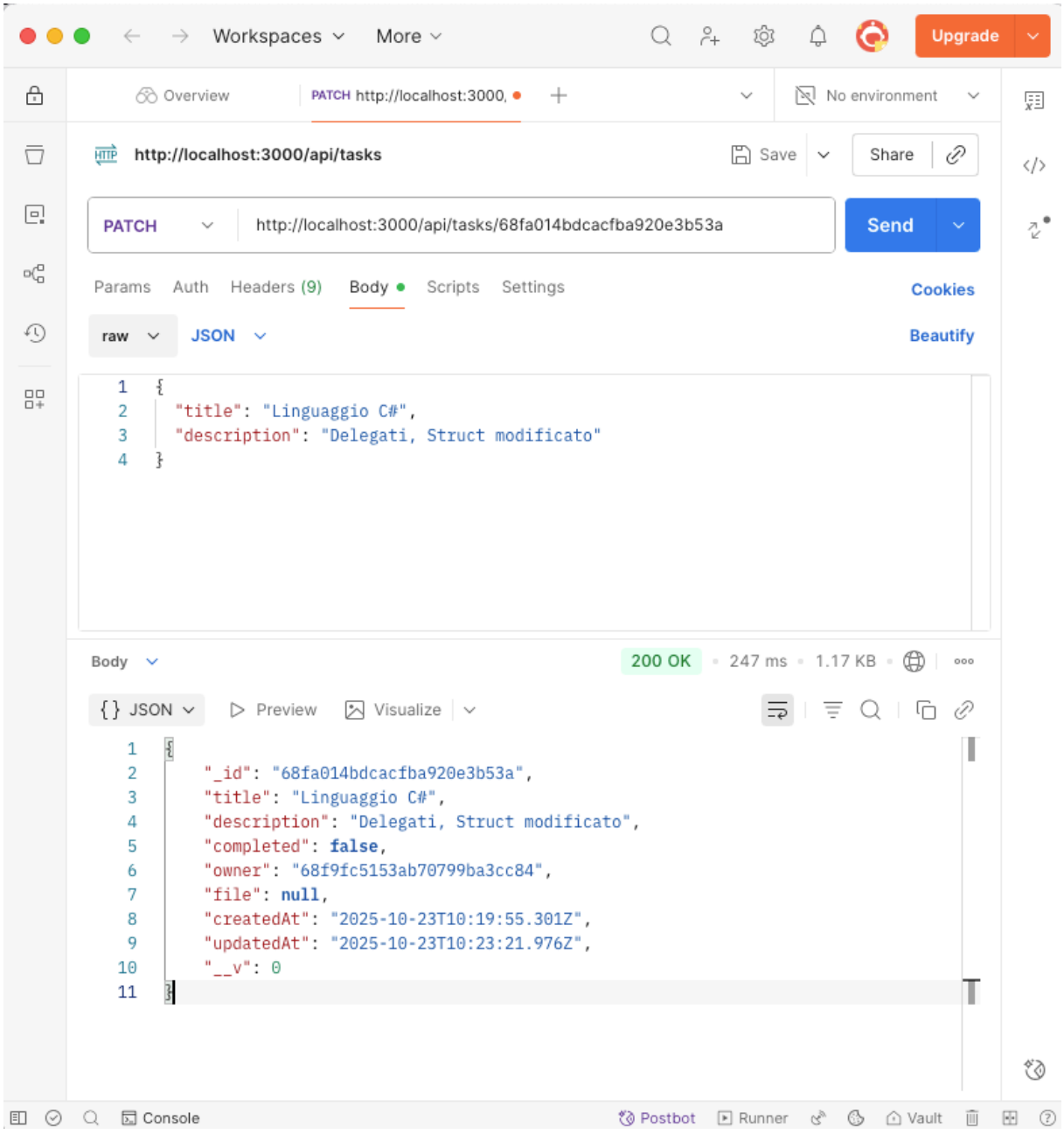
Ottieni tutti i Task

GET http://localhost:3000/api/tasks



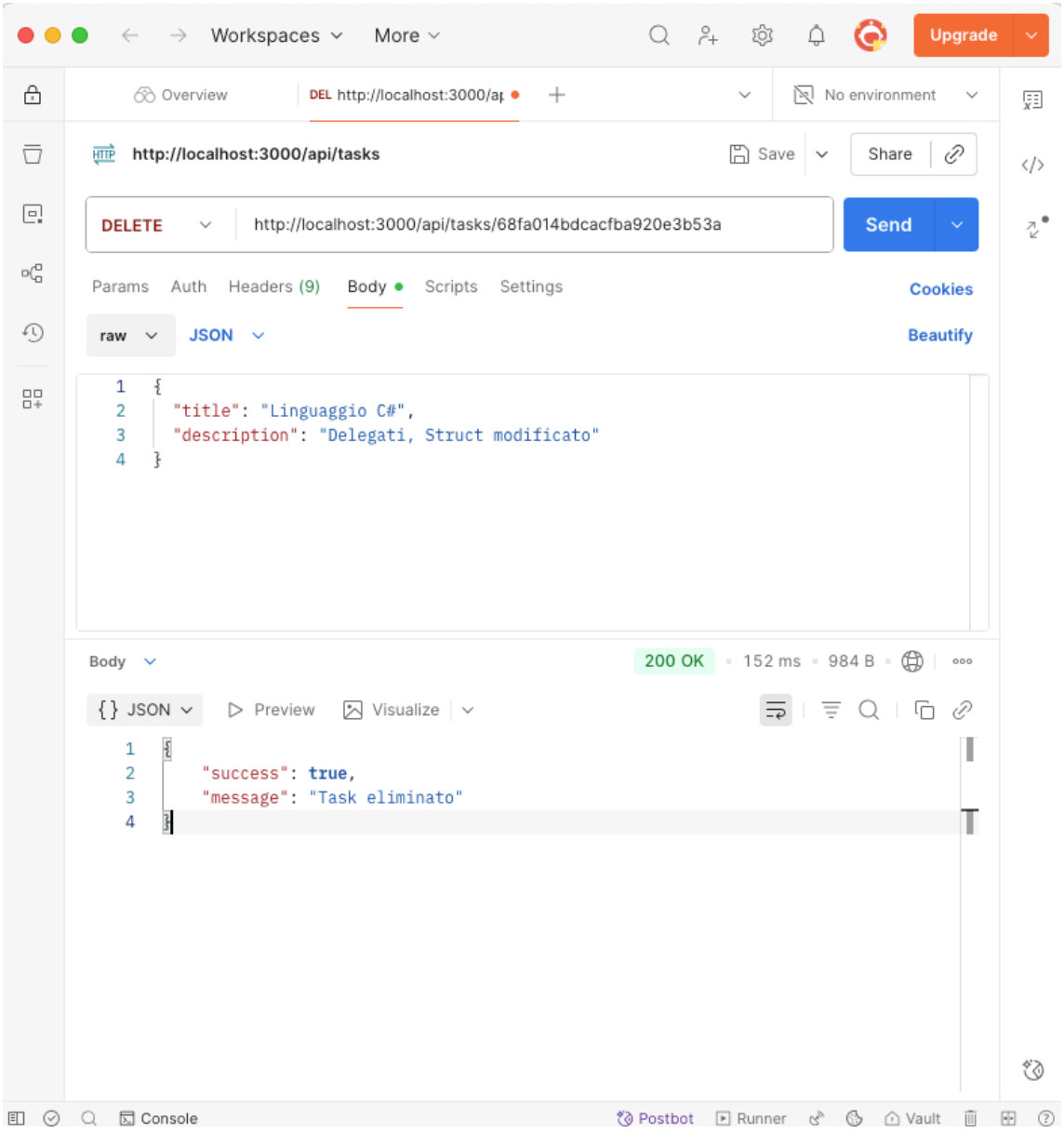
Aggiorna un Task

PATCH http://localhost:3000/api/tasks/ID



Elimina un Task

DELETE http://localhost:3000/api/tasks/ID



Obiettivi finali:

Registrazione/Login utente JWT.

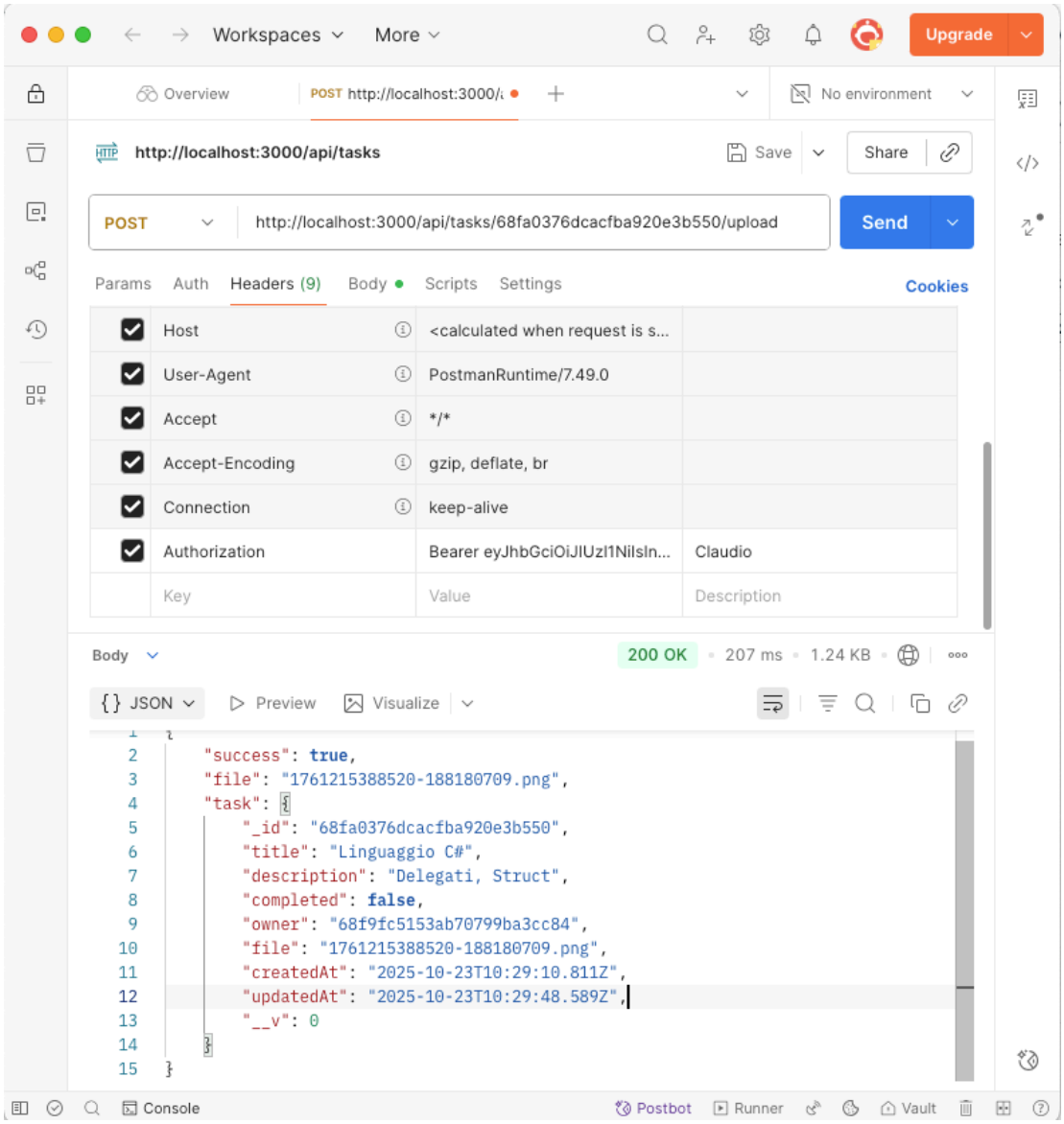
CRUD completo dei Task autenticati.

Validazioni Joi su input.

Sicurezza di base e separazione dei livelli (model, service, controller, route).

Upload di un file a un Task

POST http://localhost:3000/api/tasks/ID/upload



Obiettivi finali:

Validazione JWT;

CRUD task con ownership;

Upload immagini/pdf con Multer;

File serviti via Express statico.

Altri test con Postman:

- **Invalid login:** password sbagliata → 401
- **Accesso senza token**: /tasks → 401
- **Creazione task con body mancante**: title → 400
- **Upload file non permesso**: tipo .exe → 400
- **Delete task di un altro user:** → 403