“TaskManager” (Full Backend)

> Suggerimento didattico: 2–3 tappe al giorno. Ogni tappa ha un **Goal**, una **Checklist** e un **Test rapido**. Evita di passare alla successiva senza chiudere la presente.

TAPPA 0 — Setup e fondamenta

Goal
Ambiente pronto, repo Git, app Express minimal che risponde.

Fai così

1. Crea cartella progetto e inizializza `npm`.
2. Crea repo Git, `.gitignore` (includi `.env`, `node_modules`, `uploads/`).
3. Struttura base cartelle:

```
src/
├─ server.js
├─ config/ (env, db)
├─ routes/
├─ controllers/
├─ services/
├─ models/
├─ middleware/
├─ utils/ (logger)
└─ uploads/
```

4. Aggiungi una rotta “health” (`/api/health`) per verifiche.

Checklist

- `npm start` avvia il server
- `/api/health` risponde `{status:"ok"}`
- Primo commit su Git

Test rapido
Browser o Postman → `GET /api/health` → 200 OK.

TAPPA 1 — Database & Modelli (User, Task)

Goal
Connessione a **MongoDB Atlas** + modelli Mongoose definiti.

Fai così

1. Crea cluster gratuito su Atlas, utente DB, allowlist IP (anche 0.0.0.0/0 per il laboratorio).
2. Salva la **connection string** in `.env` come `MONGO_URI`.

3. Crea `User` (email, passwordHash, ruolo opzionale) e `Task` (title, description?, completed=false, owner: ref User).
4. Connetti Mongoose all'avvio e logga stato.

Checklist

- `MONGO_URI` in `.env` (non nel repo!)
- Modello `User` e `Task` creati
- Connessione DB riuscita in console

Test rapido
Prova a inserire da una rotta temporanea o da un service una singola entità (anche mock) e verifica in Atlas.

TAPPA 2 — Validazione con Joi

Goal
Bloccare input invalidi prima dei controller.

Fai così

1. Crea middleware generico `validate(schema)`.
2. Definisci schemi per auth (register/login) e task (create/update).
3. Applica `validate(schema)` alle rotte che ricevono body (POST/PATCH).

Checklist

- Middleware `validate` attivo
- Schemi Joi per auth e task
- Errori 400 chiari con dettagli

Test rapido
Invia body errati (email non valida, title vuoto) → ottieni 400 con messaggio leggibile.

TAPPA 3 — Autenticazione JWT (Register/Login + Middleware)

Goal
Utente può registrarsi, fare login e ottenere un token da usare nelle rotte protette.

Fai così

1. Rotte: `POST /api/auth/register`, `POST /api/auth/login`.

2. Hash password con bcrypt; non salvare password in chiaro.
3. Genera **JWT** con `userId` (e `role` opzionale).
4. Middleware `auth` che legge `Authorization: Bearer <token>` e valorizza `req.user`.

Checklist

- Registrazione con email univoca
- Login restituisce token valido
- Middleware `auth` rifiuta richieste senza token (401)

Test rapido

- `POST /api/auth/register` → 201
- `POST /api/auth/login` → token
- Usa token per accedere a `GET /api/profile` (rotta protetta di prova) → 200.

TAPPA 4 — CRUD Task (scoped all'utente)

Goal
Ogni utente gestisce **solo i propri task**.

Fai così

1. Rotte protette (`auth`):
    - `GET /api/tasks` (solo dell'utente loggato)
    - `POST /api/tasks` (crea; `owner = req.user.userId`)
    - `PATCH /api/tasks/:id` (verifica appartenenza; aggiorna)
    - `DELETE /api/tasks/:id` (solo owner)
2. Validazione Joi su POST/PATCH.
3. Controllo autorizzazione: se `task.owner !== req.user.userId` → 403.

Checklist

- CRUD funzionanti
- Filtraggio per owner in GET
- 403 se si tenta di agire su task di altri

Test rapido
Crea due utenti, due token. Ciascuno vede/modifica/elimina solo i propri task.

TAPPA 5 — Upload di file con Multer

Goal
Allegare un’immagine a un task (opzionale ma consigliato).

Fai così

1. Configura `multer` con `dest: 'uploads/'`, `limits` e `fileFilter` (accetta solo `image/*`).
2. Rotta: `POST /api/tasks/:id/upload` (protetta).
3. Verifica proprietà prima di accettare upload (owner).
4. Salva nel Task i metadati file (nome originale, storedName/path, mimetype, size).
5. Esporre `/uploads` come static (solo per laboratorio).

Checklist

- Upload consentito solo su task dell’owner
- File salvati sotto `/uploads`
- Dati file persistiti nel DB

Test rapido
Postman → form-data → key `file` → invia immagine. Verifica risposta e percorso.

TAPPA 6 — Sicurezza “base di produzione”

Goal
Ridurre i rischi con poche mosse efficaci.

Fai così

1. `helmet()` per header di sicurezza.
2. `express-rate-limit` su `/api/auth/` (es. 100 richieste/15 min).
3. `cors` mirato (whitelist: `http://localhost:5173` o dominio frontend).
4. Risposte errori **non verbose**: niente stack in JSON.

Checklist

- Helmet attivo
- Rate limit su login/register
- CORS configurato
- Messaggi d’errore sobri

Test rapido
Simula molte richieste login veloci → oltre soglia ottieni 429 Too Many Requests.

TAPPA 7 — Logging & Error Handling

Goal
Osservabilità base e coerenza nella gestione errori.

Fai così

1. `morgan('dev')` per log HTTP (sviluppo).
2. Logger applicativo (es. winston) per errori e info di sistema.
3. Middleware `notFound` (404) + `errorHandler` (500, AppError, Joi).

Checklist

- Logger unico richiamabile
- 404 gestito con JSON coerente
- Errori centralizzati

Test rapido
Chiamare rotte inesistenti → 404 JSON. Forzare errore nei controller → finisce in `errorHandler`.

TAPPA 8 — Deployment (Render/Railway) + variabili d’ambiente

Goal
Mettere l’API online, con DB Atlas e variabili d’ambiente sicure.

Fai così

1. `package.json` con `"start": "node src/server.js"`.
2. Repo su GitHub aggiornato.
3. Piattaforma (Render/Railway): collega repo, setta env (`MONGO_URI`, `JWT_SECRET`, `NODE_ENV=production`), **non** toccare `PORT`.
4. Deploy → prendi URL pubblico.

Checklist

- `/api/health` online
- CRUD funzionanti online

- Login/token OK anche su cloud

Test rapido
Postman sul dominio pubblico: registra, logga, crea task, lista task, upload file.

TAPPA 9 — Documentazione minima (Swagger) & README

Goal
Spiegare come si usa l'API e come avviarla.

Fai così

1. Crea `docs/openapi.yaml` con almeno auth + tasks.
2. `README.md`: come si avvia, env richieste, rotte principali, URL pubblico.

Checklist

- README chiaro
- OpenAPI con esempi
- (Opzionale) Swagger UI servita su `/api/docs`

Test rapido
Un compagno non del tuo gruppo riesce ad avviare il progetto seguendo il README? Se sì, hai documentato bene.

TAPPA 10 — QA finale (Quality Assurance)

Goal
Assicurare qualità e coerenza.

Fai così

1. **Verifiche funzionali**: tutti i casi (happy path + errori comuni).
2. **Verifiche sicurezza**: CORS, rate limit, no stack trace.
3. **Verifiche struttura**: controller "snelli", service "intelligenti".
4. **Verifiche log**: ci sono informazioni sufficienti per capire i problemi?

Checklist "pronto per consegna"

- URL pubblico + repo Git
- Screenshot Postman: register, login, CRUD, upload
- Validazioni Joi attive
- Error handling coerente

- Helmet + rate-limit attivi
- README + (opzionale) Swagger

Consigli pratici

- **Chiudi una tappa alla volta**: non saltare.
- **Commits frequenti** con messaggi chiari (feat/fix/chore).
- **Testa spesso**: ogni rotta dopo averla creata.
- **Logga con criterio**: i log sono tuoi alleati.
- **Non copiare-incollare a caso**: scrivi tu il codice per capirlo davvero.