DOCUMENTAZIONE API

Autenticazione

- Attualmente: **non obbligatoria** per test (demo login con root@cowork.it / CowRoot).
- Codice di riferimento:
    - backend/src/middleware/auth.js:

      l file auth.js contiene un middleware per la gestione dell'autenticazione, che verifica la validità del token JWT in ogni richiesta per garantire l'accesso. Inoltre, definisce la funzione requireRole che, basata sul ruolo dell'utente, restringe l'accesso a specifiche parti dell'applicazione per motivi di sicurezza.

Users

- **POST /users** → registra un nuovo utente.
- **GET /users/:id** → mostra profilo utente.
- **PUT /users/:id** → aggiorna profilo.
- **DELETE /users/:id** → elimina utente.
- **Codice**: backend/src/models/user.js:

  Il file user.js definisce il modello User tramite Sequelize, specificando i campi della tabella Users nel database PostgreSQL (email, password_hash, role) e i relativi vincoli. Questo modello viene utilizzato dal backend per creare, leggere, aggiornare ed eliminare utenti.

Spaces

- **GET /spaces** → lista spazi coworking.
- **POST /spaces** → aggiunge uno spazio (solo admin).
- **PUT /spaces/:id** → aggiorna spazio.
- **DELETE /spaces/:id** → elimina spazio.
- **Codice**:
    - backend/src/controllers/locationsController.js :

  Il file **locationsController.js** definisce il **"controllore"** dell'API, ovvero la logica che elabora le richieste del client per la gestione degli spazi. Questo file contiene le funzioni che gestiscono le richieste GET e POST per gli spazi.

- backend/src/models/space.j :

Il file **space.js** definisce il **"modello"** per uno spazio, che rappresenta la struttura dei dati come sono memorizzati nel database. Questo file, che usa la libreria Sequelize, descrive le proprietà che un oggetto "Spazio" dovrebbe avere, come name, type, e capacity, e come queste sono correlate a una tabella nel database.

Bookings

- **POST /bookings** → crea una prenotazione.
- **GET /bookings/:id** → dettaglio prenotazione.
- **GET /bookings?user=:id** → prenotazioni di un utente.
- **PUT /bookings/:id** → modifica prenotazione.
- **DELETE /bookings/:id** → cancella prenotazione.
- **Codice**:

  - backend/src/controllers/bookingsController.js:

Il file **bookingsController.js** è il **"controllore"** dell'API, poiché contiene la logica che gestisce le richieste relative alle prenotazioni. Questo file contiene le funzioni che gestiscono le richieste GET e POST per le prenotazioni, operando su un array in memoria che simula un database.

  - backend/src/models/booking.js:

Il file **booking.js** è il **"modello"** per una prenotazione, che definisce la struttura dei dati come verrebbero memorizzati in un database relazionale. Questo file, che utilizza la libreria Sequelize, descrive le proprietà come user_id, space_id e start_ts che una prenotazione dovrebbe avere.

Payments

- **POST /bookings/:id/pay**: Avvia un pagamento per una specifica prenotazione. Questo endpoint è gestito dal file **payments.js** e crea una sessione di checkout con Stripe per il pagamento.
- **POST /payments/webhook**: Riceve le notifiche da Stripe al completamento di un pagamento e aggiorna lo stato della prenotazione nel database.
- **Funzionalità "auto-pay"**: È collegata alle prenotazioni. Se attiva, dopo la creazione di una prenotazione tramite il bookingsController.js, il sistema invia una richiesta a payments.js per avviare il processo di pagamento.
- **Codice**: La gestione dei pagamenti è implementata principalmente in **backend/src/routes/payments.js**, che si integra con il controller delle prenotazioni per recuperare i dettagli necessari.

Gestione errori

- **200 OK** → richiesta corretta.
- **201 Created** → risorsa creata.
- **400 Bad Request** → input mancante/errato.
- **404 Not Found** → risorsa inesistente.
- **500 Internal Server Error** → errore server.

    - Implementato in backend/src/middleware/security.js :

    Il file **backend/src/middleware/security.js** definisce un **middleware**, che è un software che elabora le richieste del client prima che raggiungano il codice dell'applicazione. In questo caso, il middleware si occupa di **sicurezza**, aggiungendo intestazioni HTTP e applicando limitazioni sul numero di richieste per prevenire attacchi, ma non gestisce direttamente i codici di stato di errore che hai menzionato.

Test

- Test API automatici: backend/jest.config.js, test con Jest:

    - **Definizione:** Questo file specifica come Jest deve eseguire i test. La sua unica riga di codice, testEnvironment: 'node', indica che i test devono essere eseguiti in un ambiente Node.js, il che è appropriato per il testing del codice backend.
    - **API nel codice:** Non ci sono API esposte in questo file. È un semplice file di configurazione che fornisce a Jest i parametri necessari per l'esecuzione dei test automatici. È l'elemento chiave che permette a Jest di testare il codice del backend in modo corretto.

- Pagamenti: simulati con **Stripe test mode**.
- Seed database: backend/scripts/seed.js:

    - **Definizione:** Questo script si connette a un database PostgreSQL utilizzando la libreria pg. Inserisce dati predefiniti nelle tabelle Users, Locations e Spaces.
    - **API nel codice:** Non ci sono API nel senso stretto del termine in questo script. Il codice utilizza le API della libreria pgper interagire con il database, eseguendo query SQL per inserire dati di test. I dati inseriti servono a "seminare" il database, rendendolo pronto per lo sviluppo, i test o un'installazione iniziale.