Documentazione API

Sistema di Gestione Spazi di Coworking

Tettamanti Andrea
Mascetti Luca
Musetti Gregorio
Vernavà Lorenzo

14 settembre 2025

Sommario

Questa documentazione descrive in dettaglio le API REST del sistema CoWorkSpace, una piattaforma per la gestione di spazi di coworking. Il documento include le specifiche degli endpoint, gli schemi dei dati, i metodi di autenticazione e gli esempi di utilizzo. L'API segue i principi REST e utilizza JSON per lo scambio dei dati, implementando un sistema completo di autenticazione JWT e autorizzazione basata sui ruoli.

Indice

Panoramica del Sistema API

Architettura Generale

L'architettura generale dell'applicazione segue il seguente schema:

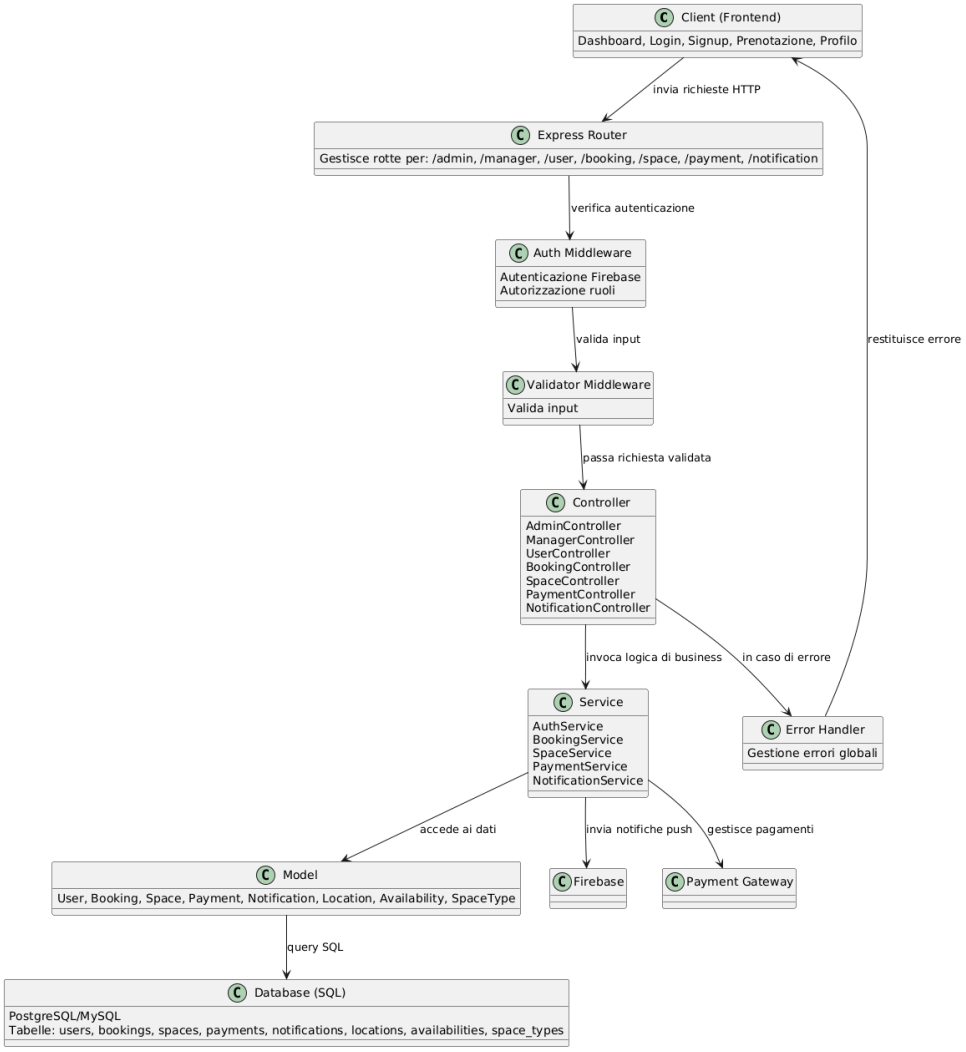

Figura 1: Architettura Generale del Sistema CoWorkSpace

Architettura REST

L'API CoWorkSpace implementa un'architettura REST (Representational State Transfer) che segue i principi di design moderni per sistemi distribuiti. Le caratteristiche principali includono:

- **Stateless**: Ogni richiesta è indipendente e contiene tutte le informazioni necessarie
- **Resource-oriented**: Gli endpoint sono organizzati attorno alle risorse del sistema
- **HTTP Methods**: Utilizzo appropriato dei verbi HTTP (GET, POST, PUT, DELETE)
- **JSON Format**: Tutti i dati sono scambiati in formato JSON

- **Consistent Response**: Struttura uniforme delle risposte API

URL Base e Versioning

```
# Development
https://localhost:3000/api

# Production
https://api.coworkspace.com/api
```

Listing 1: URL Base dell'API

L'API utilizza un approccio basato su path per l'organizzazione degli endpoint senza versioning esplicito nell'URL, seguendo il principio di evoluzione backward-compatible.

Formato delle Risposte

Tutte le risposte dell'API seguono una struttura uniforme per garantire consistenza e facilità di parsing:

1.4.1 Risposta di Successo

```
{
  "success": true,
  "message": "Operazione completata con successo",
  "data": {
    // Contenuto specifico della risposta
  }
}
```

Listing 2: Struttura Risposta di Successo

1.4.2 Risposta di Errore

```
{
  "success": false,
  "message": "Descrizione dell'errore per l'utente",
  "error": "Dettagli tecnici dell'errore",
  "errors": [
    {
      "field": "email",
      "message": "Email non valida"
    }
  ]
}
```

Listing 3: Struttura Risposta di Errore

Codici di Stato HTTP

L'API utilizza i codici di stato HTTP standard per comunicare l'esito delle operazioni:

Codice	Significato	Utilizzo nell'API
2xx - Successo		
200	OK	Operazione completata con successo
201	Created	Risorsa creata (registrazione, prenotazione)
202	Accepted	Richiesta accettata ma in elaborazione
204	No Content	Operazione completata senza contenuto
4xx - Errori Client		
400	Bad Request	Dati di input non validi
401	Unauthorized	Mancanza di autenticazione
403	Forbidden	Mancanza di autorizzazione
404	Not Found	Risorsa non trovata
409	Conflict	Conflitto con stato corrente (email già esistente)
422	Unprocessable Entity	Validazione business logic fallita
429	Too Many Requests	Rate limiting superato
5xx - Errori Server		
500	Internal Server Error	Errore interno del server
503	Service Unavailable	Servizio temporaneamente non disponibile

Tabella 1: Codici di Stato HTTP Utilizzati

Rate Limiting

Il sistema implementa rate limiting per proteggere l'API da abusi e garantire qualità del servizio:

Endpoint	Limite	Finestra Temporale
Authentication	5 tentativi	15 minuti
Password Reset	5 tentativi	15 minuti

Tabella 2: Configurazione Rate Limiting

Quando il limite viene superato, l'API restituisce un codice 429 con header informativi:

```
HTTP/1.1 429 Too Many Requests
X-RateLimit-Limit: 5
X-RateLimit-Remaining: 0
X-RateLimit-Reset: 1640995200
Retry-After: 900

{
  "success": false,
  "message": "Troppi tentativi. Riprova tra 15 minuti."
}
```

Listing 4: Response Headers Rate Limiting

CORS e Sicurezza

L'API implementa politiche CORS (Cross-Origin Resource Sharing) differenziate per ambiente:

1.7.1 Sviluppo

- Origini permesse: `localhost:3000`, `127.0.0.1:5500`
- Metodi: GET, POST, PUT, DELETE, PATCH
- Headers: Content-Type, Authorization
- Credentials: Supportate

1.7.2 Produzione

- Origini: Solo quelle specificate in `FRONTEND_URL`
- Politiche più restrittive per sicurezza
- Helmet.js per header di sicurezza

Middleware di Sicurezza

Il sistema utilizza diversi middleware per garantire la sicurezza:

Middleware	Funzione
Helmet	Content Security Policy, prevenzione attacchi XSS
CORS	Controllo accessi cross-origin
Rate Limiting	Prevenzione attacchi DoS e brute force
Input Validation	Validazione e sanitizzazione input utente
JWT Authentication	Verifica token di autenticazione
Role-based Authorization	Controllo permessi basato sui ruoli

Tabella 3: Middleware di Sicurezza Implementati

Gestione degli Errori

Il sistema implementa una gestione degli errori centralizzata con logging strutturato:

```
{
  "success": false,
  "message": "Dati non validi",
  "errors": [
    {
      "type": "field",
      "field": "email",
      "message": "Email deve essere un indirizzo valido",
      "value": "email-non-valida"
    },
    {
      "type": "field",
```

```
      "field": "password",
      "message": "Password deve contenere almeno 8 caratteri",
      "value": "***"
    }
  ]
}
```

Listing 5: Esempio Gestione Errore di Validazione

Gestione Transazioni e Rollback

Il sistema implementa transazioni database per garantire l'integrità dei dati nelle operazioni complesse:

1.10.1 Operazioni Transazionali

Le seguenti operazioni utilizzano transazioni atomiche:

- **Elaborazione Pagamenti**: Creazione pagamento + aggiornamento stato prenotazione
- **Gestione Disponibilità**: Aggiornamento multiplo slot temporali
- **Operazioni SpaceType**: Modifica tipi spazio con validazioni incrociate
- **Prenotazioni Complesse**: Creazione prenotazione + blocco disponibilità + notifiche

1.10.2 Pattern di Implementazione

```
// Utility function per transazioni
const transaction = async (callback) => {
  const client = await pool.connect();
  try {
    await client.query('BEGIN');
    const result = await callback(client);
    await client.query('COMMIT');
    return result;
  } catch (error) {
    await client.query('ROLLBACK');
    throw error;
  } finally {
    client.release();
  }
};
```

Listing 6: Esempio Transazione Automatica

1.10.3 Strategie di Rollback

- **Rollback Automatico**: In caso di errore, tutte le operazioni della transazione vengono annullate
- **Gestione Connessioni**: Rilascio automatico delle connessioni database

- **Error Propagation**: Gli errori vengono propagati al livello superiore per logging
- **Stato Consistente**: Il database rimane sempre in uno stato coerente

1.10.4 Esempio Pratico: Pagamento con Aggiornamento Prenotazione

```
try {
  await client.query('BEGIN');

  // 1. Crea il pagamento
  const payment = await Payment.create({
    booking_id, amount, payment_method,
    status: 'completed', transaction_id
  });

  // 2. Aggiorna stato prenotazione
  await Booking.update(booking_id, { status: 'confirmed' });

  await client.query('COMMIT');
  return payment;
} catch (error) {
  await client.query('ROLLBACK');
  throw error;
}
```

Listing 7: Transazione Pagamento Completa

Paginazione

Per gli endpoint che restituiscono liste di dati, l'API supporta paginazione basata su offset:

```
GET /api/bookings?page=2&limit=10&sort=created_at&order=desc
```

Listing 8: Parametri di Paginazione

```
{
  "success": true,
  "data": {
    "items": [...],
    "pagination": {
      "page": 2,
      "limit": 10,
      "total": 150,
      "totalPages": 15,
      "hasNext": true,
      "hasPrev": true
    }
  }
}
```

Listing 9: Risposta con Metadati di Paginazione

Content-Type e Accept Headers

L'API supporta esclusivamente JSON per input e output:

```
Content-Type: application/json
Accept: application/json
```

Listing 10: Headers Richiesti

Documentazione Interattiva

L'API include documentazione Swagger/OpenAPI accessibile durante lo sviluppo:

- **URL Sviluppo**: `http://localhost:3000/api-docs`
- **Formato**: OpenAPI 3.0.0
- **Caratteristiche**: Test interattivi, schemi completi, esempi

Health Check e Monitoring

Il sistema include endpoint per monitoraggio dello stato:

```
GET /api/health
```

Listing 11: Health Check Endpoint

```
{
  "success": true,
  "data": {
    "status": "ok",
    "timestamp": "2024-01-15T10:30:00Z",
    "version": "1.0.0",
    "database": "connected",
    "uptime": 3600
  }
}
```

Listing 12: Risposta Health Check

Autenticazione e Autorizzazione

Sistema di Autenticazione JWT

L'API utilizza JSON Web Tokens (JWT) per l'autenticazione degli utenti. Il sistema implementa un approccio stateless che garantisce scalabilità e sicurezza.

2.1.1 Processo di Login

1. L'utente invia credenziali (email e password)
2. Il server verifica le credenziali contro il database
3. Se valide, genera un JWT token con payload utente
4. Il token viene restituito al client per le richieste successive

```
POST /api/users/login
Content-Type: application/json

{
  "email": "mario.rossi@email.com",
  "password": "password123"
}
```

Listing 13: Richiesta di Login

```
{
  "success": true,
  "message": "Login effettuato con successo",
  "data": {
    "user": {
      "user_id": 1,
      "name": "Mario",
      "surname": "Rossi",
      "email": "mario.rossi@email.com",
      "role": "user",
      "created_at": "2024-01-15T10:30:00Z"
    },
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}
```

Listing 14: Risposta Login Successful

2.1.2 Struttura del JWT Token

Il JWT token contiene le seguenti informazioni nel payload:

```
{
  "user_id": 1,
  "email": "mario.rossi@email.com",
  "role": "user",
  "iat": 1640995200,  // Issued at
  "exp": 1641081600   // Expiration (24 ore)
}
```

Listing 15: Payload JWT Token

Utilizzo del Token

Per accedere agli endpoint protetti, il client deve includere il token nell'header Authorization:

```
GET /api/users/profile
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Content-Type: application/json
```

Listing 16: Header Authorization

Sistema di Autorizzazione Basato sui Ruoli

Il sistema implementa tre livelli di autorizzazione basati sui ruoli utente:

2.3.1 Ruoli Disponibili

Ruolo	Descrizione
user	Utente standard che può visualizzare spazi disponibili, creare prenotazioni e gestire il proprio profilo
manager	Manager di location che ha tutti i permessi utente plus gestione degli spazi nelle proprie location, visualizzazione prenotazioni della propria location
admin	Amministratore sistema con accesso completo: gestione utenti, promozione manager, gestione globale del sistema

Tabella 4: Ruoli Utente del Sistema

2.3.2 Matrice dei Permessi

Operazione	Pubblico	User	Manager	Admin
Visualizzare spazi pubblici	✓	✓	✓	✓
Registrazione/Login	✓	✓	✓	✓
Gestire proprio profilo	–	✓	✓	✓
Creare prenotazioni	–	✓	✓	✓
Visualizzare proprie prenotazioni	–	✓	✓	✓
Gestire spazi propria location	–	–	✓	✓
Visualizzare prenotazioni location	–	–	✓	✓
Gestire disponibilità	–	–	✓	✓
Gestire tutti gli utenti	–	–	–	✓
Promuovere manager	–	–	–	✓
Gestire tutte le location	–	–	–	✓
Accesso dashboard admin	–	–	–	✓

Tabella 5: Matrice Permessi per Ruolo

Middleware di Autenticazione

Il sistema utilizza middleware personalizzati per proteggere gli endpoint:

2.4.1 authMiddleware.protect

Verifica la presenza e validità del JWT token:

```
// Verifica presenza header Authorization
// Estrae il token dal formato "Bearer <token>"
// Verifica validità del token JWT
// Decodifica payload e ottiene informazioni utente
// Aggiunge req.user per controller successivi
```

Listing 17: Funzionamento authMiddleware.protect

2.4.2 authMiddleware.authorize

Controlla i permessi basati sui ruoli:

```
// Solo manager e admin possono accedere
router.get('/spaces',
  authMiddleware.protect,
  authMiddleware.authorize(['manager', 'admin']),
  spaceController.getSpaces
);

// Solo admin può accedere
router.get('/admin/users',
  authMiddleware.protect,
  authMiddleware.authorize(['admin']),
  adminController.getAllUsers
);
```

Listing 18: Esempi Utilizzo Authorization

Gestione delle Password

Il sistema implementa pratiche di sicurezza avanzate per la gestione delle password:

2.5.1 Hashing delle Password

- Utilizza bcrypt con salt factor 12
- Le password non vengono mai memorizzate in plain text
- Hash diversi anche per password identiche grazie al salt

2.5.2 Reset Password

Il sistema supporta due modalità di reset password:

1. **Password dimenticata**: Processo pubblico via email
2. **Cambio password**: Dal profilo utente autenticato

```
POST /api/users/request-password-reset
Content-Type: application/json

{
  "email": "mario.rossi@email.com"
}
```

Listing 19: Richiesta Reset Password

Rate Limiting per Sicurezza

Protezione specifica per endpoint di autenticazione:

Endpoint	Limite	Protezione
/users/login	5 richieste/15min per IP	Attacchi brute force
/users/request-password-reset	5 richieste/15min per IP	Spam di reset
/users/register	Rate limiting generale	Registrazioni massive

Tabella 6: Rate Limiting Endpoints Autenticazione

Gestione Token FCM

Il sistema supporta notifiche push tramite Firebase Cloud Messaging:

```
POST /api/users/fcm-token
Authorization: Bearer <jwt-token>
Content-Type: application/json

{
  "fcm_token": "fGH1jK2L3m4N5o6P7q8R9s0T..."
}
```

Listing 20: Salvataggio Token FCM

Logout e Invalidazione Token

Benché JWT sia stateless, il sistema supporta logout lato client:

```
POST /api/users/logout
Authorization: Bearer <jwt-token>
```

Listing 21: Endpoint Logout

```
{
  "success": true,
  "message": "Logout effettuato con successo",
  "data": {
    "message": "Token invalidato lato client"
  }
}
```

Listing 22: Risposta Logout

Sicurezza delle Sessioni

- **Durata Token**: 24 ore (configurabile)
- **Rinnovo**: Automatic refresh non implementato per sicurezza
- **Scope**: Un token per tutte le operazioni autorizzate
- **Revoca**: Solo tramite scadenza naturale

Validazione Input

Tutti gli endpoint implementano validazione rigorosa degli input:

```
// Email: formato email valido, lunghezza massima 255
// Password: minimo 6 caratteri, massimo 255
// Rate limiting: 5 tentativi per IP ogni 15 minuti
// Sanitizzazione: rimozione caratteri pericolosi
```

Listing 23: Esempio Validazioni Login

Gestione Errori di Autenticazione

Il sistema fornisce messaggi di errore specifici ma sicuri:

Scenario	Codice	Messaggio
Token mancante	401	"Token di accesso richiesto"
Token scaduto	401	"Token scaduto, effettua nuovamente il login"
Token non valido	401	"Token non valido"
Permessi insufficienti	403	"Non hai i permessi per questa operazione"
Credenziali errate	401	"Credenziali non valide"
Account sospeso	401	"Account in attesa di approvazione"

Tabella 7: Messaggi di Errore Autenticazione

Endpoints API

Panoramica degli Endpoints

L'API CoWorkSpace è organizzata in 10 categorie principali di endpoint, ciascuna dedicata a specifiche funzionalità del sistema:

Categoria	Descrizione	Base Path
Users	Gestione utenti e autenticazione	/api/users
Locations	Gestione sedi coworking	/api/locations
Spaces	Gestione spazi prenotabili	/api/spaces
Space Types	Gestione tipologie di spazio	/api/space-types
Bookings	Gestione prenotazioni	/api/bookings
Availability	Gestione disponibilità spazi	/api/availability
Payments	Gestione pagamenti	/api/payments
Notifications	Gestione notifiche	/api/notifications
Manager	Endpoint specifici per manager	/api/manager
Admin	Endpoint amministrativi	/api/admin

Tabella 8: Categorie di Endpoint API

Endpoint Users

Gestione degli utenti, autenticazione e profili.

3.2.1 `POST /api/users/register`

Descrizione: Registrazione nuovo utente
Autenticazione: Pubblica
Parametri Body:

- **email** (*string*): Email utente (formato email valido)
- **password** (*string*): Password (minimo 8 caratteri)
- **name** (*string*): Nome utente (massimo 100 caratteri)
- **surname** (*string*): Cognome utente (massimo 100 caratteri)
- **requestManagerRole** (*boolean*): Richiesta ruolo manager (opzionale)

Risposte:

- **201**: Utente registrato con successo (può effettuare login)
- **202**: Utente registrato, in attesa approvazione manager
- **400**: Dati non validi
- **409**: Email già esistente

```
POST /api/users/register

{
  "email": "mario.rossi@email.com",
  "password": "Password123!",
  "name": "Mario",
  "surname": "Rossi",
  "requestManagerRole": false
}
```

Listing 24: Esempio Registrazione Utente

3.2.2 `POST /api/users/login`

Descrizione: Login utente
Autenticazione: Pubblica + Rate Limiting
Parametri Body:

- **email** (*string*): Email registrata
- **password** (*string*): Password utente

Risposte:

- **200**: Login effettuato con successo
- **401**: Credenziali non valide
- **429**: Troppi tentativi di login

3.2.3 `GET /api/users/profile`

Descrizione: Ottieni profilo utente corrente
Autenticazione: JWT Required

3.2.4 `GET /api/users/dashboard`

Descrizione: Dashboard con statistiche e prenotazioni
Autenticazione: JWT Required

3.2.5 `PUT /api/users/profile`

Descrizione: Aggiorna profilo utente
Autenticazione: JWT Required

3.2.6 `PUT /api/users/change-password`

Descrizione: Cambia password utente
Autenticazione: JWT Required

3.2.7 `POST /api/users/request-password-reset`

Descrizione: Richiesta reset password
Autenticazione: Pubblica + Rate Limiting

3.2.8 `POST /api/users/logout`

Descrizione: Logout utente
Autenticazione: JWT Required

3.2.9 `POST /api/users/fcm-token`

Descrizione: Salva token FCM per notifiche push
Autenticazione: JWT Required

3.2.10 `GET /api/users/check-email`

Descrizione: Verifica se email è registrata
Autenticazione: Pubblica

Endpoint Locations

Gestione delle sedi coworking.

3.3.1 `GET /api/locations`

Descrizione: Lista tutte le location
Autenticazione: Pubblica
Parametri Query (opzionali):

- **city** (*string*): Filtra per città

- **search** (*string*): Ricerca nel nome o descrizione
- **manager_id** (*integer*): Filtra per manager

3.3.2 `GET /api/locations/:id`

Descrizione: Dettagli specifica location
Autenticazione: Pubblica

3.3.3 `GET /api/locations/:id/spaces`

Descrizione: Spazi di una location
Autenticazione: Pubblica

3.3.4 `POST /api/locations`

Descrizione: Crea nuova location
Autenticazione: Admin Only

3.3.5 `PUT /api/locations/:id`

Descrizione: Aggiorna location
Autenticazione: Admin Only

3.3.6 `DELETE /api/locations/:id`

Descrizione: Elimina location
Autenticazione: Admin Only

Endpoint Spaces

Gestione degli spazi prenotabili.

3.4.1 `GET /api/spaces`

Descrizione: Lista spazi con filtri avanzati
Autenticazione: Pubblica
Parametri Query (opzionali):

- **location_id** (*integer*): Filtra per location
- **space_type_id** (*integer*): Filtra per tipo spazio
- **capacity_min** (*integer*): Capacità minima
- **capacity_max** (*integer*): Capacità massima
- **price_max** (*number*): Prezzo massimo giornaliero
- **available_date** (*date*): Data disponibilità
- **city** (*string*): Filtra per città location

- **search** (*string*): Ricerca nel nome

```
GET
   /api/spaces?location_id=1&capacity_min=4&price_max=150&available_date=2024-01-20
```

Listing 25: Esempio Ricerca Spazi con Filtri

3.4.2 `GET /api/spaces/:id`

Descrizione: Dettagli specifico spazio
Autenticazione: Pubblica

3.4.3 `GET /api/spaces/:id/availability`

Descrizione: Disponibilità spazio per periodo
Autenticazione: Pubblica

3.4.4 `POST /api/spaces`

Descrizione: Crea nuovo spazio
Autenticazione: Manager/Admin

3.4.5 `PUT /api/spaces/:id`

Descrizione: Aggiorna spazio
Autenticazione: Manager/Admin

3.4.6 `DELETE /api/spaces/:id`

Descrizione: Elimina spazio
Autenticazione: Manager/Admin

Endpoint Space Types

Gestione tipologie di spazio.

3.5.1 `GET /api/space-types`

Descrizione: Lista tutte le tipologie
Autenticazione: Pubblica

3.5.2 `GET /api/space-types/:id`

Descrizione: Dettagli tipologia specifica
Autenticazione: Pubblica

3.5.3 `POST /api/space-types`

Descrizione: Crea nuova tipologia
Autenticazione: Admin Only

3.5.4 `PUT /api/space-types/:id`

Descrizione: Aggiorna tipologia
Autenticazione: Admin Only

3.5.5 `DELETE /api/space-types/:id`

Descrizione: Elimina tipologia
Autenticazione: Admin Only

Endpoint Bookings

Gestione del sistema di prenotazioni.

3.6.1 `POST /api/bookings/check-availability`

Descrizione: Verifica disponibilità spazio
Autenticazione: Pubblica
Parametri Body:

- **space_id** (*integer*): ID dello spazio
- **start_date** (*date*): Data inizio prenotazione
- **end_date** (*date*): Data fine prenotazione

3.6.2 `POST /api/bookings`

Descrizione: Crea nuova prenotazione
Autenticazione: JWT Required
Parametri Body:

- **space_id** (*integer*): ID dello spazio da prenotare
- **start_date** (*date*): Data inizio (formato YYYY-MM-DD)
- **end_date** (*date*): Data fine (formato YYYY-MM-DD)
- **notes** (*string*): Note aggiuntive (opzionale)

```
POST /api/bookings
Authorization: Bearer <jwt-token>

{
  "space_id": 1,
  "start_date": "2024-01-20",
  "end_date": "2024-01-22",
  "notes": "Riunione team marketing"
}
```

Listing 26: Esempio Creazione Prenotazione

3.6.3 `GET /api/bookings`

Descrizione: Lista prenotazioni utente corrente
Autenticazione: JWT Required
Parametri Query (opzionali):

- **status** (*string*): Filtra per stato (confirmed, pending, cancelled, completed)
- **from_date** (*date*): Prenotazioni da questa data
- **to_date** (*date*): Prenotazioni fino a questa data
- **page** (*integer*): Numero pagina (default: 1)
- **limit** (*integer*): Elementi per pagina (default: 10)

3.6.4 `GET /api/bookings/:id`

Descrizione: Dettagli prenotazione specifica
Autenticazione: JWT Required

3.6.5 `PUT /api/bookings/:id`

Descrizione: Aggiorna prenotazione
Autenticazione: JWT Required

3.6.6 `DELETE /api/bookings/:id`

Descrizione: Cancella prenotazione
Autenticazione: JWT Required

3.6.7 `PUT /api/bookings/:id/confirm`

Descrizione: Conferma prenotazione
Autenticazione: Manager/Admin

3.6.8 `PUT /api/bookings/:id/complete`

Descrizione: Completa prenotazione
Autenticazione: Manager/Admin

Endpoint Availability

Gestione disponibilità degli spazi.

3.7.1 `GET /api/availability/space/:spaceId`

Descrizione: Disponibilità spazio per periodo
Autenticazione: Pubblica
Parametri Query:

- **from_date** (*date*): Data inizio periodo
- **to_date** (*date*): Data fine periodo

3.7.2 `POST /api/availability`

Descrizione: Imposta disponibilità spazio
Autenticazione: Manager/Admin

3.7.3 `PUT /api/availability/:id`

Descrizione: Aggiorna disponibilità
Autenticazione: Manager/Admin

3.7.4 `DELETE /api/availability/:id`

Descrizione: Rimuovi disponibilità
Autenticazione: Manager/Admin

Endpoint Payments

Gestione sistema di pagamenti.

3.8.1 `GET /api/payments`

Descrizione: Lista pagamenti utente corrente
Autenticazione: JWT Required

3.8.2 `GET /api/payments/:id`

Descrizione: Dettagli pagamento specifico
Autenticazione: JWT Required

3.8.3 `POST /api/payments`

Descrizione: Crea nuovo pagamento
Autenticazione: JWT Required
Parametri Body:

- **booking_id** (*integer*): ID prenotazione da pagare
- **payment_method** (*string*): Metodo: credit_card (solo carta di credito)
- **amount** (*number*): Importo pagamento
- **transaction_id** (*string*): ID transazione gateway (opzionale)

3.8.4 `PUT /api/payments/:id/refund`

Descrizione: Rimborsa pagamento
Autenticazione: Manager/Admin

Endpoint Notifications

Gestione sistema notifiche.

3.9.1 `GET /api/notifications`

Descrizione: Lista notifiche utente corrente
Autenticazione: JWT Required

3.9.2 `GET /api/notifications/:id`

Descrizione: Dettagli notifica specifica
Autenticazione: JWT Required

3.9.3 `PUT /api/notifications/:id/read`

Descrizione: Segna notifica come letta
Autenticazione: JWT Required

3.9.4 `POST /api/notifications/test`

Descrizione: Invia notifica di test
Autenticazione: Admin Only

Endpoint Manager

Funzionalità specifiche per manager.

3.10.1 `GET /api/manager/dashboard`

Descrizione: Dashboard manager con statistiche
Autenticazione: Manager/Admin

3.10.2 `GET /api/manager/location`

Descrizione: Location gestita dal manager
Autenticazione: Manager/Admin

3.10.3 `GET /api/manager/bookings`

Descrizione: Prenotazioni location del manager
Autenticazione: Manager/Admin

3.10.4 `GET /api/manager/spaces`

Descrizione: Spazi della location del manager
Autenticazione: Manager/Admin

3.10.5 `PUT /api/manager/spaces/:id`

Descrizione: Aggiorna spazio della propria location
Autenticazione: Manager/Admin

Endpoint Admin

Funzionalità amministrative del sistema.

3.11.1 `GET /api/admin/dashboard`

Descrizione: Dashboard amministrativa completa
Autenticazione: Admin Only

3.11.2 `GET /api/admin/users`

Descrizione: Lista tutti gli utenti
Autenticazione: Admin Only

3.11.3 `GET /api/admin/users/pending-managers`

Descrizione: Utenti in attesa approvazione manager
Autenticazione: Admin Only

3.11.4 `PUT /api/admin/users/:id/promote-manager`

Descrizione: Promuovi utente a manager
Autenticazione: Admin Only

3.11.5 `PUT /api/admin/users/:id/demote`

Descrizione: Rimuovi ruolo manager
Autenticazione: Admin Only

3.11.6 `GET /api/admin/bookings`

Descrizione: Tutte le prenotazioni sistema
Autenticazione: Admin Only

3.11.7 `GET /api/admin/stats`

Descrizione: Statistiche globali sistema
Autenticazione: Admin Only

3.11.8 `POST /api/admin/locations`

Descrizione: Crea nuova location
Autenticazione: Admin Only

3.11.9 `PUT /api/admin/locations/:id/assign-manager`

Descrizione: Assegna manager a location
Autenticazione: Admin Only

Endpoint di Sistema

3.12.1 `GET /api/health`

Descrizione: Health check sistema
Autenticazione: Pubblica

```
{
  "success": true,
  "data": {
    "status": "ok",
    "timestamp": "2024-01-15T10:30:00Z",
    "version": "1.0.0",
    "database": "connected",
    "uptime": 3600
  }
}
```

Listing 27: Risposta Health Check

3.12.2 `GET /api/`

Descrizione: Informazioni generali API
Autenticazione: Pubblica

3.12.3 `GET /api-docs`

Descrizione: Documentazione Swagger
Autenticazione: Sviluppo

Convenzioni degli Endpoint

L'API segue convenzioni REST standard:

Metodo	Utilizzo	Esempio
GET	Lettura risorse	`GET /api/spaces` - Lista spazi
POST	Creazione risorse	`POST /api/bookings` - Nuova prenotazione
PUT	Aggiornamento completo	`PUT /api/spaces/:id` - Aggiorna spazio
PATCH	Aggiornamento parziale	`PATCH /api/users/profile` - Aggiorna profilo
DELETE	Eliminazione risorse	`DELETE /api/bookings/:id` - Cancella prenotazione

Tabella 9: Convenzioni Metodi HTTP

Schemi Dati e Validazioni

Panoramica degli Schemi

L'API utilizza schemi dati strutturati che corrispondono alle entità del database. Ogni schema implementa validazioni rigorose per garantire integrità e sicurezza dei dati.

Schema User

Rappresenta gli utenti del sistema con i loro ruoli e permessi.

```
{
  "user_id": 1,
  "name": "Mario",
  "surname": "Rossi",
  "email": "mario.rossi@email.com",
  "role": "user",
  "is_password_reset_required": false,
  "fcm_token": "fGH1jK2L3m4N5o6P7q8R9s0T...",
  "manager_request_pending": false,
  "manager_request_date": null,
  "created_at": "2024-01-15T10:30:00Z"
}
```

Listing 28: Schema User Completo

Validazioni Campo User:

Campo	Tipo	Validazioni	Vincoli
name	string	Lunghezza 1-100 caratteri	Obbligatorio
surname	string	Lunghezza 1-100 caratteri	Obbligatorio
email	string	Formato email valido, max 255 caratteri	Unico, obbligatorio
password	string	Minimo 8 caratteri, hash bcrypt	Obbligatorio
role	enum	user, manager, admin	Default: user
fcm_token	string	Max 255 caratteri	Opzionale

Tabella 10: Validazioni Schema User

Schema Location

Rappresenta le sedi fisiche dei coworking.

```
{
  "location_id": 1,
  "location_name": "CoWork Milano Centro",
  "address": "Via Brera 10, Milano",
  "city": "Milano",
  "description": "Moderno spazio coworking nel cuore di Milano",
  "manager_id": 2,
  "manager": {
    "user_id": 2,
    "name": "Luca",
    "surname": "Manager",
    "email": "luca.manager@email.com"
  },
  "spaces_count": 15,
  "active_bookings": 8
}
```

Listing 29: Schema Location

Validazioni Campo Location:

Campo	Tipo	Validazioni	Vincoli
location_name	string	Max 255 caratteri, non vuoto	Obbligatorio
address	string	Max 255 caratteri, non vuoto	Obbligatorio
city	string	Max 100 caratteri, non vuoto	Obbligatorio
description	text	Testo libero	Opzionale
manager_id	integer	Riferimento utente con ruolo manager	Opzionale

Tabella 11: Validazioni Schema Location

Schema Space Type

Definisce i tipi di spazio disponibili.

```
{
  "space_type_id": 1,
  "type_name": "Ufficio Privato",
  "description": "Ufficio privato per singola persona o piccoli team",
  "spaces_count": 25
}
```

Listing 30: Schema Space Type

Tipi Predefiniti del Sistema:

Tipo	Descrizione
Ufficio Privato	Ufficio privato per singola persona o piccoli team
Sala Riunioni	Sala attrezzata per meeting e riunioni di lavoro
Open Space	Spazio aperto condiviso per lavoro collaborativo
Coworking Desk	Singola postazione di lavoro in ambiente condiviso
Phone Booth	Cabina telefonica insonorizzata per chiamate private
Sala Conferenze	Ampia sala per conferenze e presentazioni
Focus Room	Stanza silenziosa per lavoro concentrato
Lounge Area	Area relax informale per incontri casual
Training Room	Aula per formazione e workshop
Event Space	Spazio per eventi e networking

Tabella 12: Tipologie di Spazio Predefinite

Schema Space

Rappresenta gli spazi prenotabili all'interno delle location.

```
{
  "space_id": 1,
  "location_id": 1,
  "space_type_id": 1,
  "space_name": "Stanza 101",
  "description": "Ufficio privato con vista sul cortile",
  "capacity": 4,
  "price_per_hour": 15.50,
  "price_per_day": 120.00,
  "opening_time": "09:00:00",
  "closing_time": "18:00:00",
  "available_days": [1, 2, 3, 4, 5],
  "booking_advance_days": 30,
  "status": "active",
  "location": {
    "location_name": "CoWork Milano Centro",
    "city": "Milano",
    "address": "Via Brera 10, Milano"
  },
  "space_type": {
    "type_name": "Ufficio Privato",
    "description": "Ufficio privato per singola persona o piccoli team"
  }
}
```

Listing 31: Schema Space Completo

Validazioni Campo Space:

Campo	Tipo	Validazioni	Vincoli
space_name	string	Max 255 caratteri, non vuoto	Obbligatorio
capacity	integer	Maggiore di 0, massimo 100	Obbligatorio
price_per_hour	decimal	Maggiore di 0, max 2 decimali	Obbligatorio
price_per_day	decimal	Maggiore di 0, max 2 decimali	Obbligatorio
opening_time	time	Formato HH:MM:SS	Default: 09:00:00
closing_time	time	Formato HH:MM:SS, dopo opening_time	Default: 18:00:00
available_days	array	Numeri 1-7 (1=Lunedì, 7=Domenica)	Default: [1,2,3,4,5,6,7]
status	enum	active, maintenance, inactive	Default: active

Tabella 13: Validazioni Schema Space

Schema Booking

Rappresenta le prenotazioni degli spazi.

```
{
  "booking_id": 1,
  "user_id": 1,
  "space_id": 1,
  "start_date": "2024-01-20",
  "end_date": "2024-01-22",
  "total_days": 3,
  "total_price": 360.00,
  "status": "confirmed",
  "payment_status": "completed",
  "notes": "Riunione team marketing",
  "created_at": "2024-01-15T10:30:00Z",
  "user": {
    "name": "Mario",
    "surname": "Rossi",
    "email": "mario.rossi@email.com"
  },
  "space": {
    "space_name": "Stanza 101",
    "capacity": 4,
    "price_per_day": 120.00,
    "location": {
      "location_name": "CoWork Milano Centro",
      "city": "Milano"
    }
  },
  "payment": {
    "payment_id": 1,
    "amount": 360.00,
    "payment_method": "credit_card",
    "status": "completed",
    "payment_date": "2024-01-15T10:35:00Z"
  }
}
```

Listing 32: Schema Booking Completo

Stati Prenotazione:

Stato	Descrizione
pending	Prenotazione creata, in attesa di conferma/pagamento
confirmed	Prenotazione confermata e pagata
completed	Prenotazione utilizzata e completata
cancelled	Prenotazione cancellata dall'utente o sistema

Tabella 14: Stati Prenotazione

Validazioni Campo Booking:

Campo	Tipo	Validazioni	Vincoli
start_date	date	Formato YYYY-MM-DD, >= data corrente	Obbligatorio
end_date	date	Formato YYYY-MM-DD, >= start_date	Obbligatorio
total_days	integer	Calcolato automaticamente	Auto-generato
total_price	decimal	Calcolato automaticamente	Auto-generato
status	enum	pending, confirmed, cancelled, completed	Default: pending
notes	text	Max 1000 caratteri	Opzionale

Tabella 15: Validazioni Schema Booking

Schema Payment

Gestisce i pagamenti delle prenotazioni.

```
{
  "payment_id": 1,
  "booking_id": 1,
  "amount": 360.00,
  "payment_date": "2024-01-15T10:35:00Z",
  "payment_method": "credit_card",
  "status": "completed",
  "transaction_id": "txn_1234567890",
  "booking": {
    "booking_id": 1,
    "start_date": "2024-01-20",
    "end_date": "2024-01-22",
    "space_name": "Stanza 101"
  }
}
```

Listing 33: Schema Payment

Metodi di Pagamento:

Metodo	Descrizione
credit_card	Pagamento con carta di credito

Tabella 16: Metodi di Pagamento Supportati

Stati Pagamento:

Stato	Descrizione
pending	Pagamento in attesa di elaborazione
completed	Pagamento completato con successo
failed	Pagamento fallito
refunded	Pagamento rimborsato

Tabella 17: Stati Pagamento

Schema Notification

Sistema completo di notifiche multi-canale.

```json
{
  "notification_id": 1,
  "user_id": 1,
  "type": "email",
  "channel": "booking_confirmation",
  "recipient": "mario.rossi@email.com",
  "subject": "Conferma prenotazione #1",
  "content": "La tua prenotazione è stata confermata",
  "template_name": "booking_confirmation.html",
  "template_data": {
    "userName": "Mario",
    "spaceName": "Stanza 101",
    "startDate": "2024-01-20",
    "endDate": "2024-01-22"
  },
  "status": "delivered",
  "metadata": {
    "provider": "sendgrid",
    "messageId": "abc123"
  },
  "booking_id": 1,
  "sent_at": "2024-01-15T10:35:00Z",
  "delivered_at": "2024-01-15T10:36:00Z",
  "read_at": null,
  "retry_count": 0,
  "created_at": "2024-01-15T10:35:00Z"
}
```

Listing 34: Schema Notification

Tipi di Notifica:

Tipo	Descrizione
email	Notifica via email
push	Notifica push mobile
sms	Notifica SMS

Tabella 18: Tipi di Notifica

Canali di Notifica:

Canale	Descrizione
booking_confirmation	Conferma prenotazione
booking_cancellation	Cancellazione prenotazione
payment_success	Pagamento riuscito
payment_failed	Pagamento fallito
payment_refund	Rimborso pagamento
booking_reminder	Promemoria prenotazione
user_registration	Benvenuto nuovi utenti
password_reset	Reset password

Tabella 19: Canali di Notifica

Schema Availability

Gestisce la disponibilità giornaliera degli spazi.

```
{
  "availability_id": 1,
  "space_id": 1,
  "availability_date": "2024-01-20",
  "is_available": true,
  "space": {
    "space_name": "Stanza 101",
    "capacity": 4,
    "price_per_day": 120.00
  }
}
```

Listing 35: Schema Availability

Validazioni Globali del Sistema

4.10.1 Validazioni Date

- Tutte le date devono essere in formato ISO 8601 (YYYY-MM-DD)
- Le date di prenotazione non possono essere nel passato
- La data fine deve essere maggiore o uguale alla data inizio
- Massimo 90 giorni di anticipo per le prenotazioni

4.10.2 Validazioni Email

```
^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$
```

Listing 36: Regex Validazione Email

4.10.3 Validazioni Password

- Lunghezza minima: 8 caratteri
- Deve contenere almeno una lettera maiuscola
- Deve contenere almeno una lettera minuscola
- Deve contenere almeno un numero
- Caratteri speciali consigliati ma non obbligatori

4.10.4 Validazioni Numeriche

- Prezzi: massimo 2 decimali, maggiori di 0
- Capacità: numeri interi positivi, massimo 100
- ID: numeri interi positivi

Gestione Errori di Validazione

Il sistema restituisce errori di validazione strutturati:

```
{
  "success": false,
  "message": "Dati non validi",
  "errors": [
    {
      "type": "field",
      "field": "email",
      "message": "Email deve essere un indirizzo valido",
      "value": "email-non-valida",
      "location": "body"
    },
    {
      "type": "field",
      "field": "start_date",
      "message": "Data inizio non può essere nel passato",
      "value": "2023-01-01",
      "location": "body"
    },
    {
      "type": "business_rule",
      "message": "Lo spazio non è disponibile nelle date selezionate",
      "details": {
        "space_id": 1,
        "conflicting_dates": ["2024-01-20", "2024-01-21"]
      }
    }
  ]
}
```

Listing 37: Esempio Errore Validazione Multipla

Esempi di Utilizzo e Casi d'Uso

Panoramica dei Casi d'Uso

Questa sezione presenta esempi pratici di utilizzo dell'API CoWorkSpace, organizzati per scenari comuni di integrazione. Gli esempi includono le chiamate API complete con headers, parametri e risposte.

Caso d'Uso 1: Registrazione e Login Utente

Scenario completo di registrazione di un nuovo utente e successivo login.

5.2.1 Passo 1: Verifica Email Disponibilità

```
GET /api/users/check-email?email=mario.rossi@email.com
Content-Type: application/json
```

Listing 38: Verifica Email Esistente

```
{
  "success": true,
  "data": {
    "email": "mario.rossi@email.com",
    "exists": false
  }
}
```

Listing 39: Risposta Email Non Esistente

5.2.2 Passo 2: Registrazione Nuovo Utente

```
POST /api/users/register
Content-Type: application/json

{
  "email": "mario.rossi@email.com",
  "password": "Password123!",
  "name": "Mario",
  "surname": "Rossi",
  "requestManagerRole": false
}
```

Listing 40: Registrazione Utente

```json
{
  "success": true,
  "message": "Utente registrato con successo",
  "data": {
    "user": {
      "user_id": 15,
      "name": "Mario",
      "surname": "Rossi",
      "email": "mario.rossi@email.com",
      "role": "user",
      "created_at": "2024-01-15T10:30:00Z"
    },
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "canLogin": true
  }
}
```

Listing 41: Risposta Registrazione Riuscita

5.2.3 Passo 3: Login Utente

```
POST /api/users/login
Content-Type: application/json

{
  "email": "mario.rossi@email.com",
  "password": "Password123!"
}
```

Listing 42: Login Utente

Caso d'Uso 2: Ricerca e Prenotazione Spazio

Workflow completo di ricerca spazi disponibili e creazione prenotazione.

5.3.1 Passo 1: Ricerca Spazi con Filtri

```
GET
   /api/spaces?city=Milano&capacity_min=4&price_max=150&available_date=2024-01-20
Content-Type: application/json
```

Listing 43: Ricerca Spazi Disponibili


```
{
  "success": true,
  "data": {
    "spaces": [
      {
        "space_id": 1,
        "space_name": "Stanza 101",
        "capacity": 6,
        "price_per_day": 120.00,
        "price_per_hour": 15.50,
        "description": "Ufficio privato con vista sul cortile",
        "location": {
          "location_id": 1,
          "location_name": "CoWork Milano Centro",
          "city": "Milano",
          "address": "Via Brera 10, Milano"
        },
        "space_type": {
          "type_name": "Ufficio Privato"
        },
        "availability": {
          "is_available": true,
          "next_available_date": "2024-01-20"
        }
      }
    ],
    "total": 1,
    "filters_applied": {
      "city": "Milano",
      "capacity_min": 4,
      "price_max": 150,
      "available_date": "2024-01-20"
    }
  }
}
```


Listing 44: Risposta Spazi Disponibili

5.3.2 Passo 2: Verifica Disponibilità Specifica

```
POST /api/bookings/check-availability
Content-Type: application/json

{
  "space_id": 1,
  "start_date": "2024-01-20",
  "end_date": "2024-01-22"
}
```

Listing 45: Verifica Disponibilità Periodo

```
{
  "success": true,
  "data": {
    "available": true,
    "space_id": 1,
    "period": {
      "start_date": "2024-01-20",
      "end_date": "2024-01-22",
      "total_days": 3
    },
    "pricing": {
      "price_per_day": 120.00,
      "total_price": 360.00,
      "currency": "EUR"
    },
    "space_details": {
      "space_name": "Stanza 101",
      "capacity": 6,
      "location_name": "CoWork Milano Centro"
    }
  }
}
```

Listing 46: Risposta Disponibilità Confermata

5.3.3 Passo 3: Creazione Prenotazione

```
POST /api/bookings
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Content-Type: application/json

{
  "space_id": 1,
  "start_date": "2024-01-20",
  "end_date": "2024-01-22",
  "notes": "Riunione team marketing - progetto Q1"
}
```

Listing 47: Creazione Prenotazione

```
{
  "success": true,
  "message": "Prenotazione creata con successo",
  "data": {
    "booking": {
      "booking_id": 25,
      "user_id": 15,
      "space_id": 1,
      "start_date": "2024-01-20",
      "end_date": "2024-01-22",
      "total_days": 3,
      "total_price": 360.00,
      "status": "pending",
      "payment_status": "pending",
      "notes": "Riunione team marketing - progetto Q1",
      "created_at": "2024-01-15T11:00:00Z"
    },
    "next_steps": {
      "payment_required": true,
      "payment_deadline": "2024-01-16T11:00:00Z"
    }
  }
}
```

Listing 48: Risposta Prenotazione Creata

Caso d'Uso 3: Gestione Pagamento

Workflow di pagamento per confermare una prenotazione.

5.4.1 Passo 1: Creazione Pagamento

```
POST /api/payments
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Content-Type: application/json

{
  "booking_id": 25,
  "payment_method": "credit_card",
  "amount": 360.00,
  "transaction_id": "txn_cc_1234567890"
}
```

Listing 49: Processamento Pagamento

```
{
  "success": true,
  "message": "Pagamento completato con successo",
  "data": {
    "payment": {
      "payment_id": 18,
      "booking_id": 25,
      "amount": 360.00,
      "payment_method": "credit_card",
      "status": "completed",
      "transaction_id": "txn_cc_1234567890",
      "payment_date": "2024-01-15T11:05:00Z"
    },
    "booking_updated": {
      "booking_id": 25,
      "status": "confirmed",
      "payment_status": "completed"
    },
    "notifications_sent": [
      "booking_confirmation_email",
      "payment_success_email"
    ]
  }
}
```


Listing 50: Risposta Pagamento Completato

Caso d'Uso 4: Dashboard Utente

Visualizzazione completa del profilo utente con statistiche.

```
GET /api/users/dashboard
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Content-Type: application/json
```

Listing 51: Richiesta Dashboard Utente


```
{
  "success": true,
  "data": {
    "user": {
      "user_id": 15,
      "name": "Mario",
      "surname": "Rossi",
      "email": "mario.rossi@email.com",
      "role": "user",
      "created_at": "2024-01-15T10:30:00Z"
    },
    "statistics": {
      "total_bookings": 5,
      "completed_bookings": 3,
      "confirmed_bookings": 1,
      "cancelled_bookings": 1,
      "total_spent": 890.00,
      "total_days_booked": 12,
      "locations_visited": 2,
      "space_types_used": 3
```

```
    },
    "bookings": {
      "upcoming": [
        {
          "booking_id": 25,
          "start_date": "2024-01-20",
          "end_date": "2024-01-22",
          "space_name": "Stanza 101",
          "location_name": "CoWork Milano Centro",
          "status": "confirmed"
        }
      ],
      "recent": [
        {
          "booking_id": 24,
          "start_date": "2024-01-10",
          "end_date": "2024-01-12",
          "space_name": "Sala Riunioni A",
          "location_name": "CoWork Roma",
          "status": "completed"
        }
      ],
      "total": 5
    },
    "preferences": {
      "top_locations": [
        {
          "location_name": "CoWork Milano Centro",
          "city": "Milano",
          "booking_count": 3
        }
      ],
      "top_space_types": [
        {
          "type_name": "Ufficio Privato",
          "booking_count": 3
        }
      ]
    }
  }
}
```


Listing 52: Risposta Dashboard Completa

Caso d'Uso 5: Workflow Manager

Gestione degli spazi da parte di un manager di location.

5.6.1 Passo 1: Dashboard Manager

```
GET /api/manager/dashboard
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Listing 53: Dashboard Manager

5.6.2 Passo 2: Visualizza Prenotazioni Location

```
GET /api/manager/bookings?status=pending&from_date=2024-01-15
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Listing 54: Prenotazioni da Gestire

5.6.3 Passo 3: Conferma Prenotazione

```
PUT /api/bookings/25/confirm
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Content-Type: application/json

{
  "manager_notes": "Prenotazione approvata - cliente verificato"
}
```

Listing 55: Conferma Prenotazione Manager

Caso d'Uso 6: Gestione Notifiche

Sistema di notifiche e comunicazioni utente.

5.7.1 Salvataggio Token FCM

```
POST /api/users/fcm-token
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Content-Type: application/json

{
  "fcm_token": "fGH1jK2L3m4N5o6P7q8R9s0T1u2V3w4X5y6Z..."
}
```

Listing 56: Registrazione per Notifiche Push

5.7.2 Recupero Notifiche

```
GET /api/notifications?status=unread&limit=10
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Listing 57: Lista Notifiche Utente

Caso d'Uso 7: Workflow Admin

Operazioni amministrative di sistema.

5.8.1 Dashboard Amministrativa

```
GET /api/admin/dashboard
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Listing 58: Statistiche Sistema

5.8.2 Promozione Manager

```
PUT /api/admin/users/15/promote-manager
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Content-Type: application/json

{
  "location_id": 1,
  "admin_notes": "Utente qualificato per gestione location Milano"
}
```

Listing 59: Promuovi Utente a Manager

Gestione Errori Comuni

5.9.1 Errore Autenticazione

```
{
  "success": false,
  "message": "Token scaduto, effettua nuovamente il login",
  "error": "TokenExpiredError",
  "code": "AUTH_TOKEN_EXPIRED"
}
```

Listing 60: Token Scaduto

5.9.2 Errore Disponibilità

```
{
  "success": false,
  "message": "Lo spazio non è disponibile nelle date selezionate",
  "error": "SPACE_NOT_AVAILABLE",
  "details": {
    "space_id": 1,
    "requested_period": {
      "start_date": "2024-01-20",
      "end_date": "2024-01-22"
    },
    "conflicting_bookings": [
      {
        "booking_id": 23,
        "start_date": "2024-01-21",
        "end_date": "2024-01-23"
      }
    ],
    "alternative_dates": [
      "2024-01-24",
      "2024-01-25",
      "2024-01-26"
    ]
  }
}
```

Listing 61: Spazio Non Disponibile

Best Practices per Integrazione

5.10.1 Autenticazione

- Memorizza il JWT token in modo sicuro (localStorage per web, KeyChain per mobile)
- Implementa refresh automatico prima della scadenza
- Gestisci logout automatico su errori 401
- Non inviare credenziali in query parameters

5.10.2 Chiamate API

- Usa sempre HTTPS in produzione
- Implementa retry logic per errori 5xx
- Rispetta i limiti di rate limiting
- Valida input lato client prima dell'invio

5.10.3 Gestione Errori

- Implementa handling specifico per ogni codice di errore
- Mostra messaggi user-friendly basati su `message`
- Logga dettagli tecnici da `error` per debugging
- Implementa fallback per errori di rete

Esempi SDK/Client

5.11.1 JavaScript/TypeScript Client

```
class CoWorkSpaceAPI {
  constructor(baseURL, token = null) {
    this.baseURL = baseURL;
    this.token = token;
  }

  async login(email, password) {
    const response = await fetch(`${this.baseURL}/users/login`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email, password })
    });

    const data = await response.json();
    if (data.success) {
      this.token = data.data.token;
    }
    return data;
  }
```

```
20
21    async getSpaces(filters = {}) {
22      const queryString = new URLSearchParams(filters).toString();
23      const response = await
            fetch(`${this.baseURL}/spaces?${queryString}`);
24      return response.json();
25    }
26
27    async createBooking(spaceId, startDate, endDate, notes = '') {
28      const response = await fetch(`${this.baseURL}/bookings`, {
29        method: 'POST',
30        headers: {
31          'Content-Type': 'application/json',
32          'Authorization': `Bearer ${this.token}`
33        },
34        body: JSON.stringify({
35          space_id: spaceId,
36          start_date: startDate,
37          end_date: endDate,
38          notes
39        })
40      });
41      return response.json();
42    }
43  }
44
45  // Utilizzo
46  const api = new CoWorkSpaceAPI('https://api.coworkspace.com/api');
47  await api.login('user@email.com', 'password');
48  const spaces = await api.getSpaces({ city: 'Milano', capacity_min: 4 });
```

Listing 62: Esempio Client JavaScript

Testing API

Esempi di test per validare le funzionalità API.

5.12.1 Test con curl

```
1  curl -X POST https://api.coworkspace.com/api/users/login \
2    -H "Content-Type: application/json" \
3    -d '{"email":"user@email.com","password":"password123"}'
```

Listing 63: Test Login con curl

5.12.2 Test con Postman

Importa la collezione Postman disponibile all'endpoint:

```
1  GET /api/postman-collection
```