Specifica Tecnica e Architetturale del Backend

1. Panoramica del Sistema

Il backend della Piattaforma Donazioni è costruito utilizzando un'architettura **RESTful** basata su **Node.js** e **Express**. Il sistema gestisce l'interazione tra due attori principali: **Donatori** (Utenti privati o commerciali) e **Associazioni** (Enti benefici), facilitando il ciclo di vita delle donazioni di beni materiali.

Il database utilizzato è **MongoDB**, gestito tramite l'ODM **Mongoose**, scelto per la flessibilità nella gestione di documenti complessi (es. dati geospaziali e array di oggetti).

2. Architettura del Software

Il progetto segue il pattern architetturale **MVC (Model-View-Controller)** adattato per API, con una netta separazione tra configurazione del server, definizione delle rotte e logica di business.

Componenti Principali

- **Entry Point (server.js):** È il punto di ingresso dell'applicazione. Si occupa di:
  - Caricare le variabili d'ambiente (dotenv).
  - Stabilire la connessione al database (connectDB).
  - Avviare il server HTTP sulla porta specificata.
  - Gestire errori critici di avvio (es. DB non raggiungibile) terminando il processo in sicurezza.
- **App Configuration (app.js):** Configura l'istanza di Express.
  - Abilita **CORS** per permettere richieste dal frontend.
  - Configura il parsing del JSON.
  - Attiva il middleware di logging (se in modalità DEBUG).
  - Monta il mainRouter sotto il prefisso /api.
- **Routing (mainRouter.js):** Agisce come hub centrale per lo smistamento del traffico. Aggrega i router specifici (auth, donation, report) e li assegna ai rispettivi path (es. /api/auth).
- **Controllers:** Contengono la logica di business pura. Ricevono l'input dalla richiesta, elaborano i dati interagendo con i Model e restituiscono la risposta.

Struttura delle Directory

```
/
  ├── app.js           # Configurazione Express App
  ├── server.js        # Entry point e connessione DB
  ├── api
  │   ├── controllers/     # Logica di business (Auth, Donation, Report)
  │   ├── models/          # Schemi Mongoose (User, Donation, Report)
  │   ├── routes/          # Definizioni degli endpoint
  │   │   ├── mainRouter.js     # Aggregatore principale
  │   │   ├── auth.routes.js
  │   │   ├── donation.routes.js
  │   │   └── report.routes.js
  └── middleware/      # Funzioni intermedie
      ├── auth.middleware.js
      └── logger.middleware.js
```

3. Modelli Dati (Classi Principali)

Di seguito è riportata la struttura dettagliata dei modelli Mongoose utilizzati nel database.

3.1 User (Utente)

Rappresenta tutti gli attori della piattaforma.

Campo	Tipo	Obbligatorio	Descrizione
_id	ObjectId	Sì (Auto)	Identificativo univoco generato da MongoDB.
email	String	Sì	Univoca, lowercase e trim.
password	String	Condizionale	Hash (Argon2). Obbligatoria se non si usa Google login.
googleId	String	No	ID univoco per utenti registrati via Google OAuth.
role	String	Sì	Enum: ['DONOR', 'ASSOCIATION',

			'ADMIN']. Default: DONOR.
name	String	Sì	Nome completo o ragione sociale.
phoneNumber	String	Sì	Contatto telefonico.
address	String	Sì	Indirizzo fisico principale.
solidarityPoints	Number	No	Punti accumulati (Gamification). Default: 0.
profile	Object	No	Dati specifici del ruolo.
profile.donorType	String	No	Enum: ['PRIVATE', 'COMMERCIAL']. Solo se ruolo è DONOR.
profile.commercialHours	String	No	Orari di apertura (Solo se COMMERCIAL).
redeemedRewards	Array	No	Array di ObjectId (Ref: Reward) per premi riscattati.

Funzionalità accessorie:

- **Password Hashing:** Middleware pre('save') per l'hashing automatico delle password modificate.
- **Verifica Password:** Metodo comparePassword(candidate) per il login locale.

3.2 Donation (Donazione)

Rappresenta un lotto di beni donato.

Campo	**Tipo**	**Obbligatorio**	**Descrizione**

donorId	ObjectId	Sì	Ref: User. L'utente che ha creato la donazione.
associationId	ObjectId	No	Ref: User. L'associazione che ha accettato la donazione.
status	String	Sì	Enum: ['AVAILABLE', 'ACCEPTED', 'COMPLETED', 'CANCELLED']. Default: AVAILABLE.
items	Array	Sì	Vettore di oggetti dettagliati.
items[].type	String	Sì	Categoria dell'oggetto (es. Cibo, Vestiario).
items[].name	String	Sì	Nome/Descrizione dell'oggetto.
items[].quantity	String	Sì	Quantità (es. "5 kg", "2 scatoloni").
pickupTime	Date	Sì	Data e ora proposte per il ritiro.
pickupLocation	Object	Sì	Dati sulla posizione di ritiro.
pickupLocation.address	String	Sì	Indirizzo testuale del ritiro.
pickupLocation.geo	Object	Sì	Oggetto GeoJSON per query spaziali.
pickupLocation.geo.type	String	Sì	Default: 'Point'.

pickupLocation.geo.coordinates	Array[Number]	Sì	Coordinate [Longitudine, Latitudine].
notes	String	No	Note aggiuntive per il ritiro (es. "Citofono rotto").
evaluation	Object	No	Feedback post-ritiro.
evaluation.rating	Number	No	Voto da 1 a 10.
evaluation.comment	String	No	Commento testuale.

Indici:

- 2dsphere su pickupLocation.geo: Ottimizza le query di prossimità geografica.

3.3 Report (Segnalazione)

Sistema di ticketing per segnalare problemi.

Campo	Tipo	Obbligatorio	Descrizione
reporterId	ObjectId	Sì	Ref: User. Chi ha aperto la segnalazione.
reportedUserId	ObjectId	No	Ref: User. L'utente segnalato (opzionale).
donationId	ObjectId	No	Ref: Donation. La donazione problematica (opzionale).
type	String	Sì	Enum: ['MALFUNCTION', 'USER_BEHAVIOR'].

description	String	Sì	Descrizione dettagliata del problema.
status	String	Sì	Enum: ['OPEN', 'IN_PROGRESS', 'CLOSED']. Default: OPEN.

4. Sicurezza e Middleware

Autenticazione (JWT)

Il sistema utilizza **JSON Web Tokens (JWT)** per l'autenticazione stateless.

1. **Middleware isAuth:** Intercetta ogni richiesta protetta.
2. Verifica la firma del token Bearer.
3. Estrae l'ID utente e interroga il database per ottenere l'istanza aggiornata dell'utente.
4. Allega l'oggetto user alla req per i passaggi successivi.

Controllo Accessi (RBAC)

Sono implementati middleware specifici che agiscono *dopo* isAuth:

- **isDonor**: Verifica che req.user.role === 'DONOR'.
- **isAssociation**: Verifica che req.user.role === 'ASSOCIATION'.
- **isAdmin**: Verifica che req.user.role === 'ADMIN'.

Monitoring e Debugging

- **Logger Middleware:** Un middleware personalizzato (logger.middleware.js) intercetta richieste e risposte se la variabile d'ambiente DEBUG è attiva.
  - **Request:** Logga metodo, URL, headers e body.
  - **Response:** Intercetta res.send per loggare status code, durata dell'esecuzione (in ms) e body della risposta.

5. Cicli di Funzionamento (Logica di Business)

5.1 Ciclo di Vita della Donazione

Il flusso principale dell'applicazione segue questi stati rigorosi:

1. **Creazione (Stato: AVAILABLE)**
   - **Attore:** Donatore.
   - **Azione:** Invia lista oggetti (items), orario ritiro e posizione.
   - **Sistema:** Valida data futura e integrità dati, salva con stato AVAILABLE.

2. **Modifica (Parziale)**
    - **Attore:** Donatore.
    - **Vincolo:** Possibile *solo* se lo stato è AVAILABLE.
    - **Logica:** Supporta aggiornamenti parziali. Se viene inviata una nuova data, viene validata nuovamente.
3. **Accettazione (Stato: ACCEPTED)**
    - **Attore:** Associazione.
    - **Azione:** Prenota una donazione disponibile.
    - **Sistema:** Esegue un lock atomico (tramite findOneAndUpdate) per prevenire race conditions. Assegna associationId.
4. **Completamento (Stato: COMPLETED)**
    - **Attore:** Associazione.
    - **Azione:** Conferma ritiro e invia feedback (rating).
    - **Sistema (Transazione):** Utilizza una **transazione MongoDB** per garantire l'integrità:
        - Passo A: Aggiorna stato donazione a COMPLETED e salva la review.
        - Passo B: Incrementa i solidarityPoints del Donatore.
        - *Rollback:* Se uno dei due passaggi fallisce, l'intera operazione viene annullata.

5.2 Ciclo di Autenticazione Google

Per garantire sicurezza e coerenza dati:

1. **Frontend:** Ottiene il credential token da Google.
2. **Backend (handleGoogleToken):** Verifica il token direttamente con le API Google.
3. **Bivio Logico:**
    - *Utente Esiste:* Genera JWT di sessione e logga l'utente.
    - *Utente Nuovo:* Genera un JWT temporaneo (registrationToken) con durata breve (15 min).
4. **Completamento (register-google):** Il client invia i dati mancanti (ruolo, telefono) + il registrationToken. Il backend crea l'utente finale.

6. Stack Tecnologico

Componente	Tecnologia	Note
Runtime	Node.js	Event-driven, non-blocking I/O
Framework	Express.js	Gestione routing e middleware
Database	MongoDB	Database NoSQL orientato ai documenti

ODM	Mongoose	Modellazione dati e validazione schema
Auth	JWT + Passport	Gestione token e strategie OAuth
Security	Argon2	Hashing sicuro delle password
Networking	CORS	Gestione Cross-Origin Resource Sharing
Utils	Dotenv	Gestione variabili d'ambiente