Relazione Fast Food

Struttura del progetto

Monorepo full-stack con due sottoprogetti principali:

- `backend/`: server **Express** in JavaScript (ESM), connessione a **MongoDB**, API REST, Swagger/OpenAPI, seeding.
- `frontend/`: app **Vite + React** (JavaScript), routing client-side, UI componentizzata, i18n.

L'adozione di **npm workspaces** centralizza dipendenze e build (`package-lock.json` unico).

Presentazione del sito web (struttura e UI)

L'interfaccia è organizzata in **pagine/viste** (route) con layout coerente:

- **Landing** (`/`): Hero, sezione "API health", CTA verso l'app.
- **Autenticazione** (`/auth`): form di login/registrazione con ruoli.
- **Dashboard Client** (`/dashboard/client`): card operative (ristoranti, ordini, carrello) + highlight personalizzati.
- **Dashboard Ristoratore** (`/dashboard/restaurant`): card operative (menu, ordini, statistiche).
- **Liste e dettagli**: ristoranti, menu, carrello, ordini, gestione menu, creazione piatti.
- **Impostazioni** (`/settings`): modifica profilo, ruoli, preferenze, dati ristorante.

Componenti UI principali: - `Navbar`, `Breadcrumbs`, `DashboardCard`, `MenuTable`, `OrderCard`, `Modal`, `Spinner`, `Toast`. - Layout a **card** e **grid** per rendere omogenee le viste. - **Tabelle** con ricerca, azioni contestuali, stati vuoti, loading.

Scelte di presentazione: - Design coerente basato su card e sezioni. - Navigazione guidata con breadcrumb. - Pulsanti con varianti semantiche (primary, secondary, sky). - Gestione stati (loading/error/empty) visibile.

Architettura funzionale

1. Autenticazione e profili

- Login con **JWT** (`/api/auth/login`) e salvataggio token lato client.
- Rotte protette lato frontend (`RequireAuth`).
- Profilo con **ruoli multipli** (cliente/ristoratore), modificabili in `Settings`.

- Disattivazione profilo tramite `active=false`.

2. Dati e collezioni (MongoDB)

- `users`: utenti con ruoli + `clientData/restaurantData`.
- `meals`: catalogo piatti base (seed da `meal.json` + custom).
- `menus`: piatti associati al ristorante con prezzo/categoria/ingredienti.
- `orders`: ordini con stati, costi, ETA, consegna.

3. Backend REST

Principali endpoint: - `GET /api/restaurants`, `GET /api/restaurants/:id` - `GET /api/menus/:restaurantId`, `PATCH/DELETE /api/menus/...` - `GET /api/meals`, `POST /api/meals` - `POST /api/orders`, `GET /api/orders`, `PATCH /api/orders/:id/status` - `GET /api/orders/preparation-counts` - `POST/GET/PUT/DELETE /api/users/:id`

Le API sono documentate con **Swagger** (`/api/docs`, `/api/docs.json`).

Operazioni richieste e come sono state implementate

Registrazione e login

- **Frontend**: `AuthPage` con `AuthForm`.
- **Backend**: `POST /api/users` per registrazione, `POST /api/auth/login` per login.
- Scelta ruolo durante signup; profilo aggiornabile da `Settings`.

Visualizzazione dati (piatti, clienti, ristoratori, acquisti)

- **Ristoranti**: lista clienti con dati base (`ClientRestaurants`).
- **Piatti**: menu per ristorante (`ClientRestaurantMenu`).
- **Acquisti/ordini**: pagina ordini cliente (`ClientOrders`).
- **Ordini ristorante**: pagina ordini ristoratore (`RestaurantOrders`).

Ricerca ristoranti (nome/luogo)

- **ClientRestaurants** filtra su `name` e `address` in tempo reale.

Ricerca piatti (tipologia/nome/prezzo)

- **ClientRestaurantMenu** filtra su:
  - nome, categoria,
  - prezzo (testuale),
  - ingredienti (estratti).

Ricerca piatti per ingredienti

- Implementata nella ricerca del menu: gli ingredienti sono inclusi nella stringa di ricerca (vedi `getMealIngredients`).

Ricerca piatti per allergie

- **Non c'è una gestione esplicita delle allergie**. Il progetto include una ricerca per ingredienti che può essere usata come filtro manuale, ma non una lista "allergeni" dedicata.

Ricerca ristorante per piatto

- **Non esiste una pagina dedicata di ricerca globale piatto → ristorante**.
- La ricerca è per ristorante e poi per piatto nel menu selezionato. È una scelta semplificativa adottata nella UI.

Login + ordine piatti

- **Flusso**: selezione piatti in menu → carrello → checkout.
- Carrello salvato in **localStorage** (`utils/cart.js`) per persistenza.
- Checkout crea ordini separati per ristorante con dettagli completo.

Gestione consegne

- Selezione **pickup / delivery** nel checkout.
- **Delivery** richiede indirizzo (blocca checkout se mancante).
- Stima **costo e tempo** con OpenStreetMap (Nominatim) + Haversine.
- **Stati ordine**:
  - `ordered → preparation → readyForPickup → delivered`
- L'API backend applica regole di transizione in base al ruolo.

Statistiche ristorante

- `RestaurantStats` calcola:
  - numero ordini,
  - ricavi totali,
  - piatti più venduti,
  - miglior cliente.

Scelte implementative principali (e motivazioni)

JWT per autenticazione

- Token firmato lato backend, più sicuro e standard per API REST.
- Permette protezione ruoli e accesso alle API.

Persistenza locale carrello e ordini

- **localStorage** usato per carrello e storicizzazione ordini:
  - garantisce esperienza fluida anche senza refresh backend immediato.
  - riduce la complessità di gestione stato globale.

Delivery estimation lato backend

- In `ordersController` la stima è **server-side** usando:
  - geocoding OpenStreetMap,
  - Haversine per distanza,
  - costo e tempo dinamici.
- Scelta: centralizzare la logica per coerenza tra client e server.

Modello menu con piatti catalogo + custom

- Menù del ristorante deriva da:
  - piatti catalogo (meal.json) o
  - piatti creati dal ristoratore.
- Prezzo, categoria, ingredienti e foto sono personalizzabili.
- Foto gestite via **Cloudinary** per storage esterno.

Componentizzazione UI

- Riutilizzo componenti (tables, cards, modals).
- Migliora manutenzione e coerenza del layout.

i18n

- Traduzioni IT/EN integrate (`i18n.js`).
- Scelta utile per demo e adattabilità.

Allineamento con la traccia

Requisiti coperti: - Gestione profilo utente (registrazione, modifica, disattivazione). - Gestione ristorante (menu, piatti custom, CRUD menu). - Gestione ordini (checkout, stati, delivery/pickup). - Gestione consegne (stima costo/tempo, workflow). - Visualizzazione statistiche ristorante. - Visualizzazione acquisti cliente.

Parzialmente coperti: - Ricerca ristorante per piatto (non globale). - Ricerca piatti per allergie (non dedicata, solo tramite ingredienti).

Conclusione

Il progetto implementa un **sistema completo di ordinazione FastFood** con ruoli distinti, backend REST e frontend ricco di funzionalità. Le scelte architetturali (JWT, REST, DAO, localStorage, componentizzazione, Swagger) hanno privilegiato **manutenibilità, chiarezza del flusso e coerenza tra frontend e backend**. Il risultato è un'applicazione modulare, estendibile e allineata alla maggior parte delle richieste della traccia, con alcune semplificazioni esplicitate.