FastFood

Progetto di Programmazione Web e Mobile
Anno Accademico 2025/2026

Marco Galluccio
Matricola: 24939A

4 febbraio 2026

Indice

1 Introduzione

Questo documento costituisce la relazione tecnica del progetto **FastFood**, sviluppato nell'ambito del corso di *Programmazione Web e Mobile* per l'Anno Accademico 2025/2026.

L'obiettivo del progetto è la realizzazione di una **web application full-stack** per la gestione di ordini online per ristoranti appartenenti a una catena di Fast Food. L'applicazione consente agli utenti di autenticarsi e interagire con il sistema in base al ruolo scelto in fase di registrazione (*cliente* o *ristoratore*), permettendo la consultazione dei ristoranti, la gestione dei menu, la creazione, il tracciamento dello stato degli ordini e la gestione di essi.

Lo sviluppo dell'applicazione ha seguito le fasi classiche di un progetto software:

- analisi dei requisiti;
- progettazione dell'architettura;
- progettazione dell'interfaccia utente;
- implementazione del backend e del frontend;
- verifica del corretto funzionamento delle funzionalità richieste.

1.1 Autore

Il progetto è stato sviluppato individualmente da:

- **Marco Galluccio** (Matricola: 24939A)

1.2 Link e informazioni utili

- Repository GitHub del progetto: `https://github.com/marcogalluccio1/pwm_project`
- Email: marco.galluccio1@studenti.unimi.com

2 Descrizione generale dell'applicazione

L'applicazione è stata sviluppata come una **Single Page Application (SPA)** con architettura **REST**, composta da un frontend realizzato in React e un backend sviluppato in Node.js con database MongoDB.

L'applicazione gestisce quattro macro-scenari principali:

- gestione del profilo utente;
- gestione del ristorante;
- gestione degli ordini;

- gestione delle consegne (non implementata, ma supporto alla sua futura implementazione).

Ogni funzionalità è stata implementata nel rispetto della separazione tra frontend e backend, con comunicazione tramite API REST protette dove richiesto da un layer dia utenticazione.

3 Struttura dell'Applicazione

L'applicazione è stata progettata seguendo un'architettura **client-server** basata sul paradigma **REST**. La struttura è suddivisa in due componenti principali:

- Frontend
- Backend

Questa separazione consente una migliore manutenibilità del codice, una chiara suddivisione delle responsabilità e una maggiore scalabilità dell'intero sistema.

4 Frontend

Il frontend dell'applicazione FastFood è stato sviluppato utilizzando **React** e **Vite**. Questa scelta consente una navigazione fluida senza ricaricamenti completi della pagina e una migliore esperienza utente.

Il frontend si occupa esclusivamente della presentazione dei dati e dell'interazione con l'utente, delegando al backend tutta la logica applicativa e la gestione della persistenza.

4.1 Tecnologie utilizzate

- **React**: libreria JavaScript per la costruzione di interfacce utente basate su componenti riutilizzabili.
- **Vite**: tool di build e sviluppo che garantisce tempi di avvio e hot reload rapidi.
- **React Router**: gestione della navigazione client-side tra le diverse pagine.
- **CSS**: definizione dello stile grafico delle pagine e dei componenti.

Nota sugli stili grafici Gli stili grafici sono stati realizzati utilizzando CSS, senza l'uso di framework come Bootstrap, al fine di mantenere maggiore libertà progettuale e sperimentare direttamente con la definizione degli stili. In alcune fasi di rifinitura è stato utilizzato il supporto dell'intelligenza artificiale come strumento di ottimizzazione e assistenza allo sviluppo.

4.2 Struttura del Frontend

Il codice frontend è organizzato in modo modulare:

- `/pages`: pagine principali dell'applicazione (home, ristoranti, ordini, checkout, profilo, ecc.);
- `/components`: componenti riutilizzabili (barra di navigazione);
- `/api`: funzioni JS per l'interazione con le API REST del backend;
- `/auth`: gestione dello stato di autenticazione e del ruolo utente;
- `/styles`: fogli di stile CSS riutilizzati nelle pagine.
- `/assets`: Immagini utilizzate nelle varie pagine.

Nota sulle immagini Le immagini utilizzate all'interno dell'applicazione sono state generate tramite l'ausilio dell'intelligenza artificiale, con l'obiettivo di ottenere contenuti visivi gradevoli coerenti con il contesto dell'applicazione e maggiormente personalizzati.

4.3 Home Page e Navigazione

La home page rappresenta il punto di accesso principale all'applicazione ed è stata progettata per adattarsi dinamicamente allo stato di autenticazione e al ruolo dell'utente.

4.3.1 Top Bar e Area Personale

Nella home e nelle principali pagine dell'applicazione è presente una **top bar** che consente la navigazione tra le principali sezioni.

Nella parte superiore destra della top bar:

- se l'utente non è autenticato, sono presenti i pulsanti di login e registrazione;
- se l'utente è autenticato, è presente un menù a tendina con collegamenti all'area personale.

L'area personale dell'utente **cliente** permette di:

- visualizzare e modificare i propri dati;
- cancellare il proprio account, nel rispetto dei vincoli applicativi.
- gestire preferenze e metodo di pagamento;
- gestire i propri ordini, attivi e passati

L'area personale dell'utente **ristoratore** permette di:

- visualizzare e modificare i propri dati;

- cancellare il proprio account, nel rispetto dei vincoli applicativi.
- creare e gestire le informazioni del proprio ristorante;
- gestire la coda di tutti gli ordini del proprio ristorante

4.3.2 Home per Utente non Autenticato

Quando l'utente non è autenticato, la home permette di:

- fare il login o registratsi;
- accedere alle pagine principali di piatti e ristoranti;
- visualizzare un **piatto casuale**, con lo scopo di incentivare la registrazione.

In questa modalità non è possibile effettuare ordini nè gestire locali.

4.3.3 Home per Cliente

Un utente autenticato come cliente visualizza:

- accesso alla propria area personale.
- possibilità di iniziare il processo di ordinazione;
- accesso ai piatti suggeriti in base alle proprie preferenze impsotate;

La home guida il cliente verso la selezione di un ristorante, che rappresenta il primo passo del flusso di ordinazione.

4.3.4 Home per Ristoratore

Un utente autenticato come ristoratore visualizza:

- accesso all'area personale.
- accesso alla gestione degli ordini ricevuti;
- accesso alla gestione del proprio ristorante e del menu;

In questo caso la home è orientata alla gestione del ristorante.

5 Backend

Il backend di FastFood è stato sviluppato in **Node.js** utilizzando **Express** come framework web. La sua responsabilità principale è fornire al frontend un insieme di **API REST** per la gestione dei dati e dei processi applicativi: autenticazione, gestione ristoranti, piatti, menù e ordini, includendo la gestione degli stati dell'ordine.

L'applicazione backend si occupa inoltre di:

- validazione dei dati in ingresso e corretta gestione dei codici HTTP in risposta;

- autenticazione e autorizzazione basata su token (JWT);
- persistenza su database MongoDB tramite ODM (Mongoose);
- documentazione delle API tramite Swagger;
- inizializzazione (setup) dei dati statici dei piatti a partire dal file JSON fornito.

5.1 Tecnologie utilizzate e scopo

- **Node.js**: runtime JavaScript per gestire richieste concorrenti e I/O asincrono.
- **MongoDB**: database NoSQL document oriented.
- **Mongoose**: Object Data Modeling (ODM) per MongoDB, usato per definire *models* con schema, validazioni e metodi di utilità.
- **JWT (jsonwebtoken)**: implementa l'autenticazione stateless tramite tokens; il token viene generato al login/registrazione e passato dal frontend nell'header `Authorization: Bearer <token>`.
- **Express**: framework per la gestione delle rotte HTTP e della logica middleware.
- **bcryptjs**: gestione password in modo sicuro tramite hashing (evitando la memorizzazione in chiaro).
- **dotenv**: caricamento delle variabili d'ambiente (es. URI MongoDB e segreto JWT)
- **compression**: middleware per comprimere le risposte HTTP (migliorando prestazioni e latenza, specialmente per payload JSON)
- **swagger-jsdoc** + **swagger-ui-express**: generazione swagger a partire dai commenti nelle route e pubblicazione dell'interfaccia grafica di documentazione su `/api/docs`.

5.2 Entry point e configurazione server

Il punto di ingresso è `src/app.js`. In questo file vengono eseguite le operazioni fondamentali:

- lettura configurazione tramite `dotenv`;
- connessione a MongoDB tramite il modulo `src/database/db.js`;
- setup Swagger e pubblicazione della documentazione su `/api/docs`;
- registrazione dei router REST (auth, restaurants, meals, orders);
- gestione della distribuzione della SPA React: il backend serve la cartella build del frontend e reindirizza tutte le route non-API a `index.html`.

Questa scelta consente di deployare l'applicazione come **prodotto unico** (backend + frontend), mantenendo comunque l'architettura REST per la comunicazione interna.

5.3 Organizzazione del codice

Il backend è strutturato per responsabilità, seguendo una separazione chiara tra routing, logica applicativa, modelli e middleware.

- `src/routes/`: definisce gli endpoint REST e mappa ciascuna route al controller appropriato.
  - `authRoutes.js`
  - `restaurantRoutes.js`
  - `mealRoutes.js`
  - `orderRoutes.js`
- `src/controllers/`: contiene la logica principale degli endpoint (validazioni, accesso ai modelli, risposte HTTP).
  - `authController.js`: registrazione, login, gestione profilo (`/me`), update e cancellazione account con vincoli (es. blocco cancellazione se esistono ordini attivi).
  - `restaurantController.js`: CRUD e operazioni legate ai ristoranti.
  - `mealController.js` e `menuController.js`: gestione e ricerca dei piatti/menu.
  - `orderController.js`: creazione ordine, calcolo tempo di attesa stimate, recupero ordini utente/ristorante e aggiornamento stato ordine.
  - `statsController.js`: endpoint dedicato alla visualizzazione di statistiche del ristorante.
- `src/models/`: modelli Mongoose che rappresentano le entità coinvolte.
  - `User.js`: utenti customer/seller, credenziali (hash), dati profilo, pagamento e preferenze.
  - `Restaurant.js`: informazioni del ristorante e collegamento al seller.
  - `Meal.js`: piatti con attributi descrittivi.
  - `Order.js`: ordine con associazioni piatti + quantità, riferimento a customer/restaurant e stato e informazioni sulla modalità di consegna.
- `src/middleware/`: middleware per autenticazione/autorizzazione.
  - `authMiddleware.js`: validazione token JWT (`requireAuth`) e controllo ruolo (`requireRole`).
- `src/database/`: connessione centralizzata al database.

  - `db.js`: stabilisce la connessione a MongoDB tramite `MONGO_URI`.

- `src/services/`: logica di servizio non direttamente legata alle route.
  - `mealSetup.js`: caricamento dei piatti iniziali dal file `data/meals.json` all'avvio dell'applicazione, come richiesto dalla traccia.

- `src/swagger.js`: definizione delle opzioni OpenAPI utilizzate per generare la documentazione tramite `swagger-jsdoc`.

5.4 Documentazione API

Tutte le API sono documentate tramite **Swagger**. Le annotazioni OpenAPI sono integrate direttamente nei file di routing (commenti `@swagger`), e la documentazione è consultabile tramite interfaccia web su:

`/api/docs`

6 Database

Per la persistenza dei dati è stato utilizzato **MongoDB**, un database NoSQL orientato ai documenti.

L'accesso al database è gestito tramite **Mongoose**, che fornisce uno strato di Object Data Modeling (ODM) permettendo la definizione di schemi, validazioni e relazioni tra documenti.

6.1 Modello Utente

Il modello `User` rappresenta sia clienti sia ristoratori. La distinzione tra i due ruoli avviene tramite il campo `role`.

Gli utenti memorizzano:

- dati di autenticazione;
- ruolo (cliente o ristoratore);
- informazioni di profilo;
- metodo di pagamento (necessario per l'ordinazione).

```
const UserSchema = new mongoose.Schema(
  {
    role: { type: String, enum: ["customer", "seller"],
       required: true },
    email: { type: String, required: true, unique: true,
       lowercase: true, trim: true },
    passwordHash: { type: String, required: true, select:
       false },
    firstName: { type: String, required: true, trim: true },
```

```
    lastName: { type: String, required: true, trim: true },
    vatNumber: {
      type: String,
      required: function () { return this.role === "seller";
         },
      trim: true,
    },
    payment: {
      method: {
        type: String,
        enum: ["card", "prepaid", "cash"],
        default: null,
      },
      cardBrand: { type: String, trim: true },
      cardLast4: { type: String, trim: true },
      holderName: { type: String, trim: true },
    },

    preferences: {
      favoriteMealTypes: [{ type: String, trim: true }],
      marketingOptIn: { type: Boolean, default: false },
    },
  },
  { timestamps: true }
);
```

La scelta di rendere il metodo di pagamento obbligatorio solo per i clienti consente di validare correttamente il flusso di checkout.

6.2 Modello Ristorante

Il modello `Restaurant` rappresenta un ristorante gestito da un ristoratore. Ogni ristorante è associato a un singolo utente di tipo seller in una relazione 1:1.

```
const RestaurantSchema = new mongoose.Schema(
  {
    sellerId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "User",
        required: true,
        unique: true,
        index: true
    },
    name: { type: String, required: true, trim: true },
    phone: { type: String, required: true, trim: true },
    address:  {type: String, required: true, trim: true },
    city: { type: String, required: true, index: true, trim:
       true },
    menuItems: [
      {
```

```
        mealId: { type: mongoose.Schema.Types.ObjectId, ref: "
            Meal", required: true },
        price: { type: Number, required: true, min: 0 },
      },
    ],
  },
  { timestamps: true }
);
```

Questa relazione consente di vincolare la gestione del ristorante al solo ristoratore proprietario.

6.3 Modello Meal

Il modello `Meal` rappresenta un piatto disponibile nel sistema. I piatti possono essere:

- piatti standard caricati dal file JSON iniziale;
- piatti personalizzati creati dai ristoratori.

Il prezzo non è memorizzato nel modello `Meal`, poiché dipende dal ristorante che lo inserisce nel proprio menu.

```
const MealSchema = new mongoose.Schema(
  {
    sourceId: { type: String, index: true },
    name: { type: String, required: true, trim: true, index:
        true },
    category: { type: String, required: true, trim: true,
        index: true },
    thumbnailUrl: { type: String, required: true, trim: true
        },
    ingredients: [{ type: String, trim: true, index: true }],
    isGlobal: { type: Boolean, default: true },
    createdBySellerId: { type: mongoose.Schema.Types.ObjectId,
        ref: "User" },
  },
  { timestamps: true }
);
```

La proprietà `isGlobal` tiene conto del fatto che un meal proviene dal file *meals.json* o se è stato inserito manualmente da un ristoratore (il cui id viene inserito in `createdBySellerId`).

Queste proprietà seguono una scelta implementativa per la quale un ristoratore può aggiungere al menù solo i piatti globali o quelli da lui creato, non è infatti permesso di aggiungere a un menù piatti personalizzati di altri ristoranti.

6.4 Modello Ordine

Il modello `Order` rappresenta un ordine effettuato da un cliente presso un singolo ristorante.

Ogni ordine contiene:

- riferimento al cliente;
- riferimento al ristorante;
- lista dei piatti ordinati con quantità e prezzo;
- stato dell'ordine;
- informazioni sul metodo di consegna (e su distanza ed indirizzo)
- informazioni sul pagamento.
- stima dell'orario di consegna.

```
const OrderSchema = new mongoose.Schema(
  {
    customerId: { type: mongoose.Schema.Types.ObjectId, ref: "
       User", required: true, index: true },
    restaurantId: { type: mongoose.Schema.Types.ObjectId, ref:
        "Restaurant", required: true, index: true },
    items: { type: [OrderItemSchema], required: true },
    fulfillment: { type: String, enum: ["pickup", "delivery"],
        required: true },
    deliveryAddress: { type: String, trim: true },
    distanceKm: { type: Number, min: 0 },
    status: {
      type: String,
      enum: ["ordered", "preparing", "delivering", "delivered"
         ],
      default: "ordered",
      index: true,
    },
    subtotal: { type: Number, required: true, min: 0 },
    deliveryFee: { type: Number, required: true, min: 0 },
    total: { type: Number, required: true, min: 0 },
    paymentMethod: { type: String, enum: ["card", "prepaid", "
       cash"], required: true},
    estimatedReadyAt: { type: Date },
  },
  { timestamps: true }
);
```

Come descritto in precedenza, la modalità di consegna a domicilio non è stata implementata, ma sono comunque state inserite le proprietà necessarie per una eventuale futura implementazione.

6.4.1 Stima del tempo di attesa (pickup)

Per gli ordini da asporto (*pickup*), lato backend viene calcolata una stima del tempo di attesa al momento della creazione dell'ordine. La stima è salvata nel campo `estimatedReadyAt` dell'ordine e rappresenta una previsione di quando l'ordine sarà pronto per il ritiro.

La stima viene calcolata considerando:

- la **coda attuale** del ristorante, ossia tutti gli ordini con stato `ordered` o `preparing`;
- la **dimensione del nuovo ordine** (numero totale di item/quantità);
- due parametri configurabili via variabili d'ambiente:
  - `PREP_BASE_MINUTES_PER_ORDER`: tempo base fisso per gestione ordine;
  - `PREP_MINUTES_PER_ITEM`: tempo stimato per singola unità ordinata.

6.4.2 Calcolo della coda

Per stimare il carico di lavoro corrente, il backend esegue un'aggregazione MongoDB sugli ordini del ristorante, filtrando per gli stati ancora attivi (`ordered`, `preparing`) e sommando le quantità di tutti gli item in coda. In questo modo si ottiene un valore `queueQty` che approssima il numero totale di unità ancora da preparare.

Questa scelta implementativa permette di fornire al cliente una stima dinamica e più realistica, poiché il tempo dipende dal carico di lavoro del ristorante e non da un valore fisso.

7 Autenticazione e ruoli

L'autenticazione è implementata tramite **JWT**:

- in fase di login/registrazione viene generato un token firmato con `JWT_SECRET`;
- il token include le informazioni minime necessarie (id utente e ruolo);
- le route protette richiedono il token nell'header `Authorization`.

La password non viene mai salvata in chiaro: viene gestita tramite **bcryptjs** e memorizzata come hash, riducendo i rischi in caso di accesso non autorizzato al database.

L'applicazione come detto in precedenza, prevede due tipologie di utenti:

- **Cliente**
- **Ristoratore**

Questa distinzione è centrale nella progettazione dell'applicazione e influenza sia le funzionalità disponibili lato frontend, sia i vincoli applicativi lato backend.

7.1 Vincolo sull'Ordinazione

Una scelta implementativa fondamentale è che **solo gli utenti autenticati come cliente possono effettuare ordini**. Gli utenti ristoratori non hanno accesso alle funzionalità di ordinazione.

Questo vincolo è stato introdotto per:

- mantenere una chiara separazione dei ruoli;
- evitare conflitti tra produttori e consumatori;
- rispecchiare il comportamento reale di una piattaforma di food delivery.

Il controllo viene applicato:

- lato frontend, mostrando o nascondendo i pulsanti di ordinazione in base al ruolo;
- lato backend, tramite middleware di autorizzazione che verificano il ruolo dell'utente prima di consentire la creazione di un ordine.

8 Esplorazione di Piatti e Ristoranti

8.1 Lista Completa dei Piatti

Dalla home page è possibile accedere a una pagina che mostra **tutti i piatti disponibili** nel sistema, includendo:

- piatti standard caricati inizialmente dal file *meals.json*;
- piatti personalizzati aggiunti dai ristoratori.

Questa pagina ha finalità esplorativa e non consente l'ordinazione diretta.
I piatti possono essere filtrati per:

- nome;
- categoria.

Aprendo la pagina di un piatto è possibile visualizzare tutte le sue informazioni quali:

- nome;
- thumbnail;
- categoria.
- ingredienti;

Dalla pagina di un piatto è inoltre presente un pulsante che permette di cercare i ristoranti che hanno quel dato piatto nel proprio menù.

8.2 Lista dei Ristoranti

È poi disponibile una pagina dedicata alla visualizzazione di tutti i ristoranti registrati. I ristoranti possono essere filtrati per:

- nome;
- città.

Da questa pagina si può accedere al dettaglio di uno specifico ristorante e al relativo menù; se loggati come clienti da qui si può iniziare il flusso di ordinazione.

9 Menu del Ristorante

Dalla pagina di dettaglio di un ristorante, il cliente loggato può visualizzare il menù completo e procedere all'ordinazione.

Il menu è navigabile ed è filtrabile per:

- nome del piatto;
- categoria;
- prezzo minimo e massimo.

9.1 Filtro per Prezzo

Il filtro per prezzo è disponibile **solo all'interno del menu di un singolo ristorante**. Questa scelta è motivata dal fatto che il prezzo non è una proprietà intrinseca del piatto, ma viene definito quando un piatto viene aggiunto al menu da un ristoratore.

Di conseguenza:

- lo stesso piatto può avere prezzi diversi in ristoranti diversi;
- il filtro per prezzo non è applicabile all vista globale dei piatti.

10 Gestione degli Ordini

La gestione degli ordini rappresenta la funzionalità centrale dell'applicazione.

Un cliente autenticato può:

- selezionare un ristorante;
- aggiungere uno o più piatti del menù al carrello;
- confermare l'ordine tramite la pagina di checkout.

Ogni ordine è associato a un singolo ristorante e segue un flusso di stato ben definito.

10.1 Stati dell'Ordine

Lo stato di un ordine può assumere i seguenti valori:

- **ordinato**
- **in preparazione**
- **in consegna**
- **consegnato**

Il ristoratore può aggiornare lo stato dell'ordine, mentre il cliente può monitorarne l'avanzamento tramite un'interfaccia grafica dedicata.

10.1.1 Modalità di ordinazione

L'ordinazione prevede due modalità disponibili:

- **Da asporto**
- **A domicilio**

Non avendo lavorato in gruppo, la parte in merito alle consegne a domicilio non è stata implementata, ma è già stato fornito il supporto a una sua afutura aggiunta tramite interfacce grafiche ed API per permettere al cliente di segnalare che l'ordinazione è stata consegnata.

Notare come lo stato dell'ordine per la modalità da asporto è gestito totalmente dal ristoratore e segue il flusso: **ordinato** $\rightarrow$ **in preparazione** $\rightarrow$ **consegnato**

10.2 Coda delle Ordinazioni

La coda delle ordinazioni è gestita lato backend nel controller degli ordini.

Gli ordini associati a un ristorante vengono recuperati e ordinati in base:

- allo stato dell'ordine;
- al tempo di consegna previsto.

Questo approccio consente al ristoratore di:

- visualizzare chiaramente gli ordini in attesa;
- gestire correttamente la preparazione;
- aggiornare lo stato dell'ordine seguendo il flusso previsto.

La coda riflette quindi l'ordine reale di lavorazione delle ordinazioni.

11 Checkout e Pagamento

L'ordinazione viene finalizzata tramite una classica pagina di **checkout**, che mostra un riepilogo completo dell'ordine:

- piatti selezionati;
- quantità;
- totale dell'ordine.

Per completare l'ordine è necessario che l'utente abbia configurato un **metodo di pagamento** nel proprio profilo.

In assenza di un metodo di pagamento valido, l'utente viene reindirizzato alla sua pagina personale per completare queste informazioni prima di poter procedere.

12 API REST e Documentazione

La comunicazione tra frontend e backend avviene tramite **API REST**, utilizzando i principali metodi HTTP (`GET`, `POST`, `PUT`, `DELETE`).

Tutte le API sono documentate tramite **Swagger**, che consente di:

- visualizzare gli endpoint disponibili;
- testare le API direttamente dal browser;
- comprendere struttura e payload delle richieste.
- comprendere come sono strutturare le risposte JSON.

La documentazione Swagger è accessibile all'indirizzo:

`/api/docs`

13 Guida all'avvio dell'applicazione e prove di funzionamento

In questa sezione viene descritto come avviare l'applicazione FastFood in modalità **single server** e vengono mostrate alcune prove di funzionamento che dimostrano la corretta esecuzione delle funzionalità principali richieste.

13.0.1 Ottenimento del codice sorgente

Il codice del progetto può essere ottenuto in due modi alternativi:

- clonando la repository GitHub del progetto;
- estraendo l'archivio `.zip` fornito nella consegna.

Nel primo caso, il progetto può essere clonato tramite il comando:

```
git clone https://github.com/marcogalluccio1/pwm_project.git
```

In alternativa, è possibile scaricare l'archivio `.zip` ed estrarlo in una directory locale. In entrambi i casi la struttura del progetto risulta identica.

13.0.2 Prerequisiti

Per l'esecuzione dell'applicazione è necessario disporre di:

- Node.js installato;
- un'istanza MongoDB accessibile (anche in versione cloud, ad esempio MongoDB Atlas);
- un file `.env` correttamente configurato nella directory del backend, seguendo il file di esempio `.env.example`.

13.0.3 Installazione delle dipendenze

Il progetto è suddiviso in due directory principali, `backend` e `frontend`, ciascuna con un proprio file `package.json`. È pertanto necessario installare separatamente le dipendenze di backend e frontend.

Dalla root eseguire:

```
cd backend
npm install
```

Successivamente, tornare alla root ed eseguire:

```
cd frontend
npm install
```

13.0.4 Build del frontend

Per la modalità single server è necessario generare la build del frontend, che verrà servita staticamente dal backend.

Dalla directory del frontend eseguire:

```
npm run build
```

Questo comando genera la cartella `frontend/dist`, contenente i file statici dell'applicazione frontend.

13.0.5 Avvio dell'applicazione

Una volta completata la build del frontend, l'applicazione viene avviata eseguendo il server backend, che si occupa anche di servire la build del frontend.

Dalla root eseguire:

```
npm start
```

A questo punto l'intera applicazione è accessibile tramite browser all'indirizzo:

`http://localhost:3000`

All'avvio, il backend inizializza automaticamente:

- la connessione al database MongoDB;
- il caricamento dei piatti iniziali dal file `meals.json`;
- la documentazione Swagger, disponibile all'endpoint `/api/docs`.

13.1 Esempio di utilizzo dell'applicazione

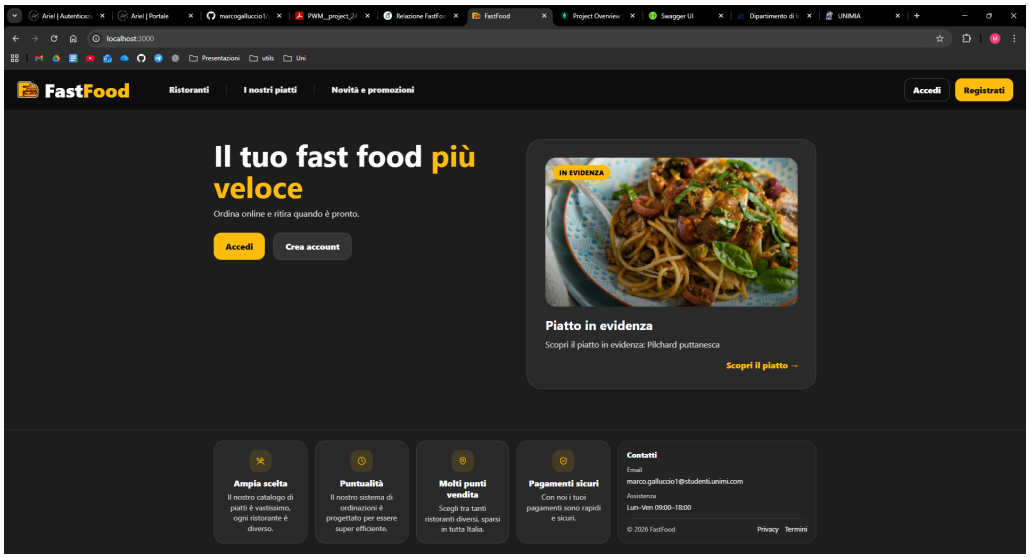
Di seguito viene riportato un flusso tipico di utilizzo dell'applicazione:

- registrazione di un nuovo utente come cliente o ristoratore;
- login all'applicazione;
- visualizzazione della home page, che varia dinamicamente in base al ruolo;
- esplorazione di ristoranti e piatti;
- creazione di un ordine (utente cliente);
- gestione degli ordini ricevuti (utente ristoratore).

13.2 Prove di funzionamento

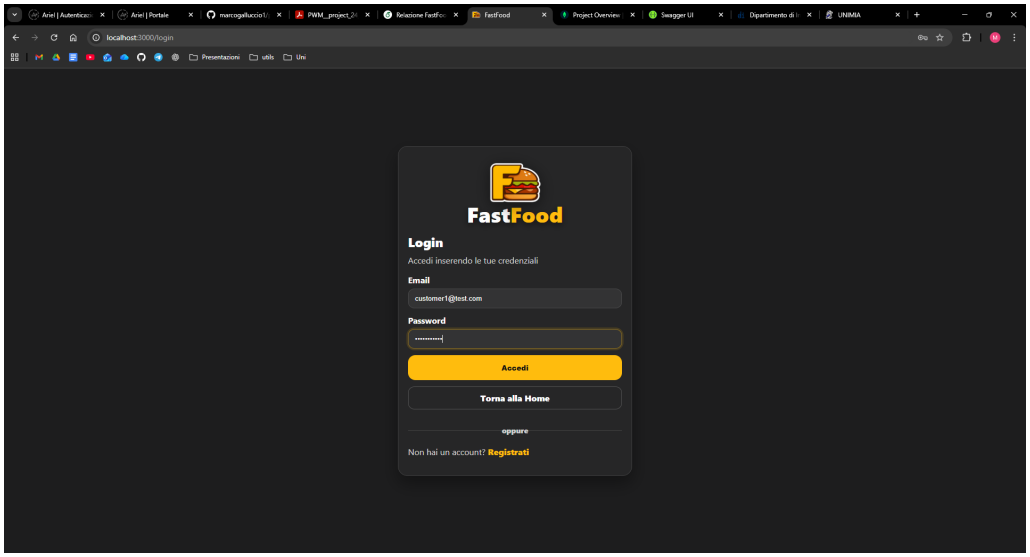
Le seguenti schermate dimostrano il corretto funzionamento delle principali funzionalità dell'applicazione.

13.2.1 Home page



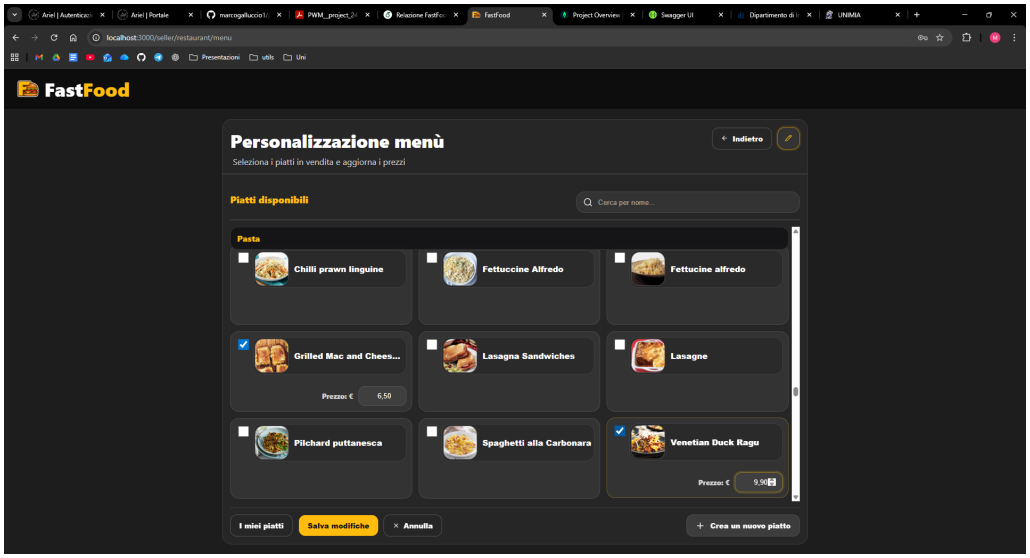
La schermata mostra la home page dell'applicazione, che varia dinamicamente in base allo stato di autenticazione e al ruolo dell'utente. In questo esempio l'utente non è ancora loggato.

13.2.2 Login



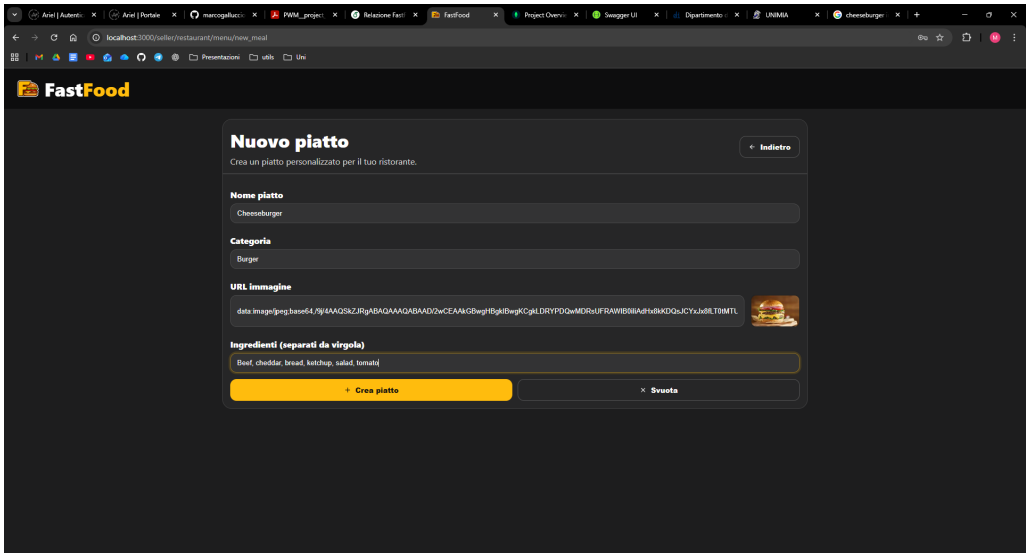
La schermata mostra la pagina di login in cui inserire email e password.

13.2.3 Gestione menù ristorante



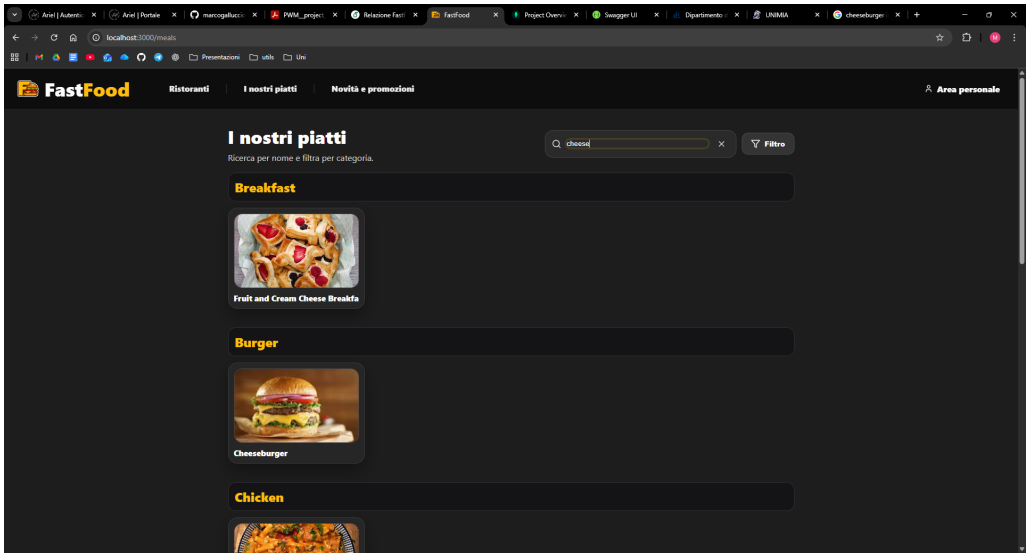
La schermata mostra la pagina di personalizzazione del menù da parte di un ristoratore.

13.2.4 Creazione piatto



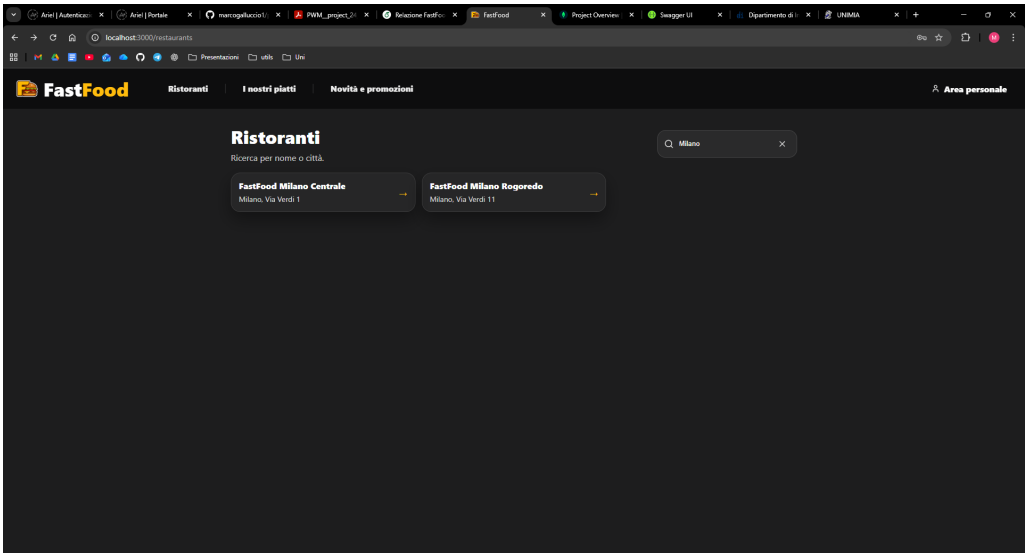
La schermata mostra la pagina in cui il ristoratore crea il suo piatto, impostando tutte le informazioni necessarie.

13.2.5 Ricerca piatti



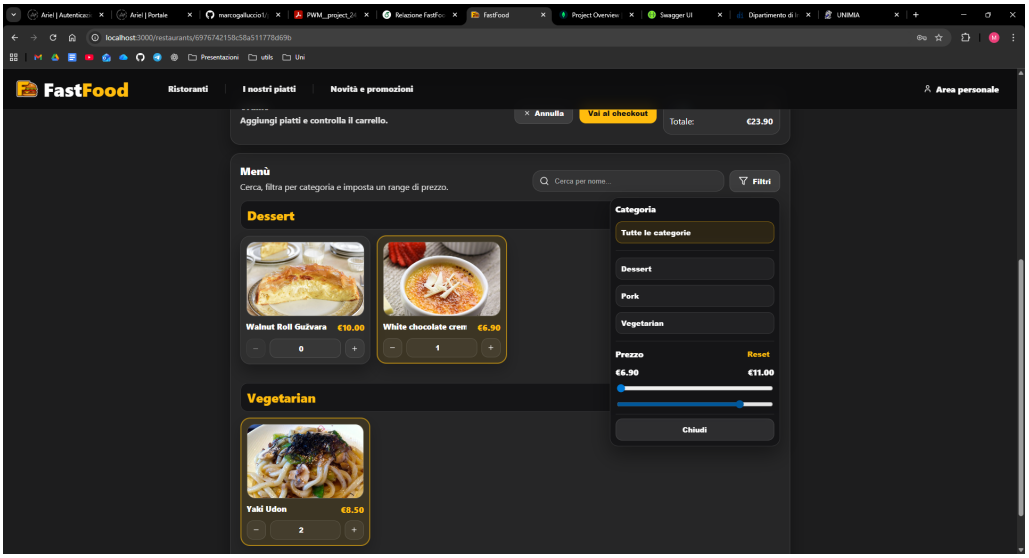
La schermata mostra la pagina di visualizzazione dei piatti, con ricerca e che include i piatti aggiunti dai ristoratori.

13.2.6 Ricerca ristoranti



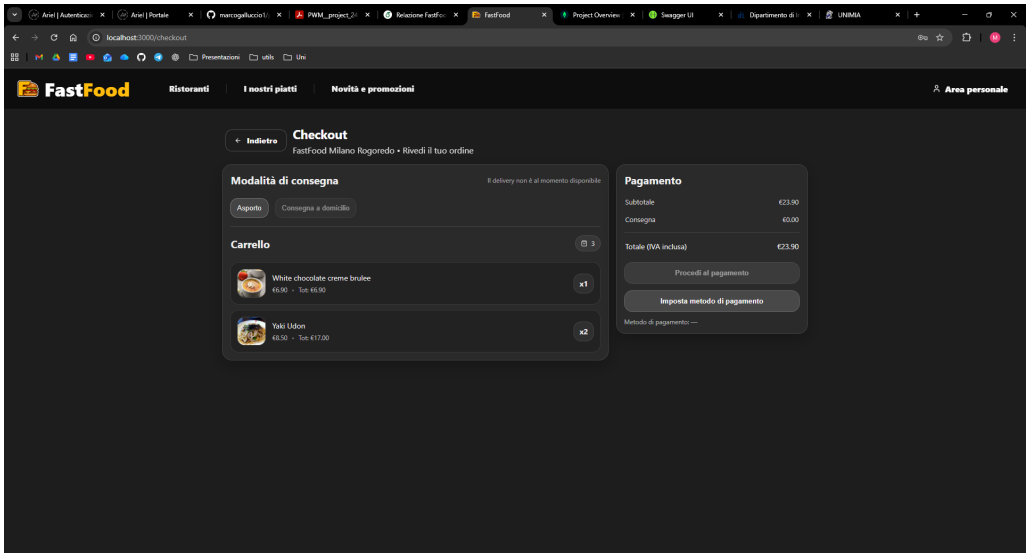
La schermata mostra la pagina di ricerca dei ristoranti, qui si può filtrare per nome o città.

13.2.7 Scelta dei piatti



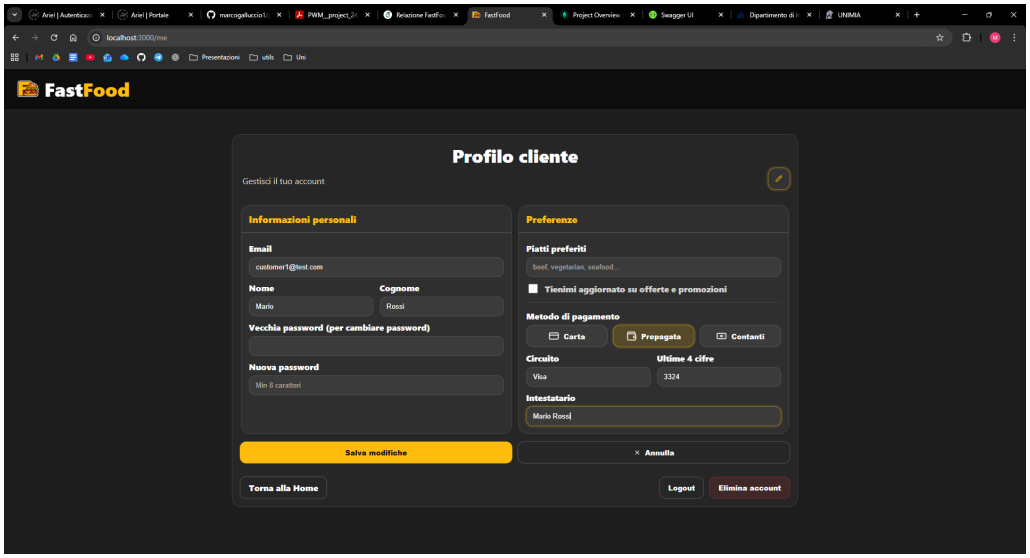
Nella pagina di un ristorante scelto, è possibile visionare il menù e il cliente può aggiungere i piatti che vuole al carrello, filtrandoli per nome, categoria o prezzo.

13.2.8 Checkout ordine



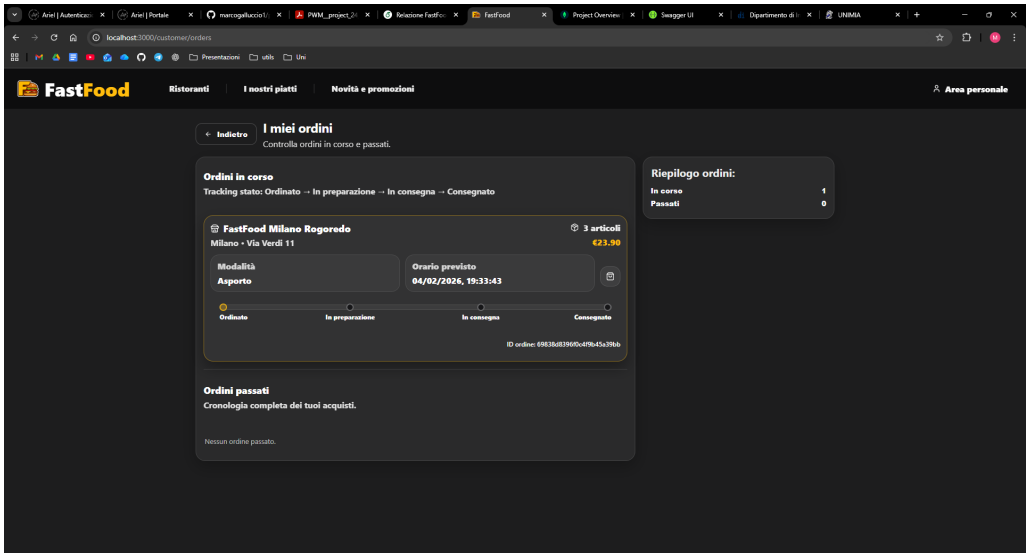
Dopo aver confermato l'ordine, si arriva alla pagina di checkout dell'ordine, con il riepilogo del totale e dei prodotti.

13.2.9 Modifica preferenze utente



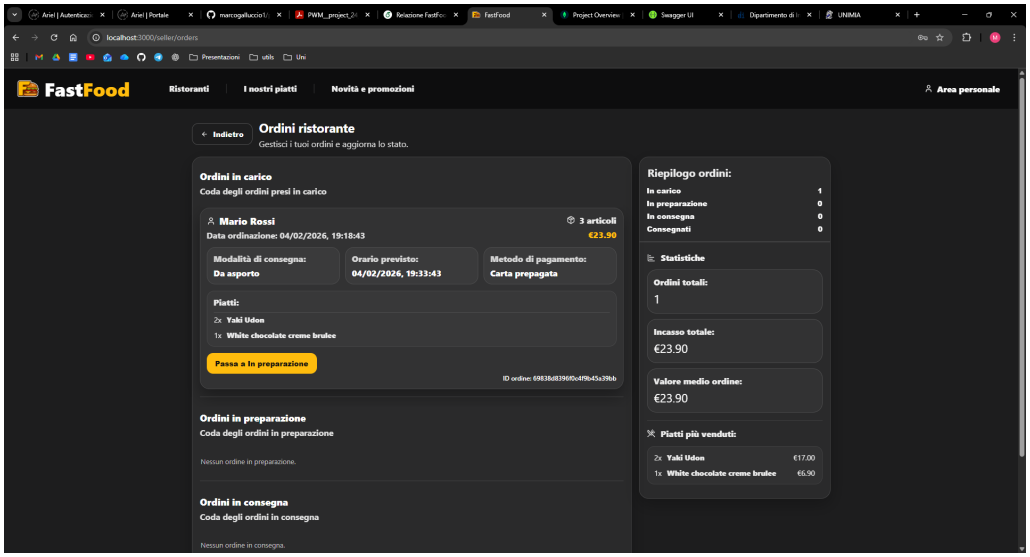
Non avendo inserito una modalità di pagamento nel profilo, si viene reindirizzati alla pagina di gestione dei dati utente.

13.2.10 Gestione ordini cliente



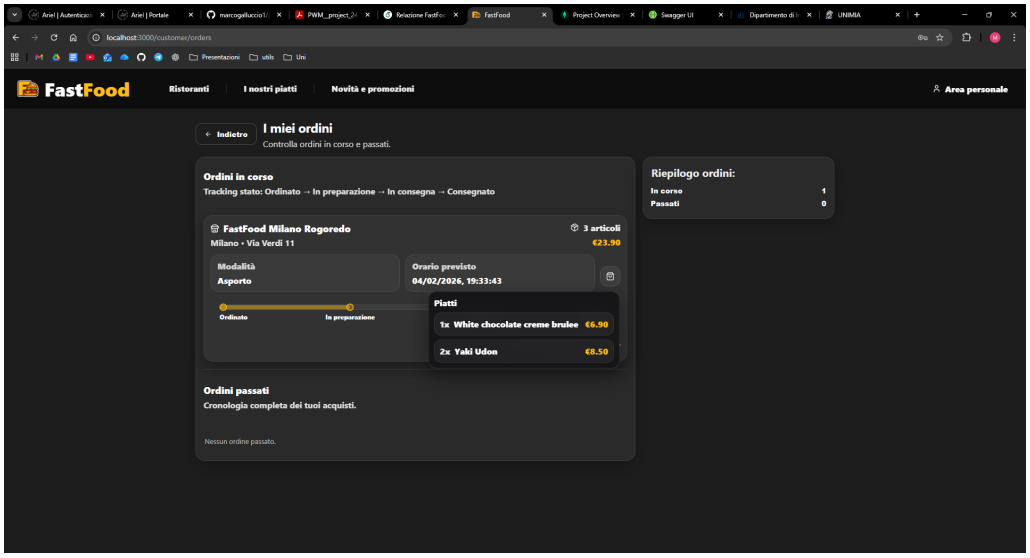
In questa pagina il cliente ha una panoramica dei propri ordini, attivi o passati.

13.2.11 Gestione ordini ristoratore



Il ristoratore può vedere i suoi ordini pendenti e ha una panoramica generale con anche le statistiche sulla destra.

13.2.12 Stato ordine



Dopo che il ristoratore ha messo a preparere l'ordine e aggiornato lo stato dell'ordinazione, il cliente può verificare il nuovo stato dell'ordine. Inoltre passando con il mouse sul carrello è possibile vedere i dettagli dei prodotti.