

# Relazione

## Progetto FastFood

Programmazione Web e Mobile

A.A. 2025/2026

*Simone Miglio*

Matricola: 978605

21 gennaio 2026

# Indice

|   |           |
|---|-----------|
| <b>Sommario Esecutivo</b>   | <b>3</b>  |
| <b>1 Architettura del Sistema</b>                                       | <b>4</b>  |
| 1.1 Struttura Generale dell'Applicazione . . . . .                      | 4         |
| 1.2 Tecnologie Utilizzate . . . . .                                     | 4         |
| 1.3 Organizzazione del Codice . . . . .                                 | 4         |
| <b>2 Progettazione del Database</b>                                     | <b>5</b>  |
| 2.1 Schema Concettuale . . . . .  | 5         |
| 2.2 Modelli di Dati Implementati . . . . .                              | 5         |
| 2.2.1 Utenti e Dati Sensibili . . . . .                                 | 5         |
| 2.2.2 Gestione Ordini (Order) . . . . .                                 | 6         |
| 2.2.3 Ciclo di Vita dell'Ordine . . . . .                               | 6         |
| <b>3 Implementazione del Backend</b>                                    | <b>7</b>  |
| 3.1 Configurazione del Server . . . . .                                 | 7         |
| 3.2 Sistema di Autenticazione . . . . .                                 | 7         |
| 3.3 Gestione delle API REST . . . . .                                   | 8         |
| 3.4 Documentazione e Testing API con Swagger . . . . .                  | 8         |
| <b>4 Implementazione del Frontend</b>                                   | <b>9</b>  |
| 4.1 Architettura Modulare JavaScript . . . . .                          | 9         |
| 4.2 Componenti UI Riutilizzabili . . . . .                              | 9         |
| 4.3 Gestione del Carrello . . . . .                                     | 10        |
| <b>5 Stima dei Tempi di Attesa</b>                                      | <b>10</b> |
| 5.1 Algoritmo di Calcolo . . . . .                                      | 10        |
| <b>6 Sfide Tecniche e Soluzioni</b>                                     | <b>11</b> |
| 6.1 Client-Side Adblocking su API . . . . .                             | 11        |
| 6.2 Accessibilità e Dark Mode . . . . .                                 | 11        |
| 6.3 Middleware Redirect per SPA . . . . .                               | 11        |
| <b>7 Operazioni Implementate</b>  | <b>12</b> |
| 7.1 Registrazione e Autenticazione Utenti . . . . .                     | 12        |
| 7.2 Gestione Ristoranti e Menu . . . . .                                | 12        |
| 7.3 Sistema di Ricerca Avanzata . . . . .                               | 13        |
| 7.4 Gestione Completa degli Ordini . . . . .                            | 13        |
| <b>8 Mappa dei Requisiti</b>  | <b>14</b> |
| <b>9 Testing e Qualità del Codice</b>                                   | <b>14</b> |
| 9.1 Suite di Test Implementata . . . . .                                | 14        |
| 9.2 Validazione e Sicurezza . . . . .                                   | 15        |
| 9.2.1 Validazione Rigorosa degli Input e Protezione Injection . . . . . | 15        |
| 9.2.2 Protezione Middleware . . . . .                                   | 15        |
| <b>10 Considerazioni di Design e User Experience</b>                    | <b>16</b> |

|   |           |
|---|-----------|
| 10.1 Responsive Design . . . . .            | 16        |
| 10.2 Accessibilità e Usabilità . . . . .    | 16        |
| <b>11 File HTML e Interfacce</b>            | <b>17</b> |
| 11.1 Pagine Principali . . . . .            | 17        |
| <b>12 Prove di Funzionamento</b>            | <b>19</b> |
| 12.1 Interfacce Specializzate . . . . .     | 19        |
| <b>13 Conclusioni e Sviluppi Futuri</b>     | <b>20</b> |
| 13.1 Risultati Raggiunti . . . . .          | 20        |
| 13.2 Possibili Miglioramenti . . . . .      | 20        |
| <b>A Codice Sorgente Completo</b>           | <b>21</b> |
| A.1 Modello User Completo . . . . .         | 21        |
| A.2 Configurazione Server Express . . . . . | 22        |
| A.3 Auth Controller Completo . . . . .      | 26        |
| A.4 API Client Frontend . . . . .           | 28        |
| A.5 Test Integrazione Auth . . . . .        | 30        |
| A.6 Cart Manager . . . . .                  | 32        |
| A.7 Restaurant Controller . . . . .         | 33        |
| A.8 Order Controller . . . . .              | 38        |

## Sommario Esecutivo

La presente relazione illustra la progettazione e implementazione dell'applicazione web **FastFood**, un sistema completo per la gestione di ordini online per ristoranti. Il progetto è stato sviluppato seguendo i quattro macro-scenari richiesti: gestione profili utente, gestione ristoranti, gestione ordini e gestione consegne, implementando un'architettura full-stack moderna con tecnologie Node.js, MongoDB, HTML5, CSS3 e JavaScript.

Il sistema supporta due tipologie di utenti distinte—clienti e ristoratori—with interfacce dedicate e funzionalità specifiche per ciascun ruolo. L'architettura implementa garantisce scalabilità, sicurezza e manutenibilità del codice.

Il codice sorgente completo è disponibile nel repository Forgejo: <https://forgejo.it/simonemiglio/FastFood>

# 1 Architettura del Sistema

## 1.1 Struttura Generale dell'Applicazione

Il sistema FastFood è stato progettato seguendo un'architettura *client-server* con separazione netta tra frontend e backend. L'applicazione adotta il paradigma **REST** per la comunicazione tra client e server, garantendo scalabilità e manutenibilità del codice.

La struttura del progetto è organizzata in due componenti principali: il backend Node.js che gestisce la logica di business e l'accesso ai dati, e il frontend che implementa l'interfaccia utente responsiva. Il backend utilizza MongoDB come database NoSQL per la persistenza dei dati, mentre il frontend è costituito da pagine HTML5 statiche arricchite con JavaScript per l'interattività.

L'architettura adottata permette la gestione di due tipologie di utenti distinte—clienti e ristoratori—with interfacce e funzionalità specifiche per ciascun ruolo. Il sistema implementa un sistema di autenticazione basato su *JSON Web Token* (JWT) per garantire la sicurezza delle sessioni utente e l'autorizzazione degli accessi alle risorse protette.

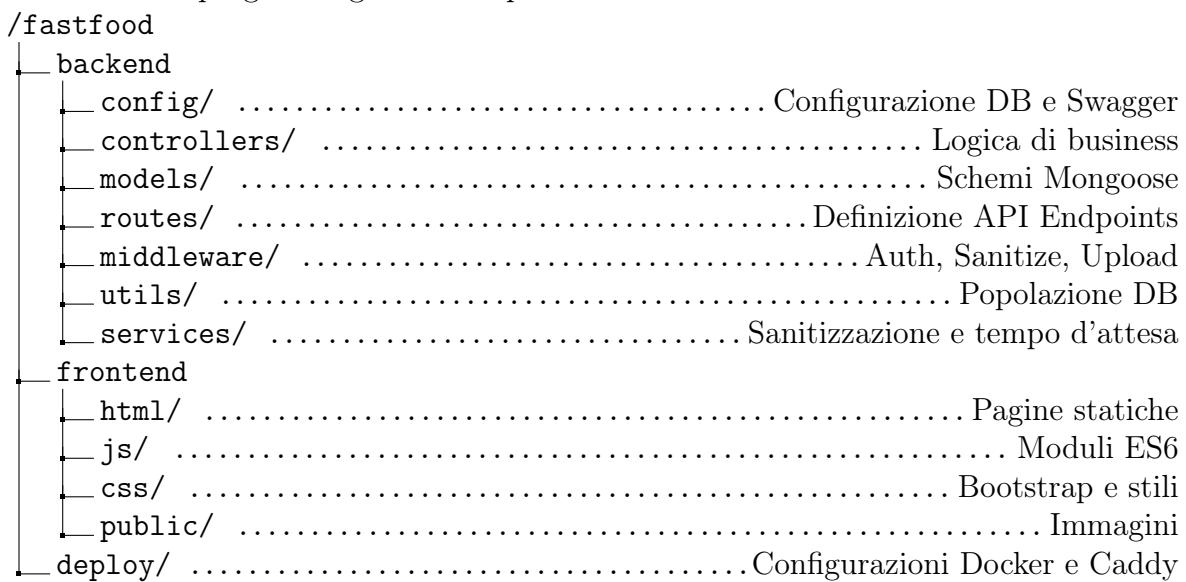
## 1.2 Tecnologie Utilizzate

Il backend è implementato utilizzando Node.js con il framework Express.js per la gestione delle route HTTP e dei middleware. Per l'accesso ai dati è stata scelta MongoDB con Mongoose ODM per la modellazione degli oggetti e la validazione dei dati. La sicurezza dell'applicazione è garantita attraverso l'uso di bcrypt per l'hashing delle password, helmet per la protezione delle intestazioni HTTP, e cors per la gestione delle richieste cross-origin.

Il frontend utilizza HTML5 semantico per la struttura delle pagine, CSS3 con Bootstrap 5 per lo styling responsivo, e JavaScript vanilla con ES6+ per l'interattività. L'applicazione implementa un'architettura Single Page Application (SPA) per alcune sezioni, mantenendo al contempo la capacità di navigazione tradizionale tra le pagine principali.

## 1.3 Organizzazione del Codice

La struttura del progetto segue le best practices MVC:



## 2 Progettazione del Database

### 2.1 Schema Concettuale

Il database è stato progettato per supportare efficacemente i quattro macro-scenari richiesti. Il modello dati è centrato attorno a cinque entità principali: User, Restaurant, Dish, Order e CustomerData.

L'entità User rappresenta il profilo base di tutti gli utenti del sistema, contenendo informazioni di autenticazione e dati personali comuni. Attraverso il campo userType, il sistema distingue tra clienti ("customer") e ristoratori ("owner"), implementando una gerarchia di utenti che consente l'applicazione di logiche di autorizzazione specifiche.

Listing 1: Schema User (Estratto - Vedi Appendice A.1)

```

1 // models/User.js
2 const userSchema = new mongoose.Schema({
3   username: {
4     type: String,
5     required: true,
6     match: [usernameRegex, '...']
7   },
8   // ... campi firstName, lastName, email ...
9   type: {
10     type: String,
11     required: true,
12     enum: Object.values(USER_TYPES)
13   }
14   // ...
15 });

```

### 2.2 Modelli di Dati Implementati

#### 2.2.1 Utenti e Dati Sensibili

Il modello User gestisce l'identità fondamentale, distinguendo tra clienti e ristoratori. Per i clienti, i dati sensibili (indirizzo, carte di pagamento) sono segregati nel modello CustomerData per una migliore organizzazione logica e sicurezza. I campi sensibili come il numero della carta di credito vengono parzialmente offuscati (masked) prima del salvataggio o sono memorizzati solo se strettamente necessario, validati tramite Regex rigorose.

Listing 2: Schema CustomerData per dati sensibili

```

1 // models/CustomerData.js
2 const CustomerDataSchema = new mongoose.Schema({
3   user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', unique:
4     true },
5   address: {
6     streetAddress: { type: String, required: true },
7     city: { type: String, required: true },
8     // Validazione provincia (2 lettere maiuscole)
9     province: { type: String, match: [/^([A-Z]{2})$/], 'Province must
      be 2 letters' },
10    zipCode: { type: String, match: [/^\d{5}$/], 'ZIP must be 5
      digits' }
11  }
12 });

```

```

10 },
11   cards: [
12     cardNumber: { type: String, match: [/^(\*\{12\}\d{4}|\d{16})$/], 'Invalid card' },
13     expiryDate: { type: String, match: [/(0[1-9]|1[0-2])\//\d{2}$/], 'MM/YY' }
14   ]
15 });

```

## 2.2.2 Gestione Ordini (Order)

Il modello `Order` gestisce il ciclo di vita delle ordinazioni. Funge da entità di collegamento tra il cliente, il ristorante e il piatto scelto. Oltre a memorizzare quantità e prezzo calcolato lato server, traccia lo stato di avanzamento dell'ordine (es. ricevuto, in preparazione, completato) e definisce indici per ottimizzare la cronologia ordini.

Listing 3: Schema del modello Order

```

1 // models/Order.js
2 const OrderSchema = new mongoose.Schema({
3   customer: {
4     type: mongoose.Schema.Types.ObjectId,
5     ref: 'User',
6     required: true
7   },
8   restaurant: {
9     type: mongoose.Schema.Types.ObjectId,
10    ref: 'Restaurant',
11    required: true
12  },
13   dish: {
14     type: mongoose.Schema.Types.ObjectId,
15     ref: 'Dish',
16     required: true
17   },
18   amount: { type: Number, required: true, min: 1 },
19   price: { type: Number, required: true }, // Prezzo calcolato
20   state: {
21     type: String,
22     enum: ['received', 'preparing', 'ready', 'completed'],
23     default: 'received'
24   },
25   createdAt: { type: Date, default: Date.now }
26 });
27
28 // Indici per ottimizzare le query cronologiche per cliente e ristorante
29 OrderSchema.index({ customer: 1, createdAt: -1 });
30 OrderSchema.index({ restaurant: 1, createdAt: -1 });

```

## 2.2.3 Ciclo di Vita dell'Ordine

L'entità `Order` è il cuore pulsante dell'applicazione. Essa traccia il flusso completo:

- Creazione:** Il cliente finalizza il carrello; viene creato un documento `Order` in stato 'received'.

2. **Preparazione:** Il ristoratore accetta l'ordine, passando lo stato a 'preparing'.
3. **Consegna:** Al termine della preparazione, lo stato diventa 'delivering'.
4. **Chiusura:** Alla consegna avvenuta, lo stato finale è 'completed'.

Questo flusso è garantito da una macchina a stati implementata nel backend che valuta le transizioni valide (es. impossibile passare da 'received' a 'completed' direttamente).

Il modello Order gestisce l'intero ciclo di vita degli ordini, dal momento della creazione fino alla consegna finale. Include informazioni sul cliente, il ristorante, i piatti ordinati con le rispettive quantità, lo stato dell'ordine e i dettagli di consegna. Gli stati dell'ordine (mappati internamente come 'received', 'preparing', 'delivering', 'completed') seguono il flusso richiesto: ordinato, in preparazione, in consegna, consegnato.

## 3 Implementazione del Backend

### 3.1 Configurazione del Server

Il server principale è configurato nel file `server.js` che inizializza l'applicazione Express con tutti i middleware necessari per sicurezza, parsing delle richieste e gestione degli errori. La configurazione include `helmet` per la sicurezza delle intestazioni, `cors` per le richieste cross-origin, e `express-rate-limit` per la protezione contro attacchi di forza bruta.

Il backend è sviluppato in **Node.js v20** utilizzando la sintassi ES6 Modules (`import/export`).

Listing 4: Server Express (Estratto - Vedi Appendice A.2)

```

1 // server.js
2 const startServer = async () => {
3   // ... connessione DB ...
4   const app = express();
5
6   app.use(helmet({ contentSecurityPolicy: { ... } }));
7   app.use(cors({ origin: process.env.ALLOWED_ORIGINS }));
8
9   // Middleware e Route
10  app.use('/auth', authRoutes);
11  app.use('/api', apiLimiter, apiRoutes);
12
13  return app;
14};

```

### 3.2 Sistema di Autenticazione

L'autenticazione è implementata attraverso il controller `authController.js` che gestisce registrazione, login e logout degli utenti. Il sistema utilizza `bcrypt` per l'hashing sicuro delle password e `JWT` per la gestione delle sessioni. I token JWT sono memorizzati come cookie HTTP-only per garantire maggiore sicurezza contro attacchi XSS.

Listing 5: Auth Controller (Estratto - Vedi Appendice A.3)

```

1 // controllers/authController.js
2 export const registerUser = async (req, res, next) => {
3   // ... logica registrazione ...

```

```

4     res.status(201).json({ message: 'User successfully registered!' });
5 }
6
7 export const loginUser = async (req, res, next) => {
8     // ... verifica credenziali e generazione JWT ...
9     res.cookie('token', token, { httpOnly: true });
10    res.json({ message: 'Login successful' });
11}

```

### 3.3 Gestione delle API REST

Le API REST sono organizzate in moduli separati per ogni entità del sistema. Il file apiRoutes.js centralizza tutte le route di business logic, mentre authRoutes.js gestisce specificatamente le operazioni di autenticazione. Ogni route implementa appropriati middleware di autenticazione e autorizzazione.

Il sistema di middleware implementa tre livelli di controllo: auth.js per verificare l'autenticazione generale, onlyCustomers.js per limitare l'accesso alle funzionalità cliente, e onlyOwners.js per le operazioni riservate ai ristoratori. Questo approccio garantisce che solo gli utenti autorizzati possano accedere alle rispettive funzionalità.

### 3.4 Documentazione e Testing API con Swagger

Il sistema include una suite completa basata su Swagger che assolve due funzioni critiche: documentazione e testing. Oltre a generare automaticamente le specifiche OpenAPI 3.0 dai commenti JSDoc, l'interfaccia Swagger UI permette agli sviluppatori di **testare manualmente** gli endpoint in tempo reale, simulando richieste client senza la necessità di sviluppare interfacce frontend dedicate, accelerando il ciclo di sviluppo e debug.

Listing 6: Configurazione Swagger

```

1 // config/swagger.js
2 const swaggerJsdock = require('swagger-jsdoc');
3 const swaggerUi = require('swagger-ui-express');
4
5 const options = {
6     definition: {
7         openapi: '3.0.0',
8         info: {
9             title: 'FastFood API',
10            version: '1.0.0',
11            description: 'API documentation for FastFood ordering system',
12        },
13        servers: [
14            {
15                url: 'http://localhost:3000',
16                description: 'Development server',
17            },
18        ],
19    },
20    apis: ['./routes/*.js', './models/*.js'],
21};
22
23 const specs = swaggerJsdock(options);
24 module.exports = { swaggerUi, specs };

```

## 4 Implementazione del Frontend

### 4.1 Architettura Modulare JavaScript

Il frontend è strutturato seguendo un approccio modulare con separazione delle responsabilità. Il modulo `api.js` centralizza tutte le comunicazioni con il backend, esportando la funzione asincrona `fetchApi` che gestisce autenticazione, errori e stati di caricamento.

Listing 7: API Client (Estratto - Vedi Appendice A.4)

```

1 // frontend/js/api.js
2 export async function fetchApi(url, options = {}, button = null) {
3     if (button) setLoadingState(button, true);
4     try {
5         const res = await fetch(config.API_BASE_URL + url, {
6             credentials: 'include',
7             ...options
8         });
9         // ... gestione errori e response ...
10        return await res.json();
11    } catch (err) { return null; }
12    finally { if (button) setLoadingState(button, false); }
13 }
```

### 4.2 Componenti UI Riutilizzabili

Il modulo `components.js` esporta funzioni pure per la creazione dinamica di elementi DOM, garantendo coerenza grafica e facilità di manutenzione.

Listing 8: Componenti UI riutilizzabili

```

1 // frontend/js/components.js
2 export function createCard({ imageSrc, title, bodyText, onClick,
3     colClass }) {
4     const col = document.createElement('div');
5     col.className = colClass || 'col-12 col-md-6 col-lg-4';
6
7     const card = document.createElement('div');
8     card.className = 'card h-100';
9     if (onClick) {
10         card.style.cursor = 'pointer';
11         card.onclick = onClick;
12     }
13
14     const img = document.createElement('img');
15     img.className = 'card-img-top image-clip';
16     img.src = imageSrc;
17
18     const body = document.createElement('div');
19     body.className = 'card-body d-flex flex-column';
20     body.innerHTML = `<h5 class="card-title">${title}</h5>
21             <p class="card-text mt-auto">${bodyText} || ,></p>
22         `;
23     card.append(img, body);
24     col.append(card);
25 }
```

```
24     return col;
25 }
```

### 4.3 Gestione del Carrello

Il sistema di gestione del carrello è implementato attraverso il modulo `cartManager.js` che utilizza `localStorage` per la persistenza. Utilizza un formato chiave stringa `restaurantId:dishId` per gestire le quantità e impedisce l'aggiunta di articoli da ristoranti diversi nello stesso ordine.

Listing 9: Gestore Carrello (Estratto - Vedi Appendice A.6)

```
1 // frontend/js/cartManager.js
2 export function addToCart(item, amount) {
3     const cart = getCart();
4     // ... controllo consistenza ristorante ...
5     if (existingRestaurants.length > 0 && !existingRestaurants.includes(
6         restaurantId)) {
7         if (!confirm('...')) return false;
8         localStorage.removeItem('cart');
9     }
10    // ... aggiornamento quantit e salvataggio ...
11    saveCart(currentCart);
12    return true;
13 }
```

## 5 Stima dei Tempi di Attesa

### 5.1 Algoritmo di Calcolo

Il sistema fornisce una stima precisa calcolando il tempo necessario per smaltire la coda di preparazione. L'algoritmo non usa costanti fisse, ma:

- Identifica la posizione dell'ordine corrente nella coda del ristorante.
- Somma i tempi di preparazione specifici (definiti nel menu) di tutti gli ordini precedenti.
- Sottrae il tempo già trascorso se il primo ordine della coda è già nello stato ‘preparing’.

Listing 10: Calcolo tempo di attesa reale

```
1 // controllers/orderController.js
2 export const waitEstimation = async (req, res, next) => {
3     // ... recupero ordine e ristorante ...
4
5     // Trova l'indice dell'ordine nella coda prioritaria del ristorante
6     const queuePosition = restaurant.queue.findIndex(id => id.toString() === orderId);
7     const orderIdsAhead = restaurant.queue.slice(0, queuePosition + 1);
8
9     // Aggregazione: somma i tempi di preparazione dei piatti in coda
```

```

10  const result = await Order.aggregate([
11    { $match: { _id: { $in: orderIdsAhead } } },
12    { /* $lookup per recuperare prepTime dal menu del ristorante */
13    },
14    {
15      $group: {
16        _id: null,
17        totalTime: { $sum: { $multiply: ['$amount', '$menuData.
18          prepTime'] } }
19      }
20    }
21  ];
22
23  let waitingTime = result[0].totalTime || 0;
24
25  // Raffinamento: sottrae il tempo già trascorso per l'ordine in cottura
26  if (firstOrder.state === 'preparing' && restaurant.
27    lastPreparationStart) {
28    const elapsed = (Date.now() - new Date(restaurant.
29    lastPreparationStart)) / 60000;
30    waitingTime = Math.max(waitingTime - elapsed, 0);
31  }
32
33  res.json({ time: Math.round(waitingTime * 10) / 10 });
34

```

## 6 Sfide Tecniche e Soluzioni

Durante lo sviluppo sono state affrontate diverse sfide tecniche significative:

### 6.1 Client-Side Adblocking su API

È stato riscontrato un problema critico dove le richieste verso l'endpoint `/api/estimate` venivano bloccate da alcune estensioni browser (es. AdBlock, uBlock) a causa del matching della parola chiave "estimate" o "metrics". **Soluzione:** L'endpoint è stato rinominato in `/api/eta/:id`. Questo caso studio evidenzia l'importanza strategica nel naming delle API pubbliche: evitare termini "trigger" (come *ad*, *track*, *analytics*) previene falsi positivi nei filtri di privacy.

### 6.2 Accessibilità e Dark Mode

L'implementazione della modalità scura (Dark Mode) ha introdotto sfide di contrasto, in particolare nel form di checkout dove il numero della carta di credito risultava invisibile (bianco su sfondo bianco o nero su nero) a causa di classi CSS conflittuali (`text-dark`). **Soluzione:** Rimozione delle classi di colore forzate per permettere l'ereditarietà corretta dallo stile del tema attivo (Bootstrap 5 dark mode).

### 6.3 Middleware Redirect per SPA

Inizialmente, il middleware di controllo setup (`setupCheck.js`) ritornava errori JSON 403 anche per le richieste di navigazione browser, impedendo il redirect fluido alla pagina

di completamento profilo. **Soluzione:** Implementazione di content-negotiation basata sull'header `Accept`. Se il client richiede HTML, il server risponde con un redirect HTTP 302; se richiede JSON (chiamata API), risponde con errore 403.

## 7 Operazioni Implementate

### 7.1 Registrazione e Autenticazione Utenti

Il sistema implementa un flusso di registrazione completo che distingue tra clienti e ristoratori fin dal momento della creazione dell'account. Durante la registrazione, gli utenti selezionano il proprio tipo di account, influenzando le funzionalità disponibili e l'interfaccia presentata dopo il login.

Il processo di autenticazione verifica le credenziali utente e genera token JWT sicuri per mantenere la sessione. Il sistema implementa controlli di sicurezza come rate limiting per prevenire attacchi di forza bruta e validazione server-side di tutti i dati in input per prevenire injection attacks.

### 7.2 Gestione Ristoranti e Menu

Il sistema permette ai ristoratori la gestione completa del profilo e del menu. La creazione del ristorante collega l'entità al 'userId' del proprietario e gestisce l'upload delle immagini tramite middleware. La modifica dei dati supporta l'aggiornamento di campi nidificati (come l'indirizzo) e la gestione dinamica del menu, permettendo l'inserimento di piatti con prezzi e descrizioni dettagliate.

Listing 11: Creazione Ristorante con Upload Immagini

```

1 // controllers/restaurantController.js
2 export const addRestaurant = async (req, res, next) => {
3     try {
4         const user = req.user; // Identificato dal token JWT
5         const restaurant = req.body.restaurant;
6
7         const newRestaurant = new Restaurant({
8             owner: user.userId,
9             name: restaurant.name,
10            address: restaurant.address, // Oggetto nidificato (via,
11            citt , cap)
12            vatNumber: restaurant.vatNumber,
13            phoneNumber: restaurant.phoneNumber,
14            // Gestione percorso immagine se caricata
15            image: req.file ? '/images/uploads/${req.file.filename}' :
16                undefined
17        });
18
19        await newRestaurant.save();
20        // ... generazione token aggiornato ...
21        res.status(201).json({ message: 'Restaurant created successfully
22        .', });
23    } catch (err) { next(err); }
24}
```

### 7.3 Sistema di Ricerca Avanzata

Il backend implementa una logica di ricerca flessibile basata su espressioni regolari (Regex) per garantire risultati anche con corrispondenze parziali (case-insensitive). Le API supportano:

- **Ricerca Geografica:** Filtri per indirizzo e città, oppure localizzazione "Nearby" basata sulla corrispondenza del CAP (Zip Code).
- **Paginazione:** Gestione efficiente di grandi volumi di dati tramite parametri 'skip' e 'limit'.

Listing 12: Logica di Ricerca e Paginazione

```

1 // controllers/restaurantController.js
2 export const searchRestaurants = async (req, res, next) => {
3     const { name, street, city, page = 1 } = req.query;
4     const limit = 20;
5     const skip = (page - 1) * limit;
6
7     const filter = { active: true }; // Solo ristoranti attivi
8
9     // Filtri dinamici con Regex (Case Insensitive)
10    if (name) filter.name = { $regex: name, $options: 'i' };
11    if (street) filter['address.streetAddress'] = { $regex: street,
$options: 'i' };
12    if (city) filter['address.city'] = { $regex: city, $options: 'i' };
13
14    const total = await Restaurant.countDocuments(filter);
15    const restaurants = await Restaurant.find(filter)
16        .skip(skip)
17        .limit(limit);
18
19    res.json({ total, restaurants });
20 }
```

### 7.4 Gestione Completa degli Ordini

Il sistema di gestione ordini copre l'intero ciclo di vita dall'aggiunta al carrello fino alla consegna finale. Gli ordini seguono il flusso di stati richiesto: ordinato, in preparazione, in consegna, consegnato. I clienti possono tracciare lo stato dei propri ordini in tempo reale, mentre i ristoratori possono gestire la coda degli ordini e aggiornare gli stati di preparazione.

Listing 13: Controller per la gestione ordini

```

1 // controllers/orderController.js
2 exports.updateOrderStatus = async (req, res) => {
3     // ... verifica owner ...
4     const validTransitions = {
5         'ordinato': ['in preparazione'],
6         'in preparazione': ['in consegna', 'consegnato'],
7         // ...
8     };
9     if (!validTransitions[order.status]?.includes(status)) {
```

```

10     return res.status(400).json({ message: 'Invalid status transition' });
11 }
12 // ... salvataggio ...
13 }

```

## 8 Mappa dei Requisiti

La seguente tabella mappa i requisiti funzionali richiesti alle sezioni della relazione e al codice implementato, dimostrando la copertura completa delle specifiche.

| Requisito                                  | Codice / File                | Sezione Relazione   |
|--|------------------------------|---------------------|
| Gestione Utenti (Clienti/Ristoratori)      | User.js, authController.js   | 7.1 Registrazione   |
| Gestione Dati Sensibili (Carte, Indirizzi) | CustomerData.js              | 3.2.1 Utenti e Dati |
| Gestione Ristoranti e Menu                 | Restaurant.js, Dish.js       | 7.2 Ristoranti      |
| Ricerca (Filtri multipli)                  | restaurantController.js      | 7.3 Ricerca         |
| Gestione Carrello (Storage Locale)         | cartManager.js               | 5.3 Carrello        |
| Creazione Ordini                           | Order.js, orderController.js | 7.4 Ordini          |
| Gestione Stati Ordine (Macchina a stati)   | orderController.js           | 3.2.3 Ciclo Ordine  |
| Stima Tempi Attesa (Algoritmo)             | api/eta endpoint             | 6. Stima Tempi      |

Tabella 1: Mappa di copertura dei requisiti

## 9 Testing e Qualità del Codice

### 9.1 Suite di Test Implementata

Il progetto include una suite completa che utilizza **Jest** e **Supertest**. I test di integrazione fanno uso di **MongoMemoryServer** per garantire un ambiente database isolato e volatile per ogni esecuzione. Particolare attenzione è stata posta ai **Test di Sicurezza** (**security.test.js**), che non si limitano al controllo accessi ma verificano proattivamente:

- La **Sanitizzazione degli Input**: rimozione automatica di chiavi pericolose (es. operatori \$gt) per prevenire NoSQL Injection.
- La **Sicurezza degli Upload**: rifiuto sistematico di file che non siano immagini valide.

Listing 14: Test Integrazione: Registrazione con Upload (Estratto)

```

1 // __tests__/auth.integration.test.js
2 describe('Auth Integration Tests', () => {
3     // Setup MongoMemoryServer e connessione DB (beforeAll/afterAll)
4     // ...
5
6     it('should register a new user', async () => {
7         const res = await request(app)
8             .post('/auth/register')

```

```

9     .field('username', 'testuser')
10    .field('email', 'test@example.com')
11    .field('password', 'Password123!')
12    .field('type', 'customer')
13    // Simulazione upload file immagine
14    .attach('image', Buffer.from('fakeimage'), 'test.jpg');

15
16    expect(res.statusCode).toBe(201);
17    expect(res.body).toHaveProperty('message', 'User successfully
18 registered!');
19 });
}

```

## 9.2 Validazione e Sicurezza

Il sistema implementa una strategia di difesa in profondità (Defense in Depth) combinando validazione dei dati, sicurezza del trasporto e protezione contro attacchi comuni.

### 9.2.1 Validazione Rigorosa degli Input e Protezione Injection

La sicurezza dei dati è garantita a più livelli. L'utilizzo di **Mongoose ODM** protegge nativamente dalla maggior parte delle NoSQL Injection sanificando le query. A questo si aggiunge un livello di validazione esplicita per la logica di business: ogni campo sensibile è sottoposto a stretta verifica tramite Espressioni Regolari (Regex) definite nel backend.

- **Username:** 3-20 caratteri alfanumerici.  
Regex: `/^\w{1,20}$/`
- **Password:** Min 8 car, 1 Maiusc, 1 num, 1 speciale.  
Regex: `/^(?=.*[a-z])(?=.*[A-Z])(?=.*[\d])(?=.*[!@#$%^&*]).{8,32}$/`
- **Email:** Formato standard email.  
Regex: `/^[\s@]+@[^\s@]+\.\.[^\s@]{2,}$/`
- **Provincia:** 2 lettere maiuscole.  
Regex: `/^[A-Z]{2}$/`
- **Carta Credito:** 16 cifre o Mascherata.  
Regex: `/^(*\{12}\d\{4}|\d\{16})$/`

### 9.2.2 Protezione Middleware

L'applicazione utilizza un set robusto di middleware di sicurezza:

- **Helmet:** Configura header HTTP sicuri, inclusa una Content-Security-Policy (CSP) che restringe le fonti di immagini e script a 'self' e domini fidati.
- **Express-Rate-Limit:** Limita le richieste ripetute dallo stesso IP (max 100 req/15min per login) per mitigare attacchi Brute Force.
- **HttpOnly Cookies:** I token JWT sono inaccessibili via JavaScript lato client, prevenendo furti di sessione tramite XSS.

- **CORS:** Configurato per accettare richieste solo dall'origine autorizzata definita nelle variabili d'ambiente.

## 10 Considerazioni di Design e User Experience

### 10.1 Responsive Design

L'interfaccia utente è completamente responsiva, utilizzando Bootstrap 5 come framework CSS base con personalizzazioni specifiche per il brand FastFood. Il design si adatta fluidamente a dispositivi mobili, tablet e desktop, garantendo un'esperienza ottimale su tutti i form factor.

Listing 15: Stili personalizzati responsivi

```

1  /* frontend/css/style.css */
2  .restaurant-card {
3      transition: transform 0.2s ease-in-out, box-shadow 0.2s ease-in-out;
4      cursor: pointer;
5  }
6
7  .restaurant-card:hover {
8      transform: translateY(-5px);
9      box-shadow: 0 8px 25px rgba(0,0,0,0.15);
10 }
11
12 .dish-card {
13     border: none;
14     border-radius: 15px;
15     overflow: hidden;
16     transition: all 0.3s ease;
17 }
18
19 .dish-card:hover {
20     transform: scale(1.02);
21     box-shadow: 0 10px 30px rgba(0,0,0,0.1);
22 }
23
24 @media (max-width: 768px) {
25     .restaurant-grid {
26         grid-template-columns: 1fr;
27     }
28
29     .dish-grid {
30         grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
31     }
32 }
```

### 10.2 Accessibilità e Usabilità

Il sistema implementa principi di accessibilità web seguendo le linee guida WCAG, includendo etichette appropriate per form, contrasti di colore adeguati, navigazione da tastiera e supporto per screen reader. L'interfaccia utilizza icone intuitive e feedback visivi chiari per guidare l'utente attraverso i vari flussi operativi.

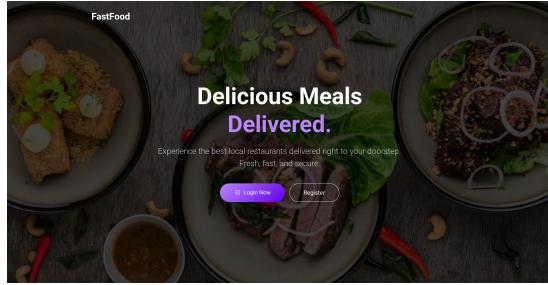


Figura 1: Home Desktop

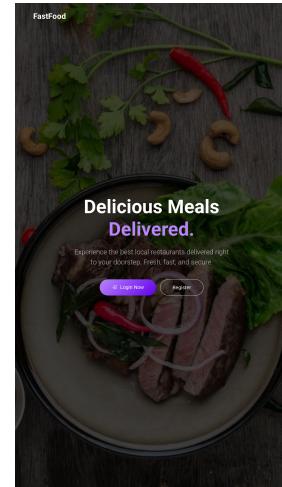


Figura 2: Home Mobile

La user experience è ottimizzata attraverso caricamento lazy delle immagini, stati di loading chiari durante le operazioni asincrone, messaggi di errore informativi e conferme per azioni critiche come eliminazione di dati o finalizzazione di ordini.

## 11 File HTML e Interfacce

### 11.1 Pagine Principali

Il sistema include diverse pagine HTML specializzate per ogni funzionalità. La pagina login.html fornisce l'interfaccia di autenticazione con validazione in tempo reale, mentre register.html gestisce la registrazione di nuovi utenti con selezione del tipo di account.

Listing 16: Pagina di login principale

```

1 <!DOCTYPE HTML>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0
" >
6     <title>FastFood</title>
7     <link rel="stylesheet" href="/css/style.css">
8     <link rel="stylesheet" href="/bootstrap/css/bootstrap.min.css">
9 </head>
10 <body data-bs-theme="dark">
11     <div id="content">
12         <div class="d-flex flex-column justify-content-center align-
13             items-center vh-100">
14             <div class="section-card col-11 col-sm-8 col-md-6 col-lg-4">
15                 <h2 class="text-center mb-4">Login to FastFood</h2>
16                 <div class="w-auto">
17                     <label for="username" class="visually-hidden">
18                         Username</label>
19                         <input type="text" id="username" class="form-control
20                             rounded-bottom-0" placeholder="Username" required>
21                         <label for="password" class="visually-hidden">
22                             Password</label>

```

```
19          <input type="password" id="password" class="form-
control rounded-top-0" placeholder="Password" required>
20      </div>
21      <div class="mt-3 d-grid gap-2">
22          <button type="submit" id="login-btn" class="btn btn-
primary">Login</button>
23      </div>
24      <div class="text-center mt-3">
25          <a href="/register">Don't have an account? Register
26      </a>
27      </div>
28  </div>
29</div>
30
31 <script src="/bootstrap/js/bootstrap.bundle.min.js"></script>
32 <script type="module" src="/js/layout.js"></script>
33 <script type="module" src="/js/errorManager.js"></script>
34 <script type="module" src="/js/login.js"></script>
35</body>
36</html>
```

## 12 Prove di Funzionamento

In questa sezione vengono presentati screenshot dimostrativi delle funzionalità principali dell'applicazione, a riprova del soddisfacimento dei requisiti.

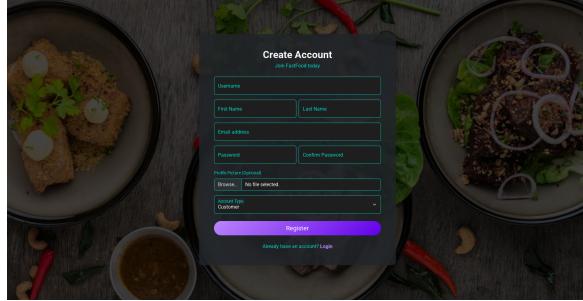


Figura 3: Registrazione Utente

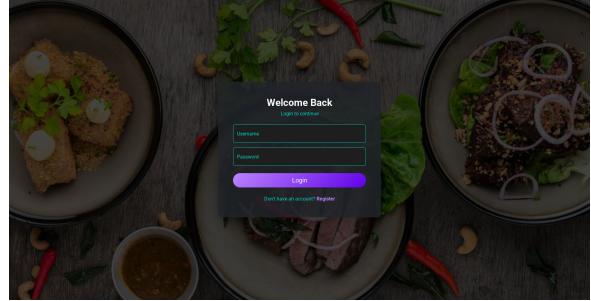


Figura 4: Login Utente

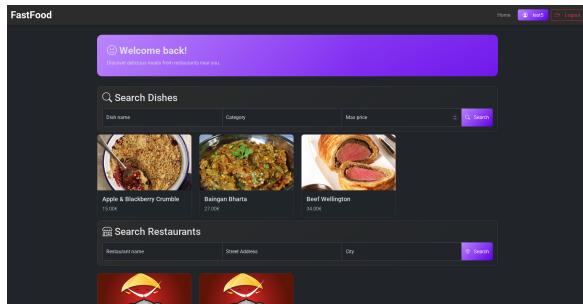


Figura 5: Page Ricerca Ristoranti

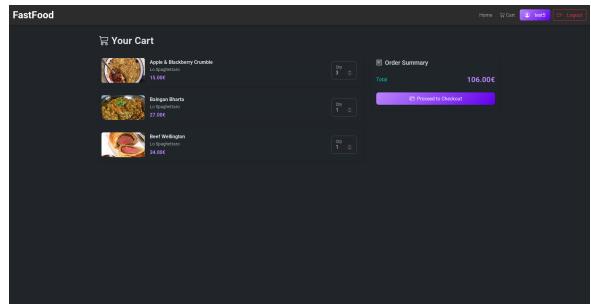


Figura 6: Gestione Carrello

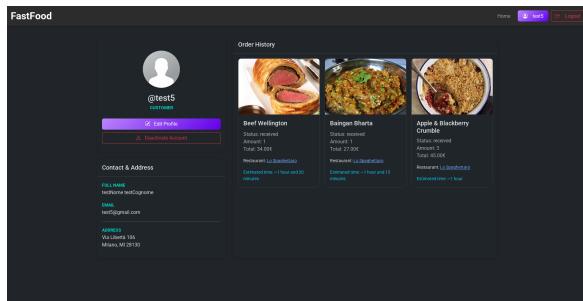


Figura 7: Storico Ordini Cliente

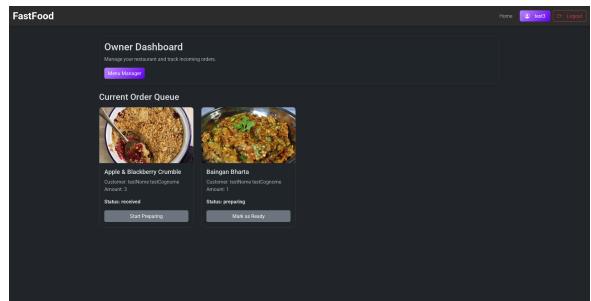


Figura 8: Dashboard Ristoratore

### 12.1 Interfacce Specializzate

Le pagine homeCustomer.html e homeOwner.html forniscono dashboards specifiche per ogni tipo di utente. La pagina restaurantPage.html permette ai clienti di visualizzare i menu e aggiungere piatti al carrello, mentre menuManager.html consente ai ristoratori di gestire i propri menu.

Il sistema include anche pagine per la gestione del profilo (editProfile.html), visualizzazione del carrello (cart.html), processo di checkout (checkout.html), e configurazione iniziale per nuovi ristoranti (addRestaurant.html).

## 13 Conclusioni e Sviluppi Futuri

### 13.1 Risultati Raggiunti

Il progetto FastFood implementa con successo tutti i quattro macro-scenari richiesti, fornendo una soluzione completa per la gestione di ordini online per ristoranti. Il sistema supporta efficacemente sia clienti che ristoratori con interfacce dedicate e funzionalità specifiche per ogni tipologia di utente.

L'architettura modulare e scalabile consente facili estensioni e manutenzioni future. La separazione chiara tra frontend e backend attraverso API REST ben documentate facilita l'integrazione con sistemi esterni e lo sviluppo di applicazioni mobile native utilizzando le stesse API.

### 13.2 Possibili Miglioramenti

Futuri sviluppi potrebbero includere l'implementazione di notifiche push per aggiornamenti in tempo reale sugli ordini, integrazione con sistemi di pagamento online per transazioni sicure, e funzionalità di rating e recensioni per migliorare la visibilità e ricerca dei ristoranti.

Altri miglioramenti potrebbero comprendere l'implementazione di un sistema di raccomandazioni basato sulle preferenze utente, supporto per ordini programmati, e integrazione con servizi di tracking GPS per consegne in tempo reale. Il sistema potrebbe anche beneficiare di funzionalità di analytics avanzate per aiutare i ristoratori a ottimizzare le loro operazioni basandosi sui dati di vendita e comportamento dei clienti.

La piattaforma FastFood rappresenta una base solida e completa per un sistema di food delivery moderno, implementando tutte le funzionalità richieste con un focus particolare su sicurezza, usabilità e scalabilità. L'architettura adottata garantisce che il sistema possa evolversi e crescere per soddisfare future esigenze di business mantenendo elevati standard di qualità e performance.

## A Codice Sorgente Completo

In questa appendice sono riportati i listati completi delle componenti principali del sistema.

### A.1 Modello User Completo

Listing 17: Modello User Completo

```

1 // models/User.js
2 import mongoose from 'mongoose';
3 import bcrypt from 'bcrypt';
4 import { USER_TYPES } from '../utils/constants.js';
5
6 const usernameRegex = /^[^\w{1,20}]/;
7 const nameRegex = /^[A-Za-z - ,.'-]{1,50}/u;
8 const emailRegex = /^[^@\s@]+@[^\s@]+\.[^\s@]{2,}/;
9 const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@#$%^&()_-+={}|\\;:<>?./]).{8,32}$/;
10
11 const userSchema = new mongoose.Schema({
12   username: {
13     type: String,
14     required: true,
15     unique: true,
16     minLength: 3,
17     maxLength: 20,
18     match: [usernameRegex, 'Username must be 3-20 characters long,',
19     'must contain only letters, numbers and underscores.']
20   },
21   firstName: {
22     type: String,
23     required: true,
24     minLength: 3,
25     maxLength: 50,
26     match: [nameRegex, 'First name must be 3-50 characters long and',
27     'contain only letters, spaces, apostrophes, or hyphens.']
28   },
29   image: {
30     type: String,
31     default: '/images/default-profile.png'
32   },
33   lastName: {
34     type: String,
35     required: true,
36     minLength: 3,
37     maxLength: 50,
38     match: [nameRegex, 'The last name must be 3-50 characters long,',
39     'must contain only letters, apostrophes, dots, dashes and spaces..']
40   },
41   email: {
42     type: String,
43     required: function () { return this.active === true },
44     unique: true,
45     sparse: true,
46     maxLength: 100,

```

```

44     match: [emailRegex, 'Invalid email address.']
45   },
46   password: {
47     type: String,
48     required: true,
49     minLength: 8,
50     maxLength: 100,
51   },
52   type: {
53     type: String,
54     required: true,
55     enum: Object.values(USER_TYPES)
56   },
57   active: {
58     type: Boolean,
59     default: true
60   }
61 })
62
63 userSchema.virtual('confirmPassword')
64   .get(function () {
65     return this._confirmPassword;
66   })
67   .set(function (value) {
68     this._confirmPassword = value;
69   });
70
71 userSchema.pre('validate', function (next) {
72   if (this.isNew && this.password !== this.confirmPassword) {
73     this.invalidate('confirmPassword', 'The passwords don\'t match
    .');
74   }
75   next();
76 });
77
78 userSchema.pre('save', async function (next) {
79   if (!this.isModified('password')) {
80     return next();
81   }
82
83   if (!passwordRegex.test(this.password)) {
84     return next(new Error('The password must be 8-32 characters long
      , must contain at least a lower case letter, an uppercase letter, a
      number and a special character.'));
85   }
86
87   const salt = await bcrypt.genSalt(10);
88   this.password = await bcrypt.hash(this.password, salt);
89   next();
90 });
91
92 const User = mongoose.model('User', userSchema);
93 export default User;

```

## A.2 Configurazione Server Express

Listing 18: Server.js Completo

```

1 // server.js
2 import express from 'express';
3 import cookieParser from 'cookie-parser';
4 import mongoSanitize from 'express-mongo-sanitize';
5 import helmet from 'helmet';
6 import xss from 'xss-clean';
7 import rateLimit from 'express-rate-limit';
8 import hpp from 'hpp';
9 import cors from 'cors';
10 import path from 'path';
11 import dotenv from 'dotenv';
12 import connectDB from './config/db.js';
13 import authRoutes from './routes/authRoutes.js';
14 import pagesRoutes from './routes/pagesRoutes.js';
15 import apiRoutes from './routes/apiRoutes.js';
16 import swaggerUi from 'swagger-ui-express';
17 import swaggerSpec from './config/swagger.js';
18 import sanitizeMiddleware from './middleware/sanitize.js';
19 import compression from 'compression';
20 import morgan from 'morgan';

21
22 import { importMeals } from './utils/mealsImporter.js';
23
24 dotenv.config();
25
26 const app = express();
27
28 app.set('trust proxy', 2);
29
30 const startServer = async () => {
31     try {
32         await connectDB();
33         await importMeals();

34         const __dirname = path.resolve();

35         app.use(compression());

36         if (process.env.NODE_ENV === 'development') {
37             app.use(morgan('dev'));
38         } else {
39             app.use(morgan('combined'));
40         }

41         app.use(
42             helmet({
43                 contentSecurityPolicy: {
44                     directives: {
45                         ...helmet.contentSecurityPolicy.
46                         getDefaultDirectives(),
47                         "img-src": ["'self'", "data:", "https://www.
48                         themealdb.com"],
49                         "connect-src": ["'self'"],
50                         },
51                         },
52                         },
53                         })
54

```

```

55     );
56
57     app.use(express.json());
58     app.use(cookieParser());
59
60     app.use((req, res, next) => {
61       if (req.body) {
62         req.body = mongoSanitize.sanitize(req.body);
63       }
64       next();
65     });
66
67     app.use(hpp());
68
69     const allowedOrigins = process.env.ALLOWED_ORIGINS
70     ? process.env.ALLOWED_ORIGINS.split(',').map(o => o.trim())
71     : [
72       'http://localhost:5000',
73       'http://127.0.0.1:5000',
74       'http://localhost:3000'
75     ];
76
77     if (process.env.ALLOWED_ORIGINS) {
78       const envOrigins = process.env.ALLOWED_ORIGINS.split(',');
79       allowedOrigins.push(...envOrigins.map(origin => origin.trim()));
80     }
81
82     app.use(cors({
83       origin: function (origin, callback) {
84         if (!origin) return callback(null, true);
85         if (allowedOrigins.indexOf(origin) === -1) {
86           const msg = 'The CORS policy for this site does not
allow access from the specified Origin.';
87           return callback(new Error(msg), false);
88         }
89         return callback(null, true);
90       },
91       credentials: true
92     }));
93
94     app.use(sanitizeMiddleware);
95
96     app.get('/', (req, res) => {
97       if (req.cookies.token) {
98         res.redirect('/home');
99       } else {
100         res.sendFile(path.join(__dirname, 'frontend/public/
landing.html'));
101       }
102     });
103
104     app.use(express.static(path.join(__dirname, 'frontend/public')));
105   ;
106   app.use('/js', express.static(path.join(__dirname, 'frontend/js')));
107   app.use('/css', express.static(path.join(__dirname, 'frontend/
css'))));

```

```
107     app.use('/images', express.static(path.join(__dirname, 'frontend/public/images')));
108     app.use('/bootstrap', express.static(path.join(__dirname, 'node_modules/bootstrap/dist')));
109
110     const authLimiter = rateLimit({
111         windowMs: 15 * 60 * 1000,
112         max: process.env.NODE_ENV === 'production' ? 20 : 100,
113         message: { error: 'Too many attempts, please try again later' }
114     });
115
116     const apiLimiter = rateLimit({
117         windowMs: 15 * 60 * 1000,
118         max: 200,
119         standardHeaders: true,
120         legacyHeaders: false,
121         message: { error: 'Too many requests from this IP, please try again after 15 minutes' }
122     });
123
124     app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerSpec));
125
126     app.use('/auth', authLimiter, authRoutes);
127     app.use('/', pagesRoutes);
128     app.use('/api', apiLimiter, apiRoutes);
129
130     app.use((err, req, res, next) => {
131         if (process.env.NODE_ENV !== 'test') {
132             console.error(err);
133         }
134
135         if (err.name === 'ValidationError') {
136             const messages = Object.values(err.errors).map(e => e.message);
137             return res.status(400).json({ error: messages });
138         }
139
140         if (err.code === 11000) {
141             const field = Object.keys(err.keyPattern)[0];
142             let message = `The ${field} is already in use.`;
143             if (field === 'owner') message = 'User already has a restaurant.';
144             return res.status(409).json({ error: message });
145         }
146
147         if (err.statusCode) {
148             return res.status(err.statusCode).json({ error: err.message });
149         }
150
151         if (err.name === 'MulterError') {
152             return res.status(400).json({ error: err.message });
153         }
154
155         if (err.message && err.message.includes('password') ||
156             err.message && err.message.includes('Password')) {
```

```

157         return res.status(400).json({ error: err.message });
158     }
159
160     res.status(500).json({ error: 'Server Error' });
161 });
162
163     return app;
164
165 } catch (error) {
166     console.error('Failed to start server:', error);
167     process.exit(1);
168 }
169 };
170
171 const requiredEnvVars = ['MONGO_URI', 'JWT_SECRET'];
172 for (const envVar of requiredEnvVars) {
173     if (!process.env[envVar]) {
174         console.error(`Missing required environment variable: ${envVar}`);
175         process.exit(1);
176     }
177 }
178
179 const server = await startServer().then(app => {
180     const PORT = process.env.PORT || 5000;
181     const runningServer = app.listen(PORT, () => {
182         console.log(`Server running on port ${PORT}`);
183     });
184
185     process.on('unhandledRejection', (err, promise) => {
186         console.log(`Error: ${err.message}`);
187         runningServer.close(() => process.exit(1));
188     });
189
190     return runningServer;
191 });
192
193 export default server;

```

### A.3 Auth Controller Completo

Listing 19: Auth Controller Completo

```

1 // controllers/authController.js
2 import User from '../models/User.js';
3 import CustomerData from '../models/CustomerData.js';
4 import Restaurant from '../models/Restaurant.js';
5 import bcrypt from 'bcrypt';
6 import jwt from 'jsonwebtoken';
7
8 export const registerUser = async (req, res, next) => {
9     try {
10         const { username, firstName, lastName, email, password,
11             confirmPassword, type } = req.body;
12
13         if (password !== confirmPassword) {

```

```

13         return res.status(400).json({ error: 'Passwords do not match
14     .'});
15 }
16
17     const newUser = new User({
18         username,
19         firstName,
20         lastName,
21         email,
22         password,
23         confirmPassword,
24         type,
25         image: req.file ? '/images/uploads/${req.file.filename}' :
26         undefined
27     });
28
29     await newUser.save();
30
31     res.status(201).json({ message: 'User successfully registered!' });
32 } catch (err) {
33     next(err);
34 }
35
36 export const loginUser = async (req, res, next) => {
37     try {
38         const { username, password } = req.body;
39         const user = await User.findOne({ username }, '_id username type
40         password');
41         if (!user) return res.status(404).json({ error: 'User not found.' });
42
43         const isMatch = await bcrypt.compare(password, user.password);
44         if (!isMatch) return res.status(400).json({ error: 'Wrong
45         Password.' });
46
47         let setupComplete = false;
48         let extraData = null;
49
50         if (user.type === 'customer') {
51             const customerData = await CustomerData.findOne({ user: user
52             _id });
53             if (customerData) {
54                 setupComplete = true;
55                 extraData = customerData._id;
56             }
57         } else {
58             const restaurant = await Restaurant.findOne({ owner: user.
59             _id });
60             if (restaurant) {
61                 setupComplete = true;
62                 extraData = restaurant._id;
63             }
64         }
65
66         const token = jwt.sign({
67             userId: user._id,
68             type: user.type,
69             extraData: extraData
70         });
71
72         res.cookie('token', token, { maxAge: 1000 * 60 * 60 * 24, httpOnly:
73         true });
74     }
75 }
76
77 module.exports = {
78     registerUser,
79     loginUser
80 }

```

```
        type: user.type,
        setupComplete
    }, process.env.JWT_SECRET, { expiresIn: '1h' });

res.cookie('token', token, {
    httpOnly: true,
    secure: process.env.NODE_ENV === 'production',
    sameSite: 'strict',
    maxAge: 1 * 60 * 60 * 1000
});

res.json({
    message: 'User logged in successfully.',
    username: user.username,
    userId: user._id,
    extraData: extraData
});

} catch (err) {
    next(err);
}

}

export const logoutUser = async (req, res) => {
    res.clearCookie('token');
    res.status(200).json({ message: 'User logged out successfully.' });
}

export const checkLogin = async (req, res) => {
    const { token } = req.cookies || {};
    if (!token) return res.status(200).json({ ok: false });
    try {
        jwt.verify(token, process.env.JWT_SECRET);
        return res.status(200).json({ ok: true });
    } catch {
        return res.status(200).json({ ok: false });
    }
}
```

## A.4 API Client Frontend

Listing 20: API Client Completo

```
1 // frontend/js/api.js
2 import { addMessage } from './errorManager.js';
3 import { setLoadingState } from './uiUtils.js';
4 import { config } from './config.js';
5
6 const PUBLIC_PAGES = ['/login', '/register', '/landing.html', '/'];
7
8 export async function fetchApi(url, options = {}, button = null) {
9     if (button) setLoadingState(button, true);
10
11     try {
12         if (!navigator.onLine) {
13             addMessage('No internet connection. Please check your
network.');
```

```

14         return null;
15     }
16
17     const defaultHeaders = { 'Content-Type': 'application/json' };
18
19     if (options.body instanceof FormData) {
20         delete defaultHeaders['Content-Type'];
21     }
22
23     const fetchConfig = {
24         credentials: 'include',
25         cache: 'no-cache',
26         ...options,
27         headers: {
28             ...defaultHeaders,
29             ...options.headers,
30         },
31     };
32
33     const res = await fetch(config.API_BASE_URL + url, fetchConfig);
34
35     if (res.status === 401) {
36         const currentPath = window.location.pathname;
37         const isPublicPage = PUBLIC_PAGES.some(p => currentPath ===
38             p || currentPath.endsWith(p));
39
40         if (!isPublicPage) {
41             addMessage('Your session has expired. Please log in
again.');
42             setTimeout(() => {
43                 window.location.href = '/login';
44             }, 1500);
45             return null;
46         }
47
48         if (res.status === 403) {
49             addMessage('You do not have permission to perform this
action.');
50             return null;
51         }
52
53         if (res.status === 404) {
54             const data = await res.json().catch(() => ({}));
55             addMessage(data.error || 'The requested resource was not
found.');
56             return null;
57         }
58
59         if (res.status === 429) {
60             addMessage('Too many requests. Please wait a moment and try
again.');
61             return null;
62         }
63
64         if (res.status >= 500) {
65             addMessage('Server error. Please try again later.');
66             return null;

```

```

67     }
68
69     const contentType = res.headers.get('content-type') || '';
70     let data;
71     if (contentType.includes('application/json')) {
72         data = await res.json();
73     } else {
74         const text = await res.text();
75         console.warn('Non-JSON response from', url, 'status:', res.
76 status, 'body:', text);
77         addMessage('Unexpected server response format.');
78         return null;
79     }
80
81     if (!res.ok) {
82         const errorMessage = Array.isArray(data.error)
83             ? data.error.join('<br>')
84             : (data.error || data.message || 'An unknown error
85 occurred.');
86         addMessage(errorMessage);
87         return null;
88     }
89
90     return data;
91 } catch (err) {
92     console.error('API Fetch Error:', err);
93
94     if (err.name === 'TypeError' && err.message.includes('Failed to
95 fetch')) {
96         addMessage('Unable to connect to server. Please check your
97 connection.');
98     } else if (err.name === 'AbortError') {
99         addMessage('Request was cancelled.');
100    } else {
101        addMessage('A network or server error occurred. Please try
102 again.');
103    }
104
105    return null;
106 } finally {
107     if (button) setLoadingState(button, false);
108 }
109 }
```

## A.5 Test Integrazione Auth

Listing 21: Test Integrazione Completo

```

1 // __tests__/auth.integration.test.js
2 import request from 'supertest';
3 import mongoose from 'mongoose';
4 import { jest } from '@jest/globals';
5 import { MongoMemoryServer } from 'mongodb-memory-server';
6 import path from 'path';
7
8 // Mock dependencies
9 jest.unstable_mockModule('../utils/mealsImporter.js', () => ({
```

```
10     importMeals: jest.fn(),
11 });
12
13 // Dynamic import for app to ensure mocks are applied
14 const { default: app } = await import('../server.js');
15
16 let mongoServer;
17
18 beforeEach(async () => {
19     mongoServer = await MongoMemoryServer.create();
20     const uri = mongoServer.getUri();
21     await mongoose.disconnect(); // Disconnect any existing connection
22     await mongoose.connect(uri);
23 });
24
25 afterEach(async () => {
26     await mongoose.disconnect();
27     await mongoServer.stop();
28 });
29
30 describe('Auth Integration Tests', () => {
31     let userCookie;
32
33     it('should register a new user', async () => {
34         const res = await request(app)
35             .post('/auth/register')
36             .field('username', 'testuser')
37             .field('firstName', 'Test')
38             .field('lastName', 'User')
39             .field('email', 'test@example.com')
40             .field('password', 'Password123!')
41             .field('confirmPassword', 'Password123!')
42             .field('type', 'customer')
43             .attach('image', Buffer.from('fakeimage'), 'test.jpg'); // Mock image upload
44
45         expect(res.statusCode).toBe(201);
46         expect(res.body).toHaveProperty('message', 'User successfully registered!');
47     });
48
49     it('should login with valid credentials', async () => {
50         const res = await request(app)
51             .post('/auth/login')
52             .send({
53                 username: 'testuser',
54                 password: 'Password123!'
55             });
56
57         expect(res.statusCode).toBe(200);
58         expect(res.body).toHaveProperty('message', 'User logged in successfully.');
59         expect(res.headers['set-cookie']).toBeDefined();
60
61         userCookie = res.headers['set-cookie'];
62     });
63
64     it('should not login with invalid password', async () => {
```

```

65     const res = await request(app)
66       .post('/auth/login')
67       .send({
68         username: 'testuser',
69         password: 'WrongPassword!'
70       });
71
72     expect(res.statusCode).toBe(400);
73     expect(res.body).toHaveProperty('error', 'Wrong Password.');
74   });
75
76   it('should logout successfully', async () => {
77     const res = await request(app)
78       .get('/auth/logout')
79       .set('Cookie', userCookie);
80
81     expect(res.statusCode).toBe(200);
82     expect(res.body).toHaveProperty('message', 'User logged out
successfully.');
83   });
84 });

```

## A.6 Cart Manager

Listing 22: Cart Manager Completo

```

1 // frontend/js/cartManager.js
2 export function getCart() {
3   return JSON.parse(localStorage.getItem('cart')) || {};
4 }
5
6 function saveCart(cart) {
7   localStorage.setItem('cart', JSON.stringify(cart));
8   showCartButton();
9 }
10
11 export function addToCart(item, amount) {
12   const cart = getCart();
13
14   const restaurantId = item.restaurant || item.restaurantId;
15   const dishId = typeof item.dish === 'string' ? item.dish : item.dish
?._id;
16
17   if (!restaurantId || !dishId) {
18     console.warn('addToCart: missing restaurant/dish id', { item });
19     return false;
20   }
21
22   const existingRestaurants = [...new Set(Object.values(cart).map(i =>
i.restaurant))];
23   if (existingRestaurants.length > 0 && !existingRestaurants.includes(
restaurantId)) {
24     const confirmed = confirm(
25       'Your cart contains items from another restaurant. ' +
26       'Do you want to clear the cart and add this item?'
27     );

```

```

28     if (!confirmed) {
29         return false;
30     }
31     localStorage.removeItem('cart');
32 }
33
34 const currentCart = getCart();
35 const key = `${restaurantId}:${dishId}`;
36
37 const qty = parseInt(amount, 10);
38 if (qty > 0) {
39     currentCart[key] = {
40         restaurant: restaurantId,
41         dish: dishId,
42         amount: qty
43     };
44 } else {
45     delete currentCart[key];
46 }
47
48 saveCart(currentCart);
49 return true;
50 }
51
52 export function showCartButton() {
53     const cart = getCart();
54     const cartBtn = document.getElementById('cart-btn');
55     if (cartBtn) {
56         cartBtn.hidden = Object.keys(cart).length === 0;
57     }
58 }
```

## A.7 Restaurant Controller

Listing 23: Restaurant Controller Completo

```

1 // controllers/restaurantController.js
2 import Restaurant from '../models/Restaurant';
3 import CustomerData from '../models/CustomerData';
4 import Order from '../models/Order';
5 import jwt from 'jsonwebtoken';
6
7 export const addRestaurant = async (req, res, next) => {
8     try {
9         const user = req.user;
10        const restaurant = req.body.restaurant;
11
12        const newRestaurant = new Restaurant({
13            owner: user.userId,
14            name: restaurant.name,
15            address: restaurant.address,
16            vatNumber: restaurant.vatNumber,
17            phoneNumber: restaurant.phoneNumber,
18            image: req.file ? '/images/uploads/${req.file.filename}' :
19                undefined
20        });
21    }
22    catch (err) {
23        next(err);
24    }
25}
```

```

20     await newRestaurant.save();
21
22     const token = jwt.sign(
23         { userId: user.userId, type: user.type, setupComplete: true
24     },
25         process.env.JWT_SECRET,
26         { expiresIn: '1h' }
27     );
28     res.cookie('token', token, {
29         httpOnly: true,
30         secure: process.env.NODE_ENV === 'production',
31         sameSite: 'strict',
32         maxAge: 60 * 60 * 1000
33     });
34
35     res.status(201).json({ message: 'Restaurant created successfully
36 .', restaurant: newRestaurant._id });
37 } catch (err) {
38     next(err);
39 }
40
41 export const getNearby = async (req, res, next) => {
42     try {
43         const user = req.user;
44
45         const userAddress = await CustomerData.findOne({ user: user.
46 userId });
46         const nearbyRestaurants = await Restaurant.find({ 'address.
47 zipCode': userAddress.address.zipCode, active: true });
48
49         res.set('Cache-Control', 'no-store');
50         res.json({ nearbyRestaurants: nearbyRestaurants });
51     } catch (err) {
52         next(err);
53     }
54 }
55
55 export const getMenu = async (req, res, next) => {
56     try {
57         const restaurantId = req.params.id;
58
59         const restaurant = await Restaurant.findById(restaurantId).
60 populate('menu.dish');
60         if (!restaurant) return res.status(404).json({ error: '
61 Restaurant not found.' });
62
62         res.json(restaurant);
63     } catch (err) {
64         next(err);
65     }
66 }
67
68 export const editMenu = async (req, res, next) => {
69     try {
70         const { restaurant, newMenu } = req.body;
71

```

```

72     await Restaurant.findByIdAndUpdate(restaurant, { menu: newMenu
73   });
74   res.json({ message: 'Menu updated successfully.' });
75 } catch (err) {
76   next(err);
77 }
78
79 export const editRestaurant = async (req, res, next) => {
80   try {
81     const user = req.user;
82     const body = req.body;
83
84     const restaurant = await Restaurant.findOne({ owner: user.userId
85   });
85     if (!restaurant) return res.status(404).json({ error: 'Restaurant not found.' });
86
87     const getValue = (bracketKey, nestedPath) => {
88       if (body[bracketKey] !== undefined) return body[bracketKey];
89
90       if (nestedPath && body.newRestaurant) {
91         const keys = nestedPath.split('.');
92         let val = body.newRestaurant;
93         for (const k of keys) {
94           if (val && val[k] !== undefined) val = val[k];
95           else return undefined;
96         }
97         return val;
98       }
99       return undefined;
100     };
101
102     const newName = getValue('newRestaurant[name]', 'name');
103     if (newName) restaurant.name = newName;
104
105     const streetAddress = getValue('newRestaurant[address][
106 streetAddress]', 'address.streetAddress');
106     const city = getValue('newRestaurant[address][city]', 'address.
107 city');
107     const province = getValue('newRestaurant[address][province]', 'address.
108 province');
108     const zipCode = getValue('newRestaurant[address][zipCode]', 'address.
109 zipCode');
110
110     if (streetAddress) restaurant.address.streetAddress =
111       streetAddress;
111     if (city) restaurant.address.city = city;
112     if (province) restaurant.address.province = province;
113     if (zipCode) restaurant.address.zipCode = zipCode;
114
115     const phone = getValue('newRestaurant[phoneNumber]', 'phone
116 number');
116     if (phone != null) restaurant.phoneNumber = phone;
117
118     const vat = getValue('newRestaurant[vatNumber]', 'vatNumber');
119     if (vat) restaurant.vatNumber = vat;
120

```

```

121     if (body.active !== undefined) restaurant.active = body.active;
122     if (body.newRestaurant?.active !== undefined) restaurant.active
123 = body.newRestaurant.active;
124
125     if (req.file) {
126         restaurant.image = '/images/uploads/${req.file.filename}';
127     }
128
129     await restaurant.save();
130     res.json({ message: 'Restaurant updated successfully.' });
131 } catch (err) {
132     next(err);
133 }
134
135 export const getAnalytics = async (req, res, next) => {
136     try {
137         const user = req.user;
138         const { start, end } = req.query;
139
140         const restaurant = await Restaurant.findOne({ owner: user.userId
141 });
141         if (!restaurant) return res.status(404).json({ error: 'Restaurant
142 not found.' });
143
144         const toDate = (v, fallback) => {
145             if (!v) return fallback;
146
147             const num = Number(v);
148             if (Number.isFinite(num)) {
149                 return new Date(num > 1e12 ? num : num * 1000);
150             }
151
152             try {
153                 const d = new Date(v);
154                 if (isNaN(d.getTime())) {
155                     console.warn('Invalid date format: ${v}');
156                     return fallback;
157                 }
158                 return d;
159             } catch (error) {
160                 console.warn('Date parsing error for value ${v}:', error
161 );
162                 return fallback;
163             }
164
165         const startDate = toDate(start, new Date(0));
166         const endDate = toDate(end, new Date());
167
168         if (startDate > endDate) {
169             return res.status(400).json({ error: 'Start date cannot be
170 after end date.' });
171
172         const matchStage = {
173             restaurant: restaurant._id,
174             createdAt: { $gte: startDate, $lte: endDate }

```

```

174     };
175
176     const totalStatsPipeline = [
177       { $match: matchStage },
178       {
179         $group: {
180           _id: null,
181           totalOrders: { $sum: 1 },
182           totalEarned: { $sum: '$price' }
183         }
184       }
185     ];
186
187     const mostOrderedPipeline = [
188       { $match: matchStage },
189       {
190         $group: {
191           _id: '$dish',
192           totalAmount: { $sum: '$amount' },
193           totalEarned: { $sum: '$price' }
194         }
195       },
196       { $sort: { totalAmount: -1 } },
197       { $limit: 1 },
198       {
199         $lookup: {
200           from: 'dishes',
201           localField: '_id',
202           foreignField: '_id',
203           as: 'dishData'
204         }
205       },
206       { $unwind: { path: '$dishData', preserveNullAndEmptyArrays:
true } }
207     ];
208
209     const [totalResult, mostOrderedResult] = await Promise.all([
210       Order.aggregate(totalStatsPipeline),
211       Order.aggregate(mostOrderedPipeline)
212     ]);
213
214     const totals = totalResult[0] || { totalOrders: 0, totalEarned:
0 };
215     const mostOrdered = mostOrderedResult[0] || null;
216
217     res.json({
218       totalOrders: totals.totalOrders,
219       totalEarned: totals.totalEarned,
220       avgEarned: totals.totalOrders > 0 ? totals.totalEarned /
totals.totalOrders : 0,
221       mostOrdered: mostOrdered
222     });
223
224   } catch (err) {
225     next(err);
226   }
227 }
228

```

```

229 export const searchRestaurants = async (req, res, next) => {
230   try {
231     const { name, street, city, page = 1 } = req.query;
232     const limit = 20;
233     const skip = (page - 1) * limit;
234
235     const filter = {};
236     if (name) filter.name = { $regex: name, $options: 'i' };
237     if (street) filter['address.streetAddress'] = { $regex: street,
238       $options: 'i' };
239     if (city) filter['address.city'] = { $regex: city, $options: 'i' };
240   };
241   filter.active = true;
242
243   const total = await Restaurant.countDocuments(filter);
244   const restaurants = await Restaurant.find(filter).skip(skip).
245   limit(limit);
246
247   res.set('Cache-Control', 'no-store');
248   res.json({ total, restaurants });
249 } catch (err) {
250   next(err);
251 }
252 }
```

## A.8 Order Controller

Listing 24: Order Controller Completo

```

1 // controllers/orderController.js
2 import mongoose from 'mongoose';
3 import Order from '../models/Order.js';
4 import Restaurant from '../models/Restaurant.js';
5 import { ORDER_STATES } from '../utils/constants.js';
6
7 export const getOrders = async (req, res, next) => {
8   try {
9     const user = req.user;
10    const page = req.query.page || 1;
11    const limit = 10;
12    const skip = (page - 1) * limit;
13
14    let filter = {};
15    if (user.type === 'customer') filter.customer = user.userId;
16    else {
17      const restaurant = await Restaurant.findOne({ owner: user.
18 userId });
19      if (!restaurant) return res.json({ total: 0, orders: [] });
20      filter.restaurant = restaurant._id;
21    }
22
23    const total = await Order.countDocuments(filter);
24    const orders = await Order.find(filter).populate('dish').
25    populate('restaurant').populate('customer').sort({ createdAt: -1 }).
26    skip(skip).limit(limit);
27
28  }
29
30  res.json({ total, orders });
31}
```

```

25     console.log(`getOrders DEBUG: Found ${total} orders (page
26     returns ${orders.length})`);
27     if (orders.length > 0) {
28       console.log(`getOrders DEBUG First Order Sample:`, JSON.
29     stringify(orders[0]));
30   }
31
32   const cleanOrders = orders.map(order => {
33     return {
34       _id: order._id,
35       dish: order.dish,
36       restaurant: order.restaurant,
37       customer: order.customer,
38       amount: order.amount,
39       price: order.price,
40       state: order.state,
41       createdAt: order.createdAt
42     }
43   });
44
45   res.json({ total, orders: cleanOrders });
46 } catch (err) {
47   next(err);
48 }
49
50 export const newOrder = async (req, res, next) => {
51   let session = null;
52   if (process.env.SKIP_TRANSACTIONS !== 'true') {
53     session = await mongoose.startSession();
54     session.startTransaction();
55   }
56   try {
57     const user = req.user;
58     for (const key in req.body) {
59       const order = req.body[key];
60
61       const restaurant = await Restaurant.findById(order.
62         restaurant);
63       if (!restaurant) {
64         throw new Error(`Restaurant not found: ${order.
65         restaurant}`);
66
67       const menuItem = restaurant.menu.find(
68         item => item.dish.toString() === order.dish
69     );
69       if (!menuItem) {
70         throw new Error(`Dish ${order.dish} not found in
71         restaurant menu`);
72     }
73
74       const serverCalculatedPrice = menuItem.price * order.amount;
75
76       const newOrder = new Order({
77         customer: user.userId,
78         restaurant: order.restaurant,
79       });
80     }
81   }
82
83   res.status(201).json(newOrder);
84 }
85
86 module.exports = router;

```

```

78         dish: order.dish,
79         amount: order.amount,
80         price: serverCalculatedPrice,
81         state: 'received',
82         createdAt: Date.now()
83     );
84
85     const saveOptions = session ? { session } : {};
86     await newOrder.save(saveOptions);
87
88     const updateOptions = session ? { session } : {};
89     await Restaurant.findByIdAndUpdate(order.restaurant, { $push
90 : { queue: newOrder._id } }, updateOptions);
91
92     if (session) {
93         await session.commitTransaction();
94     }
95     res.status(201).json({ message: 'Order received.' });
96 } catch (err) {
97     if (session) {
98         await session.abortTransaction();
99     }
100    next(err);
101} finally {
102    if (session) {
103        session.endSession();
104    }
105}
106}
107
108 export const confirmPickup = async (req, res, next) => {
109     try {
110         const user = req.user;
111         const id = req.params.id;
112
113         const order = await Order.findOne({ _id: id, customer: user.
114 userId });
115         if (!order) return res.status(404).json({ error: 'Order not
116 found.' });
117         if (order.state !== 'ready') return res.status(400).json({ error
118 : 'Order not ready for pickup.' });
119
120         order.state = 'completed';
121         await order.save();
122
123         res.json({ message: 'Order completed.' });
124     } catch (err) {
125         next(err);
126     }
127 }
128
129 export const getQueue = async (req, res, next) => {
130     try {
131         const user = req.user;
132         const restaurant = await Restaurant.findOne({ owner: user.userId
133 }).select('_id');
134         if (!restaurant) return res.status(404).json({ message: 'Restaurant
135 not found.' });
136
137         const queue = restaurant.queue;
138         const orders = queue.map(order => {
139             const dish = order.dish;
140             const amount = order.amount;
141             const price = calculatePrice(dish, amount);
142             const state = order.state;
143             const createdAt = order.createdAt;
144
145             return { dish, amount, price, state, createdAt };
146         });
147
148         res.json({ queue: orders });
149     } catch (err) {
150         next(err);
151     }
152 }
153
154
```

```

    Restaurant not found.' });

131     const queue = await Order.aggregate([
132       { $match: { restaurant: restaurant._id, state: { $in: [
133         ORDER_STATES.RECEIVED, ORDER_STATES.PREPARING] } } },
134       { $sort: { createdAt: 1 } },
135       { $lookup: { from: 'users', localField: 'customer',
136         foreignField: '_id', as: 'customer' } },
137       { $unwind: '$customer' },
138       { $lookup: { from: 'dishes', localField: 'dish',
139         foreignField: '_id', as: 'dish' } },
140       { $unwind: '$dish' },
141       { $lookup: { from: 'restaurants', localField: 'restaurant',
142         foreignField: '_id', as: 'restaurantDoc' } },
143       { $unwind: '$restaurantDoc' },
144       {
145         $project: {
146           _id: 1,
147           customer: { name: { $concat: ['$customer.firstName',
148             ' ', '$customer.lastName'] } },
149           dish: { name: '$dish.name', image: '$dish.image' },
150           amount: 1,
151           price: 1,
152           state: 1,
153           prepTime: {
154             $let: {
155               vars: {
156                 menuItem: {
157                   $arrayElemAt: [
158                     { $filter: { input: '$restaurantDoc.menu',
159                       as: 'item', cond: { $eq: ['$item.dish', '$dish._id'] } } }, 0
160                     ]
161                   }
162                 }
163               }
164             }
165           }
166         }
167       ]
168     );
169   }

170   res.json({ queue: queue });
171 } catch (err) {
172   next(err);
173 }
174 }

175 }

176 const restaurant = await Restaurant.findOne({ owner: user.userId
177 }).select('_id queue');
178   if (!restaurant) return res.status(404).json({ error: 'Restaurant not found.' });
179

```

```
179     const targetOrderId = orderId || restaurant.queue[0];
180     if (!targetOrderId) return res.json({ message: 'Queue is empty.' });
181   }
182
183     const orderToAdvance = await Order.findById(targetOrderId);
184     if (!orderToAdvance || String(orderToAdvance.restaurant) !==
185       String(restaurant._id)) {
186       return res.status(404).json({ error: 'Order not found in
187         this restaurant.' });
188     }
189
190     const isHeadOfQueue = restaurant.queue[0]?.toString() ===
191       targetOrderId.toString();
192
193     if (orderToAdvance.state === ORDER_STATES.RECEIVED) {
194       const updated = await Order.findOneAndUpdate(
195         { _id: targetOrderId, state: ORDER_STATES.RECEIVED },
196         { $set: { state: ORDER_STATES.PREPARING } },
197         { new: true }
198       );
199       if (!updated) return res.status(409).json({ error: 'Order
200         state changed concurrently.' });
201
202       if (isHeadOfQueue) {
203         await Restaurant.findByIdAndUpdate(restaurant._id, {
204           lastPreparationStart: Date.now()
205         })
206       } else if (orderToAdvance.state === ORDER_STATES.PREPARING) {
207         const updated = await Order.findOneAndUpdate(
208           { _id: targetOrderId, state: ORDER_STATES.PREPARING },
209           { $set: { state: ORDER_STATES.READY } },
210           { new: true }
211         );
212         if (!updated) return res.status(409).json({ error: 'Order
213         state changed concurrently.' });
214
215         await Restaurant.findByIdAndUpdate(restaurant._id, { $pull:
216           { queue: targetOrderId } });
217
218         res.json({ message: 'Queue advanced successfully.' });
219     } catch (err) {
220       next(err);
221     }
222   }
223
224   export const waitEstimation = async (req, res, next) => {
225     try {
226       const orderId = req.params.id;
227       console.log(`waitEstimation DEBUG: Start for OrderID=${orderId}`);
228
229       const order = await Order.findById(orderId);
230       if (!order) {
231         console.log(`waitEstimation DEBUG: Order not found ${orderId}`);
232       }
233       return res.status(404).json({ error: 'Order not found.' });
234     }
235   }
236 }
```

```

227
228     const restaurant = await Restaurant.findById(order.restaurant);
229     if (!restaurant) {
230         console.log(`waitEstimation DEBUG: Restaurant not found for
231 order ${orderId}`);
232         return res.status(404).json({ error: 'Restaurant not found.' });
233     }
234     console.log(`waitEstimation DEBUG: Restaurant found: ${
235 restaurant._id}, Queue: ${restaurant.queue.length}`);
236
237     const queuePosition = restaurant.queue.findIndex(id => id.
238 toString() === orderId);
239     if (queuePosition === -1) {
240         return res.json({ time: 0 });
241     }
242
243     const orderIdsAhead = restaurant.queue.slice(0, queuePosition +
244 1);
245
246     const result = await Order.aggregate([
247         { $match: { _id: { $in: orderIdsAhead } } },
248         {
249             $lookup: {
250                 from: 'restaurants',
251                 let: { restaurantId: '$restaurant', dishId: '$dish' },
252                 pipeline: [
253                     { $match: { $expr: { $eq: ['$id', '$$restaurantId'] } } },
254                     { $unwind: '$menu' },
255                     { $match: { $expr: { $eq: ['$menu.dish', '$$dishId'] } } },
256                     { $project: { prepTime: '$menu.preparationTime' } }
257                 ],
258                 as: 'menuData'
259             }
260         },
261         { $unwind: { path: '$menuData', preserveNullAndEmptyArrays:
262 true } },
263         {
264             $group: {
265                 _id: null,
266                 totalTime: { $sum: { $multiply: ['$amount', {
267 $ifNull: ['$menuData.prepTime', 0] }] } }
268             }
269         }
270     ]);
271
272     let waitingTime = result[0]?.totalTime || 0;
273
274     const firstOrderId = restaurant.queue[0];
275     if (firstOrderId) {
276         const firstOrder = await Order.findById(firstOrderId);
277         if (firstOrder?.state === ORDER_STATES.PREPARING &&
278 restaurant.lastPreparationStart) {
279             const elapsedMinutes = (Date.now() - new Date(restaurant

```

```
273     .lastPreparationStart).getTime() / (60 * 1000);
274         waitingTime = Math.max(waitingTime - elapsedMinutes, 0);
275     }
276
277     res.json({ time: Math.round(waitingTime * 10) / 10 });
278 } catch (err) {
279     next(err);
280 }
281 }
```

---

The screenshot displays the Swagger UI for the FastFood API. The interface is organized into several sections:

- User & Profile** (GET /api/profile, GET /api/cards, POST /api/finalize, POST /api/card/add, PUT /api/address/update, PUT /api/profile/update, DELETE /api/deactivate, DELETE /api/card/delete/{id})
- Orders** (GET /api/orders, GET /api/queue, GET /api/estimate/{id}, POST /api/order, PUT /api/confirmpickup/{id}, PUT /api/order/update)
- Restaurants** (GET /api/nearby, GET /api/menu/{id}, GET /api/restaurant/analytics, GET /api/restaurant/search, POST /api/restaurant/add, PUT /api/menu/update, PUT /api/restaurant/update)
- Dishes** (GET /api/dishes, GET /api/dishes/search, POST /api/dish/add)
- Auth** (POST /auth/register, POST /auth/login, GET /auth/logout, GET /auth/check)
- Schemas** (Address, Card, User, Dish, MenuItem, Restaurant, Order)

Figura 9: Interfaccia Swagger per Testing API