📄 Relazione Tecnica: Modulo 2 - Area Cliente e Ordinazioni

Autore: Alessandro Di Stasi (Matr. 358140)

Progetto: Food Delivery - Ingegneria del Software

---

1. Introduzione e Obiettivi (Approccio MVP)

Il Modulo 2 gestisce l'esperienza *core* dell'utente cliente all'interno della piattaforma.

Lo sviluppo ha seguito la filosofia **MVP (Minimum Viable Product)**: l'obiettivo è stato realizzare un prodotto essenziale ma perfettamente funzionante, capace di gestire l'intero ciclo di vita di un ordine (dalla scelta del ristorante al checkout).

Questa strategia ha permesso di concentrarsi sulla robustezza delle funzionalità critiche (Carrello, Transazioni, Integrità Dati), rilasciando un sistema stabile e pronto all'uso, ma progettato con un'architettura modulare pronta ad accogliere future evoluzioni (es. tracciamento rider o pagamenti elettronici).

Dal punto di vista tecnico, è stata adottata un'**architettura Client-Server a livelli (Layered Architecture)**. Il Frontend comunica con il Backend esclusivamente tramite **API RESTful JSON**, garantendo una netta separazione tra interfaccia e logica di business.

---

2. Matrice di Conformità (Requirements Traceability Matrix)

La seguente matrice dimostra la copertura dei requisiti funzionali del MVP assegnato al Modulo 2.

ID Req.	Descrizione Requisito	Componente Software (File)	Test di Riferimento
REQ-01	**Visualizzazione Ristoranti**	**FE:** Cliente.html	Test Funzionale (E2E) - Caricamento Home

	Lista esercenti con dettagli (nome, img, tag) e stato apertura.	**BE:** Get_All_Restaurants.php	
REQ-02	**Consultazione Menù** Selezione ristorante e visualizzazione piatti per categoria.	**FE:** Cliente_Presenter.js **BE:** Get_Menu.php	Test Funzionale (E2E) - Apertura Modale
REQ-03	**Gestione Carrello (Client-Side)** Aggiunta/rimozione piatti, calcolo live, vincolo unicità esercente.	**FE:** Cliente_Presenter.js (Logica localStorage)	Test Unità JS (Vincolo esercente unico)
REQ-04	**Creazione Ordine (Checkout)** Invio ordine, validazione prezzi (Security) e persistenza DB.	**BE:** Create_Order.php **BE:** Order_Factory.php	**Test Integrazione:** test_orders_runner.php

REQ-05	**Generazione Codice Ritiro** Ogni ordine confermato ha un codice univoco.	**BE:** Takeaway_Order.php (Metodo generatePickupCode)	Verifica su DB (Colonna codice_ritiro)

---

3. Architettura e Design Patterns

La logica di business è stata implementata puntando su estensibilità e incapsulamento.

- **Factory Method Pattern:** Utilizzato per la creazione degli ordini. La logica di istanziazione è centralizzata nella classe Order_Factory. Questo disaccoppia l'API dalle classi concrete, rispettando il principio *Open/Closed*: l'aggiunta di futuri tipi di ordine non richiederà modifiche al codice esistente.
- **Connection Provider:** La classe Database non è un Singleton statico, ma funge da provider che restituisce una connessione PDO pulita e isolata ad ogni richiesta.

---

4. Componenti Implementati

4.1 Livello di Presentazione (Frontend)

- **Cliente.html:** Interfaccia utente responsive (Bootstrap) per la griglia ristoranti e il carrello.
- **Cliente_Presenter.js:** Gestisce la logica applicativa lato client (Pattern Presenter). Si occupa di mantenere lo stato del carrello nel localStorage, impedire l'inserimento di piatti da ristoranti diversi e inviare il payload JSON.

4.2 Livello API (Services)

- **Get_All_Restaurants.php:** Endpoint GET che restituisce la lista esercenti.
- **Get_Menu.php:** Endpoint GET che restituisce il catalogo prodotti.
- **Create_Order.php:** Endpoint POST transazionale. Riceve il carrello, ricalcola i prezzi per sicurezza (evitando *Parameter Tampering*), invoca la Factory e salva l'ordine.

4.3 Livello di Dominio (Business Logic)

- **Order_Factory.php:** Implementazione del Creator (Factory Method).
- **Takeaway_Order.php:** Prodotto Concreto. Contiene la logica di business per gli ordini d'asporto: generazione codice ritiro e gestione della transazione SQL (INSERT in tabelle Ordini e Dettagli).

---

5. Modelli UML

5.1 Diagramma delle Classi (Static View)

Il diagramma evidenzia il **Factory Method Pattern**:

- **Order_Factory**: Classe statica che riceve i dati grezzi e restituisce un oggetto Takeaway_Order.
- **Takeaway_Order**: Incapsula le proprietà dell'ordine e il metodo process() per il salvataggio atomico su database.

5.2 Diagramma di Sequenza (Dynamic View)

Modella il flusso "Creazione Ordine":

1. **Frontend**: Invia POST /orders con il carrello JSON.
2. **API Service**: Valida il Token JWT e l'integrità del payload.
3. **Factory**: Istanzia l'oggetto ordine corretto.
4. **Database**: Esegue BEGIN TRANSACTION -> Inserimento Testata -> Inserimento Righe -> COMMIT.
5. **Response**: Ritorna HTTP 201 Created con il codice di ritiro.

---

6. Testing e Validazione

6.1 Test di Integrazione Backend

È stato predisposto uno script di collaudo (test_orders_runner.php) per verificare il corretto funzionamento della catena **API -> Factory -> Database**.

- **Oggetto del Test:** Endpoint Create_Order.php.
- **Risultato:** ✅ **SUPERATO**. L'ordine viene creato correttamente nel DB e il server risponde con il codice di ritiro generato.

6.2 Test di Sicurezza

È stato verificato che il sistema ricalcoli i prezzi lato server. Modificando manualmente il JSON inviato dal browser (es. mettendo prezzo 0€), il backend ha ignorato il dato falso e salvato l'ordine con il prezzo corretto prelevato dal database.