Prova tecnica Watuppa

2. Storia della documentazione

- Creato da: Anghel Stefan
- Approvato da: -
- Stato del documento: Completed
- Versione: 1.0
- Storico:
    - 2025-07-23: Completamento del progetto e della documentazione
    - 2025-07-21: Inizio redazione della documentazione
    - 2025-07-21: Avvio del progetto

3. Introduzione

La presente documentazione descrive le fasi di analisi, progettazione, sviluppo e consegna della prova tecnica per l’azienda "Watuppa", un progetto volto alla realizzazione di una parte frontend in Vue.js con dei componenti grafici, e un backend in PHP dedicata alle API.
L’obiettivo è dimostrare competenze tecniche full stack nella creazione di un'applicazione web moderna, intuitiva e integrata con un backend custom sviluppato in PHP.

4. Approccio al progetto

L’approccio seguito per la realizzazione del progetto è stato modulare e iterativo:

- Analisi: identificazione dei requisiti essenziali (frontend, backend, gestione ordini).
- Progettazione: definizione dell’architettura Vue 3 + PHP 8.2 + MySQL.
- Sviluppo: creazione di componenti UI dinamici, API REST, e strutture dati.
- Testing: verifica funzionalità e comunicazione tra frontend e backend.
- Documentazione: stesura della presente relazione tecnica.

5. Architettura del progetto

Frontend

- Framework: Vue 3
- Build Tool: Vite
- Linguaggio: TypeScript
- Struttura:
    - `index.html`: punto d'ingresso dell'app
    - `src/`: risiedono le logiche e i componenti
    - `src/components/`: risiedono i vari componenti utilizzati
    - `src/router/`: risiede il router dedicato alla navigazione fra le varie viste disponibili
    - `src/views/`: risiedono le varie viste disponibili

Backend

- Linguaggio: PHP
- Architettura: REST API
- Database: MySQL
- Struttura:
    - `config/`: risiedono i file dedicati alla gestione delle configurazioni, come la gestione dei cors o delle variabili d'ambiente
    - `database/`: risiedono il file per la gestione del database che avviene tramite la classe **MysqliDb**
    - `database/migrations`: file per la creazione delle tabelle nel database, chiamate 'migration'
    - `database/seeders/`: file per il popolamento delle tabelle nel database, chiamati 'seeder'
    - `public/`: cartella in cui viene effettuato l'accesso da fonti esterne, per la precisione nel file index.php
    - `scripts/`: risiedono i file per lanciare tutte le migration e tutti i seeder nell'ordine corretto
    - `web/`: risiede il file 'router.php' per la gestione delle chiamate api effettuate verso il backend, risiede anche la cartella 'api'.
    - `web/api/`: risiedono tutti i file dedicate alle api

6. Installazione e Avvio del Progetto

Configurazione delle variabili d'ambiente

Backend - File .env

Creare il file .env nella cartella backend con le seguenti variabili e impostare i valori coerenti con il proprio ambiente:

- **DB_HOST**=localhost
- **DB_USERNAME**=root
- **DB_PASSWORD**=password
- **DB_NAME**=php_vue
- **FRONTEND_URL**=http://localhost:5174

Frontend - File .env

Creare il file .env nella cartella frontend con le seguenti variabili e impostare i valori coerenti con il proprio ambiente:

- **VITE_BACKEND_URL**=http://localhost:8080/api

Frontend:

```
cd "frontend"
```

Installare le dipendenze:

```
npm install
```

Avviare il progetto Frontend:

```
npm run dev
```

Per buildare il progetto in ES5:

```
npm run build
```

Backend:

```
cd "backend"
```

Installare le dipendenze:

```
composer install
```

Inizializzare il Database :

note: Il **database deve esistere** ma **non deve contenere tabelle.** Se il **database** contiene già **tabelle** o è già stato **inizializzato, salta** questo **passaggio**

```
php scripts/run_migrations.php
```

```
php scripts/run_seeders.php
```

Avviare il backend :

```
php -S localhost:8080 -t public
```

7. API Backend

Endpoint: POST /api/filtered_orders

Request:


```
{
    "email": "user@example.com"
}
```


Response Success:


```
{
    "user": {
        "id": 1,
        "name": "Mario",
        "surname": "Rossi",
        "email": "mario.rossi@example.com"
    },
    "orders": [
        {
            "order_id": 1,
            "note": "Ordine urgente - consegnare entro venerd\u00ec",
            "created_at": "2025-07-19 17:42:21",
            "quantity": 1,
            "item_title": "iPhone 15 Pro",
            "price": "1199.00",
            "brand_name": "Apple",
            "category_name": "Elettronica"
        },
    ]
}
```

Response Error:


```
{
    "error": "Utente non trovato"
}
```


Endpoint: GET /api/products

Request: {}

Response Success:


```
{
        "products": [
                {
                    "id": 6,
                    "title": "Adidas Ultraboost 22",
                    "description": "Scarpe da running",
                    "price": "180.00",
                    "brand": "Adidas",
                    "category": "Sport e Fitness"
                }
        ]
},
```

8. Dettagli tecnici

build ES5 (legacy) per browser obsoleti

Durante la fase di build del progetto frontend, è stato configurato il plugin `@vitejs/plugin-legacy` all'interno del file `vite.config.ts`. Questo plugin ha lo scopo di **generare automaticamente una versione del bundle compatibile con browser datati**, come Internet Explorer 11.

```
legacy({

  targets: ['defaults', 'IE 11'],

  additionalLegacyPolyfills: ['regenerator-runtime/runtime']

})
```

Questa configurazione produce **due versioni del bundle**:

- una moderna (`index-*.js`) per browser recenti che supportano `ES modules` e `async/await`,
- una **legacy** (`index-legacy-*.js` e `polyfills-legacy-*.js`) in **ES5**, per browser più vecchi.

Inoltre, Vite inserisce automaticamente nell'`index.html` gli script `type="module"` e `nomodule` per garantire che **ogni browser carichi solo la versione corretta**.

Schema E-R