FIGURE 1 – Tableau de bord de l'application MAELED

École Marocaine des Sciences de l'Ingénieur
Année Universitaire 2025 – 2026

Projet Web JavaScript
Application Back-Office

MAELED
Système de gestion d'un restaurant

Réalisé par :
Lakrami Iyad
Lanaia Ayman
Maker Aymen

Option : 3IIR

Encadrant : Y. Belhaiba

Table des matières

Introduction

Dans le cadre du module **Développement Web JavaScript**, ce projet vise la réalisation d'une application **back-office complète** destinée à la gestion d'un restaurant.

L'application développée, nommée **MAELED**, permet de gérer efficacement les données essentielles telles que : les menus, les commandes, les réservations, le personnel et l'inventaire. Le projet a été réalisé exclusivement en **HTML5, CSS3 et JavaScript**, sans backend, en utilisant **localStorage** pour simuler une base de données persistante.

Chapitre 1

Contexte et Objectifs

Les applications back-office sont indispensables pour automatiser et structurer les processus internes d'une activité. Dans un restaurant, elles permettent notamment :

— le suivi des commandes clients,
— la gestion dynamique du menu,
— l'organisation du personnel,
— le contrôle des stocks,
— l'analyse des performances.

L'objectif pédagogique principal est de :

— maîtriser la manipulation du DOM,
— implémenter des opérations CRUD complètes,
— structurer une application JavaScript modulaire,
— proposer une interface claire et professionnelle.

Chapitre 2

Analyse Fonctionnelle

2.1 Besoins fonctionnels

L'application MAELED répond aux besoins suivants :

— gestion du menu (plats, prix, disponibilité),
— gestion des commandes avec sélection des articles,
— gestion des réservations avec contrôle des conflits,
— gestion du personnel,
— gestion de l'inventaire.

2.2 Flux de fonctionnement

Le flux général de l'application est le suivant :

$$\text{Connexion} \rightarrow \text{Dashboard} \rightarrow \text{Gestion CRUD} \rightarrow \text{Analyse}$$

Chapitre 3

Architecture Technique

3.1 Organisation des fichiers

- `login.html` : authentification
- `dashboard.html` : tableau de bord
- `menu.html` : gestion du menu
- `orders.html` : gestion des commandes
- `reservations.html` : gestion des réservations
- `staff.html` : gestion du personnel
- `inventory.html` : gestion de l'inventaire
- `*.js` : logique métier

3.2 Choix techniques

Le projet est développé en **JavaScript vanilla** afin de renforcer la compréhension du DOM, des événements et des structures de données.

3.2.1 Stockage local

```
function saveData(key, data){
  localStorage.setItem(key, JSON.stringify(data));
}

function loadData(key){
  return JSON.parse(localStorage.getItem(key) || '[]');
}
```

Listing 3.1 – Gestion centralisée du localStorage

Chapitre 4

Système d'Authentification

L'accès au back-office est protégé par un système d'authentification simple basé sur une session stockée localement.

```
if (email === "admin@app.com" && password === "admin123") {
  localStorage.setItem("maeled_session", "true");
  window.location.href = "dashboard.html";
}
```

Listing 4.1 – Authentification utilisateur

Un contrôle empêche l'accès aux pages protégées sans session valide.

Chapitre 5

Gestion des Entités (CRUD)

5.1 Gestion du Menu

Chaque plat est défini par un nom, une catégorie, un prix et un état de disponibilité.

```
menuItems.push({
  id: Date.now(),
  name: name,
  category: category,
  price: price,
  available: true
});
saveData("maeled_menu", menuItems);
```

Listing 5.1 – Ajout d'un plat au menu

5.2 Gestion des Commandes

Une commande peut contenir plusieurs plats avec quantités. Le total est calculé dynamiquement.

```
const total = order.items.reduce(
  (sum, item) => sum + item.price * item.quantity,
  0
);
```

Listing 5.2 – Calcul du total d'une commande

5.3 Gestion des Réservations

L'application empêche deux réservations sur la même table au même créneau.

```
const conflict = reservations.find(r =>
  r.table === table &&
  r.date === date &&
  r.time === time
);

if (conflict) {
  alert("Cette table est d j  r serv e.");
}
```

Listing 5.3 – Détection d'un conflit de réservation

5.4 Gestion du Personnel

```
staff.push({
  id: Date.now(),
  name: name,
  role: role,
  status: "actif"
});
saveData("maeled_staff", staff);
```

Listing 5.4 – Ajout d'un employé

5.5 Gestion de l'Inventaire

```
inventory.push({
  id: Date.now(),
  product: product,
  quantity: quantity,
  unit: unit
});
saveData("maeled_inventory", inventory);
```

Listing 5.5 – Ajout d'un produit en stock

Chapitre 6

Dashboard et Indicateurs (KPIs)

Le tableau de bord permet une visualisation synthétique de l'activité du restaurant.

6.1 Indicateurs affichés

— nombre total de commandes,
— chiffre d'affaires,
— commandes en cours,
— état du stock.

```
const revenue = orders.reduce(
  (sum, order) => sum + order.total,
  0
);
```

Listing 6.1 – Calcul du chiffre d'affaires

Chapitre 7

Difficultés et Solutions

7.1 Sélection des plats

La sélection des articles lors de la création d'une commande a nécessité la mise en place de formulaires dynamiques avec recalcul automatique du total.

7.2 Uniformisation des formulaires

Tous les modules CRUD utilisent désormais des formulaires modals, offrant une expérience utilisateur cohérente.

7.3 Gestion des images

L'image d'un plat est devenue optionnelle lors de la modification, afin d'améliorer l'ergonomie.

Conclusion

Ce projet a permis de développer une application back-office fonctionnelle et structurée, tout en consolidant les compétences en JavaScript, manipulation du DOM et logique CRUD. Il constitue une base solide pour des évolutions futures intégrant un backend réel et une base de données.