1. Introduction générale :

Ce mini-projet présente une application web front-end nommée Coffee Shop. L'objectif principal est de créer une interface web permettant à un utilisateur de consulter des produits, d'ajouter des articles à un panier, de simuler une commande (checkout) et de gérer les messages/contact et commandes via une page de gestion locale. Le projet est entièrement réalisé en HTML, CSS et JavaScript côté client et utilise localStorage pour la persistance légère des données (panier, commandes, messages).

Le rendu fourni contient le code source (pages HTML, fichiers CSS, JavaScript) et les ressources (images).

2. Contexte général du projet :

Contexte pédagogique : mini-projet de première unité de Programmation Web visant à pratiquer les fondamentaux du développement front-end (structure HTML, styles CSS, comportement JS, interaction locale et gestion de l'état via localStorage), et à produire une présentation claire et documentée du travail.

Environnement technique :

Technologies : HTML5, CSS3, JavaScript

Persistance : localStorage

Type de projet : application monopage/ multi-page statique (fichiers .html reliés par liens)

Fichiers fournis :

index.html (page d'accueil / catalogue)

order.html (page de commande / panier)

data.html (page de gestion des messages et des commandes sauvegardées)

script.js, order.js, data.js (comportement JS)

styles.css, data.css, order.css (styles)

dossier assets/png/ (images produits et illustrations)

3. Analyse et conception :

3.1 Objectifs fonctionnels

Afficher une liste de produits avec image, titre et prix.

Permettre l'ajout d'articles au panier et la modification de la quantité.

Afficher le résumé du panier (sous-total, total).

Simuler un processus de checkout (formulaire de commande).

Enregistrer localement les commandes (clé orders dans localStorage) et les messages de contact (clé contactMessages).

Fournir une page de gestion (data.html) permettant de visualiser et supprimer les messages et commandes stockés localement.

Interface responsive (pour mobile).

3.2 Contraintes techniques

Persistance locale uniquement (les données sont liées au navigateur et à la machine).

Compatibilité moderne ES6 (usage d'addEventListener, querySelector, template literals, fonctions fléchées).

3.3 Architecture et organisation des pages

Pages principales :

index.html — Page d'accueil / mise en valeur du produit / contact.

Contient un formulaire de contact (nom, email, message).

Boutons/liaisons vers la page de commande.

order.html — Catalogue & Panier.

Affiche la grille de produits (products-grid).

Gestion du panier : affichage des éléments, quantité, suppression, calculs de somme, bouton de checkout.

Scripts : order.js gère rendu des produits, actions du panier et checkout.

data.html — Page d'administration locale.

Affiche messages et commandes sauvegardés.

Boutons pour supprimer messages/commandes individuellement ou tout effacer.

Scripts : data.js gère renderMessages() et renderOrders() et la suppression.

3.4 Modèle de données (localStorage)

cart ou cartItems (gestion interne du panier) — ensemble d'objets produit {id, title, price, qty, img, ...}.

orders — tableau d'objets commandes conservées (données client + items + total + date).

contactMessages — tableau d'objets messages {name, email, message, date}.

4. Mise en œuvre :

4.1 Arborescence :

index.html
order.html
data.html
script.js
order.js
data.js
styles.css
order.css
data.css
assets/png/ (images: about.png, bestcoffee.png, image1.png, image2.png, image3.png, image4.png, image5.png, image6.png)

4.2 Technologies et bibliothèques :

Langages : HTML5 / CSS3 / JavaScript

Aucune dépendance externe (frameworks) — code léger, vanilla JS.

Images et styles inclus localement.

4.3 Description des fichiers principaux :

index.html

Titre : Coffee Shop.

Contenu : bannière, slogan (<h1> : "Faites de votre journée une réussite avec notre café spécial !"), section contact (formulaire).

Intègre script.js en bas de page.

order.html

Grille de produits (#products-grid) : chaque produit affiché avec image, titre, prix et bouton « Ajouter au panier ».

Panneau panier (éléments .cart-item, résumé subtotal, total).

Contrôles : ouvrir/fermer navigation mobile (openNav, closeNav), gestion du panier (openCart, renderCart, changeQty, removeFromCart, updateSummary) — fonctions présentes dans order.js.

Checkout : formulaire (checkoutForm) qui crée un objet commande et l'enregistre sous orders dans localStorage. Un toast / notification est utilisé pour confirmer l'action.

data.html

Conteneurs #messages et #orders pour afficher respectivement les messages de contact et les commandes sauvegardées.

Fonctions : renderMessages() et renderOrders() dans data.js, ainsi que boutons pour effacer individuellement ou tout supprimer (clearMessagesBtn, clearOrdersBtn).

order.js

Gestion complète du panier et affichage des produits.

Calculs : subtotal, total, items etc.

Interaction responsive : si largeur <= 820px, comportement mobile (scroll, highlight).

Persistance et mise à jour du DOM.

data.js

Lecture et rendu des données stockées dans localStorage.

Boutons de suppression et commandes de vidage.

script.js

Fonctions globales communes (gestion de la navigation, du menu mobile, écouteurs DOMContentLoaded, gestion du formulaire de contact pour stocker contactMessages).

Exemple : lors de la soumission du formulaire, création d'un objet contactMessage avec {name, email, message, date} et ajout à localStorage.

4.4 Fonctions clés (exemples observés) :

renderMessages() — lit contactMessages et affiche chaque message avec bouton Supprimer.

renderOrders() — lit orders et affiche chaque commande avec détails.

renderCart() — affiche éléments du panier et met à jour résumé (sous-total/total).

changeQty(index, qty) — modifie la quantité d'un élément du panier.

removeFromCart(index) — supprime un élément du panier.

updateSummary() — recalcul du total.

openCart() / closeNav() — gestion UI pour mobile/desktop.

4.5 Guide d'utilisation (comment tester rapidement) :

Ouvrir index.html dans un navigateur moderne (Chrome, Firefox, Edge).

Sur la page d'accueil, tester le formulaire de contact (saisie et envoi).

Après envoi, vérifier que les messages sont stockés : ouvrir data.html et voir la liste sous Messages.

Aller sur order.html (ou bouton/liaison depuis index) :

Ajouter plusieurs produits au panier.

Ouvrir le panier, modifier quantités, supprimer un article.

Cliquer sur Checkout, remplir le formulaire de commande : la commande sera sauvegardée dans localStorage sous la clé orders.

Retourner sur data.html pour visualiser et éventuellement supprimer les commandes sauvegardées.

Pour réinitialiser les données, utiliser les boutons Clear présents sur data.html (ou effacer manuellement l'entrée localStorage via l'inspecteur du navigateur).

4.6 Tests effectués (observations)

Le comportement du panier et du checkout a été testé localement en ajoutant et supprimant articles : le DOM est mis à jour correctement et les données persistent entre rechargements (via localStorage).

Le formulaire de contact enregistre bien des messages et permet leur suppression.

Test basique de responsivité : le code contient des contrôles pour mobile (largeur <= 820px) et une navigation adaptée.

5. Conclusion générale :

Le mini-projet **Coffee Shop** met en œuvre les principales notions demandées en Programmation Web1 :

- structuration des pages HTML,
- mise en forme CSS,
- comportement dynamique via JavaScript,
- manipulation du DOM,
- stockage local (`localStorage`) pour simuler persistance des données,
- pages distinctes pour l'interface utilisateur et pour une page de gestion (mini-admin).

Le code est organisé et directement exécutable localement (ouvrir `index.html` / `order.html` / `data.html`). Il s'agit d'une base solide pour un site e-commerce de démonstration.

5.1 Points forts

- Solution complète côté client, facile à déployer.
- Fonctionnalités attendues (panier, checkout, stockage commandes/messages) implémentées.
- Interface responsive prise en compte.
- Code clair (séparation des responsabilités : `order.js`, `data.js`, `script.js`).