ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

EMSI LES ORANGERS
ANNÉE UNIVERSITAIRE 2025/2026
3IIR G8

Rapport De Projet

CryptoFolio

Réaliser par :

- El Mehdi Goual
- Ali Znibar
- Ilyass Firar

Encadrer par :

- Mme Khadija Telmcani

Table des matières

1. Introduction

1.1 Contexte et objectif du projet

CryptoFolio est une application web pédagogique permettant de gérer un portefeuille de cryptomonnaies. Face à l'essor des actifs numériques, ce projet répond au besoin d'un outil simple et gratuit pour suivre ses investissements crypto.

Objectifs principaux

Pédagogique : Apprendre les fondamentaux du développement web moderne sans frameworks (SPA, manipulation DOM, APIs REST, LocalStorage, sécurité web).**Fonctionnel :** Créer et gérer des actifs crypto, les organiser en portefeuilles, suivre les prix en temps réel via CoinGecko API, visualiser les performances avec graphiques, calculer automatiquement les gains/pertes.

Technique : Démontrer qu'une application complète est possible avec HTML5, CSS3 (Tailwind), JavaScript vanilla, et LocalStorage uniquement.

Avantages

- Gratuit (pas d'abonnement)
- Privé (données locales uniquement)
- Simple (interface intuitive)
- Rapide (tout fonctionne localement)

2. Présentation du projet

2.1 Description générale

CryptoFolio est une Single Page Application (SPA) pour gérer un portefeuille de cryptomonnaies. Elle permet de créer des actifs (nom, quantité, prix d'achat), les organiser en portfolios, récupérer les prix actuels via CoinGecko API, et visualiser les performances avec des graphiques interactifs.

Architecture :

```
CryptoFolio/
├── index.html      (Structure + layout)
├── script.js       (Logique)
└── style.css       (Styles personnalisés)
```

Caractéristiques techniques :

- Type : Single Page Application (SPA)
- Frontend : HTML5, CSS3 (Tailwind), JavaScript vanilla
- Visualisation : Chart.js (bar + pie charts)
- API : CoinGecko (prix actuels + historiques)
- Persistance : LocalStorage (~5-10 MB)

2.2 Public cible et utilité

Public visé :

- **Investisseurs crypto débutants** : outil simple pour suivre leurs premiers holdings
- **Étudiants en développement** : exemple concret de SPA sans framework
- **Développeurs** : référence pour intégration d'APIs REST
- **Passionnés de crypto** : outil gratuit et confidentiel

Utilité pratique :

- Centraliser tous ses holdings crypto
- Calculer automatiquement la performance du portefeuille
- Visualiser la répartition des actifs (allocation)
- Comparer prix d'achat vs prix actuel
- Prendre des décisions d'investissement éclairées

Cas d'usage concret :

El Mehdi achète 0.01 BTC à 30,000 USD → Elle l'ajoute dans CryptoFolio
→ Clique "Prix(API)" pour voir le cours actuel (43,000 USD)
→ L'app calcule : +13,000 USD de gain (+43.3%)
→ Le graphique montre que BTC = 60% de son portfolio

3. Technologies utilisées

3.1 Langages

HTML5 : Structure sémantique avec balises (<main>, <aside>, <section>), formulaires avec validation native (required, type="number", type="date").

CSS3 / Tailwind : Design via CDN Tailwind (classes : flex, grid, bg-slate-50, text-sky-700). Fichier style.css pour composants personnalisés (.card, .btn-primary, .input).

JavaScript (Vanilla) : utilisant ES6+ (arrow functions, async/await, destructuring, template literals, Map/Set, array methods).

Exemples de code utilitaire :

```
const $ = (s) => document.querySelector(s);
const uid = () => Math.random().toString(16).slice(2) + Date.now().toString(16);
const n = (x) => (Number.isFinite(+x) ? +x : 0);
const usd = (x) => new Intl.NumberFormat("en-US", {style:"currency", currency:"USD"}).format(n(x));
const get = (k) => JSON.parse(localStorage.getItem(k) || "[]");
const set = (k, v) => localStorage.setItem(k, JSON.stringify(v));
const esc = (s) => String(s??"").replaceAll("<","&lt;").replaceAll(">","&gt;");
```

Figure 1 : Exemples de code utilitaire

3.2 Bibliothèques et API

Chart.js 4.4.1 (CDN) : Visualisations avec bar chart (valeur par asset) et pie chart (allocation %).

```
barChart = new Chart($("#bar"), {
  type: "bar",
  data: { labels: ["BTC","ETH"], datasets: [{label:"Value (USD)", data:[2500,1800]}] },
  options: { responsive: true, scales: { y: { beginAtZero: true } } }
});
```

CoinGecko API v3 : Récupération prix crypto.

Endpoint 1 - Prix actuels :

```
// GET /api/v3/simple/price?ids=bitcoin,ethereum&vs_currencies=usd
// Retourne: { "bitcoin": { "usd": 43000 }, "ethereum": { "usd": 2400 } }
```

Endpoint 2 - Prix historique :

```
// GET /api/v3/coins/bitcoin/history?date=14-01-2025
// Retourne: { "market_data": { "current_price": { "usd": 42000 } } }
```

LocalStorage API : Persistance client-side.

- Clés : cryptofolio_assets, cryptofolio_portfolios
- Format : JSON stringifié
- Capacité : ~5-10 MB par domaine

Structure Asset :

```
{
  id: "abc123",
  name: "Bitcoin",
  symbol: "BTC",
  coingeckoId: "bitcoin",
  quantity: 0.5,
  buyPrice: 30000,
  buyDate: "2025-01-10",
  portfolioId: "port_1",
  currentPrice: 43000,
  lastUpdated: "2026-01-14T15:30:00Z"
}
```

4. Fonctionnalités principales

4.1 Gestion des assets

CRUD complet :

- **Create :** Formulaire avec validation (name, symbol, coingeckoId, quantity, buyPrice, buyDate, portfolio)
- **Read :** Table avec recherche temps réel + tri (alphabétique, par montant investi)
- **Update :** Pré-remplissage formulaire, détection mode édition via ID caché
- **Delete :** Confirmation utilisateur avant suppression

Création asset :

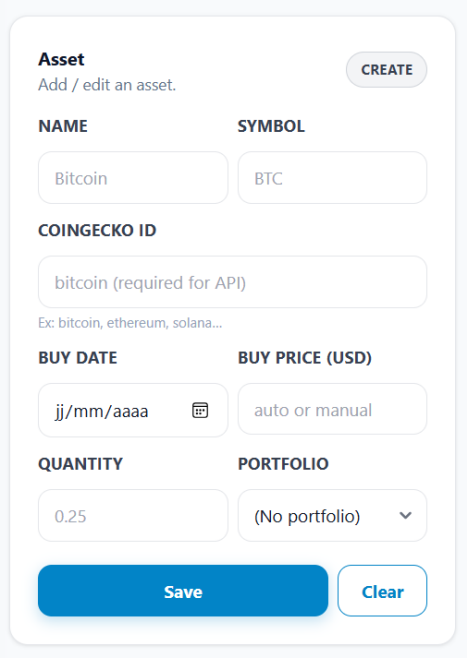

Figure 2 : Création asset

4.2 Gestion des portfolios

Création/édition : Formulaire simple (name, description). Update dropdown assets automatiquement.

Suppression avec détachement : Avant de supprimer un portfolio, on détache tous les assets liés (portfolioId = "").

```
// Détacher assets
set(K.a, get(K.a).map(a =>
  a.portfolioId === id ? { ...a, portfolioId: "" } : a
));
// Supprimer portfolio
set(K.p, get(K.p).filter(x => x.id !== id));
```

4.3 Mise à jour des prix (API)

Prix actuels : Bouton "Prix(API)" dans Dashboard récupère tous les prix via CoinGecko.

```
async function updPrices() {
  const A = get(K.a);
  const ids = [...new Set(A.map(a => a.coingeckoId).filter(Boolean))];
  const data = await cgCurrent(ids); // Appel API
  const now = new Date().toISOString();
  set(K.a, A.map(a => ({
    ...a,
    currentPrice: n(data[a.coingeckoId]?.usd),
    lastUpdated: now
  })));
}
```

Prix historique (auto-remplissage) : Quand l'utilisateur saisit date + coingeckoId, après 600ms (debounce), appel API historique pour pré-remplir le buyPrice.

```
async function autoFillBuyPrice() {
  const id = aCg.value.trim().toLowerCase();
  const dt = aDate.value;
  if (!id || !dt) return;

  aBuy.disabled = true;
  aBuy.placeholder = "Fetching...";
  try {
    const p = await cgHistoryPrice(id, dt);
    aBuy.value = Number(p).toFixed(2);
    al("Buy price auto-filled ✅", "info", 2000);
  } catch (e) {
    al(`Historical price unavailable ⚠️`, "warning");
  } finally {
    aBuy.disabled = false;
  }
}
```

4.4 Dashboard et graphiques

Calcul KPIs :

```
const inv = A.reduce((s, a) => s + n(a.quantity) * n(a.buyPrice), 0); // Total investi
const cur = A.reduce((s, a) => s + n(a.quantity) * n(a.currentPrice), 0); // Valeur actuelle
const pnl = cur - inv; // Profit/Loss
const pct = inv ? (pnl / inv) * 100 : 0; // Rendement %
```

Affichage :

- 4 KPIs : Total investi, Valeur actuelle, P/L (coloré vert/rouge), % rendement
- Table holdings : liste tous les assets avec qty, buy price, current price, value, P/L
- Graphiques : bar chart (valeur USD par asset) + pie chart (allocation %)

Mise à jour charts :

```
function updateCharts(labels, values) {
  barChart.data.labels = labels;
  barChart.data.datasets[0].data = values;
  barChart.update();

  pieChart.data.labels = labels;
  pieChart.data.datasets[0].data = values;
  pieChart.update();}
```

5. Architecture et fonctionnement

5.1 Principe SPA

Une **Single Page Application** charge une seule page HTML et met à jour le contenu dynamiquement via JavaScript, sans rechargement complet.

Implémentation :

```
// 3 vues cachées dans HTML
<section id="v-dash" class="view">Dashboard</section>
<section id="v-assets" class="view hidden">Assets</section>
<section id="v-ports" class="view hidden">Portfolios</section>

// Navigation JavaScript
const V = { dash: $("#v-dash"), assets: $("#v-assets"), ports: $("#v-ports") };

function show(v) {
  // Cacher/montrer sections
  Object.entries(V).forEach(([k, e]) => e.classList.toggle("hidden", k !== v));
  // Bouton actif
  $$(".nav-btn").forEach(b => b.classList.toggle("active", b.dataset.v === v));
  // Rafraîchir contenu
  if (v === "assets") rA();
  if (v === "ports") rP();
  if (v === "dash") dash();
}
```

Avantages : Navigation instantanée, expérience fluide, préservation de l'état, moins de bande passante.

5.2 Organisation du code

11 modules dans script.js :

Module	Rôle
1. Constantes & Outils	Raccourcis DOM, formatage, sécurité
2. Alertes	Messages colorés utilisateur
3. SPA Navigation	Switching de pages
4. Charts	Initialisation et update graphiques
5. CRUD Portfolios	Create, Edit, Delete portfolios
6. CRUD Assets	Create, Edit, Delete assets
7. API Prix Actuels	Récupération prix temps réel
8. API Prix Historique	Auto-remplissage buyPrice
9. Dashboard	Calcul KPIs + tables + charts
10. Nettoyage	Clear all localStorage
11. Initialisation	Boot app (initCharts, rP, rA, dash, show)

Conventions de nommage :

- $ = querySelector unique
- $$ = querySelectorAll
- r = render (rA, rP)
- f = form (fA, fP)
- al = alert
- cg = CoinGecko

5.3 Stockage des données

LocalStorage :

- Clés : cryptofolio_assets, cryptofolio_portfolios
- Format : JSON stringifié
- Avantages : Simple, persistant, zéro latence
- Inconvénients : Limité 5-10 MB, pas sécurisé, pas partagé entre appareils

Relations :

Portfolio (1) ←→ (N) Assets
Un portfolio peut contenir plusieurs assets
Un asset appartient à 0 ou 1 portfolio

Opérations CRUD :

```
// Create
const arr = get(K.a); arr.push(newAsset); set(K.a, arr);

// Read
const bitcoin = get(K.a).find(a => a.symbol === "BTC");

// Update
const arr = get(K.a);
const i = arr.findIndex(a => a.id === "abc123");
arr[i] = { ...arr[i], currentPrice: 45000 };
set(K.a, arr);

// Delete
set(K.a, get(K.a).filter(a => a.id !== "abc123"));
```

Sécurité :

- Validation avant stockage (if (quantity <= 0) return;)
- Échappement HTML avant rendu (${esc(asset.name)})
- Pas de données sensibles dans LocalStorage

6. Interface utilisateur

6.1 Design et navigation

Layout 2 colonnes :

- **Sidebar (fixe)** : Logo + navigation (Dashboard/Assets/Portfolios) + version
- **Main (scrollable)** : Topbar sticky (titre + hint) + alertes + contenu vue active

Palette de couleurs (Tailwind) :

- Sky (bleu) : actions, focus (bg-sky-500, text-sky-700)
- Slate (gris) : textes, backgrounds (bg-slate-50, text-slate-900)
- Green : succès, gains (text-green-700)
- Red : danger, pertes (text-red-700)
- Yellow : warnings (bg-yellow-50)

Composants CSS personnalisés :

```
.card { box-shadow: 0 10px 30px rgba(15,23,42,.08); }
.btn-primary { background: rgb(2 132 199); color: white; border-radius: .9rem; }
.input { border: 1px solid rgb(226 232 240); border-radius: .9rem; }
.nav-btn { transition: .15s ease; }
.nav-btn:hover { transform: translateX(2px); }
.nav-btn.active { background: rgba(56,189,248,.20); }
```

6.2 Expérience utilisateur

Système d'alertes :

```
al("Asset added ☑", "success", 2500);    // Vert, 2.5s
al("Editing asset ✎", "info", 1800);     // Bleu, 1.8s
al("Price unavailable ⚠", "warning", 3000); // Jaune, 3s
al("API error ✕", "danger", 4000);       // Rouge, 4s
```

Apparence : bandeau coloré en haut, bouton X pour fermer, auto-disparition après timeout.

États de chargement :

- Bouton "Prix(API)" : [Prix (API)] → [Loading...] (disabled) → [Prix (API)]
- Champ buyPrice auto-fill : placeholder "auto or manual" → "Fetching..." (disabled) → valeur remplie

Feedback visuel :

- P/L coloré : vert si positif, rouge si négatif
- Hover effects : boutons changent couleur, lignes table fond gris
- Confirmations : confirm() avant suppression

Validation temps réel :

- Champs requis : bordure rouge si vide au submit
- Type number : accepte seulement chiffres
- Messages d'erreur : alertes si quantity ≤ 0, coingeckoId manquant, etc.

7. Conclusion

CryptoFolio est une application web SPA permettant de gérer un portefeuille de cryptomonnaies avec des données en temps réel grâce à l'API CoinGecko. Ce projet a permis de mettre en pratique le JavaScript, la manipulation du DOM et l'utilisation d'APIs externes. Il constitue une base fonctionnelle pouvant être améliorée dans le futur.