

Fondamentaux du Web

HTML, CSS, JavaScript

Lab 8 — Fetch + async/await

Brancher Jàng Afrig sur les vrais pays

Licence 1 DUT1-Informatique — 2025–2026

Informations pratiques

Le moment pivot. Votre quiz fonctionne, mais avec dix pays écrits à la main. Aujourd'hui, on le branche sur l'API **REST Countries** : Jàng Afrig connaîtra désormais *tous* les pays d'Afrique, sans que vous ayez à les saisir.

Avant de commencer :

- Le quiz du Lab 7 fonctionne (tag v0.7).
- Les notions du CM4 : `fetch`, `async/await`, `try/catch`, JSON.
- Une connexion Internet pendant la séance.

Validation de fin : le quiz Capitales fonctionne avec les pays réels chargés depuis l'API, un écran de chargement s'affiche pendant la requête, et le tag v0.8 est poussé avec un rapport.

Objectifs du Lab 8

À la fin, vous saurez :

1. Faire un vrai appel réseau avec `fetch` + `async/await`.
2. **Adapter** les données reçues à la forme attendue par votre quiz (avec `map`).
3. Charger les données **au démarrage** de la page, dans une variable globale.
4. Protéger l'application contre une coupure réseau (`try/catch`).
5. Afficher un **indicateur de chargement** (spinner CSS) pendant l'attente.

Table des matières

Rappel express du CM4	2
1 Voir l'API en vrai	2
2 Écrire <code>chargerPays()</code> dans <code>data.js</code>	2
3 Charger au démarrage (variable globale)	4
4 L'écran de chargement	4

5	Le spinner qui tourne (CSS)	5
6	Gérer le réseau coupé proprement	5
7	Exercices d'application	6
8	Aide-mémoire du Lab 8	6
9	Sauvegarder mon travail (routine Git)	6
10	Résumé et transition	7

Rappel express du CM4

Une API se demande avec une URL et le verbe **GET**; elle répond en JSON; on attend la réponse avec `await fetch(...)` puis `await reponse.json()`; et on protège le tout avec `try/catch`.

1 Voir l'API en vrai

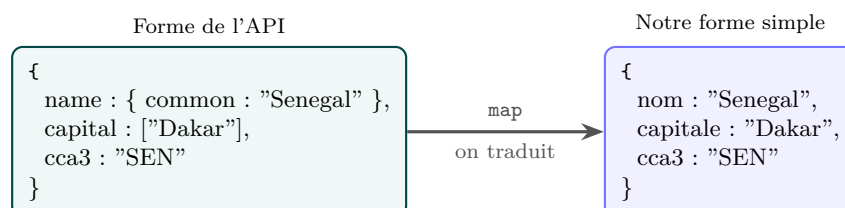
Avant de coder, regardons ce que l'API renvoie. Collez cette adresse dans la barre d'un navigateur :

```
https://restcountries.com/v3.1/region/africa?fields=name,capital,cca3,flags,languages,borders
```

Vous voyez un long texte JSON : une liste d'objets, un par pays africain. Repérez, pour un pays, les champs `name`, `capital`, `cca3`.

La forme de l'API n'est pas la nôtre

Dans REST Countries, le nom est *niché* (`name.common`) et la capitale est un *tableau* (`capital[0]`). Or notre quiz du Lab 7 attend des objets simples `{ nom, capitale }`. Il va donc falloir **traduire** la forme de l'API vers la nôtre. C'est l'idée centrale de ce Lab.



2 Écrire chargerPays() dans data.js

On transforme `data.js` : il ne contient plus de pays écrits à la main, mais une fonction qui les **récupère et les traduit**. Remplacez l'ancien tableau `const pays = [...]` par ceci (gardez `choisirAuHasard` et `melanger`) :

```
// data.js -- accès aux données + utilitaires

const URL_API =
  "https://restcountries.com/v3.1/region/africa?fields=name,capital,cca3,flags,
  languages,borders";
```

```
async function chargerPays() {
  try {
    const reponse = await fetch(URL_API);
    const donnees = await reponse.json();

    // traduire chaque pays de la forme API vers notre forme simple
    const liste = donnees.map((p) => {
      return {
        cca3: p.cca3,
        nom: p.name.common,
        capitale: p.capital ? p.capital[0] : "-",
        drapeau: p.flags.svg,
        langues: p.languages,
        voisins: p.borders
      };
    });
    return liste;
  } catch (erreur) {
    console.log("Erreur de chargement :", erreur);
    return []; // liste vide : on évite de planter
  }
}

function choisirAuHasard(tableau) {
  const index = Math.floor(Math.random() * tableau.length);
  return tableau[index];
}

function melanger(tableau) {
  const copie = [...tableau];
  for (let i = copie.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1));
    const temp = copie[i];
    copie[i] = copie[j];
    copie[j] = temp;
  }
  return copie;
}
```

Pourquoi `p.capital ? p.capital[0] : "-"` ?

Quelques rares territoires n'ont pas de capitale renseignée. Ce **ternaire** (CM3) dit : «si `capital` existe, prends le premier élément, sinon mets un tiret». C'est de la programmation *défensive* : on prévoit le cas qui manque.

Vérification dans la console

Dans la console : `const test = await chargerPays();` puis `console.table(test);`. Vous voyez vos pays, déjà à la bonne forme (nom, capitale). On garde aussi drapeau, langues, voisins : ils serviront au Lab 9.

3 Charger au démarrage (variable globale)

Le quiz a besoin des pays *avant* qu'on puisse jouer. On les charge donc **au démarrage** de la page, et on les range dans une variable globale. En haut de `script.js` :

```
// L'état du jeu
let pays = [];           // sera rempli par l'API au chargement
let questions = [];
let indexQuestion = 0;
let score = 0;
let pseudo = "";

// Au chargement de la page : on va chercher les pays
async function initialiser() {
  pays = await chargerPays();
  console.log(`${pays.length} pays chargés.`);
  afficherEcran(ecranAccueil); // on révèle l'accueil une fois prêt
}
```

Le tableau `pays` est devenu une variable, pas une constante

Avant, `pays` était un `const` figé dans `data.js`. Maintenant, c'est un `let` dans `script.js`, car son contenu *arrive plus tard* (quand l'API répond). Le reste du quiz (`genererQuestion`, etc.) n'a pas besoin de changer : il utilise toujours `pays`, `.nom` et `.capitale`.

4 L'écran de chargement

Pendant que l'API répond (une seconde ou deux), on montre un écran d'attente. Ajoutez une section **en premier** dans le `<main>` de `index.html`, et mettez la classe `cachee` sur `#accueil` :

```
<main>
  <!-- ÉCRAN 0 : CHARGEMENT (visible au départ) -->
  <section id="chargement">
    <div class="spinner"></div>
    <p>Chargement des pays ...africains</p>
  </section>

  <!-- ÉCRAN 1 : ACCUEIL (caché au départ) -->
  <section id="accueil" class="cachee">
    ... (contenu inchangé du Lab 7) ...
  </section>

  <!-- les écrans Quiz et Résultat ne changent pas -->
</main>
```

Puis, dans `script.js`, on **sélectionne** cet écran et on l'**ajoute** à `afficherEcran`, pour qu'il se cache comme les autres :

```
const ecranChargement = document.querySelector("#chargement");

function afficherEcran(ecran) {
  ecranChargement.classList.add("cachee");
  ecranAccueil.classList.add("cachee");
  ecranQuiz.classList.add("cachee");
  ecranResultat.classList.add("cachee");
  ecran.classList.remove("cachee");
}
```

```
}
```

Enfin, tout en bas de `script.js`, lancez le chargement :

```
initialiser(); // démarre le chargement dès que la page est prête
```

5 Le spinner qui tourne (CSS)

Un petit disque qui tourne, signe universel de « ça charge ». Ajoutez à la fin de `style.css` :

```
.spinner {
  width: 48px;
  height: 48px;
  border: 6px solid #eeeeee;
  border-top-color: #c0392b; /* brique de notre palette */
  border-radius: 50%;
  margin: 2rem auto;
  animation: tourner 1s linear infinite;
}

@keyframes tourner {
  to { transform: rotate(360deg); }
}
```

Une nouveauté CSS : @keyframes

Au Lab 4, vous aviez les *transitions* (passer d'un état à un autre). `@keyframes` va plus loin : il décrit une **animation** qui se répète. Ici, on fait tourner le disque de 0 à 360° en continu (*infinite*). La bordure colorée sur un seul côté crée l'effet de rotation.

6 Gérer le réseau coupé proprement

Notre `try/catch` renvoie déjà une liste vide en cas de problème. Transformons ce silence en **message clair** pour le joueur. Dans `initialiser` :

```
async function initialiser() {
  pays = await chargerPays();

  if (pays.length === 0) {
    ecranChargement.textContent =
      "Impossible de charger les pays. Vérifie ta connexion et recharge la page.";
    return; // on s'arrête là, on ne montre pas l'accueil
  }

  console.log(`${pays.length} pays chargés.`);
  afficherEcran(ecranAccueil);
}
```

Vérification — le test du mode avion

Activez le mode avion (ou coupez le Wi-Fi), rechargez la page : au lieu de planter, l'application affiche un message poli. Réactivez la connexion, rechargez : le quiz revient.
Une vraie application gère les imprévus.

7 Exercices d'application

Exercice 1 — Combien de pays ?

Dans la console, affichez `pays.length`. Combien l'API renvoie-t-elle de pays pour l'Afrique ?

Note : le nombre tourne autour de 54 ; il peut varier légèrement (54, 58, 59...) selon les territoires inclus par l'API. L'essentiel est d'avoir de quoi faire un beau quiz.

Exercice 2 — Trouver le Sénégal

Retrouvez l'objet du Sénégal grâce à son code `cca3`, et affichez sa capitale.

Indice : `find` renvoie le *premier* élément qui satisfait la condition.

```
const senegal = pays.find((p) => p.cca3 === "SEN");
console.log(senegal.capitale); // "Dakar"
```

Exercice 3 — Les pays francophones

Avec `filter`, fabriquez la liste des pays dont les langues incluent le français.

Indice : dans notre objet, `p.languages` est un objet comme `{ fra: "French", ... }`. La clé `fra` est présente si le pays parle français.

```
const francophones = pays.filter((p) => p.languages && p.languages.fra);
console.log(`${francophones.length} pays francophones`);
```

8 Aide-mémoire du Lab 8

Besoin	Code
Fonction asynchrone	<code>async function f() { ... }</code>
Lancer une requête	<code>const r = await fetch(url);</code>
Lire le JSON reçu	<code>const data = await r.json();</code>
Filet de sécurité	<code>try { ... } catch (e) { ... }</code>
Traduire une liste	<code>liste.map((p) => { return {...}; })</code>
Trouver un élément	<code>tableau.find((p) => condition)</code>
Champ niché (API)	<code>p.name.common</code> / <code>p.capital[0]</code>

9 Sauvegarder mon travail (routine Git)

Routine Git de fin de Lab — tag v0.8

```
git status
git add .
git commit -m "Lab 8 : donnees reelles via REST Countries (fetch)"
git tag v0.8
git push
git push --tags
```

Vérifiez sur GitHub que le commit *et* le tag v0.8 apparaissent.

Rapport à produire

Déposez `rapport-lab-8-NOM-Prenom.pdf` dans `rapports/`, avec quatre sections :

1. **Ce que j'ai fait** : `chargerPays`, la traduction avec `map`, le chargement au démarrage, le spinner.
2. **Difficultés rencontrées** : erreurs réseau, champs nichés, ordre de chargement...
3. **Comment je les ai résolues**.
4. **Lien vers mon repo** avec le tag `v0.8`.

10 Résumé et transition

Jàng Afrig est désormais connecté au monde réel : il charge tous les pays d'Afrique depuis une API, les traduit à sa forme interne, gère l'attente et les pannes. Le mode Capitales tourne avec de vraies données — et vous n'avez plus jamais à mettre à jour une liste à la main.

Au **Lab 9**, on exploite la richesse de ces données pour ajouter de **nouveaux modes de jeu** : Drapeaux (grâce à `drapeau`), Langues (grâce à `langues`), Voisins (grâce à `voisins`). Le squelette du quiz est déjà là : il ne reste qu'à varier les questions.

Les vraies données sont là. Au Lab 9, le jeu prend de l'ampleur.

Dr. El Hadji Bassirou TOURÉ — École Supérieure Polytechnique — UCAD
Licence 1 DUT1-Informatique — Fondamentaux du Web — 2025–2026