JOUR 4 – Applications Dynamiques avec APIs et Stockage

Objectif du Jour

Maîtriser la communication avec des APIs et le stockage local en JavaScript.

PARTIE 1 : Comprendre les APIs

Qu'est-ce qu'une API ?

Une **API** permet à votre application de communiquer avec un serveur pour échanger des données.

Flux de communication :

```
1. Vous envoyez une requête (fetch)
2. Le serveur traite
3. Vous recevez une réponse (JSON)
```

Les 3 Concepts Clés

1. `fetch()` - Envoyer une requête

```
fetch('https://api.exemple.com/data')
```

2. `async/await` - Attendre la réponse

```
async function getData() {
    const response = await fetch(url);  // Attend la réponse
    const data = await response.json(); // Convertit en objet
}
```

3. JSON - Format d'échange

```
// Serveur envoie :
'{"name":"Ahmad","age":25}'

// JavaScript reçoit :
{name: "Ahmad", age: 25}
```

PROJET 1 : Application Météo

HTML Minimal

```html
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Météo App</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <h1>🌤️ Météo</h1>
        <input type="text" id="city-input" placeholder="Ville...">
        <button onclick="getWeather()">Rechercher</button>
        <div id="result"></div>
    </div>
    <script src="weather.js"></script>
</body>
</html>
```

JavaScript Expliqué (weather.js)

```javascript
// ============================================
// 1. CONFIGURATION
// ============================================

const API_KEY = 'VOTRE_CLE_API'; // Obtenez sur openweathermap.org
const API_URL = 'https://api.openweathermap.org/data/2.5/weather';

// ============================================
// 2. FONCTION PRINCIPALE
// ============================================

async function getWeather() {
    // Étape 1 : Récupérer la ville
    const city = document.getElementById('city-input').value.trim();
    const resultDiv = document.getElementById('result');

    // Étape 2 : Validation
    if (!city) {
        resultDiv.innerHTML = '<p class="error">⚠️ Entrez une ville</p>';
        return;
    }

    // Étape 3 : Chargement
    resultDiv.innerHTML = '<p>⏳ Chargement...</p>';

    try {
        // Étape 4 : Construire l'URL
        const url = `${API_URL}?q=${city}&appid=${API_KEY}&units=metric&lang=fr`;

        // Étape 5 : Envoyer la requête
        const response = await fetch(url);

        // Étape 6 : Vérifier la réponse
        if (!response.ok) {
            throw new Error(response.status === 404 ? 'Ville non trouvée' : 'Erreur API');
        }

        // Étape 7 : Convertir en JSON
        const data = await response.json();

        // Étape 8 : Afficher
        displayWeather(data);

        // Étape 9 : Sauvegarder
```

```
        localStorage.setItem('lastCity', city);

    } catch (error) {
        resultDiv.innerHTML = `<p class="error">❌ ${error.message}</p>`;
    }
}

// ============================================
// 3. AFFICHAGE DES DONNÉES
// ============================================

function displayWeather(data) {
    const resultDiv = document.getElementById('result');

    // Extraire les données
    const temp = Math.round(data.main.temp);
    const desc = data.weather[0].description;
    const humidity = data.main.humidity;
    const wind = Math.round(data.wind.speed * 3.6);

    // Créer le HTML
    resultDiv.innerHTML = `
        <div class="weather-card">
            <h2>${data.name}, ${data.sys.country}</h2>
            <p class="temp">${temp}°C</p>
            <p>${desc}</p>
            <p>💧 ${humidity}% | 💨 ${wind} km/h</p>
        </div>
    `;
}

// ============================================
// 4. ÉVÉNEMENTS
// ============================================

// Recherche avec Entrée
document.getElementById('city-input').addEventListener('keypress', (e) => {
    if (e.key === 'Enter') getWeather();
});

// Charger dernière ville
window.onload = () => {
    const lastCity = localStorage.getItem('lastCity');
    if (lastCity) {
        document.getElementById('city-input').value = lastCity;
    }
};
```

🔍 Explication Détaillée

A. Structure de la Réponse API

Quand vous appelez l'API météo, vous recevez un objet comme ceci :

```
{
    name: "Paris",
    sys: { country: "FR" },
    main: {
        temp: 18.5,
        feels_like: 17.2,
        humidity: 65
    },
    weather: [
        { description: "nuageux" }
    ],
    wind: { speed: 3.5 }
}
```

B. Async/Await vs Promises

```
// ❌ Sans async/await (compliqué)
fetch(url)
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.error(error));

// ✅ Avec async/await (simple)
async function getData() {
    try {
        const response = await fetch(url);
        const data = await response.json();
        console.log(data);
    } catch (error) {
        console.error(error);
    }
}
```

C. Gestion des Erreurs

```
try {
    // Code qui peut échouer
    const response = await fetch(url);

    // Vérifier le statut HTTP
    if (!response.ok) {
        throw new Error('Erreur réseau');
    }

} catch (error) {
    // Code exécuté en cas d'erreur
    console.error('Une erreur est survenue:', error.message);
}
```

🎯 PARTIE 2 : Le Stockage Local

Pourquoi localStorage ?

Sauvegarder des données dans le navigateur qui **persistent après fermeture**.

Les 4 Opérations

```
// 1⃣ SAUVEGARDER
localStorage.setItem('nom', 'Ahmad');

// 2⃣ RÉCUPÉRER
const nom = localStorage.getItem('nom'); // 'Ahmad'

// 3⃣ SUPPRIMER
localStorage.removeItem('nom');

// 4⃣ TOUT SUPPRIMER
localStorage.clear();
```

⚠ Règle Importante : JSON

localStorage stocke UNIQUEMENT du TEXTE

```
// ❌ ERREUR
const user = { name: 'Ahmad', age: 25 };
localStorage.setItem('user', user); // "[object Object]" ❌

// ✅ CORRECT
localStorage.setItem('user', JSON.stringify(user)); // Objet → Texte

// Récupération
const userData = JSON.parse(localStorage.getItem('user')); // Texte → Objet
console.log(userData.name); // 'Ahmad' ✅
```

Vérifier l'Existence

```
// ✅ Méthode sûre
const data = localStorage.getItem('key');
if (data) {
    // La donnée existe
    console.log(JSON.parse(data));
} else {
    // La donnée n'existe pas
    console.log('Aucune donnée');
}

// ✅ Avec valeur par défaut
const todos = JSON.parse(localStorage.getItem('todos')) || [];
```

🎯 PROJET 2 : Application TODO

HTML Minimal

```html
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>TODO App</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <h1>📝 TODO List</h1>
        <input type="text" id="todo-input" placeholder="Nouvelle tâche...">
        <button onclick="addTodo()">Ajouter</button>
        <ul id="todo-list"></ul>
        <button onclick="clearAll()">Tout Effacer</button>
        <div id="stats"></div>
    </div>
    <script src="todo.js"></script>
</body>
</html>
```

JavaScript Complet et Commenté (todo.js)

```javascript
// ============================================
// 1. INITIALISATION
// ============================================

// Charger les tâches depuis localStorage
// Si rien n'existe, créer un tableau vide
let todos = JSON.parse(localStorage.getItem('todos')) || [];

// Afficher les tâches au chargement de la page
renderTodos();

// ============================================
// 2. AJOUTER UNE TÂCHE
// ============================================

function addTodo() {
    // Récupérer le texte
    const input = document.getElementById('todo-input');
    const text = input.value.trim();

    // Validation
    if (!text) {
        alert('⚠ Entrez une tâche');
        return;
    }

    // Créer l'objet tâche
    const newTodo = {
        id: Date.now(),          // ID unique basé sur timestamp
        text: text,              // Texte de la tâche
        done: false,             // État par défaut
        createdAt: new Date()    // Date de création
    };

    // Ajouter au début du tableau
    todos.unshift(newTodo);

    // Sauvegarder et afficher
    saveTodos();
    renderTodos();

    // Réinitialiser l'input
```

```
    input.value = '';
    input.focus();
}

// ============================================
// 3. MARQUER COMME TERMINÉ
// ============================================

function toggleTodo(id) {
    // Transformer le tableau
    todos = todos.map(todo => {
        // Si c'est la bonne tâche, inverser son état
        if (todo.id === id) {
            return { ...todo, done: !todo.done };
        }
        // Sinon, la retourner telle quelle
        return todo;
    });

    saveTodos();
    renderTodos();
}

// ============================================
// 4. SUPPRIMER UNE TÂCHE
// ============================================

function deleteTodo(id) {
    if (confirm('Supprimer cette tâche ?')) {
        // Filtrer pour garder tout SAUF cette tâche
        todos = todos.filter(todo => todo.id !== id);

        saveTodos();
        renderTodos();
    }
}

// ============================================
// 5. TOUT EFFACER
// ============================================

function clearAll() {
    if (todos.length === 0) {
        alert('Aucune tâche à supprimer');
        return;
    }

    if (confirm(`Supprimer les ${todos.length} tâches ?`)) {
        todos = [];
        saveTodos();
        renderTodos();
    }
}

// ============================================
// 6. SAUVEGARDER DANS LOCALSTORAGE
// ============================================

function saveTodos() {
    // Convertir le tableau en texte JSON et sauvegarder
    localStorage.setItem('todos', JSON.stringify(todos));
}

// ============================================
// 7. AFFICHER LES TÂCHES
```

```
// ============================================

function renderTodos() {
    const list = document.getElementById('todo-list');

    // Si vide, afficher un message
    if (todos.length === 0) {
        list.innerHTML = '<li class="empty">Aucune tâche</li>';
        updateStats();
        return;
    }

    // Créer le HTML pour chaque tâche
    list.innerHTML = todos.map(todo => `
        <li class="${todo.done ? 'done' : ''}">
            <input
                type="checkbox"
                ${todo.done ? 'checked' : ''}
                onchange="toggleTodo(${todo.id})"
            >
            <span onclick="toggleTodo(${todo.id})">${todo.text}</span>
            <button onclick="deleteTodo(${todo.id})">🗑</button>
        </li>
    `).join('');

    updateStats();
}

// ============================================
// 8. METTRE À JOUR LES STATISTIQUES
// ============================================

function updateStats() {
    const total = todos.length;
    const done = todos.filter(todo => todo.done).length;
    const remaining = total - done;

    document.getElementById('stats').innerHTML = `
        <p>Total: ${total} | Terminées: ${done} | Restantes: ${remaining}</p>
    `;
}

// ============================================
// 9. ÉVÉNEMENTS
// ============================================

// Ajouter avec la touche Entrée
document.getElementById('todo-input').addEventListener('keypress', (e) => {
    if (e.key === 'Enter') addTodo();
});
```

🎓 Concepts JavaScript Importants

A. Méthodes de Tableau

```
// map() - Transformer chaque élément
const nombres = [1, 2, 3];
const doubles = nombres.map(n => n * 2); // [2, 4, 6]

// filter() - Garder certains éléments
const pairs = nombres.filter(n => n % 2 === 0); // [2]

// find() - Trouver un élément
const deux = nombres.find(n => n === 2); // 2

// some() - Vérifier si au moins un élément correspond
const hasThree = nombres.some(n => n === 3); // true
```

B. Spread Operator (...)

```
// Copier un objet en modifiant une propriété
const todo = { id: 1, text: 'Tâche', done: false };
const updated = { ...todo, done: true };
// { id: 1, text: 'Tâche', done: true }
```

C. Template Literals

```
// ❌ Concaténation classique
const html = '<p>' + nom + ' a ' + age + ' ans</p>';

// ✅ Template literals
const html = `<p>${nom} a ${age} ans</p>`;
```

D. Arrow Functions

```
// ❌ Fonction classique
function double(x) {
    return x * 2;
}

// ✅ Arrow function
const double = (x) => x * 2;

// ✅ Avec plusieurs instructions
const process = (x) => {
    const result = x * 2;
    return result + 1;
};
```

🎯 Exercices Pratiques

Exercice 1 : API Blagues

Utilisez l'API gratuite : `https://official-joke-api.appspot.com/random_joke`

```
async function getJoke() {
    const response = await fetch('https://official-joke-api.appspot.com/random_joke');
    const joke = await response.json();
    console.log(joke.setup);
    console.log(joke.punchline);
}
```

Mission : Créez une page qui affiche une blague aléatoire au clic.

Exercice 2 : Compteur avec localStorage

```
let count = parseInt(localStorage.getItem('count')) || 0;

function increment() {
    count++;
    localStorage.setItem('count', count);
    display();
}

function display() {
    document.getElementById('count').textContent = count;
}
```

Mission : Ajoutez des boutons décrémentation et reset.

Exercice 3 : TODO Avancé

Ajoutez ces fonctionnalités :

- Catégories (Travail, Personnel, Urgent)
- Date d'échéance
- Tri par priorité
- Recherche de tâches

Débogage et Outils

Console du Navigateur (F12)

```
// Afficher des valeurs
console.log('Valeur:', variable);

// Afficher un objet
console.table(todos);

// Mesurer le temps
console.time('fetch');
await fetch(url);
console.timeEnd('fetch');

// Grouper les logs
console.group('Mes Logs');
console.log('Log 1');
console.log('Log 2');
console.groupEnd();
```

Voir localStorage

1. **F12** → Onglet **Application** (Chrome) ou **Stockage** (Firefox)
2. **Local Storage** → **Votre site**
3. Vous voyez toutes les clés/valeurs

Erreurs Courantes

```
// ❌ Oublier await
const data = fetch(url).json(); // Retourne une Promise, pas les données

// ✅ Correct
const response = await fetch(url);
const data = await response.json();

// ❌ Oublier JSON.parse
const todos = localStorage.getItem('todos'); // Retourne un STRING

// ✅ Correct
const todos = JSON.parse(localStorage.getItem('todos'));
```

✅ Checklist de Maîtrise

- ☐ Je comprends le concept d'API
- ☐ Je sais utiliser `fetch()` avec `async/await`
- ☐ Je gère les erreurs avec `try/catch`
- ☐ Je sais convertir JSON → Objet et vice-versa
- ☐ Je maîtrise les 4 opérations localStorage
- ☐ Je sais utiliser `map()`, `filter()`, `find()`
- ☐ J'ai créé une application fonctionnelle
- ☐ Mes données persistent après rechargement

📚 Ressources Complémentaires

APIs Gratuites pour Pratiquer

- **Météo :** openweathermap.org
- **Blagues :** official-joke-api.appspot.com
- **Citations :** api.quotable.io
- **Pays :** restcountries.com
- **Pokémon :** pokeapi.co

Commandes Git pour Sauvegarder

```
git add .
git commit -m "Jour 4: APIs et localStorage terminé"
git push
```

Prêt pour le Jour 5 ? Vous allez apprendre les animations CSS et JavaScript avancé ! 🚀