TP Final — Multiverse Dating (Rencontres Interdimensionnelles)

> ***"Le Pitch : Vous allez développer une application de rencontre pour tout l'univers ! L'utilisateur doit pouvoir chercher des partenaires selon des critères précis (Espèce, Statut vital...), voir les profils sous forme de cartes (style Tinder), et constituer son équipe de "Crushs". Ce TP utilise l'API publique The Rick and Morty API."***

L'API

Nous utilisons **The Rick and Morty API** : une API publique, gratuite, sans authentification.

URL de base : `https://rickandmortyapi.com/api/character`

Points forts :

- Personnages avec : `name`, `status` (Alive, Dead, unknown), `species` (Alien, Human, Robot), `gender`, `image` (URL directe), `location`
- Filtres via l'URL (Formulaires et Query Params)
- Pagination (Async avancé)
- Documentation : **https://rickandmortyapi.com/documentation**

Exemple de requête :

```
https://rickandmortyapi.com/api/character/?name=rick&status=alive
```

Structure de la réponse :

```
{
  "info": {
    "count": 826,
```

```
  "results": [
    {
      "id": 1,
      "name": "Rick Sanchez",
      "status": "Alive",
      "species": "Human",
      "gender": "Male",
      "image": "https://rickandmortyapi.com/api/character/avatar/1.jpeg",
      "location": {
        "name": "Citadel of Ricks"
      },
      "origin": {
        "name": "Earth (C-137)"
      }
    }
  ]
}
```


Structure du Projet

Le Formulaire de Recherche (Construction d'URL)

Objectif

Créer l'interface de filtrage et comprendre comment un Formulaire HTML pilote une requête API.

Consignes

1. **Créer un formulaire HTML** avec les champs suivants :
    - Input `text` pour le **nom** du personnage
    - Select pour le **statut** : Alive, Dead, Unknown

- Écouter l'événement `submit` du formulaire
- Utiliser `e.preventDefault()` pour empêcher le rechargement de la page
- Récupérer les valeurs des champs
- Construire la chaîne de paramètres (Query String) à ajouter à l'URL de base

3. **Résultat attendu** :
   - Un `console.log` qui affiche l'URL complète prête à être appelée
   - Exemple : `https://rickandmortyapi.com/api/character/?name=rick&status=alive&gender=male`

Exemple de code de départ

```
<form id="searchForm">
  <input type="text" id="nameInput" placeholder="Nom du personnage">

  <select id="statusSelect">
    <option value="">Tous les statuts</option>
    <option value="alive">Alive</option>
    <option value="dead">Dead</option>
    <option value="unknown">Unknown</option>
  </select>

  <select id="genderSelect">
    <option value="">Tous les genres</option>
    <option value="female">Female</option>
    <option value="male">Male</option>
    <option value="genderless">Genderless</option>
    <option value="unknown">Unknown</option>
  </select>

  <button type="submit">Rechercher</button>
</form>
```

Aide

Fetch & Rendu de Carte (Async & Template)

Objectif

Récupérer les données de l'API et afficher les personnages sous forme de cartes.

Consignes

1. **Utiliser** `fetch()` avec l'URL construite au Jour 1
2. **Gérer la Promesse** avec `async/await`
3. **Data Mapping** : L'API renvoie un objet `{ info: {...}, results: [...] }`. Il faut cibler `results`
4. **DOM Manipulation** : Pour chaque personnage trouvé, générer une carte HTML avec :
   - L'image du personnage
   - Son nom dans un `<h2>`
   - Un badge indiquant son statut (Alive/Dead/Unknown)
   - Son espèce et son genre
5. **Gestion d'erreur** : Si l'API renvoie une erreur 404 (aucun personnage trouvé), afficher un message "Personne dans cette dimension"

Exemple de carte HTML

```
<div class="character-card">
  <img src="[URL_IMAGE]" alt="[NOM]">
  <h2>[NOM]</h2>
  <span class="badge status-[STATUS]">[STATUS]</span>
  <p>Espèce: [SPECIES]</p>
```

Aide

- Créez une fonction `renderCharacters(characters)` qui génère le HTML
- Utilisez `innerHTML` ou `createElement` pour insérer les cartes
- Ajoutez du CSS pour rendre les cartes attrayantes

Le “Deck” de Rencontre (DOM Mutation)

Objectif

Créer une interaction style Tinder avec une pile de cartes où l’utilisateur peut “liker” ou “passer”.

Consignes

1. **Stocker les résultats** de l’API dans un tableau global `currentCandidates`
2. **Afficher uniquement** l’index 0 du tableau (une seule carte visible à la fois)
3. **Ajouter deux boutons et deux actions** :
   - ❌ (Passer) ou swipe vers la gauche
   - 💚 (Liker) ou swipe vers la droite
4. **Mutation du DOM** :
   - ❌ : Utiliser `currentCandidates.shift()`, supprimer la carte du DOM, afficher la suivante
   - 💚 : Ajouter le personnage dans un tableau `myMatches`, puis passer au suivant
5. **Fin du deck** : Afficher un message quand il n’y a plus de personnages

Aide

- Utilisez `array.shift()` pour retirer le premier élément

La Liste des Matchs (Side Effects & LocalStorage)

Objectif

Gérer l'état de l'application sur le long terme avec le localStorage.

Consignes

1. **Afficher la liste des matchs** dans une colonne latérale (sidebar)
    - Afficher seulement les avatars en petit format
    - Afficher le nom au survol ou en dessous
2. **Sauvegarde** : À chaque "Like", sauvegarder le tableau `myMatches` dans le `localStorage`
3. **Restauration** : Au chargement de la page, vérifier si des matchs existent dans le storage et les afficher
4. **Suppression** : Permettre de cliquer sur un match pour le supprimer
    - Mise à jour du DOM
    - Mise à jour du localStorage

Exemple de code localStorage

```
// Sauvegarder les matchs
function saveMatches() {
    localStorage.setItem('myMatches', JSON.stringify(myMatches));
}

// Charger les matchs
function loadMatches() {
    const stored = localStorage.getItem('myMatches');
    if (stored) {
        myMatches = JSON.parse(stored);
```

```
// Supprimer un match
function removeMatch(characterId) {
    myMatches = myMatches.filter(match => match.id !== characterId);
    saveMatches();
    renderMatches();
}
```

Aide

- Utilisez `JSON.stringify()` pour sauvegarder et `JSON.parse()` pour lire
- Appelez `loadMatches()` au chargement de la page
- Créez une fonction `renderMatches()` pour afficher la liste

UX & Pagination (Aller plus loin)

Objectif

Peaufiner l'expérience utilisateur avec la pagination et une modal de détails.

Consignes

1. **Pagination** :
    - L'objet `info` de l'API contient un champ `next` (l'URL de la page suivante)
    - Ajouter un bouton "Charger plus de profils" qui utilise cette URL
    - Ajouter les nouveaux personnages au deck actuel
2. **Modal de détails** :
    - Quand on clique sur la photo d'un match, ouvrir une `<dialog>` (balise native HTML)
    - Afficher les détails complets :

- Localisation actuelle (`location.name`)
- Nombre d'épisodes (`episode.length`)

Exemple de Modal HTML5

```
<dialog id="characterModal">
  <div class="modal-content">
    <button id="closeModal">✕</button>
    <img id="modalImage" src="" alt="">
    <h2 id="modalName"></h2>
    <p><strong>Statut:</strong> <span id="modalStatus"></span></p>
    <p><strong>Espèce:</strong> <span id="modalSpecies"></span></p>
    <p><strong>Origine:</strong> <span id="modalOrigin"></span></p>
    <p><strong>Localisation:</strong> <span id="modalLocation"></span></p>
  </div>
</dialog>
```

Critères d'Évaluation

Fonctionnel (12 points)

- ✅ Formulaire de recherche fonctionnel (2 pts)
- ✅ Fetch API et gestion des erreurs (2 pts)
- ✅ Affichage des cartes de personnages (2 pts)
- ✅ Système de "like/pass" fonctionnel (2 pts)
- ✅ Sauvegarde et restauration avec localStorage (2 pts)
- ✅ Pagination ou Modal (2 pts)

Code (5 points)

- ✅ Code propre et bien structuré (2 pts)

Design (3 points)

- ✅ Interface agréable et responsive (2 pts)
- ✅ UX fluide et intuitive (1 pt)

Bonus (Points supplémentaires)

- 🌟 **Animations CSS** lors du passage de cartes (transition, transform)
- 🌟 **Filtres multiples** (combiner plusieurs critères de recherche)
- 🌟 **Compteur** de likes/pass affichés en temps réel
- 🌟 **Dark mode** avec switch toggle
- 🌟 **Partage** de matchs (copier un lien avec les IDs)
- 🌟 **Easter eggs** sur certains personnages à mentionner durant la correction (Rick Sanchez, Morty...)

Ressources

- 📚 Documentation de l'API
- 📚 MDN - Fetch API
- 📚 MDN - LocalStorage
- 📚 MDN - Dialog Element
- 📚 MDN - URLSearchParams

Rendu

- `style.css`
- `script.js`
- Un fichier `README.md` expliquant votre démarche et les fonctionnalités implémentées

Date limite : A voir sur l'ENT (Retard = 0)

Livraison : ENT

Bon courage pour ce voyage interdimensionnel ! 🚀👽