React JS pour Master ISITCOM

Séance 0 : Prérequis et Mise en Place de l'Environnement

Dr . Abdelweheb GUEDDES

29 septembre 2025

Table des matières

1 Introduction : Objectifs de la Séance 0

Bienvenue dans cette formation React JS ! Avant de plonger dans l'écosystème React, il est impératif de s'assurer que nous partageons tous un socle commun de connaissances et d'outils.

Cette séance préliminaire a trois objectifs principaux :

1. **Valider les prérequis** en JavaScript moderne (ES6+) qui sont absolument essentiels pour écrire du code React efficace.
2. **Comprendre le "Pourquoi"** de React en analysant la manipulation traditionnelle du DOM et ses limites.
3. **Installer et configurer** un environnement de développement complet et fonctionnel sur vos machines.

À qui s'adresse cette séance ?

Cette séance est un rappel et une mise à niveau. Nous partons du principe que vous avez déjà des bases solides en HTML, CSS et JavaScript. Si les concepts ci-dessous vous semblent totalement nouveaux, un travail personnel approfondi sera nécessaire pour suivre le reste de la formation.

2 Rappels Fondamentaux (Théorie)

2.1 Le JavaScript "Moderne" (ES6+) : Un Prérequis Non-Négociable

React s'appuie massivement sur les fonctionnalités introduites depuis la version ES6 (ECMAScript 2015) de JavaScript. Maîtriser les concepts suivants est indispensable.

2.1.1 Variables : `let` et `const`

Oubliez 'var'. Utilisez 'const' par défaut pour les valeurs qui ne seront pas réassignées, et 'let' pour celles qui le seront. Cela améliore la prévisibilité et la robustesse de votre code.

2.1.2 Les Fonctions Fléchées (Arrow Functions)

Une syntaxe plus concise pour écrire des fonctions, qui résout également les problèmes de contexte de 'this'.

```
// Fonction classique
function add(a, b) {
  return a + b;
}

// Fonction fléchée
const addArrow = (a, b) => a + b;
```

Listing 1 – Syntaxe classique vs. fonction fléchée

2.1.3 La Déstructuration (Destructuring)

Une manière simple d'extraire des valeurs d'objets ou de tableaux. Vous l'utiliserez constamment avec les props et les hooks en React.

```
// Pour un objet
const person = { name: 'Alice', age: 25 };
const { name, age } = person; // Crée les variables name et age
console.log(name); // 'Alice'

// Pour un tableau (très utilisé avec le hook useState)
const [first, second] = ['pomme', 'banane'];
console.log(first); // 'pomme'
```

Listing 2 – Déstructuration d'objets et de tableaux

2.1.4 Modules : `import` et `export`

Le système de modules ES6 est la base de l'organisation de tout projet React. Il permet de diviser le code en fichiers réutilisables.

```
// Fichier: math.js
export const PI = 3.14;
export const add = (a, b) => a + b;

// Fichier: main.js
import { PI, add } from './math.js';
console.log(add(PI, 10));
```

Listing 3 – Exemple de modules

2.1.5 Autres Notions Clés

Assurez-vous d'être également à l'aise avec les concepts suivants, qui sont omniprésents en React.

— **Les Template Literals (chaînes de caractères littérales)**
Introduites avec ES6, elles permettent de manipuler les chaînes de caractères de manière plus souple et lisible en utilisant des backticks (“ ‘ “). Leurs deux avantages majeurs sont :

1. L'interpolation facile de variables et d'expressions avec la syntaxe `${expression}`.
2. La création de chaînes de caractères sur plusieurs lignes sans avoir à utiliser l'opérateur `+`.

```
const user = { name: 'Alex', level: 10 };

// Avant (avec l'opérateur +)
const classicWelcome = 'Bienvenue, ' + user.name + ' !\n' +
                       'Vous etes au niveau ' + user.level + '.';

// Maintenant (avec les Template Literals)
const modernWelcome = `Bienvenue, ${user.name} !
Vous etes au niveau ${user.level}.`;

console.log(classicWelcome);
console.log(modernWelcome); // Le resultat est identique, la syntaxe est plus
    propre.

```

Listing 4 – Concaténation classique vs. Template Literals

— **Les méthodes de tableau : `.map()`, `.filter()`, `.reduce()`**
Ces méthodes permettent de parcourir des tableaux de manière déclarative, sans utiliser de boucles `for` classiques. Elles sont fondamentales en programmation fonctionnelle et donc en React.

— `.map()` : Transforme chaque élément d'un tableau et retourne un **nouveau** tableau de même longueur avec les éléments transformés. C'est la méthode la plus utilisée en React pour afficher des listes d'éléments.

```
const numbers = [1, 2, 3, 4];
const squares = numbers.map(num => num * num);
// squares vaut maintenant [1, 4, 9, 16]

// En React, on l'utilise pour transformer des donnees en elements JSX
const fruits = ['pomme', 'banane', 'fraise'];
const fruitListItems = fruits.map(fruit => `<li key={fruit}>${fruit}</li>`);
// fruitListItems contient des elements de liste prets a etre affiches

```

Listing 5 – Utilisation de .map()

— `.filter()` : Crée un **nouveau** tableau contenant uniquement les éléments du tableau d'origine qui passent un test (une fonction qui retourne `true` ou `false`).

```
const products = [
  { name: 'Souris', price: 25 },
  { name: 'Clavier', price: 80 },
  { name: 'Ecran', price: 250 },
  { name: 'Tapis', price: 15 }
];

const affordableProducts = products.filter(product => product.price < 50);
// affordableProducts ne contient que la souris et le tapis

```

Listing 6 – Utilisation de .filter()

— `.reduce()` : Applique une fonction "réductrice" à chaque élément du tableau pour le réduire à une seule valeur (par exemple, une somme, un objet, une chaîne).

```
const amounts = [10, 25, 150, 65];
const total = amounts.reduce((accumulator, currentValue) => accumulator +
    currentValue, 0);
// total vaut 250 (0+10 -> 10+25 -> 35+150 -> 185+65)

```

Listing 7 – Utilisation de .reduce()

— **Les Promesses (Promises) et la syntaxe `async/await`**

Le code asynchrone est crucial pour les applications web qui doivent attendre des réponses réseau (appels API) sans bloquer l'interface utilisateur. Les *Promises* sont des objets qui représentent la complétion (ou l'échec) future d'une opération asynchrone.

La syntaxe `async/await` est une surcouche qui rend le code asynchrone beaucoup plus lisible, en lui donnant l'apparence d'un code synchrone.

```
// Simulation d'une fonction qui recupere des donnees utilisateur apres 1
    seconde
function fetchUserData(userId) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (userId === 1) {
        resolve({ id: 1, name: 'John Doe', email: 'john.doe@example.com' });
      } else {
        reject(new Error('Utilisateur non trouve'));
      }
    }, 1000);
  });
}

// Fonction asynchrone qui utilise 'await' pour attendre la reponse
async function displayUserData() {
  try {
    console.log('Recuperation des donnees...');
    const user = await fetchUserData(1); // Met en pause l'execution de CETTE
    fonction
    console.log(`Nom: ${user.name}, Email: ${user.email}`);
  } catch (error) {
    console.error(`Erreur: ${error.message}`);
  }
}

displayUserData();

```

Listing 8 – Gérer un appel API avec async/await

Importance pour React

Ces trois concepts sont au cœur du développement React moderne. Vous utiliserez les **Template Literals** pour construire des chaînes de caractères, `.map()` quotidiennement pour afficher des listes, et `async/await` pour interagir avec des serveurs externes afin de récupérer ou d'envoyer des données. Une maîtrise solide de ces outils est indispensable.

3 La Manipulation du DOM : L'Ancienne vs. la Nouvelle Approche

3.1 Le DOM : Document Object Model

Le DOM est une interface de programmation qui représente un document HTML ou XML sous forme d'une arborescence. Chaque nœud de l'arbre est un objet représentant une partie du document (un élément, un attribut, du texte). JavaScript peut lire et modifier cet arbre.

3.2 L'Approche Impérative (JavaScript "Vanilla")

C'est la manière "traditionnelle" d'interagir avec le DOM. Vous donnez des instructions étape par étape sur la façon de modifier l'interface.

```html
<!DOCTYPE html>
<html>
<body>
  <div id="root"></div>
  <script src="script.js"></script>
</body>
</html>
```

Listing 9 – Un simple fichier HTML

```javascript
const root = document.getElementById('root');
const fruits = ['Fraise', 'Banane', 'Orange'];

const title = document.createElement('h1');
title.textContent = 'Liste de fruits';
root.appendChild(title);

const ul = document.createElement('ul');
fruits.forEach(fruit => {
  const li = document.createElement('li');
  li.textContent = fruit;
  ul.appendChild(li);
});

root.appendChild(ul);
```

Listing 10 – script.js : Création d'une liste de manière impérative

Les Limites de l'Approche Impérative

Ce code fonctionne, mais il devient rapidement ingérable sur des applications complexes.

- **Verbeux et complexe :** Le code est long pour une tâche simple.
- **Gestion de l'état difficile :** Comment mettre à jour la liste si un fruit est ajouté ? Il faut écrire du code pour trouver le bon ‘<li>‘ à supprimer ou à ajouter, ce qui est source d'erreurs.
- **Problèmes de performance :** Chaque manipulation directe du DOM est une opération coûteuse pour le navigateur.

3.3 L'Approche Déclarative : La Promesse de React

Avec React, vous ne manipulez plus le DOM directement. Vous décrivez l'interface utilisateur que vous souhaitez obtenir pour un état de données donné, et React se charge de mettre à jour le DOM pour vous de manière optimisée.

```
function FruitList() {
  const fruits = ['Fraise', 'Banane', 'Orange'];

  return (
    <div>
      <h1>Liste de fruits</h1>
      <ul>
        {fruits.map(fruit => (
          <li key={fruit}>{fruit}</li>
        ))}
      </ul>
    </div>
  );
}
```

Listing 11 – Équivalent conceptuel en React (JSX)

Ce code est plus lisible, plus concis et plus facile à maintenir. La complexité de la manipulation du DOM est abstraite par React. C'est le problème fondamental que React vient résoudre.

4 Mise en Place de l'Environnement (Travaux Pratiques)

4.1 Outils Indispensables

Pour suivre cette formation, vous devez installer les outils suivants.

4.1.1 Node.js et npm

— **Qu'est-ce que c'est ?** Node.js est un environnement qui permet d'exécuter du JavaScript en dehors d'un navigateur. npm (Node Package Manager) est le gestionnaire de paquets qui vient avec Node.js. Il permet d'installer des bibliothèques comme React.

— **Action :** Téléchargez et installez la version **LTS** (Long-Term Support) depuis https://nodejs.org/.

— **Vérification :** Ouvrez un terminal et tapez les commandes suivantes. Vous devriez voir des numéros de version s'afficher.

```
    node -v
    npm -v

```

4.1.2 Un Éditeur de Code : Visual Studio Code

— **Qu'est-ce que c'est ?** Un éditeur de code moderne, gratuit et très populaire, avec un excellent support pour JavaScript et React.

— **Action :** Téléchargez et installez VS Code depuis https://code.visualstudio.com/.

4.1.3 Extensions VS Code Recommandées

Une fois VS Code installé, ouvrez le panneau des extensions (Ctrl+Shift+X) et installez :

— **ESLint :** Pour analyser votre code et détecter les erreurs potentielles et les problèmes de style.

— **Prettier - Code formatter :** Pour formater automatiquement votre code et garantir un style cohérent.

— **Simple React Snippets :** Pour générer rapidement des squelettes de code React.

4.2 Vérification Finale de l'Installation

Suivez ces étapes pour confirmer que tout est prêt.

1. Créez un nouveau dossier sur votre ordinateur, par exemple ‘verif-env‘.
2. Ouvrez ce dossier dans VS Code.
3. Ouvrez le terminal intégré de VS Code (‘Ctrl+‘ ou ‘View > Terminal‘).
4. Tapez la commande ‘npm init -y‘. Cela crée un fichier ‘package.json‘ qui décrira votre projet.
5. Créez un fichier ‘index.js‘ et ajoutez-y la ligne suivante :

```
console.log('Mon environnement est pret pour React ! Hello ISITCOM !');

```

6. Dans le terminal, exécutez le fichier avec la commande :

```
node index.js

```

7. Si le message s'affiche dans votre terminal, votre environnement est fonctionnel.

5 Travail Préparatoire pour la Séance 1

ACTION REQUISE avant la Séance 1

- **Assurez-vous que tous les outils** listés dans la section 3 sont installés et fonctionnels sur votre machine.
- **Revoyez attentivement les concepts JavaScript ES6+** présentés. Si l'un d'eux n'est pas clair, faites des recherches complémentaires (MDN Web Docs est une excellente ressource).
- **Il n'y a pas de compte rendu à soumettre pour cette Séance 0**, mais il est de votre responsabilité d'arriver à la Séance 1 avec un environnement de développement parfaitement opérationnel. Ne pas le faire vous pénalisera fortement pour la suite des travaux pratiques.