Membres du groupe :

François Rood Marvens
Leconte Romney
Ambroise Cédric
Aladin Carlenson
Daniel Ricolson

I- Installation de Node.js

1. Installation de Node.js sur différents systèmes d'exploitation

Windows

- Rendez-vous sur le site officiel de Node.js (https://nodejs.org).
- Téléchargez le programme d'installation Windows (.msi) pour la version LTS (Long Term Support) recommandée.
- Exécutez le fichier **.msi** et suivez les étapes du programme d'installation.
- (Alternative sans droits admin) : Téléchargez le binaire au format ZIP, extrayez-le, et configurez les variables d'environnement manuellement.

macOS

- Téléchargez le programme d'installation macOS (.pkg) depuis le site officiel de Node.js (https://nodejs.org) et suivez les instructions.

Linux

- Les méthodes varient selon la distribution. Utilisez généralement le gestionnaire de paquets (ex: **apt** pour Ubuntu/Debian) ou téléchargez le binaire depuis le site officiel.

2. Vérification de l'installation (PowerShell)

Après l'installation, vérifiez les versions de Node.js et npm via les commandes :

```
node -v  # Affiche la version de Node.js
npm -v   # Affiche la version de npm
```

3. Commandes de base de npm

- Installer un module :

```
npm install <nom-du-module>  # Installe un module localement
```

- Initialiser un projet (crée un **package.json**) :

```
npm init
```

- Installer un module globalement :

```
npm install -g <nom-du-module>
```

- Mettre à jour les modules :

```
npm update
```

II- Création d'un projet React

1- Méthodes pour initialiser un projet React

Il existe deux méthodes principales pour créer un projet React :

- Create React App (CRA) :

Utilisez la commande **npx create-react-app nom-projet** pour générer un projet préconfiguré avec Webpack, Babel, et ESLint. C'est l'outil historique le plus populaire, recommandé pour les débutants.

- Vite :

Lancez **npm create vite@latest** ou **yarn create vite**, puis suivez les instructions. Vite est plus rapide que CRA grâce à sa compilation à la volée (hot module replacement) et à une configuration minimaliste. Il est idéal pour les projets modernes nécessitant des performances accrues.

2. Structure des dossiers et fichiers

La structure de base est similaire pour CRA et Vite, avec quelques nuances :

- Dossier **public/** : Contient les fichiers statiques (ex. `index.html`, icônes).
- Dossier **src/** :
  - **main.js** (ou **index.js**) : Point d'entrée de l'application.
  - **App.js** : Composant racine React.
  - **components/**, **assets/** : Dossiers personnalisés pour organiser le code.
- Fichiers de configuration :
  CRA masque les configurations (Webpack/Babel), tandis que Vite expose un fichier **vite.config.js** modifiable.

3. Le fichier package.json

Ce fichier définit les métadonnées et les dépendances du projet :

- Propriétés principales :

- **name**, **version** : Nom et version du projet.
- **dependencies** : Paquets nécessaires en production (ex. **react**, **react-dom**).
- **devDependencies** : Outils de développement (ex. **eslint**, **webpack**).
- **scripts** : Commandes raccourcies (ex. **start** pour CRA, **dev** pour Vite).

Différences entre CRA et Vite :

- CRA inclut des scripts comme **"start"**, **"build"** , tandis que Vite utilise **"dev"**, **"build"** et **"preview"** .
- Les dépendances de Vite sont généralement plus légères.

III- . Démarrage et utilisation basique

1. Commandes pour démarrer le serveur de développement

Pour démarrer un serveur de développement, plusieurs approches peuvent être utilisées selon l'environnement ou les outils employés :

- **Dans un environnement visuel (IDE ou interface graphique)** : Vous pouvez généralement démarrer un serveur en cliquant avec le bouton droit de la souris sur le serveur dans la vue "Serveurs", puis en sélectionnant **Démarrer.**

- **Avec Django** : Si vous travaillez sur un projet Django, vous pouvez lancer le serveur de développement à l'aide de la commande suivante dans votre terminal :

(PowerShell)

```
python manage.py runserver
```

Cette commande permet de démarrer le serveur local qui écoute généralement sur http://127.0.0.1:8000/ par défaut.

- **Node.js / React** : Pour les projets utilisant Node.js comme backend ou React pour le frontend, vous devez utiliser une commande similaire à celle-ci :

(PoweShell)

```
npm start
```

Cela lance le serveur de développement et ouvre normalement l'application dans votre navigateur par défaut.

2. Création d'un composant React simple

Un composant React peut être créé sous forme de fonction ou de classe. Voici comment créer un composant fonctionnel simple :

fichier jsx :

```
import React from 'react';

function MonComposant() {
  return (
    <div>
      <h1>Bienvenue sur mon application React !</h1>
      <p>Ceci est un composant React simple.</p>
    </div>
  );
}

export default MonComposant;
```

Explications :

- Le composant **MonComposant** est une fonction JavaScript qui retourne du JSX (JavaScript XML), une syntaxe semblable à HTML.
- Ce composant peut ensuite être importé et utilisé dans d'autres parties de votre application React.

3. Mécanisme de base des props et du state

Props (Propriétés)

Les props sont des données passées d'un composant parent à un composant enfant. Elles permettent de rendre les composants réutilisables et dynamiques. Exemple :

fichier jsx :

```
function Bienvenue(props) {
  return <h1>Bonjour, {props.nom} !</h1>;
}

function App() {
  return (
    <div>
      <Bienvenue nom="Alice" />
      <Bienvenue nom="Bob" />
    </div>
  );
}
```

- Dans cet exemple, le composant **Bienvenue** reçoit une prop **nom** depuis son parent **App**.
- Les props sont en lecture seule ; elles ne doivent pas être modifiées par le composant qui les reçoit.

State (État)

Le state représente les données internes d'un composant. Contrairement aux props, le state peut être modifié à l'intérieur du composant lui-même. Voici un exemple :

fichier jsx :

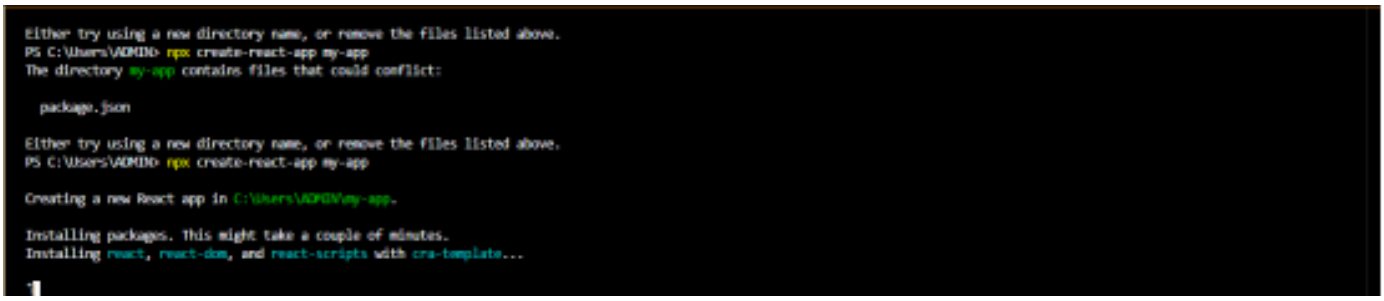
```
import React, { useState } from 'react';

function Compteur() {
```

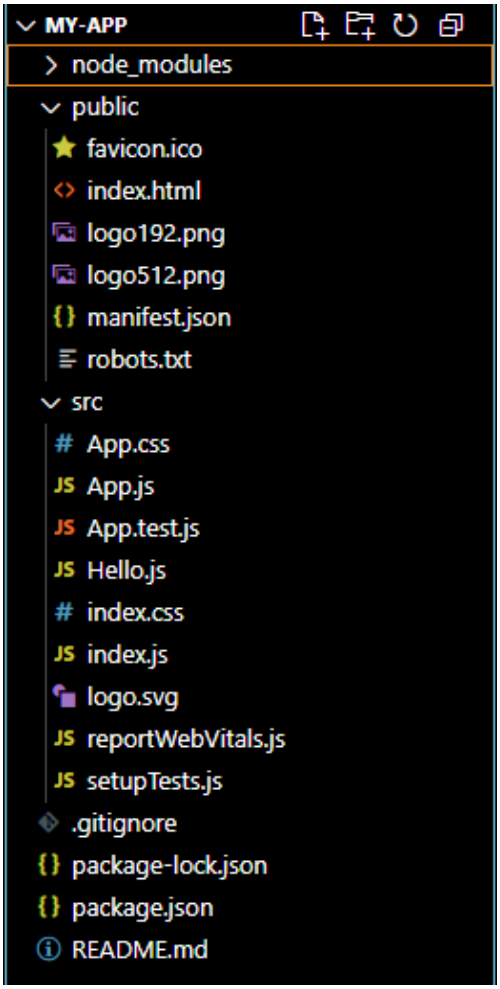
```
  const [compte, setCompte] = useState(0);

  return (
    <div>
      <p>Le compteur est à : {compte}</p>
      <button onClick={() => setCompte(compte + 1)}>Incrémenter</button>
    </div>
  );
}
```

- Ici, nous utilisons le hook **useState** pour initialiser une variable d'état **compte** avec une valeur initiale de **0**.
- Lorsque l'utilisateur clique sur le bouton, la fonction **setCompte** met à jour la valeur de **compte**, ce qui provoque un nouveau rendu du composant.

IV- Mise en pratique

1.Créer une application





2. Initialisation des méthodes

```jsx
import React from "react";
import Hello from "./Hello"; // Import du second composant

function App() {
  return (
    <div>
      <h1>Hello World</h1>
      <Hello name="Nova" />
    </div>
  );
}

export default App;
```

3. Lancement de l’application

```
Compiled successfully!

You can now view my-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.236.200:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

4. Résultat final