TD4 : NPM + Intro à React

Exercice 1 : NPM – package.json & lifecycle

1. Créer un fichier `package.json` pour un projet React en renseignant :
    - nom
    - version
    - description
2. Ajouter les scripts suivants dans `package.json` :
    - `dev`
    - `test`
    - `build`
    - `deploy`
3. Ajouter des scripts lifecycle :
    - `prebuild`
    - `postinstall`
4. Expliquer le rôle de chaque script ajouté.
5. Ajouter les dépendances nécessaires au projet React.
6. Quelle est la différence entre :
    - `dependencies`
    - `devDependencies`
7. Décrire le cycle complet suivant :
    `init → install → dev → test → build → deploy`
8. Donner les commandes NPM correspondantes à chaque étape.
9. Ajouter un script `clean` permettant de supprimer le dossier `build`.
10. Expliquer le rôle de `node_modules`.

========== Correction ==========

(1-3). Exemple de fichier package.json

```
{
  "name": "mon-projet-react",
  "version": "1.0.0",
  "description": "Projet React avec NPM, scripts et lifecycle",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "test": "vitest",
    "prebuild": "npm run clean",
    "build": "vite build",
    "deploy": "echo Deploiement du projet...",
    "postinstall": "echo Installation terminee",
```

```
    "clean": "rmdir /s /q build"
  },
  "dependencies": {
    "react": "^19.0.0",
    "react-dom": "^19.0.0"
  },
  "devDependencies": {
    "vite": "latest",
    "vitest": "latest",
  }
}
```

4. Role de chaque script

Script	Role
dev	Lance le serveur de developpement React avec Vite.
test	Execute les tests du projet avec Vitest.
prebuild	Script lifecycle execute automatiquement avant npm run build. Ici, il lance clean.
build	Genere une version de production optimisee du projet dans dist.
deploy	Lance une commande de deploiement. Dans cet exemple, il affiche seulement un message.
postinstall	Script lifecycle execute automatiquement apres npm install.
clean	Supprime les dossiers generes, comme dist et build.

5. Dependances necessaires au projet React

- react : bibliothèque principale pour créer les composants et interfaces utilisateur.
- react-dom : permet d'afficher les composants React dans le DOM du navigateur.
- vite : outil de développement et de build rapide.
- vitest : outil pour exécuter les tests.

6. Difference entre dependencies et devDependencies

Element	Utilisation	Exemples
dependencies	Packages nécessaires au fonctionnement de l'application en production.	react, react-dom
devDependencies	Packages utiles seulement pendant le développement, les tests ou le build.	vite, vitest

7. Cycle complet : init -> install -> dev -> test -> build -> deploy

Etape	Commande NPM	Description
init	npm init -y	Cree rapidement le fichier package.json.
install	npm install	Installe toutes les dependances dans node_modules.
dev	npm run dev	Lance le projet en mode developpement.
test	npm test	Execute les tests.
build	npm run build	Genere la version de production.
deploy	npm run deploy	Execute le script de deploiement.

8. Commandes utiles pour créer le projet

```
npm init
npm install react react-dom
npm install -D vite vitest
npm run dev
npm test
npm run build
npm run deploy
```

9. Script clean pour supprimer le dossier build

Solution recommandee, compatible Windows/Linux/macOS :

```
"clean": "rm -rf build"
```

Sous Windows uniquement, on peut aussi utiliser :

```
"clean": "rmdir /s /q build"
```

10. Role de node_modules

- node_modules est le dossier ou NPM installe toutes les dependances du projet.
- Il est genere automatiquement par npm install.
- Il peut etre tres volumineux et ne doit generalement pas etre envoye sur GitHub.
- Pour reconstruire node_modules, il suffit de garder package.json et package-lock.json, puis d executer npm install.

Exercice 2 : Composant React – Produit (Composants réutilisables)

1. Créer un composant React `ProduitCard`.

2. Définir les propriétés (`props`) attendues pour ce composant.
3. Décomposer `ProduitCard` en composants réutilisables :
    - titre
    - prix
    - catégorie
    - statut du stock
4. Créer chaque composant séparément.
5. Réassembler ces composants dans `ProduitCard`.
6. Afficher dynamiquement :
    - nom du produit
    - prix
    - catégorie
    - statut (en stock / rupture)

7. Comment passer des données d'un composant parent vers un composant enfant ?
8. Pourquoi découper un composant en plusieurs sous-composants ?
9. Ajouter une condition d'affichage pour le stock.
10. Proposer une amélioration pour rendre le composant réutilisable dans plusieurs pages.

========== Correction ==========

```
import React from "react";
import { createRoot } from "react-dom/client";

// Composant Titre
function ProduitTitre({ nom }) {
  return <h2>{nom}</h2>;
}

// Composant Prix
function ProduitPrix({ prix }) {
  return <p>Prix : {prix} DH</p>;
}

// Composant Catégorie
function ProduitCategorie({ categorie }) {
  return <p>Catégorie : {categorie}</p>;
}

// Composant Stock
function ProduitStock({ enStock }) {
  return (
```

```
    <p>
      Statut : {enStock ? "En stock" : "Rupture"}
    </p>
  );
}

// Composant principal
function ProduitCard({ nom, prix, categorie, enStock }) {
  return (
    <div>
      <ProduitTitre nom={nom} />
      <ProduitPrix prix={prix} />
      <ProduitCategorie categorie={categorie} />
      <ProduitStock enStock={enStock} />
    </div>
  );
}

// Composant parent
function App() {
  return (
    <ProduitCard
      nom="Clavier Gaming"
      prix={250}
      categorie="Accessoires"
      enStock={true}
    />
  );
}

createRoot(document.getElementById("root")).render(<App />);
```

7. Passer des données parent → enfant :

On passe les données avec les **props** :

```
<ProduitTitre nom={nom} />
```

8. Pourquoi découper ?

Pour rendre le code plus clair, plus facile à modifier et réutilisable.

9. Condition d'affichage stock :

```
{enStock ? "En stock" : "Rupture"}
```

10. Amélioration :

Créer un composant réutilisable avec du style et l'utiliser avec plusieurs produits :

```
<ProduitCard nom="Souris" prix={120} categorie="Accessoires"
enStock={false} />
```