Initialiser son premier projet React.

Pour poursuivre notre travail sur CinéTech, nous allons mettre en place notre premier projet React.

Ce TP nous permet de découvrir comment une application React peut être initialisée, comment adapter son fonctionnement à nos besoins, et comment préparer une base de projet simple et propre pour la suite.

Nous allons également prendre en main l'arborescence, comprendre les rôles des fichiers principaux et nettoyer le projet pour repartir avec une structure minimale.
À la fin de ce TP, notre application React sera opérationnelle et prête à accueillir les prochaines notions que nous allons aborder.

© 2025 – Grégory Boudringhin EI

1. Initialiser un projet React moderne

Vite est aujourd'hui l'outil recommandé pour démarrer une application React.
Il est rapide, léger, simple à configurer et parfaitement adapté au développement moderne.
C'est sur cette base que nous allons construire le projet CinéTech.

La première étape consiste à ouvrir notre répertoire de travail dans le nuage pédagogique.

```
Menu → File → Open Folder
```

Puis saisir le chemin suivant :

```
/var/www/html
```

Tous nos projets React seront créés à cet emplacement.
Avant d'aller plus loin, nous devons vérifier que le terminal utilise le bon utilisateur, afin d'éviter les problèmes de droits sur les fichiers.

Une fois la vérification faite, nous pouvons créer notre premier projet React.
Pour cela, exécutons la commande suivante dans le terminal :

```
> npm create vite@latest cinetech-react
```

Vite pose ensuite plusieurs questions :

- Installation du paquet *create-vite* → **Entrer** (la valeur par défaut est "oui")
- Choix du framework → **React**
- Choix du langage → **JavaScript**
- Utilisation de *rolldown-vite* → **Non**
- Installation automatique des dépendances et lancement de l'application → **Non**

Une fois le projet créé, installons les dépendances :

```
> cd cinetech-react
> npm install
```

Nous pouvons vérifier que tout fonctionne en lançant le serveur de développement :

```
> npm run dev
```

Le terminal indique que le serveur s'est correctement lancé.
Cependant, il est impossible d'accéder à l'application depuis l'extérieur.

© 2025 – Grégory Boudringhin EI

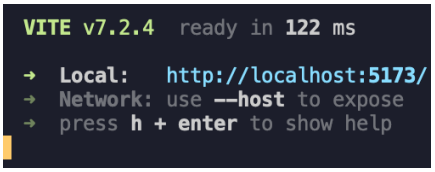

Figure 1 : Retour de la console lorsque le serveur de développement est démarré

En condition normale, elle serait déjà disponible.
Ce comportement est lié au fonctionnement du Nuage Pédagogique : il n'autorise pas le port utilisé par défaut par Vite.
Nous allons corriger ce problème dans la prochaine partie.

2. Adapter Vite pour le Nuage Pédagogique

Le nuage pédagogique n'expose que le port 3000.
Vite utilise par défaut le port 5173, il est donc nécessaire d'ajuster la configuration pour que l'application soit accessible.

Avant d'effectuer les actions suivantes, le serveur doit être coupé. Dans le terminal il suffit d'effectuer :

```
Ctrl + C
```

Dans vite.config.js, ajouter les quelques lignes pour obtenir ce résultat :

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";

export default defineConfig({
  plugins: [react()],
  server: {
    port: 3000,
    allowedHosts: true
  }
});
```

Nous relançons le serveur :

```
> npm run dev
```

Remarque :
Pour accéder au résultat, il est impératif d'utiliser le bouton d'accès à la page web du Nuage Pédagogique.

L'application React est désormais accessible depuis le nuage pédagogique sur le port 3000.

© 2025 – Grégory Boudringhin EI

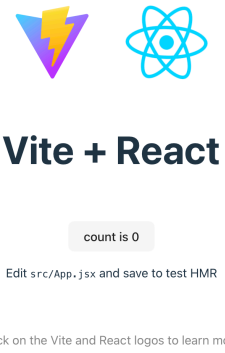

Figure 2 : Résultat obtenu par défaut du projet React généré avec ViteJS

3. Nettoyage du projet

Le projet créé contient plusieurs fichiers inutiles pour la suite du module.
L'objectif est de simplifier l'environnement afin de travailler sur une base propre. Nous allons donc supprimer une partie de ces fichiers.

Travail à réaliser :

- Supprimer les éléments suivants
  - le dossier /src/assets/,
  - le fichier App.css,
  - le fichier App.jsx,
  - le fichier index.css,
  - le fichier /public/vite.svg,
- Remplacer tout le code du fichier main.jsx par le code ci-dessous.

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

createRoot(document.getElementById('root')).render(
  <h1>CinéTech — Projet prêt</h1>
);
```

Nous pouvons maintenant vérifier que le rendu est correct dans le navigateur. Le titre « CinéTech – Projet Prêt » devrait s'afficher.

4. Nettoyage du projet

Avant d'aller plus loin, comprendre la structure d'un projet React est fondamental pour savoir où placer le code, les différents styles et les images. Mais cela permet aussi de comprendre comme fonctionne l'application.

Cette organisation n'est pas le fruit du hasard : elle a pour objectif de donner une structure claire, cohérente et facile à maintenir à un projet React. Elle permet de distinguer

© 2025 – Grégory Boudringhin EI

proprement le code de l'application, les fichiers accessibles publiquement et les ressources externes. Voici un aperçu des éléments les plus courants :

Figure 3 : Aperçu de la structure de base d'une application React

Détails de la structure de base	
node_modules/	Ce répertoire rassemble toutes les dépendances externes d'un projet (bibliothèques, outils de compilation comme React, ReactDOM, Webpack, Babel, etc.). Son contenu est automatiquement géré par npm ou yarn lors de l'installation ou de la suppression de packages. Aucune modification manuelle ne doit y être apportée. En raison de sa taille souvent importante, ce dossier est exclu du système de versionnement grâce au fichier `.gitignore`.
public/	Ce dossier regroupe l'ensemble des fichiers statiques non traités par le processus de build de Webpack ou Vite. Ces éléments, tels que les images, les icônes ou encore les manifestes, sont copiés directement dans le répertoire de build final sans transformation ni optimisation spécifique. Ce dossier sert donc de point d'entrée pour les ressources accessibles publiquement lors de l'exécution de l'application.
src/	Le dossier « src » constitue la zone principale de travail d'un projet React. Il regroupe l'ensemble du code source de l'application. Tous les éléments qu'il contient sont analysés, transformés et regroupés par les outils de build avant d'être intégrés dans la version finale de l'application.
src/main.jsx	Parmi les fichiers essentiels, on retrouve notamment le fichier `main.jsx`. Ce fichier sert de point d'entrée à la partie JavaScript de l'application. Il importe les modules nécessaires au fonctionnement de React et ReactDOM, récupère l'élément racine défini dans `public/index.html`, puis déclenche le rendu du composant principal. Cette étape initialise l'application et établit la première liaison entre React et le DOM, marquant le début de l'exécution du projet.

© 2025 – Grégory Boudringhin EI

.gitignore	Un fichier de configuration utilisé par Git pour définir les éléments qui ne doivent pas être suivis par le système de versionnement. Ce fichier liste généralement le dossier node_modules/, les répertoires issus du build tels que build/ ou dist/, ainsi que les fichiers d'environnement locaux comme .env.local. Cette exclusion permet de garder un dépôt propre et de ne versionner que les éléments réellement nécessaires au projet.
eslint.config.js	Un fichier de configuration destiné à ESLint, utilisé pour définir les règles de qualité et de cohérence du code dans un projet. Ce fichier précise les normes à respecter, les plugins à utiliser, les environnements pris en charge ainsi que les comportements à adopter lors de l'analyse du code. Grâce à cette configuration, l'outil peut détecter les erreurs, signaler les mauvaises pratiques et encourager une écriture de code plus propre et plus homogène au sein du projet.
Index.html	Fichier HTML principal d'une application monopage (SPA). Ce document sert de structure de base dans laquelle l'application React est intégrée. Il contient généralement une balise dédiée au point d'ancrage du rendu, souvent nommée root. C'est dans cet élément que l'arborescence React est montée lors de l'exécution. Ce fichier peut également accueillir des métadonnées, des liens vers des polices externes, des icônes ou tout autre élément nécessaire au chargement initial de l'application.
package.json	Fichier d'identification central d'un projet Node.js. Ce document rassemble des informations essentielles telles que le nom du projet, sa version, ainsi que l'ensemble des dépendances nécessaires. Ce fichier joue un rôle clé dans la gestion, la configuration et l'automatisation du projet.
package-lock.json	Fichier généré automatiquement qui consigne les versions exactes de chaque dépendance installée dans le projet. Sa présence garantit une installation identique sur chaque machine et assure des builds reproductibles, sans risque lié à des mises à jour imprévues de paquets. Ce fichier joue un rôle essentiel dans la stabilité du projet et doit impérativement être ajouté au dépôt Git afin de préserver la cohérence de l'environnement de travail.
README.md	Fichier rédigé en Markdown qui sert de documentation principale pour le projet. Ce document décrit généralement l'objectif du projet, les étapes d'installation, les commandes permettant son exécution, ainsi que les indications destinées aux contributeurs. Il constitue le point de référence pour comprendre, utiliser et faire évoluer le projet.

© 2025 – Grégory Boudringhin EI

vite.config.js	Fichier de configuration utilisé par Vite pour définir le comportement du système de build et de développement. Ce document permet de préciser les plugins à charger, les options de compilation, les chemins personnalisés, ainsi que divers réglages liés au serveur de développement. Grâce à cette configuration, le projet bénéficie d'un environnement optimisé, rapide et adapté aux besoins spécifiques de l'application.

© 2025 – Grégory Boudringhin EI