**1. Création du plugin avec Vite**

**1.1. Initialisation du projet avec Vite**

**Étape 1 : Exécuter la commande de création de projet avec Vite**

Dans un terminal, tapez la commande suivante pour initialiser un nouveau projet avec Vite et React :

bash

Copier le code

npm create vite@latest my-datepicker-plugin --template react

* **npm create vite@latest** : Cette commande initialise un nouveau projet en utilisant Vite.
* **my-datepicker-plugin** : Le nom du projet, qui sera le nom du plugin que nous allons créer.
* **--template react** : Indique à Vite que nous utilisons **React** comme framework pour ce projet.

**Étape 2 : Sélectionner JavaScript (ou TypeScript) selon vos besoins**

Lorsque Vite vous demande de choisir une variante, sélectionnez **JavaScript** pour un projet basé uniquement sur JavaScript (ou TypeScript si vous préférez). Dans ce cas, nous choisissons **JavaScript** pour simplifier.

**1.2. Comprendre la structure du projet générée par Vite**

Une fois l'installation terminée, Vite génère une structure de projet. Voici à quoi elle ressemble :

bash

Copier le code

/my-datepicker-plugin

│

├── /src

│ ├── /assets # Répertoire pour les fichiers statiques (images, icônes, etc.)

│ ├── /components # Répertoire pour les composants React

│ ├── App.jsx # Composant racine de l'application

│ ├── main.jsx # Point d'entrée de l'application React

│

├── /public # Contient le fichier index.html

│ ├── index.html # Fichier principal HTML du projet

│

├── package.json # Fichier de gestion des dépendances du projet

├── vite.config.js # Fichier de configuration pour Vite

└── README.md # Fichier README pour documenter le projet

* **/src** : Le répertoire où se trouve le code source principal du plugin.
* **/public** : Contient le fichier HTML principal.
* **package.json** : Contient les informations sur le projet (dépendances, scripts, etc.).
* **vite.config.js** : Configuration spécifique à Vite.

**1.3. Création du composant DatePicker**

Notre objectif est de créer un composant **DatePicker** que nous pourrons publier et utiliser dans d'autres projets.

**Étape 1 : Créer un fichier DatePicker.jsx dans /src/components/**

Voici le code du composant DatePicker :

jsx

Copier le code

// DatePicker.jsx

import React, { useState } from 'react';

// Le composant DatePicker reçoit trois props :

// - label : l'étiquette affichée pour l'input de date

// - selectedDate : la date actuellement sélectionnée

// - onChange : une fonction appelée lorsque l'utilisateur sélectionne une nouvelle date

const DatePicker = ({ label, selectedDate, onChange }) => {

// Fonction appelée lorsque l'utilisateur sélectionne une date

const handleDateChange = (e) => {

// Appelle la fonction onChange en passant la nouvelle date

onChange(new Date(e.target.value));

};

return (

<div>

{/\* Affiche l'étiquette (label) \*/}

<label>{label}</label>

{/\* Input de type date qui affiche la date sélectionnée \*/}

<input

type="date"

value={selectedDate.toISOString().split('T')[0]} // Formate la date pour l'input

onChange={handleDateChange} // Appelle handleDateChange lorsqu'une date est sélectionnée

/>

</div>

);

};

export default DatePicker;

**Explications du composant DatePicker :**

* **Props** : Le composant reçoit trois props :
  1. label : Le texte affiché au-dessus de l'input de date.
  2. selectedDate : La date actuellement sélectionnée (fournie par le parent).
  3. onChange : Une fonction qui est appelée lorsque l'utilisateur sélectionne une nouvelle date.
* **Méthode handleDateChange** : Cette fonction est déclenchée à chaque fois que l'utilisateur modifie la date dans l'input. Elle utilise la fonction onChange pour renvoyer la nouvelle date au composant parent.

**Étape 2 : Tester le composant dans App.jsx**

Maintenant que notre composant DatePicker est créé, nous devons l'intégrer dans notre application pour le tester.

1. Ouvrez le fichier **App.jsx** dans le répertoire /src et modifiez-le pour inclure et tester le DatePicker :

jsx

Copier le code

// App.jsx

import React, { useState } from 'react';

import DatePicker from './components/DatePicker'; // Import du composant DatePicker

function App() {

// État pour stocker la date de naissance

const [dateOfBirth, setDateOfBirth] = useState(new Date());

return (

<div>

<h1>Date Picker Plugin</h1>

{/\* Utilisation du DatePicker \*/}

<DatePicker

label="Select Date of Birth"

selectedDate={dateOfBirth}

onChange={setDateOfBirth} // Met à jour la date de naissance lorsque l'utilisateur sélectionne une nouvelle date

/>

{/\* Affiche la date sélectionnée \*/}

<p>Date sélectionnée : {dateOfBirth.toDateString()}</p>

</div>

);

}

export default App;

**Explications :**

* Nous avons créé un état (dateOfBirth) pour stocker la date sélectionnée par l'utilisateur.
* Le composant DatePicker est utilisé avec les props suivantes :
  + label : affiche "Select Date of Birth".
  + selectedDate : passe la date actuelle de dateOfBirth comme valeur.
  + onChange : met à jour la date de naissance lorsque l'utilisateur sélectionne une nouvelle date.

**1.4. Lancer l'application et tester localement**

Pour tester votre composant DatePicker dans l'application, exécutez cette commande dans le terminal :

bash

Copier le code

npm run dev

Cela va démarrer le serveur de développement **Vite** et ouvrir l'application dans le navigateur à l'adresse suivante : **http://localhost:3000**.

**Vous devriez voir l'interface suivante :**

* Un champ pour sélectionner une date avec un label "Select Date of Birth".
* La date sélectionnée est affichée en dessous sous forme de texte.

**1.5. Structuration pour publication**

**Explication du fichier package.json**

Le fichier package.json est crucial dans tout projet Node.js (incluant React, car React fonctionne avec Node.js). Il contient toutes les informations sur votre projet, ses dépendances, les scripts que vous pouvez exécuter, ainsi que des métadonnées pour le publier sur npm. Voici un breakdown de chaque partie de votre fichier :

json

Copier le code

{

"name": "my-datepicker-plugin",

"version": "1.0.0",

"description": "A custom React date picker component",

"main": "src/components/DatePicker.jsx",

"keywords": [

"react",

"datepicker",

"plugin"

],

"author": "Votre nom",

"license": "MIT",

"dependencies": {

"react": "^18.3.1",

"react-dom": "^18.3.1"

},

"devDependencies": {

"@vitejs/plugin-react": "^4.3.1",

"vite": "^5.4.1"

}

}

**1. Informations générales**

json

Copier le code

"name": "my-datepicker-plugin",

"version": "1.0.0",

"description": "A custom React date picker component",

"main": "src/components/DatePicker.jsx",

* **name** : Le nom de votre projet. Ce nom est important, car il est utilisé lorsque quelqu'un souhaite installer votre package via npm (par exemple, npm install my-datepicker-plugin). Ce nom doit être unique dans l'écosystème npm.
* **version** : La version de votre plugin. La gestion des versions est essentielle pour les mises à jour futures. Le schéma le plus courant est **SemVer** (Versionnage Sémantique), où :
  + Le premier chiffre (1) représente une **major version**. Cela signifie des modifications importantes ou des changements qui cassent la compatibilité.
  + Le deuxième chiffre (0) est la **minor version**, pour les nouvelles fonctionnalités qui ne cassent pas la compatibilité.
  + Le dernier chiffre (0) est le **patch**, qui correspond aux corrections de bugs sans changement dans les fonctionnalités.
* **description** : Une brève description de ce que fait le package. C'est utile pour les personnes qui recherchent des plugins sur npm.
* **main** : Le point d'entrée principal de votre package. C'est le fichier qui sera utilisé lorsqu'une personne importera votre package dans un projet. Ici, c'est **src/components/DatePicker.jsx**, car c'est le fichier du composant que vous voulez partager.

**2. Mots-clés (keywords)**

json

Copier le code

"keywords": [

"react",

"datepicker",

"plugin"

],

Les **keywords** (ou mots-clés) sont utilisés par npm pour aider les gens à trouver votre plugin dans l'annuaire. Si quelqu'un cherche un plugin avec des mots comme "React" ou "Datepicker", ces mots-clés augmenteront la visibilité de votre package.

**3. Auteur (author)**

json

Copier le code

"author": "Votre nom",

C'est ici que vous placez votre nom (ou votre organisation) en tant qu'auteur du plugin. Cela permet aux utilisateurs de voir qui a créé et maintient le projet.

**4. Licence (license)**

json

Copier le code

"license": "MIT",

La **licence** spécifie les droits légaux pour l'utilisation, la modification et la distribution de votre plugin. Ici, la licence **MIT** est une licence open-source très permissive qui permet à quiconque d'utiliser et de modifier votre code librement, tant qu'ils incluent la même licence dans leurs distributions.

**5. Dépendances (dependencies)**

json

Copier le code

"dependencies": {

"react": "^18.3.1",

"react-dom": "^18.3.1"

},

Les **dependencies** sont des bibliothèques externes dont votre projet a besoin pour fonctionner correctement. Lorsque quelqu'un installe votre package, npm installera également toutes les dépendances listées ici.

* **react** : React est nécessaire, car votre plugin est un composant React. La version indiquée est ^18.3.1, ce qui signifie que tout utilisateur ayant la version 18.3.1 ou supérieure (mais inférieure à la 19) peut utiliser votre plugin.
* **react-dom** : React DOM est aussi requis, car il gère la manipulation du DOM dans les projets React.

**6. Dépendances de développement (devDependencies)**

json

Copier le code

"devDependencies": {

"@vitejs/plugin-react": "^4.3.1",

"vite": "^5.4.1"

}

Les **devDependencies** sont des dépendances nécessaires uniquement pour développer et compiler votre projet. Elles ne sont pas installées lorsque quelqu'un utilise votre plugin. Dans votre cas, vous avez besoin de **Vite** pour le développement, et de son plugin **React** pour que Vite fonctionne bien avec React.

* **vite** : Vite est l'outil de développement rapide que vous utilisez pour développer ce projet.
* **@vitejs/plugin-react** : Ce plugin permet à Vite de prendre en charge les fichiers et les fonctionnalités spécifiques à React.

**Résumé des dépendances :**

* **dependencies** : Ce sont les bibliothèques qui seront installées avec votre plugin pour qu'il fonctionne correctement.
* **devDependencies** : Ce sont des outils que vous utilisez seulement pendant le développement, comme **Vite**, mais qui ne seront pas installés lorsque quelqu'un utilisera votre package.

**Conclusion :**

Le fichier package.json que vous avez préparé est une composante essentielle de votre plugin. Il décrit tout ce qui concerne votre projet, ses dépendances, et fournit les informations nécessaires pour que d'autres puissent utiliser et installer votre package. Une fois tout bien configuré, vous pouvez publier le plugin sur **npm**, comme nous l'avons vu dans l'étape suivante.

### **Étape 2 : Publication du plugin sur npm**

Voici les étapes détaillées pour publier un plugin sur **npm**, avec des explications à chaque étape. Nous partirons du principe que vous avez déjà un plugin fonctionnel avec **Vite** (ou un autre outil), et un fichier package.json bien configuré.

**1. Préparer le projet pour la publication**

Avant de publier, il est important de s'assurer que tout est en ordre :

* **Vérifiez le package.json :**
  + Le champ **name** est unique sur npm.
  + Le champ **version** respecte le versionnage sémantique (comme 1.0.0).
  + Le champ **main** pointe vers le fichier principal de votre plugin.
* **Nettoyez votre projet** :
  + Supprimez tout fichier inutile ou temporaire.
  + Assurez-vous que votre code est fonctionnel et bien testé.

Voici un exemple de package.json préparé pour la publication :

json

Copier le code

{

"name": "my-datepicker-plugin", // Nom unique du plugin

"version": "1.0.0", // Version initiale

"description": "A custom React date picker component", // Description du plugin

"main": "src/components/DatePicker.jsx", // Point d'entrée principal

"keywords": ["react", "datepicker", "plugin"], // Mots-clés pour aider à trouver votre plugin

"author": "Votre nom", // Votre nom ou nom d'organisation

"license": "MIT", // Licence open-source

"dependencies": {

"react": "^18.3.1",

"react-dom": "^18.3.1"

},

"devDependencies": {

"@vitejs/plugin-react": "^4.3.1",

"vite": "^5.4.1"

}

}

**2. Créer un compte npm (si ce n'est pas déjà fait)**

Si vous n'avez pas encore de compte npm, rendez-vous sur [**https://www.npmjs.com/**](https://www.npmjs.com/) et inscrivez-vous.

**3. Se connecter à npm via la ligne de commande**

Pour publier un package, vous devez d'abord vous authentifier sur npm via la ligne de commande.

Dans votre terminal, entrez la commande suivante pour vous connecter :

bash

Copier le code

npm login

Vous devrez fournir les informations suivantes :

* **Nom d'utilisateur npm**.
* **Mot de passe**.
* **Adresse e-mail**.

**Exemple :**

bash

Copier le code

$ npm login

Username: marco3866

Password: \*\*\*\*\*\*\*\*\*\*\*

Email: (this IS public) your.email@example.com

**4. Construire le projet**

Avant de publier, il faut s'assurer que votre projet est compilé et prêt à être utilisé.

Pour les projets React avec **Vite**, vous pouvez construire votre projet avec la commande suivante dans le répertoire racine de votre projet :

bash

Copier le code

npm run build

Cela générera un répertoire dist/ contenant le build de votre plugin. Ce sera le code qui sera distribué et installé lorsque quelqu'un utilisera votre plugin.

**5. Publier le plugin sur npm**

Maintenant que tout est prêt, vous pouvez publier votre plugin.

Voici la commande pour publier le projet :

bash

Copier le code

npm publish --access public

**Explication :**

* **npm publish** : Cette commande envoie le package à npm.
* **--access public** : Ce paramètre garantit que votre package est publié publiquement et accessible à tous.

**Note :** Si c'est votre première publication sur npm, vous devrez probablement confirmer votre adresse e-mail pour terminer la publication.

**Exemple :**

bash

Copier le code

npm publish --access public

+ my-datepicker-plugin@1.0.0

**6. Vérification de la publication**

Une fois le package publié, vous pouvez vérifier sa disponibilité sur le site de npm.

1. Accédez à [**https://www.npmjs.com/**](https://www.npmjs.com/).
2. Connectez-vous avec vos informations d'identification.
3. Sous l'onglet **Packages**, vous devriez voir votre plugin **my-datepicker-plugin**.

**7. Testez votre plugin après publication**

Pour vérifier que votre plugin fonctionne correctement, vous pouvez l'installer dans un autre projet.

**Étape 1 : Allez dans un autre projet React (ou créez-en un nouveau)**

* Si vous n'avez pas encore de projet React, vous pouvez en créer un avec la commande suivante :

bash

Copier le code

npx create-react-app my-new-project

**Étape 2 : Installez votre plugin dans ce projet**

Maintenant que votre plugin est publié, vous pouvez l'installer comme n'importe quel autre package npm.

bash

Copier le code

npm install my-datepicker-plugin

**Étape 3 : Utilisez votre plugin dans votre projet**

Exemple d'utilisation dans un fichier App.js :

javascript

Copier le code

import React, { useState } from 'react';

import DatePicker from 'my-datepicker-plugin'; // Importer votre plugin

function App() {

const [selectedDate, setSelectedDate] = useState(new Date());

return (

<div className="App">

<h1>Test du plugin DatePicker</h1>

<DatePicker

selectedDate={selectedDate}

onChange={setSelectedDate}

label="Select a date"

/>

</div>

);

}

export default App;

Si tout fonctionne bien, vous verrez votre composant **DatePicker** s'afficher correctement, et il sera fonctionnel dans votre projet.

**8. Gérer les versions et les mises à jour**

Lorsque vous apportez des modifications à votre plugin et que vous souhaitez les publier, vous devez :

1. **Mettre à jour la version dans package.json** : Si vous avez fait une mise à jour mineure (comme ajouter une fonctionnalité sans casser la compatibilité), mettez à jour la **minor version**. Exemple :

json

Copier le code

"version": "1.1.0"

1. **Re-publier le package** :

bash

Copier le code

npm publish

Cela mettra à jour votre package sur npm et les utilisateurs pourront installer la dernière version.

**Résumé des étapes pour publier un plugin sur npm :**

1. Créez le plugin avec un outil comme **Vite**.
2. Assurez-vous que le fichier package.json est correctement configuré.
3. Connectez-vous à **npm** via la commande npm login.
4. Construisez votre projet avec npm run build.
5. Publiez votre package avec npm publish --access public.
6. Testez votre plugin en l'installant dans un autre projet avec npm install <nom-du-package>.

Ces étapes couvrent le processus complet de la création à la publication et à l'utilisation de votre plugin sur **npm**. Cela vous permet de partager votre travail avec toute la communauté des développeurs.

**Étape 3 : Utilisation du plugin dans un projet existant**

Maintenant que votre plugin est publié sur **npm**, vous pouvez l'utiliser dans n'importe quel projet React en suivant ces étapes. Nous allons voir comment l'installer et l'utiliser dans votre projet existant.

**1. Installer le plugin dans le projet**

**Étape 1 : Accéder au projet existant**

Allez dans le répertoire de votre projet React existant où vous souhaitez utiliser votre plugin.

bash

Copier le code

cd /path/to/your/react-project

**Étape 2 : Installer le plugin**

Utilisez npm install pour ajouter votre plugin à votre projet. Remplacez <plugin-name> par le nom de votre package que vous avez publié.

bash

Copier le code

npm install my-datepicker-plugin

**Explication :**

* Cette commande ajoute votre plugin (qui est maintenant sur npm) aux **dépendances** de votre projet.
* Cela met à jour automatiquement votre fichier package.json pour inclure le plugin sous la section dependencies.

Voici à quoi peut ressembler votre fichier package.json après l'installation :

json

Copier le code

{

"name": "hrnet-react",

"version": "1.0.0",

"private": true,

"dependencies": {

"my-datepicker-plugin": "^1.0.0", // Le plugin que vous avez ajouté

"react": "^18.0.0",

"react-dom": "^18.0.0",

"react-scripts": "5.0.0"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

},

...

}

**2. Importer et utiliser le plugin dans votre projet**

Une fois le plugin installé, vous pouvez l'importer et l'utiliser dans vos composants.

Nous allons reprendre l'exemple du fichier **FormComponent** que vous avez partagé précédemment, où vous souhaitez intégrer le plugin **DatePicker** dans un formulaire.

**Exemple complet d'utilisation du plugin :**

javascript

Copier le code

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

// Importer votre plugin datepicker

import DatePicker from 'my-datepicker-plugin';

import SelectMenuComponent from '../../components/SelectMenuComponent/SelectMenuComponent';

import ConfirmationModal from '../../components/ConfirmationModal/ConfirmationModal';

import './FormComponent.css';

const FormComponent = ({ onSubmit }) => {

const [dateOfBirth, setDateOfBirth] = useState(new Date());

const [startDate, setStartDate] = useState(new Date());

const [departments, setDepartments] = useState([]);

const [states, setStates] = useState([]);

const [selectedDepartment, setSelectedDepartment] = useState('');

const [selectedState, setSelectedState] = useState('');

const [errors, setErrors] = useState({});

const [showErrorModal, setShowErrorModal] = useState(false); // État pour gérer la popup d'erreur

// Charger les départements

useEffect(() => {

fetch('/Data/departments.json')

.then((response) => response.json())

.then((data) => {

setDepartments(data);

setSelectedDepartment(data[0]);

});

}, []);

// Charger les états

useEffect(() => {

fetch('/Data/states.json')

.then((response) => response.json())

.then((data) => {

setStates(data);

setSelectedState(data[0].abbreviation);

});

}, []);

const validateForm = (formData) => {

const newErrors = {};

if (!formData.firstName) newErrors.firstName = 'First Name is required';

if (!formData.lastName) newErrors.lastName = 'Last Name is required';

if (!formData.street) newErrors.street = 'Street is required';

if (!formData.city) newErrors.city = 'City is required';

if (!formData.zipCode) newErrors.zipCode = 'Zip Code is required';

setErrors(newErrors);

return Object.keys(newErrors).length === 0;

};

const handleSubmit = (e) => {

e.preventDefault();

const employeeData = {

firstName: e.target['first-name'].value,

lastName: e.target['last-name'].value,

dateOfBirth,

startDate,

department: selectedDepartment,

street: e.target['street'].value,

city: e.target['city'].value,

state: selectedState,

zipCode: e.target['zip-code'].value

};

if (validateForm(employeeData)) {

onSubmit(employeeData); // Si le formulaire est valide, soumettre les données

} else {

setShowErrorModal(true); // Afficher la popup si le formulaire n'est pas valide

}

};

const closeModal = () => {

setShowErrorModal(false); // Fermer la popup d'erreur

};

return (

<>

<form onSubmit={handleSubmit}>

{/\* First Name \*/}

<label htmlFor="first-name">First Name</label>

<input type="text" id="first-name" />

{errors.firstName && <p className="error">{errors.firstName}</p>}

{/\* Last Name \*/}

<label htmlFor="last-name">Last Name</label>

<input type="text" id="last-name" />

{errors.lastName && <p className="error">{errors.lastName}</p>}

{/\* Date of Birth \*/}

<DatePicker

label="Date of Birth"

selectedDate={dateOfBirth}

onChange={setDateOfBirth}

/>

{/\* Start Date \*/}

<DatePicker

label="Start Date"

selectedDate={startDate}

onChange={setStartDate}

/>

{/\* Address Fields \*/}

<fieldset className="address">

<legend>Address</legend>

<label htmlFor="street">Street</label>

<input type="text" id="street" />

{errors.street && <p className="error">{errors.street}</p>}

<label htmlFor="city">City</label>

<input type="text" id="city" />

{errors.city && <p className="error">{errors.city}</p>}

<SelectMenuComponent

label="State"

options={states.map((state) => state.name)}

value={selectedState}

onChange={(e) => setSelectedState(e.target.value)}

/>

<label htmlFor="zip-code">Zip Code</label>

<input type="number" id="zip-code" />

{errors.zipCode && <p className="error">{errors.zipCode}</p>}

</fieldset>

{/\* Department \*/}

<SelectMenuComponent

label="Department"

options={departments}

value={selectedDepartment}

onChange={(e) => setSelectedDepartment(e.target.value)}

/>

<button type="submit">Save</button>

</form>

{showErrorModal && (

<ConfirmationModal

message="Certaines informations sont manquantes. Veuillez remplir tous les champs obligatoires."

onClose={closeModal}

/>

)}

</>

);

};

export default FormComponent;

**Explications du code :**

1. **Installation du plugin dans le projet :**
   * Le plugin **my-datepicker-plugin** est installé avec **npm install** et apparaît maintenant dans la liste des dépendances du projet.
2. **Importation du plugin :**
   * Dans le fichier FormComponent.js, nous importons le plugin en ajoutant cette ligne en haut du fichier :

javascript

Copier le code

import DatePicker from 'my-datepicker-plugin';

1. Cela permet d'utiliser le composant **DatePicker** dans le code.
2. **Utilisation du plugin :**
   * Nous remplaçons les anciens sélecteurs de date par le composant **DatePicker** du plugin.
   * Le plugin est utilisé deux fois :
     + Une fois pour la **Date of Birth**.
     + Une autre fois pour la **Start Date**.

**Exemple :**

javascript

Copier le code

<DatePicker

label="Date of Birth"

selectedDate={dateOfBirth}

onChange={setDateOfBirth}

/>

* + Nous passons **selectedDate** et **onChange** comme props, pour que le composant **DatePicker** fonctionne de manière contrôlée dans notre formulaire.

1. **Gestion des erreurs :**
   * Si des champs obligatoires sont manquants, une erreur est affichée sous les champs, et une popup de confirmation est montrée à l'utilisateur.

**3. Tester le plugin dans l'application**

Pour vous assurer que tout fonctionne correctement, démarrez l'application React :

bash

Copier le code

npm start

* **Assurez-vous que les composants DatePicker sont bien rendus** et que la sélection de date fonctionne comme prévu.
* **Vérifiez les interactions** : sélection de date, soumission de formulaire et gestion des erreurs.

Si tout fonctionne, cela signifie que votre plugin est bien intégré à votre projet !

**Résumé de l'étape 3 :**

1. **Installer le plugin dans le projet existant** avec npm install.
2. **Importer le plugin** dans le fichier de votre projet où vous souhaitez l'utiliser.
3. **Utiliser le plugin** en intégrant le composant dans votre code.
4. **Tester le fonctionnement** de l'application pour s'assurer que tout est correct.

Cela complète l'intégration et l'utilisation de votre plugin **DatePicker** dans un projet React existant.