Documentation du Stage : Modèles de Tâches

1. Introduction

Ce document décrit le processus de création, gestion et interaction avec les modèles de tâches dans le cadre de mon stage. Les modèles de tâches permettent de définir des éléments de travail spécifiques dans une application de gestion de projet. Les sections suivantes détaillent l'architecture des composants, l'intégration du contexte, et l'usage des fonctionnalités comme le Drag and Drop.

2. Code pour la gestion des modèles de tâches

2.1 Création du contexte

Le code suivant montre comment nous avons utilisé React et son context API pour gérer l'état des modèles de tâches à travers l'application. Le contexte est utilisé pour encapsuler les états relatifs aux modèles de tâches et permettre une gestion centralisée.

```jsx
import ScrollToError from "../utils/functions/utils/Scroll/ScrollToError"
import { useLocation } from "react-router-dom"
import { useEffect, useState } from "react"
import { createContext } from "react"

export const TaskModelContext = createContext()

export const TaskModelProvider = ({ children }) => {
    const location = useLocation()
    const [taskModel, setTaskModel] = useState(null)
    const [taskModels, setTaskModels] = useState([])
    const [errorsValidation, setErrorsValidation] = useState([])

    /**
     * UEF to scroll to first error in document
     */
    useEffect(() => {
        if (errorsValidation?.length) ScrollToError(errorsValidation)
    }, [errorsValidation])

    /**
     * UEF to reset errorsValidation
     */
    useEffect(() => {
        setErrorsValidation([])
    }, [location])

    return (
        <TaskModelContext.Provider
            value={{
                // States
                taskModel,
                setTaskModel,
                taskModels,
                setTaskModels,
                errorsValidation,
                setErrorsValidation,
            }}
        >
            {children}
        </TaskModelContext.Provider>
    )
}
```

3. Ajout de la route pour intégrer le contexte

Afin d'intégrer le contexte à la page des modèles de tâches, la route correspondante a été mise à jour pour englobler le composant TaskModels dans le provider TaskModelProvider. Cela permet de rendre le contexte accessible à tous les composants enfants dans cette route.

```
{
    path: "/:role/modèles-de-tâches",
    element: (
        <PrivateLayout>
            <TaskModelProvider>
                <TaskModels />
            </TaskModelProvider>
        </PrivateLayout>
    ),
    errorElement: <NotFoundPage layoutType="private" />,
},
```

4. Page principale pour la gestion des modèles de tâches

La page principale pour la gestion des modèles de tâches a été créée avec le composant TaskModels. Ce composant inclut une liste de missions que l'utilisateur peut réorganiser grâce à une fonctionnalité Drag and Drop.

5. Composants pour le Drag and Drop

Le Drag and Drop est géré à travers trois composants principaux: Draggable, Droppable et DragAndDrop. Ces composants permettent de rendre les éléments réorganisables et de gérer les zones de dépôt pour réorganiser les missions.

5.1 Draggable

Le composant Draggable représente un élément déplaçable, avec la possibilité de spécifier des contraintes et des comportements de déplacement.

```jsx
import { useDraggable } from "@dnd-kit/core"
/**
 * Function to render the draggable element
 * @param {String} id - The id of the draggable element
 * @param {JSX.Element} children - The children of the draggable element
 * @param {String} itemClassName - The class name of the element
 * @param {Function} getStyleDraggable - Function to get the style of the draggable element
 * @returns {JSX.Element}
 */
export default function Draggable({ id, children, itemClassName, getStyleDraggable }) {
    const { attributes, listeners, setNodeRef, transform } = useDraggable({
        id: id,
        activationConstraint: {
            tolerance: 5,
            delay: 150,
        },
    })

    return (
        <div ref={setNodeRef} style={getStyleDraggable(transform)} className={itemClassName}>
            <div {...listeners} {...attributes} className="drag-handle">
                ⠿
            </div>
            <div>{children}</div>
        </div>
    )
}
```

5.2 Droppable

Le composant Droppable représente une zone où les éléments déplaçables peuvent être déposés. Il permet de gérer le style et les comportements visuels lorsque l'élément est déplacé dans la zone de dépôt.

```jsx
import { useDroppable } from "@dnd-kit/core"

/**
 * Function to render a droppable element
 * @param {String} id - The id of the droppable element
 * @param {JSX.Element} children - The children of the droppable element
 * @param {Function} getStyleDroppable - Function to get the style of the droppable element
 * @returns {JSX.Element}
 */
export default function Droppable({ id, children, getStyleDroppable }) {
    const { setNodeRef, isOver } = useDroppable({ id: id })

    return (
        <div ref={setNodeRef} style={getStyleDroppable(isOver)} className="droppable-wrapper">
            {children}
        </div>
    )
}
```

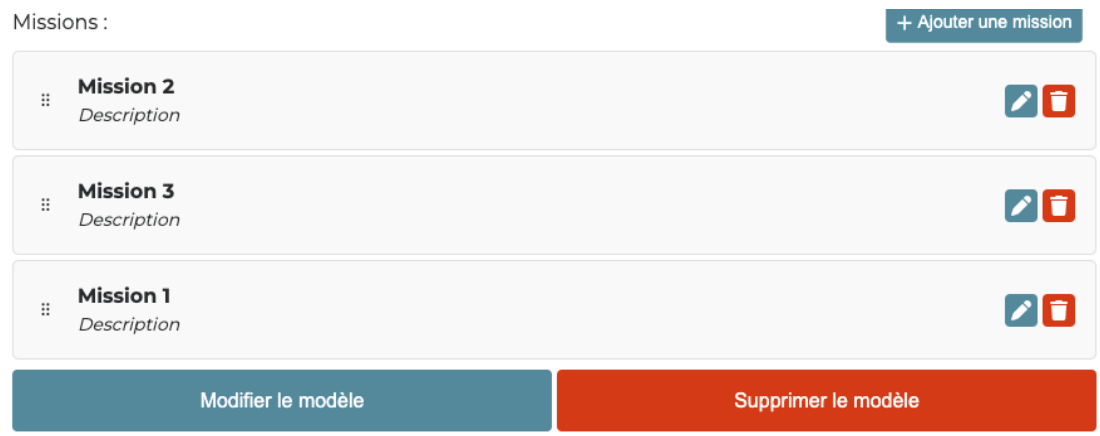
5.3 DragAndDrop

Le composant DragAndDrop combine les composants Draggable et Droppable pour gérer l'ensemble de l'interaction de déplacement des éléments.

```jsx
import { closestCenter, DndContext } from "@dnd-kit/core"
import Droppable from "./Droppable"
import Draggable from "./Draggable"
/**
 * Function to render the drag and drop component
 * @param {Array} items - The items to drag and drop
 * @param {Function} renderItem - Function to render the item
 * @param {String} itemClassName - The class name of the item
 * @param {Function} onDragEnd - Function to call when the drag ends
 * @param {Function} getStyleDroppable - Function to get the style of the droppable element
 * @returns {JSX.Element}
 */
export default function DragAndDrop({ items, renderItem, itemClassName, onDragEnd, getStyleDroppable, getStyleDraggable }) {
    return (
        <DndContext collisionDetection={closestCenter} onDragEnd={onDragEnd}>
            <div>
                {items.map((item, index) => (
                    <Droppable key={item.id} id={item.id} getStyleDroppable={getStyleDroppable}>
                        <Draggable id={item.id} itemClassName={itemClassName} getStyleDraggable={getStyleDraggable}>
                            {renderItem(item, index)}
                        </Draggable>
                    </Droppable>
                ))}
            </div>
        </DndContext>
    )
}
```

6. Rendu sur la page

Le rendu sur la page affiche la liste des missions, avec des boutons permettant d'afficher les détails de chaque mission. Les utilisateurs peuvent réorganiser les missions grâce à l'interface Drag and Drop, et chaque mission est décrite par un titre et une description.



7. Liste des missions

Voici un aperçu de l'affichage de base de la liste des missions dans l'application, avec les missions et leurs boutons d'interaction.

Insérer l'image ici : [Image 9] - Liste des missions

9. Conclusion

Ce projet démontre la gestion des modèles de tâches dans une application utilisant React. L'intégration des fonctionnalités de Drag and Drop a permis d'offrir une expérience utilisateur fluide pour la réorganisation des missions. Ce travail de développement a été réalisé dans le cadre de mon stage.