WorkTime

[App.jsx 2](#_Toc167982870)

[Login.jsx 2](#_Toc167982871)

[ApiData.jsx 3](#_Toc167982872)

[Rendu 3](#_Toc167982873)

[GoogleLogin.jsx 4](#_Toc167982874)

[store.js 5](#_Toc167982875)

[Dashboard.jsx 7](#_Toc167982876)

[Menu.jsx 8](#_Toc167982877)

[Form.jsx 9](#_Toc167982878)

[DisplayElements.jsx 10](#_Toc167982879)

# App.jsx

La fonction App est le composant principal de l'application.

Il utilise <Routes> pour définir différentes routes dans l'application.

Chaque <Route> correspond à une URL spécifique et rend le composant associé :

* + / rend le composant Login.
  + /dashboard rend le composant Dashboard.
  + /fiche-de-paie rend le composant FicheDePaie.
  + /dashboard/facture rend le composant Facture.
  + \* rend le composant Page404 pour toutes les autres URL non définies (page d'erreur 404).

# Login.jsx

Le composant Login retourne un JSX qui définit la structure et le contenu de la page de connexion.

**Formulaire de connexion** :

* Contient des champs pour le nom, le prénom, l'email et le mot de passe.
* Un composant ApiData est inclus (probablement pour récupérer ou gérer des données via une API).
* Un bouton pour soumettre le formulaire.
* Un lien pour réinitialiser le mot de passe.

**Connexion via Google** : Intégré via le composant GoogleLogin.

# ApiData.jsx

#### Importations

* **useOutsideClick** : Un hook personnalisé pour détecter les clics en dehors d'un élément.

#### Composant ApiData

Le composant ApiData gère les données des entreprises et fournit une fonctionnalité de recherche avec un menu déroulant.

#### États

* lien : URL de l'API.
* entreprise : Données des entreprises.
* nomEntreprises : Liste des noms des entreprises.
* option : Option sélectionnée dans le menu déroulant.
* searchTerm : Terme de recherche pour filtrer les options.
* showDropdown : Booléen pour afficher ou masquer le menu déroulant.

#### Effets

* **fetchLien** : Récupère l'URL de l'API et la stocke dans lien.
* **fetchEntreprise** : Utilise l'URL pour récupérer les données des entreprises et les stocke dans entreprise.
* **Tri et extraction des noms** : Trie les entreprises par leur nom et extrait les noms pour nomEntreprises.

#### Fonctionnalités

* **handleOptionClick** : Met à jour l'option sélectionnée et cache le menu déroulant.
* **useOutsideClick** : Cache le menu déroulant si un clic est détecté en dehors du menu.

#### Recherche et Affichage

* Filtre les options en fonction de searchTerm.
* Affiche les options filtrées dans un menu déroulant.

### Rendu

* **Input** : Champ de saisie pour la recherche d'entreprise.
* **Dropdown** : Contenu du menu déroulant affichant les options filtrées.

# GoogleLogin.jsx

#### Importations

* **useSelector** de react-redux : Pour accéder à l'état Redux.
* **setUser** du fichier store : Une action Redux pour définir l'utilisateur connecté.

#### Composant GoogleLogin

Le composant GoogleLogin gère la connexion via Google et redirige l'utilisateur vers le tableau de bord après une connexion réussie.

#### États et Actions

* **dispatch** : Utilisé pour envoyer des actions à Redux.
* **navigate** : Utilisé pour rediriger l'utilisateur vers une autre route.
* **firebaseApp** : Récupère l'état de la configuration Firebase à partir de Redux.

#### Fonctionnalités

* **handleGoogleLogin** : Gère la connexion via Google.
  + Utilise getAuth pour obtenir l'instance d'authentification.
  + Utilise GoogleAuthProvider pour configurer le fournisseur de connexion Google.
  + Utilise signInWithPopup pour ouvrir une fenêtre de connexion.
  + Si la connexion est réussie, l'utilisateur est stocké dans l'état Redux avec setUser, et l'utilisateur est redirigé vers /dashboard.
  + En cas d'erreur, l'erreur est loguée dans la console.

#### Rendu

* **Bouton de connexion Google** : Un bouton stylisé pour initier la connexion via Google. Lorsqu'il est cliqué, il appelle handleGoogleLogin.
* **SVG** : Une icône Google stylisée incluse dans le bouton.

# store.js

#### Importations

* **Redux Toolkit** (configureStore, createSlice, createAsyncThunk) : Pour configurer et gérer l'état global de l'application.
* **firebaseApp** : La configuration Firebase utilisée pour l'authentification et d'autres services.

#### Slices

##### userSlice

* **State Initial** : Contient userData (données de l'utilisateur), loading (chargement en cours), et error (erreur éventuelle).
* **Reducers** :
  + setUser : Définit les données de l'utilisateur (userData).
  + clearUser : Efface les données de l'utilisateur.
* **ExtraReducers** : Gère les états pending, fulfilled et rejected de fetchUserData pour les appels asynchrones.

##### firebaseSlice

* **State Initial** : Initialise l'état avec firebaseApp.
* **Reducers** :
  + setFirebase : Met à jour l'état de la configuration Firebase.

##### clientsSlice

* **State Initial** : Initialise avec une liste vide.
* **Reducers** :
  + setClients : Met à jour l'état avec les clients d'une entreprise spécifique.

##### categoriesSlice

* **State Initial** : Initialise avec une liste vide.
* **Reducers** :
  + setCategories : Met à jour l'état avec les catégories de prestations d'une entreprise spécifique.

#### Thunk Asynchrone

* **fetchUserData** : Une fonction asynchrone pour récupérer les données utilisateur à partir d'un userId.
  + En cas de succès, retourne les données de l'utilisateur.
  + En cas d'erreur, retourne une valeur de rejet avec un message d'erreur.

#### Exportations des Actions

* setUser, clearUser, setFirebase, setClients, setCategories : Actions pour mettre à jour les états respectifs.
* fetchUserData : Thunk pour récupérer les données utilisateur.

#### Configuration du Store

* Utilise configureStore pour configurer le store Redux avec les reducers des différents slices.
* **Middleware** : Personnalisation du middleware pour ignorer les vérifications de sérialisation pour certaines actions et chemins spécifiques.

# Dashboard.jsx

#### Importations

* **form** : Pour définir l’état d’ouverture et de fermeture du formulaire d’ajout de prestation.

#### Composant

Le composant Dashboard représente la page principale de l’application où les utilisateurs connectés peuvent interagir avec différentes fonctionnalités.

*Fonctionnalités*

**Récupération des données utilisateur** : Une fonction asynchrone fetchData est définie à l'intérieur de useEffect. Cette fonction est responsable de récupérer les données utilisateur depuis Firebase et de les mettre à jour dans le store Redux. Elle utilise onAuthStateChanged pour surveiller les changements d'état d'authentification et réagit en conséquence. Si un utilisateur est connecté, les données utilisateur sont récupérées depuis Firebase et mises à jour dans le store Redux à l'aide des actions setUser et fetchUserData. Si aucun utilisateur n'est connecté, l'utilisateur est redirigé vers la page d'accueil.

**Mise à jour des informations Firebase dans le store Redux** : Après la récupération des données utilisateur, les informations Firebase sont mises à jour dans le store Redux à l'aide de dispatch(setFirebase(firebaseApp)).

# Menu.jsx

#### Importations

* **const [showDropdown, setShowDropdown] = useState(false);** : Ce state local (showDropdown) est utilisé pour contrôler l'affichage du menu déroulant. Initialement, le menu déroulant est caché (false).

#### Composant

Le composant Menu représente le menu avec les différents boutons.

*Fonctionnalités*

**const ref = useOutsideClick(() => { setShowDropdown(false); });** : Cette ligne utilise le custom hook useOutsideClick pour détecter les clics à l'extérieur du composant. Lorsqu'un clic est détecté en dehors du composant, la fonction setShowDropdown est appelée pour cacher le menu déroulant.

**const toggleDropdown = () => { setShowDropdown((prevState) => !prevState); };** : Cette fonction toggleDropdown est appelée lors du clic sur le bouton du menu. Elle inverse la valeur du state showDropdown, permettant ainsi de basculer l'affichage du menu déroulant entre visible et caché.

**const handleMenuClick = () => { toggleDropdown(); onClose(); };** : Cette fonction handleMenuClick est appelée lors du clic sur le bouton du menu. Elle appelle d'abord toggleDropdown pour afficher ou cacher le menu déroulant, puis onClose pour fermer d'autres éléments du menu (par exemple, un formulaire modal).

# Form.jsx

Ce composant Form représente un formulaire permettant à l'utilisateur d'ajouter une nouvelle entrée de données, probablement une prestation de travail. Voici une explication détaillée de chaque partie du code :

**Initialisation des états et constantes :**

**initialPrestationState** : Cette constante initialise l'état initial du formulaire de prestation de travail avec des valeurs par défaut.

**Effets secondaires avec useEffect** :

Deux effets secondaires sont utilisés pour récupérer les clients et les catégories de prestations à partir de Firebase et les mettre à jour dans le store Redux. Ces effets se déclenchent lorsque l'ID de l'entreprise de l'utilisateur change, et lors du montage initial du composant.

**Gestion des événements et validation :**

Les fonctions **handleDateChange** et **handleChange** sont utilisées pour mettre à jour l'état du formulaire lorsque l'utilisateur saisit des données.

La fonction **validateForm** est utilisée pour vérifier si toutes les données du formulaire sont remplies correctement. Si la validation échoue, un message d'erreur approprié est affiché.

La fonction **handleSubmit** est appelée lors de la soumission du formulaire. Elle valide d'abord le formulaire, puis calcule la durée totale de la prestation avant de mettre à jour les données dans la base de données Firebase.

**event.preventDefault();** : annule le comportement à l’événement de soumission de formulaire, pour ne pas recharger la page.

# Utils.jsx

Ces fonctions utilitaires sont utilisées dans le composant DisplayElements et d’autres pour effectuer divers calculs et traitements sur les données de prestation de travail.

**calculateTimeDifference(startTime, endTime)**

Cette fonction calcule la différence de temps entre une heure de début et une heure de fin au format "HH". Elle retourne un objet contenant le nombre d'heures et de minutes de différence.

**groupByMonth(elements)**

Cette fonction prend une liste d'éléments de prestation de travail et les regroupe par mois. Elle retourne un objet où chaque clé est le mois (au format "mois année") et la valeur est un tableau d'éléments de prestation pour ce mois.

**totalHoursMinutes(groupedElements)**

Cette fonction prend les éléments de prestation groupés par mois et calcule le total des heures et des minutes travaillées pour chaque mois. Elle retourne un objet où chaque clé est le mois et la valeur est un objet contenant le total des heures et des minutes.

**sortedMonths(groupedElements)**

Cette fonction prend les éléments de prestation groupés par mois et les trie dans l'ordre décroissant des mois. Elle retourne un tableau contenant les noms des mois triés.

**compareMonthToPrevious(groupedElements)**

Cette fonction compare le temps de travail de chaque mois par rapport au mois précédent et calcule le pourcentage de différence. Elle retourne un objet où chaque clé est le mois et la valeur est le pourcentage de différence par rapport au mois précédent. Si aucun mois précédent n'existe, la valeur est null.

# DisplayElements.jsx

Le composant DisplayElements est responsable de l'affichage des éléments de prestation de travail dans l'interface utilisateur.

**Utilisation de Redux et Firebase**

* Utilisation de useSelector pour extraire les données de l'utilisateur et de Firebase du store Redux.
* Utilisation de useDispatch pour envoyer des actions Redux.
* Dispatch de l'action fetchUserData pour récupérer les données de l'utilisateur lorsqu'il est disponible.
* Mise à jour des prestations en temps réel à partir de Firebase grâce à onValue.
* Mise à jour des clients à partir des filtres grâce à setClients.

**Fonctionnalités**

Les useEffect

* Ajouter le uid du user dans le store
* Ajout de prestations en temps réel et ajout des données user en useState
* Ajout des clients selon le retour des filtres

applyFilters

* Si le rendu des filtres est null, affiche tout
* Si non afficher selon les valeurs des variables que le filtre rend

Fontions de Utils.jsx

Utilisations des différentes fonctions de Utils.jsx

ficheDePaie

Fonction de préparation à la page FicheDePaie :

* Extraction des données du user
* Extraction du nom de l’entreprise du user
* Ajout de toute ces données dans un objet
* Naviguer sur la page /fiche-de-paie avec les donnes préparées

Fonction de préparation à la page Facture

* Navigue sur la page facture avec les prestations et les mois

pourcentCardDesign

* Utilisation d’un package de design de carte statistique
* Affichage des pourcentages des heures comparé au mois dernier

toogleShowAddForm

* Affichage d’un formulaire d’édition avec les données d’une prestation

closeModal

* Fermeture du formulaire

# SortInput.jsx

Ce composant SortInput est une interface utilisateur qui permet à l'utilisateur de filtrer les éléments de prestation de travail en fonction de plusieurs critères tels que la catégorie, le client, le mois et la date.

**Fonctionnalités**

Les useEffect

* Ajouter les clients par entreprise dans le store
* Ajouter les catégories par entreprise dans le store

handleChange

* Agit sur le changement de chaque input
* Ajout la valeur et la clef du champs dans prestations

handleSubmit

* Agit sur le click du bouton « Envoyer »
* Ajout des prestations dans la variable filters

handleDateChange

* Agit sur le champ date
* Ajout la valeur de la date dans la variable prestations

handleResetFilters

* Agit sur le clique du bouton reset
* Met à nul tous les champs, donc la variables filters