Gérer l'état de son application avec Redux

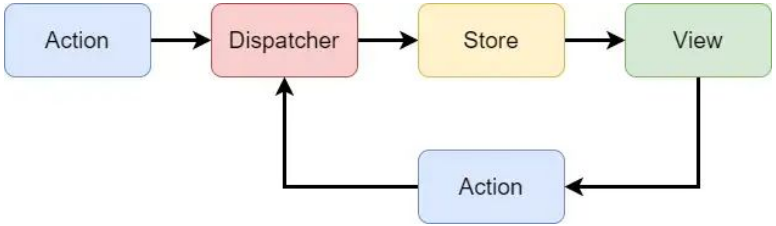
Redux

Présentation

Redux :

- Est une librairie de gestion de l'état de l'application
- Inspiré de l'architecture Flux
- Utilise des actions et des reducers pour modifier l'état de l'application

Flux :



Présentation

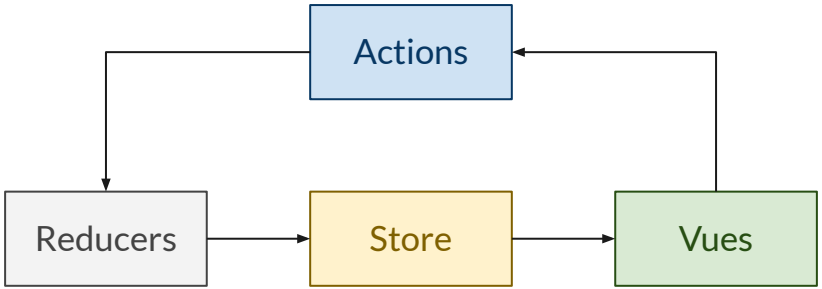
Redux est composé de plusieurs concepts :

`Actions` : Elles représentent des événements entraînant une modification de l'état de l'application

`Reducers` : Ce sont des *fonctions pures* qui sont appelées à chaque fois qu'une nouvelle action se produit. Elles prennent en paramètre l'état de l'application et l'action levée, et retournent un nouvel état.

`Store` : Il n'y en a qu'un par application. C'est lui qui stocke l'état de l'application, et qui transmet les actions levées aux reducers afin de modifier son état.

Présentation



Présentation

Installation :

```
$ npm install @reduxjs/toolkit react-redux
```

Le store 1/4

Exemple de création du store avec TypeScript :

```
import { configureStore } from '@reduxjs/toolkit';
import { TypedUseSelectorHook, useDispatch, useSelector } from 'react-redux';

const store = configureStore({
  reducer: {
    first: firstReducer,
    second: secondReducer
  }
});

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;

export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector;
export const useAppDispatch = () => useDispatch<AppDispatch>();

export default store;
```

Le store 2/4

```
const store = configureStore({
  reducer: {
    first: firstReducer,
    second: secondReducer
  }
});
```

`configureStore` : Crée le store en lui passant la liste des **reducers** de l'application.

Le store 3/4

```
export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;
```

Création d'alias de types TypeScript afin de faciliter l'utilisation :

- de l'état de l'application,
- et de la fonction dispatch permettant de transmettre les actions levées aux reducers

Le store 4/4

```
export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector;
export const useAppDispatch = () => useDispatch<AppDispatch>();
```

`useAppSelector` : Hook personnalisé basé sur le hook **`useSelector`** permettant de récupérer l'état de l'application dès que nécessaire dans les différents composants React.

`useAppDispatch` : Hook personnalisé basé sur le hook **`useDispatch`** permettant de transmettre aux reducers les actions levées.

Utilisation du store

Il est ensuite nécessaire de renseigner le store à la racine du projet :

```
import store from './store';
import { Provider } from 'react-redux';

root.render(
  <Provider store={store}>
    <App />
  </Provider>
);
```

Gérer l'état de son application avec Redux

Les reducers 1/5

```ts
import { createSlice, PayloadAction } from '@reduxjs/toolkit';

interface State { value: number }
const initialState: State = { value: 0 };

export const counterSlice = createSlice({
  name: 'counter',
  initialState: initialState,
  reducers: {
    increment: (state) => ({ value: state.value + 1 }),
    decrement: (state) => ({ value: state.value - 1 }),
    incrementByAmount: (state, action: PayloadAction<number>) => ({
      ...state,
      value: state.value + action.payload
    }),
  },
});

export const { increment, decrement, incrementByAmount } = counterSlice.actions;
export default counterSlice.reducer;
```

Les reducers 2/5

```
interface State {
  value: number
}

const initialState: State = {
  value: 0
};
```

Création de la structure de données représentant l'état de l'application, ainsi que d'une variable contenant l'état initial.

Les reducers 3/5

```
export const counterSlice = createSlice({
  name: 'counter',
  initialState: initialState,
  ...
```

`createSlice` : Fonction utilitaire permettant de créer un reducer (un "morceau" du store) avec un nom et un état initial.

Les reducers 4/5

```
  ...
  reducers: {
    increment: (state) => ({ value: state.value + 1 }), // Simple si le state ne possède qu'une seule valeur
    decrement: (state) => ({ value: state.value - 1 }),
    incrementByAmount: (state, action: PayloadAction<number>) => ({
      ...state,                                 // Utilisation du spread operator au cas où le state possèderait plusieurs valeurs
      value: state.value + action.payload       // et l'on ne souhaite en modifier qu'une partie
    }),
  },
});
```

Défini la liste des fonctions **reducers** prenant en paramètre l'état actuel et l'action levée, et retournant un nouvel état.

Les reducers 5/5

```
export const { increment, decrement, incrementByAmount } = counterSlice.actions;

export default counterSlice.reducer;
```

Exporte les actions et le reducer créés.

Récupération du state et utilisation d'actions

Dans les composants React :

```
import { useAppDispatch, useAppSelector } from './store';
import { increment, incrementByAmount } from './counterReducer';

const Counter = () => {
  const dispatch = useAppDispatch();
  const { value } = useAppSelector((state) => state.counter);

  return (
    <>
      Counter: {value} <br/>

      <button onClick={() => dispatch(increment())}>Increment !</button> <br/>

      <button onClick={() => dispatch(incrementByAmount(10))}>Increment by 10 !</button>
    </>
  );
};
```

Actions asynchrones

Redux thunk

Certaines actions impliquent des traitements longs devant être traités de manière asynchrone (comme par exemple récupérer des données d’une API REST).

Redux fournit la possibilité de les gérer grâce aux "**thunks**" et à la fonction utilitaire `createAsyncThunk`.

Création d'une async thunk 1/2

```
import { createAsyncThunk, createSlice, PayloadAction } from '@reduxjs/toolkit';

interface State {
  activities: string[],
  fetchingActivity: boolean
}

const initialState: State = { activities: [], fetchingActivity: false };

export const fetchActivity = createAsyncThunk(
  'tasks/fetchActivity',
  () => axios.get('https://www.boredapi.com/api/activity')
      .then((res) => res.data.activity)
);

const addActivity = (state: State, action: PayloadAction<string>): State => ({
  ...state,
  activities: state.activities.concat([action.payload])
});
```

Création d'une async thunk 2/2

```
export const activitiesSlice = createSlice({
  name: 'activities',
  initialState: initialState,
  reducers: {
    addActivity: addActivity // Il est possible d'extraire la fonction de modification du state dans une variable
  },
  extraReducers: (builder) => {
    builder.addCase(fetchActivity.pending, (state) => ({
      ...state,
      fetchingActivity: true
    }));

    builder.addCase(fetchActivity.fulfilled, addActivity);
  }
});

export const { addActivity } = activitiesSlice.actions;
export default activitiesSlice.reducer;
```

Création d'une async thunk 1/2

```
export const fetchActivity = createAsyncThunk(
  'tasks/fetchActivity',
  () => axios.get('https://www.boredapi.com/api/activity')
      .then((res) => res.data.activity)
);
```

`createAsyncThunk` : Crée une action asynchrone en lui donnant un identifiant, ainsi qu'une fonction représentant le traitement à effectuer et devant retourner une **Promise**.

Création d'une async thunk 2/2

```
extraReducers: (builder) => {
  builder.addCase(fetchActivity.pending, (state) => ({
    ...state,
    fetchingActivity: true
  }));

  builder.addCase(fetchActivity.fulfilled, addActivity); // Il est possible d'extraire la fonction de modification du state
}                                                        // dans une variable
```

`extraReducers` : Permet de définir des reducers supplémentaires basés par exemple sur des async thunks (ou action asynchrone).

Pour chaque action asynchrone, il est possible de gérer le cycle de vie de la Promise, et de modifier le state en conséquence.

Utilisation d'une action asynchrone

Dans les composants React :

```
import { useAppDispatch, useAppSelector } from './store';
import { fetchActivity } from './activitiesReducer';

const DisplayActivity = () => {
  const dispatch = useAppDispatch();
  const { activities } = useAppSelector((state) => state.activities);

  useEffect(() => {
    dispatch(fetchActivity());
  }, []);

  return (...);
};
```

Exercice

Exercice 1/2

Utilisez maintenant Redux pour stocker l'état de votre application :

- Créez un reducer charactersReducer
- Créez les actions :
    - addCharacter
    - deleteCharacter
- Créez également l'action asynchrone fetchCharacters qui récupèrera les personnages depuis l'API https://thronesapi.com

Exercice 2/2

Dans la liste des personnages, ajoutez la possibilité de supprimer un personnage en cliquant sur un bouton/une icône de suppression.