PHASE 3

Redux

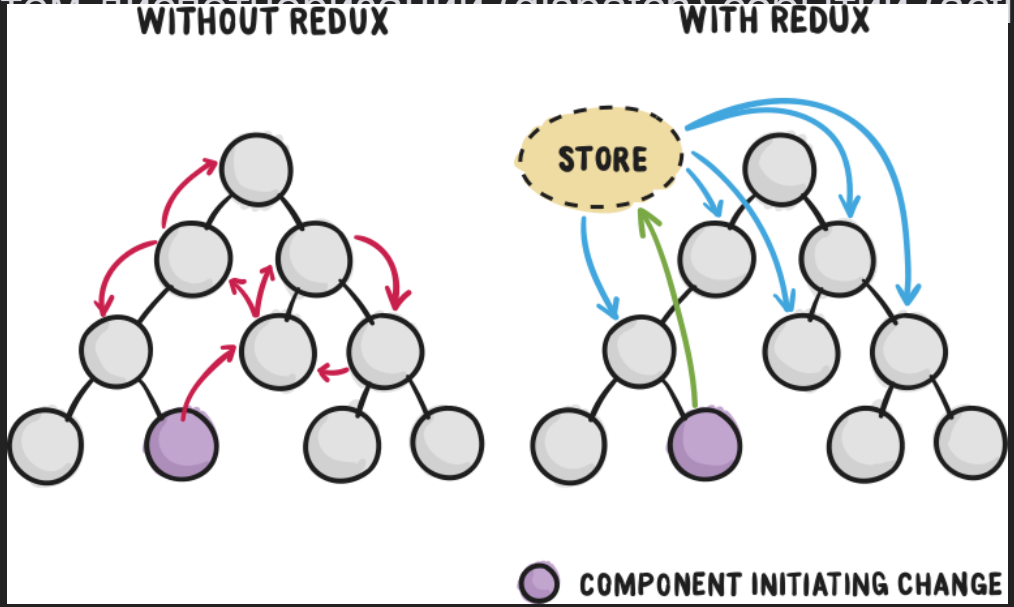
Redux

ELBRUS/BOOTCAMP

Что такое Redux

Redux

Redux - это библиотека управления состоянием для приложений на JavaScript. Она позволяет хранить состояние приложения в одном месте (store) и обновлять его путем диспетчеризации (dispatch) событий (actions).

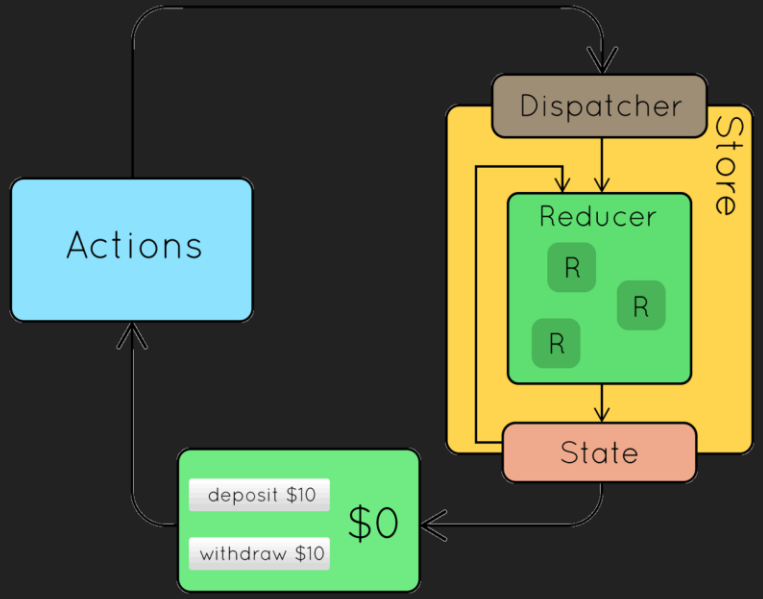


Redux - store

Redux store - это объект, который хранит состояние всего приложения в Redux. Он является центральным хранилищем состояния для всех компонентов приложения, и изменение состояния осуществляется только через него.

Store содержит состояние приложения в виде объекта, а также предоставляет несколько методов, которые позволяют компонентам взаимодействовать с этим состоянием. Все изменения состояния происходят путем диспетчеризации **(dispatch)** действий **(actions)**, которые описывают, какие изменения должны произойти в состоянии.

Store также содержит редьюсеры **(reducers)**, которые определяют, как изменения состояния будут происходить в ответ на действия. Эти редьюсеры объединяются в корневой редьюсер **(root reducer)**, который передается в функцию **configureStore()** при создании объекта **store**.



Redux actions

Actions (действия) - это объекты, которые описывают, что произошло в приложении и какие изменения необходимо внести в состояние приложения в Redux. Они содержат минимум два свойства: type и, обычно, payload.

Свойство **type** - это строка, которая указывает на тип действия, которое произошло. Тип действия может быть произвольным, но обычно он записывается в верхнем регистре и описывает, что произошло в приложении. Например, ADD_TODO или INCREMENT_COUNTER.

Свойство payload (нагрузка) - это данные, которые передаются вместе с действием и содержат информацию о том, какие данные должны быть изменены в состоянии. Это свойство не является обязательным, но может быть очень полезным для передачи данных между компонентами и редьюсерами.

Пример Action:

```
{
    type: 'ADD_TODO',
    payload: {
    id: 1,
    text: 'Купить молоко',
    completed: false
    }
}
```

Redux actions creators

В примере actions поле payload принимает новый элемент списка дел, а это значит что нам придется каждый раз описывать объект с неизменяемым полем `type:'ADD_TODO'`

и динамическим полем `payload`, чтобы избавиться от дублирования кода actions записывают в виде функций (actions creators):

```
const newToDo = {
  id: 1,
  text: 'Купить молоко',
  completed: false
  }

const newTodoAction = (newToDo) => ({
  type: 'ADD_TODO',
  payload: newToDo
})
```

Redux dispatch

Dispatch (диспетчер) - это метод объекта Redux store, который используется для отправки действий (actions) в редьюсеры (reducers) для обновления состояния приложения. Он позволяет компонентам взаимодействовать с состоянием приложения, не зная, какие редьюсеры будут обновлять это состояние. Поэтому поле type должно быть уникальным.

Метод dispatch() принимает в качестве аргумента объект действия (action), который описывает, какие изменения необходимо внести в состояние.

```
export default function SimpsonsPage(): JSX.Element {
 const dispatch = useAppDispatch();
 const data = useAppSelector(state => state.persons)

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

const clearHandler = (): void => {
 dispatch(clearPersonsAction())
}
 return (
   <Row>
   <CountForm/>
   <Button onClick={clearHandler}>Clear</Button>
     {data.map(el => <PersonCard key={el.id} person={el}/>)}
   </Row>
 );
}
```

Redux reducers

Reducers (редьюсеры) в Redux - это чистые функции, которые принимают предыдущее состояние приложения и объект действия (action), и возвращают новое состояние приложения на основе переданного действия. Reducer определяет, как должно изменяться состояние приложения в ответ на действия.

Reducers должны быть чистыми функциями, то есть они не должны изменять предыдущее состояние приложения, а всегда возвращать новое состояние. Это позволяет Redux контролировать состояние приложения и предотвращает мутирование состояния, что может привести к ошибкам в приложении.

Пример редьюсера управляющего состоянием списка дел:

```
function todoReducer(state = [], action) {
    switch (action.type) {
      case 'ADD_TODO':
        return [...state, action.payload];
      case 'DELETE_TODO':
        return state.filter(el => el.id !== action.payload)
      default:
        return state;
    }
  }
```

Redux ToolKit & Typescript

Установка зависимостей

Redux Toolkit - это официальный пакет инструментов, который предоставляет более простой и быстрый способ написания кода на Redux. Он упрощает настройку Redux и улучшает производительность, предоставляя различные встроенные функции и утилиты.

Библиотека react-redux - это официальное расширение библиотеки Redux, которое предоставляет интеграцию Redux с библиотекой React для упрощения управления состоянием в приложениях на React.

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

Добавление Redux в приложение

Для того что бы подключить Redux в наше приложение необходимо:

1. Создать store
2. Создать types для store, dispatch и асинхронных actions
3. Создать кастомные типизированные хуки useAppDispatch, useAppSelector
4. Создать types для actions
5. Создать actions creators, которые будем отправлять в reducer
6. Создать reducer который будет управлять состоянием
7. Добавить reducer в store
8. Обернуть приложение в Provider

Создание store

./client/redux/store.ts

```
import { configureStore } from '@reduxjs/toolkit';

const store = configureStore({
  reducer: {}
});

export default store;
```

Types для для store

./client/types/reduxTypes.ts

```ts
import type { AnyAction, ThunkAction } from '@reduxjs/toolkit';
import type store from '../redux/store';

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;
export type AppThunk<ReturnType = void> = ThunkAction<ReturnType, RootState, unknown, AnyAction>;
```

Redux custom hooks

./client/features/reduxHooks.ts

```ts
import type { TypedUseSelectorHook } from 'react-redux';
import { useSelector, useDispatch } from 'react-redux';
import type { AppDispatch, RootState } from '../types/reduxTypes';

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

Actions Types

./client/types/counterActionsTypes.ts

```ts
export const INCREMENT_COUNTER = 'INCREMENT_COUNTER';
export type IncrementActionType = {
  type: typeof INCREMENT_COUNTER;
};


export const DECREMENT_COUNTER = 'DECREMENT_COUNTER';
export type DecrementActionType = {
  type: typeof DECREMENT_COUNTER;
};


export type CounterActionsTypes = IncrementActionType | DecrementActionType
```

Actions

```
import type {
  DecrementActionType,
  IncrementActionType,
} from '../../types/CounterTypes';
import { DECREMENT_COUNTER, INCREMENT_COUNTER } from '../../types/CounterTypes';

export const incrementCounterAction = (): IncrementActionType => ({
  type: INCREMENT_COUNTER,
});

export const decrementCounterAction = (): DecrementActionType => ({
  type: DECREMENT_COUNTER,
});
```

Reducer

```
import type { Reducer } from '@reduxjs/toolkit';
import type { CounterActionsTypes } from '../../types/CounterTypes';
import { DECREMENT_COUNTER, INCREMENT_COUNTER } from '../../types/CounterTypes';
const counterReducer: Reducer<number, CounterActionsTypes> = (
  state = 0,
  action,
) => {
  switch (action.type) {
    case INCREMENT_COUNTER:
      return state + 1;

    case DECREMENT_COUNTER:
      return state - 1;
    default:
      return state;
  }
};

export default counterReducer;
```

Добавление reducer в store

```
import { configureStore, combineReducers } from '@reduxjs/toolkit';
import counterReducer from './reducers/counterReducer';

const store = configureStore({
  counter: counterReducer,
});

export default store;
```

Оборачиваем приложение в Provider

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import axios from 'axios';
import { BrowserRouter } from 'react-router-dom';
import { Provider } from 'react-redux';
import store from './redux/store';
import App from './App';
import 'bootstrap/dist/css/bootstrap.min.css';

axios.defaults.baseURL = 'http://localhost:3001';

const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement,
);
root.render(
  <BrowserRouter>
    <Provider store={store}>
      <App />
    </Provider>
  </BrowserRouter>,
);
```