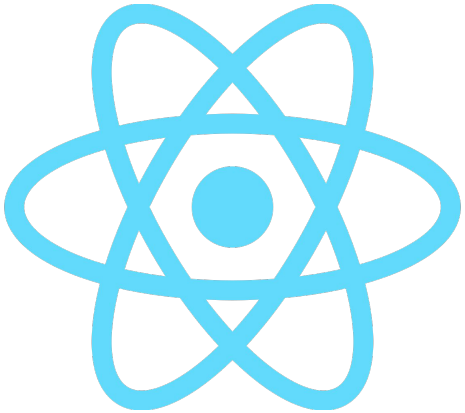
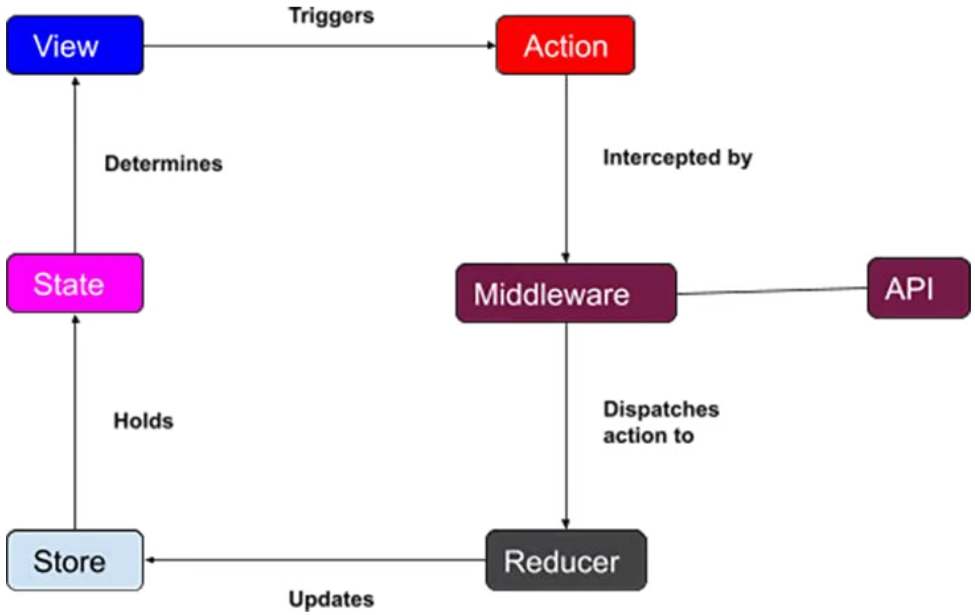


Тема занятия:

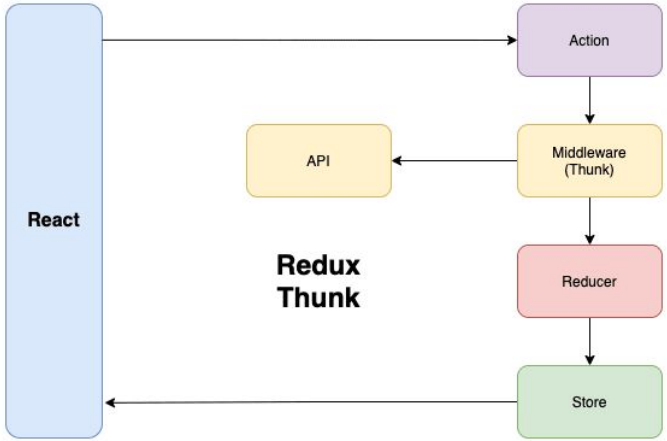
Redux Thunk

Middleware - промежуточная функция, которое берет входные данные, делает с ними что-то (например асинхронное действие - отправку запроса на сервер) и передаёт дальше.

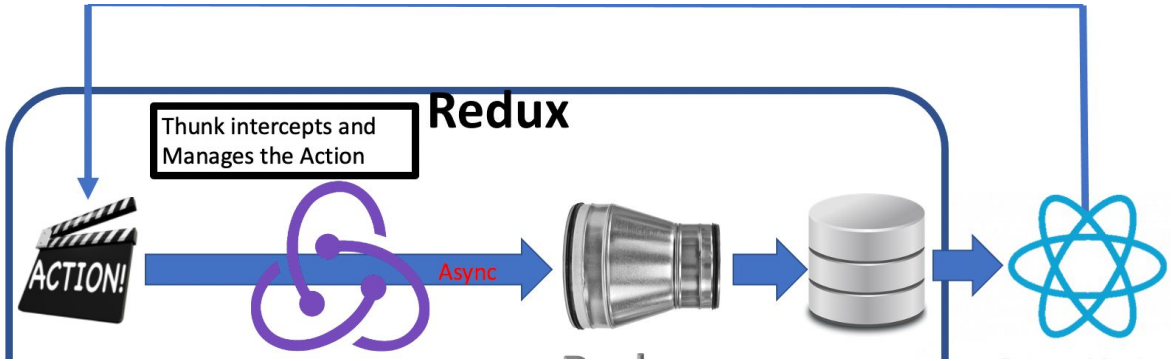
Например: есть конвейер, по которому движется пальто. На конвейере работают Зина и Людмила. Зина пришивает пуговку, Людмила прикладывает бирку. Внезапно, появляется middleware Лена, встает между Зиной и Людмилой и красит пуговку в модный цвет. Так как Лена после покраски не уносит пальто с собой, а передает дальше, то Людмила как ни в чем не бывало приделывает бирку и пальто готово. Только теперь оно более модное. Усиленное.



Redux Thunk — это middleware для библиотеки управления состоянием Redux. Он предоставляет возможность создавать и обрабатывать асинхронные действия (actions) в Redux.



В **Redux Toolkit** существует удобная функция **createAsyncThunk**, которая упрощает создание асинхронных действий, и она уже использует **Redux Thunk** внутри себя.



Шаг 1

Создайте slice с использованием **createAsyncThunk** и **extraReducers** (здесь добавляем действия, в зависимости от значений: fetchUserData.pending, …fulfilled, …rejected):

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

interface UserData {
  // Определите тип данных, которые мы получаем от сервера
}

// Создайте асинхронное действие с использованием createAsyncThunk
export const fetchUserData = createAsyncThunk('user/fetchUserData', async
() => {
  const response = await fetchDataFromServer();
  return response;
});

interface UserState {
  data: UserData | null;
  status: 'idle' | 'loading' | 'succeeded' | 'failed';
  error: string | null;
}

const initialState: UserState = {
  data: null,
  status: 'idle',
  error: null,
};
```

уникальный идентификатор для данного `createAsyncThunk`

Шаг 1

Создайте slice с использованием **createAsyncThunk** и **extraReducers** (здесь добавляем действия, в зависимости от значений: fetchUserData.pending, …fulfilled, …rejected):

```
const userSlice = createSlice({
  name: 'user',
  initialState,
  reducers: {},
  extraReducers: (builder) => {
    builder
      .addCase(fetchUserData.pending, (state) => {
        state.status = 'loading';
      })
      .addCase(fetchUserData.fulfilled, (state, action:
PayloadAction<UserData>) => {
        state.status = 'succeeded';
        state.data = action.payload;
      })
      .addCase(fetchUserData.rejected, (state, action:
PayloadAction<string>) => {
        state.status = 'failed';
        state.error = action.error.message;
      });
  },
});

export default userSlice.reducer;
```

Шаг 2

Используем созданный Slice в настройке Store

```
import { configureStore } from '@reduxjs/toolkit';
import userReducer from './slices/userSlice';

const store = configureStore({
  reducer: {
    user: userReducer,
    // Другие срезы могут быть добавлены здесь
  },
});

// Ниже будут все настройки, которые были указаны при создании проекта
…
```

Шаг 3

Используем **useDispatch** и **useSelector** в нашем компоненте:

```
import React, { useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { fetchUserData } from './slices/userSlice';
import { RootState} from './store/store';

const MyComponent = () => {
  const dispatch = useDispatch();
  const userData = useSelector((state: RootState) => state.user.data);
  const status = useSelector((state: RootState) => state.user.status);
  const error = useSelector((state: RootState) => state.user.error);

  useEffect(() => {
    // Вызовите асинхронное действие
    dispatch(fetchUserData());
  }, [dispatch]);

  if (status === 'loading') {
    return <div>Loading...</div>;
  }

  if (status === 'failed') {
    return <div>Error: {error}</div>;
  }

  return (
    <div>
      {/* Рендер компонента, используя userData из хранилища Redux */}
    </div>
  );
};

export default MyComponent;
```