В React абстрактные компоненты — это компоненты, которые не зависят от конкретной реализации UI, и их основной задачей является предоставление "скелета" или общей логики, на которой можно строить что-то более конкретное. Это похоже на создание "шаблонов" компонентов, которые можно гибко настраивать для разных случаев.

В чем смысл абстрактных компонентов?

Абстрактные компоненты фокусируются на:

Разделении логики и представления (UI):

   Они отделяют бизнес-логику, обработку данных, состояния и общие вещи от конкретного отображения.

Переиспользуемости:

   Общая функциональность сконцентрирована в одном месте, что позволяет избегать дублирования кода.

Масштабируемости:

   Такие компоненты легко модифицировать, расширять и использовать в различных местах приложения с разными настройками.

Другими словами, абстрактные компоненты улучшают структуру кода, обеспечивая гибкость без нарушения принципов DRY (Don't Repeat Yourself).

Как это выглядит?

Абстрактный компонент — это базовый компонент, содержащий только:

общую функциональность,

шаблонную структуру,

или базовые механики для взаимодействия.

Конкретные детали реализации передаются через props или рендер-пропсы (render props), контекст или обработчики событий.

Простой пример: Абстрактный список данных

Допустим, у нас есть несколько списков данных (товары, пользователи и т. д.). Они выглядят похоже, но данные могут быть разными. Абстрактный компонент может помочь обобщить их логику.

Абстрактный компонент:

import { useState, useEffect } from "react";

function AbstractList({ fetchData, renderItem }) {

    const [data, setData] = useState([]);

    const [loading, setLoading] = useState(true);

    useEffect(() => {

        fetchData().then(response => {

            setData(response);

            setLoading(false);

        });

    }, [fetchData]);

    if (loading) return Загрузка...;

    return (

            {data.map((item, index) => (

                {renderItem(item)}

            ))}

    );

}

export default AbstractList;

fetchData\*\* — это функция, которая загружает конкретные данные.

renderItem\*\* — функция, отвечающая за отображение одного элемента.

Конкретная реализация:

Теперь мы можем использовать этот абстрактный компонент для любого списка данных:

Компонент, который отображает список пользователей:

import AbstractList from "./AbstractList";

function UserList() {

    const fetchUsers = async () => {

        // Например, запрос API

        return [

            { id: 1, name: "Алиса" },

            { id: 2, name: "Боб" }

        ];

    };

    const renderUserItem = (user) => {user.name};

    return ;

}

export default UserList;

Компонент для списка продуктов:

import AbstractList from "./AbstractList";

function ProductList() {

    const fetchProducts = async () => {

        return [

            { id: 1, name: "Товар 1", price: 100 },

            { id: 2, name: "Товар 2", price: 200 },

        ];

    };

    const renderProductItem = (product) => (

            {product.name} - {product.price} ₽

    );

    return ;

}

export default ProductList;

Способы создания абстрактных компонентов

Есть несколько популярных подходов для создания абстрактных компонентов, чтобы сделать их гибкими:

1. Использование props

Это самый простой способ. Абстрактный компонент принимает пропсы и использует их для настройки поведения.

Пример:

function Button({ onClick, label, size = "medium" }) {

    return (

            {label}

    );

}

Компонент-кнопка абстрактен: он принимает функционал (onClick), текст (label) и размер.

2. Использование Render Props

Рендер-пропсы позволяют передавать React-компоненты как пропсы, чтобы точно настроить вывод.

Пример из выше: функция renderItem в списке.

Ещё пример:

function Modal({ header, children, footer }) {

    return (

            {header()}

            {children}

            {footer()}

    );

}

Использование:

 Заголовок}

    footer={() => Закрыть}

    Это тело модального окна.

3. Использование слотов (children)

Если компонент предполагает рендеринг дочерних элементов, children можно использовать как "слоты".

Пример:

function Card({ children }) {

    return {children};

}

Использование:

    Заголовок

    Контент

4. HOC (Higher-Order Component)

Это старый подход, когда логику выносят в отдельную функцию, которая оборачивает другие компоненты. Сейчас чаще используются хуки и рендер-пропсы, но знать этот способ полезно.

Пример:

function withLoading(Component) {

    return function WrappedComponent({ isLoading, ...props }) {

        if (isLoading) return Загрузка...;

        return ;

    };

}

Использование:

const UserListWithLoading = withLoading(UserList);

;

Преимущества абстрактных компонентов

Упрощение кода: Общая логика находится в одном месте, а не повторяется везде.

Переиспользуемость: Один абстрактный компонент можно использовать для множества конкретных задач.

Лёгкость изменений: Изменив абстрактный компонент, вы автоматически изменяете все связанные с ним реализации.

Гибкость: Они дают возможность описывать структуру компонентов при помощи пропсов, рендер-пропсов и слотов.

Когда использовать абстрактные компоненты?

У вас есть похожая функциональность в разных частях приложения.

При проектировании повторяющихся элементов: формы, списки, модальные окна, кнопки и т. д.

Вы хотите создать более DRY (Don’t Repeat Yourself) код.

Вам нужно отделить бизнес-логику от визуального представления.

Заключение

Абстрактные компоненты в React — это мощный инструмент для улучшения структуры и читабельности кода. Они помогают обобщить логику, минимизировать дублирование и упростить поддержку и расширение приложения. Ключевая идея — создать универсальный блок, который легко адаптируется к различным ситуациям через настройки и пропсы.

Учитесь находить общий функционал, выносить его в абстракцию и рендерить специфики через пропсы или children — это важный шаг к созданию качественных и масштабируемых React-приложений.