

# Лекция 7

# React Hooks. FSD

Разработка интернет приложений

Канев Антон Игоревич

# Функциональные компоненты

- Описание компонентов с помощью чистых функций создает меньше кода, а значит его легче поддерживать.
- Чистые функции намного проще тестировать. Вы просто передаете props на вход и ожидаете какую то разметку.
- В будущем чистые функции будут выигрывать по скорости работы в сравнении с классами из-за отсутствия методов жизненного цикла
- Все это стало возможным благодаря хукам <https://react.dev/reference/react>

# Хуки

```
import React, { useEffect, useState } from 'react'
import Axios from 'axios'
```

```
export default function Hello() {
```

```
    const [Name, setName] = useState('')
```

```
    useEffect(() => {
        Axios.get('/api/user/name')
        .then(response => {
            setName(response.data.name)
        })
    }, [])
```

```
    return (
        <div>
            My name is {Name}
        </div>
    )
```

```
}
```

```
import React, { Component } from 'react'
import Axios from 'axios'
```

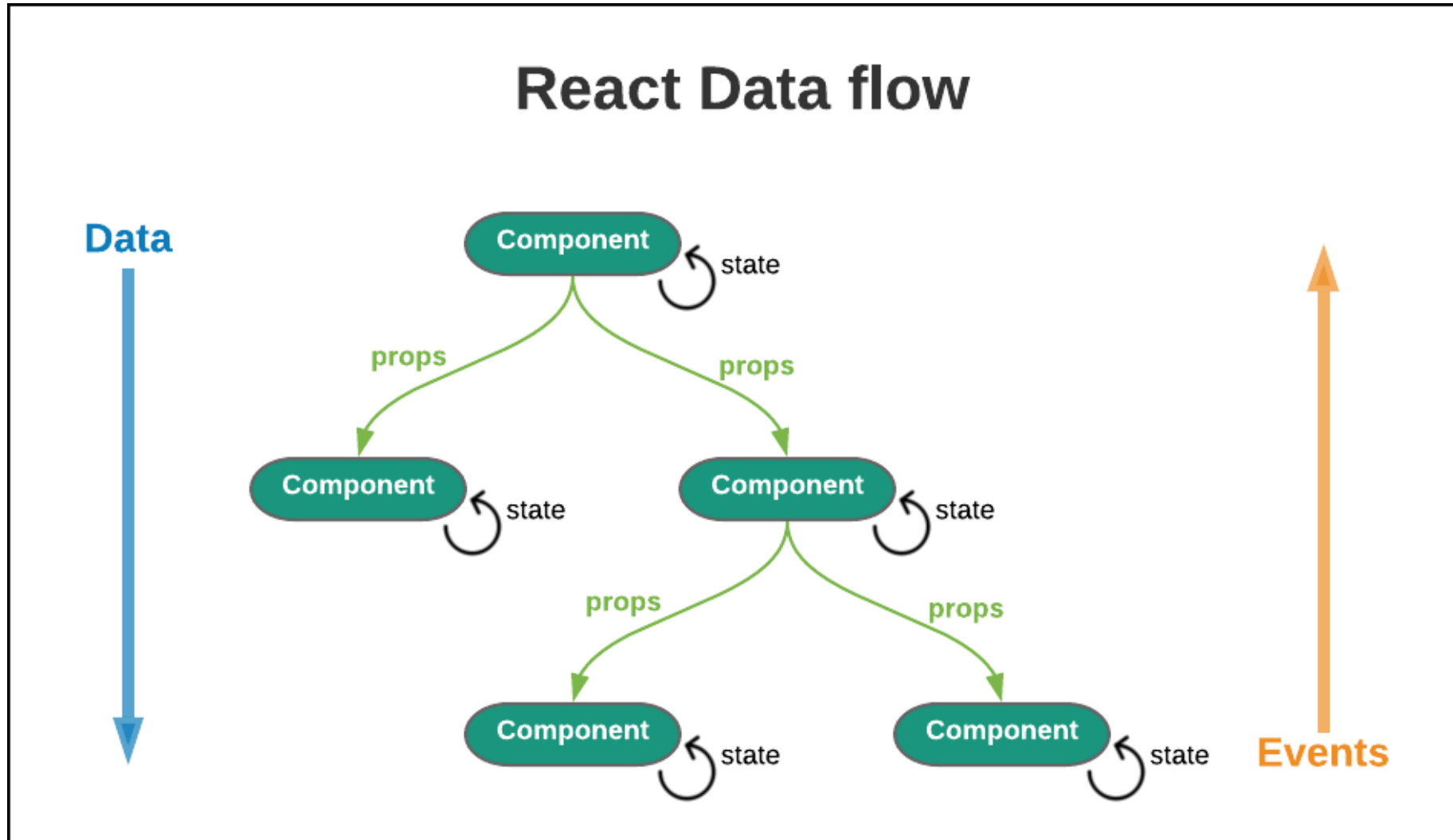
```
export default class Hello extends Component {
```

```
    constructor(props) {
        super(props);
        this.state = { name: '' };
    }
```

```
    componentDidMount() {
        Axios.get('/api/user/name')
        .then(response => {
            this.setState({ name: response.data.name })
        })
    }
```

```
    render() {
        return (
            <div>
                My name is {this.state.name}
            </div>
        )
    }
```

# Поток данных и сообщений



# Состояние

- Компонент нуждается в state, когда данные в нём со временем изменяются.
- Например, компоненту Checkbox может понадобиться состояние isChecked.
- Разница между пропсами и состоянием заключается в основном в том, что состояние нужно для управления компонентом, а пропсы для получения информации.

# Хуки. useState

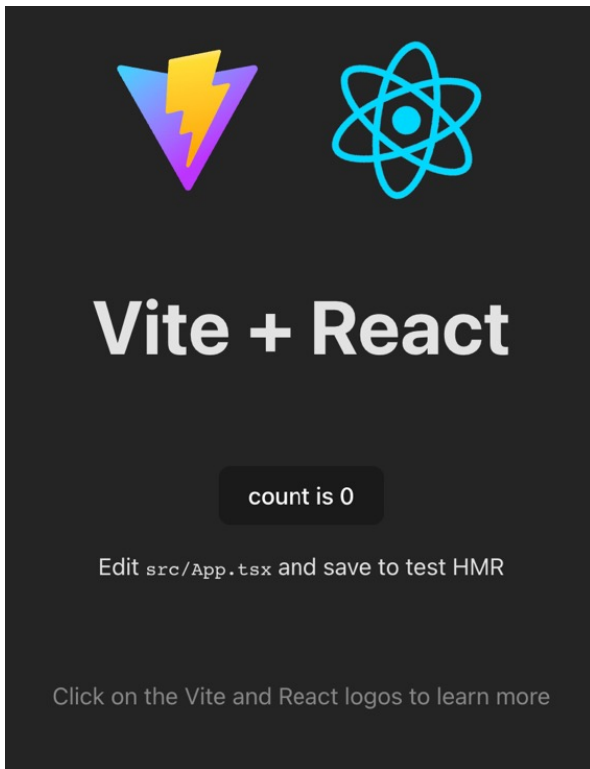
- Хуки позволяют работать с состоянием компонентов, с методами их жизненного цикла, с другими механизмами React без использования классов.

```
class Example extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      count: 0  
    };  
  }  
}
```

```
import React, { useState } from 'react';  
  
function Example() {  
  // Объявление новой переменной состояния «count»  
  const [count, setCount] = useState(0);  
  return <div onClick={()=>setCount(count=>count++)}>{count}</div>  
}
```

# App.tsx

- Первый и главный компонент нашего приложения
- В шаблоне уже есть кнопка, хук состояния



```
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'

function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      <div>
        <a href="https://vitejs.dev" target="_blank">
          <img src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.tsx</code> and save to test HMR
        </p>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </>
  )
}

export default App
```

# Жизненный цикл приложения

- **1: Монтирование**

компонент запускает `getDerivedStateFromProps()`, потом запускается `render()`, возвращающий JSX. React «монтируется» в DOM

- **2: Обновление**

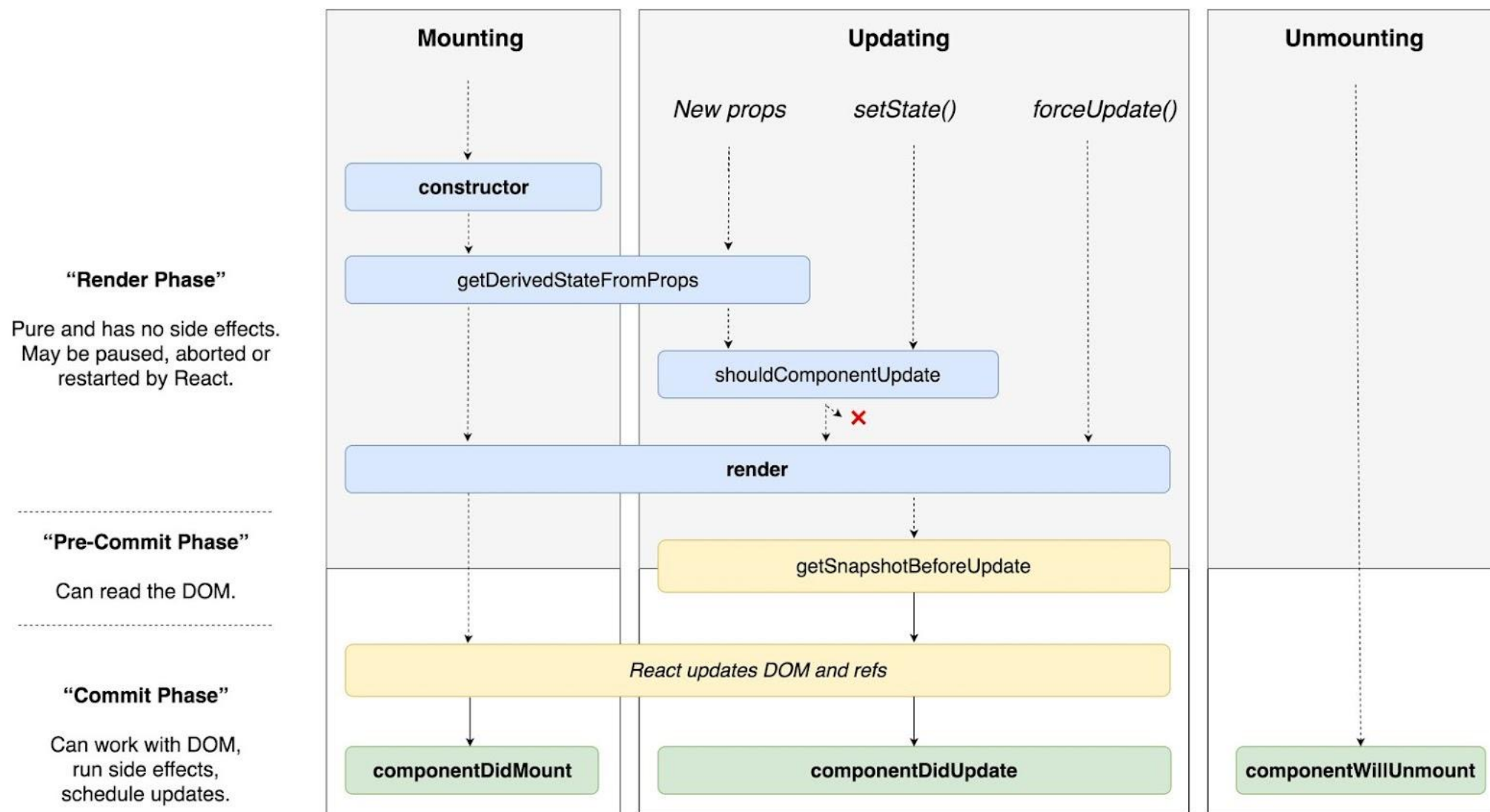
Данный этап запускается во время каждого изменения состояния либо свойств

- **3: Размонтирование**

React выполняет запуск `componentWillUnmount()` непосредственно перед удалением из DOM



# Методы жизненного цикла компонента



# useEffect

- componentDidMount()
- componentDidUpdate()
- componentWillUnmount()

```
useEffect( effect: ()=>{  
    console.log('Этот код выполняется только на первом рендере компонента')  
    // В данном примере можно наблюдать Spread syntax (Трехточие перед массивом)  
    setNames( value: names=>[...names, 'Бедный студент'])  
  
    return () => {  
        console.log('Этот код выполняется, когда компонент будет размонтирован')  
    }  
}, deps: [])  
  
useEffect( effect: ()=>{  
    console.log('Этот код выполняется каждый раз, когда изменится состояние showNames ')  
    setRandomName(names[Math.floor( x: Math.random()*names.length)])  
}, deps: [showNames])
```

# Другие хуки

- **useContext**: позволяет работать с контекстом — с механизмом для организации совместного доступа к данным без передачи свойств.
- **useReducer**: усложняет useState добавляя разделение логики в зависимости от action. Вместе с useContext дают аналог Redux.

```
const ThemeContext = createContext(null);

export default function MyApp() {
  return (
    <ThemeContext.Provider value="dark">
      <Form />
    </ThemeContext.Provider>
  )
}
```

```
function Button({ children }) {
  const theme = useContext(ThemeContext);
  const className = 'button-' + theme;
  return (
    <button className={className}>
      {children}
    </button>
  );
}
```

# Другие хуки

- **useMemo**: используется для возврата мемоизированного значения. Может применяться, чтобы функция возвратила кешированное значение.
- Можно сохранить результаты вычислений между вызовами render

```
import { useMemo } from 'react';
```

```
function TodoList({ todos, tab, theme }) {  
  const visibleTodos = useMemo(() => filterTodos(todos, tab), [todos, tab]);  
  // ...  
}
```

# React Router Hooks

- **useLocation**: все данные о текущем пути url
- **useNavigate**: объект истории браузера
- **useParams**: параметры из url
- <https://reactrouter.com/>

```
1  import * as React from 'react';
2  import { Routes, Route, useParams } from 'react-router-dom';
3
4  function ProfilePage() {
5    // Get the userId param from the URL.
6    let { userId } = useParams();
7    // ...
8  }
9
10 function App() {
11   return (
12     <Routes>
13       <Route path="users">
14         <Route path=":userId" element={<ProfilePage />} />
15         <Route path="me" element={...} />
16       </Route>
17     </Routes>
18   );
19 }
```

# GitHub Pages



<https://rashidshamloo.hashnode.dev/deploying-vite-react-app-to-github-pages>

- Мы можем бесплатно развернуть наше React приложение на GitHub Pages
- Необходимо выполнить настройки в GitHub: Settings/Pages
- Также необходимо настроить развертывание Vite в проекте

## GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

✓ Your site is published at <https://fccvienna.github.io/>

### Source

Your GitHub Pages site is currently being built from the master branch. [Learn more.](#)

master branch ▾

Save

User pages must be built from the master branch.

### Theme Chooser

Select a theme to build your site with a Jekyll theme. [Learn more.](#)

Choose a theme

### Custom domain

Custom domains allow you to serve your site from a domain other than `fccvienna.github.io`. [Learn more.](#)

Save

### ✓ Enforce HTTPS

— Required for your site because you are using the default domain (`fccvienna.github.io`)

HTTPS provides a layer of encryption that prevents others from snooping on or tampering with traffic to your site.

# GitHub Pages

- **1. Установить gh-pages пакет**

`npm install gh-pages --save-dev`

- **2. В файле package.json добавить строки перед "build": "vite build",**  
"predeploy": "npm run build",  
"deploy": "gh-pages -d dist",

- **3. В файле vite.config.js добавить строку перед plugins: [react()],**  
base: "YOUR\_REPOSITORY\_NAME",

- **4. Выполнить развертывание/обновление**

`npm run deploy`

Теперь есть ветка gh-pages в репозитории, а приложение развернуто в GitHub

<https://rashidshamloo.hashnode.dev/deploying-vite-react-app-to-github-pages>

# FSD

Рассмотрим приложение социальной сети.

- app/ содержит настройку роутера, глобальные хранилища и стили.
- pages/ содержит компоненты роутов на каждую страницу в приложении, преимущественно композирующие, по возможности, без собственной логики.

В рамках этого приложения рассмотрим карточку поста в ленте новостей.

- widgets/ содержит "собранную" карточку поста, с содержимым и интерактивными кнопками, в которые вшиты запросы к бэкенду.
- features/ содержит всю интерактивность карточки (например, кнопку лайка) и логику обработки этой интерактивности.
- entities/ содержит скелет карточки со слотами под интерактивные элементы. Компонент, демонстрирующий автора поста, также находится в этой папке, но в другом слайсе.



Layers

Slices

Segments

- <https://feature-sliced.design/ru/docs/get-started/overview>



# Архитектура FSD

**Слои** стандартизированы во всех проектах и расположены вертикально. Модули на одном слое могут взаимодействовать лишь с модулями, находящимися на слоях строго ниже. На данный момент слоев семь (снизу вверх):

1. **shared** — переиспользуемый код, не имеющий отношения к специфике приложения/бизнеса (например, UIKit, libs, API)
2. **entities** (сущности) — бизнес-сущности (например, User, Product, Order)
3. **features** (фичи) — взаимодействия с пользователем, действия, которые несут бизнес-ценность для пользователя (например, SendComment, AddToCart, UsersSearch)
4. **widgets** (виджеты) — композиционный слой для соединения сущностей и фич в самостоятельные блоки (например, IssuesList, UserProfile).
5. **pages** (страницы) — композиционный слой для сборки полноценных страниц из сущностей, фич и виджетов.
6. **processes** (процессы, устаревший слой) — сложные сценарии, покрывающие несколько страниц (например, авторизация)
7. **app** — настройки, стили и провайдеры для всего приложения.

# Архитектура FSD

- Затем есть **слайсы**, разделяющие код по предметной области. Они группируют логически связанные модули, что облегчает навигацию по кодовой базе. Слайсы не могут использовать другие слайсы на том же слое, что обеспечивает высокий уровень *связности* (cohesion) при низком уровне *зацепления* (coupling).
- В свою очередь, каждый слайс состоит из **сегментов**. Это маленькие модули, главная задача которых — разделить код внутри слайса по техническому назначению. Самые распространенные сегменты — ui, model (store, actions), api и lib(utils/hooks), но в вашем слайсе может не быть каких-то сегментов, могут быть другие, по вашему усмотрению.

# Преимущества FSD

## **Единообразие**

- Код распределяется согласно области влияния (слой), предметной области (слайс) и техническому назначению (сегмент).  
Благодаря этому архитектура стандартизируется и становится более простой для ознакомления.

## **Контролируемое переиспользование логики**

- Каждый компонент архитектуры имеет свое назначение и предсказуемый список зависимостей.  
Благодаря этому сохраняется баланс между соблюдением принципа **DRY** и возможностью адаптировать модуль под разные цели.

## **Устойчивость к изменениям и рефакторингу**

- Один модуль не может использовать другой модуль, расположенный на том же слое или на слоях выше. Благодаря этому приложение можно изолированно модифицировать под новые требования без непредвиденных последствий.

## **Ориентированность на потребности бизнеса и пользователей**



- Разбиение приложения по бизнес-доменам помогает глубже понимать, структурировать и находить фичи проекта.

# Пример FSD

- В нашем GitLab доступен простой пример по FSD - рекомендуется для дипломной работы.
- В нем каждый слой состоит из слайсов, например, **Header**, **LoginPage** и так далее.

main ▾

react / src / shared / ui / Loader / Loader.tsx

 **Loader.tsx**  367 B

```
1 import { Box, CircularProgress } from '@mui/material';
2
3 import type { FC } from 'react';
4
5 export const Loader: FC = () => (
6   <Box
7     alignItems='center'
8     display='flex'
9     height='100vh'
10    justifyContent='center'
11    left='0'
12    position='fixed'
13    top='0'
14    width='100%'
15  >
16    <CircularProgress />
17  </Box>
18 );
```

**Name**

..

api

app

entities/user


features/Login


layouts/AuthorizedLayout


pages

shared

widgets/Header

 index.css

 index.tsx

 vite-env.d.ts

- <https://projects.iu5.bmstu.ru/iu5/infrastructure/departments-services/templates/react>