Лекция 9
Архитектура фронтенда

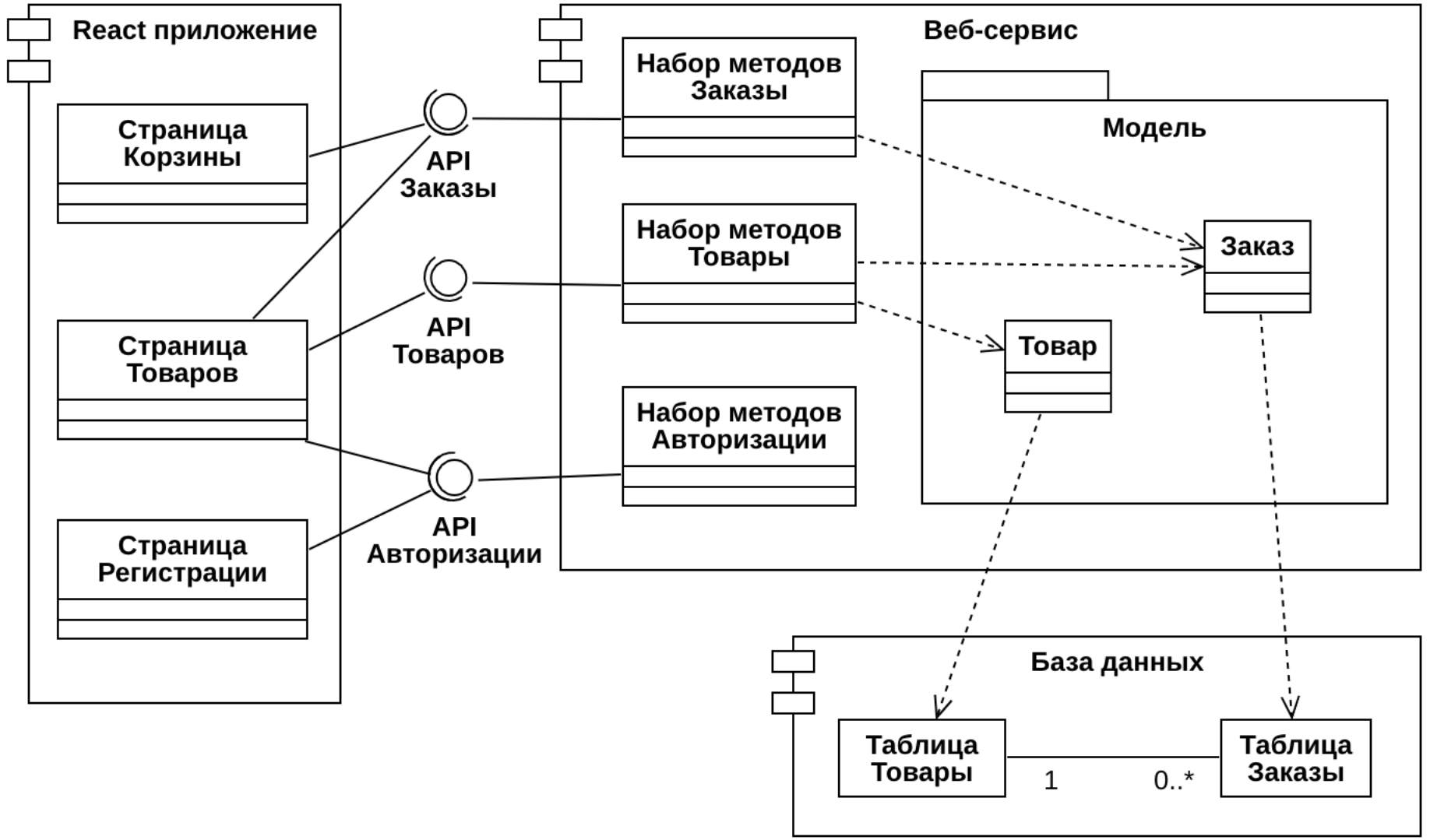
Проектирование систем и продуктовая веб-разработка

Канев Антон Игоревич

Классы фронтенда

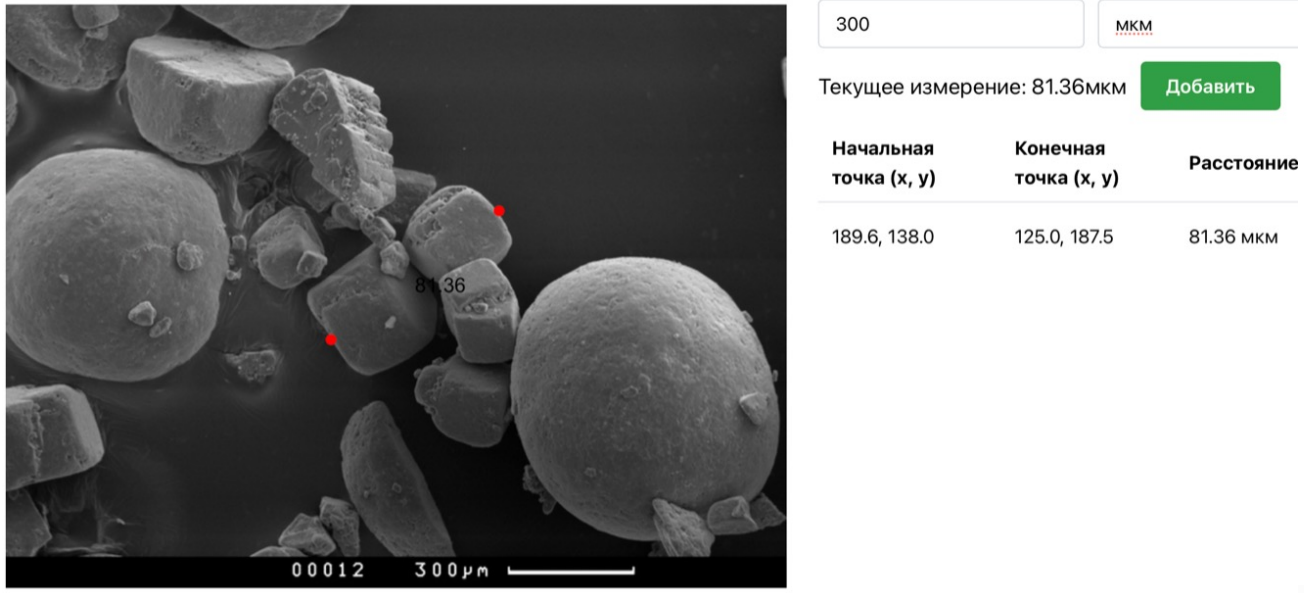
- Мы хотим объединить бэкенд и фронтенд
- Но на 1 диаграмме все не поместиться

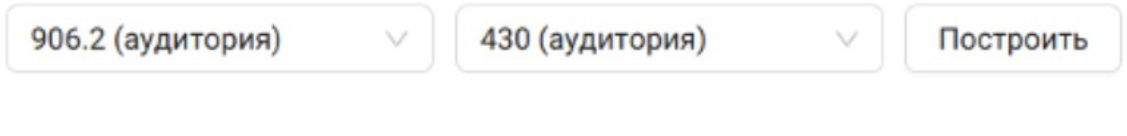
- Поэтому для 5 лабораторной оставляем **ТОЛЬКО** страницы React и домены веб-сервиса

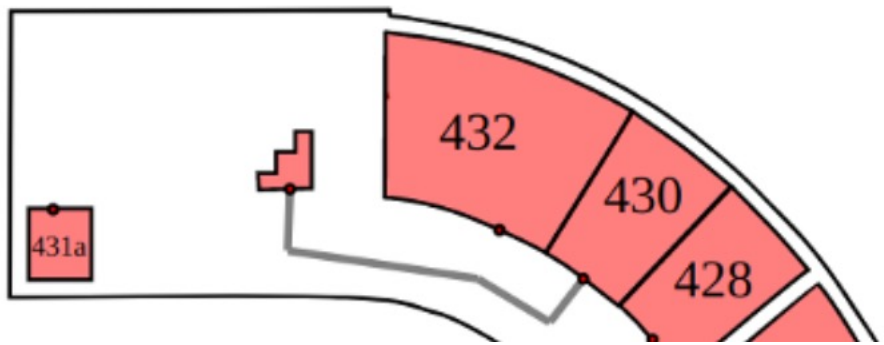


Фронтенд от ИУ5

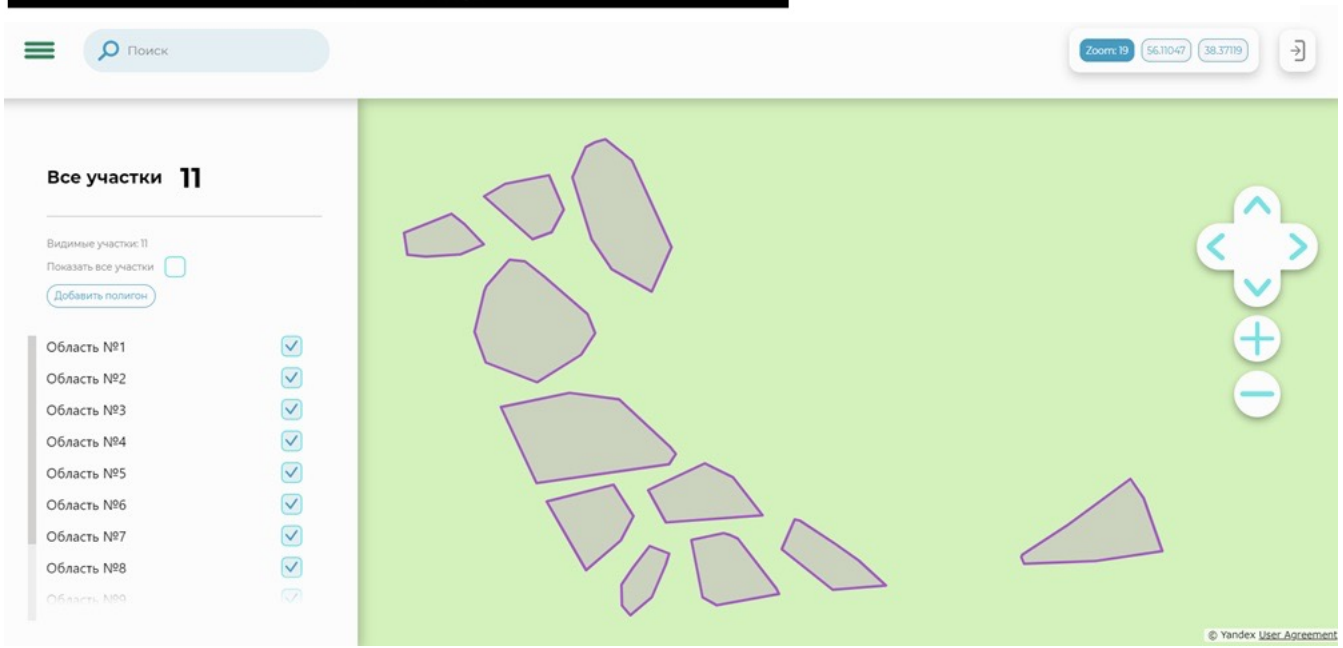
https://github.com/iu5git/PhotoPointApp







https://github.com/iu5git/CampusMap

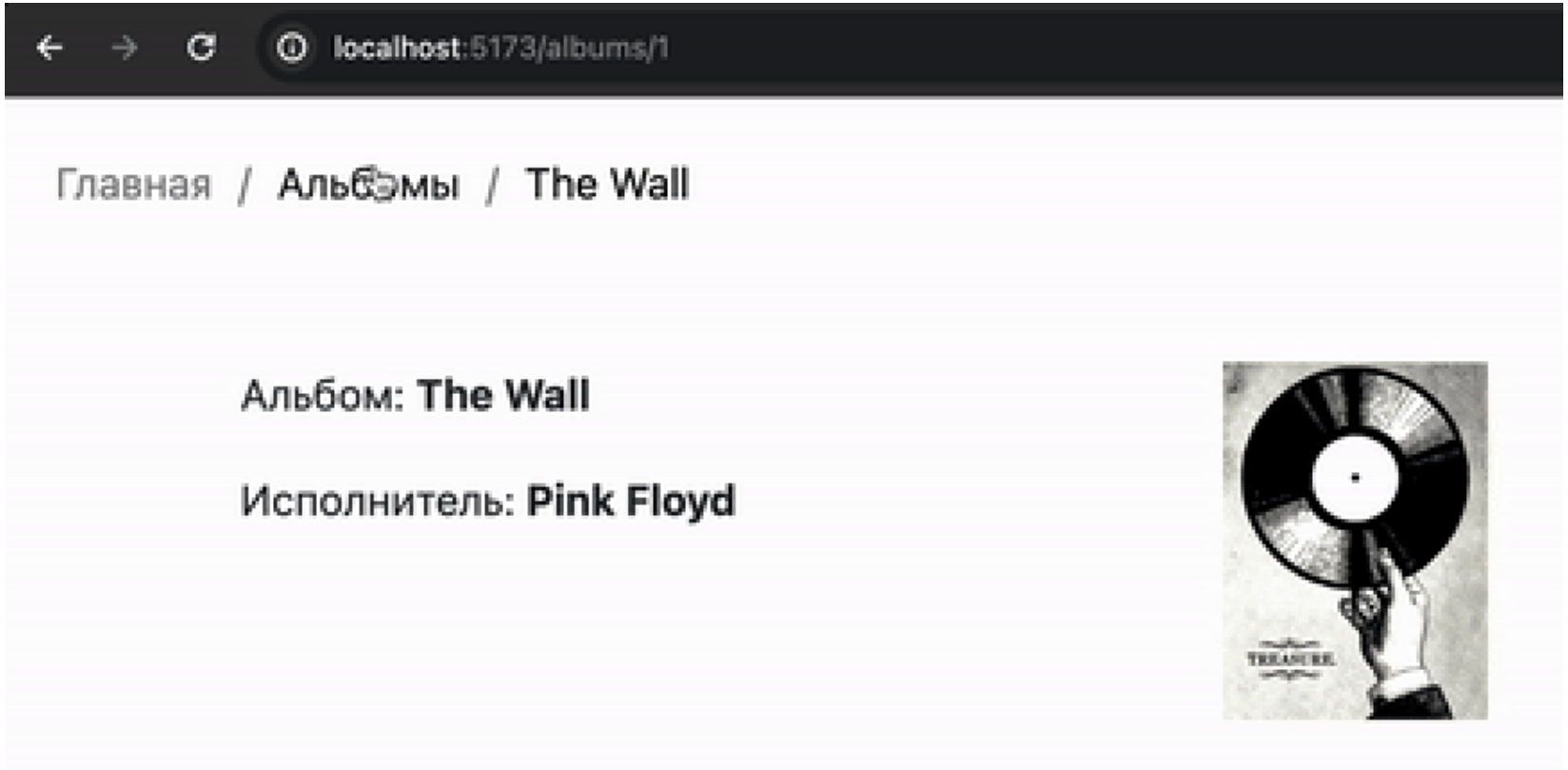


https://github.com/iu5git/MapViewApp

Breadcrumbs

Для навигации пользователя по страницам нашего приложения будем использовать

- Самописные Хлебные крошки
- Навигационную панель (меню) из react-bootstrap



Breadcrumbs, поиск

- Создаем наш собственный компонент для реализации хлебных крошек
- В поиске ждем событие onSubmit, в котором используем состояние searchValue

```
export const BreadCrumbs: FC<BreadCrumbsProps> = (props) => {
  const { crumbs } = props;

  return (
    <ul className="breadcrumbs">
      <li>
        <Link to={ROUTES.HOME}>Главная</Link>
      </li>
      {!!crumbs.length &&
        crumbs.map((crumb, index) => (
          <React.Fragment key={index}>
            <li className="slash">/</li>
            {index === crumbs.length - 1 ? (
              <li>{crumb.label}</li>
            ) : (
              <li>
                <Link to={crumb.path || ""}>{crumb.label}</Link>
              </li>
            )}
          </React.Fragment>
        ))}
    ul>
```

```
return (
  <div className="container">
    <BreadCrumbs crumbs={[{ label: ROUTE_LABELS.ALBUMS }]} />

    <InputField
      value={searchValue}
      setValue={(value) => setSearchValue(value)}
      loading={loading}
      onSubmit={handleSearch}
    />
```

Mock

- В реальной жизни бывает очень сложно связать фронтенд и бэкенд вместе: что-то не готово, проблемы с развертывание или Cors

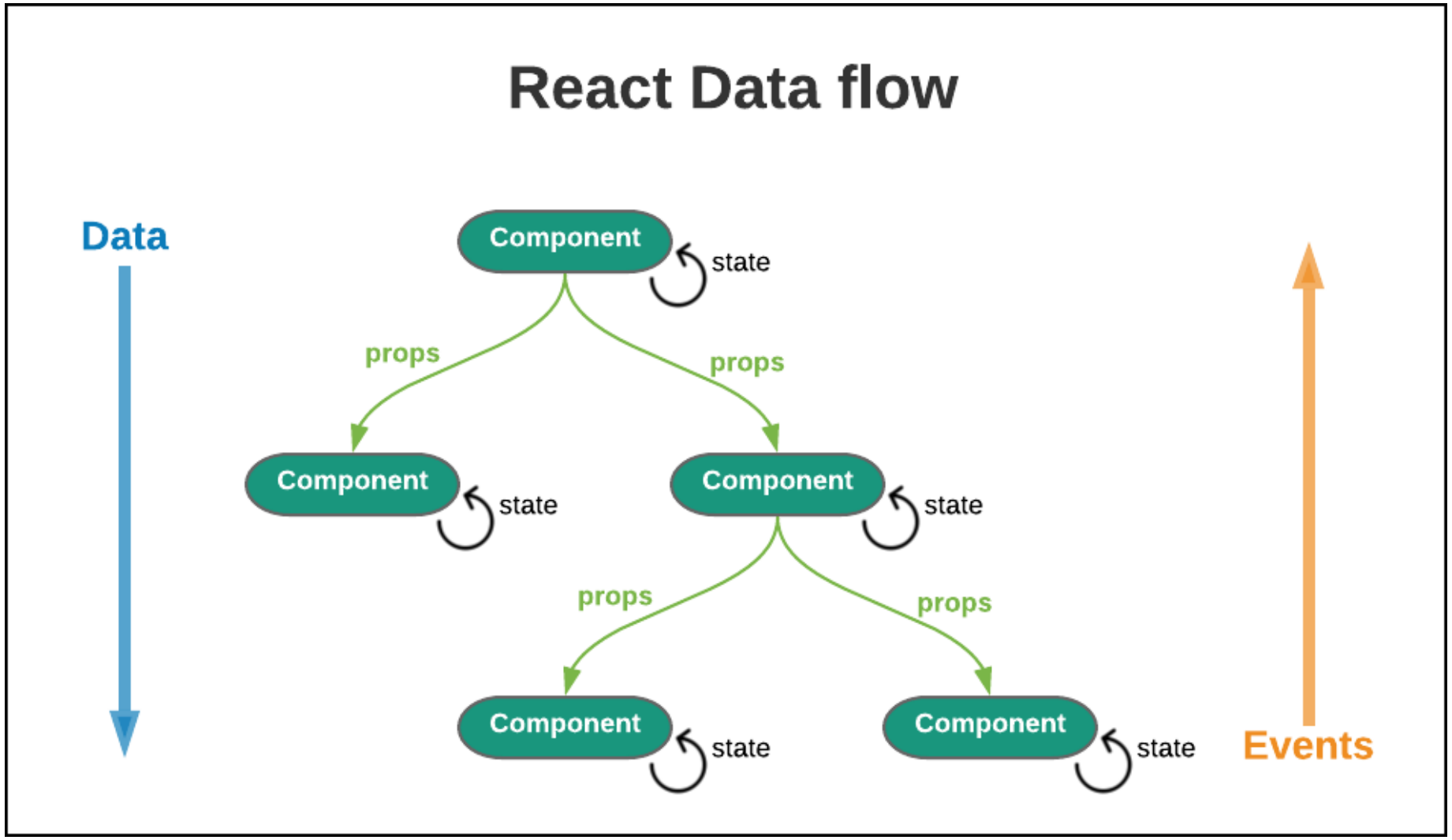
- Поэтому бэкенд проверяем через Swagger/Postman, а для фронтенда готовим Mock

```
import { ITunesResult } from "./getMusicByName";

export const SONGS_MOCK: ITunesResult = {
  resultCount: 3,
  results: [
    {
      wrapperType: "track",
      artistName: "Pink Floyd",
      collectionCensoredName: "The Wall",
      trackViewUrl: "",
      artworkUrl100: "",
    },
    {
      wrapperType: "track",
      artistName: "Queen",
      collectionCensoredName: "A Night At The Opera",
      trackViewUrl: "",
      artworkUrl100: "",
    },
    {
      wrapperType: "track",
      artistName: "AC/DC",
      collectionCensoredName: "Made in Heaven",
      trackViewUrl: "",
      artworkUrl100: "",
    },
  ],
};
```

Поток данных и сообщений

- Каждое состояние связано с его компонентом
- Чтобы передать данные между компонентами нам нужны потоки данных и события

- Это **усложняет** хранение **общих данных**, например информации о пользователе
- Выход: менеджеры состояний, и другие хуки



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

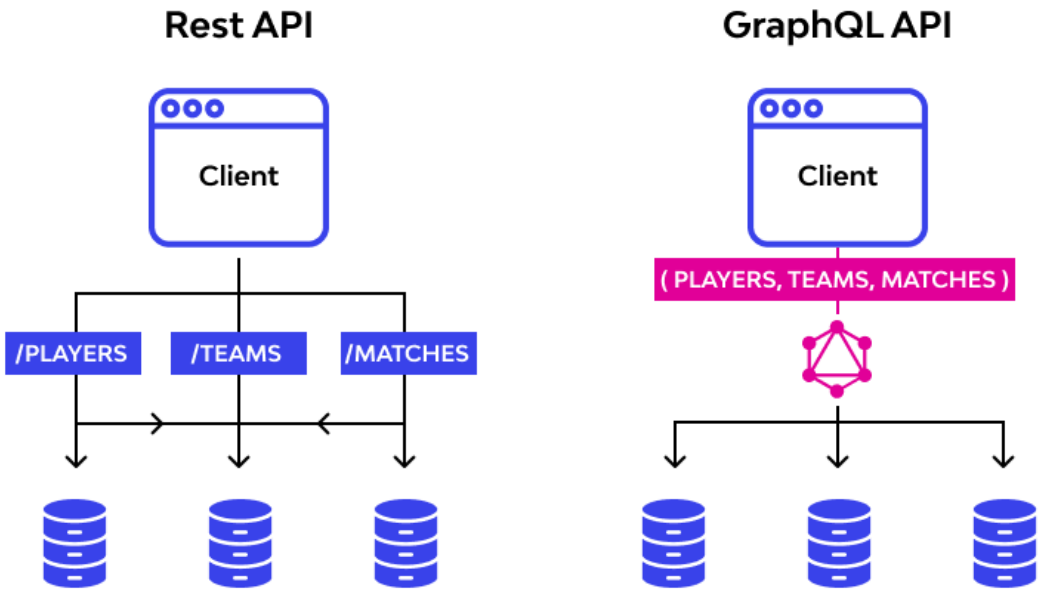
- **useLocation**: все данные о текущем пути url
- **useNavigate**: объект истории браузера
- **useParams**: параметры из url

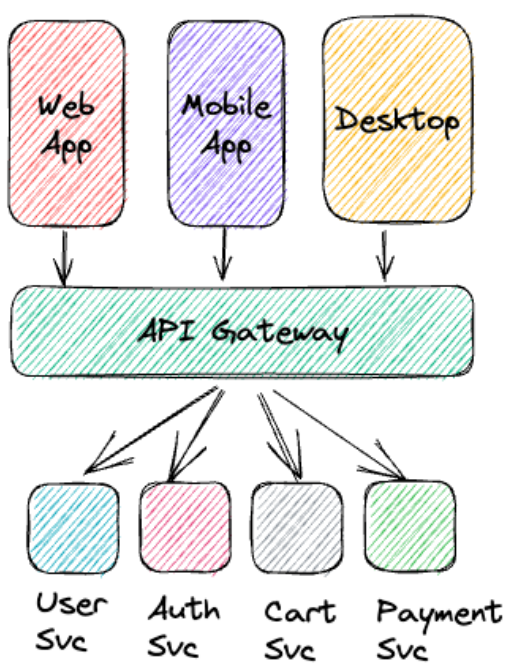
```jsx
import * as React from 'react';
import { Routes, Route, useParams } from 'react-router-dom';

function ProfilePage() {
  // Get the userId param from the URL.
  let { userId } = useParams();
  // ...
}

function App() {
  return (
    <Routes>
      <Route path="users">
        <Route path=":userId" element={<ProfilePage />} />
        <Route path="me" element={...} />
      </Route>
    </Routes>
  );
}
```

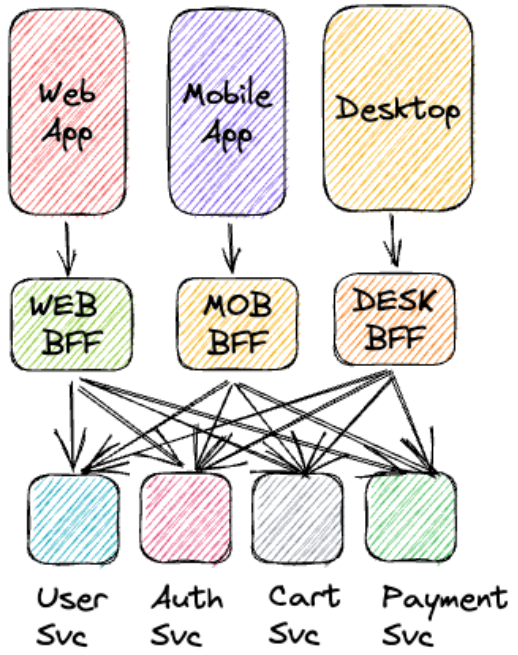
- https://reactrouter.com/

BFF и GraphQL

- GraphQL – язык запросов и сервер, который с открытым исходным кодом





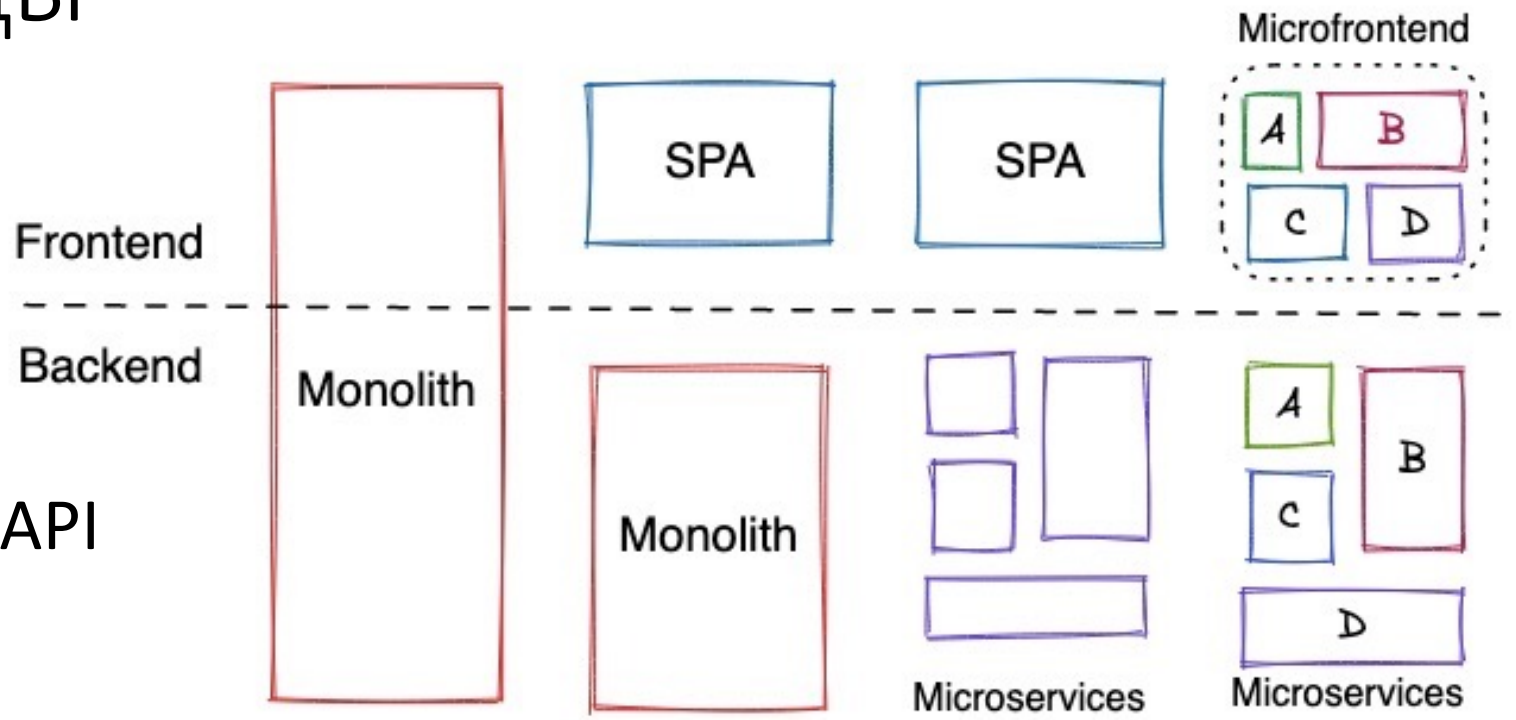


- Backend for frontend – шлюз к нашим API, адаптированный под каждого из потребителей: веб-приложение, мобильное, десктоп

Микрофронтенды

- В одном интерфейсе несколько фронтенд фреймворков: разные команды фронтенд разработчиков, разные API

- Одни из способов уйти от зависимости от одной технологии и постепенно внедрять новую



Next.js

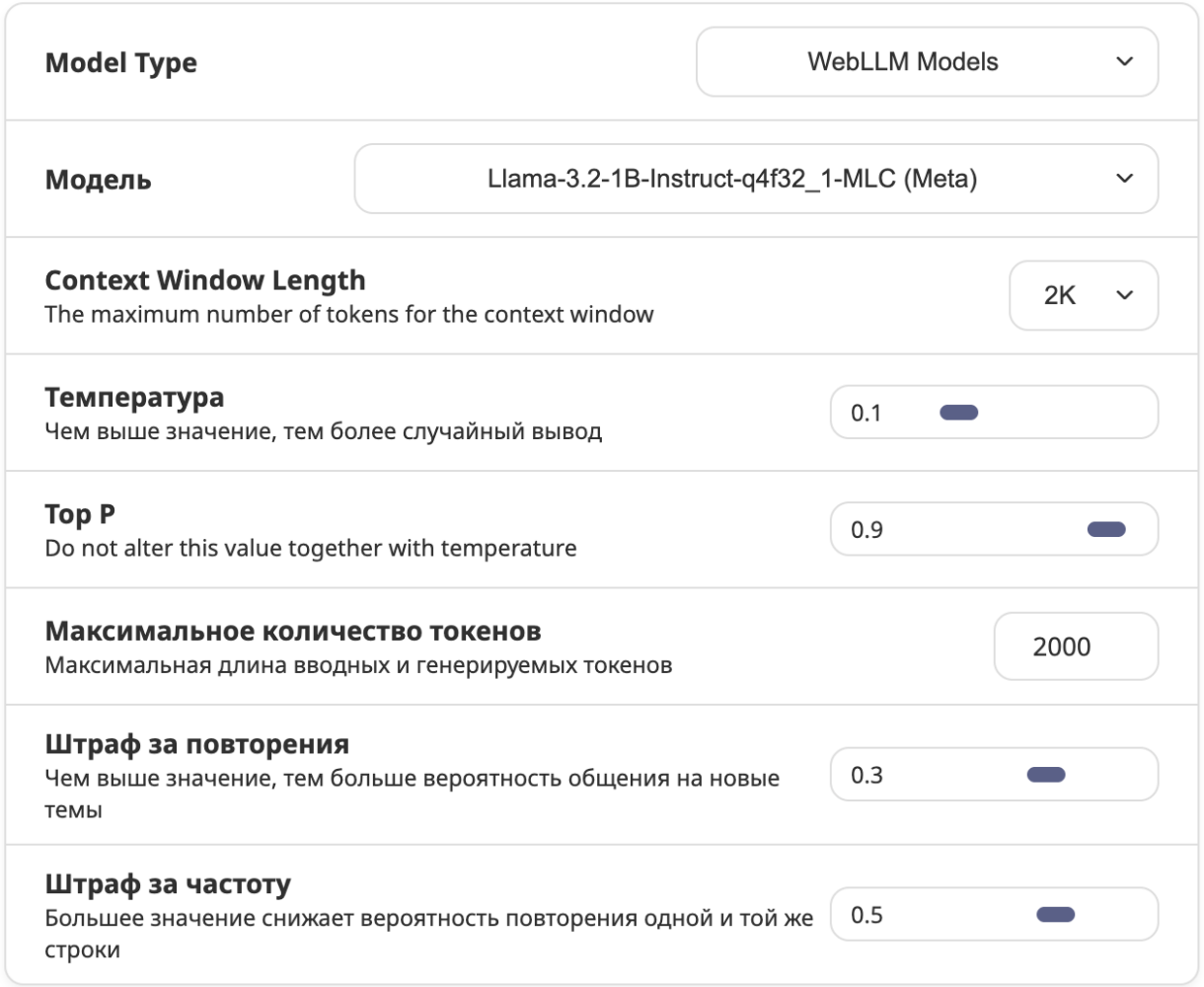
- SPA сильный инструмент, но как индексировать в поисковиках? На странице нет никаких данных
- Для этого используем серверные компоненты Next.js
- Хуков нет: данные передаем через props
- SSG (Static Site Generation) – HTML генерируется при **сборке** приложения

```
import type { InferGetStaticPropsType, GetStaticProps } from 'next'

type Repo = {
  name: string
  stargazers_count: number
}

export const getStaticProps = (async (context) => {
  const res = await fetch('https://api.github.com/repos/vercel/next.js')
  const repo = await res.json()
  return { props: { repo } }
}) satisfies GetStaticProps<{
  repo: Repo
}>

export default function Page({
  repo,
}: InferGetStaticPropsType<typeof getStaticProps>) {
  return repo.stargazers_count
}
```

WebLLM – ваша LLM прямо в браузере

https://chat.webllm.ai/

- WebLLM – пакет JavaScript, который позволяет запускать LLM прямо в браузере, использует WebGPU
- Вам требуется Google Chrome



- Можете использовать разные модели, настроить температуру
- **Температура** – параметр вариативности ответа: низкая температура оставляет только токены с высокими вероятностями
- Методичка WebLLM:

https://github.com/iu5git/Web/blob/main/tutorials/WebLLM/README.md

Открытые LLM модели

Часть популярных моделей являются открытыми и каждый желающий, может их запустить у себя (Edge Device).

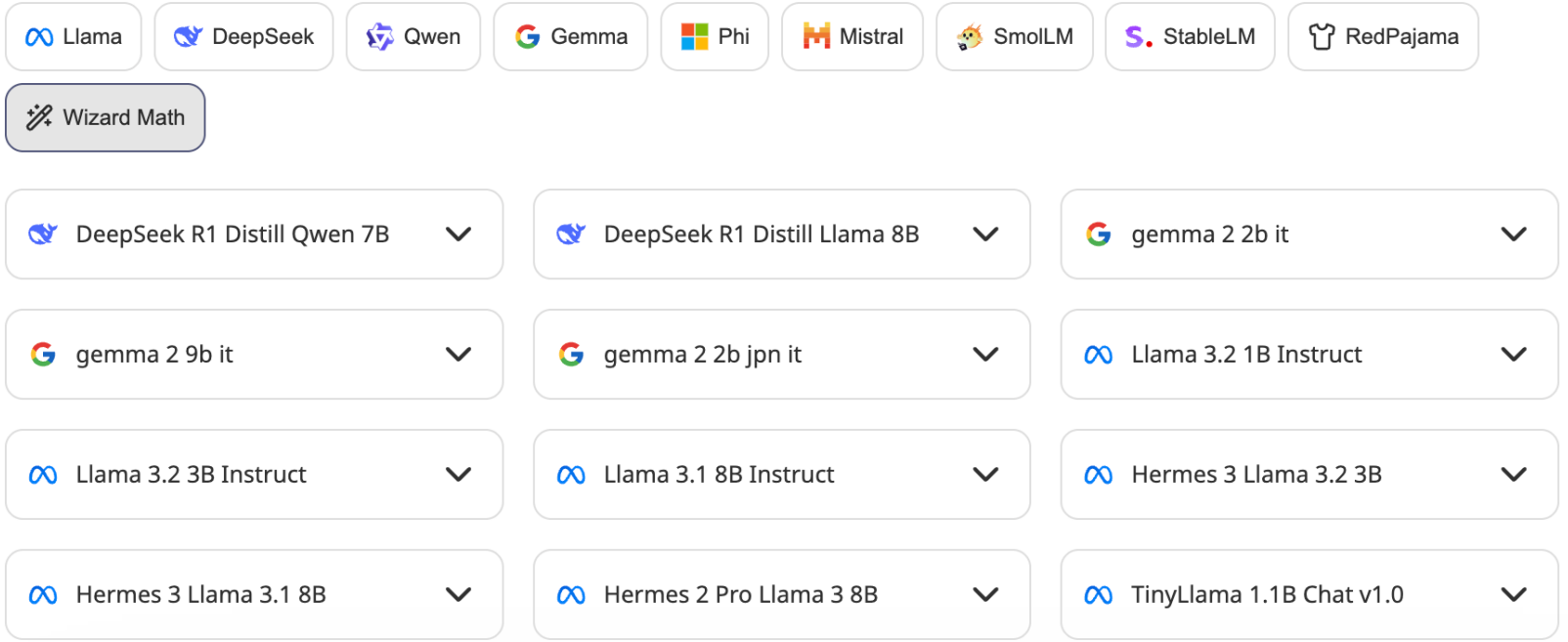
Список от mlc.ai:

Llama

Mistral от Mistral AI

Qwen от Alibaba Cloud

и тд



Также почитать:

https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard#/

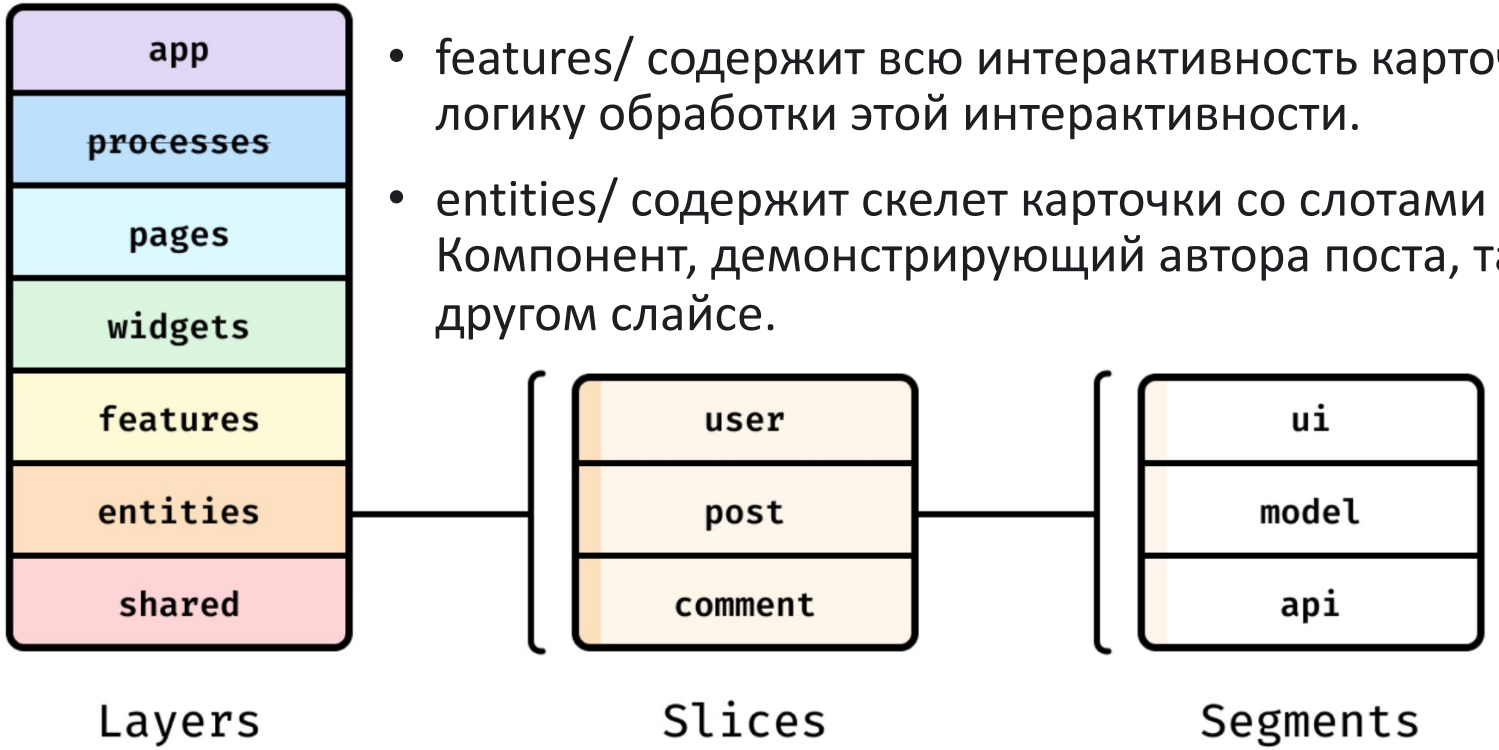
FSD

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.



- **Слайсы** разделяют код по предметной области
- В свою очередь, каждый слайс состоит из **сегментов** по техническому назначению: ui, model (store, actions), api и utils/hooks

- 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.

Преимущества FSD

Единообразие

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

Контролируемое переиспользование логики

- Каждый компонент архитектуры имеет свое назначение и предсказуемый список зависимостей, 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, 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.

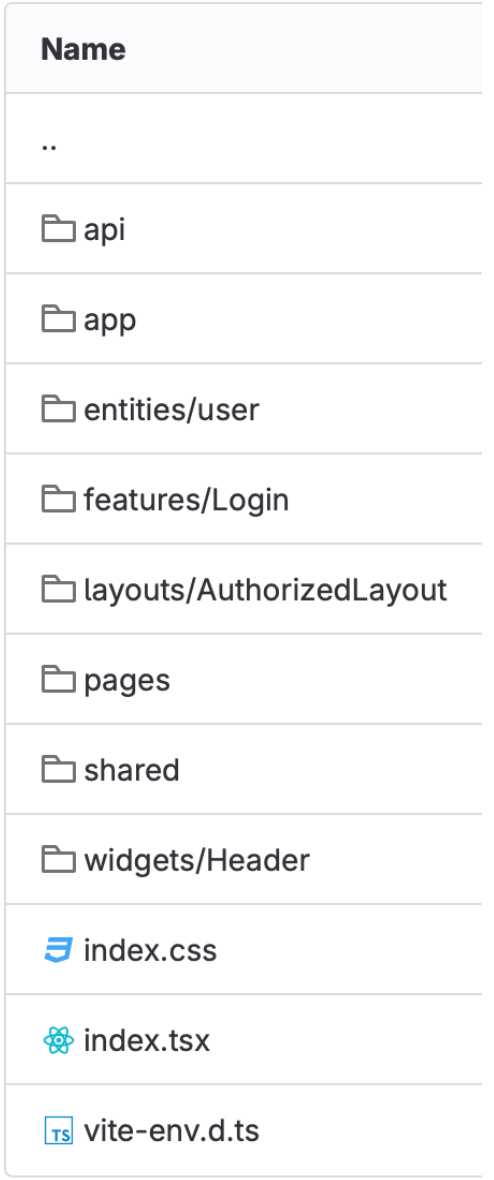
Пример FSD

- 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

```tsx
import { Box, CircularProgress } from '@mui/material';

import type { FC } from 'react';

export const Loader: FC = () => (
    <Box
        alignItems='center'
        display='flex'
        height='100vh'
        justifyContent='center'
        left='0'
        position='fixed'
        top='0'
        width='100%'
    >
        <CircularProgress />
    </Box>
);
```



- https://projects.iu5.bmstu.ru/iu5/infrastructure/department-services/templates/react