

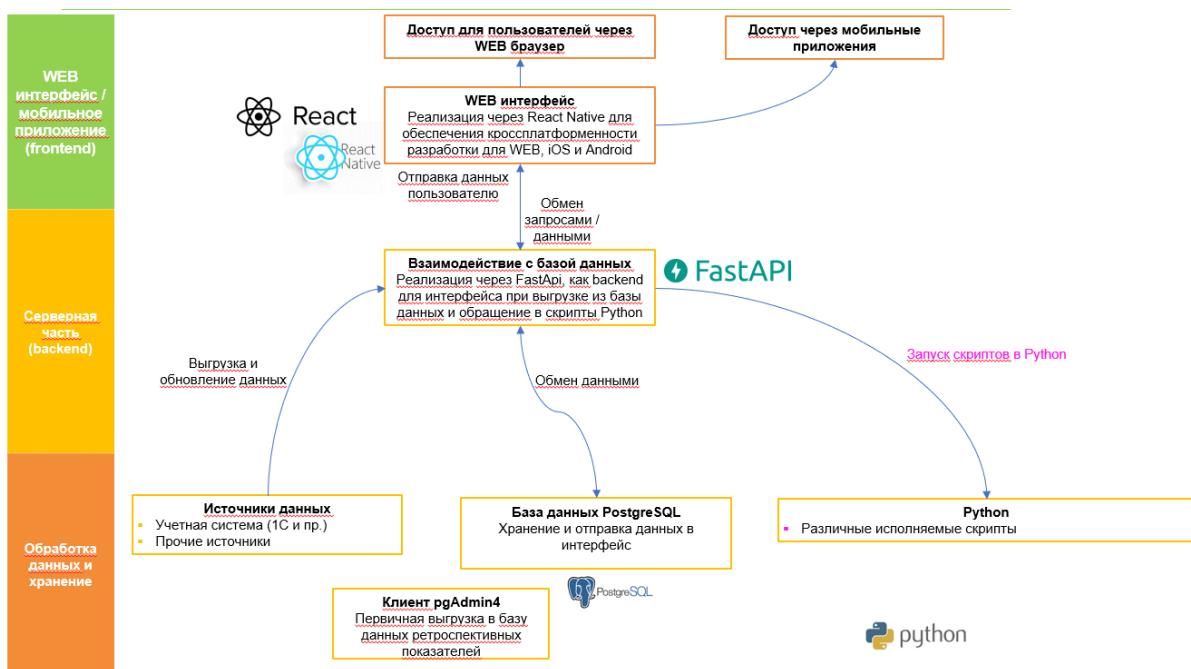
# Занятие 0: Full-Stack Development - технологический стек

[https://github.com/EugenePokh/full\\_stea\\_00\\_react\\_native\\_react\\_python\\_fastapi\\_posgre\\_sql.git](https://github.com/EugenePokh/full_stea_00_react_native_react_python_fastapi_posgre_sql.git)

Прежде чем мы погрузимся в код, кратко разберемся, на чем мы будем строить наши приложения и почему этот набор технологий — один из лучших для старта в 2025 году и beyond.

Мы будем использовать связку из двух основных направлений:

1. **Бэкенд (Backend) — «Мозги» приложения:** Python + FastAPI
2. **Фронтенд (Frontend) — «Лицо» приложения:** React + React Native
3. **Дополнительные скрипты:** Python
4. **Хранение данных:** PostgreSQL



## Фронтенд — React и React Native

### React: Самый популярный для Веба

- Библиотека (не фреймворк!) от Facebook для создания интерактивных пользовательских интерфейсов в вебе.
- **Самый удобный и востребованный: React является основным инструментом для фронтенд-разработки в большинстве современных компаний.** Изучив его, вы получаете наиболее востребованный навык на рынке.

- **Компонентный подход:** Вы создаете приложение как набор независимых блоков-компонентов (например, «Кнопка», «Меню», «Форма»). Это делает код переиспользуемым, понятным и легким в поддержке.
- **Огромное сообщество и богатая экосистема:** Существуют тысячи готовых решений, библиотек и инструментов для любых нужд.

### **React Native: Универсальное решение для мобильных платформ**

- Фреймворк, основанный на React, который позволяет создавать **нативные мобильные приложения** для iOS и Android, используя один и тот же код на JavaScript.
- **«Напиши один раз — запускай везде»:** Не нужно учить два разных языка (Swift/Kotlin) для двух платформ. До 95% кода можно использовать для обеих платформ одновременно.
- **Нативный вид и производительность:** В отличие от старых "веб-вьюшек", React Native компилируется в нативные компоненты, поэтому приложения выглядят и работают так же плавно, как и написанные на "чистом" Swift или Kotlin.
- **Экономия времени и ресурсов:** Разработка, тестирование и поддержка одной кодовой базы вместо двух — это огромная экономия.

**Итог по фронтенду:** Вы сможете создавать и современные веб-сайты, и полноценные мобильные приложения, используя одну и ту же философию разработки.

## **Бэкенд — Python и FastAPI**

### **Python: Универсальный**

- Высокоуровневый язык программирования, который славится своим простым и понятным синтаксисом. Читается почти как обычный английский.
- **Идеален для новичков:** Легко учить и понимать, что позволяет сосредоточиться на логике программирования, а не на сложном синтаксисе.
- **Мощь в Data Science и AI: На Python пишутся практически все современные нейросети и библиотеки машинного обучения** (TensorFlow, PyTorch). Изучая Python, вы получаете ключ к миру искусственного интеллекта.
- **Огромное сообщество и библиотеки:** Для любой задачи, скорее всего, уже есть готовая библиотека, что ускоряет разработку в разы.

### **FastAPI: Современный и скоростной фреймворк**

- **Что это?** Современный, быстрый (высокопроизводительный) веб-фреймворк для создания API на Python.
- **Самый современный стандарт:** Он создан с учетом всех последних возможностей Python. Код получается очень чистым и лаконичным.

- **Невероятная скорость:** По скорости работы он не уступает аналогам на Node.js и Go, что делает его одним из самых быстрых фреймворков в мире Python.
- **Автоматическая документация:** FastAPI автоматически генерирует интерактивную документацию для вашего API. Вы сразу можете тестировать свои методы прямо в браузере — это огромная экономия времени.
- **Встроенная проверка данных:** Очень сложно сделать ошибку, так как фреймворк сам проверяет типы данных, которые приходят и уходят.

**Итог по бэкенду:** Связка **Python + FastAPI** — это мощный, современный инструмент для создания серверной логики, который готов к задачам любой сложности, от простого блога до сложной системы с элементами ИИ.

---

## PostgreSQL: Надежная и мощная база данных

**PostgreSQL** — это современная, объектно-реляционная система управления базами данных с открытым исходным кодом. Мы выбираем её за её **непревзойденную надежность, строгое соответствие стандартам SQL и богатый набор возможностей**. В отличие от многих других решений, PostgreSQL поддерживает сложные типы данных, включая JSON, что позволяет ей работать как в классических реляционных моделях, так и гибко, подобно NoSQL-базам. Она идеально подходит для сложных и нагруженных приложений, обеспечивая целостность данных и мощные механизмы для выполнения самых сложных запросов. В связке с **Python/FastAPI** она создает невероятно сильный и отказоустойчивый бэкенд, готовый к масштабированию и любым бизнес-требованиям.

**Общий итог:** Мы выбрали этот стек, потому что он:

1. **Современный и востребованный.** Учите то, что используют ведущие компании прямо сейчас.
2. **Универсальный.** Позволяет вам стать Full-Stack разработчиком, который умеет делать всё: от сервера (Python/FastAPI) и веб-интерфейса (React) до мобильных приложений (React Native).
3. **Эффективный.** Эти инструменты позволяют разрабатывать сложные приложения быстрее и с меньшим количеством ошибок.
4. **Мощный.** Связка открывает двери в самые перспективные области, такие как веб-разработка, мобильная разработка и искусственный интеллект.

# **Занятие 1: Начало работы. Настройка окружения разработчика**

[https://github.com/EugenePokh/full\\_steam\\_01\\_environment.git](https://github.com/EugenePokh/full_steam_01_environment.git)

Прежде чем мы напишем первую строку кода, важно правильно подготовить рабочее место. Настройка правильного окружения — это как заложить прочный фундамент для будущего здания: оно позволит нам работать эффективно, учиться новому и легко справляться с трудностями. Сегодня мы последовательно установим все необходимые инструменты.

## **Наш план действий:**

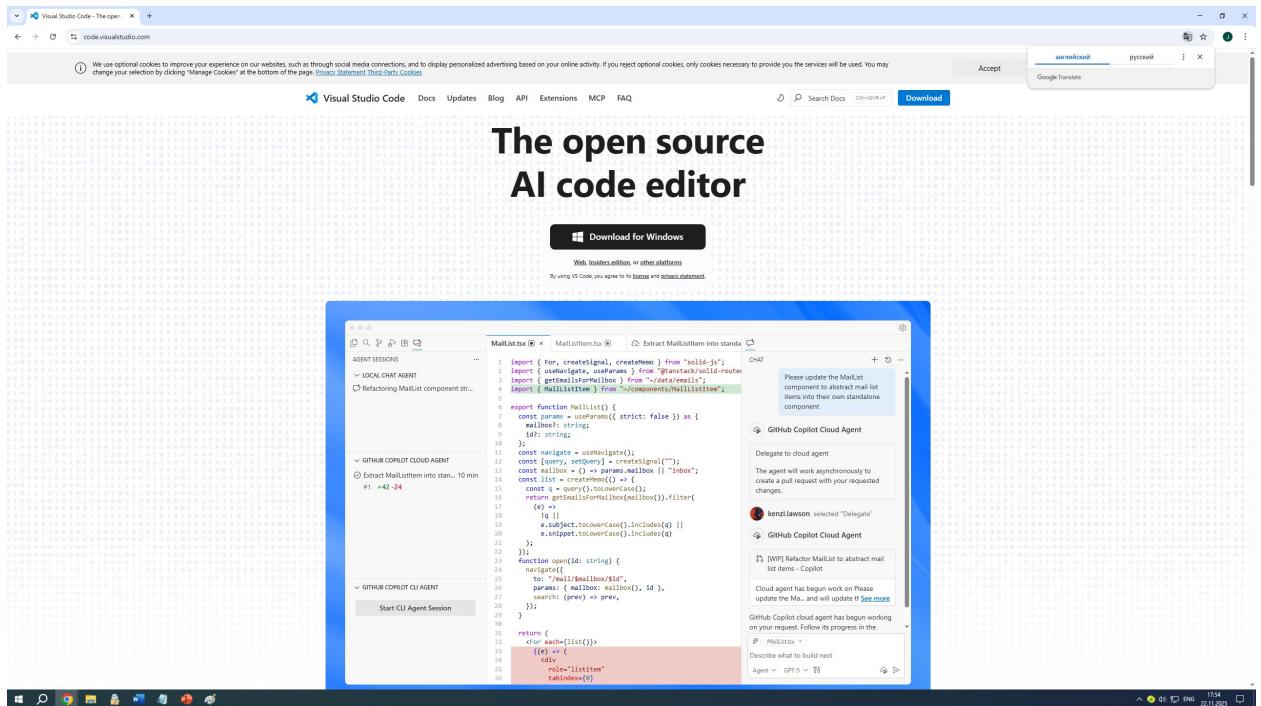
- Установим редактор кода и познакомимся с системой контроля версий.
  - Создадим своё цифровое рабочее пространство на GitHub.
  - Настроим доступ к современным инструментам для обучения — AI-ассистентам.
  - Подготовим всё необходимое для старта в мире универсальной / мобильной разработки с React Native.
- 

## **Система для работы: Visual Studio Code**

Нашим главным инструментом будет редактор кода. Это не просто блокнот, а умная рабочая среда, которая будет помогать нам писать код, подсвечивать ошибки и управлять проектом.

Мы выбираем **Visual Studio Code (VS Code)** — один из самых популярных и мощных редакторов в мире.

- **Ссылка для установки:** <https://code.visualstudio.com/>



## • Почему именно он?

- **Бесплатный и популярный:** Это стандарт для современных разработчиков.
- **Огромное количество расширений:** Для любого языка или технологии найдётся плагин, который сделает работу удобнее.
- **Встроенный терминал:** Позволяет запускать команды, не переключаясь между окнами.
- **Отличная интеграция с Git:** Позволяет работать с системой контроля версий прямо из редактора.

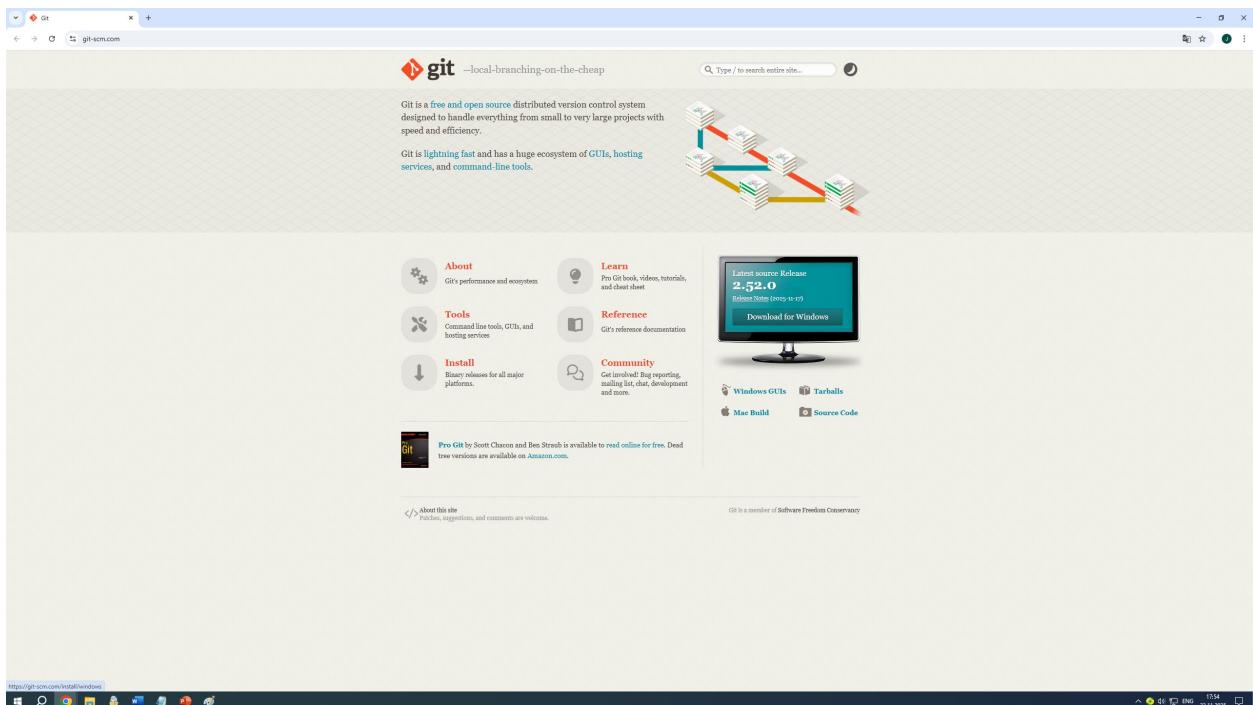
**Практический шаг:** Скачайте и установите VS Code. После запуска, вы увидите чистый и современный интерфейс, который мы будем наполнять жизнью в следующих занятиях.

## Установка Git и регистрация на GitHub

Любой серьёзный проект начинается с системы контроля версий. Проще говоря, это машинка времени для вашего кода.

**Git** — это система, которая отслеживает все изменения в ваших файлах. Она позволяет "сохраняться" в ключевые моменты, чтобы вы всегда могли вернуться к рабочей версии, если что-то пошло не так.

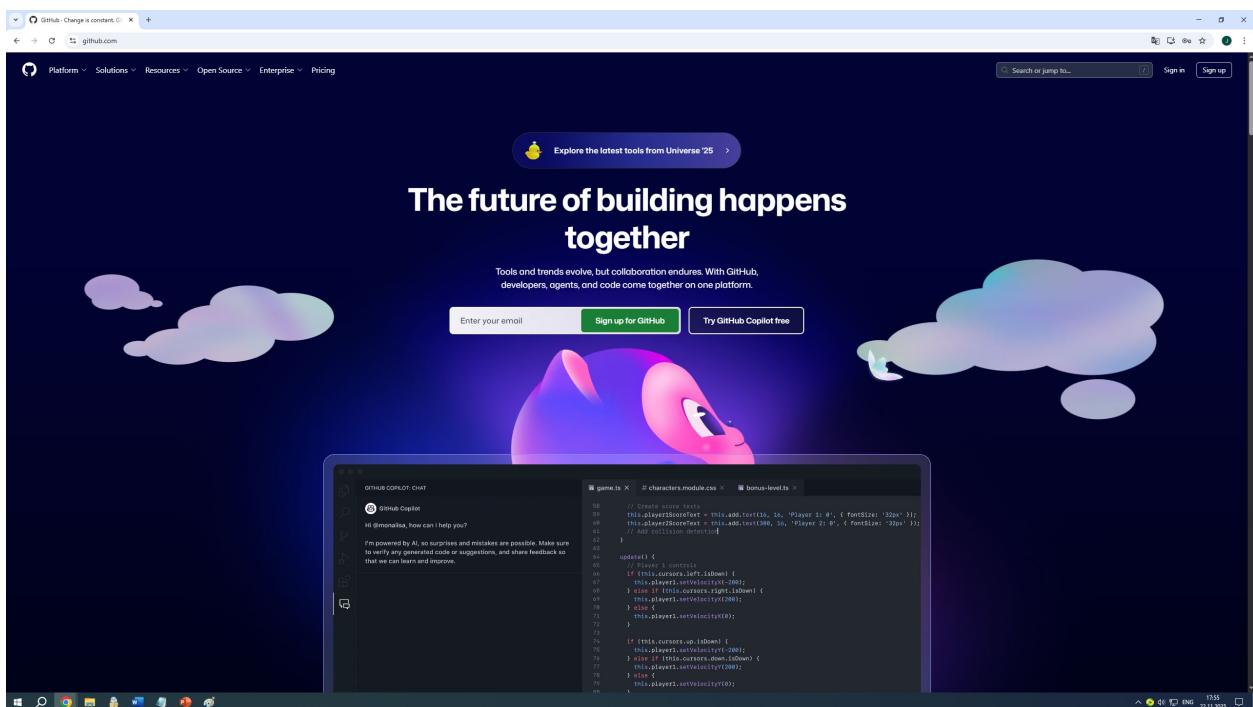
- **Ссылка для установки Git:** <https://git-scm.com/>



- **Что делаем?** Скачиваем и устанавливаем с настройками по умолчанию.

**GitHub** — это платформа для хранения кода. Это своего рода «социальная сеть для программистов», где вы можете сохранять свои проекты, делиться ими и работать в команде.

- **Ссылка для регистрации:** <https://github.com/>



## Практические шаги:

1. Зарегистрируйтесь на GitHub, используя любую удобную почту.

- После входа в аккаунт, создайте свой первый репозиторий (хранилище для кода). Для этого нажмите на кнопку «**Repositories**» в верхней части экрана, а затем — «**New repository**».
- Дайте репозиторию имя (например, `my-first-app`) и создайте его.

Теперь у вас есть своё место в мире разработки, куда мы скоро начнём выкладывать наш код.

---

## Работа через ИИ-ассистентов

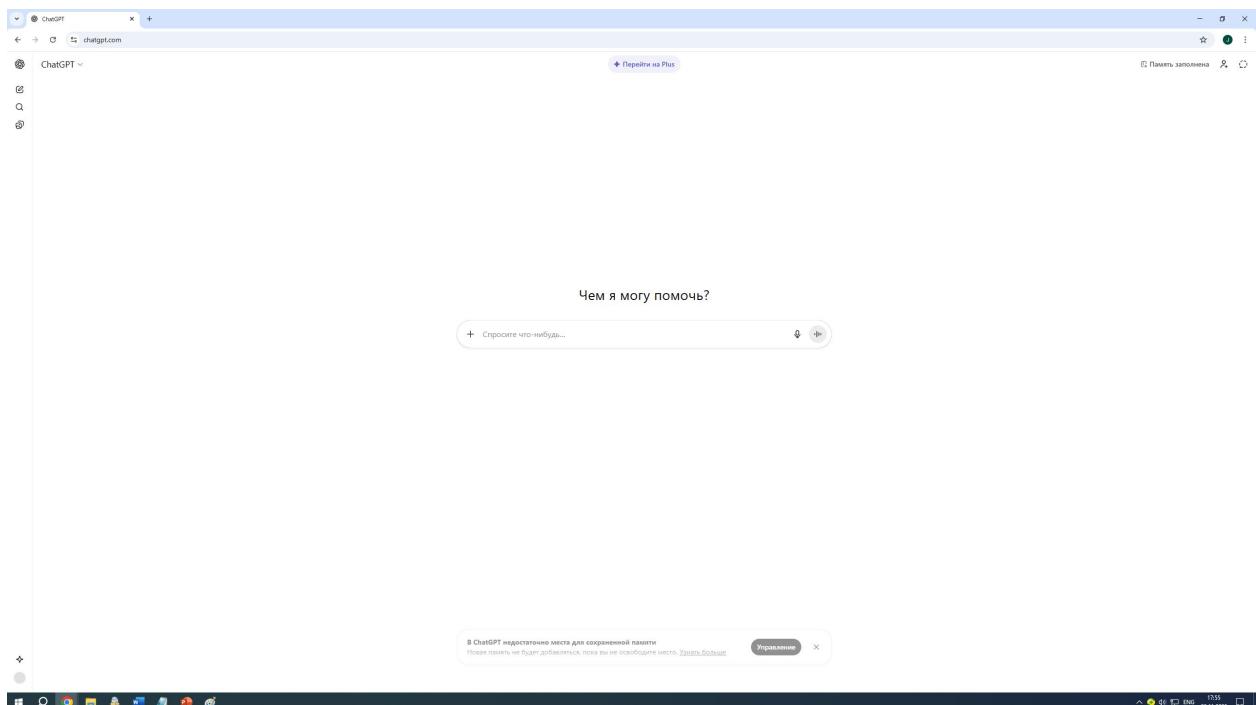
Сегодня у нас есть уникальное преимущество — мощные AI-инструменты, которые выступают в роли личного наставника 24/7. Они помогут объяснить сложные **концепции**, найти ошибку в коде или предложить разные варианты решения задачи.

Для стабильного доступа к этим сервисам может потребоваться VPN.

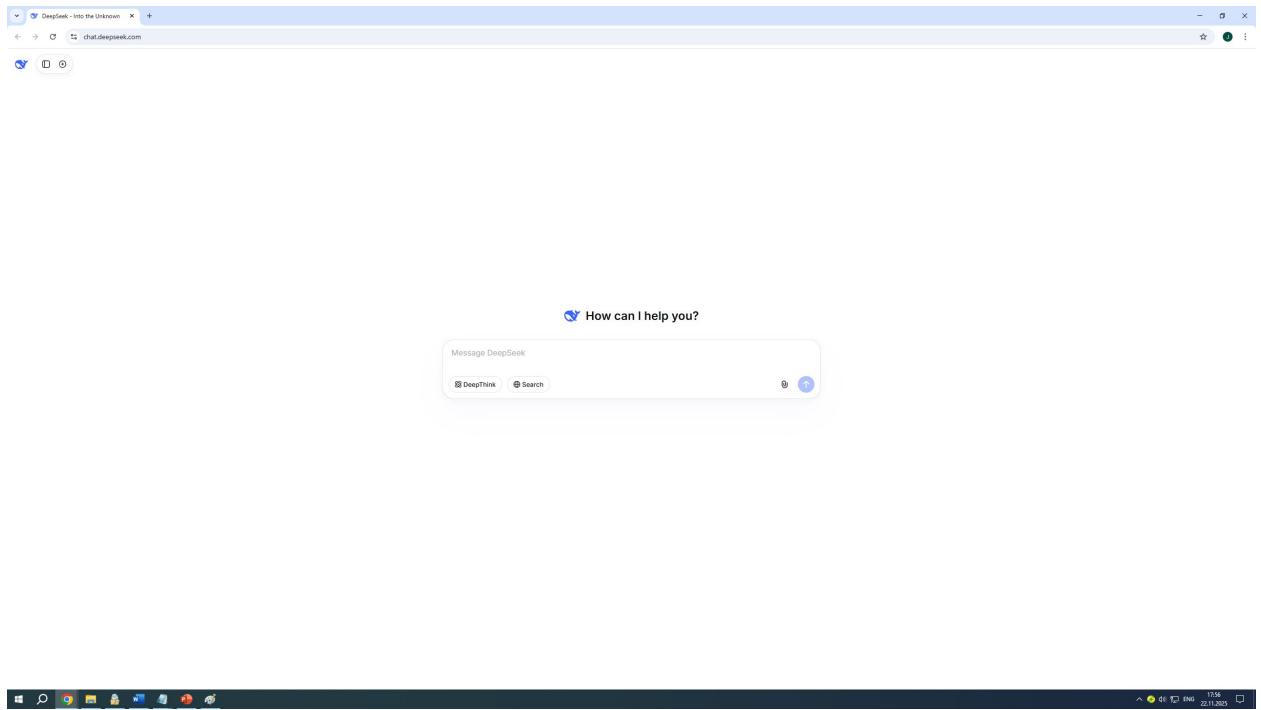
- VPN для ПК:** Удобно запускать через приложения вроде **NekoBox** (обязательно включите в настройках «**Режим TUN**» для корректной работы).

### Практические шаги по регистрации:

- Установите и активируйте VPN.
- Зарегистрируйтесь в ключевых сервисах (рекомендую использовать почтовые ящики `@gmail.com` или `@outlook.com`):
  - OpenAI ChatGPT:** <https://chatgpt.com/>



- **DeepSeek:** <https://chat.deepseek.com/>



Эти ассистенты станут вашими верными помощниками на всём пути обучения. Не стесняйтесь задавать им любые вопросы.

---

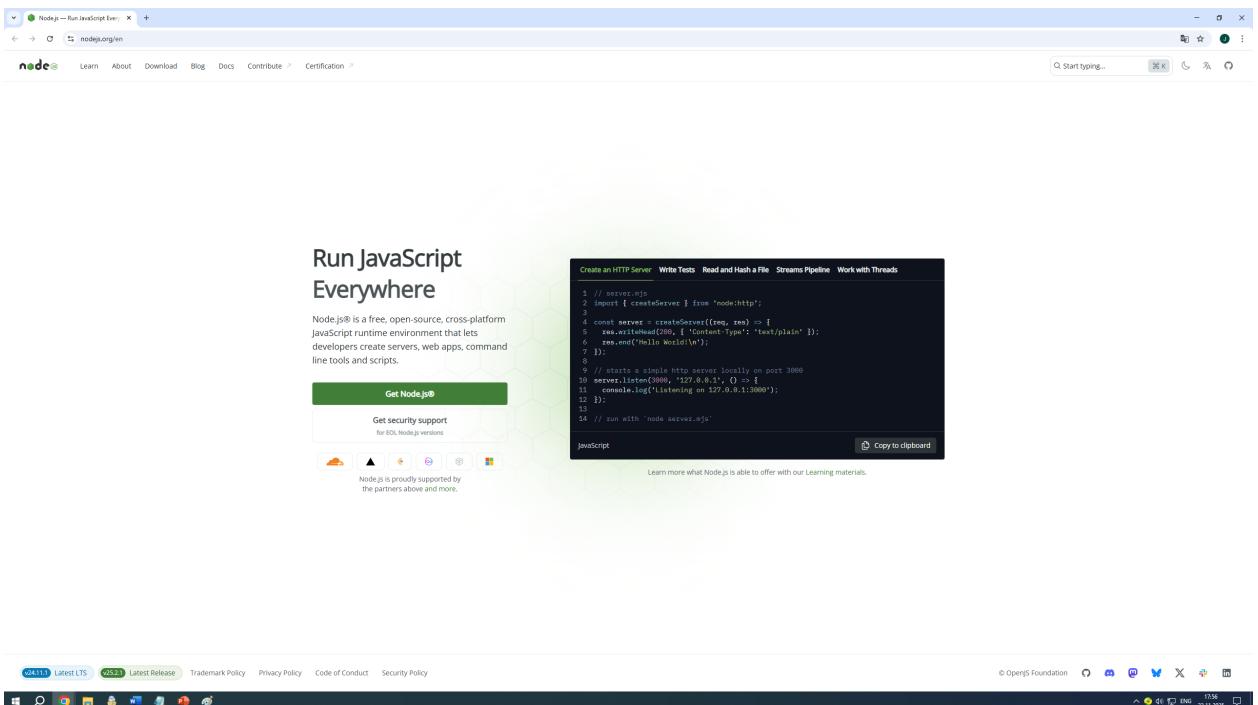
## Подготовка к React Native разработке

Мы подошли к установке ключевых технологий для создания мобильных приложений.

### Установка Node.js

Node.js — это среда выполнения для JavaScript. Она позволяет запускать JavaScript-код outside браузера, что является основой для многих инструментов разработки, включая React Native.

- **Ссылка для установки:** <https://nodejs.org/en/>

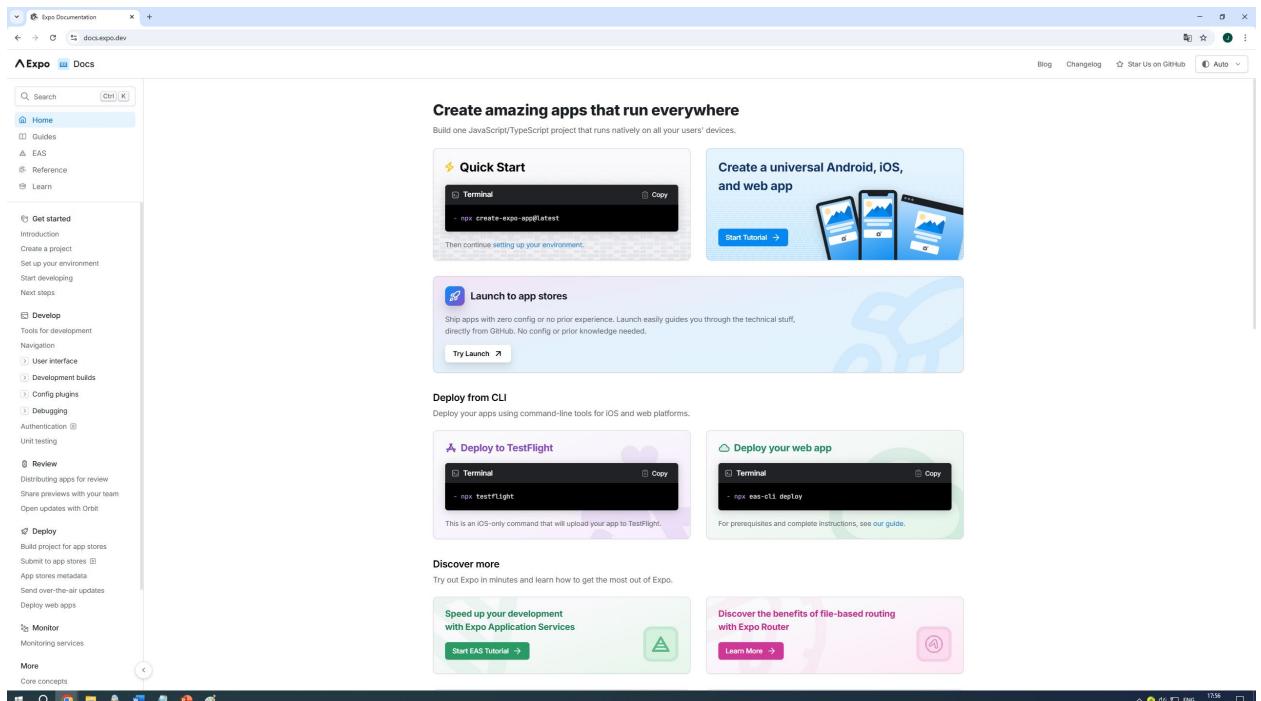


- **Что делаем?** Скачайте и установите **LTS-версию (Long-Term Support)**. Она является самой стабильной и рекомендованной к использованию.

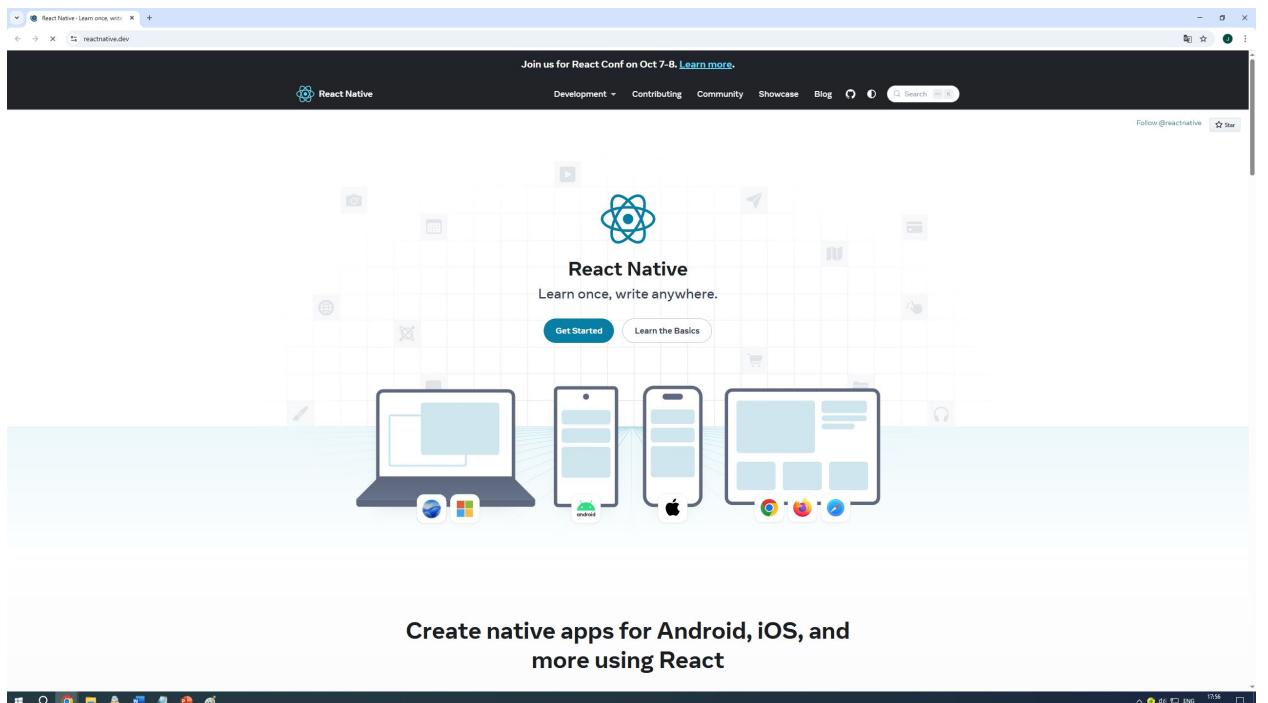
## Основные порталы для разработки

С самого начала полезно знать главные источники информации:

- **Документация Expo:** <https://docs.expo.dev/> — это фреймворк и платформа, которые значительно упрощают создание приложений на React Native.



- **Официальный сайт React Native:** <https://reactnative.dev/> — здесь находится официальная документация, руководства и новости.



## Занятие 2: Создание и запуск первого React Native приложения

[https://github.com/EugenePokh/full\\_stea 02 react native init.git](https://github.com/EugenePokh/full_stea 02 react native init.git)

В этом занятии мы перейдём от теории к практике и создадим наше первое мобильное приложение. Мы настроим виртуальное мобильное устройство, освоим базовые команды в терминале и, наконец, увидим результат своей работы в браузере и на телефоне.

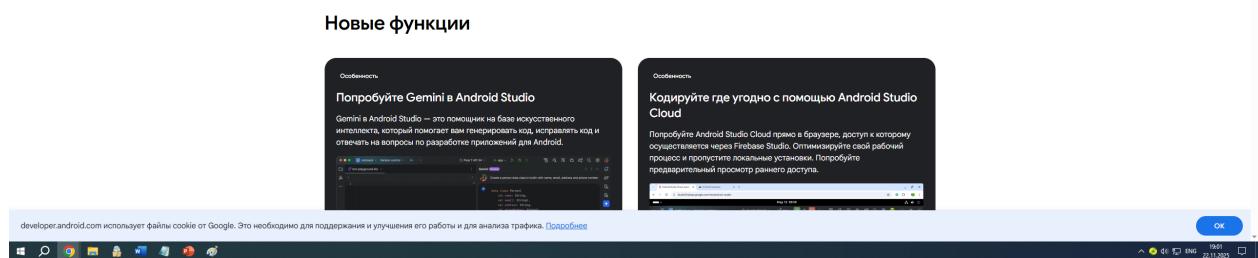
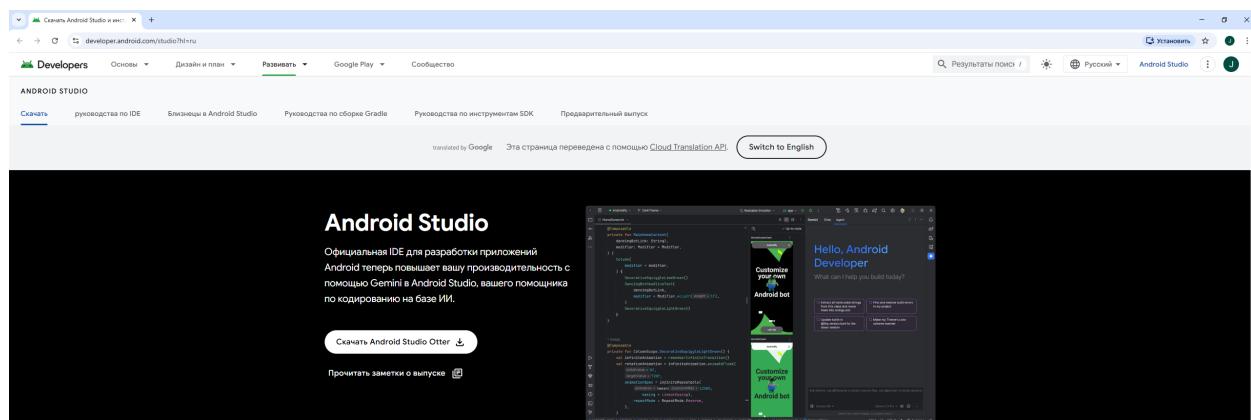
### Установка Android Studio (для пользователей Windows)

Хотя мы можем тестировать наше приложение прямо в браузере, для полноценной разработки под Android нам понадобится официальная среда от Google.

#### Что это такое?

**Android Studio** — это основная среда разработки (IDE) для создания нативных приложений под Android. Нам она нужна в первую очередь для доступа к **виртуальному устройству (эмуратору)** — программе, которая точно имитирует работу реального смартфона с Android на вашем компьютере.

- Ссылка для установки: <https://developer.android.com/studio>



- **Важный момент:** Этот шаг является *необязательным* на начальном этапе. Мы можем успешно разрабатывать и тестировать приложение в веб-браузере или на своём физическом телефоне через приложение Expo. Установите Android Studio, если вы хотите иметь полноценную среду для тестирования под Android.
- 

## Создание проекта React Native

Теперь приступим к самому главному — созданию заготовки для нашего приложения.

### Открываем рабочую директорию в VS Code

1. Запустите **Visual Studio Code**.
2. Через меню `File -> Open Folder...` откройте папку, где вы планируете хранить все свои учебные проекты.

### Работа с терминалом

Терминал — это ваш текстовый интерфейс для взаимодействия с компьютером. В VS Code его можно открыть через меню `Terminal -> New Terminal`.

### Основные команды навигации в терминале:

- `cd ..` — переместиться на один уровень вверх (выйти из текущей папки).
- `cd имя_папки` — переместиться внутрь указанной папки.

### Проверяем установку окружения

Прежде чем создавать проект, убедимся, что всё установлено корректно. Введите в терминале две команды:

```
bash
node --version
npm --version
```

Если в ответ вы увидите номера версий (например, `v18.17.0` и `9.6.7`), значит, Node.js и его менеджер пакетов `npm` готовы к работе.

### Создаём новый проект

Для создания нового приложения мы используем мощную команду:

```
bash
npx create-expo-app@latest my-first-app --template blank
```

- `npx` — утилита для запуска пакетов из npm-реестра.

- `create-expo-app@latest` — инструмент, который скачает последнюю версию заготовки для приложения.
- `my-first-app` — это название вашего проекта (вы можете назвать его как угодно, используя только латинские буквы и нижние подчёркивания).
- `--template blank` — создаёт чистый проект без лишнего предустановленного кода.

После выполнения команды в терминале вы увидите, как создаётся новая папка с вашим проектом и устанавливаются все необходимые зависимости. Этот процесс может занять несколько минут.

## Установка поддержки Web (браузера)

По умолчанию проект готов для запуска на iOS и Android. Чтобы мы могли сразу открыть его в браузере, выполните дополнительную команду, предварительно перейдя в папку проекта (`cd my-first-app`):

```
bash
npx expo install react-dom react-native-web
```

Эта команда устанавливает специальные пакеты, которые позволяют отображать React Native приложение в веб-браузере.

---

## Запуск и остановка проекта

### Запуск проекта

Самое волнующее — увидеть своё приложение вживую! Перейдите в терминале в корневую папку вашего проекта (если вы ещё не в ней) и выполните:

```
bash
npx expo start
```

Эта команда запускает **метро-бандлер** — специальный сервер разработки, который собирает ваш код и готовит его для запуска на устройствах. В терминале появится QR-код, а в браузере откроется панель управления **Expo DevTools**.

### Способы запуска приложения:

1. **В браузере (самый простой способ):**
  - В терминале, где работает `expo start`, нажмите на клавиатуре клавишу `w`.
  - В браузере автоматически откроется новая вкладка с вашим приложением.
  - Чтобы увидеть, как приложение выглядит на телефоне, нажмите `F12` и в инструментах разработчика выберите режим эмуляции мобильного устройства (например, "iPhone 12").
2. **На физическом телефоне (рекомендуется):**

- Установите официальное приложение **Expo Go** из App Store (для iPhone) или Play Market (для Android).
  - В приложении Expo Go отсканируйте QR-код из терминала или из вкладки DevTools в браузере.
  - Ваше приложение загрузится и запустится прямо на телефоне!
3. **В эмуляторе Android (для Windows, если он установлен):**
- Убедитесь, что у вас запущен эмулятор Android Studio.
  - В терминале, где работает `expo start`, нажмите клавишу `a`.

### **Остановка проекта**

Когда вы закончите работу, необходимо корректно остановить сервер разработки. Для этого в терминале, где работает команда `npm expo start`, нажмите сочетание клавиш `Ctrl + C`.

## Занятие 3: Создание структуры приложения и настройка навигации

[https://github.com/EugenePokh/full\\_steam\\_03\\_basic.git](https://github.com/EugenePokh/full_steam_03_basic.git)

В этом занятии мы превратим наш чистый проект в многостраничное приложение с навигацией. Мы создадим экраны логина и меню, свяжем их между собой и научимся сохранять прогресс в системе контроля версий Git.

---

### Установка необходимых зависимостей

Любое современное приложение строится на сторонних библиотеках, которые решают типовые задачи. Давайте установим три ключевые зависимости для нашего проекта.

Откройте терминал в корневой папке вашего проекта и выполните команды:

```
bash
# Библиотека для навигации между экранами
npm install @react-navigation/native @react-navigation/native-stack
```

```
# Локальное хранилище для сохранения данных (например, токена авторизации)
npm install @react-native-async-storage/async-storage
```

```
# Дополнительные модули и компоненты от Expo
npm install expo
```

### Что мы установили:

- `@react-navigation/native-stack` — предоставляет простой способ организации навигации в стиле стека (новый экран "ложится" поверх предыдущего).
  - `@react-native-async-storage/async-storage` — асинхронное хранилище, аналог `localStorage` в вебе. Будет использоваться для запоминания пользователя.
  - `expo` — обновляет и добавляет различные полезные модули фреймворка.
- 

### Создание структуры проекта

Хорошая организация файлов — залог поддерживаемости кода. Давайте создадим логичную структуру папок и файлов. В корне вашего проекта создайте папку `src`, а внутри нее — два файла для наших экранов.

Ваша структура проекта должна выглядеть так:

```
text
my-app/
└── assets/
    ├── adaptive-icon.png
    ├── favicon.png
    ├── icon.png
    └── splash.png
└── src/
    ├── login.js      # Экран входа в приложение
    └── menu.js       # Главный экран после успешного входа
└── .gitignore
└── App.js          # Главный файл приложения, где настраивается навигация
└── app.json
└── index.js
└── package-lock.json
└── package.json
```

---

## Создание компонентов экранов

Теперь нам нужно создать основные компоненты нашего приложения. Мы подготовим каркасы для каждого экрана.

### 1. Экран авторизации: `src/login.js`

Этот компонент будет содержать форму для ввода логина и пароля. Здесь вам предстоит реализовать:

- Поля ввода для логина и пароля
- Кнопку для входа
- Логику проверки
- Навигацию на главный экран после успешной авторизации

```
import React, { useState, useEffect } from 'react';
import {
  View,
  Text,
  TextInput,
```

```

TouchableOpacity,
StyleSheet,
Alert
} from 'react-native';
import AsyncStorage from '@react-native-async-storage/async-storage';

export default function Login({ navigation }) {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [remember, setRemember] = useState(false);

  // При запуске проверяем сохранённые данные
  useEffect(() => {
    const loadSavedData = async () => {
      try {
        const savedUsername = await AsyncStorage.getItem('username');
        const savedPassword = await AsyncStorage.getItem('password');
        if (savedUsername && savedPassword) {
          setUsername(savedUsername);
          setPassword(savedPassword);
          setRemember(true);
        }
      } catch (error) {
        console.error('Ошибка загрузки данных:', error);
      }
    };
    loadSavedData();
  }, []);

  const handleLogin = async () => {
    if (username === 'user1' && password === 'password1') {
      // Если выбрана галочка – сохраняем данные
      if (remember) {
        await AsyncStorage.setItem('username', username);
        await AsyncStorage.setItem('password', password);
      } else {
        await AsyncStorage.removeItem('username');
        await AsyncStorage.removeItem('password');
      }
      navigation.navigate('Menu');
    } else {
      Alert.alert('Ошибка', 'Неверное имя пользователя или пароль');
    }
  };

  return (
    <View style={styles.container}>
      <Text style={styles.title}>Вход в систему</Text>

      <TextInput
        style={styles.input}

```

```

        placeholder="Логин"
        value={username}
        onChangeText={setUsername}
        autoCapitalize="none"
    />

    <TextInput
        style={styles.input}
        placeholder="Пароль"
        value={password}
        onChangeText={setPassword}
        secureTextEntry
    />

    <TouchableOpacity
        style={styles.checkboxContainer}
        onPress={() => setRemember(!remember)}
    >
        <View style={[styles.checkbox, remember && styles.checkboxChecked]} />
        <Text style={styles.checkboxLabel}>Запомнить меня</Text>
    </TouchableOpacity>

    <TouchableOpacity style={styles.button} onPress={handleLogin}>
        <Text style={styles.buttonText}>Войти</Text>
    </TouchableOpacity>
</View>
);

}

const styles = StyleSheet.create({
    container: {
        flex: 1,
        justifyContent: 'center',
        padding: 24,
        backgroundColor: '#fff',
    },
    title: {
        fontSize: 26,
        fontWeight: 'bold',
        color: '#333',
        textAlign: 'center',
        marginBottom: 32,
    },
    input: {
        borderWidth: 1,
        borderColor: '#aaa',
        borderRadius: 10,
        padding: 12,
        marginBottom: 16,
        fontSize: 16,
    },
},

```

```

button: {
  backgroundColor: '#FF69B4',
  paddingVertical: 14,
  borderRadius: 10,
  alignItems: 'center',
  marginTop: 16,
},
buttonText: {
  color: '#fff',
  fontWeight: 'bold',
  fontSize: 18,
},
checkboxContainer: {
  flexDirection: 'row',
  alignItems: 'center',
},
checkbox: {
  width: 22,
  height: 22,
  borderRadius: 4,
  borderWidth: 1,
  borderColor: '#555',
  marginRight: 10,
},
checkboxChecked: {
  backgroundColor: '#FF69B4',
},
checkboxLabel: {
  fontSize: 16,
},
});

```

## 2. Главный экран: `src/menu.js`

Это основной экран приложения, который открывается после успешного входа. На этом экране находится меню:

```

import React, { useState } from 'react';
import { StyleSheet, Text, View, TouchableOpacity, ScrollView, Image,
ImageBackground } from 'react-native';
import { useNavigation } from '@react-navigation/native';
//import AsyncStorage from '@react-native-async-storage/async-storage';
import { Ionicons } from '@expo/vector-icons';

const CropsScreen = () => {
  const navigation = useNavigation();
//  const [resetModalVisible, setResetModalVisible] = useState(false);

```

```

React.useLayoutEffect(() => {
  navigation.setOptions({
    headerLeft: () => (
      <TouchableOpacity
        onPress={() => navigation.navigate('Login')}
        style={{ marginLeft: 15 }}
      >
        <Ionicons name="arrow-back" size={24} color="white" />
      </TouchableOpacity>
    ),
  });
}, [navigation]);

const crops = [
  { title: "01", source: require('../assets/icon.png'), navigateTo: 'Login' },
  { title: "02", source: require('../assets/icon.png'), navigateTo: 'Login' },
];

const renderCard = (crop, index) => (
  <TouchableOpacity
    key={index}
    style={[
      styles.card,
      crop.isDestructive && styles.destructiveCard
    ]}
    onPress={crop.action || (() => navigation.navigate(crop.navigateTo))}
  >
    <View style={styles.imageContainer}>
      <Image source={crop.source} style={styles.image} />
      <View style={styles.overlay}>
        <Text style={styles.text}>{crop.title}</Text>
      </View>
    </View>
  </TouchableOpacity>
);

return (
  <ImageBackground
    source={require('../assets/favicon.png')}
    style={styles.background}
    resizeMode="stretch">
    <View style={styles.screenContainer}>
      <ScrollView contentContainerStyle={styles.container}>
        <View style={styles.grid}>
          {crops.map((crop, index) => renderCard(crop, index))}
        </View>
      </ScrollView>
    </View>
  </ImageBackground>
);

```

```
};

const styles = StyleSheet.create({
  background: {
    flex: 1,
    width: '100%',
    height: '100%',
    justifyContent: 'center',
    alignItems: 'center',
  },
  screenContainer: {
    flex: 1,
    backgroundColor: 'rgba(206, 204, 204, 0.7)'
  },
  container: {
    flexGrow: 1,
    paddingVertical: 20,
    justifyContent: 'center',
    alignItems: 'center',
  },
  grid: {
    flexDirection: 'row',
    flexWrap: 'wrap',
    justifyContent: 'space-around',
  },
  card: {
    width: '45%',
    marginBottom: 20,
    alignItems: 'center',
  },
  destructiveCard: {
    opacity: 0.8,
  },
  imageContainer: {
    position: 'relative',
  },
  image: {
    width: 150,
    height: 150,
    borderRadius: 10,
  },
  overlay: {
    position: 'absolute',
    bottom: 0,
    left: 0,
    right: 0,
    backgroundColor: 'rgba(0, 0, 0, 0.5)',
    borderBottomLeftRadius: 10,
    borderBottomRightRadius: 10,
  },
  text: {
```

```
        color: 'rgba(255, 255, 255, 0.7)',  
        fontSize: 16,  
        textAlign: 'center',  
        paddingVertical: 5,  
    },  
    footerText: {  
        fontSize: 12,  
        color: 'rgba(255, 255, 255, 0.7)',  
        marginTop: 5,  
        textAlign: 'center',  
    },  
    // Стили для модального окна  
    modalOverlay: {  
        flex: 1,  
        justifyContent: 'center',  
        alignItems: 'center',  
        backgroundColor: 'rgba(0,0,0,0.5)',  
    },  
    modalContent: {  
        width: '80%',  
        backgroundColor: 'white',  
        borderRadius: 20,  
        padding: 25,  
        alignItems: 'center',  
        elevation: 5,  
    },  
    modalTitle: {  
        fontSize: 22,  
        fontWeight: 'bold',  
        marginBottom: 15,  
        color: '#333',  
    },  
    modalText: {  
        fontSize: 16,  
        textAlign: 'center',  
        marginBottom: 25,  
        color: '#555',  
        lineHeight: 22,  
    },  
    modalButtons: {  
        flexDirection: 'row',  
        justifyContent: 'space-around',  
        width: '100%',  
    },  
    modalButton: {  
        flexDirection: 'row',  
        alignItems: 'center',  
        justifyContent: 'center',  
        paddingVertical: 12,  
        paddingHorizontal: 20,  
        borderRadius: 10,
```

```

        width: '40%',
    },
cancelButton: {
    backgroundColor: '#f1f1f1',
    borderWidth: 1,
    borderColor: '#ddd',
},
confirmButton: {
    backgroundColor: '#e74c3c',
},
buttonIcon: {
    width: 20,
    height: 20,
    marginRight: 8,
},
buttonText: {
    fontSize: 16,
    fontWeight: 'bold',
},
});
}

export default CropsScreen;

```

### 3. Настройка навигации: App.js

Это главный файл приложения, где мы свяжем все наши экраны вместе. Вам нужно будет:

- Импортировать созданные компоненты экранов
- Настроить навигационный стек
- Определить начальный экран приложения
- Реализовать логику переключения между экранами

```

import { StatusBar } from 'expo-status-bar';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/native-stack';
import Login from './src/login';
import Menu from './src/menu';

const Stack = createStackNavigator();

console.log('Web app is loading');

export default function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>

```

```
initialRouteName="Login"
screenOptions={{
  headerStyle: {
    backgroundColor: '#FF69B4', // Ярко розовый фон заголовка
    height: 70, // Уменьшенная высота заголовка
  },
  headerTintColor: '#FFFFFF', // Белый цвет текста заголовка
  headerTitleStyle: {
    fontSize: 18, // Уменьшенный размер текста заголовка
    fontWeight: 'bold',
  },
}}
>
<Stack.Screen
  name="Login"
  component={Login}
  options={{ title: 'Добро пожаловать!' }}
/>
<Stack.Screen
  name="Menu"
  component={Menu}
  options={{ title: 'Меню' }}
/>
</Stack.Navigator>
<StatusBar style="auto" />
</NavigationContainer>
);
}
```

---

## Сохранение проекта в GitHub

Когда базовая структура приложения готова, важно сохранить нашу работу в системе контроля версий.

Настраиваем файл .gitignore

```
# Learn more https://docs.github.com/en/get-started/getting-started-with-git/ignoring-files

# dependencies
node_modules/

# Expo
.expo/
dist/
web-build/
expo-env.d.ts
```

```
# Native
.kotlin/
*.orig.*
*.jks
*.p8
*.p12
*.key
*.mobileprovision

# Metro
.metro-health-check*

# debug
npm-debug.*
yarn-debug.*
yarn-error.*

# macOS
.DS_Store
*.pem

# local env files
.env*.local

# typescript
*.tsbuildinfo

# generated native folders
/ios
/android
```

```
bash
# Инициализация Git-репозитория в папке проекта
git init

# Подключение к удаленному репозиторию на GitHub
git remote add origin https://github.com/ ваш-логин/название-репозитория.git

# Создание и переключение на основную ветку
git branch -M main

# Добавление всех файлов проекта в staging area
git add .

# Создание коммита с описанием изменений
```

```
git commit -m "Первая версия приложения: базовая структура, экраны логина и меню"

# Отправка кода в удаленный репозиторий
git push origin main
```

---

**Итог занятия:** Вы успешно создали структуру многостраничного приложения, установили необходимые библиотеки для навигации и хранения данных, подготовили компоненты экранов и сохранили прогресс в Git. В следующем занятии мы наполним наши компоненты реальной логикой и функционалом.

## Занятие 4: Расширение функциональности приложения

[https://github.com/EugenePokh/full\\_steam\\_04\\_map\\_swipe.git](https://github.com/EugenePokh/full_steam_04_map_swipe.git)

В этом занятии мы значительно расширим возможности нашего приложения, добавив два важных экрана: интерактивную карту и экран свайпов. Мы также научимся управлять подключением к удаленным репозиториям.

---

### Установка дополнительных зависимостей

Для реализации новых функций нам потребуются специализированные библиотеки.

#### Для работы с картами:

```
bash
# Нативные компоненты для отображения карт и работы с геолокацией
expo install react-native-maps expo-location
```

```
# Библиотека для работы с картами в веб-версии (альтернативный вариант)
npm install react-leaflet leaflet
```

#### Что мы устанавливаем:

- `react-native-maps` — компонент для отображения интерактивных карт в мобильном приложении
- `expo-location` — модуль для работы с геолокацией устройства
- `react-leaflet` и `leaflet` — популярная библиотека для работы с картами в веб-среде

**Примечание:** Экран свайпов не требует установки дополнительных зависимостей, так как будет реализован на базе стандартных компонентов React Native.

---

## Обновление структуры проекта

Добавим новые экраны в нашу структуру. Теперь проект выглядит так:

```
text
my-app/
├── assets/
|   ├── adaptive-icon.png
|   ├── background.jpg
|   ├── favicon.png
|   ├── icon.png
|   ├── map_icon.jpg
|   ├── robot_01.jpg
|   ├── robot_02.jpg
|   ├── robot_03.jpg
|   ├── robot_04.jpg
|   ├── robot_05.jpg
|   ├── splash-icon.png
|   └── swipe_icon.jpg
├── src/
|   ├── login.js      # Экран авторизации
|   ├── map.js        # Новый экран с картой
|   ├── menu.js       # Главное меню приложения
|   └── swipe.js      # Новый экран свайпов
└── .gitignore
├── App.js           # Главный файл с навигацией
├── app.json
├── index.js
├── package-lock.json
└── package.json
```

---

## Создание новых экранов

### 1. Экран карты: `src/map.js`

Этот компонент будет отображать интерактивную карту с возможностью определения местоположения пользователя.

```
import React, { useState, useEffect, useRef } from "react";
import { Platform, View, Text, TouchableOpacity, StyleSheet } from "react-native";
```

```

export default function UniversalMapScreen() {
  const [mapReady, setMapReady] = useState(false);
  const [location, setLocation] = useState(null);
  const [error, setError] = useState(null);
  const [MapComponent, setMapComponent] = useState(null);
  const mapRef = useRef(null);

  // Динамическая загрузка компонентов карты
  useEffect(() => {
    if (Platform.OS === 'web') {
      // Для web используем Leaflet
      initWebMap();
      setMapComponent('web');
      setMapReady(true);
    } else {
      // Для мобильных платформ динамически импортируем react-native-maps
      loadNativeMaps();
    }
  }, []);

  const loadNativeMaps = async () => {
    try {
      // Динамический импорт чтобы избежать ошибок на web
      const ReactNativeMaps = await import('react-native-maps');

      // Создаем компонент карты для нативных платформ
      const NativeMap = ({ region, onMapReady, children }) => (
        <ReactNativeMaps.default
          ref={mapRef}
          style={styles.nativeMap}
          region={region}
          onMapReady={onMapReady}
          showsUserLocation={true}
        >
          {children}
        </ReactNativeMaps.default>
      );

      setMapComponent({
        Map: NativeMap,
        Marker: ReactNativeMaps.Marker
      });
      setMapReady(true);
    } catch (error) {
      console.log('react-native-maps not available:', error);
      setError('Карта недоступна на этом устройстве');
    }
  };
}

// Web карта с Leaflet

```

```

const initWebMap = () => {
  if (window.L && document.getElementById('map-container')) {
    createWebMap();
    return;
  }
  loadLeaflet();
};

const loadLeaflet = () => {
  if (window.L) {
    createWebMap();
    return;
  }

  const script = document.createElement('script');
  script.src = 'https://unpkg.com/leaflet@1.9.4/dist/leaflet.js';
  script.onload = () => {
    setTimeout(createWebMap, 100);
  };
  script.onerror = () => setError('Не удалось загрузить карту');
  document.head.appendChild(script);

  const link = document.createElement('link');
  link.rel = 'stylesheet';
  link.href = 'https://unpkg.com/leaflet@1.9.4/dist/leaflet.css';
  document.head.appendChild(link);
};

const createWebMap = () => {
  const mapContainer = document.getElementById('map-container');
  if (!mapContainer || !window.L) {
    setTimeout(createWebMap, 100);
    return;
  }

  try {
    if (window.leafletMap) window.leafletMap.remove();

    const map = window.L.map('map-container').setView([53.1959, 45.0183], 13);

    window.L.tileLayer('https://s.tile.openstreetmap.org/{z}/{x}/{y}.png', {
      attribution: '&copy; OpenStreetMap contributors',
      maxZoom: 19,
    }).addTo(map);

    window.leafletMap = map;
    mapRef.current = map;
    setMapReady(true);
    setError(null);
  } catch (err) {
    console.error('Leaflet init error:', err);
  }
};

```

```

        setError('Ошибка инициализации карты');
    }
};

const handleFindMe = async () => {
    if (Platform.OS === "web") {
        findWebLocation();
    } else {
        await findMobileLocation();
    }
};

const findWebLocation = () => {
    if (!navigator.geolocation) {
        alert("Геолокация не поддерживается браузером");
        return;
    }

    navigator.geolocation.getCurrentPosition(
        (position) => {
            const { latitude, longitude } = position.coords;
            setLocation({ latitude, longitude });
            centerWebMap(latitude, longitude);
        },
        (error) => {
            console.error('Geolocation error:', error);
            alert('Не удалось получить местоположение');
        }
    );
};

const centerWebMap = (lat, lng) => {
    if (!window.leafletMap) return;

    window.leafletMap.setView([lat, lng], 15);

    if (window.currentMarker) {
        window.leafletMap.removeLayer(window.currentMarker);
    }

    window.currentMarker = window.L.marker([lat, lng])
        .addTo(window.leafletMap)
        .bindPopup(`<div style="text-align: center;">
            <strong>Вы здесь!</strong><br>
            Широта: ${lat.toFixed(6)}<br>
            Долгота: ${lng.toFixed(6)}
        </div>
    `)
        .openPopup();
};

```

```

const findMobileLocation = async () => {
  try {
    const { requestForegroundPermissionsAsync, getCurrentPositionAsync } =
      await import('expo-location');

    const { status } = await requestForegroundPermissionsAsync();
    if (status !== "granted") {
      alert("Разрешение на доступ к геолокации отклонено");
      return;
    }

    const loc = await getCurrentPositionAsync({});
    const newLocation = {
      latitude: loc.coords.latitude,
      longitude: loc.coords.longitude,
    };
    setLocation(newLocation);
    setError(null);

    // Центрируем нативную карту
    if (mapRef.current) {
      mapRef.current.animateToRegion({
        ...newLocation,
        latitudeDelta: 0.05,
        longitudeDelta: 0.05,
      }, 1000);
    }
  } catch (err) {
    console.error("Location error:", err);
    setError('Ошибка получения местоположения');
  }
};

// Рендер нативной карты
const renderNativeMap = () => {
  if (!MapComponent || !MapComponent.Map) return null;

  const region = location
    ?
    {
      latitude: location.latitude,
      longitude: location.longitude,
      latitudeDelta: 0.05,
      longitudeDelta: 0.05,
    }
    :
    {
      latitude: 53.1959,
      longitude: 45.0183,
      latitudeDelta: 0.05,
      longitudeDelta: 0.05,
    };
};

```

```

        return (
          <MapComponent.Map
            region={region}
            onMapReady={() => setMapReady(true)}
          >
            {location && MapComponent.Marker && (
              <MapComponent.Marker
                coordinate={location}
                title="Вы здесь 🌐"
              />
            )}
          </MapComponent.Map>
        );
      };

    // Рендер web карты
    const renderWebMap = () => (
      <View style={styles.mapContainer}>
        <div
          id="map-container"
          style={styles.webMap}
        />

        {!mapReady && !error && (
          <View style={styles.center}>
            <Text style={styles.loadingText}>Загрузка карты...</Text>
          </View>
        )}
      </View>
    );

    return (
      <View style={styles.container}>
        {/* Рендер карты в зависимости от платформы */}
        {Platform.OS === 'web' ? renderWebMap() : renderNativeMap()}

        {/* Кнопка "Найти меня" */}
        <TouchableOpacity
          style={[
            styles.button,
            !mapReady && styles.buttonDisabled
          ]}
          onPress={handleFindMe}
          disabled={!mapReady}
        >
          <Text style={styles.buttonText}>Найти меня</Text>
        </TouchableOpacity>
      </View>
    );
  }
}

```

```

    {/* Отображение ошибок */}
    {error && (
      <View style={styles.errorContainer}>
        <Text style={styles.errorText}>{error}</Text>
        {Platform.OS === 'web' && (
          <TouchableOpacity
            style={[styles.button, styles.retryButton]}
            onPress={initWebMap}
          >
            <Text style={styles.buttonText}>Повторить</Text>
          </TouchableOpacity>
        )}
      </View>
    )}

    {/* Загрузка для нативных платформ */}
    {!mapReady && Platform.OS !== 'web' && !error && (
      <View style={styles.center}>
        <Text style={styles.loadingText}>Загрузка карты...</Text>
      </View>
    )}
  </View>
);

}

// Стили
const styles = StyleSheet.create({
  container: {
    flex: 1,
    position: 'relative',
  },
  mapContainer: {
    flex: 1,
    position: 'relative',
  },
  webMap: {
    width: '100%',
    height: '100%',
    position: 'absolute',
    top: 0,
    left: 0,
    backgroundColor: '#f0f0f0'
  },
  nativeMap: {
    flex: 1
  },
  center: {
    position: 'absolute',
    top: 0,
    left: 0,
    right: 0,
  }
})

```

```
bottom: 0,
justifyContent: "center",
alignItems: "center",
backgroundColor: 'rgba(255,255,255,0.9)',
zIndex: 1000
},
button: {
  position: "absolute",
  bottom: 40,
  right: 20,
  backgroundColor: "#007AFF",
  paddingHorizontal: 25,
  paddingVertical: 14,
  borderRadius: 25,
  zIndex: 1000,
  ...Platform.select({
    ios: {
      shadowColor: '#000',
      shadowOffset: { width: 0, height: 2 },
      shadowOpacity: 0.3,
      shadowRadius: 4,
    },
    android: {
      elevation: 4,
    },
    web: {
      boxShadow: '0 4px 12px rgba(0,0,0,0.3)',
    }
  })
},
buttonDisabled: {
  backgroundColor: '#ccc',
  opacity: 0.7,
},
buttonText: {
  color: "#fff",
  fontWeight: "bold",
  fontSize: 16
},
mapControls: {
  position: "absolute",
  top: 20,
  left: 20,
  zIndex: 1000
},
controlInfo: {
  backgroundColor: 'rgba(255, 255, 255, 0.9)',
  paddingHorizontal: 12,
  paddingVertical: 8,
  borderRadius: 8,
  borderWidth: 1,
```

```
        borderColor: '#ccc'
    },
controlText: {
    fontSize: 14,
    fontWeight: 'bold',
    color: '#333'
},
loadingText: {
    fontSize: 18,
    color: '#333',
    textAlign: 'center'
},
errorText: {
    fontSize: 16,
    color: '#FF3B30',
    textAlign: 'center'
},
errorContainer: {
    position: "absolute",
    top: 20,
    left: 20,
    right: 20,
    backgroundColor: 'rgba(255, 255, 255, 0.95)',
    padding: 15,
    borderRadius: 10,
    borderWidth: 1,
    borderColor: '#FF3B30',
    zIndex: 1000,
    alignItems: 'center'
},
retryButton: {
    marginTop: 10,
    backgroundColor: "#FF3B30",
}
});
```

## 2. Экран свайпов: [src/swipe.js](#)

Здесь будет реализован интерфейс для просмотра контента с помощью жестов свайпа (как в популярных dating-приложениях).

```
// Импортируем нужные хуки и компоненты из React и React Native
import React, { useRef, useState } from "react";
import {
    View,
    Text,
    Image,
    StyleSheet,
    Dimensions,
    Animated,
```

```

    PanResponder,
} from "react-native";

// Получаем ширину и высоту экрана устройства
const { width, height } = Dimensions.get("window");

// Массив карточек (для примера используем одинаковые изображения)
const cards = [
  { id: "01", image: require("../assets/robot_01.jpg") },
  { id: "02", image: require("../assets/robot_02.jpg") },
  { id: "03", image: require("../assets/robot_03.jpg") },
  { id: "04", image: require("../assets/robot_04.jpg") },
  { id: "05", image: require("../assets/robot_05.jpg") },
];

// Карта переходов при свайпе: для каждой карточки указано, куда перейти при
// свайпе влево/вправо
const swipeMap = {
  "01": { left: "03", right: "02" },
  "02": { left: "04", right: "05" },
  "03": { left: "04", right: "05" },
  "04": { left: "01", right: "02" },
  "05": { left: "02", right: "01" },
};

// Основной компонент экрана свайпа
export default function SwipeScreen() {

  // Состояние – индекс текущей отображаемой карточки
  const [currentIndex, setCurrentIndex] = useState(0);

  // Animated.ValueXY – хранит координаты анимации по X и Y
  const position = useRef(new Animated.ValueXY()).current;

  // Создаем обработчик жестов PanResponder
  const panResponder = useRef(
    PanResponder.create({
      // Разрешаем реагировать на касания
      onStartShouldSetPanResponder: () => true,

      // Обновляем позицию карточки при движении пальца
      onPanResponderMove: (_, gesture) => {
        position.setValue({ x: gesture.dx, y: gesture.dy }); // dx – смещение по
        X, dy – по Y
      },

      // Когда пользователь отпускает палец
      onPanResponderRelease: (_, gesture) => {
        // Если свайп вправо (dx > 100) – листаем вправо
        if (gesture.dx > 100) {
          swipe("right");
        }
      }
    })
  );
}

```

```

    }
    // Если свайп влево (dx < -100) – листаем влево
    else if (gesture.dx < -100) {
        swipe("left");
    }
    // Если свайп короткий – возвращаем карточку обратно в центр
    else {
        Animated.spring(position, {
            toValue: { x: 0, y: 0 },
            useNativeDriver: false,
        }).start();
    }
},
})
.current;

// Функция для обработки свайпа
const swipe = (direction) => {
    // Получаем id текущей карточки
    const cardId = cards[currentIndex].id;

    // Определяем id следующей карточки в зависимости от направления
    const nextId = direction === "left" ? swipeMap[cardId].left :
    swipeMap[cardId].right;

    // Находим индекс карточки с нужным id
    const nextIndex = cards.findIndex((c) => c.id === nextId);

    // Анимируем уход текущей карточки за экран
    Animated.timing(position, {
        toValue: { x: direction === "left" ? -width : width, y: 0 }, // направление
        duration: 200, // скорость анимации
        useNativeDriver: false,
    }).start(() => {
        // После завершения анимации сбрасываем позицию обратно в центр
        position.setValue({ x: 0, y: 0 });
        // Устанавливаем новую активную карточку
        setCurrentIndex(nextIndex >= 0 ? nextIndex : 0);
    });
};

// Рендеринг отдельной карточки
const renderCard = (card, index) => {
    // Отображаем только текущую карточку
    if (index !== currentIndex) return null;

    // Интерполяция значения X в угол поворота (для эффекта наклона при свайпе)
    const rotate = position.x.interpolate({
        inputRange: [-width, 0, width], // диапазон смещения
        outputRange: ["-15deg", "0deg", "15deg"], // диапазон поворота
    });
}

```

```

});;

// Возвращаем Animated.View, чтобы анимации могли работать
return (
  <Animated.View
    {...panResponder.panHandlers} // подключаем обработчик свайпа
    style={[
      styles.card, // базовый стиль
      { transform: [{ translateX: position.x }, { translateY: position.y }, {
rotate }] }, // применяем анимацию перемещения и поворота
    ]}
    key={card.id} // уникальный ключ
  >
    {/* Изображение карточки */}
    <Image source={card.image} style={styles.image} resizeMode="contain" />
    {/* Текст карточки (id) */}
    <Text style={styles.cardText}>{card.id}</Text>
  </Animated.View>
);

};

// Основной рендер компонента
return (
  <View style={styles.container}>
    {/* Рендерим текущую карточку */}
    {cards.map((card, index) => renderCard(card, index))}

    {/* Нижняя панель с индикаторами карточек */}
    <View style={styles.bottomBar}>
      {cards.map((c, idx) => (
        <Text
          key={c.id}
          // Подсвечиваем активную карточку
          style={[styles.bottomText, idx === currentIndex ? styles.activeText : null]}
        >
          {c.id}
        </Text>
      )))
    </View>
  </View>
);
}

// Стили для компонентов
const styles = StyleSheet.create({
  // Основной контейнер экрана
  container: {
    flex: 1,
    backgroundColor: "#f5f5f5", // светлый фон
    justifyContent: "center",

```

```
    alignItems: "center",
  },

// Стиль карточки
card: {
  position: "absolute", // чтобы карточки накладывались друг на друга
  width: width * 0.8, // 80% ширины экрана
  height: height * 0.6, // 60% высоты экрана
  borderRadius: 10, // скругленные углы
  backgroundColor: "#fff", // белый фон карточки
  justifyContent: "center",
  alignItems: "center",
  // Тень для iOS
  shadowColor: "#000",
  shadowOpacity: 0.3,
  shadowOffset: { width: 0, height: 5 },
  shadowRadius: 5,
  // Тень для Android
  elevation: 5,
},

// Изображение на карточке
image: {
  width: "70%", // 70% ширины карточки
  height: "70%", // 70% высоты карточки
},

// Текст (id карточки)
cardText: {
  marginTop: 20,
  fontSize: 24,
  fontWeight: "bold",
},

// Нижняя панель с индикаторами
bottomBar: {
  position: "absolute",
  bottom: 40, // отступ от низа экрана
  flexDirection: "row", // расположение элементов по горизонтали
  justifyContent: "space-around", // равномерно распределяем
  width: "80%", // ширина панели
},

// Стиль текста карточки в нижней панели
bottomText: {
  fontSize: 18,
  color: "#999", // серый цвет
},

// Стиль активной карточки в панели
activeText: {
```

```

        color: "#007AFF", // синий цвет
        fontWeight: "bold",
        fontSize: 20,
      },
    });

```

### 3. Обновление главного меню: `src/menu.js`

Добавим кнопки для перехода к новым экранам.

```

import React, { useState } from 'react';
import { StyleSheet, Text, View, TouchableOpacity, ScrollView, Image,
ImageBackground } from 'react-native';
import { useNavigation } from '@react-navigation/native';
//import AsyncStorage from '@react-native-async-storage/async-storage';
import { Ionicons } from '@expo/vector-icons';

const CropsScreen = () => {
  const navigation = useNavigation();
  // const [resetModalVisible, setResetModalVisible] = useState(false);

  React.useLayoutEffect(() => {
    navigation.setOptions({
      headerLeft: () => (
        <TouchableOpacity
          onPress={() => navigation.navigate('Login')}
          style={{ marginLeft: 15 }}
        >
          <Ionicons name="arrow-back" size={24} color="white" />
        </TouchableOpacity>
      ),
    });
  }, [navigation]);

  const crops = [
    { title: "Карта", source: require('../assets/map_icon.jpg'), navigateTo: 'Map' },
    { title: "Свайп", source: require('../assets/swipe_icon.jpg'), navigateTo: 'Swipe' },
  ];

  const renderCard = (crop, index) => (
    <TouchableOpacity
      key={index}
      style={[
        styles.card,
        crop.isDestructive && styles.destructiveCard
      ]}
      onPress={crop.action || (() => navigation.navigate(crop.navigateTo))}
    >

```

```

        <View style={styles.imageContainer}>
          <Image source={crop.source} style={styles.image} />
          <View style={styles.overlay}>
            <Text style={styles.text}>{crop.title}</Text>
          </View>
        </View>
      </TouchableOpacity>
    );

    return (
      <ImageBackground
        source={require('../assets/background.jpg')}
        style={styles.background}
        resizeMode="stretch">
        <View style={styles.screenContainer}>
          <ScrollView contentContainerStyle={styles.container}>
            <View style={styles.grid}>
              {crops.map((crop, index) => renderCard(crop, index))}
            </View>
          </ScrollView>
        </View>
      </ImageBackground>
    );
  );
}

const styles = StyleSheet.create({
  background: {
    flex: 1,
    width: '100%',
    height: '100%',
    justifyContent: 'center',
    alignItems: 'center',
  },
  screenContainer: {
    flex: 1,
    backgroundColor: 'rgba(206, 204, 204, 0.7)'
  },
  container: {
    flexGrow: 1,
    paddingVertical: 20,
    justifyContent: 'center',
    alignItems: 'center',
  },
  grid: {
    flexDirection: 'row',
    flexWrap: 'wrap',
    justifyContent: 'space-around',
  },
  card: {
    width: '45%',
    marginBottom: 20,
  }
});

```

```
    alignItems: 'center',
},
destructiveCard: {
  opacity: 0.8,
},
imageContainer: {
  position: 'relative',
},
image: {
  width: 150,
  height: 150,
  borderRadius: 10,
},
overlay: {
  position: 'absolute',
  bottom: 0,
  left: 0,
  right: 0,
  backgroundColor: 'rgba(0, 0, 0, 0.5)',
  borderBottomLeftRadius: 10,
  borderBottomRightRadius: 10,
},
text: {
  color: 'rgba(255, 255, 255, 0.7)',
  fontSize: 16,
  textAlign: 'center',
  paddingVertical: 5,
},
footerText: {
  fontSize: 12,
  color: 'rgba(255, 255, 255, 0.7)',
  marginTop: 5,
  textAlign: 'center',
},
// Стили для модального окна
modalOverlay: {
  flex: 1,
  justifyContent: 'center',
  alignItems: 'center',
  backgroundColor: 'rgba(0,0,0,0.5)',
},
modalContent: {
  width: '80%',
  backgroundColor: 'white',
  borderRadius: 20,
  padding: 25,
  alignItems: 'center',
  elevation: 5,
},
modalTitle: {
  fontSize: 22,
```

```
fontWeight: 'bold',
marginBottom: 15,
color: '#333',
},
modalText: {
fontSize: 16,
textAlign: 'center',
marginBottom: 25,
color: '#555',
lineHeight: 22,
},
modalButtons: {
flexDirection: 'row',
justifyContent: 'space-around',
width: '100%',
},
modalButton: {
flexDirection: 'row',
alignItems: 'center',
justifyContent: 'center',
paddingVertical: 12,
paddingHorizontal: 20,
borderRadius: 10,
width: '40%',
},
cancelButton: {
backgroundColor: '#f1f1f1',
borderWidth: 1,
borderColor: '#ddd',
},
confirmButton: {
backgroundColor: '#e74c3c',
},
buttonIcon: {
width: 20,
height: 20,
marginRight: 8,
},
buttonText: {
fontSize: 16,
fontWeight: 'bold',
},
});
};

export default CropsScreen;
```

#### 4. Обновление навигации: App.js

Интегрируем новые экраны в общую систему навигации приложения.

```
import { StatusBar } from 'expo-status-bar';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import Login from './src/login';
import Menu from './src/menu';
import Map from './src/map';
import Swipe from './src/swipe';

const Stack = createNativeStackNavigator();

console.log('Web app is loading');

export default function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator
        initialRouteName="Login"
        screenOptions={{
          headerStyle: {
            backgroundColor: '#FF69B4', // Ярко розовый фон заголовка
            height: 70, // Уменьшенная высота заголовка
          },
          headerTintColor: '#FFFFFF', // Белый цвет текста заголовка
          headerTitleStyle: {
            fontSize: 18, // Уменьшенный размер текста заголовка
            fontWeight: 'bold',
          },
        }}
      >
        <Stack.Screen
          name="Login"
          component={Login}
          options={{ title: 'Добро пожаловать!' }}
        />
        <Stack.Screen
          name="Menu"
          component={Menu}
          options={{ title: 'Меню' }}
        />
        <Stack.Screen
          name="Map"
          component={Map}
          options={{ title: 'Карта' }}
        />
        <Stack.Screen
          name="Swipe"
          component={Swipe}
          options={{ title: 'Свайп' }}
        />
      </Stack.Navigator>
      <StatusBar style="auto" />
    
```

```
    </NavigationContainer>
);
}
```

---

## Работа с Git: создание нового репозитория

Если вы хотите сохранить проект в новом репозитории, выполните следующие команды:

```
bash
# Просмотр текущих подключений к удаленным репозиториям
git remote -v

# Удаление связи с существующим репозиторием (если нужно)
git remote remove origin

# Подключение к новому репозиторию на GitHub
git remote add origin https://github.com/ваш-логин/новый-репозиторий.git

# Отправка кода в новый репозиторий
git push origin main
```

---

**Итог занятия:** Вы успешно расширили функциональность приложения, добавив экраны карты и свайпов, обновили структуру проекта и освоили методы управления подключением к Git-репозиториям. В следующем занятии мы сосредоточимся на наполнении созданных экранов реальной логикой и интерфейсами.

## Занятие 5: Работа с базой данных и backend-часть

[https://github.com/EugenePokh/full\\_steam\\_05\\_postgresql\\_python\\_start.git](https://github.com/EugenePokh/full_steam_05_postgresql_python_start.git)

В этом занятии мы перейдем к backend-разработке и работе с данными. Мы установим и настроим базу данных PostgreSQL, подготовим окружение Python и создадим скрипт для загрузки данных из Excel-файла в базу данных.

---

### Установка и настройка PostgreSQL

PostgreSQL — это мощная объектно-реляционная система управления базами данных, которую мы будем использовать для хранения информации нашего приложения.

#### Установка:

- Скачайте установщик с официального сайта: <https://www.postgresql.org/>
- Запустите установку с настройками по умолчанию
- Запомните пароль, который вы зададите для пользователя `postgres`

#### Общие настройки подключения:

После установки вы можете подключиться к базе данных со следующими параметрами:

```
python
host='localhost',
database='postgres',
user='postgres',
password='postgres', # или тот пароль, который вы задали при установке
port=5432
```

---

### Установка Python

Python будет использоваться для backend-логики и работы с данными.

- Скачайте последнюю версию Python с официального сайта: <https://www.python.org/>
  - При установке не забудьте отметить галочку "Add Python to PATH"
- 

## Подготовка окружения для работы с данными

### Создание виртуального окружения (рекомендуется):

Виртуальное окружение позволяет изолировать зависимости проекта и избежать конфликтов версий.

```
bash
# Создание виртуального окружения
python -m venv myenv

# Активация виртуального окружения
# На Windows:
myenv\Scripts\activate
# На Mac/Linux:
source myenv/bin/activate

# Деактивация виртуального окружения
deactivate
```

### Установка необходимых библиотек:

После активации виртуального окружения установите зависимости:

```
bash
pip install pandas openpyxl psycopg2-binary sqlalchemy
```

### Что мы устанавливаем:

- `pandas` — библиотека для работы с табличными данными
  - `openpyxl` — для чтения Excel-файлов
  - `psycopg2-binary` — драйвер для подключения к PostgreSQL
  - `sqlalchemy` — ORM для удобной работы с базой данных
- 

## Структура файлов для работы с данными

Создайте следующую структуру файлов:

```
text
```

```
my-first-app/
├── data/
│   └── dataset.xlsx      # Excel-файл с исходными данными
├── backend/
│   └── upload.py         # Скрипт для загрузки данных в БД
└── ... (остальные файлы проекта)
```

---

## Подготовка файлов для работы

### 1. Файл с данными: dataset.xlsx

Файл данных содержит ретроспективные показатели для анализа и прогнозирования

Year	Источник: <a href="https://thedocs.worldbank.org/en/doc/5d903e848db1d1b83e0ec8f744e55570-0350012021/related/CMO-Historical-Data-Annual.xlsx">https://thedocs.worldbank.org/en/doc/5d903e848db1d1b83e0ec8f744e55570-0350012021/related/CMO-Historical-Data-Annual.xlsx</a>								Источник: <a href="https://ru.investing.com/commodities/us-wheat-historical-data">https://ru.investing.com/commodities/us-wheat-historical-data</a>	Источник: <a href="https://data.worldbank.org/indicator/SP.POP.TOTL">https://data.worldbank.org/indicator/SP.POP.TOTL</a>	
	Нефть		Газ		С/х продукция		Удобрения			Население в мире	
year	Crude oil, Brent (\$/bbl)	Natural gas, US (\$/mmbtu)	Soybeans (\$/mt)	Sunflower oil (\$/mt)	Wheat, US HRW wheat	Phosphate rock phosphate	DAP dap	TSP tsp	Urea urea	Potassium chloride potassium	Population, total (units)
crude油	natural gas	soybeans	sunflower	wheat	phosphate	(\$/mt)	(\$/mt)	(\$/mt)	urea	potassium	population
1960	1,63	0,14	91,83		57,99	13,00		53,00	42,25	28,50	3 031 474 234
1961	1,57	0,15	109,33		58,61	13,00		52,00	42,25	30,00	3 072 421 801
1962	1,52	0,16	100,54		64,33	11,50		51,00	42,25	30,00	3 126 849 612
1963	1,50	0,16	110,09		64,49	11,50		52,00	42,25	30,00	3 193 428 894
1964	1,45	0,15	110,51		67,53	12,50		53,00	60,50	32,50	3 260 441 925
1965	1,42	0,16	116,87		59,46	14,00		54,00	65,75	29,50	3 328 209 022
1966	1,36	0,16	125,66		62,95	13,00		50,00	59,25	27,50	3 398 480 280
1967	1,33	0,16	114,73		65,71	12,00	68,50	45,00	49,25	25,50	3 468 370 526
1968	1,32	0,16	110,80		62,77	11,50	60,50	37,50	35,50	24,00	3 540 164 023
1969	1,27	0,17	106,75		58,39	11,25	58,00	39,25	26,60	22,00	3 614 572 835
1970	1,21	0,17	116,92		54,90	11,00	54,00	42,50	18,25	31,50	3 690 209 960
1971	1,69	0,18	125,58		61,73	11,25	61,75	43,25	16,00	32,50	3 767 930 001
1972	1,82	0,19	140,00		69,81	11,50	91,00	67,50	29,25	33,50	3 843 607 574
1973	2,81	0,21	290,33		139,81	13,75	118,75	99,50	64,75	42,50	3 920 017 410
1974	10,97	0,29	276,92		179,71	54,52	332,62	303,58	285,75	60,50	3 995 888 368
1975	10,43	0,43	219,92		149,06	67,17	242,50	202,50	168,00	81,33	4 070 022 249
1976	11,63	0,58	231,17		132,92	35,83	119,92	90,92	82,00	55,50	4 143 091 942
1977	12,57	0,79	280,17		103,22	61,79	134,08	97,92	120,76	55,29	4 215 826 364
1978	12,92	0,91	268,33		127,75	31,28	139,83	98,04	136,04	59,01	4 289 795 862
1979	32,11	1,18	297,75		160,29	35,48	193,33	146,23	153,00	76,47	4 365 742 277
1980	37,89	1,59	296,23		172,73	44,67	222,21	180,25	200,56	106,50	4 442 348 279
1981	36,68	1,98	288,42		174,96	48,96	177,63	161,33	182,50	110,17	4 520 917 350
1982	33,42	2,47	244,60		160,36	42,96	166,63	138,42	123,96	83,25	4 602 701 335
1983	29,83	2,59	281,59		157,42	39,80	169,08	134,67	114,54	74,42	4 684 875 627
1984	28,80	2,66	282,08		152,33	33,07	175,63	131,25	150,17	84,13	4 766 640 881
1985	27,33	2,51	224,42		135,83	36,00	157,17	121,38	111,63	83,21	4 850 076 923
1986	14,77	1,94	208,42		114,92	36,00	138,17	121,17	70,75	72,96	4 936 006 502
1987	18,34	1,66	215,75		112,90	34,15	160,00	138,00	82,50	67,29	5 024 289 346
1988	14,97	1,68	303,50		145,20	31,60	180,17	158,38	113,54	86,83	5 113 387 878
1989	18,22	1,70	275,00		169,24	31,00	155,29	144,00	102,96	98,33	5 202 582 534
1990	23,68	1,70	246,75		135,52	31,29	154,68	131,82	115,65	94,63	5 293 395 467
1991	20,07	1,49	239,58		128,67	31,50	158,25	133,12	138,40	106,43	5 382 536 929
1992	19,31	1,77	235,50		151,20	31,50	137,87	120,74	117,60	111,76	5 470 164 577
1993	17,02	2,12	255,08		140,24	31,50	122,00	111,94	85,54	112,54	5 556 623 321
1994	15,83	1,92	251,83		149,73	31,83	160,42	132,11	93,89	112,50	5 642 046 034
1995	17,07	1,72	259,25		176,98	32,67	199,77	149,63	126,86	112,50	5 726 736 488
1996	20,65	2,73	304,83		207,60	33,03	198,10	175,83	153,69	112,50	5 811 580 202
1997	19,09	2,48	295,42		159,48	32,50	186,99	171,91	117,49	112,50	5 896 055 962
1998	12,72	2,09	243,25		126,13	32,50	186,86	173,05	83,22	112,50	5 979 726 559
1999	17,81	2,27	201,67		112,04	42,80	171,10	154,49	65,94	112,50	6 062 288 850
2000	28,27	4,31	211,83		114,09	44,00	147,67	137,71	100,92	112,50	6 144 321 462
2001	24,42	3,96	195,83		126,81	44,00	143,51	126,88	96,45	112,50	6 226 348 086
2002	24,97	3,36	212,67		148,08	44,00	152,19	133,06	94,42	112,50	6 308 140 970
2003	28,85	5,49	264,00	660,00	146,14	44,00	171,67	149,33	138,19	112,50	6 389 462 496
2004	38,30	5,89	306,50	724,00	156,88	44,00	204,74	186,31	174,91	125,55	6 470 700 346
2005	54,43	8,92	274,69	602,00	152,35	44,00	226,79	201,48	217,35	155,00	6 552 700 448
2006	65,39	6,72	268,65	730,00	192,04	44,00	229,87	201,63	222,14	178,33	6 635 110 367
2007	72,70	6,98	383,10	1469,00	255,21	52,02	391,96	339,05	307,11	195,42	6 717 567 584
2008	97,64	8,86	521,87	759,00	326,03	242,74	861,62	878,39	514,98	455,97	6 801 440 971
2009	61,86	3,95	423,62	986,00	224,07	222,75	288,47	257,42	251,08	558,48	6 885 663 352
2010	79,64	4,39	447,10	1454,00	223,58	105,31	456,43	381,89	288,90	332,15	6 969 985 525
2011	110,94	4,00	537,52	1439,77	316,26	163,55	516,29	538,26	398,92	392,82	7 054 044 372
2012	111,97	2,75	595,51	1271,90	313,24	184,93	480,24	462,00	398,60	465,38	7 141 386 257
2013	108,86	3,72	551,39	996,00	312,25	126,32	410,97	380,75	339,61	395,00	7 229 303 088
2014	98,94	4,37	484,86	900,95	284,89	110,46	429,59	382,00	308,44	296,63	7 317 040 295
2015	52,37	2,61	392,12	898,18	204,45	120,31	416,62	377,60	277,94	296,06	7 403 850 164
2016	44,05	2,49	405,45	873,00	166,63	110,46	315,81	290,27	194,13	260,33	7 490 415 449
2017	54,39	2,96	393,38	800,76	174,20	89,69	323,03	283,20	213,88	218,23	7 576 441 961
2018	71,07	3,16	394,42	703,00	209,93	87,90	393,43	346,74	249,45	215,50	7 660 371 127
2019	64,03	2,57	368,95	776,00	201,69	87,96	306,36	294,55	245,28	255,50	7 741 774 583
2020	42,30	2,01	406,64	1170,02	231,57	76,05	312,42	265,04	229,10	241,07	7 820 205 606
2021	70,44	3,85	583,32	1361,83	315,24	123,21	600,96	538,19	483,21	542,81	7 888 305 693
2022	99,82	6,37	675,40	1233,80	429,97	266,16	772,16	716,05	700,02	863,42	7 950 946 801
2023	82,62	2,54	597,90	944,03	340,43	321,66	550,05	480,22	358,00	383,22	8 189 475 205

## 2. Скрипт загрузки: upload.py

Выполняется загрузка из dataset.xlsx в базу данных

```
import pandas as pd
```

```

import psycopg2
from sqlalchemy import create_engine, text
import os

def excel_to_postgres():
    # Параметры подключения к PostgreSQL
    db_config = {
        'host': 'localhost',
        'database': 'postgres',
        'user': 'postgres',
        'password': 'postgres',
        'port': 5432
    }

    # Путь к файлу dataset.xlsx
    excel_file = 'dataset.xlsx'

    try:
        # Сначала проверим структуру файла
        print("Анализ структуры Excel файла...")

        # Читаем весь файл без пропусков для диагностики
        df_raw = pd.read_excel(excel_file, sheet_name='Y', header=None)
        print(f"Всего строк в файле: {len(df_raw)}")
        print(f"Всего столбцов в файле: {len(df_raw.columns)}")
        print("\nПервые 10 строк файла:")
        print(df_raw.head(10))

        # Покажем что в 6-й строке (индекс 5)
        print(f"\nСодержимое 6-й строки (будущие заголовки):")
        print(df_raw.iloc[5] if len(df_raw) > 5 else "6-я строка отсутствует!")

        # Теперь читаем с пропуском 5 строк
        df = pd.read_excel(
            excel_file,
            sheet_name='Y', # указываем конкретное имя листа
            skiprows=5,      # пропускаем первые 5 строк
            header=0         # используем следующую строку как заголовки
        )

        print(f"\nПосле пропуска 5 строк:")
        print(f"Прочитано {len(df)} строк из файла {excel_file}")
        print(f"Столбцы: {list(df.columns)}")

        if len(df) == 0:
            print("\nВНИМАНИЕ: Данные не найдены! Возможные причины:")
            print("1. Неправильное имя листа")
            print("2. Меньше 6 строк в файле")
            print("3. Пустые строки после заголовков")
            return
    
```

```

# Проверяем структуру данных
print("\nПервые 5 строк данных:")
print(df.head())

print("\nИнформация о типах данных:")
print(df.dtypes)

# Создаем строку подключения для SQLAlchemy
connection_string =
f"postgresql://{{db_config['user']}:{db_config['password']}@{{db_config['host']}:{db_config['port']}}/{{db_config['database']}}"

# Создаем движок SQLAlchemy
engine = create_engine(connection_string)

# Имя таблицы в базе данных
table_name = 'full_steam_dataset'

# Сохраняем данные в PostgreSQL
df.to_sql(
    table_name,
    engine,
    if_exists='replace', # заменяем таблицу если существует
    index=False,         # не сохраняем индексы pandas
    method='multi'       # для более быстрой вставки
)

print(f"\nДанные успешно сохранены в таблицу '{table_name}'")

# Проверяем сохраненные данные (исправленная версия)
with engine.connect() as conn:
    result = conn.execute(text(f"SELECT COUNT(*) FROM {table_name}"))
    count = result.scalar()
    print(f"В таблице сохранено {count} записей")

except FileNotFoundError:
    print(f"Ошибка: Файл {excel_file} не найден в текущей директории")
    print(f"Текущая директория: {os.getcwd()}")
except Exception as e:
    print(f"Произошла ошибка: {e}")
    import traceback
    traceback.print_exc()

if __name__ == "__main__":
    excel_to_postgres()

```

---

## Проверка работы

После настройки и заполнения файлов выполните:

```
bash
# Активируйте виртуальное окружение (если используете)
source myenv/bin/activate

# Запустите скрипт загрузки данных
python backend/upload.py
```

Если все настроено правильно, вы увидите сообщение об успешной загрузке данных, и ваши данные появятся в базе PostgreSQL.

---

**Итог занятия:** Вы успешно установили и настроили PostgreSQL, подготовили Python-окружение для работы с данными и создали базовую структуру для загрузки данных в базу. В следующем занятии мы создадим API с помощью FastAPI для связи между нашим приложением и базой данных.

## Занятие 6: Введение в машинное обучение и нейронные сети

[https://github.com/EugenePokh/full\\_steam\\_06\\_python\\_ii.git](https://github.com/EugenePokh/full_steam_06_python_ii.git)

В этом занятии мы перейдем к самой интересной части — работе с нейронными сетями и машинным обучением. Мы настроим окружение для сложных вычислений и создадим модуль для анализа данных и прогнозирования.

---

### Установка базовых библиотек для анализа данных

Для начала установим основные библиотеки, которые понадобятся для работы с данными и простых математических операций:

```
bash  
pip install pandas numpy psycopg2 sqlalchemy matplotlib seaborn
```

#### Что мы устанавливаем:

- `pandas` — работа с табличными данными
  - `numpy` — математические операции и работа с массивами
  - `psycopg2 + sqlalchemy` — работа с PostgreSQL
  - `matplotlib + seaborn` — визуализация данных и построение графиков
- 

### Настройка окружения для работы с видеокартой (CUDA)

Для ускорения обучения нейронных сетей мы можем использовать мощности видеокарты. Это необязательный, но рекомендуемый шаг для серьезной работы с ИИ.

#### Установка CUDA Toolkit:

- Скачайте и установите CUDA Toolkit с официального сайта NVIDIA:  
<https://developer.nvidia.com/cuda-toolkit>

- Выберите версию, совместимую с вашей видеокартой (рекомендуется CUDA 11.8 или выше)

## Установка PyTorch с поддержкой CUDA:

```
bash
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl
/cu118
```

## Установка дополнительных библиотек машинного обучения:

```
bash
pip install pandas numpy scikit-learn tensorflow matplotlib seaborn psycopg2 sqlalchemy
```

## Что мы устанавливаем:

- `torch` — основной фреймворк для работы с нейронными сетями
- `tensorflow` — альтернативный фреймворк для машинного обучения
- `scikit-learn` — библиотека для классического машинного обучения

## Создание модуля анализа данных: `analysis.py`

Этот файл будет содержать всю логику для работы с машинным обучением и нейронными сетями.

```
import pandas as pd
```

```

import numpy as np
import psycopg2
from sqlalchemy import create_engine
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import warnings
warnings.filterwarnings('ignore')

# Базовые функции из предыдущего скрипта
def connect_to_db():
    """Подключение к PostgreSQL"""
    try:
        conn = psycopg2.connect(
            host='localhost',
            database='postgres',
            user='postgres',
            password='postgres',
            port=5432
        )
        print("Успешное подключение к базе данных")
        return conn
    except Exception as e:
        print(f"Ошибка подключения: {e}")
        return None

def load_data():
    """Загрузка данных из базы"""
    conn = connect_to_db()
    if conn:
        query = "SELECT * FROM full_stake_dataset ORDER BY year"
        df = pd.read_sql(query, conn)
        conn.close()
        print(f"Загружено {len(df)} строк данных")
        return df
    else:
        raise Exception("Не удалось подключиться к базе данных")

def handle_missing_values(df):
    """Обработка пропущенных значений"""

```

```

print("\n==== ОБРАБОТКА ПРОПУЩЕННЫХ ЗНАЧЕНИЙ ====")

# Создаем копию DataFrame
df_clean = df.copy()

# Список числовых столбцов для обработки
numeric_columns = ['crude_oil', 'natural_gas', 'soybeans', 'sunflower',
'wheat',
'phosphate_rock', 'dap', 'tsp', 'urea',
'potassium_chloride', 'population']

for column in numeric_columns:
    if column in df_clean.columns:
        missing_count = df_clean[column].isnull().sum()
        if missing_count > 0:
            print(f"Столбец {column}: {missing_count} пропущенных значений")

            # Заполняем пропуски интерполяцией
            df_clean[column] = df_clean[column].interpolate(method='linear')

            # Если остались пропуски (в начале/конце), заполняем соседними
значениями
            df_clean[column] =
df_clean[column].fillna(method='bfill').fillna(method='ffill')

            print(f" → Заполнено {missing_count} пропусков")

print("Обработка пропущенных значений завершена")
return df_clean

def save_forecasts_to_db(forecasts, future_years, metrics):
    """Сохранение прогнозов в базу данных"""
    print("\n==== СОХРАНЕНИЕ В БАЗУ ДАННЫХ ====")

    try:
        engine =
create_engine('postgresql://postgres:postgres@localhost:5432/postgres')

        for model_name, values in forecasts.items():
            # Создание DataFrame с прогнозами
            forecast_df = pd.DataFrame({
                'year': future_years,
                'crude_oil_forecast': values,
                'model_name': model_name,
                'mae': metrics[model_name]['MAE'],
                'rmse': metrics[model_name]['RMSE'],
                'r2': metrics[model_name]['R2']
            })

            # Имя таблицы
            table_name = f"{model_name.lower()}_crude_oil_forecast"

```

```

        # Сохранение в базу
        forecast_df.to_sql(table_name, engine, if_exists='replace',
index=False)
        print(f"Сохранено в таблицу: {table_name}")

    except Exception as e:
        print(f"Ошибка при сохранении в базу: {e}")

# Улучшенные функции
def create_time_features(df):
    """Создание временных признаков"""
    df_temp = df.copy()

    # Лаговые признаки (значения за предыдущие годы)
    for lag in [1, 2, 3]:
        df_temp[f'crude_oil_lag_{lag}'] = df_temp['crude_oil'].shift(lag)
        df_temp[f'population_lag_{lag}'] = df_temp['population'].shift(lag)

    # Скользящие средние
    for window in [3, 5]:
        df_temp[f'crude_oil_ma_{window}'] =
df_temp['crude_oil'].rolling(window=window).mean()
        df_temp[f'soybeans_ma_{window}'] =
df_temp['soybeans'].rolling(window=window).mean()

    # Темпы роста
    df_temp['crude_oil_growth'] = df_temp['crude_oil'].pct_change()
    df_temp['population_growth'] = df_temp['population'].pct_change()

    # Заполняем пропуски, образовавшиеся при создании признаков
    df_temp = df_temp.fillna(method='bfill').fillna(method='ffill')

    return df_temp

def analyze_trends(df):
    """Анализ трендов и сезонности"""
    print("Анализ трендов...")

    try:
        from statsmodels.tsa.stattools import adfuller

        result = adfuller(df['crude_oil'].dropna())
        print(f"Тест на стационарность crude_oil: p-value = {result[1]:.4f}")
        if result[1] > 0.05:
            print(" → Ряд нестационарен, учитываем тренд")
    except ImportError:
        print(" → statsmodels не установлен, пропускаем тест на стационарность")

    # Визуализация тренда
    plt.figure(figsize=(12, 6))

```

```

plt.plot(df['year'], df['crude_oil'], label='Crude Oil')

# Линейный тренд
z = np.polyfit(df['year'], df['crude_oil'], 1)
p = np.poly1d(z)
plt.plot(df['year'], p(df['year']), "r--", alpha=0.7, label='Линейный тренд')

plt.title('Тренд crude_oil с течением времени')
plt.xlabel('Год')
plt.ylabel('Crude Oil')
plt.legend()
plt.grid(True, alpha=0.3)
plt.savefig('trend_analysis.png')
plt.close()
print("График тренда сохранен в trend_analysis.png")

def improved_feature_selection(df):
    """Улучшенный отбор признаков"""
    # Все возможные признаки (исключаем целевую переменную и год)
    all_features = [col for col in df.columns if col not in ['year',
    'crude_oil']]

    # Удаляем признаки с высокой корреляцией между собой
    correlation_matrix = df[all_features].corr()

    # Находим пары с корреляцией > 0.95
    high_corr_pairs = []
    for i in range(len(correlation_matrix.columns)):
        for j in range(i+1, len(correlation_matrix.columns)):
            if abs(correlation_matrix.iloc[i, j]) > 0.95:
                high_corr_pairs.append((
                    correlation_matrix.columns[i],
                    correlation_matrix.columns[j],
                    correlation_matrix.iloc[i, j]
                ))
    if high_corr_pairs:
        print("Высококоррелированные пары (>0.95):")
        for pair in high_corr_pairs:
            print(f" {pair[0]} - {pair[1]}: {pair[2]:.3f}")

    # Отбираем признаки с лучшей корреляцией с целевой переменной
    correlations_with_target =
    df[all_features].corrwith(df['crude_oil']).abs().sort_values(ascending=False)

    # Берем топ-8 признаков
    selected_features = correlations_with_target.head(8).index.tolist()

    # Добавляем временные признаки, если они есть
    time_features = [col for col in df.columns if any(x in col for x in ['lag',
    'ma', 'growth'])]

```

```

    selected_features.extend(time_features[:2]) # Добавляем 2 лучших временных
признака

    return selected_features

def prepare_improved_data(df, features_to_use):
    """Улучшенная подготовка данных"""
    # Убедимся, что все признаки существуют
    available_features = [f for f in features_to_use if f in df.columns]
    X = df[available_features]
    y = df['crude_oil']

    # Используем RobustScaler для устойчивости к выбросам
    scaler_X = RobustScaler()
    scaler_y = RobustScaler()

    X_scaled = scaler_X.fit_transform(X)
    y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1)).flatten()

    # Для временных рядов используем последовательное разделение
    split_idx = int(0.8 * len(X))

    X_train = X_scaled[:split_idx]
    X_test = X_scaled[split_idx:]
    y_train = y_scaled[:split_idx]
    y_test = y_scaled[split_idx:]

    print(f"Размер train: {len(X_train)}, test: {len(X_test)})")

    return X_train, X_test, y_train, y_test, scaler_X, scaler_y

def create_improved_linear_model(X_train, X_test, y_train, y_test, scaler_y):
    """Улучшенные линейные модели"""
    # Тестируем разные регуляризации
    models = {
        'Ridge': Ridge(alpha=1.0),
        'Lasso': Lasso(alpha=0.1),
        'Linear': LinearRegression()
    }

    best_model = None
    best_r2 = -np.inf
    best_pred = None

    for name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        # Обратное масштабирование
        y_test_original = scaler_y.inverse_transform(y_test.reshape(-1,
1)).flatten()

```

```

        y_pred_original = scaler_y.inverse_transform(y_pred.reshape(-1,
1)).flatten()

        r2 = r2_score(y_test_original, y_pred_original)

        if r2 > best_r2:
            best_r2 = r2
            best_model = model
            best_pred = y_pred_original

    # Метрики для лучшей модели
    y_test_original = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten()
    mae = mean_absolute_error(y_test_original, best_pred)
    rmse = np.sqrt(mean_squared_error(y_test_original, best_pred))
    r2 = best_r2

    print(f"Лучшая линейная модель: R² = {r2:.4f}")
    print(f"MAE: {mae:.2f}, RMSE: {rmse:.2f}")

    return best_model, best_pred, {'MAE': mae, 'RMSE': rmse, 'R2': r2}

def create_improved_random_forest(X_train, X_test, y_train, y_test, scaler_y):
    """Улучшенный случайный лес"""
    # Используем оптимизированные параметры
    model = RandomForestRegressor(
        n_estimators=200,
        max_depth=15,
        min_samples_split=5,
        min_samples_leaf=2,
        random_state=42
    )

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Обратное масштабирование
    y_test_original = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten()
    y_pred_original = scaler_y.inverse_transform(y_pred.reshape(-1, 1)).flatten()

    mae = mean_absolute_error(y_test_original, y_pred_original)
    rmse = np.sqrt(mean_squared_error(y_test_original, y_pred_original))
    r2 = r2_score(y_test_original, y_pred_original)

    print(f"MAE: {mae:.2f}, RMSE: {rmse:.2f}, R²: {r2:.4f}")

    return model, y_pred_original, {'MAE': mae, 'RMSE': rmse, 'R2': r2}

def create_gradient_boosting(X_train, X_test, y_train, y_test, scaler_y):
    """Градиентный бустинг"""
    model = GradientBoostingRegressor(
        n_estimators=100,

```

```

        learning_rate=0.1,
        max_depth=4,
        random_state=42
    )

model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Обратное масштабирование
y_test_original = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten()
y_pred_original = scaler_y.inverse_transform(y_pred.reshape(-1, 1)).flatten()

mae = mean_absolute_error(y_test_original, y_pred_original)
rmse = np.sqrt(mean_squared_error(y_test_original, y_pred_original))
r2 = r2_score(y_test_original, y_pred_original)

print(f"MAE: {mae:.2f}, RMSE: {rmse:.2f}, R2: {r2:.4f}")

return model, y_pred_original, {'MAE': mae, 'RMSE': rmse, 'R2': r2}

def create_improved_neural_network(X_train, X_test, y_train, y_test, scaler_y):
    """Улучшенная нейронная сеть"""
    # Упрощаем архитектуру для небольших данных
    model = Sequential([
        Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
        BatchNormalization(),
        Dropout(0.3),
        Dense(16, activation='relu'),
        BatchNormalization(),
        Dropout(0.2),
        Dense(8, activation='relu'),
        Dense(1)
    ])

    model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='mse',
        metrics=['mae']
    )

    # Колбэки для предотвращения переобучения
    callbacks = [
        EarlyStopping(patience=20, restore_best_weights=True),
        ReduceLROnPlateau(factor=0.5, patience=10)
    ]

    # Обучение с validation split
    history = model.fit(
        X_train, y_train,
        epochs=200,
        batch_size=8,

```

```

        validation_split=0.2,
        callbacks=callbacks,
        verbose=0
    )

# Предсказания
y_pred = model.predict(X_test, verbose=0).flatten()

# Обратное масштабирование
y_test_original = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten()
y_pred_original = scaler_y.inverse_transform(y_pred.reshape(-1, 1)).flatten()

mae = mean_absolute_error(y_test_original, y_pred_original)
rmse = np.sqrt(mean_squared_error(y_test_original, y_pred_original))
r2 = r2_score(y_test_original, y_pred_original)

print(f"MAE: {mae:.2f}, RMSE: {rmse:.2f}, R2: {r2:.4f}")

return model, y_pred_original, {'MAE': mae, 'RMSE': rmse, 'R2': r2}

def improved_forecast(models_dict, df, features_to_use, scalers_dict):
    """Улучшенное прогнозирование"""
    print("\n== ПРОГНОЗ НА 2024-2028 Годы ==")

    future_years = [2024, 2025, 2026, 2027, 2028]
    forecasts = {}

    # Используем последние известные значения
    last_known_data = df[features_to_use].iloc[-1:].values

    for model_name, model_info in models_dict.items():
        model = model_info['model']
        scaler_X = scalers_dict['scaler_X']
        scaler_y = scalers_dict['scaler_y']

        future_predictions = []
        current_features = last_known_data.copy()

        for year in range(5):
            current_scaled = scaler_X.transform(current_features)

            if hasattr(model, 'predict'):
                # Scikit-learn модели
                next_pred_scaled = model.predict(current_scaled)
            else:
                # Keras модели
                next_pred_scaled = model.predict(current_scaled, verbose=0)

            next_pred = scaler_y.inverse_transform(next_pred_scaled.reshape(-1, 1))[0, 0]
            future_predictions.append(next_pred)

        forecasts[model_name] = future_predictions

```

```

# Обновляем признаки для следующего прогноза
current_features[0, 0] = next_pred # Обновляем первый признак

forecasts[model_name] = np.array(future_predictions)
print(f'{model_name}: {[f'{x:.2f}' for x in future_predictions]}')

return forecasts, future_years

def plot_improved_results(df, forecasts, future_years, metrics):
    """Улучшенная визуализация результатов"""
    plt.figure(figsize=(15, 10))

    # Основной график с прогнозами
    plt.subplot(2, 2, 1)
    plt.plot(df['year'], df['crude_oil'], 'b-', label='Исторические данные',
             linewidth=2)

    colors = ['red', 'green', 'orange', 'purple', 'brown']
    for i, (model_name, values) in enumerate(forecasts.items()):
        plt.plot(future_years, values, 'o-', color=colors[i % len(colors)],
                 label=f'{model_name}', linewidth=2, markersize=6)

    plt.title('Прогноз crude_oil на 2024-2028 годы')
    plt.xlabel('Год')
    plt.ylabel('Crude Oil')
    plt.legend()
    plt.grid(True, alpha=0.3)

    # Сравнение метрик
    plt.subplot(2, 2, 2)
    metrics_df = pd.DataFrame(metrics).T
    metrics_df[['MAE', 'RMSE']].plot(kind='bar', ax=plt.gca())
    plt.title('Сравнение метрик MAE и RMSE')
    plt.xticks(rotation=45)
    plt.ylabel('Значение метрики')
    plt.grid(True, alpha=0.3)

    # R2 score
    plt.subplot(2, 2, 3)
    plt.bar(metrics_df.index, metrics_df['R2'], color=['red' if x < 0 else
'green' for x in metrics_df['R2']])
    plt.title('Коэффициент детерминации R2')
    plt.xticks(rotation=45)
    plt.ylabel('R2 Score')
    plt.axhline(y=0, color='black', linestyle='--', alpha=0.3)
    plt.grid(True, alpha=0.3)

    # Сравнение прогнозов
    plt.subplot(2, 2, 4)
    for i, (model_name, values) in enumerate(forecasts.items()):

```

```

        plt.plot(future_years, values, 'o-', label=model_name, linewidth=2)
plt.title('Сравнение прогнозов разных моделей')
plt.xlabel('Год')
plt.ylabel('Crude Oil')
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('improved_forecasts.png')
plt.close()
print("Графики сохранены в improved_forecasts.png")

def provide_improvement_recommendations(metrics_df, df):
    """Рекомендации по дальнейшему улучшению"""
    print("\n==== РЕКОМЕНДАЦИИ ПО УЛУЧШЕНИЮ ====")

    best_model = metrics_df.loc[metrics_df['R2'].idxmax()]
    best_r2 = best_model['R2']

    print(f"Лучшая модель: R2 = {best_r2:.4f}")

    if best_r2 < 0.5:
        print("Рекомендации для улучшения:")
        print("1. Соберите больше данных (больше лет наблюдений)")
        print("2. Добавьте внешние экономические показатели (ВВП, инфляция, курс валют)")
        print("3. Используйте более сложные методы feature engineering")
        print("4. Рассмотрите модели временных рядов (ARIMA, Prophet)")
        print("5. Проведите более тщательную очистку данных от выбросов")

    # Анализ данных
    print("\nИнформация о данных:")
    print(f"Количество наблюдений: {len(df)}")
    print(f"Диапазон лет: {df['year'].min()} - {df['year'].max()}")
    print(f"Изменчивость crude_oil: {df['crude_oil'].std():.2f}")

def improved_main():
    """Улучшенная версия основной функции"""
    print("==== УЛУЧШЕННОЕ ПРОГНОЗИРОВАНИЕ CRUDE_OIL ====")

    # 1. Загрузка данных
    df = load_data()
    df_clean = handle_missing_values(df)

    # 2. Расширенный анализ данных
    print("\n==== РАСШИРЕНИЙ АНАЛИЗ ДАННЫХ ====")

    # Добавляем временные признаки
    df_clean = create_time_features(df_clean)

    # Анализ трендов

```

```

analyze_trends(df_clean)

# 3. Улучшенный отбор признаков
features_to_use = improved_feature_selection(df_clean)
print(f"Отобранные признаки: {features_to_use}")

# 4. Подготовка данных с учетом временных рядов
X_train, X_test, y_train, y_test, scaler_X, scaler_y =
prepare_improved_data(df_clean, features_to_use)

models_dict = {}
metrics = {}

# 5. Улучшенные модели
# 5.1 Улучшенная линейная регрессия
print("\n==== УЛУЧШЕННАЯ ЛИНЕЙНАЯ РЕГРЕССИЯ ===")
lr_model, lr_pred, lr_metrics = create_improved_linear_model(X_train, X_test,
y_train, y_test, scaler_y)
models_dict['improved_linear'] = {'model': lr_model, 'pred': lr_pred}
metrics['improved_linear'] = lr_metrics

# 5.2 Улучшенный случайный лес
print("\n==== УЛУЧШЕННЫЙ СЛУЧАЙНЫЙ ЛЕС ===")
rf_model, rf_pred, rf_metrics = create_improved_random_forest(X_train,
X_test, y_train, y_test, scaler_y)
models_dict['improved_rf'] = {'model': rf_model, 'pred': rf_pred}
metrics['improved_rf'] = rf_metrics

# 5.3 Градиентный бустинг
print("\n==== ГРАДИЕНТНЫЙ БУСТИНГ ===")
gb_model, gb_pred, gb_metrics = create_gradient_boosting(X_train, X_test,
y_train, y_test, scaler_y)
models_dict['gradient_boosting'] = {'model': gb_model, 'pred': gb_pred}
metrics['gradient_boosting'] = gb_metrics

# 5.4 Улучшенная нейронная сеть
print("\n==== УЛУЧШЕННАЯ НЕЙРОННАЯ СЕТЬ ===")
nn_model, nn_pred, nn_metrics = create_improved_neural_network(X_train,
X_test, y_train, y_test, scaler_y)
models_dict['improved_nn'] = {'model': nn_model, 'pred': nn_pred}
metrics['improved_nn'] = nn_metrics

# 6. Прогнозирование
scalers_dict = {
    'scaler_X': scaler_X,
    'scaler_y': scaler_y,
    'X_data': scaler_X.transform(df_clean[features_to_use])
}

forecasts, future_years = improved_forecast(models_dict, df_clean,
features_to_use, scalers_dict)

```

```
# 7. Сохранение и визуализация
save_forecasts_to_db( forecasts, future_years, metrics)
plot_improved_results(df_clean, forecasts, future_years, metrics)

# 8. Анализ результатов
print("\n==== АНАЛИЗ РЕЗУЛЬТАТОВ ===")
metrics_df = pd.DataFrame(metrics).T
print(metrics_df.round(4))

# Рекомендации по улучшению
provide_improvement_recommendations(metrics_df, df_clean)

# Запускаем улучшенную версию
if __name__ == "__main__":
    improved_main()
```

---

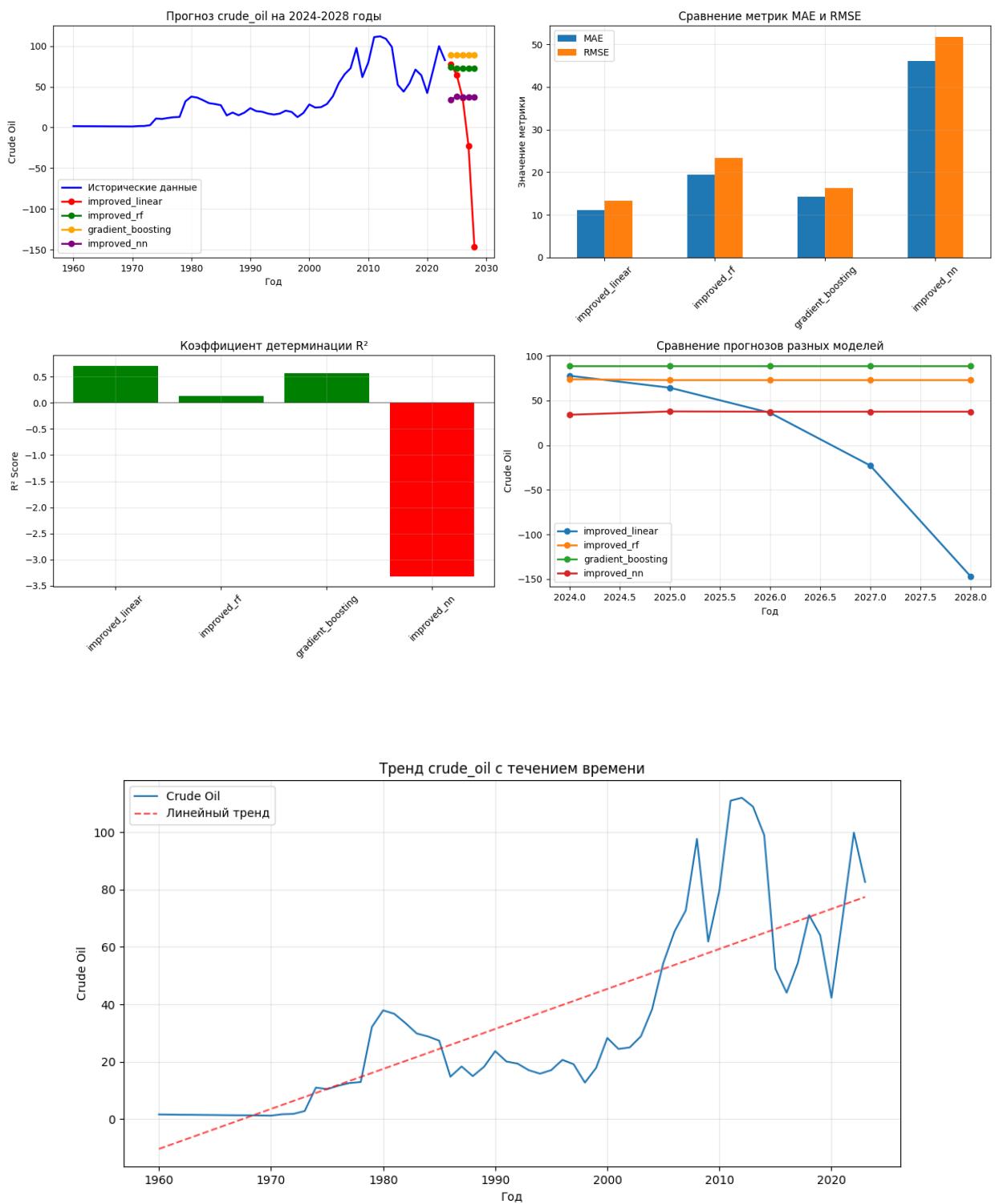
## Запуск анализа

```
python analysis.py
```

---

## Результаты анализа

После выполнения кода появятся следующие результаты:



Необходимо отметить, что прогноз на 2 года (2024 и 2025) по improved\_linear полностью соответствовал показателям стоимости нефти на конец года (см. видео).

## **Оценка качества моделей машинного обучения**

После обучения модели критически важно оценить её эффективность с помощью специальных метрик. Это позволяет понять, насколько точны прогнозы модели и пригодна ли она для реального использования.

**Основные метрики для регрессии** (когда мы предсказываем числовые значения, например, цены или количество):

- **Средняя абсолютная ошибка (MAE)** — показывает среднюю величину ошибки в абсолютном выражении. Чем меньше значение, тем лучше.
- **Среднеквадратичная ошибка (MSE)** — усиливает влияние больших ошибок за счет возведения в квадрат. Полезно, когда крупные ошибки недопустимы.
- **Коэффициент детерминации ( $R^2$ )** — показывает, какую долю дисперсии данных объясняет наша модель. Значение от 0 до 1, где 1 — идеальное предсказание. Желательным значением является от 0,7.

**Основные метрики для классификации** (когда мы предсказываем категории, например, спам/не спам):

- **Точность** — доля правильных предсказаний среди всех сделанных.
- **Полнота** — способность модели находить все relevant случаи.
- **F1-мера** — гармоническое среднее между точностью и полнотой, идеально для несбалансированных данных.
- **Матрица ошибок** — наглядная таблица, показывающая количество верных и неверных предсказаний для каждого класса.

### **Методы валидации моделей:**

- **Разделение на обучающую и тестовую выборки** — базовый метод, когда часть данных откладывается для финального тестирования.
- **Кросс-валидация** — более надежный метод, когда данные разбиваются на несколько частей и модель обучается/тестируется циклически на разных комбинациях.
- **Стратифицированная выборка** — особенно важна для классификации, сохраняет распределение классов в разных выборках.

### **Визуализация результатов:**

- **Графики остатков** — помогают выявить паттерны в ошибках предсказания.
- **Кривые обучения** — показывают, как меняется качество модели с увеличением объема данных.
- **ROC-кривые** — для классификации, демонстрируют компромисс между истинно положительными и ложно положительными срабатываниями.

### **Практические рекомендации:**

- Всегда сравнивайте несколько моделей между собой
- Обращайте внимание на переобучение — когда модель работает идеально на обучающих данных, но плохо на новых
- Учитывайте бизнес-контекст — иногда простая, но интерпретируемая модель лучше сложной "черной коробки"
- Проводите A/B тестирование перед внедрением в production

Качественная оценка модели — это не просто формальность, а важнейший этап, который определяет успех всего проекта машинного обучения.

**Итог занятия:** Вы успешно настроили окружение для работы с машинным обучением, установили все необходимые библиотеки и создали структуру для модуля анализа данных. В следующем занятии мы интегрируем наш AI-модуль с FastAPI для создания полноценного API.

## Занятие 7: Создание API и дашбордов

[https://github.com/EugenePokh/full\\_stake\\_07\\_dashboard\\_fastapi\\_backend.git](https://github.com/EugenePokh/full_stake_07_dashboard_fastapi_backend.git)

В этом занятии мы создадим полноценный backend на FastAPI и добавим в наше приложение дашборды для визуализации данных. Это ключевой этап, где фронтенд и бэкенд начинают работать вместе.

---

### Настройка FastAPI окружения

FastAPI — современный фреймворк для создания API на Python, который обеспечивает высокую производительность и простоту разработки.

#### Установка необходимых зависимостей:

bash

```
# Основные зависимости FastAPI
pip install fastapi uvicorn

# База данных
pip install psycopg2-binary sqlalchemy pandas

# Валидация данных
pip install pydantic pydantic-settings

# Дополнительные зависимости для анализа данных
pip install numpy scikit-learn matplotlib seaborn

# Для нейросетей (если используются в analysis.py)
pip install tensorflow torch scikit-learn

# Для работы с Excel/CSV
pip install openpyxl xlrd

# Для асинхронных запросов
pip install httpx aiofiles
```

#### Запуск сервера:

bash

```
python run.py
```

**Остановка сервера:** `Ctrl + C`

---

## Структура backend проекта

```
text

backend/
└── app/
    ├── __pycache__/
    │   ├── routers/
    │   │   ├── __pycache__/
    │   │   └── __init__.py
    │   └── dashboard.py      # Роуты для дашборда
    ├── config.py            # Конфигурация приложения
    └── main.py              # Основное приложение FastAPI
    └── scripts/
        ├── analysis.py      # Скрипт анализа данных
        ├── dataset.xlsx      # Исходные данные
        ├── improved_forecasts.png
        ├── trend_analysis.png
        └── upload.py          # Скрипт загрузки данных
    └── run.py                # Запуск приложения
```

---

## Настройка backend компонентов

**1. Конфигурация приложения:** `app/config.py`

Настройки базы данных, параметры подключения к PostgreSQL, настройки CORS

```
from pydantic_settings import BaseSettings
from typing import Optional

class Settings(BaseSettings):
    # Database
    database_url: str = "postgresql://postgres:postgres@localhost:5432/postgres"

    # Security
    secret_key: str = "your-secret-key-here"
```

```
debug: bool = True

# API Keys (если понадобятся)
openai_api_key: Optional[str] = None

class Config:
    env_file = ".env"

settings = Settings()
```

## 2. Основное приложение FastAPI: app/main.py

Импорт и настройка роутеров

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.routers import dashboard

app = FastAPI(title="Azot Price Portal API")

# CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Подключаем роутеры
app.include_router(dashboard.router, prefix="/api", tags=["dashboard"])

@app.get("/")
async def root():
    return {"message": "Full Stack Application Backend is running."}
```

## 3. Роуты дашборда: app/routers/dashboard.py

Эндпоинты для получения данных аналитики, запуска загрузки данных и прогноза нейронной сети

```
from fastapi import APIRouter, HTTPException, BackgroundTasks
from pydantic import BaseModel
from typing import List, Dict, Any
import pandas as pd
import psycopg2
import subprocess
import os
```

```

import sys
from sqlalchemy import create_engine

router = APIRouter(prefix="/dashboard", tags=["dashboard"])

# Настройки подключения к БД
DB_CONFIG = {
    "host": "localhost",
    "port": 5432,
    "database": "postgres",
    "user": "postgres",
    "password": "postgres"
}

def get_db_connection():
    """Создание подключения к базе данных"""
    return psycopg2.connect(**DB_CONFIG)

def get_sqlalchemy_engine():
    """Создание SQLAlchemy engine для pandas"""
    return
create_engine(f"postgresql://{DB_CONFIG['user']}:{DB_CONFIG['password']}@{DB_CONFIG['host']}:{DB_CONFIG['port']}/{DB_CONFIG['database']}")

class ForecastData(BaseModel):
    year: int
    crude_oil_forecast: float
    model_name: str
    mae: float
    rmse: float
    r2: float

class HistoricalData(BaseModel):
    year: int
    crude_oil: float

class ChartData(BaseModel):
    year: int
    linear: float
    gradient_boosting: float
    neural_network: float
    random_forest: float

class DashboardResponse(BaseModel):
    improved_linear: List[ForecastData]
    gradient_boosting: List[ForecastData]
    improved_nn: List[ForecastData]
    improved_rf: List[ForecastData]
    historical_data: List[HistoricalData]
    comparison_data: List[ChartData]
    metrics_summary: Dict[str, Any]

```

```

@router.get("/data", response_model=DashboardResponse)
async def get_dashboard_data():
    """Получение всех данных для дашборда"""
    try:
        # Получаем данные прогнозов из всех таблиц
        improved_linear = get_forecast_data("improved_linear_crude_oil_forecast")
        gradient_boosting =
get_forecast_data("gradient_boosting_crude_oil_forecast")
            improved_nn = get_forecast_data("improved_nn_crude_oil_forecast")
            improved_rf = get_forecast_data("improved_rf_crude_oil_forecast")

        # Получаем исторические данные
        historical_data = get_historical_data()

        # Подготавливаем данные для графиков
        comparison_data = prepare_comparison_data(
            improved_linear, gradient_boosting, improved_nn, improved_rf
        )

        # Сводка по метрикам
        metrics_summary = prepare_metrics_summary(
            improved_linear, gradient_boosting, improved_nn, improved_rf
        )

        return DashboardResponse(
            improved_linear=improved_linear,
            gradient_boosting=gradient_boosting,
            improved_nn=improved_nn,
            improved_rf=improved_rf,
            historical_data=historical_data,
            comparison_data=comparison_data,
            metrics_summary=metrics_summary
        )
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Ошибка получения данных: {str(e)}")

def get_forecast_data(table_name: str) -> List[ForecastData]:
    """Получение данных прогноза из указанной таблицы"""
    engine = get_sqlalchemy_engine()
    try:
        query = f"SELECT year, crude_oil_forecast, model_name, mae, rmse, r2 FROM {table_name} ORDER BY year"
        df = pd.read_sql(query, engine)

        return [
            ForecastData(
                year=int(row['year']),
                crude_oil_forecast=float(row['crude_oil_forecast']),
                model_name=row['model_name'],
                mae=row['mae'],
                rmse=row['rmse'],
                r2=row['r2']
            ) for index, row in df.iterrows()
        ]
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Ошибка получения прогноза: {str(e)}")

```

```

        model_name=str(row['model_name']),
        mae=float(row['mae']),
        rmse=float(row['rmse']),
        r2=float(row['r2']))
    )
    for _, row in df.iterrows()
]
except Exception as e:
    print(f"Error reading table {table_name}: {e}")
    return []
finally:
    engine.dispose()

def get_historical_data() -> List[HistoricalData]:
    """Получение исторических данных crude_oil"""
    engine = get_sqlalchemy_engine()
    try:
        query = "SELECT year, crude_oil FROM full_steam_dataset WHERE year
BETWEEN 1960 AND 2023 ORDER BY year"
        df = pd.read_sql(query, engine)

        return [
            HistoricalData(
                year=int(row['year']),
                crude_oil=float(row['crude_oil']) if pd.notna(row['crude_oil'])
else 0.0
            )
            for _, row in df.iterrows()
        ]
    except Exception as e:
        print(f"Error reading historical data: {e}")
        return []
    finally:
        engine.dispose()

def prepare_comparison_data(linear_data, gb_data, nn_data, rf_data):
    """Подготовка данных для графика сравнения моделей"""
    comparison_data = []

    # Проверяем, что все данные есть
    if not all([linear_data, gb_data, nn_data, rf_data]):
        return comparison_data

    for i in range(min(len(linear_data), len(gb_data), len(nn_data),
len(rf_data))):
        comparison_data.append(ChartData(
            year=linear_data[i].year,
            linear=linear_data[i].crude_oil_forecast,
            gradient_boosting=gb_data[i].crude_oil_forecast,
            neural_network=nn_data[i].crude_oil_forecast,
            random_forest=rf_data[i].crude_oil_forecast

```

```

        )))

    return comparison_data

def prepare_metrics_summary(linear_data, gb_data, nn_data, rf_data):
    """Подготовка сводки по метрикам"""
    summary = {}

    if linear_data:
        summary["linear_regression"] = {
            "mae": linear_data[0].mae,
            "rmse": linear_data[0].rmse,
            "r2": linear_data[0].r2
        }

    if gb_data:
        summary["gradient_boosting"] = {
            "mae": gb_data[0].mae,
            "rmse": gb_data[0].rmse,
            "r2": gb_data[0].r2
        }

    if nn_data:
        summary["neural_network"] = {
            "mae": nn_data[0].mae,
            "rmse": nn_data[0].rmse,
            "r2": nn_data[0].r2
        }

    if rf_data:
        summary["random_forest"] = {
            "mae": rf_data[0].mae,
            "rmse": rf_data[0].rmse,
            "r2": rf_data[0].r2
        }

    return summary

@router.post("/upload-data")
async def upload_data(background_tasks: BackgroundTasks):
    """Запуск скрипта загрузки данных"""
    try:
        # Определяем путь к скрипту относительно корня проекта
        base_dir = os.path.dirname(os.path.dirname(os.path.dirname(__file__)))
        script_path = os.path.join(base_dir, "scripts", "upload.py")

        print(f"Looking for script at: {script_path}")

        if not os.path.exists(script_path):
            raise HTTPException(status_code=404, detail=f"Скрипт upload.py не
найден по пути: {script_path}")
    
```

```

# Запускаем в фоне
background_tasks.add_task(run_script, script_path, "upload")

return {"message": "Запущен процесс выгрузки данных в PostgreSQL"}


except Exception as e:
    raise HTTPException(status_code=500, detail=f"Ошибка запуска скрипта: {str(e)}")

@router.post("/run-analysis")
async def run_analysis(background_tasks: BackgroundTasks):
    """Запуск скрипта анализа"""
    try:
        # Определяем путь к скрипту относительно корня проекта
        base_dir = os.path.dirname(os.path.dirname(os.path.dirname(__file__)))
        script_path = os.path.join(base_dir, "scripts", "analysis.py")

        print(f"Looking for script at: {script_path}")

        if not os.path.exists(script_path):
            raise HTTPException(status_code=404, detail=f"Скрипт analysis.py не найден по пути: {script_path}")

        # Запускаем в фоне
        background_tasks.add_task(run_script, script_path, "analysis")

        return {"message": "Запущен процесс анализа данных ИИ"}


    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Ошибка запуска скрипта: {str(e)}")

def run_script(script_path: str, script_name: str):
    """Запуск Python скрипта"""
    try:
        print(f"Running script: {script_path}")

        # Меняем рабочую директорию на директорию скрипта
        script_dir = os.path.dirname(script_path)
        original_cwd = os.getcwd()
        os.chdir(script_dir)

        result = subprocess.run(
            [sys.executable, script_path],
            capture_output=True,
            text=True,
            timeout=300  # 5 минут таймаут
        )

        # Возвращаемся обратно

```

```
os.chdir(original_cwd)

if result.returncode != 0:
    print(f"Ошибка выполнения скрипта {script_name}: {result.stderr}")
else:
    print(f"Скрипт {script_name} выполнен успешно: {result.stdout}")

except subprocess.TimeoutExpired:
    print(f"Скрипт {script_name} превысил время выполнения")
except Exception as e:
    print(f"Ошибка при запуске скрипта {script_name}: {str(e)}")
```

#### 4. Файл инициализации: app/routers/\_\_init\_\_.py

Экспорт роутеров

```
from . import dashboard

__all__ = ["dashboard"]
```

#### 5. Запуск приложения: run.py

Настройка сервера Uvicorn, конфигурация хоста и порта, обработка запуска

```
import uvicorn
from app.main import app

if __name__ == "__main__":
    uvicorn.run(
        "app.main:app",
        host="0.0.0.0",
        port=8000,
        reload=True
    )
```

---

## Интеграция дашбордов в React Native

### Установка зависимостей для графиков:

```
bash

npm install react-native-chart-kit
```

### Структура React Native проекта:

```
text

react_native_basic/
├── assets/
└── src/
    ├── dashboard.js      # Новый экран с дашбордами
    ├── login.js
    ├── map.js
    ├── menu.js
    └── swipe.js
└── App.js
└── ... (остальные файлы)
```

---

## Настройка React Native компонентов

### 1. Главный файл приложения: App.js

Импорт экрана дашборда, добавление маршрута навигации к дашборду, настройка стека навигации

```
import { StatusBar } from 'expo-status-bar';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import Login from './src/login';
import Menu from './src/menu';
import Map from './src/map';
import Swipe from './src/swipe';
import Dashboard from './src/dashboard';

const Stack = createNativeStackNavigator();

console.log('Web app is loading');

export default function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator
        initialRouteName="Login"
        screenOptions={{
          headerStyle: {
            backgroundColor: '#FF69B4', // Ярко розовый фон заголовка
            height: 70, // Уменьшенная высота заголовка
          },
          headerTintColor: '#FFFFFF', // Белый цвет текста заголовка
          headerTitleStyle: {
            fontSize: 18, // Уменьшенный размер текста заголовка
          }
        }}
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

```

        fontWeight: 'bold',
    },
})
>
<Stack.Screen
    name="Login"
    component={Login}
    options={{ title: 'Добро пожаловать!' }}
/>
<Stack.Screen
    name="Menu"
    component={Menu}
    options={{ title: 'Меню' }}
/>
<Stack.Screen
    name="Map"
    component={Map}
    options={{ title: 'Карта' }}
/>
<Stack.Screen
    name="Swipe"
    component={Swipe}
    options={{ title: 'Свайп' }}
/>
<Stack.Screen
    name="Dashboard"
    component={Dashboard}
    options={{ title: 'Дашбоард' }}
/>
</Stack.Navigator>
<StatusBar style="auto" />
</NavigationContainer>
);
}

```

## 2. Обновление главного меню: [src/menu.js](#)

Добавление кнопки перехода к дашборду

```

import React, { useState } from 'react';
import { StyleSheet, Text, View, TouchableOpacity, ScrollView, Image,
ImageBackground } from 'react-native';
import { useNavigation } from '@react-navigation/native';
//import AsyncStorage from '@react-native-async-storage/async-storage';
import { Ionicons } from '@expo/vector-icons';

const CropsScreen = () => {
    const navigation = useNavigation();
    // const [resetModalVisible, setResetModalVisible] = useState(false);

```

```

React.useLayoutEffect(() => {
  navigation.setOptions({
    headerLeft: () => (
      <TouchableOpacity
        onPress={() => navigation.navigate('Login')}
        style={{ marginLeft: 15 }}
      >
        <Ionicons name="arrow-back" size={24} color="white" />
      </TouchableOpacity>
    ),
  });
}, [navigation]);

const crops = [
  { title: "Карта", source: require('../assets/map_icon.jpg'), navigateTo: 'Map' },
  { title: "Свайп", source: require('../assets/swipe_icon.jpg'), navigateTo: 'Swipe' },
  { title: "Дашборд", source: require('../assets/dashboard_icon.jpg'), navigateTo: 'Dashboard' },
];

```

```

const renderCard = (crop, index) => (
  <TouchableOpacity
    key={index}
    style={[
      styles.card,
      crop.isDestructive && styles.destructiveCard
    ]}
    onPress={crop.action || (() => navigation.navigate(crop.navigateTo))}
  >
    <View style={styles.imageContainer}>
      <Image source={crop.source} style={styles.image} />
      <View style={styles.overlay}>
        <Text style={styles.text}>{crop.title}</Text>
      </View>
    </View>
  </TouchableOpacity>
);

return (
  <ImageBackground
    source={require('../assets/background.jpg')}
    style={styles.background}
    resizeMode="stretch">
    <View style={styles.screenContainer}>
      <ScrollView contentContainerStyle={styles.container}>
        <View style={styles.grid}>
          {crops.map((crop, index) => renderCard(crop, index))}
        </View>
      </ScrollView>
    </View>
  </ImageBackground>
);

```

```
        </ScrollView>
    </View>
</ImageBackground>
);
};

const styles = StyleSheet.create({
background: {
    flex: 1,
    width: '100%',
    height: '100%',
    justifyContent: 'center',
    alignItems: 'center',
},
screenContainer: {
    flex: 1,
    backgroundColor: 'rgba(206, 204, 204, 0.7)'
},
container: {
    flexGrow: 1,
    paddingVertical: 20,
    justifyContent: 'center',
    alignItems: 'center',
},
grid: {
    flexDirection: 'row',
    flexWrap: 'wrap',
    justifyContent: 'space-around',
},
card: {
    width: '45%',
    marginBottom: 20,
    alignItems: 'center',
},
destructiveCard: {
    opacity: 0.8,
},
imageContainer: {
    position: 'relative',
},
image: {
    width: 150,
    height: 150,
    borderRadius: 10,
},
overlay: {
    position: 'absolute',
    bottom: 0,
    left: 0,
    right: 0,
    backgroundColor: 'rgba(0, 0, 0, 0.5)',
```

```
borderBottomLeftRadius: 10,
borderBottomRightRadius: 10,
},
text: {
  color: 'rgba(255, 255, 255, 0.7)',
  fontSize: 16,
  textAlign: 'center',
  paddingVertical: 5,
},
footerText: {
  fontSize: 12,
  color: 'rgba(255, 255, 255, 0.7)',
  marginTop: 5,
  textAlign: 'center',
},
// Стили для модального окна
modalOverlay: {
  flex: 1,
  justifyContent: 'center',
  alignItems: 'center',
  backgroundColor: 'rgba(0,0,0,0.5)',
},
modalContent: {
  width: '80%',
  backgroundColor: 'white',
  borderRadius: 20,
  padding: 25,
  alignItems: 'center',
  elevation: 5,
},
modalTitle: {
  fontSize: 22,
  fontWeight: 'bold',
  marginBottom: 15,
  color: '#333',
},
modalText: {
  fontSize: 16,
  textAlign: 'center',
  marginBottom: 25,
  color: '#555',
  lineHeight: 22,
},
modalButtons: {
  flexDirection: 'row',
  justifyContent: 'space-around',
  width: '100%',
},
modalButton: {
  flexDirection: 'row',
  alignItems: 'center',

```

```

justifyContent: 'center',
paddingVertical: 12,
paddingHorizontal: 20,
borderRadius: 10,
width: '40%',

},
cancelButton: {
  backgroundColor: '#f1f1f1',
  borderWidth: 1,
  borderColor: '#ddd',
},
confirmButton: {
  backgroundColor: '#e74c3c',
},
buttonIcon: {
  width: 20,
  height: 20,
  marginRight: 8,
},
buttonText: {
  fontSize: 16,
  fontWeight: 'bold',
},
});
}

export default CropsScreen;

```

### 3. Экран дашборда: src/dashboard.js

Отображение графиков и запуск загрузки данных и анализа нейронными сетями

```

import React, { useState, useEffect } from 'react';
import {
  StyleSheet,
  Text,
  View,
  TouchableOpacity,
  ScrollView,
  ImageBackground,
  Alert,
  ActivityIndicator
} from 'react-native';
import { useNavigation } from '@react-navigation/native';
import { Ionicons } from '@expo/vector-icons';
import { LineChart, BarChart, PieChart } from 'react-native-chart-kit';
import { Dimensions } from 'react-native';

// Базовый URL для API запросов
const API_BASE_URL = 'http://localhost:8000/api';

```

```

const DashboardScreen = () => {
  const navigation = useNavigation();

  // Состояния компонента
  const [data, setData] = useState(null); // Данные с сервера
  const [loading, setLoading] = useState(true); // Статус загрузки
  const [actionLoading, setActionLoading] = useState({ upload: false, analysis: false }); // Статусы выполнения скриптов

  // Настройка заголовка навигации
  React.useLayoutEffect(() => {
    navigation.setOptions({
      headerLeft: () => (
        <TouchableOpacity
          onPress={() => navigation.navigate('Menu')}
          style={{ marginLeft: 15 }}
        >
          <Ionicons name="arrow-back" size={24} color="white" />
        </TouchableOpacity>
      ),
    });
  }, [navigation]);

  // Загрузка данных при монтировании компонента
  useEffect(() => {
    fetchDashboardData();
  }, []);

  // Функция загрузки данных с сервера
  const fetchDashboardData = async () => {
    try {
      setLoading(true);
      const response = await fetch(`${API_BASE_URL}/dashboard/data`);
      const result = await response.json();
      setData(result);
    } catch (error) {
      Alert.alert('Ошибка', 'Не удалось загрузить данные');
      console.error(error);
    } finally {
      setLoading(false);
    }
  };

  // Функция запуска скриптов (загрузка данных или анализ)
  const runScript = async (endpoint, actionPerformed) => {
    try {
      setActionLoading(prev => ({ ...prev, [actionName]: true }));

      const response = await fetch(`${API_BASE_URL}/dashboard/${endpoint}`, {
        method: 'POST',
      });
    }
  };
}

```

```

});;

const result = await response.json();
Alert.alert('Успех', result.message);

// Обновляем данные после выполнения скрипта
setTimeout(() => {
  fetchDashboardData();
}, 2000);

} catch (error) {
  Alert.alert('Ошибка', `Не удалось выполнить ${actionName}`);
  console.error(error);
} finally {
  setActionLoading(prev => ({ ...prev, [actionName]: false }));
}
};

// Рендер таблицы с прогнозами для каждой модели
const renderForecastTable = (title, forecastData) => (
  <View style={styles.tableContainer} key={title}>
    <Text style={styles.tableTitle}>{title}</Text>
    <View style={styles.tableHeader}>
      <Text style={styles.tableHeaderText}>Год</Text>
      <Text style={styles.tableHeaderText}>Прогноз</Text>
      <Text style={styles.tableHeaderText}>R2</Text>
    </View>
    {forecastData?.map((item, index) => (
      <View key={index} style={styles.tableRow}>
        <Text style={styles.tableCell}>{item.year}</Text>
        <Text
          style={styles.tableCell}>{item.crude_oil_forecast.toFixed(2)}</Text>
        <Text style={[
          styles.tableCell,
          { color: item.r2 > 0 ? '#FF69B4' : '#E91E63' } // Розовые цвета для
положительных/отрицательных значений
        ]}>
          {item.r2.toFixed(4)}
        </Text>
      </View>
    )));
  </View>
);

// Рендер графика сравнения моделей
const renderComparisonChart = () => {
  if (!data?.comparison_data) return null;

  const chartData = {
    labels: data.comparison_data.map(item => item.year.toString()),
    datasets: [

```

```

    },
    data: data.comparison_data.map(item => item.linear),
    color: () => '#FF69B4', // Основной розовый
    strokeWidth: 2,
  },
  {
    data: data.comparison_data.map(item => item.gradient_boosting),
    color: () => '#E91E63', // Темно-розовый
    strokeWidth: 2,
  },
  {
    data: data.comparison_data.map(item => item.neural_network),
    color: () => '#F8BB0', // Светло-розовый
    strokeWidth: 2,
  },
  {
    data: data.comparison_data.map(item => item.random_forest),
    color: () => '#AD1457', // Бордовый
    strokeWidth: 2,
  },
],
legend: ['Linear', 'Gradient Boosting', 'Neural Network', 'Random Forest']
};

return (
  <View style={styles.chartContainer}>
    <Text style={styles.chartTitle}>Сравнение прогнозов разных моделей</Text>
    <LineChart
      data={chartData}
      width={Dimensions.get('window').width - 40}
      height={220}
      chartConfig={chartConfig}
      bezier
      style={styles.chart}
    />
  </View>
);
};

// Рендер графика исторических данных и прогнозов
const renderForecastChart = () => {
  if (!data?.historical_data || !data?.comparison_data) return null;

  const historicalYears = data.historical_data.slice(-10).map(item => item.year);
  const historicalValues = data.historical_data.slice(-10).map(item => item.crude_oil);
  const forecastYears = data.comparison_data.map(item => item.year);
  const forecastValues = data.comparison_data.map(item => item.linear);

  const chartData = {

```

```

        labels: [...historicalYears, ...forecastYears],
        datasets: [
            {
                data: [...historicalValues, ...forecastValues],
                color: () => '#FF69B4', // Основной розовый цвет
                strokeWidth: 2,
            },
        ],
    };

    return (
        <View style={styles.chartContainer}>
            <Text style={styles.chartTitle}>Прогноз Crude Oil 2024-2028</Text>
            <LineChart
                data={chartData}
                width={Dimensions.get('window').width - 40}
                height={220}
                chartConfig={chartConfig}
                bezier
                style={styles.chart}
            />
        </View>
    );
};

// Конфигурация графиков
const chartConfig = {
    backgroundColor: '#ffffff',
    backgroundGradientFrom: '#ffffff',
    backgroundGradientTo: '#ffffff',
    decimalPlaces: 2,
    color: (opacity = 1) => `rgba(255, 105, 180, ${opacity})`, // Розовый
    градиент
    labelColor: (opacity = 1) => `rgba(0, 0, 0, ${opacity})`, // Черный для
    текста
    style: {
        borderRadius: 16,
    },
    propsForDots: {
        r: '4',
        strokeWidth: '2',
        stroke: '#FF69B4' // Розовые точки
    }
};

// Экран загрузки
if (loading) {
    return (
        <ImageBackground
            source={require('../assets/background.jpg')}
            style={styles.background}
    )
}

```

```

        resizeMode="stretch"
    >
    <View style={styles.loadingContainer}>
        <ActivityIndicator size="large" color="#FF69B4" /> {/* Розовый
индикатор */}
        <Text style={styles.loadingText}>Загрузка данных...</Text>
    </View>
</ImageBackground>
);
}

// Основной рендер компонента
return (
<ImageBackground
    source={require('../assets/background.jpg')}
    style={styles.background}
    resizeMode="stretch"
>
<View style={styles.screenContainer}>
<ScrollView contentContainerStyle={styles.container}>

{/* Контейнер кнопок действий */}
<View style={styles.actionsContainer}>
<TouchableOpacity
    style={[styles.actionButton, actionLoading.upload &&
styles.buttonDisabled]}
    onPress={() => runScript('upload-data', 'upload')}
    disabled={actionLoading.upload}
>
    {actionLoading.upload ? (
        <ActivityIndicator size="small" color="#ffffff" />
    ) : (
        <Ionicons name="cloud-upload" size={20} color="white" />
    )}
    <Text style={styles.actionButtonText}>
        {actionLoading.upload ? 'Выгрузка...' : 'Выгрузить данные в
PostgreSQL'}
    </Text>
</TouchableOpacity>

<TouchableOpacity
    style={[styles.actionButton, actionLoading.analysis &&
styles.buttonDisabled]}
    onPress={() => runScript('run-analysis', 'analysis')}
    disabled={actionLoading.analysis}
>
    {actionLoading.analysis ? (
        <ActivityIndicator size="small" color="#ffffff" />
    ) : (
        <Ionicons name="analytics" size={20} color="white" />
    )}

```

```

        <Text style={styles.actionButtonText}>
            {actionLoading.analysis ? 'Анализ...' : 'Выполнить анализ (ИИ)'}
        </Text>
    </TouchableOpacity>
</View>

/* Графики сравнения и прогнозов */
{renderComparisonChart()}
{renderForecastChart()}

/* Контейнер таблиц с прогнозами */
<View style={styles.tablesContainer}>
    {data?.improved_linear && renderForecastTable('Linear Regression',
data.improved_linear)}
    {data?.gradient_boosting && renderForecastTable('Gradient Boosting',
data.gradient_boosting)}
    {data?.improved_nn && renderForecastTable('Neural Network',
data.improved_nn)}
    {data?.improved_rf && renderForecastTable('Random Forest',
data.improved_rf)}
</View>

</ScrollView>
</View>
</ImageBackground>
);
};

// Стили компонента
const styles = StyleSheet.create({
    // Фоновое изображение
    background: {
        flex: 1,
        width: '100%',
        height: '100%',
    },
    // Основной контейнер с полупрозрачным серым фоном
    screenContainer: {
        flex: 1,
        backgroundColor: 'rgba(128, 128, 128, 0.7)' // Серый полупрозрачный
    },
    // Контейнер контента с отступами
    container: {
        flexGrow: 1,
        paddingVertical: 20,
        paddingHorizontal: 10,
    },
    // Контейнер загрузки
    loadingContainer: {
        flex: 1,
        justifyContent: 'center',

```

```
    alignItems: 'center',
},
// Текст загрузки
loadingText: {
  color: 'white', // Белый текст
  marginTop: 10,
  fontSize: 16,
},
// Контейнер кнопок действий
actionsContainer: {
  flexDirection: 'row',
  justifyContent: 'space-around',
  marginBottom: 20,
},
// Кнопка действия
actionButton: {
  flexDirection: 'row',
  alignItems: 'center',
  backgroundColor: '#FF69B4', // Основной розовый
  paddingVertical: 12,
  paddingHorizontal: 16,
  borderRadius: 10,
  flex: 0.45,
  justifyContent: 'center',
},
// Отключенная кнопка
buttonDisabled: {
  backgroundColor: '#9E9E9E', // Серый для отключенного состояния
},
// Текст кнопки действия
actionButtonText: {
  color: 'white', // Белый текст
  marginLeft: 8,
  fontWeight: 'bold',
  fontSize: 12,
  textAlign: 'center',
},
// Контейнер графика
chartContainer: {
  backgroundColor: 'rgba(177, 177, 177, 0.95)', // Белый фон
  borderRadius: 10,
  padding: 15,
  marginBottom: 15,
  elevation: 3,
  shadowColor: '#000', // Черная тень
  shadowOffset: { width: 0, height: 2 },
  shadowOpacity: 0.1,
  shadowRadius: 4,
},
// Заголовок графика
chartTitle: {
```

```
fontSize: 16,
fontWeight: 'bold',
marginBottom: 10,
textAlign: 'center',
color: 'white', // Темно-серый текст
},
// Стиль графика
chart: {
  borderRadius: 10,
},
// Контейнер таблиц
tablesContainer: {
  flexDirection: 'row',
  flexWrap: 'wrap',
  justifyContent: 'space-between',
},
// Контейнер отдельной таблицы
tableContainer: {
  backgroundColor: 'rgba(177, 177, 177, 0.95)', // Белый фон
  borderRadius: 10,
  padding: 10,
  marginBottom: 15,
  width: '48%',
  elevation: 2,
  shadowColor: '#000', // Черная тень
  shadowOffset: { width: 0, height: 1 },
  shadowOpacity: 0.1,
  shadowRadius: 2,
},
// Заголовок таблицы
tableTitle: {
  fontSize: 14,
  fontWeight: 'bold',
  textAlign: 'center',
  marginBottom: 8,
  color: 'white', // Темно-серый текст
},
// Шапка таблицы
tableHeader: {
  flexDirection: 'row',
  borderBottomWidth: 1,
  borderBottomColor: '#BDBDBD', // Серый разделитель
  paddingBottom: 5,
  marginBottom: 5,
},
// Текст в шапке таблицы
tableHeaderText: {
  flex: 1,
  fontWeight: 'bold',
  fontSize: 10,
  textAlign: 'center',
```

```
        color: '#424242', // Серый текст
    },
    // Стока таблицы
    tableRow: {
        flexDirection: 'row',
        paddingVertical: 3,
        borderBottomWidth: 1,
        borderBottomColor: '#EEEEEE', // Светло-серый разделитель
    },
    // Ячейка таблицы
    tableCell: {
        flex: 1,
        fontSize: 10,
        textAlign: 'center',
        color: '#212121', // Темно-серый текст
    },
});
};

export default DashboardScreen;
```

---

## Проверка работы

1. Запустите бэкенд: `python run.py`
  2. Запустите React Native приложение: `npx expo start`
  3. Проверьте переход из меню в дашборд и отображение данных с API
- 

**Итог занятия:** Вы создали полноценный бэкенд на FastAPI с API для дашбордов, настроили интеграцию с React Native приложением и добавили визуализацию данных. Теперь ваше приложение умеет не только показывать статичные данные, но и динамически обновлять аналитику с сервера.

## Занятие 8: Создание Web-версии приложения на React

[https://github.com/EugenePokh/full\\_steam\\_08\\_react\\_fastapi\\_swagger.git](https://github.com/EugenePokh/full_steam_08_react_fastapi_swagger.git)

В этом занятии мы создадим web-версию нашего приложения используя React и Vite. Это позволит нам запускать тот же функционал в браузере и расширит охват наших пользователей.

---

### FastAPI и автоматическая документация

Одним из ключевых преимуществ FastAPI является встроенная автоматическая генерация документации API.

#### Доступ к документации:

После запуска FastAPI сервера, вы можете открыть в браузере:

text

<http://localhost:8000/docs>

#### Что вы увидите в Swagger UI:

- Полный список всех доступных эндпоинтов
- Описание параметров запросов и ответов
- Возможность тестирования API прямо из браузера
- Интерактивную документацию с примерами
- Схемы данных Pydantic моделей

Эта функция значительно ускоряет разработку и тестирование API.

---

### Настройка React проекта для Web

#### Установка зависимостей:

bash

```
npm install react react-dom
```

## **Запуск development сервера:**

```
bash  
npm run dev
```

**Остановка сервера:** `Ctrl + C`

---

## **Структура React Web проекта**

```
text  
react/  
|   └── assets/  
|       ├── adaptive-icon.png  
|       ├── background.jpg  
|       ├── dashboard_icon.jpg  
|       ├── favicon.png  
|       ├── icon.png  
|       ├── map_icon.jpg  
|       ├── robot_01.jpg  
|       ├── robot_02.jpg  
|       ├── robot_03.jpg  
|       ├── robot_04.jpg  
|       ├── robot_05.jpg  
|       ├── splash-icon.png  
|       └── swipe_icon.jpg  
|   └── src/  
|       ├── App.jsx  
|       ├── Dashboard.jsx  
|       ├── Login.jsx  
|       ├── Map.jsx  
|       ├── Menu.jsx  
|       ├── Swipe.jsx  
|       └── main.jsx  
└── .gitignore  
└── index.html  
└── package-lock.json  
└── package.json  
└── vite.config.js
```

---

## Настройка Web компонентов

### 1. Главный HTML файл: index.html

Базовая структура HTML документа, корневой div для React приложения, заголовок и описание приложения

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Full Steak App</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
    <style>
      .factor-group { margin-bottom: 10px; }
      .price-result { font-size: 1.2em; font-weight: bold; color: #1D7025; }
      .section { background: #f8f9fa; padding: 15px; border-radius: 5px; margin-bottom: 20px; }
      .loading { color: #6c757d; font-style: italic; }
      .error { color: #dc3545; }
      .success { color: #198754; }
      .comparison { margin-top: 10px; padding: 10px; border-radius: 5px; }
      .comparison-positive { background-color: #d4edda; color: #155724; }
      .comparison-negative { background-color: #f8d7da; color: #721c24; }
      .login-container { max-width: 400px; margin: 100px auto; padding: 20px; }
      .card { border: none; box-shadow: 0 0 20px rgba(0,0,0,0.1); }
    </style>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

### 2. Конфигурация Vite: vite.config.js

Настройка плагинов Vite, конфигурация сервера разработки, конфигурация сборки для production

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
```

```
export default defineConfig({  
  plugins: [react()],  
  
  server: {  
    port: 3000,  
  
  },  
})
```

### 3. Точка входа React: main.jsx

Импорт основного компонента App

```
import React from 'react'  
import ReactDOM from 'react-dom/client'  
import App from './App.jsx'  
  
ReactDOM.createRoot(document.getElementById('root')).render(  
  <React.StrictMode>  
    <App />  
  </React.StrictMode>,  
)
```

### 4. Главный компонент приложения: App.jsx

Импорт всех компонентов страниц, настройка маршрутизации между страницами

```
import React, { useState, useEffect } from 'react';  
import Login from './Login.jsx';  
import Menu from './Menu.jsx';  
import Swipe from './Swipe.jsx';  
import Map from './Map.jsx';  
import Dashboard from './Dashboard.jsx';  
  
function App() {  
  const [authenticated, setAuthenticated] = useState(false);  
  const [currentScreen, setCurrentScreen] = useState('menu');  
  const [loading, setLoading] = useState(true);  
  
  useEffect(() => {  
    const token = localStorage.getItem('auth_token');  
    setAuthenticated (!!token);  
    setLoading(false);  
  }, []);
```

```

const handleLogin = () => {
  setAuthenticated(true);
  setCurrentScreen('menu');
  localStorage.setItem('auth_token', 'authenticated');
};

const handleLogout = () => {
  setAuthenticated(false);
  setCurrentScreen('menu');
  localStorage.removeItem('auth_token');
};

const navigateTo = (screen) => {
  setCurrentScreen(screen);
};

const navigateToMenu = () => {
  setCurrentScreen('menu');
};

if (loading) {
  return <div className="loading text-center mt-5">Загрузка...</div>;
}

if (!authenticated) {
  return <Login onLogin={handleLogin} />;
}

// Рендерим соответствующий экран
switch (currentScreen) {
  case 'swipe':
    return <Swipe onBack={navigateToMenu} onLogout={handleLogout} />;
  case 'dashboard':
    return <Dashboard onBack={navigateToMenu} onLogout={handleLogout} />;
  case 'map':
    return <Map onBack={navigateToMenu} onLogout={handleLogout} />;
  default:
    return <Menu onNavigate={navigateTo} onLogout={handleLogout} />;
}
}

export default App;

```

## 5. Компонент авторизации: Login.jsx

Форма ввода логина и пароля, валидация полей ввода, обработка успешной авторизации

```

import React, { useState, useEffect } from 'react';

export default function Login({ onLogin }) {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [remember, setRemember] = useState(false);

  useEffect(() => {
    const savedUsername = localStorage.getItem('username');
    const savedPassword = localStorage.getItem('password');
    if (savedUsername && savedPassword) {
      setUsername(savedUsername);
      setPassword(savedPassword);
      setRemember(true);
    }
  }, []);

  const handleLogin = () => {
    if (username === 'all1' && password === 'allPass1') {
      if (remember) {
        localStorage.setItem('username', username);
        localStorage.setItem('password', password);
      } else {
        localStorage.removeItem('username');
        localStorage.removeItem('password');
      }
      onLogin();
    } else {
      alert('Ошибка: Неверное имя пользователя или пароль');
    }
  };

  const handleKeyPress = (e) => {
    if (e.key === 'Enter') handleLogin();
  };
}

const backgroundStyle = {
  backgroundImage: 'url(..../assets/background.jpg)',
  backgroundSize: 'cover',
  backgroundPosition: 'center',
  width: '100vw',
  height: '100vh',
  display: 'flex',
  justifyContent: 'center',
  alignItems: 'center'
};

const containerStyle = {
  width: '100%',
  height: '100%',
  display: 'flex'
}

```

```

justifyContent: 'center',
alignItems: 'center',
backgroundColor: 'rgba(255, 255, 255, 0.8)'
};

const formStyle = {
  width: '85%',
  maxWidth: '400px',
  backgroundColor: 'rgba(177, 177, 177, 0.95)',
  borderRadius: '20px',
  padding: '24px',
  boxShadow: '0 4px 12px rgba(0, 0, 0, 0.25)'
};

const inputStyle = {
  width: '100%',
  border: '1px solid #ddd',
  borderRadius: '10px',
  padding: '12px',
  marginBottom: '16px',
  fontSize: '16px',
  backgroundColor: 'rgba(255, 255, 255, 0.9)',
  boxSizing: 'border-box'
};

const buttonStyle = {
  width: '100%',
  backgroundColor: '#FF69B4',
  color: 'white',
  border: 'none',
  padding: '14px',
  borderRadius: '10px',
  fontWeight: 'bold',
  fontSize: '18px',
  cursor: 'pointer',
  marginTop: '16px'
};

return (
  <div style={backgroundStyle}>
    <div style={containerStyle}>
      <div style={formStyle}>
        <h2 style={{textAlign: 'center', marginBottom: '32px', color: '#333'}}>Вход в систему</h2>

        <input
          style={inputStyle}
          type="text"
          placeholder="Логин"
          value={username}
          onChange={(e) => setUsername(e.target.value)}>
      
```

```

        onKeyPress={handleKeyPress}
    />

    <input
        style={inputStyle}
        type="password"
        placeholder="Пароль"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
        onKeyPress={handleKeyPress}
    />

        <div style={{display: 'flex', alignItems: 'center', marginBottom: '10px', cursor: 'pointer'}}>
            onClick={() => setRemember(!remember)}>
                <div style={{width: '22px', height: '22px', borderRadius: '4px', border: '1px solid #555', marginRight: '10px', backgroundColor: remember ? '#FF69B4' : '#fff'}}/>
                <span style={{fontSize: '16px', color: '#333'}}>Запомнить меня</span>
            </div>
        </div>

        <button style={buttonStyle} onClick={handleLogin}>
            Войти
        </button>
    </div>
</div>
);
}

```

## 6. Главное меню: `Menu.jsx`

Навигационные кнопки на все экраны

```

import React from 'react';

const Menu = ({ onNavigate, onLogout }) => {
    const menuItems = [
        { title: "Карта", image: "../assets/map_icon.jpg", navigateTo: 'map' },
        { title: "Свайп", image: "../assets/swipe_icon.jpg", navigateTo: 'swipe' },
        { title: "Дашборд", image: "../assets/dashboard_icon.jpg", navigateTo: 'dashboard' },
    ];

```

```

const backgroundStyle = {
  backgroundImage: 'url(..../assets/background.jpg)',
  backgroundSize: 'cover',
  backgroundPosition: 'center',
  width: '100vw',
  height: '100vh'
};

const screenStyle = {
  width: '100%',
  height: '100%',
  backgroundColor: 'rgba(206, 204, 204, 0.7)'
};

const headerStyle = {
  padding: '20px',
  display: 'flex',
  justifyContent: 'flex-end'
};

const buttonStyle = {
  backgroundColor: '#FF69B4',
  color: 'white',
  border: 'none',
  padding: '10px 20px',
  borderRadius: '10px',
  cursor: 'pointer',
  fontWeight: 'bold'
};

const containerStyle = {
  padding: '20px 0',
  display: 'flex',
  justifyContent: 'center',
  alignItems: 'center',
  minHeight: 'calc(100% - 80px)'
};

const gridStyle = {
  display: 'flex',
  flexWrap: 'wrap',
  justifyContent: 'space-around',
  maxWidth: '800px',
  width: '100%',
  gap: '20px'
};

const cardStyle = {
  width: '45%',
  maxWidth: '200px',
  marginBottom: '20px',
}

```

```

        cursor: 'pointer'
    };

const imageStyle = {
    width: '100%',
    height: '150px',
    objectFit: 'cover',
    borderRadius: '10px'
};

const overlayStyle = {
    position: 'relative',
    bottom: '25px',
    background: 'rgba(0, 0, 0, 0.5)',
    color: 'rgba(255, 255, 255, 0.7)',
    textAlign: 'center',
    padding: '5px 0',
    borderBottomLeftRadius: '10px',
    borderBottomRightRadius: '10px'
};

return (
    <div style={backgroundStyle}>
        <div style={screenStyle}>
            <div style={headerStyle}>
                <button style={buttonStyle} onClick={onLogout}>Выйти</button>
            </div>

            <div style={containerStyle}>
                <div style={gridStyle}>
                    {menuItems.map((item, index) => (
                        <div key={index} style={cardStyle} onClick={() =>
onNavigate(item.navigateTo)}>
                            <img src={item.image} alt={item.title} style={imageStyle} />
                            <div style={overlayStyle}>
                                <span>{item.title}</span>
                            </div>
                        </div>
                    )));
                </div>
            </div>
        </div>
    );
};

export default Menu;

```

## 7. Экран карты: Map.jsx

## Интеграция карт, отображение маркеров и данных, геолокация пользователя

```
import React, { useState, useEffect, useRef } from 'react';

const Map = ({ onBack, onLogout }) => {
  const [mapReady, setMapReady] = useState(false);
  const [error, setError] = useState(null);
  const mapInstanceRef = useRef(null);

  useEffect(() => {
    initWebMap();
  }, []);

  const initWebMap = () => {
    if (window.L && document.getElementById('map-container')) {
      createWebMap();
      return;
    }
    loadLeaflet();
  };

  const loadLeaflet = () => {
    if (window.L) {
      createWebMap();
      return;
    }

    const script = document.createElement('script');
    script.src = 'https://unpkg.com/leaflet@1.9.4/dist/leaflet.js';
    script.onload = () => setTimeout(createWebMap, 100);
    script.onerror = () => setError('Не удалось загрузить карту');
    document.head.appendChild(script);

    const link = document.createElement('link');
    link.rel = 'stylesheet';
    link.href = 'https://unpkg.com/leaflet@1.9.4/dist/leaflet.css';
    document.head.appendChild(link);
  };

  const createWebMap = () => {
    const mapContainer = document.getElementById('map-container');
    if (!mapContainer || !window.L) {
      setTimeout(createWebMap, 100);
      return;
    }

    try {
      if (mapInstanceRef.current) mapInstanceRef.current.remove();
      const map = window.L.map('map-container').setView([53.1959, 45.0183], 13);
      window.L.tileLayer('https://s.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '&copy; OpenStreetMap contributors',
      }).addTo(map);
    } catch (err) {
      setError(err.message);
    }
  };
}

export default Map;
```

```

        maxZoom: 19,
    }).addTo(map);
    mapInstanceRef.current = map;
    setMapReady(true);
    setError(null);
} catch (err) {
    setError('Ошибка инициализации карты');
}
};

const handleFindMe = () => {
if (!navigator.geolocation) {
    alert("Геолокация не поддерживается браузером");
    return;
}

navigator.geolocation.getCurrentPosition(
    (position) => {
        const { latitude, longitude } = position.coords;
        centerWebMap(latitude, longitude);
    },
    () => alert('Не удалось получить местоположение')
);
};

const centerWebMap = (lat, lng) => {
if (!mapInstanceRef.current) return;
mapInstanceRef.current.setView([lat, lng], 15);

if (window.currentMarker)
mapInstanceRef.current.removeLayer(window.currentMarker);
window.currentMarker = window.L.marker([lat, lng])
    .addTo(mapInstanceRef.current)
    .bindPopup(`<div style="text-align:center"><strong>Вы
здесь!</strong><br>Широта: ${lat.toFixed(6)}<br>Долгота:
${lng.toFixed(6)}</div>`)
    .openPopup();
};

const containerStyle = {
    width: '100vw',
    height: '100vh',
    position: 'relative'
};

const headerStyle = {
    position: 'absolute',
    top: 0,
    left: 0,
    right: 0,
    display: 'flex',

```

```

justifyContent: 'space-between',
padding: '20px',
zIndex: 1000
};

const buttonStyle = {
  backgroundColor: '#FF69B4',
  color: 'white',
  border: 'none',
  padding: '10px 20px',
  borderRadius: '10px',
  cursor: 'pointer',
  fontWeight: 'bold',
  zIndex: 1001
};

const mapStyle = {
  width: '100%',
  height: '100%',
  backgroundColor: '#f0f0f0'
};

const findMeStyle = {
  position: 'absolute',
  bottom: '40px',
  right: '20px',
  backgroundColor: mapReady ? '#007AFF' : '#ccc',
  color: 'white',
  border: 'none',
  padding: '14px 25px',
  borderRadius: '25px',
  fontWeight: 'bold',
  cursor: mapReady ? 'pointer' : 'not-allowed',
  zIndex: 1000
};

return (
  <div style={containerStyle}>
    <div id="map-container" style={mapStyle} />

    {!mapReady && !error && (
      <div style={{
        position: 'absolute',
        top: 0,
        left: 0,
        right: 0,
        bottom: 0,
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
        backgroundColor: 'rgba(255,255,255,0.9)'
      }}>
    
```

```

        zIndex: 1000
    }}>
    <div>Загрузка карты...</div>
</div>
)}

<button style={findMeStyle} onClick={handleFindMe} disabled={!mapReady}>
    Найти меня
</button>

{error && (
    <div style={{
        position: 'absolute',
        top: '20px',
        left: '20px',
        right: '20px',
        backgroundColor: 'rgba(255,255,255,0.95)',
        padding: '15px',
        borderRadius: '10px',
        border: '1px solid #FF3B30',
        zIndex: 1000,
        display: 'flex',
        flexDirection: 'column',
        alignItems: 'center'
    }}>
        <div style={{color: '#FF3B30', marginBottom: '10px'}}>{error}</div>
        <button style={{...buttonStyle, backgroundColor: '#FF3B30'}}>
            onClick={initWebMap}
                Повторить
            </button>
        </div>
    )}>
    </div>
);

};

export default Map;

```

## 8. Экран свайпов: Swipe.jsx

Отображение карточек для свайпа, обработка жестов свайпа, анимации переходов между карточками, логика принятия решений

```

import React, { useState, useRef } from 'react';

const cards = [
    { id: "01", image: "../assets/robot_01.jpg" },
    { id: "02", image: "../assets/robot_02.jpg" },
    { id: "03", image: "../assets/robot_03.jpg" },

```

```

{ id: "04", image: "../assets/robot_04.jpg" },
{ id: "05", image: "../assets/robot_05.jpg" },
];

const swipeMap = {
  "01": { left: "03", right: "02" },
  "02": { left: "04", right: "05" },
  "03": { left: "04", right: "05" },
  "04": { left: "01", right: "02" },
  "05": { left: "02", right: "01" },
};

const Swipe = ({ onBack, onLogout }) => {
  const [currentIndex, setCurrentIndex] = useState(0);
  const [position, setPosition] = useState({ x: 0, y: 0 });
  const [isDragging, setIsDragging] = useState(false);
  const startPos = useRef({ x: 0, y: 0 });

  // --- мышь ---
  const handleMouseDown = (e) => {
    setIsDragging(true);
    startPos.current = { x: e.clientX, y: e.clientY };
  };

  const handleMouseMove = (e) => {
    if (!isDragging) return;
    setPosition({
      x: e.clientX - startPos.current.x,
      y: e.clientY - startPos.current.y,
    });
  };

  const handleMouseUp = () => {
    if (!isDragging) return;
    handleSwipeEnd();
  };

  // --- touch ---
  const handleTouchStart = (e) => {
    const touch = e.touches[0];
    startPos.current = { x: touch.clientX, y: touch.clientY };
    setIsDragging(true);
  };

  const handleTouchMove = (e) => {
    if (!isDragging) return;
    const touch = e.touches[0];
    const deltaX = touch.clientX - startPos.current.x;
    const deltaY = touch.clientY - startPos.current.y;
    if (Math.abs(deltaX) > Math.abs(deltaY)) e.preventDefault(); // блокируем скролл
  };
}

```

```

        setPosition({ x: deltaX, y: deltaY });
    };

const handleTouchEnd = () => {
    if (!isDragging) return;
    handleSwipeEnd();
};

// --- завершение свайпа ---
const handleSwipeEnd = () => {
    const direction =
        position.x > 80 ? 'right' : position.x < -80 ? 'left' : null;

    if (direction) {
        const cardId = cards[currentIndex].id;
        const nextId = swipeMap[cardId][direction];
        const nextIndex = cards.findIndex((c) => c.id === nextId);

        setPosition({
            x: direction === 'right' ? window.innerWidth : -window.innerWidth,
            y: 0,
        });

        setTimeout(() => {
            setCurrentIndex(nextIndex >= 0 ? nextIndex : 0);
            setPosition({ x: 0, y: 0 });
            setIsDragging(false);
        }, 300);
    } else {
        setPosition({ x: 0, y: 0 });
        setIsDragging(false);
    }
};

// --- СТИЛИ ---
const backgroundStyle = {
    backgroundImage: 'url(..../assets/background.jpg)',
    backgroundSize: 'cover',
    backgroundPosition: 'center',
    width: '100vw',
    height: '100vh',
    position: 'relative',
    overflow: 'hidden',
};

const cardStyle = {
    position: 'absolute',
    width: '80%',
    maxWidth: '400px',
    height: '60%',
    maxHeight: '500px',
}

```

```

borderRadius: '20px',
backgroundColor: 'rgba(177, 177, 177, 0.95)',
display: 'flex',
flexDirection: 'column',
justifyContent: 'center',
alignItems: 'center',
cursor: isDragging ? 'grabbing' : 'grab',
transform: `translateX(${position.x}px) translateY(${position.y}px)
rotate(${position.x / 20}deg),
transition: isDragging ? 'none' : 'transform 0.3s ease-out',
boxShadow: '0 8px 32px rgba(0, 0, 0, 0.4)',
border: '1px solid rgba(255, 255, 255, 0.3)',
userSelect: 'none',
};

const buttonStyle = {
  backgroundColor: '#FF69B4',
  color: 'white',
  border: 'none',
  padding: '10px 20px',
  borderRadius: '10px',
  cursor: 'pointer',
  fontWeight: 'bold',
};

const bottomBarStyle = {
  position: 'absolute',
  bottom: '60px',
  display: 'flex',
  justifyContent: 'space-around',
  width: '80%',
  maxWidth: '400px',
  backgroundColor: 'rgba(255, 255, 255, 0.8)',
  borderRadius: '25px',
  padding: '12px 20px',
  boxShadow: '0 4px 16px rgba(0, 0, 0, 0.2)',
};

return (
  <div
    style={backgroundStyle}
    onMouseMove={handleMouseMove}
    onMouseUp={handleMouseUp}
  >
    {/* верхняя панель */}
    <div
      style={{
        position: 'absolute',
        top: 0,
        left: 0,
        right: 0,

```

```

        display: 'flex',
        justifyContent: 'space-between',
        padding: '20px',
        zIndex: 1000,
    )}
>
    <button style={buttonStyle} onClick={onBack}>← Назад</button>
</div>

/* карточка */
<div
    style={{
        width: '100%',
        height: '100%',
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
        position: 'relative',
    }}
>
<div
    style={cardStyle}
    onMouseDown={handleMouseDown}
    onTouchStart={handleTouchStart}
    onTouchMove={handleTouchMove}
    onTouchEnd={handleTouchEnd}
>
    <img
        src={cards[currentIndex].image}
        alt={`Robot ${cards[currentIndex].id}`}
        style={{
            width: '75%',
            height: '75%',
            objectFit: 'contain',
            borderRadius: '10px',
            pointerEvents: 'none',
        }}
        onDragStart={(e) => e.preventDefault()}
    />
    <div style={{ marginTop: '15px', fontSize: '28px', fontWeight: 'bold', color: '#333' }}>
        {cards[currentIndex].id}
    </div>
</div>

/* индикаторы */
<div style={bottomBarStyle}>
    {cards.map((card, idx) => (
        <span
            key={card.id}
            style={{

```

```

        fontSize: idx === currentIndex ? '22px' : '18px',
        color: idx === currentIndex ? '#FF69B4' : '#666',
        fontWeight: idx === currentIndex ? 'bold' : '600',
        transition: 'all 0.3s ease',
    )}
>
    {card.id}
</span>
))}
</div>

/* подсказка */
<div
    style={{
        position: 'absolute',
        top: '80px',
        backgroundColor: 'rgba(255, 255, 255, 0.85)',
        padding: '10px 20px',
        borderRadius: '20px',
        fontSize: '16px',
        color: '#333',
        fontWeight: '600',
    }}
>
    Свайпните влево или вправо
</div>
</div>
</div>
);
};

export default Swipe;

```

## 9. Дашиборд аналитики: Dashboard.jsx

[Загрузка данных с FastAPI бэкенда, отображение графиков и диаграмм, настройки дашборда, обновление данных]

```

import React, { useState, useEffect } from 'react';
import { Line } from 'react-chartjs-2';
import {
    Chart as ChartJS,
    CategoryScale,
    LinearScale,
    PointElement,
    LineElement,

```

```

Title,
Tooltip,
Legend,
Filler
} from 'chart.js';

ChartJS.register(
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend,
  Filler
);

const API_BASE_URL = 'http://localhost:8000/api';

const Dashboard = ({ onBack, onLogout }) => {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [actionLoading, setActionLoading] = useState({ upload: false, analysis: false });

  useEffect(() => {
    fetchDashboardData();
  }, []);

  const fetchDashboardData = async () => {
    try {
      setLoading(true);
      const response = await fetch(` ${API_BASE_URL}/dashboard/data`);
      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }
      const result = await response.json();
      setData(result);
    } catch (error) {
      console.error('Error fetching dashboard data:', error);
      alert('Ошибка: Не удалось загрузить данные с сервера');
    } finally {
      setLoading(false);
    }
  };
};

const runScript = async (endpoint, actionPerformed) => {
  try {
    setActionLoading(prev => ({ ...prev, [actionName]: true }));
    const response = await fetch(` ${API_BASE_URL}/dashboard/${endpoint}` , {

```

```

    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
  });

  if (!response.ok) {
    throw new Error(`HTTP error! status: ${response.status}`);
  }

  const result = await response.json();
  alert('Успех: ' + result.message);

  setTimeout(() => {
    fetchDashboardData();
  }, 2000);
}

} catch (error) {
  console.error(`Error running script ${actionName}:`, error);
  alert(`Ошибка: Не удалось выполнить ${actionName}`);
} finally {
  setActionLoading(prev => ({ ...prev, [actionName]: false }));
}
};

const chartOptions = {
  responsive: true,
  maintainAspectRatio: false,
  plugins: {
    legend: {
      position: 'top',
      labels: {
        usePointStyle: true,
        padding: 20,
      }
    },
  },
  scales: {
    y: {
      beginAtZero: false,
      grid: {
        color: 'rgba(255,255,255,0.1)'
      },
      ticks: {
        color: '#333'
      }
    },
    x: {
      grid: {
        color: 'rgba(255,255,255,0.1)'
      },
    }
  }
};

```

```

        ticks: {
          color: '#333'
        }
      },
    },
  );
}

const getComparisonChartData = () => {
  if (!data?.comparison_data) return null;

  return {
    labels: data.comparison_data.map(item => item.year.toString()),
    datasets: [
      {
        label: 'Linear Regression',
        data: data.comparison_data.map(item => item.linear),
        borderColor: '#FF69B4',
        backgroundColor: 'rgba(255, 105, 180, 0.1)',
        tension: 0.4,
        fill: true,
      },
      {
        label: 'Gradient Boosting',
        data: data.comparison_data.map(item => item.gradient_boosting),
        borderColor: '#E91E63',
        backgroundColor: 'rgba(233, 30, 99, 0.1)',
        tension: 0.4,
        fill: true,
      },
      {
        label: 'Neural Network',
        data: data.comparison_data.map(item => item.neural_network),
        borderColor: '#F8BBB0',
        backgroundColor: 'rgba(248, 187, 208, 0.1)',
        tension: 0.4,
        fill: true,
      },
      {
        label: 'Random Forest',
        data: data.comparison_data.map(item => item.random_forest),
        borderColor: '#AD1457',
        backgroundColor: 'rgba(173, 20, 87, 0.1)',
        tension: 0.4,
        fill: true,
      },
    ],
  };
};

const getForecastChartData = () => {
  if (!data?.historical_data || !data?.comparison_data) return null;
}

```

```

    const historicalYears = data.historical_data.slice(-5).map(item =>
item.year);
    const historicalValues = data.historical_data.slice(-5).map(item =>
item.crude_oil);
    const forecastYears = data.comparison_data.map(item => item.year);
    const forecastValues = data.comparison_data.map(item => item.linear);

    return {
        labels: [...historicalYears, ...forecastYears],
        datasets: [
            {
                label: 'Historical Data',
                data: [...historicalValues, ...forecastValues.map((_, i) => i === 0 ?
historicalValues[historicalValues.length - 1] : null)],
                borderColor: '#4CAF50',
                backgroundColor: 'rgba(76, 175, 80, 0.1)',
                tension: 0.4,
                fill: false,
            },
            {
                label: 'Linear Regression Forecast',
                data: [...historicalValues.map(() => null), ...forecastValues],
                borderColor: '#FF69B4',
                backgroundColor: 'rgba(255, 105, 180, 0.1)',
                tension: 0.4,
                fill: '+1',
                borderDash: [5, 5],
            },
        ],
    };
};

const renderForecastTable = (title, forecastData) => (
    <div key={title} style={{
        backgroundColor: 'rgba(177, 177, 177, 0.95)',
        borderRadius: '10px',
        padding: '15px',
        marginBottom: '15px',
        width: '48%',
        minWidth: '280px',
        boxShadow: '0 2px 8px rgba(0, 0, 0, 0.2)'
    }}>
        <div style={{
            fontSize: '16px',
            fontWeight: 'bold',
            textAlign: 'center',
            marginBottom: '12px',
            color: 'white'
        }}>{title}</div>

```

```

        <div style={{
            display: 'flex',
            borderBottom: '2px solid #BDBDBD',
            paddingBottom: '8px',
            marginBottom: '8px'
        }}>
            <div style={{flex: 1, fontWeight: 'bold', fontSize: '12px', textAlign: 'center', color: '#424242'}}>Год</div>
            <div style={{flex: 1, fontWeight: 'bold', fontSize: '12px', textAlign: 'center', color: '#424242'}}>Прогноз</div>
            <div style={{flex: 1, fontWeight: 'bold', fontSize: '12px', textAlign: 'center', color: '#424242'}}>R2</div>
        </div>

        {forecastData?.map((item, index) => (
            <div key={index} style={{
                display: 'flex',
                padding: '8px 0',
                borderBottom: '1px solid #EEEEEE',
                backgroundColor: index % 2 === 0 ? 'rgba(255,255,255,0.1)' : 'transparent'
            }}>
                <div style={{flex: 1, fontSize: '12px', textAlign: 'center', fontWeight: '600', color: '#212121'}}>{item.year}</div>
                <div style={{flex: 1, fontSize: '12px', textAlign: 'center', color: '#212121'}}>
                    {item.crude_oil_forecast?.toFixed(2)}
                </div>
                <div style={{
                    flex: 1,
                    fontSize: '12px',
                    textAlign: 'center',
                    color: item.r2 > 0 ? '#FF69B4' : '#E91E63',
                    fontWeight: 'bold'
                }}>
                    {item.r2?.toFixed(4)}
                </div>
            </div>
        )));
    </div>
);

const backgroundStyle = {
    backgroundImage: 'url(..../assets/background.jpg)',
    backgroundSize: 'cover',
    backgroundPosition: 'center',
    width: '100vw',
    height: '100vh'
};

const screenStyle = {

```

```

width: '100%',
height: '100%',
backgroundColor: 'rgba(128, 128, 128, 0.7)',
overflowY: 'auto'
};

const headerStyle = {
  display: 'flex',
  justifyContent: 'space-between',
  padding: '20px',
  position: 'sticky',
  top: 0,
  zIndex: 1000,
  backgroundColor: 'rgba(128, 128, 128, 0.9)',
  backdropFilter: 'blur(10px)'
};

const buttonStyle = {
  backgroundColor: '#FF69B4',
  color: 'white',
  border: 'none',
  padding: '12px 24px',
  borderRadius: '10px',
  cursor: 'pointer',
  fontWeight: 'bold',
  fontSize: '14px',
  boxShadow: '0 2px 8px rgba(0, 0, 0, 0.2)',
  transition: 'background-color 0.3s'
};

const actionBarStyle = {
  display: 'flex',
  alignItems: 'center',
  backgroundColor: '#FF69B4',
  color: 'white',
  border: 'none',
  padding: '15px 20px',
  borderRadius: '10px',
  cursor: 'pointer',
  flex: '0 1 45%',
  minWidth: '250px',
  justifyContent: 'center',
  boxShadow: '0 2px 8px rgba(0, 0, 0, 0.2)',
  transition: 'all 0.3s'
};

if (loading) {
  return (
    <div style={backgroundStyle}>
      <div style={{
        display: 'flex',

```

```

        flexDirection: 'column',
        justifyContent: 'center',
        alignItems: 'center',
        height: '100%',
        backgroundColor: 'rgba(128, 128, 128, 0.7)'
    }}>
    <div style={{{
        width: '50px',
        height: '50px',
        border: '4px solid #f3f3f3',
        borderTop: '4px solid #FF69B4',
        borderRadius: '50%',
        animation: 'spin 1s linear infinite'
    }} />
    <div style={{color: 'white', marginTop: '20px', fontSize: '18px',
fontWeight: 'bold'}}>
        Загрузка данных с сервера...
    </div>
</div>
<style>{
    @keyframes spin {
        0% { transform: rotate(0deg); }
        100% { transform: rotate(360deg); }
    }
`}</style>
</div>
);
}
}

const comparisonChartData = getComparisonChartData();
const forecastChartData = getForecastChartData();

return (
    <div style={backgroundStyle}>
        <div style={screenStyle}>
            <div style={headerStyle}>
                <button
                    style={buttonStyle}
                    onMouseOver={(e) => e.target.style.backgroundColor = '#e55a9d'}
                    onMouseOut={(e) => e.target.style.backgroundColor = '#FF69B4'}
                    onClick={onBack}
                >
                    ← Назад в меню
                </button>
            </div>

            <div style={{
                padding: '20px',
                flexGrow: 1,
                maxWidth: '1200px',
                margin: '0 auto'

```

```

    >
      <div style={{|
        display: 'flex',
        justifyContent: 'space-around',
        marginBottom: '30px',
        flexWrap: 'wrap',
        gap: '15px'
      }}>
        <button
          style={{|
            ...actionButtonStyle,
            backgroundColor: actionLoading.upload ? '#9E9E9E' : '#FF69B4'
          }}
          onMouseOver={(e) => !actionLoading.upload &&
(e.target.style.backgroundColor = '#e55a9d')}
          onMouseOut={(e) => !actionLoading.upload &&
(e.target.style.backgroundColor = '#FF69B4')}
          onClick={() => runScript('upload-data', 'upload')}
          disabled={actionLoading.upload}
        >
          {actionLoading.upload ? (
            <div style={{|
              width: '20px',
              height: '20px',
              border: '2px solid transparent',
              borderTop: '2px solid white',
              borderRadius: '50%',
              animation: 'spin 1s linear infinite',
              marginRight: '10px'
            }} />
          ) : (
            <span style={{marginRight: '10px', fontSize: '18px'}}>⏏</span>
          )}
          <span style={{fontWeight: 'bold', fontSize: '14px'}}>
            {actionLoading.upload ? 'Выгрузка...' : 'Выгрузить данные в
PostgreSQL'}
          </span>
        </button>
      <button
        style={{|
          ...actionButtonStyle,
          backgroundColor: actionLoading.analysis ? '#9E9E9E' : '#FF69B4'
        }}
        onMouseOver={(e) => !actionLoading.analysis &&
(e.target.style.backgroundColor = '#e55a9d')}
        onMouseOut={(e) => !actionLoading.analysis &&
(e.target.style.backgroundColor = '#FF69B4')}
        onClick={() => runScript('run-analysis', 'analysis')}
        disabled={actionLoading.analysis}
      >

```

```

{actionLoading.analysis ? (
  <div style={{{
    width: '20px',
    height: '20px',
    border: '2px solid transparent',
    borderTop: '2px solid white',
    borderRadius: '50%',
    animation: 'spin 1s linear infinite',
    marginRight: '10px'
  }}} />
) : (
  <span style={{marginRight: '10px', fontSize: '18px'}}>Изучение</span>
)
<span style={{fontWeight: 'bold', fontSize: '14px'}}>
  {actionLoading.analysis ? 'Анализ...' : 'Выполнить анализ (ИИ)'}
</span>
</button>
</div>

{comparisonChartData && (
  <div style={{{
    backgroundColor: 'rgba(177, 177, 177, 0.95)',
    borderRadius: '15px',
    padding: '25px',
    marginBottom: '25px',
    boxShadow: '0 4px 15px rgba(0, 0, 0, 0.2)'
  }}}>
    <div style={{{
      fontSize: '20px',
      fontWeight: 'bold',
      textAlign: 'center',
      marginBottom: '20px',
      color: 'white'
    }}}>
      Сравнение прогнозов разных моделей
    </div>
    <div style={{ height: '400px' }}>
      <Line data={comparisonChartData} options={chartOptions} />
    </div>
  </div>
) }

{forecastChartData && (
  <div style={{{
    backgroundColor: 'rgba(177, 177, 177, 0.95)',
    borderRadius: '15px',
    padding: '25px',
    marginBottom: '25px',
    boxShadow: '0 4px 15px rgba(0, 0, 0, 0.2)'
  }}}>
    <div style={{{

```

```

        fontSize: '20px',
        fontWeight: 'bold',
        textAlign: 'center',
        marginBottom: '20px',
        color: 'white'
    }}>
    Исторические данные и прогноз Crude Oil
</div>
<div style={{ height: '400px' }}>
    <Line data={forecastChartData} options={chartOptions} />
</div>
</div>
)
}

<div style={{
    display: 'flex',
    flexWrap: 'wrap',
    justifyContent: 'space-between',
    gap: '20px',
    marginTop: '30px'
}}>
    {data?.improved_linear && renderForecastTable('Linear Regression',
data.improved_linear)}
    {data?.gradient_boosting && renderForecastTable('Gradient Boosting',
data.gradient_boosting)}
    {data?.improved_nn && renderForecastTable('Neural Network',
data.improved_nn)}
    {data?.improved_rf && renderForecastTable('Random Forest',
data.improved_rf)}
    </div>
    </div>
    </div>
    </div>
);
};

export default Dashboard;

```

---

## Особенности Web-версии

- **Адаптивный дизайн** — приложение должно корректно отображаться на разных размерах экранов
- **Веб-специфичные жесты** — замена тапов на клики, адаптация свайпов для мыши
- **Браузерные API** — использование Local Storage вместо Async Storage

- **Оптимизация загрузки** — код-сплиттинг для больших приложений
- 

## Проверка работы

1. Запустите FastAPI бэкенд: `python run.py`
  2. Запустите React приложение: `npm run dev`
  3. Откройте браузер по указанному адресу (обычно `http://localhost:5173`)
  4. Протестируйте все функции приложения
- 

**Итог занятия:** Вы создали полнофункциональную web-версию вашего приложения на React, которая использует тот же бэкенд на FastAPI. Теперь ваш проект поддерживает мобильные устройства (React Native) и веб-браузеры (React), демонстрируя мощь кроссплатформенной разработки.

## Занятие 9: Сборка и публикация приложения

[https://github.com/EugenePokh/full\\_steam\\_09\\_react\\_native\\_android\\_build](https://github.com/EugenePokh/full_steam_09_react_native_android_build)

В этом занятии мы перейдем от разработки к публикации — соберем наше приложение в готовый билд для Android и рассмотрим варианты публикации в магазинах приложений.

---

### Подготовка к сборке приложения

Перед созданием билда необходимо зарегистрироваться в платформе Expo и настроить окружение для сборки.

#### Регистрация в Expo:

- Перейдите на сайт: <https://expo.dev/>
  - Создайте аккаунт (бесплатно)
  - Запомните логин и пароль — они понадобятся для входа через консоль
- 

### Настройка проекта для сборки

#### Обновление версий зависимостей:

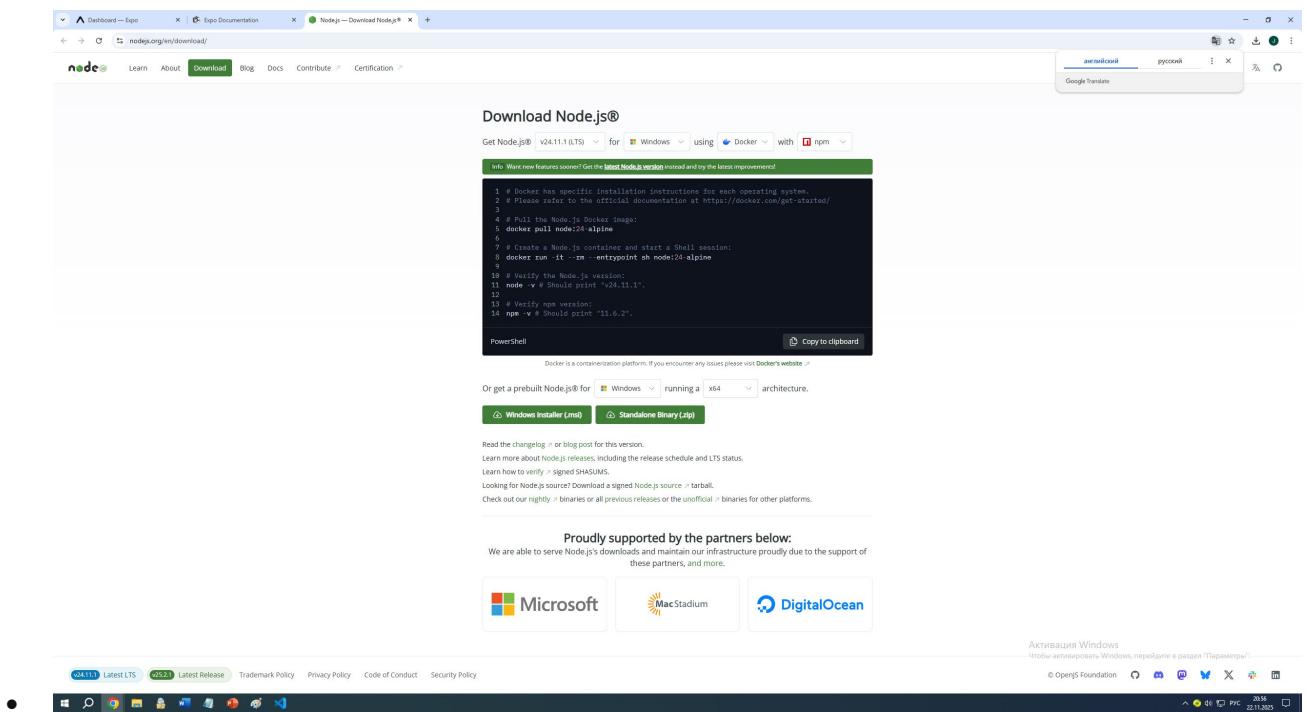
Перед сборкой необходимо обновить версии пакетов в `package.json` для совместимости с последними версиями Expo:

```
json
{
  "name": "react_native_basic",
  "version": "1.0.0",
  "main": "index.js",
  "private": true,
  "scripts": {
    "start": "expo start",
    "build": "react-native bundle --platform android --dev false --entry-file index.js --bundle-output android/app/src/main/assets/index.android.bundle --assets-dest android/app/src/main/resources/react"
  }
}
```

```
"android": "expo start --android",
"ios": "expo start --ios",
"web": "expo start --web"
},
"dependencies": {
  "expo": "~52.0.46",
  "expo-location": "~17.0.0",
  "expo-status-bar": "~2.0.0",
  "react": "18.3.1",
  "react-dom": "18.3.1",
  "react-native": "^0.76.9",
  "react-native-web": "~0.19.13",
  "@react-native-async-storage/async-storage": "^1.23.1",
  "@react-navigation/native": "^7.0.19",
  "@react-navigation/native-stack": "^7.3.3",
  "react-native-chart-kit": "^6.12.0"
},
"devDependencies": {
  "@babel/core": "^7.20.0"
}
}
```

### **Установка необходимых версий:**

- Убедитесь, что у вас установлена LTS версия Node.js v22:  
<https://nodejs.org/en/download/>



## Процесс сборки приложения

### Подготовка проекта:

bash

```
# Удаление старых зависимостей (если были проблемы)
rm -rf node_modules package-lock.json
```

```
# Установка свежих зависимостей
npm install
```

```
# Установка Expo CLI глобально
npm install -g @expo/cli
```

```
# Установка EAS CLI для сборки
npm install -g @expo/eas-cli
```

**Логин и сборка:**

```
# Вход в аккаунт Expo (следуйте инструкциям в терминале)
eas login
```

```
# Сборка Android приложения
eas build --platform android
```

Во время сборки отвечайте утвердительно на все вопросы системы.

## Получение и конвертация билда

### Скачивание билда:

- Перейдите на <https://expo.dev/>
- В разделе Overview выберите ваш проект
- Перейдите в раздел Builds и выберите последнюю успешную сборку
- Скачайте файл в формате .aab (Android App Bundle)

### Конвертация .aab в .apk:

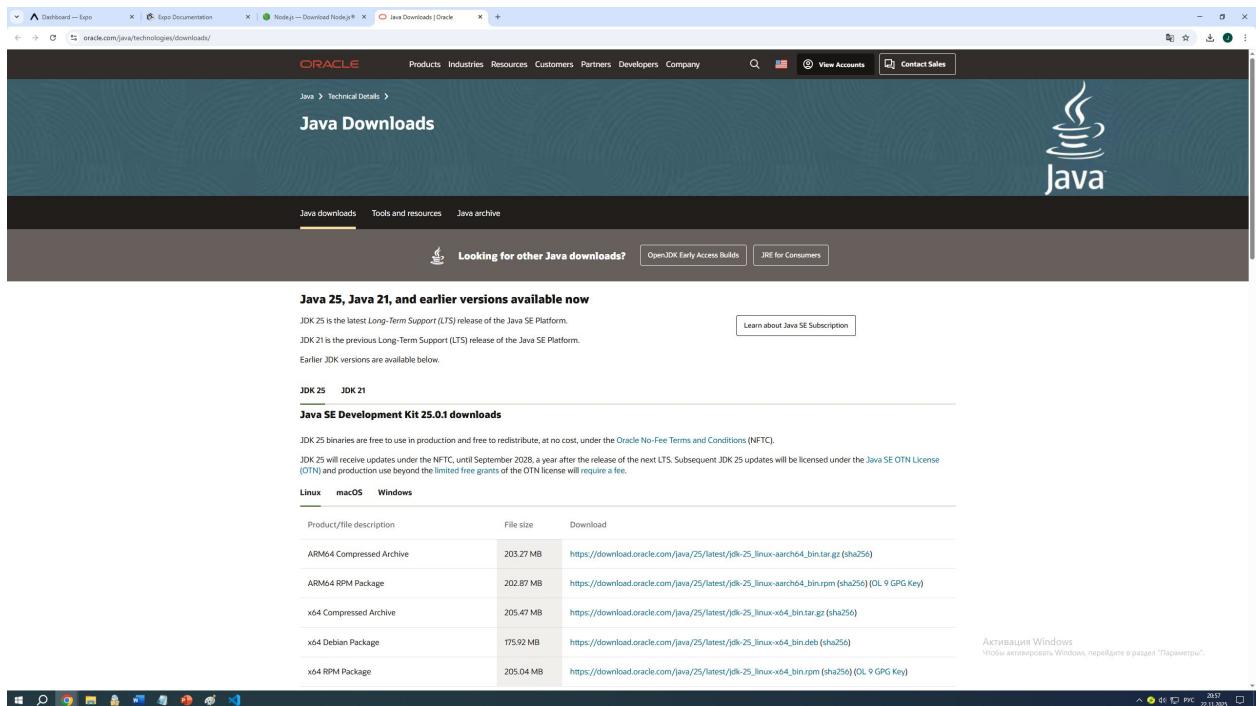
#### Создание ключа подписи:

bash

```
keytool -genkey -v -keystore "C:\path\to\keystore.jks" -alias MyKeyAlias -keyalg RSA  
-keysize 2048 -validity 10000 -storepass ваш_пароль -keypass ваш_пароль -dname "CN=Na  
me Surname, OU=Department, O=Organization, L=City, ST=Region, C=RU"
```

#### Установка Java (если не установлена):

- Скачайте с официального сайта: <https://www.oracle.com/java/technologies/downloads/>



## **Конвертация с помощью bundletool:**

bash

```
java -jar "C:\path\to\bundletool-all-1.18.1.jar" build-apks --bundle="C:\path\to\application.aab" --output="C:\path\to\application.apks" --ks="C:\path\to\keystore.jks" --ks-key-alias="MyKeyAlias" --mode=universal
```

## **Извлечение APK файла:**

1. Переименуйте application.apks в application.zip
  2. Разархивируйте файл
  3. Внутри вы найдете universal.apk — готовый к установке файл
- 

## **Важные замечания для production**

### **Замена localhost на реальный IP:**

Перед публикацией обязательно замените в файле dashboard.js:

javascript

```
const API_BASE_URL = 'http://localhost:8000/api';
```

на реальный адрес вашего сервера, иначе приложение не сможет подключиться к бэкенду.

---

## **Публикация в магазинах приложений**

### **Рекомендуемые платформы:**



**RuStore** (основная рекомендация для России)

- Регистрация: <https://console.rustore.ru/sign-in>
- Требуется статус самозанятого (можно оформить через сайт)
- Популярен в России из-за ограничений Google



**Huawei AppGallery**

- Регистрация: <https://developer.huawei.com/consumer/en/console/service>
- Доступен без иностранных карт
- Возможность вывода средств на карту МИР

## **X Google Play** (не рекомендуется для начала)

- Комиссия \$25 при регистрации
- Требует иностранную карту и резидентство
- Сложная верификация для российских разработчиков
- Необходимо привлечение 20+ тестеров для первой публикации

## **X Samsung Galaxy Store**

- Требует PayPal и иностранную карту

## **X Apple App Store**

- Годовая комиссия \$100
  - Сложности с получением выплат для российских разработчиков
  - Требует иностранную карту и резидентство
- 

## **Заключительные шаги**

После успешной сборки и тестирования APK файла:

1. Зарегистрируйтесь в выбранном магазине приложений
  2. Подготовьте описание приложения, скриншоты и иконки
  3. Загрузите приложение в магазин
  4. Пройдите модерацию
  5. Начните продвижение вашего приложения
- 

**Итог занятия:** Вы научились создавать production-сборку приложения, конвертировать ее в устанавливаемый файл и получили обзор доступных платформ для публикации. Теперь ваше приложение готово для распространения среди пользователей!

## Занятие 10: Размещение проекта на сервере

[https://github.com/EugenePokh/full\\_steam\\_10\\_timeweb.git](https://github.com/EugenePokh/full_steam_10_timeweb.git)

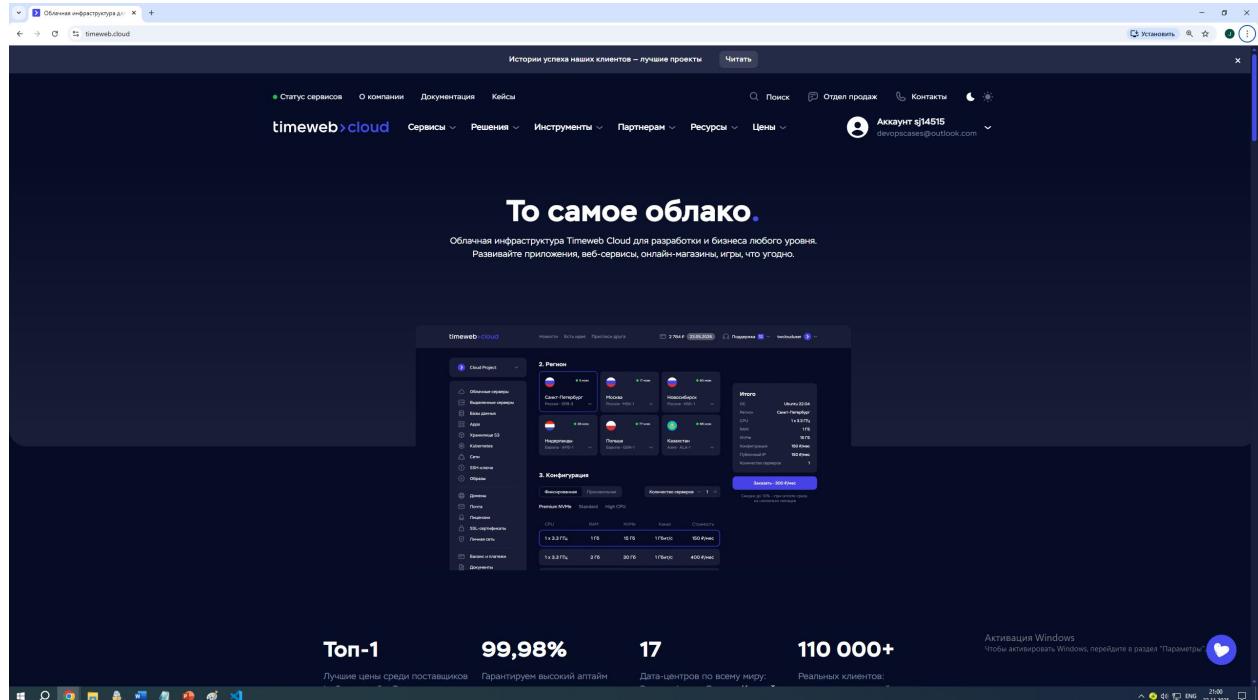
В этом заключительном занятии мы развернем наш полноценный проект на облачном сервере, чтобы сделать его доступным для пользователей из любого места.

### Регистрация и настройка облачного сервера

#### Выбор хостинг-провайдера:

Для размещения нашего проекта мы будем использовать **TimeWeb Cloud** — удобный и доступный облачный хостинг для разработчиков.

- Регистрация: <https://timeweb.cloud/>



- После регистрации создайте новый облачный сервер
- Важно:** При создании сервера обязательно добавьте **публичный IP-адрес**
- Запомните данные для доступа (IP-адрес, логин root и пароль)

## **Перенос файлов проекта на сервер**

Существует несколько способов загрузки файлов на сервер. Мы используем метод через SCP (Secure Copy).

### **Через Git Bash или терминал с SSH:**

bash

```
scp "C:\путь\к\вашему\проекту\full_steaк.zip" root@X.XXX.XXX.XXX:~
```

Где:

- C:\путь\к\вашему\проекту\full\_steaк.zip — полный путь к архиву вашего проекта на локальном компьютере
- X.XXX.XXX.XXX — публичный IP-адрес вашего сервера
- ~ — домашняя директория пользователя root на сервере

После ввода команды система запросит пароль — скопируйте его из раздела "Доступ" в панели управления TimeWeb и вставьте в терминал.

---

## **Настройка сервера и распаковка проекта**

### **Подключение к серверу:**

1. Перейдите в консоль управления на <https://timeweb.cloud/>
2. Запустите консоль вашего сервера
3. Введите логин root и пароль из раздела "Доступ"

### **Базовые команды настройки:**

bash

```
# Обновление пакетов сервера
sudo apt update
```

```
# Установка архиватора
sudo apt install unzip -y
```

```
# Распаковка проекта
unzip full_steaк.zip
```

Где full\_steaк.zip — название архива, который вы загрузили на сервер.

---

## Запуск Backend части

Backend нашего приложения состоит из FastAPI сервера и базы данных PostgreSQL.

### Установка зависимостей для backend:

bash

```
# Переходим в папку backend
cd backend

# Установка Python и системных зависимостей
sudo apt install python3 python3-pip python3-venv -y

# Установка и настройка PostgreSQL
sudo apt install postgresql postgresql-contrib -y
sudo systemctl start postgresql
sudo systemctl enable postgresql

# Создание и активация виртуального окружения
python3 -m venv myenv
source myenv/bin/activate

# Установка Python библиотек
pip install fastapi uvicorn psycopg2-binary sqlalchemy pandas numpy
pip install scikit-learn matplotlib seaborn tensorflow

# Запуск сервера
python3 run.py
```

---

## Запуск Frontend части

Frontend представляет собой React приложение, которое будет обслуживаться через Vite dev-сервер.

### Установка зависимостей для frontend:

bash

```
# Переходим в папку react
cd react

# Установка Node.js
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
```

```
sudo apt-get install -y nodejs

# Установка зависимостей проекта
npm install
npm install react react-dom

# Запуск development сервера
npm run dev
```

---

## Важные замечания по настройке

### Настройка брандмауэра:

Убедитесь, что на сервере открыты порты:

- 8000 — для FastAPI backend
- 5173 (или другой, который использует Vite) — для React frontend

### Настройка базы данных:

- Создайте базу данных и пользователя в PostgreSQL
- Обновите параметры подключения в config.py
- Запустите скрипт upload.py для наполнения базы данными

### Доменные имена:

Для production использования рекомендуется:

- Приобрести доменное имя
  - Настроить DNS записи для вашего сервера
  - Настроить Nginx как reverse proxy
  - Получить SSL сертификат (Let's Encrypt)
- 

## Проверка работоспособности

После запуска всех компонентов проверьте:

1. **Backend:** Откройте в браузере `http://X.XXX.XXX.XXX:8000/docs` — должна открыться документация Swagger
2. **Frontend:** Откройте `http://X.XXX.XXX.XXX:5173` — должно загрузиться React приложение
3. **API соединение:** Убедитесь, что фронтенд корректно подключается к бэкенду

---

## **Следующие шаги для production**

Для полноценного production-окружения рекомендуется:

- Настроить systemd службы для автоматического запуска
- Использовать Nginx + Gunicorn для FastAPI
- Настроить мониторинг и логирование
- Реализовать автоматическое развертывание (CI/CD)
- Настроить резервное копирование базы данных

## Занятие 11: Парсинг данных с веб-сайтов

[https://github.com/EugenePokh/full\\_steam\\_11\\_python\\_parsing.git](https://github.com/EugenePokh/full_steam_11_python_parsing.git)

В этом занятии мы освоим techniques для автоматического сбора данных с веб-сайтов — важный навык для наполнения вашего приложения актуальной информацией.

---

### Установка и настройка инструментов парсинга

#### Playwright — современный инструмент для автоматизации браузеров

Playwright позволяет эмулировать действия реального пользователя в браузере, что особенно полезно для работы с современными веб-приложениями.

##### Установка Playwright:

```
bash  
pip install playwright
```

##### Установка браузеров:

```
bash  
# Установка только Firefox  
playwright install firefox  
  
# Или установка всех поддерживаемых браузеров  
playwright install
```

Playwright поддерживает Chromium, Firefox и WebKit, что обеспечивает кросс-браузерную совместимость.

---

## Сравнение инструментов парсинга

### Playwright vs BeautifulSoup

#### Playwright:

- ✓ Эмулирует реальный браузер с поддержкой JavaScript
- ✓ Работает с динамическим контентом (React, Vue, Angular)
- ✓ Может выполнять сложные сценарии (клики, заполнение форм)
- ✓ Поддерживает скриншоты и видео записи
- ✗ Требует больше ресурсов
- ✗ Медленнее для простых задач

#### BeautifulSoup:

- ✓ Быстрый и легковесный
- ✓ Прост в использовании для статических сайтов
- ✓ Минимальные требования к ресурсам
- ✗ Не работает с JavaScript
- ✗ Не может взаимодействовать с элементами страницы

#### Выбор инструмента зависит от задачи:

- Используйте **BeautifulSoup** для простых статических сайтов
- Используйте **Playwright** для сложных JavaScript-приложений

---

#### Создание модуля парсинга: parsing.py

```
# Импорт необходимых библиотек
from playwright.sync_api import sync_playwright # Playwright для автоматизации
браузера
import logging # Модуль для логирования (записи ошибок и информации)

# Настройка логгера - конфигурируем систему логирования
# basicConfig() устанавливает базовые настройки логирования
# level=logging.INFO означает, что будут записываться сообщения уровня INFO и
выше (WARNING, ERROR, CRITICAL)
logging.basicConfig(level=logging.INFO)
# Создаем логгер с именем текущего модуля (файла)
```

```

# __name__ автоматически подставляет имя текущего файла
logger = logging.getLogger(__name__)

def parse_cbr_index() -> float:
    """
    Парсинг курса Доллара США с сайта ЦБ РФ

    Returns:
        float: Курс доллара США к рублю
        В случае ошибки возвращает значение по умолчанию 83.1725
    """
    try:
        # Контекстный менеджер sync_playwright() управляет жизненным циклом
        # Playwright
        # 'p' - объект Playwright, который дает доступ к браузерам
        with sync_playwright() as p:
            # Запускаем браузер Firefox в фоновом режиме (headless=True)
            # headless=True означает, что браузер работает без графического
            # интерфейса
            browser = p.firefox.launch(headless=True)

            # Создаем новую страницу (вкладку) в браузере
            page = browser.new_page()

            # Переходим на страницу с курсами валют ЦБ РФ
            page.goto("https://cbr.ru/currency_base/daily/")

            # Ждем загрузки таблицы с данными на странице
            # timeout=15000 означает, что ждем максимум 15 секунд
            # Если таблица не загрузится за 15 сек - будет исключение
            TimeoutError
            page.wait_for_selector("table.data", timeout=15000)

            # Ищем все строки (tr) в таблице с классом 'data'
            # query_selector_all возвращает список всех элементов,
            # соответствующих селектору
            rows = page.query_selector_all("table.data tr")

            # Перебираем все строки таблицы
            for row in rows:
                # В каждой строке ищем все ячейки (td)
                cells = row.query_selector_all("td")

                # Проверяем условия:
                # 1. В строке должно быть хотя бы 5 ячеек (чтобы избежать ошибок
                # индекса)
                # 2. В четвертой ячейке (индекс 3) должен быть текст "Доллар США"
                if len(cells) >= 5 and "Доллар США" in cells[3].inner_text():
                    # Получаем текст из пятой ячейки (индекс 4) - здесь находится
                    # курс

```

```

# Заменяем запятую на точку для корректного преобразования в
float
    value_text = cells[4].inner_text().replace(",", ".")  
  

# Закрываем браузер перед возвратом значения
browser.close()  
  

# Преобразуем текст в число с плавающей точкой и возвращаем
результат
    return float(value_text)  
  

# Если дошли до этой точки - значит, не нашли строку с долларом
# Закрываем браузер
browser.close()  
  

# Возвращаем значение по умолчанию
return 83.1725  
  

except Exception as e:
    # Обрабатываем любые исключения, которые могут возникнуть
    # Логируем ошибку с уровнем ERROR
    logger.error(f"Ошибка парсинга cbr_index: {e}")  
  

# Возвращаем значение по умолчанию при ошибке
return 83.1725  
  

# Добавьте этот блок для запуска
if __name__ == "__main__":
    """
Этот блок выполняется только при прямом запуске файла (не при импорте)
Он является точкой входа в программу
    """

# Выводим сообщение о начале работы
print("Запуск парсинга курса доллара...")  
  

# Вызываем нашу функцию парсинга и сохраняем результат в переменную rate
rate = parse_cbr_index()  
  

# Выводим полученный курс в удобочитаемом формате
print(f"Текущий курс доллара: {rate} руб.")  
  

# Ожидаем ввода пользователя - чтобы окно консоли не закрылось сразу
# Это полезно при запуске через двойной клик по файлу .py
input("Нажмите Enter для выхода...")

```

## **Практические рекомендации по парсингу**

### **Технические советы:**

- Используйте задержки между запросами
- Обрабатывайте исключения и таймауты
- Реализуйте механизмы повторных попыток
- Кэшируйте результаты для отладки

### **Оптимизация производительности:**

- Используйте асинхронные запросы для множества сайтов
  - Параллелизм обработку независимых данных
  - Мониторьте использование памяти при больших объемах данных
- 

## **Интеграция с основным приложением**

Парсинг можно интегрировать в ваше приложение несколькими способами:

1. **Периодический парсинг** — через cron-задачи или планировщик
  2. **По запросу пользователя** — через API эндпоинты
  3. **В реальном времени** — при определенных действиях пользователя
- 

## **Запуск парсера**

`python parsing.py`

---

**Итог занятия:** Вы освоили мощные инструменты для сбора веб-данных. Теперь вы можете автоматически наполнять ваше приложение актуальной информацией из различных источников, делая его еще более полезным для пользователей.

