Занятие 7: Создание API и дашбордов

https://github.com/EugenePokh/full_steak_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

```python
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_CONF
IG['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=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_steak_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
            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>
  );
};

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 гарфиков 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, actionName) => {
    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: () => '#F8BBD0', // 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
  },
  // Строка 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.