Государственное бюджетное профессиональное образовательное учреждение

«Нижегородский Губернский колледж»

**ОТЧЕТ**

**ПО ВЫПОЛНЕНИЮ ЗАДАНИЯ ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ**

ПМ.02 ОСУЩЕСТВЛЕНИЕ ИНТЕГРАЦИИ ПРОГРАММНЫХ МОДУЛЕЙ

|  |  |
| --- | --- |
| Руководитель\_\_\_\_\_\_\_\_\_Мамшева Ю.С. | 06.06.2025 |
| Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_Руфкин М.Н. | 06.06.2025 |
| Специальность, группа: 09.02.07, 33П |  |

Нижний Новгород

2025

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 3](#_Toc202099121)

[**ТЕОРЕТИЧЕСКАЯ ЧАСТЬ** 5](#_Toc202099122)

[**1.1 Организационная структура предприятия** 5](#_Toc202099123)

[**1.2 Анализ программного обеспечения и технического оснащения** 6](#_Toc202099124)

[**1.3 Описание инструментальных средств разработки и сопровождения программных средств** 8](#_Toc202099125)

[**ПРАКТИЧЕСКАЯ ЧАСТЬ** 9](#_Toc202099126)

[**2.1 Анализ предметной области** 9](#_Toc202099127)

[**2.2 Оформление технического задания** 11](#_Toc202099128)

[**2.3 Проектирование программного продукта** 13](#_Toc202099129)

[**2.4 Разработка программного модуля** 20](#_Toc202099130)

[**2.5 Тестирование программного продукта** 38](#_Toc202099131)

[**2.6 Разработка модуля математического моделирования** 52](#_Toc202099132)

[**ЗАКЛЮЧЕНИЕ** 55](#_Toc202099133)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 56](#_Toc202099134)

[**ПРИЛОЖЕНИЯ** 57](#_Toc202099135)

[**ПРИЛОЖЕНИЕ А** 57](#_Toc202099136)

[**ПРИЛОЖЕНИЕ Б** 58](#_Toc202099137)

[**ПРИЛОЖЕНИЕ В** 66](#_Toc202099138)

# **ВВЕДЕНИЕ**

В условиях цифровой трансформации страхового рынка, а также расширении компании, количество необходимых сотрудников для выполнения обязанностей возрастает. И одним из актуальных и удобных методов получения квалифицированного сотрудника является стажерская программа. В связи с этим можно выделить несколько основных моментов для компании СПАО «Ингосстрах»:

* Рост численности стажёров.
* Неэффективность текущих методов обучения (разрозненные Excel-отчёты, отсутствие единой базы знаний).
* Требования регуляторов.

Целью данного проекта является разработка и внедрение системы Intern Guide – цифровой платформы для адаптации, обучения и мониторинга стажёров компании СПАО «Ингосстрах». Система преследует три ключевые цели:

1. Оптимизация адаптации сокращение периода ввода в должность с 3 до 2 месяцев
2. Стандартизация обучения создание единой базы знаний по:

* Страховым продуктам (ОСАГО, КАСКО, ДМС)
* Корпоративным стандартам
* CRM-системам (1С:Страхование)

1. Автоматизация контроля переход от Excel-отчетов к цифровому мониторингу:

* Прогресса обучения
* Результатов тестирования
* Обратной связи от кураторов

**Объект исследования**: Процесс адаптации стажёров в ПАО «Ингосстрах».

**Предмет исследования**: Автоматизированная система управления обучением (LMS).

Для реализации программного продукта и выполнения поставленной задачи по разработке информационной системы, необходимо следовать следующим этапам решения:

1. Анализ требований:

* Опрос сотрудников компании

1. Проектирование:

* Прототипирование UI в Figma
* ER-диаграммы в Drowio
* UML-диаграммы

1. Реализация:

* Разработка на Python, JS, HTML.

# **ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

## **1.1 Организационная структура предприятия**

«Ингосстрах» – крупнейшая страховая компания в России, основанная в 1947 году. Основная задача данной компании – обеспечить своим клиентам финансовую защиту и душевное спокойствие.

Организационная структура предприятия строится следующим образом (Приложение А):

* Центральный офис: Москва, отвечает за общее управление дочерними предприятиями и офисами, а также занимается принятием стратегических решений в области осуществления каких-либо действий во внутреннем строе компании и внешних взаимодействий с другими компаниями.
* Региональные дирекции: в крупных городах находятся 8 региональных представительств, которые отвечают за развитие бизнеса и своевременное обслуживание клиентов, и принятие необходимых решений во всех регионах России.
* Филиалы и представительства: компания имеет более 80 различных филиалов и представительств по всей стране.

Компания «Ингосстрах» осуществляет свою деятельность в различных направлениях. Среди них можно выделить следующие основные бизнес-процессы, которые необходимы для выполнения грамотного обслуживания клиентов и полного удовлетворения их потребностям:

* Андеррайтинг: определение и оценка потенциальных и фактических рисков, на основе которых определяются условия страхования.
* Заключения договоров: оформление страховых полисов на различные виды имущества и личное здоровье.
* Урегулирование убытков: анализ и обработка страховых случаев, и выплата материальных компенсаций.
* Инвестиционная деятельность: управление страховыми резервами и инвестирование средств.
* Клиентское обслуживание: обслуживание клиентов, консультирование по вопросам касающихся области деятельности предприятия, а также решение их проблем.

«Ингосстрах» – крупная и динамично развивающаяся страховая компания с многоуровневой организационной структурой и сложными бизнес-процессами. Для эффективного функционирования компании необходимы надежные и эффективные информационные системы, в том числе базы данных.

## **1.2 Анализ программного обеспечения и технического оснащения**

В современном мире невозможно представить работу какой-либо компании без использования программного обеспечения. Оно нужно для удобного и автоматизированного использования информации и ресурсов. Компания «Ингосстрах», занимающаяся страховой деятельностью, также использует большое количество разнообразного ПО. Вот примеры некоторых из тех, что используются сотрудниками:

* Автоматизированная информационная система (АИС) — это совокупность устройств и программных средств, объединённых информационными потоками и предназначенных для сбора, хранения, обработки и выдачи информации с целью автоматизации различных процессов. Данная программа помогает управлять ресурсами компании, анализировать данные о продажах и клиентах, планировать бюджет и оптимизировать бизнес-процессы. Также она используется для обработки платежей, управления счетами, выдачи кредитов, оценки рисков и многого другого.
* Microsoft Outlook — компьютерная программа-органайзер с функциями почтового клиента компании «Майкрософт», входящая в пакет офисных программ Microsoft Office.  Используется для работы с электронными письмами и составления списков задач и контактов.

Если рассматривать используемое ПО с точки зрения хранения и использования данных, то используемое программное обеспечение представляет собой следующий список:

1. **Программное обеспечение**

Операционная система: Windows

СУБД: Oracle Database (Oracle Database Enterprise Edition)

Средства разработки баз данных: Oracle SQL Developer

Инструменты администрирования баз данных: Oracle Enterprise Manager

Системы резервного копирования и восстановления данных: Oracle Recovery Manager (RMAN), Veritas NetBackup

Системы управления полисами и договорами: собственная разработка на базе Oracle Database

Система урегулирования убытков: собственная разработка на базе Oracle Database

Система управления рисками: собственная разработка на базе Oracle Database

Система финансового учета: SAP ERP

Система клиентского обслуживания (CRM): Salesforce

1. **Техническое оснащение**

Серверы: Dell PowerEdge, HPE ProLiant

Хранилища данных: Dell EMC VMAX, HPE 3PAR

Системы бесперебойного питания: APC, Eaton

Системы кондиционирования и охлаждения: Liebert, Stulz

Компания «Ингосстрах» использует надежное и современное программное обеспечение, и техническое оснащение для обеспечения стабильной и эффективной работы своих информационных систем.

## **1.3 Описание инструментальных средств разработки и сопровождения программных средств**

Для создания системы Intern Guide выбран современный технологический стек, ориентированный на высокую производительность, масштабируемость и интеграцию с существующей ИТ-инфраструктурой СПАО «Ингосстрах».

1. **Backend**

Backend-разработка реализуется на Python с использованием следующих ключевых компонентов:

* Язык программирования: Python 3.11
* Фреймворк: FastAPI – обеспечивает высокую скорость обработки запросов и автоматическую генерацию OpenAPI-документации
* База данных: PostgreSQL 15 с расширением TimescaleDB для хранения временных рядов

1. **Frontend**

Frontend-разработка строится на классическом веб-стеке:

* JavaScript: ES6+ с использованием модульного подхода
* HTML5: Семантическая верстка
* CSS3: методология BEM для стилей
* Фреймворк: React 18

Для обеспечения качества используются несколько уровней тестирования:

1. Модульные тесты:

* Python: pytest с покрытием >85%

1. Интеграционные тесты:

* Тестирование API через Postman (коллекции в Git)

1. Нагрузочное тестирование:

* Проверка отказоустойчивости

# **ПРАКТИЧЕСКАЯ ЧАСТЬ**

## **2.1 Анализ предметной области**

На начальной стадии создания платформы для адаптации стажёров был проведен глубокий анализ предметной области. Эта работа стала отправной точкой для всего проекта, позволив сформировать четкое понимание того, какой должна быть система.

Основные этапы работы:

1. **Изучение потребностей пользователей**

Первым делом был проведен тщательный анализ того, какие задачи должна решать система. Для этого:

* Провели опрос среди работников и руководства
* Опросили стажёров прошлых наборов
* Изучили существующие процессы адаптации в компании

В результате были определены ключевые функции:

* Централизованное хранение учебных материалов
* Система проверки знаний
* Мониторинг прогресса обучения

1. **Определение структуры данных**

На этом этапе были выявлены основные сущности системы и связи между ними:

* Профили пользователей (стажёры, кураторы, администраторы)
* Учебные курсы и программы
* Тестовые задания и результаты
* Отчеты и аналитика

Для каждой сущности прописали важные атрибуты и правила взаимодействия.

1. **Создание концептуальной модели**

Заключительный этап включал:

* Визуализацию структуры системы
* Формализацию взаимосвязей между компонентами
* Разработку принципов организации данных

Проделанная работа позволила:

1. Чётко очертить границы системы
2. Выделить все значимые элементы и их характеристики
3. Установить логические связи между компонентами
4. Сформировать единое видение системы
5. Создать основу для последующей технической реализации

Полученная концептуальная модель стала надёжным фундаментом для дальнейшей разработки. Она помогает избежать многих проблем на этапе программирования, так как все ключевые решения уже продуманы и согласованы. Имея на руках результаты этого анализа, мы можем переходить к детальному проектированию архитектуры системы, разработке интерфейсов и планированию этапов реализации.

## **2.2 Оформление технического задания**

Разработка технического задания для платформы адаптации стажёров Intern Guide потребовала комплексного подхода, учитывающего специфику корпоративного обучения в страховой отрасли. Этот документ стал ключевым ориентиром для разработки, обеспечив единое видение системы и её функциональных возможностей.

Формирование технического задания началось с всестороннего исследования потребностей СПАО «Ингосстрах». Был проведён:

1. Анализ существующих процессов адаптации новых сотрудников в компании
2. Изучение современных LMS-систем
3. Опрос сотрудников

Это позволило выделить критические требования к системе:

* Интуитивно понятный интерфейс для пользователей с разным уровнем технической подготовки
* Гибкая система управления учебными материалами
* Инструменты автоматизированной проверки знаний
* Подробная аналитика успеваемости

Техническое задание для Intern Guide включает несколько взаимосвязанных разделов:

1. Бизнес-требования

* Описание целей внедрения системы
* Ключевые показатели эффективности
* Ожидаемое влияние на бизнес-процессы

1. Функциональные требования:

* Детализированные сценарии использования
* Описание ролей и их прав
* Логика работы основных модулей

1. Технические аспекты:

* Архитектурные решения
* Спецификации интеграции с корпоративными системами

1. Критерии приемки:

* Методики тестирования
* Показатели производительности

При создании технического задания особый акцент был сделан на гибкость системы, её масштабируемость, безопасность данных пользователей и аппаратных средств.

Итоговый документ представляет собой:

1. Руководство для разработчиков с четкими техническими спецификациями
2. Инструмент контроля для заказчика с измеримыми критериями приемки
3. Дорожную карту для будущего развития системы

Техническое задание для Intern Guide отличается практической ориентированностью – все требования основаны на реальных бизнес-процессах компании и подкреплены данными исследований.

Этот документ стал надежной основой для создания системы, которая реально решает проблемы адаптации новых сотрудников в крупной страховой компании.

## **2.3 Проектирование программного продукта**

UML (Unified Modeling Language) — это стандартизированный язык графического моделирования, используемый в объектно-ориентированном анализе и проектировании. UML-диаграммы помогают визуализировать, проектировать и документировать структуру и поведение программных систем.

Существует 14 типов UML-диаграмм, которые делятся на две группы:

1. **Структурные диаграммы** (отображают статическую структуру системы).
2. **Поведенческие диаграммы** (описывают динамику работы системы).

Самыми основными из них выступают 9 диаграмм: классов, кооперации, деятельности, компонентов, последовательности, потоков данных, развертывания, состояния, вариантов использования.

1. **Диаграмма классов**

Тип: структурная

Назначение: отображает статическую структуру системы в виде классов, их атрибутов, методов и взаимосвязей.

Основные элементы:

* Класс: Прямоугольник с тремя секциями:
* Имя класса
* Атрибуты
* Методы
* Связи: наследование, ассоциации, агрегация/композиция.

1. **Диаграмма последовательности**

Тип: Поведенческая

Назначение: показывает взаимодействие объектов в хронологическом порядке (как сообщения передаются между ними).

Основные элементы:

* Объекты – участники взаимодействия
* Линии жизни – вертикальные пунктирные линии, показывающие время жизни объекта
* Сообщения – стрелки между линиями жизни
* Активационные полосы – прямоугольники на линиях жизни, показывающие активность объекта

1. **Диаграмма развертывания**

Тип: Структурная

Назначение: показывает физическое размещение компонентов системы на серверах, узлах сети и устройствах.

Основные элементы:

* Узлы – физические устройства
* Артефакты – развертываемые компоненты
* Связи – показывают коммуникацию между узлами

1. **Диаграмма состояний**

Тип: Поведенческая

Назначение: описывает жизненный цикл объекта, показывая, как он переходит между состояниями под воздействием событий.

Основные элементы:

* Состояние – прямоугольник с закругленными углами
* Переход – стрелка с событием
* Начальное и конечное состояния – черные круги

1. **Диаграмма вариантов использования**

Тип: Поведенческая

Назначение: описывает взаимодействие между пользователями (акторами) и системой через сценарии использования.

Основные элементы:

* Актор – роль, взаимодействующая с системой
* Вариант использования – овал с названием действия
* Связи

1. **Диаграмма кооперации**

Тип: Поведенческая

Назначение: показывает взаимодействие объектов для выполнения конкретного сценария через обмен сообщениями.

Основные элементы:

* Объекты – прямоугольники с подчеркнутыми именами
* Сообщения – стрелки с номерами

1. **Диаграмма деятельности**

Тип: Поведенческая

Назначение: описывает бизнес-процессы и алгоритмы в виде последовательности действий.

Основные элементы:

* Действия – скругленные прямоугольники
* Ветвления – ромбы
* Потоки управления – стрелки

1. **Диаграмма компонентов**

Тип: Структурная

Назначение: отображает физические компоненты системы (модули, библиотеки) и их зависимости.

Основные элементы:

* Компоненты – прямоугольники с иконкой
* Интерфейсы – «лоллипопы»
* Зависимости – пунктирные стрелки

1. **Диаграмма потоков данных**

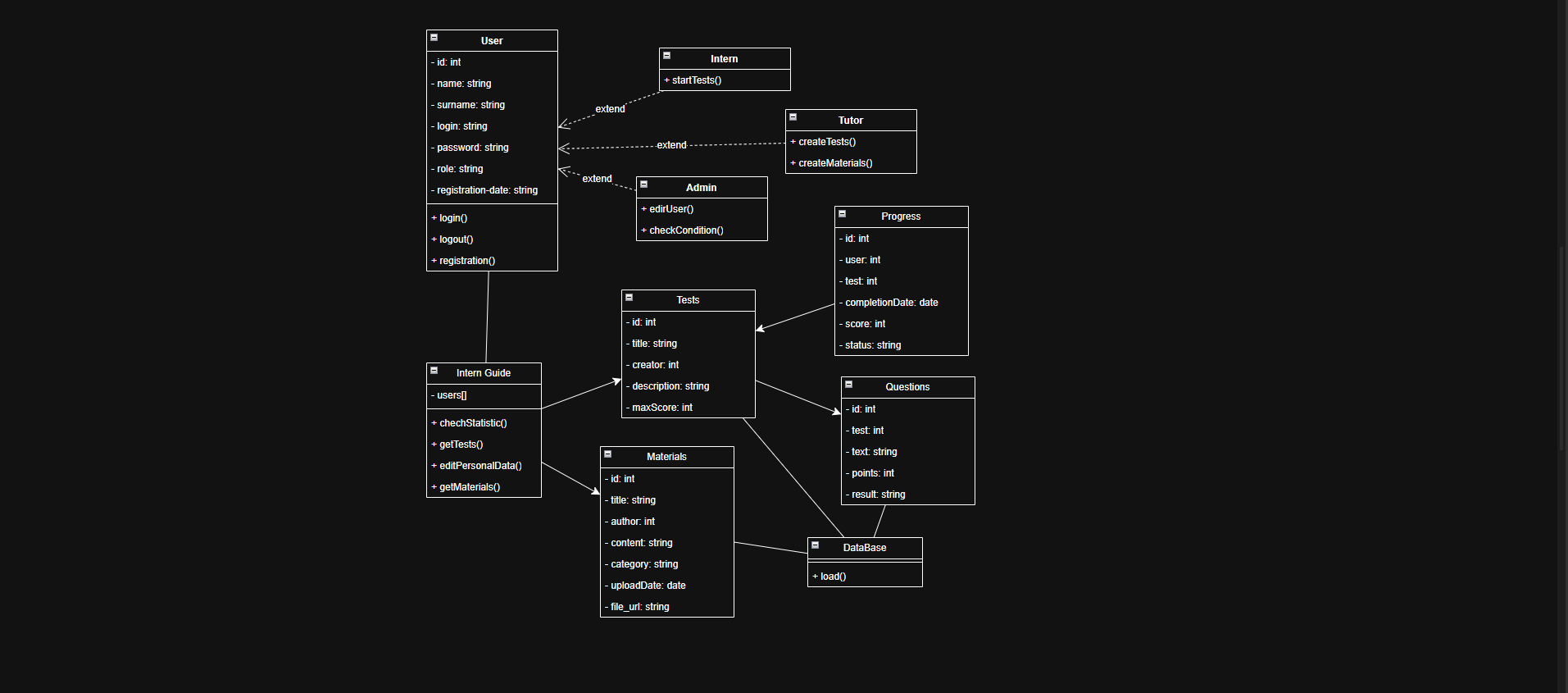
Тип: Структурная

Назначение: показывает движение информации между процессами, хранилищами и внешними сущностями.

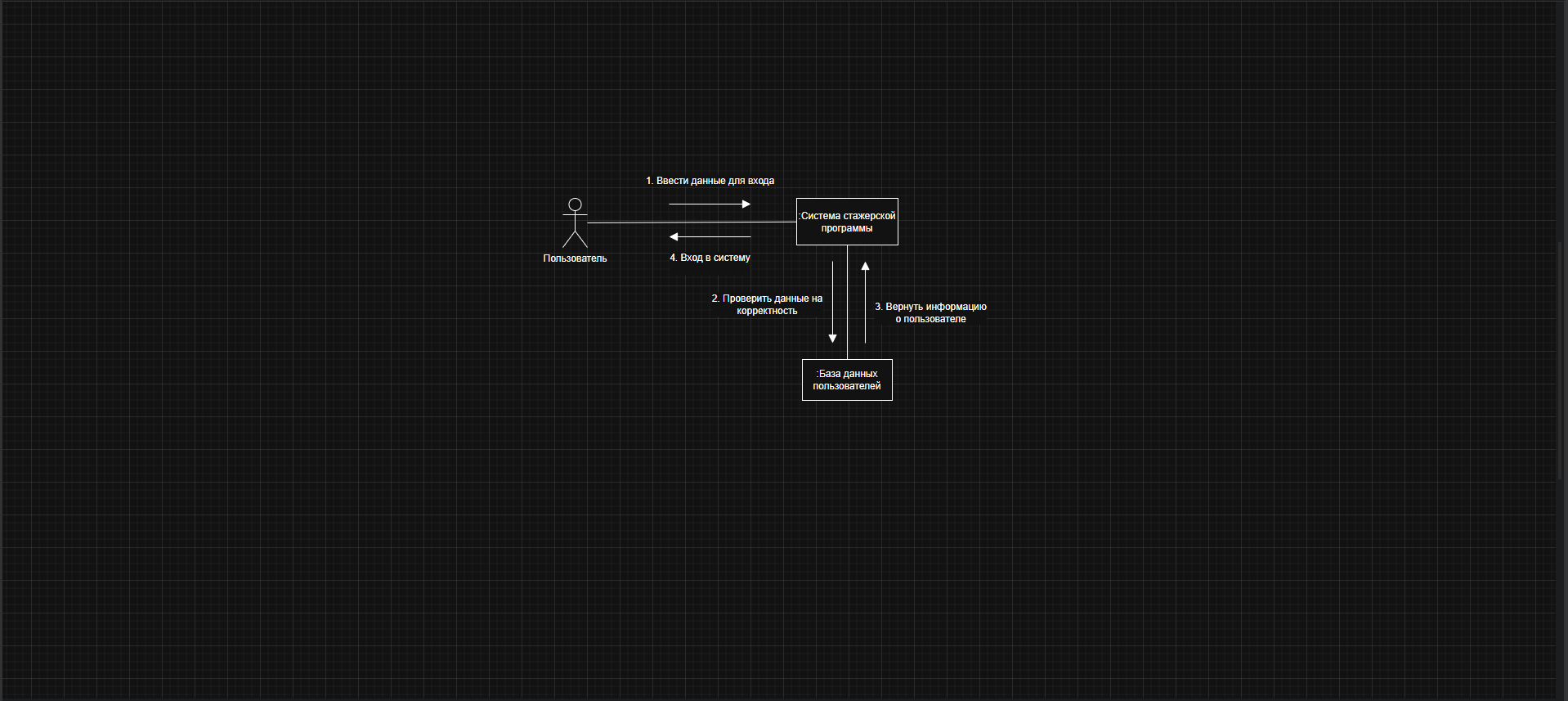
Основные элементы:

* Процессы – круги
* Хранилища – две параллельные линии
* Потоки данных – стрелки с подписями

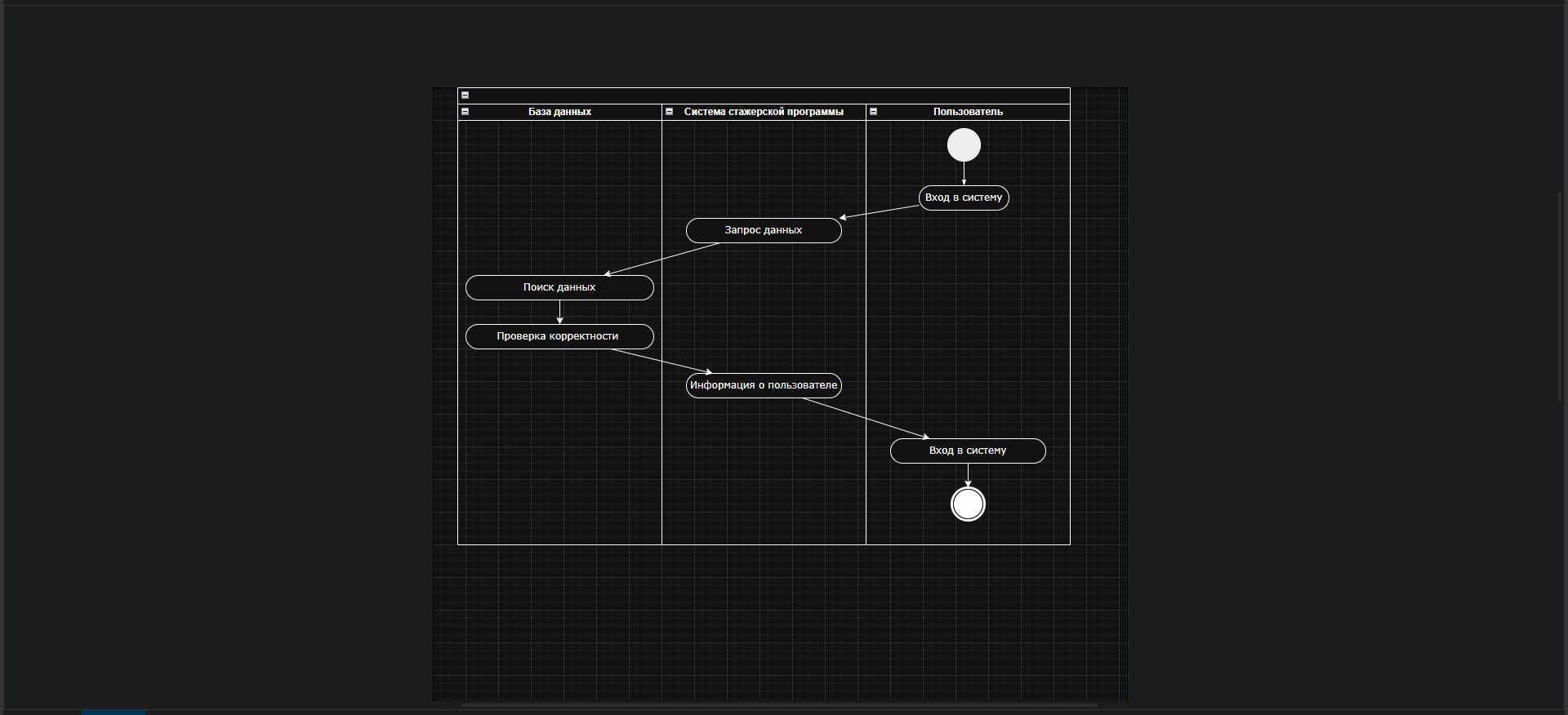
Для описания поведения и работы информационной системы Intern Guide, которая помогает стажерам лучше осваиваться на новом месте и получать различные советы, были разработаны следующие несколько диаграмм:

****

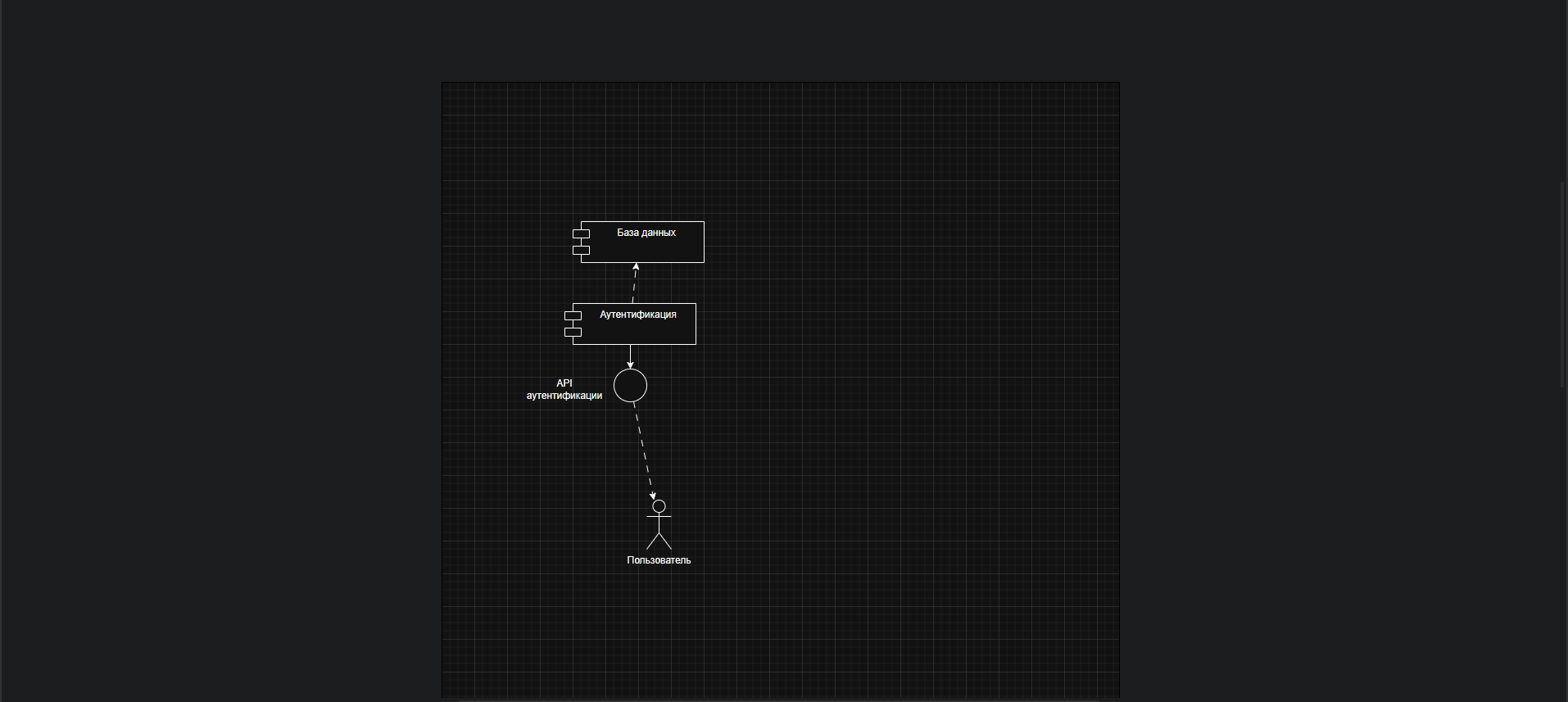
**Рисунок 1. Авторизация - диаграмма классов**

****

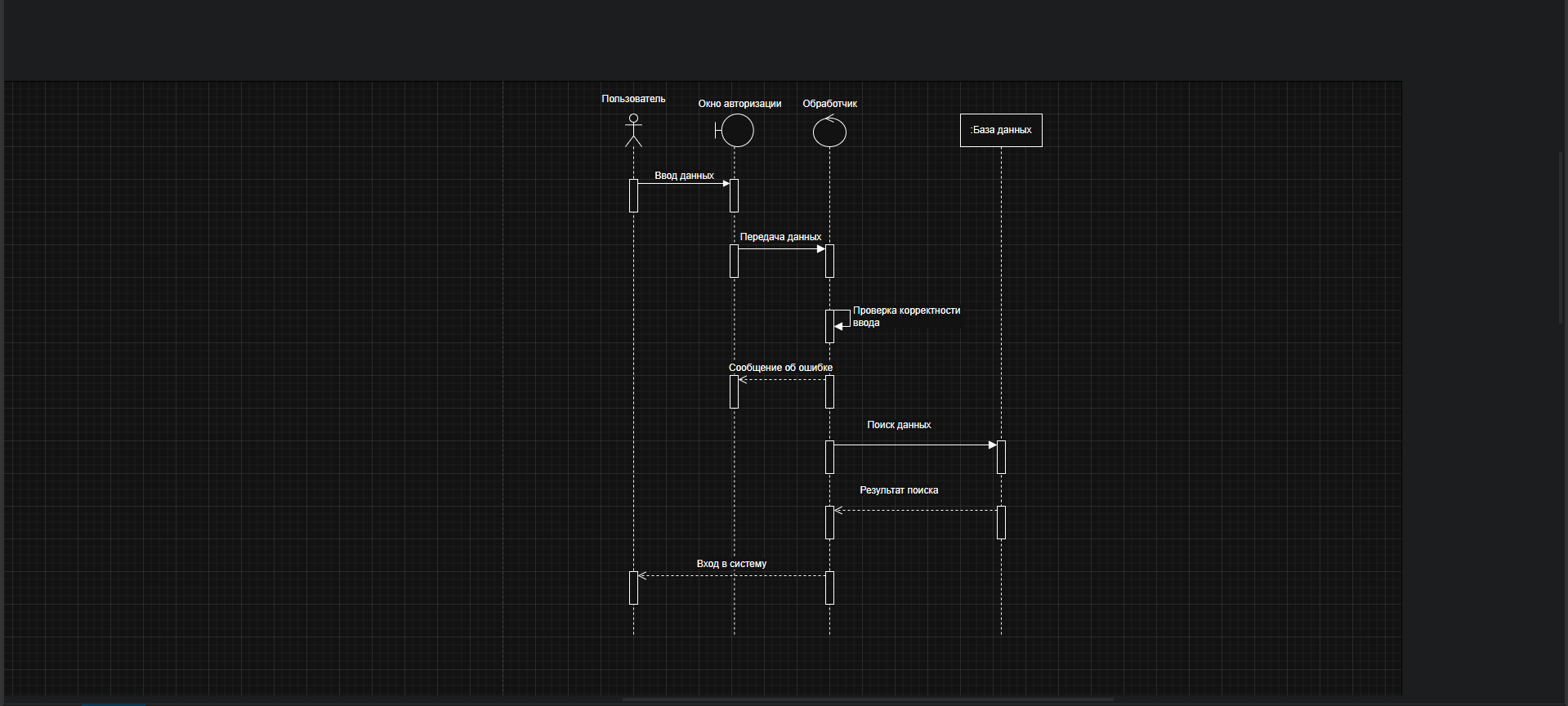
**Рисунок 2. Авторизация - диаграмма коопераций**

****

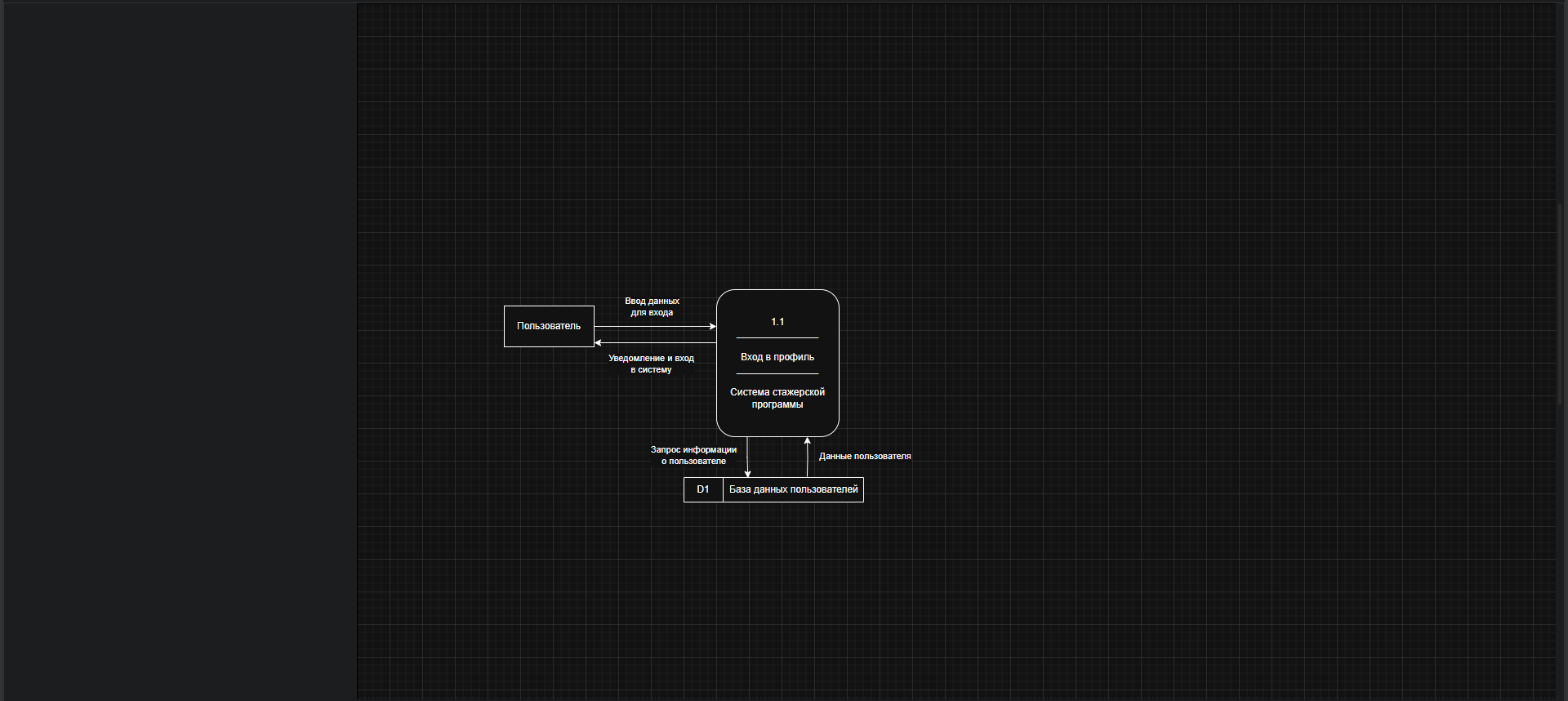
**Рисунок 3. Авторизация - диаграмма деятельности**

****

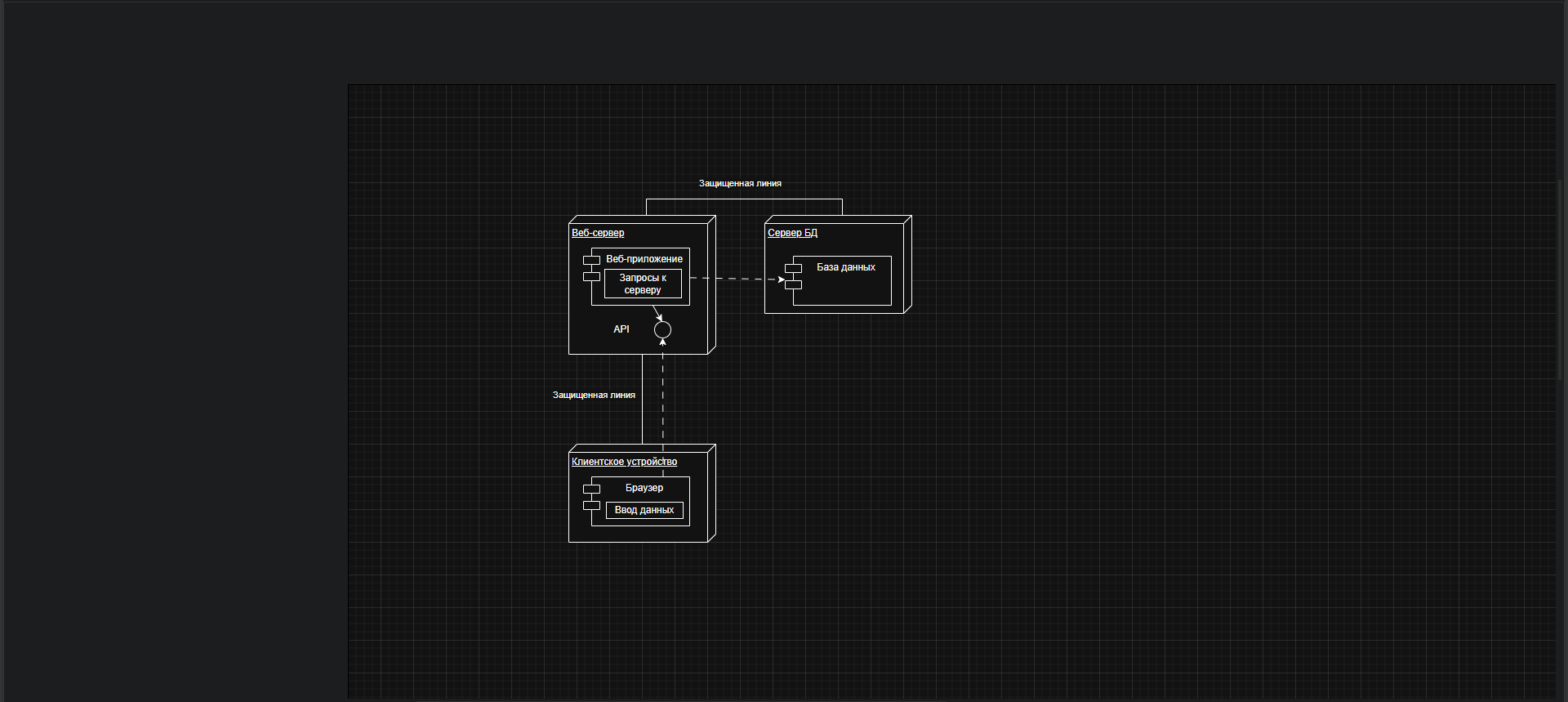
**Рисунок 4. Авторизация - диаграмма компонентов**

****

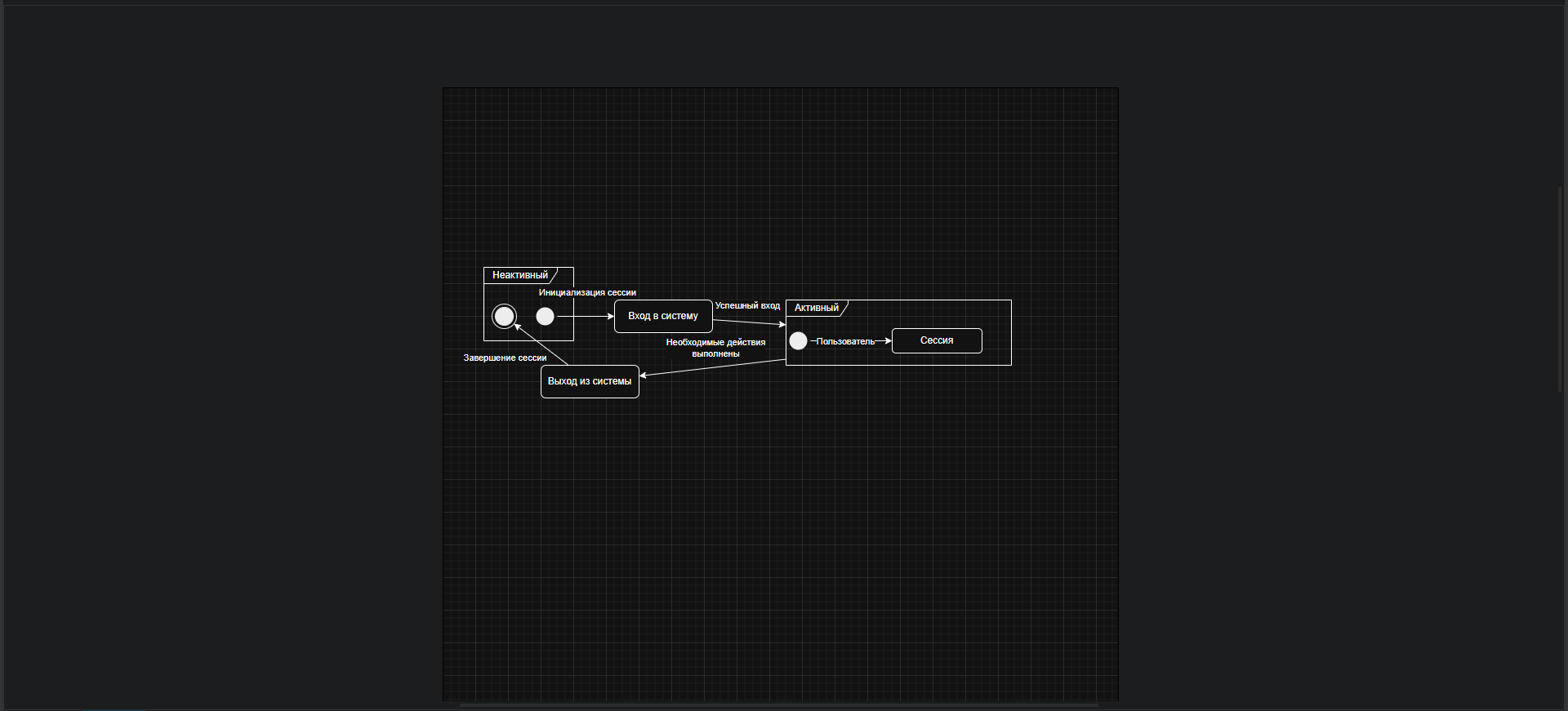
**Рисунок 5. Авторизация - диаграмма последовательности**

****

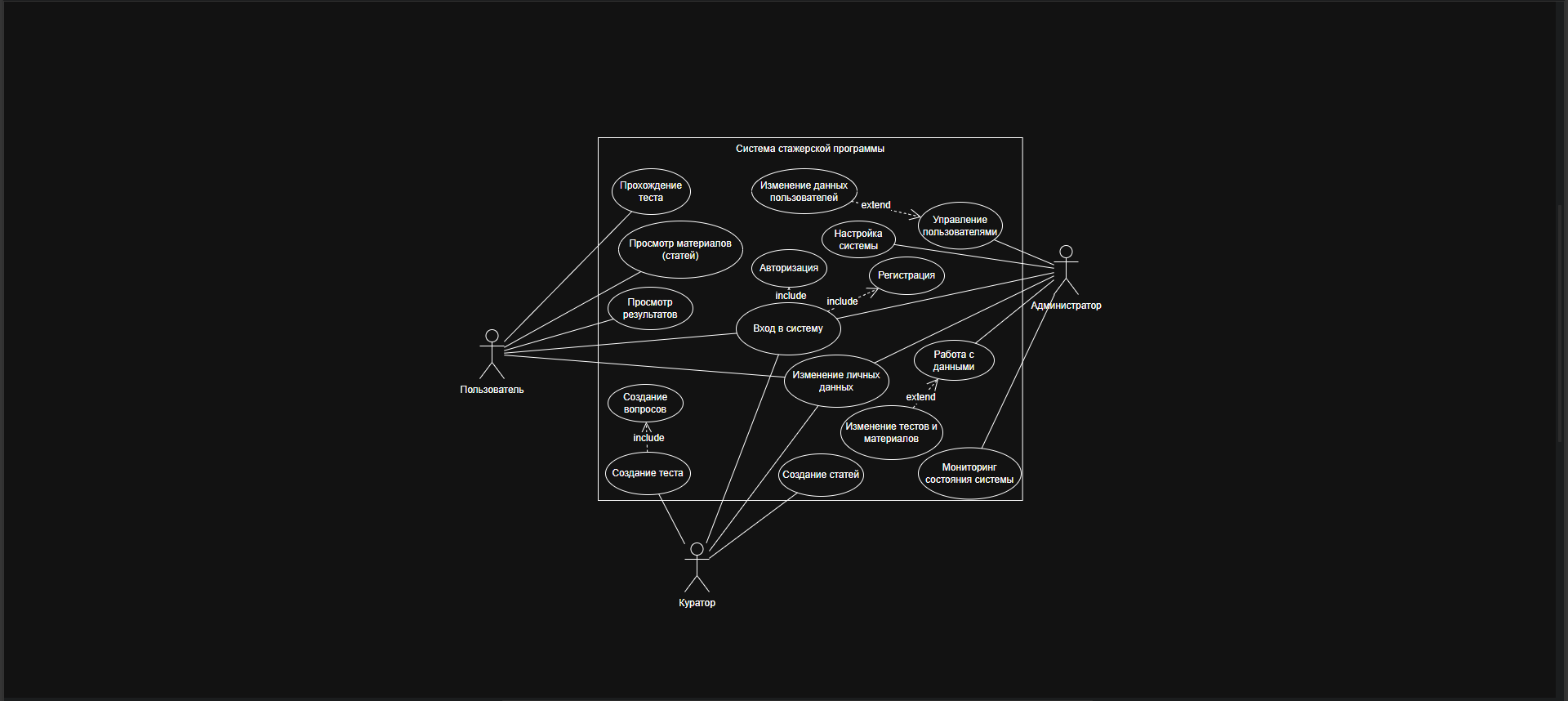
**Рисунок 6. Авторизация - диаграмма потоков данных**

****

**Рисунок 7. Авторизация - диаграмма развертывания**

****

**Рисунок 8. Авторизация - диаграмма состояния**

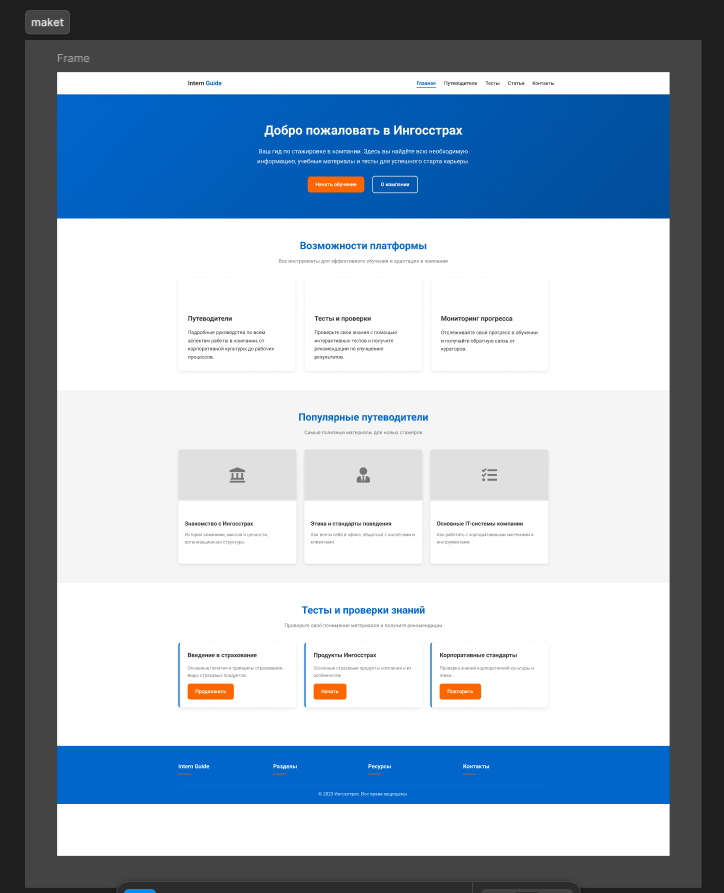
****

**Рисунок 9. Авторизация - диаграмма вариантов использования**

Дополнительные диаграммы для остальных модулей разрабатываемой системы находятся в Приложении Б.

Данный набор диаграмм обеспечивает полное визуальное представление системы на всех уровнях — от логики работы до физической инфраструктуры.

Для визуального проектирования пользовательского интерфейса сайта был разработан макет. Данный макет содержит основную информацию, которая будет представлена пользователю и будет служить опорой для реализации продукта.

****

**Рисунок 10. Макет сайта Intern Guide**

## **2.4 Разработка программного модуля**

Проект построен по модульной микросервисной архитектуре, где каждый компонент отвечает за определенную бизнес-функцию:

* **Презентационный слой:** React.js + TypeScript
* **Бизнес-логика:** Flask (Python)
* **Слой данных:** PostgreSQL

**Основные микросервисы:**

1. **Авторизация и управление пользователями (auth)**

* Регистрация, вход, ролевая модель (стажер, куратор, администратор).
* JWT-аутентификация

1. **Обучение и статьи (materials)**

* Хранение и отображение учебных материалов.
* Поиск, фильтрация и рекомендации статей.

1. **Тестирование (tests)**

* Прохождение тестов, автоматическая проверка ответов.

1. **Аналитика и отчеты (progress, users)**

* Отслеживание прогресса стажеров.
* Визуализация данных (графики, статистика).

**Технологический стек**

**Backend:**

* **Flask** — выбран за высокую производительность и асинхронность.
* **SQLAlchemy** — ORM для работы с базой данных.

**Frontend:**

* **React 18 + TypeScript** — для типобезопасности и удобства разработки.
* **CSS + HTML** — адаптивный дизайн и анимации.

**Процесс разработки**

Работа велась по гибридной методологии:

1. **Планирование**
   * Декомпозиция задач на микросервисы.
   * Проектирование API
2. **Разработка**
   * Парное программирование для критических модулей (например, система проверки тестов).
   * Code Review
3. **Тестирование**
   * **Unit-тесты:** pytest (бэкенд), ручное тестирование (frontend)
   * **Интеграционные тесты:** Postman, Cypress (E2E).
4. **Деплой**
   * Стадии: Dev → Staging → Production.

В конечном результате были реализованы следующие возможности и функциональные модули:

**Для стажеров:**

* Чтение статей с возможностью сохранения в закладки.
* Прохождение тестов с автоматической проверкой и обратной связью.
* Личный кабинет с отслеживанием прогресса.

**Для кураторов:**

* Создание и редактирование учебных материалов.
* Составление тестов (вопросы с вариантами ответов, открытые задания).
* Анализ результатов группы.

**Для администраторов:**

* Управление пользователями (роли, доступы).
* Настройка платформы (категории статей, система уведомлений).

Ниже представлен программный код программный модулей разработанной информационной системы, который реализует вышеперечисленный функционал для пользователей:

**App.py (модуль запуска серверной части программы)**

from backend import create\_app, db

from backend.routes.auth import auth\_bp

from backend.routes.materials import materials\_bp

from backend.routes.progress import progress\_bp

from backend.routes.tests import tests\_bp

from backend.routes.users import users\_bp

from flask\_cors import CORS

app = create\_app()

CORS(app, resources={

    r"/api/\*": {

        "origins": ["http://127.0.0.1:5500", "http://localhost:5500", "http://127.0.0.1:5000", "http://localhost:5000"],

        "methods": ["GET", "POST", "PUT", "DELETE", "OPTIONS"],

        "allow\_headers": ["Content-Type", "Authorization"],

        "supports\_credentials": True

    }

})

# Регистрация blueprint'ов

app.register\_blueprint(auth\_bp, url\_prefix='/api')

app.register\_blueprint(materials\_bp, url\_prefix='/api')

app.register\_blueprint(progress\_bp, url\_prefix='/api')

app.register\_blueprint(tests\_bp, url\_prefix='/api')

app.register\_blueprint(users\_bp, url\_prefix='/api')

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**config.py (модуль конфигурации программы)**

import os

from pathlib import Path

from dotenv import load\_dotenv

load\_dotenv()

BASE\_DIR = Path(\_\_file\_\_).parent.parent

DB\_PATH = BASE\_DIR / 'database' / 'intern\_guide.db'

class Config:

    SECRET\_KEY = os.getenv('SECRET\_KEY', 'ingosstrakh-intern-guide-secret-key-2023')

    SQLALCHEMY\_DATABASE\_URI = f'sqlite:///{DB\_PATH}'

    SQLALCHEMY\_TRACK\_MODIFICATIONS = False

    DEBUG = os.getenv('DEBUG', 'True') == 'True'

**Models.py (модели сущностей базы данных)**

from backend import db

from datetime import datetime

class User(db.Model):

    \_\_tablename\_\_ = 'users'

    user\_id = db.Column(db.Integer, primary\_key=True)

    name = db.Column(db.String(50), nullable=False)

    surname = db.Column(db.String(50), nullable=False)

    login = db.Column(db.String(50), unique=True, nullable=False)

    password = db.Column(db.String(200), nullable=False)

    role = db.Column(db.String(20), nullable=False, default='trainee')

    registration\_date = db.Column(db.DateTime, default=datetime.utcnow)

    materials = db.relationship('Material', backref='author', lazy=True)

    tests = db.relationship('Test', backref='creator', lazy=True)

    progress = db.relationship('Progress', backref='user', lazy=True)

class Material(db.Model):

    \_\_tablename\_\_ = 'materials'

    material\_id = db.Column(db.Integer, primary\_key=True)

    title = db.Column(db.String(100), nullable=False)

    author\_id = db.Column(db.Integer, db.ForeignKey('users.user\_id'), nullable=False)

    content = db.Column(db.Text)

    category = db.Column(db.String(50))

    upload\_date = db.Column(db.DateTime, default=datetime.utcnow)

    file\_url = db.Column(db.String(200))

    progress = db.relationship('Progress', backref='material', lazy=True)

class Test(db.Model):

    \_\_tablename\_\_ = 'tests'

    test\_id = db.Column(db.Integer, primary\_key=True)

    title = db.Column(db.String(100), nullable=False)

    creator\_id = db.Column(db.Integer, db.ForeignKey('users.user\_id'), nullable=False)

    description = db.Column(db.Text)

    max\_score = db.Column(db.Integer, nullable=False)

    creation\_date = db.Column(db.DateTime, default=datetime.utcnow)

    questions = db.relationship('Question', backref='test', lazy=True, cascade='all, delete-orphan')

    progress = db.relationship('Progress', backref='test', lazy=True)

class Question(db.Model):

    \_\_tablename\_\_ = 'questions'

    question\_id = db.Column(db.Integer, primary\_key=True)

    test\_id = db.Column(db.Integer, db.ForeignKey('tests.test\_id'), nullable=False)

    text = db.Column(db.String(500), nullable=False)

    points = db.Column(db.Integer, nullable=False, default=1)

    option\_a = db.Column(db.String(200), nullable=False)

    option\_b = db.Column(db.String(200), nullable=False)

    option\_c = db.Column(db.String(200))

    option\_d = db.Column(db.String(200))

    correct\_option = db.Column(db.String(1), nullable=False)

class Progress(db.Model):

    \_\_tablename\_\_ = 'progress'

    progress\_id = db.Column(db.Integer, primary\_key=True)

    user\_id = db.Column(db.Integer, db.ForeignKey('users.user\_id'), nullable=False)

    test\_id = db.Column(db.Integer, db.ForeignKey('tests.test\_id'))

    material\_id = db.Column(db.Integer, db.ForeignKey('materials.material\_id'))

    completion\_date = db.Column(db.DateTime, default=datetime.utcnow)

    score = db.Column(db.Integer)

    status = db.Column(db.String(20), nullable=False, default='started')

**utils.py (модуль необходимых общих утилит)**

from functools import wraps

from flask import jsonify, request

import jwt

from backend.config import Config

from backend.models import User

def token\_required(f):

    @wraps(f)

    def decorated(\*args, \*\*kwargs):

        token = None

        if 'Authorization' in request.headers:

            auth\_header = request.headers['Authorization']

            if auth\_header.startswith('Bearer '):

                token = auth\_header.split(" ")[1]

        if not token:

            return jsonify({'message': 'Token is missing!'}), 401

        try:

            data = jwt.decode(token, Config.SECRET\_KEY, algorithms=["HS256"])

            current\_user = User.query.get(data['user\_id'])

            if not current\_user:

                raise ValueError('User not found')

        except Exception as e:

            return jsonify({'message': f'Token is invalid: {str(e)}'}), 401

        return f(current\_user, \*args, \*\*kwargs)

    return decorated

def admin\_required(f):

    @wraps(f)

    def decorated(current\_user, \*args, \*\*kwargs):

        if current\_user.role != 'admin':

            return jsonify({'message': 'Admin access required!'}), 403

        return f(current\_user, \*args, \*\*kwargs)

    return decorated

def curator\_required(f):

    @wraps(f)

    def decorated(current\_user, \*args, \*\*kwargs):

        if current\_user.role not in ['curator', 'admin']:

            return jsonify({'message': 'Curator access required!'}), 403

        return f(current\_user, \*args, \*\*kwargs)

    return decorated

def generate\_password\_hash(password):

    from werkzeug.security import generate\_password\_hash as \_generate\_password\_hash

    return \_generate\_password\_hash(password)

def check\_password\_hash(hashed\_password, password):

    from werkzeug.security import check\_password\_hash as \_check\_password\_hash

    return \_check\_password\_hash(hashed\_password, password)

**progress.py (API модуль для материалов)**

from flask import Blueprint, request, jsonify

from backend.models import Progress, Material, Test, db

from backend.utils import token\_required

progress\_bp = Blueprint('progress', \_\_name\_\_)

@progress\_bp.route('/progress', methods=['GET'])

@token\_required

def get\_user\_progress(current\_user):

    progress\_records = Progress.query.filter\_by(user\_id=current\_user.user\_id).all()

    output = []

    for record in progress\_records:

        progress\_data = {

            'progress\_id': record.progress\_id,

            'test\_id': record.test\_id,

            'material\_id': record.material\_id,

            'completion\_date': record.completion\_date.strftime('%Y-%m-%d'),

            'score': record.score,

            'status': record.status

        }

        if record.material\_id:

            material = Material.query.get(record.material\_id)

            progress\_data['title'] = material.title

            progress\_data['type'] = 'material'

        elif record.test\_id:

            test = Test.query.get(record.test\_id)

            progress\_data['title'] = test.title

            progress\_data['type'] = 'test'

        output.append(progress\_data)

    return jsonify({'progress': output}), 200

@progress\_bp.route('/progress/material/<int:material\_id>', methods=['POST'])

@token\_required

def track\_material\_progress(current\_user, material\_id):

    material = Material.query.get\_or\_404(material\_id)

    progress = Progress.query.filter\_by(

        user\_id=current\_user.user\_id,

        material\_id=material\_id

    ).first()

    if progress:

        progress.status = 'completed'

    else:

        progress = Progress(

            user\_id=current\_user.user\_id,

            material\_id=material\_id,

            status='completed'

        )

        db.session.add(progress)

    db.session.commit()

    return jsonify({'message': 'Material progress updated!'}), 200

@progress\_bp.route('/progress/test/<int:test\_id>', methods=['POST'])

@token\_required

def track\_test\_progress(current\_user, test\_id):

    data = request.get\_json()

    test = Test.query.get\_or\_404(test\_id)

    progress = Progress.query.filter\_by(

        user\_id=current\_user.user\_id,

        test\_id=test\_id

    ).first()

    if progress:

        progress.score = data['score']

        progress.status = 'completed' if data['score'] >= test.max\_score \* 0.6 else 'failed'

    else:

        progress = Progress(

            user\_id=current\_user.user\_id,

            test\_id=test\_id,

            score=data['score'],

            status='completed' if data['score'] >= test.max\_score \* 0.6 else 'failed'

        )

        db.session.add(progress)

    db.session.commit()

    return jsonify({'message': 'Test progress updated!'}), 200

**users.py (API модуль для управления пользователями)**

from flask import Blueprint, request, jsonify

from backend.utils import token\_required, admin\_required, generate\_password\_hash

from backend.models import User, db

users\_bp = Blueprint('users', \_\_name\_\_)

@users\_bp.route('/users', methods=['GET'])

@token\_required

@admin\_required

def get\_users(current\_user):

    users = User.query.all()

    output = []

    for user in users:

        user\_data = {

            'user\_id': user.user\_id,

            'name': user.name,

            'surname': user.surname,

            'login': user.login,

            'role': user.role,

            'registration\_date': user.registration\_date.strftime('%Y-%m-%d')

        }

        output.append(user\_data)

    return jsonify({'users': output}), 200

@users\_bp.route('/users/<int:user\_id>', methods=['GET'])

@token\_required

def get\_user(current\_user, user\_id):

    if current\_user.user\_id != user\_id and current\_user.role != 'admin':

        return jsonify({'message': 'Unauthorized access!'}), 403

    user = User.query.get\_or\_404(user\_id)

    user\_data = {

        'user\_id': user.user\_id,

        'name': user.name,

        'surname': user.surname,

        'login': user.login,

        'role': user.role,

        'registration\_date': user.registration\_date.strftime('%Y-%m-%d')

    }

    return jsonify({'user': user\_data}), 200

@users\_bp.route('/users/<int:user\_id>', methods=['PUT'])

@token\_required

def update\_user(current\_user, user\_id):

    if current\_user.user\_id != user\_id and current\_user.role != 'admin':

        return jsonify({'message': 'Unauthorized access!'}), 403

    user = User.query.get\_or\_404(user\_id)

    data = request.get\_json()

    user.name = data.get('name', user.name)

    user.surname = data.get('surname', user.surname)

    if 'password' in data:

        user.password = generate\_password\_hash(data['password'])

    db.session.commit()

    return jsonify({'message': 'User updated!'}), 200

@users\_bp.route('/users/<int:user\_id>', methods=['DELETE'])

@token\_required

@admin\_required

def delete\_user(current\_user, user\_id):

    user = User.query.get\_or\_404(user\_id)

    if user.role == 'admin':

        return jsonify({'message': 'Cannot delete admin user!'}), 403

    db.session.delete(user)

    db.session.commit()

    return jsonify({'message': 'User deleted!'}), 200

@users\_bp.route('/users/me', methods=['GET'])

@token\_required

def get\_current\_user(current\_user):

    user\_data = {

        'user\_id': current\_user.user\_id,

        'name': current\_user.name,

        'surname': current\_user.surname,

        'login': current\_user.login,

        'role': current\_user.role,

        'registration\_date': current\_user.registration\_date.strftime('%Y-%m-%d')

    }

    return jsonify({'user': user\_data}), 200

**progress.js (модуль управления функциями прогресса)**

document.addEventListener('DOMContentLoaded', function() {

    checkAuth();

    loadProgress();

    document.getElementById('logout')?.addEventListener('click', logout);

    // Переключение вкладок

    const tabBtns = document.querySelectorAll('.tab-btn');

    tabBtns.forEach(btn => {

        btn.addEventListener('click', () => {

            tabBtns.forEach(b => b.classList.remove('active'));

            btn.classList.add('active');

            const tabId = btn.getAttribute('data-tab');

            document.querySelectorAll('.tab-content').forEach(content => {

                content.classList.remove('active');

            });

            document.getElementById(`${tabId}Tab`).classList.add('active');

        });

    });

});

function checkAuth() {

    const token = localStorage.getItem('token');

    const user = localStorage.getItem('user');

    if (!token || !user) {

        window.location.href = '/frontend/templates/auth/login.html';

        return false;

    }

    return true;

}

function loadProgress() {

    const user = JSON.parse(localStorage.getItem('user'));

    if (!user) {

        logout();

        return;

    }

    const materialsContainer = document.getElementById('materialsProgressContainer');

    const testsContainer = document.getElementById('testsProgressContainer');

    [materialsContainer, testsContainer].forEach(container => {

        if (container) {

            container.innerHTML = '<div class="loading">Загрузка прогресса...</div>';

        }

    });

    authFetch('/progress')

        .then(response => {

            if (!response.ok) {

                return response.json().then(err => {

                    throw new Error(err.message || `Ошибка сервера: ${response.status}`);

                });

            }

            return response.json();

        })

        .then(data => {

            if (!data || !data.progress) {

                throw new Error('Invalid data format');

            }

            const materialsContainer = document.getElementById('materialsProgressContainer');

            const testsContainer = document.getElementById('testsProgressContainer');

            materialsContainer.innerHTML = '';

            testsContainer.innerHTML = '';

            if (data.progress.length === 0) {

                materialsContainer.innerHTML = `

                    <div class="empty-state">

                        <i class="fas fa-book-open"></i>

                        <h3>Прогресс не найден</h3>

                        <p>Начните изучать материалы и проходить тесты</p>

                    </div>

                `;

                return;

            }

            // Прогресс по материалам

            const materialsProgress = data.progress.filter(p => p.material\_id);

            if (materialsProgress.length > 0) {

                materialsProgress.forEach(progress => {

                    const progressItem = document.createElement('div');

                    progressItem.className = 'progress-item';

                    progressItem.innerHTML = `

                        <h3>${progress.title}</h3>

                        <div class="progress-meta">

                            <span class="progress-status status-${progress.status}">

                                ${getStatusText(progress.status)}

                            </span>

                            <span>${new Date(progress.completion\_date).toLocaleDateString()}</span>

                        </div>

                    `;

                    materialsContainer.appendChild(progressItem);

                });

            } else {

                materialsContainer.innerHTML = `

                    <div class="empty-state">

                        <i class="fas fa-book-open"></i>

                        <h3>Нет пройденных материалов</h3>

                        <p>Начните изучать материалы</p>

                    </div>

                `;

            }

            const testsProgress = data.progress.filter(p => p.test\_id);

            if (testsProgress.length > 0) {

                testsProgress.forEach(progress => {

                    const progressItem = document.createElement('div');

                    progressItem.className = 'progress-item';

                    progressItem.innerHTML = `

                        <h3>${progress.title}</h3>

                        <div class="progress-meta">

                            <span class="progress-score">${progress.score} баллов</span>

                            <span class="progress-status status-${progress.status}">

                                ${getStatusText(progress.status)}

                            </span>

                        </div>

                        <div class="progress-date">

                            ${new Date(progress.completion\_date).toLocaleDateString()}

                        </div>

                    `;

                    testsContainer.appendChild(progressItem);

                });

            } else {

                testsContainer.innerHTML = `

                    <div class="empty-state">

                        <i class="fas fa-question-circle"></i>

                        <h3>Нет пройденных тестов</h3>

                        <p>Начните проходить тесты</p>

                    </div>

                `;

            }

        })

        .catch(error => {

            console.error('Error loading progress:', error);

            [materialsContainer, testsContainer].forEach(container => {

                if (container) {

                    container.innerHTML = `

                        <div class="error-state">

                            <i class="fas fa-exclamation-triangle"></i>

                            <h3>Ошибка загрузки прогресса</h3>

                            <p>${error.message || 'Попробуйте позже'}</p>

                        </div>

                    `;

                }

            });

            if (error.message.includes('authenticated')) {

                logout();

            }

        });

}

function getStatusText(status) {

    const statusText = {

        'completed': 'Завершено',

        'started': 'Начато',

        'failed': 'Не пройдено'

    };

    return statusText[status] || status;

}

**Users.js (модуль функций для управления пользователями)**

document.addEventListener('DOMContentLoaded', function() {

    checkAuth();

    loadUsers();

    const user = JSON.parse(localStorage.getItem('user'));

    if (!user || user.role !== 'admin') {

        window.location.href = 'dashboard.html';

        return;

    }

    // Инициализация модального окна

    const modal = document.getElementById('userModal');

    const addBtn = document.getElementById('addUserBtn');

    const closeBtn = document.querySelector('.close');

    const cancelBtn = document.getElementById('cancelUserBtn');

    const userForm = document.getElementById('userForm');

    const togglePassword = document.querySelector('.toggle-password');

    const passwordInput = document.getElementById('modalPassword');

    togglePassword.addEventListener('click', function() {

        const type = passwordInput.getAttribute('type') === 'password' ? 'text' : 'password';

        passwordInput.setAttribute('type', type);

        this.innerHTML = type === 'password' ? '<i class="fas fa-eye"></i>' : '<i class="fas fa-eye-slash"></i>';

    });

    addBtn.addEventListener('click', () => {

        document.getElementById('modalTitle').textContent = 'Добавить пользователя';

        document.getElementById('userId').value = '';

        userForm.reset();

        modal.style.display = 'block';

    });

    closeBtn.addEventListener('click', () => {

        modal.style.display = 'none';

    });

    cancelBtn.addEventListener('click', () => {

        modal.style.display = 'none';

    });

    window.addEventListener('click', (e) => {

        if (e.target === modal) {

            modal.style.display = 'none';

        }

    });

    // Обработка отправки формы

    userForm.addEventListener('submit', async (e) => {

    e.preventDefault();

    const userId = document.getElementById('userId').value;

    const password = document.getElementById('modalPassword').value;

    const originalPassword = document.getElementById('modalPassword').dataset.original;

    const passwordData = password === '' ? {} : { password };

    try {

        const response = await authFetch(`/admin/users/${userId}`, {

            method: 'PUT',

            headers: {

                'Content-Type': 'application/json'

            },

            body: JSON.stringify({

                name: document.getElementById('modalFirstName').value,

                surname: document.getElementById('modalLastName').value,

                login: document.getElementById('modalLogin').value,

                role: document.getElementById('modalRole').value,

                ...passwordData

            })

        });

        if (!response.ok) {

            throw new Error(await response.text());

        }

        modal.style.display = 'none';

        loadUsers();

        alert('Пользователь успешно сохранён');

    } catch (error) {

        console.error('Ошибка сохранения пользователя:', error);

        alert(`Ошибка: ${error.message}`);

    }

});

    document.getElementById('addUserBtn').addEventListener('click', function() {

        document.getElementById('modalPassword').value = '';

    });

    // Поиск пользователей

    document.getElementById('userSearch').addEventListener('input', debounce(loadUsers, 300));

});

async function loadUsers(search = '') {

    try {

        const searchTerm = document.getElementById('userSearch').value;

        const url = `/admin/users${searchTerm ? `?search=${encodeURIComponent(searchTerm)}` : ''}`;

        const response = await authFetch(url);

        if (!response.ok) {

            throw new Error('Ошибка загрузки пользователей');

        }

        const users = await response.json();

        renderUsers(users);

    } catch (error) {

        console.error('Ошибка загрузки пользователей:', error);

        alert('Не удалось загрузить список пользователей');

    }

}

function renderUsers(users) {

    const tbody = document.getElementById('usersTableBody');

    tbody.innerHTML = '';

    if (!users || users.length === 0) {

        tbody.innerHTML = '<tr><td colspan="6" class="empty-state">Пользователи не найдены</td></tr>';

        return;

    }

    users.forEach(user => {

        const tr = document.createElement('tr');

        tr.innerHTML = `

            <td>${user.user\_id}</td>

            <td>${user.name}</td>

            <td>${user.surname}</td>

            <td>${user.login}</td>

            <td>${getRoleName(user.role)}</td>

            <td>

                <button class="action-btn edit-btn" data-id="${user.user\_id}">

                    <i class="fas fa-edit"></i>

                </button>

                <button class="action-btn delete-btn" data-id="${user.user\_id}">

                    <i class="fas fa-trash"></i>

                </button>

            </td>

        `;

        tbody.appendChild(tr);

    });

    //обработчики для кнопок редактирования и удаления

    document.querySelectorAll('.edit-btn').forEach(btn => {

        btn.addEventListener('click', () => editUser(btn.dataset.id));

    });

    document.querySelectorAll('.delete-btn').forEach(btn => {

        btn.addEventListener('click', () => deleteUser(btn.dataset.id));

    });

}

function getRoleName(role) {

    switch(role) {

        case 'admin': return 'Администратор';

        case 'curator': return 'Куратор';

        case 'trainee': return 'Стажёр';

        default: return role;

    }

}

async function editUser(userId) {

    try {

        const response = await authFetch(`/admin/users/${userId}`);

        if (!response.ok) {

            throw new Error('Ошибка загрузки данных пользователя');

        }

        const user = await response.json();

        document.getElementById('modalTitle').textContent = 'Редактировать пользователя';

        document.getElementById('userId').value = user.user\_id;

        document.getElementById('modalFirstName').value = user.name;

        document.getElementById('modalLastName').value = user.surname;

        document.getElementById('modalLogin').value = user.login;

        document.getElementById('modalPassword').value = '';

        document.getElementById('modalRole').value = user.role;

        document.getElementById('modalPassword').dataset.original = user.password;

        document.getElementById('userModal').style.display = 'block';

    } catch (error) {

        console.error('Ошибка редактирования пользователя:', error);

        alert('Не удалось загрузить данные пользователя');

    }

}

async function deleteUser(userId) {

    if (!confirm('Вы уверены, что хотите удалить этого пользователя?')) {

        return;

    }

    try {

        const response = await authFetch(`/admin/users/${userId}`, {

            method: 'DELETE'

        });

        if (!response.ok) {

            throw new Error('Ошибка удаления пользователя');

        }

        loadUsers();

        alert('Пользователь успешно удалён');

    } catch (error) {

        console.error('Ошибка удаления пользователя:', error);

        alert('Не удалось удалить пользователя');

    }

}

function debounce(func, wait) {

    let timeout;

    return function() {

        const context = this, args = arguments;

        clearTimeout(timeout);

        timeout = setTimeout(() => {

            func.apply(context, args);

        }, wait);

    };

}

function checkAuth() {

    const token = localStorage.getItem('token');

    const user = localStorage.getItem('user');

    if (!token || !user) {

        window.location.href = '/frontend/templates/auth/login.html';

        return false;

    }

    return true;

}

**Progress.html (разметка страницы с прогрессом)**

<!DOCTYPE html>

<html lang="ru">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Прогресс | Intern Guide</title>

    <link rel="stylesheet" href="../../assets/css/main.css">

    <link rel="stylesheet" href="../../assets/css/progress.css">

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">

</head>

<body>

    <header>

        <div class="container header-container">

            <div class="logo">

                <img src="https://www.ph4.ru/DL/LOGO\_ICON/i/ingosstrah.gif" alt="Ingosstrakh Logo">

                <div class="logo-text">Intern <span>Guide</span></div>

            </div>

            <nav>

                <ul>

                    <li><a href="../../index.html">Главная</a></li>

                    <li><a href="../materials/index.html">Материалы</a></li>

                    <li><a href="../tests/index.html">Тесты</a></li>

                    <li><a href="index.html" class="active">Прогресс</a></li>

                    <li><a href="../dashboard.html">Профиль</a></li>

                    <li><li><a href="#" id="logoutBtn">Выйти</a></li></li>

                </ul>

            </nav>

        </div>

    </header>

    <main class="container">

        <h1>Мой прогресс</h1>

        <div class="progress-tabs">

            <button class="tab-btn active" data-tab="materials">Материалы</button>

            <button class="tab-btn" data-tab="tests">Тесты</button>

        </div>

        <div class="progress-content">

            <div id="materialsTab" class="tab-content active">

                <div class="progress-grid" id="materialsProgressContainer">

                    <!-- Прогресс по материалам будет загружен через JS -->

                </div>

            </div>

            <div id="testsTab" class="tab-content">

                <div class="progress-grid" id="testsProgressContainer">

                    <!-- Прогресс по тестам будет загружен через JS -->

                </div>

            </div>

        </div>

    </main>

    <footer>

        <div class="container">

            <div class="footer-container">

                <div class="footer-column">

                    <h3>Intern Guide</h3>

                    <p>Платформа для обучения и адаптации стажёров в компании Ингосстрах.</p>

                </div>

                <div class="footer-column">

                    <h3>Разделы</h3>

                    <ul>

                        <li><a href="../../index.html">Главная</a></li>

                        <li><a href="../materials/index.html">Материалы</a></li>

                        <li><a href="../tests/index.html">Тесты</a></li>

                        <li><a href="index.html">Прогресс</a></li>

                        <li><a href="../dashboard.html">Профиль</a></li>

                    </ul>

                </div>

                <div class="footer-column">

                    <h3>Контакты</h3>

                    <ul>

                        <li><a href="#"><i class="fas fa-envelope"></i> interns@ingos.ru</a></li>

                        <li><a href="#"><i class="fas fa-phone"></i> +7 (495) 123-45-67</a></li>

                        <li><a href="#"><i class="fas fa-map-marker-alt"></i> Москва, ул. Покровка, 42</a></li>

                    </ul>

                </div>

            </div>

            <div class="footer-bottom">

                <p>&copy; 2023 Ингосстрах. Все права защищены.</p>

            </div>

        </div>

    </footer>

    <script src="../../assets/js/auth.js"></script>

    <script src="../../assets/js/progress.js"></script>

</body>

</html>

**Users.html (разметка страницы с пользователями)**

<!DOCTYPE html>

<html lang="ru">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Управление пользователями | Intern Guide</title>

    <link rel="stylesheet" href="../assets/css/main.css">

    <link rel="stylesheet" href="../assets/css/users.css">

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">

</head>

<body>

    <header>

        <div class="container header-container">

            <div class="logo">

                <img src="https://www.ph4.ru/DL/LOGO\_ICON/i/ingosstrah.gif" alt="Ingosstrakh Logo">

                <div class="logo-text">Intern <span>Guide</span></div>

            </div>

            <nav>

                <ul>

                    <li><a href="../index.html">Главная</a></li>

                    <li><a href="materials/index.html">Материалы</a></li>

                    <li><a href="tests/index.html">Тесты</a></li>

                    <li><a href="progress/index.html">Прогресс</a></li>

                    <li><a href="dashboard.html">Профиль</a></li>

                    <li><li><a href="#" id="logoutBtn">Выйти</a></li></li>

                </ul>

            </nav>

        </div>

    </header>

    <main class="container">

        <div class="breadcrumbs">

            <a href="../index.html">Главная</a> /

            <a href="dashboard.html">Профиль</a> /

            <span>Управление пользователями</span>

        </div>

        <section class="users-management">

            <h1><i class="fas fa-users-cog"></i> Управление пользователями</h1>

            <div class="users-toolbar">

                <button id="addUserBtn" class="btn btn-primary">

                    <i class="fas fa-plus"></i> Добавить пользователя

                </button>

                <div class="search-box">

                    <input type="text" id="userSearch" placeholder="Поиск пользователей...">

                    <i class="fas fa-search"></i>

                </div>

            </div>

            <div class="users-table-container">

                <table class="users-table">

                    <thead>

                        <tr>

                            <th>ID</th>

                            <th>Имя</th>

                            <th>Фамилия</th>

                            <th>Логин</th>

                            <th>Роль</th>

                            <th>Действия</th>

                        </tr>

                    </thead>

                    <tbody id="usersTableBody">

                        <!-- Сюда будут загружаться пользователи -->

                    </tbody>

                </table>

            </div>

            <div class="pagination">

                <button id="prevPage" disabled><i class="fas fa-chevron-left"></i></button>

                <span id="pageInfo">Страница 1 из 1</span>

                <button id="nextPage" disabled><i class="fas fa-chevron-right"></i></button>

            </div>

        </section>

        <!-- Модальное окно для добавления/редактирования пользователя -->

        <div id="userModal" class="modal">

    <div class="modal-content">

        <span class="close">&times;</span>

        <h2 id="modalTitle"><i class="fas fa-user-edit"></i> Редактировать пользователя</h2>

        <form id="userForm" class="user-form">

            <input type="hidden" id="userId">

            <div class="form-row">

                <div class="form-group">

                    <label for="modalFirstName"><i class="fas fa-user"></i> Имя</label>

                    <input type="text" id="modalFirstName" required>

                </div>

                <div class="form-group">

                    <label for="modalLastName"><i class="fas fa-user-tag"></i> Фамилия</label>

                    <input type="text" id="modalLastName" required>

                </div>

            </div>

            <div class="form-group">

                <label for="modalLogin"><i class="fas fa-at"></i> Логин</label>

                <input type="text" id="modalLogin" required>

            </div>

            <div class="form-group password-group">

    <label for="modalPassword"><i class="fas fa-lock"></i> Пароль</label>

    <div class="password-input-container">

        <input type="password" id="modalPassword" placeholder="Оставьте пустым, чтобы не изменять">

        <button type="button" class="toggle-password" aria-label="Показать пароль">

            <i class="fas fa-eye"></i>

        </button>

    </div>

    <small class="password-hint">Оставьте поле пустым, чтобы сохранить текущий пароль</small>

</div>

            <div class="form-group">

                <label for="modalRole"><i class="fas fa-user-shield"></i> Роль</label>

                <div class="select-wrapper">

                    <select id="modalRole" required>

                        <option value="trainee">Стажёр</option>

                        <option value="curator">Куратор</option>

                        <option value="admin">Администратор</option>

                    </select>

                    <i class="fas fa-chevron-down"></i>

                </div>

            </div>

            <div class="form-actions">

                <button type="button" id="cancelUserBtn" class="btn btn-outline">

                    <i class="fas fa-times"></i> Отмена

                </button>

                <button type="submit" class="btn btn-primary">

                    <i class="fas fa-save"></i> Сохранить

                </button>

            </div>

        </form>

    </div>

</div>

    </main>

    <footer>

        <div class="container">

            <div class="footer-container">

                <div class="footer-column">

                    <h3>Intern Guide</h3>

                    <p>Платформа для обучения и адаптации стажёров в компании Ингосстрах.</p>

                </div>

                <div class="footer-column">

                    <h3>Разделы</h3>

                    <ul>

                        <li><a href="./index.html">Главная</a></li>

                        <li><a href="materials/index.html">Материалы</a></li>

                        <li><a href="tests/index.html">Тесты</a></li>

                        <li><a href="progress/index.html">Прогресс</a></li>

                    </ul>

                </div>

                <div class="footer-column">

                    <h3>Контакты</h3>

                    <ul>

                        <li><a href="#"><i class="fas fa-envelope"></i> interns@ingos.ru</a></li>

                        <li><a href="#"><i class="fas fa-phone"></i> +7 (495) 123-45-67</a></li>

                        <li><a href="#"><i class="fas fa-map-marker-alt"></i> Москва, ул. Покровка, 42</a></li>

                    </ul>

                </div>

            </div>

            <div class="footer-bottom">

                <p>&copy; 2025 Ингосстрах. Все права защищены.</p>

            </div>

        </div>

    </footer>

    <script src="../assets/js/auth.js"></script>

    <script src="../assets/js/users.js"></script>

</body>

</html>

Код для оставшихся модулей (auth, progress, tests), находится в приложении В.

В результате разработки был создан веб-сайт Intern Guide, который представляет собой полноценную образовательную экосистему, которая:

* Обеспечивает удобное обучение для стажеров.
* Дает кураторам инструменты для создания контента.
* Позволяет администраторам гибко управлять платформой.

## **2.5 Тестирование программного продукта**

Разработка тестовой документации для системы адаптации стажёров Intern Guide проводилась с особой тщательностью, учитывая важность системы для бизнес-процессов компании. Качество тестирования напрямую влияет на эффективность адаптации новых сотрудников.

1. **Анализ требований и подготовка**

На начальном этапе были выполнены следующие действия:

* Проведен детальный анализ технического задания с выделением ключевых требований
* Определены приоритетные направления тестирования
* Составлен перечень возможных рисков

Были выбраны основные виды тестирования:

* Функциональное тестирование всех модулей системы
* Интеграционная проверка взаимодействия компонентов
* Юзабилити-тестирование интерфейса

1. **Подготовка тестовых данных**

Для обеспечения комплексной проверки системы были подготовлены:

* Детализированные тест-кейсы с пошаговыми инструкциями
* Чек-листы для проверки ключевых функций
* Сценарии проверки граничных условий

Каждый тест-кейс содержал:

* Четкую последовательность действий
* Ожидаемые результаты
* Критерии успешного прохождения

1. **Планирование и проведение тестирования**

Был разработан детальный план тестирования. После каждого этапа тестирования формировались отчеты, содержащие:

* Общее количество выполненных тестов
* Найденные дефекты с указанием приоритетов

Реализованный подход к тестированию позволил:

* Выявить и устранить критические ошибки до запуска системы
* Оптимизировать пользовательский интерфейс
* Обеспечить стабильную работу под нагрузкой
* Повысить общее качество продукта

Таким образом, подход к тестированию PlantDoctor был максимально детализированным, что позволило создать надежный и удобный продукт для пользователей.

Для проведения тестирования функциональных возможностей программного продукта, а также взаимодействия модулей и компонентов между собой были разработаны модули автоматизированного тестирования (UNIT-тесты), которые позволяют удостовериться в правильности прохождения функциями тестов, а также ускоряют сам процесс тестирования.

**Conftest.py (устанавливает основные зависимости для тестирования)**

import pytest

from backend import create\_app

from backend.models import User, db  # Добавлен импорт User

from werkzeug.security import generate\_password\_hash

@pytest.fixture

def app():

    app = create\_app()

    app.config['TESTING'] = True

    app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///:memory:'

    app.config['SECRET\_KEY'] = 'test-secret-key'

    from backend.routes.auth import auth\_bp

    from backend.routes.materials import materials\_bp

    from backend.routes.progress import progress\_bp

    from backend.routes.tests import tests\_bp

    from backend.routes.users import users\_bp

    app.register\_blueprint(auth\_bp, url\_prefix='/auth')

    app.register\_blueprint(materials\_bp, url\_prefix='/materials')

    app.register\_blueprint(progress\_bp, url\_prefix='/progress')

    app.register\_blueprint(tests\_bp, url\_prefix='/tests')

    app.register\_blueprint(users\_bp, url\_prefix='/users')

    with app.app\_context():

        db.create\_all()

        yield app

        db.drop\_all()

@pytest.fixture

def client(app):

    return app.test\_client()

@pytest.fixture(autouse=True)

def cleanup\_db(app):

    with app.app\_context():

        db.session.remove()

        db.drop\_all()

        db.create\_all()

@pytest.fixture

def test\_user(app):

    with app.app\_context():

        user = User(

            name='Test',

            surname='User',  # Обязательное поле

            login='testuser',

            password=generate\_password\_hash('testpassword'),

            role='trainee'

        )

        db.session.add(user)

        db.session.commit()

        yield user

        db.session.delete(user)

        db.session.commit()

**Test\_auth.py (тестирование регистрации и авторизации)**

import pytest

from datetime import datetime, timedelta

import jwt

from flask import jsonify

from backend.config import Config

from backend.models import User, db

from werkzeug.security import generate\_password\_hash

from backend.routes.auth import token\_required

# Добавим временный маршрут для тестов

@pytest.fixture(autouse=True)

def add\_protected\_route(app):

    @app.route('/protected-route')

    @token\_required

    def protected\_route(current\_user):

        return jsonify({'message': 'Success'})

def test\_register\_success(client):

    # Очистка перед тестом

    with client.application.app\_context():

        User.query.filter\_by(login='newuser').delete()

        db.session.commit()

    response = client.post('/auth/register', json={

        'name': 'New',

        'surname': 'User',

        'login': 'newuser',

        'password': 'newpassword'

    })

    assert response.status\_code == 201

    assert 'успешно зарегистрирован' in response.json['message']

def test\_register\_missing\_fields(client):

    response = client.post('/auth/register', json={

        'name': 'New',

        'surname': 'User',

        'login': 'newuser'

        # Нет пароля

    })

    assert response.status\_code == 400

    assert 'обязательные поля' in response.json['message']

def test\_login\_success(client, test\_user):

    response = client.post('/auth/login', json={

        'login': 'testuser',

        'password': 'testpassword'

    })

    assert response.status\_code == 200

    assert 'token' in response.json

    assert response.json['user']['login'] == 'testuser'

def test\_login\_invalid\_credentials(client, test\_user):

    response = client.post('/auth/login', json={

        'login': 'testuser',

        'password': 'wrongpassword'

    })

    assert response.status\_code == 401

    assert 'Invalid credentials' in response.json['message']

def test\_protected\_route\_success(client, test\_user):

    token = jwt.encode({

        'user\_id': test\_user.user\_id,

        'exp': datetime.utcnow() + timedelta(hours=1)

    }, Config.SECRET\_KEY, algorithm="HS256")

    response = client.get('/protected-route', headers={

        'Authorization': f'Bearer {token}'

    })

    assert response.status\_code == 200

    assert response.json['message'] == 'Success'

def test\_protected\_route\_missing\_token(client):

    response = client.get('/protected-route')

    assert response.status\_code == 401

    assert 'Требуется токен' in response.json['message']

def test\_protected\_route\_invalid\_token(client):

    response = client.get('/protected-route', headers={

        'Authorization': 'Bearer invalidtoken'

    })

    assert response.status\_code == 401

    assert 'Неверный токен' in response.json['message']

**Test\_materials.py (тестирование работы с материалами)**

import pytest

from datetime import datetime

from backend.models import Material, User, db

from werkzeug.security import generate\_password\_hash

@pytest.fixture

def test\_curator(app, client):

    with app.app\_context():

        user = User(

            name='Curator',

            surname='Test',

            login='curator',

            password=generate\_password\_hash('curatorpass'),

            role='curator'

        )

        db.session.add(user)

        db.session.commit()

        # Login to get token

        response = client.post('/auth/login', json={

            'login': 'curator',

            'password': 'curatorpass'

        })

        return {

        'token': response.json['token'],

        'user\_id': user.user\_id

    }

def test\_create\_material(client, test\_curator):

    headers = {'Authorization': f'Bearer {test\_curator["token"]}'}

    response = client.post('/materials/materials', json={

        'title': 'New Material',

        'content': 'Test content',

        'category': 'Test category'

    }, headers=headers)

    assert response.status\_code == 201

    assert 'created' in response.json['message'].lower()

def test\_get\_materials(client, test\_curator):

    headers = {'Authorization': f'Bearer {test\_curator["token"]}'}

    response = client.get('/materials/materials', headers=headers)

    assert response.status\_code == 200

    assert isinstance(response.json['materials'], list)

def test\_update\_material(client, test\_curator):

    # First create a material

    with client.application.app\_context():

        curator = User.query.filter\_by(login='curator').first()

        material = Material(

            title='Test Material',

            author\_id=curator.user\_id,

            content='Old content',

            category='Old category'

        )

        db.session.add(material)

        db.session.commit()

        material\_id = material.material\_id

    headers = {'Authorization': f'Bearer {test\_curator["token"]}'}

    response = client.put(f'/materials/materials/{material\_id}', json={

        'title': 'Updated Title',

        'content': 'Updated content'

    }, headers=headers)

    assert response.status\_code == 200

    assert 'материал успешно обновлен' in response.json['message'].lower()

**Test\_models.py (тестирование моделей БД)**

import pytest

from datetime import datetime

from backend.models import db, User, Material, Test, Question, Progress

from backend import create\_app

from werkzeug.security import generate\_password\_hash, check\_password\_hash

@pytest.fixture

def app():

    app = create\_app()

    app.config['TESTING'] = True

    app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///:memory:'

    with app.app\_context():

        db.create\_all()

        yield app

        db.drop\_all()

def test\_user\_model(app):

    with app.app\_context():

        user = User(

            name='John',

            surname='Doe',

            login='johndoe',

            password='hashed\_password',

            role='trainee'

        )

        db.session.add(user)

        db.session.commit()

        assert user.user\_id is not None

        assert user.registration\_date is not None

        assert user.role == 'trainee'

def test\_material\_model(app):

    with app.app\_context():

        user = User(

            name='Author',

            surname='User',

            login='author',

            password='hashed\_password',

            role='curator'

        )

        db.session.add(user)

        db.session.commit()

        material = Material(

            title='Test Material',

            author\_id=user.user\_id,

            content='Test content',

            category='Test category'

        )

        db.session.add(material)

        db.session.commit()

        assert material.material\_id is not None

        assert material.upload\_date is not None

        assert material.author == user

        assert material.title == 'Test Material'

def test\_test\_model(app):

    with app.app\_context():

        user = User(

            name='Test',

            surname='Creator',

            login='creator',

            password='hashed\_password',

            role='curator'

        )

        db.session.add(user)

        db.session.commit()

        test = Test(

            title='Test Title',

            creator\_id=user.user\_id,

            description='Test description',

            max\_score=10

        )

        db.session.add(test)

        db.session.commit()

        assert test.test\_id is not None

        assert test.creation\_date is not None

        assert test.creator == user

        assert test.title == 'Test Title'

        assert test.description == 'Test description'

def test\_question\_model(app):

    with app.app\_context():

        user = User(

            name='Test',

            surname='Creator',

            login='creator',

            password='hashed\_password',

            role='curator'

        )

        db.session.add(user)

        db.session.commit()

        test = Test(

            title='Test Title',

            creator\_id=user.user\_id,

            description='Test description',

            max\_score=10

        )

        db.session.add(test)

        db.session.commit()

        question = Question(

            test\_id=test.test\_id,

            text='Test question',

            points=2,

            option\_a='Option A',

            option\_b='Option B',

            correct\_option='A'

        )

        db.session.add(question)

        db.session.commit()

        assert question.question\_id is not None

        assert question.test == test

        assert str(question) == '<Question 1>'

def test\_progress\_model(app):

    with app.app\_context():

        user = User(

            name='John',

            surname='Doe',

            login='johndoe',

            password='hashed\_password',

            role='trainee'

        )

        db.session.add(user)

        db.session.commit()

        test = Test(

            title='Test Title',

            creator\_id=user.user\_id,

            description='Test description',

            max\_score=10

        )

        db.session.add(test)

        db.session.commit()

        progress = Progress(

            user\_id=user.user\_id,

            test\_id=test.test\_id,

            score=8,

            status='completed'

        )

        db.session.add(progress)

        db.session.commit()

        assert progress.progress\_id is not None

        assert progress.completion\_date is not None

        assert progress.user == user

        assert progress.test == test

        assert progress.status == 'completed'

**Test\_progress.py (проверка прогресса)**

import pytest

from backend.models import Progress, User, Material, Test, db

from werkzeug.security import generate\_password\_hash

@pytest.fixture

def test\_data(app, client):

    with app.app\_context():

        # Создаем пользователей с обязательным surname

        author = User(

            name='Author',

            surname='AuthorSurname',  # Обязательное поле

            login='author',

            password=generate\_password\_hash('pass'),

            role='curator'

        )

        db.session.add(author)

        db.session.commit()

        user = User(

            name='Progress',

            surname='TestSurname',  # Обязательное поле

            login='progresstest',

            password=generate\_password\_hash('testpass'),

            role='trainee'

        )

        db.session.add(user)

        db.session.commit()

        # Создаем материал с указанием автора

        material = Material(

            title='Test Material',

            author\_id=author.user\_id,

            content='Test content'

        )

        db.session.add(material)

        # Создаем тест

        test = Test(

            title='Test Test',

            creator\_id=author.user\_id,

            max\_score=10

        )

        db.session.add(test)

        db.session.commit()

        # Получаем токен для пользователя

        login\_res = client.post('/auth/login', json={

            'login': 'progresstest',

            'password': 'testpass'

        })

        token = login\_res.json['token']

        yield {

            'user\_id': user.user\_id,

            'material\_id': material.material\_id,

            'test\_id': test.test\_id,

            'token': token

        }

def test\_track\_material\_progress(client, test\_data):

    headers = {'Authorization': f'Bearer {test\_data["token"]}'}

    response = client.post(

        f'/progress/progress/material/{test\_data["material\_id"]}',

        headers=headers

    )

    assert response.status\_code == 200

def test\_track\_test\_progress(client, test\_data):

    headers = {'Authorization': f'Bearer {test\_data["token"]}'}

    response = client.post(

        f'/progress/progress/test/{test\_data["test\_id"]}',

        json={'score': 8},

        headers=headers

    )

    assert response.status\_code == 200

    assert 'updated' in response.json['message'].lower()

def test\_get\_user\_progress(client, test\_data):

    headers = {'Authorization': f'Bearer {test\_data["token"]}'}

    response = client.get('/progress/progress', headers=headers)

    assert response.status\_code == 200

    assert isinstance(response.json['progress'], list)

**Tets\_tests.py (работа с тестами)**

import pytest

from backend.models import Test, Question, User, db

from werkzeug.security import generate\_password\_hash

@pytest.fixture

def test\_creator\_token(app, client):

    with app.app\_context():

        creator = User(

            name='Test',

            surname='Creator',

            login='testcreator',

            password=generate\_password\_hash('creatorpass'),

            role='curator'

        )

        db.session.add(creator)

        db.session.commit()

        response = client.post('/auth/login', json={

            'login': 'testcreator',

            'password': 'creatorpass'

        })

        return response.json['token']

def test\_create\_test(client, test\_creator\_token):

    headers = {'Authorization': f'Bearer {test\_creator\_token}'}

    test\_data = {

        'title': 'New Test',

        'description': 'Test description',

        'questions': [

            {

                'text': 'Question 1',

                'points': 2,

                'options': {'A': 'Option A', 'B': 'Option B'},

                'correct\_option': 'A'

            }

        ]

    }

    response = client.post('/tests/tests', json=test\_data, headers=headers)

    assert response.status\_code == 201

    assert 'created' in response.json['message'].lower()

def test\_get\_tests(client, test\_creator\_token):

    headers = {'Authorization': f'Bearer {test\_creator\_token}'}

    response = client.get('/tests/tests', headers=headers)

    assert response.status\_code == 200

    assert isinstance(response.json['tests'], list)

def test\_get\_test\_details(client, test\_creator\_token):

    headers = {'Authorization': f'Bearer {test\_creator\_token}'}

    with client.application.app\_context():

        creator = User.query.filter\_by(login='testcreator').first()

        test = Test(title='Sample Test', creator\_id=creator.user\_id, max\_score=10)

        db.session.add(test)

        db.session.commit()

        test\_id = test.test\_id

    response = client.get(f'/tests/tests/{test\_id}', headers=headers)

    assert response.status\_code == 200

    assert 'title' in response.json['test']

**Test\_users.py (управление пользователями)**

import pytest

from backend.models import User, db

from werkzeug.security import generate\_password\_hash

@pytest.fixture

def admin\_token(app, client):

    with app.app\_context():

        admin = User(

            name='Admin',

            surname='User',

            login='admin',

            password=generate\_password\_hash('adminpass'),

            role='admin'

        )

        db.session.add(admin)

        db.session.commit()

        response = client.post('/auth/login', json={

            'login': 'admin',

            'password': 'adminpass'

        })

        return response.json['token']

def test\_get\_users(client, admin\_token):

    headers = {'Authorization': f'Bearer {admin\_token}'}

    response = client.get('/users/users', headers=headers)

    assert response.status\_code == 200

    assert isinstance(response.json['users'], list)

def test\_update\_user\_profile(client):

    with client.application.app\_context():

        user = User(

            name='Test',

            surname='User',

            login='testuser',

            password=generate\_password\_hash('testpass'),

        )

        db.session.add(user)

        db.session.commit()

        user\_id = user.user\_id

    login\_response = client.post('/auth/login', json={

        'login': 'testuser',

        'password': 'testpass'

    })

    assert 'token' in login\_response.json

    token = login\_response.json['token']

    headers = {'Authorization': f'Bearer {token}'}

    update\_response = client.put(f'/users/users/{user\_id}', json={

        'name': 'Updated Name'

    }, headers=headers)

    assert update\_response.status\_code == 200

    assert 'updated' in update\_response.json['message'].lower()

**Test\_utils.py (тестирование утилит сайта)**

import pytest

import jwt

from flask import Flask, jsonify, request

from backend.utils import token\_required, admin\_required, curator\_required

from backend.models import User, db

from backend.config import Config

from datetime import datetime

@pytest.fixture

def app():

    app = Flask(\_\_name\_\_)

    app.config['TESTING'] = True

    app.config['SECRET\_KEY'] = Config.SECRET\_KEY

    app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///:memory:'

    db.init\_app(app)

    with app.app\_context():

        db.create\_all()

        yield app

        db.drop\_all()

def test\_token\_required\_success(app):

    with app.app\_context():

        user = User(

            name='Test',

            surname='User',

            login='testuser',

            password='hashed\_password',

            role='trainee'

        )

        db.session.add(user)

        db.session.commit()

        token = jwt.encode({

            'user\_id': user.user\_id,

            'exp': datetime.utcnow()

        }, Config.SECRET\_KEY, algorithm="HS256")

        @app.route('/protected')

        @token\_required

        def protected\_route(current\_user):

            return jsonify({'message': 'Success', 'user\_id': current\_user.user\_id})

        with app.test\_client() as client:

            response = client.get(

                '/protected',

                headers={'Authorization': f'Bearer {token}'}

            )

            assert response.status\_code == 200

            assert response.json['message'] == 'Success'

            assert response.json['user\_id'] == user.user\_id

def test\_token\_required\_missing\_token(app):

    with app.app\_context():

        @app.route('/protected')

        @token\_required

        def protected\_route(current\_user):

            return jsonify({'message': 'Success'})

        with app.test\_client() as client:

            response = client.get('/protected')

            assert response.status\_code == 401

            assert response.json['message'] == 'Token is missing!'

def test\_admin\_required\_success(app):

    with app.app\_context():

        admin = User(

            name='Admin',

            surname='User',

            login='admin',

            password='hashed\_password',

            role='admin'

        )

        db.session.add(admin)

        db.session.commit()

        @app.route('/admin')

        @token\_required

        @admin\_required

        def admin\_route(current\_user):

            return jsonify({'message': 'Admin access granted'})

        token = jwt.encode({

            'user\_id': admin.user\_id,

            'exp': datetime.utcnow()

        }, Config.SECRET\_KEY, algorithm="HS256")

        # Test with admin user

        with app.test\_client() as client:

            response = client.get(

                '/admin',

                headers={'Authorization': f'Bearer {token}'}

            )

            assert response.status\_code == 200

            assert response.json['message'] == 'Admin access granted'

def test\_admin\_required\_denied(app):

    with app.app\_context():

        user = User(

            name='Regular',

            surname='User',

            login='regular',

            password='hashed\_password',

            role='trainee'

        )

        db.session.add(user)

        db.session.commit()

        @app.route('/admin')

        @token\_required

        @admin\_required

        def admin\_route(current\_user):

            return jsonify({'message': 'Admin access granted'})

        token = jwt.encode({

            'user\_id': user.user\_id,

            'exp': datetime.utcnow()

        }, Config.SECRET\_KEY, algorithm="HS256")

        with app.test\_client() as client:

            response = client.get(

                '/admin',

                headers={'Authorization': f'Bearer {token}'}

            )

            assert response.status\_code == 403

            assert response.json['message'] == 'Admin access required!'

def test\_curator\_required\_success(app):

    with app.app\_context():

        curator = User(

            name='Curator',

            surname='User',

            login='curator',

            password='hashed\_password',

            role='curator'

        )

        db.session.add(curator)

        db.session.commit()

        @app.route('/curator')

        @token\_required

        @curator\_required

        def curator\_route(current\_user):

            return jsonify({'message': 'Curator access granted'})

        token = jwt.encode({

            'user\_id': curator.user\_id,

            'exp': datetime.utcnow()

        }, Config.SECRET\_KEY, algorithm="HS256")

        with app.test\_client() as client:

            response = client.get(

                '/curator',

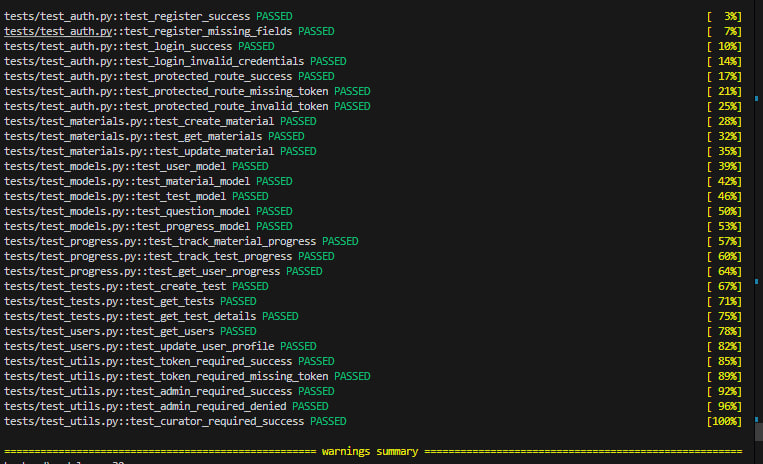
                headers={'Authorization': f'Bearer {token}'}

            )

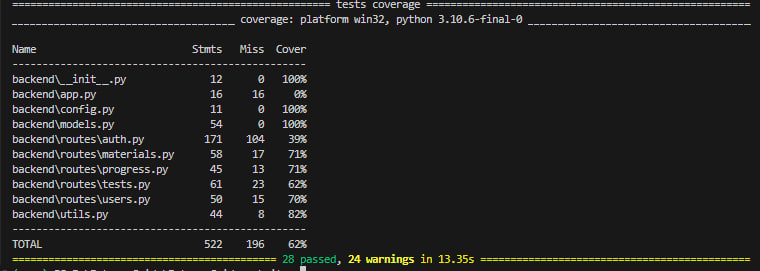
            assert response.status\_code == 200

            assert response.json['message'] == 'Curator access granted'

На рисунках ниже представлен результат выполнения данных тестов, а также процент покрытия данными тестами программного кода сайта.

****

**Рисунок 11. Результат выполнения ЮНИТ-тестов**

****

**Рисунок 12. Покрытие программного кода тестами**

## **2.6 Разработка модуля математического моделирования**

Система Internship Guide автоматически распределяет учебную нагрузку стажёров, учитывая:

* Срок стажировки (в месяцах).
* Количество рабочих часов в неделю.
* Объём материалов (статьи и тесты).

**Цель:** оптимально распределить статьи и тесты по неделям так, чтобы:

1. Все материалы были пройдены в срок.
2. Нагрузка была равномерной.
3. Количество статей и тестов в неделю было целым числом (нельзя прочитать 2.5 статьи).

Алгоритм использует Google OR-Tools (SCIP-солвер) для решения задачи оптимизации с целочисленными переменными.

**Формализация:**

* + **Переменные:**
  + articlesPerWeek – количество статей в неделю (целое число).
  + testsPerWeek – количество тестов в неделю (целое число).
* **Ограничения:**
  + Общее количество пройденных статей ≥ требуемого.
  + Общее количество пройденных тестов ≥ требуемого.
  + Еженедельная нагрузка ≤ доступного времени (статьи + тесты).
* **Целевая функция системы:** задача минимизировать суммарную нагрузку: articlesPerWeek + testsPerWeek → min

Для этого был написан следующий алгоритм:

using System;

using Google.OrTools.LinearSolver;

public class InternshipScheduler

{

public static void ScheduleInternship(int internshipMonths, int hoursPerWeek, int totalArticles, int totalTests)

{

// Конвертация месяцев в недели

int totalWeeks = internshipMonths \* 4;

int totalHours = totalWeeks \* hoursPerWeek;

// Создание решателя

Solver solver = Solver.CreateSolver("SCIP");

// Переменные (целочисленные):

// - articlesPerWeek: сколько статей в неделю (целое число)

// - testsPerWeek: сколько тестов в неделю (целое число)

Variable articlesPerWeek = solver.MakeIntVar(0, double.PositiveInfinity, "articlesPerWeek");

Variable testsPerWeek = solver.MakeIntVar(0, double.PositiveInfinity, "testsPerWeek");

// Ограничения:

// 1. Все статьи должны быть пройдены

solver.Add(articlesPerWeek \* totalWeeks >= totalArticles);

// 2. Все тесты должны быть пройдены

solver.Add(testsPerWeek \* totalWeeks >= totalTests);

// 3. Общее время в неделю (например, 1 статья = 1 час, 1 тест = 0.5 часа)

solver.Add(1 \* articlesPerWeek + 0.5 \* testsPerWeek <= hoursPerWeek);

// Цель: минимизировать суммарную нагрузку (статьи + тесты)

Objective objective = solver.Objective();

objective.SetCoefficient(articlesPerWeek, 1);

objective.SetCoefficient(testsPerWeek, 1);

objective.SetMinimization();

// Решение

Solver.ResultStatus resultStatus = solver.Solve();

// Вывод результатов

if (resultStatus == Solver.ResultStatus.OPTIMAL)

{

Console.WriteLine("Оптимальное расписание:");

Console.WriteLine($"- Статей в неделю: {articlesPerWeek.SolutionValue()}");

Console.WriteLine($"- Тестов в неделю: {testsPerWeek.SolutionValue()}");

Console.WriteLine($"- Всего статей: {articlesPerWeek.SolutionValue() \* totalWeeks} (требуется: {totalArticles})");

Console.WriteLine($"- Всего тестов: {testsPerWeek.SolutionValue() \* totalWeeks} (требуется: {totalTests})");

Console.WriteLine($"- Проверка: {(articlesPerWeek.SolutionValue() \* totalWeeks >= totalArticles ? "OK" : "Не все статьи пройдены")}");

Console.WriteLine($"- Проверка: {(testsPerWeek.SolutionValue() \* totalWeeks >= totalTests ? "OK" : "Не все тесты пройдены")}");

}

else

{

Console.WriteLine("Оптимальное решение не найдено.");

Console.WriteLine("Попробуйте увеличить количество часов или уменьшить количество материалов.");

}

}

public static void Main()

{

// Пример вызова

Console.WriteLine("Распредление нагрузки на стажера");

Console.WriteLine("---------------------------------");

Console.Write("Введите продолительность стажировки (в месяцах): ");

int internshipMonths = Convert.ToInt32(Console.ReadLine());

Console.Write("Введите количество часов в неделю на работе: ");

int hoursPerWeek = Convert.ToInt32(Console.ReadLine());

Console.Write("Введите количество статей, которые надо пройти: ");

int totalArticles = Convert.ToInt32(Console.ReadLine());

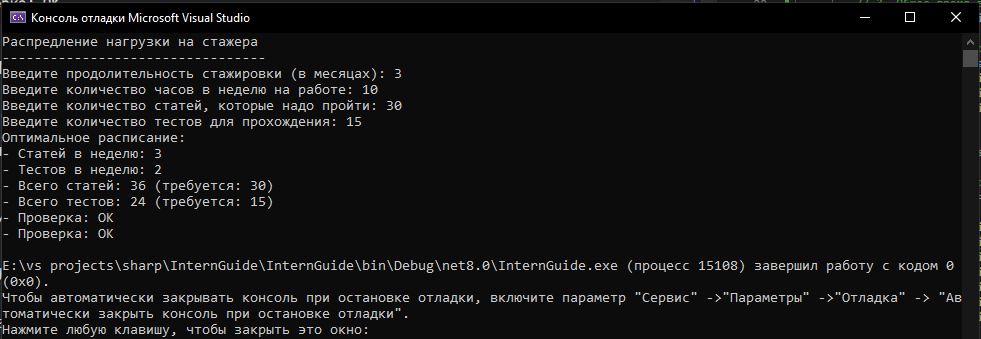
Console.Write("Введите количество тестов для прохождения: ");

int totalTests = Convert.ToInt32(Console.ReadLine());

ScheduleInternship(internshipMonths, hoursPerWeek, totalArticles, totalTests);

}

}

****

**Рисунок 13. Результат выполнения программы**

# **ЗАКЛЮЧЕНИЕ**

Работа над проектом Intern Guide для СПАО «Ингосстрах» стала важным этапом в профессиональном развитии как будущего специалиста в области разработки программного обеспечения. В процессе проектирования и реализации системы были углублены знания в современных технологиях веб-разработки, получив ценный практический опыт.

На этапе проектирования особое внимание было уделено анализу требований заказчика. Это позволило создать систему, которая не просто соответствует формальным требованиям, но и реально решает проблемы пользователей. Разработка UML-диаграмм помогла четко структурировать архитектуру приложения до начала программирования.

Практическая реализация проекта велась с использованием современного технологического стека. Для backend-части был выбран Python с фреймворком Flask, что обеспечило высокую производительность API и простоту интеграции с существующей инфраструктурой компании. Frontend-часть разрабатывалась при помощи разметки HTML и реализации функций на JavaScript, что позволило создать интуитивно понятный интерфейс с поддержкой всех необходимых функций.

Важным аспектом проекта стало тестирование системы. Реализация модульных, интеграционных и нагрузочных тестов позволила определить все ошибки и недочеты на всех этапах создания системы.

В результате была создана система, которая успешно решает поставленные задачи: сокращает время адаптации новых сотрудников, автоматизирует процессы обучения и предоставляет аналитические инструменты для кураторов.

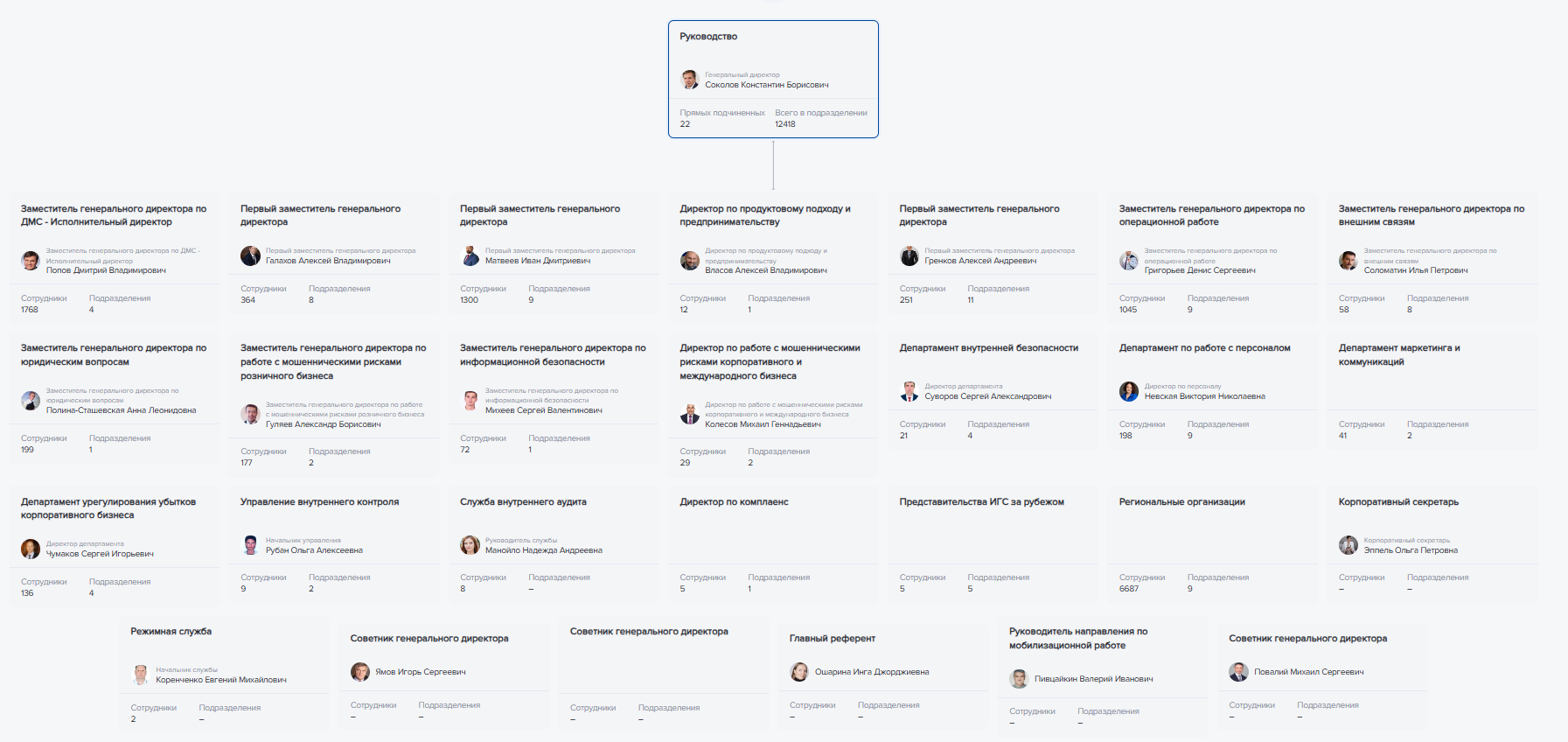
# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Современный UX/UI-дизайн / Аарон Уолтер. — М.: Манн, 2021.
2. Тестирование ПО / Роман Савин. — СПб.: БХВ, 2019.
3. UML 2.5. Подробное руководство / Лешек Мацяшек. — ДМК, 2022.
4. М. Гринвальд, С. Харрисон. Flask. Разработка веб-приложений на Python. — СПб.: Питер, 2021. — 352 с.
5. Д. Грубер. SQLAlchemy. Профессиональное использование. — М.: ДМК Пресс, 2020. — 488 с.
6. А. Швец. Python и тестирование. Полное руководство. — 2-е изд. — М.: БХВ-Петербург, 2022. — 672 с.
7. SQL для анализа данных / Кэтрин Тан. — O’Reilly, 2021.
8. Системное мышление / Доннелла Медоуз. — Манн, 2020.
9. Р. Савин. Тестирование DOT COM. — 5-е изд. — М.: Альфа-книга, 2023. — 816 с.
10. Д. Флэнаган. JavaScript. Подробное руководство. — 7-е изд. — СПб.: Символ-Плюс, 2022. — 1088 с.
11. А. Купер. Интерфейс. Основы проектирования взаимодействия. — 4-е изд. — СПб.: Питер, 2020. — 496 с.
12. Р. Мартин. Чистый код. Создание, анализ и рефакторинг. — М.: Вильямс, 2022. — 464 с.
13. Дж. Уиттакер. Как тестируют в Google. — М.: Альпина Паблишер, 2021. — 382 с.
14. В. Дронов. HTML5, CSS3 и JavaScript. Исчерпывающее руководство. — 5-е изд. — СПб.: БХВ-Петербург, 2023. — 1088 с.
15. М. Фаулер. Рефакторинг. Улучшение существующего кода. — 2-е изд. — М.: Вильямс, 2020. — 448 с.
16. К. Хорстманн. Java Script для профессионалов. — М.: Диалектика, 2022. — 1088 с.

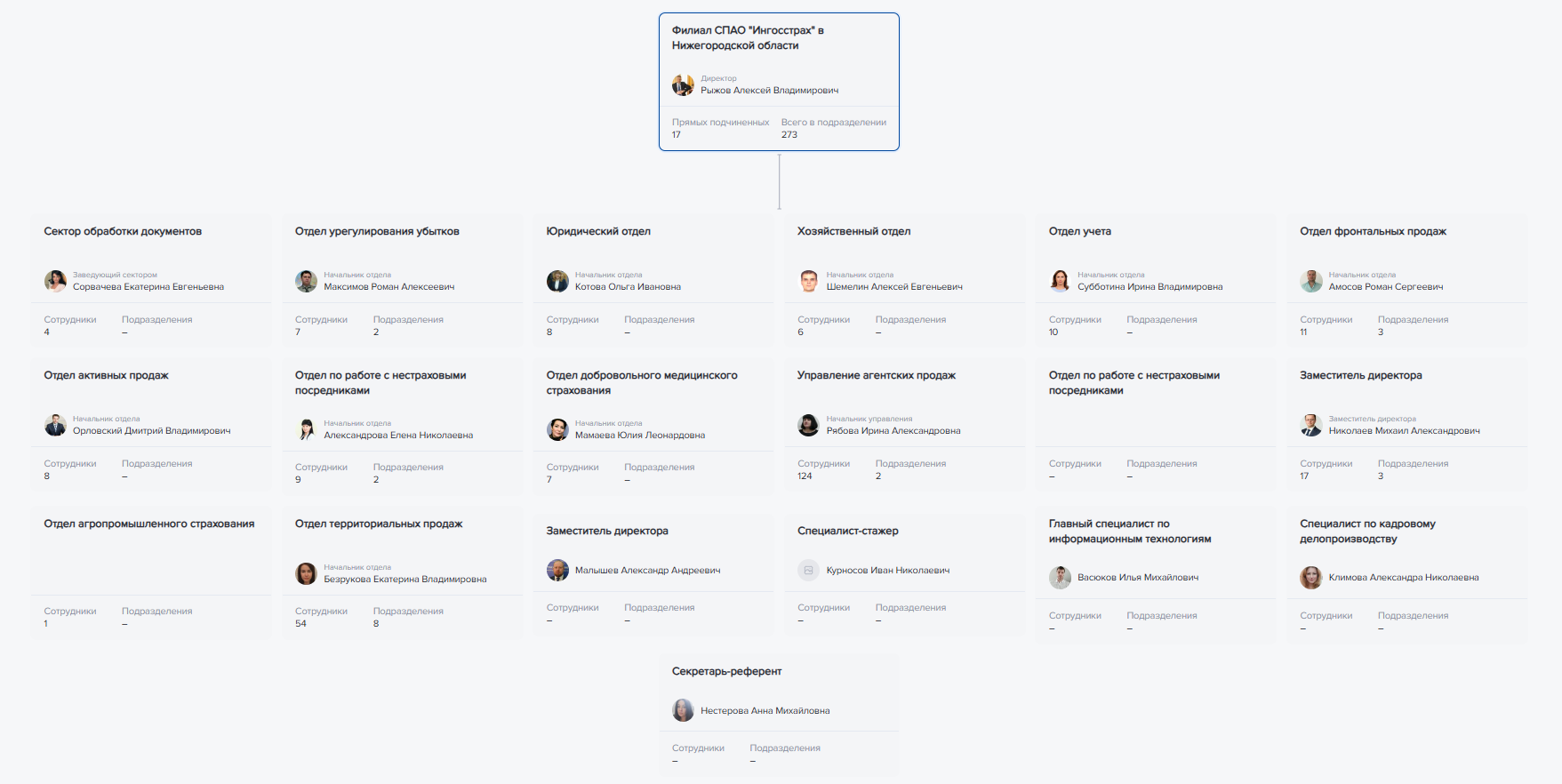
# **ПРИЛОЖЕНИЯ**

## **ПРИЛОЖЕНИЕ А**

Оргструктура предприятия

****

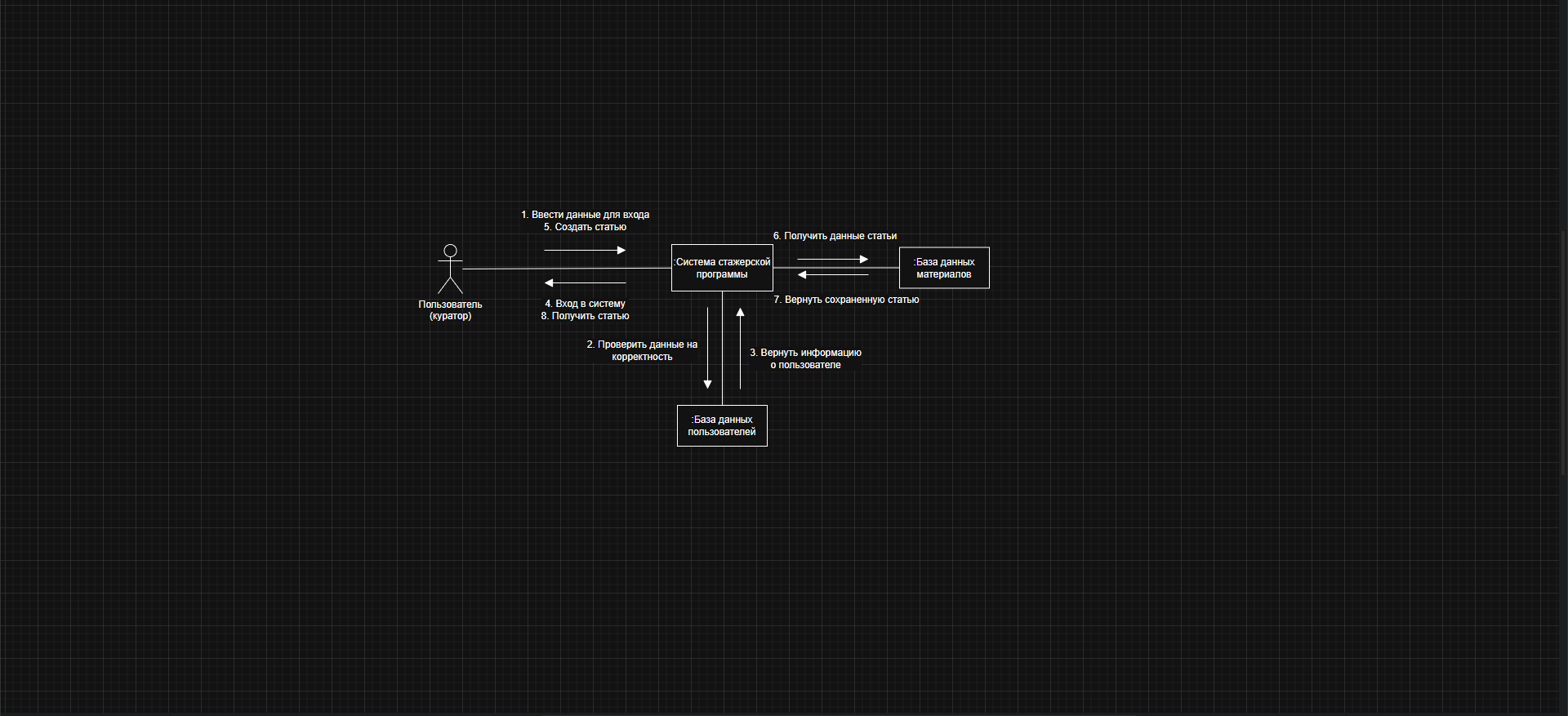
**Рисунок 14. Структура предприятия**

****

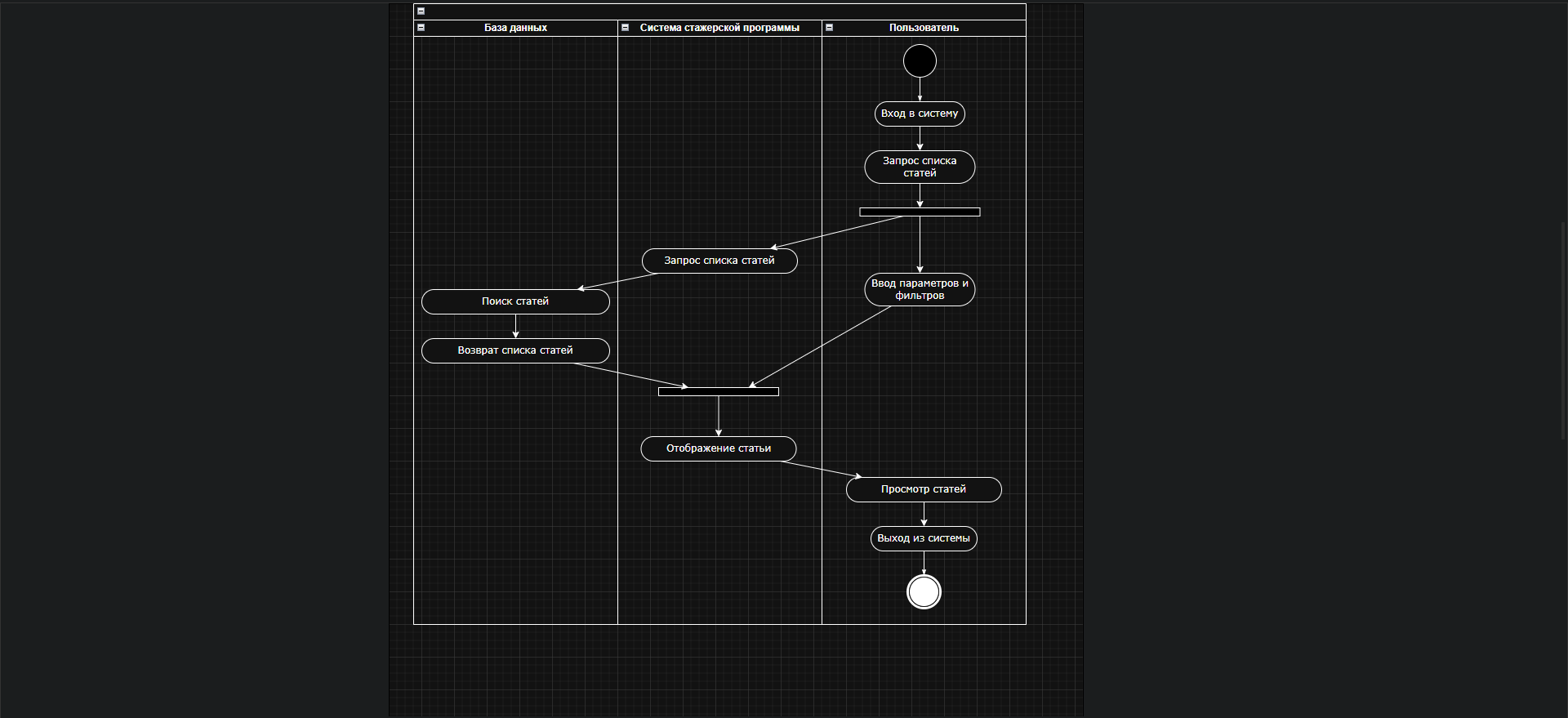
**Рисунок 15. Структура регионального предприятия**

## **ПРИЛОЖЕНИЕ Б**

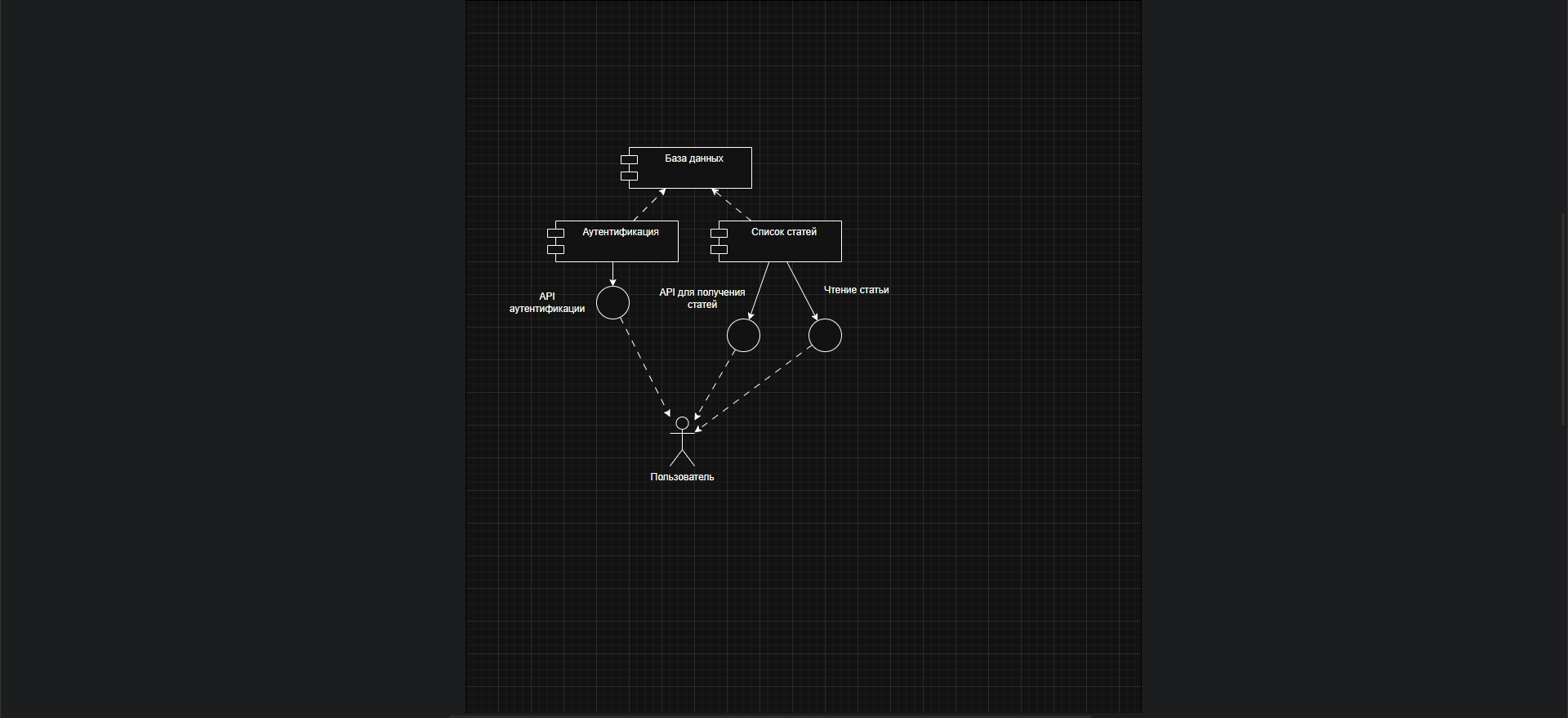
UML-диаграммы модулей программы

****

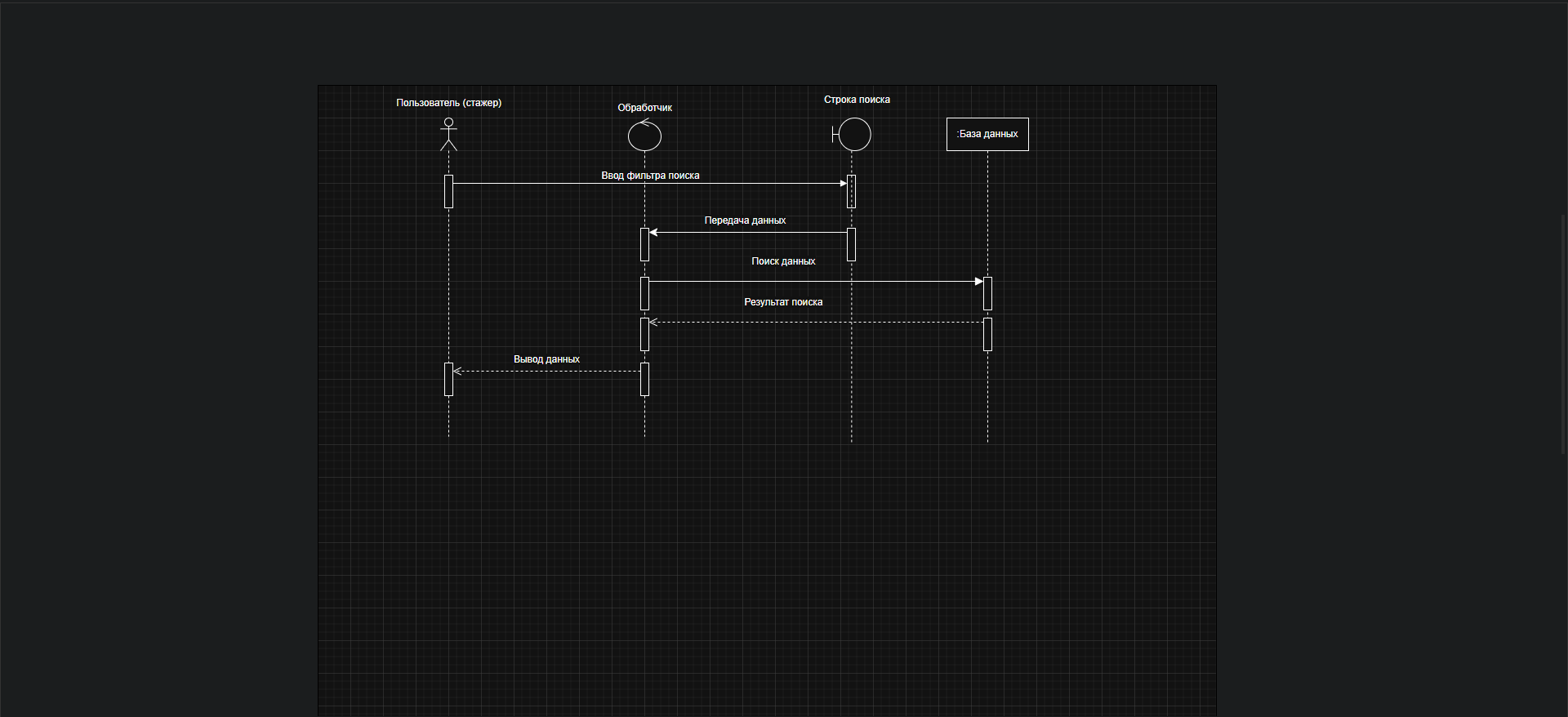
**Рисунок 16. Материалы - диаграмма коопераций**

****

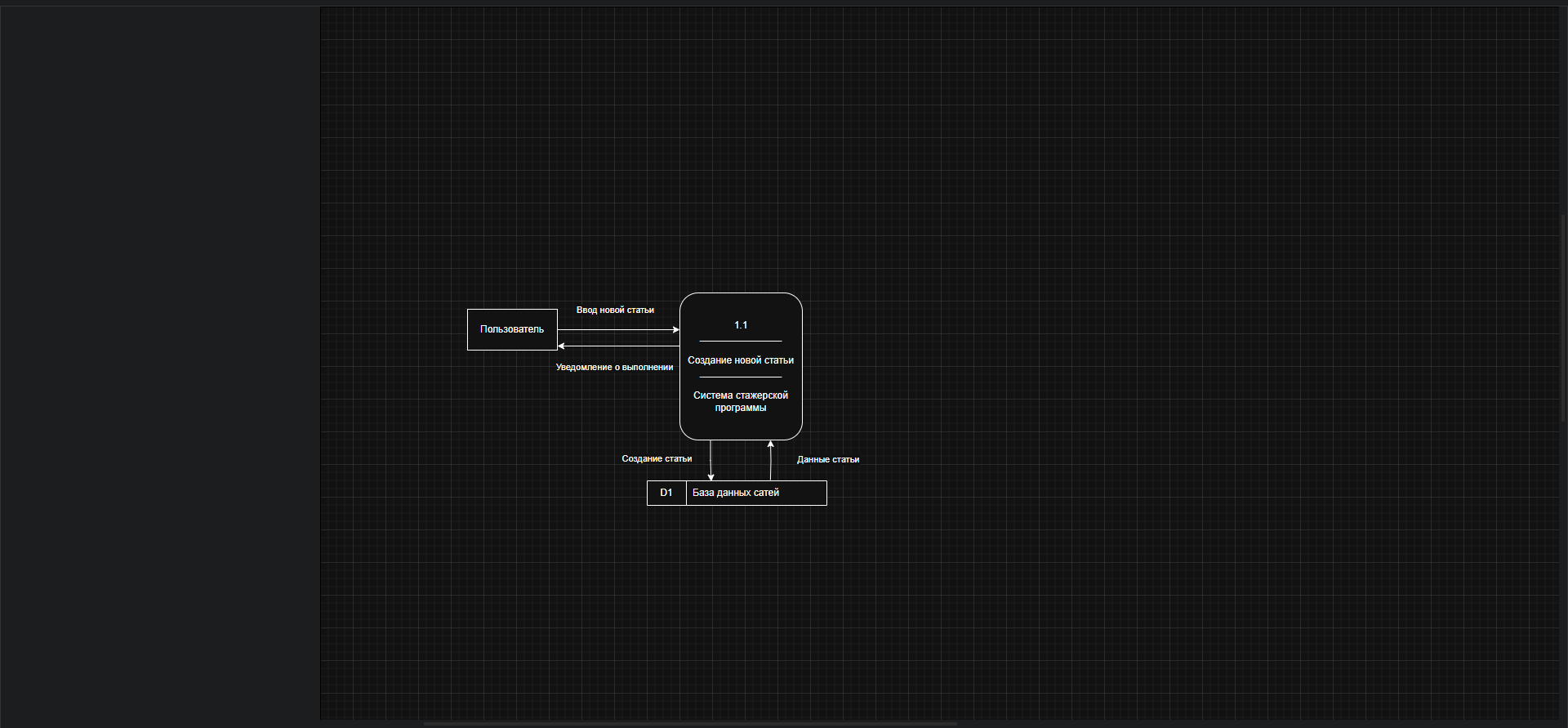
**Рисунок 17. Материалы - диаграмма деятельности**

****

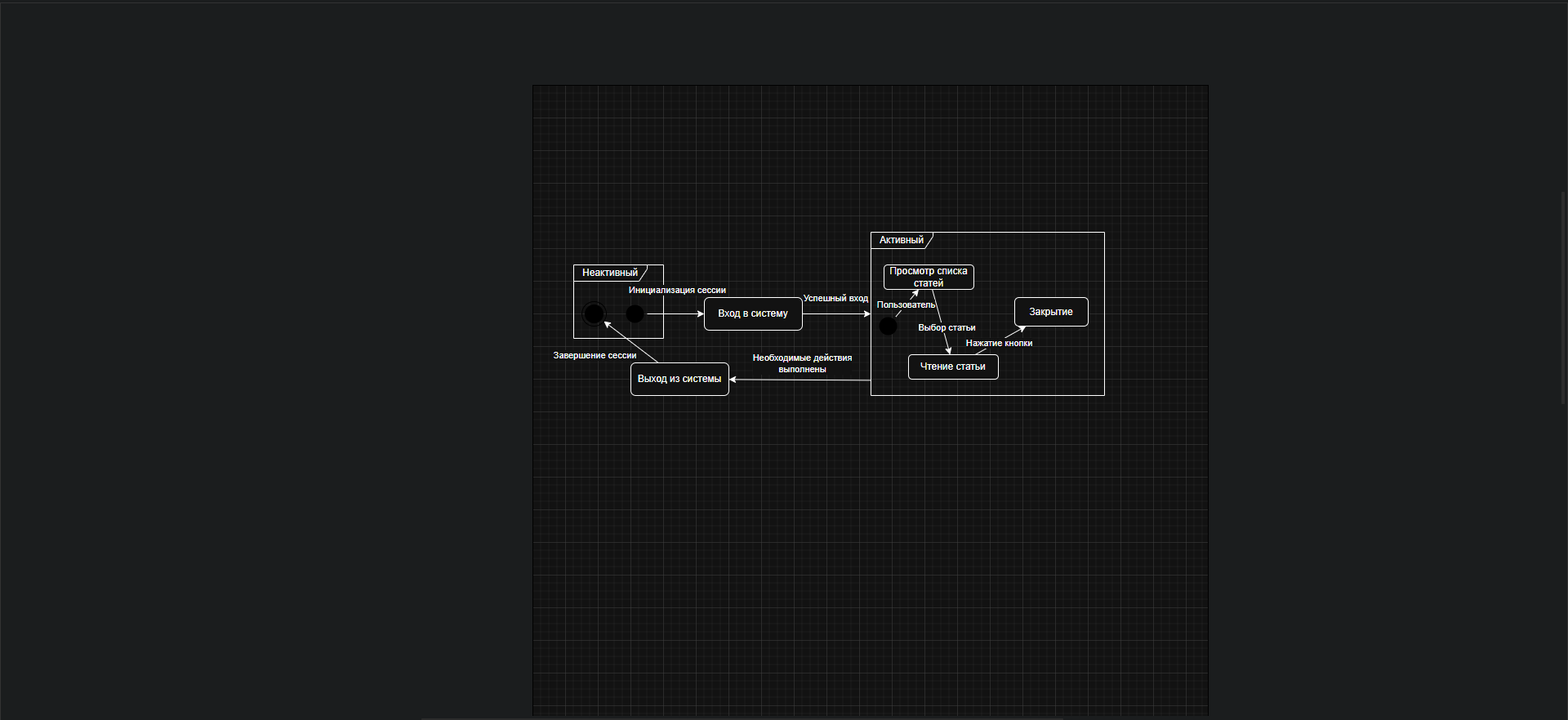
**Рисунок 18. Материалы - диаграмма компонентов**

****

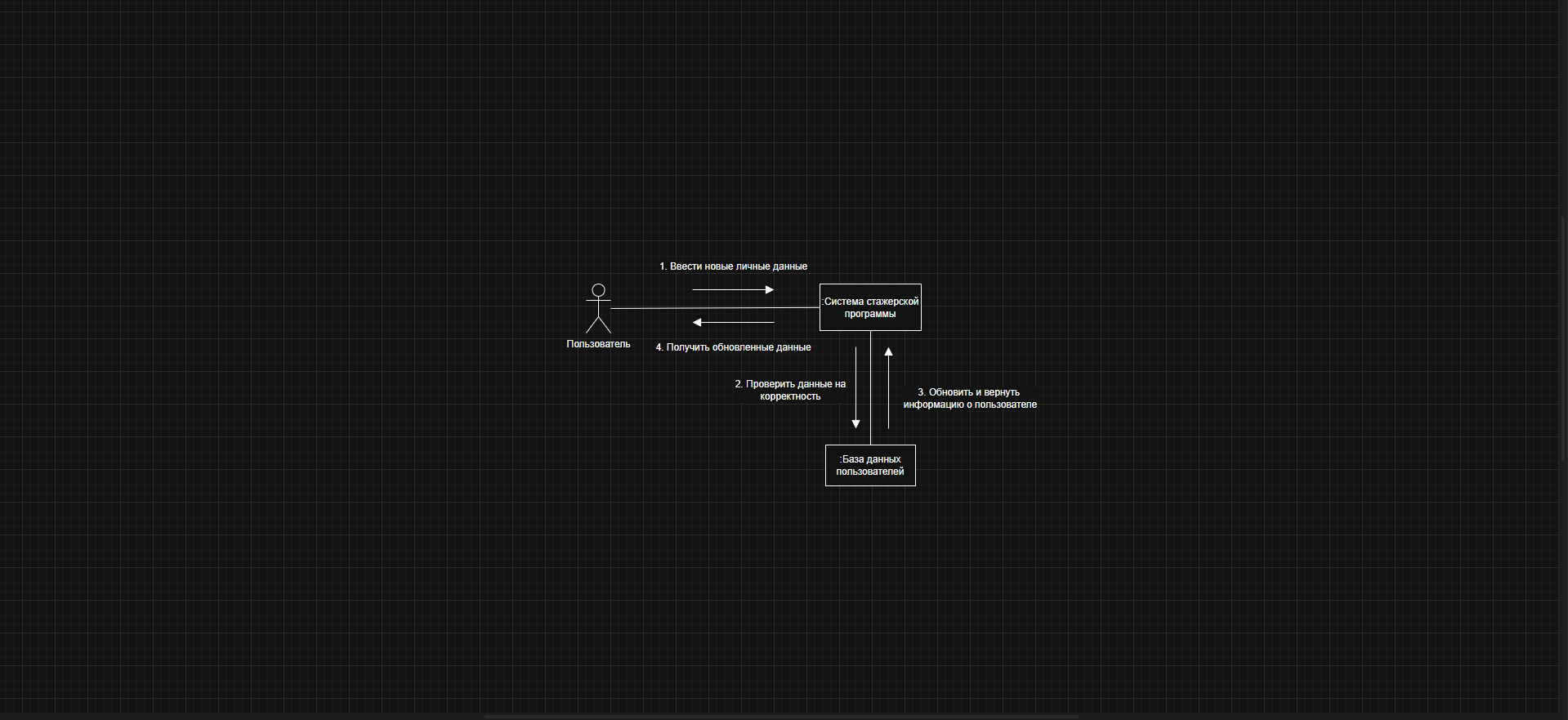
**Рисунок 19. Материалы - диаграмма последовательности**

****

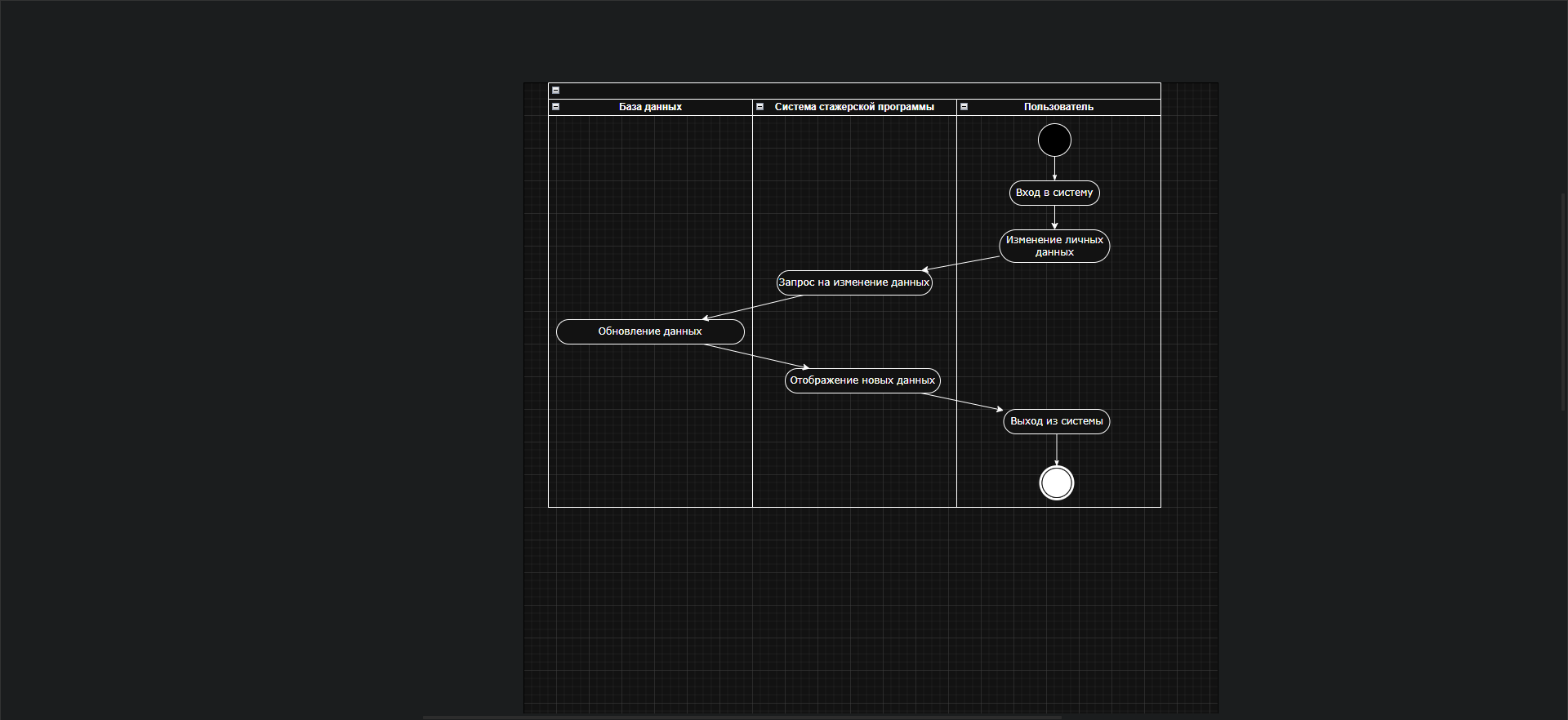
**Рисунок 20. Материалы - диаграмма потоков данных**

****

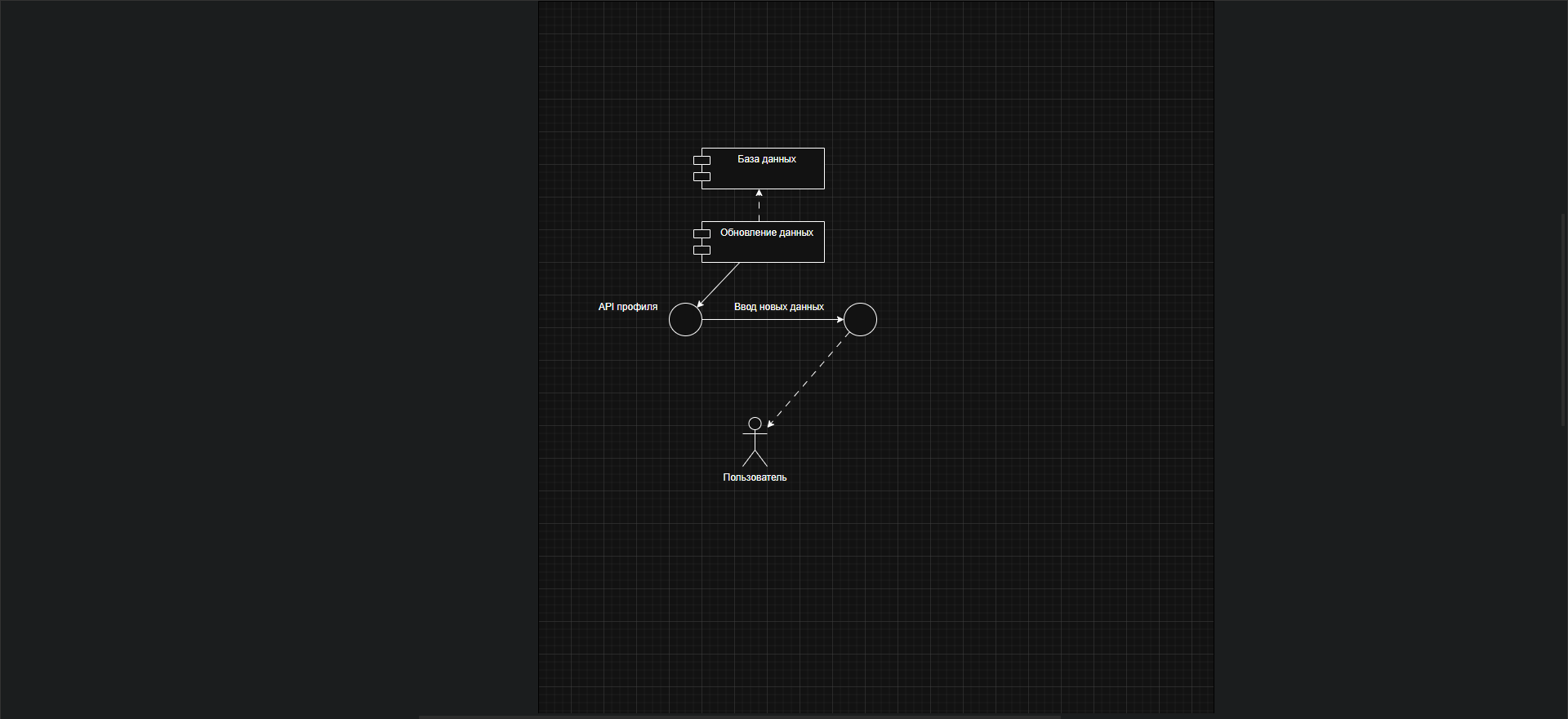
**Рисунок 21. Материалы - диаграмма состояния**

****

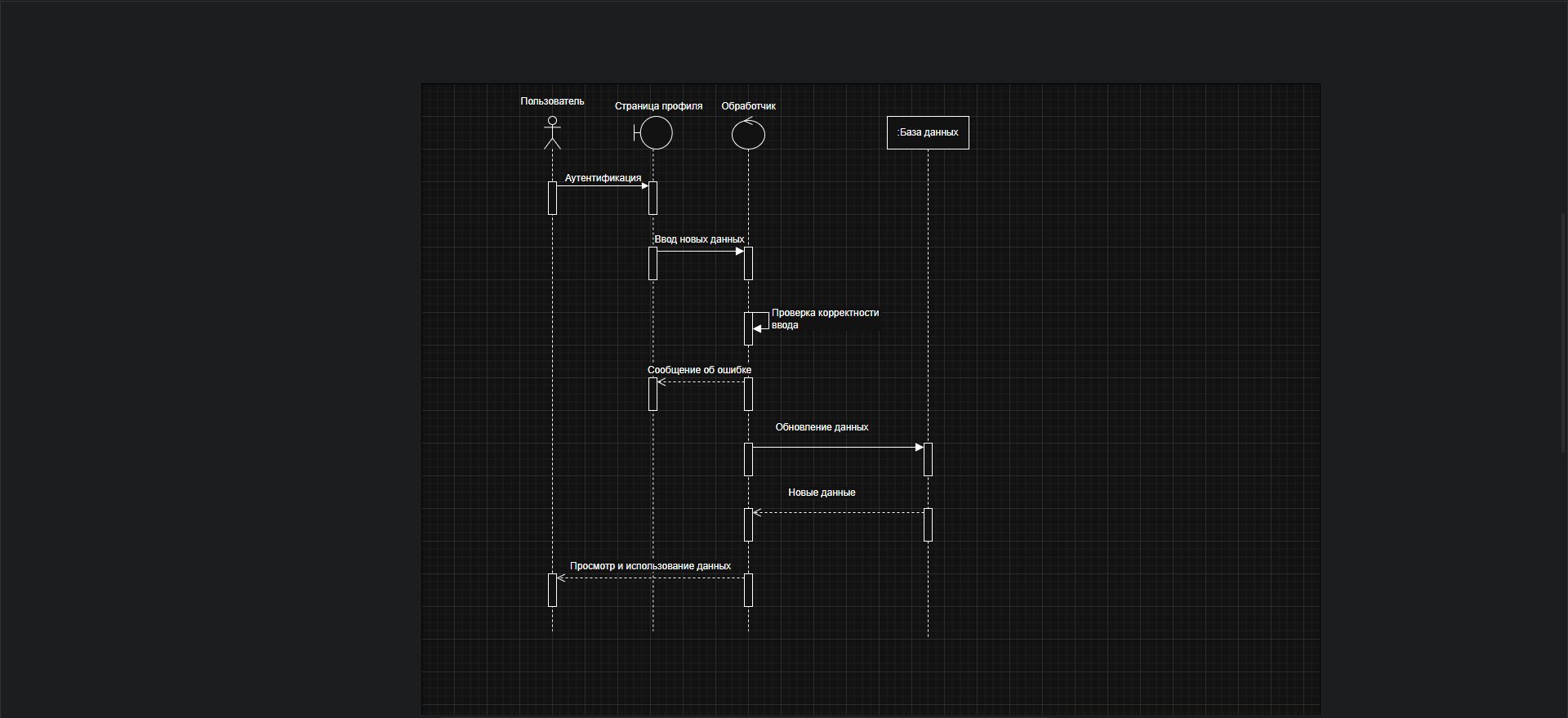
**Рисунок 22. Обновление данных - диаграмма кооперации**

****

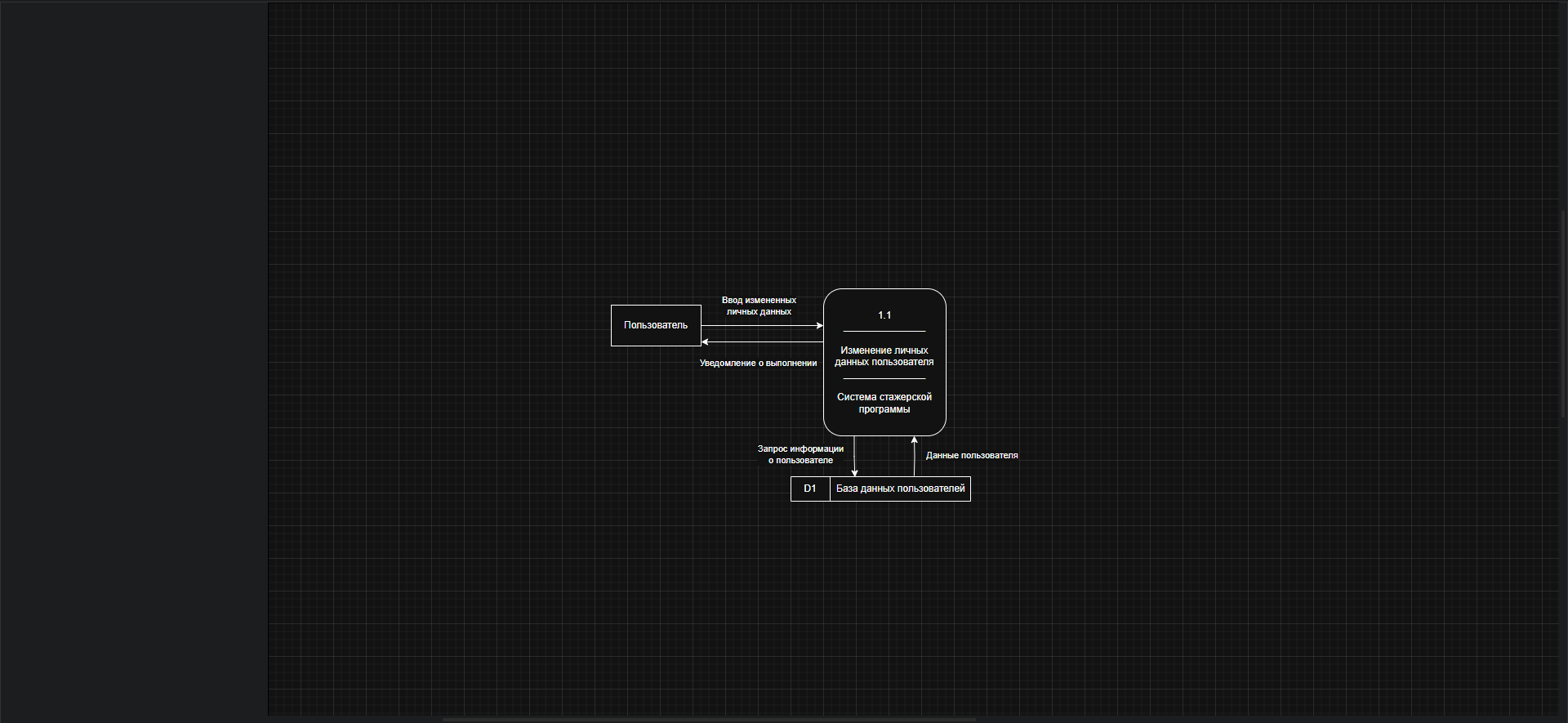
**Рисунок 23. Обновление данных - диаграмма деятельности**

****

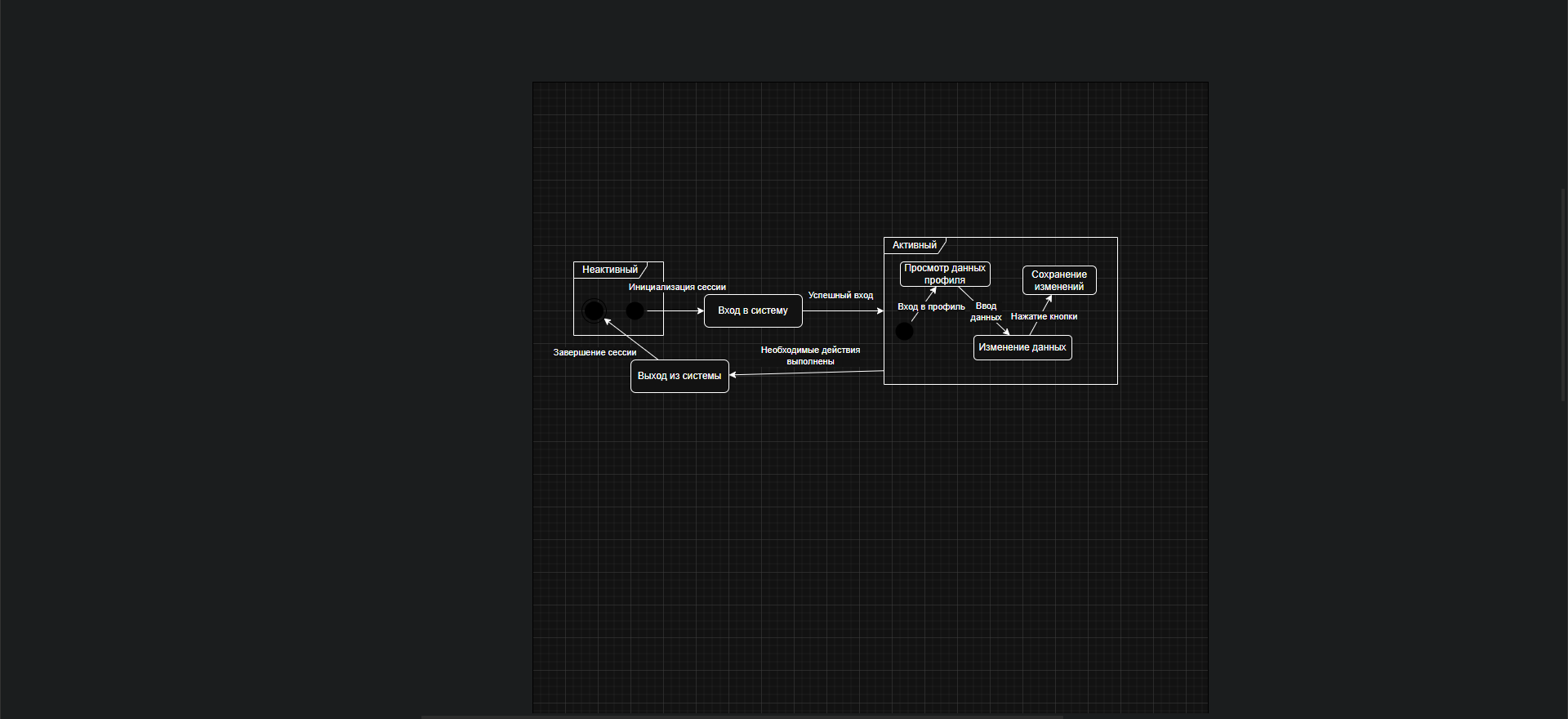
**Рисунок 24. Обновление данных - диаграмма компонентов**

****

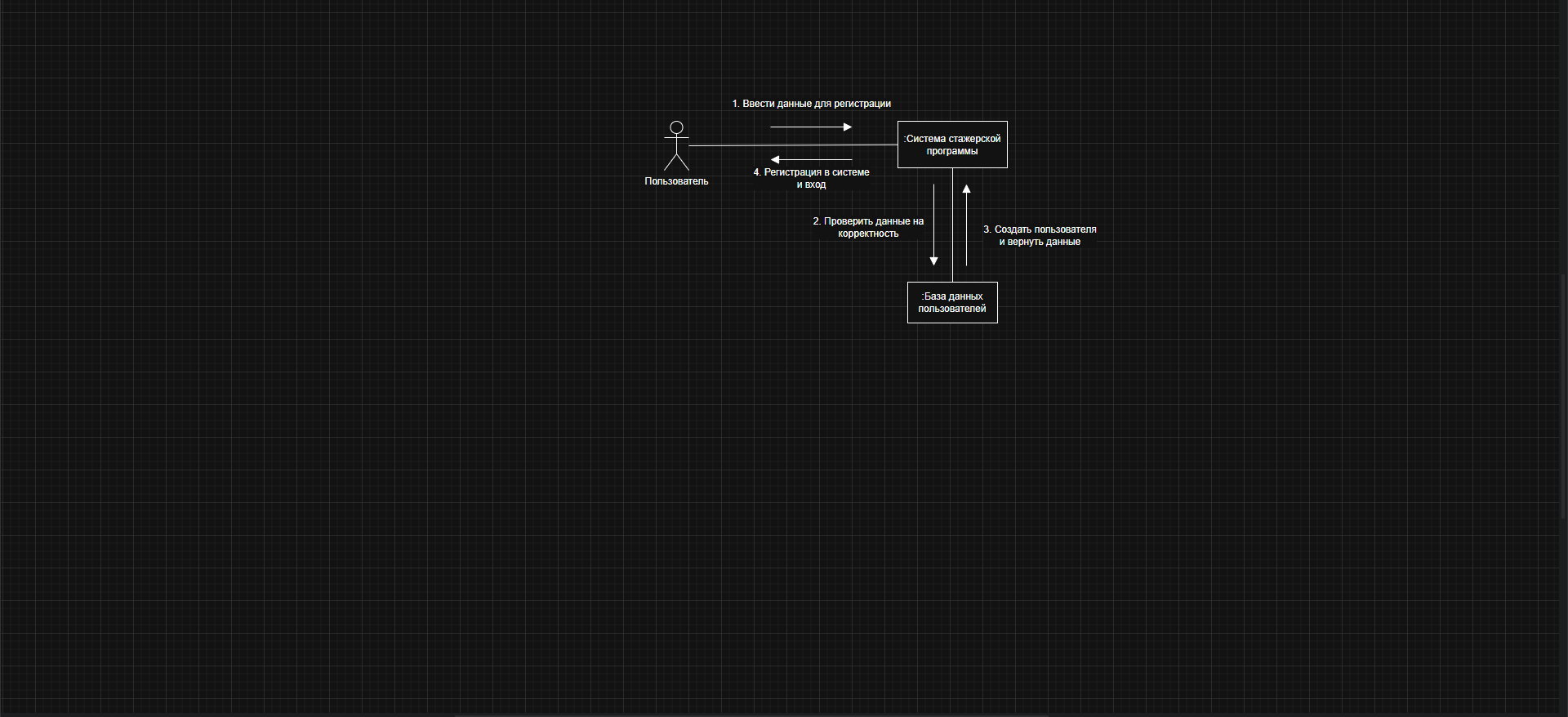
**Рисунок 25. Обновление данных - диаграмма последовательности**

****

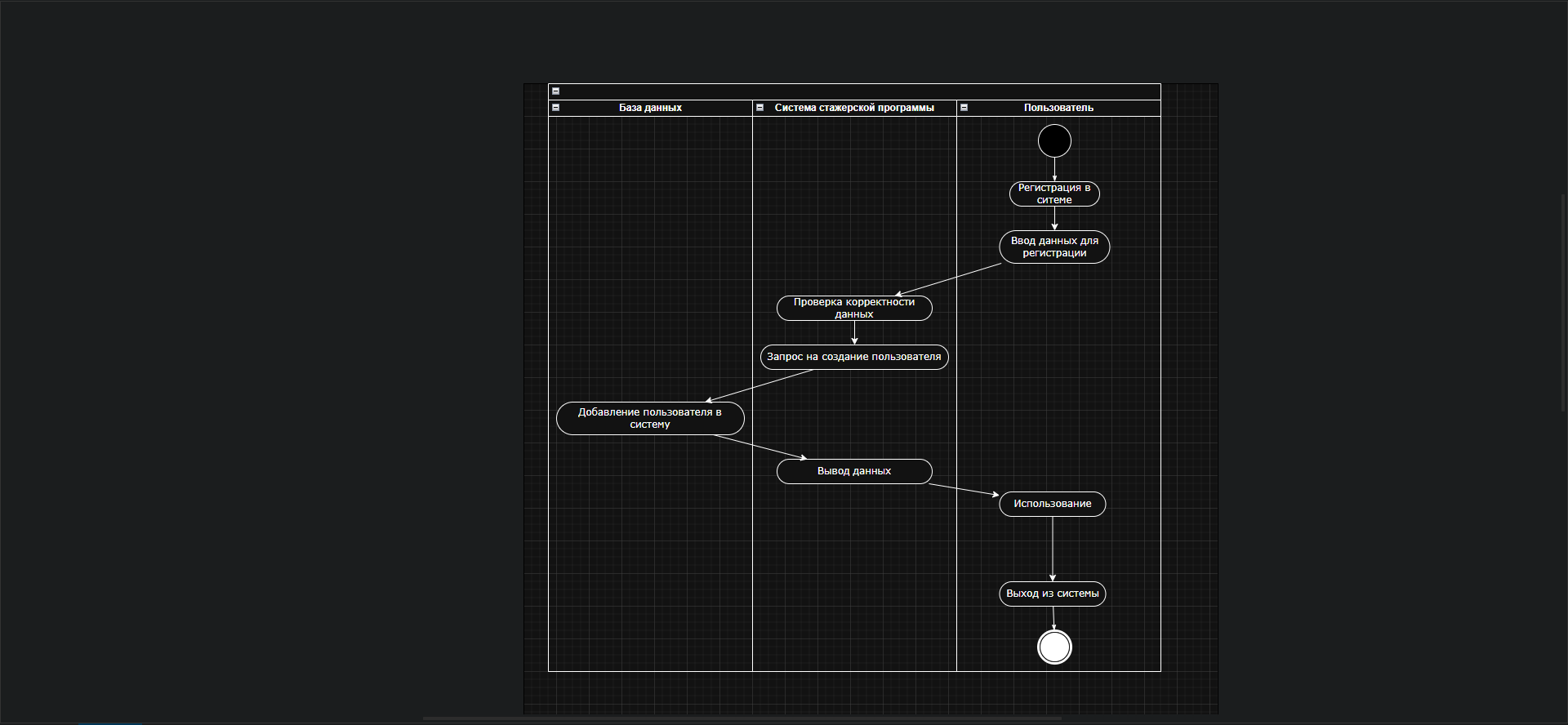
**Рисунок 26. Обновление данных - диаграмма потоков данных**

****

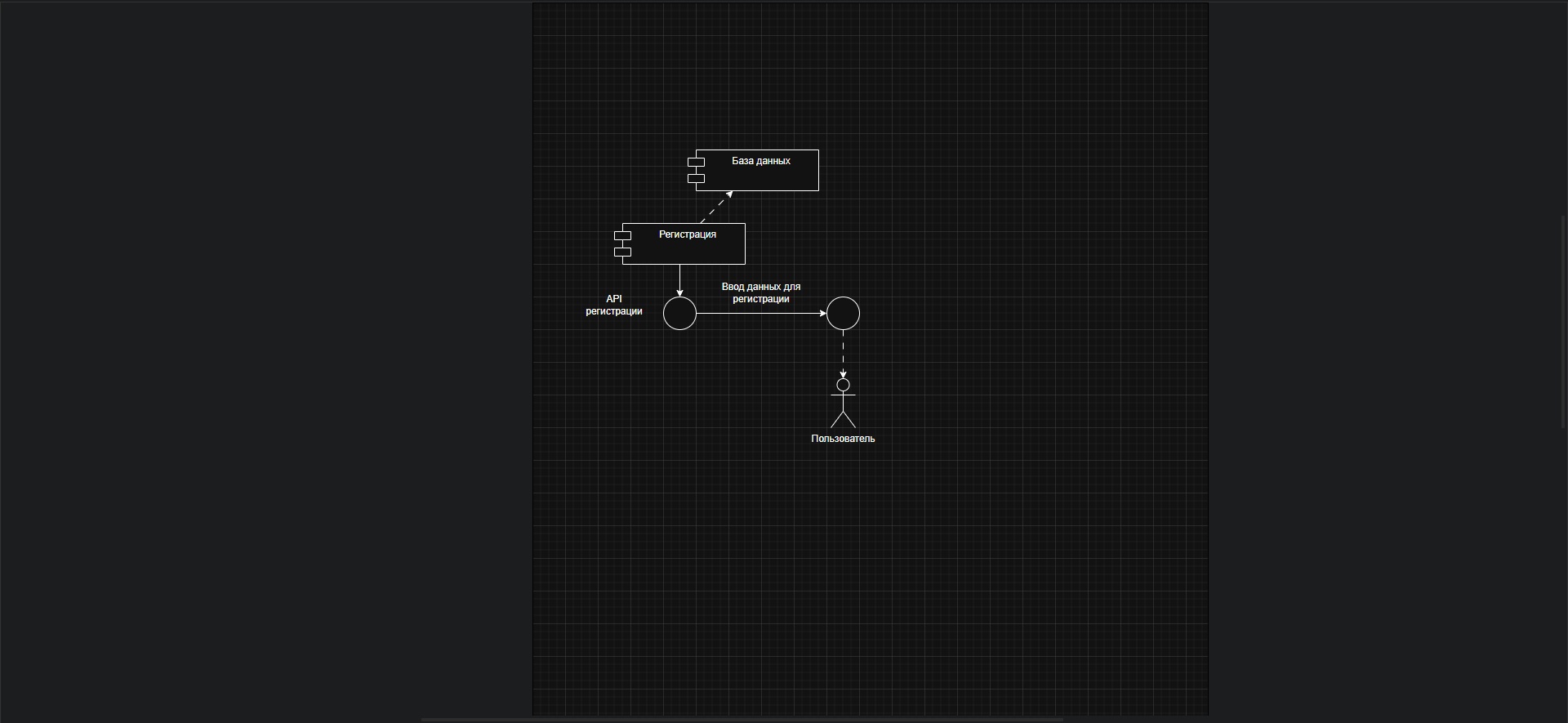
**Рисунок 27. Обновление данных - диаграмма состояния**

****

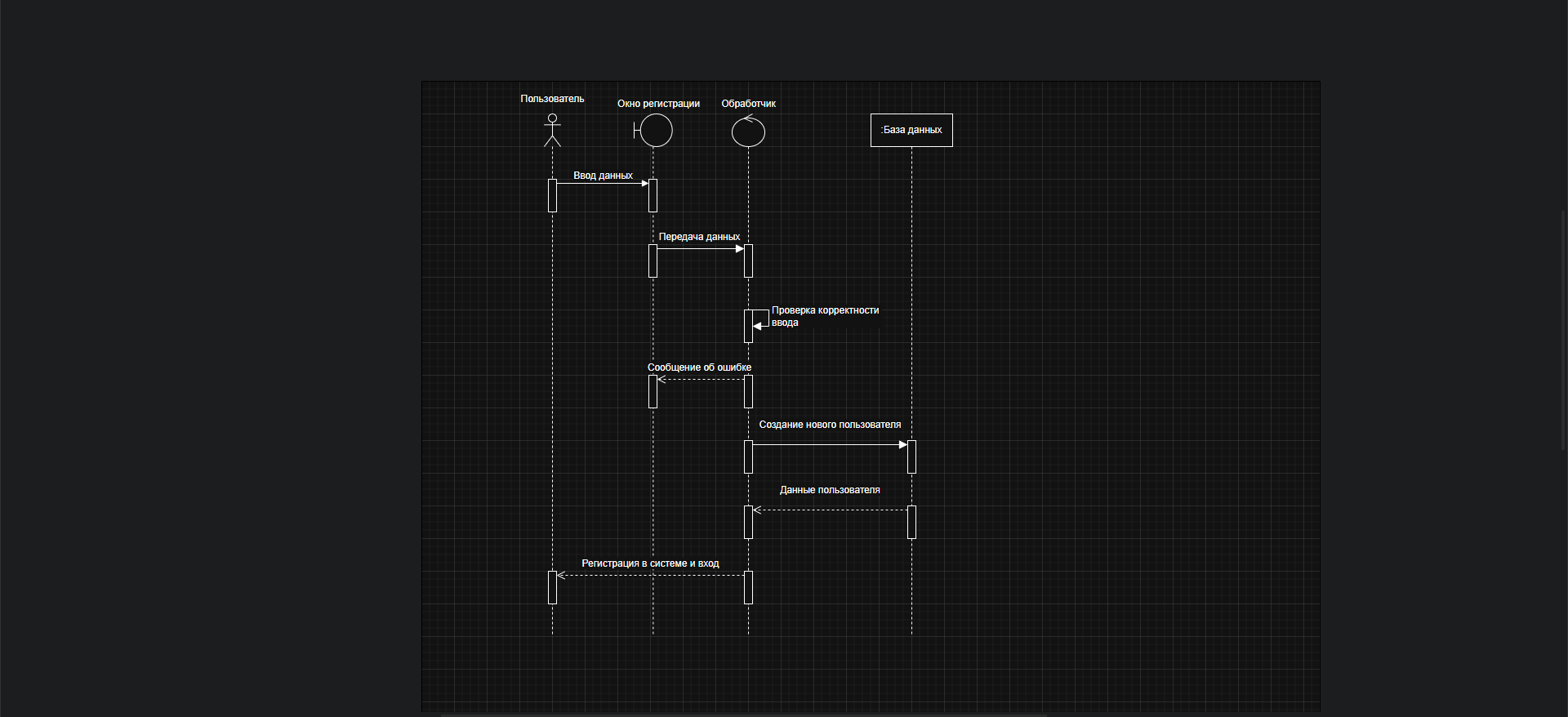
**Рисунок 28. Регистрация - диаграмма кооперации**

****

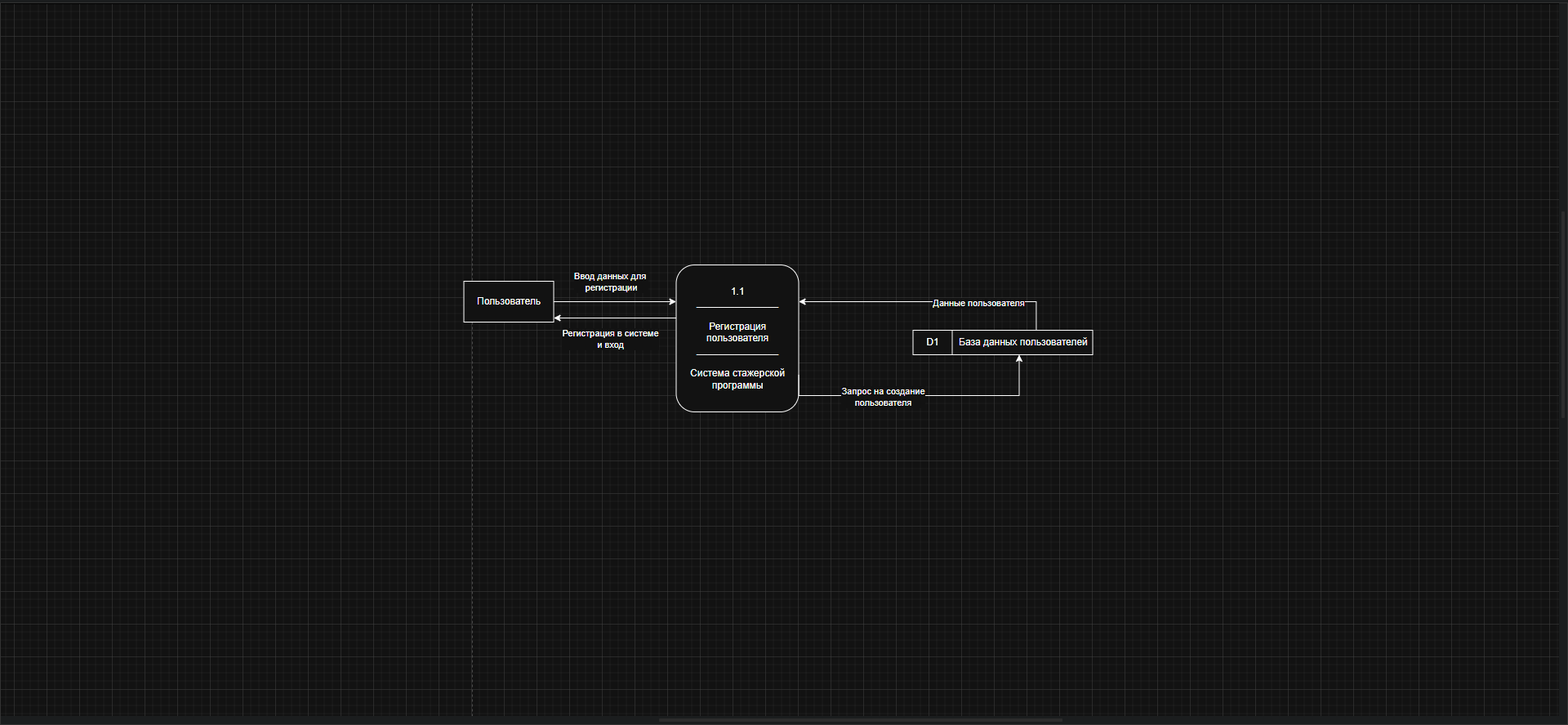
**Рисунок 29. Регистрация - диаграмма деятельности**

****

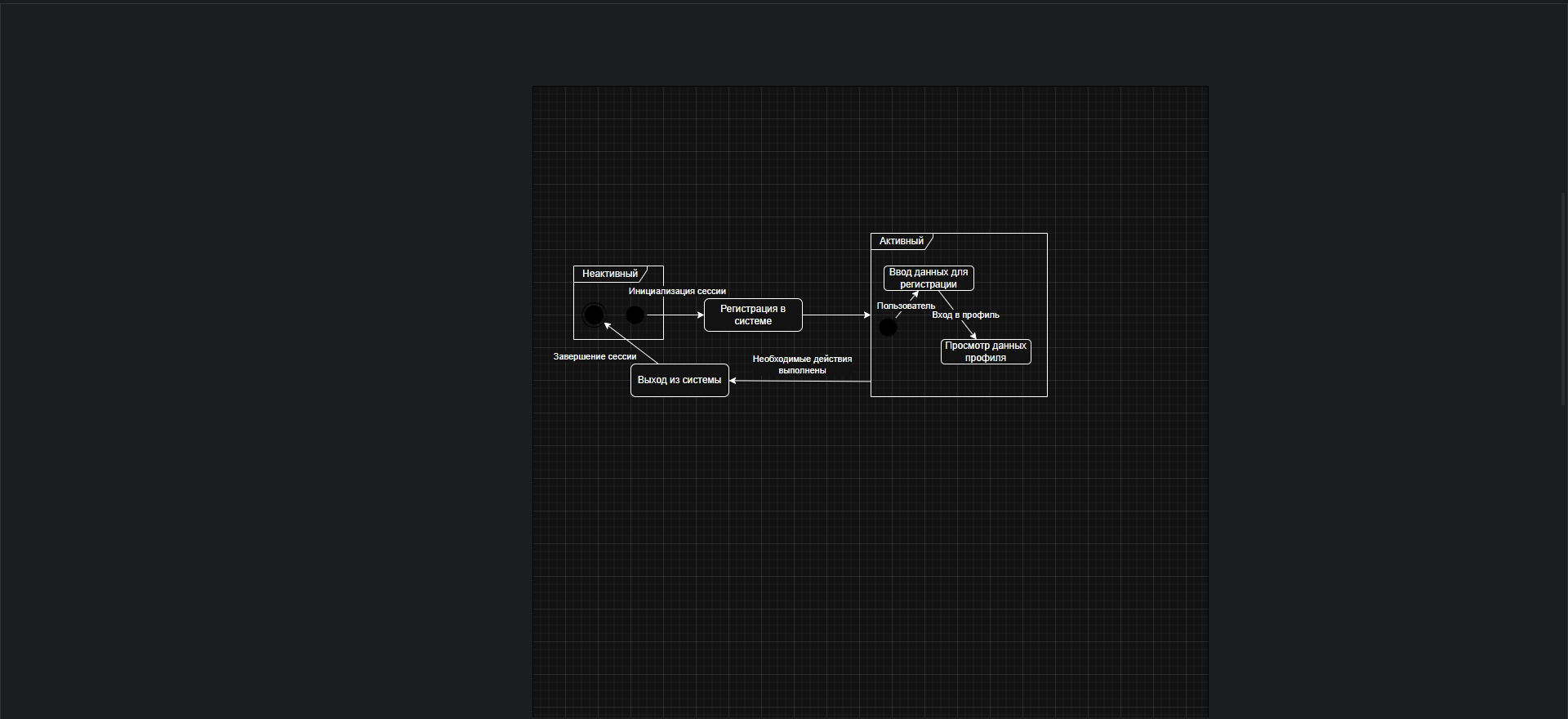
**Рисунок 30. Регистрация - диаграмма компонентов**

****

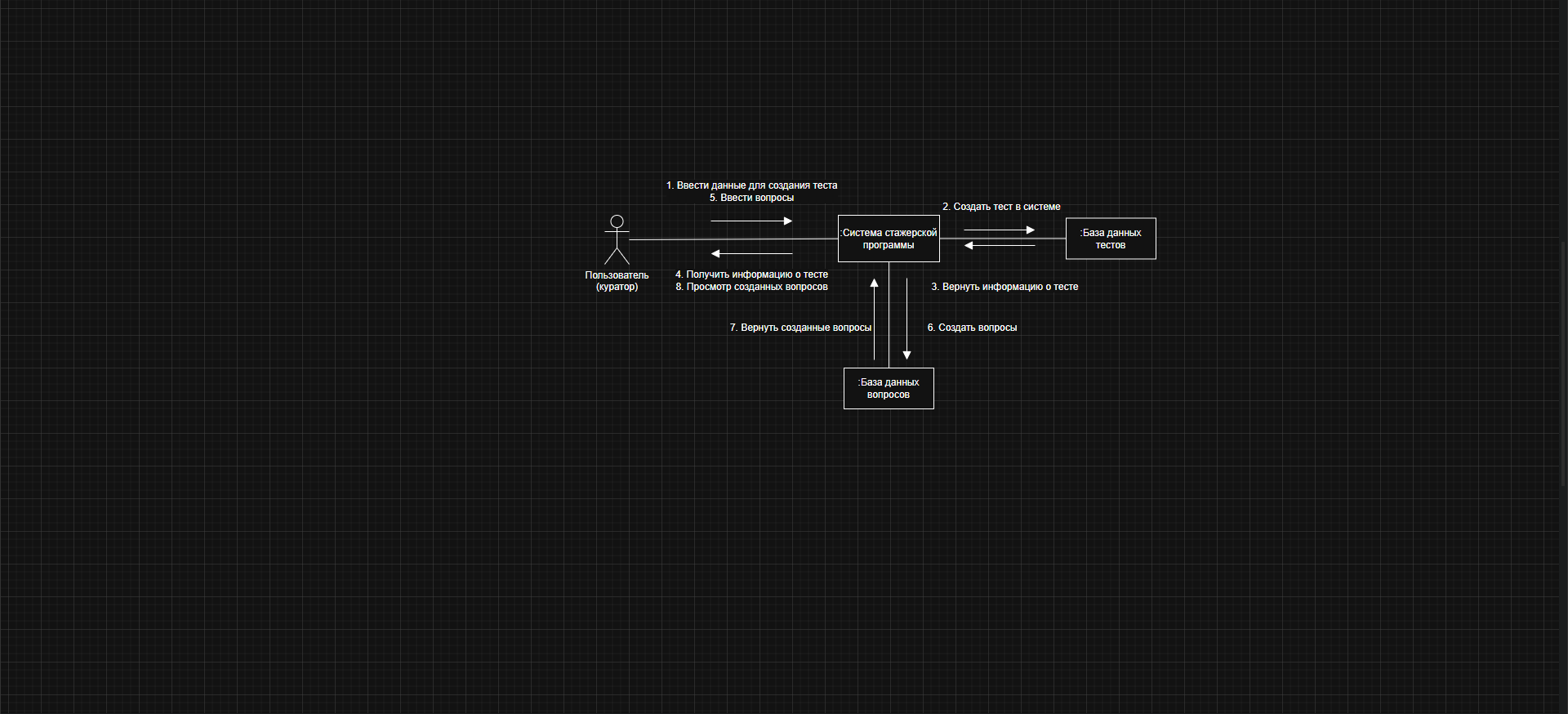
**Рисунок 31. Регистрация - диаграмма последовательности**

****

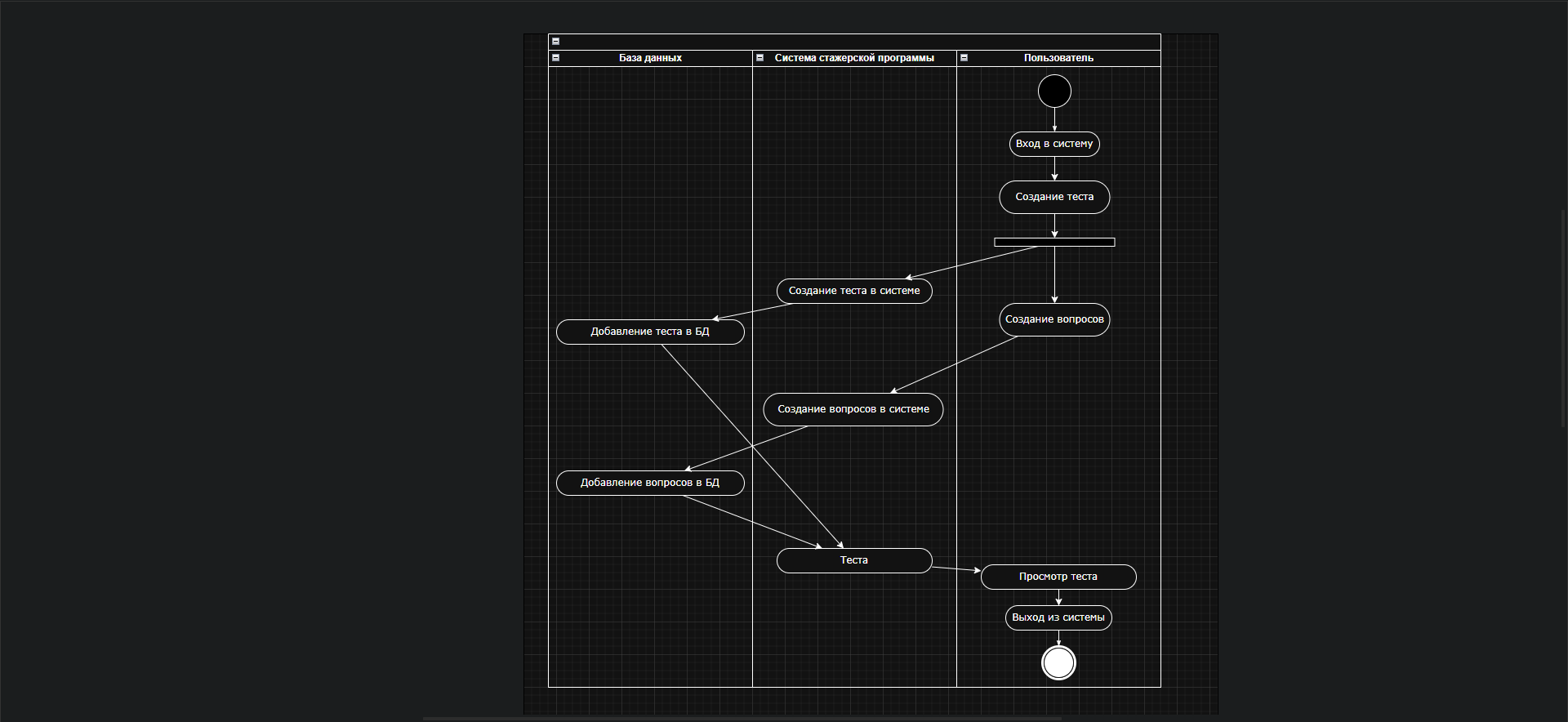
**Рисунок 32. Регистрация - диаграмма потоков данных**

****

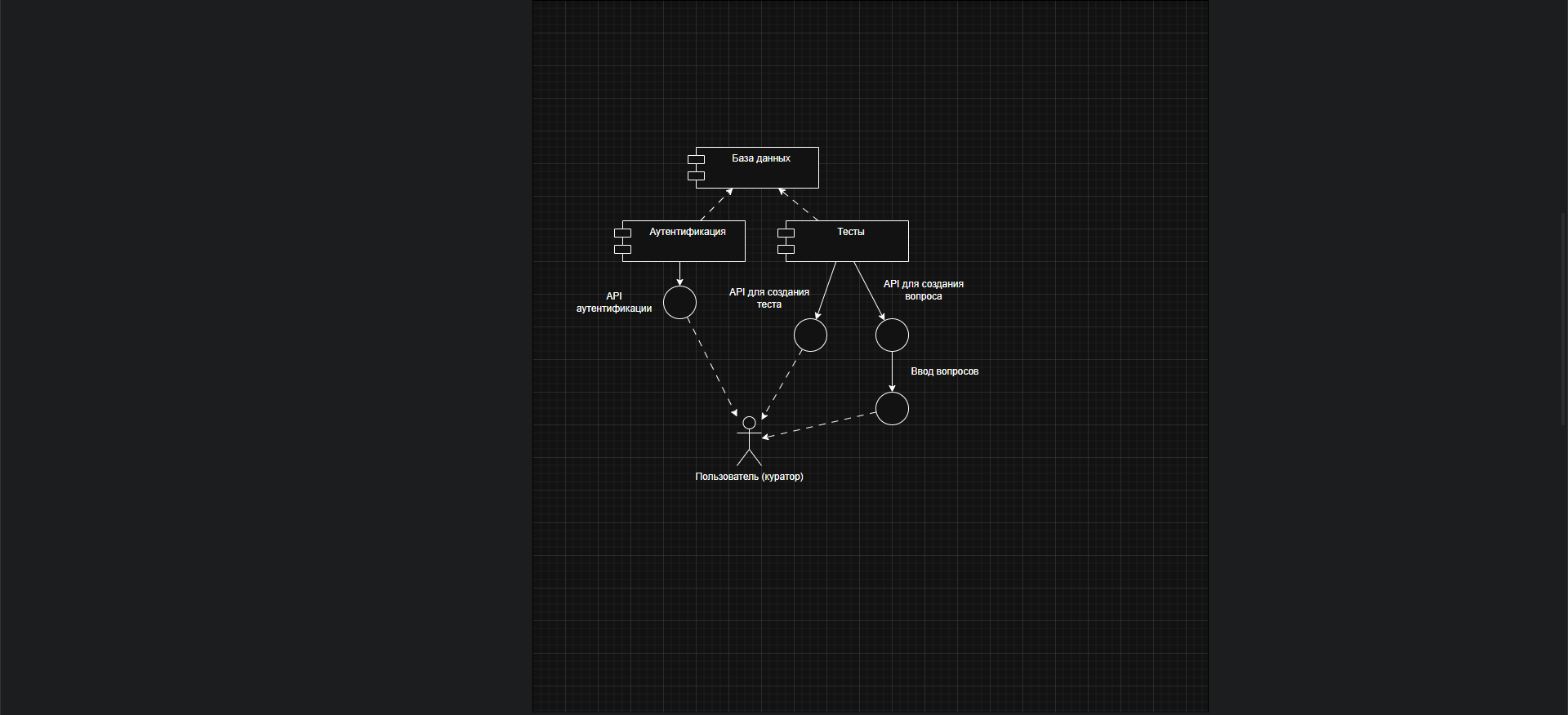
**Рисунок 33. Регистрация - диаграмма состояния**

****

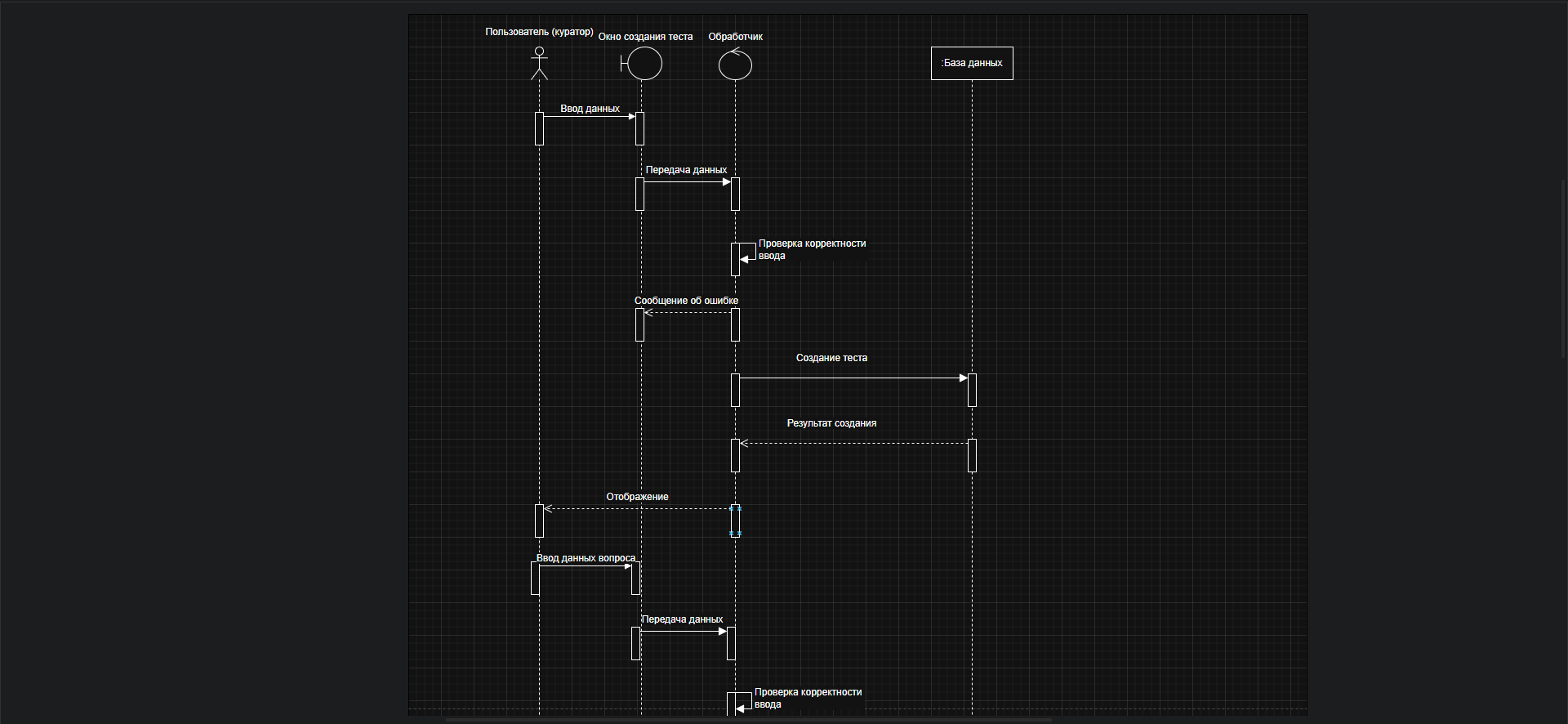
**Рисунок 34. Тесты - диаграмма кооперации**

****

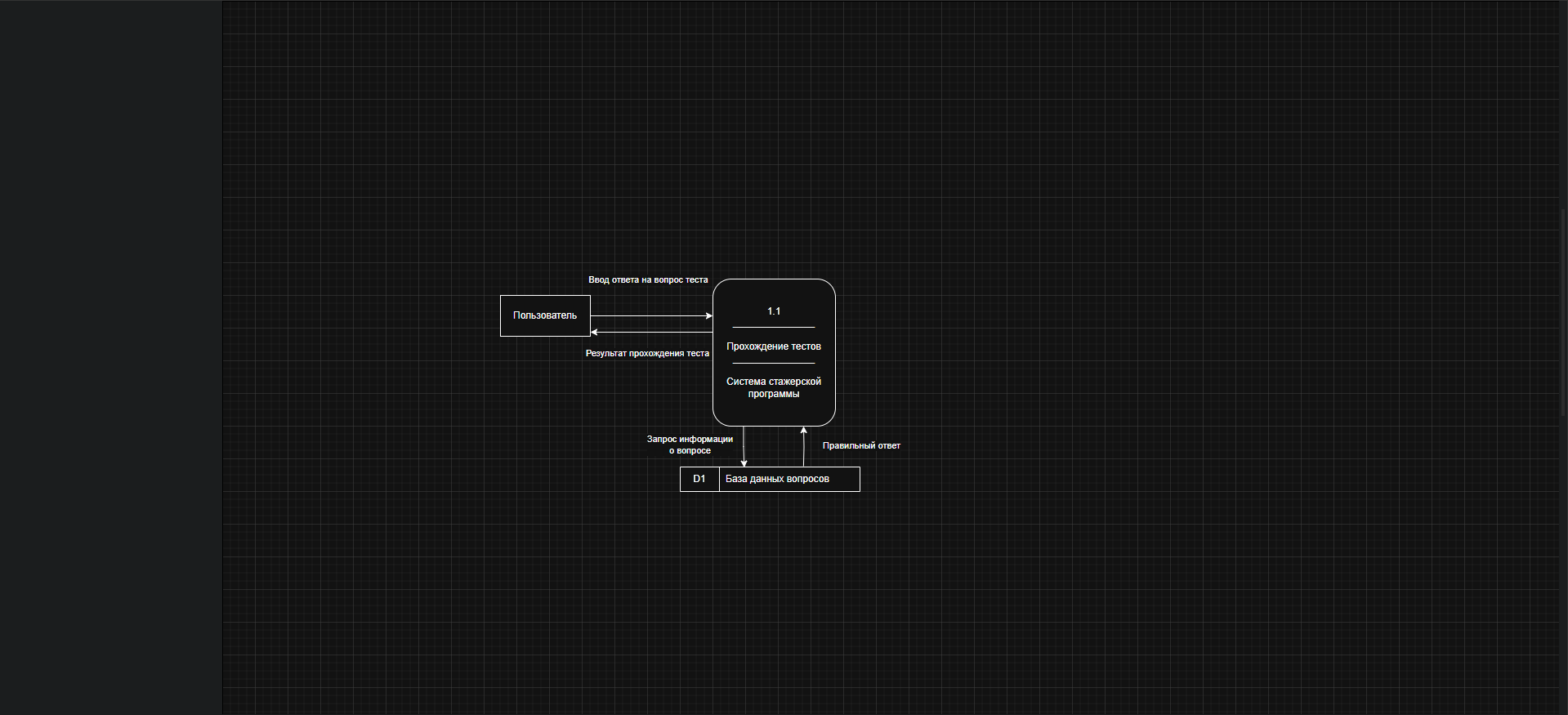
**Рисунок 35. Тесты - диаграмма деятельности**

****

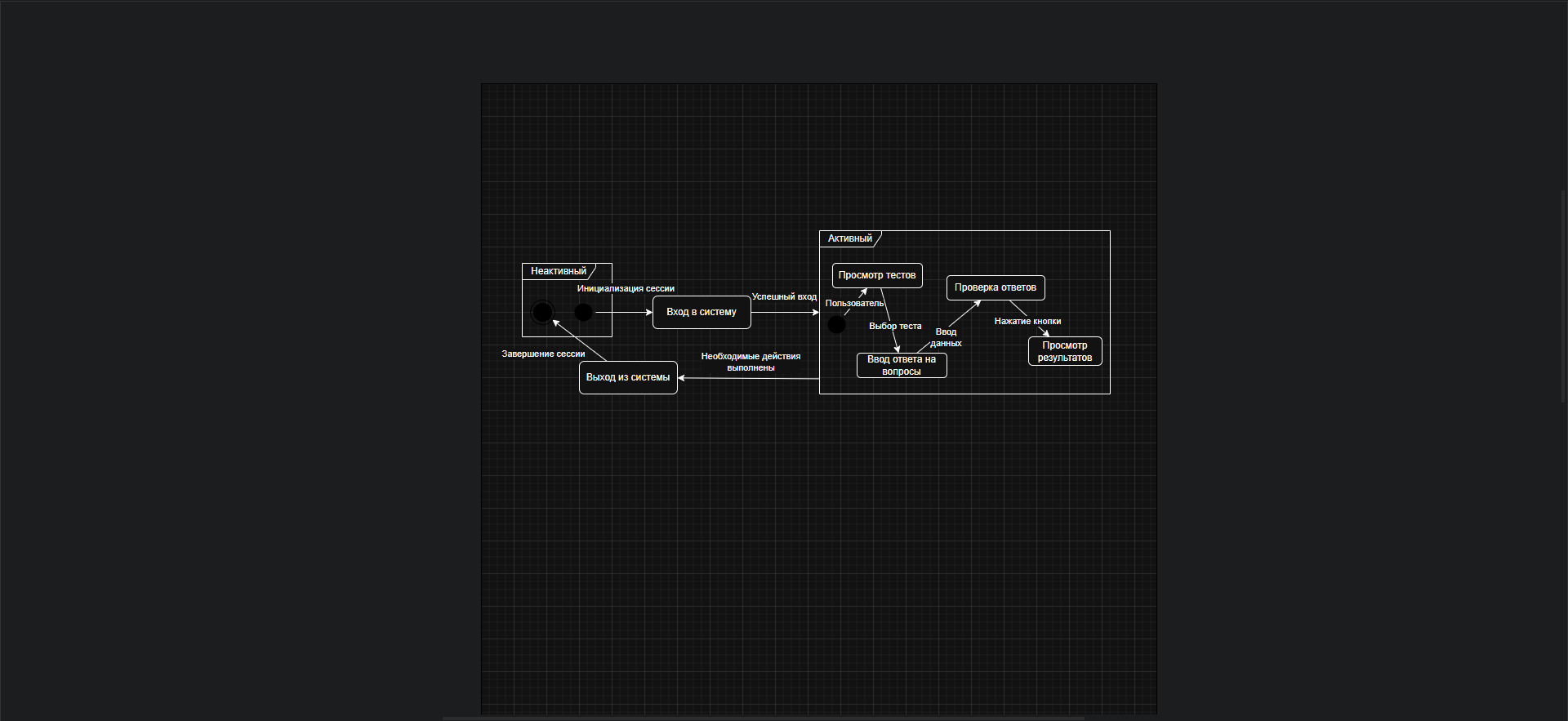
**Рисунок 36. Тесты - диаграмма компонентов**

****

**Рисунок 37. Тесты - диаграмма последовательности**

****

**Рисунок 38. Тесты - диаграмма потоков данных**

****

**Рисунок 39. Тесты - диаграмма состояния**

## **ПРИЛОЖЕНИЕ В**

**Код модулей программного продукта**

**Auth.py**

from flask import Blueprint, request, jsonify

from werkzeug.security import generate\_password\_hash, check\_password\_hash

import jwt

import datetime

from functools import wraps

from backend.models import User, Progress

from backend import db

from backend.config import Config

auth\_bp = Blueprint('auth', \_\_name\_\_)

def token\_required(f):

    @wraps(f)

    def decorated(\*args, \*\*kwargs):

        token = None

        if 'Authorization' in request.headers:

            auth\_header = request.headers['Authorization']

            if auth\_header.startswith('Bearer '):

                token = auth\_header.split(" ")[1]

        if not token:

            return jsonify({'message': 'Требуется токен авторизации'}), 401

        try:

            data = jwt.decode(token, Config.SECRET\_KEY, algorithms=["HS256"])

            current\_user = User.query.get(data['user\_id'])

            if not current\_user:

                raise ValueError('Пользователь не найден')

        except Exception as e:

            return jsonify({'message': f'Неверный токен: {str(e)}'}), 401

        return f(current\_user, \*args, \*\*kwargs)

    return decorated

@auth\_bp.route('/register', methods=['POST', 'OPTIONS'])

def register():

    if request.method == 'OPTIONS':

        return jsonify({}), 200

    try:

        data = request.get\_json()

        if not data:

            return jsonify({'message': 'Необходимо передать данные'}), 400

        required\_fields = ['name', 'surname', 'login', 'password']

        if not all(field in data for field in required\_fields):

            return jsonify({'message': 'Не все обязательные поля заполнены'}), 400

        if User.query.filter\_by(login=data['login']).first():

            return jsonify({'message': 'Пользователь с таким логином уже существует'}), 400

        hashed\_password = generate\_password\_hash(

            data['password'],

            method='pbkdf2:sha256',

            salt\_length=16

        )

        new\_user = User(

            name=data['name'],

            surname=data['surname'],

            login=data['login'],

            password=hashed\_password,

            role=data.get('role', 'trainee')

        )

        db.session.add(new\_user)

        db.session.commit()

        return jsonify({'message': 'Пользователь успешно зарегистрирован'}), 201

    except Exception as e:

        db.session.rollback()

        return jsonify({'message': f'Ошибка регистрации: {str(e)}'}), 500

@auth\_bp.route('/login', methods=['POST', 'OPTIONS'])

def login():

    if request.method == 'OPTIONS':

        return jsonify({}), 200

    data = request.get\_json()

    if not data or not data.get('login') or not data.get('password'):

        return jsonify({'message': 'Login and password required'}), 400

    user = User.query.filter\_by(login=data['login']).first()

    if not user or not check\_password\_hash(user.password, data['password']):

        return jsonify({'message': 'Invalid credentials'}), 401

    token = jwt.encode({

        'user\_id': user.user\_id,

        'exp': datetime.datetime.utcnow() + datetime.timedelta(hours=24)

    }, Config.SECRET\_KEY, algorithm="HS256")

    return jsonify({

        'token': token,

        'user': {

            'user\_id': user.user\_id,

            'name': user.name,

            'surname': user.surname,

            'role': user.role,

            'login': user.login

        }

    }), 200

@auth\_bp.route('/update-profile', methods=['POST'])

@token\_required

def update\_profile(current\_user):

    try:

        data = request.get\_json()

        if not data:

            return jsonify({'message': 'Необходимо передать данные'}), 400

        # Обновляем основные данные

        if 'name' in data:

            current\_user.name = data['name']

        if 'surname' in data:

            current\_user.surname = data['surname']

        if 'login' in data:

            current\_user.login = data['login']

        # Обновляем пароль, если он предоставлен

        if 'password' in data and data['password']:

            current\_user.password = generate\_password\_hash(

                data['password'],

                method='pbkdf2:sha256',

                salt\_length=16

            )

        db.session.commit()

        # Генерируем новый токен с обновленными данными пользователя

        token = jwt.encode({

            'user\_id': current\_user.user\_id,

            'exp': datetime.datetime.utcnow() + datetime.timedelta(hours=24)

        }, Config.SECRET\_KEY, algorithm="HS256")

        return jsonify({

            'message': 'Данные профиля успешно обновлены',

            'token': token,

            'user': {

                'user\_id': current\_user.user\_id,

                'name': current\_user.name,

                'surname': current\_user.surname,

                'role': current\_user.role,

                'login': current\_user.login,

                'password': current\_user.password

            }

        }), 200

    except Exception as e:

        db.session.rollback()

        return jsonify({'message': f'Ошибка обновления профиля: {str(e)}'}), 500

@auth\_bp.route('/delete-account', methods=['DELETE'])

@token\_required

def delete\_account(current\_user):

    try:

        # Удаляем все связанные данные пользователя (прогресс, материалы, тесты)

        # В зависимости от вашей бизнес-логики может потребоваться дополнительная обработка

        # Удаляем пользователя

        db.session.delete(current\_user)

        db.session.commit()

        return jsonify({'message': 'Аккаунт успешно удалён'}), 200

    except Exception as e:

        db.session.rollback()

        return jsonify({'message': f'Ошибка удаления аккаунта: {str(e)}'}), 500

@auth\_bp.route('/progress/stats', methods=['GET'])

@token\_required

def get\_progress\_stats(current\_user):

    try:

        # Подсчет пройденных материалов

        completed\_materials = db.session.query(Progress).filter(

            Progress.user\_id == current\_user.user\_id,

            Progress.material\_id.isnot(None),

            Progress.status == 'completed'

        ).count()

        # Подсчет пройденных тестов

        completed\_tests = db.session.query(Progress).filter(

            Progress.user\_id == current\_user.user\_id,

            Progress.test\_id.isnot(None),

            Progress.status == 'completed'

        ).count()

        return jsonify({

            'completedMaterials': completed\_materials,

            'completedTests': completed\_tests

        }), 200

    except Exception as e:

        return jsonify({'message': f'Ошибка получения статистики: {str(e)}'}), 500

# Добавим в auth.py новые маршруты для администратора

@auth\_bp.route('/admin/users', methods=['GET', 'POST'])

@token\_required

def admin\_users(current\_user):

    if current\_user.role != 'admin':

        return jsonify({'message': 'Доступ запрещен'}), 403

    if request.method == 'GET':

        # Получение списка пользователей

        search = request.args.get('search', '')

        query = User.query

        if search:

            search = f'%{search}%'

            query = query.filter(

                (User.name.ilike(search)) |

                (User.surname.ilike(search)) |

                (User.login.ilike(search))

            )

        users = query.order\_by(User.user\_id).all()

        users\_data = []

        for user in users:

            users\_data.append({

                'user\_id': user.user\_id,

                'name': user.name,

                'surname': user.surname,

                'login': user.login,

                'role': user.role

            })

        return jsonify(users\_data), 200

    elif request.method == 'POST':

        # Создание нового пользователя

        try:

            data = request.get\_json()

            required\_fields = ['name', 'surname', 'login', 'password', 'role']

            if not all(field in data for field in required\_fields):

                return jsonify({'message': 'Не все обязательные поля заполнены'}), 400

            if User.query.filter\_by(login=data['login']).first():

                return jsonify({'message': 'Пользователь с таким логином уже существует'}), 400

            hashed\_password = generate\_password\_hash(

                data['password'],

                method='pbkdf2:sha256',

                salt\_length=16

            )

            new\_user = User(

                name=data['name'],

                surname=data['surname'],

                login=data['login'],

                password=hashed\_password,

                role=data['role']

            )

            db.session.add(new\_user)

            db.session.commit()

            return jsonify({

                'message': 'Пользователь успешно создан',

                'user\_id': new\_user.user\_id

            }), 201

        except Exception as e:

            db.session.rollback()

            return jsonify({'message': f'Ошибка создания пользователя: {str(e)}'}), 500

@auth\_bp.route('/admin/users/<int:user\_id>', methods=['GET', 'PUT', 'DELETE'])

@token\_required

def admin\_user(current\_user, user\_id):

    if current\_user.role != 'admin':

        return jsonify({'message': 'Доступ запрещен'}), 403

    user = User.query.get\_or\_404(user\_id)

    if request.method == 'GET':

        return jsonify({

        'user\_id': user.user\_id,

        'name': user.name,

        'surname': user.surname,

        'login': user.login,

        'password': '\*\*\*\*\*\*\*\*',

        'role': user.role,

    }), 200

    elif request.method == 'PUT':

        try:

            data = request.get\_json()

            if 'name' in data:

                user.name = data['name']

            if 'surname' in data:

                user.surname = data['surname']

            if 'login' in data:

                if User.query.filter(User.login == data['login'], User.user\_id != user\_id).first():

                    return jsonify({'message': 'Пользователь с таким логином уже существует'}), 400

                user.login = data['login']

            if 'password' in data and data['password']:

                user.password = generate\_password\_hash(

                    data['password'],

                    method='pbkdf2:sha256',

                    salt\_length=16

                )

            if 'role' in data:

                user.role = data['role']

            db.session.commit()

            return jsonify({'message': 'Пользователь успешно обновлен'}), 200

        except Exception as e:

            db.session.rollback()

            return jsonify({'message': f'Ошибка обновления пользователя: {str(e)}'}), 500

    elif request.method == 'DELETE':

        try:

            # Проверяем, что администратор не удаляет сам себя

            if user.user\_id == current\_user.user\_id:

                return jsonify({'message': 'Вы не можете удалить свой собственный аккаунт'}), 400

            db.session.delete(user)

            db.session.commit()

            return jsonify({'message': 'Пользователь успешно удален'}), 200

        except Exception as e:

            db.session.rollback()

            return jsonify({'message': f'Ошибка удаления пользователя: {str(e)}'}), 500

**Auth.js**

const BASE\_PATH = window.location.pathname.startsWith('/frontend')

    ? '/frontend'

    : '';

const API\_URL = 'http://localhost:5000/api'; // или другой порт, если изменили

const AUTH\_PATHS = {

    login: `${BASE\_PATH}/templates/auth/login.html`,

    home: `${BASE\_PATH}/templates/dashboard.html` // или другой главный путь

};

// Универсальная функция fetch с обработкой CORS

async function makeRequest(url, options = {}) {

    try {

        const response = await fetch(`${API\_URL}${url}`, {

            ...options,

            headers: {

                ...options.headers,

                'Content-Type': 'application/json'

            },

            credentials: 'include',

            mode: 'cors'

        });

        if (!response.ok) {

            const error = await response.json().catch(() => ({}));

            throw new Error(error.message || `HTTP error ${response.status}`);

        }

        return response.json();

    } catch (error) {

        console.error('Request failed:', error);

        throw error;

    }

}

const Auth = {

    checkAuthState() {

        const token = localStorage.getItem('token');

        const user = localStorage.getItem('user');

        if (!token || !user) {

            if (!window.location.pathname.includes('/auth/')) {

                window.location.href = AUTH\_PATHS.login;

            }

            return false;

        }

        return true;

    },

    async handleLogin(event) {

        event.preventDefault();

        const form = event.target;

        const credentials = {

            login: form.login.value.trim(),

            password: form.password.value.trim()

        };

        if (!credentials.login || !credentials.password) {

            alert('Заполните все поля');

            return;

        }

        try {

            const data = await makeRequest('/login', {

                method: 'POST',

                body: JSON.stringify(credentials)

            });

            localStorage.setItem('token', data.token);

            localStorage.setItem('user', JSON.stringify(data.user));

            window.location.href = AUTH\_PATHS.home;

        } catch (error) {

            console.error('Login error:', error);

            alert(error.message || 'Ошибка входа. Проверьте логин и пароль');

        }

    },

    async handleRegister(event) {

        event.preventDefault();

        const name = document.getElementById('name').value;

        const surname = document.getElementById('surname').value;

        const login = document.getElementById('login').value;

        const password = document.getElementById('password').value;

        const confirmPassword = document.getElementById('confirmPassword').value;

        if (password !== confirmPassword) {

            alert('Пароли не совпадают');

            return;

        }

        try {

            const response = await fetch(`${API\_URL}/register`, {

                method: 'POST',

                headers: {

                    'Content-Type': 'application/json'

                },

                body: JSON.stringify({

                    name,

                    surname,

                    login,

                    password,

                    role: 'trainee'

                }),

                credentials: 'include',

                mode: 'cors'

            });

            const data = await response.json();

            if (!response.ok) {

                throw new Error(data.message || 'Ошибка регистрации');

            }

            alert('Регистрация успешна! Теперь вы можете войти.');

            window.location.href = 'login.html';

        } catch (error) {

            console.error('Registration error:', error);

            alert(`Ошибка регистрации: ${error.message}`);

        }

    },

    logout() {

        localStorage.removeItem('token');

        localStorage.removeItem('user');

        window.location.href = AUTH\_PATHS.login;

    }

};

async function authFetch(url, options = {}) {

    const token = localStorage.getItem('token');

    if (!token) {

        window.location.href = AUTH\_PATHS.login;

        throw new Error('Not authenticated');

    }

    try {

        const response = await fetch(`${API\_URL}${url}`, {

            ...options,

            headers: {

                ...options.headers,

                'Authorization': `Bearer ${token}`,

                'Content-Type': 'application/json'

            },

            credentials: 'include'

        });

        // Обработка 401 Unauthorized

        if (response.status === 401) {

            localStorage.removeItem('token');

            localStorage.removeItem('user');

            window.location.href = AUTH\_PATHS.login;

            throw new Error('Session expired');

        }

        // Обработка других ошибок

        if (!response.ok) {

            const errorData = await response.json().catch(() => ({}));

            throw new Error(errorData.message || `HTTP error! status: ${response.status}`);

        }

        return response;

    } catch (error) {

        console.error('Fetch error:', error);

        throw error;

    }

}

window.Auth = Auth;

window.authFetch = authFetch;

// Инициализация

document.addEventListener('DOMContentLoaded', () => {

    Auth.checkAuthState();

    const loginForm = document.getElementById('loginForm');

    if (loginForm) {

        loginForm.addEventListener('submit', (e) => Auth.handleLogin(e));

    }

    const registerForm = document.getElementById('registerForm');

    if (registerForm) {

        registerForm.addEventListener('submit', (e) => Auth.handleRegister(e));

    }

    const logoutBtn = document.getElementById('logoutBtn');

    if (logoutBtn) {

        logoutBtn.addEventListener('click', (e) => {

            e.preventDefault();

            Auth.logout();

        });

    }

});

**Materials.py**

from flask import Blueprint, request, jsonify

from backend.models import Material, db

from backend.utils import token\_required, curator\_required

materials\_bp = Blueprint('materials', \_\_name\_\_)

@materials\_bp.route('/materials', methods=['GET'])

@token\_required

def get\_materials(current\_user):

    materials = Material.query.all()

    output = []

    for material in materials:

        material\_data = {

            'material\_id': material.material\_id,

            'title': material.title,

            'author': f"{material.author.name} {material.author.surname}",

            'category': material.category,

            'upload\_date': material.upload\_date.strftime('%Y-%m-%d'),

            'author\_id': material.author\_id

        }

        output.append(material\_data)

    return jsonify({'materials': output}), 200

@materials\_bp.route('/materials/<int:material\_id>', methods=['GET'])

@token\_required

def get\_material(current\_user, material\_id):

    material = Material.query.get\_or\_404(material\_id)

    material\_data = {

        'material\_id': material.material\_id,

        'title': material.title,

        'author': f"{material.author.name} {material.author.surname}",

        'content': material.content,

        'category': material.category,

        'upload\_date': material.upload\_date.strftime('%Y-%m-%d'),

        'file\_url': material.file\_url

    }

    return jsonify({'material': material\_data}), 200

@materials\_bp.route('/materials', methods=['POST'])

@token\_required

@curator\_required

def create\_material(current\_user):

    data = request.get\_json()

    new\_material = Material(

        title=data['title'],

        author\_id=current\_user.user\_id,

        content=data.get('content'),

        category=data.get('category'),

        file\_url=data.get('file\_url')

    )

    db.session.add(new\_material)

    db.session.commit()

    return jsonify({'message': 'Material created!'}), 201

@materials\_bp.route('/materials/<int:material\_id>', methods=['PUT'])

@token\_required

@curator\_required

def update\_material(current\_user, material\_id):

    try:

        # Получаем JSON данные

        data = request.get\_json()

        # Проверяем наличие обязательных полей

        if not data or 'title' not in data or not data['title']:

            return jsonify({'message': 'Название обязательно'}), 400

        material = Material.query.get\_or\_404(material\_id)

        # Проверка прав доступа

        if material.author\_id != current\_user.user\_id and current\_user.role != 'admin':

            return jsonify({'message': 'Недостаточно прав для редактирования'}), 403

        # Обновляем материал

        material.title = data['title']

        material.content = data.get('content', material.content)

        material.category = data.get('category', material.category)

        db.session.commit()

        return jsonify({

            'message': 'Материал успешно обновлен',

            'material\_id': material.material\_id

        }), 200

    except Exception as e:

        db.session.rollback()

        print(f"Ошибка при обновлении: {str(e)}")

        return jsonify({'message': f'Ошибка сервера: {str(e)}'}), 500

@materials\_bp.route('/materials/<int:material\_id>', methods=['DELETE'])

@token\_required

@curator\_required

def delete\_material(current\_user, material\_id):

    material = Material.query.get\_or\_404(material\_id)

    if material.author\_id != current\_user.user\_id and current\_user.role != 'admin':

        return jsonify({'message': 'You can only delete your own materials!'}), 403

    db.session.delete(material)

    db.session.commit()

    return jsonify({'message': 'Material deleted!'}), 200

**Materials.js**

const user = JSON.parse(localStorage.getItem('user'));

document.addEventListener('DOMContentLoaded', function() {

    checkAuth();

    loadMaterials();

    const searchInput = document.getElementById('searchInput');

    const categoryFilter = document.getElementById('categoryFilter');

    if (searchInput) {

        searchInput.addEventListener('input', debounce(loadMaterials, 300));

    }

    if (categoryFilter) {

        categoryFilter.addEventListener('change', loadMaterials);

    }

    // Обработчики для страницы просмотра материала

    if (document.getElementById('materialContent')) {

        loadMaterialContent();

    }

    // Обработчики для страницы создания материала

    if (document.getElementById('createMaterialForm')) {

        setupMaterialForm();

    }

});

let allMaterials = [];

function checkAuth() {

    const token = localStorage.getItem('token');

    const user = localStorage.getItem('user');

    if (!token || !user) {

        window.location.href = '/frontend/templates/auth/login.html';

        return false;

    }

    return true;

}

function loadMaterials() {

    const searchTerm = document.getElementById('searchInput')?.value.toLowerCase() || '';

    const category = document.getElementById('categoryFilter')?.value || '';

    authFetch('/materials')

        .then(response => response.json())

        .then(data => {

            allMaterials = data.materials || [];

            // Фильтрация материалов

            const filteredMaterials = allMaterials.filter(material => {

                const matchesSearch = material.title.toLowerCase().includes(searchTerm) ||

                                    (material.content && material.content.toLowerCase().includes(searchTerm));

                const matchesCategory = !category || material.category === category;

                return matchesSearch && matchesCategory;

            });

            renderMaterials(filteredMaterials);

        })

        .catch(error => {

            console.error('Error loading materials:', error);

            renderError(error);

        });

}

function renderMaterials(materials) {

    console.log('Текущий пользователь:', JSON.parse(localStorage.getItem('user')));

    console.log('Материалы:', materials);

    const materialsContainer = document.getElementById('materialsContainer');

    if (!materialsContainer) return;

    const user = JSON.parse(localStorage.getItem('user'));

    if (!user) return;

    if (materials.length === 0) {

        materialsContainer.innerHTML = `

            <div class="empty-state">

                <i class="fas fa-book-open"></i>

                <h3>Материалы не найдены</h3>

                <p>Попробуйте изменить параметры поиска</p>

            </div>

        `;

        return;

    }

    materialsContainer.innerHTML = '';

    materials.forEach(material => {

            const isCreator = user && material && (user.user\_id == material.author\_id);

            const isAdmin = user.role === 'admin';

            const isCurator = user.role === 'curator';

            // Кураторы могут редактировать только СВОИ материалы, админы - все

            const showActions = (isCurator && isCreator) || isAdmin;

        console.log('Current user:', user);

        console.log('Material author\_id:', material.author);

        console.log('Comparison:', user.user\_id == material.author\_id);

        console.log(`Material ${material.material\_id}: isCreator=${isCreator}, isCurator=${isCurator}, showActions=${showActions}`);

        const materialCard = document.createElement('div');

        materialCard.className = 'material-card';

        materialCard.innerHTML = `

            <div class="material-image" style="background-color: ${getCategoryColor(material.category)}">

                <i class="fas fa-file-alt fa-3x"></i>

            </div>

            <div class="material-content">

                <span class="material-category">${material.category || 'Без категории'}</span>

                <h3>${material.title}</h3>

                <p>${material.content?.substring(0, 150) || 'Описание отсутствует'}...</p>

                <div class="material-meta">

                    <span>${new Date(material.upload\_date).toLocaleDateString()}</span>

                    <div class="material-actions">

                        <button class="btn-icon" onclick="viewMaterial(${material.material\_id})">

                            <i class="fas fa-eye"></i>

                        </button>

                        ${showActions ? `

                        <button class="btn-icon text-warning" onclick="event.stopPropagation(); editMaterial(${material.material\_id})">

                            <i class="fas fa-edit"></i>

                        </button>

                        <button class="btn-icon text-danger" onclick="event.stopPropagation(); deleteMaterial(${material.material\_id})">

                            <i class="fas fa-trash"></i>

                        </button>

                        ` : ''}

                    </div>

                </div>

            </div>

        `;

        materialsContainer.appendChild(materialCard);

    });

}

function getCategoryColor(category) {

    const colors = {

        'О компании': '#e6f7ff',

        'Корпоративная культура': '#fff7e6',

        'Рабочие процессы': '#e6ffed',

    };

    return colors[category] || '#f5f5f5';

}

// Функция для задержки выполнения (debounce)

function debounce(func, wait) {

    let timeout;

    return function() {

        const context = this, args = arguments;

        clearTimeout(timeout);

        timeout = setTimeout(() => {

            func.apply(context, args);

        }, wait);

    };

}

function editMaterial(materialId) {

    console.log('Editing material with ID:', materialId);

    window.location.href = `edit.html?id=${materialId}`;

}

function deleteMaterial(materialId) {

    if (confirm('Вы уверены, что хотите удалить этот материал?')) {

        authFetch(`/materials/${materialId}`, {

            method: 'DELETE'

        })

        .then(response => {

            if (response.ok) {

                alert('Материал успешно удалён');

                loadMaterials();

            }

        })

        .catch(error => {

            console.error('Error deleting material:', error);

            alert('Ошибка при удалении материала');

        });

    }

}

function loadMaterialContent() {

    const urlParams = new URLSearchParams(window.location.search);

    const materialId = urlParams.get('id');

    if (!materialId) {

        window.location.href = 'index.html';

        return;

    }

    authFetch(`/materials/${materialId}`)

        .then(response => {

            if (!response.ok) {

                throw new Error('Network response was not ok');

            }

            return response.json();

        })

        .then(data => {

            const material = data.material;

            const materialContent = document.getElementById('materialContent');

            materialContent.innerHTML = `

                <h2>${material.title}</h2>

                <div class="material-meta">

                    <span class="material-category">${material.category || 'Без категории'}</span>

                    <span>${new Date(material.upload\_date).toLocaleDateString()}</span>

                </div>

                <div class="material-text">

                    ${material.content || '<p>Содержание отсутствует</p>'}

                </div>

                ${material.file\_url ? `

                <div class="material-file">

                    <a href="${material.file\_url}" class="btn btn-outline" target="\_blank">

                        <i class="fas fa-download"></i> Скачать файл

                    </a>

                </div>

                ` : ''}

            `;

            // Показываем кнопку только для стажёров

            document.getElementById('completeMaterialBtn').addEventListener('click', () => {

                completeMaterial(materialId);

            });

        })

        .catch(error => {

            console.error('Error loading material:', error);

            if (error === 'No token found') {

                logout();

            } else {

                window.location.href = 'index.html';

            }

        });

}

function completeMaterial(materialId) {

    authFetch(`/progress/material/${materialId}`, {

        method: 'POST'

    })

    .then(response => {

        if (response.ok) {

            alert('Материал отмечен как пройденный!');

            window.location.href = '../progress/index.html';

        }

    })

    .catch(error => {

        console.error('Error completing material:', error);

        alert('Ошибка при отметке материала');

    });

}

function setupMaterialForm() {

    const form = document.getElementById('createMaterialForm');

    form.addEventListener('submit', function(event) {

        event.preventDefault();

        const title = document.getElementById('title').value;

        const category = document.getElementById('category').value;

        const content = document.getElementById('content').value;

        const fileInput = document.getElementById('file');

        // В реальном приложении здесь была бы загрузка файла

        const formData = {

            title,

            category,

            content

        };

        authFetch('/materials', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json'

            },

            body: JSON.stringify(formData)

        })

        .then(response => {

            if (response.ok) {

                alert('Материал успешно создан!');

                window.location.href = 'index.html';

            }

        })

        .catch(error => {

            console.error('Error creating material:', error);

            alert('Ошибка при создании материала');

        });

    });

}

function viewMaterial(materialId) {

    window.location.href = `view.html?id=${materialId}`;

}

// Экспортируем функции для использования в HTML

window.viewMaterial = viewMaterial;

window.editMaterial = editMaterial;

window.deleteMaterial = deleteMaterial;

**Tests.py**

from flask import Blueprint, request, jsonify

from backend.models import Test, Question, db

from backend.utils import token\_required, curator\_required

tests\_bp = Blueprint('tests', \_\_name\_\_)

@tests\_bp.route('/tests', methods=['GET'])

@token\_required

def get\_tests(current\_user):

    tests = Test.query.all()

    output = []

    for test in tests:

        test\_data = {

            'test\_id': test.test\_id,

            'title': test.title,

            'creator': f"{test.creator.name} {test.creator.surname}",

            'description': test.description,

            'max\_score': test.max\_score,

            'question\_count': len(test.questions),

            'creator\_id': test.creator\_id

        }

        output.append(test\_data)

    return jsonify({'tests': output}), 200

@tests\_bp.route('/tests/<int:test\_id>', methods=['GET'])

@token\_required

def get\_test(current\_user, test\_id):

    test = Test.query.get\_or\_404(test\_id)

    test\_data = {

        'test\_id': test.test\_id,

        'title': test.title,

        'description': test.description,

        'max\_score': test.max\_score,

        'questions': []

    }

    for question in test.questions:

        question\_data = {

            'question\_id': question.question\_id,

            'text': question.text,

            'points': question.points,

            'options': {

                'A': question.option\_a,

                'B': question.option\_b,

                'C': question.option\_c,

                'D': question.option\_d

            },

            'correct\_option': question.correct\_option

        }

        test\_data['questions'].append(question\_data)

    return jsonify({'test': test\_data}), 200

@tests\_bp.route('/tests', methods=['POST'])

@token\_required

@curator\_required

def create\_test(current\_user):

    data = request.get\_json()

    new\_test = Test(

        title=data['title'],

        creator\_id=current\_user.user\_id,

        description=data.get('description'),

        max\_score=sum(q['points'] for q in data['questions'])

    )

    db.session.add(new\_test)

    db.session.commit()

    for question in data['questions']:

        new\_question = Question(

            test\_id=new\_test.test\_id,

            text=question['text'],

            points=question['points'],

            option\_a=question['options']['A'],

            option\_b=question['options']['B'],

            option\_c=question['options'].get('C'),

            option\_d=question['options'].get('D'),

            correct\_option=question['correct\_option']

        )

        db.session.add(new\_question)

    db.session.commit()

    return jsonify({'message': 'Test created!', 'test\_id': new\_test.test\_id}), 201

@tests\_bp.route('/tests/<int:test\_id>', methods=['PUT'])

@token\_required

@curator\_required

def update\_test(current\_user, test\_id):

    test = Test.query.get\_or\_404(test\_id)

    if test.creator\_id != current\_user.user\_id and current\_user.role != 'admin':

        return jsonify({'message': 'You can only edit your own tests!'}), 403

    data = request.get\_json()

    test.title = data.get('title', test.title)

    test.description = data.get('description', test.description)

    # Удаляем старые вопросы

    Question.query.filter\_by(test\_id=test\_id).delete()

    # Добавляем новые вопросы

    test.max\_score = sum(q['points'] for q in data['questions'])

    for question in data['questions']:

        new\_question = Question(

            test\_id=test.test\_id,

            text=question['text'],

            points=question['points'],

            option\_a=question['options']['A'],

            option\_b=question['options']['B'],

            option\_c=question['options'].get('C'),

            option\_d=question['options'].get('D'),

            correct\_option=question['correct\_option']

        )

        db.session.add(new\_question)

    db.session.commit()

    return jsonify({'message': 'Test updated!'}), 200

@tests\_bp.route('/tests/<int:test\_id>', methods=['DELETE'])

@token\_required

def delete\_test(current\_user, test\_id):

    test = Test.query.get\_or\_404(test\_id)

    if test.creator\_id != current\_user.user\_id and current\_user.role != 'admin':

        return jsonify({'message': 'You can only delete your own tests!'}), 403

    db.session.delete(test)

    db.session.commit()

    return jsonify({'message': 'Test deleted!'}), 200

**Tests.js**

const user = JSON.parse(localStorage.getItem('user'));

document.addEventListener('DOMContentLoaded', function() {

    checkAuth();

    loadTests();

    const searchInput = document.getElementById('searchInput');

    if (searchInput) {

        searchInput.addEventListener('input', debounce(loadTests, 300));

    }

    // Загрузка тестов для списка

    if (document.getElementById('testsContainer')) {

        loadTests();

    }

    // Загрузка теста для прохождения

    if (document.getElementById('testForm')) {

        loadTestForTaking();

    }

    if (document.getElementById('editTestForm')) {

        setupTestForm();

    }

});

let allTests = [];

let currentTest = null;

let currentQuestionIndex = 0;

let userAnswers = {};

function checkAuth() {

    const token = localStorage.getItem('token');

    const user = localStorage.getItem('user');

    if (!token || !user) {

        window.location.href = '/frontend/templates/auth/login.html';

        return false;

    }

    return true;

}

function loadTests() {

    const searchTerm = document.getElementById('searchInput')?.value.toLowerCase() || '';

    authFetch('/tests')

        .then(response => response.json())

        .then(data => {

            allTests = data.tests || [];

            // Фильтрация тестов

            const filteredTests = allTests.filter(test => {

                return test.title.toLowerCase().includes(searchTerm) ||

                       (test.description && test.description.toLowerCase().includes(searchTerm));

            });

            renderTests(filteredTests);

        })

        .catch(error => {

            console.error('Error loading tests:', error);

            renderError(error);

        });

}

function renderTests(tests) {

    console.log('Текущий пользователь:', JSON.parse(localStorage.getItem('user')));

    console.log('Тесты:',  tests);

    const testsContainer = document.getElementById('testsContainer');

    if (!testsContainer) return;

    const user = JSON.parse(localStorage.getItem('user'));

    if (!user) return;

    if (tests.length === 0) {

        testsContainer.innerHTML = `

            <div class="empty-state">

                <i class="fas fa-question-circle"></i>

                <h3>Тесты не найдены</h3>

                <p>Попробуйте изменить параметры поиска</p>

            </div>

        `;

        return;

    }

    testsContainer.innerHTML = '';

    tests.forEach(test => {

        const isCreator = user && test && (user.user\_id == test.creator\_id);

        const isAdmin = user.role === 'admin';

        const isCurator = user.role === 'curator';

        const showActions = (isCreator && isCurator) || isAdmin;

        console.log(`Test ${test.test\_id}: isCreator=${isCreator}, isCurator=${isCurator}, showActions=${showActions}`);

        const testCard = document.createElement('div');

        testCard.className = 'test-card';

        testCard.innerHTML = `

            <h3>${test.title}</h3>

            <p>${test.description || 'Описание отсутствует'}</p>

            <div class="progress-container">

                <div class="progress-label">

                    <span>Вопросов:</span>

                    <span>${test.question\_count}</span>

                </div>

            </div>

            <div class="test-actions">

                <button class="btn btn-primary" onclick="startTest(${test.test\_id})">

                    Начать тест

                </button>

                <div class="test-actions">

                ${showActions ? `

                <button class="btn btn-outline-warning" onclick="event.stopPropagation(); editTest(${test.test\_id})">

                    <i class="fas fa-edit"></i>

                </button>

                <button class="btn btn-outline-danger" onclick="event.stopPropagation(); deleteTest(${test.test\_id})">

                    <i class="fas fa-trash"></i>

                </button>

                ` : ''}

                </div>

            </div>

            <div class="test-author">

                Автор: ${test.creator || 'Неизвестен'}

            </div>

        `;

        testsContainer.appendChild(testCard);

    });

}

// Функция debounce (такая же как в materials.js)

function debounce(func, wait) {

    let timeout;

    return function() {

        const context = this, args = arguments;

        clearTimeout(timeout);

        timeout = setTimeout(() => {

            func.apply(context, args);

        }, wait);

    };

}

function editTest(testId) {

    console.log('Attempting to edit test:', testId);

    // Открываем в новой вкладке для отладки

    window.open(`edit.html?id=${testId}`, '\_blank');

    // Или для обычного использования:

    // window.location.href = `edit.html?id=${testId}`;

}

function deleteTest(testId) {

    if (confirm('Вы уверены, что хотите удалить этот тест?')) {

        authFetch(`/tests/${testId}`, {

            method: 'DELETE'

        })

        .then(response => {

            if (response.ok) {

                alert('Тест успешно удалён');

                loadTests();

            }

        })

        .catch(error => {

            console.error('Error deleting test:', error);

            alert('Ошибка при удалении теста');

        });

    }

}

function loadTestForTaking() {

    const urlParams = new URLSearchParams(window.location.search);

    const testId = urlParams.get('id');

    if (!testId) {

        window.location.href = 'index.html';

        return;

    }

    authFetch(`/tests/${testId}`)

        .then(response => {

            if (!response.ok) {

                throw new Error('Network response was not ok');

            }

            return response.json();

        })

        .then(data => {

            currentTest = data.test;

            userAnswers = {};

            document.getElementById('testTitle').textContent = currentTest.title;

            document.getElementById('testDescription').textContent = currentTest.description || '';

            document.getElementById('totalQuestions').textContent = currentTest.questions.length;

            showQuestion(0);

        })

        .catch(error => {

            console.error('Error loading test:', error);

            if (error === 'No token found') {

                logout();

            } else {

                window.location.href = 'index.html';

            }

        });

}

function showQuestion(index) {

    if (!currentTest || index < 0 || index >= currentTest.questions.length) {

        return;

    }

    currentQuestionIndex = index;

    const question = currentTest.questions[index];

    document.getElementById('currentQuestion').textContent = index + 1;

    document.getElementById('progressFill').style.width = `${(index / currentTest.questions.length) \* 100}%`;

    const questionContainer = document.getElementById('questionContainer');

    questionContainer.innerHTML = `

        <div class="question">

            <h3>${question.text}</h3>

            <div class="question-points">${question.points} баллов</div>

            <div class="options">

                ${Object.entries(question.options).map(([key, value]) => `

                    <div class="option">

                        <input type="radio"

                               id="option-${key}"

                               name="answer"

                               value="${key}"

                               ${userAnswers[question.question\_id] === key ? 'checked' : ''}>

                        <label for="option-${key}">

                            <span class="option-letter">${key}</span>

                            <span class="option-text">${value}</span>

                        </label>

                    </div>

                `).join('')}

            </div>

        </div>

    `;

    // Настройка кнопок навигации

    document.getElementById('prevQuestionBtn').disabled = index === 0;

    document.getElementById('nextQuestionBtn').style.display = index === currentTest.questions.length - 1 ? 'none' : 'inline-block';

    document.getElementById('submitTestBtn').style.display = index === currentTest.questions.length - 1 ? 'inline-block' : 'none';

    // Сохранение ответа при изменении

    const radioInputs = document.querySelectorAll('input[name="answer"]');

    radioInputs.forEach(input => {

        input.addEventListener('change', function() {

            userAnswers[question.question\_id] = this.value;

        });

    });

}

function startTest(testId) {

    window.location.href = `take.html?id=${testId}`;

}

function nextQuestion() {

    saveCurrentAnswer();

    if (currentQuestionIndex < currentTest.questions.length - 1) {

        showQuestion(currentQuestionIndex + 1);

    }

}

function prevQuestion() {

    saveCurrentAnswer();

    showQuestion(currentQuestionIndex - 1);

}

function saveCurrentAnswer() {

    const selectedOption = document.querySelector('input[name="answer"]:checked');

    if (selectedOption) {

        const questionId = currentTest.questions[currentQuestionIndex].question\_id;

        userAnswers[questionId] = selectedOption.value;

    }

}

function submitTest(event) {

    event.preventDefault();

    saveCurrentAnswer();

    // Подсчёт баллов

    let score = 0;

    currentTest.questions.forEach(question => {

        if (userAnswers[question.question\_id] === question.correct\_option) {

            score += question.points;

        }

    });

    authFetch(`/progress/test/${currentTest.test\_id}`, {

        method: 'POST',

        headers: {

            'Content-Type': 'application/json'

        },

        body: JSON.stringify({ score })

    })

    .then(response => {

        if (response.ok) {

            alert(`Тест завершён! Ваш результат: ${score} из ${currentTest.max\_score} баллов`);

            window.location.href = '../progress/index.html';

        }

    })

    .catch(error => {

        console.error('Error submitting test:', error);

        alert('Ошибка при отправке результатов теста');

    });

}

window.startTest = startTest;

window.nextQuestion = nextQuestion;

window.prevQuestion = prevQuestion;

window.editTest = editTest;

window.deleteTest = deleteTest;

document.getElementById('nextQuestionBtn')?.addEventListener('click', nextQuestion);

document.getElementById('prevQuestionBtn')?.addEventListener('click', prevQuestion);

document.getElementById('submitTestBtn')?.addEventListener('click', submitTest);