

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Специальность: 09.05.01 «Применение и эксплуатация
автоматизированных систем специального
назначения»

Специализация: «Эксплуатация вычислительных машин, комплексов,
систем и сетей специального назначения»

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
СПЕЦИАЛИСТА**

на тему

Автоматизированная система для сбора и анализа информации
из социальной сети для поиска потенциальных клиентов

Студент Цыпин Леонид Владимирович (ФИО полностью)
03.06.2020 (подпись, дата)

Руководитель Митрохин М.А. (фамилия, инициалы)
03.06.20 (подпись, дата)

Нормоконтролёр Кучин А.В. (фамилия, инициалы)
03.06.2020 (подпись, дата)

Рецензент Севастьянов А.В. (фамилия, инициалы)
03.06.2020 (подпись, дата)
(должность, место работы)

Работа допущена к защите (протокол зас. кафедры от 10.06.2020 г. № 13)

Заведующий кафедрой М.А. Митрохин (фамилия, инициалы)
(подпись)

Работа защищена с отметкой «отлично» (протокол заседания ГЭК от 19.06.2020 № 15)

Секретарь ГЭК Е.И. Гурин

Пенза 2020

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«30» *сентября* 2020г.

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
СПЕЦИАЛИСТА**

1. Студент Цыпин Леонид Владимирович гр. 15В02 факультета вычислительной техники.
2. Тема работы: «Автоматизированная система для сбора и анализа информации из социальной сети для поиска потенциальных клиентов»
Тема утверждена приказом ПГУ № 0335а/о от 30.04.2020 г.
3. Руководитель работы д.т.н., доцент каф. ВТ ПГУ Митрохин М.А.
4. Задание на работу (назначение разработки, исходные данные и т.п.).

Назначение разработки – Система предназначена для сбора, анализа и поиска потенциальных клиентов из социальной сети.

5. Перечень подлежащих разработке вопросов.

Требуется произвести анализ предметной области, выбрать технологии для разработки, рассчитать экономическую часть, разработать алгоритм работы системы с последующим написанием программы.

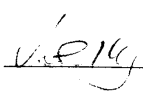
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10	09.05.20	
2.	Исследование автоматизированных систем для продвижения в социальной сети	5	16.05.20	
3.	Обзор методов и средств проектирования	10	18.05.20	
4.	Разработка алгоритма системы	30	23.05.20	
5.	Разработка системы	50	26.05.20	
7.	Эргономика рабочего места программиста	5	02.06.20	
8.	Расчет экономической части	5	02.06.20	
9..	Исправление ошибок в работе системы	10	04.06.20	
10.	Оформление выпускной работы	20	06.06.20	

Дата выдачи задания 30 «апреля» 2020 года.

Руководитель дипломного проекта  Митрохин М.А.

(подпись, дата)

Задание к исполнению принял студент  30.04.2020 Цыпин Л.В.

(подпись, дата)

Работу к защите допустить
 Декан факультета  Фионова Л.Р.
 (подпись, дата)

РЕФЕРАТ

Работа содержит 101 лист, 28 рисунков, 9 таблиц, 13 источника, 3 приложения.

Автоматизированная система для сбора и анализа информации из социальной сети для поиска потенциальных клиентов.

КЛЮЧЕВЫЕ СЛОВА: INSTAGRAM, ПРОДВИЖЕНИЕ, АККАУНТ, ХЕШТЕГ, ГЕОМЕТКА, СОЦИАЛЬНАЯ СЕТЬ.

Цель работы – разработать масштабируемую, отказоустойчивую автоматизированную систему для сбора, анализа и подбора потенциальных клиентов из социальной сети Instagram.

Объектом разработки является веб-приложение, обеспечивающее клиентам доступ к статистике и данным их личного кабинета, возможностям создавать задачи для сбора и анализа данных.

В результате проведенной работы должен быть разработан дипломный проект, удовлетворяющий поставленной задаче, а именно, создан клиент, разработана серверная часть, хранилище данных, и разработаны соответствующие алгоритмы, реализующие возможность хранить и обрабатывать данные.

Выполнено функциональное тестирование системы, которое продемонстрировало правильность работы системы.

В проектной части описан функционал системы. Описана технология разработки и алгоритм получения данных из социальной сети.

					ПГУ 1.090501.15.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата					
Разраб.		Цыпин Л.В.			Автоматизированная система для сбора и анализа информации из социальной сети для поиска потенциальных клиентов. Пояснительная записка	Лит.	Лист	Листов	
Провер.		Митрохин М.А.					2	102	
Н. Контр.		Кучин А.В.				Кафедра ВТ Гр.15В02			
Реценз.		Севастьянов А.В.							
Утверд.		М.А.Митрохин							

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 Предпроектный анализ	7
1.1 Исследование автоматизированных систем для продвижения в социальной сети	7
1.2 Постановка задачи на разработку системы	8
1.3 Анализ функциональных требований	11
1.4 Обзор методов и средств проектирования.....	15
2 Разработка программного обеспечения сайта	18
2.1 Архитектура системы.....	18
2.2 Разработка клиентской части системы.....	19
2.3 Разработка серверной части системы.....	23
2.4 Разработка базы данных.....	41
2.4.1 Разработка базы данных.....	41
2.4.2 Разработка обработчиков базы данных	42
3 Тестирование и развертывание программного обеспечения	45
3.1 Тестирование программного обеспечения	45
3.2 Развертывание программного обеспечения	53
4 Технико-экономическое обоснование разработки программы	54
4.1 Введение	54
4.2 Расчет затрат на разработку программного обеспечения	56
4.3 Расчет затрат на разработку программного обеспечения	58
4.4 Расчет капитальных вложений	59
4.5 Расчет эксплуатационных расходов	59
4.6 Определение расходов на интернет трафик.....	62
4.7 Заключение	63
5 Безопасность жизнедеятельности. Эргономика рабочего места программиста.....	64
5.1 Характеристика условий труда программиста.....	64

5.2 Эргономические требования к рабочему месту программиста	65
ЗАКЛЮЧЕНИЕ	66
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	67
Приложение А. Листинги программы	68
Приложение Б. Диаграммы.....	93
Приложение В. Презентация	98

ВВЕДЕНИЕ

В настоящее время социальный маркетинг набирает всё большую популярность, и люди все больше времени проводят в социальных сетях. Из-за огромного количества информации в интернете, людям все сложнее ориентироваться в выдаче ссылок в поисковых системах, и зачастую могут просто пройти мимо интересных предложений. По этой причине, компаниям необходимо все больше времени уделять на маркетинг в сфере социальных сетей, если они хотят увеличить своё количество клиентов и создать купол популярности в своих нишах.

По данным социологов из Фонда "Общественное мнение" (ФОМ) более 40% россиян стабильно раз в день заходят в социальную сеть, и с каждым днем это количество возрастает в разы [13]. Это служит отличной возможностью для компаний использовать социальные сети, чтобы завоевать положительное мнение у клиентов и расширить свою аудиторию. Поэтому так важно присутствовать в социальных сетях. Пользователи социальных сетей, находясь в своей зоне комфорта, будут более благоприятными для общения и изучения новых предложений компаний.

Для решения задачи поиска самых подходящих потенциальных клиентов в социальных сетях предлагается разработать систему, которая будет учитывать все необходимые факторы и собирать базу таких клиентов.

В данной выпускной квалификационной работе создана автоматизированная система по поиску, анализу и подбору потенциальных клиентов на данных социальной сети Instagram.

1 Предпроектный анализ

1.1 Исследование автоматизированных систем для продвижения в социальной сети

Существует много критериев, по которым можно сравнить сервисы по продвижению в социальной сети Instagram.

Выделим основные из них: поиск по геометкам, аналитика аккаунта, поиск по хештегам, общение с подписчиками, цена за месяц использования, (таблица 1.1).

Таблица 1.1 – Сравнение сервисов по продвижению в социальной сети Instagram Tooligram, Паразайт, Zengram

	Tooligram	Паразайт	Zengram
Поиск по геометкам	Есть	Есть	Нет
Поиск по хештегам	Есть	Нет	Есть
Общение в Direct	Есть	Есть	Есть
Аналитика аккаунта	Есть	Есть	Нет
Цена за месяц использования	2900 рублей	1599 рублей	699 рублей

Сравнение возможностей трех лидеров рынка предоставления услуг по продвижению в социальной сети Instagram показывает, что пользователи хотят автоматизировать поиск нахождения клиентов по геометкам и по хештегам, а также автоматически присылать сообщения новым подписчикам.

Но ни один из сервисов не предоставляет возможность полностью автоматизировать поиск нахождения клиентов, а именно, анализировать аккаунт, генерировать геометки и хештеги, по которым уже будет происходить сам поиск.

1.2 Постановка задачи на разработку системы

Основной задачей дипломного проекта является создание автоматизированной системы, которая будет собирать базу пользователей, анализировать и подбирать потенциальных клиентов в соответствии с параметрами аккаунта компании в социальной сети Instagram.

Главная направленность SMM продвижения схожа с SEO продвижением. SEO целенаправленно занимается продвижением во взаимодействии с поисковой системой, а SMM продвижение оказывает особое влияние на людей, учитывая их психологию, интересы, привычки, настроение.

Для того, чтобы эффективно продвигать себя или свой бренд в социальной сети, необходимо учитывать все эти факторы, а социальная сеть Instagram позволяет понять каждого пользователя по их публикациям. Поиск новых клиентов или новой аудитории должен производиться не бездумно, а по заранее подготовленным шаблонам.

На данный момент уже существуют подобные системы, но ни одна из них не умеет анализировать ваш аккаунт и создавать шаблон поиска новых клиентов по данным, собранным с ваших публикаций.

На основе анализа уже существующих сервисов было принято решение разработать свою систему.

Система должна предоставлять пользователям возможность полностью автоматизировать продвижение, предоставив при этом только свой аккаунт в социальной сети Instagram.

Рассмотрим алгоритм поиска потенциальных клиентов или аудитории в соответствии нашего бренда:

- 1) Определяем возрастную категорию;
- 2) Определяем привычки, личностные качества и интересы;
- 3) Определяем географию;
- 4) Собираем базу пользователей;
- 5) Привлекаем их внимание.

Рассмотрим на примере: у нас есть компания, занимающаяся производством и продажей классической мужской одежды в г. Пенза.

- 1) Клиентами будут люди от 16 лет;
- 2) Мужчины, любящие проводить время в дорогих ресторанах или у которых скоро выпускной или любое другое торжественное мероприятие, мужчины и женщины, которым нравятся классические костюмы;
- 3) Пензенская область;
- 4) Ищем всех пользователей с геометками Пензенская область, отбираем тех, у кого возрастная группа и интересы совпадают с теми, что мы определили;
- 5) Собираем базу из этих пользователей, а также их подписчиков и подписок;
- 6) Привлекаем их внимание сообщениями, просмотрами истории или действиями с их публикациями.

После всех этих действий необходимо подождать реакции пользователей и, если мы получили результат в виде новых клиентов или обратной коммуникации с их стороны, то повторяем эти действия каждую неделю чтобы добиться максимальных результатов. Если же мы не получили

результат, то меняем алгоритм поиска новых клиентов, пока не добьемся ожидаемых результатов.

Для того, чтобы наша компания процветала, необходимо искать клиентов и сообщать о себе на всем протяжении существования компании.

Но на это придется тратить в лучшем случае, как минимум 1 день в неделю.

Таким образом, приходим к выводу, чтобы уделять меньше времени и эффективно искать клиентов необходимо использовать для сбора, анализа и поиска потенциальных клиентов автоматизированную систему.

Система должна предоставлять пользователям возможность создавать задачи, в которых они сами выбирают то, что нужно выполнить, используя их аккаунт:

- 1) Сбор и анализ пользователей социальной сети на основе данных с аккаунта самого пользователя системы;
- 2) Сбор и анализ пользователей социальной сети по хештегам;
- 3) Сбор и анализ пользователей социальной сети по геометкам;
- 4) Просмотр историй пользователей социальной сети по базе, собранной ранее.

Для реализации системы необходимо:

- 1) База данных, которая должна сохранять весь прогресс, формировать базу найденных пользователей социальной сети и хранить сессионный токен от аккаунта пользователя системы;
- 2) Серверная часть системы, которая будет взаимодействовать с этой базой;
- 3) Клиентская часть системы, которая общается с серверной частью и с которой взаимодействует сам пользователь.

Модель предметной области представлена на рисунке 1.1.



Рисунок 1.1 – Диаграмма классов предметной области

Классы «Задача», «Статистика», «Хештег», «Геометка» являются представлениями данных, которые должны храниться в системе.

Класс «Система» взаимодействует базой и хранит данные по задачам, статистике, геометкам, хештегам, аккаунтам социальной сети Instagram.

Класс «Задача» взаимодействует с системой и Instagram аккаунтами и хранит шаблон поиска данных в социальной сети Instagram.

Класс «Статистика» взаимодействует с задачами и системой и хранит данные по прогрессу задач.

Класс «Хештег» взаимодействует с Instagram аккаунтом и хранит хештеги публикаций и других активностей аккаунта.

Класс «Геометка» взаимодействует с Instagram аккаунтом и хранит геолокации публикаций и других активностей аккаунта.

1.3 Анализ функциональных требований

Требования можно определить как описание того, что должно быть реализовано. Существуют два основных типа требования:

- функциональные требования – какое поведения должна предлагать система;
- нефункциональные требования – особое свойство или ограничение, накладываемое на систему [4].

В соответствии с этим, можно сформировать следующие требования к работе функций системы:

- Авторизация в системе;
- Добавление аккаунта Instagram;
- Удаление аккаунта Instagram;
- Создание задач;
- Удаление задач;
- Формирование статистики по задачам.

Исходя из анализа работы системы, уже сейчас можно сформировать описание основным вариантам использования системы. Данное описание представлено в таблице 1.2.

Таблица 1.2 – Краткое описание вариантов использования

Действующее лицо	Цель	Описание
Пользователь	Авторизация в системе	Пользователь вводит логин и пароль. Если данные корректны, то пользователю предоставляется доступ к остальным возможностям системы
	Добавление аккаунта Instagram	Пользователь вводит логин и пароль от своего аккаунта Instagram
	Удаление аккаунта Instagram	Пользователь выбирает добавленный аккаунт Instagram и удаляет его из системы

Продолжение таблицы 1.2

Действующее лицо	Цель	Описание
Пользователь	Создание задач	Пользователь создает задачу на аналитику или продвижение в социальной сети Instagram
	Удаление задач	Пользователь выбирает из списка задач задачу, которую он хочет удалить
	Формирование статистики по задачам	Пользователь переходит на экран со статистикой по задачам и может просмотреть их прогресс или ошибки

На основании таблицы 1.2 можно создать диаграмму вариантов использования для разрабатываемого программного обеспечения, включающую основные действия пользователя.

Диаграмма вариантов использования представлена на рисунке 1.2.

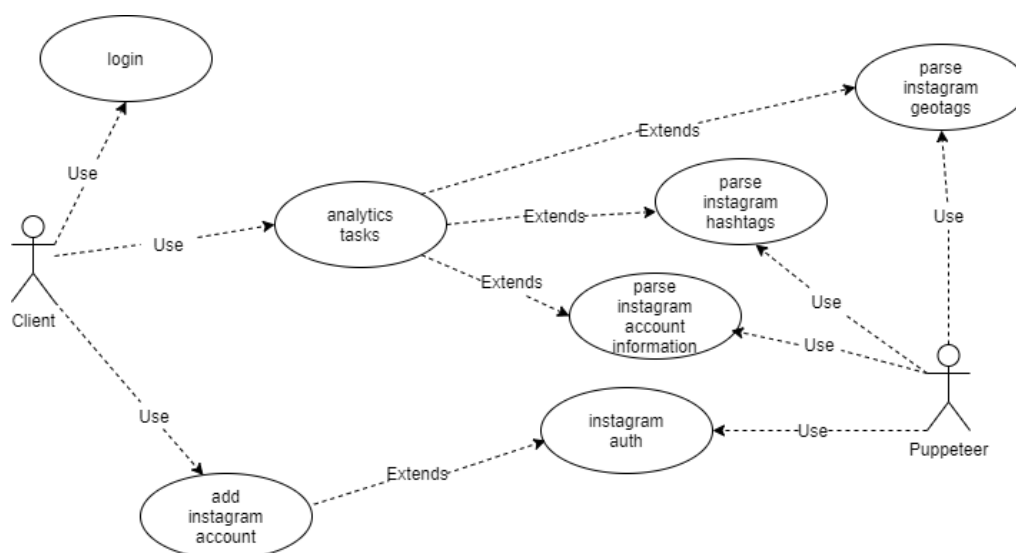


Рисунок 1.2 – Диаграмма вариантов использования

Одним из способов описания вариантов использования является сценарий. Сценарий представляет собой последовательность шагов, описывающих взаимодействие между пользователем и системой. Количество деталей в сценарии зависит от риска в соответствующем варианте использования: чем больше риск, тем больше деталей необходимо указать. Часто случается так, что на фазе исследования детально описывается небольшое количество вариантов использования [6].

Описание вариантов использования:

Вариант использования: Авторизация в системе

Основное действующее лицо: Пользователь.

Область действия: Сайт.

Уровень: Цель пользователя.

Участники и интересы: Пользователь хочет получить доступ к остальному функционалу системы.

Предусловие: Пользователя имеет доступ в интернет. Открыта страница авторизации. Пользователь не авторизован.

Минимальные гарантии: Пользователь получил ошибку авторизации.

Гарантия успеха: Пользователь авторизовался.

Основной сценарий:

- 1) Пользователь вводит логин и пароль на экране авторизации;
- 2) Система делает запрос в БД и отправляет результат на клиент;
- 3) Пользователя перенаправляют на основной экран в случае успеха, или показывают ошибку авторизации в случае неудачи.

Вариант использования: Создание задачи.

Основное действующее лицо: Пользователь.

Область действия: Сайт.

Уровень: Цель пользователя.

Участники и интересы: Пользователь хочет создать новую задачу для сбора и анализа аккаунтов социальной сети Instagram.

Предусловие: У пользователя есть доступ в интернет. Пользователь на экране создания задачи. Пользователь добавил свой аккаунт Instagram

Минимальные гарантии: Пользователь получил ошибку создания новой задачи.

Гарантия успеха: Пользователь создал новую задачу.

Основной сценарий:

- 1) Пользователь переходит на экран создания задачи;
- 2) Пользователь выбирает необходимые фильтры;
- 3) Пользователь нажимает кнопку Создать.
- 4) Пользователь получает информацию об успехе или неудаче.

Работу системы можно описать так:

- 1) Пользователь авторизовывается в системе и совершает любое действие по своему усмотрению;
- 2) Система возвращает результат пользователю;
- 3) Пользователь может выбрать следующее действие, либо выйти из системы.

1.4 Обзор методов и средств проектирования

Javascript – мультипарадигменный язык программирования, который поддерживает несколько стилей программирования.

Инструментарий был выбран исходя из производительности, масштабируемости и возможностей для быстрой разработки (таблица 1.2)

Таблица 1.2 – Инструменты разработки

№	Наименование инструмента	Причина выбора
1	Next JS	Реактивный JS фреймворк, позволяющий генерировать структуру страницы для клиентской части в серверной части
2	Nest JS	Node JS фреймворк для реализации серверной части, позволяющий масштабировать функционал
3	MongoDB	Система управления базами данных класса NoSQL в формате «ключ — значение»

Фреймворк NextJS используется для написания React приложений. Nextjs поддерживает Server-Side Rendering, что позволяет исходные данные генерировать на сервере и тем самым обеспечить быструю загрузку страниц и снизить нагрузку на браузер.

Server-Side Rendering – это выполнение JS кода на стороне сервера, чтобы отдать на клиент уже готовую HTML страницу. Это не означает, что на клиенте больше не будет выполняться JS код, он по-прежнему остается и будет функционировать, но первая итерация происходит на сервере, в том числе и запрос к API, если он вызывается при рендеринге страницы. Таким образом сокращается время на обработку JS кода браузером при загрузке страницы, но, чтобы избежать потерь данных, браузеру необходимо пройти по собранному DOM и убедиться, что он совпадает с тем, что отрендирилось на клиенте и добавить евент листенеры. В случае, если не совпадает, то браузер сам заново рендерит страницу.

Фреймворк NestJS используется для написания серверной части на NodeJS. NestJS создан для того, чтобы облегчить жизнь разработчикам, навязывая правильную архитектуру системы и правильные подходы к

разработке, что уменьшает количество ошибок и проблем при разработке и дальнейшей поддержке.

NodeJS – серверный JS, технология, позволяющая писать серверные многопоточные приложения на языке программирования JavaScript.

MongoDB – это система управления нереляционной базой данных. Она позволяет быстро масштабировать БД, не имеет схемы, позволяет быстро разрабатывать, имеет такую же модель данных, как и в серверной части.

Механизмы авторизации, аутентификации и выполнения задач пользователей в социальной сети Instagram происходит при помощи библиотеки Puppeteer. Библиотека puppeteer для Node.js позволяет автоматизировать работу с браузером Google Chrome. В частности, с помощью puppeteer можно создавать программы для автоматического сбора данных с веб-сайтов, имитирующие действия обычного пользователя. В подобных сценариях может применяться браузер без пользовательского интерфейса, так называемый «Headless Chrome». Используя puppeteer, можно управлять и браузером, который запущен в обычном режиме, что особенно полезно при отладке программ.

При помощи Puppeteer мы открываем сессию в google chrome, в которой эмулируем поведения авторизации пользователя в социальной сети Instagram, получаем сессионную информацию, данные с аккаунта и сохраняем их в нашу базу данных.

2 Разработка программного обеспечения сайта

2.1 Архитектура системы

Разрабатываемая система представляет собой трехслойную архитектуру (рисунок 2.1)

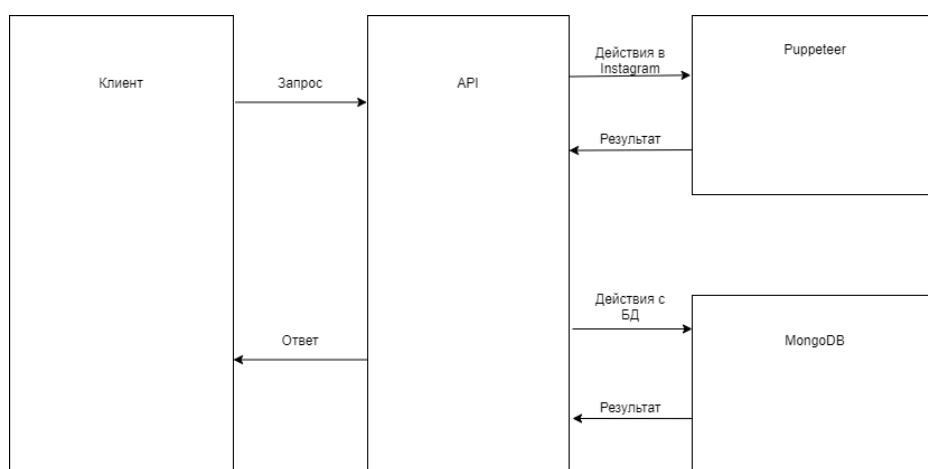


Рисунок 2.1 – Структура проекта.

Каждый из участников наделен определенным функционалом, который требуется реализовать в виде связки клиентская часть, серверная часть и база данных.

Клиент отображает графический интерфейс в браузере у пользователей и взаимодействует с API через запросы.

Сервер взаимодействует с библиотекой puppeteer для того, чтобы совершать действия в социальной сети Instagram, а также взаимодействует с СУБД MongoDB для работы с базой данных.

Нереляционная база данных, которая управляется с помощью MongoDB, хранит все данные пользователей системы.

2.2 Разработка клиентской части системы

Для того, чтобы увеличить заинтересованность у пользователей системы интерфейс клиентской части должен быть интуитивно понятен.

При первом открытии сайта, пользователь видит экран авторизации, с двумя полями для ввода логина и пароля, и кнопкой для попытки авторизации (рис. 2.2).

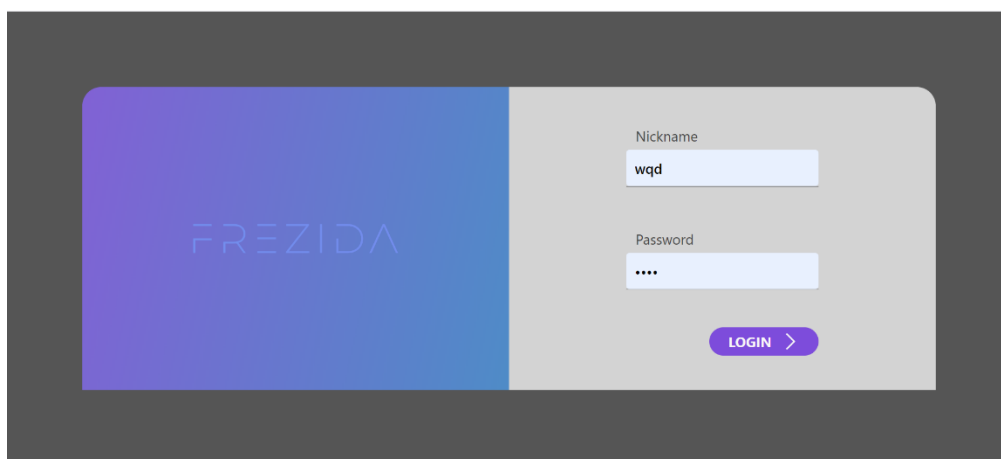


Рисунок 2.2 – Экран логина

При нажатии на кнопку авторизации вызывается обработчик onLogin:

```
const onLogin = () => {  
  login({login, password})  
    .then(() => redirectTo('/dashboard'))  
    .catch((err) => setError({login: err}))  
};
```

Обработчик onLogin делает запрос в API с введенными логином и паролем.

При успешной авторизации, у пользователя меняется экран на главный экран со списком всех возможностей системы (рис. 2.3).

Изм.	Лист	№ докум.	Подпись	Дата

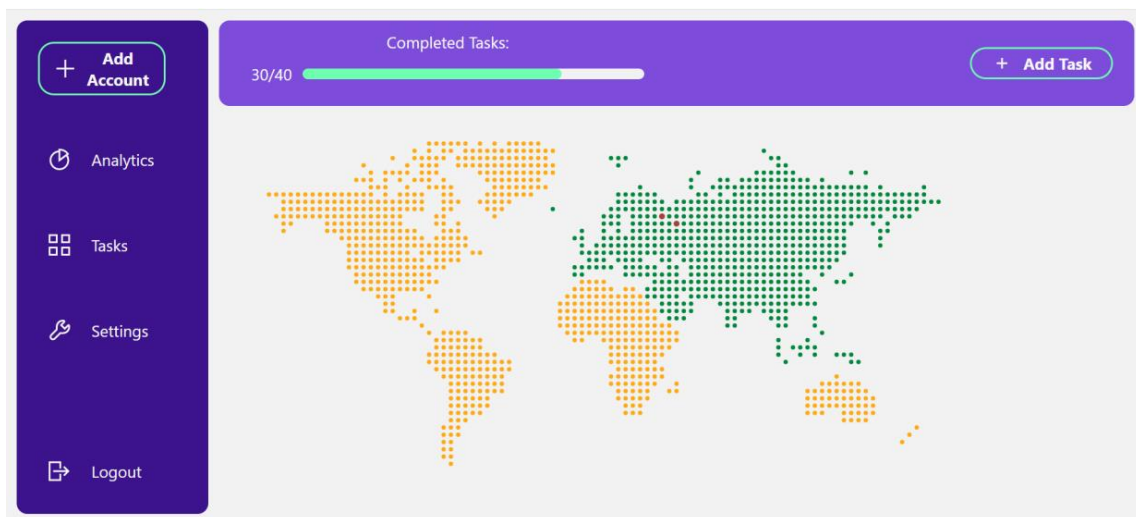


Рисунок 2.3 – Экран со списком возможностей системы

При попытке создать задачу вызывается обработчик `handleAdd`.

```
const handleAdd = () => {
  if (props.user.inst && props.user.inst.length)
    return redirectTo('/dashboard/task')
  return setIsOpen(true)
};
```

Обработчик `handleAdd` проверяет, есть ли у пользователя аккаунт Instagram, если есть, то показывает экран создания задачи.

В случае если пользователь не добавил аккаунт Instagram, появляется модальное окно, в котором сообщается о невозможности создании задачи без добавленного аккаунта Instagram и предложении его добавить (рис. 2.4).

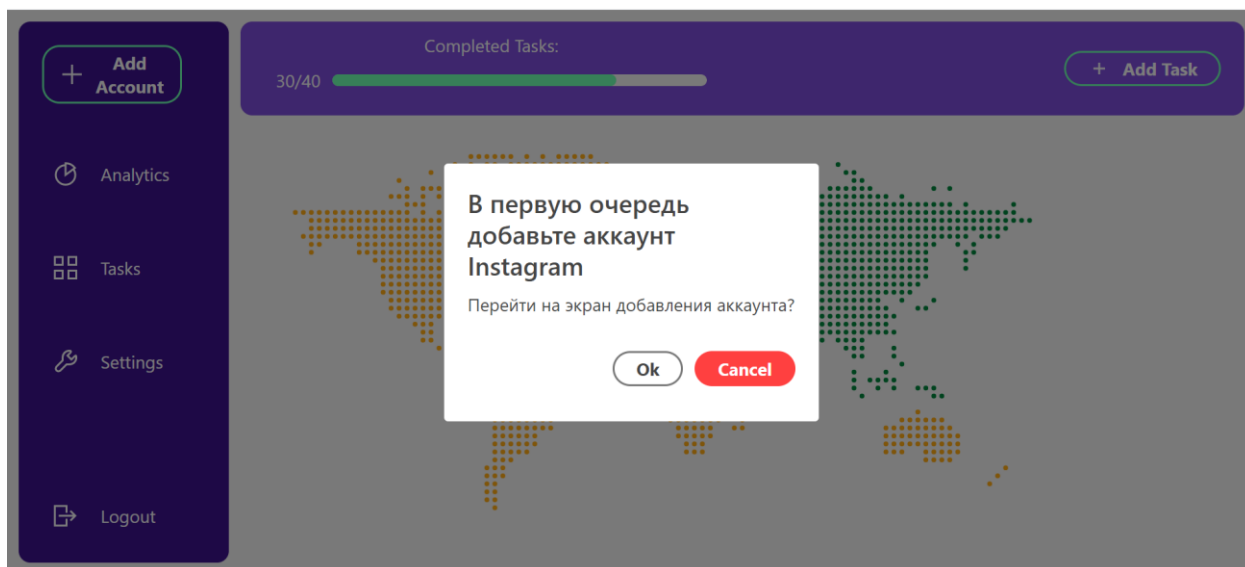


Рисунок 2.4 – Ошибка добавления задачи

Экран добавления аккаунта Instagram содержит только два поля ввода логина и пароля от аккаунта Instagram и кнопку добавления (рис 2.5).

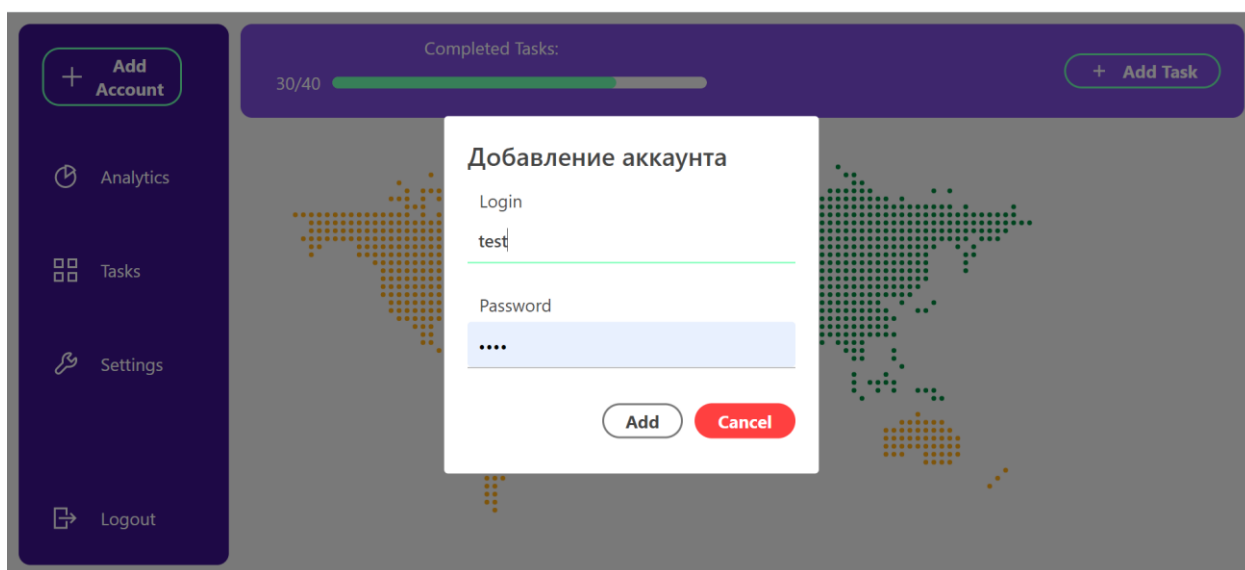


Рисунок 2.5 – Экран добавления аккаунта

При нажатии на кнопку Отмена, закрывается экран добавления аккаунта.

При нажатии на кнопку Добавить вызывается обработчик `handleAddAccount`.

```
const handleAddAccount = () => {
    addAccount({
        login,
        password
    })
    .then(() => {
        setOpenSuccess(true)
    })
    .catch((err) => setError({ inst: err }));
};
```

Обработчик `handleAddAccount` отправляет запрос в API с логином и паролем от аккаунта Instagram. Если сервер смог авторизоваться и получить сессию от аккаунта Instagram, то закрывается экран добавления аккаунта и показывается сообщение об успехе.

При условии, что был добавлен аккаунт Instagram, экран создания задачи тип задачи, выбрать фильтры для сбора, анализа и продвижения аккаунта, которые представляют из себя чекбоксы: поиск по хештегам, поиск по геометкам, поиск по подписчикам и подпискам (рис 2.6).

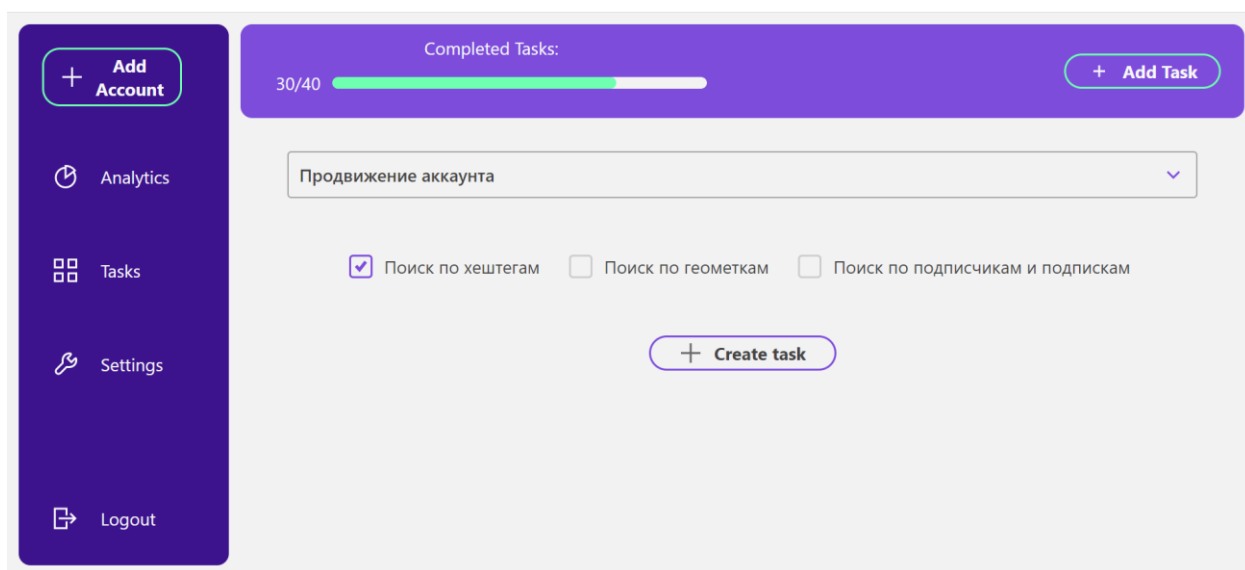


Рисунок 2.6 – Экран создания задачи

При нажатии на кнопку создания задачи, вызывается обработчик `handleCreateTask`.

```
const handleCreateTask = () => {
  createTask({
    type,
    filters,
  })
  .then(() => {
    redirectTo('/dashboard')
  })
  .catch((err) => setError({ inst: err }));
};
```

Обработчик `handleCreateTask` отправляет в API запрос с выбранными фильтрами и типом задачи. В случае успеха, открывается экран со всеми возможностями системы.

2.3 Разработка серверной части системы

Для разработки функций собственного API необходимо определиться с функционалом приложения (таблица 2.1).

Таблица 2.1 – Список задач, решаемых приложением

№	Задачи приложения	Маршрут роутинга
1	Авторизация	POST /auth
2	Авторизация аккаунта Instagram	POST /account/add
3	Получение задач пользователя	GET /tasks
4	Получение информации по определенной задаче	GET /task/:id

Продолжение таблицы 2.1

№	Задачи приложения	Маршрут роутинга
5	Создание задачи	POST /task/add
6	Удаление задачи	DELETE /task/:id

Исходя из данных таблицы 2.1 и организации проекта Nest JS структура проекта выглядит следующим образом (рисунок 2.7).

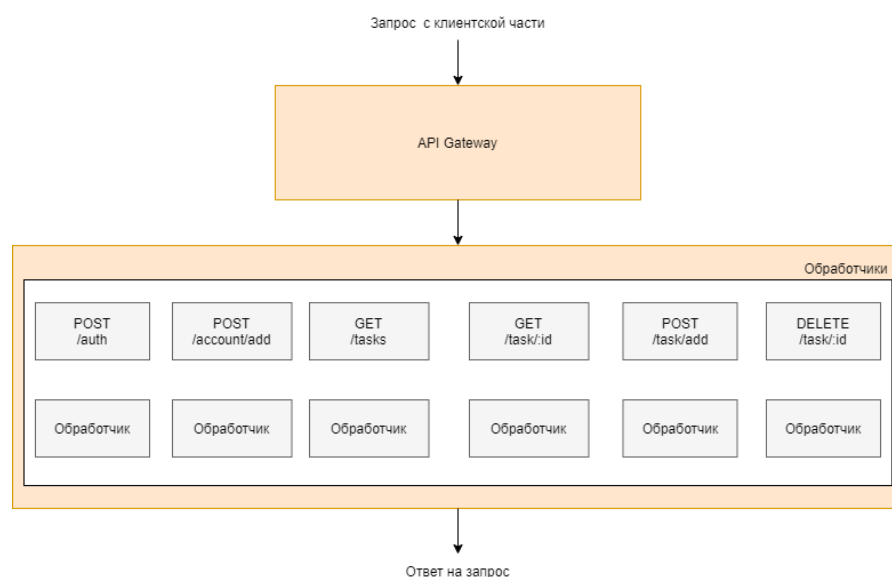


Рисунок 2.7 - Структура API приложения

Как показано на рисунке 2.7 все HTTP запросы от пользователей обрабатываются API Gateway, которая берет на себя функцию прокси сервера, сортирующего все запросы, разбирает URL HTTP запроса и формирует запрос к конкретному обработчику, для которого данный HTTP запрос предназначен.

В данном конкретном случае имеется шесть обработчиков, которые вызываются по запросу с клиентской части.

Сам обработчик в свою очередь представляет собой JS функцию (рисунок 2.8).

Каждая функция содержит пользовательский код и некоторые данные конфигурации, включая имя функции и параметры. Функции JS не сохраняют состояние и никак не зависят от базовой инфраструктуры, поэтому можно быстро загрузить столько копий функции, сколько нужно для масштабирования в соответствии с количеством входящих событий. В общем случае обработчик включает код пользователя, код для обработки служебных запросов и код для работы с СУБД MongoDB.

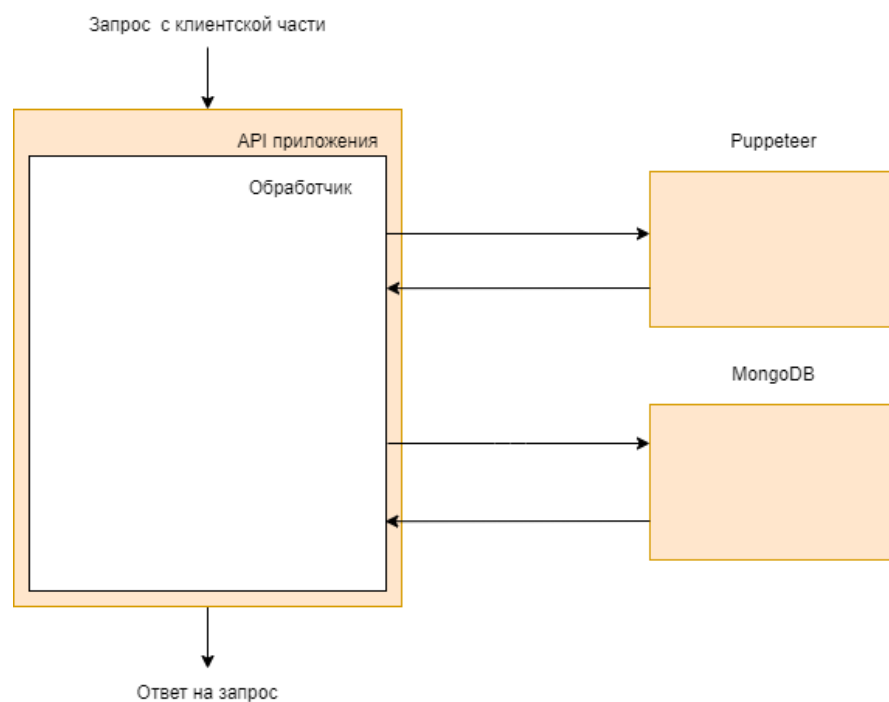


Рисунок 2.8 - Общая структура обработчиков

На рисунках 2.9 и 2.10 представлены конкретные примеры JS функций выполняющие задачи создания аналитических задач и их удаления из базы задач.

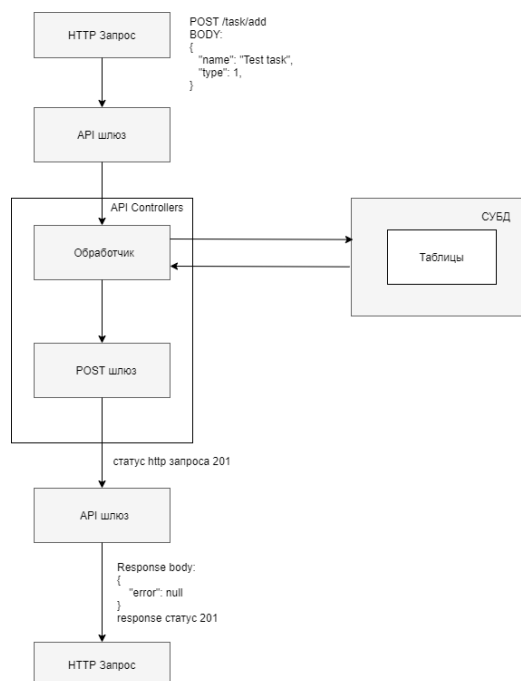


Рисунок 2.9 - Структура обработчика «добавление задачи»

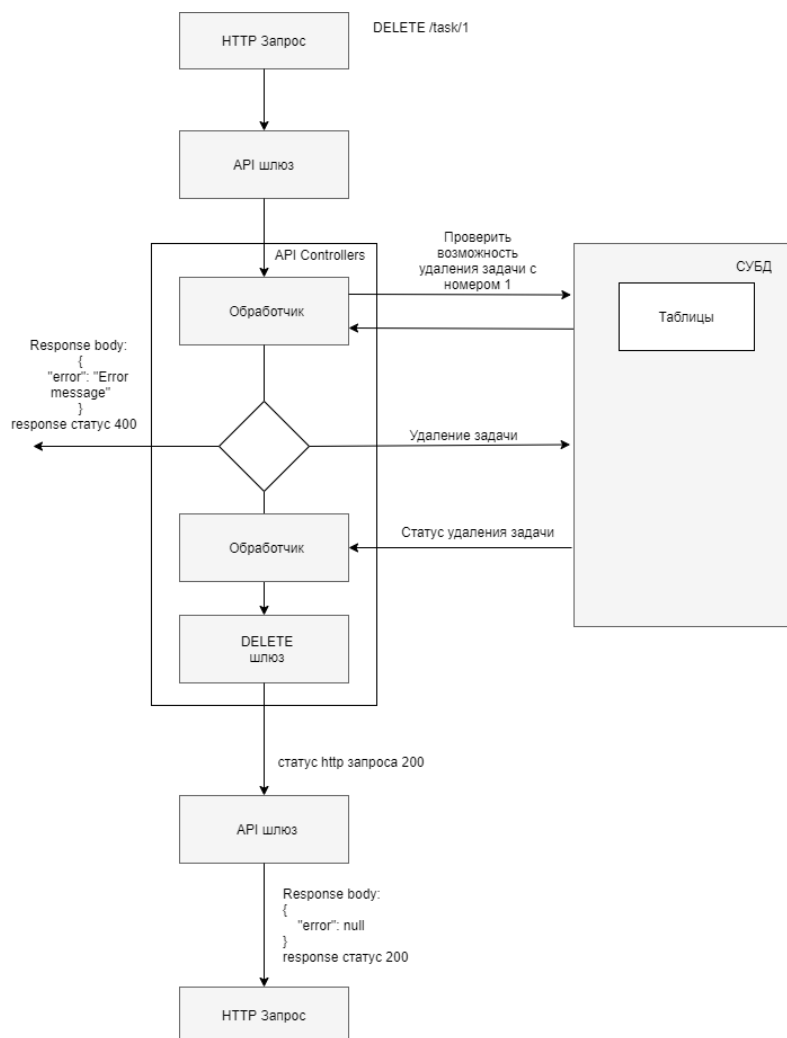


Рисунок 2.10 - Структура обработчика «удаление задачи»

Разработка обработчиков авторизации, подключения аккаунта Instagram, создание, редактирование, удаление задач для поиска и анализа потенциальных клиентов, анализ аккаунта Instagram (публикаций, историй, подписчиков и подписок) являются основой взаимодействия клиента и серверной части данного проекта, именно поэтому реализацию проекта начнем с создания маршрутов взаимодействия клиента и сервера.

Каждый обработчик получает в качестве аргумента ссылку на функцию, содержащую программный код, для обработки маршрута и URL ссылку на данные и/или структуры данных для обработки логики.

Обработчик login формируется в отдельном файле. Все обработчики проекта требуется поместить в подкаталог users/controllers (создается в проектах Nest JS автоматически, при создании сущности). Код обработчика login будет храниться в отдельном файле users.controller.ts.

В файле users.controller.ts будет класс UsersController, в котором обработчик login является методом класса, который делает запрос в БД и, если входные параметры верные, то создает сессию и возвращает токен аутентификации. Чтобы найти пользователя по входным параметрам, используется метод findOne, класса модели User MongoDB, принимающий объект из полей и значений, которые необходимо найти в базе, если MongoDB вернула данные, то пароль проверяется на совпадение с паролем, хранящемся в БД.

```
async function login({ username, password }) {  
  const user = await User.findOne({ username });  
  if (user && bcrypt.compareSync(password, user.hash)) {  
    const token = jwt.sign({ sub: user.id }, config.secret);  
    return {  
      ...user.toJSON(),  
      token  
    };  
  }  
}
```

В Nest JS точкой входа приложения по умолчанию является файл main.ts.

```
import { NestFactory, FastifyAdapter } from '@nestjs/core';  
import { ValidationPipe, Logger } from '@nestjs/common';  
import { SwaggerModule, DocumentBuilder } from '@nestjs/swagger';
```

```

import fastifyCors from 'fastify-cors';
import { AppModule } from './app.module';
import { config } from './config';
async function bootstrap() {
  const app = await NestFactory.create(AppModule, new
FastifyAdapter({
  trustProxy: true,
  }));
  const documentOptions = new DocumentBuilder()
    .setTitle(config.TITLE)
    .setDescription(config.DESCRPTION)
    .setVersion(config.VERSION)
    .setBasePath(`/${config.PREFIX}`)
    .build();
  const document = SwaggerModule.createDocument(app,
documentOptions);
  const validationOptions = {
    skipMissingProperties: true,
    validationError: { target: false },
  };
  /*-----*/
  app.useGlobalPipes(new ValidationPipe(validationOptions));
  app.setGlobalPrefix(config.PREFIX);
  app.register(fastifyCors, {
    origin: true,
  });
  SwaggerModule.setup(config.API_EXPLORER_PATH, app, document);
  await app.listen(config.PORT, config.HOST);

```

```

    Logger.log(`Server listening on port ${config.PORT}`,
    'Bootstrap');
  }
  bootstrap();

```

Из листинга выше видно, что система работает следующим образом, при запуске приложения пользователь попадает по умолчанию на стартовую страницу index.html. Далее нажимая как вариант на элементы управления и/или ссылки, он инициирует работу конкретного маршрута. В данном случае шесть функциональных маршрутов.

По умолчанию API-шлюз (main.ts) возвращает HTTP-статус 200 (201) для всех запросов. Синтаксис NodeJS позволяет помимо кодов об удачной работе маршрута отрабатывать и выдавать коды ошибок, в частности в нашем примере обрабатывается ошибка номер 404 (Данные не найдены). Данное свойство пригодится при обработке ошибок данных приложения или маршрутизации, т.к. именно этот статус ошибки сообщает о том, что были введены неверные входные параметры или, использовался неверный роут.

Функция подключения аккаунта Instagram в качестве параметров принимает логин и пароль от аккаунта Instagram. Если и тот и другой параметр проходят валидацию, то функция выполняется, сохраняя сессионную информацию в СУБД, если данные заполнены не верно или отсутствуют, функция возвращает значение null.

```

import { Injectable, Logger } from '@nestjs/common';
import * as puppeteer from 'puppeteer'
import { InstagramLoginDto, InstagramLookingDto } from
"../dto/inst.dto";
const requestUrl = 'https://www.instagram.com/'
const instHomeUrl = "

```

```

const loginUriSelector = '#react-root > section > main > article
> div.rgFsT > div:nth-child(2) > p > a'

const loginField = '#react-root > section > main > div > article
> div > div:nth-child(1) > div > form > div:nth-child(2) > div >
label > input'

const passwordField = '#react-root > section > main > div > article
> div > div:nth-child(1) > div > form > div:nth-child(3) > div >
label > input'

const loginButton = '#react-root > section > main > div > article
> div > div:nth-child(1) > div > form > div:nth-child(4) > button'

const bannerNotNowButton = 'body > div.RnEpo.Yx5HN > div > div >
div.mt3GC > button.aOOIW.HoLwm'

@Injectable()

export class InstAuthService {
  private readonly logger = new Logger('InstAuthController');
  private parseResponseLogin = async (page) => {
    let personalInfo = {
      cookies: null,
      body: null
    };
    const loginResponse = await
page.waitForResponse('https://www.instagram.com/accounts/login/a
jax/');

    const body = await loginResponse.json();

    if (body.authenticated) {
      const cookies = await page.cookies()

```

```

personalInfo = {
  cookies: cookies,
  body,
}
} else {
  this.logger.error(`Cant login:
  ${JSON.stringify(body)}`);
  personalInfo = {
    cookies: {},
    body: null
  }
}
return personalInfo
}

async login (user: InstagramLoginDto) {
  const browser = await puppeteer.launch({ headless: false
  })
  const page = await browser.newPage()
  await page.goto(requestUrl, { waitLoad: true,
  waitNetworkIdle: true })
  // Ожидаем и нажимаем на авторизацию
  await page.waitForSelector(loginUriSelector)
  page.click(loginUriSelector)
  //Ожидаем и вводим логин и пароль
  await page.waitForSelector(loginField)
  await page.type(loginField, user.login);
  await page.type(passwordField, user.password);

```

```
//Авторизируемся
```

```
page.click(loginButton)
```

```
const personalInfo = await this.parseResponseLogin(page)
```

```
this.logger.log(personalInfo)
```

```
return personalInfo
```

```
}
```

```
}
```

Как видно из листинга функции, в первую очередь подключается модули Nest JS. Далее подключается библиотека Puppeteer. Следующим шагом подготавливаются данные для парсинга страницы авторизации в социальной сети Instagram, запускается сессия google chrome. В открывшейся сессии google chrome происходит поиск полей для заполнения логина и пароля, подставляются данные из входных параметров и после этого происходит авторизация в сервисе.

Полученная сессия отправляется в СУБД, чтобы затем ее использовать для аналитических задач, созданных пользователем.

Функция получения данных с аккаунта Instagram в качестве аргумента принимает сессию аккаунта.

```
async parseAccount (session: InstagramSessionDto) {
```

```
  try {
```

```
    const currentSession = this.login(session)
```

```
    await currentSession.goto(instHomeUrl, { waitLoad: true,
waitNetworkIdle: true })
```

```

    await currentSession.waitForSelector(postSelector)

    const posts = currentSession.get(postSelector)

    return posts.map(post => {
        hashtag: post.get(hashtagSelector).split(' ')
    })
  } catch(e) {
    return e
  }
}

```

При старте выполнения этой функции запускается сессия google chrome, открывается веб-версия Instagram, подставляется в куки сессия аккаунта и, если авторизация при помощи сессии прошла успешно, то начинается парсинг данных (хештеги, геометки) с публикаций и других действий пользователя. Все это форматируется и сохраняется в базу данных. В случае, если авторизация не прошла, то функция возвращает ошибку авторизации.

Функция поиска аккаунтов по хештегам принимает в качестве аргументов сессию аккаунта и массив хештегов.

```

async parseHashtags (session: InstagramSessionDto, hashtags: HashTag[]) {
  try {
    const currentSession = this.login(session)

    await currentSession.goto(instHomeUrl, { waitLoad: true,
waitNetworkIdle: true })

    return hashtags.map(hashtag => {
      await currentSession.waitForSelector(searchSelector)

```

```

        await currentSession.type(searchSelector, hashtag);

        currentSession.click(searchButonSelector)

        await currentSession.waitForSelector(peopleSelector)
        const peoples = currentSession.get(peopleSelector)

        return peoples.map(people => {
            id: people.get(`${peopleSelector}-${people.id}`).find('id')
        })
    })
    } catch(e) {
        return e
    }
}

```

При старте выполнения этой функции запускается сессия google chrome, открывается веб-версия Instagram, подставляется в куки сессия аккаунта и, если авторизация при помощи сессии прошла успешно, то начинается поиск аккаунтов, которые публиковали записи или совершали действия с нужными нам хештегами. Все это форматируется и возвращается. В случае, если авторизация не прошла, то функция возвращает ошибку авторизации.

Функция поиска аккаунтов по геометкам принимает в качестве аргументов сессию аккаунта и массив геометок.

```

async parseGeotags (session: InstagramSessionDto, geotags: GeoTag[]) {
    try {
        const currentSession = this.login(session)

```

```

return GeoTag.map(async (geolocation) => {
    await    currentSession.goto(`${instGeoUrl}/${geolocation}`,    {
waitLoad: true, waitNetworkIdle: true })

    const peoples = currentSession.get(peopleSelector)

    return peoples.map(people => {
        id: people.get(`${peopleSelector}-${people.id}`).find('id')
    })
})
} catch(e) {
    return e
}
}

```

При старте выполнения этой функции запускается сессия google chrome, открывается веб-версия Instagram, подставляется в куки сессия аккаунта и, если авторизация при помощи сессии прошла успешно, то начинается поиск аккаунтов, которые находились, делали публикации или совершали другие действия с необходимыми нам координатами их местоположения. Все это форматируется и возвращается. В случае, если авторизация не прошла, то функция возвращает ошибку авторизации.

Функция продвижения аккаунта в Instagram принимает в качестве аргументов сессию аккаунта и фильтры.

```

async looking(session: InstagramSessionDto, filters: FilterDto[]) {
    try {
        const currentSession = this.login(session)

        let hashtags = this.parseHashTags();

```

```

let geotags = this.parseGeoTags();

for (let hl = 0; hl < hashtags.length; hl++) {

    await    page.goto('https://www.instagram.com/explore/tags/'    +
    hashtags[hl] + '/?hl=en');

    for (let r = 1; r < 4; r++) {
        for (let c = 1; c < 4; c++) {

            let br = false;

            await page.click('section > main > article > div:nth-child(3) >
div > div:nth-child(' + r + ') > div:nth-child(' + c + ') > a').catch(() => {
                br = true;
            });

            await    page.waitFor(2250    +    Math.floor(Math.random()    *
250));

            if (br) continue;

            let                hasEmptyHeart                =                await
page.$(cnf.selectors.post_heart_grey);

            let username = await page.evaluate(x => {
                let element = document.querySelector(x);
                return Promise.resolve(element ? element.innerHTML : '');
            }, cnf.selectors.post_username);

```

```

let followStatus = await page.evaluate(x => {
    let element = document.querySelector(x);
    return Promise.resolve(element ? element.innerHTML : "");
}, cnf.selectors.post_follow_link);

console.log('---> Evaluate post from ' + username);

if (hasEmptyHeart !== null && Math.random() <
cnf.settings.like_ratio) {
    await page.click(cnf.selectors.post_like_button);
    console.log('---> like for ' + username);
    await page.waitFor(10000 + Math.floor(Math.random() *
5000));
}

let isArchivedUser;
await ops.inArchive(username).then(() => isArchivedUser =
true).catch(() => isArchivedUser = false);

if (followStatus === 'Follow' && !isArchivedUser &&
Math.random() < cnf.settings.follow_ratio) {
    await ops.addFollow(username).then(() => {
        return page.click(cnf.selectors.post_follow_link);
    }).then(() => {
        console.log('---> follow for ' + username);
        return page.waitFor(10000 + Math.floor(Math.random()
* 5000));
    }).catch(() => {

```

```

        console.log('---> Already following ' + username);
    });
}

    await page.click(cnf.selectors.post_close_button).catch(() =>
console.log(':::> Error closing post'));
    }
}

}

    if (cnf.settings.do_unfollows) {

        let        cutoff        =        new        Date().getTime()        -
(cnf.settings.unfollow_after_days * 86400000);

        let follows = await ops.getFollows();
        let unfollows = [];

        follows.rows.forEach(user => {
            if (user.doc.added < cutoff) {
                unfollows.push(user.doc._id);
            }
        });

        for (let n = 0; n < unfollows.length; n++) {

            let user = unfollows[n];

            await page.goto('https://www.instagram.com/' + user + '?hl=en');

```

```

        await page.waitFor(1500 + Math.floor(Math.random() * 500));

        let followStatus = await page.evaluate(x => {
            let element = document.querySelector(x);
            return Promise.resolve(element ? element.innerHTML : "");
        }, cnf.selectors.user_unfollow_button);

        if (followStatus === 'Following') {
            console.log('---> unfollow ' + user);
            await page.click(cnf.selectors.user_unfollow_button);
            await page.waitFor(750);
            await
page.click(cnf.selectors.user_unfollow_confirm_button);

            ops.unFollow(user);
            await page.waitFor(15000 + Math.floor(Math.random() *
5000));

        } else {
            console.log('---> archive ' + user);
            ops.unFollow(user);
        }
    }
} catch (e) {
    return e
}
}

```

При старте выполнения этой функции запускается сессия google chrome, открывается веб-версия Instagram, подставляется в куки сессия

аккаунта и, если авторизация при помощи сессии прошла успешно, то вызываются функции в соответствии с фильтрами, формируется база пользователей и начинается проход по страничкам пользователей и привлечение внимания действиями с их публикациями. В случае, если авторизация не прошла, то функция возвращает ошибку авторизации.

2.4 Разработка базы данных

2.4.1 Разработка базы данных

База данных данного приложения, реализована при помощи фреймворка NestJS на базе MongoDB и объектов данных на основе JSON.

Стандартная структура NoSQL СУБД выглядит следующим образом:

```
[ {  
  "_id": 1,  
  "name" : { "first" : "John", "last" : "Backus" },  
  "contribs" : [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ],  
  "awards" : [ { "award" : "W.W. McDowell Award", "year" : 1967, "by" :  
    "IEEE Computer Society" }, { "award" : "Draper Prize", "year" : 1993, "by" :  
    "National Academy of Engineering" } ]  
}
```

В ситуации с MongoDB нам необходимо определить модели сущностей:

User – модель Пользователя:

```
import { prop, Typegoose } from '@typegoose/typegoose';  
export class User extends Typegoose {  
  @prop()  
  id: string;
```

```

    login:string;
    password: string
  }

```

InstAccount – модель аккаунта Instagram:

```

import { prop, Typegoose } from '@typegoose/typegoose';
export class InstAccount extends Typegoose {
  @prop()
  userId: string;
  session: string;
}

```

Task – модель задачи:

```

import { prop, Typegoose } from '@typegoose/typegoose';
export class InstAccount extends Typegoose {
  @prop()
  userId: string;
  id: string;
  progress: TTaskProgress;
  filters: TFilter[];
  type: TTaskType;
}

```

2.4.2 Разработка обработчиков базы данных

Преимуществом Nest JS является подключение и свободное использование автономных модулей, практически, для любых ситуаций.

Подключаем разработанные модели в модули нашего приложения и задаем окружение для доступа к базе данных.

```

import { Module } from '@nestjs/common';

```

```
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { UserModule } from './user/user.module';
import { TypegooseModule } from 'nestjs-typegoose';
@Module({
  Imports: [TypegooseModule.forRoot('mongodb://localhost:27017/nest
'), UserModule], controllers: [AppController],
  providers: [AppService], }) export class AppModule {}
```

Описываем наши контроллеры.

```
import { Injectable } from '@nestjs/common';
import { User } from './domain/user';
import { InjectModel } from 'nestjs-typegoose';
import { ReturnModelType } from '@typegoose/typegoose';
@Injectable()
export class UserService {
  constructor(@InjectModel(User) private readonly userModel:
    ReturnModelType<typeof User>) {}
  async createUser(user: User) {
    const createdUser = new this.userModel(user); return
    await createdUser.save();
  }
  async listUsers(): Promise<User[] | null> {
    return await this.userModel.find().exec();
  }
}
```

Удобство использования именно этой технологии заключается в том, что нам необходимо описать только логику на JS, а формирование запросов и

сами запросы, а также валидацию производит сам фреймворк Nest JS и его модули. Так и на примере нашей ситуации, мы описали сущность User, описали логику для авторизации и теперь, если с клиента придет запрос по роуту /login, то вызовется наш контроллер и найдет пользователя с входными параметрами среди данных в MongoDB.

					ПГУ 1.090501.15.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		44

3 Тестирование и развертывание программного обеспечения

3.1 Тестирование программного обеспечения

Тестирование и написание тест-кейсов обеспечивает проверку качества приложения и является доказательством того, что приложение работает так, как было задумано на этапе проектирования.

Все виды тестирования ПО, в зависимости от целей тестирования можно разделить на 3 группы:

- Функциональные;
- Нефункциональные;
- Связанные с изменениями;

Функциональные виды тестирования используются для написания тестов на функционал системы и могут быть представлены на следующих уровнях тестирования:

- Компонентный или модульный (Component/Unit testing);
- Интеграционный (Integration testing);
- Системный (System testing);
- Приемочный (Acceptance testing).

Нефункциональные виды тестирования используются для определения характеристик системы.

Связанные с изменениями виды тестирования используются для тестирования ПО после изменений в кодовой базе или в логике работы.

По степени автоматизации тестирование можно разделить на два вида:

- Ручное тестирование;
- Автоматизированное тестирование;

Ручное тестирование – тестирование, при котором тестировщик не использует дополнительное ПО, а составляет тест кейсы по ТЗ и, используя их, проходит по приложению как обычный пользователь.

Автоматизированное тестирование – тестирование, при котором используются дополнительные программные средства.

Для того, чтобы протестировать систему, мы будем использовать ручное и интеграционное тестирования.

Для ручного тестирования необходимо разработать тест кейсы и выполнить их, тем самым доказав работу системы (табл. 3.1).

Таблица 3.1 – План тестирования

№	Название теста	Действие	Ожидаемый результат	Полученный результат
1	Ввод неверных данных при авторизации	Пользователь вводит неверный логин или пароль на экране авторизации и нажимает кнопку Войти	Появляется сообщение с ошибкой “Введен неверный логин или пароль”	Соответствует ожидаемым результатам (рисунок 3.1)
2	Ввод корректных данных при авторизации	Пользователь вводит корректные логин и пароль на экране авторизации	У пользователя автоматически появляется экран с основным функционалом системы	Соответствует ожидаемым результатам (рисунок 3.2)
3	Добавление аккаунта Instagram с неверными данными	На экране добавления аккаунта Instagram пользователь вводит неверные логин или пароль	Появляется сообщение с ошибкой “Ошибка авторизации. Проверьте логин и пароль”	Соответствует ожидаемым результатам (рисунок 3.3)

Продолжение таблицы 3.1

№	Название теста	Действие	Ожидаемый результат	Полученный результат
4	Добавление аккаунта Instagram с верными данными	На экране добавления аккаунта Instagram пользователь вводит верные логин или пароль	Появляется сообщение об успешном добавлении аккаунта	Соответствует ожидаемым результатам (рисунок 3.4)
5	Создание задачи без добавленного аккаунта Instagram	Пользователь, у которого нет добавленного аккаунта Instagram нажимает на кнопку Создать задачу	Появляется модальное окно, в котором сообщается о невозможности создания задачи без добавленного аккаунта Instagram и предложении его добавить.	Соответствует ожидаемым результатам (рисунок 3.5)
6	Создание задачи с добавленным аккаунтом Instagram	Пользователь, у которого есть аккаунт Instagram нажимает кнопку Создать задачу	Появляется экран создания задачи	Соответствует ожидаемым результатам (рисунок 3.6)

Выполняя действия, предполагаемые тест кейсом №1, ожидаем увидеть на экране авторизации ошибку “Введен неверный логин или пароль” (рис. 3.1)

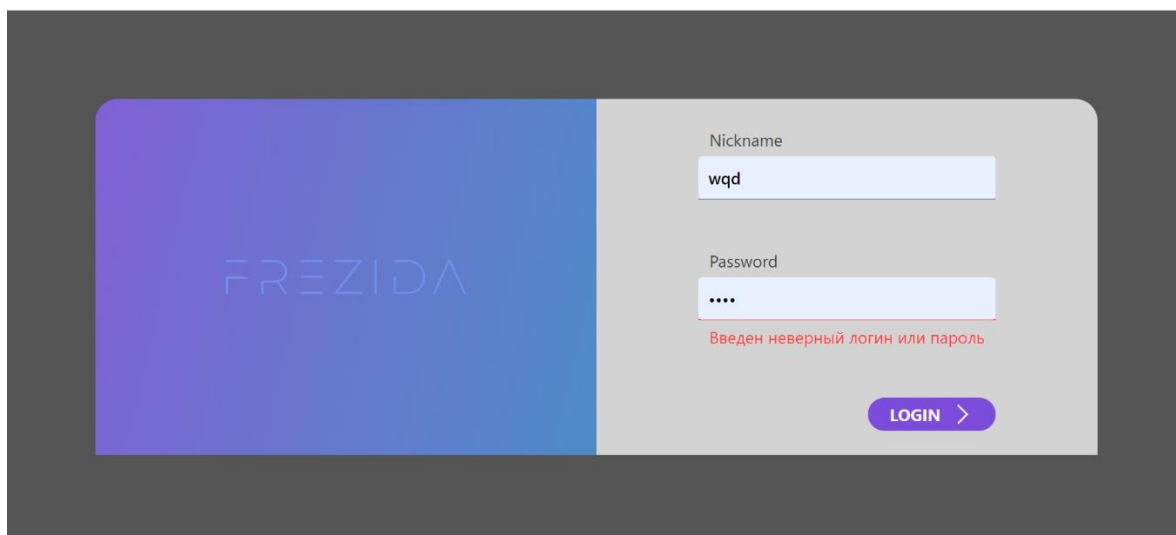


Рисунок 3.1 – Ошибка авторизации

Выполняя действия, предполагаемые тест кейсом №2, ожидаем, что не будет ошибок и появится экран, предоставляющий основной функционал системы (рис. 3.2)

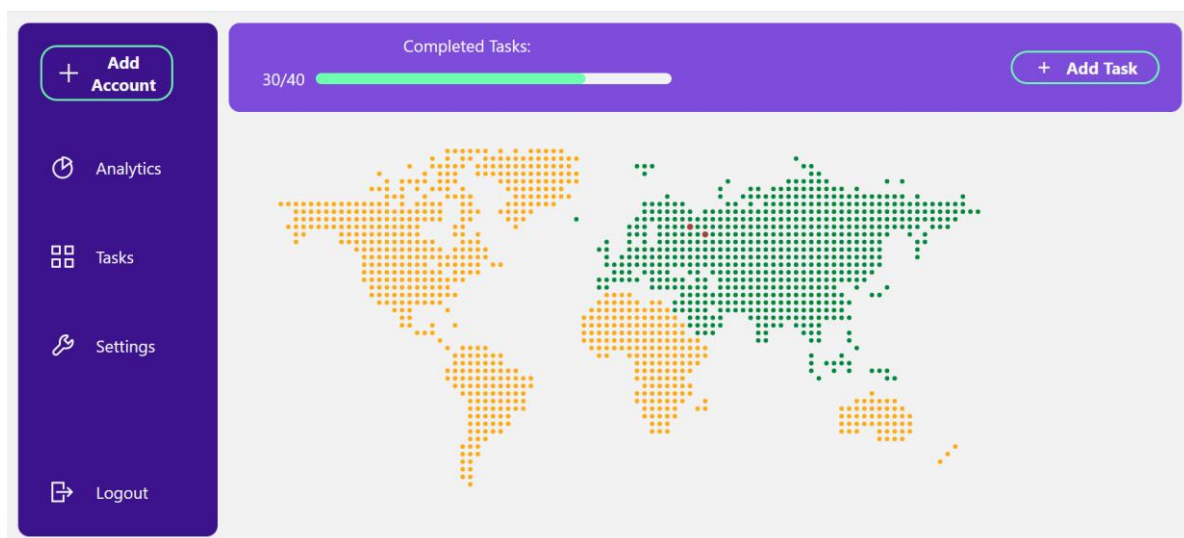


Рисунок 3.2 – Экран с основным функционалом

Выполняя действия, предполагаемые тест кейсом №3, ожидаем увидеть на экране добавления аккаунта Instagram ошибку авторизации (рис. 3.3)

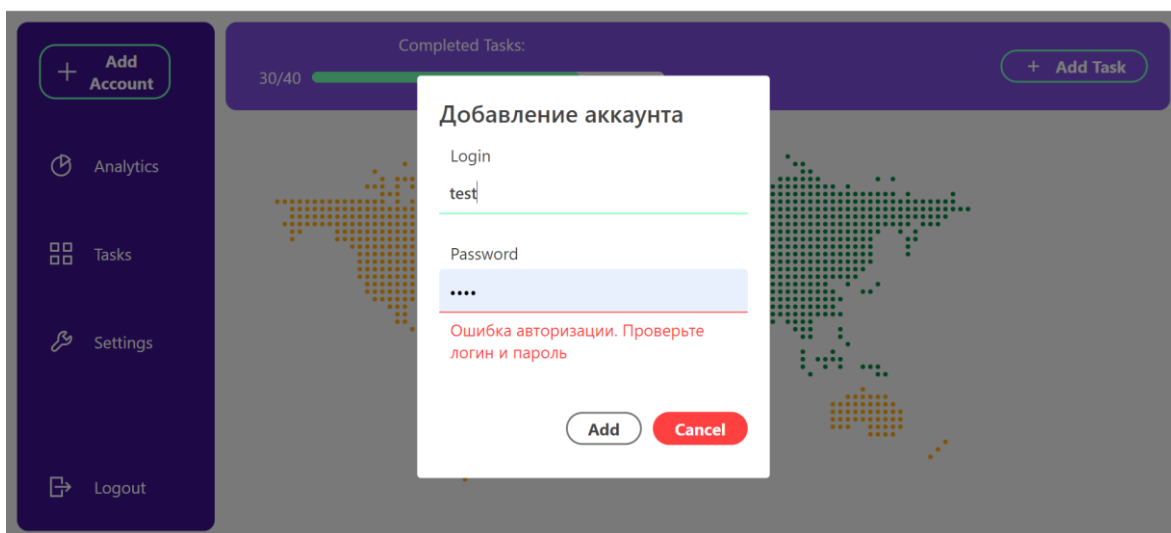


Рисунок 3.3 – Ошибка добавления аккаунта

Выполняя действия, предполагаемые тест кейсом №4, ожидаем увидеть сообщение об успешном добавлении аккаунта (рис. 3.4)

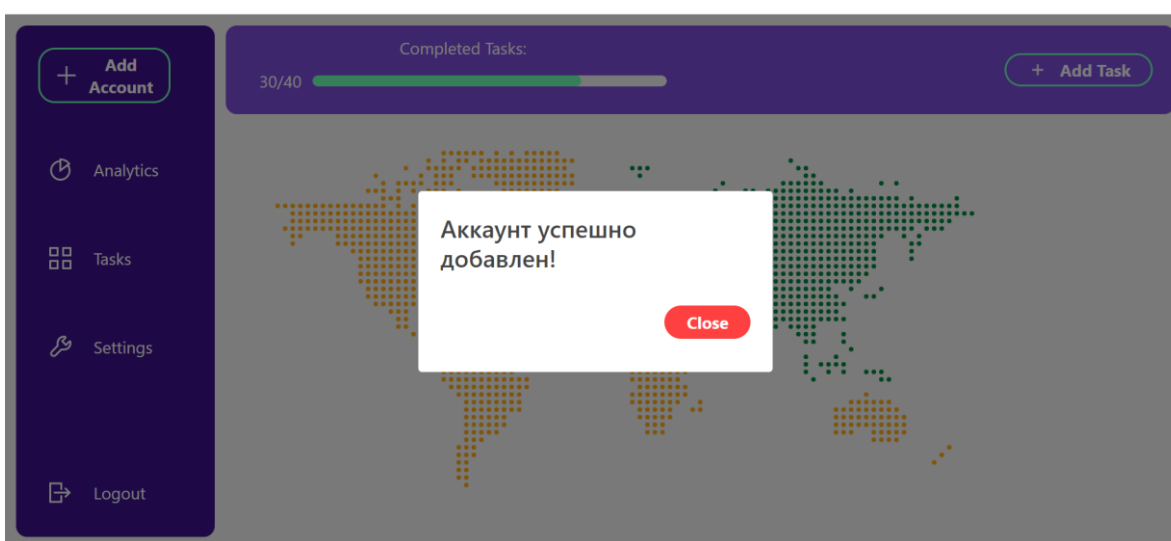


Рисунок 3.4 – Сообщение об успешном добавлении аккаунта

Выполняя действия, предполагаемые тест кейсом №5, ожидаем увидеть на сообщение о невозможности создания задачи без добавленного аккаунта Instagram (рис. 3.5)

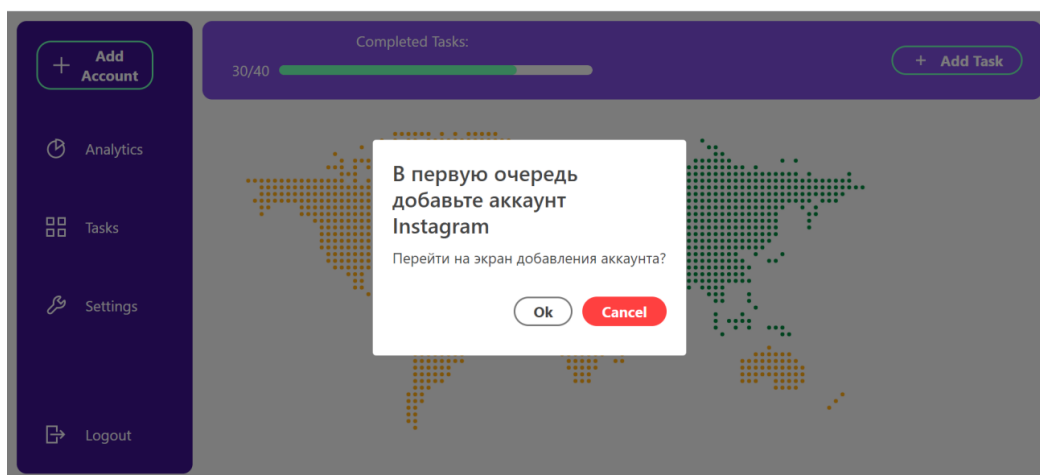


Рисунок 3.5 – Ошибка создания задачи

Выполняя действия, предполагаемые тест кейсом №6, ожидаем увидеть экран создания задачи (рис. 3.6)

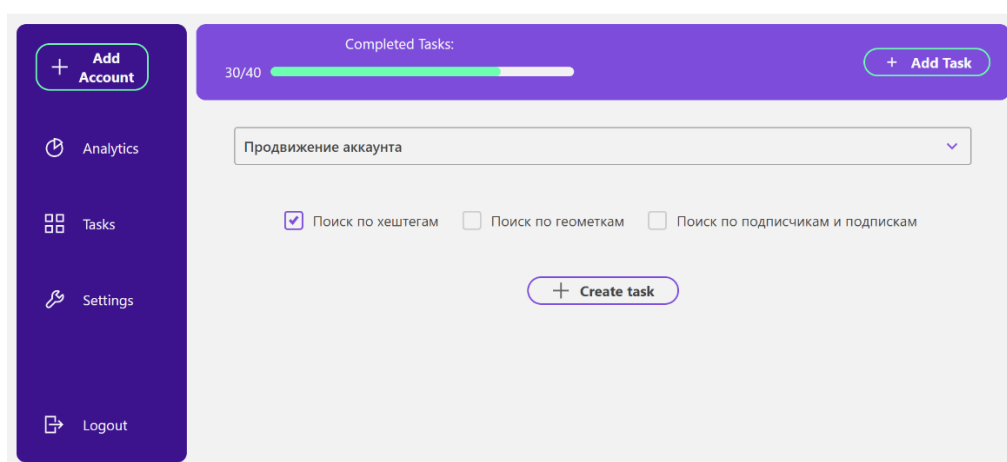


Рисунок 3.6 – Экран создания задачи

Для того, чтобы автоматически проверить работу клиентской и серверной частей системы, необходимо написать интеграционные тесты. Для этого будем использовать JS библиотеку Puppeteer.

Для того, чтобы убедиться, что клиентская часть запущена, напомним следующий тест:

```
test('Фронтенд запущен', async () => {
```

```

const client_url = 'localhost:4000';

let browser = await puppeteer.launch();
let page = await browser.newPage();

await io({
  page,
  async input() {
    await page.goto(client_url);
  },
  async output({ message }) {
    await message("Client has been opened");
  }
});

await page.close();
await browser.close();
});

```

В данном тест-кейсе мы оборачиваем открытие сайта callback функцию input, которая вызовется у метода тестировщика puppeteer в начале выполнения теста. Если на данной странице будет сообщение Client has been opened, то тест-кейс был успешно пройден, и система его автоматически засчитает успешным, иначе произошла ошибка и тест-кейс упал.

Для того, чтобы убедиться, что серверная часть запущена и работает, напомним следующий тест:

```

test(`Бекенд запущен и работает`, async () => {
  const backend_url = 'localhost:7071';

```

```

let browser = await puppeteer.launch();
let page = await browser.newPage();
await page.setRequestInterception(true);
page.on('response', resp => {
  if (response.url().endsWith("/login"))
    await io({
      page,
      input() {
      },
      async output({ message }) {
        expect(response.status()).toBe(200)
      }
    });
});

await page.evaluate(() => {
  fetch("/login", {
    method: "POST",
    body: { login: test, password: test }
  });
});

await page.close();
await browser.close();
});

```

В данном тест-кейса мы проверяем работу метода API login. Перехватываем ответ от запроса /login, и эмулируем сам запрос, с корректными данными пользователя, если статус ответа равен 200, то тест успешно пройден, иначе тест упадет.

3.2 Развертывание программного обеспечения

Система использует клиент-серверную архитектуру.

Клиентская часть служит для отображения сайта в браузере пользователей. Так как она написана на NextJS, перед запуском необходимо установить все модули, используемые в системе командой `yarn` из корня проекта. Затем запустить ее командой `yarn start`. Клиентская часть запускается на локальном ip на порте 4000.

Серверная часть представляет из себя базу данных и API. Так как она написана на NestJS, перед запуском необходимо установить `node.js` последней версии, затем установить все модули командой `yarn` из корня проекта и запустить сервер командой `yarn start`. Серверная часть запускается на локальном ip на порте 7071.

Диаграмма развертывания представлена на рисунке 3.7.

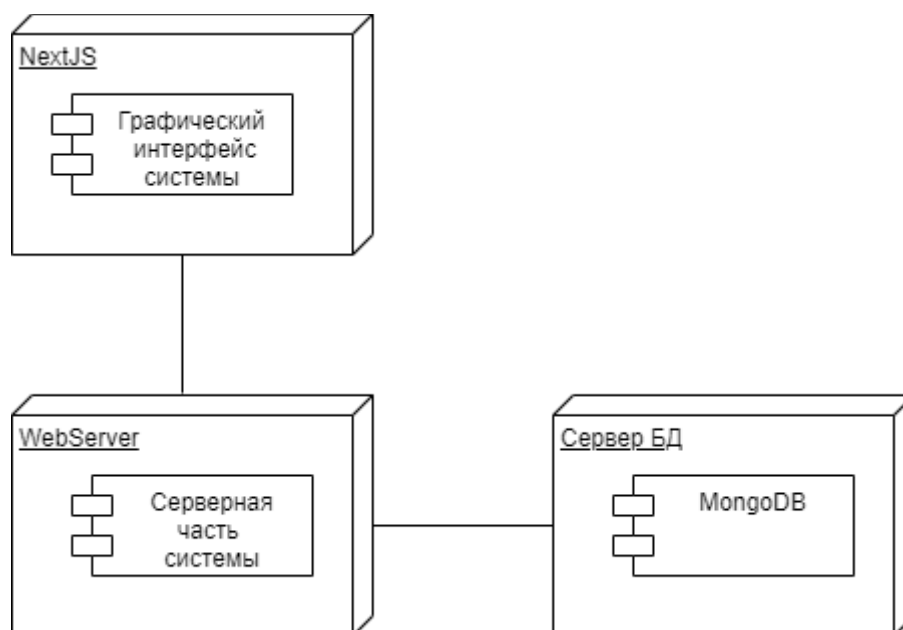


Рисунок 3.7 – Диаграмма развертывания

4 Технико-экономическое обоснование разработки программы

4.1 Введение

Целью технико-экономического обоснования разработки является количественное и качественное доказательство экономической целесообразности усовершенствования программы, а также определение организационно-экономических условий ее эффективного функционирования.

Эффективность программного модуля определяется его качеством и эффективностью процесса разработки и сопровождения. Качество программного изделия определяется тремя составляющими:

- с точки зрения специалиста-пользователя данного программного продукта;
- с позиции использования ресурсов и их оценки;
- по выполнению требований на программное изделие.

Программное изделие должно быть разработано так, чтобы оно выполняло свои функции без лишних затрат ресурсов (оперативной памяти ПК, машинного времени, пропускной способности каналов передачи данных и др. – на стадии функционирования; времени разработки и денежных ресурсов – на стадии использования программного изделия).

Содержание технико-экономического обоснования разработки программы заключается в следующем:

- расчет технико-экономических показателей и выбор базы сравнения;
- определение трудоемкости и стоимости программного обеспечения (ПО);
- расчет цены ПО;
- расчет капитальных и эксплуатационных затрат на разработку;

- определение показателей финансово-экономической эффективности.

Приведенные в данном разделе дипломной работы результирующие таблицы, перечни данных, позволяют сопоставить результаты разработки и затраты на нее, чтобы сделать вывод об эффективности проекта. Исходные данные для расчета экономических показателей приведены в таблице 4.1.

Таблица 4.1. Исходные данные для расчета экономических показателей

Обозначение	Наименование показателя	Единицы измерения	Значение показателя		
С _{ЭВМ}	Стоимость ЭВМ	тыс. руб.	20		
Д _м	Среднее количество дней в месяце	дни	22		
ρ _н	Норматив рентабельности	—	0,24		
ω _д	Коэффициент, учитывающий дополнительную заработную плату разработчика программы	—	0,20		
ω _с	Коэффициент, учитывающий начисления органам социального страхования	—	0,268		
ω _н	Коэффициент, учитывающий накладные расходы организации	—	1,35		
q _г	Количество I-задач, решаемых потребителем	зад. год	60		
t _{м.в.г}	Время решения I-ой задачи разработанной программой	маш. час	1		
n _п	Количество организаций, которые приобретут данную программу	шт.	1г.	2г.	3г.
			8	10	15

Продолжение таблицы 4.1

Z _{эл}	Тариф за 1 кВт/час	руб.	3,44
T _с	Срок службы разработанной программы	год	3
НДС	Налог на добавленную стоимость	%	20
T _р	Количество рабочих дней в году	дн	264
N _{см}	Количество смен работы ЭВМ	—	1
t _{см}	Продолжительность смены	ч	8
α	Простои ЭВМ	%	5
P	Мощность, потребляемая ЭВМ	кВт	0,4
N _{ср}	Среднее количество ремонтов в год	—	2
S _д	Стоимость деталей, заменяемых при ремонте	руб.	1300

4.2 Расчет затрат на разработку программного обеспечения

Суммарные затраты на разработку программы рассчитываются по следующей формуле:

$$S_{\text{рп}} = S_{\text{зп}} + S_{\text{нак}}, \quad (5.1)$$

где:

$S_{\text{зп}}$ — затраты по заработной плате инженера-программиста;

$S_{\text{нак}}$ — накладные расходы.

Затраты по заработной плате инженера-программиста рассчитываются по формуле:

$$S_{\text{ЗП}} = \text{ОЗП} \cdot (1+\omega_{\text{с}}) \cdot (1+\omega_{\text{д}}) \cdot t_{\text{pi}}, \quad (5.2)$$

где:

ОЗП – основная заработная плата инженера-программиста за месяц (25000 руб.);

t_{pi} – время, необходимое для разработки программы программистом i -го разряда (чел.-мес.);

$\omega_{\text{д}}$ – коэффициент, учитывающий дополнительную заработную плату разработчика программы, в долях к сумме основной заработной платы;

$\omega_{\text{с}}$ – коэффициент, учитывающий начисления органам социального страхования на заработную плату разработчика программы, в долях к сумме основной заработной плате разработчика.

Программа разрабатывалась 30 дней, если учесть, что в одном месяце 22 рабочих дня, то:

$$1,36 \cdot 22 \cdot 30 \cdot t_{\text{P12}} = (\text{чел. –мес.}) \quad (5.3)$$

Таким образом, затраты по заработной плате инженера программиста составят:

$$S_{\text{ЗП}} = 25000 \cdot (1+0,268) \cdot (1+0,20) \cdot 1,36 = 51408 \text{ (руб.)}$$

Накладные затраты рассчитываются с учетом $\omega_{\text{н}}$ – коэффициента, определяющего уровень накладных расходов организации по формуле:

$$S_{\text{НАК}} = \text{ОЗП} \cdot \omega_{\text{н}} \cdot t_{\text{p12}}, \quad (5.4)$$

$$S_{\text{НАК}} = 25000 \cdot 1,35 \cdot 1,36 = 45900 \text{ (руб.)}$$

Таким образом, суммарные затраты на разработку программного обеспечения составляют:

$$S_{\text{РП}} = 51408 + 45900 = 97308 \text{ (руб.)}$$

4.3 Расчет затрат на разработку программного обеспечения

Оптовая цена разработанной программы определяется по следующей формуле:

$$Z_{\Pi} = S_{\text{РП}} + \Pi, \quad (5.5)$$

где:

Z_{Π} – оптовая цена (цена разработчика) (руб.);

$S_{\text{РП}}$ – суммарные затраты на разработку программы (руб.);

Π – прибыль, рассчитанная по формуле:

$$\Pi = \rho_{\text{Н}} \cdot S_{\text{РП}}, \quad (5.6)$$

где:

$\rho_{\text{Н}}$ – норматив рентабельности, учитывающий прибыль организации, разрабатывающей данную программу в долях ко всем затратам данной организации на разработку программы.

Итак,

$$Z_{\Pi} = S_{\text{РП}} \cdot (1 + \rho_{\text{Н}}), \quad (5.7)$$

$$Z_{\Pi} = 97308 \cdot (1 + 0,24) = 120661,92 \text{ (руб.)}$$

Розничная цена программы рассчитывается с учетом налога на добавленную стоимость (НДС = 20%) по формуле:

$$Z_{\text{Пр}} = Z_{\Pi} + \text{НДС} = Z_{\Pi} + 100 Z_{\Pi} \cdot 20 = Z_{\Pi} \cdot (1 + 0,20), \quad (5.8)$$

$$Z_{\text{Пр}} = 120661,92 \cdot (1 + 0,20) = 144794,304 \text{ (руб.)}$$

Выручка от продаж при условии n_{Π} – количество организаций, желающих приобрести программу, составит:

$$B = Z_{\text{Пр}} \cdot n_{\Pi}, \quad (5.9)$$

$$B_1 = 144794,304 \cdot 8 = 1158354,432 \text{ (руб.)}$$

$$B_2 = 144794,304 \cdot 10 = 1447943,04 \text{ (руб.)}$$

$$B_3 = 144\,794,304 \cdot 15 = 2\,171\,914,56 \text{ (руб.)}$$

4.4 Расчет капитальных вложений

Капиталовложения, связанные с работой ЭВМ рассчитываются по формуле:

$$K_{ЭВМ} = C_{ЭВМ} + S_3, \quad (5.10)$$

где:

$C_{ЭВМ}$ – стоимость ЭВМ (руб.);

S_3 – стоимость запасных частей (руб.);

Итак, произведем расчет коэффициентов входящих в формулу расчета величины капиталовложений:

$$S_3 = 0,15 \cdot C_{ЭВМ}; \quad (5.11)$$

$$S_3 = 0,15 \cdot 20000 = 3000 \text{ (руб.)};$$

Капиталовложения в ЭВМ составляют:

$$K_{ЭВМ} = 20000 + 3000 = 23000 \text{ (руб.)}$$

4.5 Расчет эксплуатационных расходов

Эксплуатационные расходы на ЭВМ рассчитываются по формуле:

$$E = (T_{м.в.} \cdot e_ч) + \frac{Z_{п}}{T_c}, \quad (5.12)$$

где:

$T_{м.в.}$ – машинное время для решения задач с помощью разработанной программы, (маш.час/год);

$e_{ч}$ – эксплуатационные расходы, приходящиеся на 1 час работы ЭВМ;

$Z_{п}$ – цена, по которой продается программа (руб.);

$T_{с}$ – срок службы программы (г).

Полезный фонд времени работы ЭВМ рассчитывается по формуле:

$$T_{пол} = T_{общ} \cdot t_{см} \cdot N_{см} \cdot (1 - \frac{\alpha}{100}), \quad (5.13)$$

где:

$T_{общ}$ – общий фонд времени работы ЭВМ (дни);

$T_{общ} = TP$;

$N_{см}$ – количество смен работы ЭВМ;

$t_{см}$ – время одного рабочего дня (час);

α – простои ЭВМ (в % от общего фонда времени работы ЭВМ).

Полезный фонд времени работы ЭВМ получим:

$$T_{пол} = 264 \cdot 8 \cdot 1 \cdot (1 - \frac{5}{100}) = 2006,4 \text{ (Маш. час /год)}$$

Машинное время для решения задач с помощью данной программы рассчитывается по формуле:

$$T_{м.в.} = q_I \cdot t_{м.в.I}, \quad (5.14)$$

где:

q_I – количество I-задач, решаемых потребителем в год(шт.);

$t_{м.в.I}$ – время решения I-ой задачи, разработанной программой (Маш. час).

$$T_{м.в.} = 60 \cdot 1 = 60 \text{ (Маш. час / год)}.$$

Эксплуатационные расходы, приходящиеся на 1 час работы ЭВМ, оцениваются по формуле:

$$e_{\text{ч}} = \frac{(A_0 + S_{\text{ЗП}} + S_{\text{ЭЛ}} + R_{\text{РМ}})}{T_{\text{ПОЛ}}}, \quad (5.15)$$

где:

A_0 – амортизационные отчисления (руб.);

$S_{\text{ЗП}}$ – затраты по заработной плате инженера в год (руб./год);

$S_{\text{ЭЛ}}$ – стоимость потребляемой энергии (руб.);

$R_{\text{РМ}}$ – затраты на ремонт ЭВМ (руб.);

$T_{\text{ПОЛ}}$ – полезный годовой фонд работы ЭВМ, (маш.час/год).

Амортизационные отчисления рассчитываются с учетом нормы амортизации ($a_n = 12,5 \%$);

$$A_0 = a_n \cdot K_{\text{ЭВМ}} = 0,125 \cdot 23000 = 2875,0 \text{ (руб.)}. \quad (5.16)$$

Затраты по заработной плате инженера за год рассчитывается по формуле:

$$S_{\text{ЗП}} = (1 + \omega_c) \cdot (1 + \omega_d) \cdot O_{\text{ЗП}} \cdot 12, \quad (5.17)$$

где:

ω_c – коэффициент, учитывающий начисления органам социального страхования на заработную плату разработчика программы, в долях к сумме основной заработной плате разработчика.

ω_d – коэффициент, учитывающий дополнительную заработную плату разработчика программы, в долях к сумме основной заработной платы;

$O_{\text{ЗП}}$ – основная заработная плата инженера за месяц (25000 руб.);

Рассчитываем годовые затраты по заработной плате и социальным отчислениям для инженера:

$$S_{\text{ЗП}} = (1 + 0,268) \cdot (1 + 0,20) \cdot 25000 \cdot 12 = 456480 \text{ (руб./ год)}$$

Стоимость потребляемой энергии оценивается по формуле:

$$S_{\text{ЭЛ}} = P \cdot T_{\text{ПОЛ}} \cdot Z_{\text{ЭЛ}}, \quad (5.18)$$

где:

P – мощность, потребляемая ЭВМ (кВт);

$T_{\text{пол}}$ – полезный годовой фонд работы ЭВМ (маш.час/год);

$Z_{\text{эл}}$ – тариф за 1 кВт/час (руб. /кВт).

Итак, произведем расчет стоимости потребляемой энергии:

$$S_{\text{эл}} = 0,4 \cdot 2006,4 \cdot 3,44 = 2760,81 \text{ (руб.)}.$$

Затраты на ремонт ЭВМ вычисляются по формуле:

$$R_{\text{рм}} = N_{\text{ср}} \cdot S_{\text{д}}, \quad (5.19)$$

где:

$N_{\text{ср}}$ – среднее количество ремонтов в год;

$S_{\text{д}}$ – стоимость деталей заменяемых при одном ремонте, в среднем.

$$R_{\text{рм}} = 2 \cdot 1300 = 2600 \text{ (руб.)}.$$

Произведем вычисление эксплуатационных расходов, приходящихся на 1 час работы ЭВМ:

$$e_{\text{ч}} = \frac{(2875 + 182592 + 2760,81 + 2600)}{2006,5} = 95,12 \text{ (руб/час)}$$

Далее вычислим эксплуатационные расходы на ЭВМ:

$$E = (60 \cdot 95,12) + \frac{48426,7}{3} = 21849,43 \text{ (руб.)}.$$

4.6 Определение расходов на интернет трафик

Для облегчения работы программиста и повышения его производительности в проекте предусмотрено использование глобальной сети интернет.

В настоящее время на пензенском рынке интернет очень много провайдеров предлагающие свои услуги. Цены, предлагаемые основными

провайдерами, сведем в таблицу 4.2, для анализа и экономически целесообразного выбора поставщика услуги по интернет трафику.

Таблица 4.2 Таблица интернет трафиков и провайдеров

Название провайдера	Предлагаемый тариф	Стоимость тарифа (руб.)	Скорость доступа (Мбит/с)
Ростелеком	Пакет-Для впечатлений	800	100
ПАО «МТС»	ФИТ Интернет	650	100
Дом.РУ	Скорость света	700	100

Т.к. вышеперечисленные провайдеры предлагают без лимитные интернет трафики с одинаковой скоростью доступа в сеть интернет, то экономически целесообразно выбрать трафик с самой низкой стоимостью. Выбираем ПАО "МТС", тариф ФИТ Интернет, с абонентской платой в месяц 650 рублей

4.7 Заключение

Обобщенные технико-экономические показатели разработки программы сведены в таблицу 5.3

Таблица 5.3 Итоговая таблица

Показатель	Значение
Капитальные вложения (руб.)	23000
Эксплуатационные расходы (руб.)	21849,43
Затраты на разработку ПО (руб.)	97308
Затраты на интернет трафик (руб.)	650
Затраты на заработную плату программисту (руб.)	51408

5 Безопасность жизнедеятельности. Эргономика рабочего места программиста.

5.1 Характеристика условий труда программиста

Технический прогресс глобально поменял условия производственной деятельности сотрудников умственного труда. Их труд стал более интенсивнее, требующим больших затрат умственной, эмоциональной и физической энергии. Это потребовало системного решения проблем эргономики, гигиены и организации труда, регламентации режимов труда и отдыха.

В настоящее время компьютеры широко применяется во всех областях деятельности человека. При работе с компьютером, человек подвергается воздействию ряда вредных электромагнитных полей (ВЧ, УВЧ и СВЧ), шума и вибрации, статического электричества и др.

Работа с компьютером характеризуется значительным умственным напряжением и нервно-эмоциональной нагрузкой операторов, высокой напряженностью зрительной работы и большой нагрузкой на мышцы рук при работе с клавиатурой. Большое значение имеет рациональная конструкция и расположение элементов рабочего места, что важно для поддержания оптимальной рабочей позы программиста.

В процессе работы с компьютером необходимо соблюдать правильный режим труда и отдыха. В противном случае у персонала отмечаются болезненное напряжение зрительного аппарата с появлением жалоб на неудовлетворенность работой, головные боли, раздражительность, нарушение сна, усталость и болезненные ощущения в глазах, в пояснице, в области шеи и руках.

Одним из критериев, по которому определяется «пригодность» рабочего места для постоянного пребывания сотрудников, является эргономические требования к рабочему месту и освещенность рабочего места программиста.

5.2 Эргономические требования к рабочему месту программиста

Проектирование рабочих мест, снабженных мониторами, относится к числу важных проблем эргономического проектирования в области вычислительной техники.

Рабочее место и взаимное расположение всех его элементов должно соответствовать антропометрическим, физическим и психологическим требованиям. Большое значение имеет также характер работы. В частности, при организации рабочего места программиста должны быть соблюдены следующие основные условия: оптимальное размещение оборудования, входящего в состав рабочего места и достаточное рабочее пространство, позволяющее осуществлять все необходимые движения и перемещения.

Эргономическими аспектами проектирования рабочих мест программиста, являются:

1. высота рабочей поверхности,
2. размеры пространства для ног,
3. требования к расположению документов на рабочем месте (наличие и размеры подставки для документов, возможность различного размещения документов, расстояние от глаз пользователя до экрана, документа, клавиатуры и т.д.),
4. характеристики рабочего кресла,
5. требования к поверхности рабочего стола,
6. регулируемость элементов рабочего места. Главными элементами рабочего места программиста являются стол и кресло.

Основным рабочим положением является положение сидя.

Рабочая поза сидя вызывает минимальное утомление программиста.

Рациональная планировка рабочего места предусматривает четкий порядок и постоянство размещения предметов, средств труда и документации.

Изм.	Лист	№ докум.	Подпись	Дата

ЗАКЛЮЧЕНИЕ

В процессе выполнения данной выпускной квалификационной работы проведены анализ требований, проектирование и разработка, и тестирование автоматизированной системы для сбора и анализа информации из социальной сети для поиска потенциальных клиентов.

В процессе анализа требований, были сформированы требования к системе. Проведен анализ вариантов использования. Разработана модель предметной области, представленная диаграммой классов. В результате выполненной выпускной квалификационной работы реализована клиентская и серверная части и база данных, являющиеся системой для сбора и анализа информации из социальной сети для поиска потенциальных клиентов.

Система была реализована на языке JavaScript с использованием фреймворков NestJS и NextJS. На основе анализа, полученных результатов, можно сделать вывод, что все функции системы на всех наборах тестовых данных работают корректно. Следовательно, цель выпускной квалификационной работы достигнута.

Система может быть запущена в любой операционной системе с наличием современного браузера и доступом в интернет

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Интеллектуальные модели анализа информации: курс лекций 2017 BaseGroup Labs.
2. Елманова Н. Введение в OLAP-технологии Microsoft. / Н. Елманова, Федоров А. – М.: Диалог-МИФИ, 2017.
3. П. Блаттнер - Special Edition Using MVC. – М.: Вильямс, 2016.
4. Арлоу Д. Унифицированный процесс. Практический объектноориентированный анализ и проектирование / Д. Арлоу, И. Нейштадт – СПб: Символ-Плюс, 2017.
5. Мацаяшек А. Анализ требований и проектирование системы. Разработка информационных систем с использованием UML / А. Мацаяшек – М.: Издательский дом «Вильямс», 2015.
6. Фаулер М. UML. Основы / М. Фаулер, К. Скотт – СПб: СимволПлюс, 2014.
7. Documentation NestJS - A progressive Node.js framework.
8. Vercel Next JS Docs & Learning.
9. Auth0 MongoDB, Nest JS and Typegoose.
10. Мартишин С. А., Храпченко М. В. Базы данных. Практическое применение СУБД SQL- и NoSQL-типа для применения проектирования информационных систем. М :Москва, Бином, 2018. - 301 с.
11. Фаулер М., Прамодкумар Дж. Садаладж NoSQL: новая методология разработки нереляционных баз данных, :Москва, Бином, 2018. - 261 с.
12. Гома Х. UML. Проектирование систем реального времени параллельных и распределенных приложений / Х. Гома – М.: ДМК Пресс, 2015.
13. О рекламе: федер. закон РФ № 38-ФЗ [принят 13 марта 2006 г.; в ред. от 28 дек. 2013 г.]. – URL: <http://www.consultant.ru> (дата обращения: 13.02.16).

Приложение А. Листинги программы

main.ts

```
import { NestFactory } from '@nestjs/core';  
  
import { AppModule } from './app.module';  
  
async function bootstrap() {  
  const app = await NestFactory.create(AppModule);  
  await app.listen(3000);  
}  
  
bootstrap();
```

app.module.ts

```
import { Module } from '@nestjs/common';  
  
import { AppController } from './app.controller';  
  
import { AppService } from './app.service';  
  
import { AuthModule } from './auth/auth.module';  
  
import { UsersModule } from './users/users.module';  
  
import { InstAuthModule } from './inst-auth/inst-auth.module';  
  
import { InstAuthController } from './inst-auth/inst-auth.controller';  
  
import { InstAuthService } from './inst-auth/inst-auth.service';
```

@Module({

```

imports: [AuthModule, UsersModule, InstAuthModule],
controllers: [AppController, InstAuthController],
providers: [AppService, InstAuthService],
}))
export class AppModule {}

```

app.controller.ts

```

import { Controller, Get, Request, Post, UseGuards } from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';
import { AuthService } from '../auth/auth.service';

```

@Controller()

```

export class AppController {
  constructor(private readonly authService: AuthService) {}

```

@UseGuards(AuthGuard('local'))

@Post('auth/login')

```

async login(@Request() req) {
  return this.authService.login(req.user);
}

```

@UseGuards(AuthGuard('jwt'))

@Get('profile')

```

getProfile(@Request() req) {

```

```

    return req.user;

  }

}

inst-auth.service.ts

import { Injectable, Logger } from '@nestjs/common';

import * as puppeteer from 'puppeteer'

import { InstagramLoginDto, InstagramLookingDto } from '../dto/inst.dto';

const requestUrl = 'https://www.instagram.com/'

const instHomeUrl = ""

const loginUriSelector = '#react-root > section > main > article > div.rgFsT >
div:nth-child(2) > p > a'

const loginField = '#react-root > section > main > div > article > div > div:nth-
child(1) > div > form > div:nth-child(2) > div > label > input'

const passwordField = '#react-root > section > main > div > article > div > div:nth-
child(1) > div > form > div:nth-child(3) > div > label > input'

const loginButton = '#react-root > section > main > div > article > div > div:nth-
child(1) > div > form > div:nth-child(4) > button'

const bannerNotNowButton = 'body > div.RnEpo.Yx5HN > div > div >
div.mt3GC > button.aOOIW.HoLwm'

@Injectable()

export class InstAuthService {

  private readonly logger = new Logger('InstAuthController');

  private parseResponseLogin = async (page) => {

    let personalInfo = {

      cookies: null,


```

```

        body: null

    };

const loginResponse = await
page.waitForResponse('https://www.instagram.com/accounts/login/ajax/');
const body = await loginResponse.json();

if (body.authenticated) {

    const cookies = await page.cookies()

    personalInfo = {

        cookies: cookies,

        body,

    }

} else {

    this.logger.error(`Cant login: ${JSON.stringify(body)}`)
personalInfo = {

        cookies: {},

        body: null

    }

}

return personalInfo

}

async login (user: InstagramLoginDto) {

    const browser = await puppeteer.launch({ headless: false })

    const page = await browser.newPage()

    await page.goto(requestUrl, { waitLoad: true, waitNetworkIdle: true })

    // Ожидаем и нажимаем на авторизацию

```

```

    await page.waitForSelector(loginUriSelector)

    page.click(loginUriSelector)

    //Ожидаем и вводим логин и пароль

    await page.waitForSelector(loginField)

    await page.type(loginField, user.login);

    await page.type(passwordField, user.password);

    //Авторизируемся

    page.click(loginButton)

    const personalInfo = await this.parseResponseLogin(page)

    this.logger.log(personalInfo)

    return personalInfo }

async looking (users: InstagramLookingDto) {

    const browser = await puppeteer.launch({ headless: false })

    const page = await browser.newPage()

    await page.goto(instHomeUrl, { waitLoad: true, waitNetworkIdle: true })

    await page.evaluate(() => {

        fetch("/login", {

            method: "POST",

        });

    });

}

}

inst-auth.service.spec.ts

```

```

import { Test, TestingModule } from '@nestjs/testing';
import { InstAuthService } from './inst-auth.service';

describe('InstAuthService', () => {
  let service: InstAuthService;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [InstAuthService],
    }).compile();

    service = module.get<InstAuthService>(InstAuthService);
  });

  it('should be defined', () => {
    expect(service).toBeDefined();
  });
});

```

inst-auth.module.ts

```

import { Module } from '@nestjs/common';
import { InstAuthService } from './inst-auth.service';
import { InstAuthController } from './inst-auth.controller';

```

```
@Module({
  providers: [InstAuthService],
  controllers: [InstAuthController]
})
export class InstAuthModule { }
```

inst-auth.controller.ts

```
import { Controller, Post, Body, UseGuards } from '@nestjs/common'
import { AuthGuard } from '@nestjs/passport';
import { InstAuthService } from '../inst-auth.service'
import { InstagramLoginDto, InstagramLookingDto } from '../dto/inst.dto';
```

```
@Controller('inst')
```

```
export class InstAuthController {
```

```
  constructor(private readonly instAuthService: InstAuthService) { }
```

```
  @UseGuards(AuthGuard('jwt'))
```

```
  @Post('login')
```

```
  login(@Body() user: InstagramLoginDto) {
```

```
    return this.instAuthService.login(user)
```

```
  }
```

```
  @UseGuards(AuthGuard('jwt'))
```

```

@Post('looking')

looking(@Body() users: InstagramLookingDto) {
  return this.instAuthService.looking(users)
}
}

```

inst-auth.controller.spec.ts

```

import { Test, TestingModule } from '@nestjs/testing';
import { InstAuthController } from './inst-auth.controller';

```

```

describe('InstAuth Controller', () => {

```

```

  let controller: InstAuthController;

```

```

  beforeEach(async () => {

```

```

    const module: TestingModule = await Test.createTestingModule({
      controllers: [InstAuthController],
    }).compile();

```

```

    controller = module.get<InstAuthController>(InstAuthController);
  });

```

```

  it('should be defined', () => {

```

```

    expect(controller).toBeDefined();

```

```

  });

```

					ПГУ 1.090501.15.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		75

```
});
```

```
Task/index.jsx
```

```
/* eslint-disable react/prop-types */
```

```
import React from 'react';
```

```
import {
```

```
  Box, Button, Select, CheckBox,
```

```
} from 'grommet';
```

```
import {
```

```
  Add,
```

```
} from 'grommet-icons';
```

```
import Layout from '../components/layout';
```

```
const Task = () => {
```

```
  const [checked, setChecked] = React.useState(true);
```

```
  return (
```

```
    <Layout>
```

```
      <Box pad="large">
```

```
        <Select
```

```
          options={['Анализ аккаунта', 'Подбор базы', 'Продвижение аккаунта']}>
```

```
          value="Продвижение аккаунта">
```

					ПГУ 1.090501.15.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		76

```

    onChange=(({ option }) => {} )
  />

<Box pad="large" flex direction="row" alignContent="stretch">

  <CheckBox

    checked={checked}

    label="Поиск по хештегам"

    onChange={event => setChecked(event.target.checked)}

    pad="small"

  />

  <CheckBox

    checked={false}

    label="Поиск по геометкам"

    onChange={event => setChecked(event.target.checked)}

    pad="small"

  />

  <CheckBox

    checked={false}

    label="Поиск по подписчикам и подпискам"

    onChange={event => setChecked(event.target.checked)}

    pad="small"

  />

</Box>

<Box width="small" flex alignSelf="center">

  <Button label="Create task" icon={ <Add /> } />

```

```

        </Box>

    </Box>

</Layout>

);

};

export default Task;

main.jsx

import React from 'react';
import {
    Box,
    Main as MainGrommet,
    WorldMap,
} from 'grommet';

const Main = () => (
    <MainGrommet
        pad="medium"
        overflow="auto"
        fill
        justify="center"
    >

```

					ПГУ 1.090501.15.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		78

```
<Box align="center" justify="center" pad="medium" fill>
```

```
<WorldMap
```

```
places={[
```

```
{
```

```
name: 'Moscow',
```

```
location: [55.751244, 37.618423],
```

```
color: 'neutral-4',
```

```
},
```

```
{
```

```
name: 'Penza',
```

```
location: [53.20066, 45.00464],
```

```
color: 'neutral-4',
```

```
},
```

```
]]
```

```
continents={[
```

```
{
```

```
name: 'Asia',
```

```
color: 'graph-1',
```

```
// onClick: () => setActive(!active),
```

```
},
```

```
{
```

```
name: 'Europe',
```

```
color: 'graph-1',
```

```
// onClick: () => setActive(!active),
```

					ПГУ 1.090501.15.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		79

```

    },
  ]}
  fill

  pad="large"

  onSelectPlace={(lat, lon) => {}}

  selectColor="accent-2"

  color={{
    dark: 'accent-2',
    light: 'status-warning',
  }}
/>

</Box>

</MainGrommet>

);

```

```
export default Main;
```

```
layout.jsx
```

```
/* eslint-disable react/prop-types */
```

```
import React from 'react';
```

```
import App from 'next/app';
```

```
import {
```

```
  Box,
```

```
} from 'grommet';
```

```
import Header from '../header';
```

```
import Sidebar from '../sideBar';
```

```
const Layout = ({ children }) => (
```

```
  <Box direction="row" fill overflow="hidden">
```

```
    <Box
```

```
      direction="row"
```

```
      pad="small"
```

```
    <Sidebar />
```

```
  </Box>
```

```
  <Box direction="column" fill overflow="hidden" pad={{ vertical: 'small', right: 'small' }}>
```

```
    <Box>
```

```
      <Header />
```

```
    </Box>
```

```
    <Box fill>
```

```
      { children }
```

```
    </Box>
```

```
  </Box>
```

					ПГУ 1.090501.15.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		81

```

</Box>

);

export default Layout;

sidebar.jsx

import React from 'react';

import {
  Avatar,
  Button,
  Box,
  Nav,
  Layer,
  Heading,
  Text,
  Form, FormField,
} from 'grommet';

import {
  Analytics,
  Configure,
  Projects,
  Add,
  Logout,

```

```

    } from 'grommet-icons';

import Sidebar from '../././././components/sideBar';

import redirectTo from '../././././components/redirectTo';
import Input from '../././././components/input';

const SidebarButton = ({ icon, label, ...rest }) => (
  <Box pad="small">
    <Button
      gap="medium"
      alignSelf="start"
      plain
      icon={ icon }
      label={ label }
      hoverIndicator
      focusIndicator={ false }
      { ...rest }
    />
  </Box>
);

const SidebarFooter = (isOpen, setIsOpen) => (

```

```

<Nav gap="small" overflow="hidden">

  <SidebarButton icon={ <Logout /> } label="Logout" onClick={() =>
setIsOpen(!isOpen)} />

</Nav>

);

const MainNavigation = () => (

  <Nav gap="large" responsive={false} overflow="hidden">

    <SidebarButton icon={ <Analytics /> } label="Analytics" onClick={() =>
redirectTo('analytics')} />

    <SidebarButton icon={ <Projects /> } label="Tasks" onClick={() =>
redirectTo('tasks')} />

    <SidebarButton icon={ <Configure /> } label="Settings" onClick={() =>
redirectTo('settings')} />

  </Nav>

);

const SidebarHeader = ({ openModal }) => {

  const t = 2;

  if (t === 1) {

    return (<Avatar
src="//s.gravatar.com/avatar/b7fb138d53ba0f573212ccce38a7c43b?s=80" />);

  }

  return (

    <Box overflow="hidden">

```

```

        <Button label="Add Account" icon={ <Add /> } onClick={openModal} />
    </Box>

);

};

const DashboardSideBar = () => {

    const [isOpen, setIsOpen] = React.useState();

    const [value, setValue] = React.useState({ login: "", password: "" });

    return (

        <>

            <Sidebar

                renderHeader={() => SidebarHeader({ openModal: () => setIsOpen(true) })}

                renderMain={MainNavigation}

                renderFooter={() => SidebarFooter()}

                round="small"

                overflow="hidden"

            />

            {isOpen && (

                <Layer

                    position="center"

                    modal

                    onEsc={() => setIsOpen(false)}

                    onClickOutside={() => setIsOpen(false)}

```

>

```
{/* <Form
```

```
value={ value }
```

```
onChange={nextValue => setValue(nextValue)}
```

```
onSubmit={({ value: nextValue }) => { }}
```

```
> */}
```

```
<Box pad="medium" gap="small" width="medium">
```

```
<Heading level={3} margin="none">
```

Аккаунт успешно добавлен!

```
</Heading>
```

```
{/* <FormField
```

```
label="Login"
```

```
name="login"
```

```
required
```

```
validate={ [
```

```
(name) => {
```

```
if (!name || name.length === 0) return 'Please enter your Login';
```

```
return undefined;
```

```
},
```

```
]]
```

```
>
```

```
<Input name="login" type="login" />
```

```
</FormField>
```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.15.001 ПЗ

Лист

86

```

<FormField

  label="Password"

  name="password"

  required

  error="Ошибка авторизации. Проверьте логин и пароль"

  validate={ [

    (name) => {

      if (!name || name.length === 0) return 'Please enter your Password';

      return undefined;

    },

  ]}

>

  <Input name="password" type="password" />

</FormField> */}

<Box

  as="footer"

  gap="small"

  direction="row"

  align="center"

  justify="end"

  pad={{ top: 'medium', bottom: 'small' }}

>

  { /* <Button label="Add" onClick={() => setIsOpen(false)} color="dark-3"
/> */}

```

```

    <Button
      label={ (
        <Text color="white">
          <strong>Close</strong>
        </Text>
      ) }
      onClick={ () => setIsOpen(false) }
      primary
      color="status-critical"
    />
  </Box>
</Box>
{ /* </Form> */ }
</Layer>
)
}
</>
);
};

```

```
export default DashboardSideBar;
```

```
header.jsx
```

```
import React from 'react';
```

```
import {
  Header as HeaderGrommet, Box, Meter, Text, Button, Heading, Layer,
} from 'grommet';
```

```
import {
  FormAdd,
} from 'grommet-icons';
```

```
const Header = () => {
  const [isOpen, setIsOpen] = React.useState();
```

```
  const handleAdd = () => {
    setIsOpen(true);
  };
```

```
  return (
    <Box>
      <HeaderGrommet
        pad="small"
        overflow="hidden"
        background="brand"
        height="xsmall"
        round="small"
      >
```

<Box align="center" pad={{ left: 'small' }}>

<Text>

Completed Tasks:

</Text>

<Box align="center" pad="small" direction="row">

<Text margin={{ right: 'small' }}>

30/40

</Text>

<Meter

type="bar"

background="light-2"

round

thickness="small"

max="40"

values={{ { value: 30 } }}

/>

</Box>

</Box>

<Box pad={{ right: 'small' }} align="center">

<Button label="Add Task" icon={ <FormAdd /> } onClick={ handleAdd } />

</Box>

</HeaderGrommet>

{ isOpen && (

<Layer

```

position="center"

modal

onEsc={() => setIsOpen(false)}

onClickOutside={() => setIsOpen(false)}

>

<Box pad="medium" gap="small" width="medium">

  <Heading level={3} margin="none">

    В первую очередь добавьте аккаунт Instagram

  </Heading>

  <Text>Перейти на экран добавления аккаунта?</Text>

  <Box

    as="footer"

    gap="small"

    direction="row"

    align="center"

    justify="end"

    pad={{ top: 'medium', bottom: 'small' }}

  >

    <Button label="Ok" onClick={() => setIsOpen(false)} color="dark-3" />

    <Button

      label={({

        <Text color="white">

          <strong>Cancel</strong>

        </Text>

      )}

    >

```

```
    )}  
    onClick={() => setIsOpen(false)}  
    primary  
    color="status-critical"  
  />  
</Box>  
</Box>  
</Layer>  
  )}  
</Box>  
);  
};
```

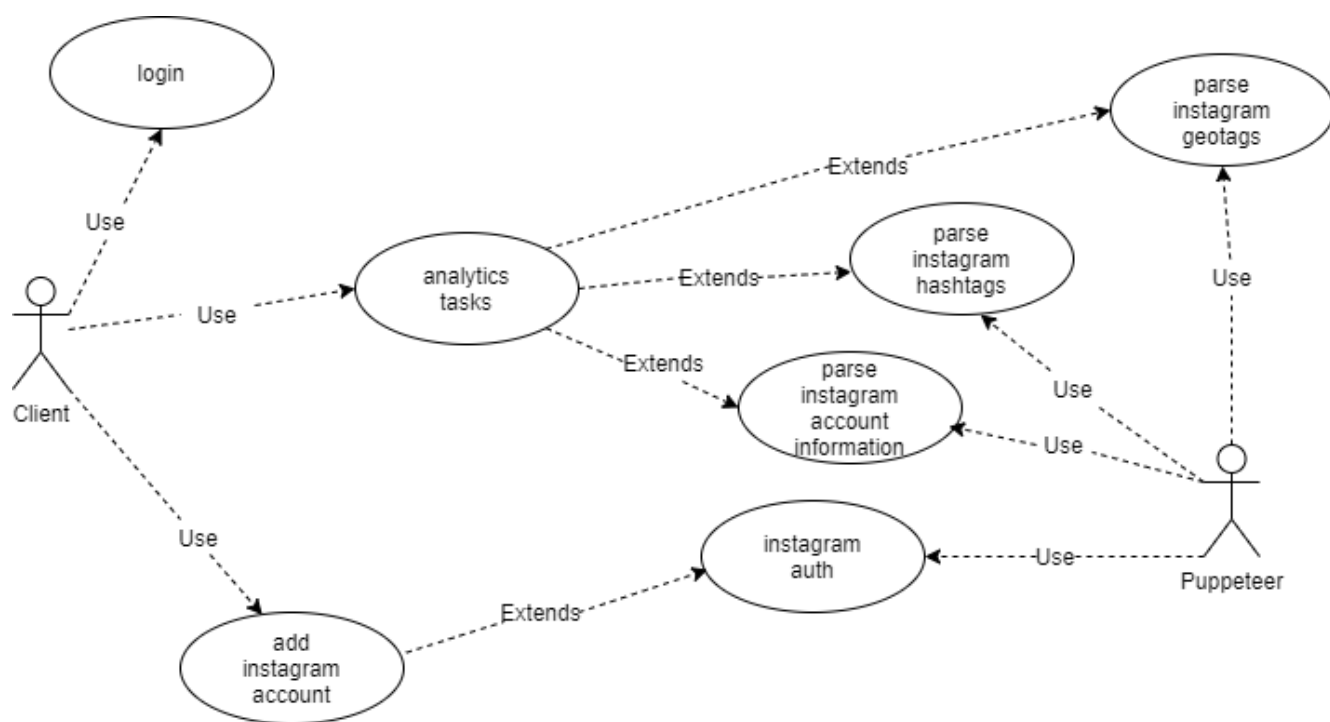
```
export default Header;
```

					ПГУ 1.090501.15.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		92

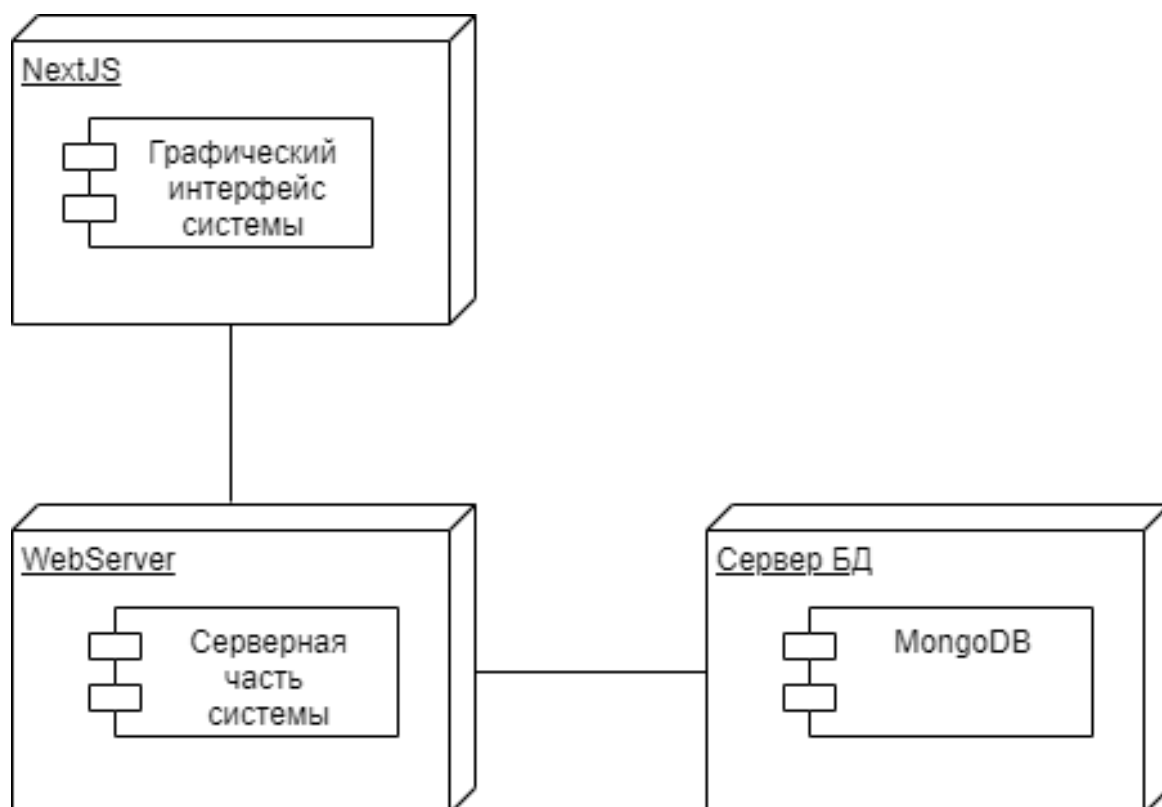
Приложение Б.

Диаграммы

					ПГУ 1.090501.15.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		93



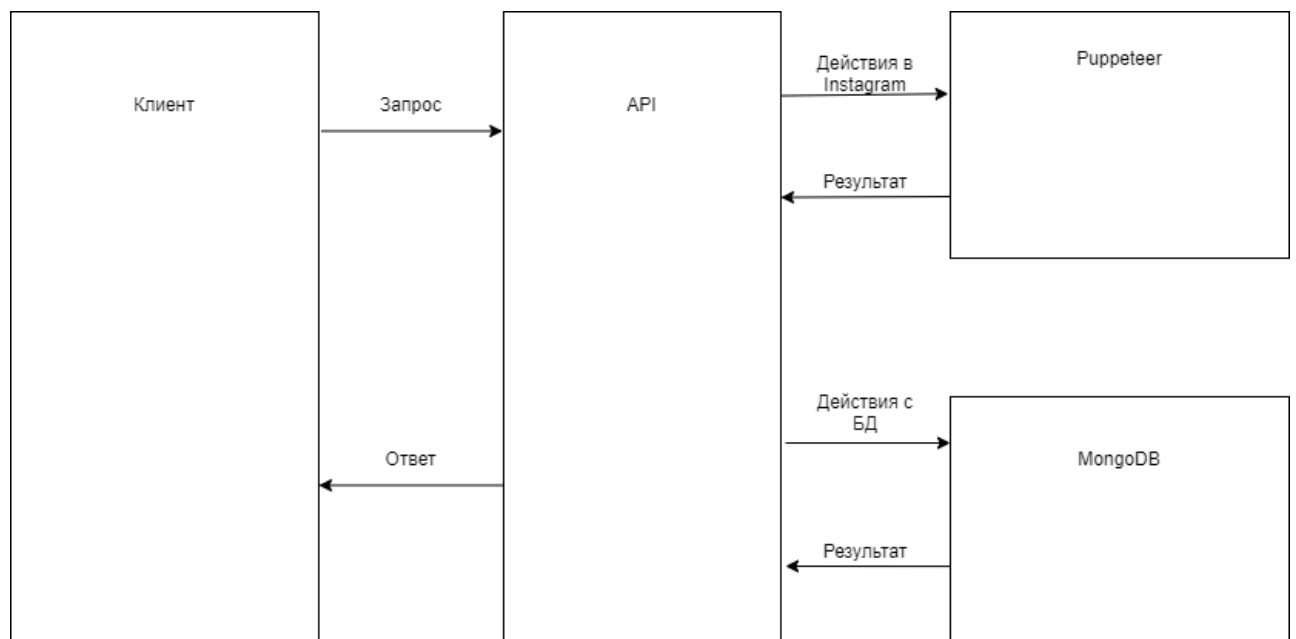
					ПГУ 1.090501.15.002 Д01		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Цыпин Л.В.			Автоматизированная система для сбора и анализа информации из социальной сети для поиска потенциальных клиентов. Диаграмма вариантов использования	Лит.	Лист
Провер.		Митрохин М.А.					Листов
Н. Контр.		Кучин А.В.					94
Реценз.		Севастьянов А.В.					1(102)
Утверд.		М.А.Митрохин				Кафедра ВТ Гр.15В02	



					ПГУ 1.090501.15.003 Д02				
Изм.	Лист	№ докум.	Подпись	Дата					
Разраб.		Цытин Л.В.			Автоматизированная система для сбора и анализа информации из социальной сети для поиска потенциальных клиентов. Диаграмма развертывания	Лит.	Лист	Листов	
Провер.		Митрохин М.А.					95	1(102)	
Н. Контр.		Кучин А.В.				Кафедра ВТ Гр.15В02			
Реценз.		Севастьянов А.В							
Утверд.		М.А.Митрохин							



					ПГУ 1.090501.15.004 Д03				
Изм.	Лист	№ докум.	Подпись	Дата					
Разраб.		Цытин Л.В.			Автоматизированная система для сбора и анализа информации из социальной сети для поиска потенциальных клиентов. Диаграмма классов предметной области	Лит.	Лист	Листов	
Провер.		Митрохин М.А.						96	1(102)
Н. Контр.		Кучин А.В.				Кафедра ВТ Гр.15В02			
Реценз.		Севастьянов А.В							
Утверд.		М.А.Митрохин							



					<p>ПГУ 1.090501.15.005 Д04</p>		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Цытин Л.В.			Автоматизированная система для сбора и анализа информации из социальной сети для поиска потенциальных клиентов. Диаграмма структуры проекта	Лит.	Лист
Провер.		Митрохин М.А.					Листов
Н. Контр.		Кучин А.В.					97
Реценз.		Севастьянов А.В.					1(102)
Утверд.		М.А.Митрохин				Кафедра ВТ Гр.15В02	

Приложение В. Презентация

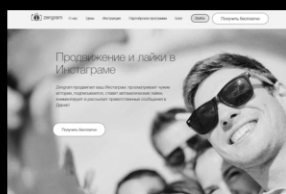
Автоматизированная система для сбора и анализа информации из социальной сети для поиска потенциальных клиентов.

Выполнил студент группы 15В02
Цыпин Л.В.

Научный руководитель
Митрохин М.А.

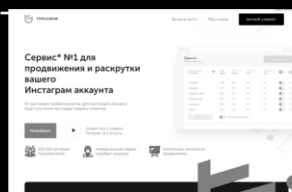
Пенза, 2020

АНАЛОГИ



ZENGRAM

TOOLIGRAM



PARASITE

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.15.001 ПЗ

Лист

98

ЗАДАЧИ

01

УНИКАЛЬНОСТЬ

02

МАСШТАБИРУЕМОСТЬ

03

ОТКАЗОУСТОЙЧИВОСТЬ

СТЕК ТЕХНОЛОГИЙ



NEXTJS

Фреймворк для
написания
клиентской части

NESTJS

Фреймворк для
написания
серверной части

MONGODB

Система
управления
нереляционной
базой данных

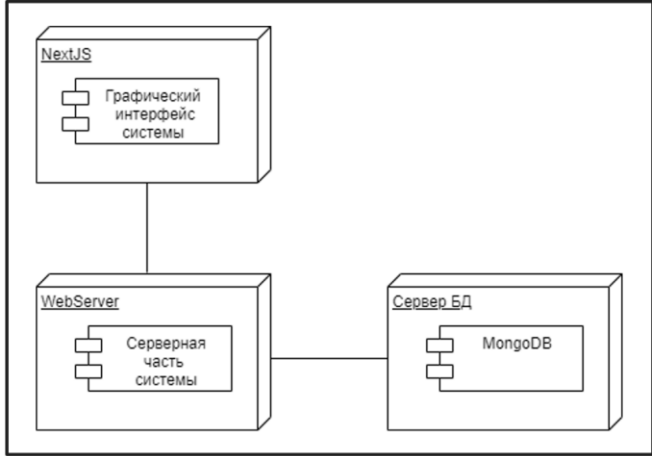
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.15.001 ПЗ

Лист

99

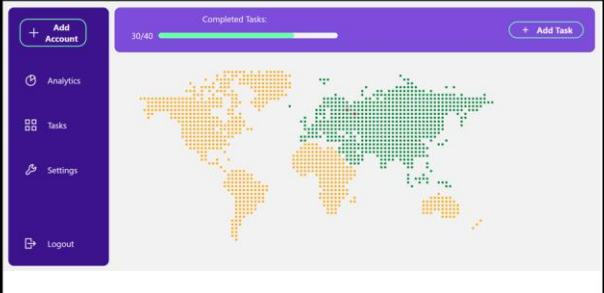
ДИАГРАММА РАЗВЕРТЫВАНИЯ



ИНТЕРФЕЙС

Главный экран

Экран, предоставляющий доступ ко всему функционалу системы



Экран создания задачи

Экран с выбором фильтров и типа задачи

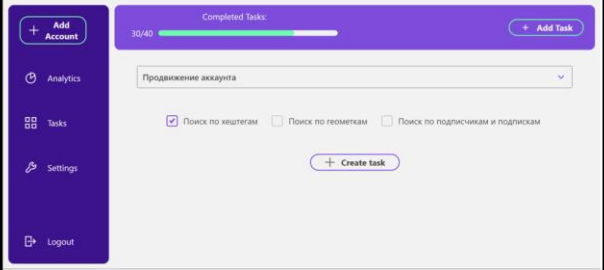
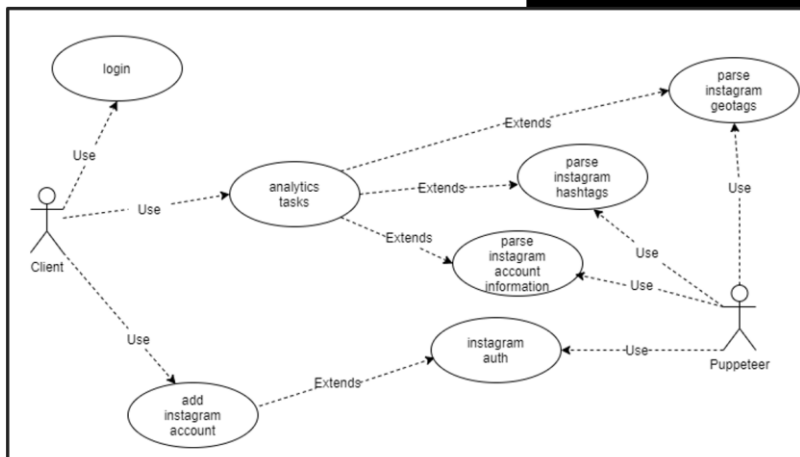
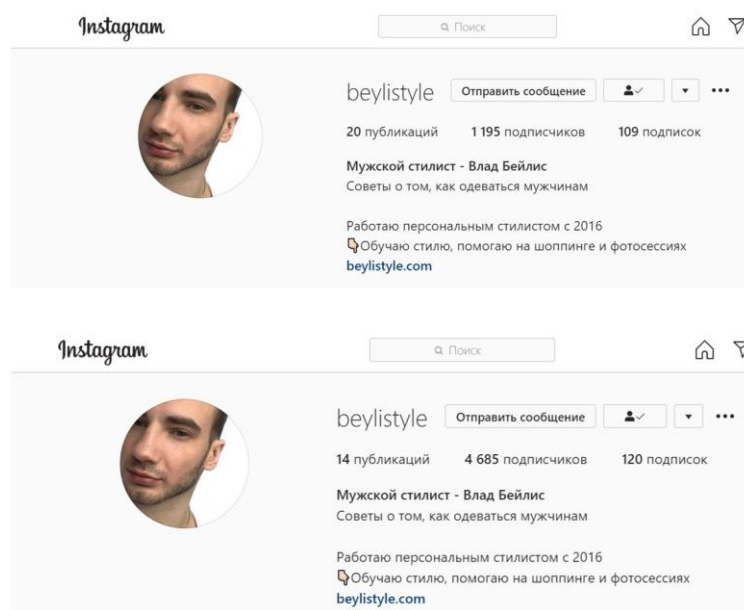


ДИАГРАММА ИСПОЛЬЗОВАНИЯ



РЕЗУЛЬТАТЫ РАБОТЫ СИСТЕМЫ



Спасибо за внимание!

Выполнил студент группы 15BO2
Цыпин Л.В.

Научный руководитель
Митрохин М.А.

Пенза, 2020

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.15.001 ПЗ

Лист

102