МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Экономики и управления на предприятии природопользования»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(бакалаврская работа)
по направлению подготовки 09.03.03 Прикладная информатика
(квалификация – бакалавр)

На тему «Разработка информационной системы «Платформа с ИИ - помощником для написания книг»»

Исполнитель Платонов Игорь Олегович

Руководитель к.т.н., доцент Попов Николай Николаевич

«К защите допускаю»

Руководитель кафедры _____

кандидат экономических наук

Майборода Евгений Викторович

«18» 01 2026 г.

Филиал Российского государственного гидрометеорологического университета в г. Туапсе
НОРМОКОНТРОЛЬ ПРОЙДЕН
«12» 01 2026 г.
Мартынова Т.В.
ПОДПИСЬ РАСШИФРОВКА ПОДПИСИ

Туапсе
2026

ОГЛАВЛЕНИЕ

Введение

В условиях современной цифровой трансформации технологии искусственного интеллекта (ИИ) демонстрируют активное проникновение во все сферы человеческой деятельности, включая творческие процессы, к которым относится создание литературных произведений. Появление высокопроизводительных языковых моделей открыло новые технологические возможности для авторов, однако вместе с тем сформировало ряд принципиальных вызовов. Большинство представленных на рынке программных решений позиционируются как генераторы контента, способные создавать текст с нуля, что сопряжено с риском утраты авторской индивидуальности, оригинальности и контроля над творческим процессом.

Актуальность данной работы обусловлена объективной потребностью в разработке интеллектуальных инструментов, функциональность которых направлена не на замещение писателя, а на дополнение и усиление его творческих возможностей. Существует выраженный спрос на платформы, в которых ИИ функционирует в роли ассистента, содействуя автору в решении рутинных задач: коррекции грамматических и стилистических ошибок, предложении альтернативных вариантов развития сюжета, проведении семантического анализа текста и расширении существующих идей. Наряду с этим, значимыми проблемами остаются обеспечение конфиденциальности данных и высокая стоимость эксплуатации облачных ИИ-сервисов. Разработка системы, способной функционировать на базе локально развернутых языковых моделей, выступает эффективным решением указанных проблем, предоставляя авторам безопасную и экономически доступную программную среду.

В рамках настоящей работы предлагается решение поставленных задач в виде информационной системы «Платформа с ИИ-помощником для написания книг». Ключевой особенностью системы является ее архитектура, ориентированная на ассистирование, а не на генерацию, что позволяет сохранить за автором центральную роль в творческом процессе.

Объектом исследования выступает процесс разработки информационных систем, направленных на поддержку творческой деятельности.

Предметом исследования является комплекс архитектурных и программных решений для создания платформы с интегрированным ИИ-помощником, способной функционировать в условиях ограниченных аппаратных ресурсов.

Целью данной выпускной квалификационной работы является разработка функционального прототипа информационной системы, предоставляющей авторам инструментарий для написания и редактирования текстов при поддержке интегрированного ИИ-помощника.

Для достижения поставленной цели были определены следующие задачи:

— Проведение анализа существующих программных средств и платформ, предназначенных для писательской деятельности.

— Проектирование микросервисной архитектуры системы и разработка структуры базы данных для хранения пользовательской информации и контента.

— Реализация сервиса аутентификации на языке программирования Go для управления доступом пользователей.

— Разработка основного сервиса приложения, реализующего бизнес-логику по управлению проектами, документами и взаимодействию с ИИ.

— Осуществление интеграции с локально развернутой языковой моделью с использованием платформы Ollama.

— Разработка пользовательского веб-интерфейса для взаимодействия с системой.

— Подготовка среды для развертывания всего комплекса сервисов с применением технологии контейнеризации Docker.

Практическая значимость работы заключается в создании готового к демонстрации прототипа, который подтверждает жизнеспособность выбранной архитектурной модели. Разработанный программный комплекс может служить технологической основой для дальнейшего развития и коммерциализации

продукта. Продемонстрирован практический подход к развертыванию сложной системы на локальном оборудовании с организацией удаленного доступа для тестирования, что является актуальным решением на этапе прототипирования.

Выпускная квалификационная работа имеет традиционную структуру и состоит из введения, трех разделов, заключения, списка использованных источников и приложений. Во введении обоснована актуальность темы, сформулированы цель и задачи. В первом разделе представлен анализ предметной области. Во втором разделе детально описан процесс проектирования и разработки системы. Третий раздел посвящен экономическому обоснованию проекта. В заключении подведены итоги проделанной работы.

1 Аналитический обзор предметной области

1.1 Анализ современных программных средств для писательской деятельности

Для формирования объективного обоснования необходимости разработки информационной системы «Платформа с ИИ-помощником для написания книг» требуется провести системный анализ существующих на рынке программных продуктов, предназначенных для авторов. Современные программные решения целесообразно классифицировать на три основные категории: классические текстовые процессоры, специализированное программное обеспечение для управления писательскими проектами и платформы, в основе функциональности которых лежит применение технологий искусственного интеллекта.

1. Классические текстовые процессоры (например, MicrosoftWord, GoogleDocs)

Данная категория инструментов является наиболее распространенной и универсальной для работы с текстовыми данными. Эти программные продукты предоставляют широкий функционал для форматирования, рецензирования и подготовки документов к печати.

Ключевые преимущества:

— Широкое распространение и доступность: Являются стандартным компонентом большинства операционных систем и офисных пакетов, что обеспечивает их повсеместное присутствие на персональных компьютерах.

— Развитые средства форматирования: Предоставляют большой набор инструментов для детальной настройки визуального представления текста, его подготовки к типографской печати или электронной публикации.

— Функционал для совместной работы: Реализованы эффективные механизмы для организации коллективной работы над документом, включая комментирование и редактирование в реальном времени (в частности, в GoogleDocs).

Ключевые недостатки:

— Отсутствие специализированных средств для управления структурой проекта: Программы ориентированы на работу с линейными документами и не содержат инструментов для управления сложными иерархическими структурами, такими как главы, персонажи, сюжетные линии и исследовательские материалы. Работа с крупными произведениями затруднена из-за сложности навигации и организации контента.

— Ограниченные или отсутствующие возможности искусственного интелекта: Интегрированные интеллектуальные функции, как правило, ограничиваются базовой коррекцией орфографии и грамматики и не предлагают средств для интеллектуальной поддержки творческого процесса.

2. Специализированное программное обеспечение для авторов (например, Scrivener, Ulysses)

Программные решения данной категории разработаны с целью нивелирования недостатков, присущих классическим текстовым процессорам, в части структурирования крупных литературных произведений.

Ключевые преимущества:

— Развитые средства организации контента: Позволяют производить декомпозицию произведения на составные части (главы, сцены, фрагменты) с возможностью их гибкой реструктуризации. Предусмотрен функционал для централизованного хранения сопутствующих материалов в рамках единого проекта.

— Режим сфокусированной работы: Предоставляют минимальный пользовательский интерфейс, позволяющий минимизировать отвлекающие факторы и сконцентрироваться на создании текста.

— Гибкие механизмы экспорта: Поддерживают компиляцию проекта в различные конечные форматы (например, .docx, .pdf, .epub) с возможностью детальной настройки параметров вывода.

Ключевые недостатки:

— Низкий уровень интеграции с ИИ: Архитектура данных программных продуктов изначально не предусматривала интеграцию с технологиями искусственного интеллекта. Соответствующие функции либо отсутствуют, либо реализованы посредством сторонних модулей и не являются центральным элементом рабочего процесса.

— Повышенные требования к освоению: Многофункциональность интерфейса и обилие настроек могут требовать от новых пользователей значительных временных затрат на изучение всего инструментария программы.

3. Платформы с доминирующей ролью ИИ (например, Jasper, NovelAI, Sudowrite)

Данная категория представляет собой новое поколение инструментов, архитектура которых построена на основе больших языковых моделей (LLM). Они предлагают авторам прямую ассистенцию ИИ в процессе написания текстов.

Ключевые преимущества:

— Высокие генеративные способности: Способны генерировать текстовые фрагменты различного объема на основе кратких описаний (промптов), продолжать повествование, создавать диалоги и описательные элементы.

— Широкий спектр ИИ-инструментов: Предлагают разнообразный функционал, включающий рерайтинг, суммирование, а также генерацию идей для сюжетных разработок.

Ключевые недостатки:

— Парадигма «ИИ как генератор контента»: Основным концептуальным недостатком является смещение роли ИИ от ассистента к основному автору. Это приводит к риску потери уникального писательского стиля, а генерируемый контент может характеризоваться отсутствием авторской индивидуальности.

— Зависимость от облачной инфраструктуры и высокая стоимость: Функционирование данных платформ основано на модели подписки

(SaaS) и требует регулярных финансовых отчислений. Обработка данных на удаленных серверах вызывает обоснованные опасения относительно конфиденциальности литературных материалов.

— Требовательность к качеству входных данных: Эффективность систем напрямую зависит от компетенций пользователя в области формулирования точных и корректных запросов к ИИ.

Проведенный сравнительный анализ позволяет выявить свободную рыночную нишу в сегменте программного обеспечения для авторов. Обобщенные результаты анализа представлены в таблице 1.

Таблица 1 – Сравнительная характеристика категорий программных средств

Категория инструмента	Уровень управления структурой	Реализация ИИ	Модель развертывания
Текстовые процессоры	Низкий	Ограниченная (коррекция орфографии)	Локальное / Облачное
Специализированные ПО	Высокий	Отсутствует или слабая	Локальное (лицензия)
ИИ-платформы	Средний	Центральная (генерация контента)	Облачное (подписка)

Как следует из представленной таблицы, на рынке отсутствует программное решение, комплексно сочетающее в себе сильные стороны всех рассмотренных категорий: развитую систему организации проекта, глубокую интеграцию ИИ в роли помощника (а не генератора) и возможность локального развертывания для обеспечения конфиденциальности данных и независимости от подписной модели. Разрабатываемый в рамках данной работы прототип нацелен на заполнение указанной ниши путем создания сбалансированного и безопасного инструмента для творческой деятельности.

1.2 Роль и место искусственного интеллекта в творческом процессе создания текстов

Современный этап развития информационных технологий характеризуется фундаментальными изменениями в подходах к обработке естественного языка. Ключевым драйвером этих изменений стало появление нейросетевых архитектур, способных не просто классифицировать информацию, но и генерировать новый контент, обладающий высокой степенью семантической связности. Как справедливо отмечает А.Ю. Черепнин, «большие языковые модели (LLM) демонстрируют беспрецедентные возможности в широком спектре задач обработки естественного языка» [27, с.1], что открывает принципиально новые горизонты для автоматизации творческих процессов, ранее считавшихся прерогативой исключительно человеческого интеллекта.

Влияние этих технологий на писательскую деятельность трудно переоценить. Генеративные модели, обученные на гигантских корпусах текстовых данных, способны имитировать стилистические особенности, поддерживать контекст повествования и предлагать варианты развития сюжетных линий. В этом контексте Б.В. Орехов указывает на важную особенность современных систем: «большие языковые модели умеют порождать правдоподобные тексты» [23, с.1]. Однако именно «правдоподобие» является как сильной, так и слабой стороной технологии: модель может генерировать грамматически корректный, но фактологически ложный или логически противоречивый текст, что требует постоянного контроля со стороны человека.

Отдельным и наиболее критическим аспектом применения больших языковых моделей в литературном творчестве является техническое ограничение длины контекстного окна (Context Window). В процессе написания крупной прозаической формы, объем которой может достигать сотен тысяч токенов, стандартные LLM неизбежно сталкиваются с проблемой «катастрофического забывания» ранних сюжетных деталей. Большинство

современных моделей потребительского сегмента способны удерживать в оперативной памяти контекст объемом от 4 до 8 тысяч токенов, что эквивалентно лишь нескольким главам текста. При превышении этого лимита модель теряет когерентность повествования: персонажи могут внезапно изменить имена, характеристики или мотивацию, а логические связи с экспозицией произведения разрываются. Существующие методы обхода данного ограничения, такие как рекурсивное суммирование (Recursive Summarization) или использование векторных хранилищ памяти (Vector Stores), хоть и смягчают проблему, но вносят искажения в семантическую ткань произведения. В связи с этим, на текущем этапе развития технологий, наиболее эффективной стратегией взаимодействия человека и машины остается фрагментарная обработка текста, где автор вручную определяет границы контекста, передаваемого модели для анализа, что позволяет сохранить глобальную целостность сюжета за счет когнитивных способностей человека, а не вычислительной мощности алгоритма.

Внедрение подобных инструментов в рабочие процессы авторов приводит к трансформации самой сути писательского труда. Происходит переход от чисто ручного создания текста к гибридным формам взаимодействия, где алгоритм берет на себя рутинные операции по перефразированию, корректуре или генерации идей. Г.В. Андрончик подчеркивает, что «синтез когнитивных технологий с классическими аналитическими инструментами способствует глубокому преобразованию бизнес-процессов, создавая подходящие условия для формирования гибких и адаптивных систем менеджмента» [3, с.1]. Применительно к литературному творчеству это означает создание адаптивной среды, где ИИ подстраивается под стиль автора, выступая не в роли соавтора, а в роли интеллектуального инструмента поддержки.

Тем не менее, интеграция ИИ в творчество сопряжена с рядом серьезных вызовов. Техническая сложность моделей и их непрозрачность («черный ящик») порождают проблемы надежности и предсказуемости результатов. Д.Е. Намиот и Е.А. Ильюшин справедливо замечают: «При этом, любое

использование LLM привносит массу рисков, свойственных генеративным моделям» [21, с.1]. К таким рискам относятся не только «галлюцинации» нейросетей, но и возможная потеря уникального авторского голоса при чрезмерном полагании на машинную генерацию.

Именно поэтому в проектируемой системе отвергается концепция полной автоматизации написания книг. А.А. Шинкарев и соавторы формулируют важный этический и практический принцип: «Передача полного контроля управлением из рук человека в «руки» ИИ на текущий момент невозможна, поскольку не установлены четкие этические рамки и зона ответственности ИИ» [30, с.22]. Следовательно, наиболее продуктивной парадигмой является «ИИ-ассистент», где человек остается демиургом и финальным верификатором любого созданного контента.

Одной из технических проблем при использовании LLM является ограниченность контекстного окна и знаний модели, заложенных на этапе обучения. П.А. Ломов в своем исследовании отмечает, что «качество ответов снижается, если вопросы начинают касаться предметов, процессов и явлений, которые в меньшей степени описаны в текстах, использованных для обучения модели» [18, с.1]. Для писателя это критично, так как работа над книгой подразумевает оперирование уникальным, созданным автором миром, о котором базовая модель ничего не знает.

В современной разработке интеллектуальных систем стандартом решения данной проблемы является архитектурный подход RAG (Retrieval Augmented Generation). Как указывают А.Г. Олейник и коллеги, «в рамках RAG-технологии реализуются механизмы поиска в базе данных информационной системы фрагментов текста, вектора которых наиболее релевантны векторизованному вопросу пользователя» [22, с.4]. Данный подход позволяет существенно повысить фактологическую точность ответов модели за счет подстановки внешней информации.

Тем не менее, несмотря на высокую технологическую привлекательность RAG, в рамках разработки данного прототипа было принято решение

использовать модель явного пользовательского контекста, отказавшись от автоматического векторного поиска на текущем этапе. Это обусловлено двумя факторами. Во-первых, внедрение векторной базы данных и механизма эмбеддингов значительно повышает требования к вычислительным ресурсам, что противоречит задаче создания легковесной системы для локального запуска на персональных компьютерах. Во-вторых, специфика писательской работы часто требует прецизионной точности: автору необходимо, чтобы ИИ обработал именно конкретный, выделенный вручную абзац или главу, а не фрагменты, которые алгоритм сочтет семантически похожими. Таким образом, в проектируемой системе контроль за контекстом полностью передается пользователю, что обеспечивает предсказуемость результата генерации, оставляя внедрение RAG перспективной задачей для следующих версий продукта.

Наконец, нельзя игнорировать вопросы информационной безопасности, особенно когда речь идет о взаимодействии с внешними моделями или интеграции пользовательских данных. Ю.Е. Лебединский и Д.Е. Намиот предупреждают о специфических угрозах: «Промпт-инъекции представляют собой метод атаки, в котором злоумышленник внедряет специально сформированные инструкции в запросы к AI-системе» [15, с.1]. В связи с этим, архитектура платформы должна предусматривать механизмы валидации и санации данных, передаваемых в языковую модель, чтобы исключить возможность манипуляции поведением ассистента.

Таким образом, роль искусственного интеллекта в современном творческом процессе определяется как вспомогательная, но высокотехнологичная функция, требующая тщательной настройки, контроля контекста и обеспечения безопасности взаимодействия.

1.3 Формулирование задач и определение требований к разрабатываемому прототипу

На основании результатов проведенного анализа существующих программных средств и в соответствии с принятой концепцией ассистирующего взаимодействия с искусственным интеллектом, были сформулированы цель, задачи, а также детализированные требования к функциональности и архитектуре разрабатываемого прототипа информационной системы.

Цель разработки: Разработка и реализация функционального прототипа клиент-серверной информационной системы, предоставляющей авторам изолированную и защищенную программную среду для работы над текстовыми произведениями с использованием интегрированного ИИ-помощника.

Основные задачи разработки:

— Спроектировать и реализовать многопользовательскую систему с обеспечением безопасных механизмов аутентификации и авторизации.

— Разработать структуру базы данных для персистентного хранения данных пользователей, проектов, документов и сопутствующих материалов.

— Реализовать серверное приложение (API), инкапсулирующее всю бизнес-логику системы.

— Осуществить интеграцию серверного приложения с локально развернутой большой языковой моделью.

— Разработать интуитивно понятный клиентский веб-интерфейс для взаимодействия пользователя с системой.

— Подготовить конфигурацию для комплексного развертывания всех компонентов системы с использованием технологии контейнеризации.

Для достижения поставленных задач были определены нижеследующие функциональные и нефункциональные требования.

1. Функциональные требования к прототипу:

Управление учетными записями пользователей:

— Система должна обеспечивать функциональность регистрации новых пользователей с использованием адреса электронной почты и пароля.

— Должен быть реализован механизм аутентификации для предоставления доступа к системе.

— Должна быть обеспечена строгая изоляция данных каждого пользователя (проектов, документов) на уровне логики приложения и структуры базы данных.

Управление проектами и документами:

— Пользователю должна быть предоставлена возможность выполнять операции создания, переименования и удаления проектов (абстракция «книга»).

— В рамках каждого проекта должна быть доступна функциональность по созданию, переименованию и удалению документов (абстракция «глава»).

— Система должна предоставлять иерархическую структуру навигации по проектам и документам.

Работа с текстовыми данными:

— Система должна предоставлять полнофункциональный текстовый редактор для создания и модификации содержимого документов.

— Должна быть реализована функция фонового автоматического сохранения текстовых данных по мере их изменения пользователем.

— Должна быть предусмотрена возможность импорта данных из файлов форматов .txt и .docx.

Функциональность ИИ-помощника:

— Функционирование ИИ должно осуществляться в ассистирующем режиме, предполагающем обработку исключительно тех текстовых фрагментов, которые инициированы пользователем (выделенный сегмент или весь документ).

— Система должна предоставлять набор предопределенных команд (запросов) к ИИ, включая:

— Коррекция орфографических и грамматических ошибок.

— Генерация предложений для развития сюжета.

— Анализ и стилистическая адаптация текста.

— Сокращение или расширение выделенного фрагмента.

— Пользователь должен иметь возможность формировать произвольные (пользовательские) запросы к ИИ для обработки текстовых данных.

Дополнительная функциональность:

— Пользователю должна быть предоставлена возможность создавать и управлять личными заметками, не ассоциированными с конкретным проектом.

2. Нефункциональные требования к прототипу:

— Архитектура: Система должна базироваться на микросервисной архитектуре для логического разделения функциональности аутентификации и основного приложения. Данный подход призван повысить масштабируемость и упростить дальнейшее сопровождение системы.

— Безопасность: Пароли пользователей должны храниться в базе данных в хешированном виде. Для авторизации запросов к API должен применяться механизм, основанный на JWT-токенах. При проектировании подсистемы безопасности был сделан выбор в пользу Stateless-архитектуры (архитектуры без сохранения состояния), реализуемой посредством стандарта JSON Web Tokens (JWT). В отличие от классического подхода с использованием сессий (Session-based auth), где сервер обязан хранить идентификатор сессии каждого активного пользователя в оперативной памяти или базе данных (Redis/Memcached), JWT позволяет передавать всю необходимую информацию о пользователе (Claims) непосредственно в теле зашифрованного токена. Это решение устраняет необходимость выполнения дополнительных запросов к базе данных для валидации сессии при каждом HTTP-вызове, что снижает латентность системы. Более того, такой подход обеспечивает горизонтальную масштабируемость: в случае необходимости запуска нескольких экземпляров сервиса app (репликация), любой из них сможет валидировать токен, имея лишь доступ к публичному ключу подписи, без необходимости синхронизации хранилища сессий между узлами кластера.

— Развертывание и переносимость: Все компоненты проекта, включая базу данных, серверные приложения и ИИ-модель, должны быть

контейнеризированы с использованием Docker и развертываться посредством единой команды DockerCompose, что обеспечивает инвариантность среды выполнения на различных машинах.Декларативное описание инфраструктуры, включая переменные окружения и сетевые настройки, реализовано в конфигурационном файле оркестратора (рисунок 1).

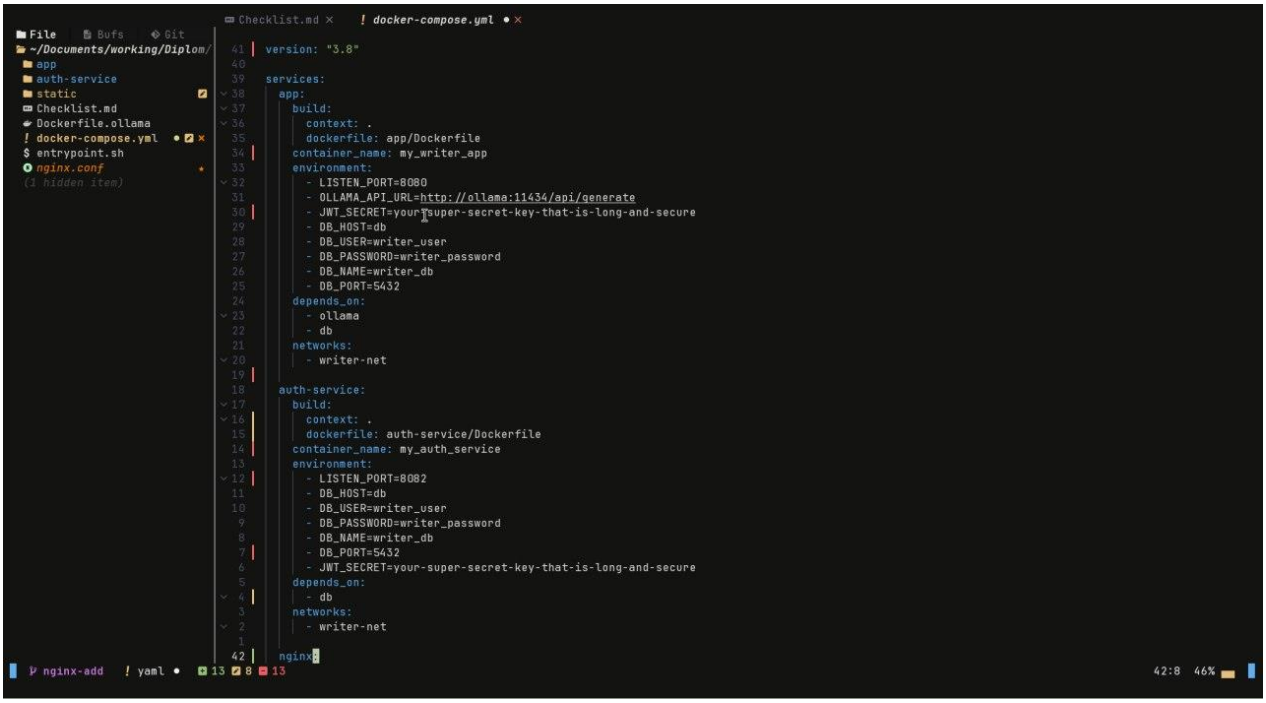

Рисунок 1 – Конфигурация оркестрации контейнеров DockerCompose

— Автономность: Система должна быть спроектирована для автономной работы с локально развернутой языковой моделью (посредством Ollama), что обеспечивает ее независимость от внешних облачных сервисов и гарантирует конфиденциальность пользовательских данных.

— Производительность: Система должна обеспечивать обработку пользовательских запросов с минимальной задержкой в рамках ограничений, накладываемых аппаратной конфигурацией, на которой развертывается прототип.

2 Проектный раздел

2.1 Проектирование архитектуры и информационной базы системы

Процесс создания современной информационной системы требует тщательного подхода к выбору архитектурных решений, поскольку именно на этом этапе закладывается фундамент производительности, масштабируемости и надежности будущего программного продукта. При проектировании системы «Платформа с ИИ-помощником для написания книг» ключевым фактором стал отказ от монолитной структуры в пользу декомпозиции на независимые компоненты. К.И. Зимина и О.Р. Лапонина справедливо указывают, что «взаимодействие микросервисов — это ключевой аспект микросервисной архитектуры, который обеспечивает работу приложения в целом» [10,с.1]. Именно качество и надежность этого взаимодействия определяют стабильность всей платформы.

Однако переход к распределенной архитектуре не должен быть самоцелью; он должен решать конкретные проблемы разработки и эксплуатации. Д.А. Кравченко отмечает: «Архитектуры микросервисов упрощают масштабирование и ускоряют разработку приложений, позволяя внедрять инновации и ускоряя вывод новых функций на рынок» [12, с.1]. В контексте разрабатываемой платформы это означает возможность независимого обновления модуля работы с ИИ без остановки основного веб-сервиса, что критически важно для обеспечения непрерывности пользовательского опыта.

При этом процесс разделения системы на компоненты требует глубокого анализа предметной области. Как подчеркивает К.А. Куликов, «ключевым вызовом является корректное определение границ между сервисами» [14, с.2]. Ошибки на этом этапе могут привести к созданию сильно связанной системы, которая унаследует недостатки монолита, но приобретет сложность распределенных вычислений. В данном проекте границы сервисов проведены на основе функционального назначения: аутентификация (auth-service) отделена от бизнес-логики (app), что соответствует принципам единственной

ответственности.Общая схема контейнерной архитектуры и взаимодействия сервисов представлена на рисунке 2.

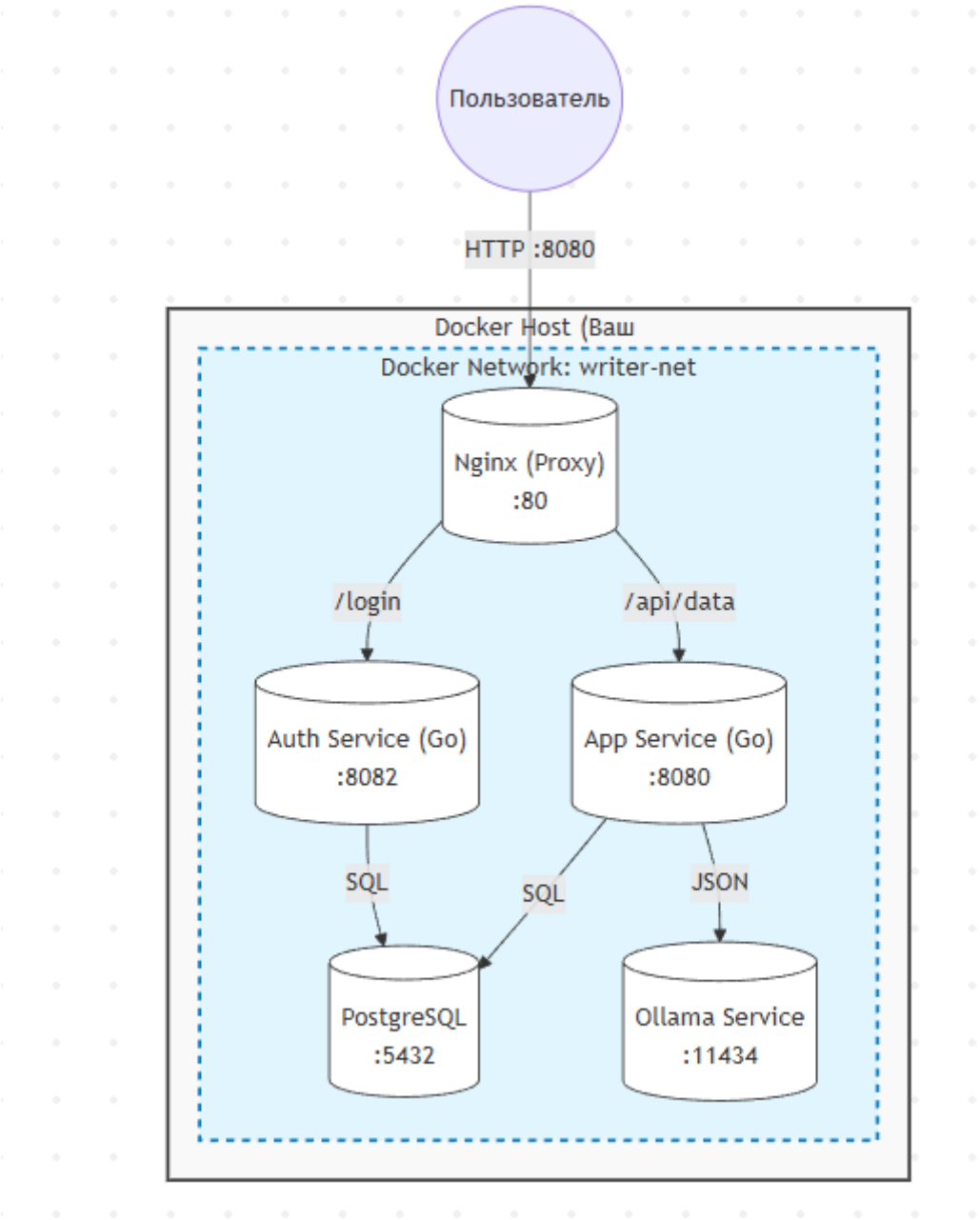

Рисунок 2 – Схема архитектуры контейнеризированного приложения

В качестве альтернативного или дополняющего подхода при проектировании внутренней структуры сервисов рассматривалась концепция модульного монолита. М.А. Земсков утверждает, что «модульные монолиты представляют собой перспективный подход в разработке масштабируемых веб-

приложений» [9, с.1]. Этот подход был частично применен внутри основного сервиса приложения, где логика работы с проектами, документами и заметками логически изолирована, но физически находится в рамках одного развертываемого модуля для упрощения отладки на ранних стадиях.

Особое внимание при проектировании уделялось вопросам информационной безопасности на уровне операционной системы и среды выполнения. Н.И. Лиманова и А.С. Анашкин напоминают, что «основополагающими принципами безопасности являются принцип наименьших привилегий и принцип необходимости» [16, с.2]. Эти принципы были положены в основу конфигурации контейнеров и настройки прав доступа к базе данных.

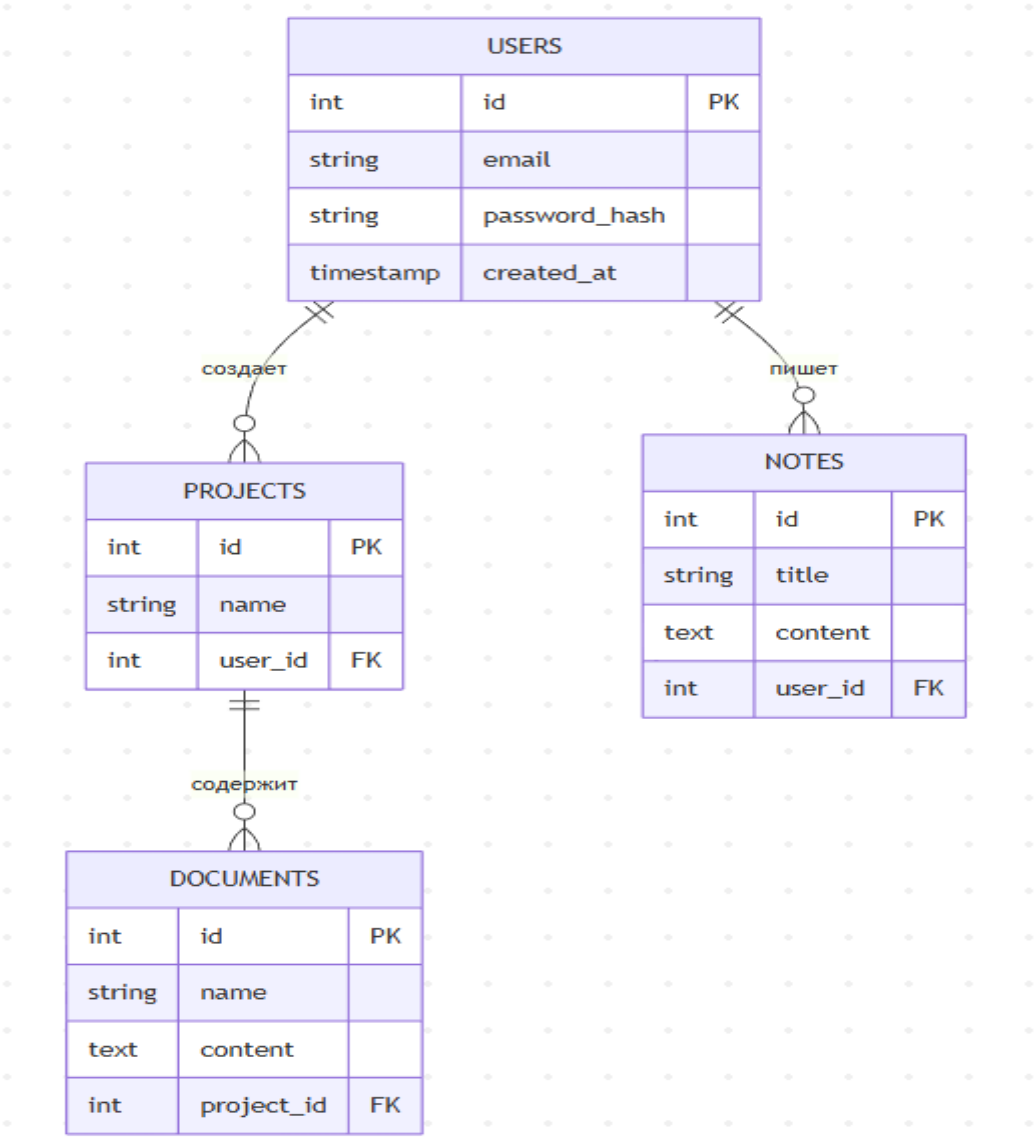

Рисунок 3 – ER-диаграмма базы данных системы

Для организации хранения данных была выбрана реляционная модель, так как она обеспечивает строгую структуру и целостность информации. А.В. Брешенков отмечает: «Существующая методология проектирования РБД позволяет разработчику обоснованно назначить ключевые и индексные поля, сформировать связи между таблицами обеспечить безопасность данных» [5, с.1]. В разработанной схеме базы данных PostgreSQL таблицы projects, documents и notes связаны внешними ключами с каскадным удалением, что гарантирует отсутствие «сиротских» записей при удалении родительских объектов. Логическая модель базы данных, отображающая связи сущностей, приведена на рисунке 3.

Выбор в пользу объектно-реляционной системы управления базами данных PostgreSQL, в противовес популярным в веб-разработке NoSQL-решениям (например, MongoDB), был продиктован спецификой предметной области и требованиями к консистентности данных. Литературное произведение, несмотря на свою текстовую природу, обладает жесткой иерархической структурой («Книга» включает «Главы», которые включают «Текст»), которая идеально ложится на реляционную модель. Использование JSON-ориентированных баз данных в данном проекте могло бы привести к дублированию данных и сложностям при реализации каскадных удалений, тогда как механизм внешних ключей (Foreign Keys) и транзакционная целостность (ACID), обеспечиваемые PostgreSQL, гарантируют, что структура проекта останется валидной даже в случае программных сбоев или аварийного завершения работы контейнеров. Кроме того, PostgreSQL предоставляет мощные встроенные средства для полнотекстового поиска и работы с JSONB-полями, что оставляет архитектурный задел для будущего внедрения функционала тегирования заметок и сложного поиска по содержимому книг без необходимости подключения дополнительных поисковых движков.

Для формализации и структурирования знаний о предметной области, особенно в части классификации пользовательских данных, использовались онтологические подходы. В.А. Лисин и Е.А. Сидорова пишут: «Одним из

методов разработки таких систем является применение онтологий – формального описания предметной области (ПрО), в качестве метакаркаса для систематизации знаний» [17, с.1]. Хотя в текущем прототипе онтология не реализована в явном виде (как RDF-граф), структура таблиц базы данных фактически отражает онтологию предметной области «литературное творчество» (Книга -> Глава -> Текст).

2.2. Разработка и реализация программных модулей

Процесс создания программного комплекса «Платформа с ИИ-помощником» базировался на фундаментальных принципах системной инженерии, предполагающих строгую декомпозицию функциональных блоков, изоляцию зон ответственности и применение промышленных стандартов контейнеризации. Как справедливо отмечают Ш. Шагулыев и соавторы, «в современном быстро меняющемся мире технологий языки программирования являются одним из ключевых элементов, формирующих цифровой мир» [29, с.1], поэтому выбор технологического стека стал фундаментом всей разработки, определяющим дальнейшие эксплуатационные характеристики системы. В качестве базового инструментария для реализации серверной части (Back-end) был выбран компилируемый язык программирования Go (Golang) версии 1.24. Данный выбор не является случайным и обусловлен результатами исследований Д.Н. Францева и О.В. Новоселовой, которые подчеркивают, что «Golang был разработан Google с акцентом на многопоточность и конкурентное программирование» [26, с.1]. Встроенные в язык примитивы — горутины (goroutines) — позволяют планировщику Go эффективно мультиплексировать тысячи одновременных HTTP-запросов на ограниченное количество потоков операционной системы. Это обеспечивает высокую пропускную способность системы при минимальном потреблении оперативной памяти, что является критически важным фактором для проектируемого решения,

функционирующего в условиях ограниченных ресурсов локальной машины пользователя.

Фундаментальным преимуществом выбора языка Go для реализации серверной части является его уникальная модель конкурентности, основанная на концепции взаимодействующих последовательных процессов (CSP). В отличие от классической потоковой модели, применяемой в языках Java или C++, где каждому клиентскому запросу выделяется поток операционной системы (OS Thread) с потреблением стека порядка 1–2 МБ, Go использует легковесные горутины (Goroutines) с начальным размером стека всего 2 КБ. Планировщик Go (Go Scheduler) реализует механизм кооперативной многозадачности, эффективно распределяя тысячи горутин на небольшом количестве системных ядер. Для системы, взаимодействующей с ИИ, это имеет критическое значение: запросы к нейросети являются длительными операциями ввода-вывода (I/O Bound). В момент ожидания ответа от Ollama горутина, обслуживающая запрос, «засыпает» и не блокирует системный поток, позволяя процессору обрабатывать другие задачи (например, автосохранение или аутентификацию). Это обеспечивает высокую отзывчивость системы даже в моменты пиковой нагрузки на модуль искусственного интеллекта.

Архитектура системы спроектирована по микросервисному принципу, что соответствует современным тенденциям разработки распределенных систем. Д.А. Кравченко указывает, что «архитектуры микросервисов упрощают масштабирование и ускоряют разработку приложений, позволяя внедрять инновации и ускоряя вывод новых функций на рынок» [12, с.1]. Руководствуясь этим тезисом, мы разделили функциональность платформы на два независимых компонента: auth-service, отвечающий исключительно за идентификацию пользователей и генерацию токенов доступа, и app, реализующий основную бизнес-логику управления проектами и документами.Файловая структура проекта организована в соответствии с микросервисной архитектурой, где каждый сервис (app, auth-service) имеет собственную директорию с исходным кодом и конфигурацией Docker (рисунок 4). Однако переход к распределенной

архитектуре несет в себе риски избыточной сложности. Как подчеркивает К.А. Куликов, «ключевым вызовом является корректное определение границ между сервисами» [14, с.2]. В рамках данного проекта границы были проведены строго по функциональному признаку (Bounded Contexts), чтобы избежать сильной связности компонентов. Взаимодействие между сервисами и клиентской частью организовано через RESTful API, поскольку, согласно А.А. Мельнику, «от качества проектирования и построения архитектуры зависит производительность системы, ее масштабируемость и сопровождаемость» [20, с.2]. Для реализации маршрутизации внутри сервисов использовался стандартный мультиплексор http.NewServeMux из стандартной библиотеки Go, что позволило избежать использования громоздких сторонних фреймворков и снизить размер итоговых бинарных файлов.

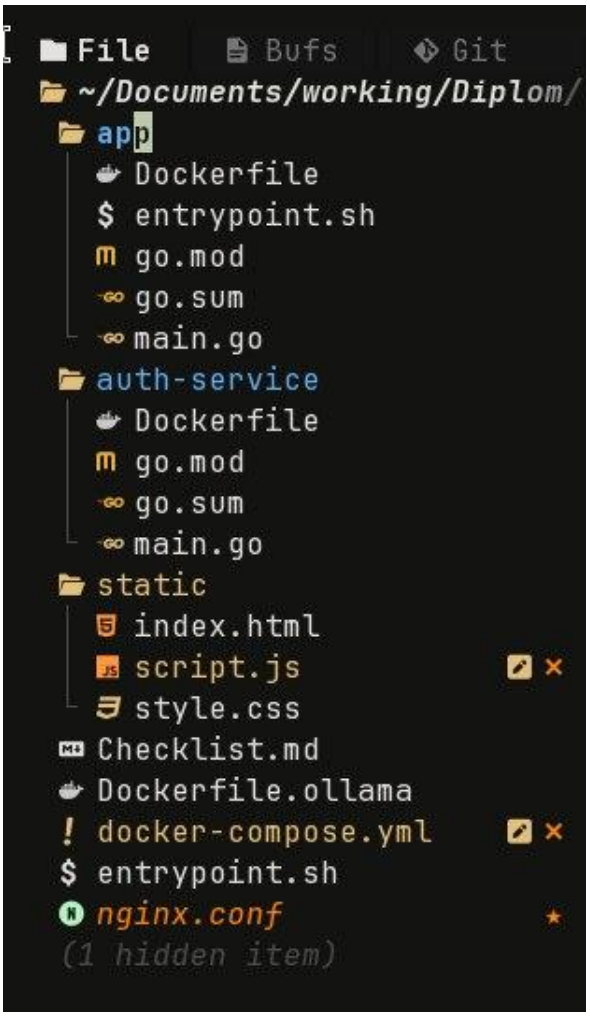

Рисунок 4 – Файловая структура проекта в среде разработки

Особое внимание в процессе разработки было уделено механизмам взаимодействия сервисов и обработки запросов. К.И. Зимина и О.Р. Лапонина отмечают, что «взаимодействие микросервисов — это ключевой аспект микросервисной архитектуры, который обеспечивает работу приложения в целом» [10, с.1]. В приложении реализован паттерн «Цепочка обязанностей» (Chain of Responsibility) посредством внедрения промежуточного программного обеспечения (Middleware). Первый слой, CORS Middleware, обеспечивает корректную обработку кросс-доменных запросов, добавляя необходимые заголовки в ответы сервера, что критично для работы браузерного клиента. Второй слой, Auth Middleware, реализует защиту API от несанкционированного доступа. Е.И. Ляшов утверждает, что «аутентификация служит ключевым элементом защиты пользовательских данных и предотвращения несанкционированного доступа» [19, с.1]. Мы использовали индустриальный стандарт JSON Web Token (JWT). Алгоритм работы middleware заключается в перехвате входящего запроса, извлечении токена, валидации его цифровой подписи с помощью секретного ключа и, что наиболее важно, внедрении извлеченного идентификатора пользователя (UserID) непосредственно в контекст запроса (context.Context). Такой подход позволяет передавать аутентификационные данные сквозь все слои приложения без необходимости повторных обращений к базе данных.

Ниже представлена схема информационных потоков (рисунок 5), демонстрирующая маршрутизацию запросов от клиента к микросервисам и базе данных, а также последовательность обработки вызовов: от первичной валидации токена безопасности на уровне промежуточного ПО (Middleware) до выполнения транзакций и извлечения данных из репозитория PostgreSQL, с последующей агрегацией полученных результатов и их сериализацией в формат JSON для отправки конечного ответа пользователю, что обеспечивает стандартизированный и безопасный обмен данными в рамках архитектуры REST API

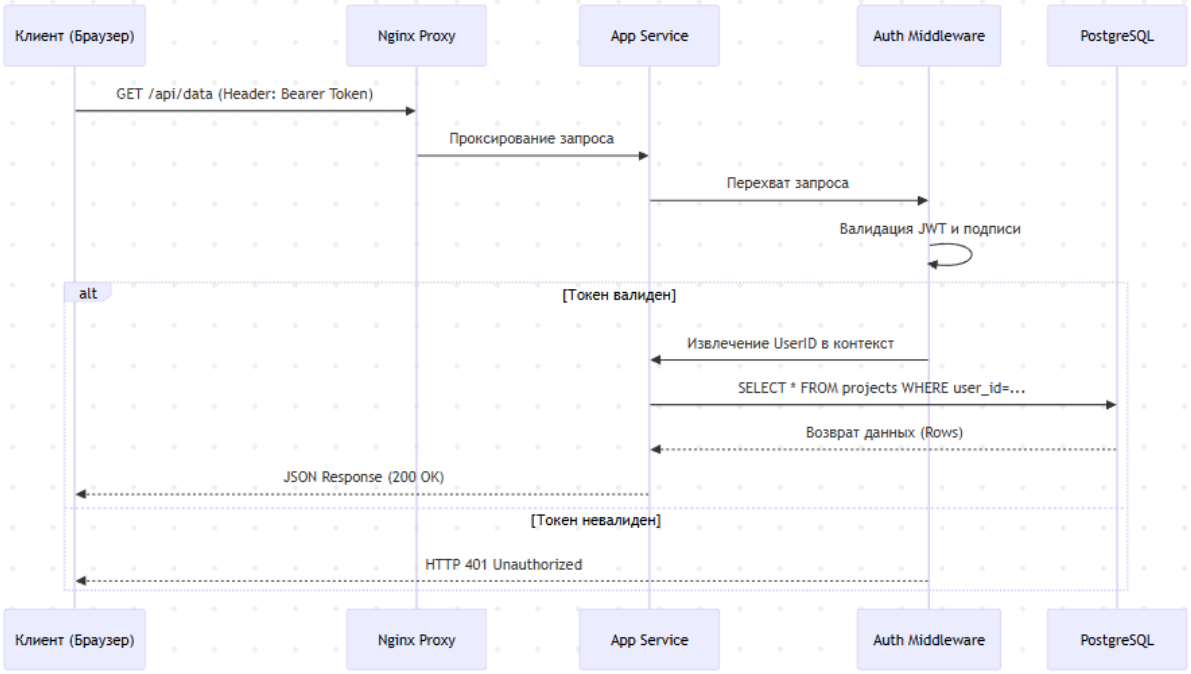

Рисунок 5 – Схема обработки авторизованного запроса

Хранение состояния системы и персистентность данных реализованы на базе реляционной системы управления базами данных PostgreSQL. А.В. Брешенков пишет: «Существующая методология проектирования РБД позволяет разработчику обоснованно назначить ключевые и индексные поля, сформировать связи между таблицами обеспечить безопасность данных» [5, с.1]. Следуя этому принципу, схема базы данных была спроектирована с использованием нормализованных структур. Таблицы projects (проекты) и documents (главы) связаны ограничением целостности по внешнему ключу с опцией ON DELETE CASCADE. Это техническое решение гарантирует, что при удалении пользователем проекта СУБД автоматически и атомарно удалит все связанные с ним текстовые документы, предотвращая нарушение целостности данных и накопление «сиротских» записей. Структура данных фактически отражает онтологию предметной области. Как отмечают В.А. Лисин и Е.А. Сидорова, онтологии служат «метакаркасом для систематизации знаний» [17, с.1], и в нашем случае иерархия «Проект — Документ — Контент» является формализованным представлением структуры литературного произведения. Для

обеспечения надежности соединения приложения с базой данных в коде применен программный паттерн Retry («Повторная попытка»): при запуске сервис в цикле опрашивает доступность порта СУБД, что предотвращает аварийное завершение работы контейнеров при гонке условий во время старта инфраструктуры.

Обеспечение безопасности, переносимости и изоляции среды выполнения реализовано через современные технологии контейнеризации. Д.Р. Чиганов отмечает: «Контейнеризация позволяет запускать приложения в единообразной среде независимо от операционной системы и окружающей инфраструктуры» [28, с.2]. Мы использовали платформу Docker и стратегию многоэтапной сборки (Multistage Build), описанную в Dockerfile. К.В. Карнаухов добавляет, что это делает процесс развертывания «портируемым и предсказуемым образом» [11, с.1]. На первом этапе сборки используется «тяжелый» образ с полным набором инструментов компиляции и зависимостей Go, где происходит сборка бинарного файла с флагами оптимизации линковщика (-ldflags="-s -w"), удаляющими отладочную информацию. На втором этапе этот бинарный файл копируется в минималистичный образ Alpine Linux, не содержащий исходного кода и лишних системных утилит. Это соответствует принципам безопасности, описанным О.В. Ехлаковым: «В современных условиях контейнеризация, прежде всего посредством Docker, зарекомендовала себя как ключевой инструмент сегрегации приложений и их зависимостей» [7, с.1]. Также при конфигурации контейнеров соблюдаются принципы наименьших привилегий, о важности которых напоминают Н.И. Лиманова и А.С. Анашкин [16, с.2], путем жесткого ограничения прав доступа внутри изолированной сети Docker Network.

Применение стратегии минимизации размера контейнеров на базе дистрибутива Alpine Linux выполняет не только функцию экономии дискового пространства, но и служит важным элементом эшелонированной защиты информационной системы. Стандартные полновесные образы операционных систем (такие как Ubuntu или Debian) содержат сотни предустановленных

библиотек, пакетных менеджеров и системных утилит, которые могут быть использованы злоумышленником в качестве векторов атаки в случае компрометации приложения (например, для эскалации привилегий или организации обратного шелла). Использование минималистичного образа (scratch или alpine), содержащего исключительно скомпилированный бинарный файл приложения и минимально необходимый набор корневых сертификатов, радикально сокращает так называемую «поверхность атаки» (Attack Surface). В такой среде отсутствуют интерпретаторы командной строки (bash/sh) и инструменты для скачивания файлов (curl/wget), что делает эксплуатацию потенциальных уязвимостей (RCE) практически невозможной, так как вредоносный код просто не сможет быть исполнен в изолированной и обедненной среде контейнера.

Интеграция с генеративными моделями искусственного интеллекта потребовала решения нетривиальных задач обеспечения высокой доступности и синхронизации процессов. Д.А. Филисов подчеркивает, что «высокая доступность и отказоустойчивость являются основополагающими принципами для высоконагруженных приложений» [25, с.2]. Поскольку загрузка нейросети Ollama и весов модели в видеопамять занимает значительное время, был разработан специализированный скрипт инициализации entrypoint.sh.Фрагмент программного кода, реализующий алгоритм проверки готовности нейросети и автоматической загрузки модели, представлен на рисунке 6. Он реализует механизм Health Check: скрипт запускает сервер Ollama в фоновом режиме и входит в цикл активного ожидания, опрашивая локальный API до получения успешного ответа. Только после подтверждения полной готовности нейросети управление передается основному процессу. А. Ю. А. Альфара и коллеги указывают на важность наблюдаемости систем: «Для повышения надежности работы программных систем используют мониторинг с последующим анализом его результатов» [1, с.1], поэтому все этапы инициализации ИИ, включая статус загрузки модели и ошибки инференса, логируются в стандартный вывод контейнера для последующего анализа администратором.Алгоритм

инициализации сервиса ИИ и проверки доступности модели показан на рисунке 7.

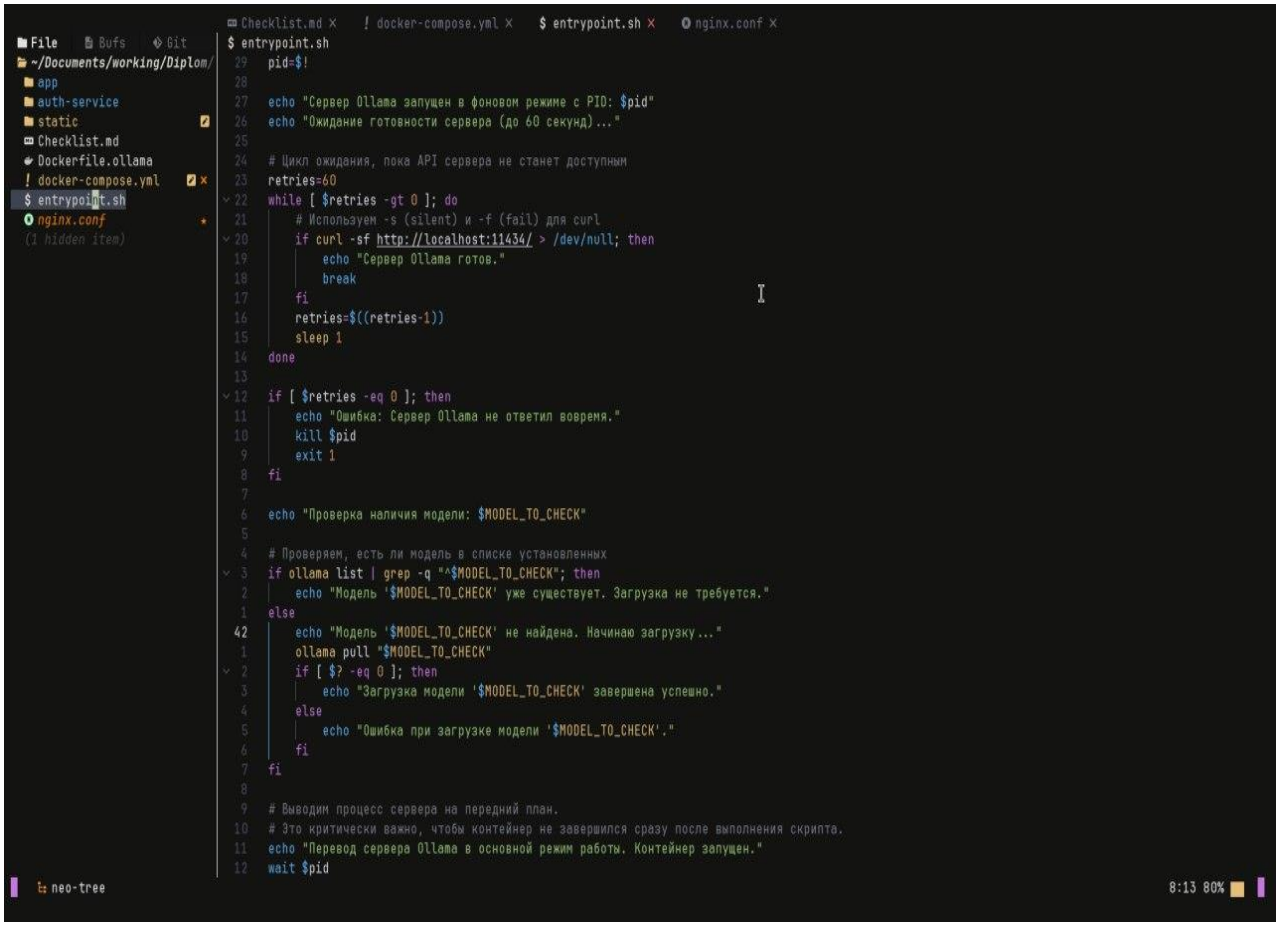

Рисунок 6 – Реализация механизма HealthCheck для сервиса ИИ

Представленный фрагмент кода демонстрирует практическую реализацию стратегии «активного ожидания» (Active Waiting), которая критически необходима для корректной оркестрации зависимых сервисов в асинхронной среде. Стандартные механизмы проверки статуса контейнера в Docker сообщают лишь о факте запуска корневого процесса, но не гарантируют готовность приложения обрабатывать входящие HTTP-запросы, что в случае с «тяжелыми» нейросетевыми бэкендами часто приводит к ошибкам соединения на начальном этапе работы (Race Condition). Скрипт циклически опрашивает локальный порт сервиса инференса, и только после стабилизации сетевого соединения переходит к этапу автоматического обеспечения контентом (Model Provisioning). Логика скрипта обеспечивает идемпотентность развертывания:

система самостоятельно верифицирует целостность локального кэша моделей и, в случае отсутствия требуемой архитектуры (в данном случае gemma3:1b), инициирует её загрузку из удаленного репозитория реестра Ollama. Такой подход исключает необходимость ручного администрирования контейнера пользователем через командную строку, делая систему полностью автономной и готовой к работе сразу после «холодного» старта (Zero-touch provisioning).

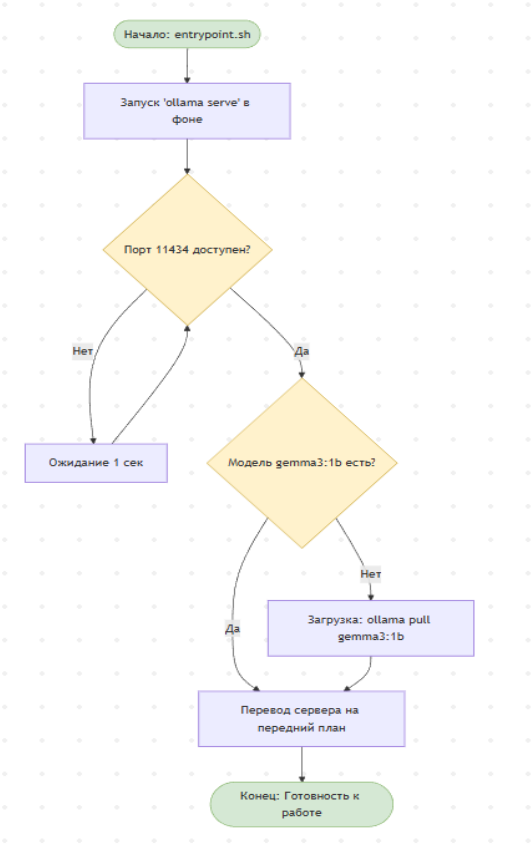

Рисунок 7 – Блок-схема алгоритма инициализации сервиса Ollama

Ключевым технологическим барьером при локальном развертывании больших языковых моделей (LLM) являются высокие требования к пропускной способности памяти и объему VRAM. Для обеспечения работоспособности системы на потребительском оборудовании (например, видеокартах с объемом памяти 4–8 ГБ) в проекте применяется технология пост-тренировочного квантования (Post-Training Quantization). Используемая платформа Ollama оперирует моделями в формате GGUF, который позволяет снизить разрядность весов нейросети с исходных 16 бит (FP16) до 4 или 5 бит (Int4/Int5) с

минимальной потерей качества генерации (перплексии). Данный подход позволяет сократить потребление оперативной памяти в 3–4 раза и существенно ускорить инференс за счет уменьшения объема данных, передаваемых по шине PCIe. Таким образом, становится возможным запуск моделей с миллиардами параметров в режиме реального времени без использования дорогостоящих кластерных вычислительных мощностей, что полностью соответствует концепции экономической эффективности разрабатываемого решения.

Для конечного пользователя сложность внутренней архитектуры скрыта за интуитивно понятным интерфейсом, так как, согласно А.Ю. Володину, «цифровые инструменты и интерфейсы задают рамки поиска и анализа, акцентируя одни аспекты и скрывая другие» [6, с.7]. Чтобы не ограничивать автора техническими деталями, все запросы маршрутизируются через единую точку входа. В этой роли выступает веб-сервер Nginx, настроенный как обратный прокси-сервер (API Gateway). Он принимает входящий трафик на стандартный HTTP-порт, терминирует соединения и распределяет запросы между микросервисами app и auth-service в зависимости от URL-пути. Это позволяет скрыть сложную топологию внутренней сети Docker, решить проблему Cross-Origin Resource Sharing (CORS) на инфраструктурном уровне и обеспечить единый фасад безопасности для всей платформы.

Внедрение в архитектуру веб-сервера Nginx в качестве обратного прокси-сервера (Reverse Proxy) обусловлено не только необходимостью маршрутизации запросов, но и требованиями к стабильности сетевого взаимодействия. Сервисы, написанные на прикладных языках (в данном случае Go), оптимизированы для выполнения бизнес-логики, но могут быть недостаточно эффективны при обработке "медленных" клиентов или некорректных HTTP-заголовков. Nginx берет на себя задачу терминации входящих соединений, буферизацию запросов и защиту от простейших DDoS-атак (например, Slowloris). Кроме того, такая архитектура позволяет реализовать паттерн «SSL Termination», вынося ресурсоемкие задачи шифрования трафика (HTTPS) на уровень инфраструктуры, освобождая вычислительные ресурсы приложения

для обработки логики. В рамках контейнерной среды Docker это также решает проблему разрешения имен (Service Discovery), позволяя сервисам обращаться друг к другу по внутренним сетевым алиасам, скрытым от внешнего наблюдателя.

2.3 Руководство пользователя и тестирование прототипа

Заключительным и наиболее ответственным этапом реализации проекта стала разработка регламента взаимодействия пользователя с системой и проведение комплексных испытаний разработанного программного обеспечения. Проектирование пользовательского интерфейса (UI/UX) опиралось на теоретические основы, сформулированные А.Ю. Володиным, который отмечает, что «цифровые инструменты и интерфейсы задают рамки поиска и анализа, акцентируя одни аспекты и скрывая другие» [6, с.7]. Следуя этому тезису, визуальная составляющая платформы была реализована как одностраничное веб-приложение (Single Page Application, SPA), функционирующее в среде современного браузера. Дизайн-концепция базируется на принципах минимализма и «фокусированного письма», чтобы визуальный шум не отвлекал автора от творческого процесса. Технически интерфейс построен на манипуляции объектной моделью документа (DOM) посредством чистого JavaScript, а взаимодействие с сервером организовано через асинхронные HTTP-запросы (AJAX) с использованием Fetch API. Это архитектурное решение позволило добиться мгновенного отклика интерфейса и обновлять данные на странице без ее полной перезагрузки, обеспечивая пользовательский опыт, сравнимый с нативными настольными приложениями.

Визуальная организация интерфейса была спроектирована с учетом принципов когнитивной эргономики и теории снижения когнитивной нагрузки. В процессе литературного творчества автор находится в состоянии потока, которое легко нарушается избыточными визуальными стимулами или сложной навигацией. Поэтому в приложении реализован паттерн «Спокойная технология» (Calm Technology): элементы управления ИИ и форматированием

скрыты по умолчанию и появляются контекстно, только при явном выделении текста или наведении курсора. Цветовая палитра интерфейса использует низкоконтрастные сочетания для фоновых элементов и акцентные цвета только для активных действий, что снижает зрительное утомление при длительных сессиях работы. Такой подход трансформирует интерфейс из инструмента управления в прозрачную среду, где техническая сложность взаимодействия с нейросетью полностью абстрагирована от пользователя, позволяя сосредоточиться исключительно на семантике текста.

Осознанным архитектурным решением при разработке клиентской части стал отказ от использования тяжеловесных JavaScript-фреймворков (таких как React, Vue или Angular) в пользу нативных возможностей браузера (Vanilla JS). Современные тенденции веб-разработки часто ведут к необоснованному усложнению технологического стека, что влечет за собой увеличение размера загружаемого кода (bundle size) и повышенное потребление оперативной памяти на устройстве клиента за счет работы виртуального DOM и механизмов реактивности. Для локально развертываемой системы, которая может функционировать на оборудовании с ограниченными ресурсами параллельно с ресурсоемкой нейросетью, критически важна оптимизация каждого мегабайта памяти. Прямое манипулирование объектной моделью документа (DOM) и использование нативного Fetch API позволили создать легковесный, мгновенно загружающийся интерфейс, который не требует этапа транспиляции или сборки (build step), упрощая процесс развертывания и отладки, а также снижая порог входа для потенциальных разработчиков, желающих модифицировать открытый исходный код проекта.

Сценарий работы с платформой начинается с процедуры аутентификации, которая является обязательным условием доступа к функционалу. При загрузке приложения клиентский скрипт инициирует проверку наличия валидного JSON Web Token (JWT) в локальном хранилище браузера (localStorage). В случае отсутствия токена или истечения срока его действия система автоматически блокирует доступ к основному контенту, накладывая затемняющий слой

(overlay) и отображая модальное окно входа. В данном модуле реализована логика динамического переключения между формами авторизации и регистрации без перезагрузки страницы. При отправке пользовательских данных скрипт выполняет их первичную валидацию и отправляет POST-запрос на шлюз аутентификации. Полученный в ответе сервера токен сохраняется в защищенном хранилище браузера и автоматически добавляется в заголовок Authorization всех последующих запросов к API, что обеспечивает прозрачность процесса безопасности для конечного пользователя.

Организация рабочего пространства и навигация по структуре литературных произведений осуществляются через левую боковую панель. Иерархическая модель данных «Проект (Книга) — Документ (Глава)» визуализирована в виде интерактивного аккордеона. Программная логика клиентской части поддерживает локальное состояние приложения в объекте appData, который синхронизируется с сервером. При выборе пользователем конкретной главы система инициирует GET-запрос для получения ее содержимого и помещает текст в центральную область редактирования. Для управления элементами структуры предусмотрена система контекстных меню, позволяющая создавать новые сущности, переименовывать их или удалять. Важно отметить, что критические операции, такие как удаление целого проекта, защищены механизмом подтверждения намерения (confirm), что минимизирует риск случайной потери значительных объемов данных.Внешний вид основного рабочего пространства пользователя представлен на рисунке 8.

Как демонстрирует представленная иллюстрация, компоновка интерфейса выполнена по принципу четкого визуального зонирования: инструменты управления структурой и навигации вынесены в отдельный боковой блок, что позволяет максимально освободить центральную область для непосредственной работы с текстом. Такое решение обеспечивает эргономичность рабочего процесса, позволяя пользователю мгновенно переключаться между менеджментом глав и творческим процессом, не теряя визуального контекста и не перегружая внимание избыточными элементами управления.

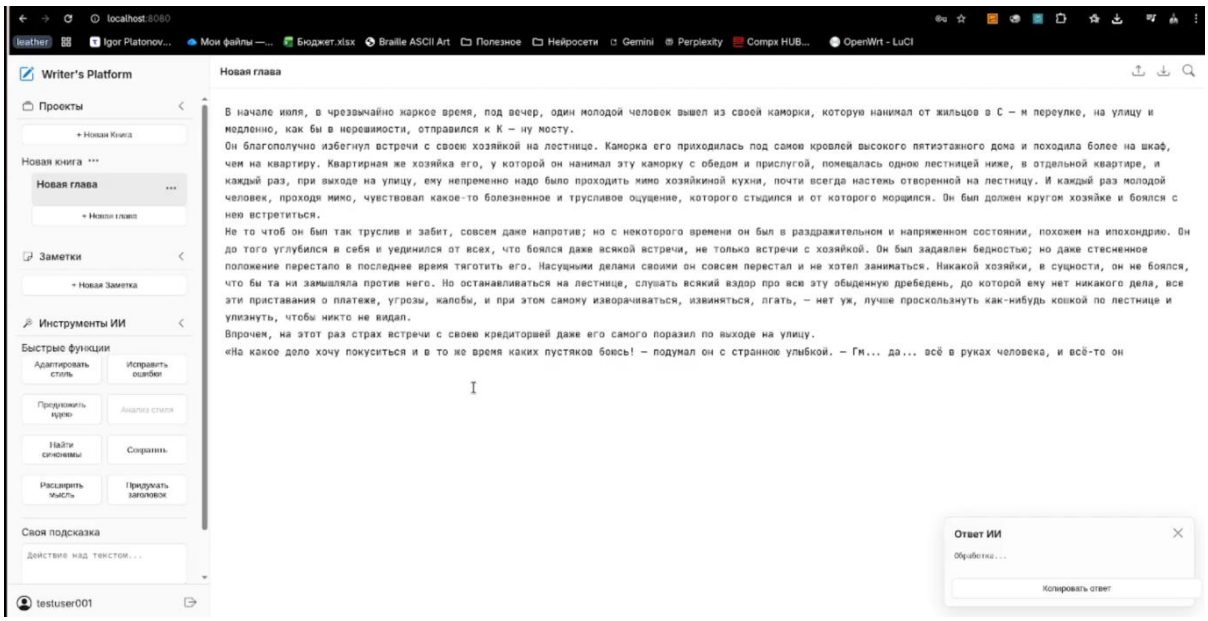

Рисунок 8 – Пользовательский интерфейс платформы: навигация и редактор

Центральным элементом системы является текстовый редактор, в который интегрирован сложный алгоритм автоматического сохранения изменений. Реализация данной функции базируется на обработке события ввода (input) с применением паттерна «debounce» (устранение дребезга). Скрипт отслеживает активность пользователя и запускает таймер отложенной записи: только если пользователь прекращает набор текста более чем на 500 миллисекунд, приложение формирует PUT-запрос с обновленным содержимым и отправляет его на сервер. Это техническое решение полностью соответствует подходу к автоматизации, описанному М.В. Безпятым, согласно которому технологии должны быть направлены на «устранение утомительных, повторяющихся задач и сведение к минимуму ручного вмешательства» [4, с.2]. В данном случае система освобождает автора от необходимости постоянно нажимать кнопку сохранения, гарантируя при этом целостность данных даже при внезапном закрытии вкладки браузера. Также в редактор встроен модуль импорта файлов, использующий File API для чтения и преобразования бинарных данных из форматов DOCX и PDF в плоский текст непосредственно на стороне клиента.

Ключевая инновация платформы — контекстно-зависимая панель инструментов искусственного интеллекта — активируется исключительно по инициативе пользователя. При выделении фрагмента текста в редакторе интерфейс отображает панель доступных команд, таких как «Адаптировать стиль», «Исправить ошибки» или «Продолжить мысль». При выборе команды клиентское приложение формирует составной промпт, конкатенируя системную инструкцию и выделенный пользовательский текст, и отправляет его на сервер инференса. Важнейшим аспектом реализации является то, что результат генерации не заменяет исходный текст автоматически, а отображается в отдельном модальном окне поверх интерфейса. Это реализует принцип методического контроля, описанный О.И. Пугач и Н.В. Старцевой: «полученные материалы подлежат обязательному контролю со стороны преподавателя» [24, с.16]. В архитектуре нашей системы роль эксперта выполняет сам автор, который принимает решение о целесообразности использования предложенных нейросетью правок, сохраняя тем самым уникальный авторский стиль и контроль над произведением.Пример взаимодействия с ИИ-помощником и отображение результата генерации показаны на рисунке 9.

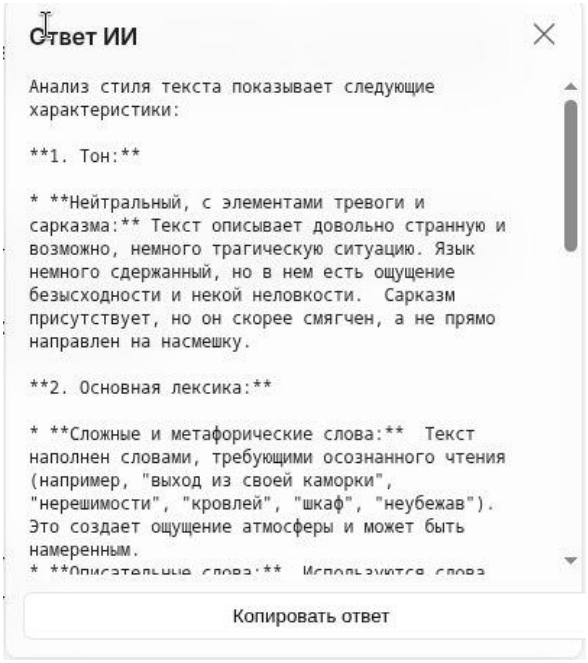

Рисунок 9 – Интерфейс получения результатов генерации от ИИ

Для подтверждения соответствия системы заявленным функциональным требованиям и стандартам надежности было проведено комплексное тестирование прототипа. Как указывают А. Ю. А. Альфара и коллеги, «для повышения надежности работы программных систем используют мониторинг с последующим анализом его результатов» [1, с.1]. Руководствуясь этим принципом, мы развернули испытательный стенд на базе технологии Docker и провели серию тестов с логированием ключевых метрик производительности контейнеров.Мониторинг состояния запущенных сервисов и анализ логов в реальном времени осуществлялся через терминальный интерфейс управления Docker (рисунок 10). В ходе проверки подсистемы безопасности тестировалась устойчивость API к несанкционированному доступу. И.Д. Крылов и соавторы предупреждают, что уязвимости веб-приложений могут привести к значительному ущербу [13, с.1], поэтому мы верифицировали сценарии атак типа IDOR (Insecure Direct Object References). Тесты подтвердили, что сервер корректно отклоняет запросы к ресурсам без валидного токена (возвращая статус HTTP 401 Unauthorized), а также блокирует попытки доступа к чужим документам путем подмены идентификаторов в URL, обеспечивая строгую изоляцию пользовательских данных.

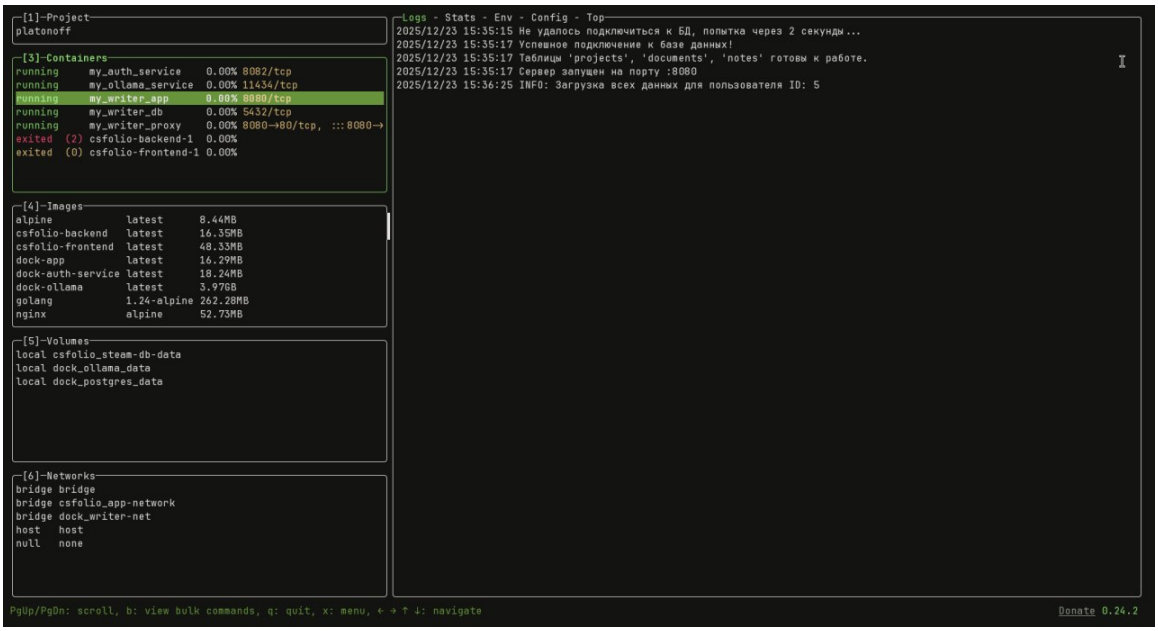

Рисунок 10 – Мониторинг статуса контейнеров и логов системы

Отдельный блок испытаний был посвящен проверке интеграции с большой языковой моделью (LLM) и оценке производительности под нагрузкой. Д.А. Филисов подчеркивает, что «высокая доступность и отказоустойчивость являются основополагающими принципами для высоконагруженных приложений» [25, с.2]. Тестирование сценария «холодного старта» показало, что разработанный скрипт инициализации (entrypoint) успешно удерживает приложение в режиме ожидания до полной загрузки весов модели в видеопамять графического ускорителя, предотвращая аварийное завершение работы контейнера. Замеры производительности на оборудовании с поддержкой CUDA показали среднюю скорость генерации порядка 40 токенов в секунду, что обеспечивает комфортную интерактивную работу без длительных задержек. Финальный этап тестирования включал проверку сценариев удаленной работы. С использованием технологии безопасного туннелирования был организован внешний доступ к локальному веб-серверу, что позволило верифицировать корректность работы адаптивной верстки на мобильных устройствах. Для проверки адаптивности интерфейса на мобильных устройствах использовалось туннелирование трафика через сервис Ngrok (рисунок 11).Совокупность положительных результатов проведенных испытаний позволяет сделать обоснованный вывод о технической зрелости прототипа и его готовности к опытной эксплуатации.

Рисунок 11 – Организация защищенного туннеля для внешнего тестирования

3 Расчет показателей экономической эффективности

3.1 Расчет стоимости разработки программного продукта

Формирование стоимости программного продукта как объекта интеллектуальной собственности требует комплексного подхода к учету всех видов ресурсов, затраченных на этапах жизненного цикла разработки. В условиях цифровой экономики программное обеспечение классифицируется как нематериальный актив (НМА), первоначальная стоимость которого складывается из совокупности фактических затрат на его создание, доведение до состояния пригодности к использованию и последующее внедрение. Применение затратного метода оценки в данном случае является наиболее обоснованным, поскольку он позволяет с высокой точностью капитализировать расходы на научно-исследовательские и опытно-конструкторские работы (НИОКР), избегая субъективности, свойственной доходному или сравнительному подходам на этапе прототипирования. Корректная калькуляция себестоимости не только служит базой для принятия управленческих решений о целесообразности дальнейшего инвестирования, но и формирует фундамент для последующего ценообразования при возможной коммерциализации продукта.

Экономическая оценка процесса создания информационной системы базируется на методе калькуляции фактической себестоимости, который является общепринятым стандартом при постановке на баланс нематериальных активов (НМА) собственной разработки. В отличие от типовой разработки веб-сайтов или стандартных учетных систем, создание платформы с интегрированным искусственным интеллектом относится к категории научно-исследовательских и опытно-конструкторских работ (НИОКР). Данная специфика кардинально меняет структуру затрат, смещая акцент с рутинного кодирования на этапы проектирования, экспериментирования и отладки сложных архитектурных решений. В этом контексте особую роль играет правильная организация инженерного труда и выбор методологии управления

проектом. Как отмечает П.А. Жигулев в своем исследовании, «гибкие методологии, напротив, более значимы в творчески ориентированных задачах, к числу которых в том числе можно отнести и процесс разработки игровых приложений» [8, с.2]. Разработка системы с генеративным ИИ по своей природе близка к созданию игровых движков, так как требует итеративного подхода и постоянной адаптации требований по мере получения результатов тестирования прототипа, что подтвердило правильность выбора гибкой модели управления при реализации данного проекта.

Доминирующую долю в структуре себестоимости программного продукта традиционно составляет фонд оплаты труда (ФОТ) инженерного персонала. Особенность данного проекта заключалась в том, что он реализовывался силами одного специалиста, что потребовало от разработчика совмещения компетенций сразу нескольких профильных ролей. Во-первых, выполнялись функции Backend-инженера: проектирование микросервисной архитектуры на языке Go, реализация многопоточной обработки запросов и обеспечение безопасности API. Во-вторых, требовались навыки DevOps-инженера для настройки среды контейнеризации Docker, написания сценариев многоэтапной сборки и оркестрации взаимодействия сервисов через Nginx. В-третьих, ключевую роль играли компетенции ML-инженера (Machine Learning) для интеграции локальной языковой модели, настройки параметров инференса и оптимизации работы нейросети в условиях ограниченных ресурсов.

Учитывая мультидисциплинарный и наукоемкий характер задач, общая трудоемкость разработки была оценена в 106 человеко-часов. Этот объем времени включил в себя не только непосредственное написание программного кода, но и предварительный анализ предметной области, проектирование схемы базы данных, а также длительные этапы интеграционного тестирования и отладки взаимодействия с нейросетью. Расчет стоимости трудозатрат производился исходя из средней рыночной ставки специалиста уровня Junior+/Middle, способного решать комплексные задачи без постоянного наставничества, которая была принята на уровне 600 рублей в час. Таким

образом, путем умножения объема затраченного времени на часовую тарифную ставку был сформирован основной фонд заработной платы разработчика, составивший 63 600 рублей.

В соответствии с действующим налоговым законодательством Российской Федерации, работодатель обязан производить отчисления во внебюджетные фонды с сумм начисленной заработной платы. Эти страховые взносы являются обязательной частью прямых затрат на производство программного обеспечения. При применении стандартной совокупной ставки в размере 30 процентов (включающей взносы в Пенсионный фонд, Фонд обязательного медицинского страхования и Фонд социального страхования), величина дополнительных начислений на фонд оплаты труда составила 19 080 рублей. Эти средства, хотя и не выплачиваются разработчику на руки, увеличивают фактическую себестоимость разработки и должны быть в обязательном порядке учтены в экономическом расчете.

Специфика разработки информационных систем, интегрирующих технологии искусственного интеллекта, накладывает особые требования к структуре материально-технического обеспечения проекта. В отличие от создания стандартных веб-приложений, где основным ресурсом выступает время программиста, разработка и отладка нейросетевых модулей сопряжена с необходимостью использования высокопроизводительных вычислительных мощностей. Процессы квантования моделей, локального инференса и тестирования гипотез требуют использования специализированных графических ускорителей и значительных объемов оперативной памяти, что переводит амортизационные отчисления из разряда второстепенных расходов в категорию значимых статей себестоимости. Игнорирование износа дорогостоящего оборудования привело бы к искусственному занижению фактической стоимости разработки и искажению показателей экономической эффективности проекта.

Вторым по значимости фактором, формирующим стоимость разработки систем с элементами искусственного интеллекта, является материально-

техническое обеспечение. Специфика работы больших языковых моделей (LLM) накладывает жесткие требования к вычислительным мощностям. И.А. Андреев и соавторы в своем исследовании подчеркивают важность детального анализа «скорости работы процессора на основе различных характеристик, таких как тактовая частота, количество ядер, объем кэш-памяти и другие» [2, с.1]. В контексте нейросетевых вычислений критическим параметром становится не только мощность центрального процессора, но и наличие производительного графического ускорителя (GPU) с поддержкой технологии CUDA и достаточным объемом видеопамяти для загрузки весов модели. Стандартные офисные компьютеры оказались непригодны для решения поставленных задач, так как инференс нейросети на CPU занимает неприемлемо много времени. В связи с этим разработка велась на высокопроизводительной рабочей станции рыночной стоимостью 90 000 рублей. Технические характеристики аппаратно-программного комплекса, использованного при разработке и расчете амортизации, приведены на рисунке 12.

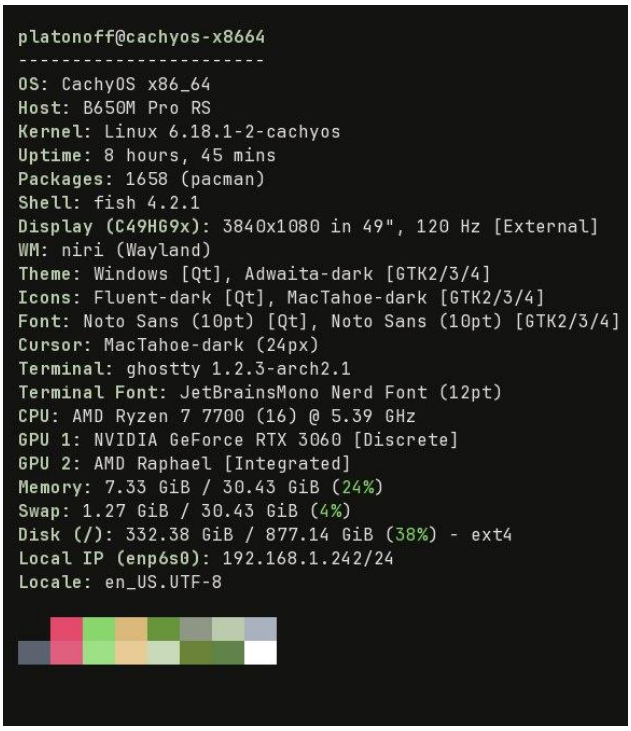

Рисунок 12 – Технические характеристики рабочей станции разработчика

Для корректного отнесения стоимости оборудования на себестоимость конкретного программного продукта был применен линейный метод начисления амортизации. Исходя из установленного срока полезного использования вычислительной техники, равного 60 месяцам (5 лет), ежемесячная сумма амортизационных отчислений составила 1 500 рублей. Поскольку активная фаза разработки, требующая использования данного оборудования, длилась один месяц, именно эта сумма была включена в расчет. Дополнительно были учтены прямые накладные расходы на электроэнергию. Функционирование мощной видеокарты и процессора под нагрузкой характеризуется высоким энергопотреблением (порядка 300-400 Вт/ч). Расчет, произведенный на основе действующего тарифа на электроэнергию и времени работы оборудования, показал, что затраты по этой статье составили 175 рублей.

Таким образом, итоговая себестоимость разработки прототипа информационной системы была сформирована путем суммирования всех статей прямых и косвенных затрат: основного фонда оплаты труда, обязательных страховых взносов, амортизационных отчислений и расходов на электроэнергию. Общая сумма капитальных вложений в создание нематериального актива составила 84 355 рублей.Структура статей затрат в общей себестоимости программного продукта наглядно представлена на рисунке 13. Данный показатель демонстрирует высокую экономическую эффективность внутренней разработки по сравнению с потенциальной стоимостью приобретения или заказа аналогичного программного решения у сторонних подрядчиков, где цена могла бы быть кратно выше за счет заложенной нормы прибыли и коммерческих рисков.

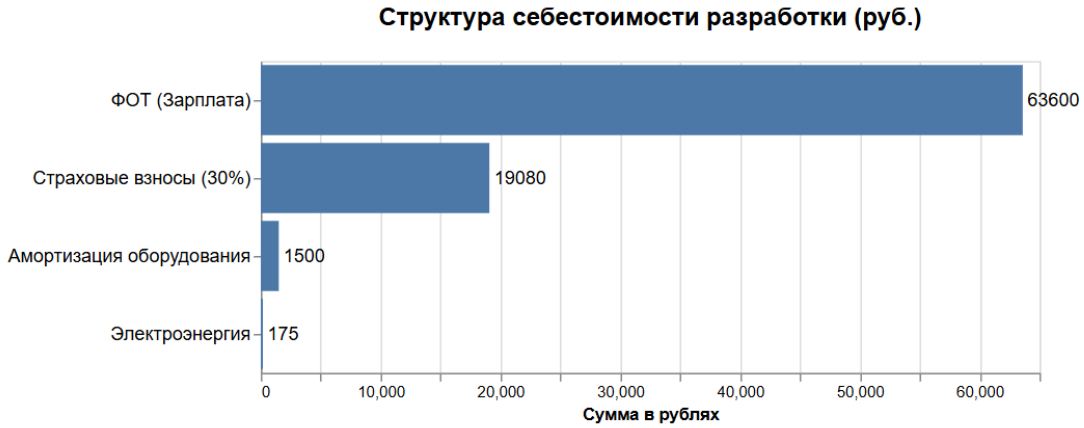

Рисунок 13 – Диаграмма структуры себестоимости разработки

3.2 Расчет эффективности эксплуатации

Комплексная оценка экономической эффективности внедрения разработанной информационной системы не ограничивается простым сопоставлением затрат на разработку и внедрение. Она требует глубокого анализа операционной деятельности и качественных изменений в рабочем процессе, которые привносит автоматизация интеллектуального труда. Фундаментальным аспектом здесь выступает изменение самой парадигмы работы автора. М.В. Безпятый в своих трудах справедливо утверждает: «Автоматизация направлена на устранение утомительных, повторяющихся задач, сведение к минимуму ручного вмешательства и повышение скорости и надежности процессов» [4, с.2]. В контексте писательской деятельности разработанная платформа берет на себя наиболее трудоемкие и рутинные функции — корректуру текста, поиск синонимов и стилистическую адаптацию. Это позволяет автору перераспределить свой временной ресурс с технической обработки текста на творческую деятельность, что опосредованно повышает производительность труда и качество конечного продукта.

Проведенный сравнительный анализ эксплуатационных расходов демонстрирует существенную экономическую выгоду перехода от использования сторонних облачных сервисов к собственной локальной инфраструктуре. На текущий момент использование популярных коммерческих

платформ генерации текста, функционирующих по модели SaaS (Software as a Service), сопряжено со значительными регулярными затратами. Стоимость профессиональной подписки на подобные сервисы варьируется в диапазоне от 20 до 30 долларов США в месяц, что в пересчете на российскую валюту составляет порядка 30 000 рублей ежегодных операционных расходов для одного пользователя. При этом данная модель расходов является постоянной и имеет тенденцию к росту ввиду инфляционных процессов и изменения курсов валют. В противовес этому, эксплуатация разработанного локального прототипа характеризуется практически нулевой маржинальной стоимостью использования. Единственной переменной статьей расходов является оплата электроэнергии, потребляемой вычислительной станцией во время активных вычислений нейросети. Расчеты показывают, что даже при интенсивном режиме работы расходы на электроэнергию не превышают 600 рублей в год. Сопоставление этих величин позволяет констатировать, что отказ от облачных подписок обеспечивает прямую экономию бюджета в размере более 29 000 рублей ежегодно, что делает проект высокорентабельным для индивидуального использования.Сравнительный анализ затрат на эксплуатацию облачных сервисов и разработанной локальной системы приведен на рисунке 14.

Рисунок 14 – Сравнительная диаграмма годовых эксплуатационных расходов

Помимо прямых финансовых показателей, необходимо учитывать экономические риски, связанные с информационной безопасностью.

Использование публичных облачных сервисов неизбежно влечет за собой передачу уникальных авторских материалов на серверы третьих лиц, что создает угрозу утечки интеллектуальной собственности. И.Д. Крылов и коллеги в своем исследовании предупреждают, что уязвимости веб-приложений и каналов передачи данных «могут привести к значительному финансовому и репутационному ущербу для организаций» [13, с.1]. В случае утечки черновика неизданной книги убытки автора могут многократно превысить стоимость любой подписки. Разработанная система полностью элиминирует этот класс рисков, так как обработка данных происходит в изолированном локальном контуре без выхода в глобальную сеть, что обеспечивает абсолютную конфиденциальность и защиту авторских прав, являясь важным конкурентным преимуществом продукта.

При этом важно подчеркнуть, что внедрение технологий искусственного интеллекта не подразумевает полного исключения человека из производственного процесса. Концепция системы строится на принципе синергии естественного и искусственного интеллекта. О.И. Пугач и Н.В. Старцева, рассматривая применение генеративных моделей, отмечают: «Полученные методические материалы подлежат обязательному контролю со стороны преподавателя» [24, с.16]. Этот методологический принцип был положен в основу бизнес-логики нашей платформы: нейросеть выступает в роли ассистента, генерирующего варианты и предложения, но финальное решение о принятии правок всегда остается за автором. Такой подход гарантирует сохранение уникального авторского стиля и высокого художественного качества текста, что невозможно при полной автоматизации, и тем самым повышает рыночную ценность создаваемых произведений.

В рамках стратегии долгосрочного развития проекта был разработан детальный инвестиционный план модернизации аппаратной части, необходимый для перехода от стадии прототипа (MVP) к промышленной эксплуатации. Текущее оборудование накладывает объективные технические ограничения на размер используемых нейросетевых моделей и длину

контекстного окна. Для качественного скачка в возможностях системы, включая внедрение технологии RAG (Retrieval Augmented Generation) для анализа полных текстов книг, планируется приобретение специализированного сервера инференса. Целевая конфигурация предполагает установку графического ускорителя профессионального или топового потребительского класса с объемом видеопамяти 24 ГБ. Такой объем VRAM критически важен, так как он позволяет загружать в память современные модели с миллиардами параметров (например, уровня 70B с квантованием), которые по качеству генерации не уступают лидирующим облачным решениям.

Ориентировочные капитальные затраты на такую модернизацию оцениваются рынком в диапазоне от 150 000 до 200 000 рублей. Несмотря на высокую стоимость первоначальных вложений, расчет возврата инвестиций (ROI) демонстрирует целесообразность такого шага, особенно при масштабировании системы на малые рабочие группы. При сценарии использования платформы небольшой редакцией или коллективом соавторов из пяти человек, совокупные ежегодные затраты на облачные подписки составили бы около 150 000 рублей. Таким образом, приобретение собственного высокопроизводительного сервера полностью окупается уже через 12–16 месяцев эксплуатации. После прохождения точки окупаемости система начинает генерировать чистую прибыль за счет экономии на операционных расходах, обеспечивая при этом организации технологический суверенитет и независимость от политики ценообразования внешних поставщиков услуг.

Заключение

В ходе выполнения выпускной квалификационной работы была достигнута поставленная цель — разработан и реализован функциональный прототип информационной системы «Платформа с ИИ-помощником для написания книг».

В процессе исследования и разработки были решены следующие задачи:

1. Проведен анализ предметной области, который показал востребованность инструментов, работающих по парадигме «ИИ как ассистент», а не генератор контента.
2. Спроектирована микросервисная архитектура, включающая сервис аутентификации (auth-service) и основной сервис приложения (app), взаимодействующие через защищенный API.
3. Реализована безопасная среда с использованием Docker-контейнеризации, базы данных PostgreSQL и обратного прокси-сервера Nginx.
4. Осуществлена интеграция с локальной языковой моделью через инструмент Ollama. Выбранная модель gemma3:1b доказала свою эффективность на оборудовании среднего ценового сегмента (RTX 3050), обеспечивая высокую скорость отклика и конфиденциальность пользовательских данных.
5. Разработан клиентский веб-интерфейс, реализующий функции управления проектами, текстового редактора и контекстного меню ИИ-инструментов.

Ключевые особенности разработанной системы:

— Автономность: Полная независимость от внешних платных API и интернет-соединения при работе с текстом.

— Конфиденциальность: Тексты произведений не покидают локальный контур пользователя.

— Гибкость: Использование Docker позволяет развернуть систему одной командой на любой машине с поддержкой виртуализации.

Ограничения и направления для развития: На текущем этапе система является прототипом (MVP). К ограничениям можно отнести необходимость наличия видеокарты с поддержкой CUDA для быстрой работы, а также использование утилиты ngrok для организации удаленного доступа, что не подходит для промышленной эксплуатации (Production). Дальнейшее развитие проекта предполагает:

— Реализацию полноценного деплоя на VDS-сервере с использованием GPU-хостинга.

— Внедрение механизма RAG (RetrievalAugmentedGeneration) для учета контекста всей книги, а не только выделенного фрагмента.

— Миграцию клиентской части на фреймворк Vue.js или React для расширения функциональности интерфейса.

Экономический расчет показал, что внедрение разработанной системы позволяет сэкономить порядка 29 600 рублей в год на подписках, что делает проект привлекательным для индивидуальных авторов и малых издательских групп.

Список литературы

1. Альфара, А. Ю. А., Королев, Д. В., Зайцев, К. С., Дунаев, М. Е. Разработка системы мониторинга для серверного приложения // International Journal of Open Information Technologies. – 2023. – № 8. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/razrabotka-sistemy-monitoringa-dlya-servernogo-prilozheniya (дата обращения: 22.12.2025).

2. Андреев, И. А., Иванов, А. Ю., Петров, Р. В. Исследование влияния различных параметров на вычислительную мощность процессора // Вестник НовГУ. – 2024. – № 3 (137). [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/issledovanie-vliyaniya-razlichnyh-parametrov-na-vychislitelnuyu-moschnost-protsessora (дата обращения: 22.12.2025).

3. Андрончик, Г. В. Оптимизация бизнес-процессов с помощью LLM // Universum: технические науки. – 2025. – № 5 (134). [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/optimizatsiya-biznes-protsessov-s-pomoschyu-llm (дата обращения: 22.12.2025).

4. Безпятый, М. В. Автоматизация и оптимизация процессов разработки и развертывания в DevOps: применение современных методов и инструментов // Инновации и инвестиции. – 2023. – № 7. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/avtomatizatsiya-i-optimizatsiya-protsessov-razrabotki-i-razvertyvaniya-v-devops-primenenie-sovremennyh-metodov-i-instrumentov (дата обращения: 22.12.2025).

5. Брешенков, А. В. Методика проектирования реляционных баз данных // Инженерный журнал: наука и инновации. – 2013. – № 11 (23). [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/metodika-proektirovaniya-relyatsionnyh-baz-dannyh (дата обращения: 22.12.2025).

6. Володин, А. Ю. Цифровая герменевтика исторического источника: формализация как толкование // Вестн. Перм. ун-та. Сер. История. – 2025. – № 2 (69). – URL: https://cyberleninka.ru/article/n/tsifrovaya-germenevtika-

istoricheskogo-istochnika-formalizatsiya-kak-tolkovanie (дата обращения: 22.12.2025).

7. Ехлаков, О. В. Системный подход к безопасности пользовательских файлов: от первичной валидации до изоляции в Docker // International Journal of Open Information Technologies. – 2025. – № 9. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/sistemnyy-podhod-k-bezopasnosti-polzovatelskih-faylov-ot-pervichnoy-validatsii-do-izolyatsii-v-docker (дата обращения: 22.12.2025).

8. Жигулев, П. А. Agile-методология управления командами (на примере разработчиков в игровой индустрии) // Экономика и бизнес: теория и практика. – 2023. – № 12-2 (106). [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/agile-metodologiya-upravleniya-komandami-na-primere-razrabotchikov-v-igrovoy-industrii (дата обращения: 22.12.2025).

9. Земсков, М. А. Использование модульных монолитов для разработки масштабируемых web-приложений // Universum: технические науки. – 2024. – № 10 (127). [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/ispolzovanie-modulnyh-monolitov-dlya-razrabotki-masshtabiruemyh-web-prilozheniy (дата обращения: 22.12.2025).

10. Зимина, К. И., Лапонина, О. Р. Механизмы межсервисной аутентификации в приложениях с микросервисной архитектурой // International Journal of Open Information Technologies. – 2023. – № 5. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/mehanizmy-mezhservisnoy-autentifikatsii-v-prilozheniyah-s-mikroservisnoy-arhitekturoy (дата обращения: 22.12.2025).

11. Карнаухов, К. В. Преимущества контейнеризации и сервисно-ориентированной архитектуры при разработке и развертывании веб-приложения // Экономика и социум. – 2018. – № 6 (49). [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/preimuschestva-konteynerizatsii-i-servisno-orientirovannoy-ar-hitektury-pri-razrabotke-i-razvertyvanii-veb-prilozheniya (дата обращения: 22.12.2025).

12. Кравченко, Д. А. Микросервисная архитектура // Интерактивная наука. – 2022. – № 4 (69). [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/mikroservisnaya-arhitektura (дата обращения: 22.12.2025).

13. Крылов, И. Д., Кича, И. В., Яковлев, Д. П. и др. Моделирование рисков информационной безопасности типа обхода Web Application Firewall // Известия ТулГУ. Технические науки. – 2023. – № 4. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/modelirovanie-riskov-informatsionnoy-bezopasnosti-tipa-obhoda-web-application-firewall (дата обращения: 22.12.2025).

14. Куликов, К. А. Стратегии декомпозиции монолита и выявление границ микросервисов // Парадигма. – 2025. – № 11-4. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/strategii-dekompozitsii-monolita-i-vyyavlenie-granits-mikroservisov (дата обращения: 22.12.2025).

15. Лебединский, Ю. Е., Намиот, Д. Е. Состязательное тестирование больших языковых моделей // International Journal of Open Information Technologies. – 2025. – № 11. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/sostyazatelnoe-testirovanie-bolshih-yazykovyh-modeley (дата обращения: 22.12.2025).

16. Лиманова, Н. И., Анашкин, А. С. Основные механизмы безопасности в Linux // Бюллетень науки и практики. – 2024. – № 2. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/osnovnye-mehanizmy-bezopasnosti-v-linux (дата обращения: 22.12.2025).

17. Лисин, В. А., Сидорова, Е. А. Модель конструктора информационных порталов на основе онтологий предметной области и ресурсов // Информационные и математические технологии в науке и управлении. – 2025. – № 2 (38). [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/model-konstruktora-informatsionnyh-portalov-na-osnove-ontologiy-predmetnoy-oblasti-i-resursov (дата обращения: 22.12.2025).

18. Ломов, П. А. Использование онтологий для контекстуализации запросов к большим языковым моделям // Онтология проектирования. – 2025. – № 2 (56). [Электронный ресурс]. URL: https://cyberleninka.

19. ruarticle/n/ispolzovanie-ontologiy-dlya-kontekstualizatsii-zaprosov-k-bolshim-yazykovym-modelyam (дата обращения: 22.12.2025).

20. Ляшов, Е. И. Виды аутентификации в современных веб-приложениях // Universum: технические науки. – 2024. – № 12 (129). [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/vidy-autentifikatsii-v-sovremennyh-veb-prilozheniyah (дата обращения: 22.12.2025).

21. Мельник, А. А. Основные принципы построения и проектирования интерфейса API // Вестник науки. – 2025. – № 10 (91). [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/osnovnye-printsipy-postroeniya-i-proektirovaniya-interfeysa-api (дата обращения: 22.12.2025).

22. Намиот, Д. Е., Ильюшин, Е. А. ОкибербезопасностиИИ-агентов // International Journal of Open Information Technologies. – 2025. – № 9. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/o-kiberbezopasnosti-ii-agentov (датаобращения: 22.12.2025).

23. Олейник, А. Г. и др. Использование RAG-технологии для создания интеллектуальной информационной системы поддержки исследовательского поиска // Труды Кольского научного центра РАН. Серия: Технические науки. – 2024. – № 3. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/ispolzovanie-rag-tehnologii-dlya-sozdaniya-intellektualnoy-informatsionnoy-sistemy-podderzhki-issledovatelskogo-poiska (дата обращения: 22.12.2025).

24. Орехов, Б. В. Текст и знание в аспекте больших языковых моделей // Историческая информатика. – 2023. – № 4 (46). [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/tekst-i-znanie-v-aspekte-bolshih-yazykovyh-modeley (дата обращения: 22.12.2025).

25. Пугач, О. И., Старцева, Н. В. Применение генеративного искусственного интеллекта в методической работе преподавателя вуза:

разработка контрольно-измерительных материалов // Russian Journal of Education and Psychology. – 2024. – № 5. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/primenenie-generativnogo-iskusstvennogo-intellekta-v-met-odicheskoy-rabote-prepodavatelya-vuza-razrabotka-kontrolno-izmeritelnyh (дата обращения: 22.12.2025).

26. Филисов, Д. А. Архитектура высоконагруженных приложений // Universum: технические науки. – 2023. – № 10-1 (115). [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/arhitektura-vysokonagruzhennyh-prilozheniy (дата обращения: 22.12.2025).

27. Францев, Д. Н., Новоселова, О. В. Сравнение языков программирования для выполнения многопоточных задач // Вестник магистратуры. – 2025. – № 3-2 (162). [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/sravnenie-yazykov-programmirovaniya-dlya-vypolneniya-mnogopotochnyh-zadach (дата обращения: 22.12.2025).

28. Черепнин, А. Ю. Интерпретация P-tuning в больших языковых моделях // Парадигма. – 2025. – № 10-1. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/interpretatsiya-p-tuning-v-bolshih-yazykovyh-modelyah (дата обращения: 22.12.2025).

29. Чиганов, Д. Р. Docker: ключ к контейнеризации и масштабируемости // Вестник науки. – 2023. – № 7 (64). [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/docker-klyuch-k-konteynerizatsii-i-masshtabiruemosti (дата обращения: 22.12.2025).

30. Шагулыев, Ш., Мередова, О., Чарыева, А. Языки программирования // Инновационная наука. – 2024. – № 9-2. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/yazyki-programmirovaniya (дата обращения: 22.12.2025).

31. Шинкарев, А. А. и др. Эволюция использования и применения машинного обучения и искусственного интеллекта в разработке корпоративных информационных систем и в системах поддержки принятия решений // Вестник ЮУрГУ. Серия: Компьютерные технологии, управление, радиоэлектроника. –

2025. – № 4. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/evolyutsiya-ispolzovaniya-i-primeneniya-mashinnogo-obucheniya-i-iskusstvennogo-intellekta-v-razrabotke-korporativnyh (дата обращения: 22.12.2025).