Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
федерального государственного бюджетного образовательного учреждения высшего образования
**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»**
(МИ ВлГУ)

Факультет ИТР

Кафедра ИС

УТВЕРЖДАЮ

Заведующий кафедрой

_____ Д.Е. Андрианов
(подпись)

«_____» _____ 2025 г.

БАКАЛАВРСКАЯ РАБОТА

Тема Разработка интерактивного симулятора экономических моделей

МИВУ.09.03.02-04.000 БР

Руководитель

Комкова С.В
(фамилия, инициалы)

_____
(подпись) (дата)

Студент ИС-121
(группа)

Есин Д.И.
(фамилия, инициалы)

_____
(подпись) (дата)

Муром 2025

БЛАНК ЗАДАНИЯ

Бакалаврская работа посвящена разработке интерактивного симулятора экономических моделей, предназначенного для образовательных целей. В проекте использованы технологии JavaFX для создания клиентского приложения и Spring Boot для реализации серверной части. Основное внимание уделено автоматизации процессов моделирования экономических сценариев, визуализации результатов и интеграции искусственного интеллекта для анализа данных. Разработанный симулятор предоставляет студентам и преподавателям удобный инструмент для изучения экономических теорий, проведения экспериментов и формирования отчётов, что способствует повышению качества обучения и углублению понимания экономических процессов.

Табл. 1. Ил. 30. Библ. 16.

The bachelor's thesis focuses on the development of an interactive economic models simulator designed for educational purposes. The project utilizes JavaFX for the client application and Spring Boot for the server implementation. The main emphasis is on automating economic scenario modeling, visualizing results, and integrating artificial intelligence for data analysis. The developed simulator provides students and educators with a user-friendly tool for studying economic theories, conducting experiments, and generating reports, thereby enhancing the quality of education and deepening the understanding of economic processes.

Tabl. 1. Fig. 30. Bibl. 16.

Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
федерального государственного бюджетного образовательного учреждения высшего образования
**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»**
(МИ ВлГУ)

Факультет ИТР

Кафедра ИС

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Тема: Разработка интерактивного симулятора экономических моделей

МИВУ.09.03.02-04.000 ПЗ

Муром 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современное образование в области экономики требует от студентов и начинающих специалистов глубокого понимания сложных экономических процессов и моделей. Однако изучение теоретических основ без практического применения часто оказывается недостаточным для формирования у обучающихся устойчивых навыков анализа и принятия обоснованных решений. В связи с этим важной задачей является создание программного обеспечения, способного наглядно демонстрировать и объяснять экономические модели и процессы, облегчая понимание и закрепление учебного материала.

Целью дипломной работы является разработка интерактивного программного комплекса – симулятора экономических моделей, предназначенного для образовательных целей и ознакомления студентов и начинающих экономистов с принципами работы фундаментальных экономических теорий и инструментов анализа данных. Симулятор будет предоставлять пользователям возможность не только изучать и визуализировать основные экономические модели, но и самостоятельно проводить эксперименты и наблюдать влияние различных параметров на поведение системы, закрепляя таким образом полученные теоретические знания на практике.

Использование современных информационных технологий в образовании позволяет значительно повысить эффективность обучения. Интерактивные симуляторы, такие как разрабатываемый в рамках данной дипломной работы программный комплекс, способствуют развитию аналитического мышления, формированию самостоятельных исследовательских навыков и способствуют вовлечению студентов в активную учебную деятельность.

Разработка данного программного обеспечения направлена на повышение интереса студентов к изучению экономики, улучшение усвоения теоретических знаний и формирование устойчивых навыков применения экономических моделей в реальных ситуациях. Симулятор призван сделать процесс обучения

интерактивным и наглядным, что позволит студентам более эффективно и с большим интересом осваивать учебный материал.

Для достижения поставленной цели были определены следующие задачи:

– проведение анализа существующих образовательных программ и симуляторов экономических процессов, выявление их преимуществ и недостатков;

– формулировка функциональных требований к разрабатываемому программному обеспечению, включая требования к пользовательскому интерфейсу и методическим материалам;

– выбор и обоснование технологий для разработки программного комплекса;

– проектирование архитектуры клиент-серверного приложения, обеспечивающей масштабируемость и удобство использования;

– разработка структуры базы данных, проектирование и реализация клиентского и серверного компонентов приложения;

– интеграция системы искусственного интеллекта для обеспечения дополнительной аналитической поддержки и объяснения результатов моделирования;

– тестирование и отладка программного обеспечения с целью выявления и устранения возможных ошибок и недостатков.

Разработанный программный комплекс может быть использован в учебном процессе для повышения качества преподавания экономических дисциплин, формирования практических навыков у обучающихся и повышения их заинтересованности в изучении экономики как науки.

1 Анализ технического задания

В соответствии с представленным техническим заданием требуется разработать интерактивный симулятор экономических моделей, ориентированный на использование в учебном процессе и способствующий повышению качества подготовки студентов и начинающих экономистов. Основная задача проектируемой системы заключается в наглядной демонстрации базовых экономических теорий, предоставлении пользователям интерактивных инструментов для экспериментов с различными параметрами моделей и визуализации полученных результатов в удобной форме. Программный комплекс должен обеспечивать удобный доступ к образовательным материалам, простоту создания и настройки экономических сценариев, а также возможность анализа результатов моделирования с использованием интегрированных средств искусственного интеллекта для лучшего понимания изучаемых процессов.

1.1 Обзор программ аналогов

При проектировании образовательного симулятора экономических моделей важно учитывать существующие решения, выполняющие схожие функции математического моделирования и визуализации данных. Анализ аналогов позволяет выявить их сильные и слабые стороны и сформировать требования к разрабатываемой системе. В качестве примеров были выбраны три распространённых инструмента: Eviews, Gretl, Microsoft Excel.

EViews — мощный пакет для эконометрического и статистического анализа данных.

Плюсы: широкий набор встроенных методов оценки моделей, удобный синтаксис команд, экспорт результатов в разные форматы.

Минусы: высокая стоимость лицензии, сложность освоения, отсутствие интерактивной визуализации учебных примеров и AI-поддержки.

Gretl — бесплатный инструмент эконометрического анализа с открытым исходным кодом.

Плюсы: доступность, поддержка большинства эконометрических тестов и моделей, скриптовый интерфейс.

Минусы: базовый, устаревший интерфейс; ограниченные возможности визуализации; нет функционала для анализа финансовой отчётности и интеграции AI.

Microsoft Excel (с надстройками) — универсальная табличная среда, часто используемая для построения простейших экономических моделей.

Плюсы: богатые возможности графиков и формул, лёгкий экспорт.

Минусы: отсутствие специализированных статистических и эконометрических модулей «из коробки», слабая поддержка крупномасштабных расчётов, нет встроенных инструментов AI-анализа.

Таблица 1 - Сравнительный анализ программ аналогов

Критерий	Eviews	Gretl	Microsoft Excel
Математическое моделирование	Да	Да	Да
Интерактивная визуализация	Нет	Нет	Да
Анализ финансовой документации	Нет	Нет	Да
Интеграция AI	Нет	Нет	Нет
Экспорт отчётов	Да	Да	Да

Анализ показал, что существующие инструменты хорошо справляются с классическими математическими задачами, но не обеспечивают интеграцию методов искусственного интеллекта для анализа финансовых документов и предоставления информации об экономической модели посредством LLM. Это обосновывает необходимость разработки специализированного симулятора, сочетающего гибкость математического моделирования, богатую визуализацию и AI-поддержку в одном учебном приложении.

1.2 Формирование требований к информационной системе

Проектируемая система будет предоставлять пользователю возможность выбирать экономические модели, редактировать их параметры и сохранять состояния моделей для каждого пользователя. Через удобный интерфейс можно будет задавать исходные предпосылки моделирования, запускать расчёты и сохранять все полученные результаты для последующего анализа.

После выполнения вычислений система будет отображать готовые графики и диаграммы — линейные графики, гистограммы, тепловые карты и другие настроенные виды визуализации. Пользователь сможет в любой момент переключаться между различными типами представлений и возвращаться к ранее сохранённым сессиям без повторного ввода параметров.

Важным элементом функционала станут внешние AI-модули, которые автоматически проанализируют загруженные финансовые отчёты и сопроводительную документацию. Система будет автоматически извлекать значения для параметров из приложенных документов, а так же давать пользователю полную справку по модели: объяснять графики, помогать интерпретировать результат, предлагать значения параметров.

Для обеспечения надёжности и производительности система будет использовать механизмы авторизации и регистрации, хранить пользовательскую сессию и сохранять в базу данных результаты предыдущих расчётов.

1.3 Выбор средств программной реализации

Для реализации серверной части информационной системы предъявляются требования к масштабируемости и удобству дальнейшего развития проекта, при этом клиентская часть должна быть выполнена в виде десктопного приложения с интуитивно понятным интерфейсом, подходящим для образовательных целей.

При выборе языка программирования для серверной части рассматривались такие распространённые решения, как Java и Python. Python изначально привлекал внимание своей простотой, скоростью разработки прототипов и обилием готовых решений. Вместе с тем, серверное приложение, работающее с интенсивными вычислениями, большим количеством параллельных запросов и сложными аналитическими алгоритмами, требует значительно большей производительности, стабильности и возможности удобной масштабируемости. Именно в этих аспектах Java значительно превосходит Python, особенно при использовании новейших функций JDK 21. В результате этих преимуществ Java стала оптимальным выбором для реализации серверной составляющей информационной системы.

Серверный фреймворк выбирался среди популярных платформ: Spring Framework и Jakarta EE. Jakarta EE предоставляет широкий набор корпоративных инструментов, однако он характеризуется высокой сложностью настройки и значительным порогом входа, что увеличивает временные и трудовые затраты при разработке и сопровождении проекта. Напротив, Spring Framework отличается более гибким и простым подходом к конфигурации, модульностью и возможностью удобной интеграции с широким спектром современных технологий и библиотек. Именно поэтому Spring Framework был выбран как основа для разработки серверной части.

Важной составляющей серверного приложения является эффективное и надёжное кеширование данных, позволяющее повысить быстродействие и снизить нагрузку на базу данных. Рассматривались два варианта реализации кеширования: Redis и Ehcache. Ehcache — локальный in-memory кеш, интегрируемый непосредственно в приложение, который характеризуется лёгкостью настройки и отсутствием необходимости дополнительной инфраструктуры. Тем не менее, в условиях интенсивной параллельной работы и необходимости распределённого хранения данных между несколькими инстансами приложения более актуальным решением является Redis.

В качестве системы управления базами данных рассматривались две популярные реляционные базы данных: MySQL и PostgreSQL. MySQL отличается простотой настройки, хорошей производительностью и широкой поддержкой в различных приложениях и сервисах. Однако PostgreSQL обладает расширенными функциональными возможностями, такими как поддержка сложных запросов, богатый набор аналитических функций, транзакционная целостность данных и, что особенно важно, поддержка работы с JSON-данными и их эффективной обработки. Эти возможности крайне важны в контексте проектируемой системы, которая активно использует сложные структуры данных для хранения и анализа экономических моделей. Ввиду перечисленных преимуществ PostgreSQL был выбран в качестве основной базы данных системы.

Для хранения загружаемых пользователем документов и сопутствующей информации была необходимость выбрать объектное хранилище. Рассматривались решения MinIO и Ceph. Ceph является мощным распределённым хранилищем, обладающим высокой надёжностью, производительностью и масштабируемостью, однако оно характеризуется значительной сложностью развёртывания, конфигурации и управления, что может быть избыточным для данного проекта. MinIO, напротив, представляет собой лёгкое, эффективное и простое в эксплуатации объектное хранилище. MinIO отличается простотой настройки и удобством поддержки, что делает его оптимальным выбором для проекта, где важны удобство эксплуатации и гибкость управления данными.

Одним из ключевых компонентов системы является модуль анализа текстовых данных и документов, реализованный с помощью большой языковой модели (LLM). В рамках этого требования рассматривались открытые модели, такие как Mistral 7B и LLaMA 2. LLaMA 2 широко используется и обладает хорошими показателями производительности и точности при решении широкого спектра задач обработки естественного языка. Однако модель Mistral 7B отличается большей эффективностью, скоростью работы и лучшей адаптацией к

задачам анализа и генерации рекомендаций в экономической и финансовой сферах, при этом обладая меньшими требованиями к вычислительным ресурсам и более быстрым временем отклика. Эти характеристики сделали Mistral 7B предпочтительным выбором, позволяющим реализовать точный и оперативный анализ документации и финансовой отчётности и помогать пользователям в изучении экономических процессов.

При выборе технологий для клиентского приложения рассматривались платформы Java и C#. C# обладает сильными инструментами для создания удобных и производительных десктопных приложений. Однако выбор Java обусловлен необходимостью использования единого стека технологий на сервере и клиенте, что существенно облегчает поддержку и дальнейшее развитие проекта. Для реализации клиентского интерфейса сравнивались технологии Swing и JavaFX. Swing является проверенным решением, однако он устарел и уступает JavaFX в возможностях создания современного и привлекательного интерфейса, особенно с точки зрения визуализации экономических моделей и результатов расчётов. JavaFX обеспечивает более качественную визуализацию, интерактивность и удобство использования, что крайне важно для образовательного применения разрабатываемой системы.

Для управления процессом сборки и зависимостями были рассмотрены Gradle и Maven. Maven отличается стандартизированным подходом и простотой настройки для типовых задач. Однако Gradle обеспечивает более гибкую конфигурацию, высокую скорость сборки за счёт инкрементального подхода и эффективного кеширования. В результате выбор был сделан в пользу Gradle.

Таким образом, выбранные технологии обеспечивают необходимый уровень производительности, надёжности и удобства, соответствуют стандартам разработки и делают информационную систему максимально удобной и эффективной как для пользователей, так и для разработчиков.

2 Проектирование системы

При проектировании информационной системы, предназначенной для автоматизации экономического моделирования и анализа финансовых данных, необходимо соблюдать ряд общих требований и принципов, обеспечивающих надёжность, эффективность и простоту последующего сопровождения системы.

Важнейшим требованием является соблюдение модульного принципа построения системы. Компоненты должны разрабатываться максимально независимо друг от друга, что обеспечит высокую масштабируемость и лёгкость внесения изменений. Каждый компонент должен иметь чётко определённые функции и способы взаимодействия с другими компонентами, которые документируются на этапе проектирования. Это необходимо для снижения рисков при дальнейшем развитии и доработке системы.

Кроме того, проектирование должно предусматривать интеграцию с внешними AI-сервисами для расширения возможностей анализа данных и повышения точности моделей. Взаимодействие с внешними системами должно быть реализовано с использованием стандартных интерфейсов и протоколов (например, REST API), что позволит легко изменять или добавлять интеграции по мере необходимости.

Кроме того, важно учитывать не только технические, но и организационные аспекты разработки системы. Гибкий подход к управлению проектом, своевременное тестирование помогут создать продукт, который не только соответствует текущим требованиям, но и способен адаптироваться к будущим изменениям.

Таким образом, соблюдение перечисленных требований обеспечит создание надёжной, эффективной и легко поддерживаемой информационной системы, полностью удовлетворяющей заявленным функциональным и техническим характеристикам.

2.1 Определение бизнес-процессов

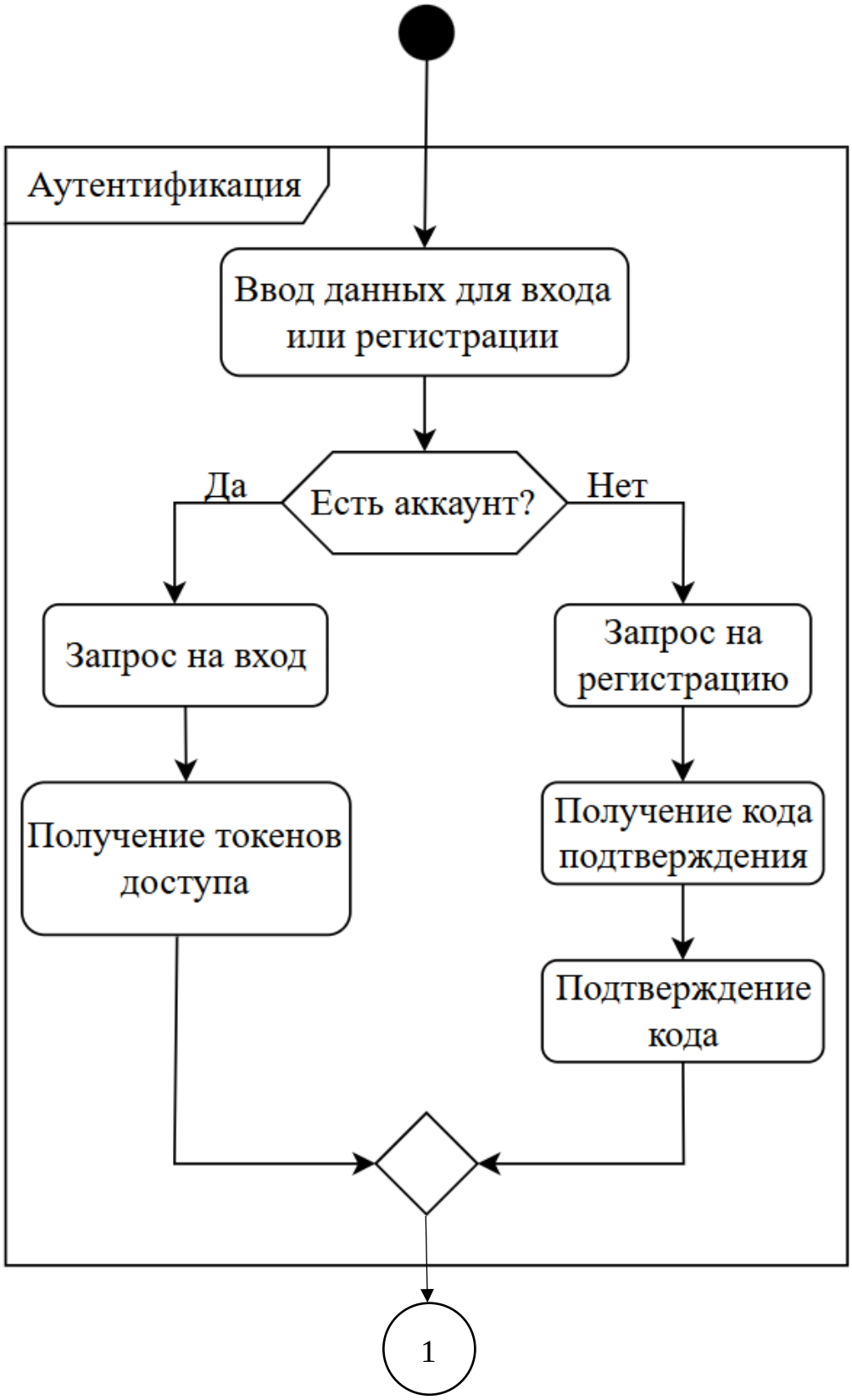

Рисунок 1 – Диаграмма бизнес-процессов

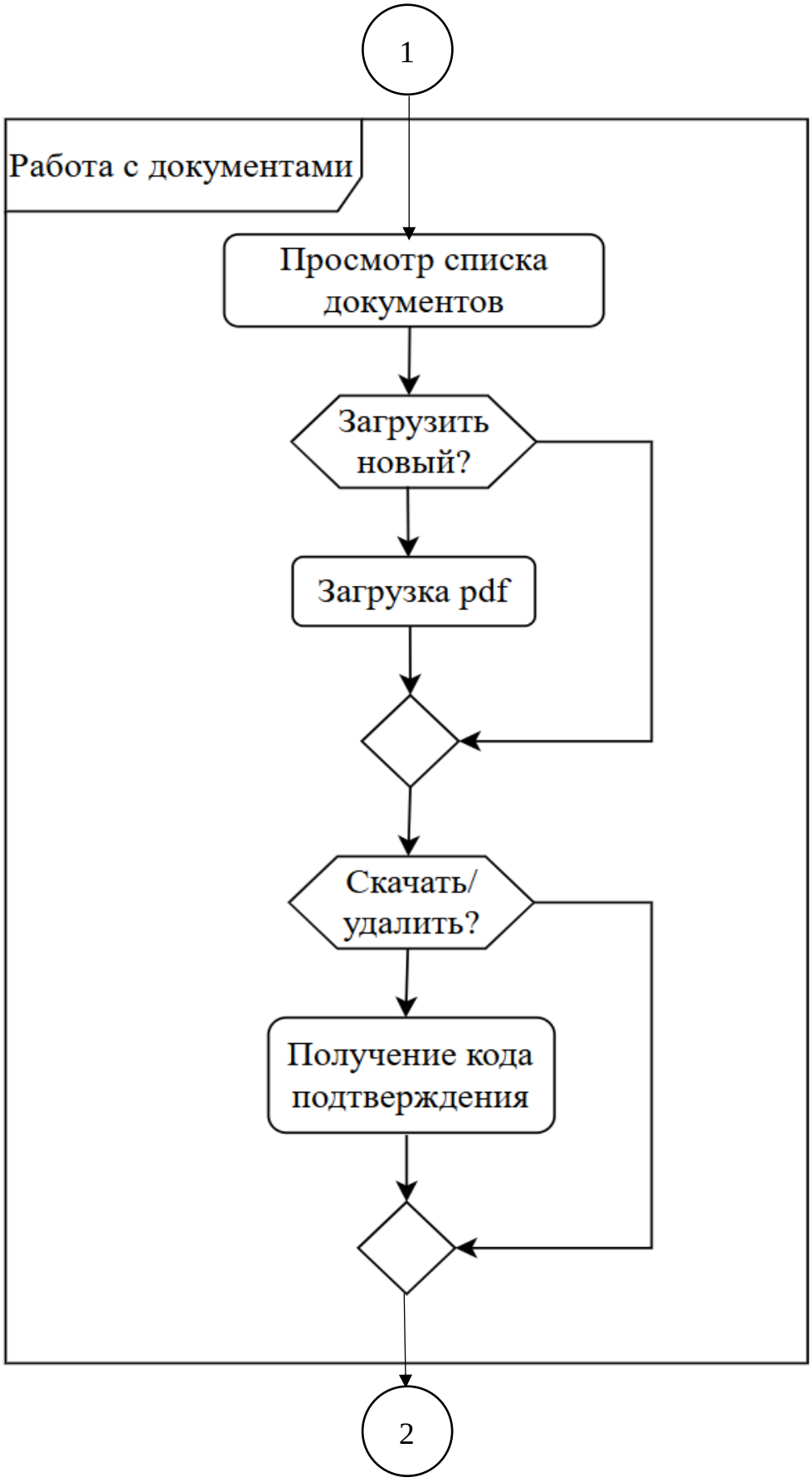

Рисунок 1 – Диаграмма бизнес-процессов (продолжение)

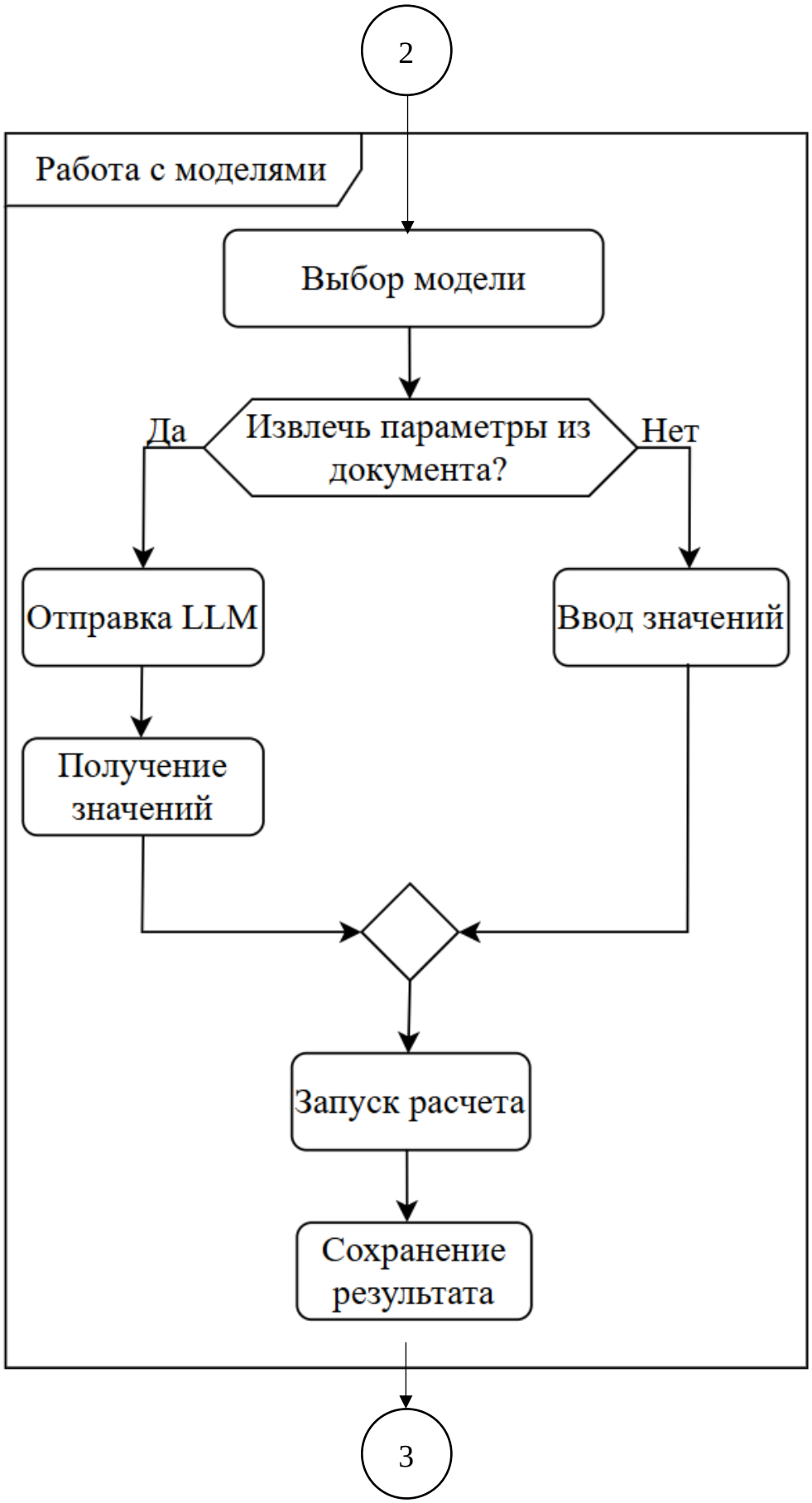

Рисунок 1 – Диаграмма бизнес-процессов (продолжение)

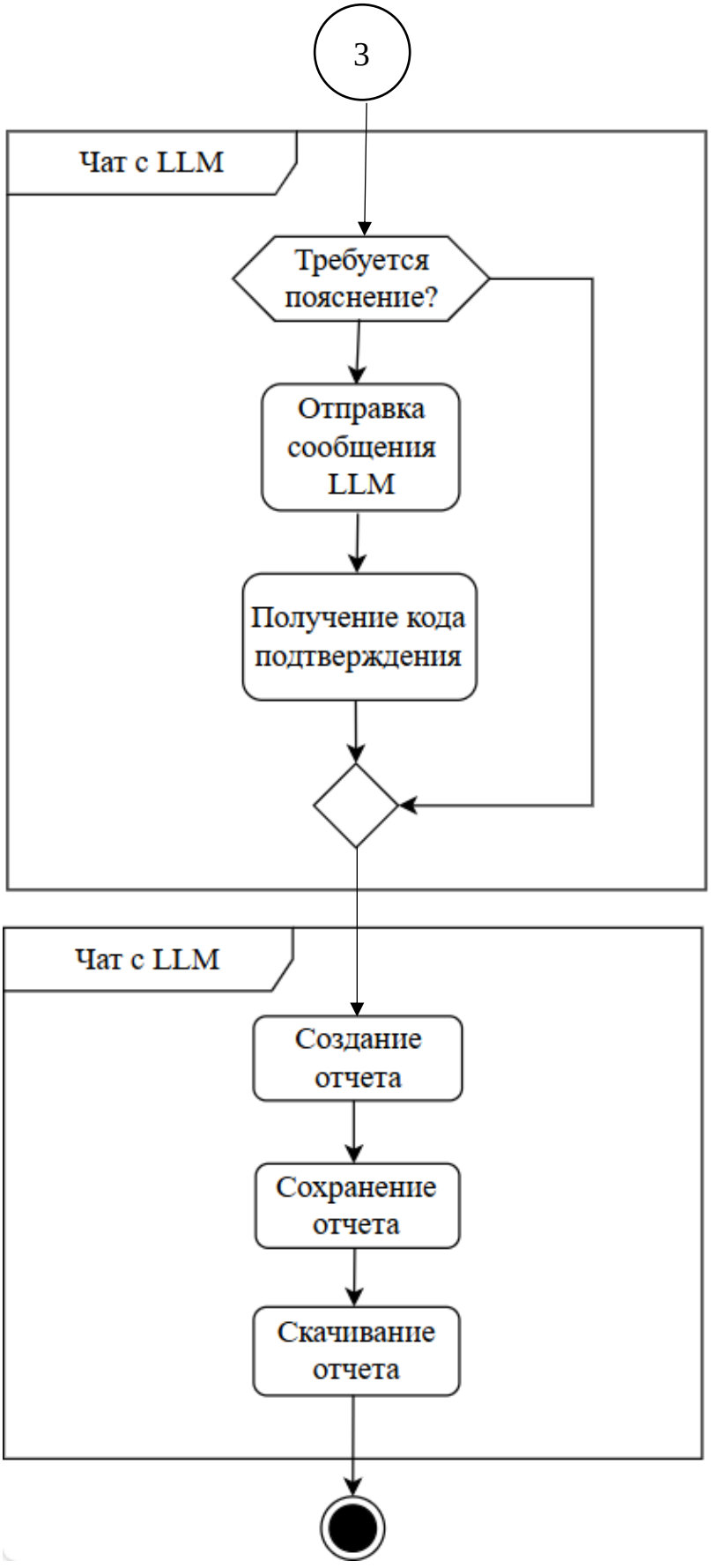

Рисунок 1 – Диаграмма бизнес-процессов (продолжение)

Диаграмма бизнес-процессов, изображенная на рисунке 1, отражает основные этапы взаимодействия пользователя с информационной системой и внутреннюю логику функционирования программного комплекса. Начальный этап работы начинается с запуска клиентского приложения, после чего пользователь проходит процедуру аутентификации. В зависимости от наличия учётной записи система предусматривает два сценария: либо вход с уже существующими данными, либо регистрация с подтверждением по электронной почте. Это обеспечивает необходимый уровень безопасности и позволяет идентифицировать пользователей на всех последующих этапах работы.

После успешной аутентификации пользователь получает доступ к блоку управления документами, где может просматривать список ранее загруженных файлов, загружать новые документы в систему или, при необходимости, скачивать и удалять уже имеющиеся. Документы хранятся на сервере с использованием объектного хранилища MinIO, что обеспечивает надёжность хранения и быстроту доступа к пользовательским данным.

Следующий важный этап работы с системой — взаимодействие с экономическими моделями. Пользователь выбирает нужную модель, получает параметры, которые можно скорректировать вручную или извлечь автоматически из загруженных документов с помощью интеграции LLM-сервиса. После задания всех необходимых параметров инициируется расчет модели на сервере, а результаты сохраняются для последующего анализа.

Для углубления понимания полученных данных и дополнительного сопровождения предусмотрен отдельный блок интеллектуального чата. В случае необходимости пользователь может отправить запрос на пояснение результатов или параметров модели искусственному интеллекту, получив развернутый ответ непосредственно в приложении.

Завершающий бизнес-процесс связан с формированием отчётов. Пользователь может сгенерировать отчёт, который будет содержать параметры модели, графическую визуализацию результатов и выдержки из чата с LLM.

Готовый отчёт в формате PDF автоматически сохраняется в хранилище. Он может быть скачан пользователем.

2.2 Определение сценариев использования

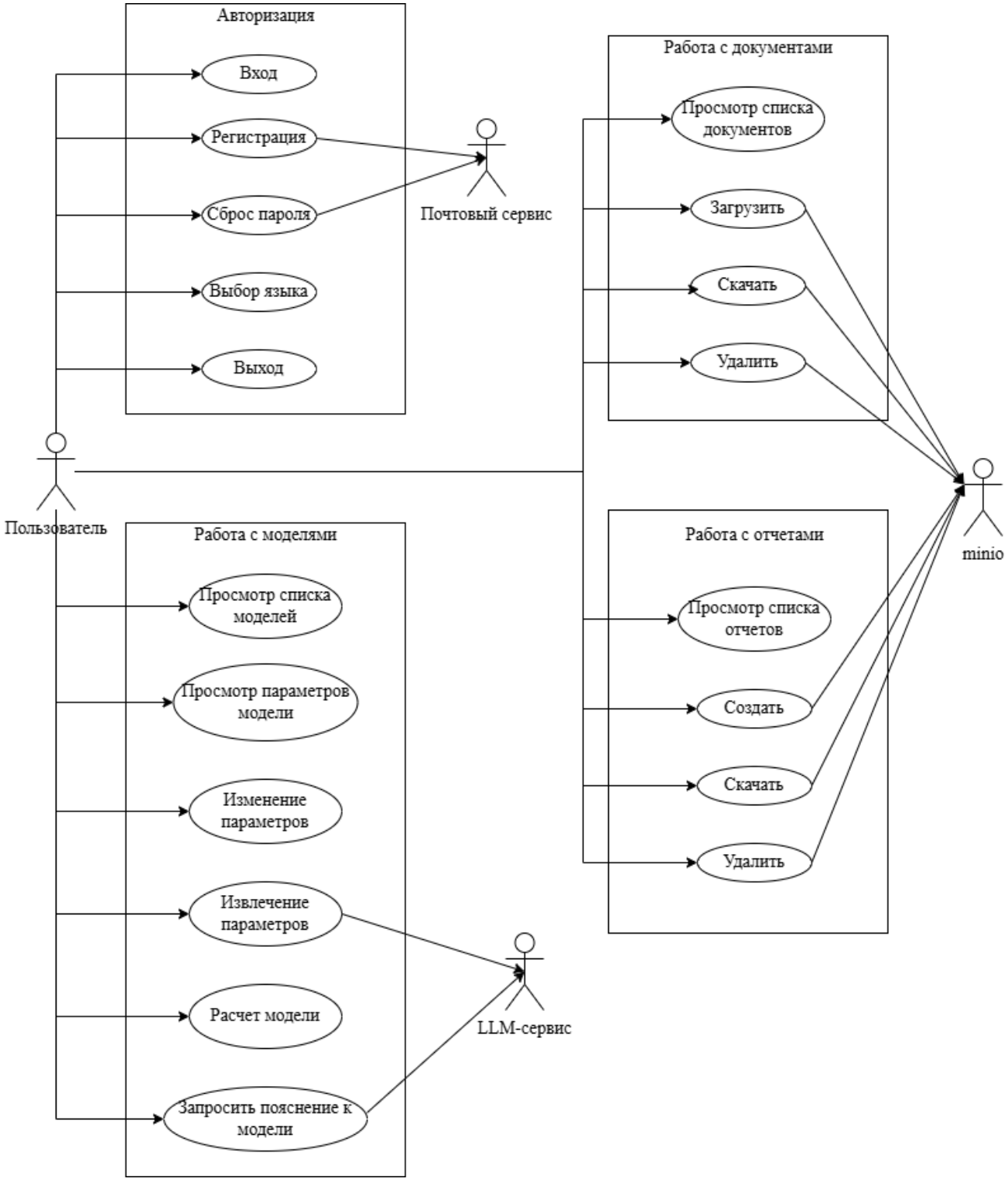

Рисунок 2 – Диаграмма сценариев использования

Диаграмма сценариев использования, изображенная на рисунке 2, отражает все ключевые пользовательские действия и взаимодействия с внешними сервисами в рамках работы с симулятором экономических моделей. В верхней части схемы расположены основные функциональные блоки: аутентификация, работа с документами и моделями. Каждый блок объединяет набор операций, доступных пользователю. Например, модуль аутентификации позволяет регистрироваться, входить в систему, сбрасывать пароль, подтверждать электронную почту, выходить из аккаунта и выбирать язык интерфейса. Все эти сценарии направлены на обеспечение безопасности и индивидуализации работы в системе.

Работа с документами включает возможности просмотра, загрузки, скачивания и удаления файлов, что обеспечивает гибкое управление пользовательскими данными. Параллельно пользователь может взаимодействовать с экономическими моделями: получать список доступных моделей, просматривать и изменять их параметры, запускать расчёты.

Во втором ряду схемы размещены интеллектуальные и отчётные функции. Благодаря интеграции LLM-сервиса реализованы такие сценарии, как извлечение параметров из документов и ведение диалога с искусственным интеллектом для получения пояснений или рекомендаций по моделям. Итоговые данные пользователь может оформить в виде отчёта, просмотреть список существующих отчетов, а также скачать или удалить их при необходимости.

Важной особенностью данной диаграммы является отображение связей с внешними сервисами — почтовым сервером, LLM и MinIO-хранилищем. Подтверждение e-mail и сброс пароля осуществляются с помощью почтового сервера, обмен документами выполняется через MinIO, а интеллектуальные функции реализуются за счёт подключения к LLM. Такая архитектура обеспечивает модульность системы и её готовность к расширению, а также прозрачность пользовательских сценариев и взаимодействия компонентов.

2.3 Проектирование архитектуры

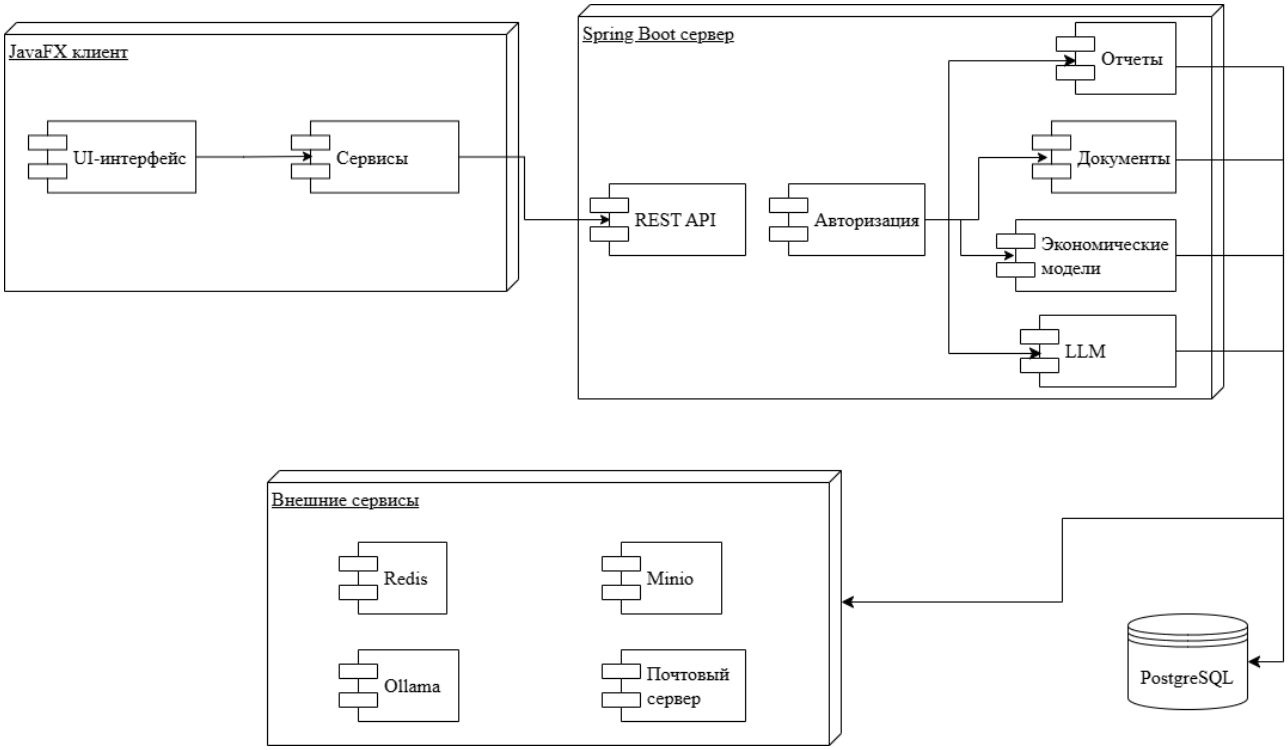

Рисунок 3 – Диаграмма компонентов

Архитектура информационной системы «Симулятор экономических моделей» построена по принципу разделения клиентской и серверной частей, что обеспечивает масштабируемость, безопасность и удобство поддержки проекта. Диаграмма компонентов, изображенная на рисунке 3, наглядно иллюстрирует взаимодействие между основными модулями, а также их интеграцию с внешними сервисами и инфраструктурными компонентами.

Клиентское приложение реализовано на платформе JavaFX и состоит из ряда функциональных модулей, каждый из которых отвечает за отдельный аспект работы пользователя: управление аутентификацией, экономическими моделями, документами, отчётами и диалогом с LLM-сервисом. Центральное место занимает пользовательский интерфейс, который координирует все взаимодействия между функциональными блоками клиента. Вся коммуникация с серверной частью осуществляется через REST API, что обеспечивает

унифицированный обмен данными и облегчает возможную интеграцию с другими внешними или мобильными клиентами в будущем.

Серверная часть системы реализована с использованием Spring Boot, что позволяет эффективно управлять бизнес-логикой, обработкой запросов и обеспечением безопасности. Каждый серверный модуль отвечает за отдельный сервис: аутентификация, работа с моделями, документами, отчётами, поддержка многоязычности и взаимодействие с LLM. Сервер оперирует централизованной базой данных PostgreSQL для хранения всех ключевых сущностей, а также интегрирован с системой кеширования Redis, что существенно ускоряет работу с наиболее востребованными и повторяющимися данными. Такой подход особенно актуален для обработки частых запросов, например, при работе с пользовательскими сессиями или кэшировании результатов вычислений экономических моделей.

Особое место в архитектуре занимает поддержка работы с внешними сервисами. Для хранения документов и отчётных файлов используется объектное хранилище MinIO, что обеспечивает надёжное и масштабируемое хранение данных вне основной базы. Для рассылки сервисных писем — подтверждения регистрации, сброса пароля и других уведомлений — сервер интегрирован с почтовым сервером. Интеллектуальные функции реализуются посредством отдельного LLM-сервиса, в качестве которого используется модель Mistral 7B. Это позволяет обрабатывать сложные текстовые задачи, такие как извлечение параметров из документов и генерация объяснений к расчётам, что расширяет образовательный и аналитический функционал системы.

Взаимодействие между всеми компонентами тщательно продумано для обеспечения целостности данных, высокой отказоустойчивости и безопасности пользовательской информации. Использование единой архитектурной схемы позволяет легко дорабатывать систему, расширять её функционал и интегрировать новые сервисы без необходимости переработки всей инфраструктуры. Такой подход формирует современную платформу,

отвечающую актуальным требованиям образовательных и аналитических приложений в сфере экономики.

Таким образом, представленная архитектура обеспечивает не только технологическую целостность и гибкость разработки, но и высокую адаптивность к будущим изменениям и масштабированию. Благодаря модульности и четкому разграничению зон ответственности каждый компонент системы может развиваться независимо, что значительно облегчает внедрение новых функций.

2.4 Проектирование взаимодействия модулей программы

Диаграмма последовательностей, приведенная в приложении А, отражает пошаговое взаимодействие пользователя с системой и распределение ответственности между основными сервисами.Пользователь инициирует процесс аутентификации, вводя свои данные для входа. Клиентское приложение формирует запрос к сервису аутентификации, получает в ответ токены доступа и открывает возможность дальнейшей работы. После входа пользователь может просмотреть доступные экономические модели: клиент отправляет соответствующий запрос сервису моделей и получает актуальный список.

Дальнейшая работа строится вокруг выбора конкретной модели. При открытии модели клиент взаимодействует с сервисом моделей для получения подробных данных и параметров, необходимых для дальнейшего анализа или расчётов. В случае необходимости пользователь может загрузить документ, например, финансовый отчёт или другую сопутствующую документацию. После загрузки документа через сервис документов, клиент может инициировать процедуру интеллектуального извлечения параметров, направляя запрос к внешнему LLM-сервису, который возвращает распознанные и структурированные данные.

Расчёт экономической модели реализуется отдельным сценарием. Пользователь инициирует вычисления, клиентское приложение формирует запрос, который передаётся в сервис моделей, а затем направляется модулю вычислений. После завершения расчётов результат возвращается пользователю через всю цепочку сервисов. Финальным этапом пользователь может сформировать отчёт на основе полученных данных и построенных графиков: клиент отправляет запрос сервису отчётов и после успешной генерации получает доступ к результату.

Таким образом, диаграмма последовательностей наглядно иллюстрирует четко выстроенный процесс взаимодействия пользователя с системой — от аутентификации до получения итоговых отчётов.

2.5 Проектирование структуры базы данных

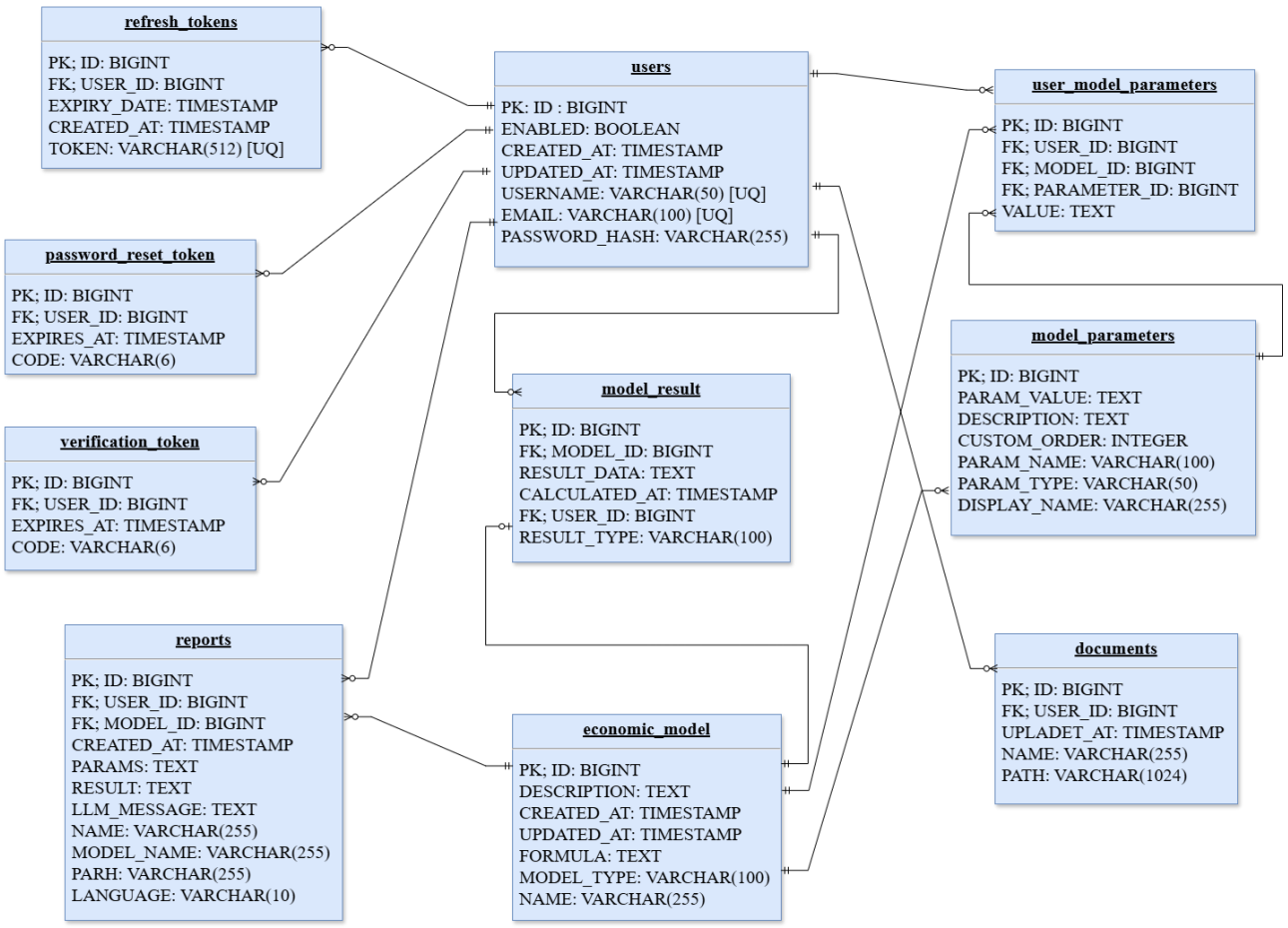

Рисунок 4 – ER-диаграмма

База данных информационной системы «Симулятор экономических моделей», чья структура отражена на рисунке 4, спроектирована таким образом, чтобы обеспечить целостность, безопасность и удобство хранения всех необходимых данных для поддержки бизнес-логики приложения. В основе лежит сущность пользователя, к которой жестко привязаны практически все основные элементы данных. Пользовательская таблица содержит как уникальные идентификаторы и аутентификационную информацию (логин, email, хэш пароля), так и служебные поля, отражающие состояние аккаунта и временные метки создания и обновления.

Документы, загружаемые пользователями, имеют собственную таблицу и содержат ссылки на пользователя, а также параметры файла, такие как имя, путь хранения и дату загрузки. Такая организация облегчает управление пользовательскими файлами, их поиск и удаление. Механизм работы с экономическими моделями представлен отдельной таблицей, где для каждой модели фиксируются основные характеристики: тип, название, описание, формула и временные метки. Каждый экземпляр модели может быть дополнен индивидуальным набором параметров, хранимым в отдельной таблице model_parameters, где каждому параметру присваивается уникальное имя, тип, значение и сопроводительное описание. Это позволяет гибко масштабировать функциональность без необходимости изменений в общей структуре базы.

Особое внимание уделено пользовательским параметрам, связанным одновременно с моделью, пользователем и конкретным параметром модели — это реализовано через связующую таблицу user_model_parameter, что обеспечивает персонализацию и независимость работы каждого пользователя с выбранными моделями.

Все результаты моделирования записываются в отдельную таблицу, где хранятся связи с пользователем и моделью, тип результата, исходные данные и дата расчета. Это позволяет сохранять полную историю расчетов, проводить аналитику и формировать отчёты.

Система отчетности строится на таблице reports, в которой хранятся метаданные отчётов, такие как название, язык, параметры, результаты моделирования и сообщения LLM, а также ссылки на соответствующего пользователя и модель. Данные об отчетах включают как итоговую информацию, так и путь к файлу, что облегчает хранение и загрузку из внешнего хранилища.

Безопасность работы с системой реализована с помощью трёх таблиц токенов: refresh_tokens для управления сессиями, verification_tokens для подтверждения email при регистрации и password_reset_tokens для восстановления доступа к аккаунту. Все эти таблицы имеют связь с пользователем и содержат уникальные коды, сроки действия и временные метки, что важно для предотвращения несанкционированного доступа и поддержания актуальности токенов.

2.6 Проектирование структуры классов сервера

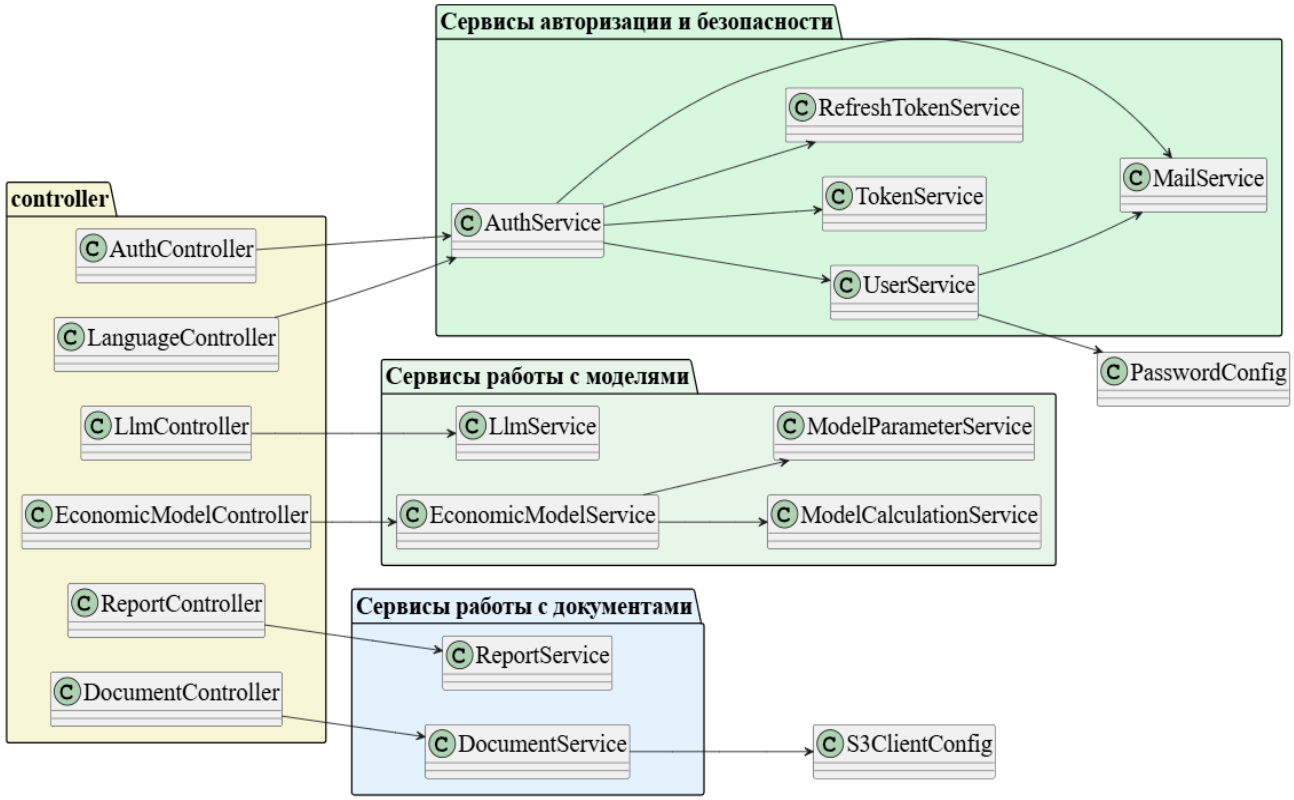

Рисунок 5 – Диаграмма классов сервера

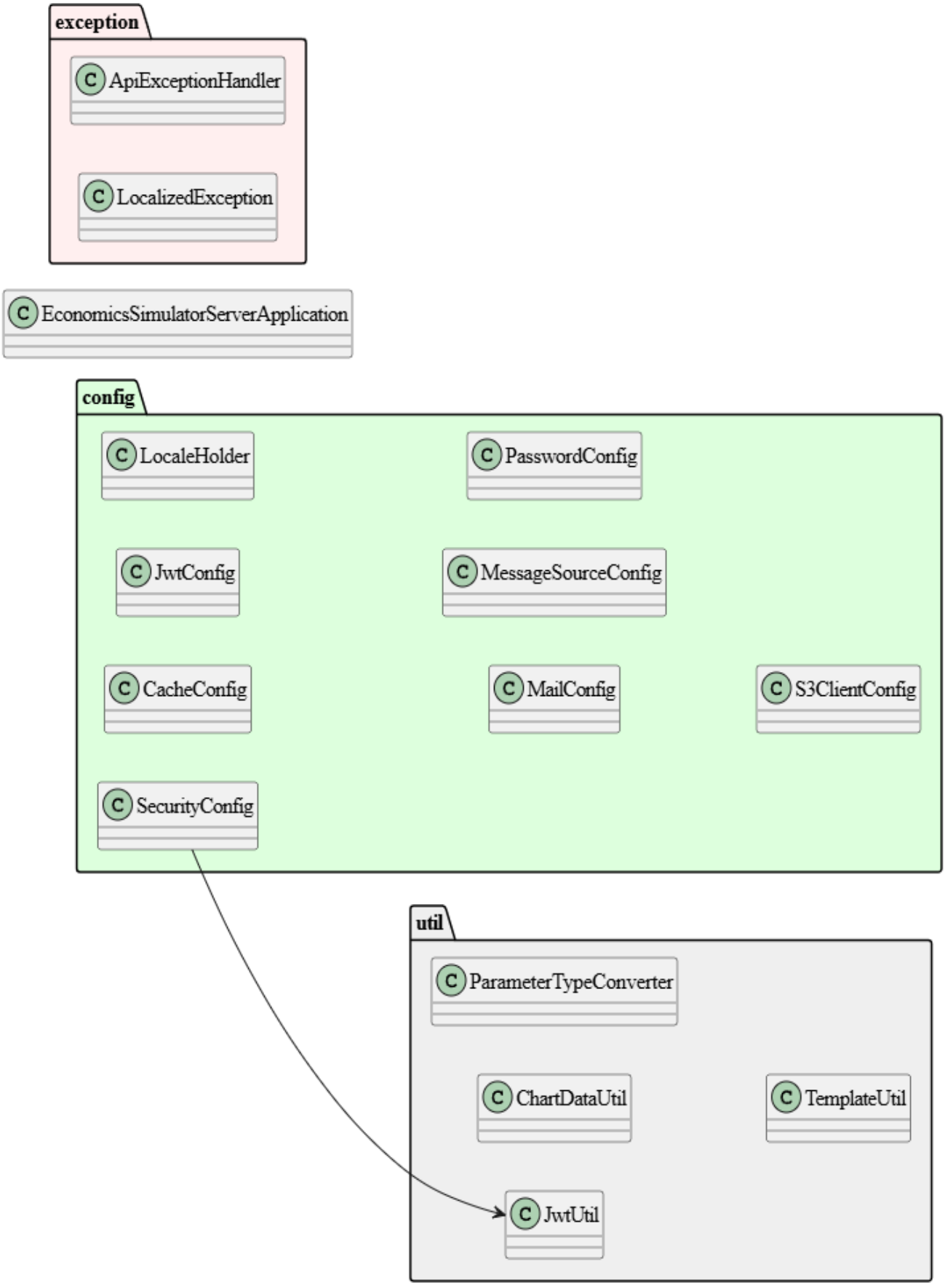

Рисунок 5 – Диаграмма классов сервера (продолжение)

Рисунок 5 – Диаграмма классов сервера (продолжение)

Архитектура серверной части, изображенная на рисунке 5, информационной системы реализована по модульному принципу с чётким разделением ответственности между слоями, что отражено на диаграмме классов. Центральное место занимают контроллеры, предоставляющие точки входа для клиентских запросов и обеспечивающие маршрутизацию вызовов к соответствующим сервисам. Контроллеры инкапсулируют взаимодействие с бизнес-логикой, изолируя внешний интерфейс от внутренних реализаций, что способствует безопасности и масштабируемости системы.

Слой сервисов играет ключевую роль в обработке бизнес-логики приложения. Каждый сервис отвечает за отдельный аспект работы системы: управление пользователями и аутентификацией, обработку документов, работу с экономическими моделями, взаимодействие с LLM-сервисом, формирование и хранение отчётов. Это позволяет гибко управлять изменениями и развивать отдельные направления функционала без влияния на другие части приложения. Дополнительно сервисы взаимодействуют с вспомогательными компонентами — такими как почтовый сервис для отправки уведомлений, конфигурационные классы для подключения к внешним ресурсам, сервис кэширования и другие.

Данные, которыми оперирует система, представлены отдельными агрегирующими классами и интерфейсами — DTOs, сущностями, репозиториями и решателями экономических моделей (Solvers). DTO (Data Transfer Object) классы используются для передачи структурированных данных между слоями приложения, обеспечивая удобство сериализации и безопасности обмена информацией. Сущности отражают реальные таблицы базы данных и хранят основное состояние бизнес-объектов, а репозитории выступают в роли прослойки между сервисами и СУБД, инкапсулируя доступ к данным.

Особое место в архитектуре занимает слой вычислений и бизнес-логики, представленный множеством специализированных решателей (Solvers) для экономических моделей, каждый из которых реализует определённый метод расчёта или анализа данных. Наличие фабрики решателей позволяет

динамически выбирать и масштабировать алгоритмы моделирования без необходимости переработки всей структуры приложения.

Технические пакеты config, util и exception содержат необходимые для работы приложения служебные классы и конфигурации. Конфигурационные классы обеспечивают параметры подключения к внешним сервисам и настройку безопасности. Утилитарные классы предоставляют вспомогательные методы для сериализации данных, работы с токенами и прочей инфраструктурной логики, а обработчики исключений централизуют контроль ошибок на сервере, способствуя стабильной и предсказуемой работе системы.

Вся архитектура ориентирована на поддержание высокой модульности, расширяемости и удобства поддержки. Такое построение системы облегчает внедрение новых функций, тестирование и локализацию изменений, а также способствует безопасности и производительности серверного приложения.

2.7 Проектирование структуры классов клиента

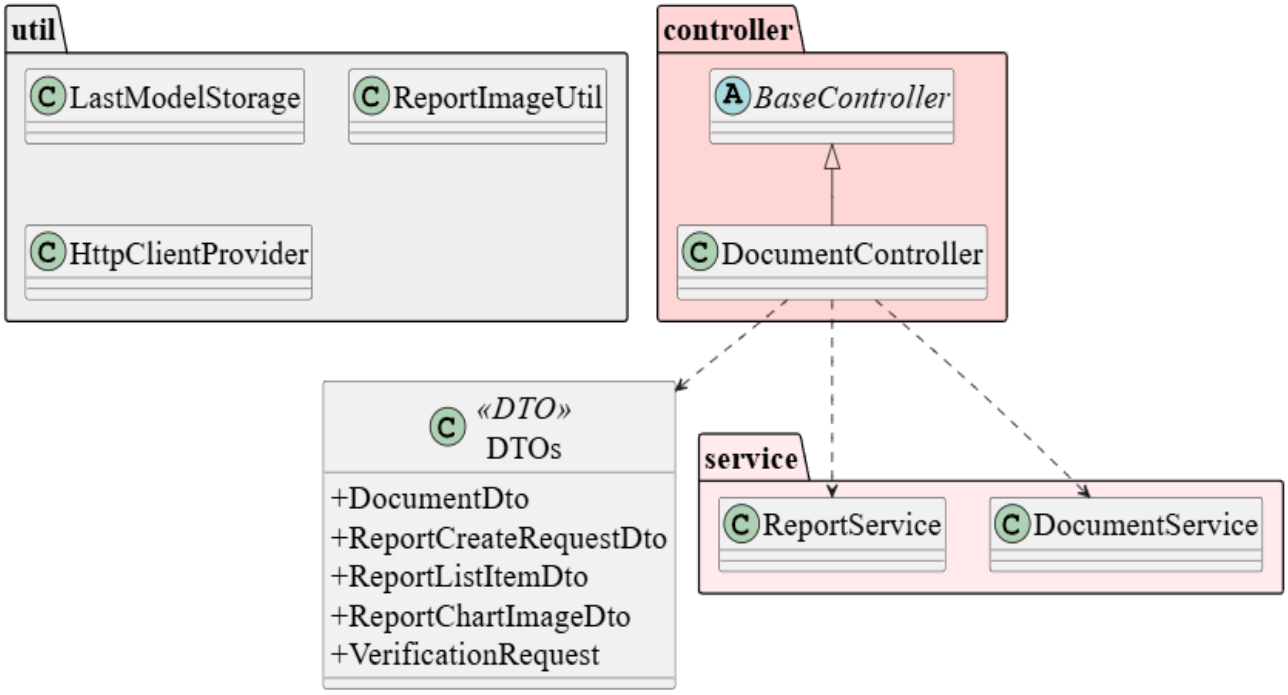

Рисунок 6 – Диаграмма классов клиента

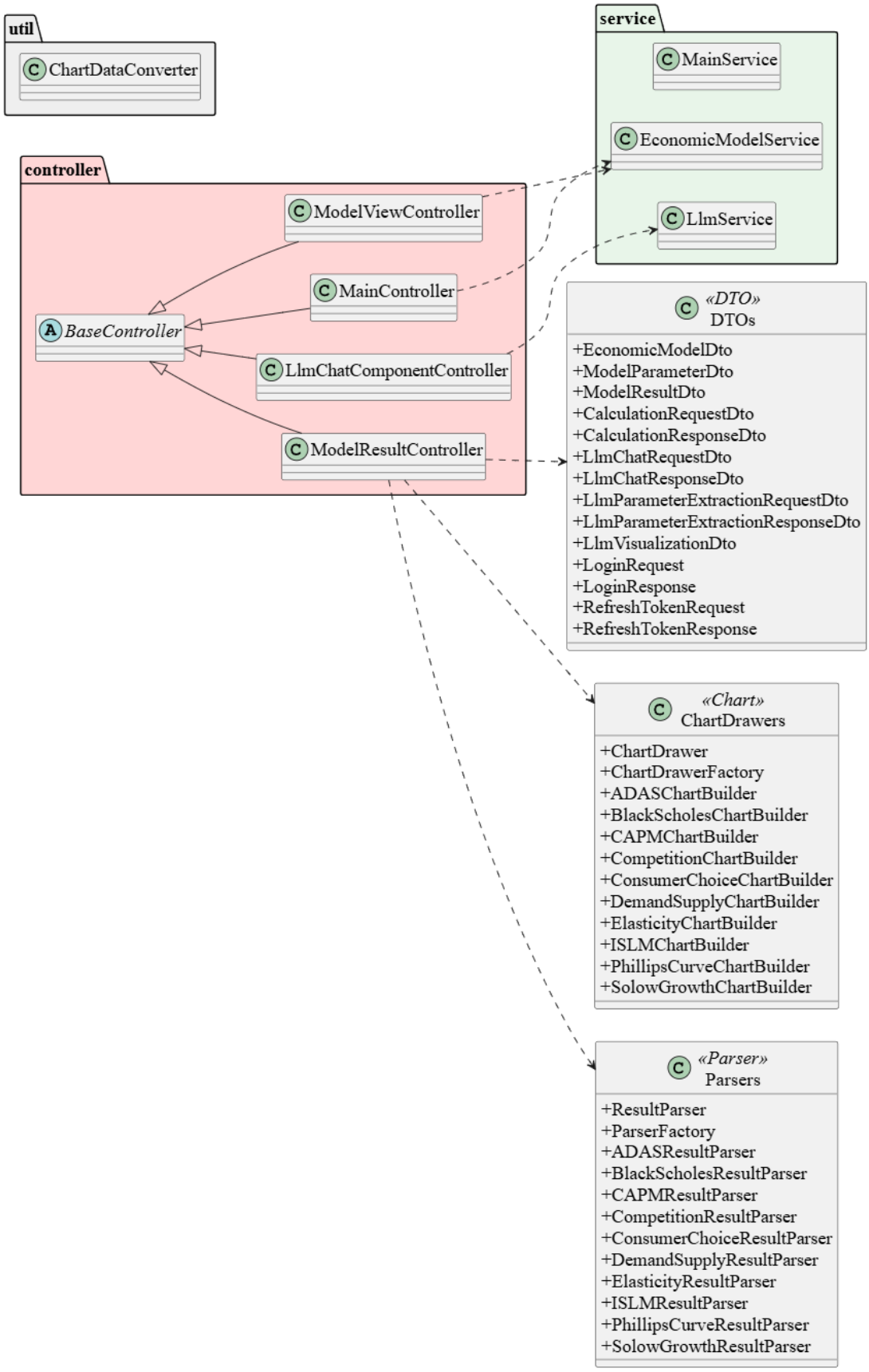

Рисунок 6 – Диаграмма классов клиента (продолжение)

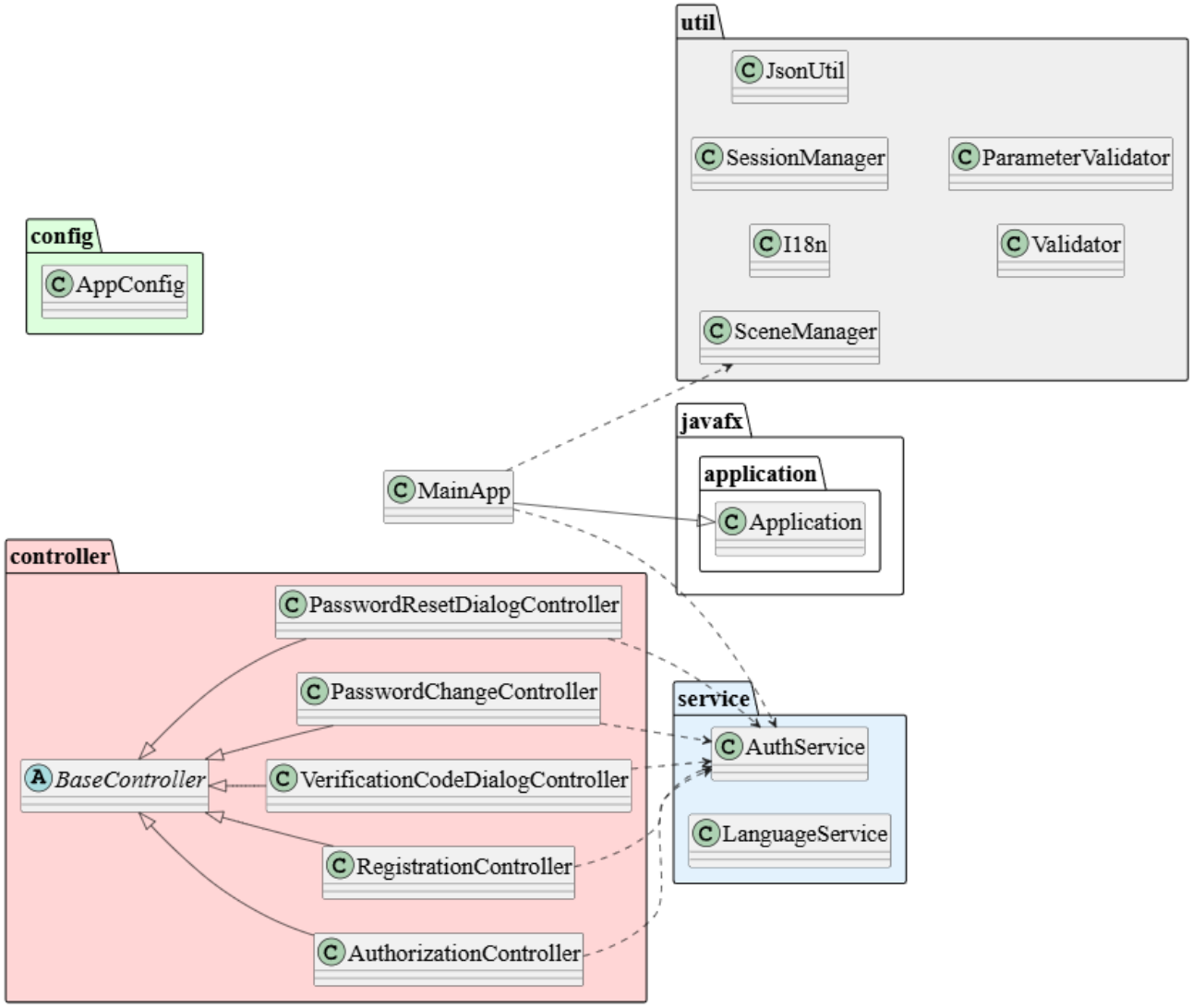

Рисунок 6 – Диаграмма классов клиента (продолжение)

Клиентская часть информационной системы реализована с учётом принципов модульности и строгого разделения ответственности, что обеспечивает удобство поддержки, расширяемость и стабильность работы приложения. Диаграмма классов клиента, показанная на рисунке 6, наглядно отражает структуру и основные взаимосвязи между ключевыми компонентами пользовательского интерфейса, вспомогательных утилит, сервисов бизнес-логики, контроллеров и специализированных модулей обработки данных.

Верхний уровень архитектуры включает пакеты конфигурации и утилит. Конфигурационные классы отвечают за централизованное хранение параметров, используемых во всём приложении, таких как базовые адреса серверов и

глобальные настройки. Утилитарные компоненты реализуют общие функции, необходимые для корректной работы клиента: локализацию и управление языками интерфейса, обработку HTTP-запросов, сериализацию и десериализацию данных, управление пользовательской сессией и предпочтениями, а также базовые проверки корректности пользовательского ввода и вспомогательные преобразования данных для визуализации и формирования отчётов.

Основной логический центр приложения представлен двумя крупными слоями — контроллерами и сервисами. Контроллеры, как производные от абстрактного базового класса, реализуют обработку пользовательских действий, обеспечивают инициализацию интерфейса и связывают графическую составляющую с бизнес-логикой. Каждый контроллер отвечает за отдельный экран или функциональный блок приложения, что способствует изоляции логики и облегчает внесение изменений или добавление новых пользовательских сценариев без риска для остальной системы.

Сервисный слой инкапсулирует бизнес-логику клиентской части. Сервисы обеспечивают взаимодействие с серверной частью приложения, обрабатывают данные, реализуют логику авторизации, управления документами, получения и обработки экономических моделей, а также интеграцию с интеллектуальными сервисами и формирование отчётов. Такая структура позволяет повторно использовать бизнес-функциональность и централизованно обрабатывать ошибки, возникающие при коммуникации с сервером.

Особое место занимают три специализированных агрегатора: DTOs, ChartDrawers и Parsers. Агрегатор DTOs объединяет все структуры данных, используемые для обмена информацией между слоями клиента и для передачи параметров серверу, обеспечивая строгую типизацию и безопасность данных на протяжении всего жизненного цикла работы с информацией. Модуль ChartDrawers реализует стратегию построения различных графиков и диаграмм для визуализации экономических моделей, что значительно облегчает

интеграцию новых способов представления данных и расширяет возможности пользовательского интерфейса. Агрегатор Parsers включает в себя логику интерпретации и форматирования полученных от сервера результатов расчётов, делая их удобочитаемыми и понятными для пользователя.

Наследование и композиция классов реализованы с учётом лучших практик объектно-ориентированного проектирования. Общие методы и свойства вынесены в базовые абстракции, что снижает дублирование кода и повышает поддерживаемость системы. Связи между классами отображают реальные потоки данных и управляющие зависимости, что обеспечивает прозрачность взаимодействия и облегчает отладку.

Вся архитектура клиентской части строится вокруг принципа "тонкого клиента" с фокусом на управлении интерфейсом и бизнес-логикой взаимодействия с сервером.

2.8 Проектирование дизайна

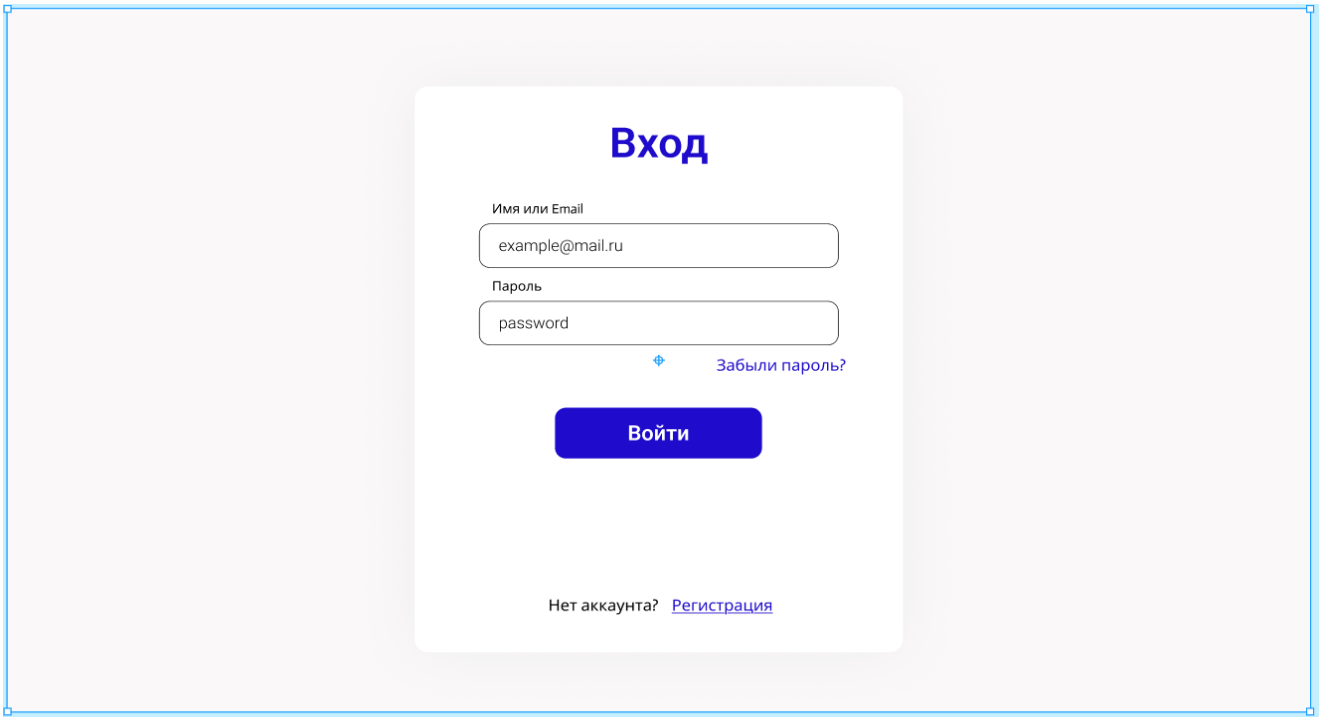

Рисунок 7 – Дизайн экрана в хода

На рисунке 7 представлена форма авторизации пользователя, реализованная в рамках клиентского приложения симулятора экономических моделей. В центре экрана размещён компактный модальный блок с визуальным акцентом на заголовке "Вход", выполненном крупным синим шрифтом, что сразу привлекает внимание пользователя и чётко обозначает назначение текущего окна. Под заголовком располагаются два поля для ввода: первое предназначено для ввода имени пользователя или электронной почты, второе — для пароля. Оба поля имеют современное оформление с плавными закруглёнными границами, что соответствует актуальным тенденциям в дизайне пользовательских интерфейсов и делает процесс ввода более комфортным визуально.

Общий стиль экрана подчеркивает минимализм и чистоту, что позволяет пользователю сфокусироваться непосредственно на задаче входа, не отвлекаясь на лишние детали. Такое решение способствует повышению удобства и скорости взаимодействия, а также соответствует принципам доступности, необходимым для образовательных приложений, ориентированных на широкую аудиторию.

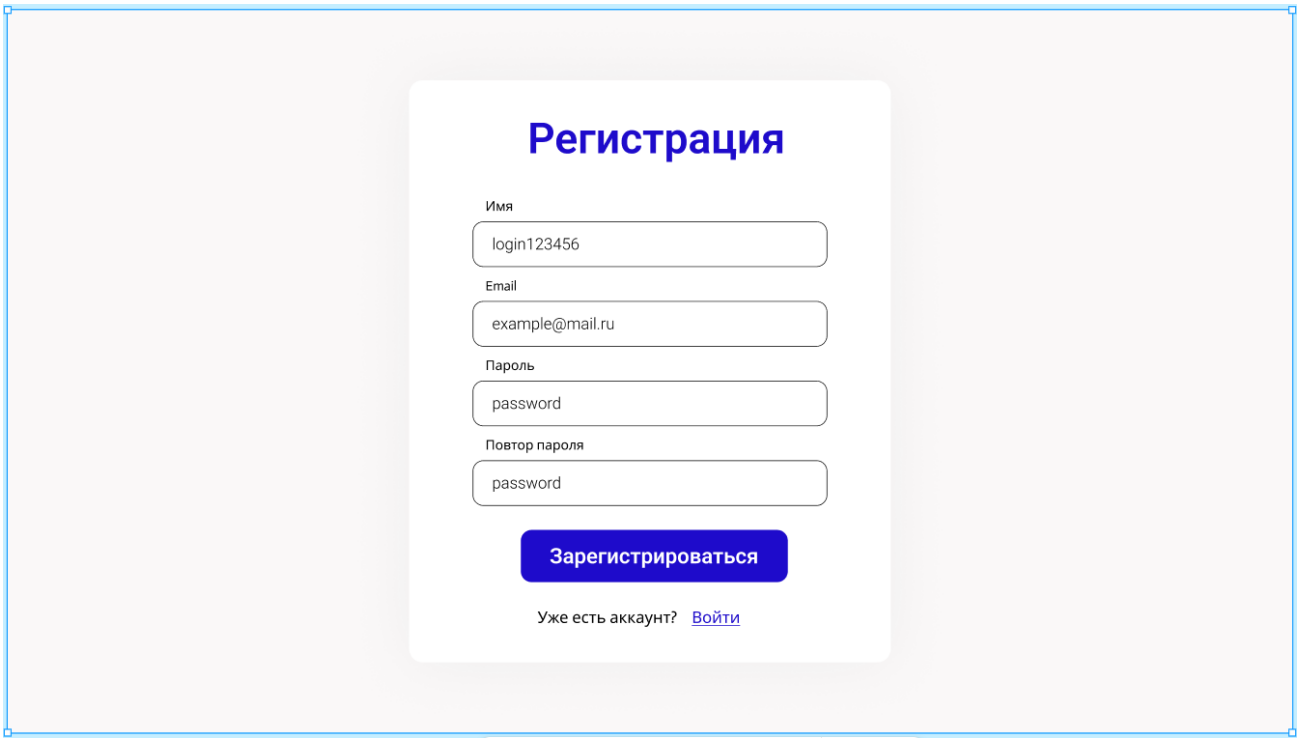

Рисунок 8 – Экран регистрации

Экран регистрации, показанный на рисунке 8, пользователя выполнен в едином стиле с формой авторизации и отличается лаконичностью и простотой восприятия. В верхней части располагается крупный заголовок "Регистрация", подчёркивающий назначение окна. Форма содержит четыре поля для ввода: имя, email, пароль и подтверждение пароля. Все поля визуально оформлены в современном минималистичном стиле с плавными закруглениями. Основная кнопка "Зарегистрироваться" выделена ярким синим цветом и хорошо заметна на светлом фоне. В нижней части экрана размещена ссылка для перехода к форме входа, что обеспечивает удобство навигации между основными действиями. Общий дизайн ориентирован на простоту и понятность для пользователей.

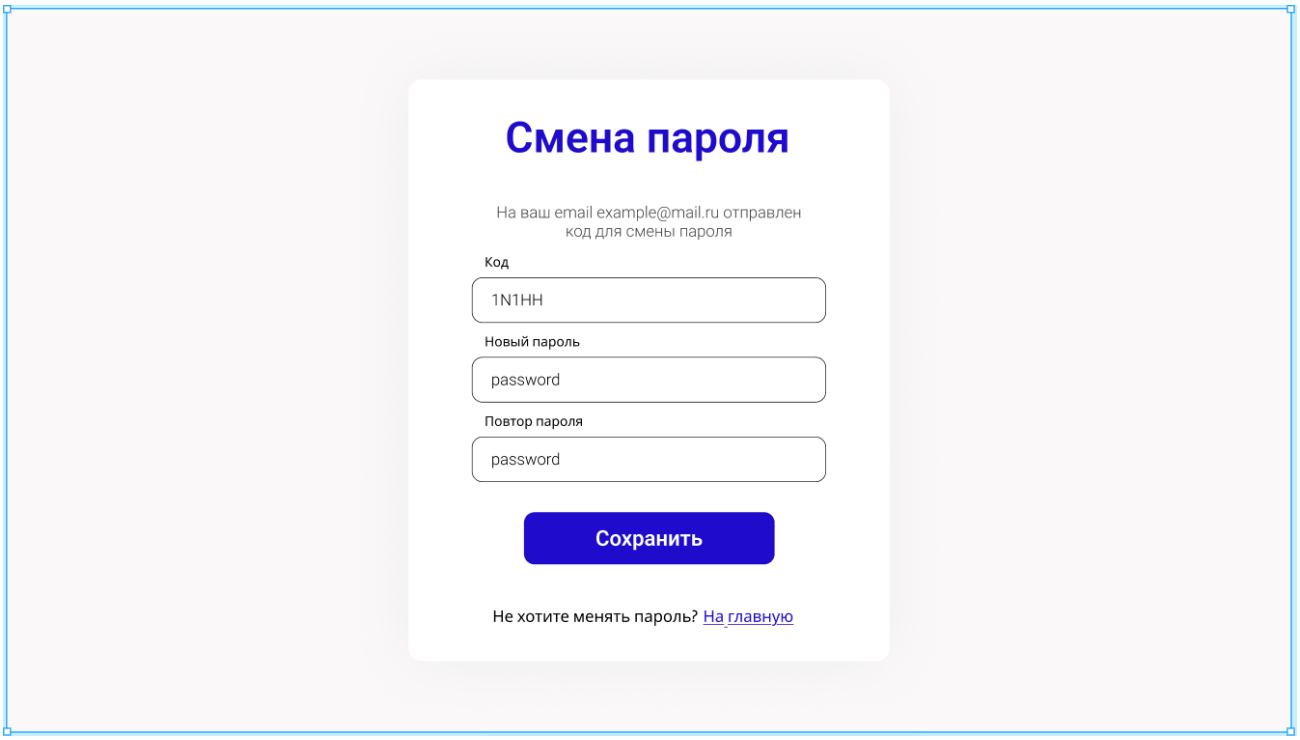

Рисунок 9 – Экран смены пароля

Экран смены пароля, изображенный на рисунке 9, выполнен в том же минималистичном стиле, что и остальные формы аутентификации. В центре располагается заголовок "Смена пароля", под которым кратко объясняется порядок действий. Пользователь вводит полученный по email код, новый пароль и подтверждение пароля. Все поля оформлены просто и удобно для ввода. Яркая

синяя кнопка "Сохранить" акцентирует внимание, а дополнительная ссылка позволяет быстро вернуться на главный экран. Дизайн способствует быстрому и комфортному восстановлению доступа к системе.

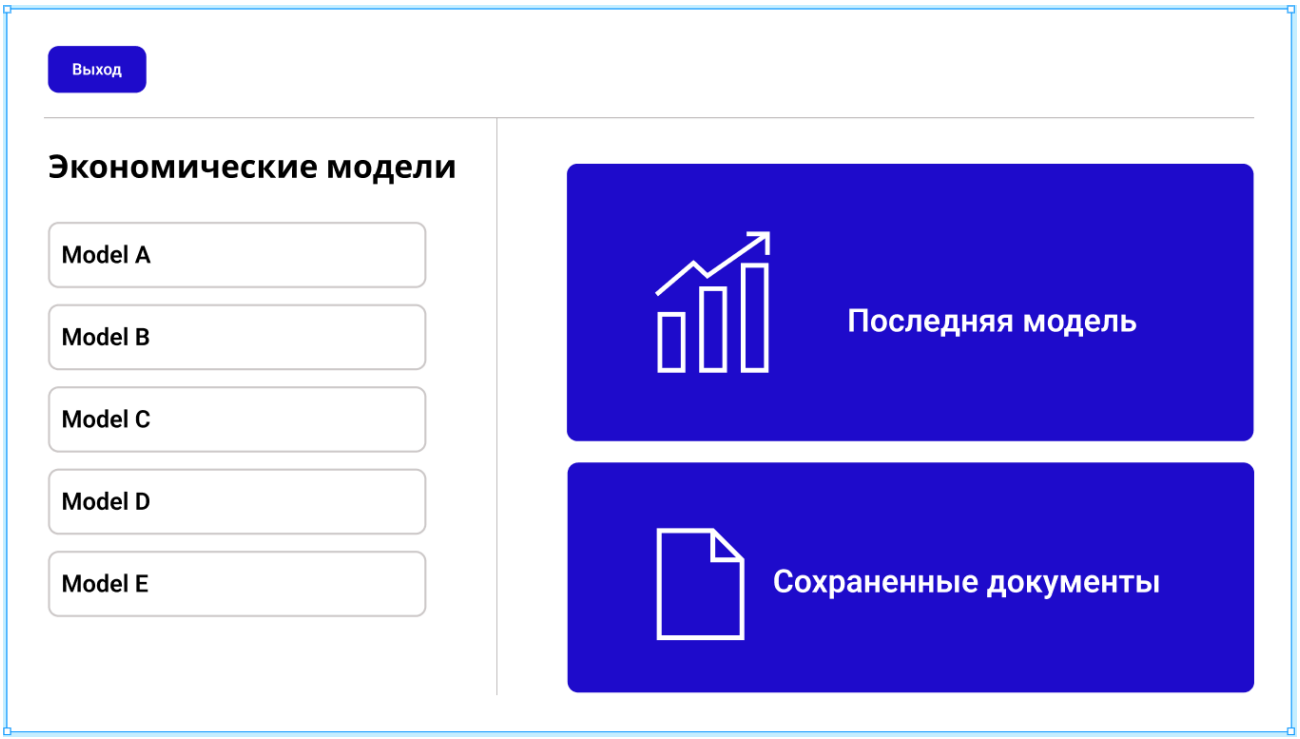

Рисунок 10 – Главный экран

Главный экран, показанный на рисунке 10, приложения предназначен для навигации между основными разделами и выбора экономических моделей для дальнейшей работы. Слева размещён вертикальный список всех доступных моделей, оформленный в виде отдельных кнопок, что позволяет пользователю быстро выбрать интересующую его модель. Заголовок блока выполнен крупным шрифтом для удобства ориентации на странице.

Правая часть экрана разделена на два крупных функциональных блока, каждый из которых выделен насыщенным синим фоном и снабжён крупной иконкой. Верхний блок предназначен для быстрого перехода к последней используемой модели — здесь реализована концепция "быстрого доступа", чтобы пользователь мог сразу вернуться к незавершённой работе. Нижний блок предназначен для перехода к просмотру и управлению сохранёнными

документами. Текстовые подписи и графические элементы визуально разграничивают назначение каждого блока и создают современный, лаконичный облик интерфейса.

Верхняя панель содержит заметную кнопку "Выход", вынесенную отдельно для быстрого завершения сессии. Вся структура экрана построена таким образом, чтобы максимально упростить доступ к основным возможностям системы и обеспечить комфортную навигацию даже для новых пользователей. Поддерживается баланс между визуальной привлекательностью и функциональностью, что положительно сказывается на пользовательском опыте при работе с симулятором экономических моделей.

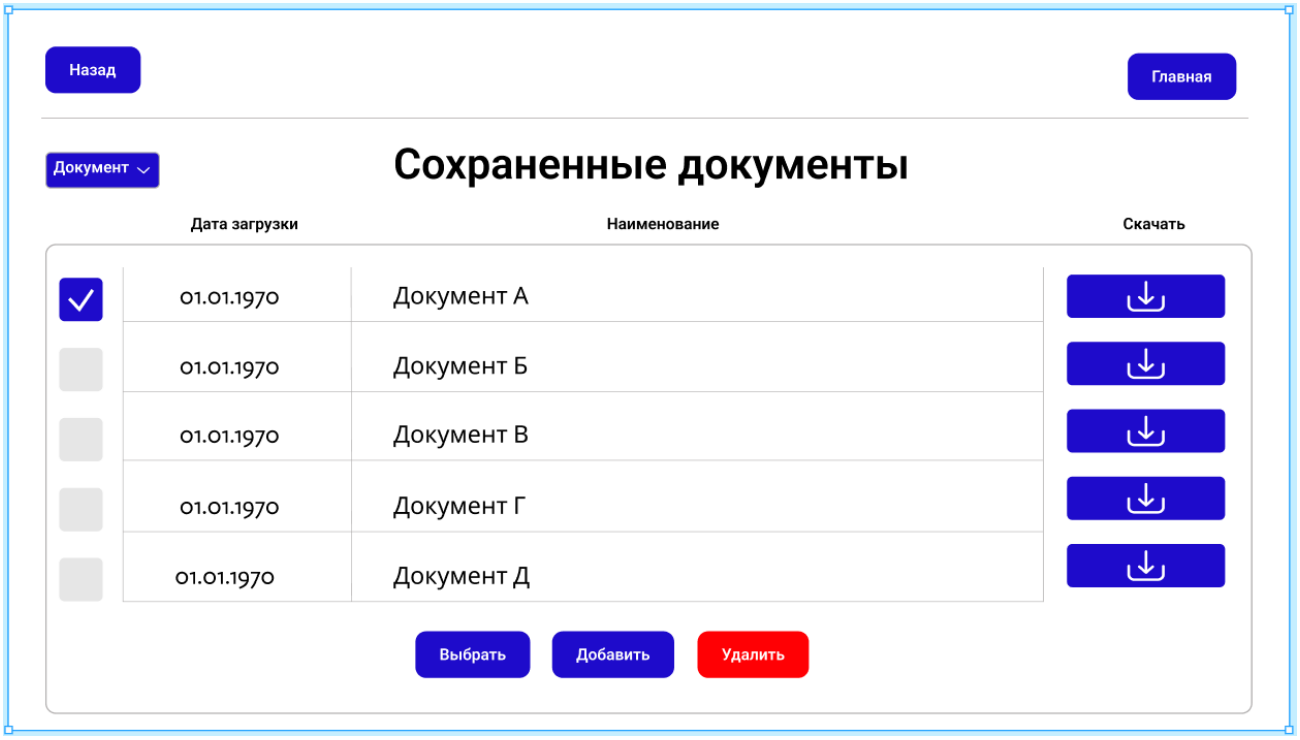

Рисунок 11 – Экран управления сохраненными документами

Экран управления сохранёнными документами, показанный на рисунке 11, реализован в виде удобного интерактивного списка, где пользователь может легко просматривать, выбирать, загружать и удалять необходимые файлы. В верхней части экрана размещены крупный заголовок и две заметные кнопки навигации — «Назад» и «Главная», что обеспечивает быстрый возврат к

предыдущим разделам или на главный экран приложения. Над таблицей присутствует выпадающий список для фильтрации документов по типу, что позволяет при большом количестве файлов быстро находить нужные материалы.

Центральную часть интерфейса занимает табличное представление всех сохранённых пользователем документов. Для каждого документа отображается дата загрузки, название, а также предоставлена отдельная кнопка для скачивания файла, выделенная ярким цветом и крупной иконкой. Это делает процесс загрузки простым и интуитивно понятным даже для неопытных пользователей. Слева от каждой строки расположена область для выбора документа, позволяющая отмечать один или несколько файлов для последующих действий, таких как удаление или работа с выбранным документом.

В нижней части экрана располагается панель управления с тремя кнопками — «Выбрать», «Добавить» и «Удалить». Кнопки отличается цветовым оформлением для быстрого визуального различия функций: синяя для выбора, красная для удаления и нейтральная для добавления.

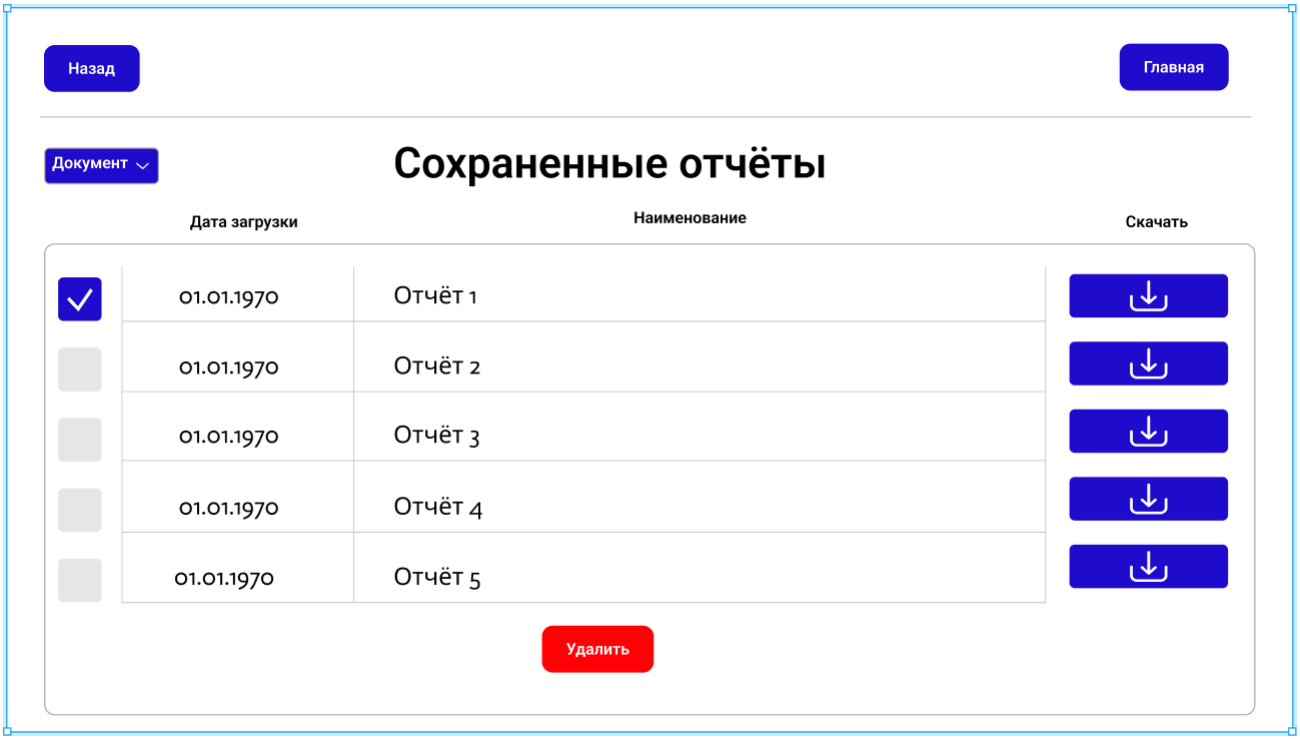

Рисунок 12 – Экран управления сохраненными отчетами

Экран управления сохранёнными отчётами, представленный на рисунке 12, оформлен в едином стиле с остальными разделами приложения и предназначен для быстрого доступа к сформированным пользователем результатам моделирования. В центре страницы размещена таблица с отчетами, для каждого из которых отображаются дата загрузки, название и кнопка для скачивания файла. Пользователь может отметить нужный отчёт для дальнейших действий, например, удаления через выделенную красную кнопку внизу. Такой подход обеспечивает удобство и простоту управления отчётами, позволяя легко находить, скачивать или удалять нужные документы без лишних переходов. Продуманная организация элементов интерфейса создаёт комфортную рабочую среду, способствуя продуктивному взаимодействию пользователя с системой. Всё это делает процесс работы с отчётами интуитивно понятным и максимально удобным в повседневном использовании.

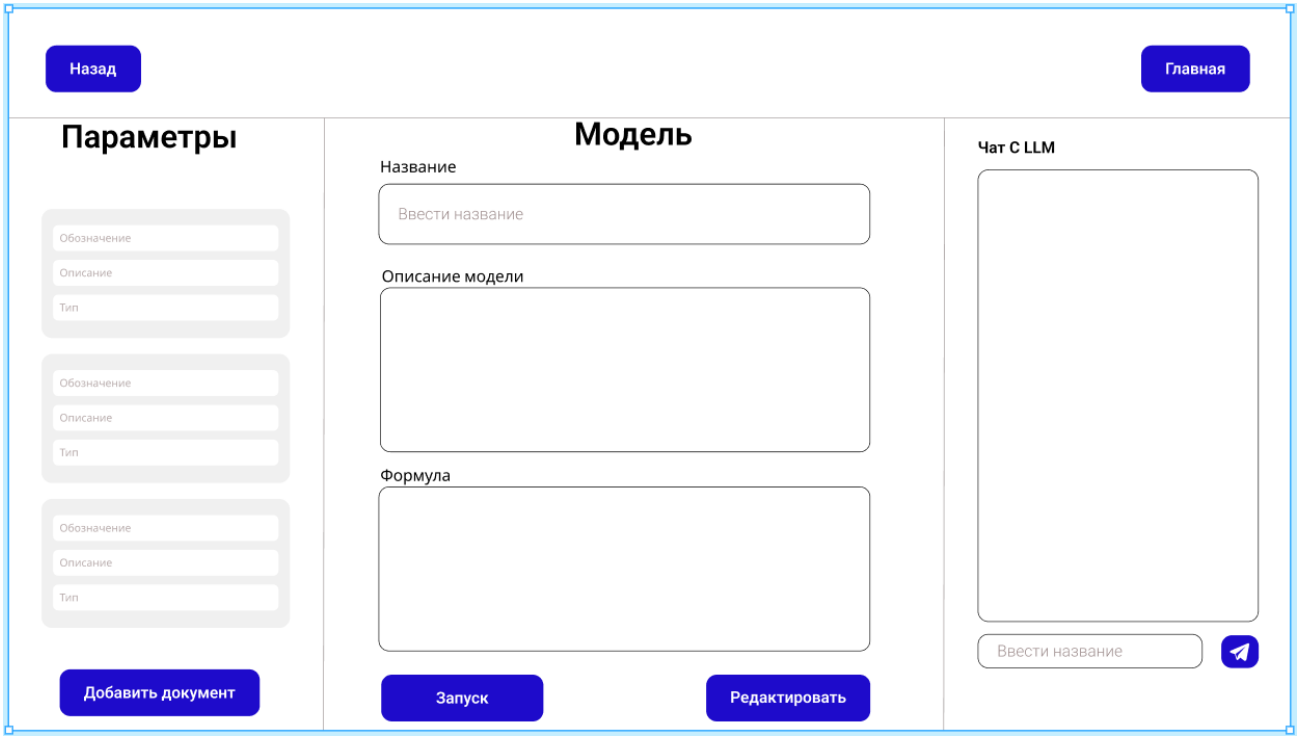

Рисунок 13 – Экран редактирования и просмотра экономической модели

Экран редактирования и просмотра экономической модели, изображенный на рисунке 13, реализован в виде трёх логических блоков, обеспечивающих

наглядность и удобство работы пользователя с каждым аспектом моделирования. Слева размещается панель параметров, где для каждого параметра предусмотрены отдельные поля для описания и ввода значений, а также кнопка для добавления нового документа, который может быть использован для автоматического извлечения данных. Центральная часть экрана посвящена настройке самой модели: здесь пользователь может ввести или скорректировать название, подробное описание и формулу модели. Кнопки «Запуск» и «Редактировать» позволяют в один клик выполнить вычисления либо сохранить изменения, что минимизирует количество лишних действий и способствует высокой скорости работы.

Правую часть экрана занимает чат с LLM (языковой моделью), который открывает возможность для интерактивного взаимодействия: пользователь может задавать вопросы, уточнять непонятные моменты по работе модели или получать пояснения к результатам.

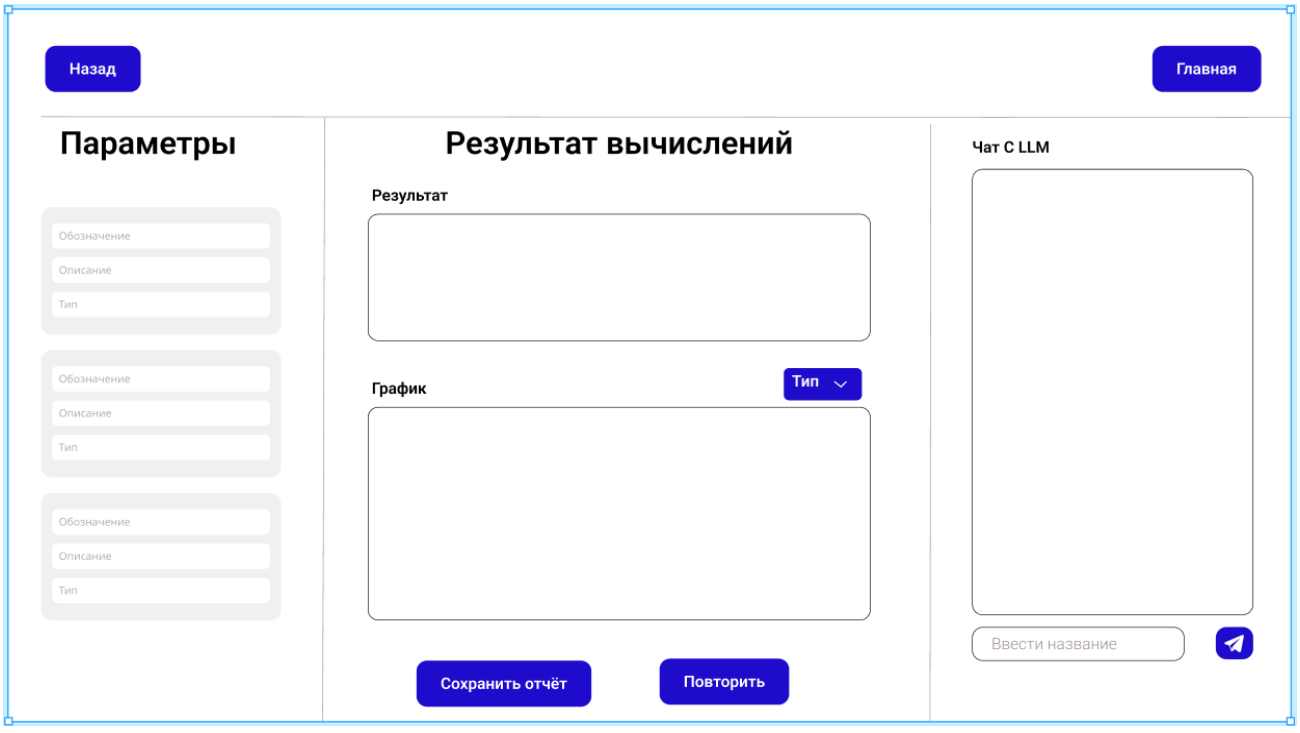

Рисунок 14 – Экран отображения результатов вычислений

Экран отображения результатов вычислений, представленный на рисунке 14, построен по тому же принципу, что и окно редактирования модели, но его основная задача — предоставить пользователю исчерпывающую информацию о завершённом расчёте. В левой части вновь располагается панель с исходными параметрами, что позволяет при необходимости сразу оценить, какие данные легли в основу проведённого анализа. Центральная зона выделена под блок результатов: в верхней части размещается текстовое поле, в котором отображается краткое текстовое описание итогов вычислений, а чуть ниже — визуализатор графика, где пользователь может выбрать тип отображения графических данных с помощью выпадающего списка. Такой подход обеспечивает гибкость визуального представления и позволяет глубже анализировать динамику или структуру результатов, не покидая текущего экрана.

Функциональные кнопки «Сохранить отчёт» и «Повторить» дают возможность либо быстро зафиксировать полученные данные для последующего использования, либо провести перерасчёт с новыми параметрами. Это повышает удобство и способствует непрерывности исследовательского процесса. В правой части интерфейса, как и на других ключевых экранах, присутствует чат с LLM, который позволяет оперативно получить пояснения по результатам, задать дополнительные вопросы или уточнить детали проведённого анализа. Совокупность этих элементов обеспечивает пользователю прозрачный, логичный и интуитивно понятный сценарий взаимодействия с результатами работы симулятора, что особенно важно для образовательной среды.

Разработка дизайна системы требует баланса между детализацией и читаемостью, чтобы обеспечить ясное понимание архитектуры без избыточной сложности. Важно фокусироваться на ключевых компонентах и их взаимодействии, избегая перегруженности второстепенными элементами.

3 Техническая часть

Проект “Симулятор экономических моделей” базируется на комбинации Spring Boot сервера и JavaFX клиента. Главная идея заключается в том, чтобы хранить экономические модели в базе данных, вычислять их результаты на сервере, а клиентское приложение обеспечивало удобный интерфейс для ввода параметров и отображения вычислений. В центре архитектуры лежит база данных PostgreSQL, где хранятся пользователи, модели, параметры, результаты и служебные данные. Сервер выступает связующим звеном, контролирующим права доступа, вычисляющим модели и отдающим данные клиенту.

Для аутентификации используется JWT: сервер генерирует access- и refresh-токены. При обращении к защищенным ресурсам клиент передает токен в заголовке, а сервер проверяет его и извлекает данные пользователя. Это обеспечивает безопасное использование API и позволяет разделить доступ к различным моделям и результатам. Кроме того, в проект добавлена локализация сообщений, что позволяет работать как на русском, так и на английском языках. Все сообщения выводятся на стороне клиента в зависимости от выбранного языка, а сервер умеет возвращать текст в нужной локали.

Серверные сервисы отвечают за загрузку моделей, формирование параметров и расчет результатов. Главный сервис загрузки моделей считывает конфигурацию из базы данных, сопоставляет параметры с соответствующим solver и сохраняет результаты в таблицу model_results.

Вся работа над сервером и клиентом выполняется в рамках единого репозитория. Это облегчает управление версиями и позволяет синхронизировать обновления кода. Однако модули четко разделены: запуск сервера и клиента происходит отдельно, и каждый может работать без знания внутренней структуры другого. Такой подход упрощает тестирование и внедрение новых функций: сервер можно обновить, не перекомпилируя клиент, и наоборот.

Во время разработки большое внимание уделялось организации пользовательского интерфейса. Задача состояла в том, чтобы максимально упростить ввод параметров и визуализировать результаты. Для этого в JavaFX используются компоненты TableView и формы с текстовыми полями и выпадающими списками. При изменении параметров пользователь сразу видит влияние на результаты.

Дизайн также учитывает многопоточность. Сервер выполняет вычисления моделей в отдельных потоках, что позволяет обслуживать нескольких пользователей одновременно. Клиентская часть тоже использует фоновые задачи, чтобы не блокировать интерфейс при запросах к серверу. Таким образом, даже при долгих вычислениях пользователь продолжает работать с интерфейсом, а результаты приходят по завершении операции.

Разработка проводилась с применением принципов модульности и переиспользования кода. Сервисы и репозитории разбиты на небольшие части, чтобы их можно было легко тестировать и дополнять. Для хранения настроек и параметров используется файл конфигурации и таблицы базы данных. Такая гибкая структура дает возможность в будущем добавлять новые типы моделей без переписывания всей системы.

В итоге проект представляет собой комплексное решение для моделирования экономических процессов. Сервер управляет данными и расчетами, клиент обеспечивает удобный доступ к функциям, а база данных хранит все необходимое для работы системы. В следующих разделах мы подробно рассмотрим каждый из компонентов.

Таким образом, "Симулятор экономических моделей" представляет собой целостную систему, сочетающую мощный серверный функционал с удобным клиентским интерфейсом.

3.1 Разработка базы данных

Основой проекта является надежная база данных, которая должна поддерживать все операции с пользователями, моделями и результатами. Мы выбрали PostgreSQL как систему управления, поскольку она обладает широким функционалом и хорошо интегрируется с Spring Boot. Исходная ER-диаграмма определяет структуру будущей базы. В ходе разработки таблицы дополнялись дополнительными полями и индексами, которые ускоряют поиск и обеспечивают целостность данных.

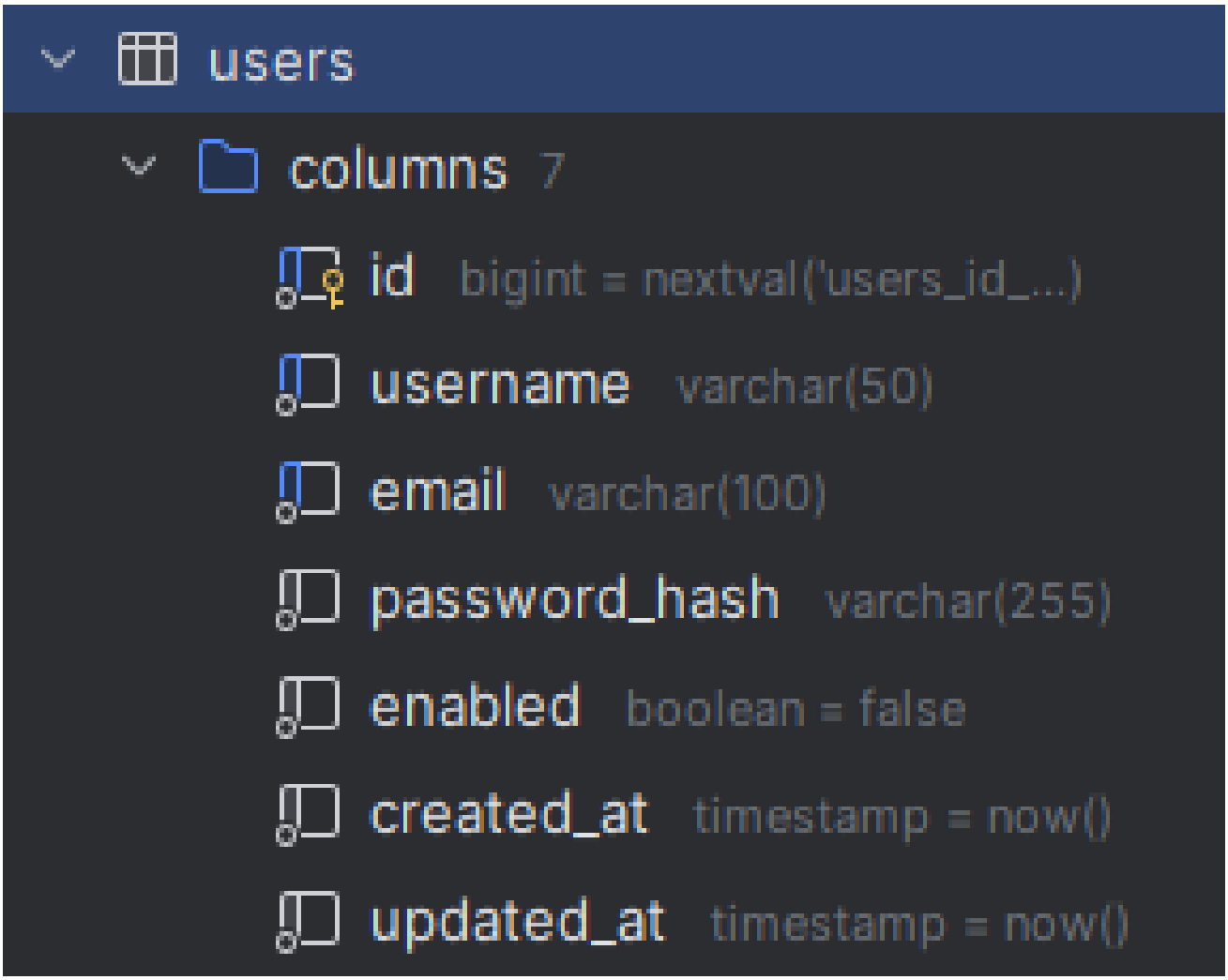

Рисунок 15 – Таблица users

Таблица users, изображенная на рисунке 15, хранит учетные записи, и здесь много внимания уделяется безопасности. Помимо стандартных полей вроде имени пользователя, email и хэшированного пароля, присутствует флаг enabled, являющийся флагом подтверждения пользователем своего email. Временные метки created_at и updated_at позволяют отслеживать изменения, что может пригодиться для аудита.

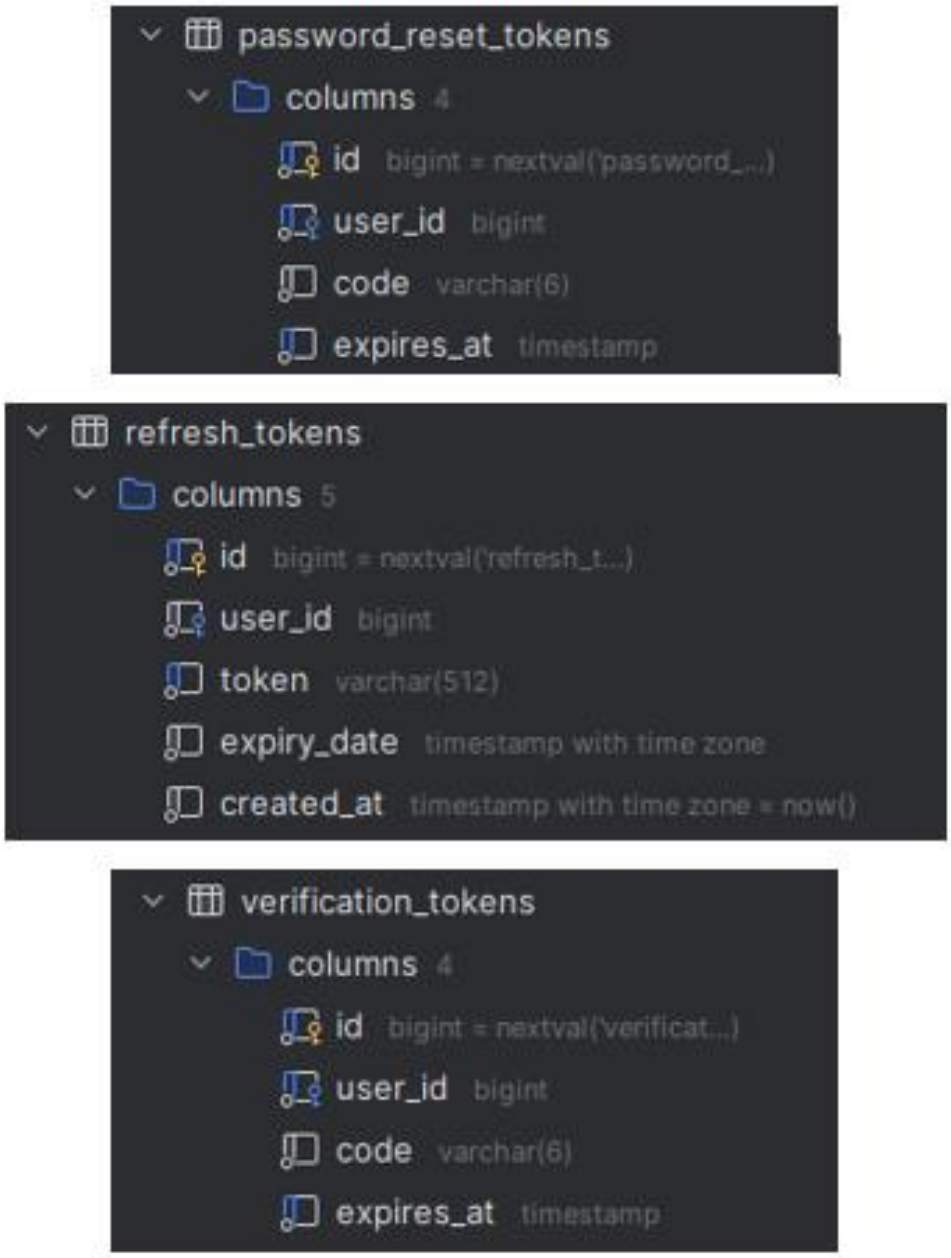

Рисунок 16 – Таблицы токенов

Для каждого пользователя создаются связанные записи refresh-токенов, верификационных кодов и кодов сброса пароля — это помогает управлять сессиями и восстановлением доступа. В таблицах токенов, показанных на рисунке 16, verification_tokens и password_reset_tokens сохраняются коды подтверждения при регистрации и восстановлении пароля. Каждая запись имеет срок годности, по истечении которого код становится недействительным. Это обеспечивает безопасность и контроль над жизненным циклом токенов.

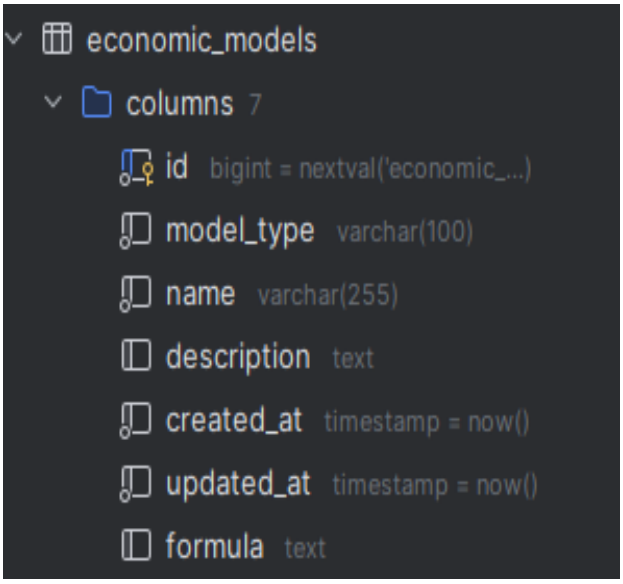

Рисунок 17 – Таблица economic_models

Таблица economic_models, показанная на рисунке 17 — это ядро системы. Здесь хранятся типы моделей, названия, описания и формулы. Каждая модель имеет собственный идентификатор, по которому потом связывается с параметрами, результатами и отчетами. Формула может быть представлена в любом удобном формате, например в виде строки для подстановки в математический движок. Важным дополнением являются временные метки, которые позволяют отслеживать, когда модель была создана или обновлена. Это полезно при разработке новой версии модели или анализе ее изменений.

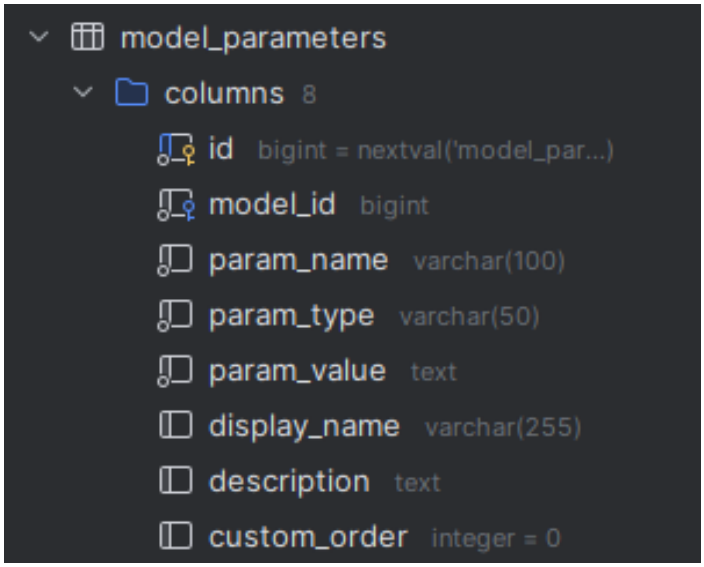

Рисунок 18 – Таблица model_parameters

Следующей значимой таблицей является model_parameters, изображенная на рисунке 19. В ней перечислены все параметры, которые используются в моделях. Каждый параметр связан с конкретной моделью через model_id, имеет имя, тип и значение. Дополнительно указаны отображаемое название (display_name) и описание, чтобы клиент мог легко представить параметры пользователю. Поле custom_order помогает сортировать параметры в нужном порядке. Такая гибкая структура делает модель расширяемой: можно добавить новые параметры, не меняя клиентский код.

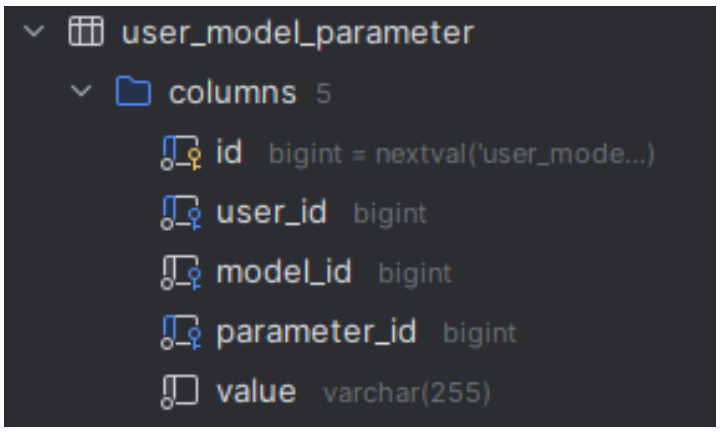

Рисунок 19 – Таблица user_model_parameter

Таблица user_model_parameter, изображенная на рисунке 19, предназначена для хранения индивидуальных значений параметров, которые вводит пользователь. Это особенно важно для сценариев, где различные пользователи хотят протестировать свои модели с уникальными данными.

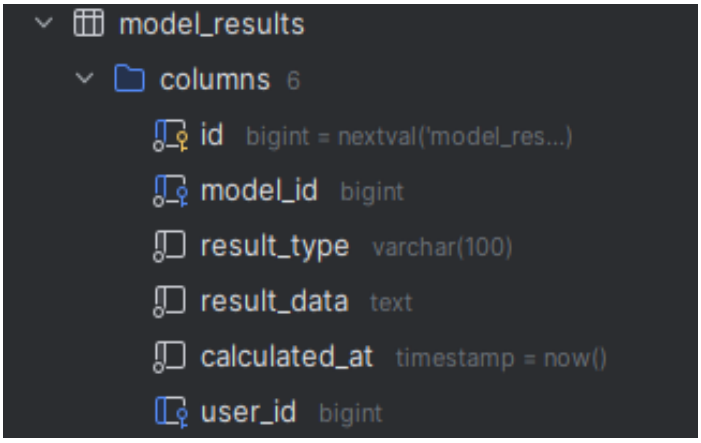

Рисунок 20 – Таблица model_results

Важную роль играет таблица model_results, представленная на рисунке 20. Как только пользователь запускает расчет модели, результаты сохраняются здесь. Это может быть текст, JSON или другая структура, в которой удобно хранить выходные данные. Таблица также фиксирует дату вычислений и связывает их с пользователем и моделью.

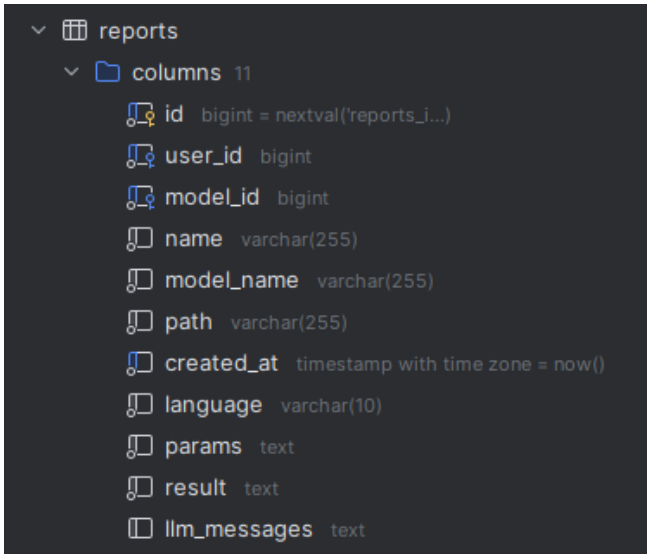

Рисунок 21 – Таблица reports

Таблица reports, изображенная на рисунке 21, используется для генерации и хранения отчетов. В ней лежат ссылки на пользователя, модель, название отчета и путь к сгенерированному файлу. Также хранятся параметры, которые применялись для расчета, и результат в текстовом виде. Поле llm_messages пригодилось для сохранения сообщений языковой модели, если в отчете есть объяснительный текст.

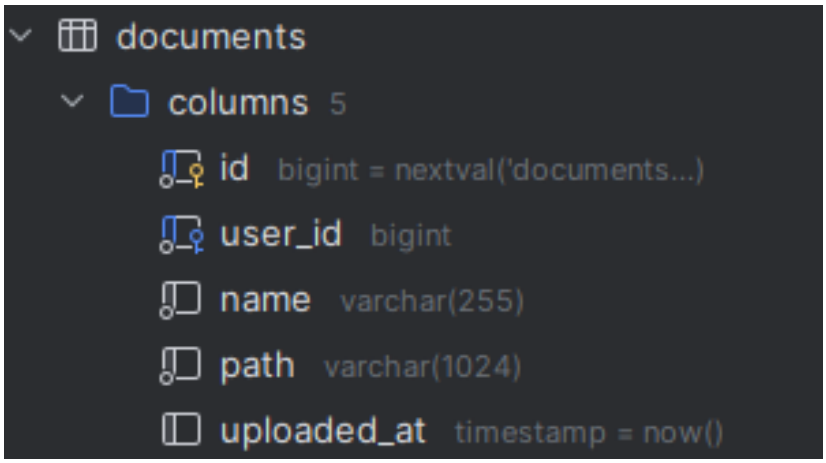

Рисунок 22 – Таблица documents

Переходя к documents, изображенной на рисунке 22, стоит упомянуть, что эта таблица служит для хранения путей к загруженным файлам. Пользователь может добавлять документы, которые нужны для моделирования. Кроме пути, фиксируется дата загрузки и идентификатор пользователя. Все файлы лежат на сервере, и таблица обеспечивает лишь индексирование и быстрый доступ к ним.

Такой набор таблиц формирует фундаментальную структуру базы. Связи между ними, обозначенные внешними ключами, гарантируют целостность данных. Пользователь связан с моделями, параметрами и результатами, а модели — с параметрами и отчетами. Благодаря этому можно в любой момент получить полную картину работы пользователя с конкретной моделью, что делает систему удобной для анализа и расширения.

3.2 Разработка сервера

Сервер — центральная часть приложения, отвечающая за обработку запросов, выполнение расчетов и управление данными. Он построен на Spring Boot, поскольку этот фреймворк предоставляет мощный инструментарий для веб-приложений, безопасности и работы с базой данных. При старте приложение загружает необходимые конфигурации, соединяется с PostgreSQL и начинает принимать HTTP-запросы. Каждый запрос проходит через цепочку фильтров, где проверяется наличие токена и права пользователя. Конфигурация безопасности реализована в классе SecurityConfig.

```
/**
 * Настраивает фильтры безопасности и правила доступа.
 *
 * @param http объект для конфигурации HTTP безопасности
 * @return цепочка фильтров безопасности
 * @throws Exception при ошибках конфигурации
 */
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
    http
            .csrf(AbstractHttpConfigurer::disable)
            .cors(cors -> {})
            .sessionManagement(sm ->

sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                    .requestMatchers("/auth/**").permitAll()
                    .anyRequest().authenticated())
            .addFilterBefore(
                    new JwtAuthenticationFilter(jwtUtil, jwtConfig,
userDetailsService),
                    UsernamePasswordAuthenticationFilter.class);

    return http.build();
}

        /**
         * Выполняет проверку JWT и устанавливает аутентификацию в
контекст.
         *
         * @param request  входящий HTTP-запрос
         * @param response ответ HTTP
```

```
         * @param filterChain цепочка фильтров
         * @throws ServletException в случае ошибок фильтрации
         * @throws IOException при ошибках ввода-вывода
         */
        @Override
        protected void doFilterInternal(HttpServletRequest request,
                                        HttpServletResponse
response,
                                        FilterChain filterChain)
                throws ServletException, IOException {

            String authHeader =
request.getHeader(jwtConfig.getHeader());
            String prefix = jwtConfig.getTokenPrefix();

            if (authHeader != null &&
authHeader.startsWith(prefix)) {
                String token =
authHeader.substring(prefix.length());
                String username = jwtUtil.extractUsername(token);

                if (username != null &&

SecurityContextHolder.getContext().getAuthentication() == null) {

                    var userDetails =
userDetailsService.loadUserByUsername(username);

                    if (jwtUtil.isTokenValid(token, userDetails)) {
                        var authToken = new
UsernamePasswordAuthenticationToken(
                                userDetails, null,
userDetails.getAuthorities());

SecurityContextHolder.getContext().setAuthentication(authToken);
                    }
                }
            }
            filterChain.doFilter(request, response);
        }
```

Здесь задаются правила доступа к различным эндпоинтам. Например, URL-адреса авторизации и регистрации открыты для всех, а доступ к расчетам моделей разрешен только аутентифицированным пользователям. В конфигурации подключаются фильтры JWT, которые извлекают токен из заголовков и проверяют его валидность. Если проверка проходит успешно, данные

пользователя помещаются в контекст безопасности, благодаря чему контроллеры получают информацию о текущем пользователе.

```
/**
 * Получить список моделей для текущего пользователя.
 *
 * @return список моделей в обёртке {@link ResponseEntity}
 */
@GetMapping(«/»)
public ResponseEntity<List<EconomicModelDto>> getAllModels() {
    Long userId = getCurrentUserId();
    return
ResponseEntity.ok(economicModelService.getAllModels(userId));
}

/**
 * Получить модель по её идентификатору.
 *
 * @param id идентификатор модели
 * @return найденная модель
 */
@GetMapping(«/{id}»)
public ResponseEntity<EconomicModelDto> getModelById(@PathVariable
Long id) {
    Long userId = getCurrentUserId();
    return ResponseEntity.ok(economicModelService.getModelById(id,
userId));
}

/**
 * Получить параметры указанной модели пользователя.
 *
 * @param modelId идентификатор модели
 * @return список параметров модели
 */
@GetMapping(“/{modelId}/parameters”)
public ResponseEntity<List<ModelParameterDto>> getUserParameters(
        @PathVariable Long modelId
) {
    Long userId = getCurrentUserId();
    return
ResponseEntity.ok(modelParameterService.getParametersByModelId(56od
emed,
userId));}
```

Контроллеры сервера — это классы, маркированные @RestController. Они принимают HTTP-запросы и возвращают ответы в формате JSON. Например,

EconomicModelController реализует методы для получения списка моделей, запуска расчета и получения результатов. Каждый метод обрабатывает входящие данные, вызывает необходимые сервисы и формирует ответ.

```
/**
 * Авторизация пользователя.
 *
 * @param req запрос с логином и паролем
 * @return пары токенов доступа и обновления
 */
@PostMapping("/login")
public LoginResponse login(@RequestBody @Valid LoginRequest req) {
    return authService.login(req);
}
/**
 * Регистрация нового пользователя.
 *
 * @param req данные, необходимые для регистрации
 * @return {@link ApiResponse} с результатом операции
 */
@PostMapping("/register")
@ResponseStatus(HttpStatus.CREATED)
public ApiResponse register(@RequestBody @Valid RegistrationRequest
req) {
    return authService.register(req);
}
```

В AuthController находятся методы регистрации и входа пользователя. После успешной авторизации сервер создает пары access- и refresh-токенов.

```
/**
 * Выполняет расчёт экономической модели и сохраняет результат для
пользователя.
 *
 * @param request данные для расчёта
 * @param userId  идентификатор пользователя
 * @return результат вычислений
 */
@Transactional
public CalculationResponseDto calculate(CalculationRequestDto
request,Long userId) {
    EconomicModel model =
economicModelRepository.findById(request.modelId())
            .orElseThrow(() -> new
LocalizedException("error.model_not_found"));
```

```
    EconomicModelSolver solver =
solverFactory.getSolver(request.modelType());
    if (solver == null)
        throw new LocalizedException("error.solver_not_found");

    CalculationResponseDto resultDto = solver.solve(request);

    if (resultDto.result() != null) {
        Optional<ModelResult> existingOpt =
modelResultRepository.findByUserIdAndModelId(userId,
model.getId());
        ModelResult modelResult;
        if (existingOpt.isPresent()) {
            modelResult = existingOpt.get();

modelResult.setResultType(resultDto.result().resultType());

modelResult.setResultData(resultDto.result().resultData());
        } else {
            modelResult = new ModelResult();
            modelResult.setModel(model);

modelResult.setUser(userRepository.getReferenceById(userId));

modelResult.setResultType(resultDto.result().resultType());

modelResult.setResultData(resultDto.result().resultData());
        }
        modelResultRepository.save(modelResult);
    }

    return resultDto;
}
```

Бизнес-логика сосредоточена в сервисах. ModelCalculationService выполняет основную работу по запуску модели. Он получает параметры от пользователя, подставляет их в формулу и вызывает вычислитель. Если расчет успешен, сервис сохраняет результат в таблицу model_results и возвращает его пользователю. Помимо расчета есть сервисы для работы с пользователями, управления токенами и обработки документов. Разделение на контроллеры и сервисы облегчает поддержку, поскольку можно менять внутреннюю реализацию, не влияя на API.

```
/**
 * Репозиторий для CRUD-операций над сущностями {@link
com.example.economicssimulatorserver.entity.EconomicModel}.
 */

@Repository
public interface EconomicModelRepository extends
JpaRepository<EconomicModel, Long> {
}
/**
 * Репозиторий для сохранения и получения сущностей {@link
com.example.economicssimulatorserver.entity.ModelResult}.
 */
@Repository
public interface ModelResultRepository extends
JpaRepository<ModelResult, Long> {

    /**
     * Получает результат пользователя по конкретной модели.
     *
     * @param userId  идентификатор пользователя
     * @param modelId идентификатор модели
     * @return результат модели, если найден
     */
    Optional<ModelResult> findByUserIdAndModelId(Long userId, Long
modelId);
}
```

Слой хранения данных реализован через репозитории JPA. Каждая сущность из базы данных имеет свой интерфейс репозитория, который наследует JpaRepository. Так мы получаем набор готовых методов для операций CRUD, а также возможность писать собственные запросы на JPQL. Например, репозиторий EconomicModelRepository позволяет искать модели по типу или имени, а ModelResultsRepository предоставляет методы для получения истории расчетов пользователя. Благодаря этим репозиториям код контроллеров и сервисов остается компактным и понятным.

```
String subject =
messageSource.getMessage("msg.email_verification_subject", null,
locale);

/**
 * Обрабатывает пользовательские исключения с кодом локализованного
сообщения.
```

```
     *
     * @param ex     выброшенное исключение
     * @param locale текущая локаль клиента
     * @return {@link ApiResponse} с текстом ошибки
     */
    @ExceptionHandler(LocalizedException.class)
    public ResponseEntity<ApiResponse> handleLocalizedException(
            LocalizedException ex, Locale locale) {

        String code = ex.getCode();
        String message;
        try {
            message = messageSource.getMessage(code, ex.getArgs(),
    locale);
        } catch (Exception e) {
            logException(ex);
            message = code;
        }
        return ResponseEntity.badRequest().body(new ApiResponse(false,
    message));
    }
```

Важной частью сервера является международная локализация. Она настроена через стандартные механизмы Spring Boot и позволяет хранить текст сообщений в различных файлах ресурсов. Контроллеры могут возвращать сообщения на языке, указанном в заголовке запроса. Клиент, в свою очередь, передает локаль, и сервер подбирает нужную строку. Это удобно при выводе ошибок или системных сообщений.

```
/**
 * Возвращает все экономические модели с параметрами пользователя.
 *
 * @param userId идентификатор пользователя
 * @return список моделей
 */
@Transactional()
@Cacheable(value = "models", key = "#userId")
public List<EconomicModelDto> getAllModels(Long userId) {
    return economicModelRepository.findAll().stream()
            .map(model -> toDto(model, userId))
            .collect(Collectors.toList());
}
```

Для ускорения работы системы реализовано кеширование моделей. Когда сервер загружает модель из базы данных, он сохраняет ее в памяти, чтобы при следующем запросе не обращаться к базе заново. Этот механизм особенно полезен, если моделей немного, но к ним обращаются часто. Кеш также можно сбрасывать при обновлении модели, чтобы не использовать устаревшие данные. Подобный подход снижает нагрузку на базу и повышает производительность.

Разработка сервера велась с учетом расширяемости. Если в будущем появятся новые типы моделей или нужно будет добавить внешние сервисы, достаточно создать новые контроллеры и сервисы, не трогая существующий код. Благодаря Spring Boot и JPA многие рутинные задачи решаются автоматически, а разработчикам остается сосредоточиться на бизнес-логике и алгоритмах.

```
/**
 * Возвращает солвер по его идентификатору.
 *
 * @param modelType имя модели из запроса
 * @return экземпляр солвера или {@code null}, если он не найден
 */
public EconomicModelSolver getSolver(String modelType) {
    return solverMap.get(modelType);
}
/**
 * Рассчитывает показатели CAPM по заданным параметрам.
 *
 * @param request параметры рынка и характеристики портфеля
 * @return ответ с данными для графиков
 */
@Override
public CalculationResponseDto solve(CalculationRequestDto request)
{
    Map<String, String> paramMap = request.parameters().stream()
            .collect(Collectors.toMap(ModelParameterDto::paramName,
ModelParameterDto::paramValue));

    double Rf    = (Double)
ParameterTypeConverter.fromString(paramMap.get("Rf"),    "double");
    double Rm    = (Double)
ParameterTypeConverter.fromString(paramMap.get("Rm"),    "double");
    double beta  = (Double)
ParameterTypeConverter.fromString(paramMap.get("beta"),  "double");
    double alpha = (Double)
ParameterTypeConverter.fromString(paramMap.get("alpha"), "double");
    double sigma = (Double)
```

```
ParameterTypeConverter.fromString(paramMap.get("sigma"), "double");

    double expectedReturn = Rf + beta * (Rm - Rf) + alpha;

    List<Map<String, Number>> smlPoints = new ArrayList<>();
    smlPoints.add(Map.of("risk", 0.0, "return", Rf));
    smlPoints.add(Map.of("risk", beta, "return", expectedReturn));
    Map<String, Object> smlData = new LinkedHashMap<>();
    smlData.put("sml", smlPoints);

    Map<String, Object> efData = new LinkedHashMap<>();
    List<Map<String, Number>> portfolios = new ArrayList<>();
    for (double b = 0.0; b <= 2.0; b += 0.5) {
        double r = Rf + b * (Rm - Rf) + alpha;
        double s = b;
        portfolios.add(Map.of("risk", s, "return", r));
    }
    efData.put("portfolios", portfolios);
    List<Map<String, Number>> frontierPoints = new ArrayList<>();
    for (double b = 0.0; b <= 2.0; b += 0.2) {
        double r_i = Rf + b * (Rm - Rf);
        frontierPoints.add(Map.of("risk", b, "return", r_i));
    }
    efData.put("frontier", frontierPoints);

    List<Map<String, Object>> decompositionList = new
ArrayList<>();
    decompositionList.add(Map.of(
            "label", "Portfolio",
            "alpha", alpha,
            "beta",  beta
    ));
    decompositionList.add(Map.of(
            "label", "Market",
            "alpha", 0.0,
            "beta",  1.0
    ));
    decompositionList.add(Map.of(
            "label", "Risk-free",
            "alpha", 0.0,
            "beta",  0.0
    ));
    Map<String, Object> decompData = new LinkedHashMap<>();
    decompData.put("decomposition", decompositionList);

    Map<String, Object> allCharts = new LinkedHashMap<>();
    allCharts.put("sml",                smlData);
    allCharts.put("efficient_frontier", efData);
    allCharts.put("decomposition",      decompData);
    ModelResultDto result = new ModelResultDto(
            null,
            request.modelId(),
```

```
            "chart",
            ParameterTypeConverter.toString(allCharts, "json"),
            null
    );
    return new CalculationResponseDto(result,
request.parameters());
}
```

Важной частью работы сервера является модуль вычислений. Он реализован по паттерну «фабрика», что позволяет увеличить переиспользование кода и выбирать необходимый тип решения под каждую модель.

3.3 Разработка клиента

Клиентская часть разработана на JavaFX — это позволило создать современный десктопный интерфейс с широкими возможностями кастомизации. При запуске клиент отображает окно авторизации, где пользователь вводит свои учетные данные. Если вход прошел успешно, приложение загружает главную сцену с доступными функциями.

```
/**
 * Переключает сцену и передаёт контроллер потребителю для
дополнительной настройки.
 *
 * @param fxml путь к FXML-файлу
 * @param controllerConsumer действие над контроллером
 * @param <T>  тип контроллера
 */
public static <T> void switchTo(String fxml,
java.util.function.Consumer<T> controllerConsumer) {
    try {
        if (currentFxml != null)
            history.push(currentFxml);
        currentFxml = fxml;
        Locale locale = I18n.getLocale();
        ResourceBundle bundle =
ResourceBundle.getBundle("messages", locale);
        FXMLLoader loader = new
FXMLLoader(SceneManager.class.getResource(ROOT + fxml));
        loader.setResources(bundle);
        Parent root = loader.load();
```

```
            T controller = loader.getController();
            controllerConsumer.accept(controller);
            scene.setRoot(root);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Сцены переключаются через класс SceneManager, который хранит ссылки на FXML-файлы и обеспечивает кэширование, чтобы ускорить повторное открытие окон.

```
    /**
     * Пытается выполнить вход. При успехе открывает главный экран и
     * создаёт экземпляр {@link EconomicModelService}. Ошибки валидации
    и
     * исключения сервиса выводятся в поле состояния.
     */
    @FXML
    public void doLogin() {
        showError(statusLabel, "");
        String login = usernameEmailField.getText().trim();
        String pass = passwordField.getText();

        if (login.isEmpty() || pass.isEmpty()) {
            showError(statusLabel, "common.status_field");
            return;
        }

        loginButton.setDisable(true);
        new Thread(() -> {
            try {
                auth.login(new LoginRequest(login, pass));
                Platform.runLater(() -> {
                    showSuccess(statusLabel,
    "auth.status_label.successful");
                    BaseController.provide(EconomicModelService.class,
    new EconomicModelService());

    SceneManager.switchToWithoutContorller("main.fxml");
                });
            } catch (IllegalArgumentException ex) {
                Platform.runLater(() -> {
                    showError(statusLabel, tr("error.base") +
    ex.getMessage());
                    passwordField.setText("");
                });
            } catch (Exception ex) {
```

```
            Platform.runLater(() -> showError(statusLabel,
tr("error.base") + ex.getMessage()));
        } finally {
            Platform.runLater(() -> loginButton.setDisable(false));
        }
    }).start();
}
```

На каждом экране присутствуют контроллеры JavaFX, управляемые аннотацией @FXML. Например, AuthorizationController обрабатывает ввод данных пользователя, отправляет запросы на сервер и при успешной авторизации сохраняет полученный refresh-токен. Затем он инициирует переход на главную сцену. FXML-файл определяет расположение элементов, а контроллер реагирует на события и взаимодействует с сервисами.

```
public CalculationResponseDto calculate(CalculationRequestDto
request) throws IOException, InterruptedException {
    String endpoint = "calculate";
    return post(baseUri, endpoint, request,
CalculationResponseDto.class, true, null);
}
@Override
protected <T> T post(
        URI baseUri,
        String endpoint,
        Object body,
        Class<T> respType,
        boolean auth,
        String accessToken
) throws IOException, InterruptedException {
    String currentToken =
SessionManager.getInstance().getAccessToken();
    try {
        return super.post(baseUri, endpoint, body, respType, true,
currentToken);
    } catch (RuntimeException ex) {
        if (ex.getMessage() != null &&
ex.getMessage().startsWith("HTTP 401")) {
            if (AuthService.getInstance().refreshTokens()) {
                currentToken =
SessionManager.getInstance().getAccessToken();
                return super.post(baseUri, endpoint, body,
respType, true, currentToken);
            } else {
                AuthService.getInstance().logout();
```

```
                Platform.runLater(() ->
SceneManager.switchTo("authorization.fxml", c -> ((BaseController)
c).clearFields()));
                throw new
IllegalArgumentException(org.example.economicssimulatorclient.util.
I18n.t("error.session_expired"));
            }
        }
        throw ex;
    }
}
```

В папке service хранятся классы, обеспечивающие коммуникацию с сервером. Здесь реализованы методы отправки запросов, получения и обновления токенов, загрузки моделей и результатов. Например, при отправке запроса на расчет модели сервис добавляет access-токен в заголовок и ожидает ответ в формате JSON. При истечении access-токена сервис автоматически использует refresh-токен, чтобы получить новый и повторить запрос. Такой механизм позволяет пользователю работать длительное время без повторного входа.

```
/**
 * Сохраняет токены в хранилище настроек.
 */
public void saveTokens(String accessToken, String refreshToken) {
    prefs.put(ACCESS_TOKEN_KEY, accessToken);
    prefs.put(REFRESH_TOKEN_KEY, refreshToken);
}
```

Клиент также хранит данные в объекте SessionManager, который отвечает за сохранение токена в файле и его чтение при старте приложения. Благодаря этому пользователь не вводит логин и пароль каждый раз. При необходимости можно выйти из системы, и тогда SessionManager удалит сохраненные данные. Этот модуль упрощает работу с учетной записью и обеспечивает безопасность, так как токены хранятся в зашифрованном виде.

```
/**
 * Создаёт элементы ввода для каждого параметра модели.
 */
private void fillParameters() {
    parameterList.getChildren().clear();
    valueFields.clear();
    for (ModelParameterDto param : parameters) {
        VBox paramBox = new VBox(3);
        paramBox.getStyleClass().add("parameter-card");
        paramBox.setSpacing(2);

        Label symbol = new
Label(localizedValue(param.paramName()));
        symbol.getStyleClass().add("parameter-name");
        Label descr = new
Label(localizedValue(param.description()));
        descr.getStyleClass().add("parameter-desc");
        descr.setWrapText(true);

        TextField value = new TextField(param.paramValue());
        value.setEditable(false);
        value.setPromptText("Значение");
        value.setPromptText(I18n.t("model.value_placeholder"));
        valueFields.add(value);

        paramBox.getChildren().addAll(symbol, descr, value);
        parameterList.getChildren().add(paramBox);
    }
}
```

Ключевой функцией клиента является отображение моделей и ввод параметров. Для каждой модели создается динамическая форма, где параметры выводятся в том порядке, который указан в custom_order. Значения параметров проверяются на стороне клиента — например, для чисел используется валидатор, который не дает ввести некорректные символы. После заполнения формы нажатием кнопки пользователь отправляет параметры на сервер, и по завершении расчета отображается результат.

```
/**
 * Загружает новый PDF-документ и обновляет список.
 */
private void onAdd() {
    if
(!I18n.t("docs.type.documents").equals(typeComboBox.getValue())) {
        showError(statusLabel, "docs.add_only_documents");
```

```
            return;
        }
        FileChooser chooser = new FileChooser();
        chooser.setTitle(I18n.t("common.choose_file"));
        chooser.getExtensionFilters().add(new
FileChooser.ExtensionFilter("PDF Files", "*.pdf"));
        File file =
chooser.showOpenDialog(tableGrid.getScene().getWindow());
        if (file != null) {

            runAsync(() -> {
                try {
                    documentService.uploadDocument(file);
                } catch (IOException | InterruptedException e) {
                    throw new RuntimeException(e);
                }
                Platform.runLater(() -> {
                    showSuccess(statusLabel, "docs.document_uploaded");
                    loadDocuments();
                });
            }, ex -> Platform.runLater(() -> showError(statusLabel,
I18n.t("docs.upload_error") + ex.getMessage())));
        }
}
```

Клиент также умеет работать с файлами. Через экран документов пользователь загружает необходимые данные или отчеты. В этом модуле вызывается сервис, который отправляет файл на сервер и получает подтверждение загрузки. Список документов выводится в таблице с возможностью скачивания и удаления.

Элементы группируются по смыслу: панель моделей, область работы с параметрами и секция результатов. Дополнительные окна открываются только при необходимости и закрываются, когда задача выполнена.

```
/**
 * Подготавливает кнопку смены языка и назначает обработчик.
 */
protected void initLangButton() {
    if (langButton != null) {
        updateLangButtonText();
        langButton.setOnAction(e -> onLangButtonClicked());
    }
}
/**
 * Возвращает перевод строки по ключу из ResourceBundle.
```

```
     * Если ключ не найден, оборачивает его в восклицательные знаки.
     *
     * @param key ключ сообщения
     * @return локализованная строка либо "!key!"
     */
    public static String t(String key) {
        if (bundle == null) setLocale(locale);
        if (bundle.containsKey(key)) {
            return bundle.getString(key);
        }
        return "!" + key + "!";
    }
```

При проектировании были учтены требования к локализации. Все строки, включая заголовки окон и подписи кнопок, вынесены в файл ресурсов, и их можно перевести на любой язык. Переход между языками осуществляется через кнопку. Это позволяет пользователям работать на русском или английском, не перегружая интерфейс лишним текстом.

3.4 Разработка дизайна

Пользовательский интерфейс создан с учетом принципов чистого дизайна и удобства работы. Исходное требование заключалось в том, чтобы пользователь мог быстро задать параметры, запустить модель и увидеть результат. На главной сцене отображаются основные действия: выбор модели, просмотр документов и вход в настройки. Такая компоновка сводит к минимуму количество кликов, необходимых для запуска модели.

```
<AnchorPane xmlns="http://javafx.com/javafx/8"
            xmlns:fx="http://javafx.com/fxml/1"

fx:controller="org.example.economicssimulatorclient.controller.Main
Controller"
            stylesheets="@main.css"
            styleClass="anchor-pane">

    <BorderPane  fx:id="topBar" styleClass="top-bar"
          AnchorPane.topAnchor="0" AnchorPane.leftAnchor="0"
AnchorPane.rightAnchor="0">
```

```
        <left>
            <VBox alignment="CENTER" prefHeight="60">
                <Button fx:id="exitButton" text="%common.exit"
styleClass="exit-button" onAction="#onExitButtonClicked"/>
            </VBox>
        </left>
        <center>
            <Label fx:id="statusLabel" styleClass="status-label"
alignment="CENTER_RIGHT"/>
        </center>
        <right>
            <VBox alignment="CENTER" prefHeight="60"
styleClass="exit-container">
                <Button fx:id="langButton" styleClass="exit-button"
/>
            </VBox>
        </right>
    </BorderPane >

    <Separator styleClass="top-separator"
               AnchorPane.leftAnchor="0" AnchorPane.rightAnchor="0"
AnchorPane.topAnchor="56"/>

    <HBox AnchorPane.topAnchor="60" AnchorPane.bottomAnchor="0"
          AnchorPane.leftAnchor="0" AnchorPane.rightAnchor="0">

        <VBox fx:id="sidePanel" styleClass="side-panel">
            <Label text="%main.models" styleClass="section-title"/>
            <VBox fx:id="modelList" styleClass="model-list"
spacing="16" />
        </VBox>

        <Separator orientation="VERTICAL" styleClass="separator"/>

        <GridPane fx:id="mainGrid" styleClass="main-grid">

            <VBox fx:id="tileButton" styleClass="tile-button, tile-
button-wide"
                  alignment="CENTER" spacing="14"
                  GridPane.rowIndex="0" GridPane.columnIndex="0"
GridPane.columnSpan="2">
                <StackPane styleClass="icon-placeholder">
                    <FontIcon iconLiteral="fa-bar-chart"
styleClass="tile-svg"/>
                </StackPane>
                <Label text="%main.last_model" styleClass="tile-
text"/>
            </VBox>


            <VBox fx:id="tileDocuments" styleClass="tile-button,
tile-button-wide"
```

```
                    alignment="CENTER" spacing="14"
                    GridPane.rowIndex="1" GridPane.columnIndex="0"
GridPane.columnSpan="2">
                <StackPane styleClass="icon-placeholder">
                    <FontIcon iconLiteral="fa-file-pdf-o"
styleClass="tile-svg"/>
                </StackPane>
                <Label text="%docs.saved" styleClass="tile-text"/>
            </VBox>

        </GridPane>

    </HBox>
</AnchorPane>
```

Для описания внешнего вида использована технология FXML. В файле main.fxml задается расположение элементов главного окна: верхняя панель инструментов, боковая панель с доступными моделями и центральная рабочая область. Каждый элемент снабжен id и привязан к контроллеру. Структура FXML легко читается и может быть изменена без перекомпиляции кода, что ускоряет разработку и тестирование интерфейса.

```
.anchor-pane {
    -fx-background-color: #fff;
}

.top-bar {
    -fx-background-color: transparent;
    -fx-pref-height: 60px;
    -fx-padding: 0 0 0 32;
    -fx-alignment: center;
}
.status-label {
    -fx-font-size: 22px;
    -fx-font-family: "Roboto", Arial, sans-serif;
    -fx-font-weight: 600;
    -fx-text-fill: #1e0acb;
    -fx-alignment: center;
}
.exit-container{
    -fx-background-color: transparent;
    -fx-padding: 0 32 0 32;
}
.exit-button {
    -fx-background-color: #1e0acb;
```

```
    -fx-background-radius: 12;
    -fx-text-fill: #fff;
    -fx-font-size: 16px;
    -fx-font-family: "Roboto", Arial, sans-serif;
    -fx-font-weight: 600;
    -fx-cursor: hand;
    -fx-border-color: transparent;
    -fx-effect: null;
}

.top-separator {
    -fx-background-color: #e0e0e0;
    -fx-pref-height: 1;
    -fx-max-height: 1;
    -fx-min-height: 1;
}

.side-panel { -fx-padding: 52 32 0 32; }
.section-title {
    -fx-font-size: 32px;
    -fx-font-family: "Roboto", Arial, sans-serif;
    -fx-font-weight: bold;
    -fx-text-fill: #000;
    -fx-padding: 0 0 32 0;
}

.model-list-button {
    -fx-background-color: #fff;
    -fx-border-color: #cdc9c9;
    -fx-border-radius: 12;
    -fx-background-radius: 12;
    -fx-padding: 12 0 12 18;
    -fx-font-size: 20px;
    -fx-font-family: "Roboto", Arial, sans-serif;
    -fx-font-weight: 600;
    -fx-text-fill: #222;
    -fx-min-height: 56px;
    -fx-pref-height: 56px;
    -fx-alignment: center-left;
    -fx-cursor: hand;
    -fx-margin: 0 0 16 0;
}
.model-list-button:hover { -fx-background-color: #f5f5ff; }

.main-grid { -fx-hgap: 24; -fx-vgap: 24; -fx-padding: 32 32 32 32;
-fx-alignment: top-center; }
.tile-button {
    -fx-background-color: #1e0acb;
    -fx-background-radius: 16;
    -fx-border-radius: 16;
    -fx-cursor: hand;
    -fx-padding: 0;
```

```
    -fx-alignment: center;
    -fx-border-width: 0;
    -fx-font-size: 32px;
    -fx-font-family: "Roboto", Arial, sans-serif;
    -fx-font-weight: 600;
    -fx-text-fill: #fff;
    -fx-pref-width: 674px;
    -fx-pref-height: 600px;
}
.tile-button:hover { -fx-background-color: #17089f; }
.tile-button.tile-button-wide {
    -fx-pref-width: 1372px;
    -fx-pref-height: 500px;
}

.separator {
    -fx-background-color: #cdc9c9;
    -fx-pref-width: 1;
    -fx-min-width: 1;
    -fx-max-width: 1
}

.tile-svg {
    -fx-icon-color: #fff;
    -fx-icon-size: 256px;
}

.icon-placeholder {
    -fx-background-radius: 50;
    -fx-min-width: 256;
    -fx-min-height: 256;
    -fx-pref-width: 256;
    -fx-pref-height: 256;
    -fx-max-width: 256;
    -fx-max-height: 256;
    -fx-alignment: center;
    -fx-background-color: transparent;
}

.tile-text {
    -fx-text-fill: #fff;
    -fx-font-size: 32px;
    -fx-font-family: "Roboto", Arial, sans-serif;
    -fx-font-weight: 600;
    -fx-padding: 20 0 0 0;
    -fx-alignment: center;
}
```

К оформлению приложения применяются стили из main.css. Файл CSS описывает цвета, шрифты, отступы и эффекты при наведении курсора. В

частности, кнопки выхода, смены языка и запуска модели оформлены классом .exit-button, который задает фон, цвет текста и радиус скругления. Благодаря CSS можно быстро менять внешний вид элементов, не трогая логику работы.

Основное внимание в дизайне уделено простоте. Было принято решение избегать излишнего количества окон и переключателей, оставив только ключевые функции. Стадии процесса четко разделены: сначала выбор модели, затем ввод данных, после чего вывод результата. Такой подход снижает когнитивную нагрузку и делает работу с программой приятной.

```
Locale locale = I18n.getLocale();
ResourceBundle bundle = ResourceBundle.getBundle("messages",
locale);
FXMLLoader loader = new
FXMLLoader(SceneManager.class.getResource(ROOT + fxml));
```

Пользователь может настроить язык интерфейса. При переключении на другой язык все надписи обновляются автоматически. Это возможно благодаря используемой в JavaFX системе ресурсов, где каждое текстовое значение подставляется через ключ. Заранее был продумали способ расширения — в будущем можно добавить новые языки, просто создав файл ресурсов.

```
/**
 * Разбирает ответ сервера по модели AD-AS и возвращает краткое
 * текстовое изложение основных результатов.
 *
 * @param json исходные данные модели
 * @return человекочитаемое представление результатов
 */
@Override
public String parse(String json) {
    StringBuilder sb = new StringBuilder();
    JSONObject root = new JSONObject(json);

    JSONObject equilibriumBlock =
root.optJSONObject("equilibrium");
    if (equilibriumBlock != null) {
        sb.append(I18n.t("result.adas.title")).append("\n");
        JSONObject eqPoint =
equilibriumBlock.optJSONObject("equilibrium");
```

```
        if (eqPoint != null) {
            double x = eqPoint.optDouble("x", Double.NaN);
            double y = eqPoint.optDouble("y", Double.NaN);

sb.append(String.format(I18n.t("result.adas.equilibrium_point"), x,
y));
        }
        JSONArray lras = equilibriumBlock.optJSONArray("LRAS");
        if (lras != null && !lras.isEmpty()) {
            JSONObject lrasPt = lras.getJSONObject(0);
            sb.append(String.format(I18n.t("result.adas.lras"),
lrasPt.optDouble("x", Double.NaN)));
        }
    }

    JSONObject shifts = root.optJSONObject("shifts");
    if (shifts != null) {

sb.append("\n").append(I18n.t("result.adas.shifts_title")).append("
\n");
        JSONArray ad2 = shifts.optJSONArray("AD2");
        if (ad2 != null && !ad2.isEmpty()) {
            JSONObject ad2first = ad2.getJSONObject(0);

sb.append(String.format(I18n.t("result.adas.ad2_shift"),
ad2first.optDouble("x", Double.NaN), ad2first.optDouble("y",
Double.NaN)));
        }
        JSONArray sras2 = shifts.optJSONArray("SRAS2");
        if (sras2 != null && !sras2.isEmpty()) {
            JSONObject sras2first = sras2.getJSONObject(0);

sb.append(String.format(I18n.t("result.adas.sras2_shift"),
sras2first.optDouble("x", Double.NaN), sras2first.optDouble("y",
Double.NaN)));
        }
    }

    JSONObject gaps = root.optJSONObject("gaps");
    if (gaps != null) {

sb.append("\n").append(I18n.t("result.adas.gaps_title")).append("\n
");
        double potentialY = gaps.optDouble("potentialY",
Double.NaN);
        double actualY = gaps.optDouble("actualY", Double.NaN);

sb.append(String.format(I18n.t("result.adas.potential_output"),
potentialY));

sb.append(String.format(I18n.t("result.adas.actual_output"),
actualY));
```

```
            double gap = actualY - potentialY;
            sb.append(String.format(I18n.t("result.adas.gap"), gap));
        }

        if (sb.isEmpty()) {
            sb.append(I18n.t("result.adas.no_data"));
        }

        return sb.toString().trim();
    }
    /**
     * Построитель графиков модели совокупного спроса и предложения
    (AD-AS).
     * Поддерживает визуализацию равновесия, сдвигов кривых и расчёта
    разрывов.
     */
    public class ADASChartBuilder implements ChartDrawer {

        /**
         * Создаёт график в рамках модели AD-AS.
         *
         * @param chartKey  тип визуализации: {@code "equilibrium"},
    {@code "shifts"} или {@code "gaps"}
         * @param chartData исходные данные для построения
         * @return JavaFX-узел с графиком
         */
        @Override
        public Node buildChart(String chartKey, Map<String, Object>
    chartData) {

            Node node;
            switch (chartKey) {
                case "equilibrium":
                    node = buildEquilibriumChart(chartData);
                    break;
                case "shifts":
                    node = buildShiftsChart(chartData);
                    break;
                case "gaps":
                    node = buildGapsChart(chartData);
                    break;
                default:
                    Label lbl = new Label(I18n.t("chart.not_impl") +
    chartKey);
                    lbl.setStyle("-fx-text-fill: red;");
                    StackPane errorPane = new StackPane(lbl);
                    errorPane.setStyle("-fx-background-color: white; -
    fx-background-radius: 18; -fx-border-radius: 18; -fx-border-color:
    #fff;");
                    errorPane.setPadding(new Insets(16));
                    return errorPane;
            }
```

```
        if (node instanceof LineChart) {
            styleChart((LineChart<?, ?>) node);
        }

        return node;
    }

    private void styleChart(LineChart<?, ?> chart) {
        chart.setPrefWidth(760);
        chart.setPrefHeight(420);
        chart.setStyle("-fx-background-color: white; -fx-border-
color: transparent;");
        chart.setCreateSymbols(false);
        chart.applyCss();

        Node plotBackground = chart.lookup(".chart-plot-
background");
        if (plotBackground != null)
            plotBackground.setStyle("-fx-background-color:
white;");

        Node verticalGrid = chart.lookup(".chart-vertical-grid-
lines");
        if (verticalGrid != null)
            verticalGrid.setStyle("-fx-stroke: #ebebeb;");

        Node horizontalGrid = chart.lookup(".chart-horizontal-grid-
lines");
        if (horizontalGrid != null)
            horizontalGrid.setStyle("-fx-stroke: #ebebeb;");

        Node legend = chart.lookup(".chart-legend");
        if (legend != null)
            legend.setStyle("-fx-background-color: white; -fx-
border-color: #ededed; -fx-border-radius: 8; -fx-padding: 4 12 4
12;");

        chart.setPadding(new Insets(0, 0, 0, 0));
    }

    private Node buildEquilibriumChart(Map<String, Object>
chartData) {
        NumberAxis xAxis = new NumberAxis();
        NumberAxis yAxis = new NumberAxis();
        xAxis.setLabel(I18n.t("chart.output"));
        yAxis.setLabel(I18n.t("chart.price_level"));

        LineChart<Number, Number> chart = new LineChart<>(xAxis,
yAxis);
        chart.setTitle(I18n.t("chart.adas.equilibrium.title"));
        chart.setAnimated(false);
```

```
        addSeriesToChart(chart, chartData, "AD", "AD (" +
I18n.t("chart.demand") + ")");
        addSeriesToChart(chart, chartData, "SRAS", "SRAS (" +
I18n.t("chart.supply") + ")");
        addSeriesToChart(chart, chartData, "LRAS", "LRAS (" +
I18n.t("chart.supply") + ")");

        // Точка равновесия
        if (chartData.containsKey("equilibrium")) {
            Map<String, Number> eq = (Map<String, Number>)
chartData.get("equilibrium");
            Number x = eq.get("Y");
            Number y = eq.get("P");
            if (x != null && y != null) {
                XYChart.Series<Number, Number> eqPoint = new
XYChart.Series<>();
                eqPoint.setName(I18n.t("chart.equilibrium_point"));
                eqPoint.getData().add(new XYChart.Data<>(x, y));
                chart.getData().add(eqPoint);
            }
        }
        return chart;
    }

    private Node buildShiftsChart(Map<String, Object> chartData) {
        NumberAxis xAxis = new NumberAxis();
        NumberAxis yAxis = new NumberAxis();
        xAxis.setLabel(I18n.t("chart.output"));
        yAxis.setLabel(I18n.t("chart.price_level"));

        LineChart<Number, Number> chart = new LineChart<>(xAxis,
yAxis);
        chart.setTitle(I18n.t("chart.adas.shifts.title"));
        chart.setAnimated(false);

        addSeriesToChart(chart, chartData, "AD", "AD (" +
I18n.t("chart.before") + ")");
        addSeriesToChart(chart, chartData, "AD2", "AD (" +
I18n.t("chart.after") + ")");
        addSeriesToChart(chart, chartData, "SRAS", "SRAS (" +
I18n.t("chart.before") + ")");
        addSeriesToChart(chart, chartData, "SRAS2", "SRAS (" +
I18n.t("chart.after") + ")");
        addSeriesToChart(chart, chartData, "LRAS", "LRAS");

        return chart;
    }

    private Node buildGapsChart(Map<String, Object> chartData) {
        NumberAxis xAxis = new NumberAxis();
        NumberAxis yAxis = new NumberAxis();
        xAxis.setLabel(I18n.t("chart.output"));
```

```
        yAxis.setLabel(I18n.t("chart.price_level"));

        LineChart<Number, Number> chart = new LineChart<>(xAxis,
yAxis);
        chart.setTitle(I18n.t("chart.adas.gaps.title"));
        chart.setAnimated(false);

        addSeriesToChart(chart, chartData, "AD", "AD");
        addSeriesToChart(chart, chartData, "SRAS", "SRAS");
        addSeriesToChart(chart, chartData, "LRAS", "LRAS");

        if (chartData.containsKey("potentialY") &&
chartData.containsKey("actualY")) {
            double potentialY = ((Number)
chartData.get("potentialY")).doubleValue();
            double actualY = ((Number)
chartData.get("actualY")).doubleValue();
            double minP = yAxis.getLowerBound();
            double maxP = yAxis.getUpperBound();
            Rectangle gap = new Rectangle();
            gap.setX(Math.min(potentialY, actualY));
            gap.setY(minP);
            gap.setWidth(Math.abs(potentialY - actualY));
            gap.setHeight(maxP - minP);
            gap.setFill(Color.rgb(255, 150, 150, 0.3));
            chart.setTitle(chart.getTitle() + String.format("
(Разрыв: %.2f)", actualY - potentialY));
        }
        return chart;
    }

    private void addSeriesToChart(LineChart<Number, Number> chart,
Map<String, Object> data, String key, String title) {
        if (data.containsKey(key)) {
            List<Map<String, Number>> points = (List<Map<String,
Number>>) data.get(key);
            XYChart.Series<Number, Number> series = new
XYChart.Series<>();
            series.setName(title);
            for (Map<String, Number> point : points) {
                Number x = point.get("x");
                Number y = point.get("y");
                if (x != null && y != null) {
                    series.getData().add(new XYChart.Data<>(x, y));
                }
            }
            chart.getData().add(series);
        }
    }
}
```

Для повышения читабельности результатов применяются таблицы и графики. Результаты при помощи парсинга приводятся в читаемый вид, а динамические графики строятся на основе библиотек JavaFX. Парсеры и построение графиков реализованы по паттерну «фабрика», для унификации программного кода и удобства его организации. Визуальное представление помогает лучше понять структуру и динамику модели, что особенно важно при анализе экономических процессов.

```
    <Label fx:id="titleLabel" text="%llm.chat" styleClass="llm-
chat-title"/>

    <ScrollPane fx:id="chatScroll" fitToWidth="true"
prefHeight="250" styleClass="llm-chat-scroll">

        <VBox fx:id="chatArea" spacing="5" styleClass="llm-chat-
area"/>

    </ScrollPane>

    <HBox spacing="5" styleClass="llm-chat-input-row">

        <TextField fx:id="inputField" promptText="%llm.prompt"
HBox.hgrow="ALWAYS" styleClass="llm-chat-input"/>

        <VBox fx:id="sendButton" styleClass="llm-chat-send-btn">
            <FontIcon iconLiteral="fa-send-o" styleClass="tile-
svg"/>
        </VBox>

    </HBox>
</VBox>
```

Одним из оригинальных решений была интеграция LLM-подсказок в отчеты. Пользователь может запросить пояснения по полученным результатам, и сервер отправляет сообщения языковой модели, сохраняемые в reports. Эти сообщения отображаются в клиентском интерфейсе и могут быть локализованы, а также сохранены в отчет. Дизайн предусматривает область для таких пояснений, располагая их рядом с основными результатами.

3.5 Подключение вспомогательных решений

При разработке программы, было задействовано несколько внешних технологий, необходимых для реализации вышеизложенного функционала. Такими технологиями стали:

– Redis, используемый для распределенного кеширования;

– Minio, выступающий в роли локального S3 хранилища файлов;

– Ollama, являющаяся средой запуска LLM,

– Сервис smtp, предоставляемый Gmail.

Для подключения к redis необходимо указать следующие настройки в app.properties сервера.

```
spring.cache.type=redis
# Адрес и порт сервера Redis
spring.data.redis.host=127.0.0.1
spring.data.redis.port=6379
#spring.data.redis.port=6380
# Таймаут подключения к Redis (например, 2 секунды)
spring.data.redis.timeout=2s
```

После чего redis необходимо установить и развернуть на сервере.

Для установки redis на сервер его необходимо загрузить командами:

– apt update && sudo apt upgrade -y;

– apt install redis-server -y;

– redis-server.

При использовании minio необходимо так же развернуть его при помощи следующих команд:

– wget https://dl.min.io/server/minio/release/linux-amd64/minio;

– chmod +x minio;

– mv minio /usr/local/bin/;

– mkdir ~/minio-data;

– MINIO_ROOT_USER=minioadmin

MINIO_ROOT_PASSWORD=minioadmin123 minio server ~/minio-data --console-address ":9001".

После чего нужно зайти в панель администрирования minio, расположенную по адресу localhost:9001, и создать бакеты documents и reports.

```
minio.url=http://localhost:9000
minio.access-key=minioadmin
minio.secret-key=minioadmin123
minio.bucket=documents
minio.bucket.reports=reports
```

Для подключения к серверу используются эти настройки в app.properties.

Если сервер устанавливается локально, то для работы redis и minio необходимо развернуть среду wsl.

Для использования ollama достаточно просто загрузить исполняемый файл программы с официального ресурса, и установить в его среду необходимую модель командой ollama pull mistral.

```
llm.ollama.host=http://localhost:11434
llm.ollama.model=mistral
llm.ollama.timeout=30s
llm.ollama.retry-on-parse-error=true
llm.ollama.max-retries=10
```

Эти настройки позволят подключаться серверу к модели, и отправлять ей запросы.

Для использования smtp необходимо включить функцию smtp в настройках Gmail, после чего задать настройки для отправки писем.

```
# SMTP-сервер для отправки email (Gmail)
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=optimushd25@gmail.com
```

4 Эксплуатационная часть

В этой части описываются основные вопросы эксплуатации программного комплекса, включая организацию тестирования, сопровождение и руководство для различных категорий пользователей. Сначала приводятся примеры ручного тестирования, где демонстрируются основные пользовательские сценарии, а также отмечаются найденные и исправленные ошибки. Затем рассматриваются результаты автоматизированных unit- и интеграционных тестов, описываются подходы к автоматической проверке корректности работы системы и приводятся итоги тестовых прогонов.

4.1 Ручное тестирование

Ручное тестирование является одним из ключевых этапов проверки работоспособности программного комплекса перед его внедрением и эксплуатацией. Данный процесс позволяет выявить ошибки, которые могли остаться незамеченными на этапах автоматизированной проверки, а также оценить удобство использования программы с точки зрения конечного пользователя. В ходе ручного тестирования были последовательно воспроизведены типовые сценарии работы: регистрация и вход в систему, загрузка документов, проведение расчетов моделей, формирование отчетов и смена языка интерфейса. Такой подход позволяет не только убедиться в стабильности функционирования всех основных компонентов, но и выявить возможные недостатки пользовательского интерфейса, логики обработки ошибок или взаимодействия с внешними сервисами. Результаты ручного тестирования становятся основой для дальнейшей доработки системы и повышения качества.

Тестирование авторизации.

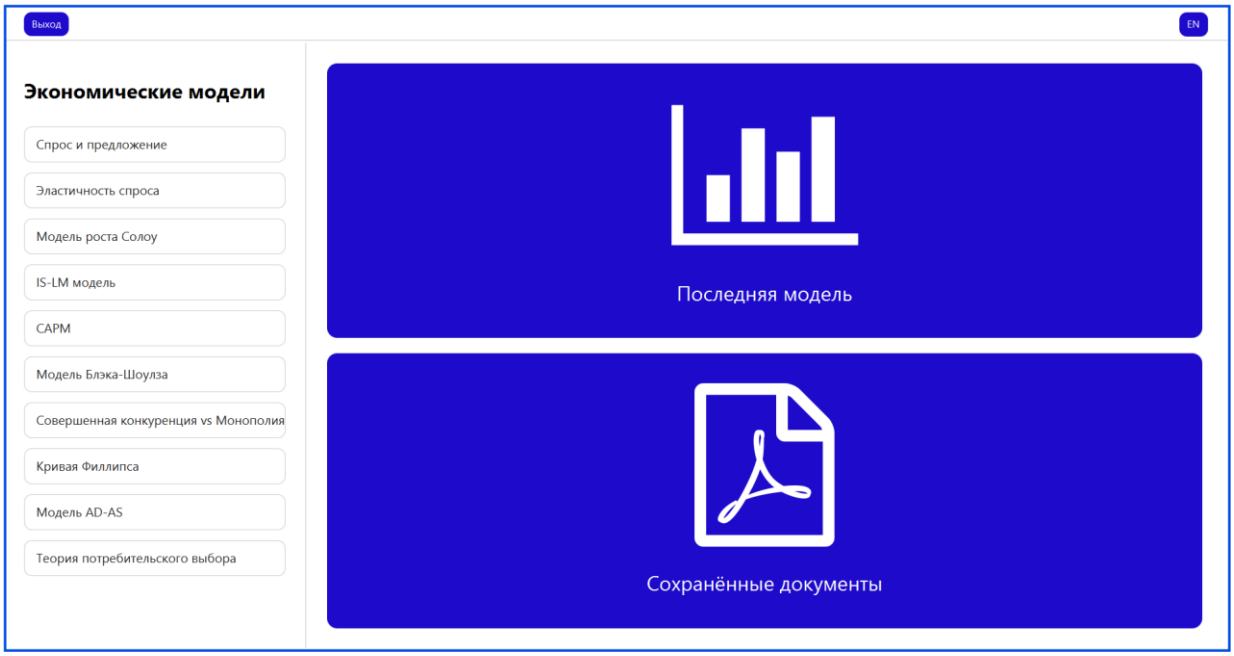

Рисунок 23 – Тестирование автризации

Пользователь вводит данные авторизации, после чего успешно попадает на основной экран. Результат тестирования изображен на рисунке 23.

Тестирование загрузки документов.

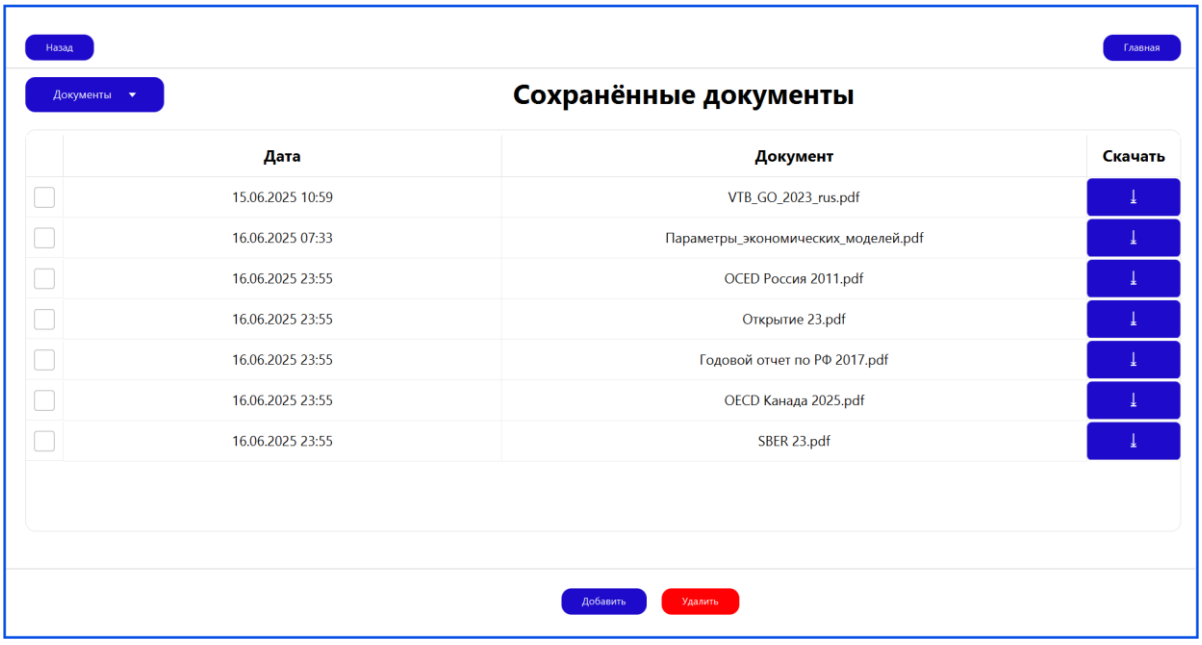

Рисунок 24 – Тестирование загрузки документов

Документ успешно загружается в систему, после чего отображается в таблице документов. Результат тестирования изображен на рисунке 24.

Тестирование извлечения данных из документа.

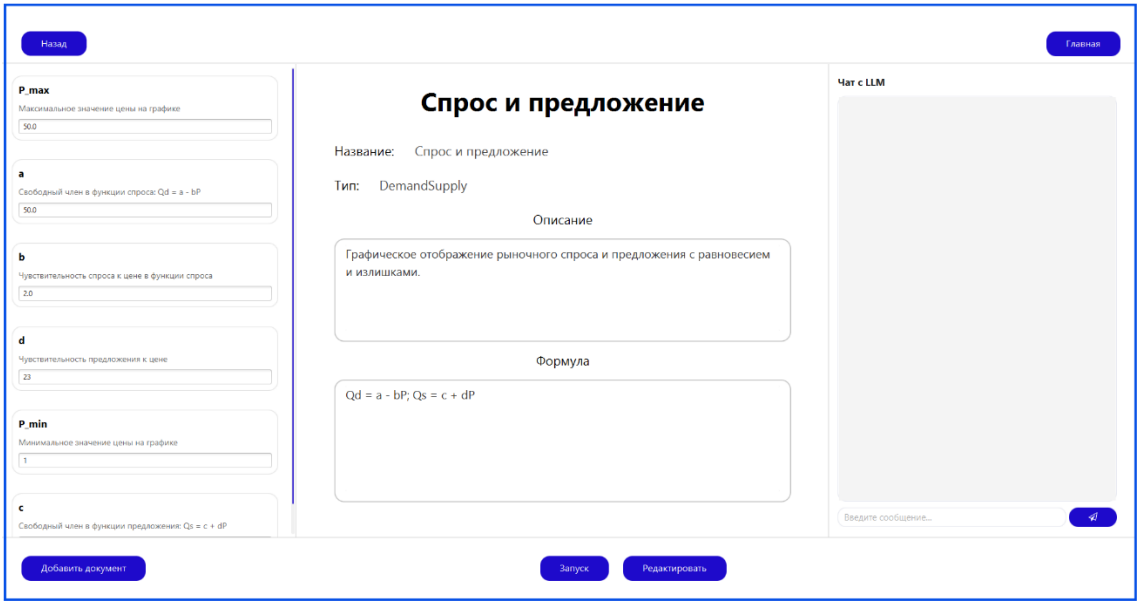

Рисунок 25 – Тестирование извлечения данных из документа

Пользователь успешно выбрал желаемый документ, после чего вернулся на экран просмотра модели, а значения параметров были подставлены из документа. Результат тестирования изображен на рисунке 25.

Тестирование расчета модели.

Рисунок 26 – Тестирование расчета модели

При запуске расчета сервер успешно вернул результаты, а клиент построил все требуемые виды визуализации. Так же клиент произвел парсиннг результата вычислений, переформатировав его в удобный читаемый вид. Результат тестирования изображен на рисунке 26.

Тестирование чата с LLM.

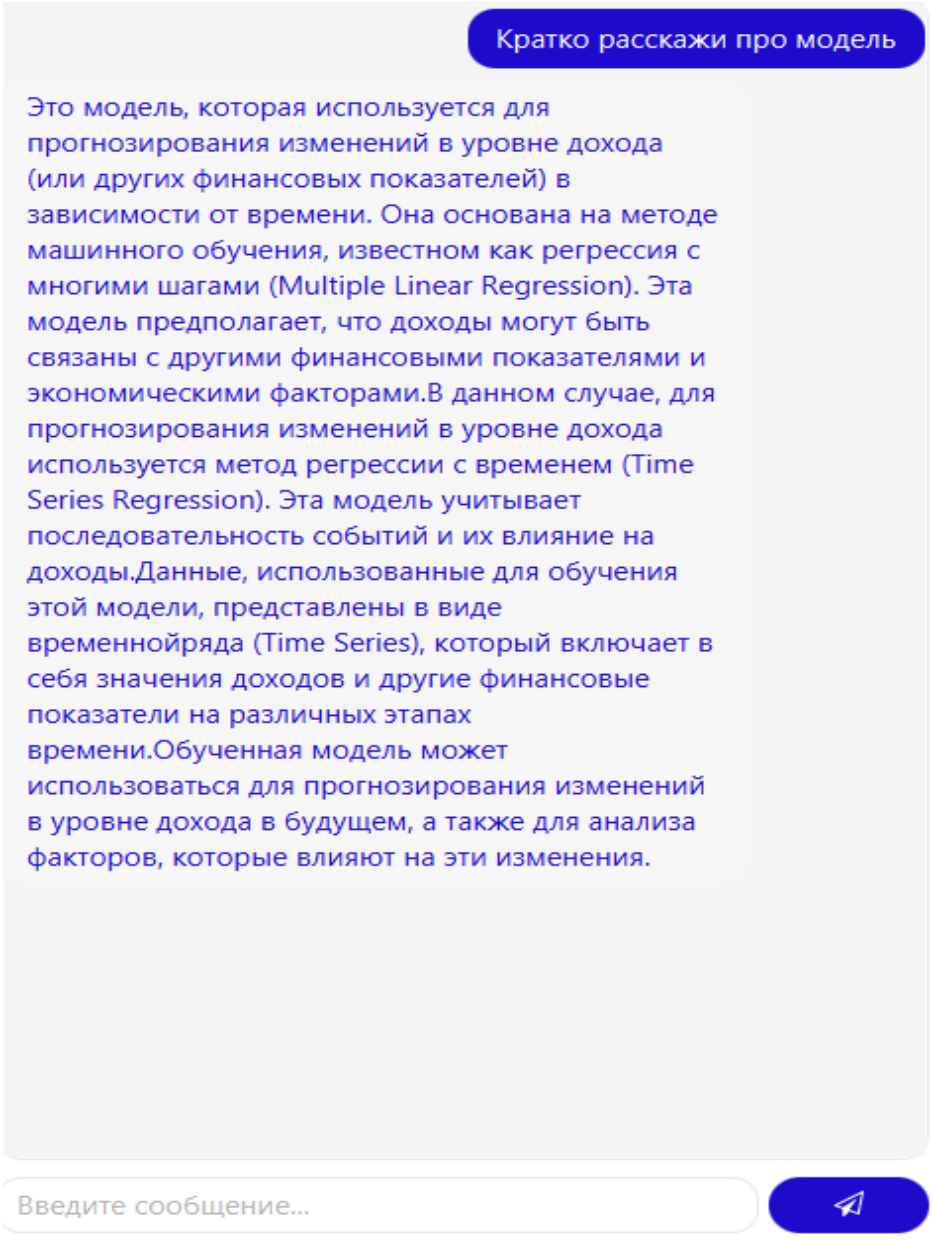

Рисунок 27 – Тестирование чата с LLM

Модель успешно дала ответ на вопрос. Ответ является осмысленным, и соответствует теме запроса и текущей модели. Результат тестирования изображен на рисунке 27.

Тестирование формирования отчета.

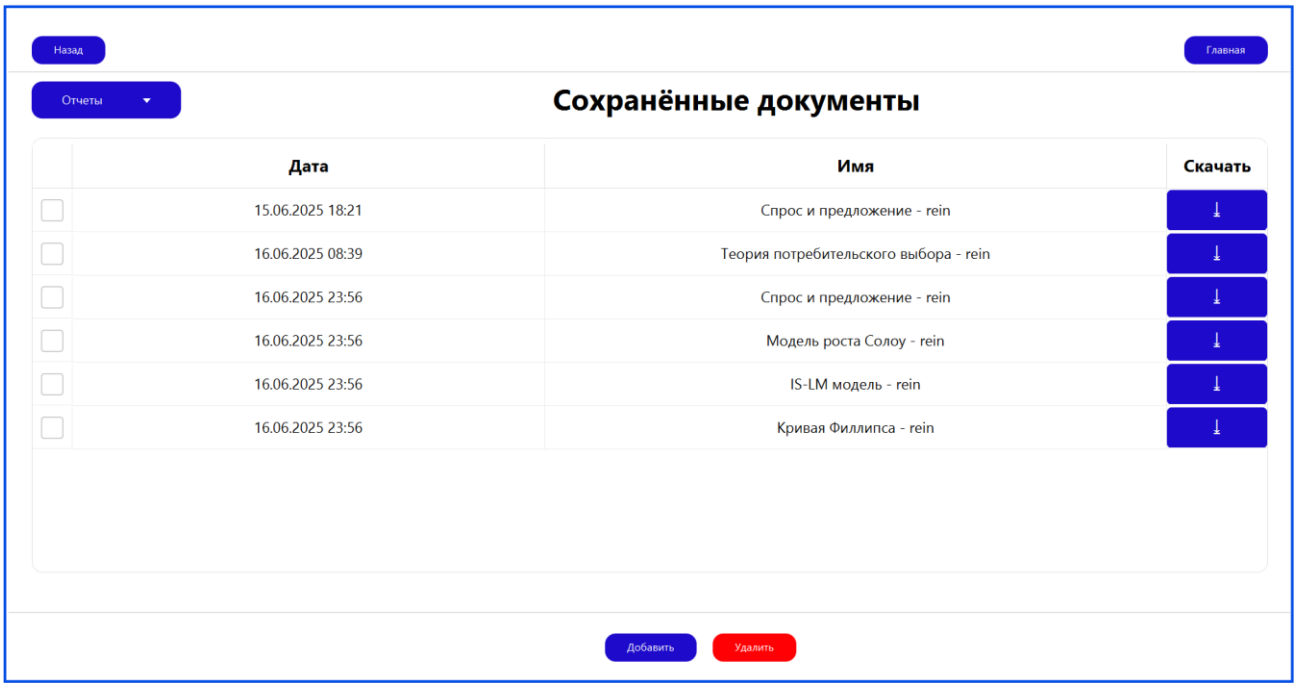

Рисунок 28 – Тестирование формирования отчета

Отчет успешно сформирован. Он содержит всю необходимую информацию. Результат тестирования изображен на рисунке 28.

В результате ручного тестирования было подтверждено корректное функционирование всех основных компонентов программного комплекса и их взаимодействия друг с другом. Пользовательские сценарии, включающие работу с документами, расчетами моделей и формированием отчетов, были успешно выполнены без критических сбоев или потери данных.

В процессе проверки удалось выявить и своевременно устранить ряд незначительных недочетов, связанных в основном с визуальным отображением отдельных элементов и обработкой нестандартных ситуаций. В целом, по итогам тестирования система продемонстрировала достаточную стабильность и готовность к дальнейшей эксплуатации.

4.2 Автоматическое тестирование

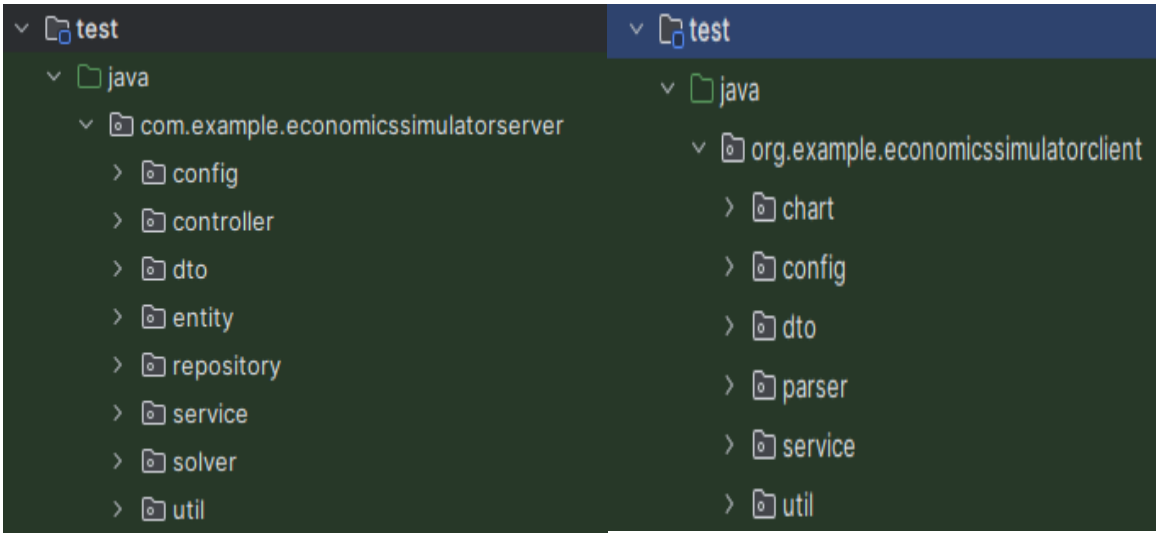

Рисунок 29 – Структура тестов

В проекте выделены отдельные каталоги с тестами для клиента и сервера, что изображено на рисунке 30. Все тестовые классы используют JUnit 5, при необходимости подключается Mockito и Spring Boot Test. Для клиента тесты лежат в нескольких подпакетах, проверяющих функциональность DTO, парсеров, сервисов и утилит. На сервере отдельные тесты охватывают контроллеры, конфигурацию, сущности, репозитории и бизнес-логику.

Тесты клиента организованы следующим образом: в каталоге src/test/java/org/example/economicssimulatorclient присутствуют шесть основных подпакетов. В них находятся тесты для визуализации графиков, проверки конфигурационных классов, сериализации/десериализации DTO, корректной работы парсеров результатов, сервисов и различных утилит. Несмотря на то что тестов меньше, чем производственного кода, они охватывают ключевые сценарии пользовательского интерфейса и сетевого взаимодействия.

```
@Test
void testRecordFieldsAndEquality() {
    ModelParameterDto param = new ModelParameterDto(1L, 1L, "a",
"double", "42", "A", "desc", 1);
    CalculationRequestDto dto1 = new CalculationRequestDto(1L,
```

```
"model", List.of(param));
    CalculationRequestDto dto2 = new CalculationRequestDto(1L,
"model", List.of(param));

    assertThat(dto1.modelId()).isEqualTo(1L);
    assertThat(dto1.modelType()).isEqualTo("model");
    assertThat(dto1.parameters()).containsExactly(param);

    assertThat(dto1).isEqualTo(dto2);
    assertThat(dto1.hashCode()).isEqualTo(dto2.hashCode());
    assertThat(dto1.toString()).contains("modelId=1",
"modelType=model");
}
```

Проверка DTO. Большинство тестов на стороне клиента посвящены DTO. Они проверяют корректность геттеров, equals/hashCode, а также работу методов toString. Такие тесты обеспечивают стабильность моделей данных и предотвращают ошибки при обмене информацией с сервером.

```
@Test
void returnsNoDataIfNothingPresent() {
    var parser = new CAPMResultParser();
    String json = "{}";
    String result = parser.parse(json);
    assertThat(result).contains("No CAPM data to display.");
}

@Test
void parsesSMLFrontierAndDecomposition() {
    var parser = new CAPMResultParser();
    String json = """
    {
      "sml": { "sml": [ {"risk": 0.1, "return": 0.05}, {"risk":
0.9, "return": 0.25} ] },
      "efficient_frontier": { "portfolios": [ {"risk": 0.2,
"return": 0.07}, {"risk": 0.5, "return": 0.14} ] },
      "decomposition": { "decomposition": [ {"label": "A", "alpha":
0.1, "beta": 0.2} ] }
    }
    """;
    String result = parser.parse(json);
    assertThat(result).contains("Security Market
Line").contains("Efficient frontier").contains("decomposition");
}
```

Парсеры результатов. Отдельные классы парсеров разбирают текстовые или JSON-ответы от сервера. Тесты создают примерные входные данные и проверяют, что итоговые объекты содержат нужные поля и не теряют данные. Это критично для отображения результатов вычислений и построения графиков.

```
@Test
void getAllModels_returnsList() throws Exception {
    EconomicModelDto[] models = {
            new EconomicModelDto(
                    1L,
                    "DemandSupply",
                    "Test model",
                    "Description",
                    List.of(
                            new ModelParameterDto(1L, 1L, "a",
"double", "100", "A", "desc", 1)
                    ),
                    List.of(
                            new ModelResultDto(1L, 1L, "output",
"{\"value\":42}", "2024-06-13T12:00:00")
                    ),
                    "2024-06-13T11:00:00",
                    "2024-06-13T12:00:00",
                    "Q = a - bP"
            )
    };
    doReturn(models).when(service)
            .get(any(), eq(""), eq(EconomicModelDto[].class),
anyBoolean(), any());

    List<EconomicModelDto> result = service.getAllModels();
    assertEquals(1, result.size());
    EconomicModelDto model = result.get(0);
    assertEquals("Test model", model.name());
    assertEquals("DemandSupply", model.modelType());
    assertNotNull(model.parameters());
    assertEquals("A", model.parameters().get(0).displayName());
    assertNotNull(model.results());
    assertEquals("output", model.results().get(0).resultType());
}
```

Сервисы на клиенте. Для сервисов используются Mockito и MockWebServer. Тесты имитируют вызовы REST-API сервера, проверяя, как клиентские сервисы формируют запросы и обрабатывают ответы. Таким

образом, гарантируется корректное взаимодействие с сервером даже без запуска реального backend.

```
@Test
void toJsonAndFromJsonRoundtrip() throws Exception {
    ApiResponse resp = new ApiResponse(true, "message");
    String json = JsonUtil.toJson(resp);
    assertNotNull(json);
    assertTrue(json.contains("message"));

    ApiResponse resp2 = JsonUtil.fromJson(json, ApiResponse.class);
    assertEquals(resp, resp2);
}
```

Утилитарные классы. В пакет util входят тесты вспомогательных функций. Они проверяют преобразование данных, управление пользовательскими сессиями, конфигурацию HTTP-клиента и международизацию. Поскольку эти классы активно используются в разных частях приложения, их стабильность повышает надёжность всей клиентской части.

Итоги клиентского покрытия. В каталоге исходников содержится примерно 80 классов, а тестовых классов — около 50. Соотношение близко к 60 %, что позволяет покрыть основной функционал.

Тесты сервера покрывают ключевые узлы взаимодействия склиентами, а так же большую часть бизнес-логики.

```
@Test
@DisplayName("POST /auth/register - successful registration")
void testRegister() throws Exception {

Mockito.when(authService.register(any(RegistrationRequest.class)))
            .thenReturn(new ApiResponse(true,
"msg.verification_code_sent"));

    mockMvc.perform(post("/auth/register")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content("""

{"username":"testuser","email":"test@email.com","password":"pass123
"}
                                """))
```

```
            .andExpect(status().isCreated())
            .andExpect(jsonPath("$.success").value(true))

.andExpect(jsonPath("$.message").value("msg.verification_code_sent"
));
}
```

В пакете controller находятся интеграционные тесты с использованием MockMvc, которые проверяют REST-эндпойнты без фильтров безопасности. Вызовы к сервисам подменяются Mockito, чтобы сфокусироваться на корректности HTTP-ответов.

```
@Test
void testJavaMailSender() {
    MailProperties props = new MailProperties();
    props.setHost("smtp.example.com");
    props.setPort(2525);
    props.setUsername("user");
    props.setPassword("pass");

    MailConfig config = new MailConfig(props);
    JavaMailSender sender = config.javaMailSender();

    assertThat(sender).isNotNull();
}
```

Тесты конфигурации. В пакете config проверяются классы настройки безопасности, почтового сервиса, локализации и JWT. Такие тесты помогают убедиться, что приложение правильно инициализирует необходимые компоненты и использует верные параметры конфигурации.

```
@Test
void builderAndAllArgsConstructorAndSetters() {
    LocalDateTime now = LocalDateTime.now();
    EconomicModel model = EconomicModel.builder()
            .id(10L)
            .modelType("macro")
            .name("IS-LM")
            .description("desc")
            .parameters(List.of())
            .results(List.of())
            .createdAt(now)
```

```
                .updatedAt(now)
                .formula("x+y=z")
                .build();

        assertThat(model.getId()).isEqualTo(10L);
        assertThat(model.getModelType()).isEqualTo("macro");
        assertThat(model.getName()).isEqualTo("IS-LM");
        assertThat(model.getDescription()).isEqualTo("desc");
        assertThat(model.getParameters()).isEmpty();
        assertThat(model.getResults()).isEmpty();
        assertThat(model.getCreatedAt()).isEqualTo(now);
        assertThat(model.getUpdatedAt()).isEqualTo(now);
        assertThat(model.getFormula()).isEqualTo("x+y=z");
    }
```

Тесты сущностей и репозиториев. В подпакете entity содержатся простые тесты для JPA-сущностей: проверяется установка и получение полей, корректность equals/hashCode. В репозиториях используются аннотации @DataJpaTest, что позволяет запускать тесты на встроенной базе H2 и проверять операции сохранения и поиска без полноценного сервера БД.

```
@Test
void uploadDocument_nonPdf_throws() {
    when(multipartFile.getContentType()).thenReturn("image/png");
    when(multipartFile.getOriginalFilename()).thenReturn("b.png");
    assertThatThrownBy(() -> service.uploadDocument(1L,
multipartFile, "desc"))
            .isInstanceOf(LocalizedException.class);
}

@Test
void getById_notFound_throws() {

when(documentRepository.findById(55L)).thenReturn(Optional.empty())
;
    assertThatThrownBy(() ->
service.getById(55L)).isInstanceOf(LocalizedException.class);
}
```

Сервисы сервера покрыты юнит-тестами с использованием Mockito. Имитация зависимостей (например, почтового сервиса или менеджера аутентификации) даёт возможность протестировать логику регистрации, входа,

работы с документами и отчётами. Обработка исключений и успешные сценарии в каждом методе проверяются отдельно.

```
@Test
void solve_returnsValidResult() {
    CalculationRequestDto req = new CalculationRequestDto(
            1L,
            "ADAS",
            List.of(
                    new ModelParameterDto(1L, 1L, "Y_pot",
"double", "1000", "Potential Y", null, null),
                    new ModelParameterDto(2L, 1L, "P0", "double",
"100", "Initial Price", null, null),
                    new ModelParameterDto(3L, 1L, "AD_slope",
"double", "-3", "AD Slope", null, null),
                    new ModelParameterDto(4L, 1L, "AS_slope",
"double", "2", "AS Slope", null, null),
                    new ModelParameterDto(5L, 1L, "shock",
"double", "50", "Shock", null, null)
            )
    );
    CalculationResponseDto resp = solver.solve(req);
    assertThat(resp).isNotNull();
    assertThat(resp.result()).isNotNull();
    assertThat(resp.result().resultType()).isEqualTo("chart");
    assertThat(resp.result().resultData()).contains("equilibrium");
    assertThat(resp.updatedParameters()).hasSize(5);
}
```

Тесты решателей моделей. Пакет solver включает тесты для каждой экономической модели (ISLM, ADAS, Phillips Curve и другие). В них создаются примеры параметров, вызывается решатель, и проверяется, что расчёты не падают и возвращают ожидаемую структуру данных. Это важно для корректной работы аналитической части приложения.

```
@Test
void testGenerateAndValidateToken() {
    User user = new User("testuser", "pass", java.util.List.of());
    String token = jwtUtil.generateToken(user);
    assertThat(jwtUtil.isTokenValid(token, user)).isTrue();

assertThat(jwtUtil.extractUsername(token)).isEqualTo("testuser");
}
```

```
@Test
void testExtraClaims() {
    User user = new User("john", "pass", java.util.List.of());
    String token = jwtUtil.generateToken(user, Map.of("role",
"ADMIN"));
    assertThat(jwtUtil.isTokenValid(token, user)).isTrue();
}
```

Утилиты и JWT. Тесты пакета util проверяют вспомогательные методы, например создание JWT-токенов и их валидацию. Благодаря этим проверкам можно полагаться на стабильность механизма авторизации и прав доступа.

Проект стремится покрыть тестами ключевые элементы: модели данных, REST-эндпойнты, репозитории и алгоритмические модули. Тесты присутствуют как на уровне единичных классов (юнит-тестирование), так и на уровне взаимодействия контроллеров с моками сервисов.

Использование аннотаций Spring Boot для тестов упрощает конфигурацию и изоляцию компонентов. Mockito позволяет легко подменять зависимости, а H2 обеспечивает быструю проверку JPA-операций. Благодаря этому разработчики могут быстро запускать тесты без необходимости поднимать полный сервер или внешние сервисы.

Суммарно для клиента и сервера написано 285 тестов. Все тесты проходят успешно, из чего можно сделать вывод, что проект готов к эксплуатации.

4.3 Руководство администратора

Раздел предназначен для системных администраторов, которые занимаются развертыванием, настройкой и поддержкой серверной части информационной системы. Здесь подробно описываются базовые действия по подготовке и конфигурации виртуального выделенного сервера (VDS), базы данных PostgreSql, средств кеширования, LLM модели и s3 хранилища.

Для стабильной работы системы рекомендуется использовать сервер с параметрами, обеспечивающими производительность и безопасность. Оптимально:

– ОС Linux (Ubuntu 20.04 LTS, CentOS 8 и др.);

– минимум 4 ядра CPU с частотой не ниже 2.5 ГГц;

– ОЗУ не менее 6 ГБ;

– дисковое пространство от 20 ГБ, предпочтительно SSD;

– графический ускоритель с объемом видеопамяти не меньше 8ГБ;

– открытые порты: 8080 (Spring), 9000 (minio), 6379 (redis), 11434 (ollama);

– установленные JDK 21+ и PostgreSQL 16.0+.

Первостепенно, необходимо установить вспомогательное ПО.

Для установки redis необходимо выполнить этот набор команд:

– sudo apt update;

– sudo apt install redis-server -y.

После чего настроить запуск в качестве службы:

– sudo nano /etc/redis/redis.conf;

– supervised systemd.

Следующий этап – перезапуск redis:

– sudo systemctl restart redis.service;

– sudo systemctl enable redis;

– sudo systemctl status redis.

Теперь redis будет доступен в качестве провайдера кеша для сервера.

Для установки ollama необходимо выполнить этот набор команд:

– curl -fsSL https://ollama.com/install.sh | sh;

– ollama pull mistral.

Для установки minio необходимо выполнить этот набор команд:

– wget https://dl.min.io/server/minio/release/linux-amd64/minio;

– chmod +x minio;

– sudo mv minio /usr/local/bin.

После чего необходимо настроить minio в режиме службы, для этого нужно создать пользователя:

– sudo useradd -r minio-user -s /sbin/nologin;

– sudo mkdir /usr/local/share/minio;

– sudo mkdir /etc/minio;

– sudo chown minio-user:minio-user /usr/local/share/minio;

– sudo chown minio-user:minio-user /etc/minio.

Далее необходимо настроить конфигурацию, путем создания конфигурационного файла по команде sudo nano/etc/systemd/system/minio.service. В него необходимо поместить следующее:

```
[Unit]
Description=MinIO
Wants=network-online.target
After=network-online.target
[Service]
User=minio-user
Group=minio-user
ExecStart=/usr/local/bin/minio server /usr/local/share/minio --
console-address ":9001"
Environment="MINIO_ROOT_USER=minioadmin"
Environment="MINIO_ROOT_PASSWORD=minioadmin123"
Restart=always
LimitNOFILE=65536
[Install]
WantedBy=multi-user.target
```

После сохранение файла конфигурации нужно запустить службу этим набором команд:

– sudo systemctl daemon-reexec;

– sudo systemctl daemon-reload;

– sudo systemctl enable minio;

– sudo systemctl start minio.

Следующим этапом будет развертывание базы данных на сервере.

Для этого необходимо ее установить при помощи следующих команд:

– sudo apt update;

– sudo apt install postgresql postgresql-contrib -y.

По завершению установки необходимо создать базу данных, и импортировать ее из дампа. Переходим в консоль postgreSql через команду sudo -u postgres psql, где создаем базу данных.

```
CREATE USER postgres WITH PASSWORD masterkey;
CREATE DATABASE economicsdb OWNER postgres;
```

Импортируем бекап в базу данных посредством команды psql -U postgres -d economicsdb -f /home/user/economicsdb.sql.

Последним этапом развертывания станет запуск сервера путем запуска исполняемого .jar файлы в фоновом режиме командой nohup java -jar EconomicsSimulatorServer.jar > server.log 2>&1 &.

Для корректности работы так же необходимо убедится что брандмауер пропускает соединения по необходимым портам. Для этого нужно ввести следующий набор команд:

– sudo ufw allow ssh;

– sudo ufw allow 8080/tcp;

– sudo ufw allow 9000,9001/tcp;

– sudo ufw allow 11434/tcp;

– sudo ufw enable.

Чек-лист перед вводом в эксплуатацию:

– ОС обновлена, установлены PostgreSQL, Redis, Ollama, Minio;

– сервер слушает порт 8080;

– redis слушает порт 6379;

– ollama слушает порт 11434;

– minio слушает порт 9000

– spring Boot сервер запущен в фоновм режиме и работает стабильно.

В заключение, данный раздел предоставляет системному администратору комплексное руководство по развёртыванию и первичной настройке всех необходимых компонентов серверной инфраструктуры информационной системы. Четкое следование перечисленным этапам позволяет обеспечить работоспособность каждого из используемых сервисов, гарантирует корректное взаимодействие между ними и формирует основу для дальнейшей стабильной эксплуатации. Реализация всех указанных настроек и проверка соответствия параметров серверного окружения позволяют избежать распространённых ошибок и упростить процесс поддержки, что особенно важно при внедрении системы в учебные или производственные среды.

4.4 Руководство программиста

В данном разделе рассматриваются ключевые особенности структуры клиентской и серверной частей программного комплекса, а также их основные архитектурные принципы. Описывается организация взаимодействия между различными модулями, подходы к обеспечению безопасности, локализации и хранению пользовательских данных. Основное внимание уделено логике построения приложения, разделению бизнес-функций и поддержанию модульности кода, что позволяет гарантировать надежность, масштабируемость и удобство сопровождения всей системы. Такой подход обеспечивает гибкую интеграцию новых возможностей, а также стабильную работу всех компонентов комплекса как в учебных, так и в производственных условиях.

Клиент «Экономического симулятора» реализован на JavaFX. Его стартовая точка — класс MainApp, где происходит инициализация JavaFX и выбор начальной сцены. FXML-файлы лежат в каталоге /org/example/economicssimulatorclient/ и загружаются через SceneManager, который кэширует созданные сцены. При запуске приложение проверяет наличие сохранённого токена и решает, показывать ли форму входа или сразу открывать главное окно.

Все контроллеры наследуют функциональность BaseController. Этот абстрактный класс предоставляет методы для локализации сообщений, отображения статусов и запуска фоновых задач. В нём реализован механизм сохранения и получения одноэкземплярных сервисов, а также очистка полей формы при необходимости.

За локализацию отвечает класс I18n, который хранит текущую локаль и соответствующий ей файл сообщений. Через его метод t можно получить перевод по ключу, либо сам ключ восклицательными знаками, если перевода нет.

Состояние сессии пользователя ведётся в SessionManager. Токены сохраняются в java.util.prefs.Preferences, а также имеется флаг, показывающий, что пользователь только что вышел из системы. Менеджер позволяет сохранять, получать и удалять токены, а также подменять экземпляр для тестов.

Взаимодействие с сервером реализовано через сервисы. AuthService обрабатывает регистрацию, подтверждение почты, вход и обновление токенов. В случае ошибки выводится локализованное сообщение, извлечённое из ответа сервера. После успешного входа токены сохраняются в SessionManager и сбрасывается признак выхода.

Помимо аутентификации, клиент содержит EconomicModelService, DocumentService и другие сервисы для работы с экономическими моделями, документами и отчётами. Каждый сервис наследует BaseService, что позволяет использовать общие методы для отправки HTTP-запросов и обработки ошибок.

FXML-представления и их контроллеры обеспечивают ввод параметров моделей, просмотр документов и взаимодействие с системой. Например, существуют формы регистрации, авторизации, изменения пароля и основные окна, отображающие результаты расчётов и графики.

Для построения графиков используется вспомогательный класс ChartDataConverter. Он преобразует данные, полученные от сервера, в структуру, подходящую для компонентов JavaFX. Утилиты отвечают за валидацию параметров, сериализацию JSON и хранение последних параметров модели.

Клиент также содержит компонент для общения с языковой моделью, отображаемый на экранах просмотра модели и результата. Эта часть опирается на сервис LlmService, который отправляет сообщения на сервер и выводит ответы в интерфейсе пользователя.

Сервер построен на Spring Boot. Главный класс EconomicsSimulatorServerApplication включает аннотацию @EnableCaching и запускает приложение через SpringApplication.run.

Безопасность реализована через конфигурацию SecurityConfig. Здесь определены правила доступа и фильтр, проверяющий JWT в каждом запросе. Если токен валиден, в контекст безопасности помещается аутентифицированный пользователь. Для неаутентифицированных запросов, кроме /auth/**, доступ закрыт. Настройка отключает CSRF, включает stateless-сессии и подключает фильтр аутентификации перед UsernamePasswordAuthenticationFilter.

Контроллеры REST описаны подробными Javadoc. Например, DocumentController позволяет загружать и скачивать PDF-документы пользователя. Для получения файла проверяется наличие документа у пользователя, а ответ формируется через InputStreamResource с корректными HTTP-заголовками.

Слой обработчиков исключений представлен классом ApiExceptionHandler. Он преобразует исключения в локализованные ответы типа ApiResponse. Для кастомных исключений LocalizedException используется код сообщения, находящийся в ресурсах локализации. При валидационных ошибках собираются все сообщения и объединяются в одну строку. Непредусмотренные исключения приводят к ответу со статусом 500 и стандартным кодом ошибки.

Само LocalizedException служит базовым unchecked-исключением с поддержкой локализации. В нём сохраняются код сообщения и параметры для подстановки. Этот код также используется в качестве текста исключения по умолчанию, что позволяет быстро вернуть пользователю локализованную ошибку в обработчике.

Бизнес-логика разделена на сервисы. EconomicModelService управляет экономическими моделями и их параметрами. Он может возвращать все модели с параметрами пользователя, загружать отдельную модель и преобразовывать сущности в DTO. Методы помечены аннотациями @Transactional и @Cacheable для оптимизации работы и сохранения целостности данных.

Репозитории, такие как UserRepository, обеспечивают доступ к данным в базе. Для каждого типа сущности создан свой интерфейс, расширяющий JpaRepository. В них присутствуют методы поиска по имени пользователя, email и проверка существования, что помогает валидировать запросы пользователя.

Отдельным слоем выделены решатели экономических моделей (solvers). К примеру, PhillipsCurveSolver рассчитывает кривую Филлипса, формирует данные для графиков и возвращает их в виде DTO. Colверы реализуют интерфейс EconomicModelSolver и помечены @Component, чтобы Spring мог их автоматически обнаружить.

Сервер взаимодействует с клиентом через DTO-классы, описывающие запросы и ответы. Они передают параметры расчётов, результаты, идентификаторы сущностей и текстовые сообщения. DTO позволяют изолировать внутренние модели БД от внешнего API и гарантировать стабильность форматов данных.

Для конфигурирования сервиса используются классы, определяющие параметры JWT, почтовой рассылки, S3-хранилища, кэша и другие аспекты. Благодаря этому настроить приложение можно через свойства и профили Spring.

4.5 Руководство пользователя

Программный комплекс «Экономический симулятор» предназначен для студентов, преподавателей и всех, кто интересуется современными инструментами анализа экономических моделей. Приложение предоставляет простой и интуитивно понятный интерфейс, а также широкий спектр функций

для моделирования, работы с документами и формирования аналитических отчетов. Перед началом работы с программой рекомендуется ознакомиться с требованиями к системе и основными принципами использования.

Системные требования:

– ОС Linux (Ubuntu 20.04 LTS, CentOS 8 и др.) или Windows 11;

– минимум 2 ядра CPU с частотой не ниже 2.5 ГГц;

– ОЗУ не менее 2 ГБ;

– дисковое пространство от 200Мб;

– JRE для запуска исполняемого .jar.

После первого запуска приложения пользователю будет предложено пройти авторизацию. Для входа необходимо ввести имя пользователя или адрес электронной почты и пароль в специальной форме. Если у пользователя ещё нет аккаунта, предусмотрена возможность регистрации — для этого нужно перейти по соответствующей ссылке и заполнить предложенную анкету, после чего на указанный адрес электронной почты поступит код подтверждения. В случае утери пароля можно воспользоваться функцией восстановления, следуя простым инструкциям на экране.

После успешной авторизации открывается главное окно программы, в котором отображается список доступных экономических моделей. Пользователь может выбрать интересующую модель для изучения или проведения расчетов. Помимо этого, всегда доступна функция быстрого перехода к последней открытой модели, а также возможность открыть раздел для управления загруженными файлами и отчетами. Интерфейс приложения поддерживает переключение языка между русским и английским, что делается через соответствующую кнопку в меню.

В режиме работы с конкретной моделью пользователь видит описание модели, а также перечень параметров, которые можно изменять. После внесения изменений предусмотрена возможность сохранения новых параметров и запуска расчетов, результаты которых отображаются в отдельном окне — здесь же можно

сформировать и сохранить отчет. При необходимости можно воспользоваться разделом работы с документами: загруженные PDF-файлы можно просматривать, извлекать из них параметры и автоматически подставлять их в текущую модель. Кроме того, встроен чат-ассистент, который помогает разобраться в принципах работы моделей, отвечает на вопросы и предоставляет текстовые пояснения.

Раздел управления документами позволяет загружать новые PDF-документы, просматривать и упорядочивать уже имеющиеся файлы, удалять ненужные документы, а также скачивать необходимые отчеты. При необходимости пользователь может выбрать определённый файл для извлечения параметров и автоматического возврата этих данных в активную модель, что особенно удобно при работе с загруженной отчетностью.

Для завершения работы предусмотрена функция выхода: достаточно нажать соответствующую кнопку в правом верхнем углу главного окна, после чего сессия будет завершена. При последующем запуске приложения потребуется повторно выполнить вход в систему.

Описанный процесс работы с приложением обеспечивает удобный и безопасный доступ к экономическим моделям, начиная с простой и понятной процедуры авторизации. Приложение предлагает интуитивно понятный интерфейс с возможностью выбора моделей, настройки параметров, выполнения расчетов и работы с документами. Дополнительные функции, такие как мультиязычная поддержка, чат-ассистент и автоматическое извлечение данных из документов, значительно упрощают взаимодействие с системой. Руководство содержит всю необходимую информацию для комфортного освоения приложения и эффективного решения задач пользователя.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы была реализована поставленная задача — создание интерактивного программного комплекса для моделирования экономических процессов, ориентированного на образовательные цели. В рамках исследования был проведён тщательный анализ существующих решений и аналогов, что позволило выявить основные тенденции, а также недостатки и пробелы современных образовательных платформ. Это стало основой для формирования собственных функциональных требований и определения приоритетов в проектировании архитектуры будущей системы.

В процессе разработки был выбран оптимальный стек технологий, обеспечивающий надёжность, производительность и масштабируемость решения. Основное внимание уделялось реализации удобного клиентского интерфейса, который обеспечивает интуитивную навигацию и простоту взаимодействия пользователя с экономическими моделями. Особое место заняла интеграция средств визуализации и современных технологий искусственного интеллекта, позволивших автоматизировать обработку документов, а также предоставлять студентам дополнительные объяснения и аналитику по результатам моделирования.

Созданная архитектура клиент-серверного приложения обеспечивает разделение бизнес-логики и пользовательского интерфейса, что не только упрощает поддержку и развитие системы, но и гарантирует её гибкость при возможных изменениях требований или расширении функционала. На всех этапах проектирования и реализации были использованы современные принципы модульности, переиспользования кода и обеспечения безопасности. Надёжная база данных, распределённое кеширование, локализация и поддержка различных сценариев использования позволяют системе адаптироваться под разные образовательные задачи.

Тестирование программного комплекса показало его работоспособность, удобство для конечного пользователя и высокую степень автоматизации многих рутинных операций. Проведённая отладка и оптимизация отдельных модулей позволили обеспечить стабильную работу при одновременном обращении нескольких пользователей, а реализованные механизмы логирования и обработки ошибок упростили сопровождение и выявление возможных неисправностей.

Таким образом, итогом работы стал современный и многофункциональный симулятор экономических моделей, способный эффективно дополнять традиционные методы преподавания и значительно повышать интерес обучающихся к изучению экономики. Внедрение такого комплекса в образовательный процесс способствует формированию устойчивых аналитических и исследовательских навыков, а также практическому освоению теоретических основ экономических дисциплин.

Полученные в процессе разработки знания и результаты могут быть использованы как основа для дальнейшего совершенствования системы, расширения её функциональности, внедрения новых экономических моделей и интеграции дополнительных аналитических сервисов. Реализованный программный продукт способен адаптироваться к новым образовательным вызовам, что делает его перспективным инструментом для применения в учебных заведениях и для самостоятельного изучения экономики.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Блаватник А. В. Микроэкономика: Учебник для бакалавриата и магистратуры. — М.: Юрайт, 2023. — 480 с.

2 Григорьев А. В. Java. Полное руководство. — М.: Диалектика, 2019. — 800 с.

3 Шилдт Г. Java. Библиотека профессионала. — М.: Вильямс, 2020. — 960 с.

4 Ляшенко А. В. Spring Framework 5 для профессионалов. — М.: БХВ-Петербург, 2020. — 560 с.

5 Филиппов В. А. JavaFX 9. Создание графических приложений. — СПб.: Питер, 2018. — 384 с.

6 Смирнов И. В. Клиент-серверные приложения на Java. — М.: Наука и техника, 2017. — 320 с.

7 Ковалёв М. Ю. Разработка корпоративных приложений с Spring Boot. — СПб.: Питер, 2021. — 400 с.

8 Freeman S. Pro JavaFX 9: A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients. — Berkeley: Apress, 2017. — 650 p.

9 Spring Boot Documentation [Электронный ресурс]. URL: https://docs.spring.io/spring-boot/docs/current/reference/html/ (дата обращения: 04.05.2025).

10 JavaFX Documentation [Электронный ресурс]. URL: https://openjfx.io/ (дата обращения: 20.05.2025).

11 PostgreSQL Documentation [Электронный ресурс]. URL: https://www.postgresql.org/docs/ (дата обращения: 08.05.2025).

12 Redis Documentation [Электронный ресурс]. URL: https://redis.io/docs/ (дата обращения: 16.05.2025).

13 MinIO Documentation [Электронный ресурс]. URL: https://min.io/docs/minio/linux/index.html (дата обращения: 27.05.2025).

14 Ollama Documentation [Электронный ресурс]. URL: https://ollama.com/library (дата обращения: 05.06.2025).

15 Гавриленко Л. С. Информационные системы: основы проектирования. — М.: КНОРУС, 2022. — 256 с.

16 Сидоров П. Н. Информационные системы: основы проектирования. — М.: КНОРУС, 2019. — 288 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Диаграмма последовательностей

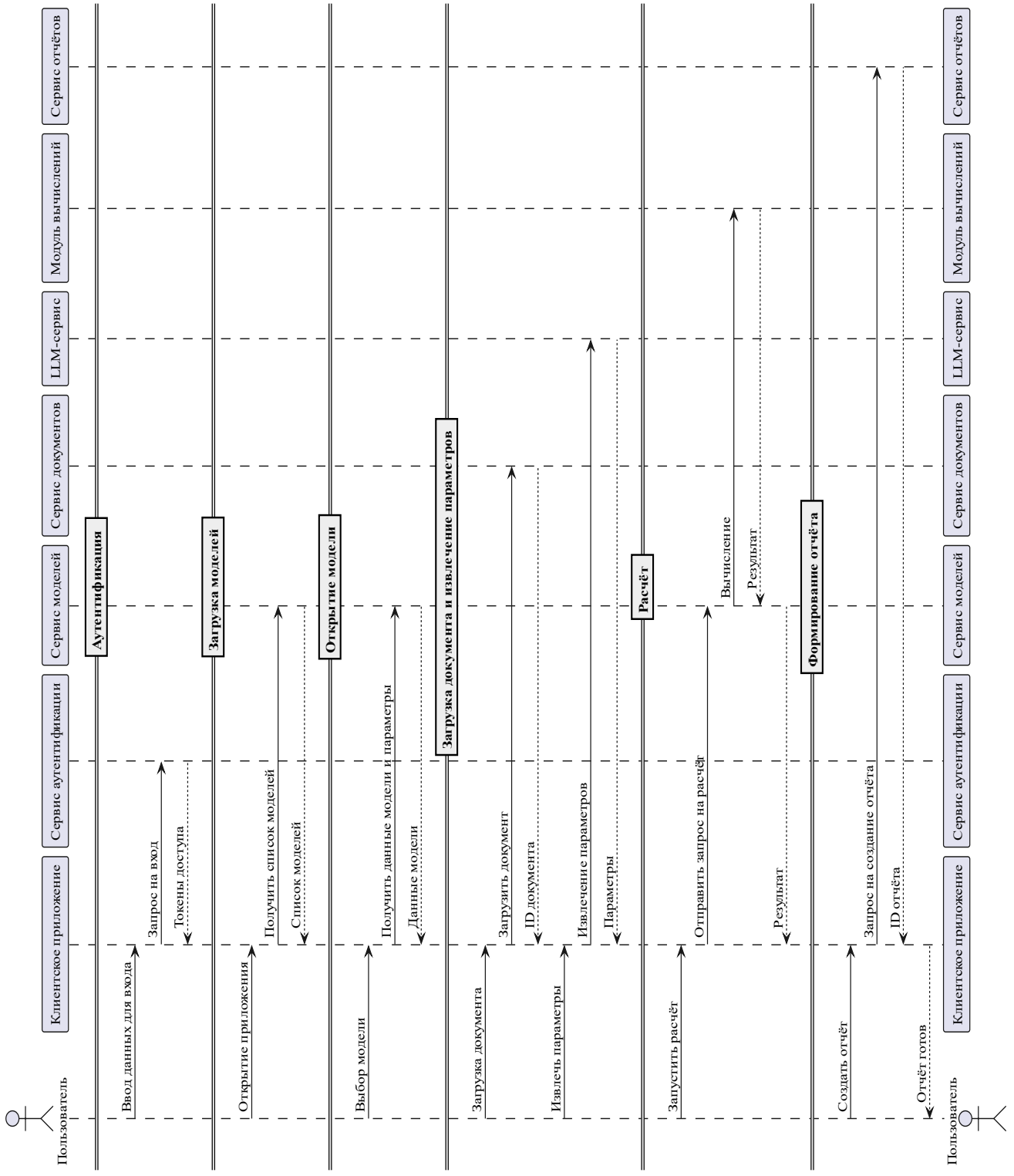

Рисунок А.1 – Диаграмма последовательностей