Министерство науки и высшего образования Российской Федерации

**Муромский институт**

федерального государственного бюджетного образовательного учреждения высшего образования

**«Владимирский государственный университет   
имени Александра Григорьевича и Николая Григорьевича Столетовых»**

**(МИ ВлГУ)**

Факультет ИТР

Кафедра ИС

КУРСОВАЯ

РАБОТА

по Проектирование информационных систем

(наименование дисциплины)

Тема Проектирование информационной системы интерактивного‎   
 симулятора экономических моделей, разработка MVP

Руководитель

Еремеев С. В.

(оценка) (фамилия, инициалы)

Члены комиссии

(подпись) (дата)

Студент ИС-121

(подпись) (Ф.И.О.) (группа)

Есин Д. И.

(подпись) (Ф.И.О.) (фамилия, инициалы)

(подпись) (дата)

Муром 2025

БЛАНК ЗАДАНИЯ

Курсовая работа посвящена проектированию информационной системы – симулятора экономических моделей, предназначенного для автоматизации математического и имитационного моделирования экономических процессов на предприятиях. В ней представлены анализ предметной области и аналогов, описаны основные этапы проектирования и разработки прототипа ИС с использованием технологий искусственного интеллекта. Проект ориентирован на экономистов, аналитиков и специалистов в области стратегического управления бизнесом.

The term paper is devoted to the design of an information system – an economic model simulator aimed at automating mathematical and simulation modeling of business processes in enterprises. It contains an analysis of the subject area and existing analogues, describes the main stages of designing and developing an IS prototype using artificial intelligence technologies. The project is intended for economists, analysts, and specialists in the field of strategic business management.

Министерство науки и высшего образования Российской Федерации

**Муромский институт**

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: Проектирование информационной системы интерактивного симулятора экономических моделей, разработка MVP

МИВУ.09.03.02-04.000 ПЗ

Муром 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 7](#_Toc193716321)

[1 Анализ технического задания 9](#_Toc193716322)

[1.1 Программы аналоги 9](#_Toc193716323)

[1.2 Функциональные требования 11](#_Toc193716324)

[1.3 Выбор средств программной реализации 12](#_Toc193716325)

[2 Разработка проекта Информационной системы 15](#_Toc193716326)

[2.1 Проектирование бизнес-процессов 16](#_Toc193716327)

[2.2 Описание сценариев использования 20](#_Toc193716328)

[2.3 Проектирование архитектуры программы 21](#_Toc193716329)

[2.4 Проектирование взаимодействия модулей 23](#_Toc193716330)

[2.5 Проектирование базы данных 25](#_Toc193716331)

[2.6 Проектирование структуры классов сервера 26](#_Toc193716332)

[2.7 Проектирование структуры классов клиента 28](#_Toc193716333)

[3 Разработка прототипа программы 30](#_Toc193716334)

[3.1 Функциональные требования к прототипу 31](#_Toc193716335)

[3.2 Конфигурация и развертывание 32](#_Toc193716336)

[3.3 Разработка основных компонентов 33](#_Toc193716337)

[4 Тестирование 40](#_Toc193716338)

[4.1 Тестирование клиента 40](#_Toc193716339)

[4.2 Тестирование сервера 44](#_Toc193716340)

[ЗАКЛЮЧЕНИЕ 46](#_Toc193716341)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 47](#_Toc193716342)

# 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 специализированного 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.

## Программы аналоги

При проектировании информационной системы важно учитывать уже существующие решения, которые выполняют схожие функции. Анализ аналогов позволяет выявить их сильные и слабые стороны, что поможет избежать ошибок при разработке собственной системы. Для оценки преимуществ и недостатков были выбраны три наиболее распространённых аналога: AnyLogic, Bizagi Modeler и Vensim.

AnyLogic – это многофункциональное программное решение для имитационного моделирования бизнес-процессов, социальных и экономических систем. Программа поддерживает различные подходы к моделированию (агентное, системно-динамическое, дискретно-событийное), а также имеет широкие возможности визуализации и анализа данных. Основным недостатком является высокая сложность изучения для неподготовленных пользователей и отсутствие встроенных возможностей для анализа документации с использованием технологий AI.

Bizagi Modeler – популярный инструмент моделирования бизнес-процессов, который позволяет проектировать и анализировать процессы с помощью наглядных BPMN-диаграмм. Программа отличается простотой освоения и хорошими возможностями для экспорта результатов. Недостатком Bizagi Modeler является ограниченность функционала исключительно моделированием бизнес-процессов, а также отсутствие встроенных модулей математических расчётов и интеграции AI.

Vensim – инструмент системно-динамического моделирования экономических и социальных процессов, позволяющий создавать и тестировать сложные математические модели. Программа обладает удобным интерфейсом для разработки моделей и проведения расчётов, а также мощными средствами анализа и визуализации результатов. Недостатком Vensim является отсутствие поддержки имитационного моделирования агентов и невозможность интеграции внешних AI-сервисов.

Таблица 1- Сравнительный анализ программ аналогов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Программа** | **Математическое моделирование** | **Имитационное моделирование** | **Анализ финансовой отчетности** | **Интеграция AI-сервисов** | **Экспорт отчетности** |
| AnyLogic | Да | Да | Нет | Нет | Да |
| Bizagi Modeler | Нет | Нет | Нет | Нет | Да |
| Vensim | Да | Нет | Нет | Нет | Да |

Анализ существующих решений показал, что большинство программ-аналогов не поддерживают весь комплекс необходимых возможностей, включая интеграцию искусственного интеллекта и автоматический анализ финансовой отчетности. Эти недостатки требуют дополнительных затрат на интеграцию сторонних решений и значительно усложняют рабочий процесс. Таким образом, необходима разработка специализированной информационной системы, объединяющей математическое и имитационное моделирование, а также возможности AI-анализа документации и финансовых показателей в рамках единого программного решения.

## Функциональные требования

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, сравнивать несколько расчётов 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, сохранять пользовательские сессии и кэшировать результаты предыдущих расчётов. Параллельное выполнение вычислений позволит нескольким пользователям одновременно работать с разными моделями без задержек, а встроенные средства защиты данных обеспечат конфиденциальность и целостность пользовательской информации.

## Выбор средств программной реализации

Для реализации серверной части разрабатываемой информационной системы предъявляются требования к языку программирования, фреймворку, реализации кеширования, а также к системе управления базами данных. Клиентская часть должна быть реализована в виде десктопного приложения с дружественным пользовательским интерфейсом.

1. Выбор языка программирования сервера:

Выбор стоит между Java и Python. Python отличается простой синтаксиса, скоростью прототипирования и большим количеством библиотек для работы с данными и AI. Однако Java превосходит Python по производительности, многопоточности и масштабируемости, особенно актуальным для серверных приложений с интенсивными вычислениями и параллельной обработкой данных. Кроме того, использование новейшей версии Java (jdk-21) позволяет использовать современные механизмы виртуальных потоков (Virtual Threads) и значительно оптимизировать работу приложения. Именно поэтому Java была выбрана для реализации серверной части системы.

1. Выбор серверного фреймворка:

Выбор стоит между Spring-framework и Jakarta EE. Jakarta EE предоставляет обширные возможности и является стандартом для разработки корпоративных приложений на Java, однако характеризуется более высоким порогом вхождения и сложностью конфигурации по сравнению с Spring Framework. Spring обладает простотой конфигурации, высоким уровнем интеграции с популярными технологиями и большим сообществом разработчиков, что ускоряет разработку, упрощает поддержку и дальнейшее развитие проекта. Эти преимущества стали решающими для выбора Spring Framework.

1. Выбор реализации кеширования:

Выбор стоит между Ehcache и Redis. Redis представляет собой более мощное решение, поддерживающее распределённый кеш и удобное масштабирование. Однако в рамках данного проекта, учитывая его размер и требования, использование такого мощного инструмента, как Redis, является избыточным. Ehcache является лёгким и быстрым локальным кешем, интегрируемым непосредственно в приложение, не требующим отдельной инфраструктуры. Это позволяет значительно снизить сложность реализации и поддержки, сохранив при этом высокую производительность. Поэтому выбор сделан в пользу Ehcache.

1. Выбор базы данных:

PostgreSQL сравнивался с популярной реляционной базой данных MySQL. MySQL обладает хорошей производительностью, простотой использования и поддерживается большим количеством приложений и сервисов. Однако PostgreSQL предоставляет расширенные возможности работы с JSON, поддерживает сложные запросы и транзакции, которые крайне важны в контексте реализации экономических моделей, хранения расчётов и аналитических данных. Также PostgreSQL имеет более продвинутые механизмы обеспечения целостности и надёжности данных. Именно эти преимущества сделали PostgreSQL предпочтительным вариантом для хранения данных в системе.

1. Выбор языка программирования клиентской части:

Выбор стоит между Java и C#. Платформа .NET и язык C# обладают удобными инструментами и хорошей производительностью для разработки десктоп-приложений. Однако выбор Java обусловлен необходимостью использовать единый стек технологий на клиенте и сервере, что повышает простоту поддержки. Кроме того, JavaFX предоставляет мощные инструменты и библиотеки для реализации удобного и быстрого пользовательского интерфейса. Поэтому Java была выбрана и для реализации клиентского приложения.

1. Выбор технологии для клиентского интерфейса:

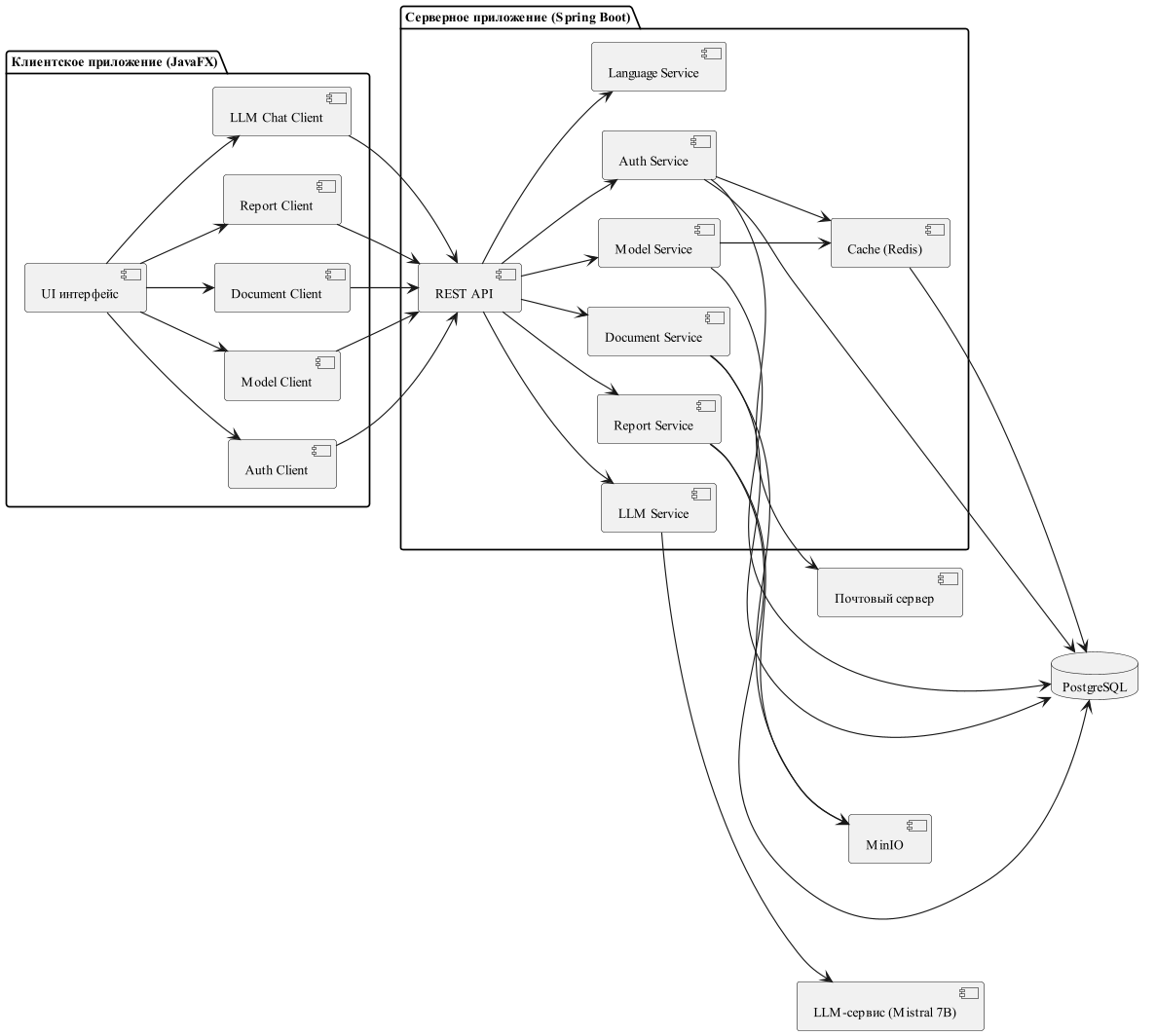
JavaFX сравнивался с альтернативным решением — Swing. Несмотря на то, что Swing — это зрелая и проверенная временем технология, JavaFX является более современным и функциональным решением, обеспечивающим широкие возможности для создания интерактивных и визуально привлекательных интерфейсов, включая встроенную поддержку графиков, диаграмм и анимаций. Именно это послужило причиной выбора JavaFX для реализации интерфейса пользователя в клиентском приложении.

1. Выбор системы сборки:

Для реализации сборки и управления зависимостями серверного и клиентского приложений были рассмотрены две наиболее распространённые системы — Gradle и Maven. Maven является широко используемым инструментом, обладающим стандартной и понятной структурой проекта, простой конфигурацией и развитой экосистемой плагинов. Основное преимущество Maven заключается в унифицированном подходе к сборке и строгом соответствии стандартам, однако это же является его недостатком: любая нестандартная задача требует более сложной настройки. В свою очередь, Gradle представляет собой более современную и гибкую систему сборки. Gradle позволяет описывать процесс сборки проекта при помощи сценариев на языке Groovy или Kotlin, что значительно упрощает реализацию сложных и нестандартных задач. Также Gradle эффективнее управляет зависимостями, поддерживает инкрементальную сборку, параллельное выполнение задач и обладает более высокой скоростью сборки крупных проектов за счёт продвинутого механизма кеширования. Исходя из вышеперечисленного, а также учитывая необходимость гибкости, производительности и лёгкости дальнейшего сопровождения проекта, выбор сделан в пользу системы сборки Gradle.

Таким образом, выбранные технологии и архитектурные решения обеспечат надежность, удобство и безопасность работы информационной системы.

# 2.2 Разработка проекта Информационной 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.

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

# 2.1 Проектирование бизнес-процессов



Рисунок 1.1 – Диаграмма бизнес-процессов ч.1

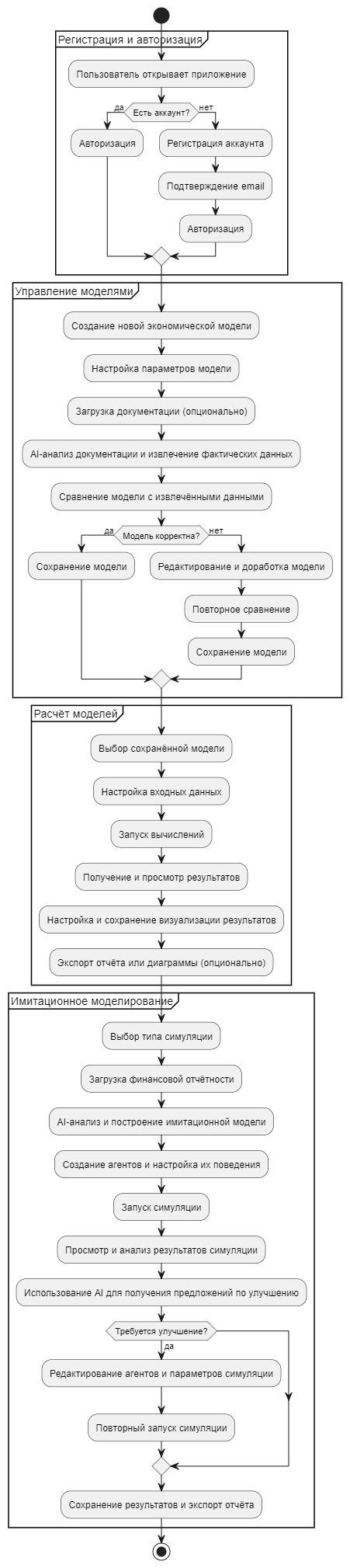


Рисунок 1.2 – Диаграмма бизнес-процессов ч.2

Диаграмма бизнес-процессов – графическое представление последовательности действий или шагов, выполняемых в рамках конкретного процесса организации. Она наглядно отображает порядок выполнения операций, участников процесса, взаимодействия между ними и принимаемые решения.

Диаграмма бизнес-процессов, представленная на рисунке 1.1 и рисунке 1.2 представляет собой описание процесса работы программы.

1. Регистрация и авторизация:

На данном этапе пользователь запускает приложение и проходит процесс авторизации. Если у пользователя уже есть аккаунт, то он вводит данные и сразу получает доступ к системе. Если же аккаунта нет, пользователю необходимо пройти процедуру регистрации, подтвердить email и затем авторизоваться в системе. После успешной авторизации пользователь переходит к дальнейшей работе с системой.

1. Управление моделями:

На данном этапе пользователь создаёт новую экономическую модель, задаёт и настраивает её параметры. При необходимости он может загрузить документацию, которая анализируется с помощью AI, извлекающего из неё фактические данные. Затем система сравнивает введённую модель с полученными данными. Если модель корректна, то она сохраняется; в противном случае пользователю предлагается внести необходимые правки и снова выполнить проверку перед сохранением.

1. Расчёт моделей:

На этапе расчётов пользователь выбирает сохранённую модель и задаёт входные данные для вычислений. Система выполняет расчёты на основе указанных параметров и возвращает результаты. Пользователь получает возможность просмотреть результаты расчётов, настроить визуализацию и при необходимости экспортировать отчёт или диаграмму в удобном формате для последующего использования или анализа.

1. Имитационное моделирование:

В данном разделе пользователь выбирает тип имитационной модели и загружает необходимую финансовую отчётность. После этого AI-модуль автоматически анализирует отчёты и формирует имитационную модель с соответствующими агентами. Пользователь запускает симуляцию и просматривает её результаты, получая рекомендации от AI по возможным улучшениям. При необходимости пользователь может внести изменения в поведение агентов и параметры модели, повторить симуляцию, а затем сохранить и экспортировать результаты работы.

# 2.2 Описание сценариев использования

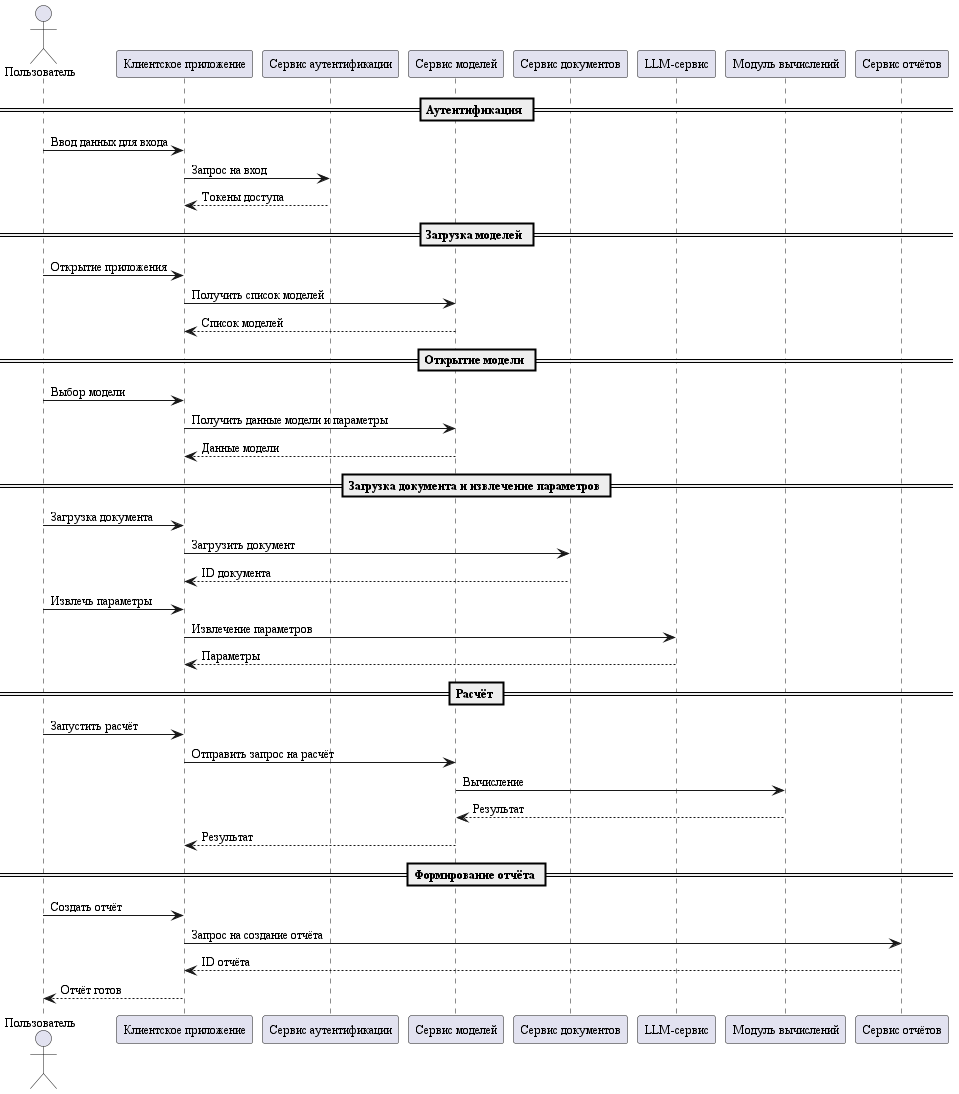


Рисунок 2 – Диаграмма сценариев использования

Диаграмма сценариев использования (Use Case) наглядно отображает функционал информационной системы с точки зрения взаимодействия пользователей с различными компонентами.

На рисунке 2 представлены основные сценарии использования проектируемой программы. Отображены следующие возможности: авторизация и регистрация, управление экономическими моделями, вычисления и расчёты, имитационное моделирование и управление профилем пользователя. В качестве акторов на диаграмме представлены пользователь, система и внешний AI-модуль.

Пользователь взаимодействует с системой посредством регистрации, авторизации и управления экономическими и имитационными моделями, включая создание, редактирование, удаление моделей и запуск расчётов и симуляций. Система обеспечивает обработку запросов, выполнение расчётов и визуализацию результатов, а также осуществляет передачу документации на анализ внешнему AI-модулю. AI-модуль проводит интеллектуальный анализ предоставленных данных, возвращая системе результаты, на основе которых она формирует рекомендации для пользователя.

# 2.3 Проектирование архитектуры программы

Для проектирования информационной системы был проведён сравнительный анализ многослойной (трёхуровневой) и микросервисной архитектур. Микросервисная архитектура характеризуется разделением функционала на небольшие независимые сервисы, каждый из которых обладает своей базой данных и может развёртываться и масштабироваться отдельно. Несмотря на высокую гибкость и масштабируемость, микросервисная архитектура требует сложной инфраструктуры, а так же комплексную поддержку и интеграцию отдельных компонентов.

В отличие от неё, выбранная трёхуровневая архитектура представляет собой баланс между гибкостью и простотой реализации. Данная архитектура предполагает разделение системы на чётко структурированные уровни (клиентский интерфейс, сервер с бизнес-логикой и база данных), что значительно упрощает проектирование, разработку и поддержку системы. Для дипломного проекта и заданных условий это наиболее рациональное решение, обеспечивающее необходимую производительность, надёжность, простоту сопровождения и дальнейшее расширение функционала.

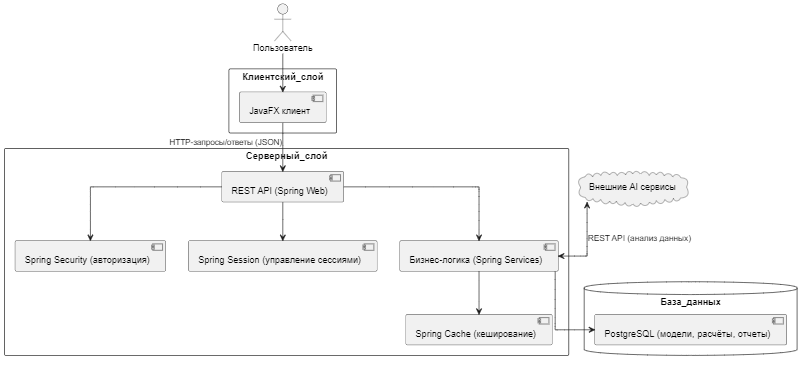


Рисунок 3 – Диаграмма архитектуры

Представленная на рисунке 3 диаграмма наглядно иллюстрирует выбранную трёхуровневую архитектуру разрабатываемой системы. Пользователь взаимодействует с клиентским приложением, разработанным на JavaFX, которое посредством REST API отправляет HTTP-запросы на сервер. Сервер реализован на платформе Spring Framework и включает компоненты авторизации (Spring Security), управления пользовательскими сессиями (Spring Session), слой бизнес-логики (Spring Services) и кеширования (Spring Cache). Серверный слой взаимодействует с внешними AI-сервисами через REST API для анализа данных и обращается к реляционной базе данных PostgreSQL для хранения экономических моделей, расчётов и отчётов.

# 2.4 Проектирование взаимодействия модулей

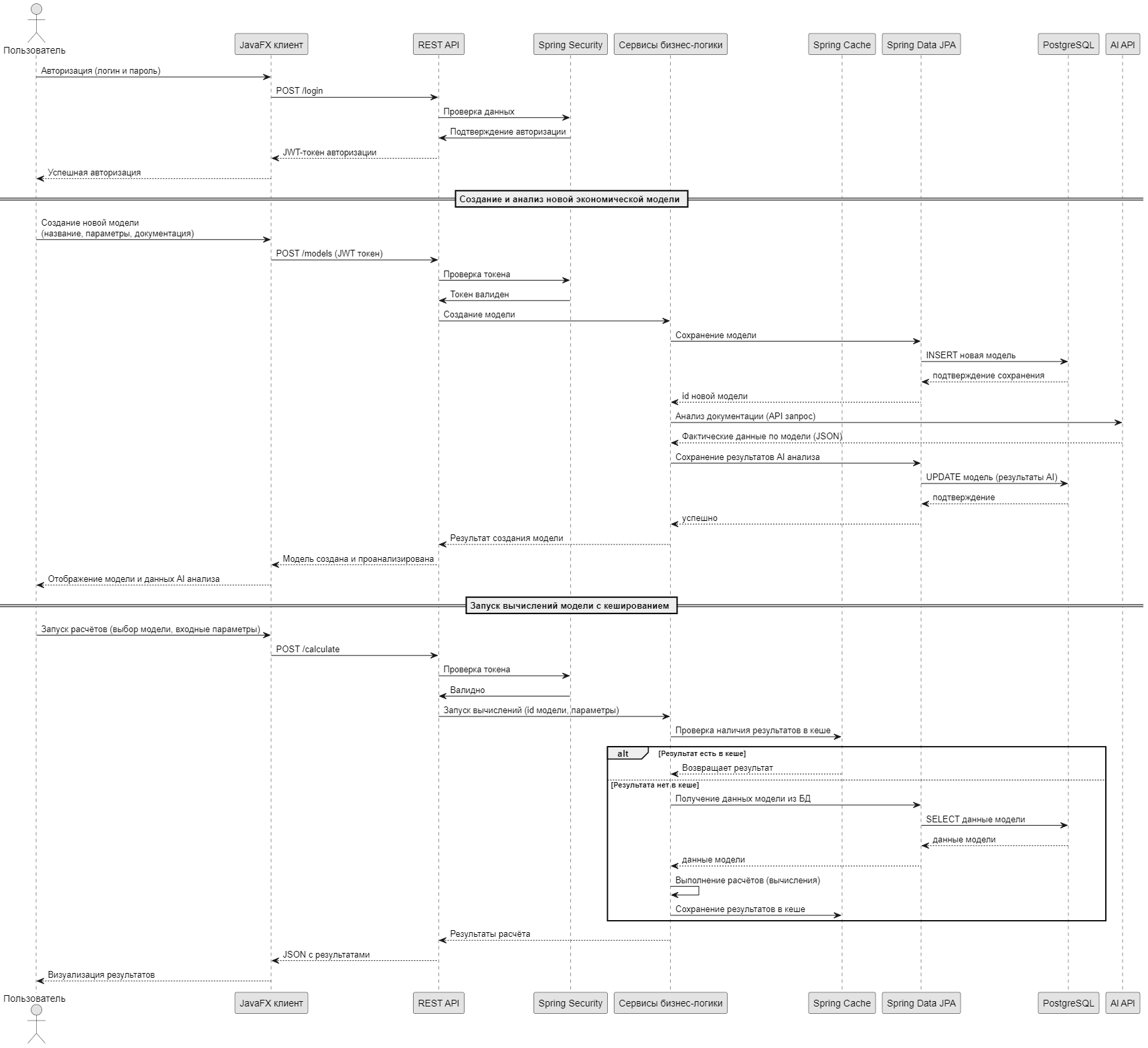


Рисунок 4 – Диаграмма последовательностей ч.1

Диаграмма последовательностей (Sequence diagram) представляет собой инструмент визуализации, который иллюстрирует порядок взаимодействия объектов или компонентов системы во времени. Она отображает обмен сообщениями между объектами, определяет порядок выполнения действий, а также условия, при которых происходят те или иные события.

Представленная на рисунке 4 диаграмма последовательностей отображает три основных сценария взаимодействия компонентов системы:

1. Авторизация пользователя:

Пользователь вводит свои учётные данные (логин и пароль), после чего клиент отправляет запрос на сервер через REST API. Сервер выполняет проверку данных при помощи компонента Spring Security и возвращает клиенту JWT-токен, подтверждающий успешную авторизацию.

1. Создание и анализ новой экономической модели:

Пользователь инициирует создание новой экономической модели, указывая её параметры и загружая документацию. Данные передаются через REST API на сервер, где происходит сохранение модели в базе данных PostgreSQL. Затем сервер отправляет документацию внешнему AI API для анализа и извлечения фактических данных. Полученные данные сохраняются и передаются клиенту, где отображаются пользователю.

1. Запуск вычислений модели с кешированием:

Пользователь выбирает модель и указывает входные параметры для запуска расчётов. Сервер проверяет наличие результатов вычисления в кеше. Если результаты уже есть, они возвращаются пользователю напрямую из кеша. Если результатов нет, система запрашивает данные из базы данных, проводит вычисления и сохраняет результаты в кеше, после чего отправляет их обратно клиенту для последующей визуализации.

# 2.5 Проектирование базы данных

ER-диаграмма (диаграмма «сущность-связь») – это графическое представление структуры базы данных, иллюстрирующее сущности (таблицы), их атрибуты (поля) и связи между ними.

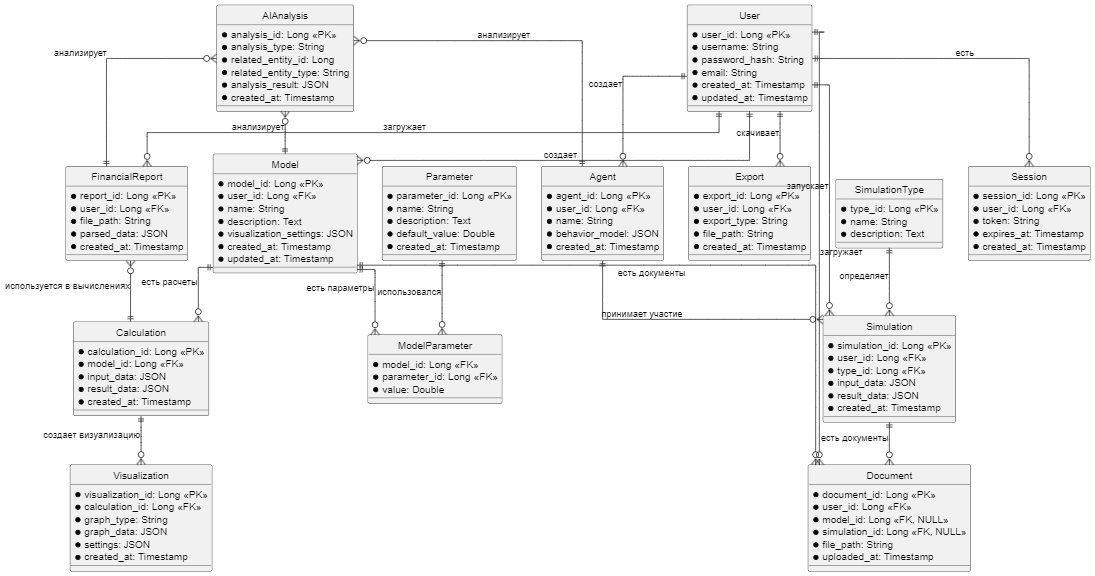


Рисунок 5 – Диаграмма базы данных

Представленная на рисунке 5 ER-диаграмма отображает структуру базы данных проектируемой информационной системы. Основными сущностями являются пользователь (User), экономические модели (Model), параметры моделей (Parameter и ModelParameter), результаты расчётов (Calculation), финансовая отчётность (FinancialReport), имитационные модели и агенты (Simulation, Agent), документы (Document) и анализы с помощью искусственного интеллекта (AIAnalysis). Взаимосвязи показывают, как пользователь создаёт и взаимодействует с моделями и документами, запускает расчёты и симуляции, а также использует AI-анализы. Дополнительные сущности, такие как сессия (Session) и экспорт данных (Export), обеспечивают управление пользовательскими сессиями и возможность экспорта полученных данных.

# 2.6 Проектирование структуры классов сервера

Диаграмма классов (Class Diagram) – это визуальное представление структуры системы, отражающее классы, их атрибуты, методы, а также взаимоотношения между классами. Она используется для проектирования программного обеспечения, обеспечивая ясность и наглядность архитектуры, упрощая понимание взаимодействия различных компонентов системы.

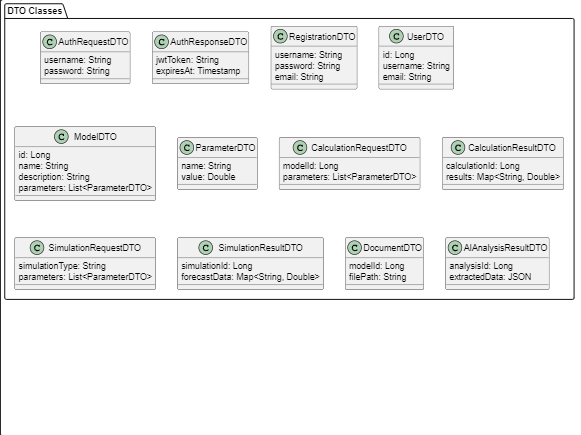


Рисунок 6.1 – Диаграмма классов сервера ч.1

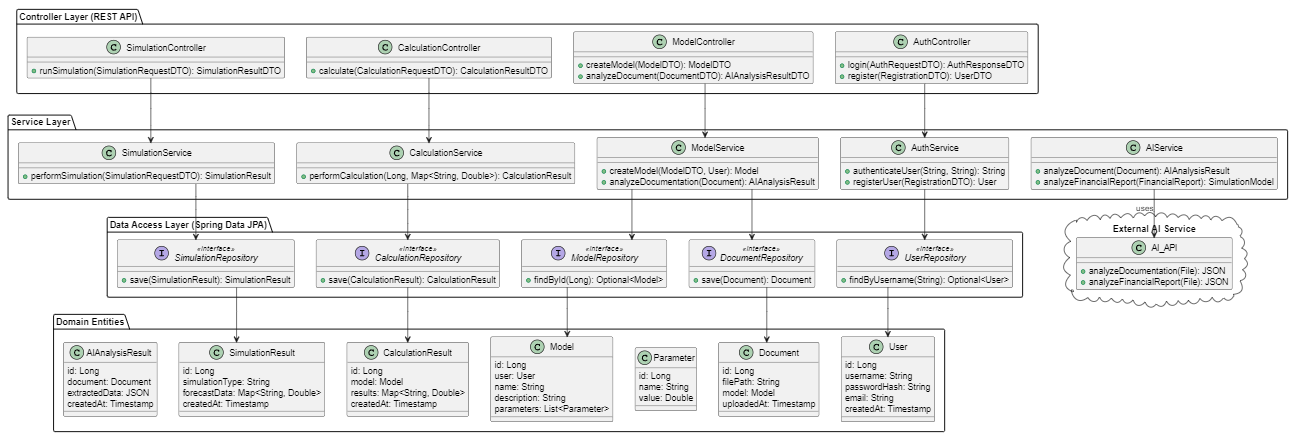


Рисунок 6.2 – Диаграмма классов сервера ч.2

Представленная на рисунке 6.1 и рисунке 6.2 диаграмма классов серверной части информационной системы организована по слоям, согласно трехуровневой архитектуре: Controller Layer (REST API), Service Layer, Data Access Layer (Spring Data JPA), Domain Entities и DTO-классы. Контроллеры принимают запросы от клиентов и вызывают соответствующие методы сервисного слоя, который реализует бизнес-логику, в том числе взаимодействует с внешним AI-сервисом. Сервисный слой взаимодействует с репозиториями слоя доступа к данным (Spring Data JPA), которые в свою очередь работают с базой данных через доменные сущности. DTO-классы обеспечивают передачу данных между клиентом и сервером, скрывая внутреннюю структуру доменных сущностей и предоставляя чёткий интерфейс взаимодействия.

# 2.7 Проектирование структуры классов клиента

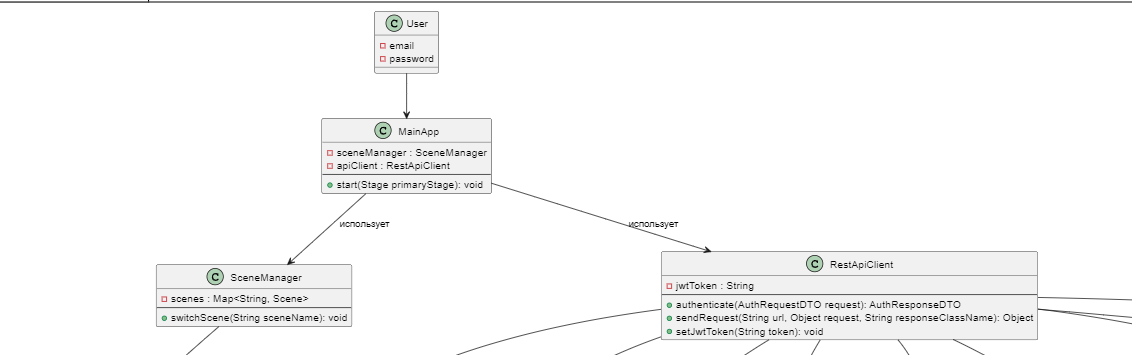


Рисунок 7.1 – Диаграмма классов клиентов ч.1

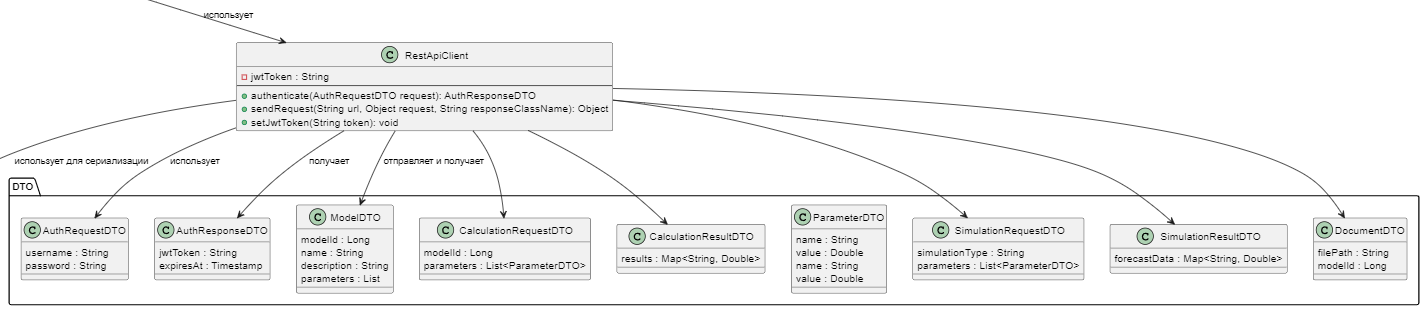


Рисунок 7.2 – Диаграмма классов клиентов ч.2

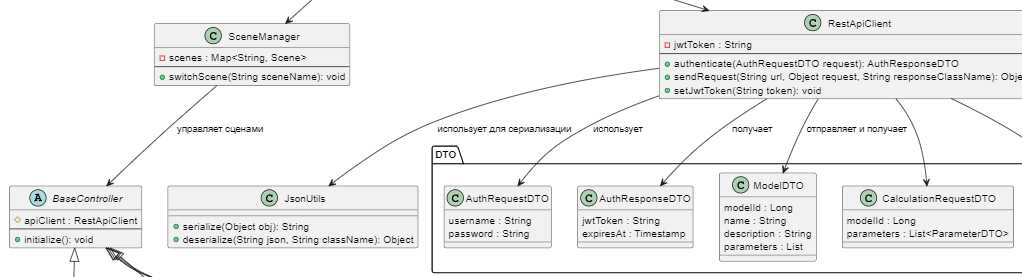


Рисунок 7.3 – Диаграмма классов клиентов ч.3

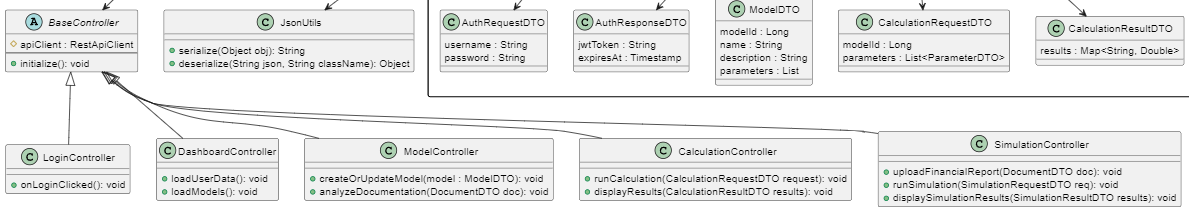


Рисунок 7.4 – Диаграмма классов клиентов ч.4

Представленная на рисунке 7.1, рисунке 7.2, рисунке 7.3 и рисунке 7.4 диаграмма классов отображает структуру клиентского приложения, разработанного с использованием технологии JavaFX. В центре схемы расположен основной класс приложения – MainApp, который управляет запуском и жизненным циклом клиентского приложения. Он взаимодействует с классом SceneManager, отвечающим за переключение и управление различными экранами приложения, а также с классом RestApiClient, обеспечивающим взаимодействие с сервером через REST API.

Контроллеры приложения наследуются от абстрактного класса BaseController, который обеспечивает общий доступ к REST API клиенту. Конкретные контроллеры, такие как LoginController, DashboardController, ModelController, CalculationController и SimulationController, реализуют логику пользовательского взаимодействия и обмена данными с сервером. Для обмена информацией используются специальные DTO-классы (AuthRequestDTO, ModelDTO, CalculationRequestDTO и другие), которые сериализуются и десериализуются при помощи класса JsonUtils.

Класс User представляет собой сущность пользователя, которая передает данные в приложение для авторизации и работы с системой. Таким образом, диаграмма ясно отражает модульную структуру клиентского приложения, обеспечивающую удобство реализации интерфейса пользователя, взаимодействия с сервером и эффективного обмена данными

# Разработка прототипа программы

В рамках курсовой работы был реализован прототип информационной системы — симулятора экономических моделей. Прототип охватывает базовую функциональность, необходимую для демонстрации основных возможностей системы: регистрацию и авторизацию пользователей, создание экономических моделей, запуск расчётов и математического моделирования, а также визуализацию результатов. Реализация выполнена по трёхуровневой архитектуре: клиентское приложение (JavaFX), серверная часть (Spring Framework) и база данных (PostgreSQL).

Клиентская часть разработана в виде настольного приложения с использованием JavaFX и FXML, что позволило создать интерактивный интерфейс для работы с моделями и результатами симуляций. Серверная часть реализована на базе Spring Boot и обеспечивает REST API для обработки пользовательских запросов, выполнения бизнес-логики и взаимодействия с базой данных. Обмен данными между клиентом и сервером осуществляется в формате JSON с использованием DTO-структур.

Таким образом, разработанный MVP демонстрирует ключевые элементы системы и создаёт основу для её дальнейшего расширения, включая углублённый AI-анализ, построение прогнозов, сравнение с реальными финансовыми показателями и расширенную поддержку пользовательских сценариев.

## 3.1 Функциональные требования к прототипу

Разработанный прототип реализует минимально жизнеспособную версию (MVP) симулятора экономических моделей, включающую ограниченный, но демонстрационно-полноценный набор функций. Ниже перечислены основные функциональные возможности, реализованные на уровне клиента и сервера на основе представленного исходного кода.

1. Клиентская часть:

Приложение предоставляет возможность регистрации нового пользователя и авторизации с использованием формы ввода email и пароля. После успешного входа пользователь видит список доступных экономических моделей, загружаемых с сервера, и может выбрать интересующую модель для дальнейшей работы. При выборе модели динамически создаётся интерфейс для ввода параметров, необходимых для симуляции. Затем пользователь может запустить симуляцию и получить результаты, представленные в виде интерактивного линейного графика и текстового отчёта, что обеспечивает наглядное представление работы системы.

1. Серверная часть:

Сервер принимает запросы для регистрации и авторизации пользователей, управления экономическими моделями, а также для создания сессий и проведения симуляций. Реализована интеграция с вычислительным движком, который выполняет математические расчёты на основе введённых пользователем параметров, и возвращает результаты в виде JSON-объектов, содержащих числовые данные и информацию для построения графиков. Дополнительно сервер обрабатывает запросы по получению параметров модели, что позволяет клиенту динамически формировать интерфейс ввода, и осуществляет хранение всех данных в базе данных. Такой подход обеспечивает надёжность, масштабируемость и гибкость системы, создавая фундамент для дальнейшего развития проекта.

## 3.2 Конфигурация и развертывание

В рамках проекта серверная и клиентская части настроены с использованием Gradle, что обеспечивает централизованное управление зависимостями, задачами сборки и конфигурацией модульной системы. Серверный проект основан на Spring Boot и использует build.gradle для настройки зависимостей Spring Framework, Jackson для работы с JSON, а также логирование и поддержку работы с PostgreSQL. Конфигурация компилятора указана для JDK 21, что позволяет использовать современные возможности языка и улучшенные механизмы многопоточности. Кроме того, файл application.properties содержит параметры подключения к базе данных и настройки JPA, что обеспечивает корректное функционирование серверной логики и взаимодействие с БД.

Клиентская часть проекта построена на JavaFX, что позволяет создать интерактивный и удобный пользовательский интерфейс. В build.gradle для клиента используются плагины org.openjfx.javafxplugin и org.beryx.jlink для интеграции модулей JavaFX и формирования исполняемого образа. Здесь заданы зависимости на JavaFX 21, а также различные библиотеки для улучшения визуального оформления (ControlsFX, TilesFX, BootstrapFX и др.). Основной класс приложения и модульная структура проекта указаны в конфигурации, что гарантирует корректный запуск приложения через Gradle и последующую сборку установочного образа с помощью jpackage.

Развертывание осуществляется отдельно для серверной и клиентской частей. Серверное приложение поднимается через запуск .jar файлв, что запускает встроенный сервер Spring Boot, связывает его с базой данных и обеспечивает работу REST API. Клиентское приложение собирается с использованием задач jlink и jpackage, формируя автономный установочный образ (.exe) для настольного приложения. Такой подход позволяет эффективно управлять жизненным циклом проекта, упрощает развертывание и гарантирует согласованность конфигураций на всех этапах эксплуатации системы.

## 3.3 Разработка основных компонентов

Серверная часть проекта построена на базе Spring Boot, что обеспечивает быструю и гибкую разработку REST API и интеграцию с базой данных PostgreSQL. В архитектуре сервера основное внимание уделено разделению ответственности между контроллерами, сервисным слоем, репозиториями и мапперами.

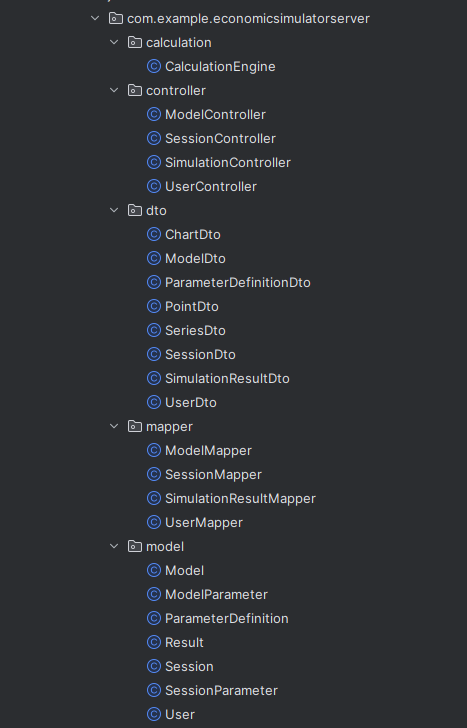


Рисунок 8.1 – Структура проекта сервера ч.1

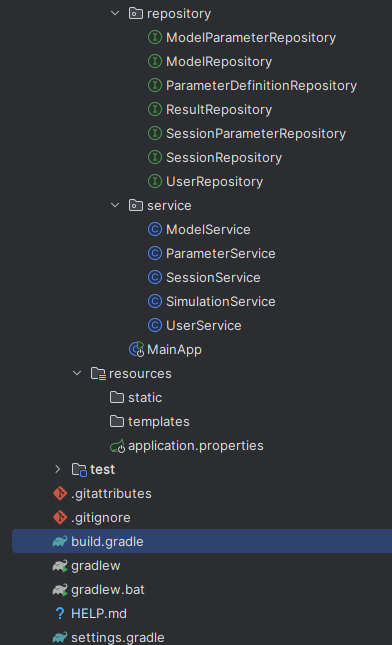


Рисунок 8.2 – Структура проекта сервера ч.2

Контроллеры (такие как UserController, ModelController, SessionController и SimulationController) являются входной точкой для HTTP-запросов. Каждый контроллер принимает данные в виде DTO, валидирует их и передаёт в соответствующий сервисный класс для дальнейшей обработки.

Например, UserController обрабатывает регистрацию и вход пользователей, используя UserService, который проверяет корректность данных, взаимодействует с базой через UserRepository и преобразует доменные сущности в DTO посредством UserMapper.

Контролллеры серверной части:

* UserController: управляет регистрацией и авторизацией пользователей;
* SessionController: управляет сессиями пользователей;
* ModelController:осуществляет взаимодействие с экономическими моделями;
* SimulationController: управляет симуляциями.

Сервисный слой реализован классами UserService, ModelService, SessionService, SimulationService и ParameterService. Они инкапсулируют бизнес-логику приложения: создание, обновление и удаление экономических моделей, формирование сессий для симуляций, запуск расчётов и симуляций. Особое внимание уделено вычислительному компоненту — классу CalculationEngine, который принимает математическую формулу и переменные, используя библиотеку exp4j для парсинга и вычисления, а также кэширует ExpressionBuilder для повышения производительности. Кроме того, в сервисном слое реализованы методы обновления параметров сессии, где происходит удаление старых значений и вставка новых в таблицу session\_parameter с помощью соответствующего репозитория.

Репозитории, реализованные через Spring Data JPA (например, UserRepository, ModelRepository, SessionRepository, ResultRepository), обеспечивают стандартные операции CRUD и специализированные методы поиска, позволяя легко извлекать и сохранять данные.

Мапперы (такие как ModelMapper, SessionMapper, SimulationResultMapper и UserMapper) осуществляют преобразование между доменными сущностями и DTO, что позволяет отделить внутреннюю модель данных от формата, используемого при обмене информацией с клиентом. Такой многоуровневый подход обеспечивает масштабируемость, удобство тестирования и последующее расширение функционала, а также гарантирует, что каждый компонент системы выполняет строго определённую роль в общей архитектуре.

Клиентская часть разработана с использованием JavaFX и FXML, что позволяет создать современный и интерактивный графический интерфейс для конечного пользователя.

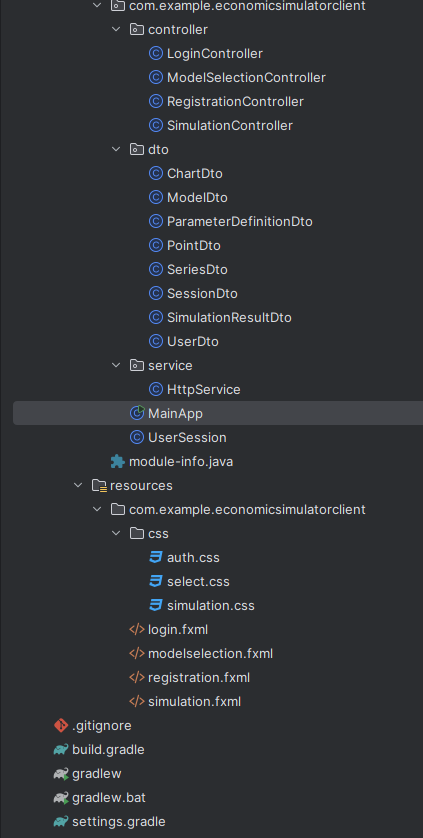


Рисунок 9 – Структура проекта клиента

Основной класс приложения — MainApp — является точкой входа и отвечает за загрузку начальных сцен, установку размеров окна и переключение между различными экранами посредством статического метода switchScene().

public static void switchScene(String fxmlFile) {  
 try {  
 boolean wasFullScreen = *primaryStage*.isFullScreen();  
 double currentWidth = *primaryStage*.getWidth();  
 double currentHeight = *primaryStage*.getHeight();  
  
 FXMLLoader loader = new FXMLLoader(MainApp.class.getResource("/com/example/economicsimulatorclient/" + fxmlFile));  
 Scene newScene = new Scene(loader.load());  
 *primaryStage*.setScene(newScene);  
fullscreen  
 if (wasFullScreen) {  
 *primaryStage*.setFullScreen(true);  
 } else {  
 // Иначе возвращаем прежний размер  
 *primaryStage*.setWidth(currentWidth);  
 *primaryStage*.setHeight(currentHeight);  
 *primaryStage*.setMinWidth(800);  
 *primaryStage*.setMinHeight(600);  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

Пользовательский интерфейс разделён на несколько экранов, каждый из которых управляется отдельным контроллером. Например, LoginController обеспечивает аутентификацию пользователей, реализуя обработчики событий для входа и перехода на экран регистрации, а RegistrationController обрабатывает процесс регистрации, проверяя корректность введённых данных и обеспечивая обратную связь через визуальные элементы (например, метки ошибок и подсветку полей ввода). Контроллер ModelSelectionController загружает список экономических моделей, получаемых с сервера через класс HttpService, и отображает их в списке, используя кастомные ячейки для улучшенного визуального восприятия. На экране симуляции, управляемом SimulationController, динамически создаются поля ввода для параметров, получаемых с сервера, и пользователь может задать новые значения для симуляции.

Контролллеры клиентской части:

* RegistrationController: управляет экраном регистрацией пользователей;
* LoginController: управляет экраном авторизацией пользователей;
* ModelSelectionController: управляет экраном выбора моделей;
* SimulationController: управляет экраном отображения симуляции.

HttpService играет ключевую роль в обмене данными между клиентом и сервером — он отправляет HTTP-запросы (GET, POST) и обрабатывает ответы, преобразуя их из JSON в объекты DTO с помощью библиотеки Jackson. Клиент отображает результаты симуляции в виде интерактивного линейного графика, используя компоненты JavaFX, такие как LineChart, XYChart.Series и другие, а также выводит текстовые отчёты, что позволяет пользователю получить наглядное представление о работе системы.

public SimulationResultDto runSimulation(Long sessionId, Map<String, String> params) throws IOException, InterruptedException {

String url = BASE\_URL + "/api/simulations/" + sessionId + "/run";

HttpRequest.Builder builder = HttpRequest.newBuilder()

.uri(URI.create(url))

.header("Content-Type", "application/json");

if (params != null && !params.isEmpty()) {

String requestBody = objectMapper.writeValueAsString(params);

builder.POST(HttpRequest.BodyPublishers.ofString(requestBody));

} else {

builder.POST(HttpRequest.BodyPublishers.noBody());

}

HttpRequest request = builder.build();

HttpResponse<String> response = httpClient.send(request, HttpResponse.BodyHandlers.ofString());

if (response.statusCode() != 200) {

throw new RuntimeException("Failed to run simulation. HTTP status: " + response.statusCode());

}

return objectMapper.readValue(response.body(), SimulationResultDto.class); }

Преимущества выбранного подхода заключаются в чётком разделении ответственности между слоями приложения, что существенно упрощает поддержку, тестирование и дальнейшее расширение функционала. Каждый компонент системы – от клиентской части, реализованной на JavaFX, до серверной, построенной на Spring Boot с использованием репозиториев для доступа к данным – выполняет строго определённую задачу. Такой подход позволяет вносить изменения в бизнес-логику или структуру базы данных, не затрагивая другие слои, что обеспечивает гибкость системы.

Серверные утилиты в системе выполняют ресурсоёмкие операции, такие как вычисление экономических моделей с использованием класса CalculationEngine, который кэширует парсеры выражений и оптимизирует работу с математическими формулами. Это позволяет значительно ускорить расчёты и повысить производительность при обработке сложных экономических сценариев.

Взаимодействие между клиентской и серверной частями построено на основе REST API, что обеспечивает эффективный обмен данными и чёткое разделение функциональных обязанностей. Клиентское приложение формирует HTTP-запросы, используя объекты DTO для передачи данных, а сервер обрабатывает их через контроллеры, вызывающие соответствующие сервисы, которые, в свою очередь, взаимодействуют с базой данных. Такой подход обеспечивает масштабируемость системы, позволяя легко добавлять новые функции и адаптировать существующие сервисы для решения дополнительных задач, что является существенным преимуществом в условиях динамично развивающихся экономических моделей.

# 4 Тестирование

# 4.1 Тестирование клиента

1. Тестирование экрана регистрации:

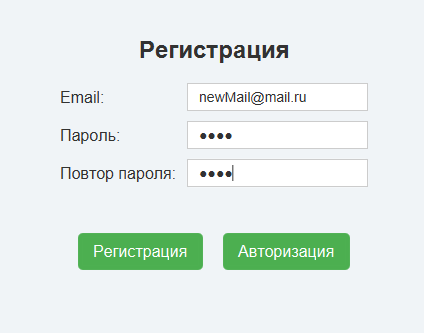


Рисунок 10.1 – Тестирование регистрации ч.1



Рисунок 10.2 – Тестирование регистрации ч.2

Из рисунка 10.1 и рисунка 10.2 видно, что пользователь может вводить данные, и его профиль фиксируется в таблице Users в базе данных.

1. Тестирование авторизации:

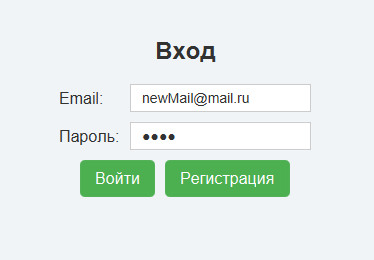


Рисунок 11.1 – Тестирование авторизации ч.1

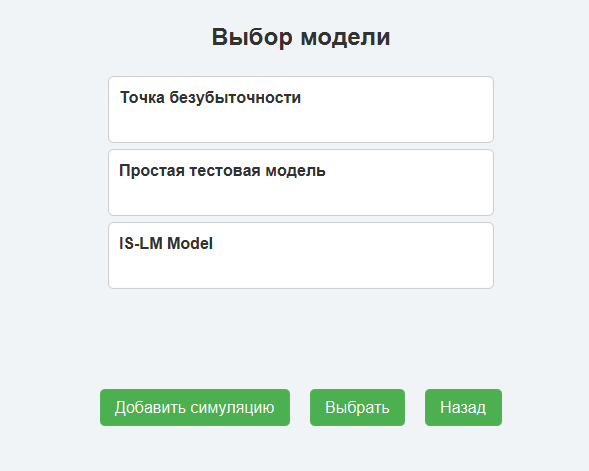


Рисунок 11.2 – Тестирование авторизации ч.2

Из рисунка 11.1 и рисунка 11.2 следует, что при верном заполнении формы авторизации, пользователь попадает на следующий экран с выбором экономической модели.

1. Тестирование выбора модели:

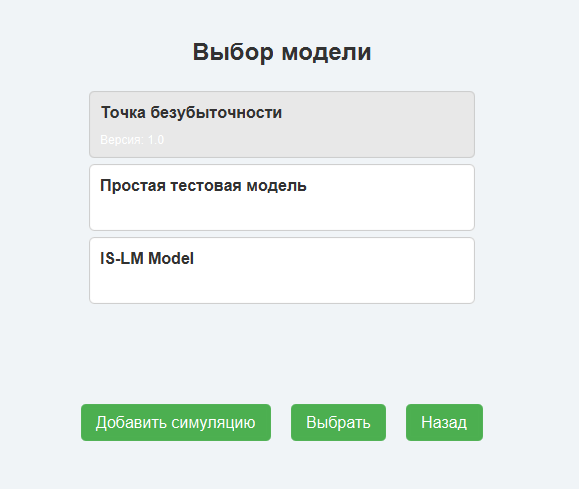


Рисунок 12.1 – Тестирование выбора модели ч.1

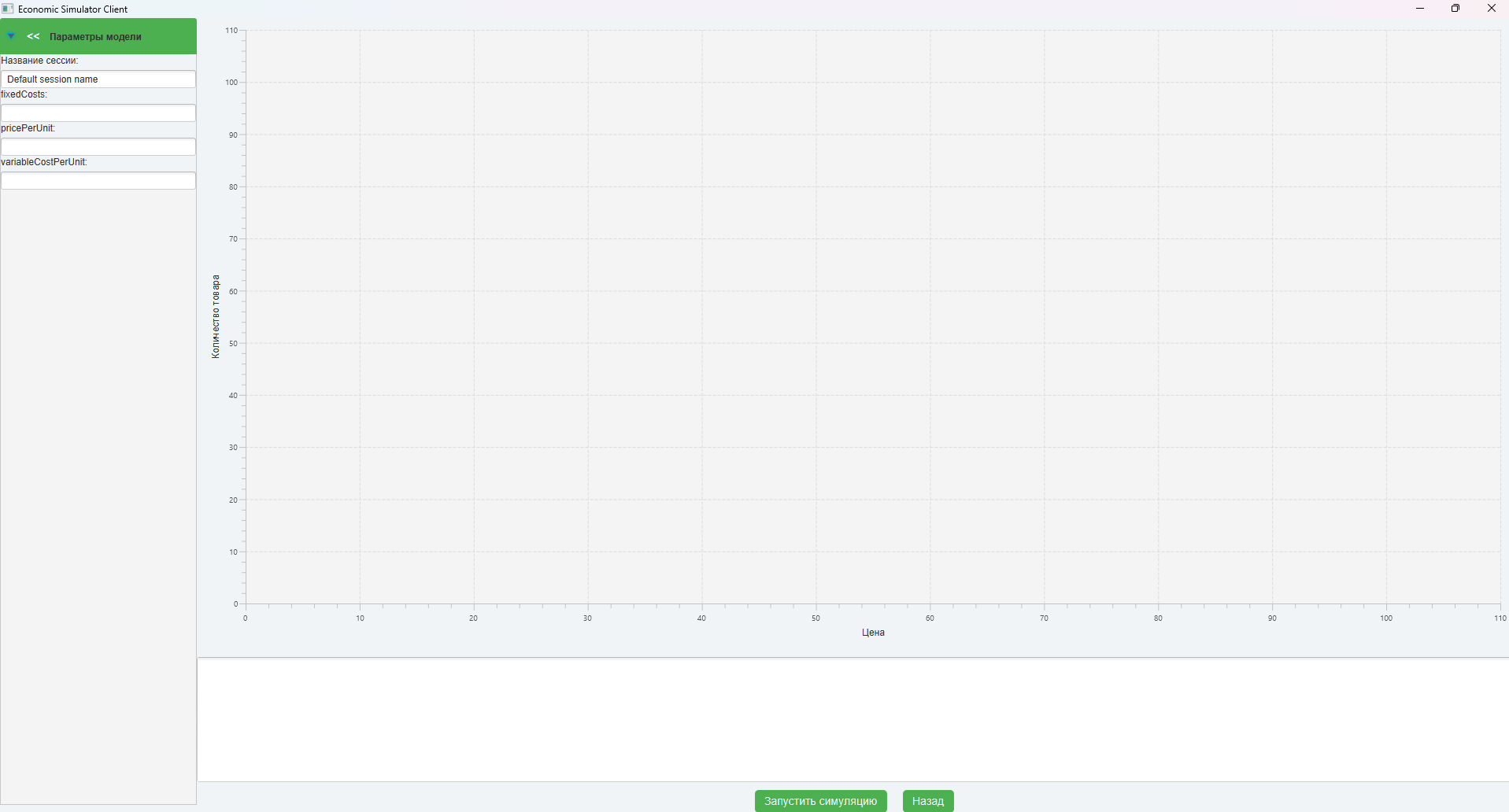


Рисунок 12.2 – Тестирование выбора модели ч.2

На рисунке 12.1 и рисунке 12.2 показано, что при выборе модели, пользователь попадает на экран с визуализацией и вводом параметров. Можно утверждать об успешности теста, изходя из того что на экране ввода параметров отображаются только те параметры, которые принадлежат выбранной модели.

1. Тестирование симуляции:

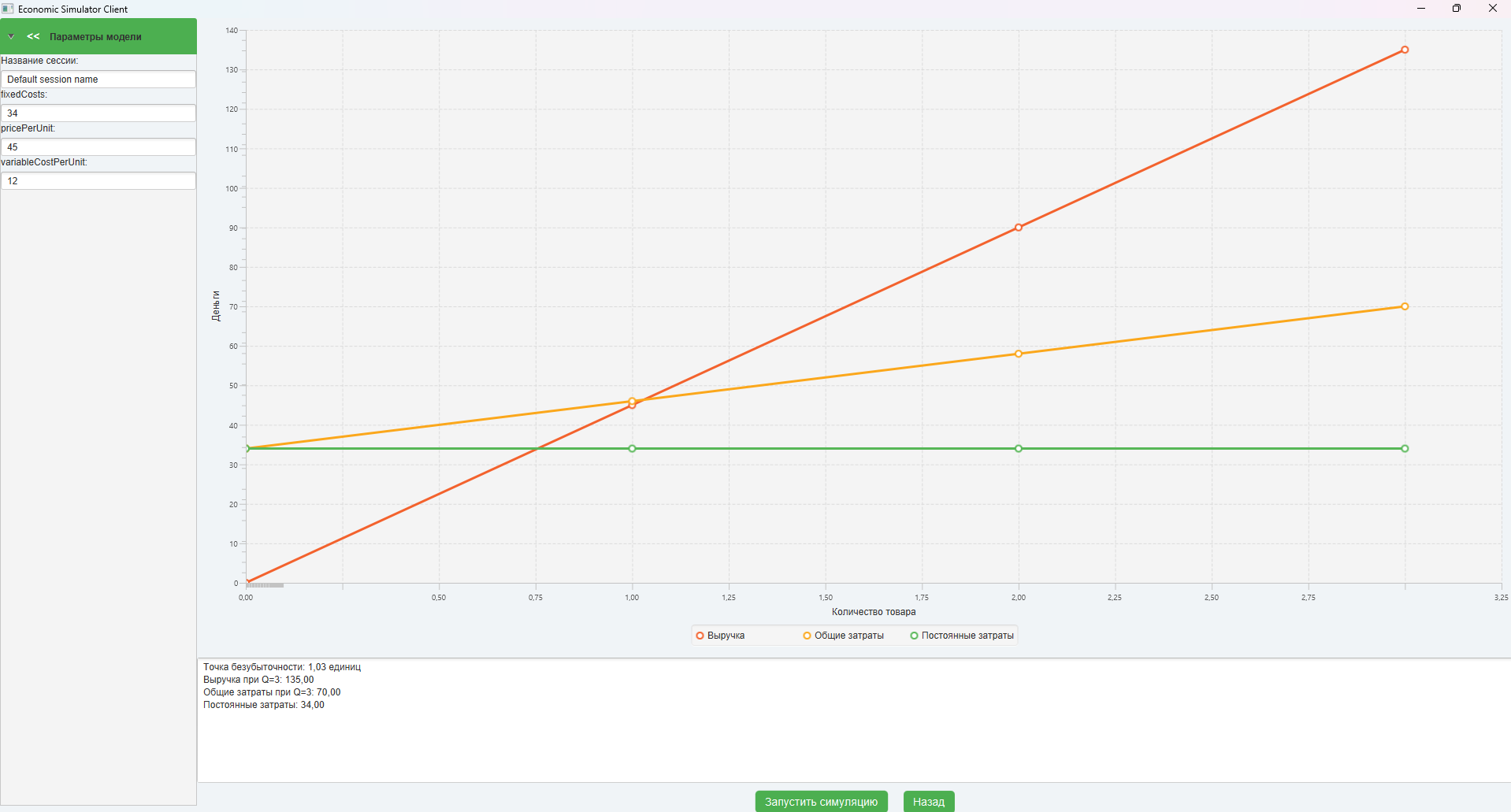


Рисунок 13 – Тестирование симуляции

На рисунке 13 показан результат вычисления точки безубыточности, а так же построение графика по рассчитанным данным.

# 4.2 Тестирование сервера

1. Запуск сервера:

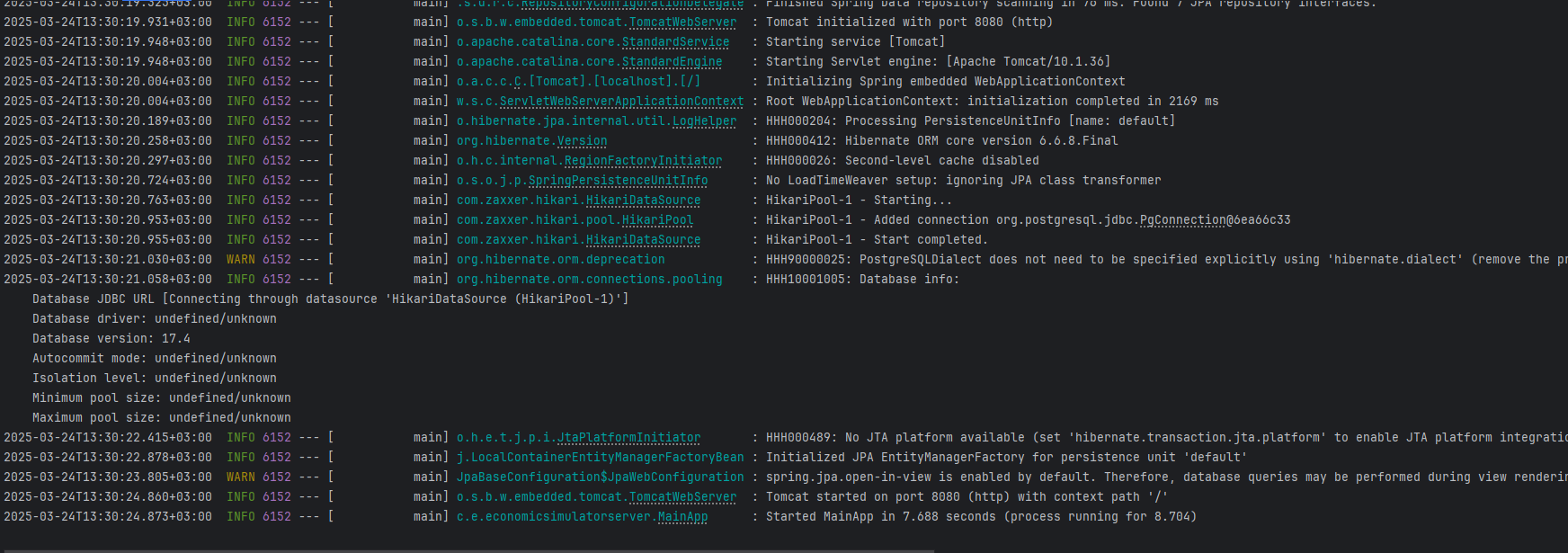


Рисунок 14 – Запуск сервера

На рисунке 14 показан результат запуска сервера. Исходя из логов, мы видим что сервер запущен и ожидает запросов от клиента.

1. Тестирование получения данных с клиента:



Рисунок 15 – Получение данных авторизации

На рисунке 15 показано получение данных об авторизации пользователя. Сервер успешно нашел данные пользователя и осуществил вход.

1. Тестирование расчётов симуляции на сервере:

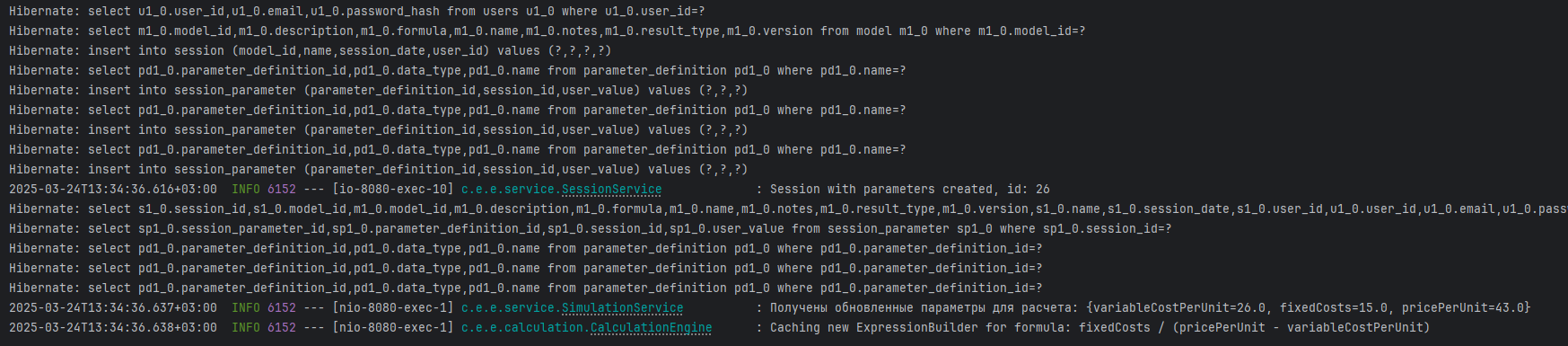


Рисунок 16 – Расчёт симуляции

На рисунке 16 показано получение данных о параметрах, введенных пользователем. Сервер успешно получил данные и произвел вычисления, после чего произвел отправку данных на клиент.

# ЗАКЛЮЧЕНИЕ

Проектирование и разработка информационной системы – симулятора экономических моделей, реализованного с использованием JavaFX, Spring Boot и PostgreSQL, представляет собой актуальное и практически значимое решение для анализа и прогнозирования экономических процессов. Система позволяет автоматизировать сложные вычислительные операции, проводить моделирование различных экономических сценариев и визуализировать результаты симуляций, что значительно упрощает принятие управленческих решений и повышает точность прогнозов. Использование JavaFX обеспечивает создание удобного, интуитивно понятного клиентского интерфейса, а Spring Boot – гибкую, масштабируемую и безопасную серверную логику, что в совокупности способствует эффективной работе приложения.

В процессе разработки были построены подробные диаграммы, отражающие архитектуру системы, бизнес-процессы, сценарии использования, взаимодействие компонентов и структуру базы данных. Такой подход позволил четко разделить функциональные обязанности между слоями, что облегчило тестирование и сопровождение отдельных модулей.

Практическая ценность разработанной системы заключается в возможности быстрого и точного анализа экономических показателей, автоматизации ресурсоемких процессов моделирования и визуализации результатов в виде графиков и текстовых отчетов. Это позволяет оптимизировать процессы управления, снизить затраты времени и ресурсов на проведение аналитических исследований и повысить объективность принятия решений. Внедрение данной информационной системы станет важным шагом в цифровизации экономического анализа и улучшении качества управления предприятиями.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Oracle Java SE Documentation URL:<https://docs.oracle.com/en/java/javase>.
2. Open JFX Documentation – URL: <https://open.jvx.io>.
3. Spring Framework Documentation – URL: <https://spring.io/projects/spring-framework>.
4. Мэнкив, Н. Г. Принципы экономики / Н. Г. Мэнкив. — М.: ДМК Пресс, 2020. — 592 с.
5. Самуэльсон, П. А., Нордхаус, У. Д. Экономика / П. А. Самуэльсон, У. Д. Нордхаус. — М.: ДМК Пресс, 2020. — 784 с.
6. Гарипов, Р. Р. Spring Framework: разработка корпоративных приложений на Java / Р. Р. Гарипов. — М.: ДМК Пресс, 2020. — 352 с.
7. Гончаров, А. Ю. JavaFX: разработка клиентских приложений на Java / А. Ю. Гончаров. — М. : Лань, 2019. — 288 с.
8. Лаура, Л. PostgreSQL: основные понятия и практическое применение / Л. Лаура. — СПб.: БХВ-Петербург, 2020. — 416 с.