**Изображение выглядит как логотип

Автоматически созданное описаниеМИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**(национальный исследовательский университет)»

**Институт (Филиал):** № 8 «Компьютерные науки и прикладная математика»

**Кафедра:** 806

**Группа:** М8О-401Б-21

**Направление подготовки:** 01.03.02 Прикладная математика и информатика

**Профиль:** Информатика

**Квалификация: бакалавр**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

на тему: «Разработка веб-приложения для решения задач смешано-целочисленного линейного программирования»

Автор ВКРБ: Знай Артемий Олегович (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Руководитель: Судаков Владимир Анатольевич (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Консультант: (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Консультант: (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Рецензент: (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

**К защите допустить**

Заведующий кафедрой № 806 «Вычислительная математика  
и программирование» Крылов Сергей Сергеевич (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Москва 202**Изображение выглядит как логотип

Автоматически созданное описание**5

# **РЕФЕРАТ**

Выпускная квалификационная работа бакалавра состоит из 47 страниц, 21 рисунка, 10 использованных источников, 1 приложения.

MILP, ВЕБ-ПРИЛОЖЕНИЕ, ОПТИМИЗАЦИЯ, ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС, КЛИЕНТ, СЕРВЕР, РЕШАТЕЛЬ

Объектом разработки в данной работе является веб-приложение, предназначенное для решения задач смешано-целочисленного линейного программирования (MILP). Приложение ориентировано на упрощение процесса оптимизации сложных бизнес-задач путем предоставления интуитивно понятного интерфейса, позволяющего загружать условия задач, настраивать параметры оптимизации, выбирать решатель и получать оптимальное решение в режиме реального времени.

Цель работы – разработка веб-приложения, которое упростит процесс решения задач MILP для конечных пользователей. Для достижения поставленной цели проведен анализ существующих методов оптимизации, разработан механизм валидации и преобразования входных данных, реализована серверная логика, а также построен клиентский интерфейс. Основное внимание уделено организации обмена информацией между клиентом и сервером.

Основной результат работы – веб-приложение, позволяющее не только эффективно решать задачи смешано-целочисленного линейного программирования, но и проводить анализ чувствительности решения к параметрам модели. Разработанное решение включает удобный пользовательский интерфейс, возможность загрузки условий задачи через Excel-файлы и централизованное хранение результатов в базе данных. Достижения работы демонстрируют, что современный стек веб-технологий может быть успешно использован для интеграции математических методов оптимизации в информационные системы предприятий, способствуя сокращению временных затрат на принятие управленческих решений и повышению эффективности бизнес-процессов.

# **СОДЕРЖАНИЕ**

[РЕФЕРАТ 2](#_Toc197281536)

[ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ 4](#_Toc197281538)

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 5](#_Toc197281539)

[ВВЕДЕНИЕ 6](#_Toc197281540)

[1 ПОДХОДЫ К СОЗДАНИЮ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧ СМЕШАНО-ЦЕЛОЧИСЛЕННОГО ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ 8](#_Toc197281541)

[1.1 Веб-приложение 8](#_Toc197281542)

[1.2 Задачи смешано-целочисленного линейного программирования 11](#_Toc197281543)

[2 АРХИТЕКТУРА ПРОЕКТА 14](#_Toc197281544)

[2.1 Архитектура бэкенда 14](#_Toc197281545)

[2.1.1 Технологии бэкенда 16](#_Toc197281546)

[2.1.2 Структура данных 19](#_Toc197281547)

[2.1.3 Реализация серверной части 20](#_Toc197281548)

[2.2 Архитектура фронтенда 25](#_Toc197281549)

[2.2.1 Технологии фронтенда 28](#_Toc197281550)

[2.2.2 Реализация клиентской части 31](#_Toc197281551)

[2.3 Модульное взаимодействие 36](#_Toc197281552)

[3 ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 38](#_Toc197281553)

[3.1 Формат входных данных 38](#_Toc197281554)

[3.2 Пользовательские сценарии 39](#_Toc197281555)

[ЗАКЛЮЧЕНИЕ 43](#_Toc197281556)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 45](#_Toc197281557)

[ПРИЛОЖЕНИЕ A QR-код репозитория с исходным кодом 46](#_Toc197281558)

**ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ**

В настоящей выпускной квалификационной работе бакалавра применяют следующие термины с соответствующими определениями:

|  |  |
| --- | --- |
| Бэкенд | – внутренняя часть сайта или приложения, скрытая от пользователя |
| Фронтенд | – публичная часть веб-сервисов, с которой пользователь взаимодействует напрямую |
| Веб-приложение | – клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера |
| Веб-сервер | – программное обеспечение, принимающее запросы от клиентов и выдающее ответы на них |
| Эндпоинт | – конечная точка веб-сервиса, к которой фронтенд обращается для выполнения определённых операций или получения данных |

# **ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ**

В настоящей выпускной квалификационной работе бакалавра применяют следующие сокращения и обозначения:

|  |  |
| --- | --- |
| MILP | – Mixed Integer Linear Programming, или смешанное целочисленное линейное программирование |
| API | – Application Programming Interface, или программный интерфейс приложения |
| REST | – Representational State Transfer, или передача состояния представления |
| ORM | – Object-Relational Mapping, или объектно-реляционное отображениe |
| SQL | – Structured Query Language, или язык структурированных запросов |
| СУБД | – система управления базами данных |
| JSON | – JavaScript Object Notation, или текстовый формат обмена данными, основанный на JavaScript |

# **ВВЕДЕНИЕ**

В современных условиях глобальной конкуренции и постоянного усложнения процессов принятия решений оптимизация деятельности предприятий приобретает всё большую значимость. Одной из ключевых направления в этой области является решение задач смешано-целочисленного линейного программирования (MILP), позволяющих моделировать и оптимизировать широкий спектр производственных, логистических и финансовых процессов. MILP-задачи характеризуются наличием как непрерывных, так и дискретных переменных, что делает их решение крайне полезным для систем, где принимаются как решения по объёму производства, так и по выбору конкретных стратегий распределения ресурсов.

Современные методы оптимизации, основанные на математическом моделировании, давно показали свою эффективность при решении сложных задач управления, однако их применение зачастую ограничено трудностями интеграции с реальными информационными системами предприятия. В этом контексте разработка веб-приложения для решения задач MILP представляет особый интерес, поскольку оно позволяет объединить мощные вычислительные возможности специализированных решателей (таких как GLPK, CBC, SCIP) с удобным и доступным интерфейсом, что облегчает практическое использование методов оптимизации даже пользователями без глубоких математических знаний.

Разрабатываемое веб-приложение призвано обеспечивать доступ к современным алгоритмам оптимизации через единый портал, где пользователь может загрузить исходные данные в виде таблиц Excel, выбрать решатель и получить оптимальное решение задачи с анализом чувствительности. Такая интеграция способствует не только автоматизации процессов принятия решений, но и позволяет проводить детальный анализ параметров модели, оценивать влияние изменений входных данных на итоговую оптимизацию и, таким образом, принимать более обоснованные решения в условиях неопределенности и динамичного развития рынка.

Ключевой особенностью работы является построение архитектуры веб-приложения, которая объединяет фронтенд, реализованный с использованием современного фреймворка React, и бэкенд на Python с использованием фреймворка Flask и библиотеки Pyomo. Такой подход позволяет реализовать масштабируемую систему, способную работать с большими объёмами данных и предоставлять гибкие настройки параметров оптимизации. Важным элементом является реализация механизма обмена данными между клиентской и серверной частями с использованием протокола SSE, что обеспечивает оперативное получение промежуточных результатов и обратную связь пользователю.

Настоящая работа направлена на исследование и разработку современных технологий для решения MILP-задач с применением веб-приложений. В её рамках проводится анализ существующих методов оптимизации, разрабатывается и реализуется программное решение, позволяющее не только решать задачи оптимизации, но и проводить анализ чувствительности решения к изменениям исходных параметров модели. Ожидается, что результаты данной работы будут иметь практическое значение для предприятий, стремящихся повысить эффективность управления своими ресурсами, а также станут вкладом в развитие методов прикладной оптимизации и интеграции математического моделирования в информационные системы.

Таким образом, разработка веб-приложения для решения задач смешано-целочисленного линейного программирования является актуальной с научной и практической точек зрения, способствуя созданию удобных инструментов для оптимизации и принятия обоснованных решений в условиях сложных и динамичных производственных процессов.

# **1 ПОДХОДЫ К СОЗДАНИЮ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧ СМЕШАНО-ЦЕЛОЧИСЛЕННОГО ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ**

## **1.1 Веб-приложение**

В последние десятилетия информационные технологии кардинально изменили способы взаимодействия человека с вычислительными системами. Одним из главных достижений этого периода стало появление и массовое распространение веб-приложений. Веб-приложение – это программный продукт, который работает в сети Интернет и предоставляет пользователю функциональность в виде интерактивного интерфейса, доступного через стандартный веб-браузер. В отличие от традиционных настольных программ, веб-приложения не требуют установки на устройство пользователя, что значительно упрощает их развёртывание, обновление и масштабирование.

Исторически развитие веб-приложений началось с простой публикации статических веб-страниц на сервере. С появлением технологий CGI (Common Gateway Interface) и ранних серверных языков, таких как Perl и ASP, началась эра динамических сайтов, способных генерировать содержимое в зависимости от пользовательского ввода. В дальнейшем, с появлением языка JavaScript и фреймворков на стороне клиента, а затем серверных платформ вроде PHP, Python, Ruby, Java, веб-приложения эволюционировали в полнофункциональные системы, способные удовлетворять самые разнообразные потребности – от электронной коммерции до корпоративных систем управления.

Ключевым отличием веб-приложений является их распределённая архитектура. Обычно такие системы состоят из фронтенда, реализованного с использованием HTML, CSS и JavaScript, а также бэкенда, который отвечает за бизнес-логику, хранение данных и обработку запросов пользователей. Благодаря этому разделению разработчики могут обновлять или масштабировать как интерфейс, так и серверную часть независимо друг от друга. В последние годы активно развивается концепция микросервисной архитектуры, когда система разбивается на самостоятельные, слабо связанные компоненты, что повышает отказоустойчивость и упрощает поддержку крупных приложений.

Существуют различные виды веб-приложений, каждый из которых имеет свои особенности:

* одностраничные приложения (Single Page Applications, SPA) – позволяют изменять содержимое страницы без её полной перезагрузки. Это обеспечивает более плавное и отзывчивое взаимодействие с пользователем. К числу популярных технологий относятся React, Angular, Vue.js и другие,
* традиционные многостраничные приложения – работают по классической схеме, когда каждый запрос пользователя приводит к загрузке новой HTML-страницы. Такой подход прост в реализации, однако может уступать по скорости и интерактивности одностраничным приложениям,
* гибридные приложения – сочетают в себе возможности веб-приложений и нативных мобильных решений. Такие приложения часто создаются с использованием фреймворков типа Ionic или React Native, позволяющих использовать веб-технологии для разработки мобильных программ,
* прогрессивные веб-приложения (PWA) – это современные решения, объединяющие преимущества веб- и мобильных приложений. PWA предлагают оффлайн-режим, нотификации, возможность установки на рабочий стол и удобство обновлений, что делает их высокофункциональными и удобными для конечного пользователя.

Особенности веб-приложений многогранны. Среди основных преимуществ можно выделить:

* доступность. Пользователю не требуется установка специального программного обеспечения, достаточно стандартного браузера,
* кросс-платформенность. Веб-приложение работает на различных устройствах и операционных системах, что существенно расширяет аудиторию,
* централизованное управление. Обновления и исправления применяются на сервере, и все пользователи сразу получают актуальную версию,
* масштабируемость. Благодаря распределённой архитектуре и возможности использования облачных вычислений, веб-приложения легко масштабируются для обработки растущего объёма данных и запросов.

Однако есть и определённые минусы. Среди недостатков можно отметить:

* зависимость от соединения с интернетом. Для корректной работы требуется стабильный доступ к сети, что не всегда возможно в отдалённых регионах или при перебоях,
* вопросы безопасности. Веб-приложения становятся объектом атак из-за публичного доступа, поэтому обеспечение безопасности данных и защиты от внешних угроз становится критически важным,
* производительность. В некоторых случаях, особенно при работе с тяжелым клиентским JavaScript или большим объёмом данных, может наблюдаться снижение производительности по сравнению с нативными приложениями.

Таким образом, веб-приложения представляют собой гибкий и доступный инструмент для реализации самых разнообразных бизнес-задач, от простых сервисов до сложных корпоративных систем. Их архитектура и возможности постоянно развиваются, что позволяет интегрировать в них новые технологии и подходы, включая современные методы оптимизации и анализа, такие как задачи смешано-целочисленного линейного программирования. Именно поэтому разработка веб-приложения для решения таких задач является актуальной и перспективной областью исследований, способной существенно улучшить процессы принятия решений и оптимизации в различных сферах деятельности.

## **1.2 Задачи смешано-целочисленного линейного программирования**

Задачи смешано-целочисленного линейного программирования представляют собой класс оптимизационных задач, где целевая функция и ограничения выражены линейными зависимостями, а переменные могут быть как непрерывными, так и дискретными (обычно целочисленными). Формальная модель MILP-задачи имеет следующий общий вид:

где — линейная целевая функция;

— коэффициенты, которые задаются в ℝ.

где — коэффициенты, которые задаются в ℝ;

— означает один из следующих вариантов: ≤, ≥ или =;

— коэффициенты, которые задаются в ℝ.

где — переменные, значение которых должно быть целым.

где — переменные, значение которых должно быть вещественным.

Такое смешение типов переменных позволяет моделировать реальные ситуации, в которых помимо количественных (непрерывных) решений требуется также принимать решения о выборе вариантов или о допустимости целых значений – например, решение о том, включать ли конкретный производственный цех в план или нет, или сколько единиц продукции производить в условиях ограничения на ресурсы.

Особенности MILP-задач заключаются в следующем:

* линейность модели. Как целевая функция, так и все ограничения выражаются через линейные комбинации переменных. Это позволяет применять теоретические методы линейного программирования, но усложняется наличие целочисленных переменных,
* смешанный характер переменных. Наличие целочисленных переменных делает задачи NP-трудными, так как пространство возможных решений дискретное и обычно экспоненциально растёт с увеличением числа таких переменных,
* применимость для широкого спектра задач. MILP модели находят применение в логистике, планировании производства, управлении запасами, финансовом моделировании и многих других сферах, где необходимо учитывать как количественные, так и дискретные критерии.

Применение MILP-моделирования очень популярно в тех случаях, когда необходимо принять решения при наличии ограниченных ресурсов. Например, в сфере транспортной логистики MILP позволяет определить оптимальные маршруты грузоперевозок с учётом целых маршрутов и выбором из множества альтернатив, в планировании производства – оптимально распределить задания на производственные линии, где некоторые переменные представляют собой «да» или «нет» решения, а другие – непрерывные потоки материалов.

Среди методов решения MILP-задач можно выделить:

* классический метод ветвей и границ (branch and bound), который последовательно делит большое пространство решений на подсистемы и отсеивает неэффективные ветви,
* метод ветвей и усечений (branch and cut), представляющий собой улучшение метода branch and bound с использованием дополнительных неравенств (усечений), которые ускоряют процесс поиска,
* различные эвристические и метаэвристические алгоритмы, такие как генетические алгоритмы или алгоритм имитации отжига, применяемые для очень больших или сложных MILP-задач, когда точные методы оказываются вычислительно затратными.

Более подробно эти методы описаны, например, в работе Немхаузера и Уолси [1].

Стоит отметить, что MILP-задачи, будучи NP-трудными, в общем случае не имеют полиномиального алгоритма решения. Современные решатели, такие как CBC, SCIP, Gurobi и CPLEX, используют комбинацию различных методов, включая техники ветвей и границ, предусекционные методы и мультипликаторы Лагранжа для уменьшения исследуемого пространства, что позволяет находить оптимальные или приближённые решения за разумное время даже для относительно крупных моделей.

На практике эффективность MILP моделей определяется не только качеством математической формулировки задачи, но и выбранным решателем, а также приемами предобработки и анализа чувствительности модели. Именно поэтому разработка веб-приложения для решения таких задач требует гибкого подхода: оно должно предоставлять возможность выбора решателя, настройки параметров модели и демонстрации анализа влияния изменений входных данных на оптимальное решение.

Таким образом, MILP-проблемы являются универсальным инструментом для оптимизации в различных сферах деятельности. Они позволяют интегрировать дискретные и непрерывные решения в единую модель и, благодаря современным алгоритмам, находить эффективные решения для комплексных и многокомпонентных ситуаций, что делает их незаменимыми как в исследовательской, так и в прикладной деятельности.

# **2 АРХИТЕКТУРА ПРОЕКТА**

## **2.1 Архитектура бэкенда**

Архитектура бэкенда представляет собой системное решение, обеспечивающее обработку бизнес-логики, хранение данных, взаимодействие с внешними сервисами и предоставление API для клиентских приложений. При разработке бэкенд-части существует несколько вариантов архитектурных подходов, каждый из которых имеет свои преимущества и ограничения.

Монолитная архитектура.

Суть подхода: вся бизнес-логика, обработки запросов, работа с базой данных и коммуникация с внешними сервисами реализуются в одном приложении, развёрнутом как единое целое.

Преимущества:

* простота разработки и развёртывания, особенно для небольших и средних проектов,
* нет дополнительных накладных расходов для межсервисных коммуникаций, всё находится в одном кодовом пространстве,
* лёгкость отладки и тестирования за счёт наличия единой кодовой базы.

Недостатки:

* с увеличением проекта кодовая база становится сложной для поддержки и масштабирования,
* любое изменение или обновление требует перезагрузки всего приложения, что может привести к простоям,
* ограниченные возможности масштабирования: масштабируется вся система, даже если требуются дополнительные ресурсы только для отдельных функций.

Микросервисная архитектура.

Суть подхода: функциональность разделяется на независимые сервисы, каждый из которых отвечает за конкретную задачу (например, аутентификация, обработка оптимизационных вычислений, управление базой данных).

Преимущества:

* возможность масштабирования каждого сервиса отдельно в зависимости от нагрузки,
* повышенная отказоустойчивость: сбой одного сервиса не приводит к краху всей системы,
* более гибкая разработка, так как команды могут работать независимо над различными компонентами.

Недостатки:

* усложнение архитектуры за счёт межсервисного взаимодействия (например, через REST, gRPC или message queues),
* усиленные требования к инфраструктуре (контейнеризация, оркестрация, мониторинг),
* трудности при отладке и тестировании интеграционных сценариев.

В рамках настоящего проекта для решения задач смешано-целочисленного линейного программирования выбрана монолитная архитектура бэкенда. Такой выбор обусловлен следующими соображениями:

* простота разработки. Проект ориентирован на решение узкой прикладной задачи – оптимизации MILP, поэтому реализация всей бизнес-логики на одном серверном приложении позволяет быстро воплотить функциональные требования, не усложняя архитектуру межсервисными коммуникациями,
* централизованность функционала. В рамках монолита удобно объединить обработку загрузки файлов, построение математических моделей с помощью Pyomo, выбор и интеграцию решателей (GLPK, CBC, SCIP и т.д.), а также работу с базой данных и механизм обмена данными посредством SSE. Такой подход облегчает единое управление состоянием и упрощает отладку,
* масштабируемость на начальном этапе. При умеренной нагрузке на систему монолитное приложение позволяет эффективно работать, а если в будущем возникнет необходимость расширения, его можно будет постепенно переписывать в микросервисную архитектуру.

Таким образом, монолитная архитектура является оптимальным выбором для данного проекта с точки зрения скорости разработки, простоты поддержки и достаточной масштабируемости для текущих задач. В будущем, при расширении проекта, возможно разделение на микросервисы по мере необходимости, но на данном этапе интегрированное решение позволяет сфокусироваться на основной функциональности – решении MILP-задач в веб-среде.

### **2.1.1 Технологии бэкенда**

В рамках реализации веб-приложения для решения задач смешано-целочисленного линейного программирования был выбран стек технологий, основанный на языке программирования Python. Это объясняется не только его выразительностью и богатой экосистемой, но и возможностью быстрого прототипирования оптимизационных моделей. В данном проекте задействованы следующие ключевые технологии:

Python.

Язык Python выбран благодаря своей универсальности и широкому спектру библиотек для численных расчетов, аналитики и веб-разработки. Его читаемость, динамическая типизация и большое сообщество разработчиков позволяют оперативно внедрять новые решения и поддерживать актуальность проекта.

В проекте используются современные версии Python, что обеспечивает совместимость с актуальными библиотеками и инструментами разработки.

Flask.

Фреймворк Flask обеспечивает легковесную и гибкую архитектуру веб-приложения. Благодаря своей минималистичной природе Flask позволяет разработчику самостоятельно выбирать необходимые компоненты и интегрировать их в единое решение [2].

В проекте Flask служит основой для обработки HTTP-запросов, организации маршрутизации, подключения middleware и реализации REST API, через которое осуществляется взаимодействие с фронтендом.

Преимущество Flask заключается в простоте настройки и широких возможностях расширения за счёт сторонних расширений.

Flask-CORS.

Для обеспечения корректного междоменных запросов (CORS – Cross-Origin Resource Sharing) используется расширение Flask-CORS. Оно позволяет разрешить доступ к API из различных источников, что особенно актуально при разработке распределённых систем, где фронтенд и бэкенд могут находиться на разных доменах или портах.

Это расширение минимальными усилиями устраняет проблемы, связанные с политикой одного источника браузеров, обеспечивая стабильное взаимодействие между клиентской и серверной частями.

Pyomo.

Основной инструмент для математического моделирования – это библиотека Pyomo, предназначенная для формулирования и решения задач оптимизации. Она позволяет описывать модели смешанного целочисленного линейного программирования в декларативном виде, а также интегрируется с различными решателями, такими как GLPK, CBC или SCIP [3].

Pyomo удовлетворяет требованиям по гибкости и масштабируемости модели, предоставляя возможность работы с линейными ограничениями, целевыми функциями и синтаксическим оформлением модели в виде объекта Python.

Решатели оптимизации (GLPK, CBC, SCIP).

По умолчанию в проекте используется решатель GLPK [4], однако предусмотрена возможность поддержания альтернативных решателей, таких как CBC и SCIP. Это даёт возможность пользователю выбирать наиболее подходящий инструмент в зависимости от специфики решаемой задачи.

Использование разных решателей позволяет провести сравнительный анализ эффективности, а также обеспечить более надёжное и гибкое решение MILP-задач при разных условиях.

SQLAlchemy и PostgreSQL.

Для работы с базой данных применяется SQLAlchemy – ORM-фреймворк, позволяющий абстрагироваться от специфики SQL-синтаксиса и обеспечить удобное манипулирование данными через объектную модель [5].

В качестве СУБД используется PostgreSQL, что обеспечивает высокую надёжность, масштабируемость и гибкость при хранении результатов оптимизационных расчётов, условий задач и логов работы приложения.

Такой подход позволяет реализовать централизованное ведение данных, а также легко интегрировать дополнительные возможности по аналитике и отчетности.

Pandas и openpyxl.

Для чтения и обработки входных Excel-файлов задействованы библиотеки Pandas и openpyxl. Pandas обеспечивает мощные инструменты работы с табличными данными, а openpyxl служит движком для чтения файлов формата xlsx.

Эти технологии позволяют корректно обрабатывать, валидировать и конвертировать данные, загруженные пользователем, что является важным этапом подготовки данных для оптимизационной модели.

Многопоточность.

В проекте используется модуль concurrent.futures с ThreadPoolExecutor для реализации фоновой обработки длительных вычислений. Такой подход позволяет освобождать основной поток сервера для обработки новых запросов, одновременно выполняя вычислительно затратные задачи в фоне.

Это позитивно сказывается на отклике системы и обеспечивает удобную реализацию SSE-уведомлений о ходе выполнения оптимизации.

В итоге выбранный стек технологий сочетает в себе проверенные временем инструменты и современные подходы, позволяющие создать надёжное и масштабируемое веб-приложение. Монолитная архитектура, построенная на Flask, демонстрирует высокую скорость разработки, централизованное управление бизнес-логикой и интеграцию с мощной библиотекой Pyomo, что позволяет реализовывать сложные MILP задачи. Использование ORM SQLAlchemy и СУБД PostgreSQL обеспечивает устойчивость хранения данных, а применение Pandas и openpyxl делает возможным оперативное считывание данных из Excel-файлов. В совокупности, данные технологии создают основу для разработки эффективного и гибкого веб-приложения, способного решать задачи оптимизации в реальном времени и удовлетворять потребности современных пользователей.

### **2.1.2 Структура данных**

База данных состоит из одной таблицы tasks, состоящей из 9 полей:

1. поле id — id записи в таблице,
2. поле task\_id — id задачи, который назначается каждой задаче, при получении условий на сервере,
3. поле conditions – исходное условие задачи. Формат хранения – JSON,
4. поле solution – решение задачи. Формат хранения – JSON. Может произойти, например, ошибка, тогда решатель не вернет нам валидное решение, поэтому это поле в таблице может быть пустым,
5. поле solver – решатель, который выбрал пользователь,
6. поле conditions\_excel - поле для хранения сгенерированного Excel файла в виде base64 строки,
7. поле upload\_time - время загрузки задачи на сервер,
8. поле solve\_time – время завершения решения задачи,
9. поле canceled – булевый флаг, который символизирует об отмене пользователем решения задачи.

Подробная структура данных изображена на рисунке 1.

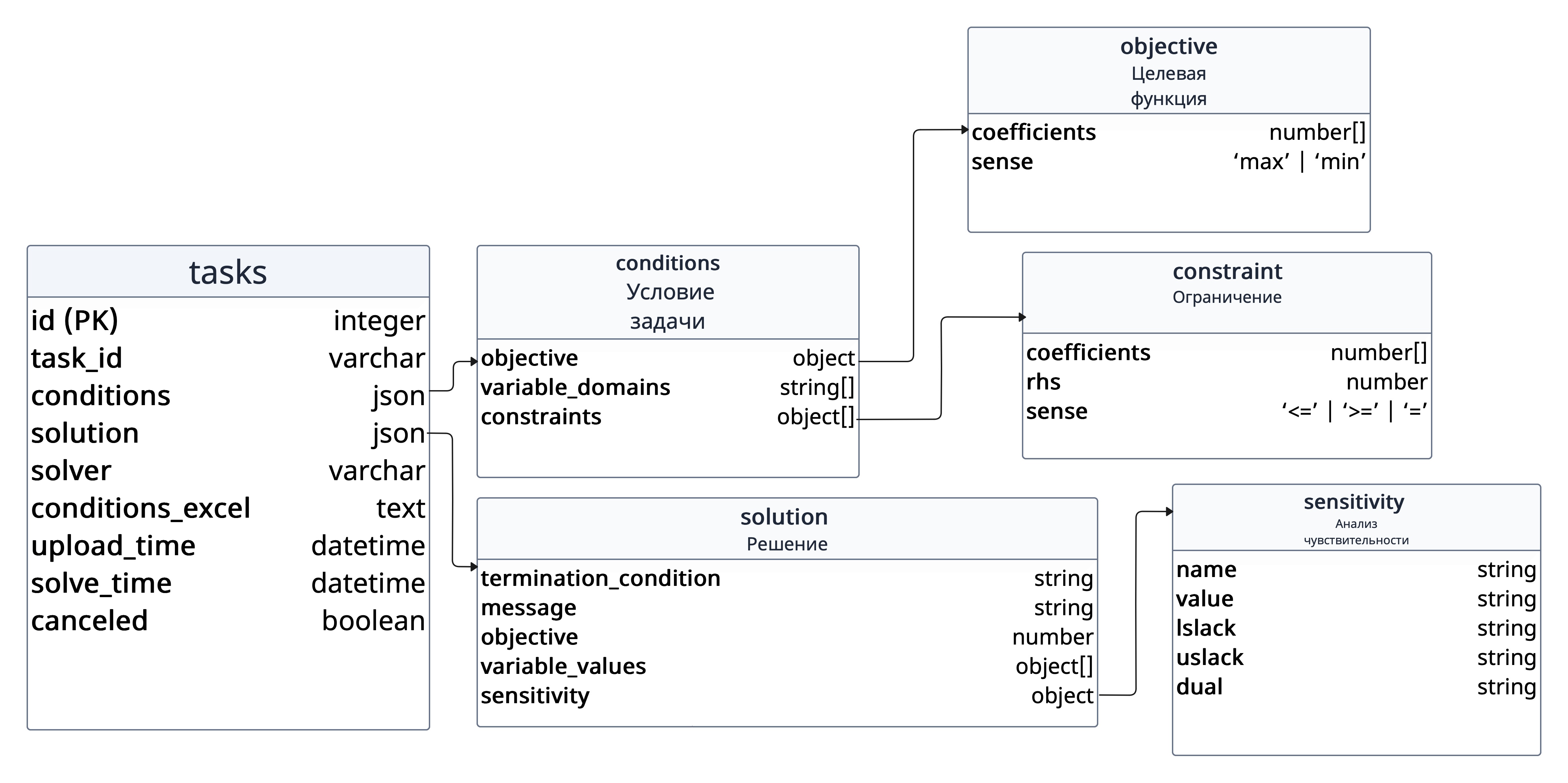


Рисунок 1 — Структура данных

### **2.1.3 Реализация серверной части**

При инициализации приложения происходит создание приложения с помощью фреймворка Flask, CORS настройка и подключение к базе данных PostgreSQL (рисунок 2). Адрес СУБД берется из переменной окружения DATABASE\_URL.

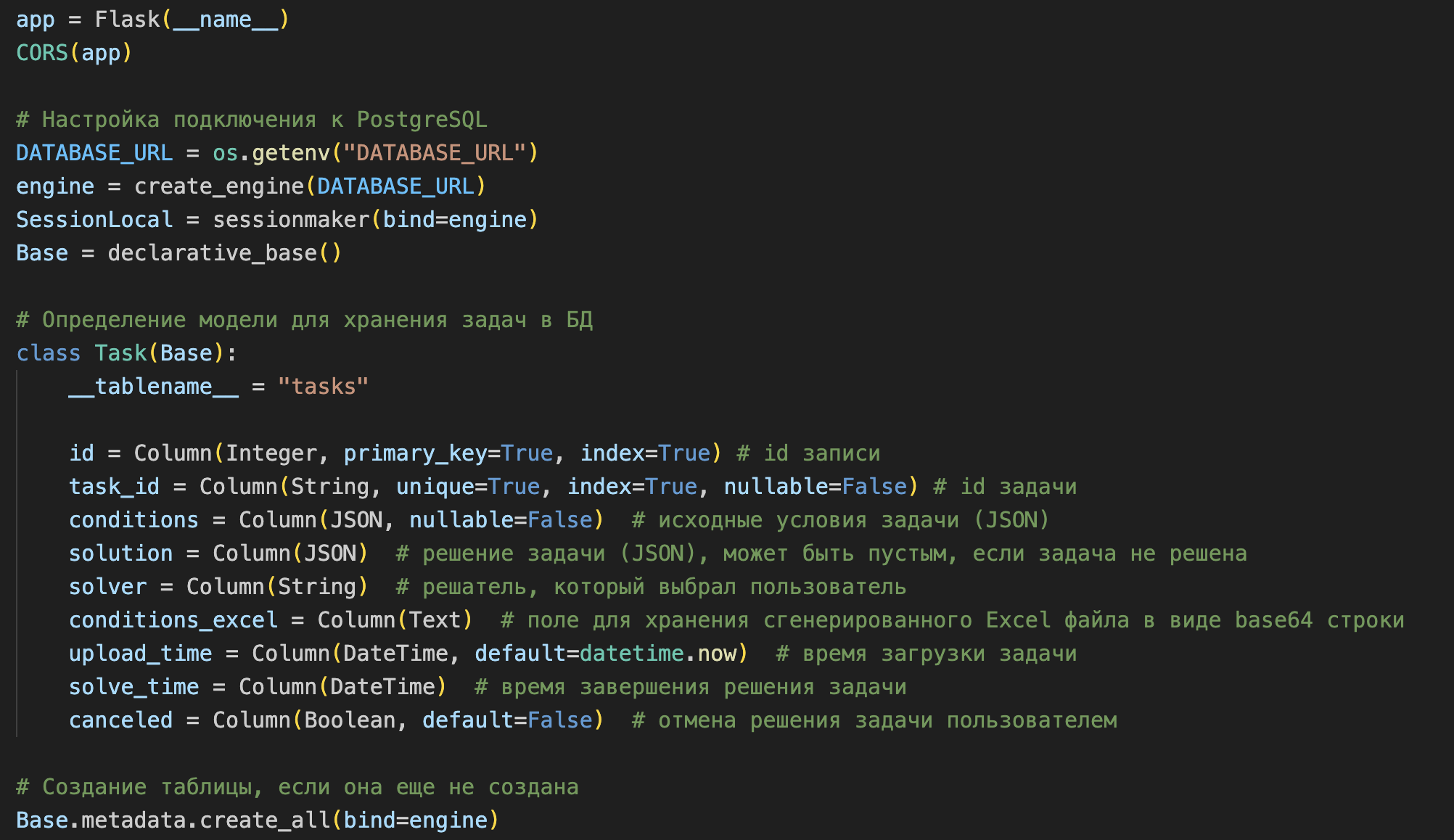


Рисунок 2 — Создание приложения и первичная настройка

Для загрузки задачи, которая была создана с помощью полей ввода на сайте, создан эндпоинт «/task» и методом POST (рисунок 3). В функции обработчике этого эндпоита происходит извлечение из запроса данных о самой задаче и о том какой решатель будет использоваться. Далее из данных о задаче создается модель, чтобы передать ее в выбранный решатель библиотеки Pyomo. Также в функции происходит генерация task\_id, конвертация условий в формат excel, запись всей текущей информации в базу данных и вызов фоновой функции решателя. Если все исполнилось корректно, то возвращаем ответ, в котором содержится task\_id, чтобы клиент мог опрашивать сервер о статусе выполнения конкретной задачи.

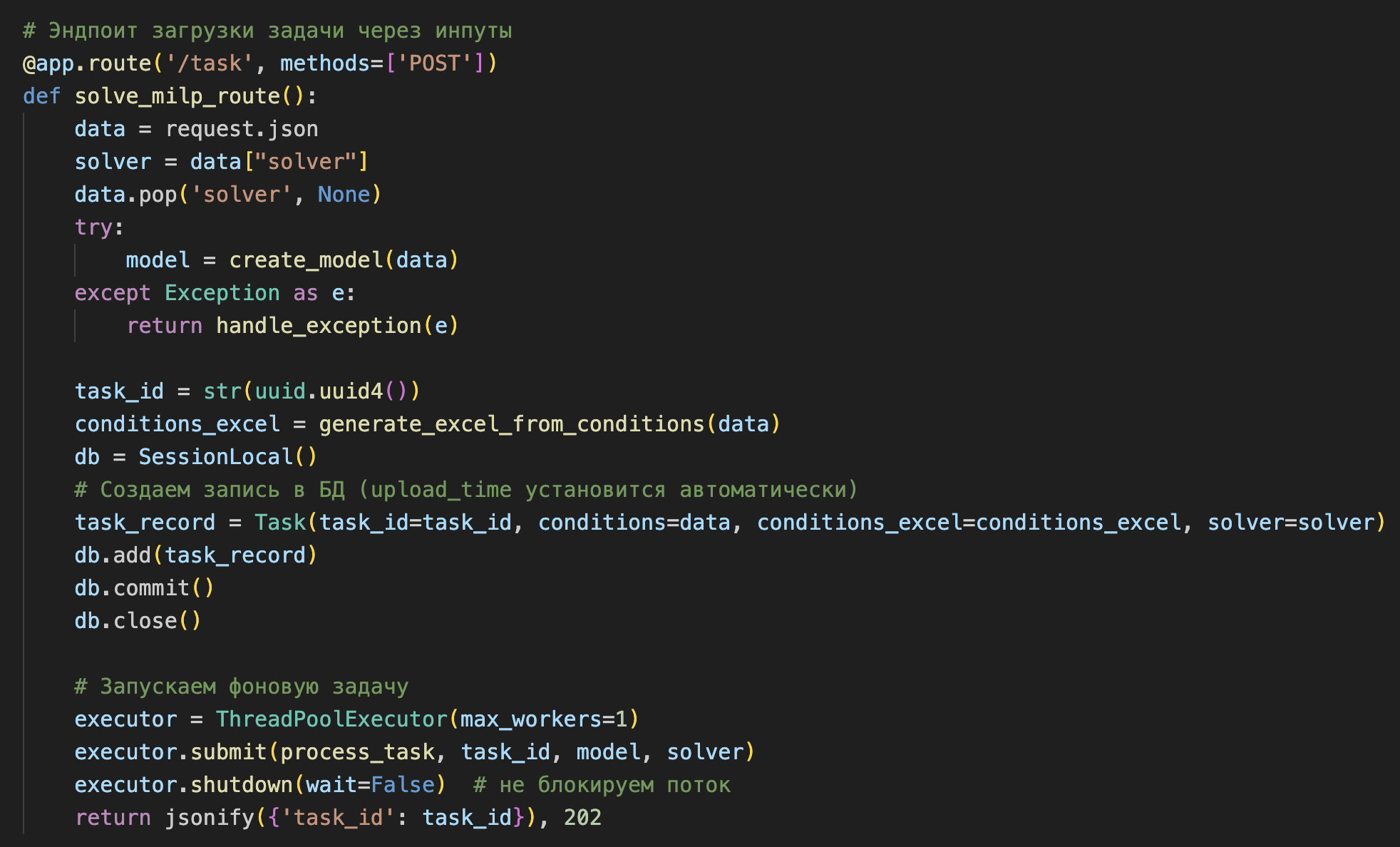


Рисунок 3 — Эндпоинт загрузки задачи через инпуты

Еще один способ загрузки задачи – с помощью файла формата xlsx. Для этого был разработан отдельный эндпоинт, доступный по пути «/task/excel», с методом POST. В функции обработчике происходит парсинг данных с помощью pandas, валидация данных, конвертация в json формат, создание модели, конвертация условия в base64 строку, запись всей текущей информации в базу данных и вызов той же фоновой функции решателя, что и в предыдущем эндпоинте. Если все прошло успешно, то возвращается task\_id задачи. Реализация этого эндпоинта изображена на рисунке 4.

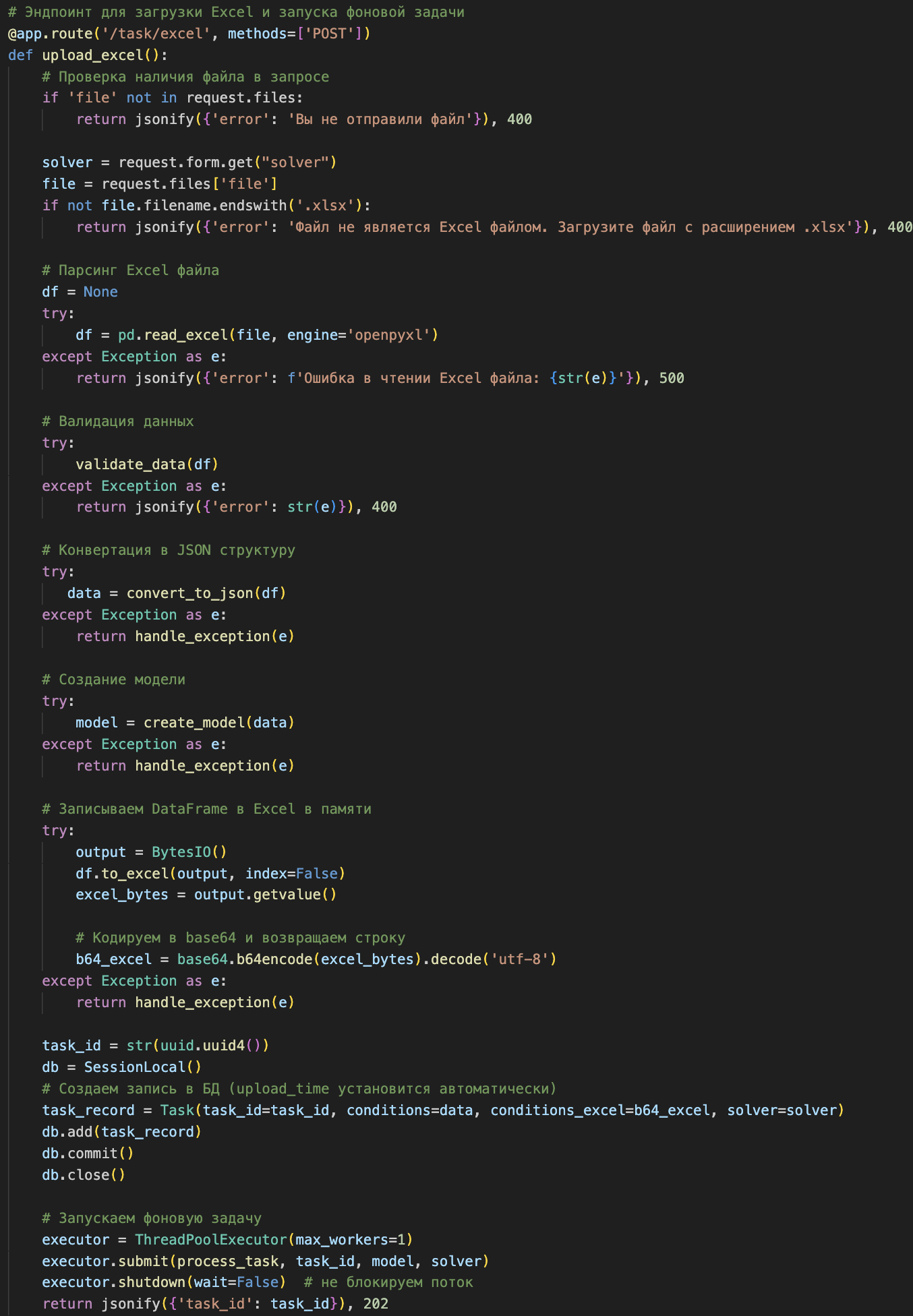


Рисунок 4 — Эндпоинт загрузки задачи через файл

Для получения информации о текущем статусе решении задачи, доступен эндпоинт «/task/task\_progress/<task\_id>» с методом GET (рисунок 5). Клиент создает SSE соединение по этому адресу и получает каждую секунду состояние решения задачи. Для этого в функции event\_stream каждую секунду происходит обращение к базе данных по task\_id, если время решения не установлено, значит задача в процесс решения, и на клиент отправляется соответствующее сообщение. Если solve\_time известен, то на клиент отправляется информация о решении, условии задачи, решатели, которым производилось решение и времени решении. В случае обнаружения ошибки, она отправляется на клиент.

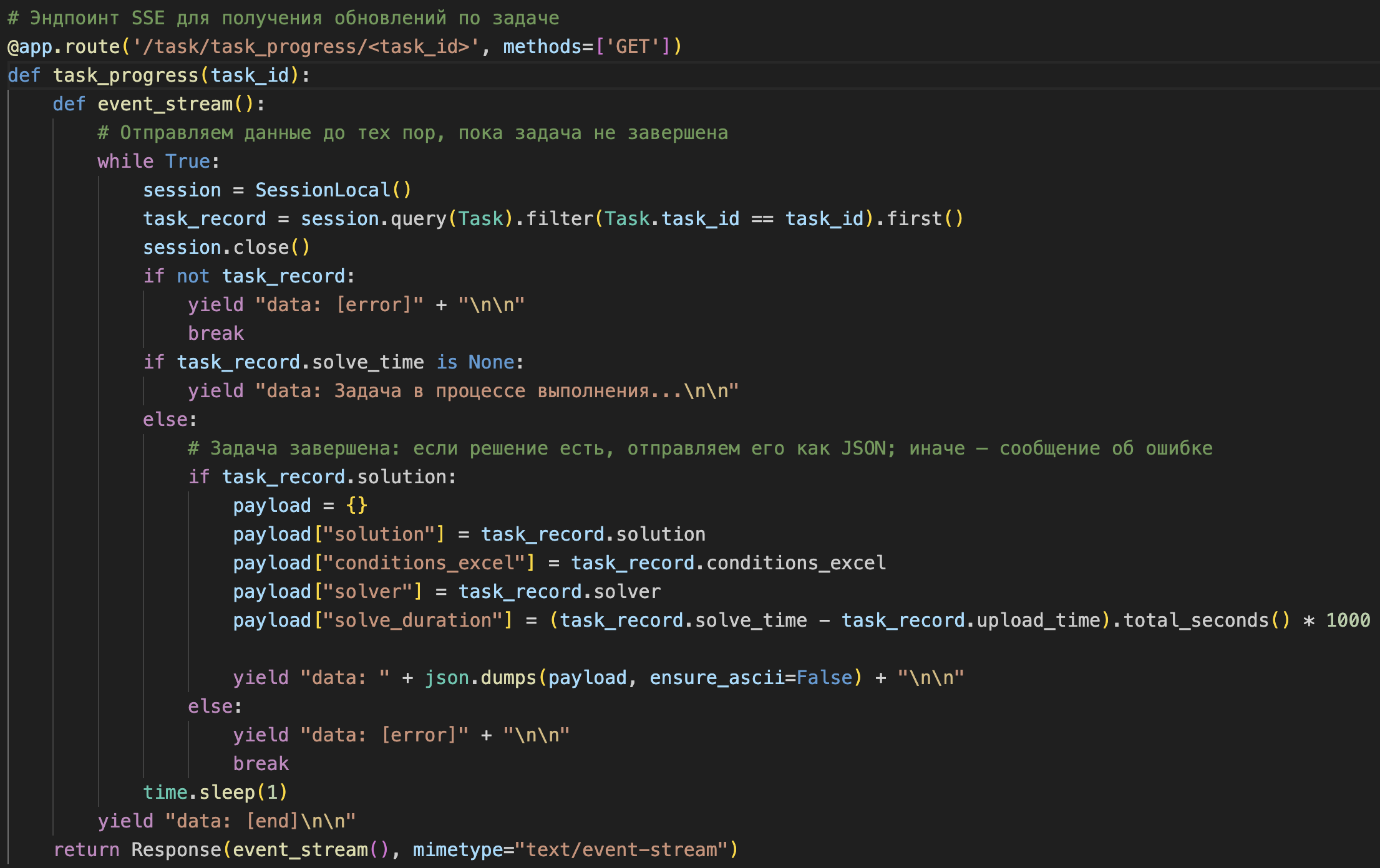


Рисунок 5 — Эндпоинт для получения обновлений по задаче

У пользователя есть возможность остановить решение загруженной задачи, например, если его не устраивает, что задача решается слишком долго. Для этого создан эндпоинт «/task/cancel\_task/<task\_id>» c методом POST (рисунок 6). Его функционал заключается в установлении булевого флага поля canceled в значение True. После чего клиенту возвращается сообщение об успешности отмены решения.

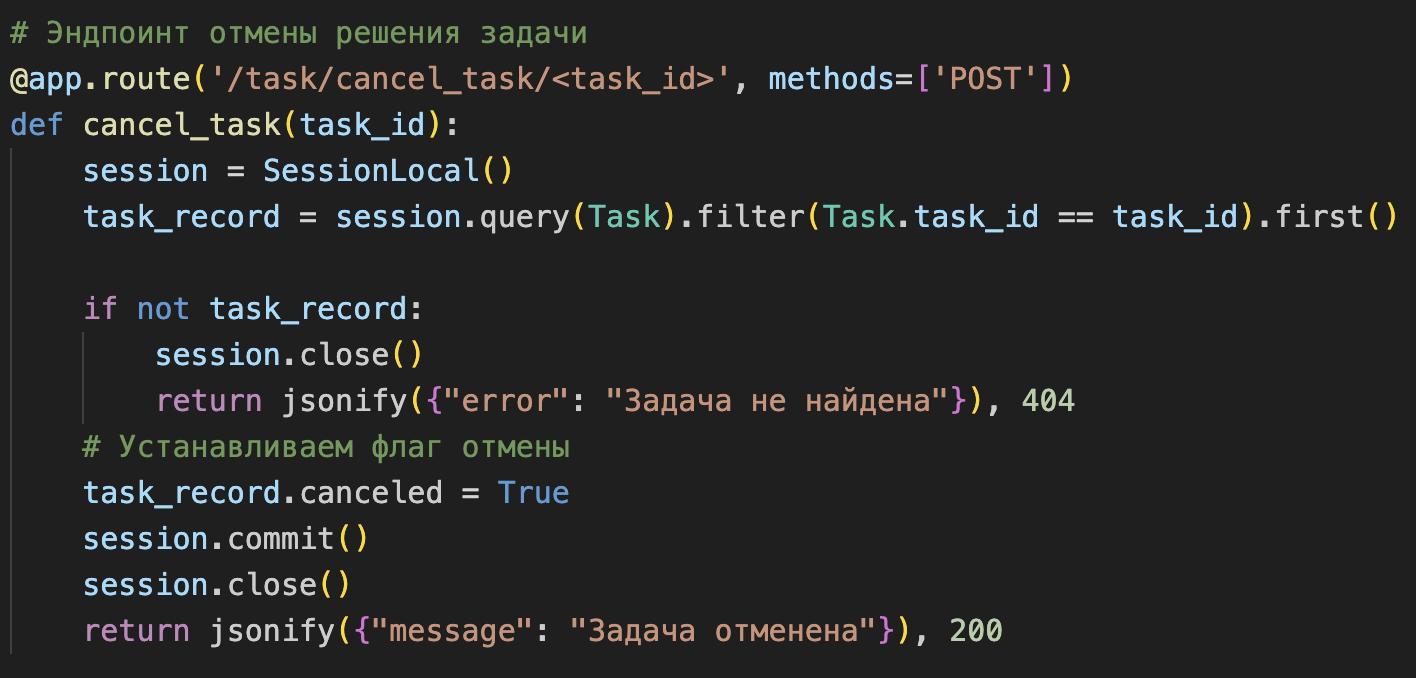
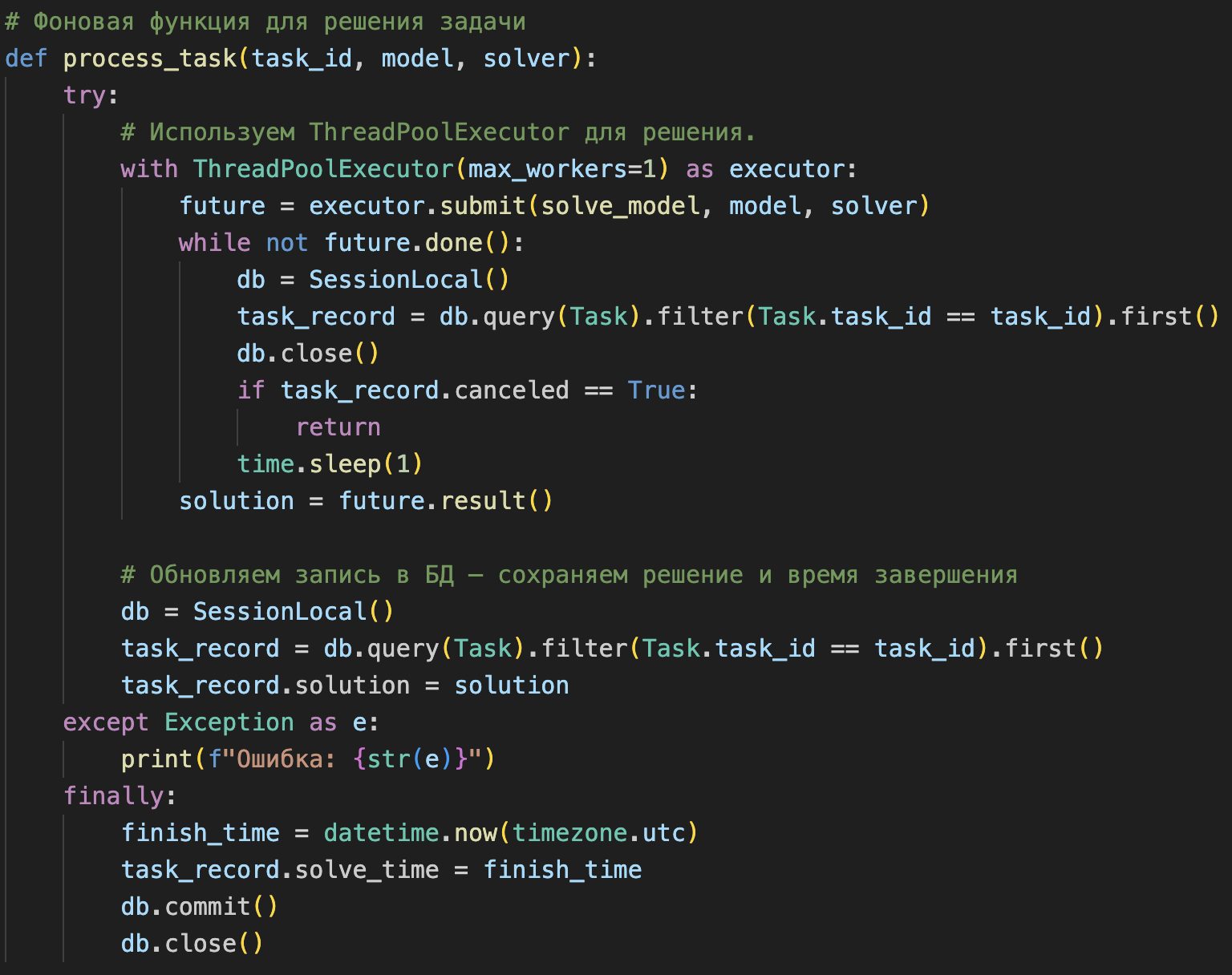


Рисунок 6 — Эндпоинт отмены решения задачи

Непосредственно остановка решения происходит в фоновой функции решения - в цикле, пока решение задачи не завершено, проверяем у текущей задачи в таблице поле canceled. Если флаг равен True, то мы останавливаем решение. Если решение не прервали, мы сохраняем результаты в базу данных. При возникновении ошибки логируем ее. Описанная логика отражена на рисунке 7.

 Рисунок 7 — Фоновая функция решения задачи

На рисунке 8 отображена файловая структура серверной части проекта.



Рисунок 8 — Фоновая функция решения задачи

## **2.2 Архитектура фронтенда**

Фронтенд представляет собой ту часть системы, с которой взаимодействует конечный пользователь. При разработке клиентских приложений существует несколько архитектурных подходов, позволяющих организовать код в соответствии с принципами масштабируемости, удобства сопровождения и повторного использования компонентов. Рассмотрим основные подходы и их особенности:

Монолитный подход.

Суть: все элементы пользовательского интерфейса, логика обработки данных и представления объединены в единое целое без ярко выраженной декомпозиции на самостоятельные компоненты.

Преимущества:

* простота первоначальной разработки и быстрая сборка прототипа,
* нет необходимости в сложной настройке модульной системы или сборщика, так как все находится в одном проекте.

Недостатки:

* при росте приложения код становится сложно поддерживать, поскольку функциональные слои тесно переплетены,
* отсутствие чёткого разделения обязанностей приводит к проблемам с масштабированием и модификацией проекта,
* трудности с повторным использованием компонентов в других проектах.

MVC (Model-View-Controller).

Приложение разделяется на три слоя: модель (данные и бизнес-логика), представление (UI и отрисовка) и контроллер (обработка пользовательских запросов и установка связей между моделью и видом).

Преимущества:

* чёткое разделение ответственности делает код более структурированным и легко поддерживаемым,
* облегчается тестирование каждого слоя отдельно,
* упрощается повторное использование компонентов при правильной декомпозиции.

Недостатки:

* может потребоваться дополнительное время на настройку и проектирование архитектуры,
* в современных динамических веб-приложениях правила MVC могут быть не всегда оптимальны для высокой интерактивности и гибкости.

Компонентный (модульный) подход.

Суть: приложение разбивается на небольшие, независимые модули (компоненты, виджеты), каждый из которых инкапсулирует свою логику работы с данными, отрисовки и взаимодействия с пользователем. Компоненты могут быть переиспользованы в разных частях приложения, а поддержка глобальных слоёв (например, вопросов взаимодействия с API или общих утилит) осуществляется через отдельные модули.

Преимущества:

* высокая степень повторного использования кода: один и тот же компонент можно применять в нескольких местах,
* локализация логики: каждый виджет «знает» только о своих внутренностях, что уменьшает вероятность возникновения ошибок при изменении функционала одного из модулей,
* упрощённое тестирование и отладка благодаря инкапсуляции кода внутри модулей,
* лёгкость масштабирования приложения и поддержки разных функциональных блоков независимо друг от друга.

Недостатки:

* первоначальная настройка модульной структуры требует тщательного проектирования и архитектурных решений,
* может возникать избыточность, если структура компонентов не продумана и не стандартизирована, что приводит к дублированию функционала между модулями.

Выбранный подход во фронтенд-части проекта основывается на модульном подходе. В данном решении код разделён на виджеты – независимые модули, отвечающие за конкретные функциональные участки (например, ввод условий задачи, отображение результатов, управление загрузкой файлов). Каждый виджет инкапсулирует как свои визуальные компоненты, так и внутреннюю логику работы с данными, что позволяет отделить отрисовку от бизнес-логики.

Кроме того, в архитектуре присутствуют общие слои, такие как модуль API для взаимодействия с бэкендом и модуль shared, включающий общие утилиты, компоненты и конфигурации, востребованные в отдельных виджетах. Такое разделение способствует централизованному управлению и упрощает повторное использование кода.

В итоге, модульный подход во фронтенде позволяет обеспечить гибкость, масштабируемость и высокое качество кода, что особенно важно для проектов, где интерфейс должен эффективно взаимодействовать с динамичными оптимизационными алгоритмами бэкенда. Этот архитектурный выбор облегчает поддержку системы, уменьшает связанные риски и помогает в быстром прототипировании и дальнейшей эволюции приложения по мере расширения функционала и появления новых требований.

### **2.2.1 Технологии фронтенда**

В реализации клиентской части проекта использован современный и гибкий стек технологий, позволяющий создавать высокопроизводительные и удобные в обслуживании веб-приложения. В состав этого стека входят Vite, TypeScript, React, Gravity UI и RTK Query. Каждая из этих технологий вносит свой вклад в быстродействие, надежную типизацию, удобство в разработке и эффективное взаимодействие с серверной частью.

Vite:

* выступает в роли современного сборщика и инструмента для разработки фронтенда. Он отличается мгновенной загрузкой модулей, благодаря нативному использованию ES-модулей в браузере и быстрому серверу разработки [6],
* преимуществом Vite является его высокая скорость старта проекта и обработки изменений в режиме реального времени, что существенно ускоряет процесс отладки и прототипирования,
* благодаря минимальной конфигурации и высокой оптимизации результирующего файла, Vite становится идеальным выбором для разработки модульных приложений, требующих частых перезагрузок и высокой отзывчивости.

TypeScript:

* это надстройка над JavaScript, предоставляющая статическую типизацию и возможности объектно-ориентированного программирования. Использование данного языка позволяет обнаруживать ошибки во время компиляции, что снижает вероятность возникновения ошибок при выполнении и улучшает качество кода [7],
* типизация помогает разработчикам легко ориентироваться в большом объеме модульного кода и способствует документированию API компонентов, что крайне важно при работе в команде и поддержке проекта на длительном сроке.

React:

* одна из самых популярных библиотек для создания пользовательских интерфейсов. Он строится на компонентном подходе, что позволяет разбить приложение на небольшие независимые блоки, каждый из которых инкапсулирует свою логику и представление.
* преимущество React заключается в его виртуальном DOM, который оптимизирует процесс обновления интерфейса, а также в большом сообществе, предоставляющем готовые решения для сложных задач. Это позволяет быстро реализовывать интерактивные и динамические компоненты, адаптированные под нужды конечного пользователя [8].

Gravity UI:

* представляет собой систему пользовательских интерфейсов, разработанную для упрощения создания стильных и отзывчивых компонентов. Ее использование позволяет стандартизировать внешний вид приложения, обеспечивая единообразие элементов интерфейса и повышение удобства использования [9],
* предлагает набор готовых компонентов, таких как кнопки, формы, карточки и другие, что ускоряет процесс разработки дизайна и обеспечивает соответствие современным трендам в веб-дизайне,
* хорошо интегрируется с React, что позволяет быстро настраивать и адаптировать визуальные компоненты под индивидуальные требования проекта.

RTK Query:

* это современный инструмент, входящий в состав Redux Toolkit, предназначенный для управления состоянием, связанным с асинхронными запросами. Он упрощает процесс взаимодействия с API и позволяет централизованно организовать запросы на сервер, их кэширование и обновления данных в приложении [10],
* благодаря RTK Query осуществляется эффективное и надежное подключение к серверной части, минимизируется избыточность кода при реализации API-запросов, а также повышается отзывчивость и консистентность данных в пользовательском интерфейсе,
* использование данного инструмента особенно актуально для проектов, где требуется динамическое обновление информации (например, получение статуса решения оптимизационной задачи), что позволяет обеспечить непрерывное взаимодействие между клиентской и серверной частями.

В совокупности, выбранный стек технологий обеспечивает гибкость, высокую производительность и масштабируемость фронтенда. Vite и TypeScript делают разработку более быстрой и надежной, а React с компонентным подходом и Gravity UI позволяют создавать современный, единообразный и отзывчивый интерфейс, который легко поддерживать и расширять. RTK Query обеспечивает эффективное управление асинхронными данными, упрощая процесс интеграции с сервером. Такое гармоничное сочетание технологий способствует быстрому прототипированию и реализации функциональности, необходимой для решения задач смешано-целочисленного линейного программирования в веб-приложении.

### **2.2.2 Реализация клиентской части**

Начальной точкой приложения является создание главного компонента, в котором находятся все виджеты приложения (рисунок 9). В этом месте также подключаются компоненты, которые обеспечивают доступ в любой части кода к данным, стилям темы и компоненту отображения ошибки.

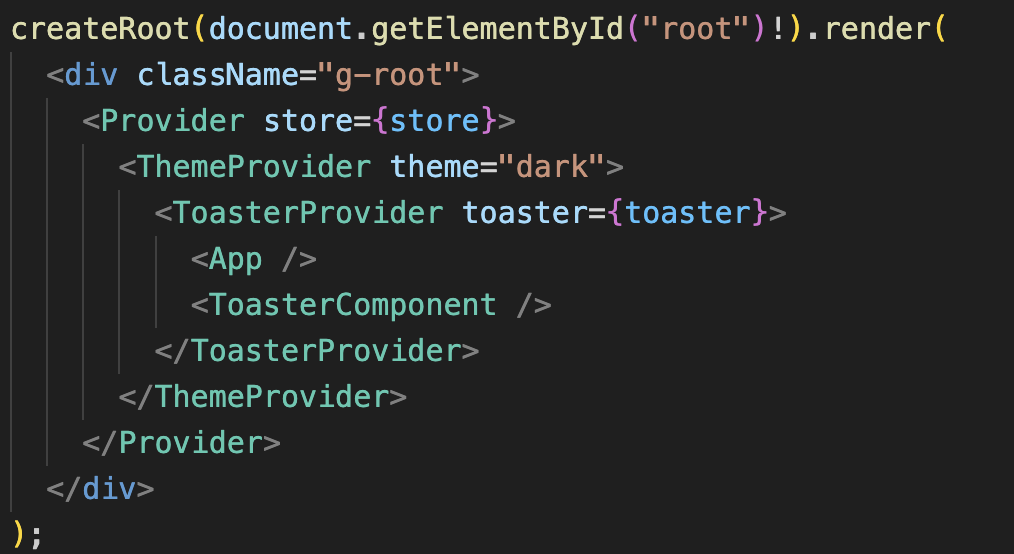


Рисунок 9 — Точка входа клиентского приложения

В компоненте App подключаются все реализованные виджеты, и задается структура веб-страницы (рисунок 10).

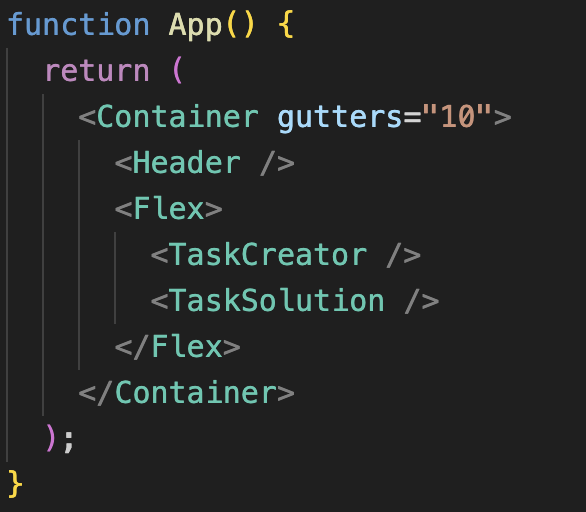


Рисунок 10 — Компонент App

Для того, чтобы пользователь мог ввести условия своей задачи, был реализован виджет TaskCreator. В нем реализована форма ввода параметров задачи, поддержана валидация введенных коэффициентов, если пользователь ввел некорректные данные, возле поля ввода будет выведено информативное сообщение об ошибке и будет заблокирована возможность отправки задачи на сервер до исправления всех ошибок. Функция валидация данных изображена на рисунке 11.



Рисунок 11 — Функция валидации в клиентской части

Помимо формы с полями ввода, компонент TaskCreator импортирует отдельный виджет ExcelUploader, который позволяет отправлять условия задачи в формате Excel. Этот виджет включает себя кнопку, при нажатии на которую открывается модальное окно с требованиями к файлу и кнопкой загрузки файла.

Для того, чтобы была возможность сохранять данные пользователя в приложении, библиотекой redux предоставляется функция createSlice, которая позволяет создать локальное хранилище данных, также в ней описываются функции, с помощью которых можно изменять данные. На рисунке 12 изображено использование такой функции в виджете ExcelUploader.



Рисунок 12 — Создание хранилища данных для ExcelUploader

После успешной отправки задачи на сервер, мы получаем от него id задачи. Так как решение может затянуться, сохраняем id задачи в localStorage – хранилище данных, предоставляемое каждым браузером, позволяющее сохранить данные в виде ключ-значение даже после перезагрузки страницы и закрытия вкладки. После этого открываем SSE соединение, которое позволяет нам получать сообщения от бэкенда с некоторой периодичностью, эти сообщения будут содержать информацию либо о том, что задача еще в процессе решения, тогда мы продолжаем показывать состояние загрузки, либо конечное решение, которое мы показываем пользователю.

Для отображения данных, которые нам прислал бэкенд был реализован виджет TaskSolution. Если бэкенд, успешно вернул нам решение задачи, мы должны максимально информативно и удобно отобразить все данные. В случае ошибки, которая могла возникнуть как при отправке, так и при валидации на сервере, отображаем ошибку. Решение может затянуться, поэтому мы должны показывать пользователю, что решение находится в процессе. Для этого воспользуемся готовым компонентом Progress из библиотеки gravityUI. На рисунке 13 показано, как он используется в коде виджета TaskSolution.

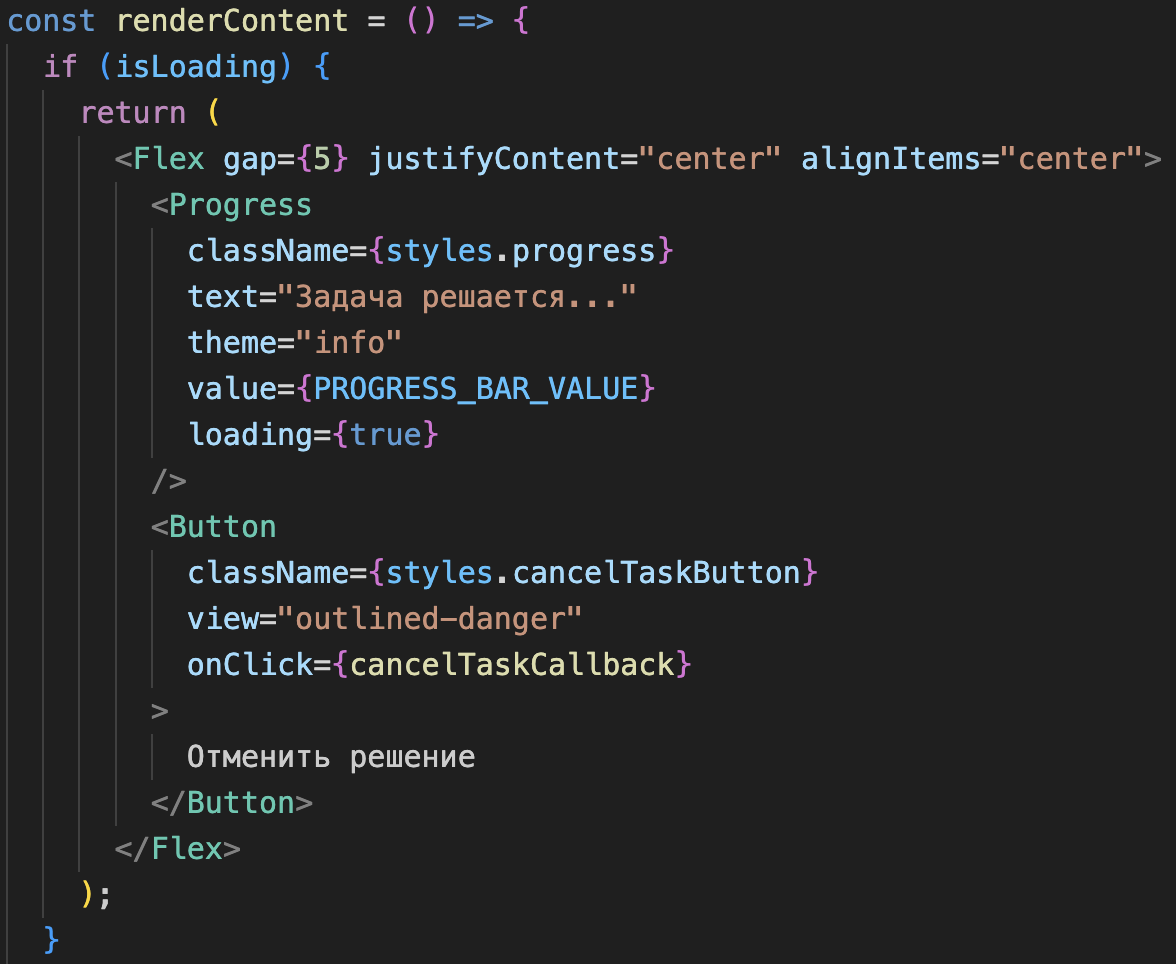


Рисунок 13 — Использование готового компонента Progress

Файловая структура фронтенда указана на рисунке 14.

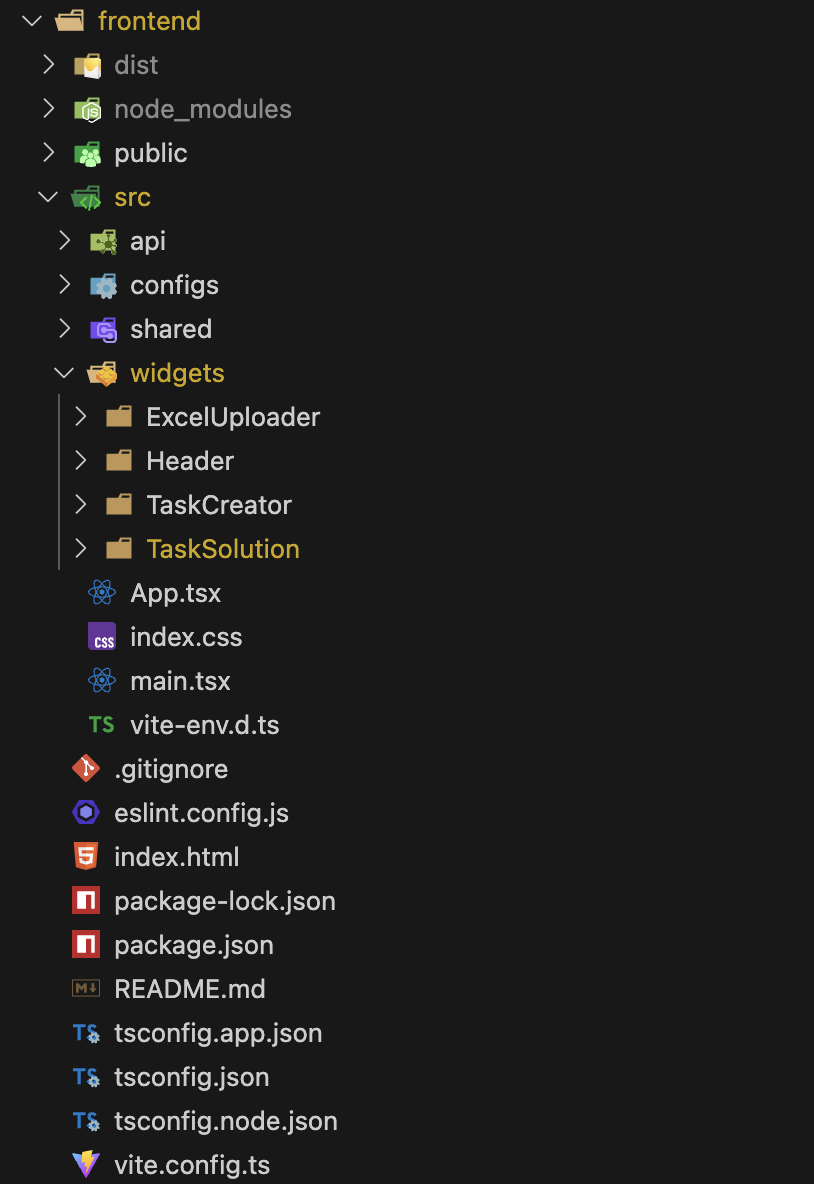


Рисунок 14 — Файловая структура фронтенда

## **2.3 Модульное взаимодействие**

Схема модульного взаимодействия, которая включает в себя визуальную и серверную части, отображена на рисунке 15.

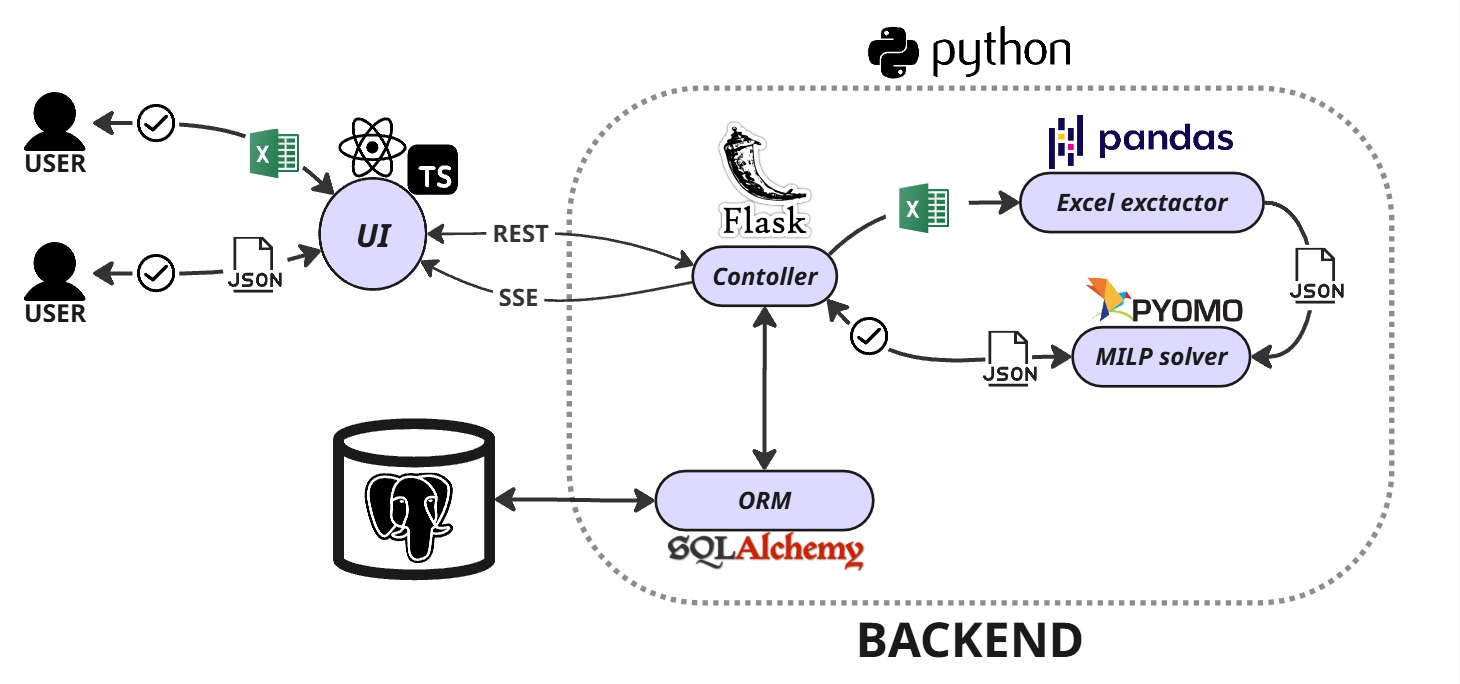


Рисунок 15 — Модульное взаимодействие

# **3 ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

## **3.1 Формат входных данных**

Для загрузки задачи на сервер по адресу «/task» используется формат данных, описанный на рисунке 16.



Рисунок 16 — Формат входных данных эндпоинта «/task»

Формат входных данных при загрузке задачи с помощью файла по адресу «/task/excel» представляет собой объект FormData с полями, которые представлены на рисунке 17.

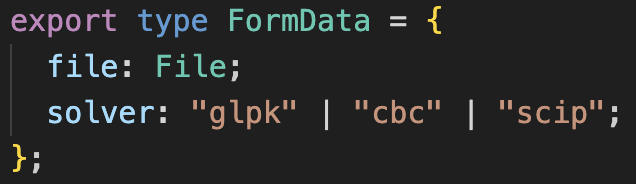


Рисунок 17 — Формат входных данных эндпоинта «/task/excel»

Для отмены решения в параметрах маршрутизации самого запроса строкой передается taskId. Таким образом, эндпоинт принимает вид «task/cancel/{taskId}». Аналогичный параметр передается в эндпоинте получения статуса решения: «/task/task\_progress/{taskId}».

## **3.2 Пользовательские сценарии**

В результате мы получили одностраничный сайт, который предоставляет удобный интерфейс загрузки задач смешано-целочисленного линейного программирования. На рисунке 18 изображено разработанное веб-приложение.

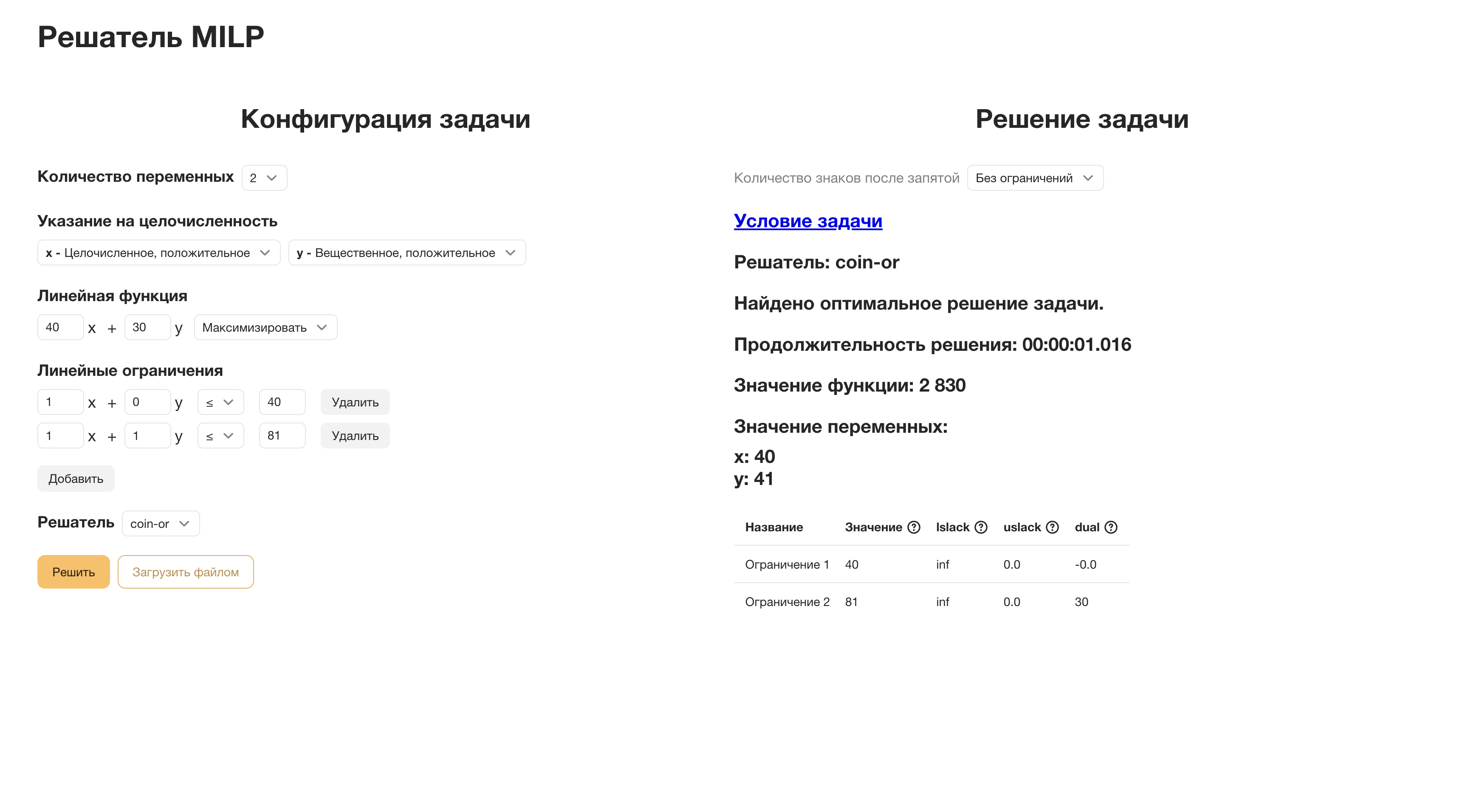


Рисунок 18 — Веб-приложение для решения задач смешано-целочисленного линейного программирования

В левой части экрана находится форма, с помощью которой пользователь может быстро ввести небольшую задачу. Ему предоставлена возможность выбрать количество переменных, их указание на целочисленность, ввести коэффициенты функции и направление оптимизации, настроить линейные ограничения, а также выбрать один из представленных решателей.

В случае, если задача объемная, пользователь может загрузить ее файлом, нажав на соответствующую кнопку. Перед ним откроется модальное окно, изображенное на рисунке 19.

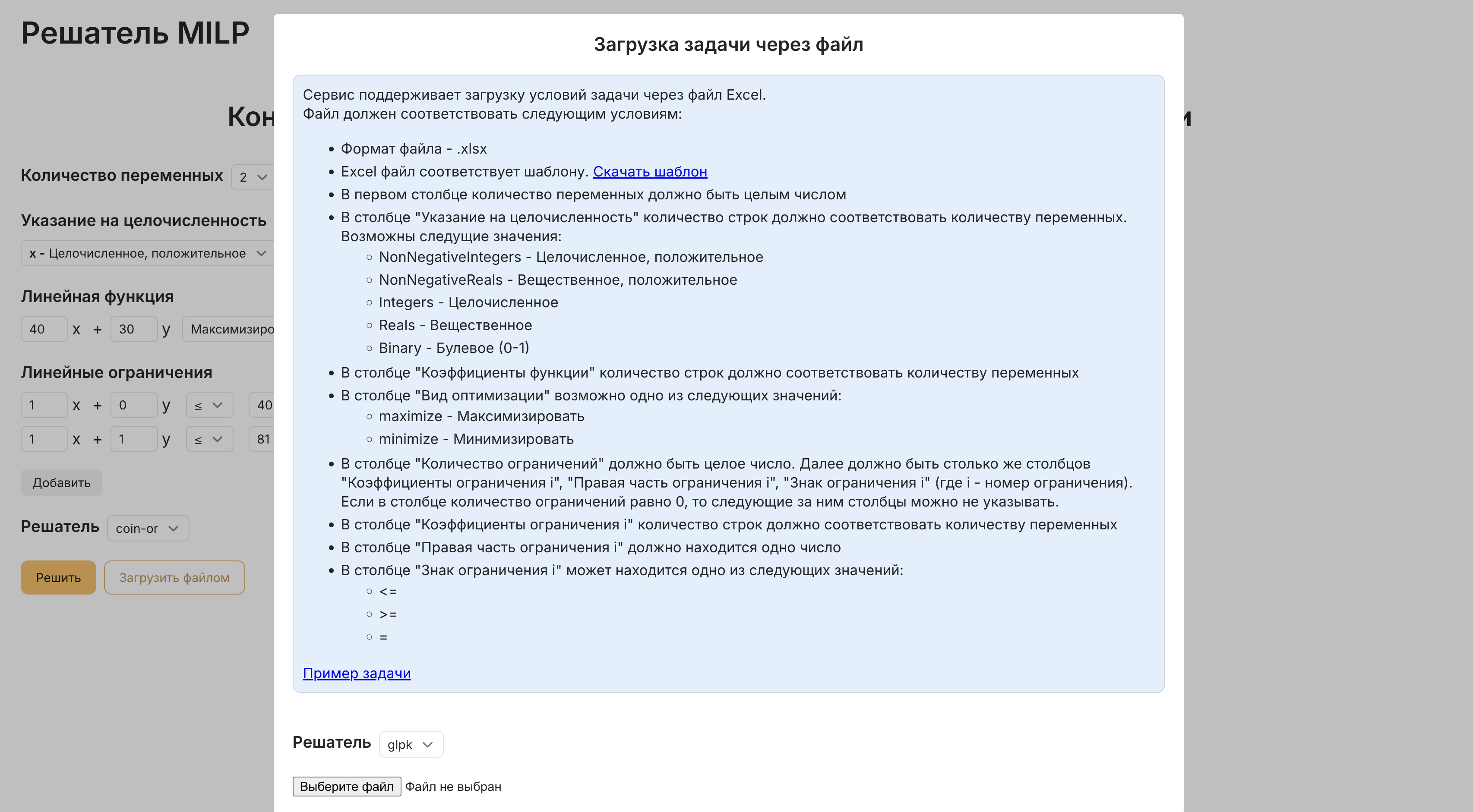


Рисунок 19 — Загрузка задачи через файл

В модальном окне указана подробная информация, о том, как должен быть заполнен файл Excel, чтобы сервис его корректно обработал. Помимо текстовых указания, есть шаблон и пример заполненной задачи, которые можно скачать.

После успешной загрузки задачи на сервер, бэкенд начинает решать задачу, в это время пользователю показывается, что задача находится в процессе решения и что этот процесс можно отменить (рисунок 20).

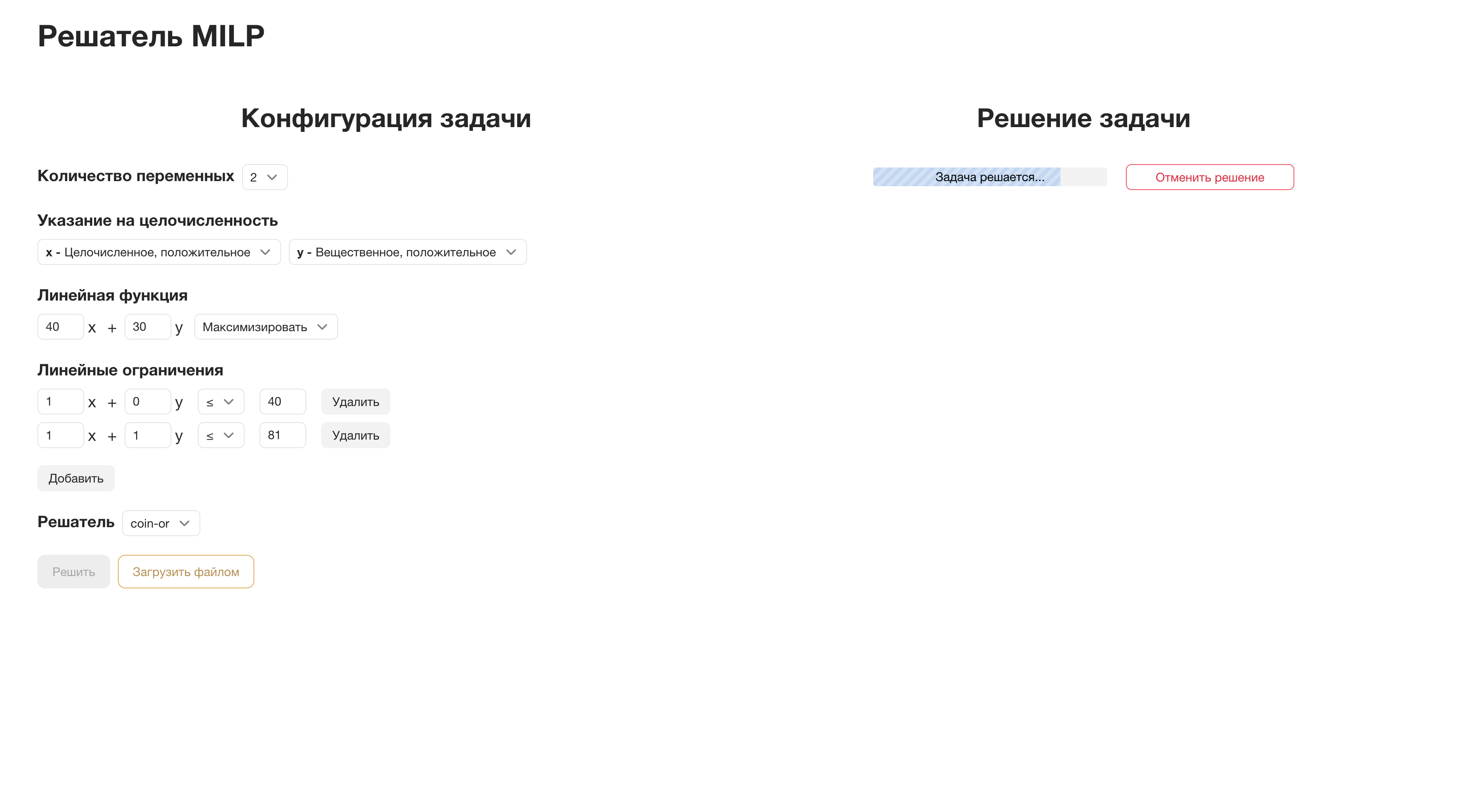


Рисунок 20 — Задача в процессе решения

В случае, если на сервере случилась ошибка, например во время валидации данных из файла, то на клиент возвращается ошибка, которая отображается всплывающим сообщением, показанным на рисунке 21.

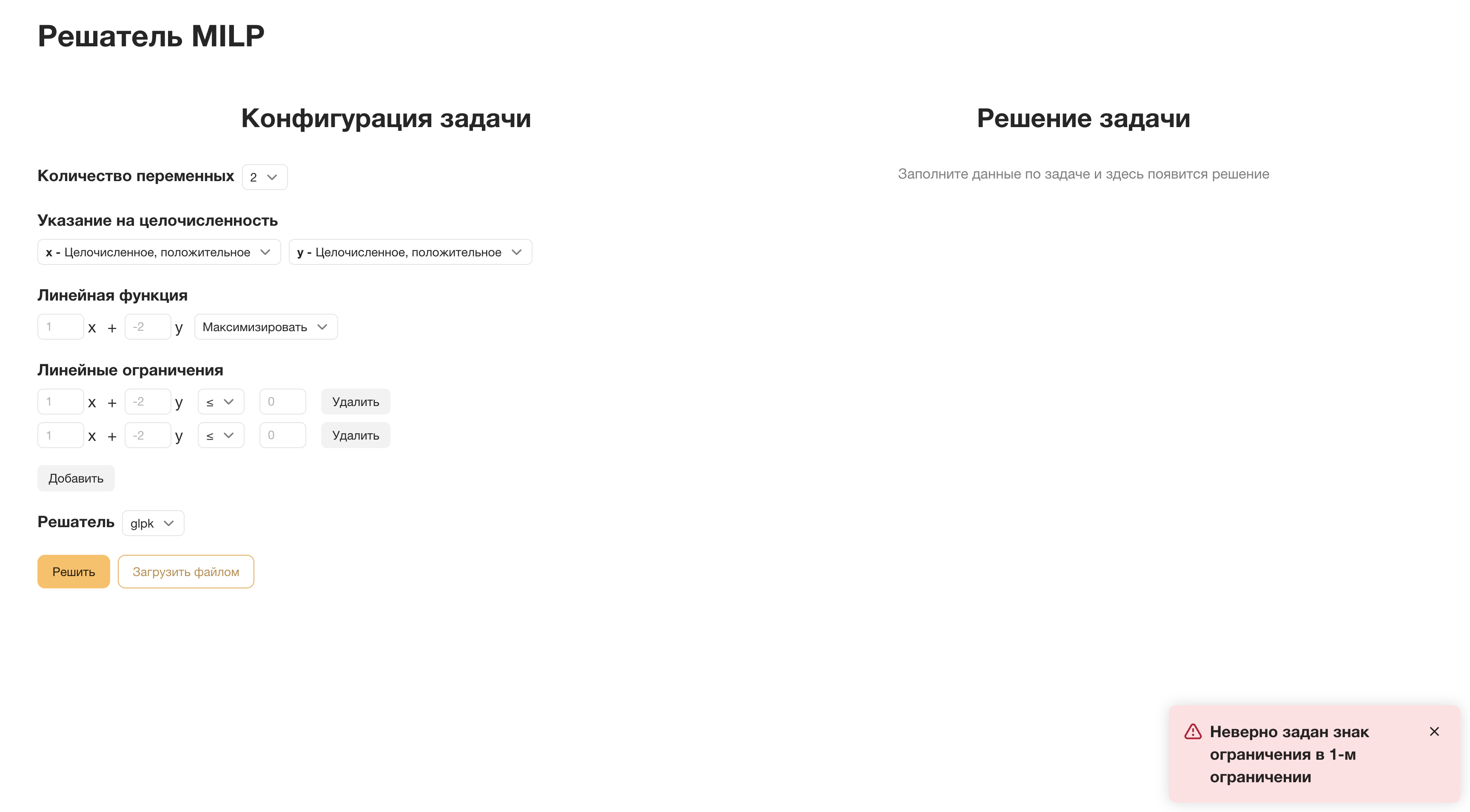


Рисунок 21 — Сообщение об ошибке

По окончании решения задачи, в правой части экрана отображается следующая информация: условие задачи, которое можно скачать в Excel формате, информация о том, какой решатель использовался в процессе решения, сколько занял весь процесс решения задачи, само решение с возможностью настройки отображения количества знаков после запятой, а также анализ чувствительности решения к изменению параметров.

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения выпускной квалификационной работы бакалавра на тему «Разработка веб-приложения для решения задач смешано-целочисленного линейного программирования» было создано комплексное программное решение, объединяющее современные методы математического моделирования и передовые веб-технологии. Разработанное веб-приложение позволяет пользователям с удобным и интуитивно понятным интерфейсом загружать условия задачи в виде Excel‑файлов или вводить их вручную, выбирать нужный решатель и получать оптимальное решение задачи в режиме реального времени с обратной связью посредством SSE.

Архитектурное решение проекта построено на монолитном подходе, что обеспечило быструю разработку и централизованное управление бизнес-логикой, а также позволило интегрировать все необходимые функции – от валидации и преобразования входных данных до формирования математической модели с использованием библиотеки Pyomo и последующего вызова решателей оптимизации (GLPK, CBC, SCIP). Использование ORM‑фреймворка SQLAlchemy в связке с PostgreSQL обеспечило надежное хранение информации о загруженных задачах, результатах вычислений, временных метках загрузки и завершения решения, а также возможность последующего анализа статистики работы системы и планирования доработок.

Фронтенд приложения реализован с применением современных инструментов – Vite, TypeScript, React, Gravity UI и RTK Query – что позволило создать гибкий и отзывчивый пользовательский интерфейс, предоставляющий пользователю не только возможность ввода и загрузки условий оптимизации, но и оперативное информирование о статусе решения задачи благодаря SSE-соединению. Модульная архитектура клиентской части обеспечивает высокую степень переиспользования компонентов и простоту поддержки.

Таким образом, реализованное веб-приложение удовлетворяет всем требованиям, установленным в техническом задании, и демонстрирует полную работоспособность по обработке и решению задач смешано-целочисленного линейного программирования. Полученные результаты подтверждают практическую полезность разработанного решения: система способна ускорить принятие управленческих решений в условиях реальных бизнес-задач, обеспечить гибкость и масштабируемость оптимизационных расчетов, а также предоставить удобные средства анализа и мониторинга процессов оптимизации.

Достижение поставленных целей и реализация полного функционала свидетельствуют о высоком уровне использования современных информационных и математических технологий, а также об успешной интеграции методов оптимизации в бизнес-процессы. Работа может служить отправной точкой для дальнейших исследований и доработок, направленных на расширение функционала и повышение эффективности систем принятия управленческих решений в условиях динамичного развития рынка.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Немхаузер Г. Л., Уолси М. Л. Integer and Combinatorial Optimization. // John Wiley & Sons, 1999. — 784 с.
2. Flask Documentation. — URL: https://flask.palletsprojects.com/ (дата обращения 12.01.2025).
3. Pyomo — Python Optimization Modeling Objects. — URL: https://www.pyomo.org/ (дата обращения 12.01.2025).
4. GLPK — GNU Linear Programming Kit. — URL: https://www.gnu.org/software/glpk/ (дата обращения 12.01.2025).
5. SQLAlchemy Documentation. — URL: https://docs.sqlalchemy.org/ (дата обращения 17.01.2025).
6. Vite Documentation. — URL: https://vitejs.dev/ (дата обращения 13.03.2025).
7. TypeScript Handbook. — URL: https://www.typescriptlang.org/docs/handbook/intro.html (дата обращения 05.03.2025).
8. React Documentation. — URL: https://reactjs.org/ (дата обращения 08.03.2025).
9. Gravity UI — Современные подходы к разработке интерфейсов. — URL: https://gravityui.com/ (дата обращения 10.03.2025).
10. RTK Query Documentation. — URL: https://redux-toolkit.js.org/rtk-query/overview (дата обращения 10.01.2025).

# **ПРИЛОЖЕНИЕ A QR-код репозитория с исходным кодом**

