МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 6-05-0611-01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

Тема: База данных фриланс-площадки

**Исполнитель**

студент 2 курса 1 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. С. Жук

подпись, дата

**Руководитель**

ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_ М. Г. Савельева

должность, учен. степень, ученое звание подпись, дата

Допущен(а) к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ М. Г. Савельева

подпись дата инициалы и фамилия

**Содержание**

[Введение 4](#_Toc197277932)

[1 Постановка задачи 5](#_Toc197277933)

[1.1 Обзор аналогичных решений 5](#_Toc197277934)

[1.1.1 Upwork 5](#_Toc197277935)

[1.1.2 Freelancer 6](#_Toc197277936)

[1.1.3 Fiverr 6](#_Toc197277937)

[1.2 Требования к проекту 7](#_Toc197277938)

[1.3 Вывод по разделу 8](#_Toc197277939)

[2 Проектирование и разработка базы данных 9](#_Toc197277940)

[2.1 Определение вариантов использования 9](#_Toc197277941)

[2.2 Диаграммы UML, взаимодействие всех компонентов 12](#_Toc197277942)

[2.3 Вывод по разделу 12](#_Toc197277943)

[3 Разработка объектов базы данных 13](#_Toc197277944)

[3.1 Разработка таблиц базы данных 13](#_Toc197277945)

[3.2 Разработка схемы базы данных 15](#_Toc197277946)

[3.3 Разработка процедур и функций базы данных 15](#_Toc197277947)

[3.4 Разработка представлений базы данных 15](#_Toc197277948)

[3.5 Разработка ролей базы данных 16](#_Toc197277949)

[3.6 Разработка индексов базы данных 17](#_Toc197277950)

[3.7 Разработка триггеров базы данных 17](#_Toc197277951)

[3.8 Вывод по разделу 17](#_Toc197277952)

[4 Описание процедур экспорта и импорта данных 18](#_Toc197277953)

[4.1 Процедура импорта данных из JSON-файла 18](#_Toc197277954)

[4.2 Процедура экспорта данных в JSON-файл 19](#_Toc197277955)

[4.3 Вывод по разделу 19](#_Toc197277956)

[5 Тестирование производительности 20](#_Toc197277957)

[5.1 Заполнение таблицы 20](#_Toc197277958)

[5.2 Тестирование производительности базы данных 20](#_Toc197277959)

[5.3 Вывод по разделу 21](#_Toc197277960)

[6 Описание технологии и её применение в базе данных 22](#_Toc197277961)

[6.1 Технология системы аудита базы данных 22](#_Toc197277962)

[6.2 Вывод по разделу 23](#_Toc197277963)

[7 Руководство пользователя 24](#_Toc197277964)

[7.1 Вывод по разделу 26](#_Toc197277965)

[Заключение 27](#_Toc197277966)

[Список использованных источников 28](#_Toc197277967)

[Приложение А Структура базы данных 29](#_Toc197277968)

[Приложение Б Процедуры и функции 36](#_Toc197277969)

Введение

Развитие цифровых технологий и тенденция к удалённой работе стали мощным стимулом для создания онлайн-платформ, объединяющих фрилансеров и заказчиков в различных сферах. Фриланс-площадки играют важную роль в современной цифровой экономике, обеспечивая удобный доступ к заданиям и проектам, возможность выбора специалистов и формирование безопасного пространства для сотрудничества.

Ключевым элементом такой системы является база данных, которая отвечает за централизованное хранение и обработку информации о пользователях, проектах, заказах, отзывах, сообщениях и транзакциях. Именно правильно спроектированная база данных обеспечивает целостность, надёжность, безопасность и масштабируемость всей площадки. Дополнительно, особое внимание уделяется работе с мультимедийными данными, которые могут включать изображения, вложения в портфолио, документы и другие формы визуальной информации.

Целью данной курсовой работы является разработка реляционной базы данных для фриланс-площадки. База данных сможет обеспечивать поддержку пользовательских ролей, безопасный доступ к данным через процедуры, резервное копирование, аудит действий, а также функции импорта и экспорта данных в формате JSON.

Поставленная цель обусловила следующие задачи:

* провести анализ аналогичных фриланс-сервисов и определить успешные решения в рамках проектирования базы данных;
* сформулировать функциональные требования к базе данных;
* описать сценарии взаимодействия пользователей с системой и разработать диаграммы прецедентов для всех ролей;
* рассмотреть взаимодействие компонентов базы данных;
* разработать структуру базы данных, включающую таблицы, связи, ограничения, а также поддерживающую работу с ролями, мультимедийным контентом;
* обеспечить возможность импорта/экспорта данных в формате JSON;
* реализовать технологию системы аудита базы данных;
* провести тестирование производительности базы данных.

Для реализации базы данных будет использована СУБД «PostgreSQL». Данная СУБД предоставляет широкие возможности для создания сложных, высоконагруженных и надёжных баз данных. Кроме того, СУБД обеспечивает высокую степень масштабируемости, оптимизацию выполнения запросов и гибкость при проектировании структуры данных.

Таким образом, проектируемая база данных будет представлять собой надёжную основу фриланс-платформы, обеспечивая стабильную работу, гибкость взаимодействия пользователей и возможность дальнейшего масштабирования системы в соответствии с растущими требованиями.

Постановка задачи

Одним из основных этапов в создании базы данных является анализ функциональных требований и определение задач, которые она должна решать для платформы. Предполагается, что пользователи будут взаимодействовать в разных ролях, включая администратора, модератора, фрилансера и заказчика. Каждая роль обладает индивидуальными правами доступа и выполняет определённые действия, которые должны быть реализованы через архитектуру базы данных.

Вся информация должна храниться в базе данных с обеспечением логирования действий и контроля целостности. Особое внимание уделяется необходимости импорта и экспорта данных, поддержке мультимедиа и масштабируемости структуры.

Обзор аналогичных решений

Upwork

**Upwork** – одна из крупнейших международных фриланс-платформ, ориентированная на долгосрочное сотрудничество между заказчиками и исполнителями [1]. С точки зрения проектирования базы данных, платформа представляет интерес благодаря следующим функциям:

* **Система заявок и предложений:** фрилансеры подают индивидуальные заявки на проекты с условиями выполнения. В БД это будет реализовано через таблицу orders, где фиксируются отклики (id\_project, id\_freelancer, creation\_date, status).

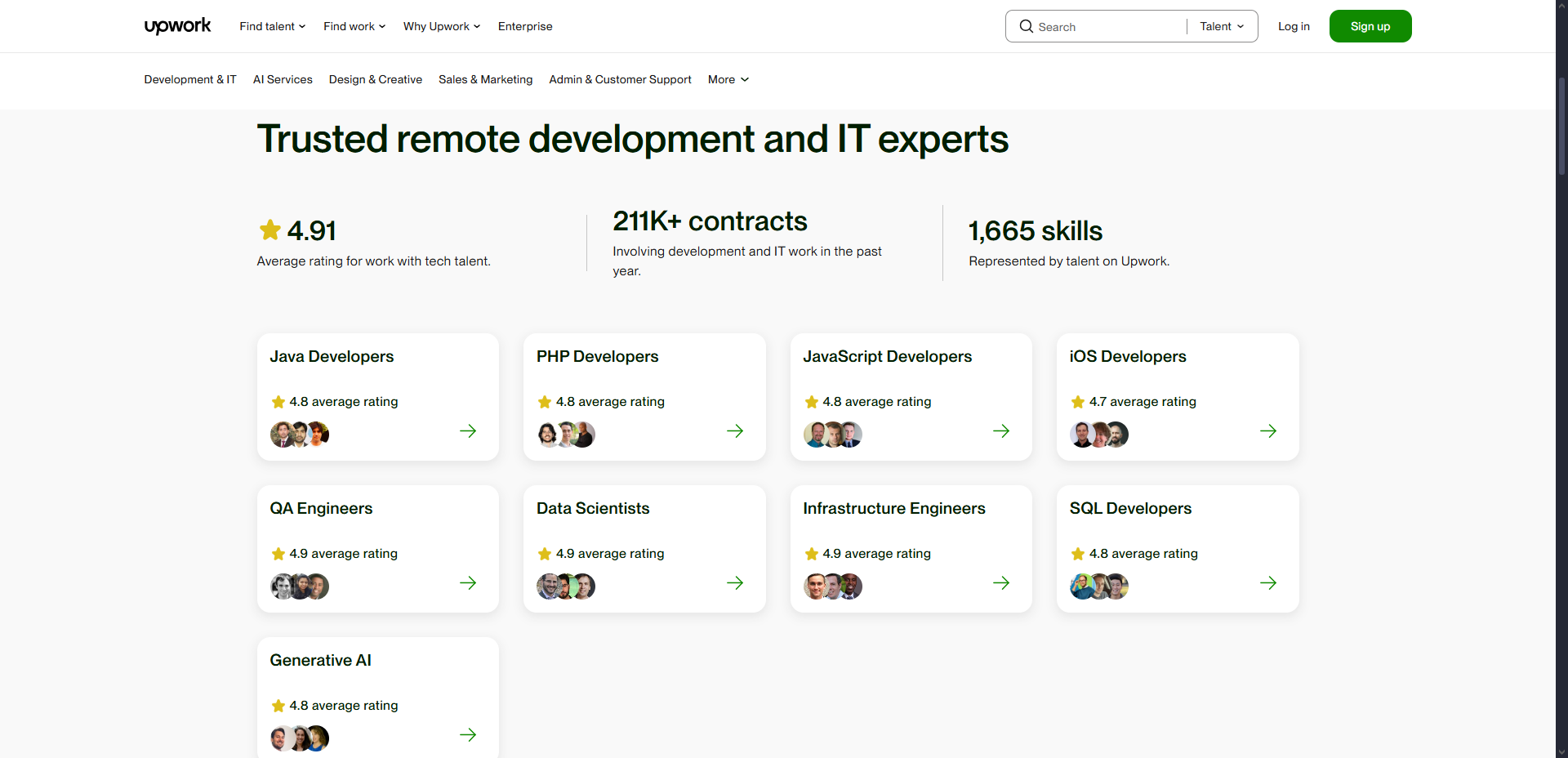


Рисунок 1.1 – Исполнители в категории «Разработчики»

* **Фильтрация исполнителей по параметрам:** на платформе применяются фильтры по навыкам, рейтингу, стоимости. Таблицы users и portfolio будут хранить информацию, необходимую для фильтрации, включая skills, experience, rating.
* **Система рейтингов и отзывов:** отзывы и оценки отображаются в профилях исполнителей. В БД будут использоваться таблицы reviews и users с полями rating и comment, поддерживающими обратную связь и рейтинг.

Freelancer

**Freelancer** – австралийская фриланс-платформа, охватывающая широкий спектр услуг от разработки программного обеспечения до дизайна и маркетинга [2]. С точки зрения базы данных, интерес представляет поддержка нескольких режимов взаимодействия между заказчиком и исполнителем:

* **Система конкурсов:** заказчик публикует задание, фрилансеры подают работы, проводится выбор победителя. Модель поддерживается через projects + orders с доработкой под конкурсную логику (этап подачи, голосование). В таблице orders можно будет фиксировать статус «конкурс», а в projects – поле competition\_mode.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 1.2 – Пример конкурсов от заказчиков на Freelancer

* **Командная работа и распределение задач:** возможность кооперации между фрилансерами. Таблицы projects и orders позволят реализовать базовую поддержку групповых задач, но в рамках проекта она может быть ограничена привязкой нескольких исполнителей через дополнительную сущность (например, team\_membership).

Fiverr

**Fiverr** – международная платформа, ориентированная на быстрые заказы по фиксированной стоимости [3]. Основная особенность – представление услуг в виде готовых предложений (гигов):

* **Готовые пакеты услуг:** фрилансеры предлагают подготовленные заранее услуги. Это будет реализовано через projects, где фрилансер сам создаёт описание услуги, прикрепляет media, а заказчик оформляет заказ через orders.

Изображение выглядит как текст, снимок экрана, веб-страница, Веб-сайт

Автоматически созданное описание

Рисунок 1.3 – Пример предоставляемых услуг от фрилансеров на Fiverr

* **Уровни фрилансеров:** присваиваются в зависимости от качества, скорости и отзывов. В таблице users могут быть добавлены поля level, performance\_score, основанные на анализе полей из reviews и orders.

Требования к проекту

В процессе разработки базы данных для фриланс-платформы были определены функциональные требования, обеспечивающие стабильную и безопасную работу системы, а также поддерживающие масштабируемость и надёжность при высоких нагрузках. Основные требования базируются на задачах, выявленных в ходе анализа предметной области и аналогичных решений. В перечень требований входит:

1. Многоуровневая ролевая модель.

* Реализация четырёх основных ролей: администратор, модератор, фрилансер, заказчик;
* Разграничение доступа к данным и действиям на уровне процедур и интерфейсов;
* Хранение информации о ролях и правах доступа в таблице users с атрибутом role.

1. Работа с проектами и заказами.

* Ведение записей о проектах, заказах, конкурсах.
* Хранение статуса, дедлайна, стоимости, условий работы – таблицы projects, orders.

1. Система отзывов и рейтингов.

* Возможность оценивать выполненные заказы и оставлять комментарии – таблица reviews;
* Расчёт агрегированного рейтинга пользователя с отображением в его профиле.

1. Обработка жалоб и модерация.

* Функционал подачи и обработки жалоб – таблица complaints;
* Привязка жалобы к заказу, проекту или пользователю с фиксацией статуса рассмотрения.

1. Коммуникация между пользователями.

* Реализация системы сообщений и диалогов через таблицы conversations и message.

1. Мультимедиаконтент.

* Возможность хранения и отображения изображений, документов, ссылок на видео в таблице media, связанной с проектами и портфолио фрилансеров.

1. Фильтрация и поиск.

* Поддержка фильтрации по категориям, цене, навыкам, рейтингу – через поля таблиц users, projects, orders, portfolio.

1. Импорт и экспорт данных.

* Поддержка загрузки и выгрузки данных в формате JSON;
* Реализация соответствующих хранимых процедур (предполагаемые import\_json\_data и export\_json\_data).

Вывод по разделу

В результате анализа предметной области и существующих решений сформулированы ключевые требования к функциональности и структуре базы данных фриланс-платформы. Были изучены три ведущие платформы-аналоги – Upwork, Freelancer и Fiverr, каждая из которых реализует собственную модель взаимодействия между пользователями. Их подходы к организации заявок, конкурсных заданий, рейтинговой системы и фильтрации были адаптированы и учтены при формировании архитектуры базы данных.

Выделены основные сущности: пользователи, проекты, заказы, сообщения, отзывы, жалобы и мультимедийные материалы. Для каждой из них определены таблицы, взаимосвязи и ключевые атрибуты, обеспечивающие реализацию типовых бизнес-процессов. Особое внимание уделено поддержке ролевой модели, безопасному доступу к данным через процедуры, возможности масштабирования, а также обеспечению аудита действий пользователей.

Для реализации обозначенных функциональных требований будет использоваться СУБД PostgreSQL, которая позволит спроектировать все таблицы, представления, процедуры и другие объекты базы данных, так как данная СУБД позволит обеспечить высокую степень масштабируемости, оптимизацию выполнения запросов и гибкость при проектировании структуры данных.

Проектирование и разработка базы данных

Определение вариантов использования

**Пользователь** является основной ролью для взаимодействия с сервисом. Он имеет доступ к личному кабинету, где может изменять свои данные, а также в личном кабинете находятся все его уведомления о присланных ему заявках на проекты. Также, у него есть возможность создавать различные портфолио, например разработка алгоритмов или вёрстка сайтов, с указанием своих навыков, по которым в будущем его могут найти другие такие же пользователи, чтобы получать от них заявки на работу с проектами.

Каждый пользователь фриланс-сервиса может выполнять функции как исполнителя, так и заказчика, поэтому дальше будет рассмотрение пользователя с двух сторон.

Ниже представлена диаграмма вариантов использования пользователя со стороны заказчика:



Рисунок 2.1 – Диаграмма вариантов использования пользователя со стороны заказчика

Любой заказчик может создавать проекты с указанием названия, описания, статуса проекта, а также с прикреплением возможных медиафайлов. Далее, сохраняется возможность редактировать созданные проекты, либо удалить вовсе. Все проекты можно фильтровать по статусу, выводя только нужные, например открытый к выполнению, в прогрессе, либо черновик проекта.

Заказчик может управлять своими заказами на проекты, изменяя статус заказа и его дедлайн. Если статус указывается как «Отменён», то заказ удаляется из списка и заносится в архив заказов, проект остаётся в общем списке для оформления нового заказа на выполнение. Если же статус заказа «Выполнен», то заказ также удаляется из списка и заносится в архив, но проект тоже удаляется из общего списка и считается как таковым выполненным полностью.

Для своих проектов заказчики могут через поиск по навыкам искать исполнителей для выполнения своих доступных проектов, то есть тех, на которых в данный момент нет активных заказов.

После выбора подходящего исполнителя заказчик отправляет уведомление исполнителю о выполнении проекта. Это уведомление исполнитель может как принять, тогда создаётся новый заказ, так и отклонить.

После завершение заказа заказчик может оставить отзыв о исполнителе со своей стороны, указав при этом рейтинг его работы. Также, появляется возможность подать жалобу только на тех пользователей, с которыми шла какая-либо работа, будь то выполнение либо отмена заказа.

Далее следует диаграмма вариантов использования пользователя со стороны исполнителя:

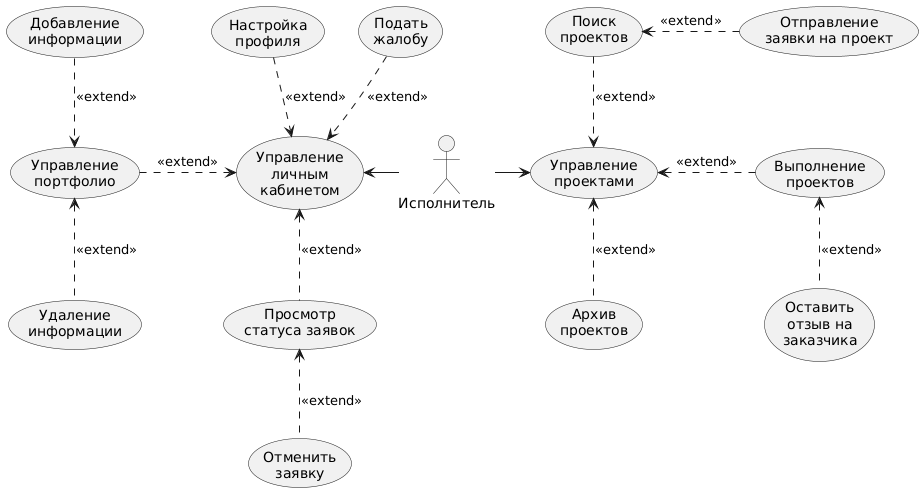


Рисунок 2.2 – Диаграмма вариантов использования пользователя со стороны исполнителя

В своём функционале исполнитель имеет много возможностей, как и заказчик, однако у него есть особые возможности.

Исполнитель может создавать различные портфолио для демонстрации своих навыков, указывая описание портфолио, возможные медиафайлы, навыки и опыт работы.

Для исполнителя основной деятельностью является отклик на проекты для их выполнения. Для удобства есть фильтры по статусу для просмотра только нужных проектов. После отклика на проект создаётся заказ, которым исполнитель, как и заказчик, может управлять: изменять статус, дедлайн заказа, а также заносить в архив.

После выполнения заказа исполнитель со своей стороны может оставить отзыв с рейтингом заказчику, если он хочет указать ключевые моменты работы с ним. Также есть возможность отправить жалобу на заказчика после взаимодействия с ним, просто так жалобы создавать нельзя.

Для решения каких-либо жалоб на сервисе была создана роль **модератор**. Его функциональные возможности представлены на диаграмме вариантов использования снизу:

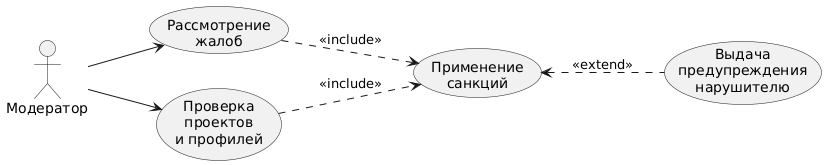


Рисунок 2.3 – Диаграмма вариантов использования в роли Модератор

Любые жалобы от пользователей поступают модератору на рассмотрение. Модератор может как решить жалобу, если такова считается весомой и действительной, так и удалить жалобу, если нет оснований для её существования. После решения жалобы могут быть применены санкции к нарушителю, например выдача предупреждения.

Для регулирования всего сервиса и его базы данных была создана роль **администратора**, функциональные возможности которого представлены на диаграмме вариантов использования ниже:

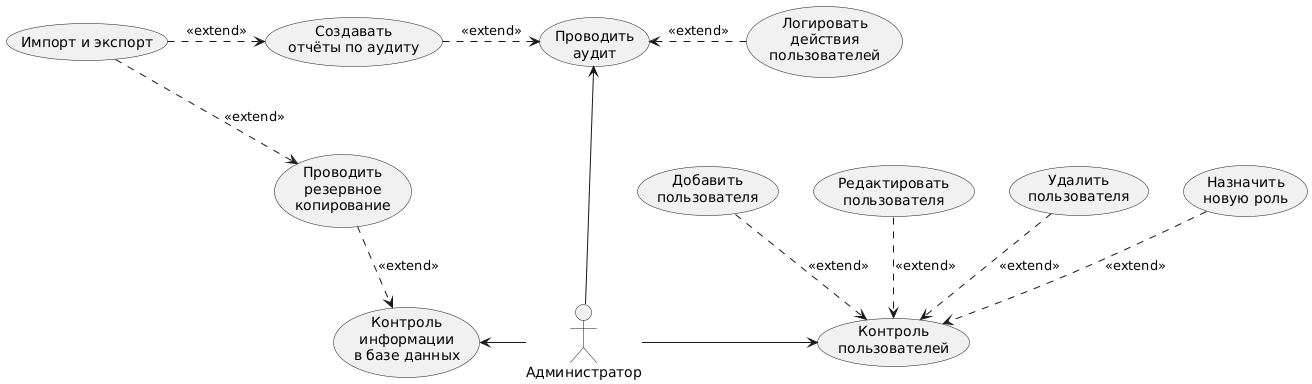


Рисунок 2.4 – Диаграмма вариантов использования в роли Администратор

Администратор имеет полный контроль над пользователями и их проектами. Он может создавать новых пользователей, редактировать их данные или удалять их, назначать им роли пользователя, модератора либо нового администратора. Администратор также отвечает за создание резервных копий всей базы данных, проводя своевременное экспортирование всех записей из базы на диск с последующей возможностью восстановление из них всех данных.

Диаграммы UML, взаимодействие всех компонентов

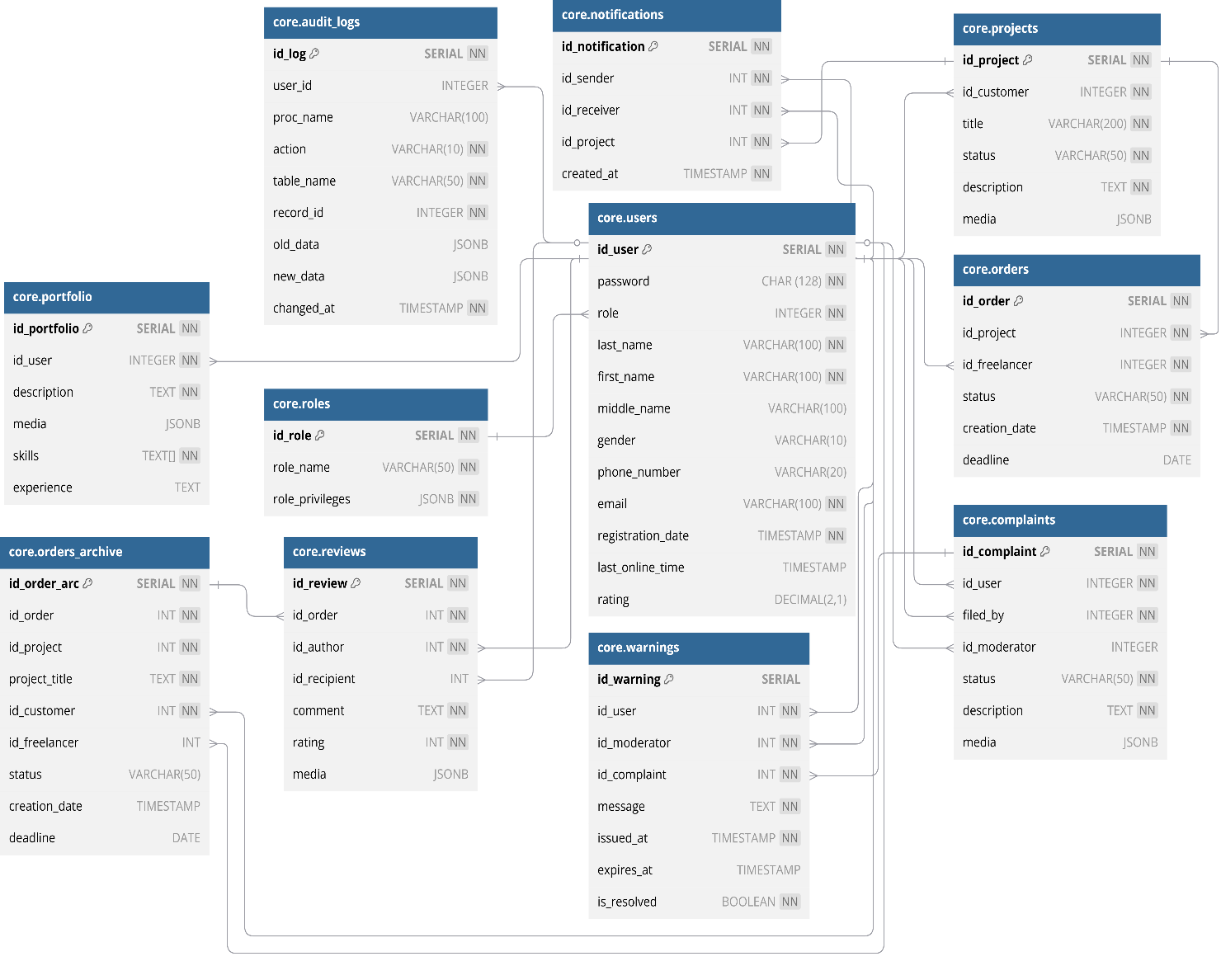


Рисунок 2.5 – Диаграмма базы данных

В соответствии с ранее описанными функциональными возможностями ролей база данных были разработаны следующие таблицы: users – хранение информации пользователей; roles – информация о ролях и привилегиях; projects – информация о проектах; reviews – отзывы пользователей; orders – заказы пользователей; orders\_archive – архив выполненных заказов; complaints – жалобы; warnings – предупреждения пользователей; portfolio – портфолио пользователей; audit\_logs – логи аудита; notifications – уведомления пользователей.

Вывод по разделу

По итогам проектирования и разработки базы данных были определены основные роли сервиса, функциональные возможности ролей. Также были спроектированы основные таблицы базы данных на основе анализа ролей сервиса. Таким образом база данных была хорошо и чётко спроектирована, что обеспечит надёжную работу сервиса, а также разграничит деятельность ролей.

Разработка объектов базы данных

Разработка таблиц базы данных

По итогам проектирования базы данных выделилось 11 таблиц, в которых будет храниться вся информация сервиса. Однако, необходимо определить структуры таблиц, по которым вся информация будет разбита.

Таблица 3.1 – users

|  |  |  |
| --- | --- | --- |
| **Имя столбца** | **Описание** | **Тип данных** |
| id\_user | Уникальный идентификатор пользователя | SERIAL |
| password | Захешированный пароль | CHAR(128) |
| role | ID роли пользователя | INTEGER |
| last\_name | Фамилия | VARCHAR(100) |
| first\_name | Имя | VARCHAR(100) |
| middle\_name | Отчество | VARCHAR(100) |
| gender | Пол | VARCHAR(10) |
| phone\_number | Номер телефона | VARCHAR(20) |
| email | Электронная почта | VARCHAR(100) |
| registration\_date | Дата регистрации | TIMESTAMP |
| last\_online\_time | Последняя активность | TIMESTAMP |
| rating | Рейтинг пользователя | DECIMAL(2,1) |

Таблица 3.2 – roles

|  |  |  |
| --- | --- | --- |
| **Имя столбца** | **Описание** | **Тип данных** |
| id\_role | Уникальный идентификатор роли | SERIAL |
| role\_name | Название роли | VARCHAR(50) |
| role\_privileges | JSON-объект с привелегиями для роли | JSONB |

Таблица 3.3 – projects

|  |  |  |
| --- | --- | --- |
| **Имя столбца** | **Описание** | **Тип данных** |
| id\_project | Уникальный идентификатор проекта | SERIAL |
| id\_customer | ID пользователя-заказчика | INTEGER |
| title | Название проекта | VARCHAR(200) |
| status | Статус проекта | VARCHAR(50) |
| description | Описание проекта | TEXT |
| media | Ссылки/данные медиа по проекту | JSONB |

Таблица 3.4 – orders\_archive

|  |  |  |
| --- | --- | --- |
| **Имя столбца** | **Описание** | **Тип данных** |
| id\_order\_arc | Уникальный идентификатор записи архива заказа | SERIAL |
| id\_order | Оригинальный ID заказа | INTEGER |
| id\_project | ID проекта, к которому относился заказ | INTEGER |
| project\_title | Заголовок проекта на момент архивации | TEXT |
| id\_customer | ID заказчика | INT |
| id\_freelancer | ID исполнителя | INT |
| status | Итоговый статус заказа | INT |
| creation\_date | Дата/время создания заказа | TIMESTAMP |
| deadline | Дедлайн заказа | DATE |

Таблица 3.5 – reviews

|  |  |  |
| --- | --- | --- |
| **Имя столбца** | **Описание** | **Тип данных** |
| id\_review | Уникальный идентификатор отзыва | SERIAL |
| id\_author | ID пользователя, оставившего отзыв | INTEGER |
| id\_recipient | ID пользователя, получившего отзыв | INTEGER |
| comment | Текст отзыва | TEXT |
| rating | Оценка | INTEGER |
| media | Ссылки или данные медиа, приложенные к отзыву | JSONB |

Таблица 3.6 – orders

|  |  |  |
| --- | --- | --- |
| **Имя столбца** | **Описание** | **Тип данных** |
| id\_order | Уникальный идентификатор заказа | SERIAL |
| id\_project | ID проекта, по которому создаётся заказ | INTEGER |
| id\_freelancer | ID исполнителя, взявшего заказ | INTEGER |
| status | Текущий статус заказа | VARCHAR(50) |
| creation\_date | Время создания заказа | TIMESTAMP |
| deadline | Дата сдачи заказа | DATE |

Таблица 3.7 – complaints

|  |  |  |
| --- | --- | --- |
| **Имя столбца** | **Описание** | **Тип данных** |
| id\_complaint | Уникальный идентификатор жалобы | SERIAL |
| id\_user | ID пользователя, в отношении которого жалуются | INTEGER |
| filed\_by | ID пользователя, подавшего жалобу | INTEGER |
| id\_moderator | ID модератора, назначенного для рассмотрения | INTEGER |
| status | Статус жалобы | VARCHAR(50) |
| description | Текст жалобы | TEXT |
| media | Ссылки или данные медиа, приложенные к жалобе | JSONB |

Таблица 3.8 – warnings

|  |  |  |
| --- | --- | --- |
| **Имя столбца** | **Описание** | **Тип данных** |
| id\_warning | Уникальный идентификатор предупреждения | SERIAL |
| id\_user | ID пользователя-нарушителя | INTEGER |
| id\_moderator | ID модератора, выдавшего предупреждение | INTEGER |
| id\_complaint | ID жалобы, по которой выдано предупреждение | INTEGER |
| message | Текст предупреждения | TEXT |
| issued\_at | Время выдачи предупреждения | TIMESTAMP |
| expires\_at | Дата/время истечения срока действия предупреждения | TIMESTAMP |
| is\_resolved | Флаг, закрыто ли предупреждение | BOOLEAN |

Таблица 3.9 – portfolio

|  |  |  |
| --- | --- | --- |
| **Имя столбца** | **Описание** | **Тип данных** |
| id\_portfolio | Уникальный идентификатор записи портфолио | SERIAL |
| id\_user | ID пользователя, владеющего портфолио | INTEGER |
| description | Описание работ/проекта в портфолио | TEXT |
| media | Ссылки или данные медиа (картинки, файлы) | JSONB |
| skills | Список навыков пользователя | TEXT |
| experience | Описание опыта работы | TEXT |

Таблица 3.10 – audit\_logs

|  |  |  |
| --- | --- | --- |
| **Имя столбца** | **Описание** | **Тип данных** |
| id\_log | Уникальный идентификатор записи аудита | SERIAL |
| user\_id | ID пользователя, совершившего действие | INTEGER |
| proc\_name | Имя хранимой процедуры или функции | VARCHAR(100) |
| action | Тип операции | VARCHAR(10) |
| table\_name | Имя таблицы, в которой произошло изменение | VARCHAR(50) |
| record\_id | PK затронутой записи в table\_name | INTEGER |
| old\_data | JSON-снимок данных до изменения | JSONB |
| new\_data | JSON-снимок данных после изменения | JSONB |
| changed\_at | Время фиксации изменения | TIMESTAMP |

Таблица 3.11 – notifications

|  |  |  |
| --- | --- | --- |
| **Имя столбца** | **Описание** | **Тип данных** |
| id\_notification | Уникальный идентификатор уведомления | SERIAL |
| id\_sender | ID пользователя-отправителя уведомления | INTEGER |
| id\_receiver | ID пользователя-получателя уведомления | INTEGER |
| id\_project | ID проекта, к которому относится уведомление | INTEGER |
| created\_at | Время создания уведомления | TIMESTAMP |

Разработка схемы базы данных

Для базы данных сервиса было решено удалить стандартную схему pubic и создать новую core, чтобы с нуля разграничить все привилегии между ролями и использование представлений.

Разработка процедур и функций базы данных

Процедуры в PostgreSQL позволяют выполнять различные последовательности команд, ограничивая возможности пользователя с целью соблюдения целостности и безопасности базы данных.

Функции являются хорошим инструментом для получения каких-либо данных или значений по параметрам и в разных выходных форматах, будь то строка, таблица или скалярная величина.

В ходе разработки процедур и функций базы данных было разработано 34 хранимых процедур для выполнения CRUD-операций и определённых последовательностей, а также 20 функций для получения разными ролями различных данных через вводимые параметры. Ниже представлен пример процедур и функций:

* admin\_create\_user – создаёт пользователя с его данными;
* user\_update\_project – обновляет данные о проекте пользователя;
* user\_delete\_review – удаляет пользовательский отзыв;
* free\_projects – поиск исполнителей для найма на доступные проекты пользователя.

Разработка представлений базы данных

Представления позволяют выполнять готовые запросы на получения данных без прямого доступа к таблицам с целью сохранения безопасности работы. В рамках базы данных было разработано 16 представлений, из них 14 являются пользовательскими, 2 – модераторские.

В пользовательских представлениях содержится информация, которую ему можно видеть, то есть исключаются пароли и данные, не принадлежащие ему. Существуют «расширенные» представления, которые комбинируют информацию из разных таблиц в одном результирующем наборе, что является очень удобным источником данных.

Модераторские представления имеют более обширный доступ к данным о пользователях и о жалобах, чем пользовательские. Пример представлений базы данных:

* v\_order\_extended – представление для получение заказов с названием проектов, имён заказчиков и исполнителей;
* v\_users – пользовательское представление с данными обо всех пользователей без пароля, почты и номера телефона;
* v\_user\_notifications – представление со всеми уведомлениями и именами отправителя и получателя;
* v\_orders\_reviews – представление для получение архивного заказа с отзывами исполнителя и заказчика.

Разработка ролей базы данных

Для базы данных сервиса было разработаны следующие роли:

* svc\_admin: login-роль без привилегий суперпользователя, является ролью администратора сервиса, для которой есть все привилегии на схему базы данных, все процедуры, функции, представления и триггеры для аудита;
* svc\_user: nologin-роль (групповая) с ограниченным набором пользовательских привилегий на ряд процедур, функций и представлений, является ролью пользователя сервиса;
* svc\_mod: nologin-роль (групповая) с ограниченным набором модераторских привилегий на ряд процедур, функций и представлений, является ролью модератора сервиса;
* app\_end\_usr: login-роль с наследованием от svc\_user, является ролью-подключением пользователя для приложения, имеет все права и ограничения от svc\_user;
* app\_mod\_usr: login-роль с наследованием от svc\_moder, является ролью-подключением модератора для приложения, имеет все права и ограничения от svc\_moder;
* svc\_app: login-роль с минимальными правами на просмотр информации из представлений core.v\_users и core.v\_roles и вставкой в core.users, является стандартной ролью для приложения, которое подключается к базе данных при запуске, нужна для аутентификации и авторизации пользователя, либо регистрации нового с последующим переключением на соответствующую роль.

Разработка индексов базы данных

Для гарантированного ускорения критичных SQL-запросов в системе были созданы специализированные индексы, «повешенные» на ключевые колонки, внешние ключи и часто фильтруемые поля: на идентификаторы (PRIMARY KEY, UNIQUE), на поля связей (FK), на поля со статусами и датами, а также на колонки с вложенными данными (JSONB, тексты).

Разработка триггеров базы данных

Для базы данных были использованы триггеры, предназначенные для ведения аудита любых действий, выполненных через процедуры. Создано 10 триггеров на соответствующие таблицы, которые при любом действии (INSERT, UPDATE, DELETE) выполняют функцию аудита, записывающую данные в таблицу audit\_logs. Пример триггеров:

* trg\_audit\_roles: триггер на таблицу core.roles с выполнением функции core.fn\_audit\_trigger('id\_role');
* trg\_audit\_users: триггер на таблицу core.users с выполнением функции core.fn\_audit\_trigger('id\_user');
* trg\_audit\_projects триггер на таблицу core.projects с выполнением функции core.fn\_audit\_trigger('id\_project');
* trg\_audit\_reviews: триггер на таблицу core.reviews с выполнением функции core.fn\_audit\_trigger('id\_review');
* trg\_audit\_orders: триггер на таблицу core.orders с выполнением функции core.fn\_audit\_trigger('id\_order');
* trg\_audit\_orders\_archive: триггер на таблицу core.orders\_archive с выполнением функции core.fn\_audit\_trigger('id\_order\_arc');
* trg\_audit\_complaints: триггер на таблицу core.complaints с выполнением функции core.fn\_audit\_trigger('id\_complaint');
* trg\_audit\_warnings: триггер на таблицу core.warnings с выполнением функции core.fn\_audit\_trigger('id\_warning');
* trg\_audit\_portfolio: триггер на таблицу core.portfolio с выполнением функции core.fn\_audit\_trigger('id\_portfolio');
* trg\_audit\_notifications: триггер на таблицу core. notifications с выполнением функции core.fn\_audit\_trigger('id\_notification');

Вывод по разделу

В результате проделанной работы была построена полноценная объектная модель базы данных, обеспечивающая высокий уровень безопасности и управляемости, а также разделимость между ролями. Была определена структура данных, слой представлений к данным, процедуры и функции для работы с данными, модель ролей и привилегий, которые будут использовать процедуры и функции, а также триггеры для ведения аудита действий над таблицами.

Описание процедур экспорта и импорта данных

Процедура импорта данных из JSON-файла

Принимая во внимание технологию аудита базы данных, в данном разделе будет рассмотрена процедура импорта данных всей базы данных без таблицы audit\_logs из JSON-файлa.

Ниже представлен листинг кода процедуры импорта данных всей базы данных без таблицы audit\_logs из JSON-файлa:

CREATE OR REPLACE PROCEDURE core.admin\_import\_db(p\_file TEXT)

LANGUAGE plpgsql SECURITY DEFINER AS $$

DECLARE

dump\_json JSONB; tbls TEXT[] := ARRAY[

'roles','users','projects','orders','orders\_archive',

'complaints','portfolio','notifications','reviews','warnings'];

pks TEXT[] := ARRAY[

'id\_role','id\_user','id\_project','id\_order','id\_order\_arc',

'id\_complaint','id\_portfolio','id\_notification','id\_review',

'id\_warning']; i INT; tbl\_count INT := array\_length(tbls,1);

BEGIN

SET LOCAL audit.skip = 'on';

dump\_json := pg\_read\_file(p\_file)::jsonb;

EXECUTE format(

'TRUNCATE %s RESTART IDENTITY CASCADE', array\_to\_string(ARRAY(SELECT format('core.%I', t) FROM unnest(tbls) AS t),', '));

FOR i IN 1 .. tbl\_count LOOP

EXECUTE format(

'INSERT INTO core.%I SELECT \* FROM jsonb\_populate\_recordset(NULL::core.%I, $1 -> %L)',

tbls[i], tbls[i], tbls[i]

) USING dump\_json;

END LOOP;

FOR i IN 1 .. tbl\_count LOOP

EXECUTE format(

'SELECT setval(

pg\_get\_serial\_sequence(''core.%I'', ''%I''),

GREATEST(COALESCE((SELECT MAX(%I) FROM core.%I), 0) + 1, 1)

)',

tbls[i], pks[i], pks[i], tbls[i]

);

END LOOP;

EXCEPTION WHEN OTHERS THEN

RAISE WARNING 'Не удалось импортировать БД: %', SQLERRM;

END;$$;

Листинг 4.1 – Процедура импорта из JSON-файла

Данная процедура принимает путь к JSON-файлу из параметра, отключает аудит действий, установив локальную переменную со значением on, считывает данные с файла как jsonb в переменную dump\_json. После каскадно очищаются все таблицы, названия которых берётся из массива tbls. В цикле по списку таблиц берётся соответствующий ключ из JSON (в dump\_json это «tablename») и через jsonb\_populate\_recordset загоняется все объекты обратно в каждую таблицу, а после, вторым циклом для каждой таблицы вычисляется MAX(id\_pk) и вызывается setval на последовательность, чтобы следующие SERIAL-значения шли без пропусков и не пересекались с импортированными.

Процедура экспорта данных в JSON-файл

Так же как и с импортом, в данном разделе будет рассмотрена процедура экспорта данных всей базы данных без таблицы audit\_logs в JSON-файл, ниже приведён листинг соответствующей процедуры:

CREATE OR REPLACE PROCEDURE core.admin\_export\_db(p\_file text)

LANGUAGE plpgsql SECURITY DEFINER AS $$

DECLARE

tbls TEXT[] := ARRAY[

'roles','users','projects','orders','orders\_archive',

'complaints','portfolio','notifications','reviews','warnings'];

parts TEXT[]; sql TEXT;

BEGIN

SET LOCAL audit.skip = 'on';

parts := ARRAY( SELECT format(

'%L, (SELECT COALESCE(jsonb\_agg(to\_jsonb(t)), ''[]''::jsonb)'

|| ' FROM core.' || tbl || ' t)', tbl)

FROM unnest(tbls) AS tbl);

sql := format(

'COPY (SELECT jsonb\_build\_object(%s)::text) TO %L',

array\_to\_string(parts, ', '),

p\_file);

EXECUTE sql;

EXCEPTION WHEN OTHERS THEN

RAISE WARNING 'Не удалось экспортировать БД: %', SQLERRM;

END; $$;

Листинг 4.2 – Процедура экспорта в JSON-файл

Эта процедура выполняет экспорт всех ключевых таблиц из схемы core в один JSON-файл. Сначала происходит установка флага audit.skip, чтобы не логировать действия этой процедуры. Дальше происходит поэтапное формирование JSON-полей из массива tbls в формате 'table\_name', (SELECT COALESCE(jsonb\_agg(to\_jsonb(t)), '[]') FROM core.table\_name t) – то есть массив JSON-объектов всех строк таблицы. После с помощью jsonb\_build\_object(...) все фрагменты объединяются в один большой JSON-объект и дальше выполняется динамическая команда sql, который сохраняет полученный JSON в файл.

Вывод по разделу

В данном разделе были разработаны процедуры импорта и экспорта всей информации из базы данных в формате JSON. Процедуры используют путь к файлу как входной параметр для импорта из или экспорта в него.

Тестирование производительности

Заполнение таблицы

С целью оценки производительности и оптимизации запросов к базе данных было произведено тестирование, первым этапом которого заключалось в заполнении одной таблицы 1000000 произвольных строк.

INSERT INTO core.projects (

id\_customer, title, status, description, media

)

SELECT

(ARRAY[40,46,47,48,50])[ ((gs - 1) % 5) + 1 ],

'Project title #' || gs,

(ARRAY['draft','open','in\_progress','completed','cancelled'])

[ ((gs - 1) % 5) + 1 ],

'This is a description for project #' || gs,

CASE WHEN ((gs - 1) % 2) = 0

THEN jsonb\_build\_object(

'media\_url', 'https://example.com/media/' || gs)

ELSE jsonb\_build\_object(

'non\_media\_url', 'https://bad.link/media/' || gs)

END

FROM generate\_series(1,1000000) AS gs;

Листинг 5.1 – SQL-команда для генерации 1000000 строк

Данный запрос с помощью generate\_series(1,1000000) создаёт миллион строк и вставляет их в таблицу core.projects, автоматически генерируя для каждой записи значение id\_customer циклическим выбором из массива [40,46,47,48,50], заголовок title в виде «Project title #N» и описание description «This is a description for project #N», где N – номер строки; поле status тоже выбирается циклически из списка ['draft','open','in\_progress','completed','cancelled']; а в JSONB-поле media при чётных номерах формируется один объект { "media\_url": "https://example.com/media/N" }, а при нечётных – { "non\_media\_url": "https://bad.link/media/N" }. Таким образом за один запрос создаётся набор плавно чередующихся тестовых данных.

Тестирование производительности базы данных

Для проверки производительности запроса к базе данных на примере следующего запросы был проведён анализ планов запроса:

SELECT id\_project, id\_customer, title, description, media

FROM core.v\_projects p WHERE p.status = 'draft'

Листинг 5.2 – SQL-запрос для тестирования

После выполнения его план запрос показал следующие значения, представленные на рисунке ниже.

На нём видно, что без оптимизации запроса сервер проводит последовательное сканирование всех строк и по фильтру убирает ненужные, оставляя только те, которые соответствуют условию.

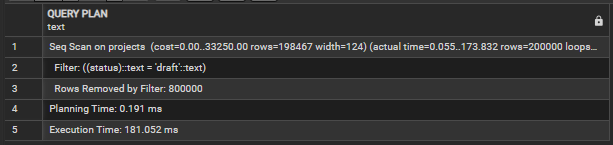


Рисунок 5.1 – План запроса до оптимизации

С целью оптимизации данного запроса и запросов, где статус может быть «draft», «open», «in\_progress», «completed», «cancelled», были созданы покрывающие индексы на столбец status со значениями id\_project, id\_customer, title, description, media:

CREATE INDEX CONCURRENTLY idx\_proj\_draft\_cover

ON core.projects(status)

INCLUDE (id\_project, id\_customer, title, description, media)

WHERE status = 'draft';

Листинг 5.3 – Пример одного из покрывающих индексов

После повторного выполнения запроса его план показал следующие значения:



Рисунок 5.2 – План запроса после оптимизации

Видно, что поиск проходит уже только по индексу, выбирая сразу нужные значения, а не выбирая из всего набора. Таким образом удалось уменьшить время выполнения (Execution Time) с 181,052 миллисекунд до 29,891.

Вывод по разделу

По итогам данного раздела было проведено тестирование таблицы базы данных. Таблица была заполнена 1 000 000 строк, к ней был выполнен запрос на поиск проектов по статусу. Была проведена оптимизация запроса путём добавления покрывающего индекса с определённым статусом, что позволило ускорить выполнение запроса в 6 раз.

Описание технологии и её применение в базе данных

Технология системы аудита базы данных

Технология аудита базы данных представляет собой совокупность приёмов и механизмов, с помощью которых система управления базами данных автоматически фиксирует всю или выборочную историю изменений данных и доступа к ним. В моей базе данных это реализовано через функцию с триггерами.

CREATE OR REPLACE FUNCTION core.fn\_audit\_trigger()

RETURNS TRIGGER LANGUAGE plpgsql AS $$

DECLARE

v\_old JSONB; v\_new JSONB; v\_id INT; v\_user INT; v\_proc TEXT;

pk\_column TEXT;

BEGIN

BEGIN

v\_user := current\_setting('audit.user\_id')::INT;

EXCEPTION WHEN OTHERS THEN RETURN NULL;

END;

IF current\_setting('audit.skip', TRUE) = 'on' THEN RETURN NULL;

END IF;

v\_proc := COALESCE(

current\_setting('audit.parent\_proc\_name', TRUE),

current\_setting('audit.proc\_name', TRUE)

);

v\_old := CASE WHEN TG\_OP = 'INSERT' THEN NULL ELSE to\_jsonb(OLD) END;

v\_new := CASE WHEN TG\_OP = 'DELETE' THEN NULL ELSE to\_jsonb(NEW) END;

IF TG\_NARGS < 1 THEN RAISE EXCEPTION 'fn\_audit\_trigger requires 1 argument: pk\_column'; END IF;

pk\_column := TG\_ARGV[0];

v\_id := COALESCE(

(v\_new ->> pk\_column)::INT,

(v\_old ->> pk\_column)::INT

);

INSERT INTO core.audit\_logs (user\_id, proc\_name, action, table\_name, record\_id, old\_data, new\_data, changed\_at)

VALUES (v\_user, v\_proc, TG\_OP, TG\_TABLE\_SCHEMA || '.' || TG\_TABLE\_NAME, v\_id, v\_old,v\_new,now());

RETURN NULL; END; $$;

Листинг 6.1 – Триггер-функция аудита базы данных

Эта триггер‐функция автоматически фиксирует все операции INSERT, UPDATE и DELETE в таблицах приложения, без дублирования кода в каждой процедуре: при срабатывании триггера она сначала пытается прочитать параметр audit.user\_id ID пользователя и сразу выходит, если это не удалось; затем проверяет флаг audit.skip, чтобы при массовых загрузках или экспортах не логифицировать изменения; после чего берёт имя вызывающей процедуры из переменных audit.proc\_name/audit.parent\_proc\_name; формирует JSON-снимки старого (OLD) и нового (NEW) состояния строки; из переданного аргумента триггера (TG\_ARGV[0]) узнаёт имя PK-столбца и достаёт его значение из JSON-снимков; и, наконец, записывает в таблицу core.audit\_logs запись с полями: кто (user\_id), какая процедура (proc\_name), тип операции (TG\_OP), где (TG\_TABLE\_SCHEMA||'.'||TG\_TABLE\_NAME, record\_id), что изменилось (old\_data, new\_data) и когда (now()).

CREATE OR REPLACE TRIGGER trg\_audit\_roles

AFTER INSERT OR UPDATE OR DELETE ON core.roles

FOR EACH ROW EXECUTE FUNCTION core.fn\_audit\_trigger('id\_role');

CREATE OR REPLACE TRIGGER trg\_audit\_users

AFTER INSERT OR UPDATE OR DELETE ON core.users

FOR EACH ROW EXECUTE FUNCTION core.fn\_audit\_trigger('id\_user');

CREATE OR REPLACE TRIGGER trg\_audit\_projects

AFTER INSERT OR UPDATE OR DELETE ON core.projects

FOR EACH ROW EXECUTE FUNCTION core.fn\_audit\_trigger('id\_project');

CREATE OR REPLACE TRIGGER trg\_audit\_reviews

AFTER INSERT OR UPDATE OR DELETE ON core.reviews

FOR EACH ROW EXECUTE FUNCTION core.fn\_audit\_trigger('id\_review');

CREATE OR REPLACE TRIGGER trg\_audit\_orders

AFTER INSERT OR UPDATE OR DELETE ON core.orders

FOR EACH ROW EXECUTE FUNCTION core.fn\_audit\_trigger('id\_order');

CREATE OR REPLACE TRIGGER trg\_audit\_orders\_archive

AFTER INSERT OR UPDATE OR DELETE ON core.orders\_archive

FOR EACH ROW EXECUTE FUNCTION core.fn\_audit\_trigger('id\_order\_arc');

CREATE OR REPLACE TRIGGER trg\_audit\_complaints

AFTER INSERT OR UPDATE OR DELETE ON core.complaints

FOR EACH ROW EXECUTE FUNCTION core.fn\_audit\_trigger('id\_complaint');

CREATE OR REPLACE TRIGGER trg\_audit\_warnings

AFTER INSERT OR UPDATE OR DELETE ON core.warnings

FOR EACH ROW EXECUTE FUNCTION core.fn\_audit\_trigger('id\_warning');

CREATE OR REPLACE TRIGGER trg\_audit\_portfolio

AFTER INSERT OR UPDATE OR DELETE ON core.portfolio

FOR EACH ROW EXECUTE FUNCTION core.fn\_audit\_trigger('id\_portfolio');

CREATE OR REPLACE TRIGGER trg\_audit\_notifications

AFTER INSERT OR UPDATE OR DELETE ON core.notifications

FOR EACH ROW EXECUTE FUNCTION core.fn\_audit\_trigger('id\_notification');

Листинг 6.2 – AFTER-триггеры аудита базы данных

Для каждой таблицы были созданы соответствующие AFTER-триггеры, где применяется эта функция с параметром, равным первичным ключом таблицы.

Вывод по разделу

В данном разделе была описана технология аудита базы данных, её реализация в виде триггер-функции и применение через AFTER-триггеры на таблицах. Реализованная технология аудита базы данных записывает информацию обо всех действиях, которые выполнялись пользователями по отношению к их процедурам.

Руководство пользователя

Для базы данных сервиса было создано приложение с использованием технологии WPF, которое взаимодействует с базой данных через ORM технологию EntityFrameworkCore.PostgreSQL. При запуске приложение сразу пробует подключиться к базе данных под стандартной ролью svc\_app, чтобы приложение могло делать аутентификацию или регистрацию пользователя.

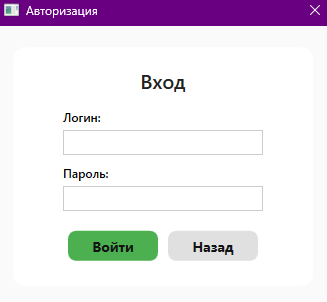


Рисунок 7.1 – Окно входа в аккаунт пользователя

После успешного входа приложение меняет подключение со стандартного на пользовательское в соответствии с его ролью, полученной с базы данных. В нашем случае это просто пользователь.

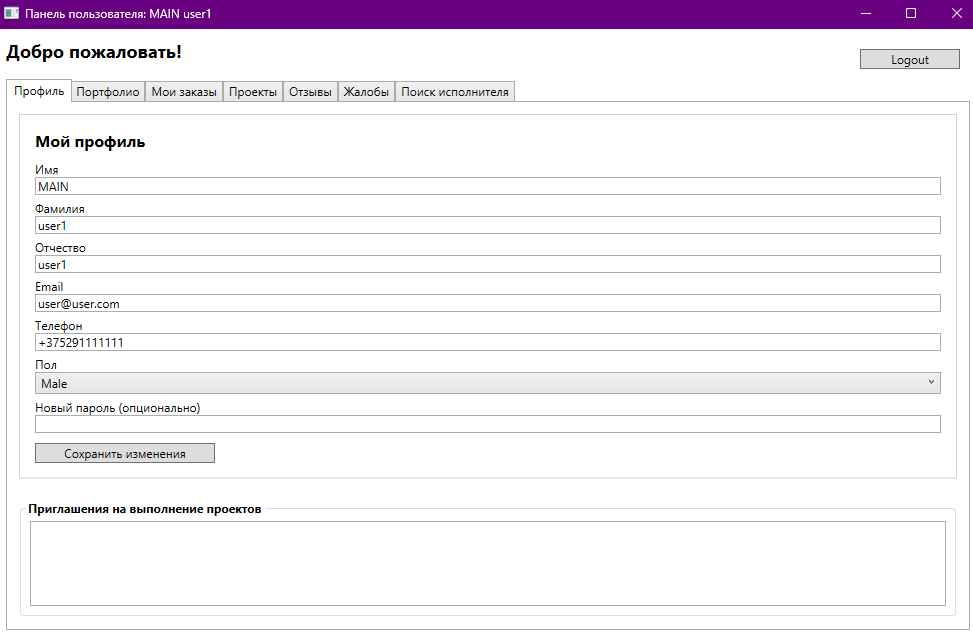


Рисунок 7.2 – Профиль пользователя

После авторизации пользователь попадает в свой личный кабинет, где может изменить свои личные данные или посмотреть уведомления о заявках.

На вкладке «Проекты» пользователь может просмотреть все доступные проекты, которые можно фильтровать по статусу проекта, а также откликаться на чужие для выполнения. Есть возможность просматривать только свои проекты, которые визуально имеют зелёный фон, на них нельзя откликаться, но можно таким образом легко и быстро редактировать или удалять их.

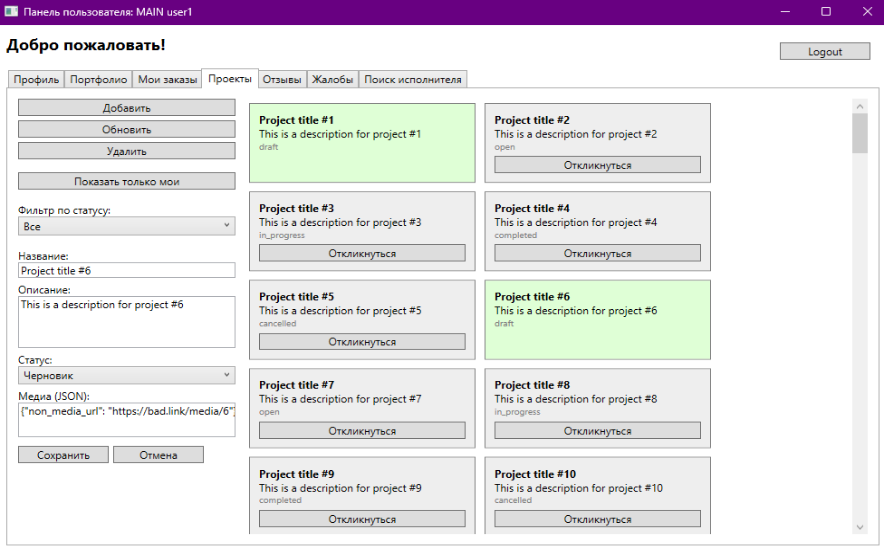


Рисунок 7.3 – Вкладка «Проекты»

На этой же вкладке можно добавлять, изменять, удалять свои проекты через соответствующие кнопки. Слева появляется форма, которая заполняется данными о проекте, если пользователь его выбрал.

У администратора визуально приложение отличается. Ему виден весь список пользователей и ролей с действиями по изменению данных.

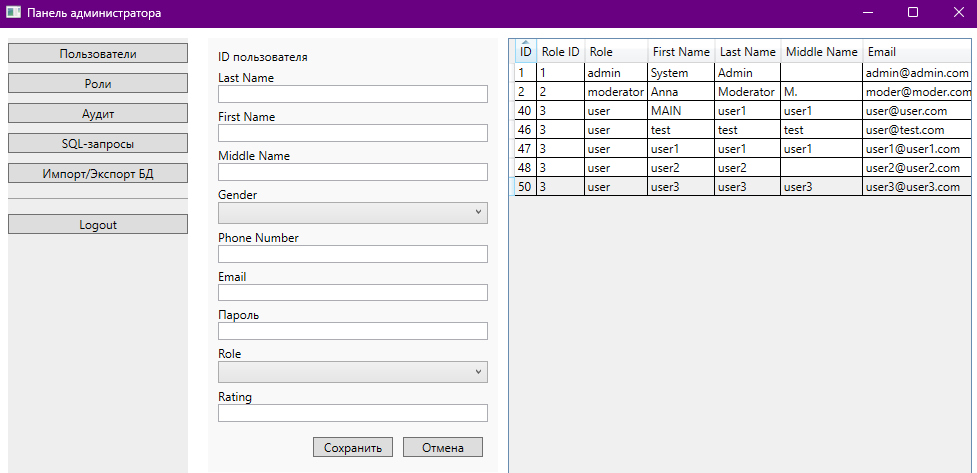


Рисунок 7.4 – Панель администратора

У модератора своя визуальная версия приложения, предназначенная ему для работы с жалобами.

Ему показывается список всех поданных жалоб, которые он «разрешает» через изменение статус и применение санкций в виде выдачи предупреждения нарушителю или удаления проекта.

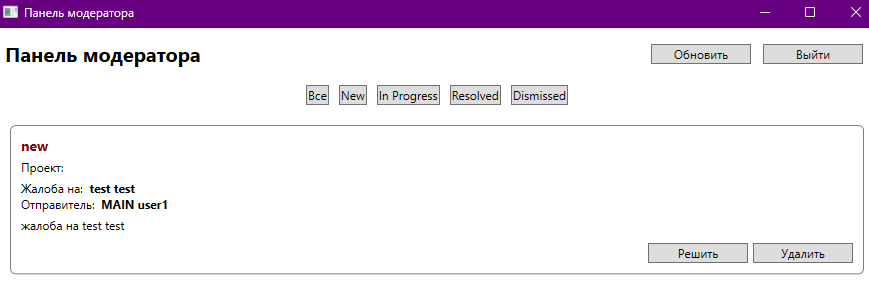


Рисунок 7.5 – Панель модератора

Модератор может переключаться между жалобами по их статусу, а также обновлять панель для получения актуального списка жалоб.

Вывод по разделу

Было разработано приложение, позволяющее удобно взаимодействовать с базой данных. Для каждой роли были созданы визуально отличимые компоненты в соответствии с правами на выполнение процедур.

Для всех пользователей был реализован общий компонент авторизации и регистрации, который работает с базой данных от стандартного подключения, которое имеет минимальный набор прав.

Все действия с данными, будь то добавление, изменение, удаление, а также поиск, фильтрация и выполнение определённых операций были созданы с использованием готовых процедур и функций в совокупности с представлениями схемы базы данных.

Таким образом в рамках курсовой работы была построена централизованная и безопасная архитектура, где вся бизнес-логика скрыта в хранимых процедурах, функциях и представлениях, а доступ к ним регулируется чёткой ролевой моделью; параллельно внедрён триггерный аудит всех изменений данных с сохранением «до»/«после» состояний и привязкой к пользователям и процедурам; помимо этого разработаны универсальные механизмы экспорта и импорта всей базы в JSON, позволяющие быстро сохранять и восстанавливать её полное состояние.

Заключение

В рамках курсовой работы была разработана база данных для фриланс-площадки на PostgreSQL с созданием приложения при помощи технологии WPF. Сервис позволяет пользователям работать с проектами других пользователей, создавая заказы и выполняя их, искать исполнителей для выполнения своих проектов, редактировать свои профили и портфолио. Администратор сервиса имеет полный контроль над пользователями и ролями, проводит аудит действий пользователей, а также может создавать резервные копии БД через экспорт и импорт JSON файлов. Модератор имеет все права для просмотра пользовательских жалоб, их решений и применений санкций по отношению к нарушителям.

Структура базы данных представима в виде таблиц, в которых хранится информация о пользователях, об их портфолио, о ролях, о проектах и заказах, об архивных заказах, о пользовательских жалобах, отзывах, предупреждениях и уведомлениях.

База данных имеет свою схему, в которой у пользователей и модераторов присутствуют жесткие ограничения на прямой доступ к объектам. С целью разграничения функциональных возможностей были созданы представления, процедуры и функции, для безопасной работы с данными БД.

Была реализована технология аудита базы данных, которая отслеживает любые операции вставки, изменения, удаления данных с последующей записью информации о том, кто это сделал, через какую процедуру, с какой таблицей и прочее в отдельную таблицу для хранения логов аудита.

Список использованных источников

1. Аналог «Upwork» [Электронный ресурс]. – Режим доступа: [https://www.upwork.com/](https://www.upwork.com/–). – Дата доступа: 29.03.2025.
2. Аналог «Freelancer» [Электронный ресурс]. – Режим доступа: <https://www.freelancer.com/>.– Дата доступа: 29.03.2025.
3. Аналог «Fiverr» [Электронный ресурс]. – Режим доступа: <https://www.fiverr.com/>. – Дата доступа: 29.03.2025.

Приложение А Структура базы данных

CREATE TABLE core.roles (

id\_role SERIAL PRIMARY KEY,

role\_name VARCHAR(50) NOT NULL UNIQUE,

role\_privileges JSONB NOT NULL

);

Листинг А.1 – Таблица roles

CREATE TABLE core.users (

id\_user SERIAL PRIMARY KEY,

password CHAR(128) NOT NULL CHECK (length(password) = 128),

role INTEGER NOT NULL REFERENCES core.roles(id\_role) ON UPDATE CASCADE ON DELETE RESTRICT,

last\_name VARCHAR(100) NOT NULL CHECK (char\_length(last\_name) > 0),

first\_name VARCHAR(100) NOT NULL CHECK (char\_length(first\_name) > 0),

middle\_name VARCHAR(100),

gender VARCHAR(10) CHECK (gender IN ('Male','Female','Other')),

phone\_number VARCHAR(20) CHECK (phone\_number ~ '^\+?[0-9]{7,20}$'),

email VARCHAR(100) NOT NULL UNIQUE CHECK (email ~\* '^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'),

registration\_date TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

last\_online\_time TIMESTAMP CHECK (last\_online\_time IS NULL OR last\_online\_time >= registration\_date),

rating DECIMAL(2,1) DEFAULT 0.0 CHECK (rating >= 0 AND rating <= 5)

);

Листинг А.2 – Таблица users

CREATE TABLE core.projects (

id\_project SERIAL PRIMARY KEY,

id\_customer INTEGER NOT NULL REFERENCES core.users(id\_user) ON UPDATE CASCADE ON DELETE CASCADE,

title VARCHAR(200) NOT NULL CHECK (char\_length(title) > 0),

status VARCHAR(50) NOT NULL CHECK (status IN ('draft','open','in\_progress','completed','cancelled')),

description TEXT NOT NULL CHECK (char\_length(description) > 0),

media JSONB

);

Листинг А.3 – Таблица projects

CREATE TABLE core.reviews (

id\_review SERIAL PRIMARY KEY,

id\_order INT NOT NULL REFERENCES core.orders\_archive(id\_order\_arc) ON UPDATE CASCADE ON DELETE CASCADE,

id\_author INT NOT NULL REFERENCES core.users(id\_user) ON UPDATE CASCADE ON DELETE CASCADE,

id\_recipient INT REFERENCES core.users(id\_user) ON UPDATE CASCADE ON DELETE SET NULL,

comment TEXT NOT NULL CHECK (char\_length(comment) > 0),

rating INT NOT NULL CHECK (rating BETWEEN 1 AND 5),

media JSONB,

CONSTRAINT uq\_review\_per\_side UNIQUE (id\_order, id\_author),

CONSTRAINT ck\_no\_self\_review CHECK (id\_recipient IS NULL OR id\_author <> id\_recipient)

);

Листинг А.4 – Таблица reviews

CREATE TABLE core.orders (

id\_order SERIAL PRIMARY KEY,

id\_project INTEGER NOT NULL UNIQUE REFERENCES core.projects(id\_project) ON UPDATE CASCADE ON DELETE CASCADE,

id\_freelancer INTEGER NOT NULL REFERENCES core.users(id\_user) ON UPDATE CASCADE ON DELETE CASCADE,

status VARCHAR(50) NOT NULL CHECK (status IN ('pending','active','completed','cancelled','disputed')),

creation\_date TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

deadline DATE CHECK (deadline IS NULL OR deadline >= creation\_date)

);

Листинг А.5 – Таблица orders

CREATE TABLE core.orders\_archive (

id\_order\_arc SERIAL PRIMARY KEY,

id\_order INT NOT NULL,

id\_project INT NOT NULL,

project\_title TEXT NOT NULL,

id\_customer INT NOT NULL REFERENCES core.users(id\_user) ON UPDATE CASCADE ON DELETE CASCADE,

id\_freelancer INT REFERENCES core.users(id\_user) ON UPDATE CASCADE ON DELETE CASCADE,

status VARCHAR(50) CHECK (status IN ('completed','cancelled')),

creation\_date TIMESTAMP,

deadline DATE

);

Листинг А.6 – Таблица orders\_archive

CREATE TABLE core.complaints (

id\_complaint SERIAL PRIMARY KEY,

id\_user INTEGER NOT NULL REFERENCES core.users(id\_user) ON UPDATE CASCADE ON DELETE CASCADE,

filed\_by INTEGER NOT NULL REFERENCES core.users(id\_user) ON UPDATE CASCADE ON DELETE CASCADE,

id\_project INT REFERENCES core.projects(id\_project) ON UPDATE CASCADE ON DELETE SET NULL,

id\_moderator INTEGER REFERENCES core.users(id\_user) ON UPDATE CASCADE ON DELETE SET NULL,

status VARCHAR(50) NOT NULL CHECK (status IN ('new','in\_progress','resolved','dismissed')),

description TEXT NOT NULL CHECK (char\_length(description) > 0),

media JSONB

);

Листинг А.7 – Таблица complaints

CREATE TABLE core.warnings (

id\_warning SERIAL PRIMARY KEY,

id\_user INT NOT NULL REFERENCES core.users(id\_user) ON UPDATE CASCADE ON DELETE CASCADE,

id\_moderator INT NOT NULL REFERENCES core.users(id\_user) ON UPDATE CASCADE ON DELETE RESTRICT,

id\_complaint INT UNIQUE NOT NULL REFERENCES core.complaints(id\_complaint) ON UPDATE CASCADE ON DELETE SET NULL,

id\_project INT UNIQUE REFERENCES core.projects(id\_project) ON UPDATE CASCADE ON DELETE SET NULL,

message TEXT NOT NULL,

issued\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

expires\_at TIMESTAMP,

is\_resolved BOOLEAN NOT NULL DEFAULT FALSE

);

Листинг А.8 – Таблица warnings

CREATE TABLE core.audit\_logs (

id\_log SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES core.users(id\_user) ON UPDATE CASCADE ON DELETE SET NULL,

proc\_name VARCHAR(100),

action VARCHAR(10) NOT NULL CHECK (action IN ('INSERT','UPDATE','DELETE')),

table\_name VARCHAR(50) NOT NULL,

record\_id INTEGER NOT NULL,

old\_data JSONB,

new\_data JSONB,

changed\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP

);

Листинг А.9 – Таблица audit\_logs

CREATE TABLE core.notifications (

id\_notification SERIAL PRIMARY KEY,

id\_sender INT NOT NULL REFERENCES core.users(id\_user) ON DELETE CASCADE,

id\_receiver INT NOT NULL REFERENCES core.users(id\_user) ON DELETE CASCADE,

id\_project INT NOT NULL REFERENCES core.projects(id\_project) ON DELETE CASCADE,

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT uq\_invite\_per\_project UNIQUE (id\_receiver, id\_project)

);

Листинг А.10 – Таблица notifications

-- 1. role's view

CREATE OR REPLACE VIEW core.v\_roles AS

SELECT id\_role, role\_name FROM core.roles;

-- 2. user's view

CREATE OR REPLACE VIEW core.v\_users AS

SELECT

u.id\_user, r.role\_name, u.last\_name, u.first\_name,

u.middle\_name, u.gender, u.registration\_date,

u.last\_online\_time, u.rating

FROM core.users u

JOIN core.roles r ON u.role = r.id\_role;

-- 3. project's view

CREATE OR REPLACE VIEW core.v\_projects AS

SELECT

id\_project, id\_customer, title,

status, description, media

FROM core.projects;

-- 4. review's view

CREATE OR REPLACE VIEW core.v\_reviews AS

SELECT

id\_review, id\_order, id\_author, id\_recipient,

comment, rating, media

FROM core.reviews;

-- 5. order's view

CREATE OR REPLACE VIEW core.v\_orders AS

SELECT

id\_order, id\_project, id\_freelancer,

status, creation\_date, deadline

FROM core.orders;

-- 6. order\_extended view

CREATE OR REPLACE VIEW core.v\_order\_extended AS

SELECT

o.id\_order AS "OrderId",

o.status AS "OrderStatus",

o.creation\_date AS "OrderCreationDate",

o.deadline AS "OrderDeadline",

p.id\_project AS "ProjectId",

p.title AS "ProjectTitle",

p.status AS "ProjectStatus",

c.id\_user AS "CustomerId",

c.first\_name || ' ' || c.last\_name AS "CustomerFullName",

f.id\_user AS "FreelancerId",

COALESCE(f.first\_name || ' ' || f.last\_name, 'Не назначен')

AS "FreelancerFullName"

FROM core.v\_orders o

JOIN core.v\_projects p ON o.id\_project = p.id\_project

JOIN core.v\_users c ON p.id\_customer = c.id\_user

LEFT JOIN core.v\_users f ON o.id\_freelancer = f.id\_user;

-- 7. order\_archive's view

CREATE OR REPLACE VIEW core.v\_orders\_archive AS

SELECT

id\_order\_arc, id\_order,

id\_project, project\_title,

id\_customer, id\_freelancer,

status, creation\_date, deadline

FROM core.orders\_archive;

-- 8. order\_archive\_extended view

CREATE OR REPLACE VIEW core.v\_order\_archive\_extended AS

SELECT

a.id\_order AS "OrderId",

a.status AS "OrderStatus",

a.creation\_date AS "OrderCreationDate",

a.deadline AS "OrderDeadline",

a.id\_project AS "ProjectId",

a.project\_title AS "ProjectTitle",

'archived' AS "ProjectStatus",

a.id\_customer AS "CustomerId",

COALESCE(c.first\_name || ' ' || c.last\_name, 'Неизвестно') AS "CustomerFullName",

a.id\_freelancer AS "FreelancerId",

COALESCE(f.first\_name || ' ' || f.last\_name, 'Не назначен')

AS "FreelancerFullName"

FROM core.orders\_archive a

LEFT JOIN core.v\_users f ON f.id\_user = a.id\_freelancer

LEFT JOIN core.v\_users c ON c.id\_user = a.id\_customer;

-- 9. portfolio's view

CREATE OR REPLACE VIEW core.v\_portfolio AS

SELECT

id\_portfolio, id\_user, description,

media, skills, experience

FROM core.portfolio;

-- 10. orders\_reviews view — «архив-заказ + оба отзыва»

CREATE OR REPLACE VIEW core.v\_orders\_reviews AS

SELECT

oa.id\_order\_arc AS order\_id,

oa.creation\_date,

oa.project\_title,

oa.id\_customer,

cu.first\_name || ' ' || cu.last\_name AS customer\_fullname,

oa.id\_freelancer,

fu.first\_name || ' ' || fu.last\_name AS freelancer\_fullname,

cr.id\_review AS customer\_review\_id,

cr.comment AS customer\_comment,

cr.rating AS customer\_rating,

fr.id\_review AS freelancer\_review\_id,

fr.comment AS freelancer\_comment,

fr.rating AS freelancer\_rating

FROM core.orders\_archive oa

JOIN core.v\_users cu ON cu.id\_user = oa.id\_customer

LEFT JOIN core.v\_users fu ON fu.id\_user = oa.id\_freelancer

LEFT JOIN core.v\_reviews cr ON cr.id\_order = oa.id\_order\_arc

AND cr.id\_author = oa.id\_customer

LEFT JOIN core.v\_reviews fr ON fr.id\_order = oa.id\_order\_arc

AND fr.id\_author = oa.id\_freelancer;

-- 11. all user's counterparts workers

CREATE OR REPLACE VIEW core.v\_all\_counterparts AS

SELECT DISTINCT CASE WHEN o.id\_customer = u.id\_user THEN o.id\_freelancer ELSE o.id\_customer END AS counterpart\_id

FROM core.orders\_archive o

JOIN core.v\_users u ON u.id\_user IN (o.id\_customer, o.id\_freelancer);

-- 12. v\_complaints

CREATE OR REPLACE VIEW core.v\_complaints AS

SELECT

id\_complaint, id\_user, filed\_by, id\_project,

id\_moderator, status, description, media

FROM core.complaints;

-- 13. v\_user\_notifications

CREATE OR REPLACE VIEW core.v\_user\_notifications AS

SELECT n.id\_notification, n.id\_project,

p.title AS project\_title,

n.id\_sender,

s.first\_name||' '||s.last\_name AS sender\_name,

n.id\_receiver,

r.first\_name||' '||r.last\_name AS receiver\_name,

n.created\_at

FROM core.notifications n

JOIN core.projects p ON p.id\_project = n.id\_project

JOIN core.v\_users s ON s.id\_user = n.id\_sender

JOIN core.v\_users r on r.id\_user = n.id\_receiver;

-- 14. v\_user\_warnings

CREATE OR REPLACE VIEW core.v\_user\_warnings AS

SELECT

w.id\_warning,

w.id\_user AS user\_id,

u.first\_name||' '||u.last\_name AS user\_name,

w.id\_moderator AS moderator\_id,

m.first\_name||' '||m.last\_name AS moderator\_name,

w.message,

w.issued\_at,

w.expires\_at,

w.is\_resolved

FROM core.warnings w

JOIN core.users u ON w.id\_user = u.id\_user

JOIN core.users m ON w.id\_moderator = m.id\_user;

/\* svc\_mod \*/

-- 1. user's view

CREATE OR REPLACE VIEW core.v\_mod\_users AS

SELECT

u.id\_user, u.role AS id\_role, r.role\_name,u.last\_name,u.first\_name,

u.middle\_name, u.gender, u.phone\_number, u.email, u.registration\_date,

u.last\_online\_time, u.rating

FROM core.users u

JOIN core.roles r ON u.role = r.id\_role;

-- 2. complaint's view

CREATE OR REPLACE VIEW core.v\_mod\_complaints AS

SELECT

c.id\_complaint,

c.id\_user AS "UserComId",

u1.first\_name || ' ' || u1.last\_name AS "UserComName",

c.filed\_by AS "FiledById",

u2.first\_name || ' ' || u2.last\_name AS "FiledByName",

c.id\_moderator AS "ModeratorId",

c.id\_project AS "ProjectId",

p.title AS "ProjectTitle",

c.status AS "Status",

c.description AS "Description",

c.media AS "Media"

FROM core.complaints AS c

JOIN core.users AS u1 ON c.id\_user = u1.id\_user

JOIN core.users AS u2 ON c.filed\_by = u2.id\_user

LEFT JOIN core.projects AS p ON c.id\_project = p.id\_project

WHERE c.id\_user <> c.filed\_by;

Листинг А.11 – Представления базы данных

Приложение Б Процедуры и функции

CREATE OR REPLACE FUNCTION core.search\_users(

current\_user\_id INTEGER,

query TEXT DEFAULT ''

) RETURNS TABLE (

id INTEGER,

"FullName" TEXT,

"SkillsPreview" TEXT,

"CanInvite" BOOLEAN

) LANGUAGE sql STABLE

SECURITY DEFINER AS $$

SELECT

u.id\_user AS id,

u.first\_name || ' ' || u.last\_name AS "FullName",

COALESCE(skills.skills\_preview, '') AS "SkillsPreview",

-- ещё не приглашён?

NOT EXISTS (

SELECT 1

FROM core.v\_user\_notifications n

WHERE n.id\_sender = current\_user\_id

AND n.id\_receiver = u.id\_user

) AS "CanInvite"

FROM core.v\_users u

LEFT JOIN core.v\_portfolio p ON p.id\_user = u.id\_user

LEFT JOIN LATERAL (

SELECT string\_agg(DISTINCT skill, ', ') AS skills\_preview

FROM unnest(p.skills) AS skill

) skills ON TRUE

WHERE u.id\_user <> current\_user\_id

AND (

coalesce(query, '') = '' -- пустой запрос → без фильтра

OR lower(u.first\_name || ' ' || u.last\_name) LIKE '%' || lower(query) || '%'

OR EXISTS (

SELECT 1

FROM unnest(p.skills) AS skill

WHERE lower(skill) LIKE '%' || lower(query) || '%'

)

)

ORDER BY u.first\_name, u.last\_name;

$$;

Листинг Б.1 – Функция поиска исполнителей по введённым навыкам

CREATE OR REPLACE PROCEDURE core.mod\_issue\_warning(

p\_performer\_id INT,

p\_complaint\_id INT,

p\_target\_user INT,

p\_project\_id INT,

p\_message TEXT,

p\_expires\_days INT DEFAULT 7,

p\_delete\_project BOOLEAN DEFAULT FALSE

)

LANGUAGE plpgsql SECURITY DEFINER AS $$

DECLARE

v\_expires TIMESTAMP := current\_timestamp + (p\_expires\_days || ' days')::interval;

BEGIN

-- для аудита: кто и в какой процедуре

PERFORM set\_config('audit.user\_id', p\_performer\_id::text, false);

PERFORM set\_config('audit.proc\_name', 'core.mod\_issue\_warning', false);

INSERT INTO core.warnings(

id\_user, id\_moderator, id\_complaint, id\_project, message, expires\_at

) VALUES (

p\_target\_user, p\_performer\_id, p\_complaint\_id, p\_project\_id, p\_message, v\_expires

);

IF p\_delete\_project THEN

DELETE FROM core.projects

WHERE id\_project = p\_project\_id;

END IF;

END;

$$;

Листинг Б.2 – Процедура выдачи предупреждения нарушителю

CREATE OR REPLACE PROCEDURE core.user\_create\_order (

p\_performer\_id INTEGER,

p\_project INTEGER,

p\_freelancer INTEGER,

p\_status VARCHAR DEFAULT 'pending',

p\_deadline DATE DEFAULT NULL

) LANGUAGE plpgsql SECURITY DEFINER AS $$

BEGIN

PERFORM set\_config('audit.user\_id', p\_performer\_id::text, false);

PERFORM set\_config('audit.proc\_name', 'core.user\_create\_order', false);

IF EXISTS ( SELECT 1 FROM core.orders WHERE id\_project = p\_project ) THEN

RAISE EXCEPTION 'На этот проект уже есть заказ. Повторный отклик невозможен.';

END IF;

INSERT INTO core.orders(id\_project,id\_freelancer,status,deadline) VALUES(p\_project,p\_freelancer,p\_status,p\_deadline);

END;

$$;

Листинг Б.3 – Процедура создание заказа

CREATE OR REPLACE PROCEDURE core.user\_archive\_order (p\_performer\_id int, p\_order int)

LANGUAGE plpgsql SECURITY DEFINER AS $$

DECLARE

v\_order core.orders%ROWTYPE;

v\_proj core.projects%ROWTYPE;

v\_cust core.users%ROWTYPE;

v\_frel core.users%ROWTYPE;

BEGIN

PERFORM set\_config('audit.user\_id', p\_performer\_id::text, false);

PERFORM set\_config('audit.proc\_name', 'core.user\_archive\_order', false);

SELECT \* INTO v\_order FROM core.orders WHERE id\_order = p\_order;

IF NOT FOUND THEN RAISE EXCEPTION 'Заказ % не найден', p\_order; END IF;

SELECT \* INTO v\_proj FROM core.projects WHERE id\_project = v\_order.id\_project;

SELECT \* INTO v\_cust FROM core.users WHERE id\_user = v\_proj.id\_customer;

SELECT \* INTO v\_frel FROM core.users WHERE id\_user = v\_order.id\_freelancer;

INSERT INTO core.orders\_archive (

id\_order, id\_project, id\_customer,

id\_freelancer, status,

creation\_date, deadline,

project\_title)

VALUES (

v\_order.id\_order, v\_order.id\_project, v\_proj.id\_customer,

v\_order.id\_freelancer, v\_order.status,

v\_order.creation\_date, v\_order.deadline, v\_proj.title)

ON CONFLICT DO NOTHING;

IF v\_order.status = 'completed' THEN

DELETE FROM core.projects WHERE id\_project = v\_order.id\_project;

ELSE

DELETE FROM core.orders WHERE id\_order = v\_order.id\_order;

END IF;

END;

$$;

Листинг Б.4 – Процедура архивирования заказа и удаление из основного списка