Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка базы данных для проведения онлайн аукциона»

Студент ИУ7-63Б _____ А. И. Дегтярев
(Группа) (Подпись, дата) (И. О. Фамилия)

Руководитель курсовой работы _____ А. П. Ковтушенко
(Подпись, дата) (И. О. Фамилия)

2022 г.

РЕФЕРАТ

Расчетно-пояснительная записка 64 с., 19 рис., 9 табл., 21 источн., 3 прил.

Ключевые слова: Базы Данных, SQL, Аукцион

Объектом разработки является базы данных для хранения информации, необходимой для проведения онлайн аукциона.

Цель работы — разработать базу данных для проведения онлайн аукциона и реализовать сервер, предоставляющий программный доступ к ней.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить и проанализировать специфику проведения аукциона;
- проанализировать способы представления данных и выбрать оптимальный для поставленной задачи;
- описать используемые структуры данных;
- описать структуру разрабатываемого программного обеспечения;
- привести основные сценарии использования приложения;
- определить средства программной реализации;
- описать процесс сборки приложения;
- протестировать разработанное программное обеспечение.

В результате выполнения работы была спроектированы и разработаны база данных для проведения онлайн аукциона и сервер, предоставляющий к ней доступ.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Аукционы набирают популярность: так в 2021 году количество торгов в России выросло на 17% и объем продаж на 18% по сравнению с 2020 годом[1]. Это показывает, что у россиян постепенно появляется интерес к теме торгов и аукционов.

Целью моей работы является разработка базы данных для проведения онлайн аукциона и реализация сервера, предоставляющего программный доступ к ней.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить и проанализировать специфику проведения аукциона;
- проанализировать способы представления данных и выбрать оптимальный для поставленной задачи;
- описать используемые структуры данных;
- описать структуру разрабатываемого программного обеспечения;
- привести основные сценарии использования приложения;
- определить средства программной реализации;
- описать процесс сборки приложения;
- протестировать разработанное программное обеспечение.

1 Аналитический раздел

В данном разделе рассмотрены определение и классификация аукционов, порядок проведения и правовой аспект английского аукциона. Также, представлены варианты использования приложения, формализованы данные, построена диаграмма сущность-связь и рассмотрены состояния сущностей систем. Проанализированы и классифицированы модели хранения данных, выбраны и обоснованы критерии сравнения моделей, и из рассмотренных моделей выбрана оптимальная для решения поставленной задачи.

1.1 Постановка задачи

Необходимо спроектировать и реализовать базу данных, для хранения всей необходимой информации для проведения английского аукциона по сети. База данных должна содержать информацию о товарах, предложениях и движениях денежных средств. Также, нужно разработать программный интерфейс, который позволит работать с этой базой данных: создавать, получать, удалять и редактировать информацию о товарах, предложениях и транзакциях.

1.2 Определение аукциона

Аукцион можно определить по одному из его основных свойств: механизму достижения рыночного равновесия, уравнивания спроса и предложений. Среди всех доступных механизмов достижения рыночного равновесия аукцион отличается явным ценообразованием: всем участвующим в аукционе сторонам заранее известны правила, определяющие конечную цену. На аукционе чаще всего продают объекты, для которых не существует установившегося рынка, то есть они или редкие, или уникальные. Например, на международных аукционах покупаются: пушно-меховые товары, немытая шерсть, подержанные автомобили, картины, лошади. [2]

1.3 Классификация аукционов

С точки зрения открытости ставок аукционы делятся на следующие виды:

- **открытый аукцион** – аукцион, в котором все ставки известны его

участникам;

- **закрытый аукцион** – аукцион, в котором ставки участников известны только организатору.

По способу установления цены аукционы делятся на следующие виды:

- **аукцион с повышением цены** начинается с минимальной цены, а ставки должны увеличиваться. Покупателем является тот, кто предложил наибольшую ставку;

- **аукцион с понижением цены** начинается с максимальной цены, которая в ходе аукциона постепенно уменьшается. Покупателем является тот, кто первым согласился на предложенную цену.

Английский аукцион является классическим, наиболее часто применяемым, открытым аукционом с повышением цены.

1.4 Стадии проведения аукциона

Различают четыре стадии проведения аукционов [3]:

- **подготовка** – на этом этапе владелец товара доставляет его на склад организатора аукциона, составляются каталоги, осуществляется рекламная деятельность;

- **осмотр товаров** – во время осмотра товара потенциальные покупатели имеют возможность ознакомиться с выставленными для продажи товарами. Осмотр является важным этапом проведения аукционных торгов, так как в случае приобретения товара претензии к его качеству впоследствии не принимаются;

- **аукционный торг** – главная стадия аукциона, на которой определяется стоимость товара

- **оформление и исполнение аукционной сделки**.

1.5 Аукционный торг

В случае английского аукциона торг проходит следующим образом [3]:

1. Аукционист объявляет начальную цену и спрашивает: "Кто больше?"

2. Покупатель, желающий приобрести лот по более высокой цене, называет новую цену, которая выше предыдущей на величину не ниже минимальной надбавки, указанной в правилах проведения торгов.

3. Аукционист называет номер покупателя, под которым он зарегистрирован на аукционе, новую цену лота и повторяет вопрос: "Кто больше?"

4. Если после троекратного повторения вопроса не следует нового предложения, аукционист ударяет молотком, подтверждая продажу лота покупателю, который последним назвал наивысшую цену.

1.6 Оформление и исполнение аукционной сделки

Согласно Гражданскому кодексу Российской Федерации по результатам торгов организатор и лицо, выигравшее торги, подписывают в день проведения аукциона или конкурса протокол о результатах торгов, который имеет силу договора. При заключении договора с лицом, выигравшим торги, сумма внесенного им перед началом торгов задатка, засчитывается в счет исполнения обязательств по заключенному договору. [4]

Для совершения операций с денежными средствами, права на которые принадлежат другому лицу-бенефициару, в Гражданском кодексе Российской Федерации предусмотрен номинальный счет [5]. Номинальный счет формально имеет одного владельца, а на нем лежат денежные средства, права на которые принадлежат другому лицу. По завершению аукционной сделки, с помощью банка, должна происходить передача прав на денежные средства от лица, выигравшего торги, к владельцу предмета торгов и организатору.

1.7 Варианты использования

В аукционе можно выделить следующих участников:

- **владелец** – пользователь, который владеет некоторым товаром и имеет желание его продать через аукцион;

- **покупатель** – пользователь, который хочет приобрести товар на аукционе;

- **менеджер** – представитель компании, контролирующий правильность заполнения анкет, информации о продуктах и управляющий номинальными счетами аукционной платформы.

На рисунке 1.1 представлена диаграмма вариантов использования приложения каждым из участников.

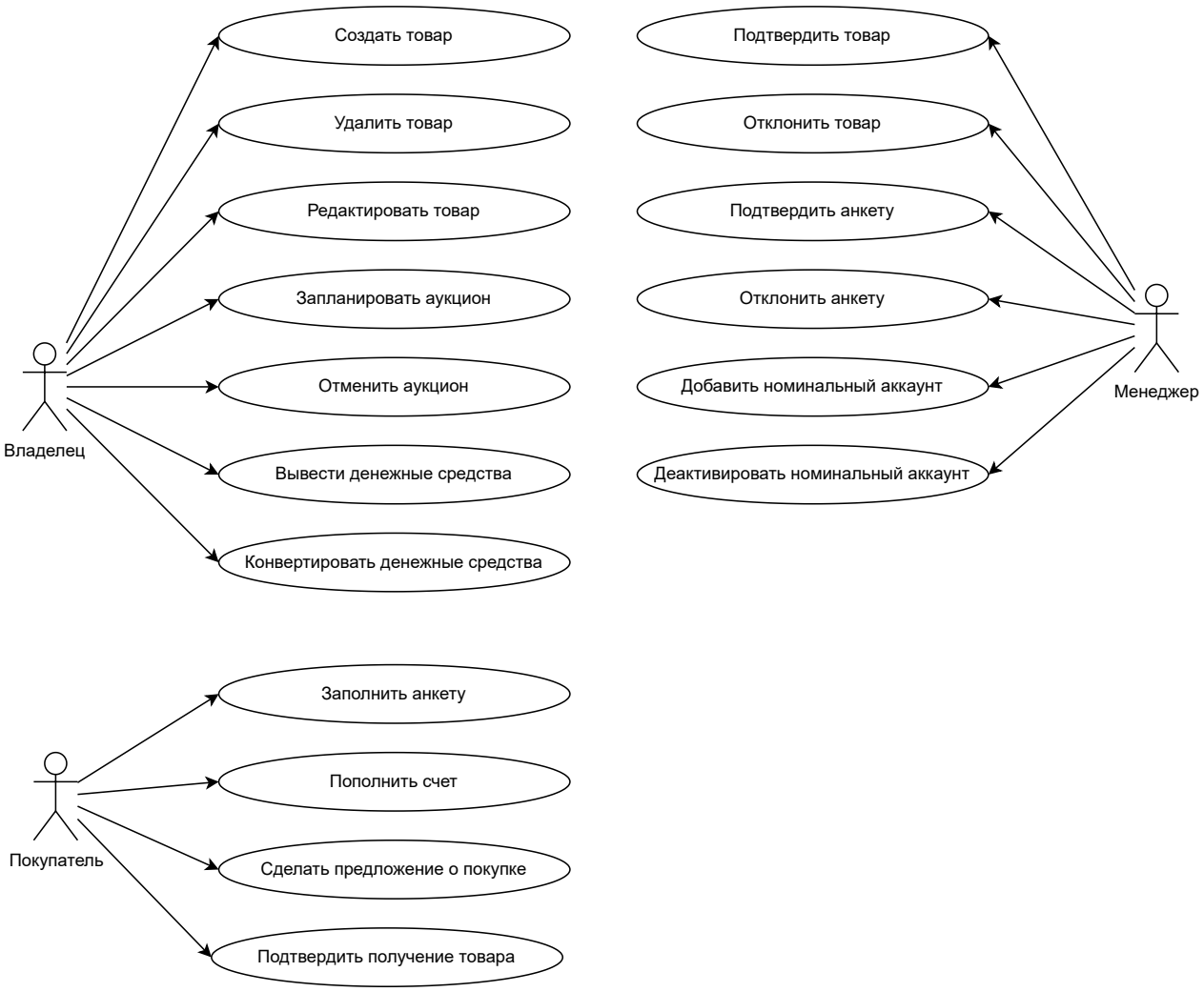

Рисунок 1.1 – Диаграмма вариантов использования

1.8 Формализация данных

База данных должна хранить информацию о:

- пользователях;
- товарах;
- аукцион;

- ставках;
- заключенных договорах;
- номинальных счетах;
- движении денежных средств.

Таблица 1.1 – Сущности и их данные

Сущность	**Данные**
Пользователь	Имя, почта, пароль, номер телефона, паспортные данные
Товар	Название, описание, фотографии, собственник
Аукцион	Товар, организатор, дата проведения
Ставка	Сумма, дата создания
Заключенный договор	Аукцион, ставка, победитель
Номинальный счет	Банк, в котором был открыт счет; реквизиты счета
Движение денежных средств	Сумма, отправитель, получатель, статус

На рисунке 1.2 представлена диаграмма модели сущность-связь в нотации Чена. На рисунках 1.3-1.8 представлены диаграммы состояний сущностей системы.

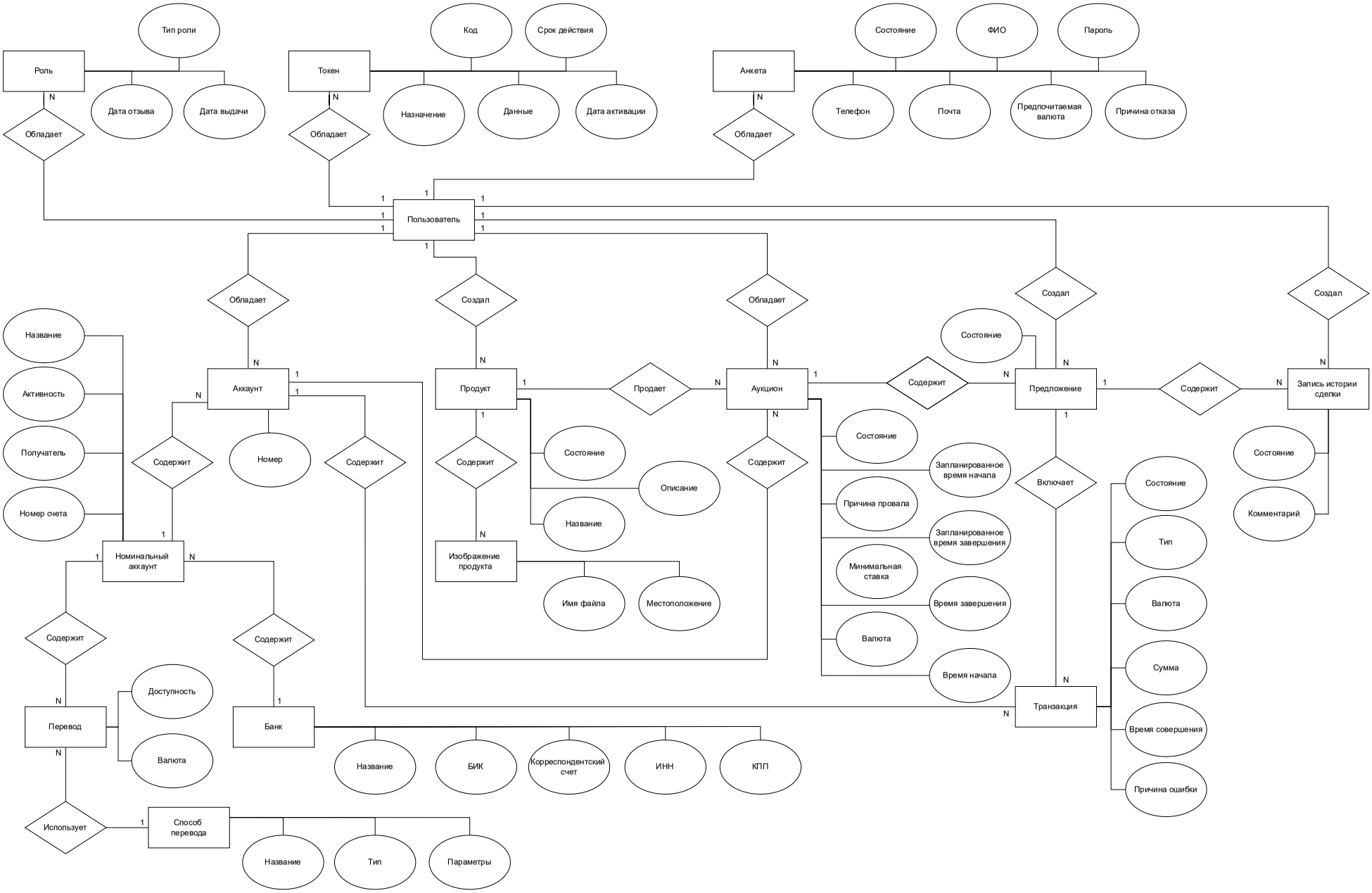

Рисунок 1.2 – Диаграмма модели сущность-связь в нотации Чена

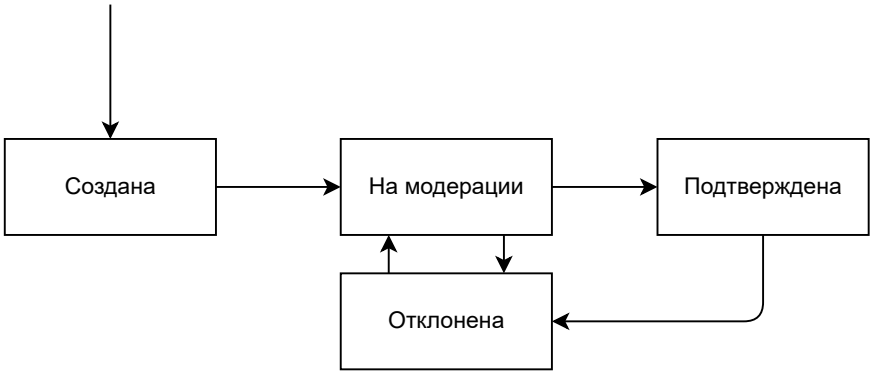

Рисунок 1.3 – Диаграмма состояний анкеты

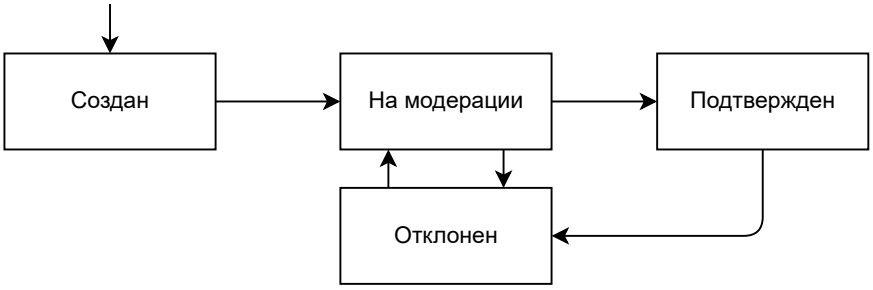

Рисунок 1.4 – Диаграмма состояний продукта

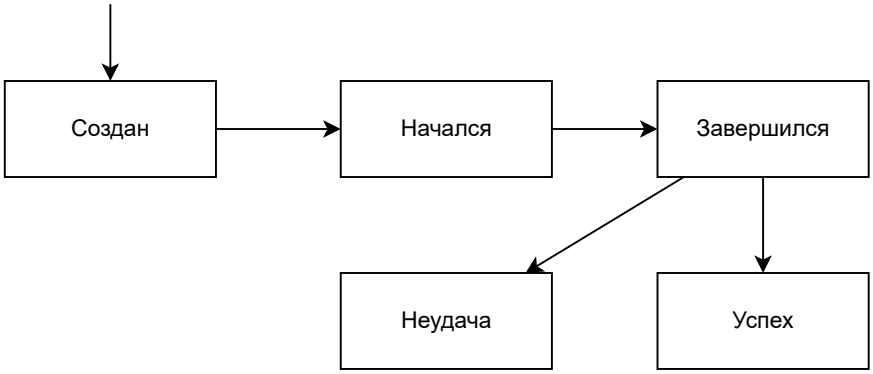

Рисунок 1.5 – Диаграмма состояний аукциона

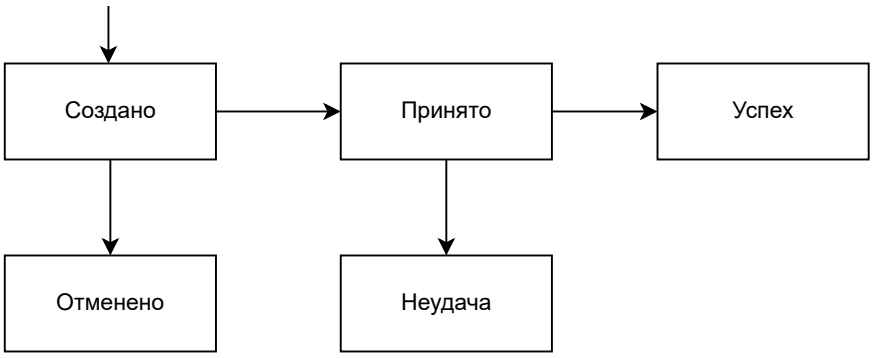

Рисунок 1.6 – Диаграмма состояний предложения

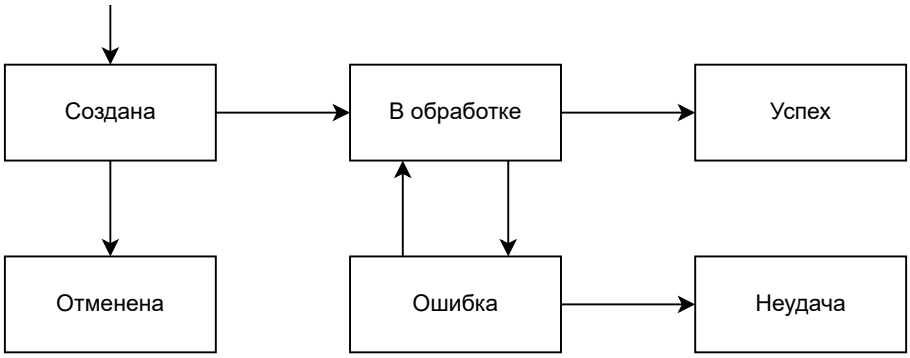

Рисунок 1.7 – Диаграмма состояний транзакции

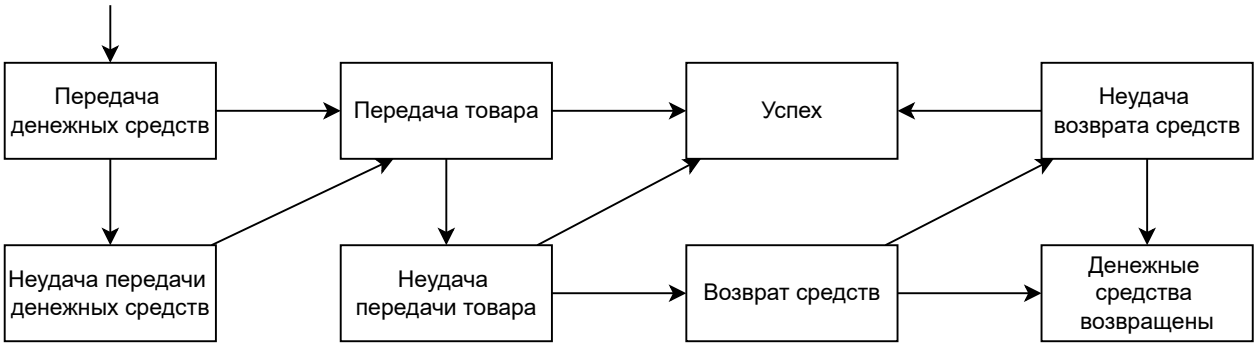

Рисунок 1.8 – Диаграмма состояний сделки

1.9 Классификация БД

База данных – совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных. [6]

Модель данных – это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует

пользователь. Эти объекты позволяют моделировать структуру данных, а операторы — поведение данных. [7]

Каждая БД строится на основе некоторой явной или неявной модели данных. Все БД, построенные на одной и той же модели данных, относят к одному типу. Поэтому, классификация БД по используемой модели данных является основной. Наиболее известными моделями являются следующие [8]:

- **дореляционные модели данных** являются предшественниками реляционных баз данных. Наиболее известные представители:
  - **сетевая модель данных** представляет данные в виде типов записей. Эта модель также представляет ограниченный тип отношения один ко многим, называемый типом набора;
  - **иерархическая модель** представляет данные в виде иерархической древовидной структуры. Каждая ветвь иерархии представляет собой ряд связанных записей;
- **реляционная модель данных** представляет данные в виде отношений или таблиц;
- **постреляционные модели данных** появились после реляционная модели и удовлетворяют требованиям, на которые реляционные базы данных не способны. К таким требованиям относятся: большое количество данных, огромное количество пользователей, сложные данные. Наиболее известными представителями постреляционных моделей являются:
  - **документоориентированная модель данных** предназначена для хранения иерархических структур данных, которые в этой модели называются документами. Документ имеет структуру дерева, в листьях которого находятся хранимые данные. Документы можно сгруппировать в коллекции;
  - **модель данных ключ-значение** определяет структуру хранения в виде ассоциативного массива. Это позволяет создавать системы, главной особенностью которых является быстрое извлечение данных, так как структура хранения жестко задана.

1.10 Выбор модели хранения

Для решения поставленной задачи были выбраны следующие критерии:

- **наличие ACID** необходимо для проведения торга, так как вовремя создания предложений возможно состояние гонки;
- **соблюдение целостности ссылочных данных** необходимо для гарантии целостности связей между сущностями системы.

В таблице 1.2 представлено сравнение наиболее известных моделей хранения данных.

Таблица 1.2 – Сравнение моделей хранения

Модель хранения данных	**ACID**	**Ссылочная целостность**
сетевая модель	-	-
иерархическая модель	-	+
реляционная модель	+	+
документоориентированная модель	-	-
модель ключ-значение	-	-

Из приведенного сравнения можно сделать вывод, что для решения поставленной задачи лучше всего подходит реляционная модель данных, к тому же формализованные данные хорошо представляются в виде отношений.

Вывод

В данном разделе были рассмотрены определение и классификация аукционов, порядок проведения и правовой аспект английского аукциона. Были представлены варианты использования приложения, формализованы данные, построена диаграмма сущность-связь, рассмотрены состояния сущностей систем. Проанализированы и классифицированы модели хранения данных, выбраны и обоснованы критерии сравнения моделей, и из рассмотренных моделей была выбрана оптимальная для решения поставленной задачи.

2 Конструкторский раздел

В данном разделе будет спроектирована база данных, приведена ER-диаграмма сущностей базы данных, описаны поля всех таблиц и объекты, необходимые для соблюдения целостности данных. Будет спроектировано приложение, предоставляющее программный интерфейс, который позволяет работать с базой данных, приведены верхнеуровневое разбиение на компоненты и описание этих компонентов. А так же будут приведены диаграммы последовательностей для основных действий в приложении.

2.1 Проектирование базы данных

На рисунке 2.1 представлена ER-диаграмма сущностей базы данных. В базе данных будет 16 таблиц, каждая из которых представляет отдельную сущность.

Описание полей таблицы `users`, представляющую пользователей системы:

- `id` – уникальный идентификатор пользователя;
- `created_at` – временная метка регистрации пользователя;
- `deleted_at` – временная метка, когда пользователь считается удаленным.

Пользователь – общая сущность для менеджера, владельца и покупателя.

В этой и остальных таблицах с полем `deleted_at` должен быть реализован подход мягкого удаления, когда данные реально не удаляются из хранилища, а просто помечаются как удаленные. [9]

Описание полей таблицы `migrations`, представляющую миграции базы данных:

- `id` – уникальный идентификатор миграции.

По наличию миграций можно определить, какие изменения были применены к базе данных, и последовательно применить недостающие.

Описание полей таблицы `user_forms`, представляющую анкеты пользователей:

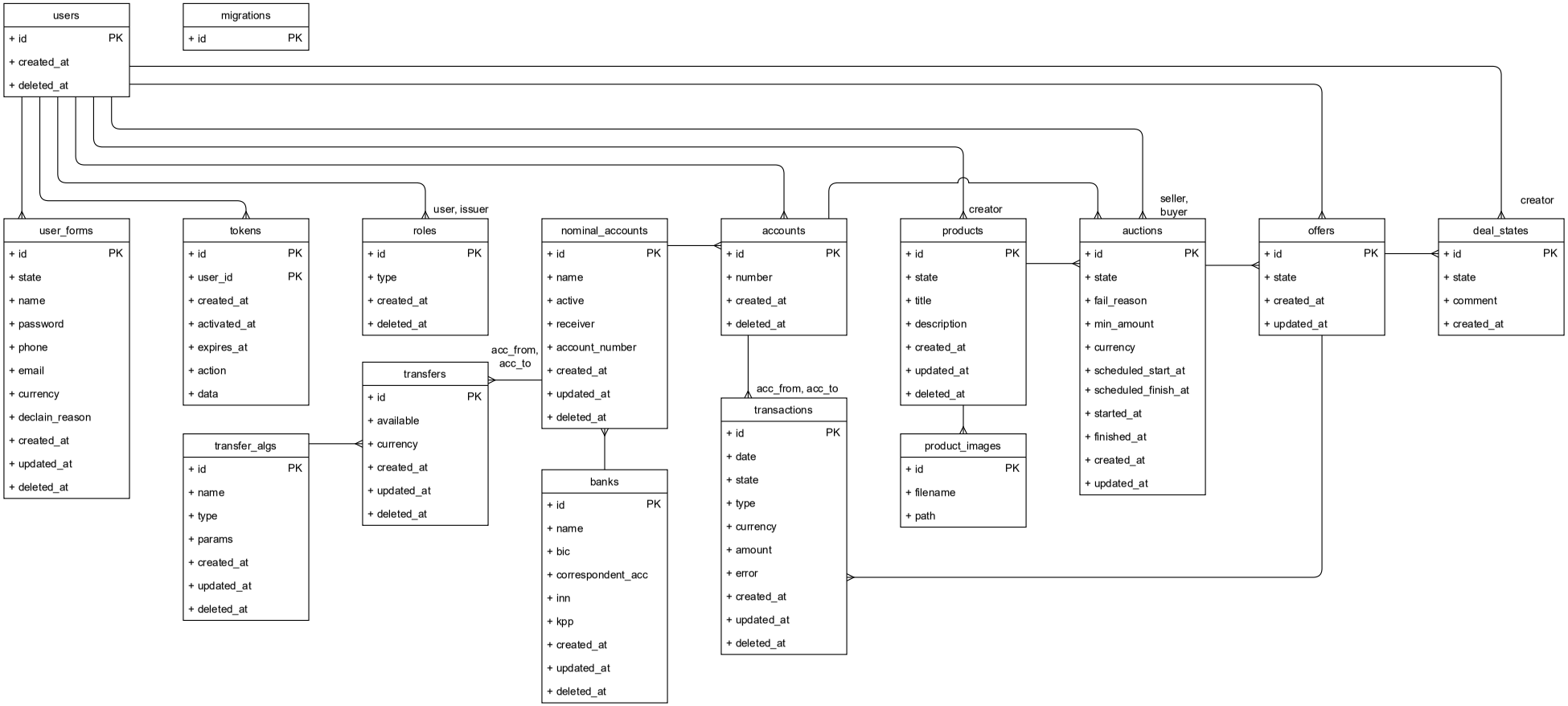

Рисунок 2.1 – ER-диаграмма сущностей базы данных в нотации Мартина

- `id` – уникальный идентификатор анкеты;
- `state` – текущее состояние анкеты. Принимает одно из значений, описанных на рисунке диаграммы состояний 1.3;
- `name` – ФИО пользователя;
- `password` – хэш пароля пользователя;
- `phone` – номер телефона пользователя;
- `email` – адрес электронной почты пользователя;
- `currency` – предпочитаемая валюта;
- `declain_reason` – причина отклонения анкете. Заполняется менеджером при переходе в состояние "Отклонена";
- `created_at`, `updated_at`, `deleted_at` – временные метки создания, последнего обновления и удаления соответственно.

Когда пользователь изменяет анкету, то она так же должна пройти проверку менеджером. Пока эта проверка не будет выполнена, будет использоваться прошлая анкета.

Описание полей таблицы `tokens`, представляющую токены:

- `id` – уникальный код;
- `user_id` – идентификатор пользователя, которому принадлежит этот код;
- `created_at`, `activated_at`, `expires_at` – временные метки создания, активации и истечения срока действия токена;
- `action` – назначение, действие токена;
- `data` – данные, необходимые для выполнения действия токена.

Описание полей таблицы `roles`, представляющую роли пользователей:

- `id` – уникальный идентификатор роли;

- `type` – тип роли. Может принимать одно из значений: менеджер, администратор;
- `created_at`, `deleted_at` – временные метки создания и удаления соответственно.

Описание полей таблицы `accounts`, представляющую счета пользователей:

- `id` – уникальный идентификатор счета;
- `number` – идентификатор счета, полученный из банка;
- `created_at`, `deleted_at` – временные метки создания и удаления соответственно.

Описание полей таблицы `nominal_accounts`, представляющую счета пользователей:

- `id` – уникальный идентификатор счета;
- `name` – название номинального аккаунта в рамках платформы;
- `receiver`, `account_number` – банковские реквизиты счета: получатель и счет соответственно;
- `created_at`, `updated_at`, `deleted_at` – временные метки создания, последнего обновления и удаления соответственно.

Описание полей таблицы `banks`, представляющую банки:

- `id` – уникальный идентификатор счета;
- `name`, `bic`, `correspondent_acc`, `inn`, `kpp` – название банка, реквизиты банка: банковский идентификационный код, корреспондентский счет, идентификационный номер налогоплательщика и код причины постановки на учёт соответственно;
- `created_at`, `updated_at`, `deleted_at` – временные метки создания, последнего обновления и удаления соответственно.

Описание полей таблицы `transfers`, представляющую порядок перевода средств между номинальными счетами:

- `id` – уникальный идентификатор перевода;
- `currency_from`, `currency_to` – валюты из которой и в которую происходит перевод;
- `created_at`, `updated_at`, `deleted_at` – временные метки создания, последнего обновления и удаления соответственно.

Описание полей таблицы `transfer_algs`, представляющую алгоритмы перевода средств между номинальными счетами:

- `id` – уникальный идентификатор алгоритма;
- `name` – название алгоритма;
- `type`, `params` – тип и параметры алгоритма перевода;
- `created_at`, `updated_at`, `deleted_at` – временные метки создания, последнего обновления и удаления соответственно.

Описание полей таблицы `transfer_algs`, представляющую алгоритмы перевода средств между номинальными счетами:

- `id` – уникальный идентификатор алгоритма;
- `name` – название алгоритма;
- `type`, `params` – тип и параметры алгоритма перевода;
- `created_at`, `updated_at`, `deleted_at` – временные метки создания, последнего обновления и удаления соответственно.

Описание полей таблицы `transactions`, представляющую движение денежных средств между счетами пользователей:

- `id` – уникальный идентификатор транзакции;
- `date` – временная метка проведения транзакции;

- `state` – состояние транзакции. Принимает одно из значений, описанных на рисунке диаграммы состояний 1.7;
- `type` – тип транзакции;
- `currency` – валюта транзакции;
- `amount` – сумма транзакции;
- `error` – ошибка, произошедшая при проведении транзакции в банке;
- `created_at`, `updated_at`, `deleted_at` – временные метки создания, последнего обновления и удаления соответственно.

Описание полей таблицы `products`, представляющую товары:

- `id` – уникальный идентификатор товара;
- `state` – состояние товара. Принимает одно из значений, описанных на на рисунке диаграммы состояний 1.4;
- `title`, `description` – название и описание товара;
- `declain_reason` – причина отклонения. Заполняется менеджером при переходе в состояние "Отклонен;
- `created_at`, `updated_at`, `deleted_at` – временные метки создания, последнего обновления и удаления соответственно.

Описание полей таблицы `product_images`, представляющую изображения товаров:

- `id` – уникальный идентификатор изображения;
- `filename` – название файла;
- `path` – путь к файлу в файловом хранилище.

Описание полей таблицы `auctions`, представляющую аукцион:

- `id` – уникальный идентификатор аукциона;

- `state` – состояние аукциона. Принимает одно из значений, описанных на на рисунке диаграммы состояний 1.5;
- `fail_reason` – причина неудачи проведения аукциона. Заполняется менеджером при переходе в состояние "Неудача";
- `min_amount` – минимальная ставка;
- `currency` – валюта проведения аукциона;
- `scheduled_start_at`, `scheduled_finish_at` – запланированные даты начала и конца аукциона;
- `started_at`, `finish_at` – реальные даты начала и завершения аукциона;
- `created_at`, `updated_at` – временные метки создания и последнего обновления соответственно.

Описание полей таблицы `offers`, представляющую предложение о покупке:

- `id` – уникальный идентификатор аукциона;
- `state` – состояние предложения. Принимает одно из значений, описанных на на рисунке диаграммы состояний 1.6;
- `created_at`, `updated_at` – временные метки создания и последнего обновления соответственно.

Описание полей таблицы `deal_states`, представляющую историю состояния сделок:

- `id` – уникальный идентификатор состояния;
- `state` – состояние сделки. Принимает одно из значений, описанных на на рисунке диаграммы состояний 1.8;
- `comment` – комментарий перехода в состояние;
- `created_at` – временные метки создания состояния сделки.

2.2 Соблюдение целостности данных

В таблицах 2.1-2.5 представлены объекты базы данных, необходимые для соблюдения целостности.

Таблица 2.1 – Пользовательские типы данных

Объект	**Описание**
Короткий идентификатор	Строка, состоящая из 11 символов, поддерживаемых унифицированным указателем ресурса (URL)
Роли пользователей	Одно из значений: «Администратор», «Менеджер»
Валюты	Перечисляемый тип, состоящий из текстовых кодов валют, поддерживаемых приложением
Тип транзакции	Одно из значений: «Пополнение», «Конвертация валюты», «Возврат», «Покупка», «Вывод»
Состояния анкеты	Перечисляемый тип, который включает в себя состояния, представленные на рисунке 1.3
Состояния товаров	Перечисляемый тип, который включает в себя состояния, представленные на рисунке 1.4
Состояния аукциона	Перечисляемый тип, который включает в себя состояния, представленные на рисунке 1.5
Состояния предложения	Перечисляемый тип, который включает в себя состояния, представленные на рисунке 1.6
Состояния транзакции	Перечисляемый тип, который включает в себя состояния, представленные на рисунке 1.7
Состояния сделки	Перечисляемый тип, который включает в себя состояния, представленные на рисунке 1.8
Код токена	шестизначное десятичное число

Таблица 2.2 – Значения по умолчанию

Таблица	**Поле**	**Значение**
любая с указанным полем	`created_at`	текущее время
	`updated_at`	текущее время
	`state`	начальное состояние на соответствующей таблице диаграмме состояний

Таблица 2.3 – Правила

Таблица	**Правило**
auctions	только при state = «Создан» может отсутствовать currency
	min_amount может быть задано, только если определено currency
	при state ≠ «Создан» должно быть заполнено started_at
	при state ≠ «Создан» ∧ state ≠ «Начался» должно быть заполнено finished_at
auctions	при state = «Успех» должно быть заполнено buyer_id
	только при state = «Создан» может отсутствовать seller_account_id
transactions	amount > 0
	account_from_id должно быть заполнено всегда кроме случая, когда type = «Пополнение» это поле должно отсутствовать
	account_to_id должно быть заполнено всегда кроме случая, когда type = «Вывод» это поле должно отсутствовать
	при type = «Пополнение» ∨ type = «Конвертация валюты» должно отсутствовать offer_id, при остальных type поле необходимо
	при state = «Успех» должно быть заполнено date

В таблице 2.4 представлены внешние ключи с много-однозначной связью.

Таблица 2.4 – Внешние ключи

Таблица	**Поле**	**Таблица, на которую указывает**
roles	issuer_id	users(id)
	user_id	
user_forms		
tokens		
nominal_accounts	bank_id	banks(id)
accounts	user_id	users(id)
	nominal_account_id	nominal_accounts(id)
products	creator_id	users(id)
auctions	product_id	products(id)
	seller_id	users(id)
	buyer_id	
product_images	product_id	products(id)
offers	auction_id	auctions(id)
	user_id	users(id)
transactions	account_from_id	accounts(id)
	account_to_id	
	offer_id	offers(id)
transfers	account_from_id	nominal_accounts(id)
	account_to_id	
	alg_id	transfer_algs(id)
deals	creator_id	users(id)
	offer_id	offers(id)

Таблица 2.5 – Триггеры

Название	**Описание**
Триггер добавления короткого идентификатора	Триггер создает уникальный в таблице короткий идентификатор. Схема триггера представлена на рисунке 2.2
Триггер добавления кода токена	Триггер создает уникальный для пользователя код токена. Схема триггера представлена на рисунке 2.3

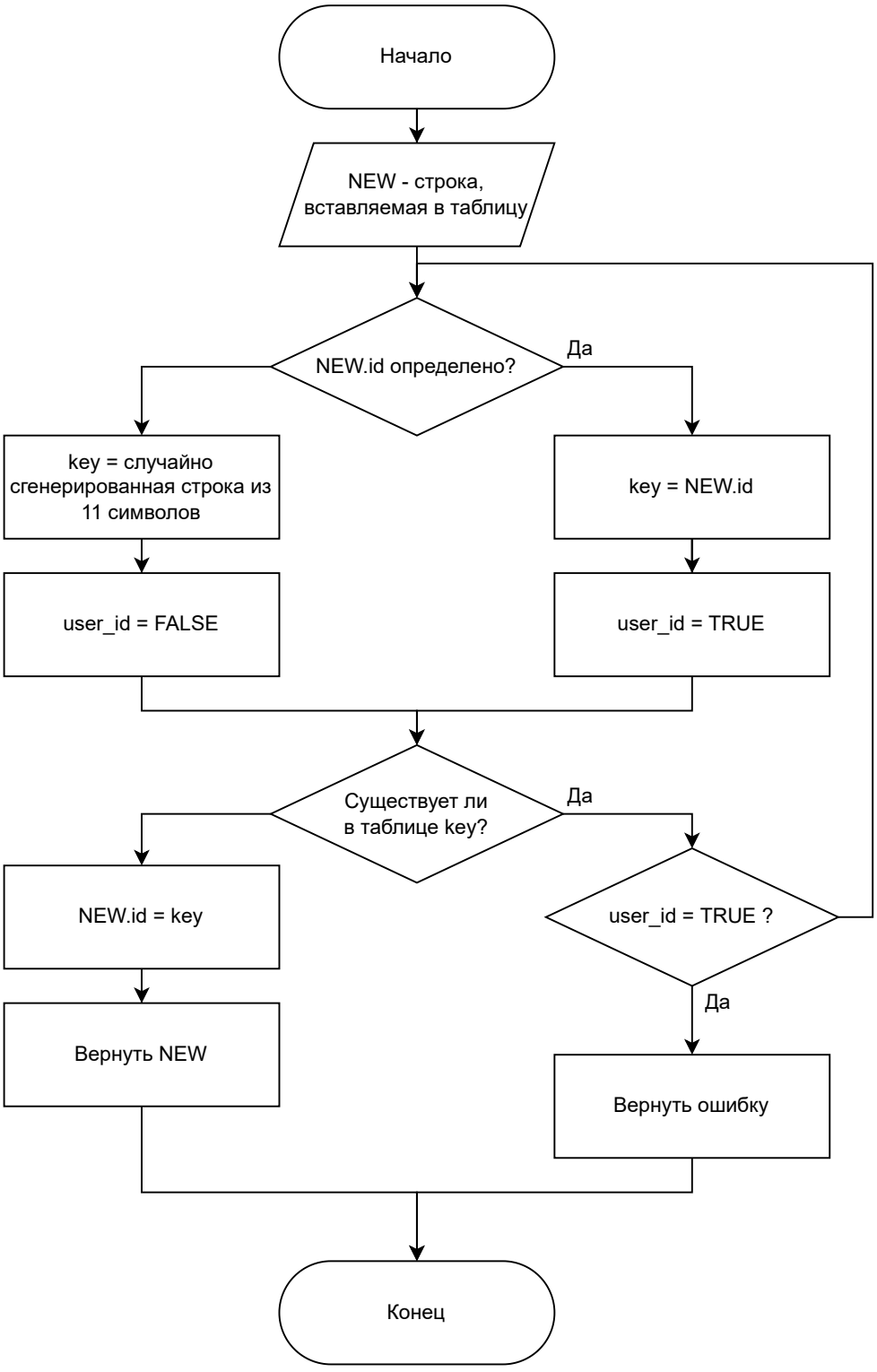

Рисунок 2.2 – Схема алгоритма триггера добавления короткого идентификатора

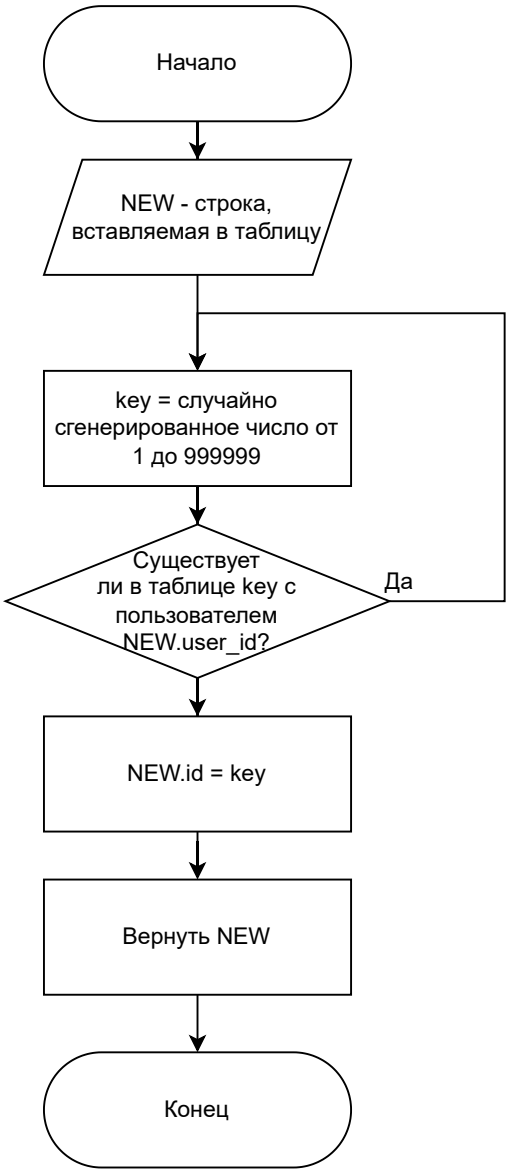

Рисунок 2.3 – Схема алгоритма триггера добавления кода токена

2.3 Проектирование приложения

По условию постановки задачи необходимо разработать программный интерфейс, который позволит работать с базой данных. На рисунке 2.4 представлено верхнеуровневое разбиение приложения на компоненты.

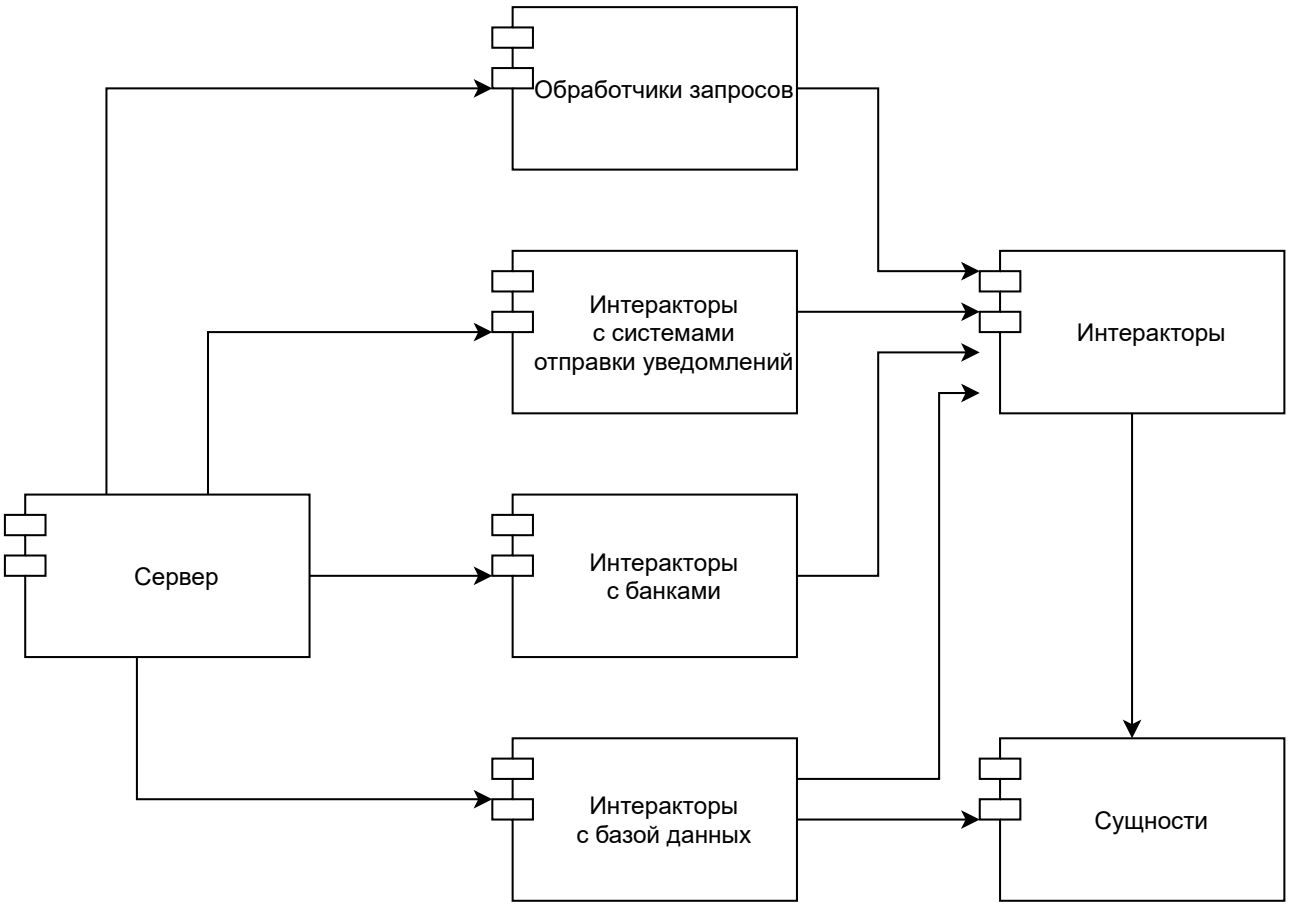

Рисунок 2.4 – Верхнеуровневое разбиение на компоненты

Описание компонентов приложения:

- **сервер** создает все остальные компоненты, производит инъекцию зависимостей, запускает основной цикл приложения;
- **обработчики запросов** преобразовывают входные данные интеракторов, вызывают их и преобразованный результат отдают наружу;
- **интеракторы с системами отправки уведомлений** производят взаимодействие с внешними системами отправки уведомлений, например, SMS или Email оповещений;
- **интеракторы с банками** абстрагируют взаимодействие с банками;

- **интеракторы с базой данных** производят взаимодействие с базой данных;
- **интеракторы** производят взаимодействие между сущностями системы;
- **сущности** – сущности системы.

2.4 Сценарии использования

На рисунках 2.5-2.11 представлены сценарии использования.

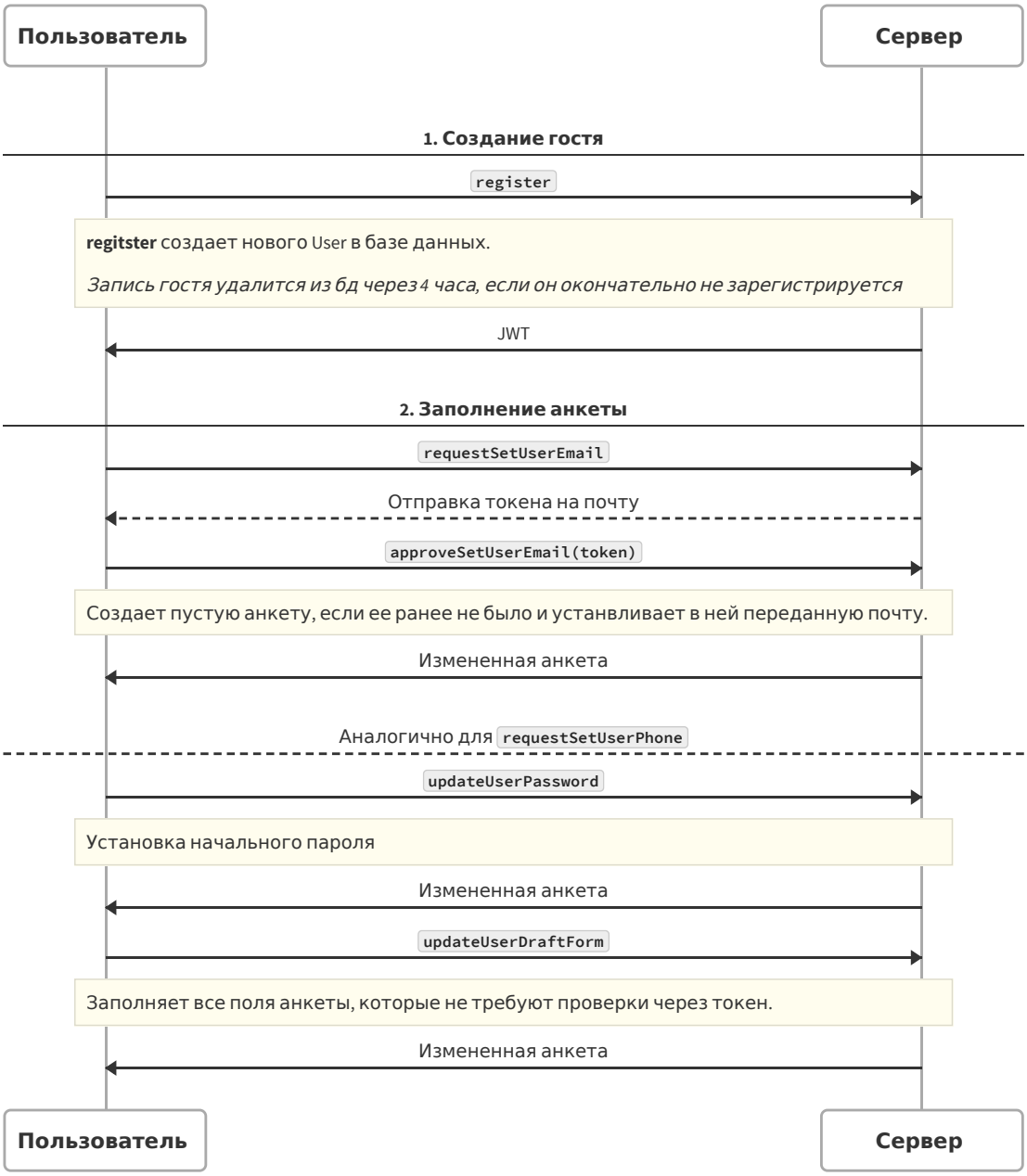

Рисунок 2.5 – Диаграмма последовательности регистрации пользователя

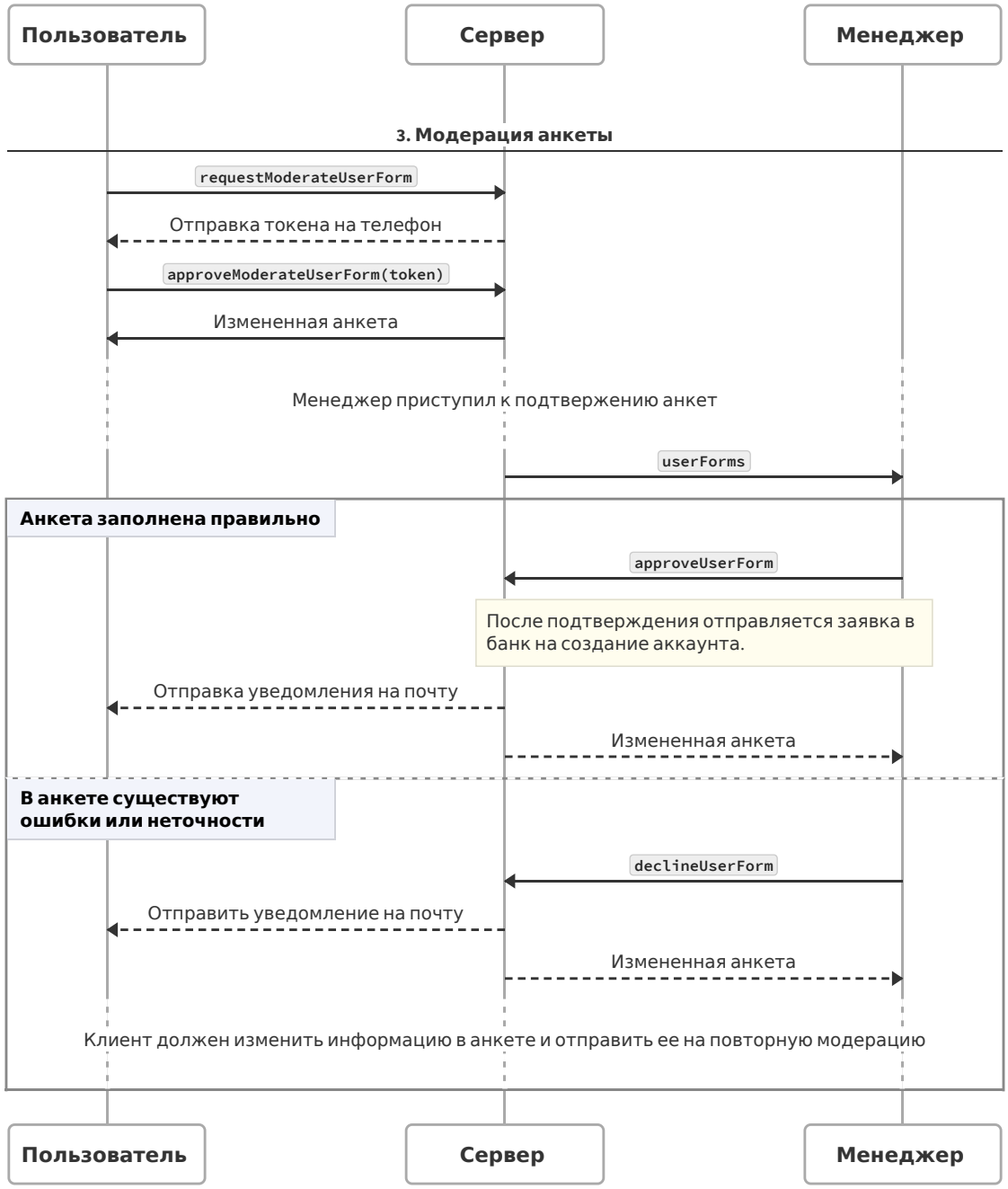

Рисунок 2.6 – Продолжение диаграммы последовательности регистрации пользователя

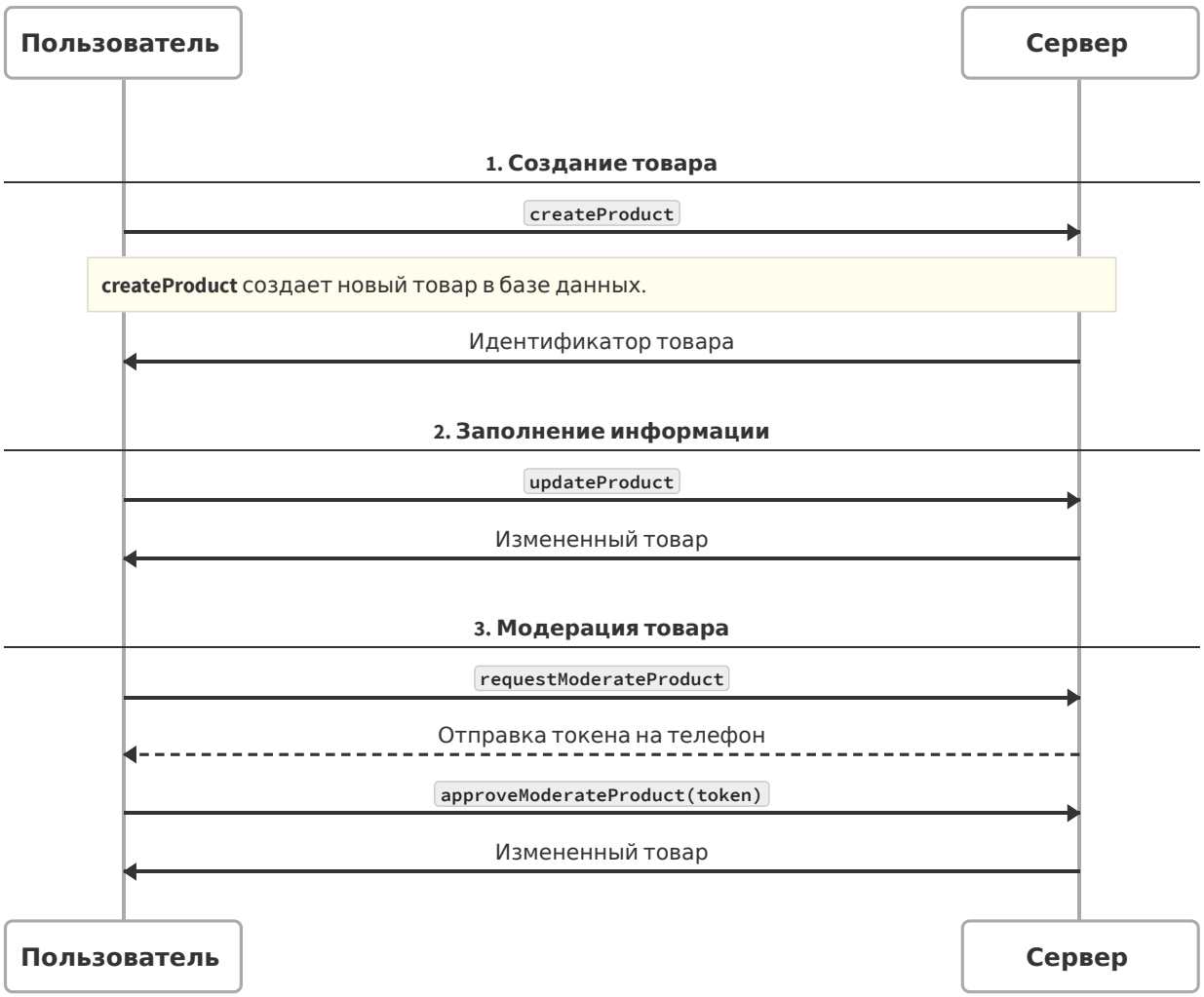

Рисунок 2.7 – Диаграмма последовательности создания товара

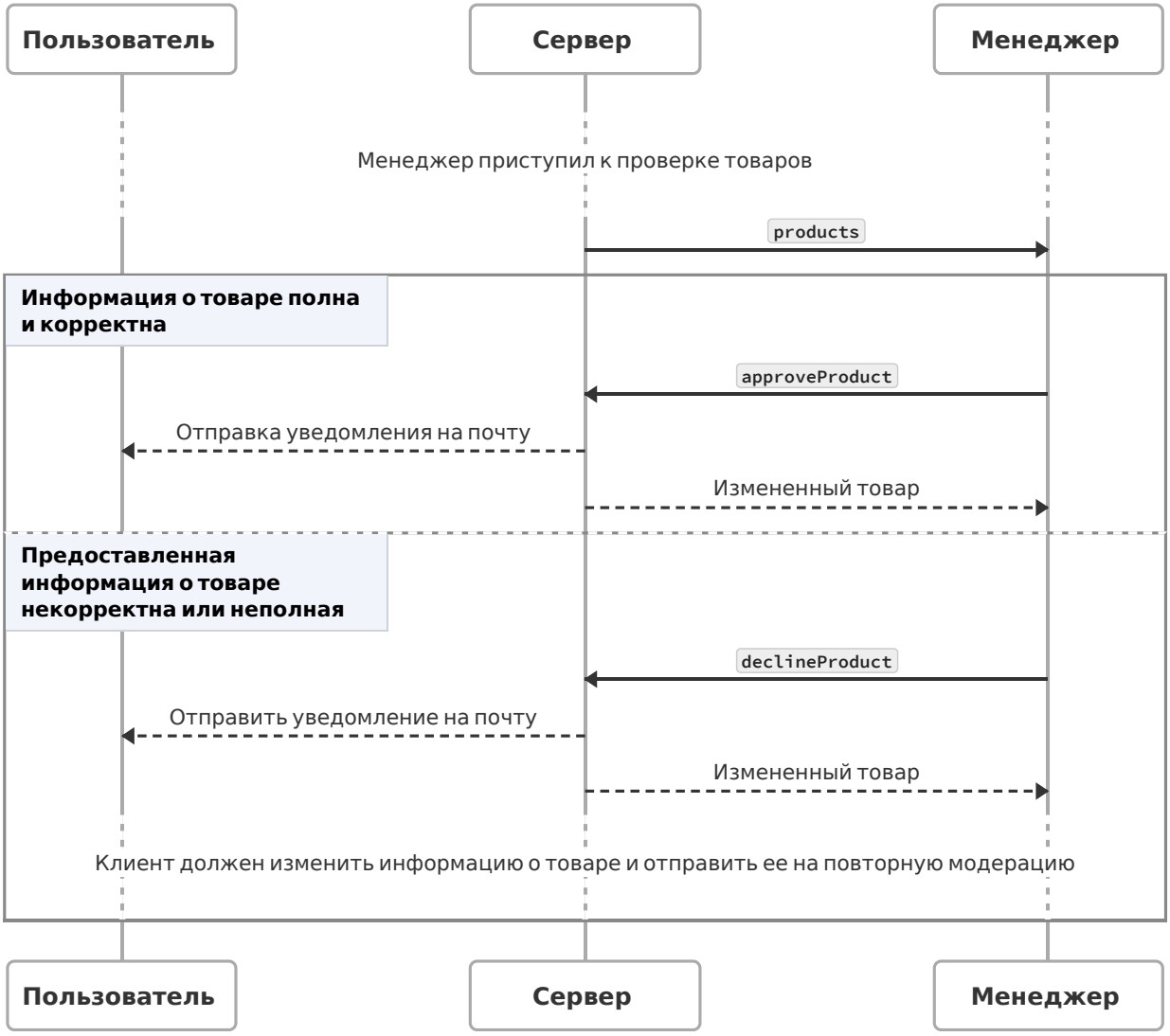

Рисунок 2.8 – Продолжение диаграммы последовательности создания товара

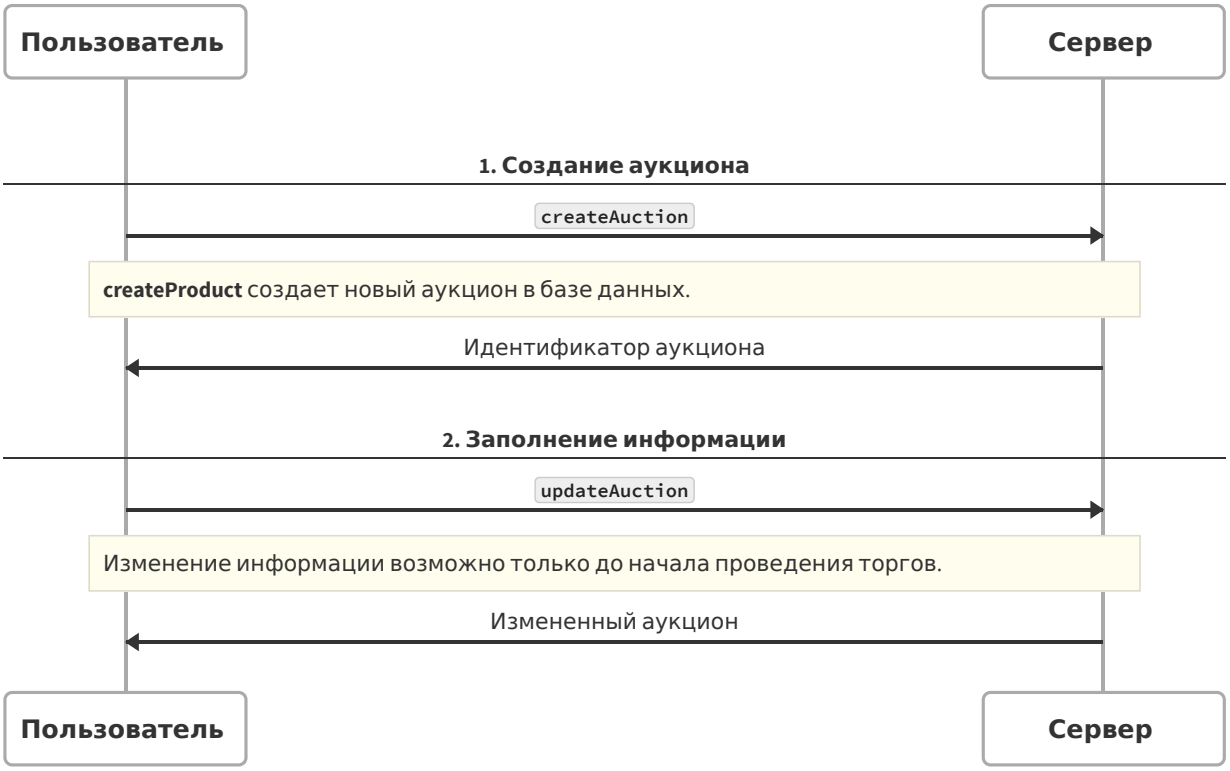

Рисунок 2.9 – Диаграмма последовательности создания аукциона

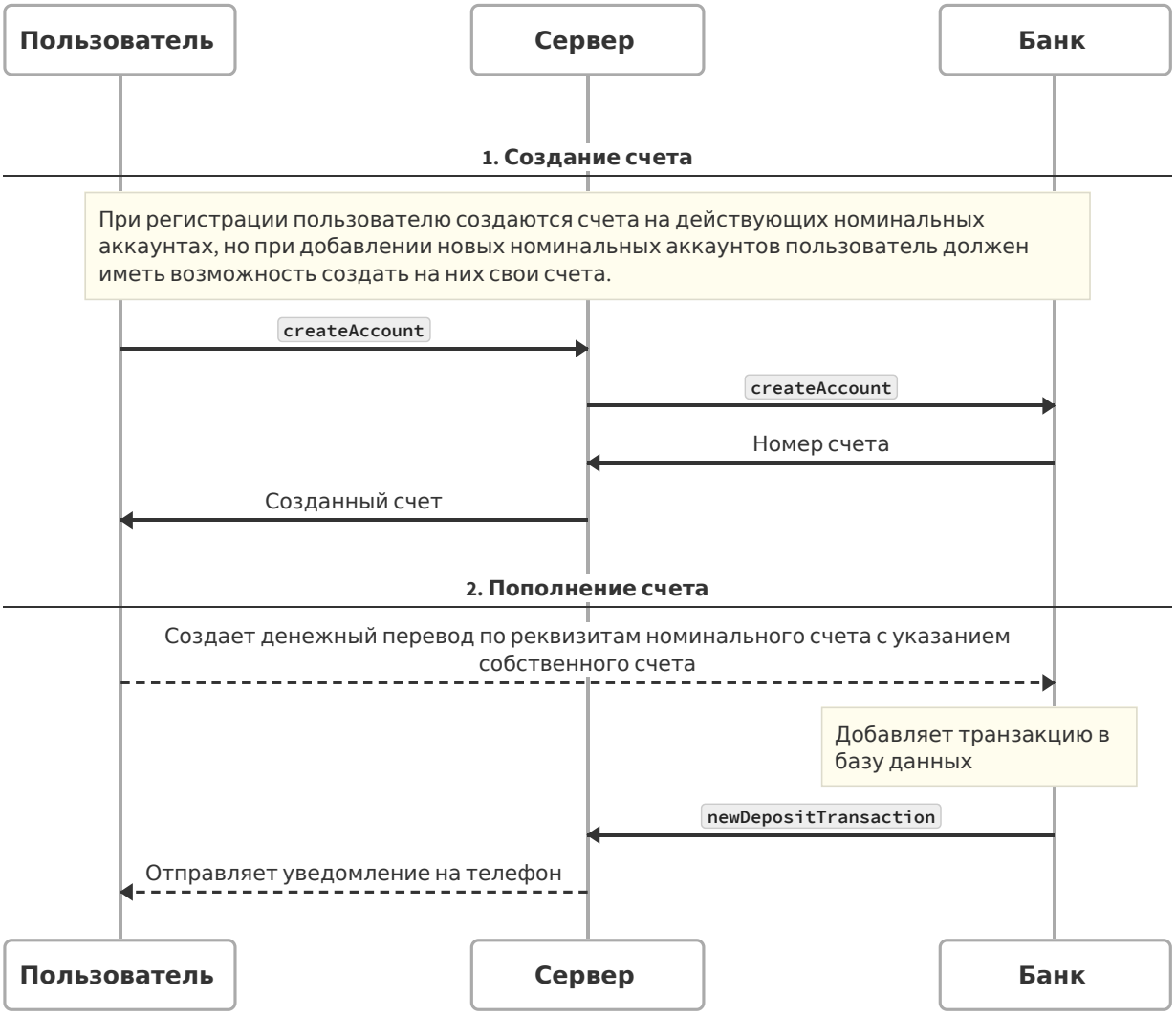

Рисунок 2.10 – Диаграмма последовательности пополнения счета

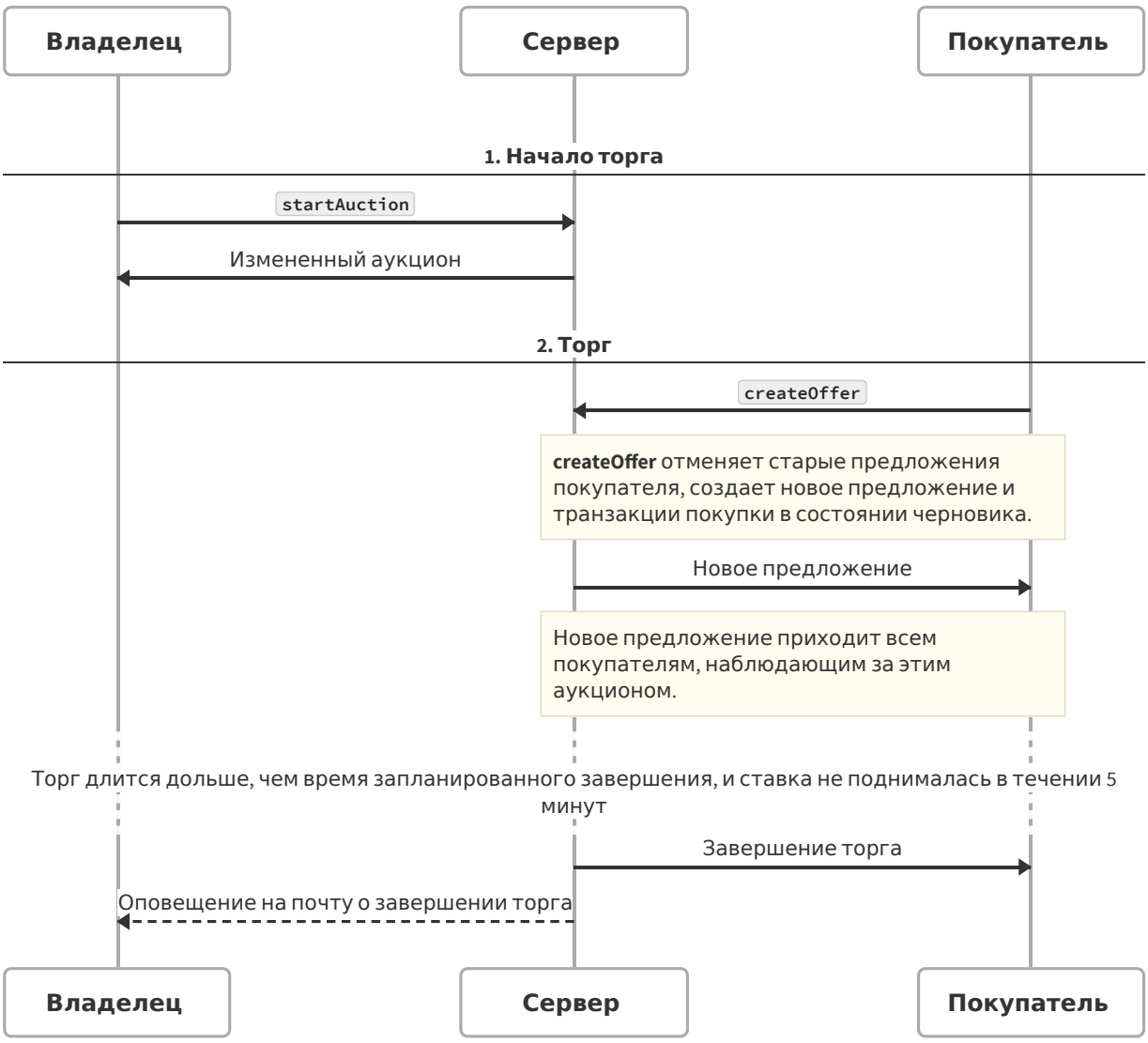

Рисунок 2.11 – Диаграмма последовательности проведения торга

Вывод

В данном разделе была спроектирована база данных, приведена ER-диаграмма сущностей базы данных, описаны поля всех таблиц и объекты, необходимые для соблюдения целостности данных. Было спроектировано приложение, предоставляющее программный интерфейс, который позволяет работать с базой данных, приведены верхнеуровневое разбиение на компоненты и описание этих компонентов. А так же были приведены диаграммы последовательностей для основных действий в приложении.

3 Технологический раздел

В данном разделе будет произведен обзор и сравнение существующих реляционных СУБД по составленным критериям и выбор СУБД, которая является оптимальной для решения поставленной задачи. Будут определены средства программной реализации и описана инициализация и система безопасности базы данных. А так же будут приведены примеры запросов к базе данных и примеры работы приложения.

3.1 Обзор существующих реляционных СУБД

3.1.1 MySQL

MySQL[10] – это база данных компании Oracle, впервые выпущенная в 1995 году. Она долгое время был на вершине многих рейтинговых списков. Это связано с тем, что это одна из первых баз данных с открытым исходным кодом, наполненная полезным и надежным функционалом. Oracle также предлагает платную версию MySQL с дополнительными функциями и поддержкой. MySQL являются хорошей отправной точкой для веб-проектов, но проигрывает в случаях, когда нужны расширенные функции защиты данных и когда нужна поддержку полуструктурированными данными, такими как JSON.

3.1.2 Oracle

Разработанная в 1979 году, Oracle[11] остается популярной базой данных, особенно для СУБД корпоративного уровня. Это одна из самых зрелых и стабильных баз данных на сегодняшний день. Он используется крупнейшими компаниями из списка Fortune 500 по всему миру для своих транзакций. Она предлагает продвинутые функции как для структурированных, так и для полуструктурированных данных, поддерживает блокчейн-таблицы, облегчает мгновенные транзакции и помогает создавать как OLAP, так и OLTP в одном экземпляре базы данных. Однако у нее есть существенный недостаток по сравнению с другими базами данных: она не имеет открытого исходного кода. Стоимость запуска базы данных Oracle для предприятия с несколькими сотнями сотрудников может достигать тысяч долларов.

3.1.3 PostgreSQL

PostgreSQL[12] была разработана в Калифорнийском университете в Беркли и выпущена в 1989 году. Изначальное название POSTGRES является преемником базы данных INGRES. В 1996 году его название стало PostgreSQL, чтобы указать на поддержку SQL. По инновациям и возможностям PostgreSQL, несомненно, находится на первом месте. PostgreSQL обрабатывает полуструктурированные данные, такие как JSON, и имеет поддержку распределенного SQL. Последнее полезно при работе с миллионами транзакций в Интернете. PostgreSQL имеет является СУБД с открытым исходным кодом и работает во всех основных операционных системах, таких как Windows, MacOS, а также в системах семейства Unix. У нее есть большое сообщество пользователей, которые разрабатывают плагины и библиотеки. Более того имеется возможность писать скрипты на Python и запускать их в PostgreSQL.

3.2 Выбор СУБД

Для выбора СУБД были составлены следующие критерии:

- **поддержка JSON** нужна для реализации сущности *Токен*. Токеном подтверждаются разные действия пользователя, для которых нужны свои данные, структура которых заранее не определена;
- **бесплатное использование** СУБД необходимо для минимизации затрат на разработку системы.

Сравнение наиболее известных СУБД представлено в таблице 3.1.

Таблица 3.1 – Сравнение СУБД

СУБД	**поддержка JSON**	**Бесплатное использование**
MySQL	-	+
Oracle	+	-
PostgreSQL	+	+

Из приведенного сравнения можно сделать вывод, что для реализации поставленной задачи лучше всего подходит PostgreSQL, так как является СУБД с открытым исходным кодом и позволяет обрабатывает JSON.

3.3 Средства реализации

Для взаимодействия с базой данных необходимо написать приложение. Приложение было спроектировано в разделе 2.3. Для его реализации были выбраны следующие технологии:

- язык программирования Go[13], как наиболее подходящий для написания веб-серверов;
- Gin Web Framework[14] – HTTP фреймворк;
- gqlgen[15] – библиотека для написания GraphQL[16] серверов;
- GORM[17] – ORM, написанная на Go;
- Docker Compose[18] позволяет развернуть и запустить отдельные части сервера.

3.4 Инициализация базы данных

Для инициализации базы данных использовался специальный скрипты миграций. Каждый скрипт писался по одному шаблону: сначала идет проверка наличия идентификатора миграции в таблице миграций, и при его отсутствии, применяется тело миграции. В конце этого тела идентификатор миграции добавляется в таблицу миграций. Весь скрипт миграции происходит одной транзакцией и завершается оператором `COMMIT`. В начале первого скрипта миграции - инициализации базы данных - происходит создание таблицы миграций.

В приложениях А и Б приведены скрипты инициализации базы данных и создания пользователей уровня базы данных соответственно.

Для развертки базы данных используется `Dockerfile`, код которого приведен на листинге 3.1. Помимо самой базы данных `Postgres` в контейнер помещается расширение `pg_cron`[19], которое позволяет периодично запускать задачи, такие как удаление гостей из таблицы пользователей.

3.5 Запросы к базе данных

На листингах 3.2 и 3.3 приведены сложные запросы к базе данных, использованные в сервере. В приложении В представлена последовательность

запросов с использованием транзакции.

Листинг 3.1 – Конфигурация контейнера с базой данных

```
FROM postgres:14.2-alpine AS build

RUN apk add --no-cache git make gcc musl-dev clang llvm

RUN git clone -b v1.4.1 https://github.com/citusdata/pg_cron.git
RUN cd pg_cron && make && make install

FROM postgres:14.2-alpine

# found this files using `find . * | grep "pg_cron"`
COPY --from=build /usr/local/lib/postgresql/bitcode/pg_cron /usr/
   local/lib/postgresql/bitcode/pg_cron
COPY --from=build /usr/local/lib/postgresql/bitcode/pg_cron.index
   .bc /usr/local/lib/postgresql/bitcode/pg_cron.index.bc
COPY --from=build /usr/local/lib/postgresql/pg_cron.so /usr/local
   /lib/postgresql/pg_cron.so
COPY --from=build /usr/local/share/postgresql/extension/pg_cron.
   control /usr/local/share/postgresql/extension/pg_cron.control
COPY --from=build /usr/local/share/postgresql/extension/pg_cron-*
    /usr/local/share/postgresql/extension/
```

Листинг 3.2 – Получение свободных средств на счете

```
SELECT trs.currency, trs.account_id, SUM(trs.amount) as amount
FROM (
  SELECT currency, account_from_id as account_id, -amount as
     amount
  FROM "transactions"
  JOIN (
      SELECT *
      FROM "accounts"
      WHERE id = 'test-account-id'
      AND "accounts"."deleted_at" IS NULL
  ) a ON account_from_id = a.id
    AND (
      transactions.state IN (
        'SUCCEEDED', 'PROCESSING', 'ERROR'
      )
      OR (
        transactions.type IN ('BUY')
        AND transactions.state IN ('CREATED')
      )
    )
  WHERE "transactions"."deleted_at" IS NULL
  UNION ALL
  SELECT currency, account_to_id as account_id, amount
  FROM "transactions"
  JOIN (
    SELECT *
    FROM "accounts"
    WHERE id = 'test-account-id'
    AND "accounts"."deleted_at" IS NULL
  ) a ON account_to_id = a.id
  AND transactions.state IN (
    'SUCCEEDED', 'PROCESSING', 'ERROR'
  )
  WHERE "transactions"."deleted_at" IS NULL
) trs
GROUP BY trs.currency, trs.account_id
```

Листинг 3.3 – Получение товаров, владельцем которых является определенный пользователь

```
SELECT *
FROM "products"
JOIN (
  SELECT *,
    ROW_NUMBER() OVER(
      PARTITION BY ofd.product_id
      ORDER BY ofd.from_date DESC
    ) as owner_n
  FROM (
    SELECT id as product_id, creator_id as owner_id, created_at
       as from_date
    FROM "products"
    WHERE "products"."deleted_at" IS NULL
    UNION ALL
    SELECT products.id as product_id, auctions.buyer_id as
       owner_id, auctions.finished_at as from_date
    FROM "products"
    JOIN auctions ON auctions.product_id = products.id
    AND auctions.state = 'SUCCEEDED'
    WHERE "products"."deleted_at" IS NULL
  ) as ofd
) ofd ON products.id = ofd.product_id
AND ofd.owner_n = 1
AND ofd.owner_id IN ('test-owner-id')
WHERE "products"."deleted_at" IS NULL
ORDER BY created_at desc
```

3.6 Система безопасности сервера БД

Безопасность сервера БД обеспечивается с помощью ролевой модели на уровне базы данных. Выделенные роли базы данных представлены в таблице 3.2. В приложении Б представлен скрипт создания ролей.

Таблица 3.2 – Роли базы данных

Роль	**Права**
server	чтение, добавление и обновление всех таблиц
bankgate	чтение, добавление и обновление транзакций
dealer	чтение и обновление аукционов и предложений; чтение, добавление и обновление истории сделок
viewer	чтение всех таблиц

3.7 Примеры работы

API сервера можно использовать при помощи любого инструмента, позволяющего отправлять HTTP запросы. Для приведенных примеров работы 3.4-3.7 была использована команда `gq` из библиотеки `graphqlurl`[20]. Она позволяет делать запросы к GraphQL серверу в одной строке и ее интерфейс аналогичен `curl`[21].

Листинг 3.4 – Регистрация гостя

```
$ URL='http://[::1]:8080/graphql'
$ gq $URL -H 'Content-Type: application/json' -q '
mutation {
  register {
    token
  }
}'
Executing query... done
{
  "data": {
    "register": {
      "token": "eyJhbG...Yw2rFg"
    }
  }
}
$ TOKEN1="eyJhbG...Yw2rFg"
```

Листинг 3.5 – Привязка пользователя к почте

```
$ # Запрос установления почты
$ gq $URL -H 'Content-Type: application/json' \
  -H "Authorization: $TOKEN1" -q '
mutation {
  requestSetUserEmail(input: {
    email: "test@example.com"
  })
}'
Executing query... done
{
  "data": {
    "requestSetUserEmail": true
  }
}
$ # Подтверждение установления новой почты
$ gq $URL -H 'Content-Type: application/json' \
  -H "Authorization: $TOKEN1" -q '
mutation {
  approveSetUserEmail(input: { token: "364956" }) {
    user {
      draftForm {
        email
      }
    }
  }
}'
Executing query... done
{
  "data": {
    "approveSetUserEmail": {
      "user": {
        "draftForm": {
          "email": "test@example.com"
        }
      }
    }
  }
}
```

Листинг 3.6 – Вход по почте и паролю

```
$ gq $URL -H 'Content-Type: application/json' -q '
mutation {
  login(input: {
    username: "test@example.com",
    password: "12345"
  }) {
    token
  }
}'
Executing query... done
{
  "data": {
    "login": {
      "token": "eyJhbG...zHi8ik"
    }
  }
}
```

Листинг 3.7 – Получение черновика анкеты текущего пользователя

```
$ gq $URL -H 'Content-Type: application/json' \
  -H "Authorization: $TOKEN1" -q '
query {
  viewer {
    draftForm {
      id
      email
      state
      phone
      name
    }
  }
}'
Executing query... done
{
  "data": {
    "viewer": {
      "draftForm": {
        "id": "bc97be4a-bc50-48eb-a7e3-7057463c695f",
        "email": "test@example.com",
        "state": "CREATED",
        "phone": "+79998887766",
        "name": "test-client"
      }
    }
  }
}
```

Вывод

В данном разделе был произведен обзор и сравнение существующих реляционных СУБД по составленным критериям, выбрана СУБД, которая является оптимальной для решения поставленной задачи. Были определены средства программной реализации и описана инициализация и система безопасности базы данных. А так же были приведены примеры запросов к базе данных и примеры работы приложения.

ЗАКЛЮЧЕНИЕ

Во время выполнения курсовой работы было сделано следующее:

- изучена и проанализирована специфика проведения аукциона;
- проанализированы способы представления данных и выбран оптимальный способ для поставленной задачи;
- описаны используемые структуры данных и объекты базы данных, необходимые для соблюдения целостности;
- описана структура разрабатываемого программного обеспечения;
- приведены основные сценарии использования приложения;
- определены средства программной реализации;
- описан процесс сборки приложения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рынок русского искусства и арт-рынок России. Итоги и основные цифры 2021 года [Электронный ресурс]. — Режим доступа: `https://artinvestment.ru/russian-art-market-reports/2021.html` дата обращения: 03.07.2022.
2. *Menezes F. M.*, *Monteiro P. K.* An introduction to auction theory. — OUP Oxford, 2004.
3. *Стровский Л. Е.* Внешнеэкономическая деятельность предприятия : учебник для студентов вузов, обучающихся по экономическим специальностям. — Москва : ЮНИТИ-ДАНА, 2017.
4. Российская Федерация. Законы. О внесении изменений в часть первую Гражданского кодекса Российской Федерации : Федеральный закон 08.03.2015 N 42-ФЗ.
5. Российская Федерация. Законы. О внесении изменений в части первую и вторую Гражданского кодекса Российской Федерации и отдельные законодательные акты Российской Федерации : Федеральный закон от 26.07.2017 N 212-ФЗ.
6. ГОСТ Р ИСО МЭК ТО 10032-2007: Эталонная модель управления данными (идентичен ISO/IEC TR 10032:2003 Information technology — Reference model of data management).
7. *Дейт К. Д.* Введение в системы баз данных. — Вильямс, 2008.
8. *Watt A.*, *Eng N.* Database design. — 2014.
9. Soft-deletion is actually pretty hard - Tree Web Solutions [Электронный ресурс]. — Режим доступа: `https://treewebsolutions.com/articles/soft-deletion-is-actually-pretty-hard-14` дата обращения: 31.05.2022.
10. MySQL [Электронный ресурс]. — Режим доступа: `https://www.mysql.com` дата обращения: 09.05.2022.
11. Database Services | Oracle [Электронный ресурс]. — Режим доступа: `https://www.oracle.com/database` дата обращения: 09.05.2022.

12. PostgreSQL: The world's most advanced open source database [Электронный ресурс]. — Режим доступа: https://www.postgresql.org дата обращения: 09.05.2022.

13. The Go Programming Language [Электронный ресурс]. — Режим доступа: https://go.dev дата обращения: 31.05.2022.

14. Gin Web Framework [Электронный ресурс]. — Режим доступа: https://gin-gonic.com дата обращения: 31.05.2022.

15. gqlgen [Электронный ресурс]. — Режим доступа: https://gqlgen.com дата обращения: 31.05.2022.

16. GraphQL | A query language for your API [Электронный ресурс]. — Режим доступа: https://graphql.org дата обращения: 31.05.2022.

17. GORM [Электронный ресурс]. — Режим доступа: https://gorm.io дата обращения: 31.05.2022.

18. Overview of Docker Compose | Docker Documentation [Электронный ресурс]. — Режим доступа: https://docs.docker.com/compose дата обращения: 02.06.2022.

19. citusdata/pg_cron: Run periodic jobs in PostgreSQL [Электронный ресурс]. — Режим доступа: https://github.com/citusdata/pg_cron дата обращения: 02.06.2022.

20. hasura/graphqurl: curl for GraphQL with autocomplete, subscriptions and GraphiQL. Also a dead-simple universal javascript GraphQL client. [Электронный ресурс]. — Режим доступа: https://github.com/hasura/graphqurl дата обращения: 03.07.2022.

21. curl [Электронный ресурс]. — Режим доступа: https://curl.se дата обращения: 03.07.2022.

ПРИЛОЖЕНИЕ А
Первичная инициализация базы данных

Листинг А.1 – Первичная инициализация базы данных

```
--
-- Initialize database with basic tables
--
CREATE TABLE IF NOT EXISTS migrations (
    id VARCHAR PRIMARY KEY
);

SELECT EXISTS (
    SELECT id FROM migrations WHERE id = :'MIGRATION_ID'
) as migrated \gset

\if :migrated
    \echo 'migration' :MIGRATION_ID 'already␣exists,␣skipping'
\else
    \echo 'migration' :MIGRATION_ID 'does␣not␣exist'

    CREATE EXTENSION IF NOT EXISTS "pgcrypto";
    CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
    CREATE EXTENSION IF NOT EXISTS "pg_cron";

    -- shorkey used for prettier urls
    CREATE DOMAIN SHORTKEY as varchar(11);

    CREATE OR REPLACE FUNCTION shortkey_generate()
    RETURNS TRIGGER AS $$
    DECLARE
        gkey TEXT;
        key SHORTKEY;
        qry TEXT;
        found TEXT;
        user_id BOOLEAN;
    BEGIN
        -- generate the first part of a query as a string with
           safely
        -- escaped table name, using || to concat the parts
        qry := 'SELECT␣id␣FROM␣' || quote_ident(TG_TABLE_NAME) ||
            '␣WHERE␣id=';
```

Листинг А.2 – Первичная инициализация базы данных

```
        LOOP
            -- deal with user-supplied keys, they don't have to
               be valid base64
            -- only the right length for the type
            IF NEW.id IS NOT NULL THEN
                key := NEW.id;
                user_id := TRUE;

                IF length(key) <> 11 THEN
                    RAISE 'User␣defined␣key␣value␣%␣has␣invalid␣
                       length.␣Expected␣11,␣got␣%.', key, length(
                       key);
                END IF;
            ELSE
                -- 8 bytes gives a collision p = .5 after 5.1 x
                   10^9 values
                gkey := encode(gen_random_bytes(8), 'base64');
                gkey := replace(gkey, '/', '_');  -- url safe
                   replacement
                gkey := replace(gkey, '+', '-');  -- url safe
                   replacement
                key := rtrim(gkey, '=');          -- cut off
                   padding
                user_id := FALSE;
            END IF;

            -- Concat the generated key (safely quoted) with the
               generated query
            -- and run it.
            -- SELECT id FROM "test" WHERE id='blahblah' INTO
               found
            -- Now "found" will be the duplicated id or NULL.
            EXECUTE qry || quote_literal(key) INTO found;

            -- Check to see if found is NULL.
            -- If we checked to see if found = NULL it would
               always be FALSE
            -- because (NULL = NULL) is always FALSE.
            IF found IS NULL THEN
                -- If we didn't find a collision then leave the
                   LOOP.
                EXIT;
            END IF;

            IF user_id THEN
                -- User supplied ID but it violates the PK unique
                    constraint
                RAISE 'ID␣%␣already␣exists␣in␣table␣%', key,
                   TG_TABLE_NAME;
            END IF;
```

Листинг А.3 – Первичная инициализация базы данных

```
            -- We haven't EXITed yet, so return to the top of the
                LOOP
            -- and try again.
        END LOOP;

        -- NEW and OLD are available in TRIGGER PROCEDURES.
        -- NEW is the mutated row that will actually be INSERTed.
        -- We're replacing id, regardless of what it was before
        -- with our key variable.
        NEW.id = key;

        RETURN NEW;
    END
    $$ language 'plpgsql';

    -- represents only user id
    CREATE TABLE users (
        id SHORTKEY PRIMARY KEY,
        created_at TIMESTAMP NOT NULL DEFAULT NOW(),
        deleted_at TIMESTAMP
    );

    -- generate shortkey for each insert
    CREATE TRIGGER trgr_users_genid
        BEFORE INSERT ON users FOR EACH ROW
        EXECUTE PROCEDURE shortkey_generate();

    CREATE INDEX idx_user_indices_deleted_at ON users(deleted_at)
       ;

    CREATE TYPE role_type AS ENUM (
        'MANAGER',
        'ADMIN'
    );

    CREATE TABLE roles (
        id SERIAL PRIMARY KEY,
        type role_type NOT NULL,
        user_id SHORTKEY NOT NULL,
        issuer_id SHORTKEY,
        created_at TIMESTAMP NOT NULL DEFAULT NOW(),
        deleted_at TIMESTAMP,
        CONSTRAINT fk_user FOREIGN KEY (user_id) REFERENCES users
           (id) ON DELETE CASCADE,
        CONSTRAINT fk_issuer FOREIGN KEY (issuer_id) REFERENCES
           users(id) ON DELETE SET NULL
    );
```

Листинг А.4 – Первичная инициализация базы данных

```
    CREATE INDEX indx_role_indices_pk ON roles(type, user_id,
       deleted_at);

    CREATE TYPE user_form_state AS ENUM (
        'CREATED',
        'MODERATING',
        'APPROVED',
        'DECLAINED'
    );

    CREATE TYPE currency AS ENUM (
        'RUB',
        'EUR',
        'USD'
    );

    -- represents user data
    CREATE TABLE user_forms (
        id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
        user_id SHORTKEY NOT NULL,
        state user_form_state NOT NULL DEFAULT 'CREATED',
        name VARCHAR,
        password VARCHAR,
        phone VARCHAR,
        email VARCHAR,
        currency currency,
        declain_reason TEXT,
        created_at TIMESTAMP NOT NULL DEFAULT NOW(),
        updated_at TIMESTAMP NOT NULL DEFAULT NOW(),
        deleted_at TIMESTAMP,
        CONSTRAINT fk_user FOREIGN KEY (user_id) REFERENCES users
           (id) ON DELETE CASCADE
    );

    CREATE INDEX idx_user_form_indices_deleted_at ON user_forms(
       deleted_at);

    -- remove expired guests record
    SELECT cron.schedule('*/5 * * * *', $$
        DELETE FROM users
        WHERE
            created_at < NOW() - interval '4 hour'
            AND id NOT IN (
                SELECT user_id
                FROM user_forms
            )
    $$);
```

Листинг А.5 – Первичная инициализация базы данных

```
CREATE TYPE token_action AS ENUM (
  'SET_USER_EMAIL',
  'SET_USER_PHONE',
  'MODERATE_USER_FORM'
);

-- tokens are sent to clients by other means
-- they must be activated to perform some actions
CREATE TABLE tokens (
    id DECIMAL(6, 0),
    user_id SHORTKEY NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    activated_at TIMESTAMP,
    expires_at TIMESTAMP NOT NULL,
    action token_action NOT NULL,
    data JSONB,
    CONSTRAINT fk_user FOREIGN KEY (user_id) REFERENCES users
       (id) ON DELETE CASCADE,
    PRIMARY KEY (id, user_id)
);

-- delete expired and not activated tokens
SELECT cron.schedule('*/5 * * * *', $$DELETE FROM tokens
   WHERE expires_at < NOW() AND activated_at IS NULL$$);

CREATE OR REPLACE FUNCTION token_generate()
RETURNS TRIGGER AS $$
DECLARE
    key DECIMAL(6, 0);
    qry TEXT;
    found DECIMAL(6, 0);
BEGIN
    -- tokens must be generated by this function
    -- id = 0 is also means empty
    IF NEW.id IS NOT NULL AND NEW.id <> 0 THEN
        RAISE 'Tokens id must be generated by database';
    END IF;

    IF NEW.user_id IS NULL THEN
        RAISE 'user_id must be provided';
    END IF;

    -- query to check if this token already exists
    qry := 'SELECT id, user_id FROM tokens WHERE user_id = '
       || quote_literal(NEW.user_id) || ' AND id =';

    LOOP
        key := floor(random() * 999999 + 1);
```

Листинг А.6 – Первичная инициализация базы данных

```
            EXECUTE qry || key::TEXT INTO found;

            IF found IS NULL THEN
                EXIT;
            END IF;
        END LOOP;

        NEW.id = key;

        RETURN NEW;
    END
    $$ LANGUAGE 'plpgsql';

    CREATE TRIGGER trgr_tokens_genid
        BEFORE INSERT ON tokens FOR EACH ROW
        EXECUTE PROCEDURE token_generate();

    CREATE TABLE banks (
        id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
        name VARCHAR NOT NULL,
        bic VARCHAR NOT NULL,
        correspondent_account VARCHAR NOT NULL,
        inn VARCHAR NOT NULL,
        kpp VARCHAR NOT NULL,
        created_at TIMESTAMP NOT NULL DEFAULT NOW(),
        updated_at TIMESTAMP NOT NULL DEFAULT NOW(),
        deleted_at TIMESTAMP
    );

    CREATE INDEX idx_bank_indices_deleted_at ON banks(deleted_at)
       ;

    CREATE TABLE nominal_accounts (
        ID UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
        name VARCHAR UNIQUE NOT NULL,
        receiver VARCHAR NOT NULL,
        account_number VARCHAR NOT NULL,
        created_at TIMESTAMP NOT NULL DEFAULT NOW(),
        updated_at TIMESTAMP NOT NULL DEFAULT NOW(),
        deleted_at TIMESTAMP,
        bank_id UUID NOT NULL,
        CONSTRAINT fk_bank FOREIGN KEY (bank_id) REFERENCES banks
           (id)
    );

    CREATE INDEX idx_nominal_account_indices_deleted_at ON
       nominal_accounts(deleted_at);
```

Листинг А.7 – Первичная инициализация базы данных

```
    CREATE TABLE accounts (
        id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
        number VARCHAR NOT NULL,
        user_id SHORTKEY NOT NULL,
        nominal_account_id UUID NOT NULL,
        created_at TIMESTAMP NOT NULL DEFAULT NOW(),
        deleted_at TIMESTAMP,
        CONSTRAINT fk_user FOREIGN KEY (user_id) REFERENCES users
           (id),
        CONSTRAINT fk_nominal_account FOREIGN KEY (
           nominal_account_id) REFERENCES nominal_accounts(id)
    );

    CREATE INDEX idx_account_indices_deleted_at ON accounts(
       deleted_at);

    CREATE TYPE product_state AS ENUM (
      'CREATED',
      'MODERATING',
      'APPROVED',
      'DECLAINED'
    );

    CREATE TABLE products (
        id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
        state product_state NOT NULL DEFAULT 'CREATED',
        title VARCHAR NOT NULL,
        description VARCHAR NOT NULL,
        creator_id SHORTKEY NOT NULL,
        created_at TIMESTAMP NOT NULL DEFAULT NOW(),
        updated_at TIMESTAMP NOT NULL DEFAULT NOW(),
        deleted_at TIMESTAMP,
        CONSTRAINT fk_creator FOREIGN KEY (creator_id) REFERENCES
            users(id)
    );

    CREATE INDEX idx_product_indices_deleted_at ON products(
       deleted_at);

    CREATE TYPE auction_state AS ENUM (
      'CREATED',
      'STARTED',
      'FINISHED',
      'FAILED',
      'SUCCEEDED'
    );

    CREATE TABLE auctions (
        id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
        state auction_state NOT NULL DEFAULT 'CREATED',
```

Листинг А.8 – Первичная инициализация базы данных

```
        product_id UUID NOT NULL,
        seller_id SHORTKEY NOT NULL,
        buyer_id SHORTKEY,
        min_amount DECIMAL(12, 2),
        "currency" currency,
        scheduled_start_at TIMESTAMP,
        scheduled_finish_at TIMESTAMP,
        started_at TIMESTAMP,
        finished_at TIMESTAMP,
        fail_reason TEXT,
        created_at TIMESTAMP NOT NULL DEFAULT NOW(),
        updated_at TIMESTAMP NOT NULL DEFAULT NOW(),
        CONSTRAINT fk_product FOREIGN KEY (product_id) REFERENCES
            products(id),
        CONSTRAINT fk_seller FOREIGN KEY (seller_id) REFERENCES
           users(id),
        CONSTRAINT fk_buyer FOREIGN KEY (buyer_id) REFERENCES
           users(id),
        CONSTRAINT chk_currency CHECK (
            state = 'CREATED' OR "currency" IS NOT NULL
        ),
        CONSTRAINT chk_min_amount CHECK (
            min_amount IS NULL OR "currency" IS NOT NULL
        ),
        CONSTRAINT chk_started_at CHECK (
            state = 'CREATED' OR started_at IS NOT NULL
        ),
        CONSTRAINT chk_finished_at CHECK (
            state = 'CREATED' OR state = 'STARTED' OR finished_at
                IS NOT NULL
        ),
        CONSTRAINT chk_buyer CHECK (
            NOT (state = 'SUCCEEDED' AND buyer_id IS NULL)
        )
    );

    CREATE TABLE product_images (
        id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
        filename VARCHAR NOT NULL,
        path VARCHAR NOT NULL,
        product_id UUID NOT NULL,
        CONSTRAINT fk_product FOREIGN KEY (product_id) REFERENCES
            products(id) ON DELETE CASCADE
    );

    INSERT INTO migrations(id) VALUES (:'MIGRATION_ID');
\endif

COMMIT;
```

ПРИЛОЖЕНИЕ Б
Настройка ролей базы данных

Листинг Б.1 – Настройка ролей базы данных

```
--
-- Create database roles for server, bankgate and viewer
--
SELECT EXISTS (
    SELECT id FROM migrations WHERE id = :'MIGRATION_ID'
) as migrated \gset

\if :migrated
    \echo 'migration' :MIGRATION_ID 'already␣exists,␣skipping'
\else
    \echo 'migration' :MIGRATION_ID 'does␣not␣exist'

    \set SERVER_USER `echo $SERVER_USER`
    \set SERVER_PASSWORD `echo $SERVER_PASSWORD`
    \set BANKGATE_USER `echo $BANKGATE_USER`
    \set BANKGATE_PASSWORD `echo $BANKGATE_PASSWORD`
    \set VIEWER_USER `echo $VIEWER_USER`
    \set VIEWER_PASSWORD `echo $VIEWER_PASSWORD`
    \set DEALER_USER `echo $DEALER_USER`
    \set DEALER_PASSWORD `echo $DEALER_PASSWORD`

    \set exit_error false

    SELECT (:'SERVER_USER' = '') as is_not_empty \gset
    \if :is_not_empty
        \warn 'SERVER_USER␣is␣empty'
        \set exit_error true
    \endif

    SELECT (:'SERVER_PASSWORD' = '') as is_not_empty \gset
    \if :is_not_empty
        \warn 'SERVER_PASSWORD␣is␣empty'
        \set exit_error true
    \endif

    SELECT (:'BANKGATE_USER' = '') as is_not_empty \gset
    \if :is_not_empty
        \warn 'BANKGATE_USER␣is␣empty'
        \set exit_error true
    \endif
```

Листинг Б.2 – Настройка ролей базы данных

```
SELECT (:'BANKGATE_PASSWORD' = '') as is_not_empty \gset
\if :is_not_empty
    \warn 'BANKGATE_PASSWORD␣is␣empty'
    \set exit_error true
\endif

SELECT (:'VIEWER_USER' = '') as is_not_empty \gset
\if :is_not_empty
    \warn 'VIEWER_USER␣is␣empty'
    \set exit_error true
\endif

SELECT (:'VIEWER_PASSWORD' = '') as is_not_empty \gset
\if :is_not_empty
    \warn 'VIEWER_PASSWORD␣is␣empty'
    \set exit_error true
\endif

SELECT (:'DEALER_USER' = '') as is_not_empty \gset
\if :is_not_empty
    \warn 'DEALER_USER␣is␣empty'
    \set exit_error true
\endif

SELECT (:'DEALER_PASSWORD' = '') as is_not_empty \gset
\if :is_not_empty
    \warn 'DEALER_PASSWORD␣is␣empty'
    \set exit_error true
\endif

\if :exit_error
    DO $$
    BEGIN
    RAISE EXCEPTION 'all␣required␣environment␣variables␣must␣
       not␣be␣empty';
    END;
    $$;
\endif

CREATE ROLE :SERVER_USER
    WITH LOGIN PASSWORD :'SERVER_PASSWORD';
GRANT SELECT, INSERT, UPDATE
    ON ALL TABLES IN SCHEMA public
    TO :SERVER_USER;
```

Листинг Б.3 – Настройка ролей базы данных

```
    CREATE ROLE :VIEWER_USER
        WITH LOGIN PASSWORD :'VIEWER_PASSWORD';
    GRANT SELECT
        ON ALL TABLES IN SCHEMA public
        TO :VIEWER_USER;

    CREATE ROLE :BANKGATE_USER
        WITH LOGIN PASSWORD :'BANKGATE_PASSWORD';

    CREATE ROLE :DEALER_USER
        WITH LOGIN PASSWORD :'DEALER_PASSWORD';
    GRANT SELECT, UPDATE
        ON auctions
        TO :DEALER_USER;

    INSERT INTO migrations(id) VALUES (:'MIGRATION_ID');
\endif

COMMIT;
```

ПРИЛОЖЕНИЕ В
Транзакция добавления предложения

Листинг В.1 – Транзакция добавления предложения

```
-- Текущий пользователь
SELECT * FROM "users"
WHERE id = 'zTUZBNT-9b0' AND "users"."deleted_at" IS NULL;
-- Аукцион, в котором планируется сделать ставку
SELECT * FROM "auctions"
WHERE id = 'a75d4700-7058-4749-b1e4-8aef840103fc';
-- Счет, с которого будет списана сумма
SELECT * FROM "accounts"
WHERE id = 'a5c4d021-67b8-4bf5-8a22-8869523c9ab9'
AND "accounts"."deleted_at" IS NULL;

BEGIN TRANSACTION;
-- Блокировка аукциона
SELECT * FROM "auctions"
WHERE id = 'a75d4700-7058-4749-b1e4-8aef840103fc'
FOR SHARE OF "auctions";
-- Блокировка счета
SELECT * FROM "accounts"
WHERE id = 'a5c4d021-67b8-4bf5-8a22-8869523c9ab9'
AND "accounts"."deleted_at" IS NULL
FOR UPDATE OF "accounts";
-- Отмена прошлого предложения, если оно есть
SELECT * FROM "offers"
WHERE auction_id IN ('a75d4700-7058-4749-b1e4-8aef840103fc')
AND state IN ('CREATED')
AND user_id IN ('zTUZBNT-9b0');
-- Здесь его не оказалось, поэтому отмены нет.
-- Получение баланса
SELECT trs.currency, trs.account_id, SUM(trs.amount) as amount
FROM (
  SELECT currency, account_from_id as account_id, -amount as
     amount
  FROM "transactions"
  JOIN (
      SELECT *
      FROM "accounts"
      WHERE id = 'a5c4d021-67b8-4bf5-8a22-8869523c9ab9'
      AND "accounts"."deleted_at" IS NULL
  ) a ON account_from_id = a.id
    AND (
      transactions.state IN (
        'SUCCEEDED', 'PROCESSING', 'ERROR'
      )
```

Листинг B.2 – Транзакция добавления предложения

```
      OR (
        transactions.type IN ('BUY')
        AND transactions.state IN ('CREATED')
      )
    )
  WHERE "transactions"."deleted_at" IS NULL
  UNION ALL
  SELECT currency, account_to_id as account_id, amount
  FROM "transactions"
  JOIN (
    SELECT *
    FROM "accounts"
    WHERE id = 'a5c4d021-67b8-4bf5-8a22-8869523c9ab9'
    AND "accounts"."deleted_at" IS NULL
  ) a ON account_to_id = a.id
  AND transactions.state IN (
    'SUCCEEDED', 'PROCESSING', 'ERROR'
  )
  WHERE "transactions"."deleted_at" IS NULL
) trs
GROUP BY trs.currency, trs.account_id

-- Создание предложения
INSERT INTO "offers" (
  "state", "auction_id", "user_id",
  "created_at", "updated_at"
)
VALUES (
  'CREATED', 'a75d4700-7058-4749-b1e4-8aef840103fc',
  'zTUZBNT-9b0', '2022-06-18␣09:56:16.263',
  '2022-06-18␣09:56:16.263'
)
RETURNING "id";
-- Создание транзакция предложения
INSERT INTO "transactions" (
  "date", "state", "type", "currency",
  "amount", "error", "account_from_id",
  "account_to_id", "offer_id", "created_at",
  "updated_at", "deleted_at"
)
VALUES (
  NULL, 'CREATED', 'BUY', 'RUB', '10',
  NULL, 'a5c4d021-67b8-4bf5-8a22-8869523c9ab9',
  '5d0c6a55-23cd-4e70-b342-5707578a92cb',
  '31e167d9-4341-4b47-8922-e8b5281fbf21',
  '2022-06-18␣09:56:16.268', '2022-06-18␣09:56:16.268',
  NULL
)
RETURNING "id";
COMMIT;
```