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:

«Реализация базы данных для системы обработки заявок пользователей в сфере жилищного хозяйства»

Студент ИУ7-64Б _____ С. Д. Параскун
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ Ю. М. Гаврилова
(Подпись, дата) (И.О.Фамилия)

2022г.

Реферат

Отчет содержит 41 страницу, 17 рисунков, 2 таблицы, 16 источников, 1 приложение.

Ключевые слова: базы данных, PostgreSQL, telegram-бот, реляционная модель, Java, Spring Boot, IntelliJ IDEA.

Объектом исследования является модель представления данных в системе сбора и обработки заявок пользователей.

Целью данной работы является проектирование и разработка базы данных для сбора и обработки заявок пользователей, интерфейсом для которой станет telegram-бот.

Результатом выполнения работы стали достигнутая цель и решенные задачи:

- формализована задача и определен необходимый функционал;
- описана структура объектов БД, а также сделан выбор СУБД для ее хранения и взаимодействия;
- создана БД и каждую ее сущность заполнена не менее 1000 записей;
- разработан алгоритм проверки наличия бан-слов в заголовке или описании при создании заявки;
- спроектировано и реализовано приложение в формате telegram-бота для сбора и обработки заявок, которое будет взаимодействовать с описанной базой данных;
- проведено исследование времени обработки операции добавления заявок в зависимости от наличия триггера и объема данных в таблице с бан-словами.

Содержание

Введение

В повседневной жизни люди нередко сталкиваются с ситуациями, в которых помощь окружающих может значительно облегчить решение какой-либо проблемы. При этом речь может идти как об обычном наличии предметов или продуктов, так и о профессиональной помощи в экстренной ситуации. Для этого нужно обмениваться информацией о своих потребностях или возможностях с другими людьми, самые территориально близкие из которых – соседи.

Можно пытаться писать каждому из них со своей просьбой, но гораздо проще разместить ее в специально созданном для этого сервисе. Так как в нынешнее время такой мессенджер как telegram [1] стремительно набирает популярность среди населения, целесообразным будет ориентироваться именно на эту площадку.

Целью курсовой работы является проектирование и разработка базы данных для сбора и обработки заявок пользователей, интерфейсом для которой станет telegram-бот.

Для достижения поставленной цели, необходимо решить следующие задачи:

- формализовать задачу и определить необходимый функционал;
- описать структуру объектов БД, а также сделать выбор СУБД для ее хранения и взаимодействия;
- создать БД и заполнить каждую ее сущность не менее 1000 записей;
- разработать алгоритм проверки наличия бан-слов в заголовке или описании при создании заявки;
- спроектировать и реализовать приложение в формате telegram-бота для сбора и обработки заявок, которое будет взаимодействовать с описанной базой данных;
- провести исследование времени обработки операции добавления заявок в зависимости от наличия триггера и объема данных в таблице с бан-словами.

Итогом работы станет база данных с разработанным интерфейсом взаимодействия с использованием различных ролевых моделей. Также должно быть предусмотрено наличие триггера на добавление новых заявок в систему.

1 Аналитический раздел

В данном разделе будут выдвинуты требования к приложению, определены пользователи системы, формализованы хранимые о системе данные, а также проведен анализ существующих решений и выбор модели базы данных.

1.1 Требования к приложению

Приложение должно поддерживать определенный функционал:

- персонализация пользователей;
- создание новых заявок и изменение информации об уже существующих;
- добавление оборудования;
- добавление навыков и отправка их на подтверждение;
- пополнение списка ролей конкретного пользователя и переключение между ними.

1.2 Пользователи системы

Для работы с системой обязательным этапом является прохождение персонализации. Пользователь может работать в системе под одной из следующих ролей.

1. Оформитель – пользователь, обладающий возможностями создания заявки, изменения ее параметров, закрытия заявки.
2. Исполнитель – пользователь, обладающий возможностями отправления своих навыков на подтверждение, взятия заявок на исполнение и их последующего закрытия.
3. Администратор – пользователь, обладающий возможностью подтверждения роли других администраторов и профессиональных навыков исполнителей.

В ходе использования приложения предусмотрена возможность смены ролей. На рисунке 1.1 представлена диаграмма использования приложения.

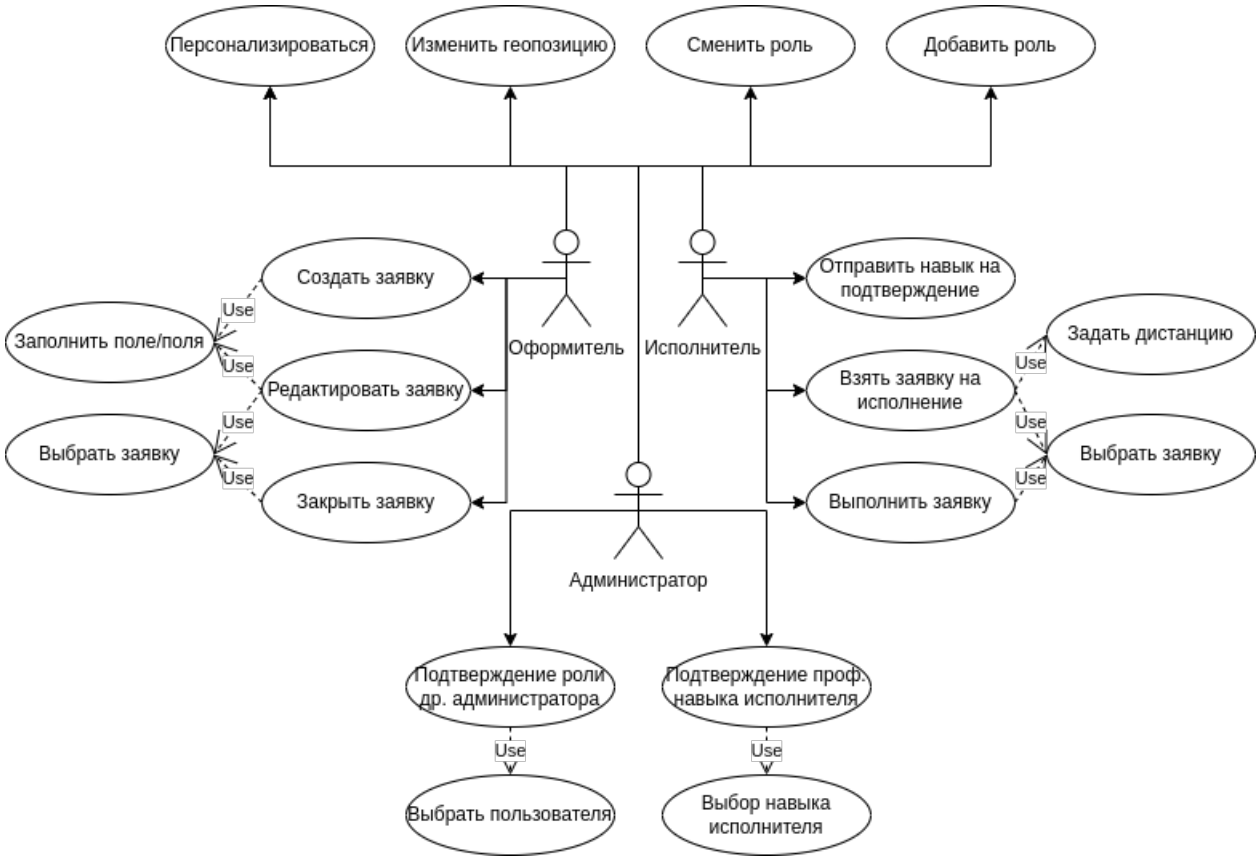

Рисунок 1.1 – Use-case диаграмма

1.3 Формализация данных

База данных состоит из нескольких таблиц:

- таблица пользователей User;
- таблица подключений Connection;
- таблица заявок Request;
- таблица оборудования Equipment;
- таблица навыков Skill;
- таблица подтверждения навыков исполнителей ExecutorSkill;

- таблица, хранящая набор данных для предотвращения использования ненормативной лексики Ban.

На рисунке 1.2 представлена ER-диаграмма сущностей в нотации Чена.

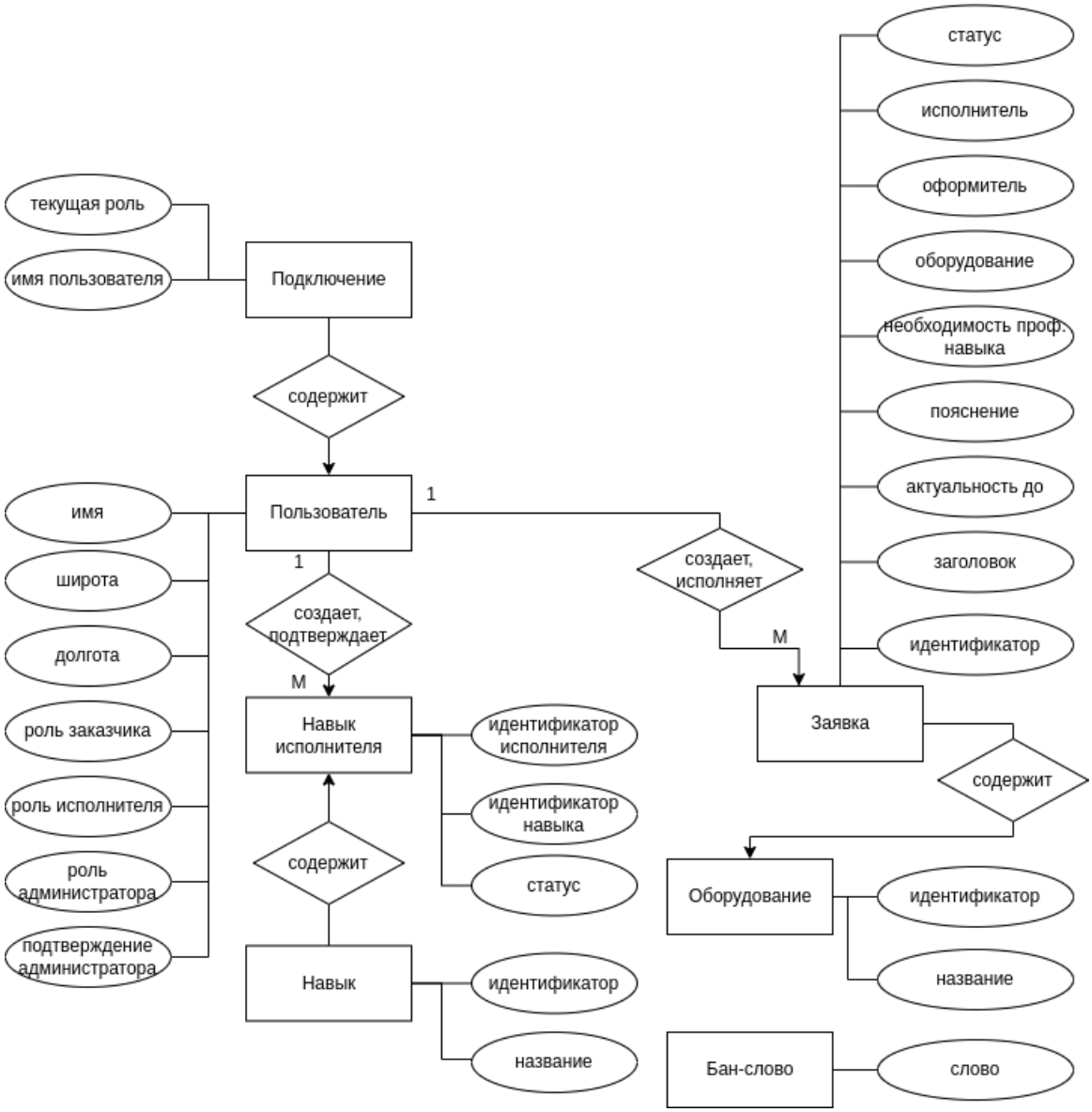

Рисунок 1.2 – ER-диаграмма в нотации Чена

1.4 Анализ существующих решений

Среди уже имеющихся проектов, решающих поставленную задачу, были выделены 3 аналога (таблица 1.1) – сервисы «YouDo» [2], «Все соседи» [3] и «Привет, сосед!» [4]. Сравнение проводилось по ряду критериев, а именно

авторизация, наличие территориальной привязки, охват аудитории, а также были выделены преимущественные особенности системы.

Таблица 1.1 – Существующие решения поставленной задачи

Название проекта	**Авторизация**	**Наличие территориальной привязки**	**Охват аудитории**	**Особенности системы**
YouDo	Электронная почта или Вконтакте, Одноклассники, Google, Mail, Apple	На уровне города	Более 2.5 миллионов пользователей	Заказо-ориентированность со стабильной системой отзывов
Все соседи	Номер телефона	Устанавливается с точностью до дома	Около 60 тыс. пользователей	Бартерная система взаимовыручки
Привет, сосед!	Электронная почти или Вконтакте, Facebook, Одноклассники, Google, Yandex, Apple	Устанавливается с точностью до дома	Около 2 тыс. пользователей	Уклон в социальную сеть с бонусной программой

Стоит упомянуть, что все вышеуказанные решения имеют мобильные версии для IOS и Android, а также Web-версии приложений.

У каждого вышеупомянутого аналога имеются недостатки среди выделенных критериев, как например слабоконкретизированная геопозиция, неудобная авторизация или малое количество участников.

Создаваемый продукт должен не просто быть на уровне с конкурирующими платформами, но и предоставлять пользователям дополнительный функционал. Такими особенностями станут:

- территориальная привязка на уровне геолокации;
- авторизация внутри мессенджера telegram, имеющего больший охват аудитории и возможность лично написать оформителю/исполнителю заявки;

- смена роли – пользователь сможет не только создавать заявки, но и помогать другим, исполняя их, в рамках одного аккаунта;
- добавление навыков и оборудования – для некоторых задач простого желания помочь недостаточно, а данный функционал поможет облегчить процесс выбора заявок для исполнения.;
- механизм предотвращения ненормативной лексики при создании заявок.

Анализ моделей баз данных

Модель данных – это совокупность правил порождения структур данных в базе данных, операций над ними, а также ограничений целостности, определяющих допустимые связи и значения данных, последовательность их изменения [5].

В настоящее время разработано множество моделей данных, рассмотрим основные из них.

1.4.1 Иерархическая модель данных

Иерархическая модель данных подразумевает что элементы, организованные в структуры, объединены иерархической или древовидной связью. В таком представлении родительский элемент может иметь несколько дочерних, а дочерний – только один родительский.

Каждая вершина дерева соответствует типу сущности ПО. Конечные вершины, то есть вершины, из которых не выходит ни одной дуги, называются листьями дерева. Каждая некорневая вершина связана с родительской вершиной иерархическим групповым отношением. Тип сущности характеризуется произвольным количеством атрибутов, связанных с ней отношением 1:1. Атрибуты, связанные с сущностью отношением 1:n, образуют отдельную сущность (сегмент) и переносятся на следующий уровень иерархии. Реализация связей типа n:m не поддерживается.

В иерархической модели не может быть сегментов, не связанных с вершинами верхнего уровня (кроме корневой). Добраться до нужной записи можно только пройдя по всему дереву сущностей и по записям текущего сегмента, так как связи между записями обычно выполнены в виде ссылок. Каждая из записей идентифицируется по полному сцепленному ключу, который в свою очередь является конкатенированным значением идентификаторов сегментов, через которые осуществляется доступ, и идентификатором самой записи.

Существенным недостатком такой модели является то, что каждая сущность может относится только к одному родительскому сегменту. Таким образом при необходимости множественного отношения происходит дублирование данных, что может привести к нарушению логической целостности БД.

1.4.2 Сетевая модель данных

Сетевая модель базы данных подразумевает, что у родительского элемента может быть несколько потомков, а у дочернего элемента – несколько предков. Структура такой модели представлена в виде графа, причем каждая вершина графа хранит экземпляры сущностей (записи одного типа) и сведения о групповых отношениях с сущностями других типов. Каждая запись может хранить произвольное количество значений атрибутов (элементов данных и агрегатов), характеризующих экземпляр сущности. Для каждого типа записи выделяется первичный ключ – атрибут, значение которого позволяет однозначно идентифицировать запись среди экземпляров записей данного типа.

Связи между записями выполняются в виде указателей, т.е. каждая запись хранит ссылку на другую однотипную запись (или признак конца списка) и ссылки на списки подчинённых записей, связанных с ней групповыми отношениями. Таким образом, в каждой вершине записи хранятся в виде связного списка.

В этой модели связь 1:n между сущностями реализуется с помощью групповых отношений, а связи 1:n между атрибутами сущности – в рамках записи. Для реализации связей типа n:m вводится вспомогательный тип записи и две связи 1:n.

Физическая независимость не обеспечивается в сетевой модели данных, потому что наборы организованы с помощью физических ссылок. Помимо этого данная модель не обеспечивает независимость данных от программ. Эти недостатки помешали обрести ей широкое распространение из-за сложного проектирования и поддержки.

1.4.3 Реляционная модель данных

Реляционная модель данных является самой распространенной в использовании. В отличие от вышеописанных, в данной модели не существует физических отношений между сущностями. Хранение информации осуществляется в виде таблиц (отношений), состоящих из рядов и столбцов. Отношение имеет имя, которое отличает его от имён всех других отношений. Атрибутам реляционного отношения назначаются имена, уникальные в рамках отношения. Обращение к отношению происходит по его имени, а к атрибуту – по именам отношения и атрибута.

В реляционных моделях данных нет необходимости просматривать все указатели, что облегчает выполнение запросов на выборку информации по сравнению с сетевыми и иерархическими моделями. Это означает, что порядок записей не является важным, также как и порядок столбцов.

Каждая запись идентифицируется уникальной комбинацией атрибутов – первичным ключом. Для связи между отношениями используется внешний ключ – копия первичного или уникального ключа атрибута сторонней сущности. Таким образом можно реализовывать связь 1:n. Для обеспечения связи n:m вводится дополнительная сущность, атрибутами которой являются внешние ключи соответствующих сущностей.

Обработка данных осуществляется с помощью декларативного языка запросов SQL [6].

1.5 Выбор модели базы данных

В данном проекте будет использоваться реляционная модель данных, потому что в рамках проекта она обладает следующими преимуществами:

- изложение информации осуществляется с помощью простых и понятных форм (таблиц);
- позволяет работать со структурированными данными, структура которых не подвергается частым изменениям;
- имеет возможность произвольного доступа к записям сущностей;
- исключает дублирование, реализуя связь между отношениями посредством внешнего ключа.

1.6 Вывод из раздела

В данном разделе были выделены ролевые модели системы, конкретизированы хранимые данные и их связь между собой, построены соответствующие диаграммы. Также был проведен анализ существующих на рынке решений, который позволил понять, какие особенности стоит добавить в разрабатываемый проект. Был осуществлен выбор модели базы данных.

2 Конструкторский раздел

В данном разделе будут представлены этапы проектирования базы данных, выделены конкретные действия ролевой модели, спроектирован триггер и описаны основы проектирования приложения.

2.1 Формализация сущностей системы

На основе выделенных ранее сущностей спроектированы следующие таблицы базы данных.

1. Таблица User – содержит информацию о пользователях системы. Включает следующие поля:
   - tgName – имя пользователя в telegram, символьный тип, является идентификатором;
   - geoLat, geoLong – местоположение пользователя, вещественный тип;
   - customerRole, executorRole, administratorRole – наличие у пользователя описываемой роли, логический тип;
   - adminVerified – статус подтверждения роли администратора у пользователя, логический тип.
2. Таблица Connection – содержит информацию о подключениях пользователей под определенной ролью. Включает следующие поля:
   - userTgName – идентификатор пользователя, целочисленный тип;
   - currentRole – текущая роль, символьный тип.

   Совокупность этих полей является идентификатором подключения.
3. Таблица Request – содержит информацию о заявках системы. Включает следующие поля:
   - id – идентификатор заявки, целочисленный тип;
   - title – название заявки, символьный тип;
   - periodOfRelevance – срок релевантности заявки, временная метка;

- explanation – пояснение к заявке, символьный тип;
- profNecessity – необходимый проф. навык, идентификатор навыка, целочисленный тип;
- equipment – необходимое оборудование, идентификатор оборудования, целочисленный тип;
- owner, executor – оформитель и исполнитель, идентификатор пользователя, символьный тип;
- status – статус заявки, символьный тип.

4. Таблица Equipment – содержит информацию об используемом оборудовании. Включает следующие поля:
   - id – идентификатор оборудования, целочисленный тип;
   - name – название оборудования, символьный тип.
5. Таблица Skill – содержит информацию о навыках в системе. Включает следующие поля:
   - id – идентификатор навыка, целочисленный тип;
   - name – название навыка, символьный тип.
6. Таблица ExecutorSkill – содержит информацию о навыках исполнителей и статусе их подтверждения. Включает следующие поля:
   - executorId – идентификатор пользователя (исполнителя), символьный тип;
   - skillId – идентификатор навыка, целочисленный тип;
   - status – статус подтвержденния навыка, символьный тип.

   Совокупность executorId и skillId является идентификатором данной таблицы.
7. Таблица Ban – содержит части слов, по которым можно определить ненормативную лексику. Состоит из единственного поля word – символьный тип, является идентификатором.

Соответствующая диаграмма по описанным выше данным представлена на рисунке 2.1.

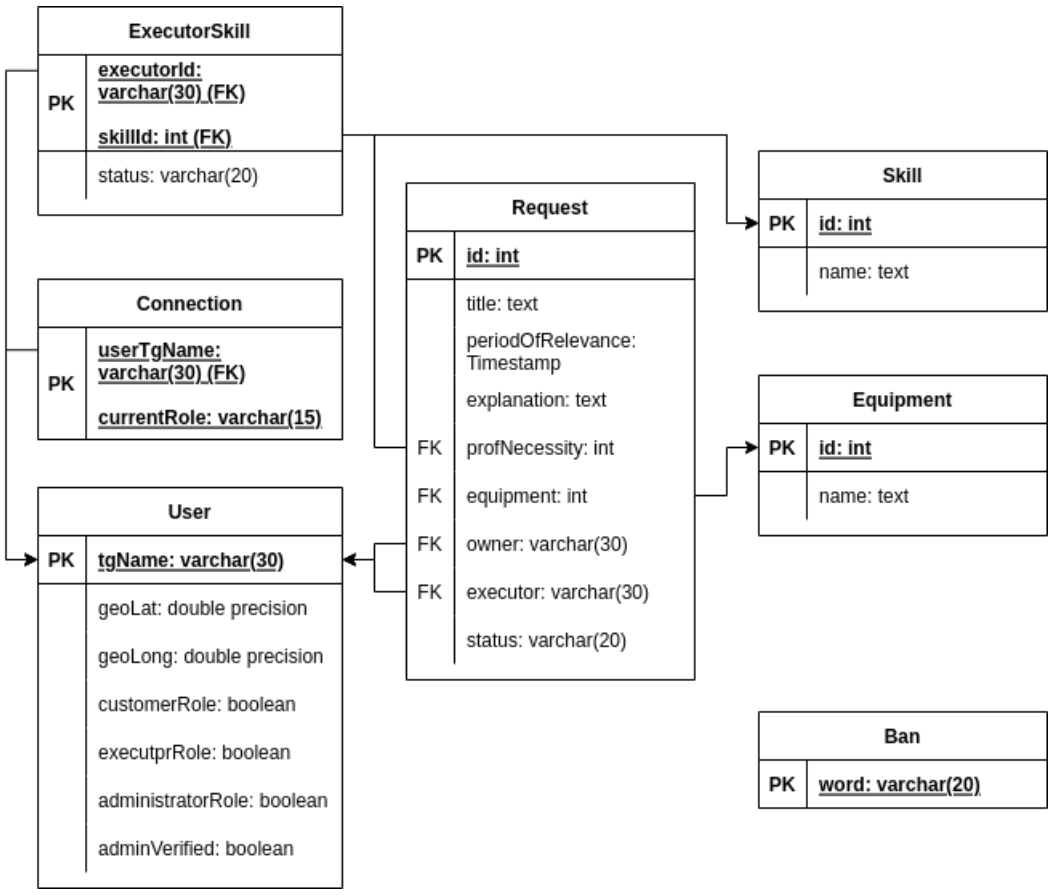

Рисунок 2.1 – ER-диаграмма системы

2.2 Ролевая модель

Для обеспечения корректной работы пользователей с системой на уровне базы данных выделена следующая ролевая модель.

1. Customer – оформитель заявки. Обладает правами:
   - INSERT/UPDATE над таблицами User, Connection;
   - все права над таблицей Request;
   - INSERT над таблицами Skill, Equipment.
2. Executor – исполнитель заявки. Обладает правами:
   - INSERT/UPDATE над таблицами User, Connection;
   - SELECT/UPDATE над таблицей Request;
   - INSERT над таблицами Skill, ExecutorSkill.

3. Administrator – администратор. Обладает правами:
    - все права над таблицами User, Connection;
    - все права над таблицей ExecutorSkill;
    - все права над таблицей Ban.

Использование ролевой модели на уровне базы данных гарантирует безопасность доступа к объектам.

2.3 Триггер

В системе предусмотрен механизм предотвращения использования ненормативной лексики. Он реализован вспомогательной таблицей Ban, хранящей части слов для распознавания, а также триггером AFTER на действие INSERT в таблицу Request. Данный триггер проверяет наличие вхождений в заголовок (поле title) и пояснение (поле explanation) частей «запрещенных» слов. Если хотя бы одно такое вхождение найдено, заявка помечается статусом BANNED.

На рисунке 2.2 представлена схема алгоритма работы выполняемой триггером функции checkBanWords().

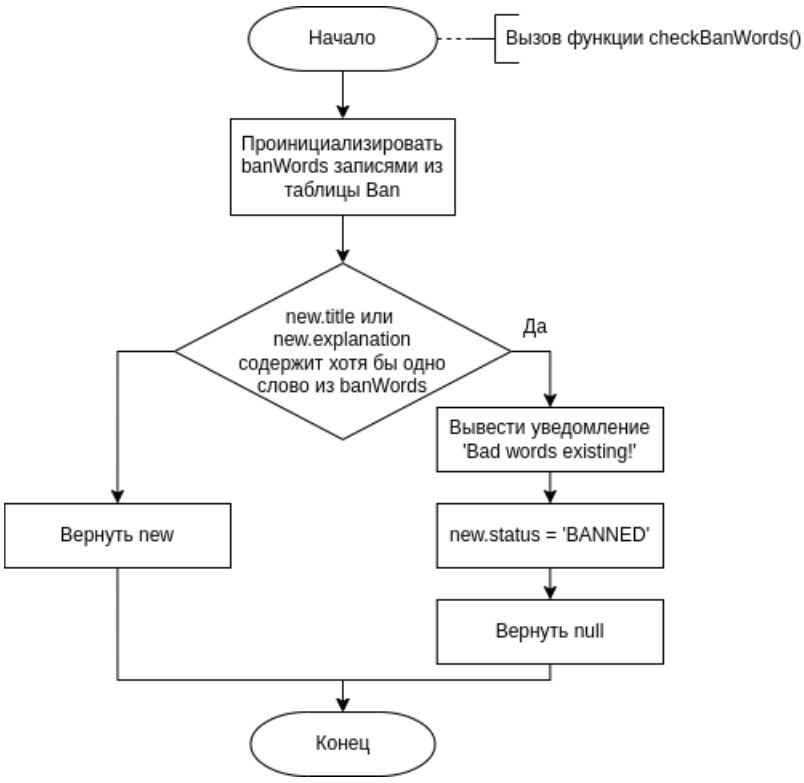

Рисунок 2.2 – Триггер AFTER на создание новой заявки

2.4 Проектирование приложения

Приложение, обеспечивающее взаимодействие с базой данных, представляет собой telegram-бота. Использование имени пользователя в telegram в качестве первичного ключа является целесообразным, так как гарантируется уникальность данного параметра на уровне мессенджера. Разработка будет осуществляется в соответствии с принципами «чистой архитектуры» [7]. Использование такого подхода позволило выделить несколько компонентов: доступ к данным, бизнес-логика, транспортный уровень и UI, реализуемый на основе telegramAPI [8].

2.5 Вывод из раздела

В данном разделе были формализованы сущности системы с последующим выделением ролевой модели, спроектирован триггер AFTER на добавление

новой заявки. Также были описаны основные моменты проектирования приложения.

3 Технологический раздел

В данном разделе будут выбраны СУБД, средства реализации приложения, описано создание базы данных, триггера, ролей с последующим выделением прав, наполнение базы, а также спроектирован пользовательский интерфейс.

3.1 Выбор СУБД

В настоящее время существует множество систем управления базами данных, работающих на основе реляционной модели. Среди самых распространенных [9] выделяют MySQL [10], PostgreSQL [11] и замыкает тройку лидеров SQLite [12]. Рассмотрим особенности каждой из них.

1. MySQL. Среди достоинств данной СУБД можно выделить простой синтаксис, высокую безопасность и масштабируемость, поддержку большей части функционала SQL. Однако, несмотря на перечисленные положительные аспекты, MySQL сильно подвергается DDos-атакам и не сопровождается бесплатной технической поддержкой.

2. PostgreSQL. В рамках использования этой СУБД имеется возможность помимо встроенного SQL использовать различные дополнения, отличается поддержкой форматов csv и json, но оперирует большим объемом ресурсов.

3. SQLite. Очевидными достоинствами является компактность базы данных, которая состоит из одного файла, и легкая переносимость. Но данная СУБД совершенно не подходит для больших БД, а также не поддерживает управление пользователями.

При реализации проекта будет использован PostgreSQL, поскольку эта СУБД обладает достаточным набором инструментов для поставленной задачи.

3.2 Средства реализации

В качестве используемого языка программирования выбрана Java [13], так как:

- данный язык является объектно-ориентированным, позволяя использовать наследование, интерфейсы, абстракции и т. д.;
- имеет JDBC [14] – API для выполнения SQL-запросов к базам данных.

В качестве среды разработки используется IntelliJ IDEA [15], поскольку данная программа:

- имеет удобный интерфейс взаимодействия и подключения используемых зависимостей;
- предоставляет возможности тестирования и запуска написанного приложения с определенными файлами конфигурации.

Создание базы данных

В соответствии с выбранной СУБД и спроектированной базой данных было осуществлено создание БД и ее сущностей. Реализация представлена в приложении А, листинг 4.1.

3.3 Создание триггера

В предыдущем разделе был спроектирован триггер AFTER на создание новой заявки в системе. Код его создания представлен в листинге 3.1

Листинг 3.1 – Реализация триггера AFTER на добавление заявки

```
1 create trigger newRequest
2 after insert
3 on sosedushka_db.request
4 for each row
5 execute function checkBanWords()
```

Для этого триггера была написана соответствующая функция с помощью PL/pgSQL [16] – процедурного расширения языка SQL, используемого в СУБД PostgreSQL. Код функции представлен в листинге 3.2.

Листинг 3.2 – Реализация функции checkBanWords()

```
1  create or replace function checkBanWords()
2  returns trigger as
3  $$
4  declare
5      banWords text ARRAY;
6  begin
7      banWords := ARRAY(select '%' || word || '%' from sosedushka_db.ban);
8      if new.title like any (banWords) or new.explanation like any (banWords)
9      then
10         raise notice 'Bad words existing!';
11         update sosedushka_db.request set status='BANNED' where id=new.id;
12         return null;
13     else
14         return new;
15     end if;
16 end
17 $$
18 language plpgsql
```

3.4 Создание ролей и выделение им прав

В конструкторском разделе была разработана ролевая модель, в которой выделены следующие роли:

- customer – оформитель заявки;
- executor – исполнитель заявки;
- administrator – администратор.

Соответствующий этой ролевой модели сценарий создания ролей и выделения им прав представлен на листинге 3.3.

Листинг 3.3 – Создание ролей и выделение им прав

```
create role customer with login password '1111';
grant insert, update on table sosedushka_db.user to customer;
grant insert, update on table sosedushka_db.connection to customer;
grant all privileges on table sosedushka_db.request to customer;
grant insert on table sosedushka_db.skill to customer;
grant insert on table sosedushka_db.equipment to customer;

create role executor with login password '2222';
grant insert, update on table sosedushka_db.user to executor;
grant insert, update on table sosedushka_db.connection to executor;
grant select, update on table sosedushka_db.request to executor;
grant insert on table sosedushka_db.skill to executor;
grant insert on table sosedushka_db.executorskill to executor;

create role administrator with login password '4813';
grant all privileges on table sosedushka_db.user to administrator;
grant all privileges on table sosedushka_db.connection to administrator;
grant all privileges on table sosedushka_db.executorskill to
administrator;
grant all privileges on table sosedushka_db.ban to administrator;
```

3.5 Наполнение базы данных

Каждая из описанных сущностей (за исключением таблицы Ban) была заполнена 1000 значений. Рассмотрим каким образом осуществлялась генерация данных.

1. Таблица User – все значения получены случайным образом в соответствии с описанными типами данных полей.

2. Таблица Connection – для каждого пользователя из таблицы User текущая роль выбиралась случайным образом из сгенерированных доступных ему ролей.

3. Таблицы Equipment и Skill – значения были получены путем поиска в интернете различных подходящих для таблиц значений.

4. Таблица ExecutorSkill – для каждого случайно выбранного пользователя из таблицы User со значением поля executorRole = True случайным

образом выбирался навык из таблицы Skill и статус навыка из перечня доступных.

5. Таблица Request – заголовок, пояснение и статус выбирались случайно из списка доступных, срок актуальности заполнен одинаковым значением, оформитель и исполнитель выбраны случайным образом из таблицы User с проверкой соответствующих ролей, навык и оборудование выбирались таким образом, чтобы области применения оборудования не противоречили с областью применения навыка.

Значения таблицы Ban были получены путем выделения основ нецензурных слов. Итоговый объем составил 50 записей.

Выгрузка значений из электронных таблиц в базу данных представлен в приложении А, листинг 4.2.

3.6 Интерфейс приложения

Для работы с базой данных был разработан интерфейс взаимодействия в виде telegram-бота.

В поле поиска чатов необходимо ввести @Sosed_Ushka_Bot, перейдя в этот диалог можно увидеть приветственное сообщение и ввести команду */start*. После этого будет предложено пройти персонализацию командой */personalize* под одной из имеющихся ролей. На рисунке 3.1 представлена иллюстрация вышеописанных действий и персонализация под ролью оформителя.

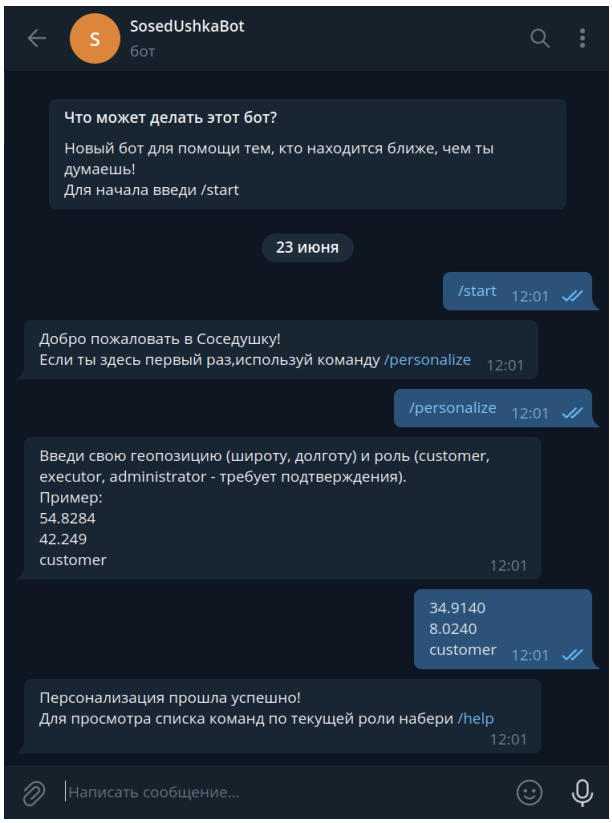

Рисунок 3.1 – Интерфейс персонализации

Чтобы увидеть список доступных команд для текущей роли можно ввести команду */help* (рисунок 3.2). Также можно сменить геопозицию пользователя командой */change_geo* (рисунок 3.3).

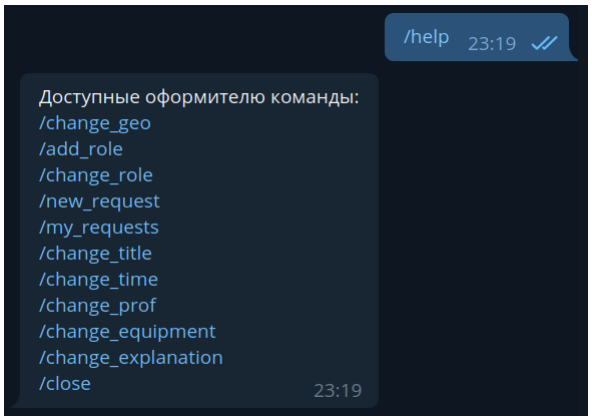

Рисунок 3.2 – Интерфейс персонализации

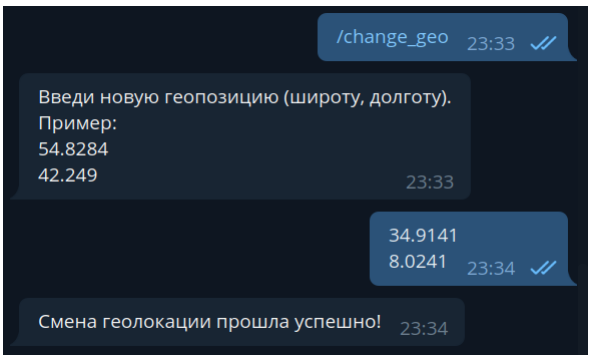

Рисунок 3.3 – Интерфейс персонализации

Рассмотрим взаимодействие с заявками. Командой */new_request* можно создать заявку, введя заголовок, время актуальности, пояснение, необходимый навык и оборудование. Ненужные поля можно оставить пустыми, кроме заголовка и актуальности. Командой */my_requests* можно получить список открытых или исполняемых заявок. Иллюстрация описанных действий отображена на рисунке 3.4.

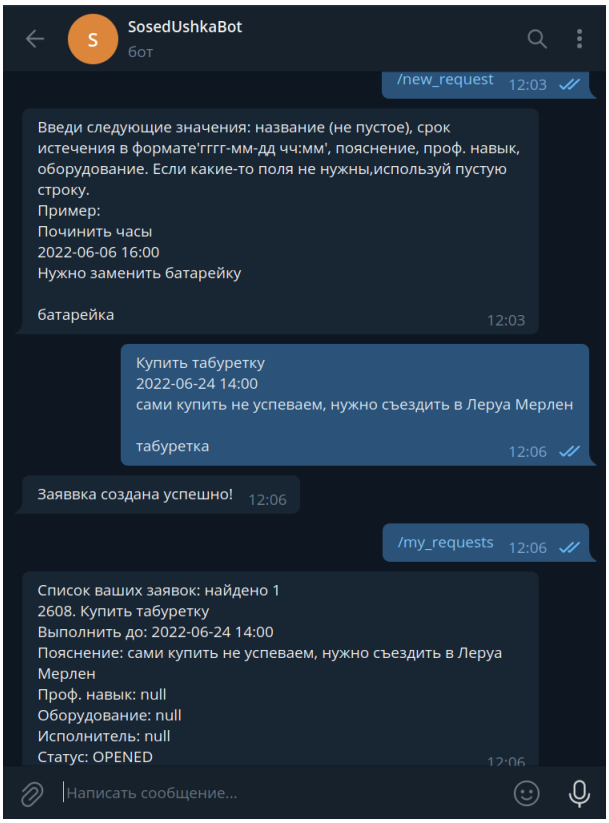

Рисунок 3.4 – Интерфейс создания и просмотра списка заявок оформителя

Каждое из полей заявки можно изменить соответствующей командой. На-

пример, пояснение изменяется использование $/change_explanation$. Также доступно закрытие заявки командой $/close$. Реализация на рисунке 3.5.

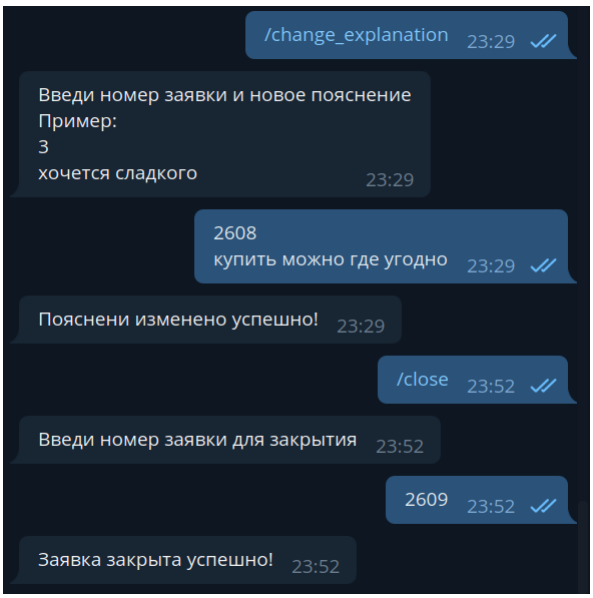

Рисунок 3.5 – Интерфейс изменения поля заявки и ее закрытия

Также имеется возможность добавления роли и соответствующего перехода к ней. Рассмотрим на смену роли от оформителя к исполнителю. Командой $/add\_role$ добавляем пользователю роль executor, и переходим к ней командой $/change\_role$. После успешного перехода появится соответствующее сообщение (рисунок 3.6).

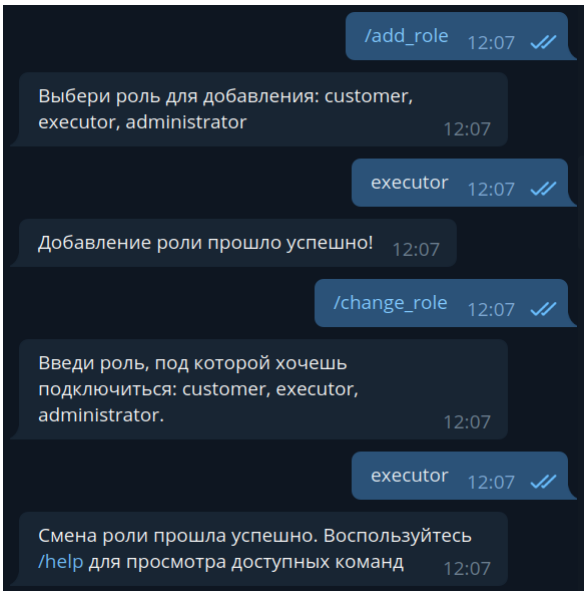

Рисунок 3.6 – Интерфейс добавления и смены роли

Исполнитель может отправить свой навык на подтверждение администратора командой */send_to_verify* (рисунок 3.7). Для этого необходимо ввести его название. Если навыка еще нет в базе, то он добавляется.

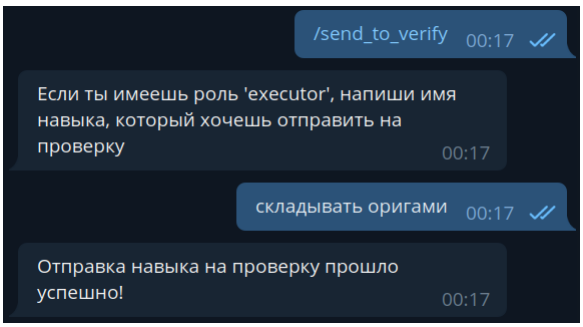

Рисунок 3.7 – Интерфейс отправки навыка исполнителя на подтверждение

Имеется возможность просмотреть список подходящих по навыкам исполнителя заявок (в том числе без необходимого навыка), указывая необходимую дальность от своей геопозиции в километрах. Это делается командой */get_request_on_my_skill*. Также можно взять одну из них на исполнение командой */take* (показано на рисунке 3.8).

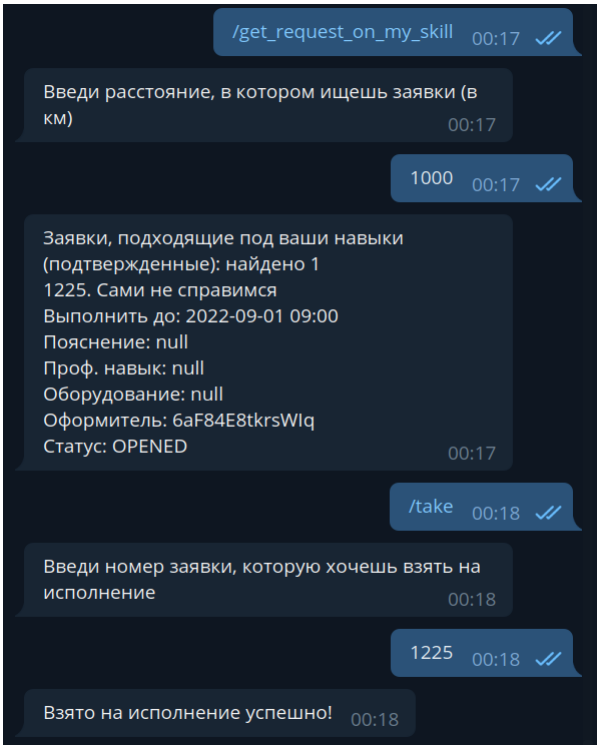

Рисунок 3.8 – Интерфейс просмотра заявок по навыку и взятия их на исполнение

Когда заявка будет выполнена, исполнитель может перевести заявку в соответствующий статус командой *done* (рисунок 3.9).

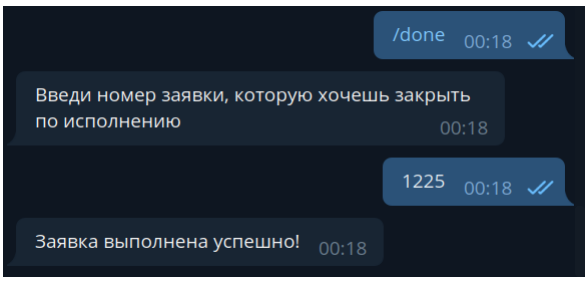

Рисунок 3.9 – Интерфейс перевода заявки в статус выполненной

Перейдем к возможностям администратора. Добавление этой роли происходит не сразу, поэтому подключение под этой ролью будет доступно только после подтверждения (рисунок 3.10).

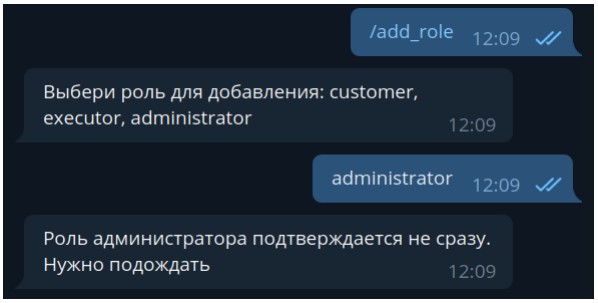

Рисунок 3.10 – Интерфейс добавления роли администратора

Администратору доступно подтверждение навыков исполнителей командой */skill_verify* с предварительным просмотром списка неподтвержденных, командой */get_skill_to_verify*. Пример работы на рисунке 3.11.

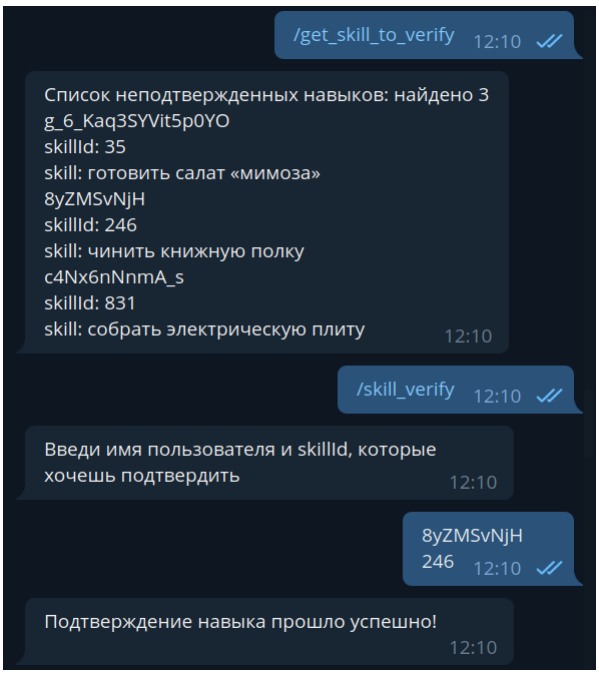

Рисунок 3.11 – Интерфейс подтверждения навыков исполнителей

Аналогичный интерфейс доступен для просмотра неподтвержденных администраторов – $/get\_admin\_to\_verify$ и $/admin\_verify$ (рисунок 3.12).

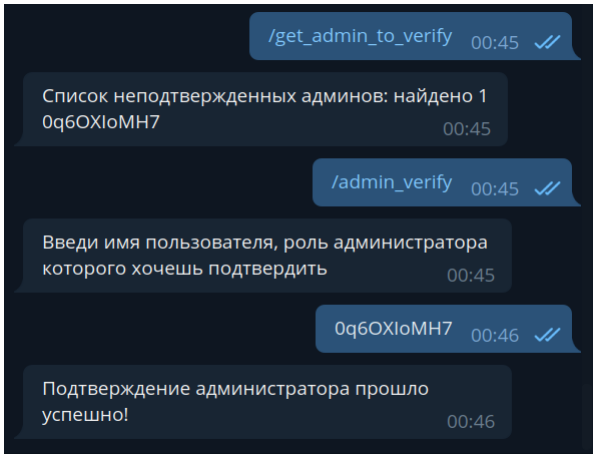

Рисунок 3.12 – Интерфейс подтверждения роли администратора

3.7 Вывод из раздела

В данном разделе был сделан выбор СУБД и средств реализации, описано создание БД, триггера, ролей с выделением прав. Также были представлены

описание генерации данных для наполнения базы и пользовательский интерфейс доступных в приложении действий.

4 Экспериментальный раздел

В данном разделе будет произведена постановка эксперимента, представлены результаты его проведения, а также сделаны выводы на основе полученных данных.

4.1 Цель эксперимента

Целью эксперимента является оценка изменения времени выполнения операции INSERT в таблицу Request при отсутствии и наличии триггера, используемого для проверки параметров создаваемой заявки.

4.2 Описание эксперимента

Замеры времени добавления заявки в систему осуществлялись при отсутствии триггера и при его наличии, причем при размерностях таблицы Ban 0, 10, 20, 50 строк.

Так как проверка осуществляется по двум полям таблицы Request, имеет место несколько ситуаций: бан-слово в поле заголовка, бан-слово в поле пояснения или отсутствие бан-слова. Первые две ситуации выделяются отдельно, так как от этого зависит количество выполняемых проверок в условии функции триггера, код которой представлен в листинге 3.1.

В таблице 4.1 представлены временные замеры по вышеупомянутым параметрам. Каждое значение получено усреднением результатов по 10 замерам. По данной таблице построен график (рисунок 4.1) отображающий исследуемые зависимости.

Таблица 4.1 – Временные замеры эксперимента (в мс)

Ситуация	**Без триггера**	**С триггером. Кол-во бан-слов:**			
		0	**10**	**20**	**50**
Бан-слово в заголовке	0.1425	0.2073	0.4291	0.4515	0.4740
Бан-слово в пояснении	0.1555	0.2008	0.4434	0.4632	0.4901
Отсутствие бан-слова	0.1491	0.1992	0.2566	0.2705	0.2829

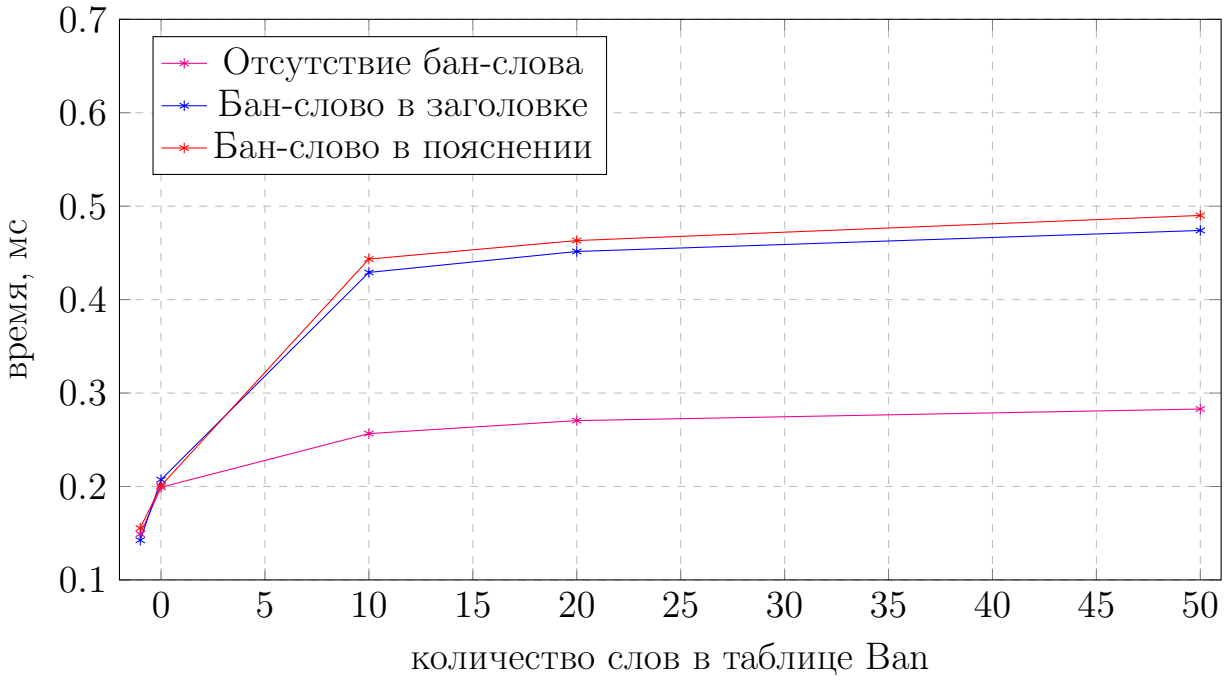

Рисунок 4.1 – Временная зависимость добавления заявки в систему от наличия триггера и размерности таблицы Ban

Из результатов эксперимента можно сделать вывод, что при отсутствии бан-слов время на добавление новой заявки в систему увеличивается до 182% в зависимости от размеров таблицы Ban. Данный прирост обусловлен получением значений всех бан-слов.

При наличии искомых слов рост времени операции вставки значений в таблицу Request увеличивается на 204-215%. Такой скачок обусловлен наличием операции обновления поля status проверяемой заявки. Также стоит отметить, что последовательность проверки полей на наличие слова приводит к разбросу в 9%, которым можно пренебречь относительно общего прироста времени выполнения.

4.3 Вывод из раздела

В данном разделе был проведен эксперимент относительно времени добавления заявки в систему при отсутствии и наличии триггера на проверку параметров заявки на нецензурную лексику. В ходе эксперимента было получено, что при отсутствии бан-слов время добавления заявки увеличивается в среднем в 1.5 раз, а при наличии – в среднем в 3 раза. Однако данный механизм является необходимым для системы общего пользования для предотвращения бескультурья.

Заключение

В ходе выполнения данной работы были выполнены следующие задачи:

- формализована задача и определен необходимый функционал;
- описана структура объектов БД, а также сделан выбор СУБД для ее хранения и взаимодействия;
- создана БД и каждую ее сущность заполнена не менее 1000 записей;
- разработан алгоритм проверки наличия бан-слов в заголовке или описании при создании заявки;
- спроектировано и реализовано приложение в формате telegram-бота для сбора и обработки заявок, которое будет взаимодействовать с описанной базой данных;
- проведено исследование времени обработки операции добавления заявок в зависимости от наличия триггера и объема данных в таблице с бан-словами.

Была достигнута поставленная цель: спроектирована и разработана база данных для сбора и обработки заявок пользователей, интерфейсом для которой стал telegram-бот.

Необходимый функционал был реализован. Так как приложение спроектировано в соответствии с принципами «чистой архитектуры», его возможности можно расширять посредством добавления новых сущностей в систему.

Также в ходе работы было выполнено исследование влияния наличия триггера проверяющего бан-слова в полях заявки на время добавления заявок. Результаты показали, что использование триггера увеличивает время операции вставки в таблицу Request в 1.5 раза при отсутствии искомых слов и в 3 раза при их наличии.

Список использованных источников

1. Telegram Messenger [Электронный ресурс]. – Режим доступа: https://telegram.org/, свободный – (14.05.2022).

2. YouDo – сервис поиска специалистов [Электронный ресурс]. – Режим доступа: https://youdo.com/, свободный – (14.05.2022).

3. Все соседи – сервис соседской взаимовыручки [Электронный ресурс]. – Режим доступа: https://vsesosedi.com/, свободный – (14.05.2022).

4. Привет, сосед! – Городская социальная сеть [Электронный ресурс]. – Режим доступа: https://privetsosed.ru/, свободный – (14.05.2022).

5. Карпова И. П. Базы данных. Учебное пособие. – М.: Московский государственный институт электроники и математики (Технический университет), 2009. – 131 с.

6. SQL – Structured Query Language [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/en-us/sql/odbc/reference/structured-query-language-sql?view=sql-server-ver15, свободный – (14.05.2022).

7. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. – СПб.: Питер, 2018. – 352 с.

8. Telegram APIs [Электронный ресурс]. – Режим доступа: https://core.telegram.org/, свободный – (20.06.2022).

9. LearnSQL – The Most Popular Databases for 2022 [Электронный ресурс]. – Режим доступа: https://learnsql.com/blog/most-popular-databases-2022/, свободный – (14.05.2022).

10. MySQL [Электронный ресурс]. – Режим доступа: https://www.mysql.com/, свободный – (14.05.2022).

11. PostgreSQL: The World's Most Advanced Open Source Relational Database [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/, свободный – (14.05.2022).

12. SQLite [Электронный ресурс]. – Режим доступа: https://www.sqlite.org/, свободный – (14.05.2022).

13. Java [Электронный ресурс]. – Режим доступа: https://www.java.com/ru/, свободный – (21.06.2022).

14. JDBC Database Access [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html, свободный – (21.06.2022).

15. IntelliJ IDEA [Электронный ресурс]. – Режим доступа: https://www.jetbrains.com/ru-ru/idea/, свободный – (21.06.2022).

16. Pl/pgSQL – SQL Procedural Language [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/docs/current/plpgsql.html, свободный – (04.06.2022).

Приложение А

Листинг 4.1 – Создание базы данных и ее сущностей

```
create database sosedushka_db;

create table sosedushka_db.user(
    tgName varchar(30) primary key,
    geoLat double precision,
    geoLong double precision,
    customerRole boolean,
    executorRole boolean,
    administratorRole boolean,
    adminVerified boolean);
create table sosedushka_db.connection(
    userTgName varchar(30) references sosedushka_db.user (tgName),
    currentRole varchar(15),
    primary key (userTgName, currentRole));
create table sosedushka_db.skill(
    id int generated by default as identity
    (start with 1 increment by 1) primary key,
    name text not null);
create table sosedushka_db.executorSkill(
    executorId varchar(30) references sosedushka_db.user (tgName),
    skillId int references sosedushka_db.skill (id),
    skillStatus varchar(20),
    primary key (executorId, skillId));
create table sosedushka_db.equipment (
    id int generated by default as identity
    (start with 1 increment by 1) primary key,
    name text);
create table sosedushka_db.request(
    id int generated by default as identity
    (start with 1 increment by 1) primary key,
    title text not null,
    periodOfRelevance timestamp,
    explanation text,
    profNecessity int references sosedushka_db.skill (id),
    equipment int references sosedushka_db.equipment (id),
    owner varchar(30) references sosedushka_db.user (tgName),
    executor varchar(30) references sosedushka_db.user (tgName),
    status varchar(20));
create table sosedushka_db.ban(
    word text not null primary key);
```

Листинг 4.2 – Заполнение базы данных

```
copy sosedushka_db.user (tgName, geoLat, geoLong, customerRole,
executorRole, administratorRole, adminVerified)
from '/tmp/db_course/user.csv'
with delimiter ',';

copy sosedushka_db.connection (userTgName, currentRole)
from '/tmp/db_course/connection.csv'
with delimiter ',';

copy sosedushka_db.skill (name)
from '/tmp/db_course/skill.csv'
with delimiter ',';

copy sosedushka_db.executorSkill (executorId, skillId, skillStatus)
from '/tmp/db_course/executorSkill.csv'
with delimiter ',';

copy sosedushka_db.equipment (name)
from '/tmp/db_course/equipment.csv'
with delimiter ',';

copy sosedushka_db.request (title, periodOfRelevance, explanation,
profNecessity, equipment, owner, executor, status)
from '/tmp/db_course/request.csv'
(format csv, null '');

copy sosedushka_db.ban (word)
from '/tmp/db_course/ban.csv'
with delimiter ',';
```