Реферат

Отчет содержит 30 страниц, 9 рисунков, 2 таблицы, 13 источников, 1 приложение.

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

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

Итогом работы стала база данных с разработанным интерфейсом взаимодействия с использованием различных ролевых моделей.

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 ее необходимым объемом данных;
- спроектировать и реализовать интерфейс доступа в формате telegram-бота;
- провести тестирование разработанного продукта.

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.

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.

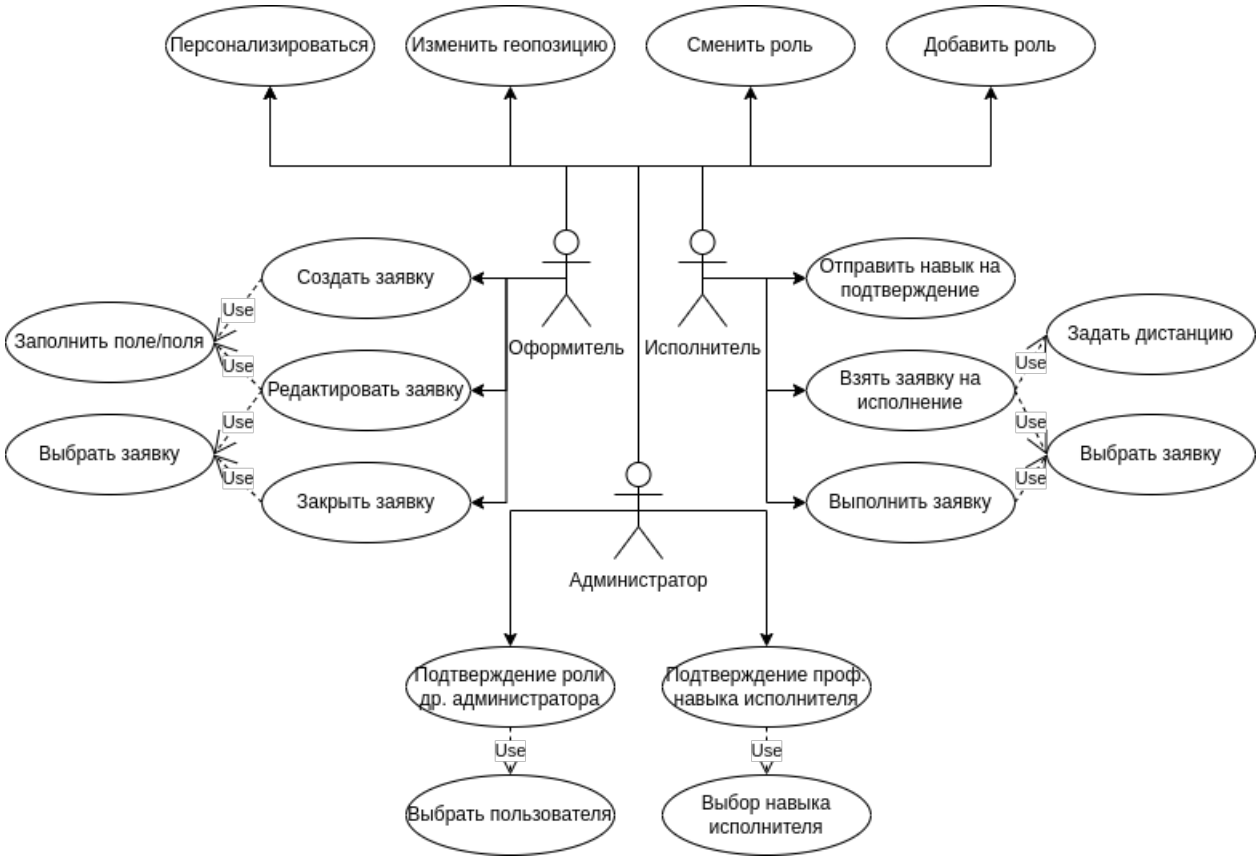

Рисунок 1.1 – Use-case диаграмма

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, реализующая связь «многие ко многим» для исполнителей и навыков ExecutorSkill;

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

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

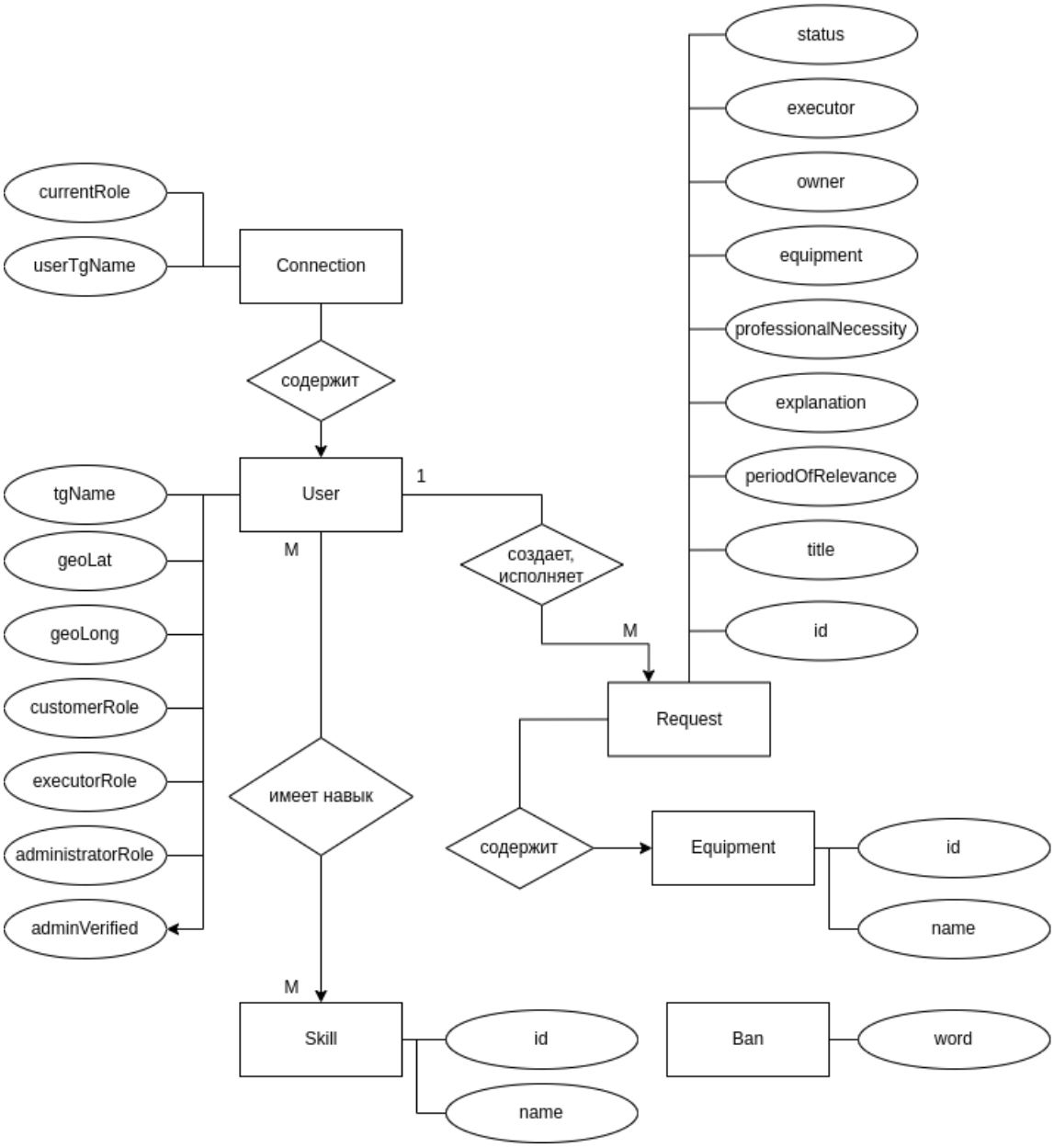

Рисунок 1.2 – ER-диаграмма в нотации Чена

1.4 Анализ существующих решений

Среди уже имеющихся проектов, решающих поставленную задачу, были выделены 3 аналога (таблица 1.1). 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 [2]	Электронная почта или Вконтакте, Одноклассники, Google, Mail, Apple	На уровне города	Более 2.5 миллионов пользователей	Заказо-ориентированность со стабильной системой отзывов
Все соседи [3]	Номер телефона	Устанавливается с точностью до дома	Около 60 тыс. пользователей	Бартерная система взаимовыручки
Привет, сосед! [4]	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. Избавиться от первого из них не составляет труда (стоит повысить точность геолокации), тогда как последние два могут быть сведены к использованию популярного мессенджера, имеющего большой охват аудитории. Таким вариантом можно смело назвать telegram, который не уступает приведенным проектам в кроссплатформенности.

1.5 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, каких особенностей в них не хватает.

2 Конструкторский раздел

В данном разделе будут представлены этапы проектирования базы данных, выделены конкретные действия ролевой модели, спроектирован триггер и интерфейс взаимодействия.

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 – реализует связь «многие ко многим» исполнителей и навыков. 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.

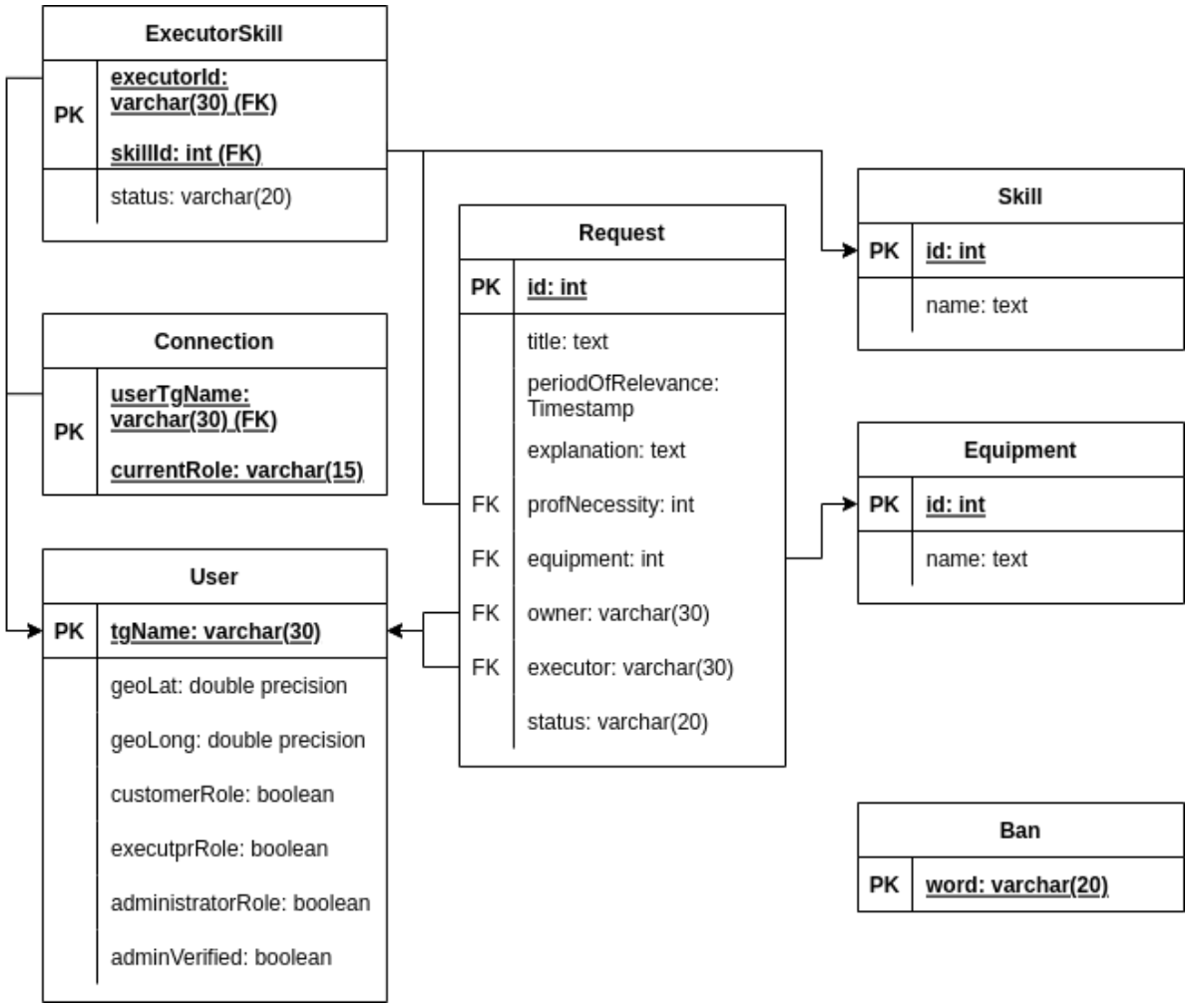

Рисунок 2.1 – ER-диаграмма системы

2.2 Ролевая модель

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.

Сценарий создания базы данных и выделение ролей представлены в приложение А.

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.

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

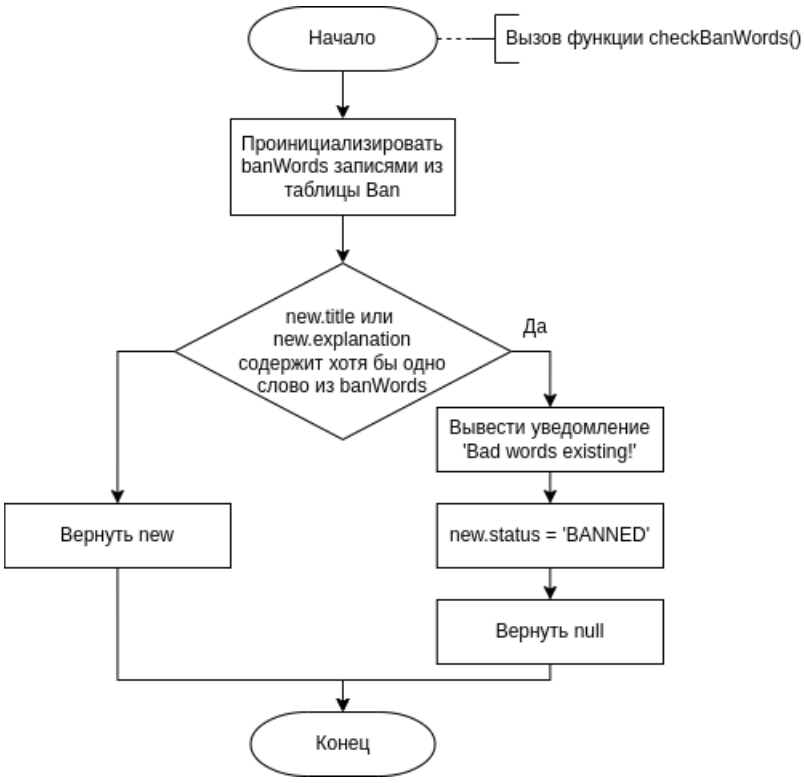

Рисунок 2.2 – Триггер AFTER на создание новой заявки

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» [5]. Использование такого подхода позволило выделить несколько компонентов: доступ к данным, бизнес-логика, транспортный уровень и UI, реализуемый на основе telegramAPI [6].

2.5 Вывод из раздела

В данном разделе были описаны этапы проектирования базы данных и приложения.

3 Технологический раздел

В данном разделе будут выбраны СУБД, средства реализации приложения, а также спроектирован пользовательский интерфейс.

3.1 Выбор СУБД

В данной работе предусмотрена четкая структурированность хранимых данных, что позволяет использовать реляционную базу данных, являющуюся удобной и наиболее широко используемой. Данный тип БД обеспечивает хранение данных в виде двумерных таблиц, столбцы которых определяют множество используемых значений, а строки – конкретные записи базы данных.

Среди самых распространенных систем управления базами данных можно выделить Microsoft SQL Server [7], PostgreSQL [8] и Oracle [9]. При реализации будет использован PostgreSQL, так как он уже использовался в ранее изученном курсе.

3.2 Средства реализации

В качестве используемого языка программирования выбрана Java [10], так как:

- данный язык является объектно-ориентированным, позволяя использовать наследование, интерфейсы, абстракции и т. д.;
- имеет JDBC [11] – API для выполнения SQL-запросов к базам данных.

В качестве среды разработки используется IntelliJ IDEA [12], поскольку данная программа:

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

3.3 Реализованные функции

Для спроектированного триггера была написана соответствующая функция с помощью PL/pgSQL [13] – процедурного расширения языка SQL, используемого в СУБД PostgreSQL. Код функции представлен в листинге 3.1.

Листинг 3.1 – Реализация триггера AFTER на добавление заявки

```
create or replace function checkBanWords()
returns trigger as
$$
declare
    banWords text ARRAY;
begin
    banWords := ARRAY(select '%' || word || '%' from sosedushka_db.ban);
    if new.title like any (banWords) or new.explanation like any (banWords)
    then
        raise notice 'Bad words existing!';
        update sosedushka_db.request set status='BANNED' where id=new.id;
        return null;
    else
        return new;
    end if;
end
$$
language plpgsql

create trigger newRequest
after insert
on sosedushka_db.request
for each row
execute function checkBanWords()
```

3.4 Интерфейс приложения

Для работы с базой данных был разработан интерфейс взаимодействия в виде telegram-бота. На рисунках 3.1-3.4 представлены примеры персонализации, взаимодействия с системой оформителя, исполнителя и администратора.

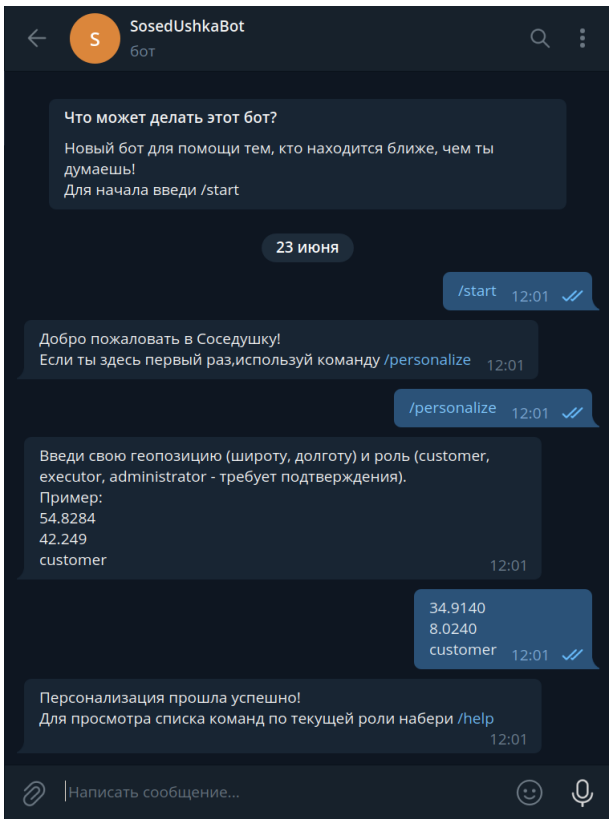

Рисунок 3.1 – Интерфейс персонализации

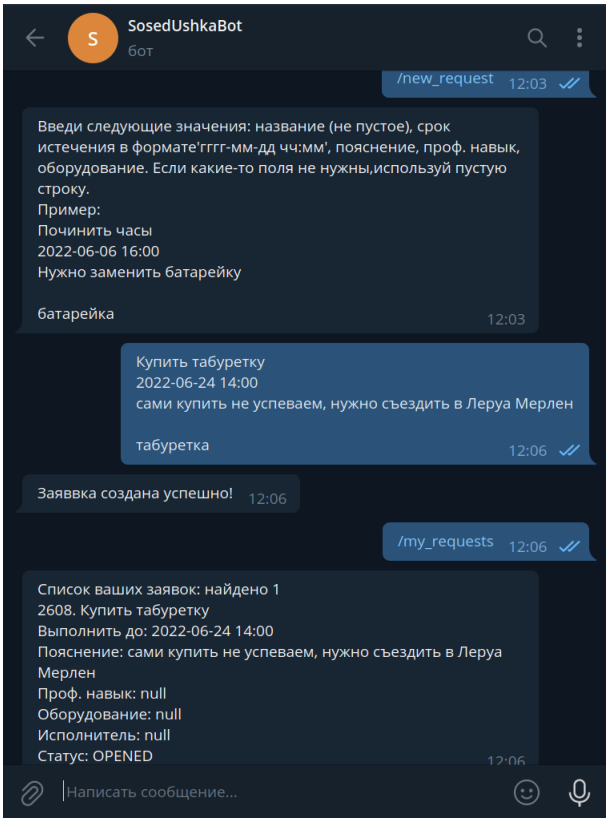

Рисунок 3.2 – Интерфейс оформителя

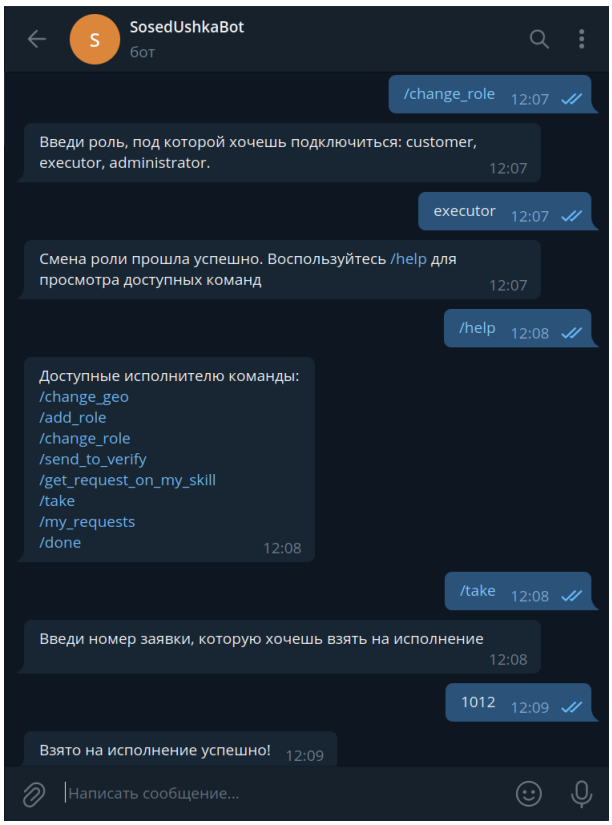

Рисунок 3.3 – Интерфейс исполнителя

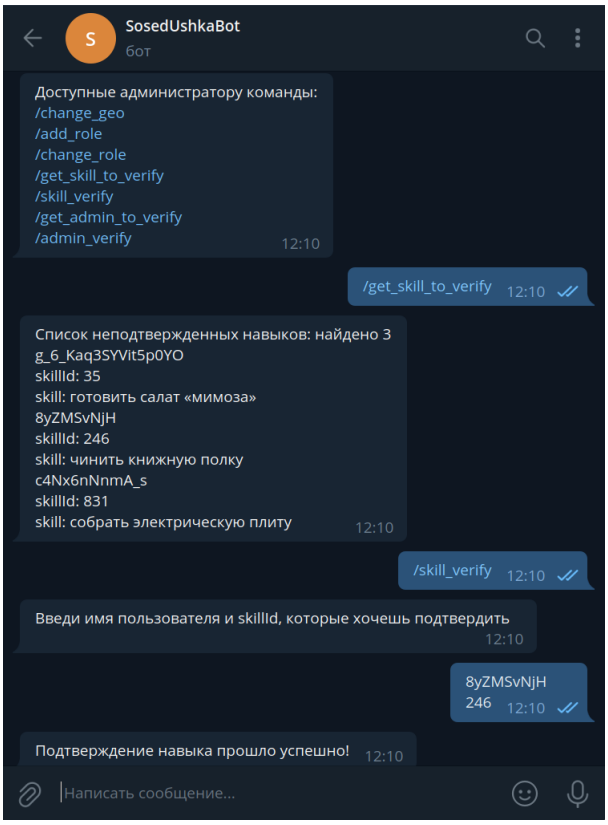

Рисунок 3.4 – Интерфейс администратора

3.5 Вывод из раздела

В данном разделе был сделан выбор СУБД и средств реализации, а также реализован триггер и пользовательский интерфейс.

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 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

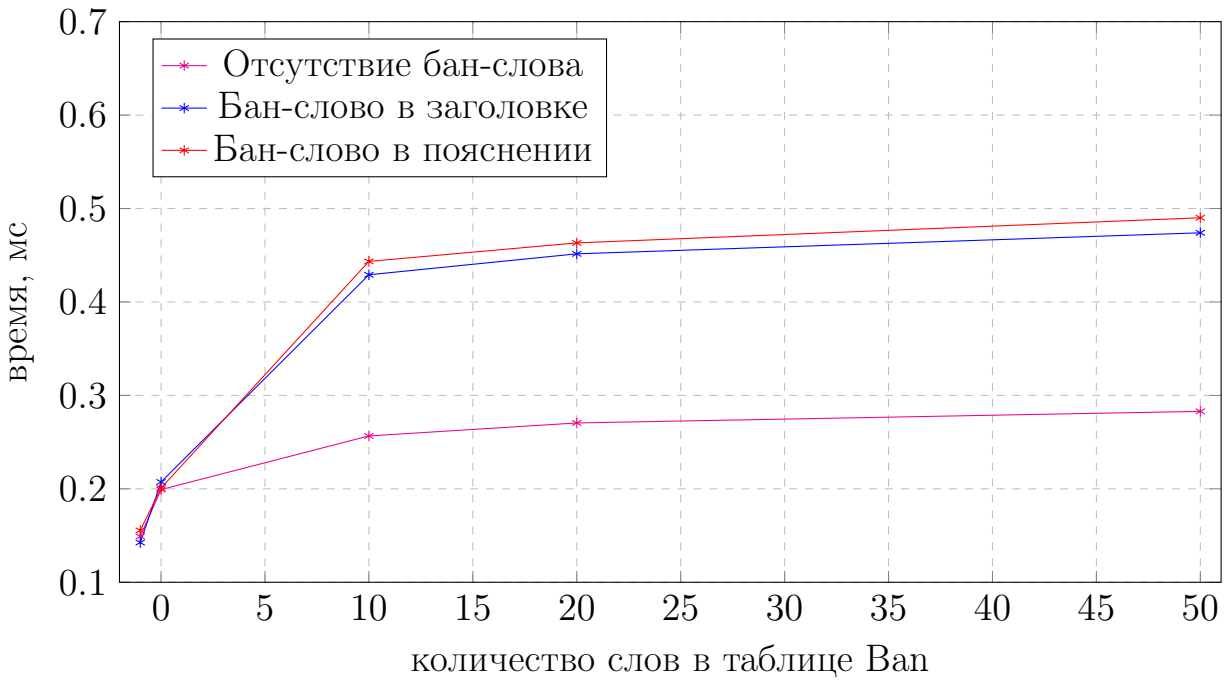

Рисунок 4.1 – Временная зависимость добавления заявки в систему от наличия триггера и размерности таблицы Ban

Из результатов эксперимента можно сделать вывод, что при отсутствии бан-слов время на добавление новой заявки в систему увеличивается на 34-82% 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 заполнена необходимым объемом данных;
- спроектирован и реализован интерфейс доступа в формате telegram-бота;
- проведено тестирование разработанного продукта.

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

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

[7] Microsoft SQL Server 2019 [Электронный ресурс]. – Режим доступа: https://www.microsoft.com/ru-ru/sql-server/sql-server-2019, свободный – (21.06.2022).

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

[9] Oracle Database [Электронный ресурс]. – Режим доступа: https://www.oracle.com/database/, свободный – (21.06.2022).

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

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

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

[13] 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 и заполнение базы данных

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

create table sosedushka_db.connection(
    userTgName varchar(30) references sosedushka_db.user (tgName),
    currentRole varchar(15),
    primary key (userTgName, currentRole)
);
copy sosedushka_db.connection (userTgName, currentRole)
    from '/tmp/db_course/connection.csv'
    with delimiter ',';

create table sosedushka_db.skill(
    id int generated by default as identity
    (start with 1 increment by 1) primary key,
    name text not null
);
copy sosedushka_db.skill (name)
    from '/tmp/db_course/skill.csv'
    with delimiter ',';

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

```
44
45 create table sosedushka_db.equipment (
46     id int generated by default as identity
47     (start with 1 increment by 1) primary key,
48     name text
49 );
50 copy sosedushka_db.equipment (name)
51     from '/tmp/db_course/equipment.csv'
52     with delimiter ',';
53
54 create table sosedushka_db.request(
55     id int generated by default as identity
56     (start with 1 increment by 1) primary key,
57     title text not null,
58     periodOfRelevance timestamp,
59     explanation text,
60     profNecessity int references sosedushka_db.skill (id),
61     equipment int references sosedushka_db.equipment (id),
62     owner varchar(30) references sosedushka_db.user (tgName),
63     executor varchar(30) references sosedushka_db.user (tgName),
64     status varchar(20)
65 );
66 copy sosedushka_db.request (title, periodOfRelevance, explanation,
67     profNecessity, equipment, owner, executor, status)
68     from '/tmp/db_course/request.csv'
69     (format csv, null '');
70
71 create table sosedushka_db.ban(
72     word text not null primary key
73 );
74 copy sosedushka_db.ban (word)
75     from '/tmp/db_course/ban.csv'
76     with delimiter ',';
77
78
79 create role customer with login password '1111';
80 grant insert, update on table sosedushka_db.user to customer;
81 grant insert, update on table sosedushka_db.connection to customer;
82 grant all privileges on table sosedushka_db.request to customer;
83 grant insert on table sosedushka_db.skill to customer;
84 grant insert on table sosedushka_db.equipment to customer;
85
86 create role executor with login password '2222';
87 grant insert, update on table sosedushka_db.user to executor;
88 grant insert, update on table sosedushka_db.connection to executor;
89 grant select, update on table sosedushka_db.request to executor;
90 grant insert on table sosedushka_db.skill to executor;
91 grant insert on table sosedushka_db.executorskill to executor;
```

```
92
93 create role administrator with login password '4813';
94 grant all privileges on table sosedushka_db.user to administrator;
95 grant all privileges on table sosedushka_db.connection to administrator;
96 grant all privileges on table sosedushka_db.executorskill to
97 administrator;
98 grant all privileges on table sosedushka_db.ban to administrator;
```