МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

*Факультет* вычислительной техники

*Кафедра* «Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

**БАКАЛАВРСКАЯ РАБОТА**

на тему

# СИСТЕМА УЧЁТА ПОКАЗАНИЙ СЧЁТЧИКОВ ВОДЫ С ИСПОЛЬЗОВАНИЕМ ЧАТ-БОТА

**Студент** 30.05.24 Сафронов Денис Витальевич
(подпись, дата) (ФИО полностью)

**Руководитель** 30.05.24 Токарев А.Н.
(подпись, дата) (фамилия, инициалы)

**Нормоконтролёр** 30.05.24 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

*Работа допущена к защите* (протокол зас. кафедры от 27.05.24 г. № 11 )

**Заведующий кафедрой** _____ М.А. Митрохин
(подпись) (фамилия, инициалы)

*Работа защищена с отметкой* отлично (протокол заседания ГЭК от 04.06.2024 № 7 )

**Секретарь ГЭК** _____ Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

*Факультет*
вычислительной техники

*Кафедра*
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«19» 04 2024 г.

# ЗАДАНИЕ
# НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Сафронов Денис Витальевич гр. 20ВВП1 факультета вычислительной техники.

2.Тема работы «Система учёта показаний счётчиков воды с использованием чат-бота»

Тема утверждена приказом ПГУ № 405/о от "19" апреля 2024г.

3. Руководитель бакалаврской работы к.т.н., доцент Токарев Андрей Николаевич

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Назначение разработки: система учёта показаний счётчиков воды с использованием чат-бота. Разработать систему для сбора и обработки данных показаний счётчиков использующую телеграм в качестве клиента, spring Framework на стороне сервера и базу данных postgresql. Язык программирования Java, postgresql, среда разработки intellij IDEA, pgadmin4.

5. Перечень подлежащих разработке вопросов

1. Анализ предметной области автоматизированного учёта показаний счётчиков.

2. Разработка системы учёта показаний счётчиков на основании проделанного анализа.

3.Анализ системы учёта показаний счётчиков.

6. Календарный график выполнения работы

| № п/п | Наименование этапов работы | Объем работы | Срок выполнения | Подпись руководителя |
|---|---|---|---|---|
| 1 | Анализ предметной области автоматизированного учёта показаний счётчиков. | 15% | 19.04.2024-30.04.2024 | |
| 2 | Выбор среды разработки и фреймворков | 15% | 1.05.2024-6.05.2024 | |
| 3 | Разработка системы учёта показаний счётчиков воды на основании проведённого анализа | 38% | 7.05.2024-15.05.2024 | |
| 4 | Анализ системы учёта счётчиков воды с использованием чат-бота | 20% | 16.05.2024-27.05.2024 | |
| 5 | Оформление выпускной квалификационной работы | 12% | 28.05.2024-31.05.2024 | |

Дата выдачи задания "19" апреля 2024г.

Руководитель бакалаврской работы _____ 19.04.24
*(подпись, дата)*

Задание к исполнению принял студент _____ 19.04.24.
*(подпись, дата)*

Работу к защите допустить
Декан факультета _____ А.Р. Фисаво 03.06.2024
*(подпись, дата)*

## Реферат

Бакалаврская работа содержит 116 листов, 70 рисунков, 2 таблицы, 7 источников, 3 приложения

ЧАТ-БОТЫ, СИТЕМА УЧЁТА, БАЗА ДАННЫХ, JAVA, TELEGRAM, POSTRESQL, INTELLIJ IDEA

Цель работы – изучение систем учёта показаний счётчиков и разработка программной системы для сбора и обработки показаний с использованием популярной социальной сети – телеграм.

Объектом разработки является автоматизированная система учёта, выполняющую запись, обработку и получение показаний, оповещение пользователя о имеющихся задолженностей и времени внесения своевременных показаний и платежей.

В результате проведённой работы разработана система, полностью соответствующая поставленной цели данной выпускной квалификационной работы. В ходе достижения результата были использованы следующие технологии: java, spring framework, telegram API, postgresql.

**Содержание**

Перечень сокращений и обозначений ..... 4
Введение ..... 5
1 Анализ предметной области ..... 6
1.1 Описание предметной области ..... 6
1.2 Анализ аналогов ..... 7
1.2.1 Сервис «Квартплата онлайн» ..... 7
1.2.2 Портал «mos.ru» ..... 12
1.2.3 Сервис «ГИС ЖКХ» ..... 15
1.2.4 Вывод ..... 18
1.3 Постановка задачи ..... 20
1.4 Выбор технологий и средств разработки приложения ..... 21
1.4.1 Архитектура приложения ..... 21
1.4.2 Выбор среды разработки ..... 26
1.4.3 Выбор базы данных ..... 27
2 Реализация приложения ..... 29
2.1 Настройка среды разработки ..... 29
2.2 Разработка БД ..... 32
2.3 Реализация телеграм бота ..... 36
2.4 Тестирование и отладка ..... 45
3 Руководство пользователя ..... 55
4 Безопасность жизнедеятельности ..... 59
Заключение ..... 61
Список использованных источников ..... 62
Приложение А – UML-диаграммы ..... 63
Приложение Б – Листинг программы ..... 68
Приложение В – Презентация ..... 111

## Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД – база данных

БЖД – безопасность жизнедеятельности

ПО – програмное обеспечение

ЯП – языки программирования

API (Application Programming Interface) – описание способов взаимодействия одной компьютерной программы с другими

IDE (Integrated development environment) – среда разработки

UML (Unified Modeling Language) – унифицированный язык моделирования

## Введение

В связи с крупномасштабным увлечением производства во многих сферах возникают проблемы учёта расхода ресурсов. Существующие системы учёта часто сталкиваются с проблемами устаревания или недоступности ПО. Вместе с тем растёт потребность в расширении функционала существующих систем.

Положительные стороны заключается в том, что телеграмм бот предоставляет простой и понятный интерфейс, что упрощает его использование. Доступность данного продукта. Мессенджер телеграмм имеет версии под все платформы, соответственно для его использование не требуется дополнительная установка клиента. Следовательно, можно выделить следующие преимущества данного продукта:

– простотой и удобством использования;

– наличием широкого спектра возможностей, что делает их универсальным инструментом для пользователей с разными интересами и потребностями;

– доступностью API и инструментов для создания и настройки ботов.

Актуальность данной темы заключается в том, что в настоящее время учёт расхода ресурсов важен во многих сферах жизни, и многие компании и организации часто заинтересованы в наличии актуального современного ПО, выполняющее этот функционал. Система учёта показаний счётчиков позволит быстро получать актуальную информацию по расходам за разный период времени. Кроме того, получить данные не только по расходам ресурсов, но и по расходам средств и на основании этих показаний в кратчайшие сроки разработать стратегию по уменьшению этих расходов.

# 1 Анализ предметной области

## 1.1 Описание предметной области

Потребность в системе учёта расхода водяных счётчиков возникает у жильцов квартир и работников предприятий. Учёт расхода воды позволяет контролировать и переопределять расходы средств и ресурсов. Позволяет получить детальный отчёт расхода ресурсов и на их основании проработать дальнейшие стратегии их использования.

Для отправки показаний счётчиков за текущий месяц требуется указать в квитанции текущие показания. Информация о расходах за предшествующие месяцы, находится на бумажных носителях(квитанциях). Таким образом учёт и хранение данных о расходе становится труднозатратным и долгим процессом. Следовательно, использование приложения имеет такие преимущества как:

– удобство использования: Учёт расходов в приложении позволяет быстро отправить и получить данные в любой момент времени с помощью своего смартфона;

– оптимизация расходов: Благодаря возможности отслеживать свой расход воды в реальном времени, пользователи могут выявлять излишние затраты и принимать меры по их сокращению, что способствует экономии средств и ресурсов;

– персонализация: Использование приложения позволяет получать уведомления об появлении новых задолженностей;

– детальный анализ: Приложение позволяет получить расход воды за определённый период времени.

В целом, учёт расходов воды позволяет сэкономить на затраченных средствах, сократить использование ресурсов воды.

## 1.2 Анализ аналогов

Перед началом создания собственного проекта, было принято решение изучить аналогичные приложения и сайты, чтобы посмотреть, как с данной задачей справились другие разработчики. Таким образом были выделены слабые и сильные стороны существующих приложений и сайтов для того, чтобы улучшить будущий продукт.

В качестве аналогов будущего проекта были выбраны сайты «mos.ru» и «квартплата онлайн» [1], «ГИС ЖКХ» [2].

### 1.2.1 Сервис «Квартплата онлайн»

Для того, чтобы начать пользоваться сервисом нужно войти в личный кабинет (рисунок 1).

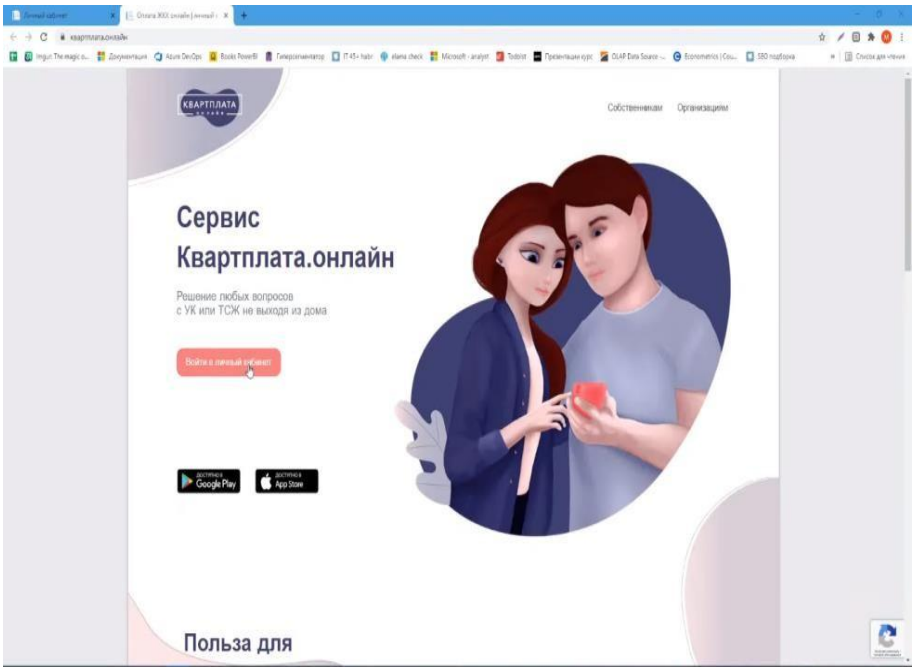



Рисунок 1 – Стартовое окно сервиса

Выбрать кнопку «Добавить помещение» в личном кабинете (рисунок 2).

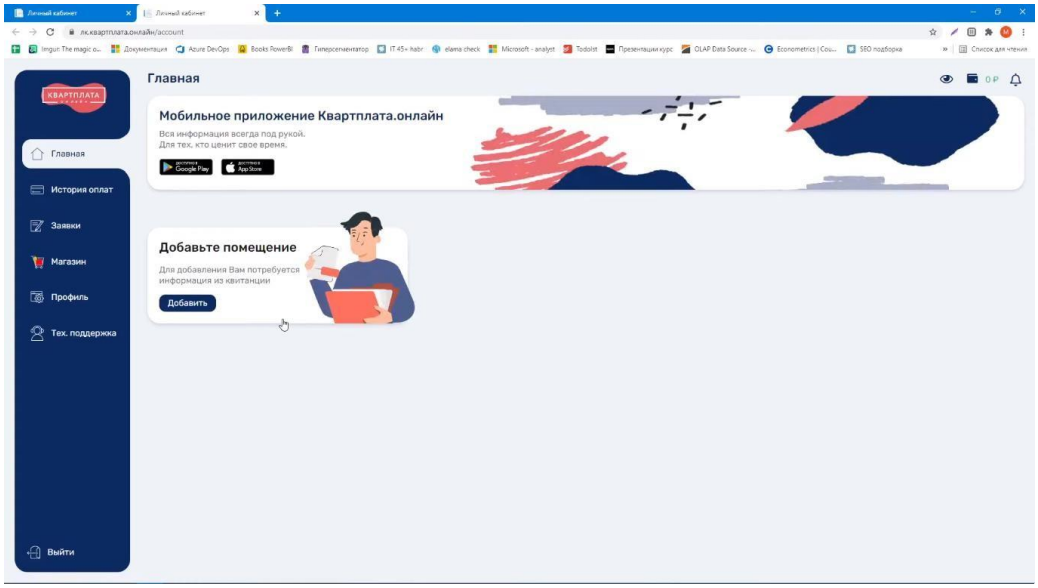



Рисунок 2 – Окно личного кабинета сервиса

Для добавления помещения нужно указать ИНН организации, номер лицевого счёта и подтверждающий номер договора (рисунки 3-5).

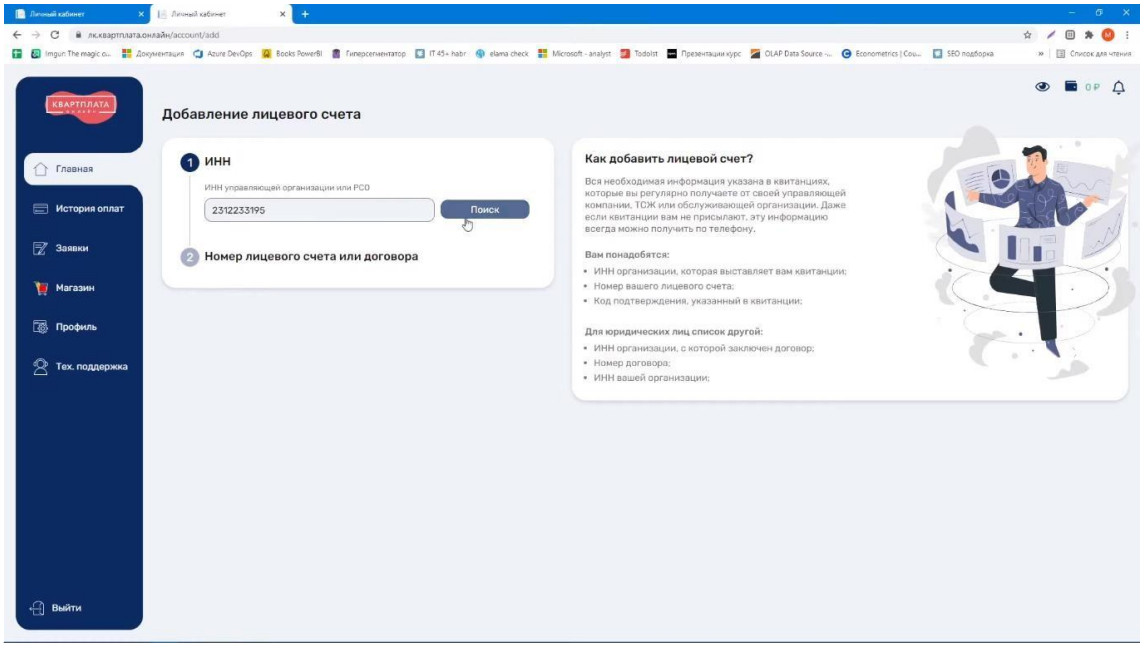



Рисунок 3 – Окно ввода ИНН

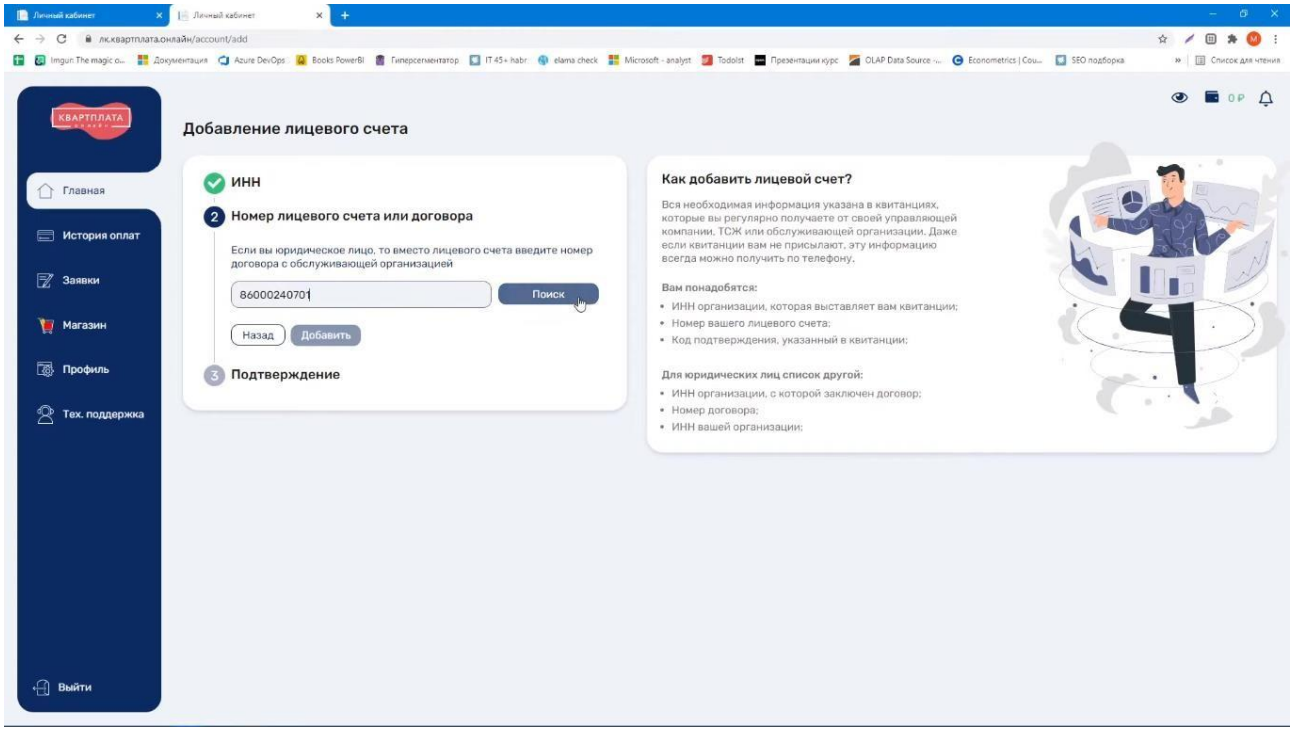



Рисунок 4 – Окно ввода номера договора

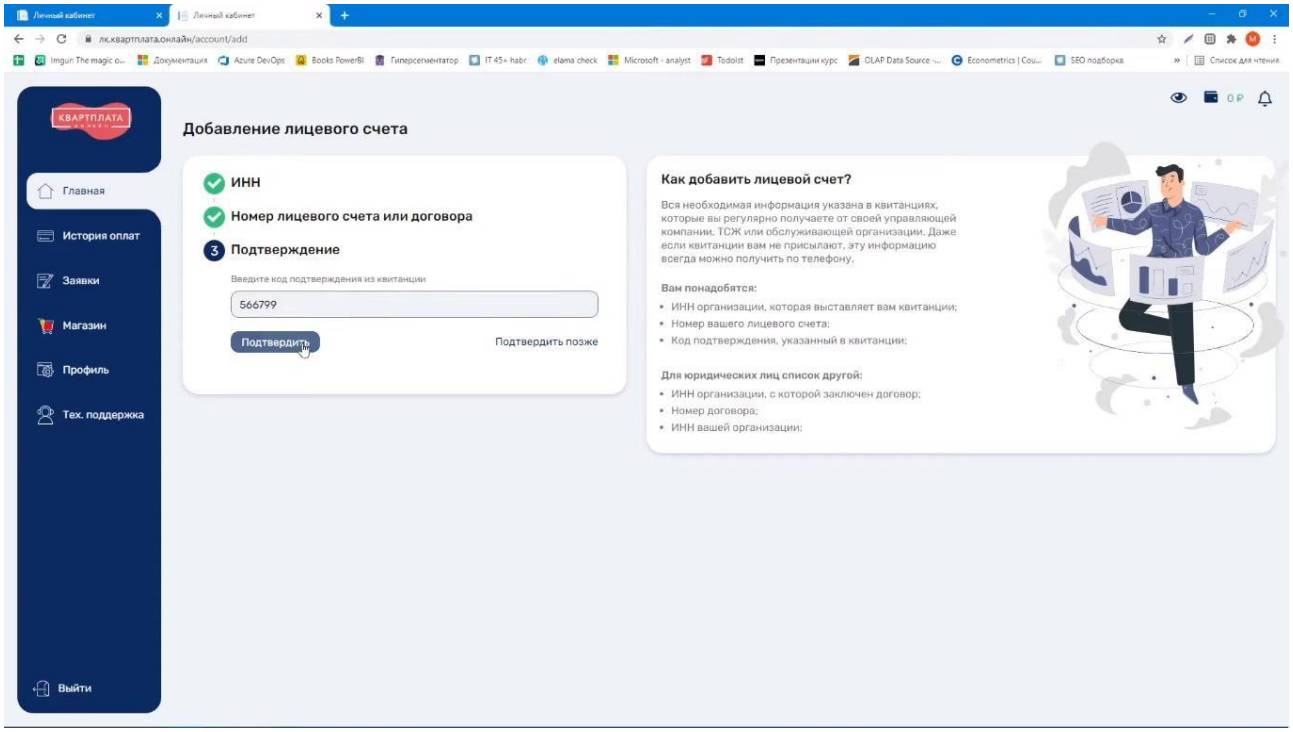



Рисунок 5 – Окно ввода подтверждающего кода

После того, как мы указали все данные договора наше помещение будет отображаться в личном кабинете пользователя (рисунок 6).

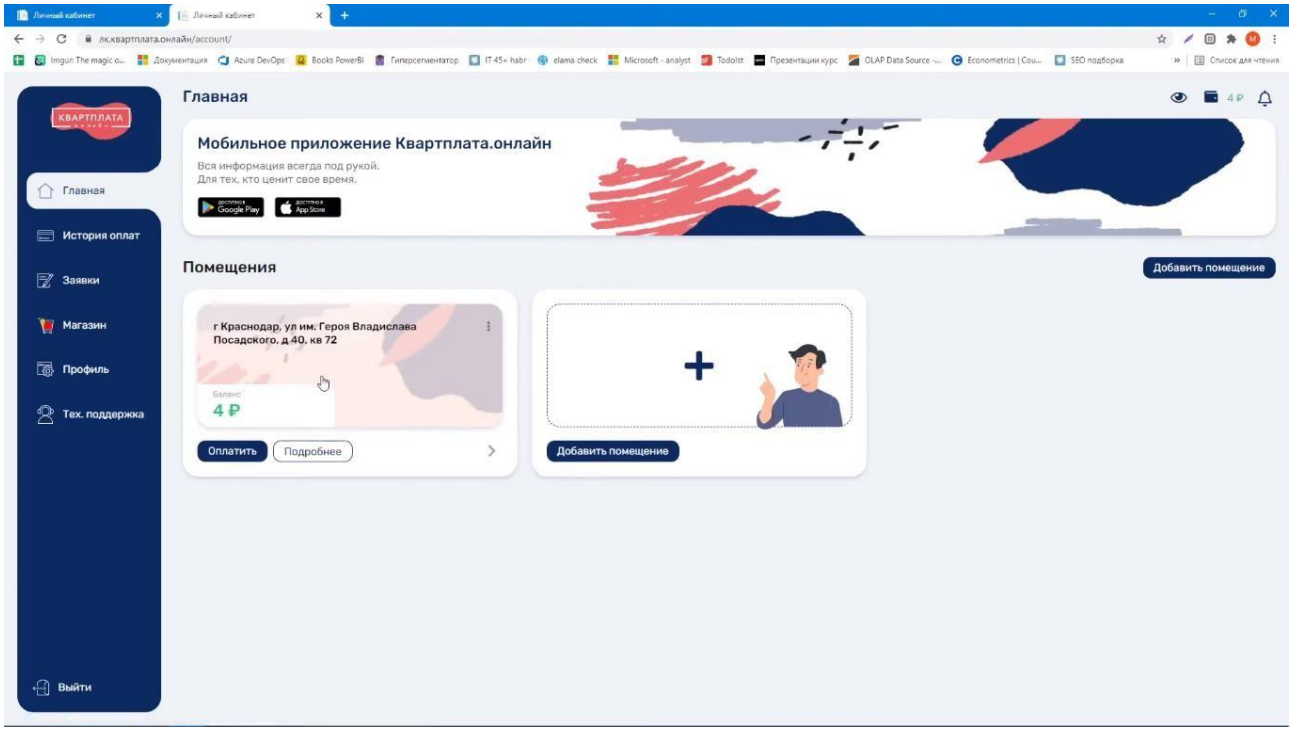



Рисунок 6 – Личный кабинет после добавление помещения

Для того, чтобы просмотреть информацию о добавленном помещении нужно кликнуть по нашему помещению (рисунок 7).

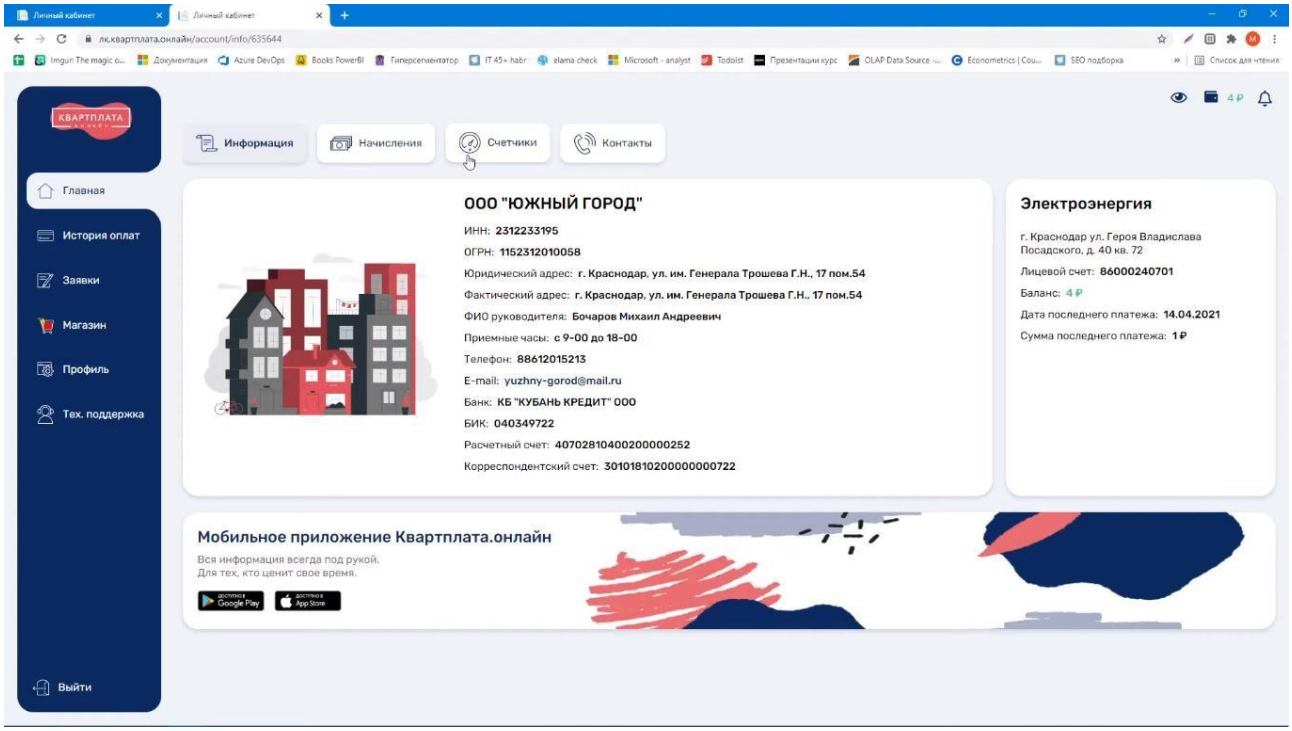



Рисунок 7 – Окно с информацией о помещении

Для того чтобы узнать показания счётчиков нужно нажать на вкладку «счётчики» (рисунок 8).

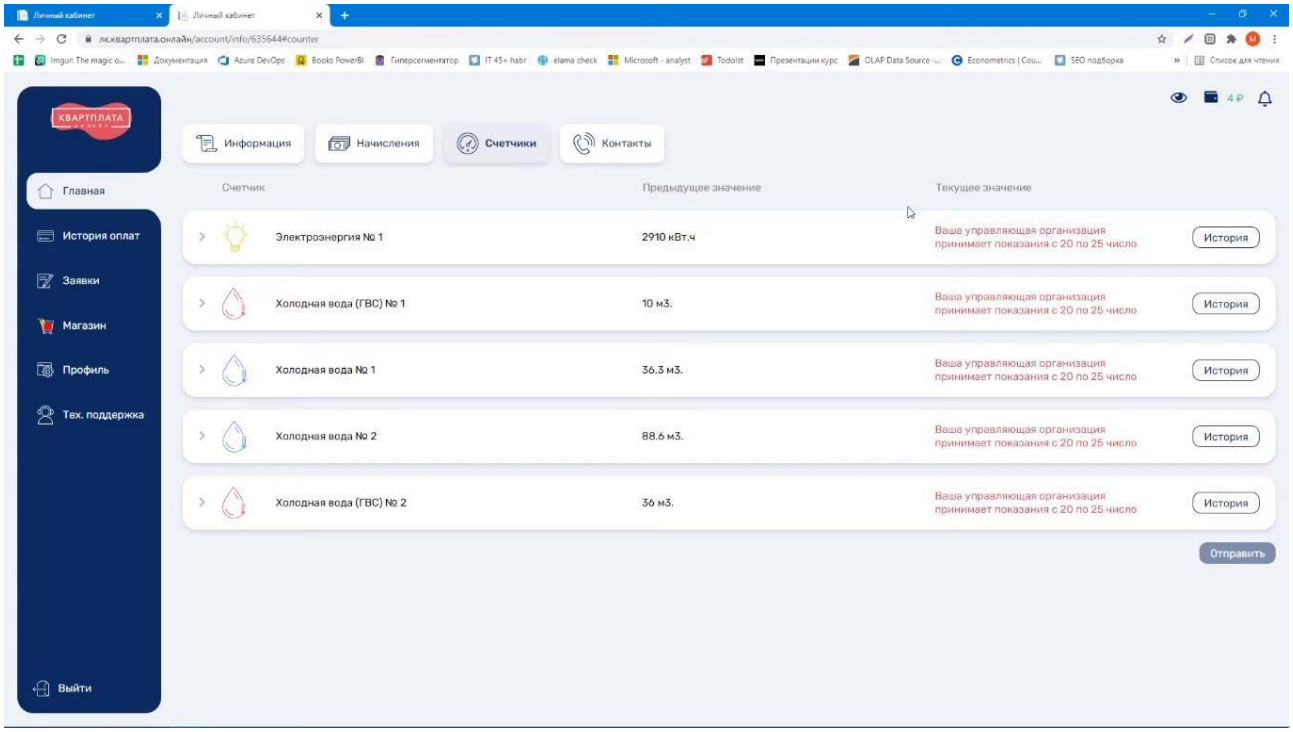



Рисунок 8 – Вкладка счётчики

Если нажать на кнопку «история» можно узнать историю показаний за каждый месяц (рисунок 9).

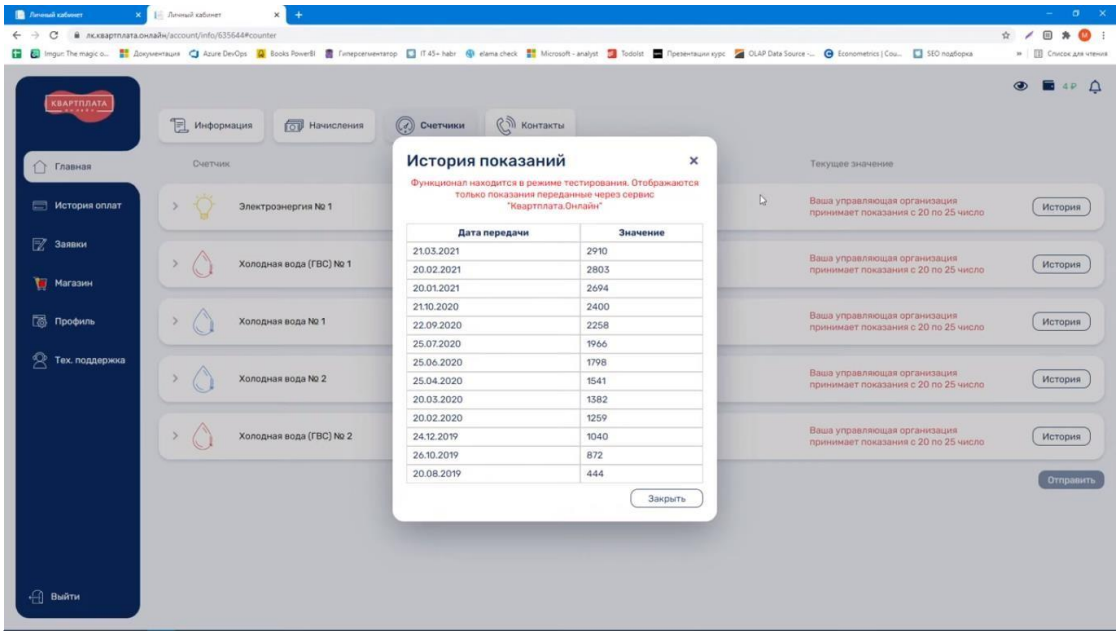



Рисунок 9 – Окно ежемесячных показаний счётчика

Для того чтобы узнать историю начислений/оплат нужно зайти во вкладку «начисления» (рисунок 10).

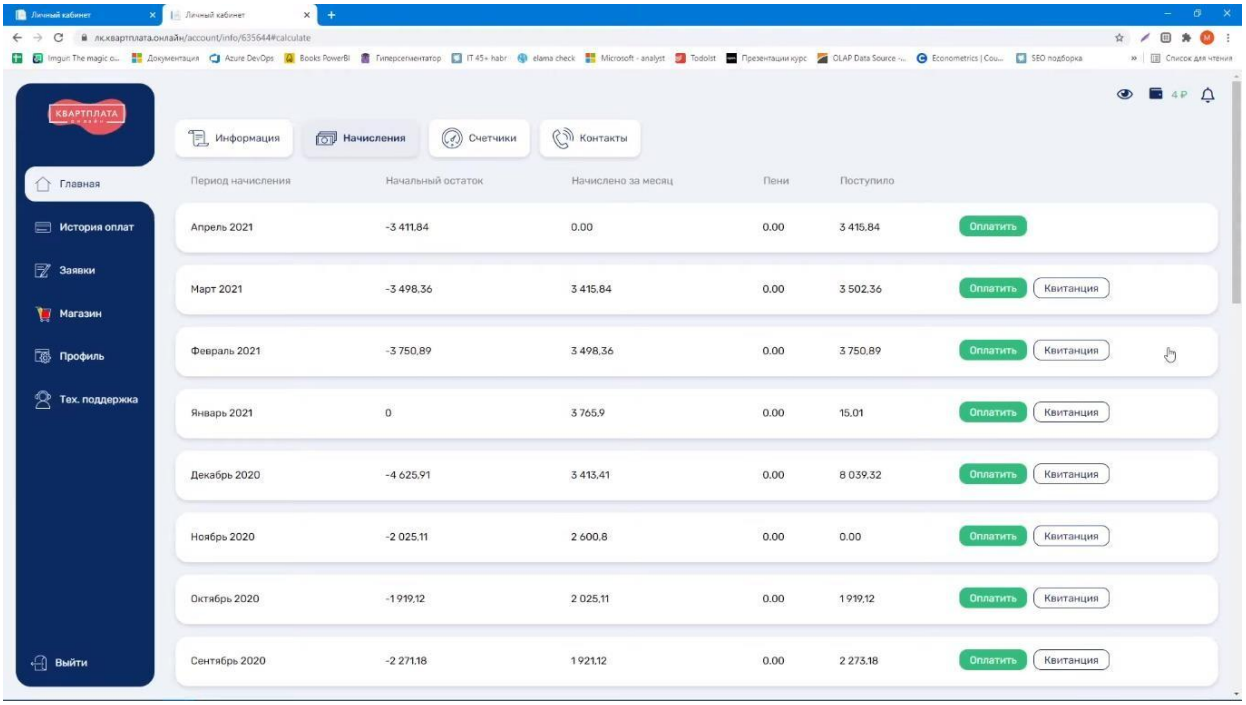



Рисунок 10 – Окно с ежемесячными начислениями

### 1.2.2 Портал «mos.ru»

Для работы с учётом расходов воды в портале mos.ru требуется авторизоваться (рисунок 11).

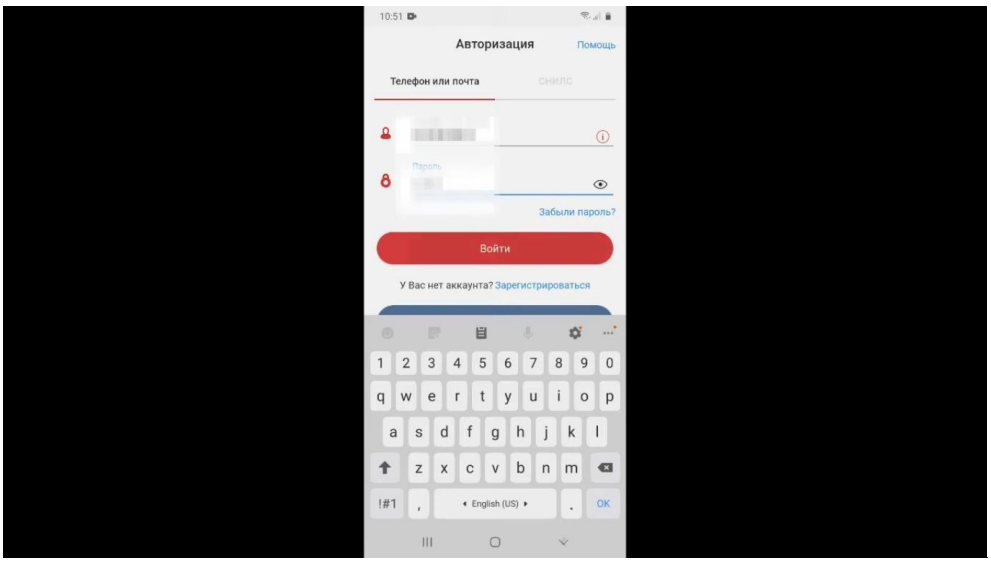



Рисунок 11 – Окно авторизации на mos.ru

В личном кабинете выбираем вкладку «квартиры» и нажимаем на кнопку «Добавить квартиру» (рисунок 12).

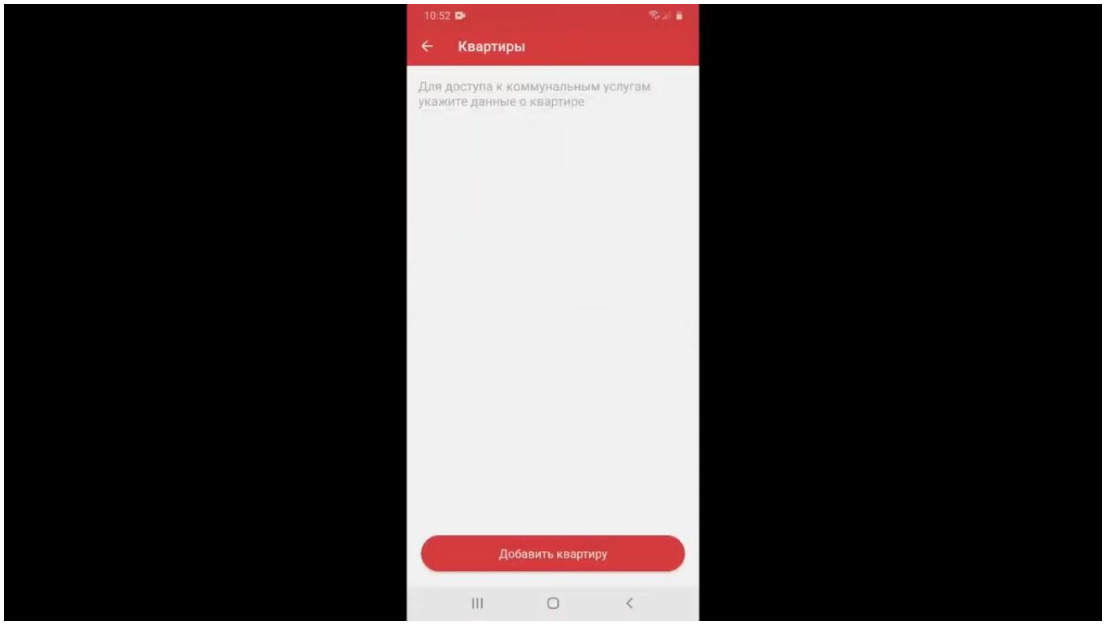



Рисунок 12 – Окно "квартиры"

Далее заполняем название квартиры, адрес, код плательщика и номер лицевого счёта (рисунки 13-14).

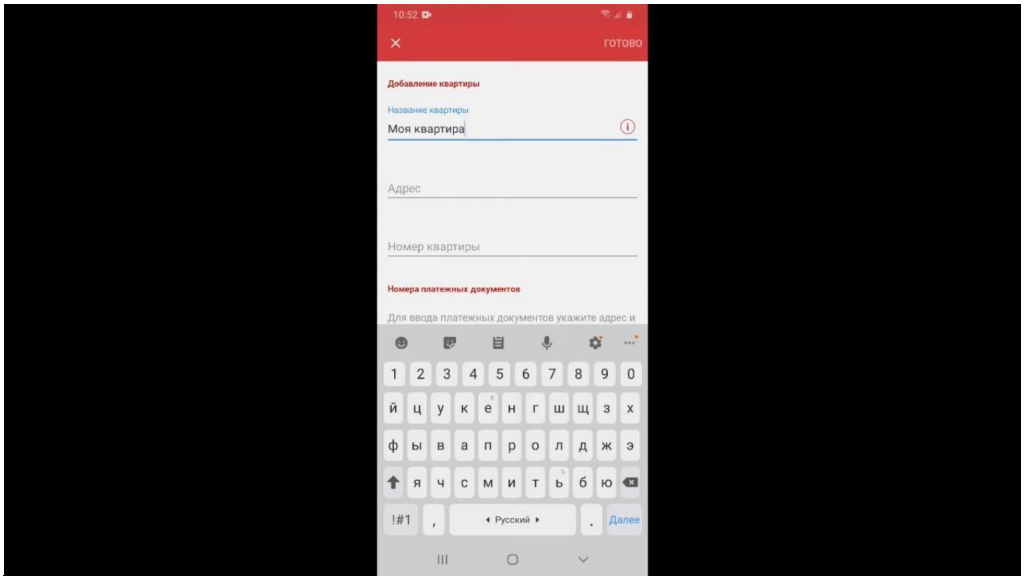



Рисунок 13 – Окно заполнения данных квартиры

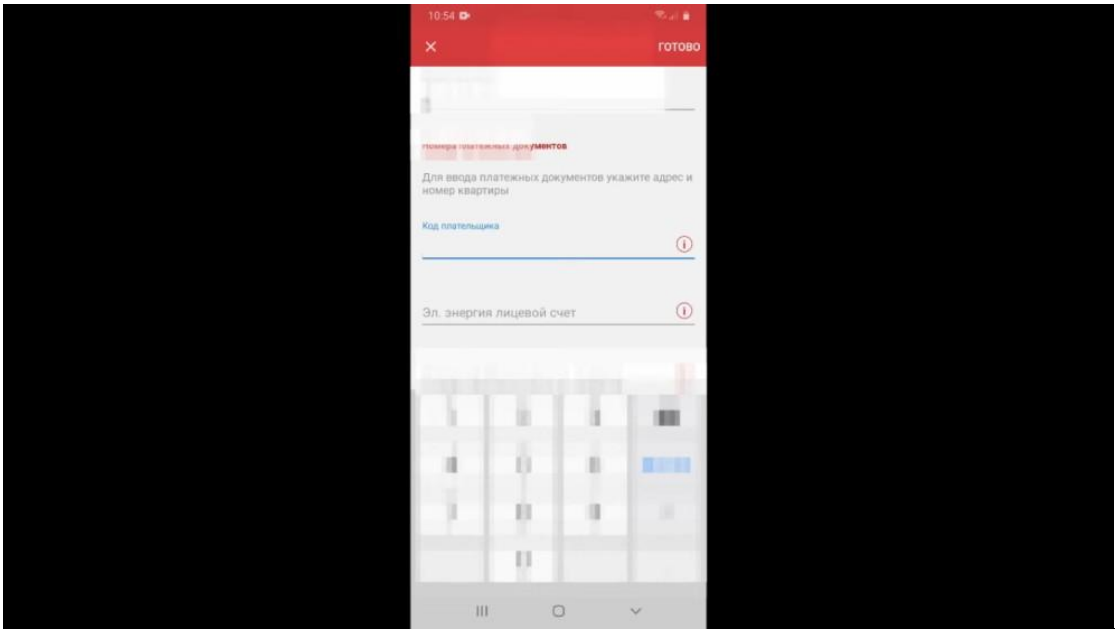



Рисунок 14 – Окно заполнения данных плательщика

Для того чтобы проверить задолженность или отправить показания за текущий месяц нужно перейти во вкладку «учёт воды» (рисунок 15).

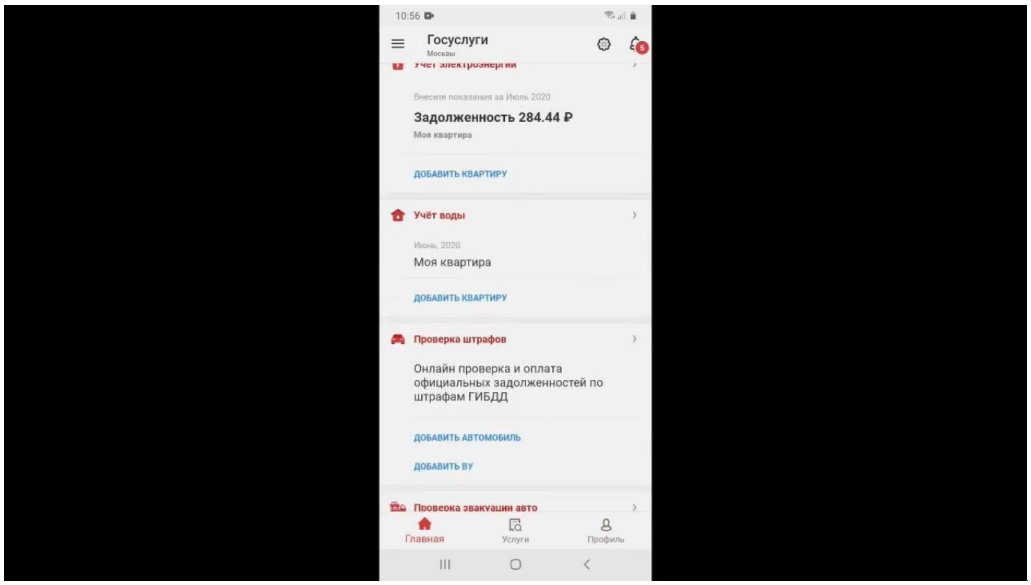



Рисунок 15 – Вкладка "главное"

В открывшемся окне мы можем отправить показания счётчиков горячей и холодной воды, а в нижней части просмотреть статистику потребления воды за прошлые месяцы (рисунок 16).

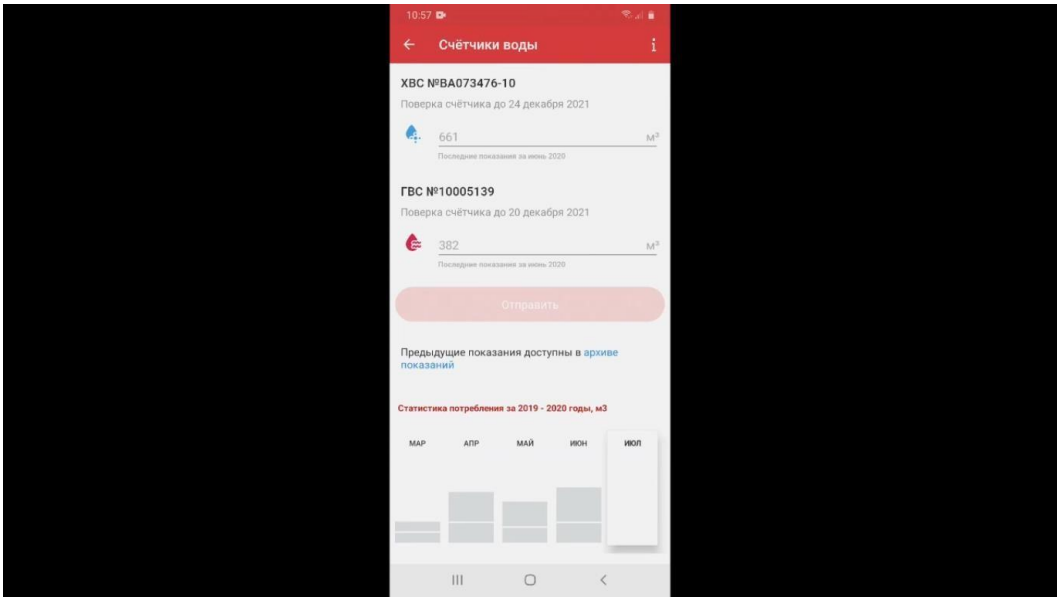



Рисунок 16 – Окно "счётчики воды"

### 1.2.3 Сервис «ГИС ЖКХ»

Для использования сервиса «ГИС ЖКХ» нужно на странице «объекты управления» заполнить поля «вид прибора» в разделе «общие сведения» (рисунок 17) и все поля в разделе «местоположения» (рисунок 18).

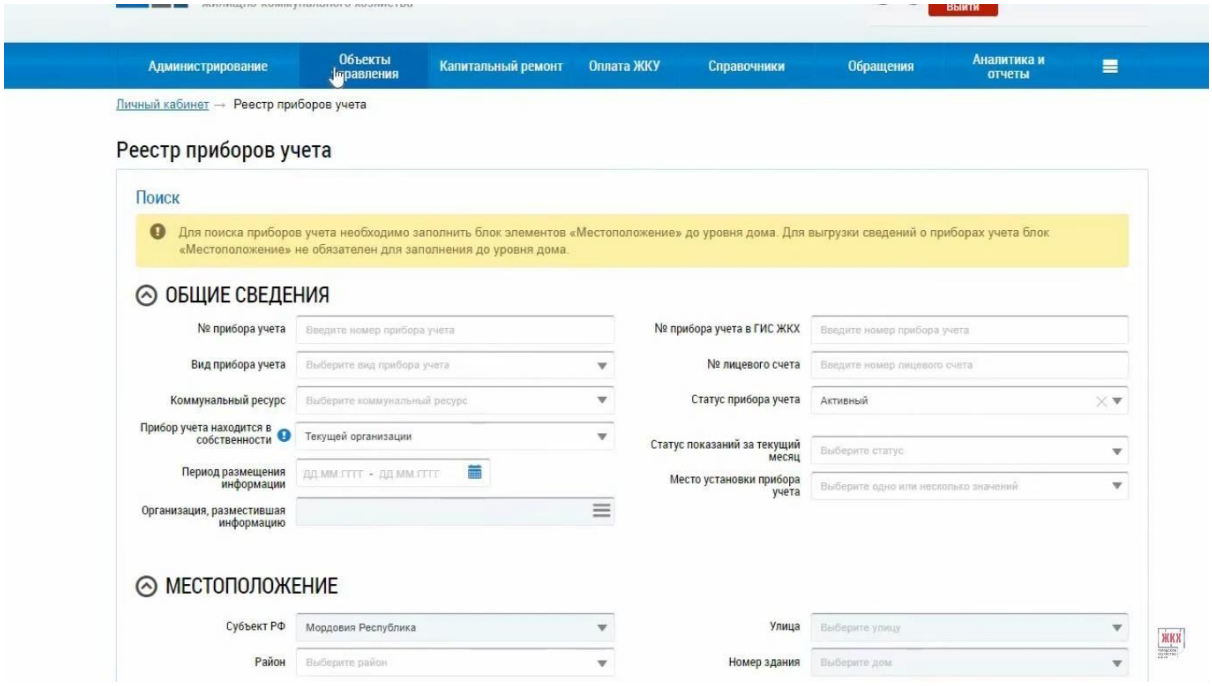



Рисунок 17 – Вкладка "объекты управления" раздел «общие сведения»

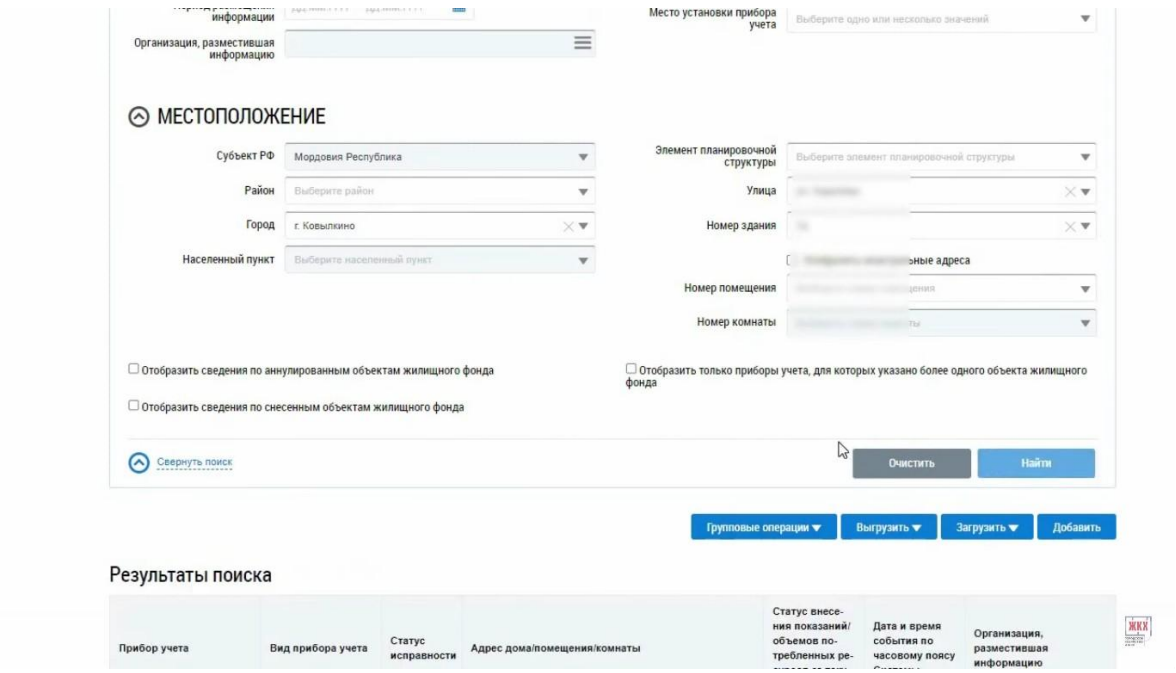



Рисунок 18 – Вкладка "объекты управления" раздел "Местоположение"

Далее нужно нажать по кнопке «найти» и в результатах поиска появятся все найденные приборы учёта (рисунок 19). Для внесения показания выбираем прибор и нажимаем «внести показания».

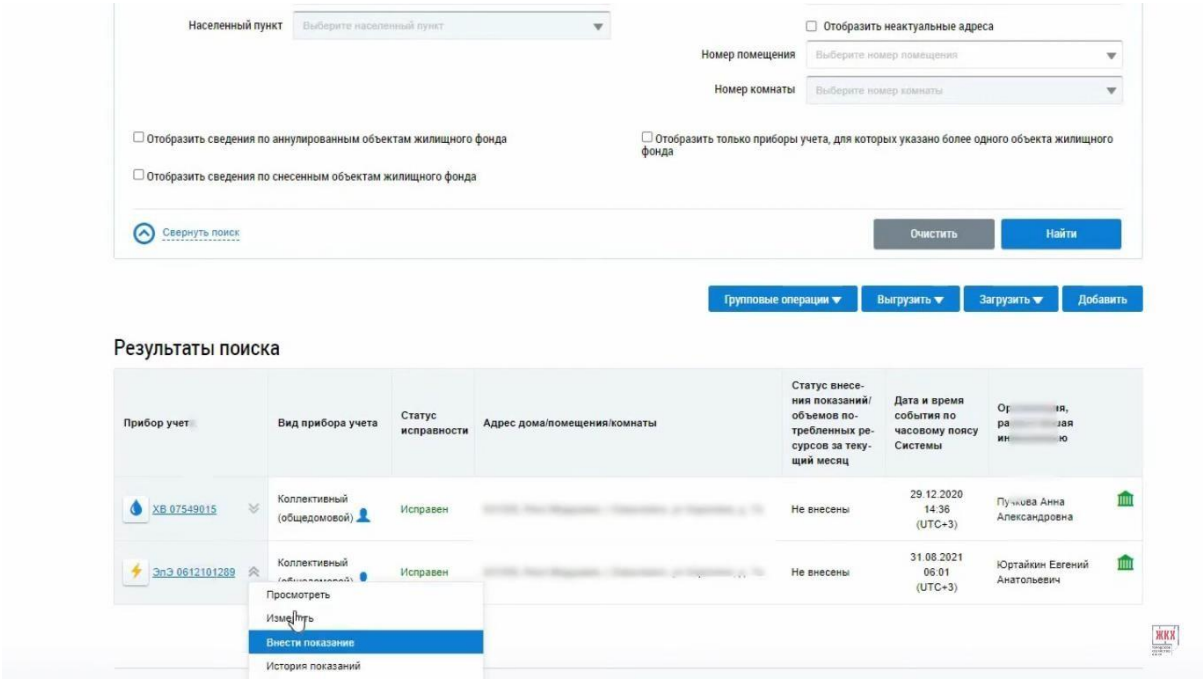



Рисунок 19 – Результаты поиска приборов

После выбора появится окно для внесения показаний (рисунок 20).

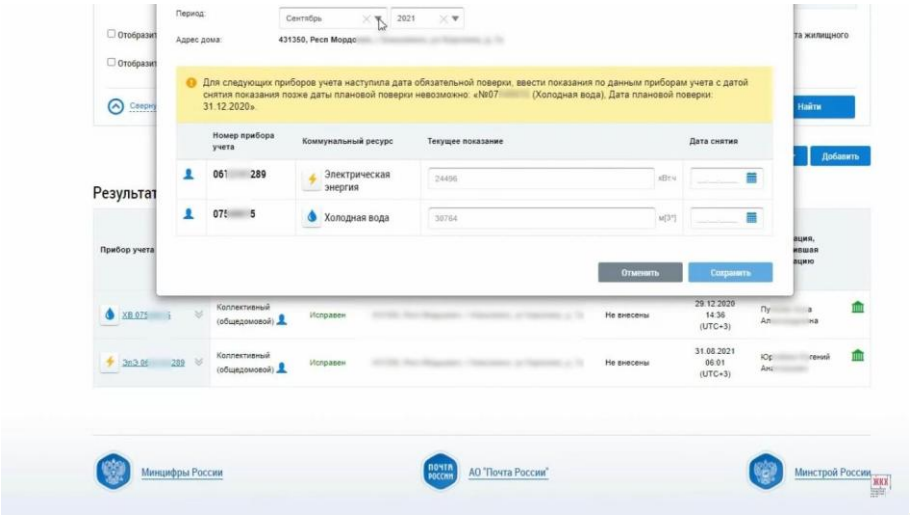



Рисунок 20 – Окно внесения показаний

Для того чтобы посмотреть показания за прошлые месяцы нужно выбрать историю показаний (рисунок 21).

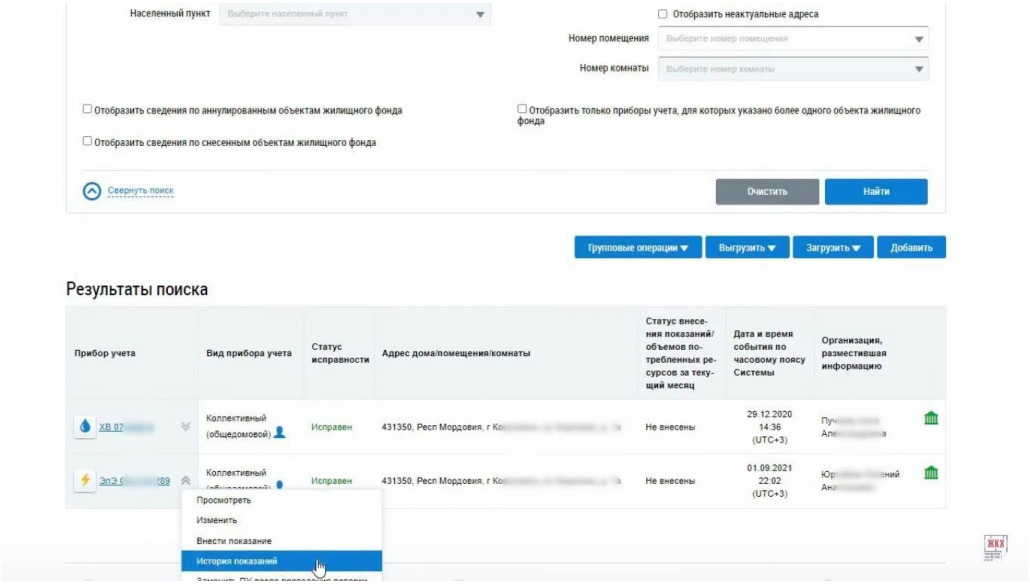



Рисунок 21 – История показаний

Откроется окно, где будет указано количество израсходованных ресурсов, дата размещения и снятия (рисунок 22)

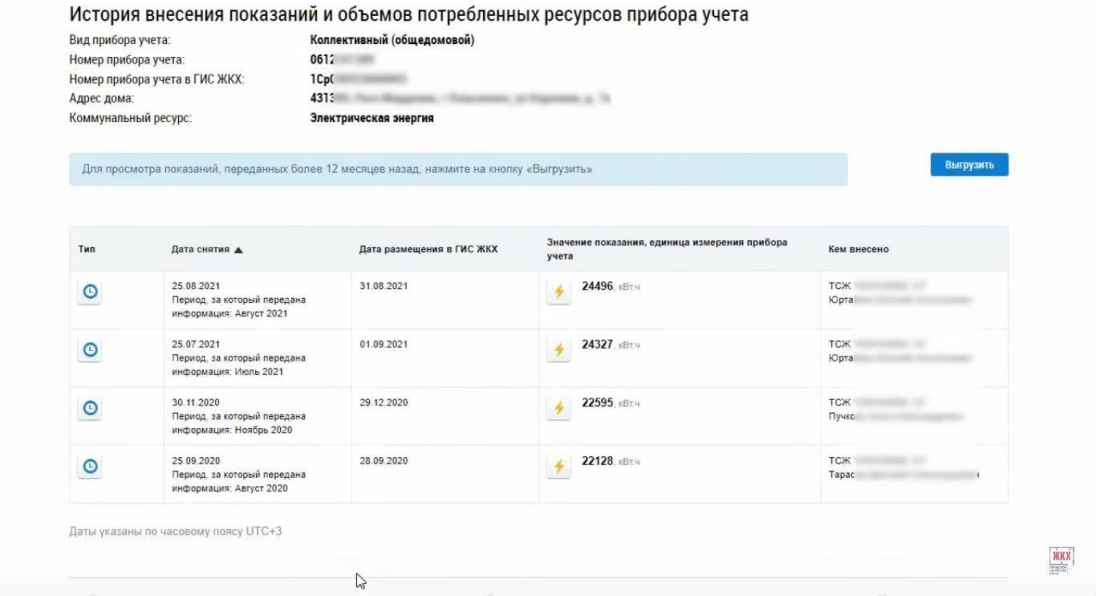



Рисунок 22 – История внесения показаний

### 1.2.4 Вывод

Результаты анализа существующих аналогов можно представить в таблице 1, показывающие минусы и плюсы рассматриваемых приложений и онлайн-сервисов.

Таблица 1 – Достоинства и недостатки аналогов

| Функционал | mos.ru | квартплата онлайн | ГИС ЖКХ |
|---|---|---|---|
| Авторизация | + | + | - |
| Возможность просмотра показаний за прошедшие месяцы | + | + | + |
| Возможность просмотра суммы оплаты за прошедшие месяцы | + | + | - |
| Возможность указать расход за текущий месяц в любой месяц | - | - | - |
| Доступ для всех жителей РФ | - | + | + |

На основании проведённого анализа было принято решение сделать систему учёта показаний счётчиков с помощью телеграм-бота. Данная система позволит проводить учёт с любого устройства в любое время.

## 1.3 Постановка задачи

Необходимо разработать телегам-бота, который будет вносить изменения показаний счётчиков и затраченных средств в базу данных и возвращать эти данные по запросу пользователя.

В данной программе будет реализована авторизация по номеру договора для получения данных определённого клиента.

Используя данное приложение, пользователь(кандидат) сможет:

– авторизироваться;

– отправлять показания за текущий месяц;

– просматривать показания за прошедшие месяцы;

– просматривать сумму затраченных средств за прошедшие месяцы.

Все эти возможности отражены на UML-диаграмме вариантов использования (рисунок 23).

Рисунок 23 – Диаграмма вариантов использования

В ходе постановки задачи были оставлены следующие технические требования.

Для использования системы и её корректной работы указываются следующие требования:

– наличие на устройстве доступа к интернету (мобильных данных или Wi-Fi);

– наличие телеграм мессенджера на устройстве (мобильного приложения, ПК версии или браузерной).

Технические требования для ПК:

– процессор с тактовой частотой 800 MHz или более мощный;

– оперативная память 2ГБ или больше;

– свободное место на жёстком диске от 170 Мб;

– архитектура с разрядностью 32 бит или 64 бит (x86 или x64);

– операционная система Windows XP, Windows Vista, Windows 7, Windows 8, Windows 10.

Технические требования для мобильных устройств:

– операционная система: iOS(12.0) или Android (4.1 и выше);

– оперативная память: 2 ГБ;

– свободное место от 170 Мб.

## 1.4 Выбор технологий и средств разработки приложения

### 1.4.1 Архитектура приложения

Выбор архитектуры приложения важен при создании программного продукта, поскольку правильно созданная архитектура позволяет вносить изменения с минимальными затратами ресурсов.

При выборе архитектуры приложения было принято решение использовать клиент-серверную архитектуру. Для реализации серверной части было принято решение разработать backend-систему бота, отвечающая за обработку полученных сообщений от клиента и взаимодействием с базой данных. Обмен сообщений между серверной и клиентской частью будет осуществляться с помощью TelegramBotApi [3]. На сервер Телеграмма переодически будут поступать http-запросы на проверку обновления чата. В случае изменеиния чата в нашу backend-систему будут приходить http-ответы. Разработанная архитектура будущего приложения представлена на рисуноке 24.

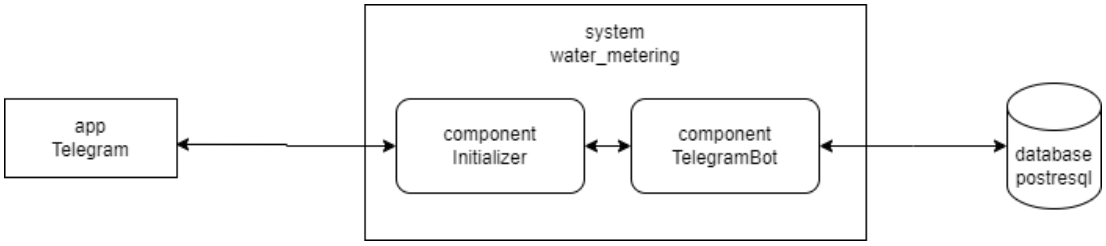



Рисунок 24 – Архитектура программы

Telegram – приложение, используемое в качестве клиента. Отображает полученные данные и отправляет запросы на их получение.

Water_metering – серверная часть приложение, обрабатывающее запросы от клиента.

Initializer – компонент серверной части, выполняющий регистрацию бота в телеграме.

TelegramBot – компонент серверной части, содержащий основную бищнес-логику приложения: обращение в БД, сортировку и фильтрацию данных.

База данных postgresql – база данных, в которой хранятся данные показаний счётчиков, номера договоров, логины и пароли пользователей. Диаграмма сущностей представлена на рисунке 25 .

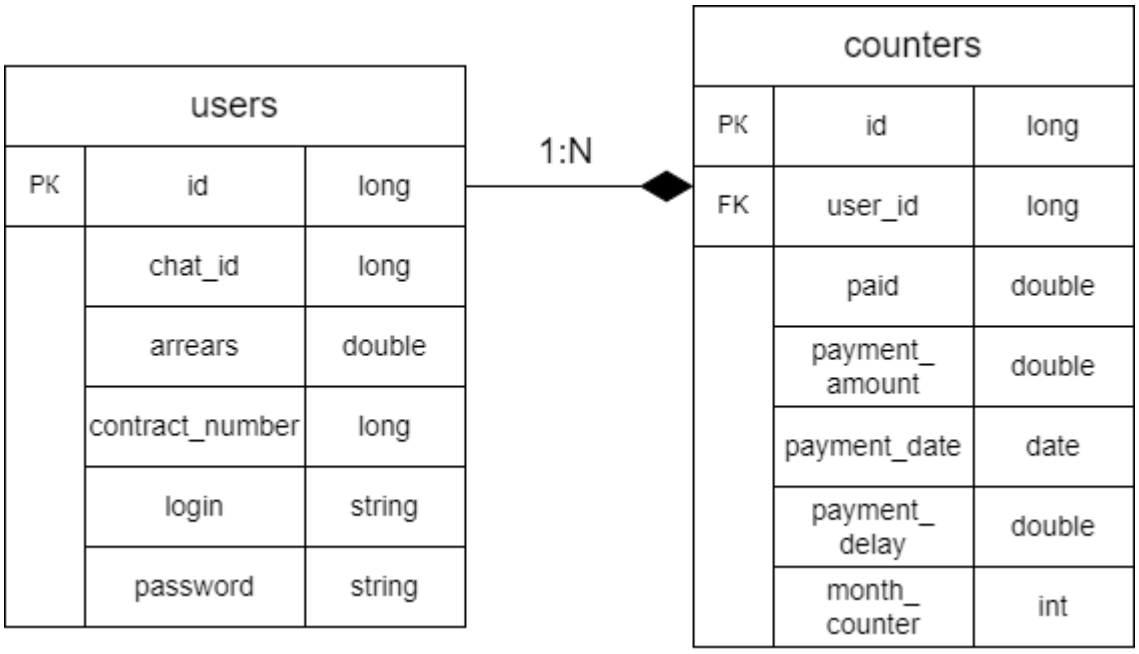



Рисунок 25 – Диаграмма сущностей

Таблица пользователя представляет сущность в которой будут храниться поля с информацией о показателях за текущий месяц, а также данные для авторизации:

– идентификатор пользователя id. Присваивается автоматически;

– id_readings. Содержит внешний ключ на id показания в таблице показаний. Присваивается автоматически;

– counter. Поле, содержащее данные показания за текущий месяц. Задаётся пользователем;

– payment. Поле, содержащее оплаченную сумму за текущий месяц, Задаётся пользователем;

– paymentDate. Поле, содержащее дату платежа за текущий месяц. Получается автоматически;

– arrears. Поле, содержащее наличие задолженности. Проверяется системой и в случае отсутствия показаний на расчётный день система высчитывает штрафную сумму и заполняет данное поле;

– ContractNumber. Поле, содержащее номер договора, по которому предоставляются услуги. Заполняется пользователем;

– login. Поле содержащее логин пользователя. Поле заполняется при регистрации пользователем и отправляется серверу для проверки авторизации;

– password. Поле содержащее пароль пользователя. Поле заполняется при регистрации пользователем и отправляется серверу для проверки авторизации.

Таблица показаний хранит в себе показания счётчиков за прошедшие месяцы:

– counterId. Поле, содержащее уникальный идентификатор показания. Заполняется автоматически;

– monthlyCounter. Поле, содержащее идентификатор пользователя. Задаётся автоматически;

– amountPaid. Поле, содержащее оплаченную сумму за месяц. Получается в первый день месяца из таблицы пользователя;

– paymentDate. Поле, содержащее дату платежа. Получается в первый день месяца из таблицы пользователя;

– delay. Поле, содержащее сумму начисленных пене за не уплату за определённый месяц. Заполняется системой в случае неуплаты или несвоевременной уплаты предоставляемых услуг.

Система проверяет, если месяц показания за текущий месяц меньше текущего, то система получает данные из таблицы пользователя и добавляет новую запись в таблицу показаний.

Система учёта счётчиков предоставляет несколько действий для пользователя: Регистрация, внесение показаний, получение показаний за предыдущие месяцы. Для начала работы с ботом требуется регистрация. Для этого требуется отправить команду /registration. После этого вводим логин, номер договора и пароль. Если номера договора или логина в БД нет, то система сообщает нам об успешной регистрации. После этого можно выполнить авторизацию и пользоваться ботом. Для авторизации нужно отправить боту команду /autorizate и ввести логин и пароль. Если система нашла эти данные в БД, то для нас будут доступны команды /getcounter, /getcounters, /setcounter.

/getcounter – команда для получения текущий показаний. Предоставляет данные полей counters, payment и paymentDate из таблицы пользователя.

/getcounters – команда для получения текущий показаний. Предоставляет данные полей monthlyCounter, amountPaid и paymentDate из таблицы показаний.

/setcounter – устанавливает значения в поле counters таблицы пользователя.

Последовательность обмена сообщений представлена на рисунке 26.

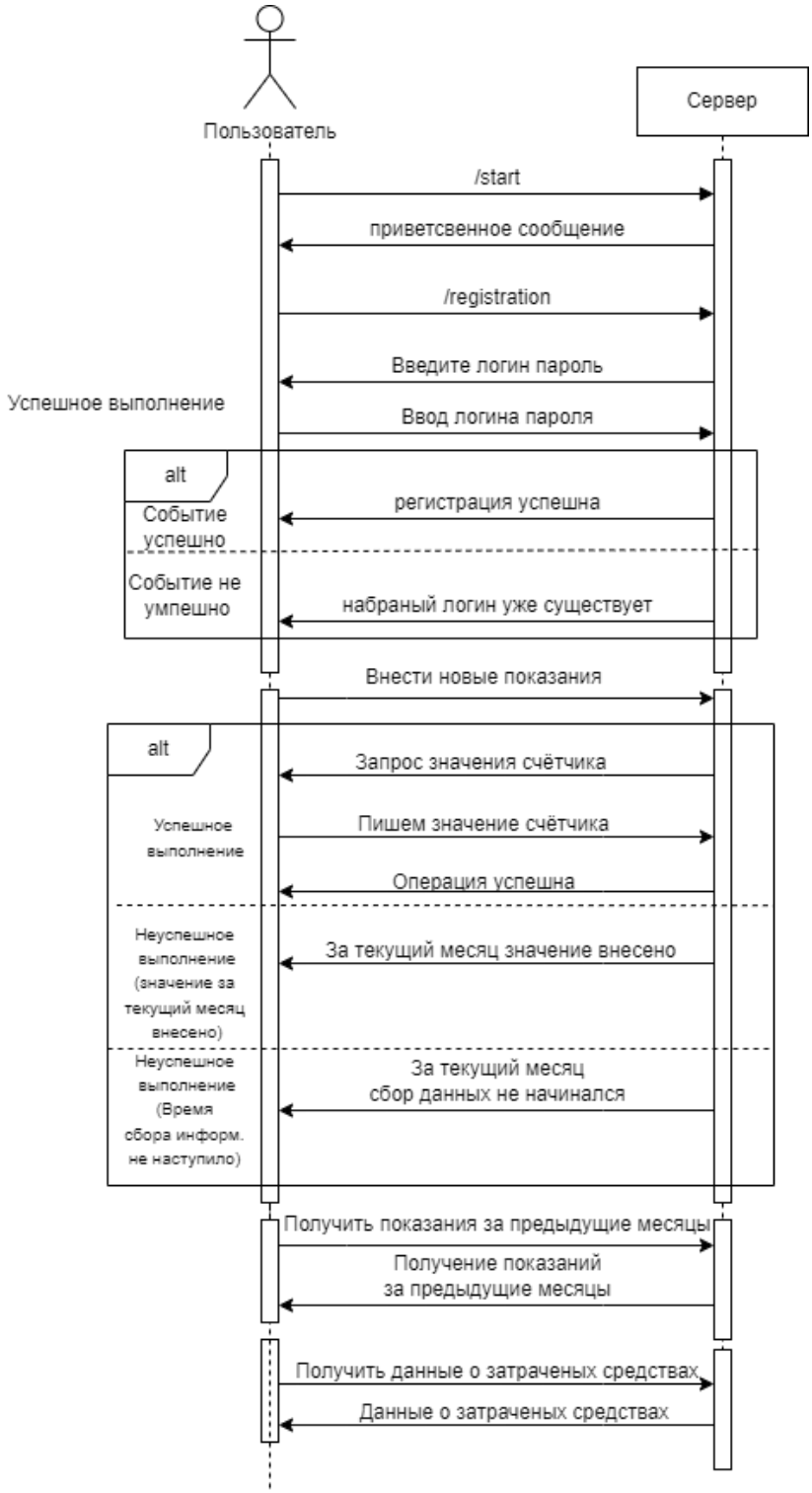



Рисунок 26 – Диаграмма последовательности для зарегистрированного пользователя.

### 1.4.2 Выбор среды разработки

На данный момент существует множество сред разработки для backend-приложений, но наиболее популярными является INTELLIJ IDEA.

Intellij idea – интегрированная среда разработки программного обеспечения для многих языков программирования, в частности одного из самых популярных языков Java. Среди возможностей можно выделить то, что среда совместима со многими популярными свободными инструментами разработки, такими как CVS, Subversion, Apache Ant, Maven и JUnit. Также Позволяет использовать систему контроля версий Git. Позволяет настроить нужное окружение. Для запуска тестов в IDE, достаточно докачать специальный плагин Java Devkit. Возможность кастомизировать рабочую панель. Из недостатков можно выделить тяжеловесность этой IDE. Для выполнения даже небольших правок могут потребоваться значительные ресурсы, поэтому если нужно выполнить какую-то простую и быструю задачу, удобнее использовать более легкий редактор.

Одним из самых популяных фреймфорков на данный момент является Spring boot. Spring – набор взаимосвязанных мини-фреймворков, созданных для работы над разными частями приложения. Фреймворк предоставляет огромный набор инструментов и библиотек, которые упрощают и ускоряют процесс разработки, позволяя сосредоточиться на бизнес-логике приложения.

Для автоматизации сборки приложения используется maven. Maven — это инструмент для автоматической сборки проектов на Java и других языках программирования. Он помогает разработчикам правильно подключить библиотеки и фреймворки, управлять их версиями, выстроить структуру проекта и составить к нему документацию.

1.4.3 Выбор базы данных

Для хранения данных требуется выбрать БД, которая будет соответствовать всем требованиям системы. Для сравнения были выбраны наиболее используемые БД- это SQL, PostgreSQL, и MySQL.

Одним из основных различий между PostgreSQL и MySQL является поддержка стандарта SQL. PostgreSQL более строго следует стандартам SQL, что делает его более мощным и гибким для сложных запросов и операций с данными. MySQL более прост в использовании и может использоваться для небольших проектов. Помимо стандартов SQL, PostgreSQL и MySQL также имеют различные функции и возможности. Выделим основные достоинства и недостатки.

SQL Server.

Достоинства:

– интеграция с экосистемой Microsoft;

– широкий функционал;

– поддержка и обновления.

Недостатки:

– высокая стоимость;

– зависимость от ОС. Система работает только с Windows.

PostgreSQL.

Достоинства:

– открытый код;

– полная совместимость с ANSI SQL;

– гибкость и расширяемость.

Недостатки:

– менее обширная поддержка и документация по сравнению с коммерческими БД;

– низкая производительность по сравнению с коммерческими БД .

MySQL.

Достоинства:

– простота использования;

– высокая производительность;

– широкое сообщество и документация.

Недостатки:

– ограниченные функциональные возможности;

– ограниченные ресурсы для масштабирования.

Поскольку для приложения требуется недорогое, гибкое и расширяемое обеспечение было принято решение использовать PostgreSQL, которая отвечает всем требованиям.

Основные возможности PostgreSQL:

– гибкий доступ к базам данных, их организация и хранение;

– управление записями в базах данных: создание, редактирование и удаление, обновление версий и так далее;

– просмотр нужной информации из базы по запросу;

– отправка транзакций, последовательных запросов, собранных в подобие скрипта;

– настройка и контроль доступа к той или иной информации, группировка пользователей по уровню прав;

– контроль версий и организация одновременного доступа к базе из разных источников;

– защита информации от возможных утечек и потерь;

контроль состояния базы в целом.

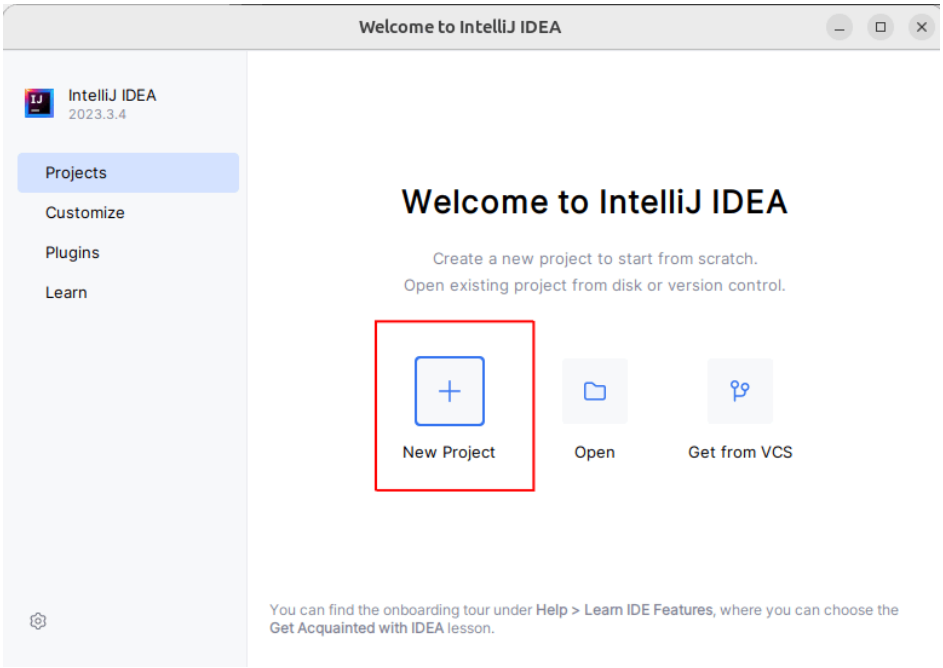
# 2 Реализация приложения

## 2.1 Настройка среды разработки

Для создания нового проекта в IntelliJ IDEA необходимо выполнить следующие действия:

– на стартовом окне Intelej IDEA выбрать "New Project" (Новый проект) из меню (рисунок 27);

– выбрать тип проекта: в открывшемся окне выбрать "Maven" [4], указать название проекта, расположение и указать версию JDK и нажать «Create» (Рисунок 28).




Изображение 27 – Стартовое окно Intelej idea

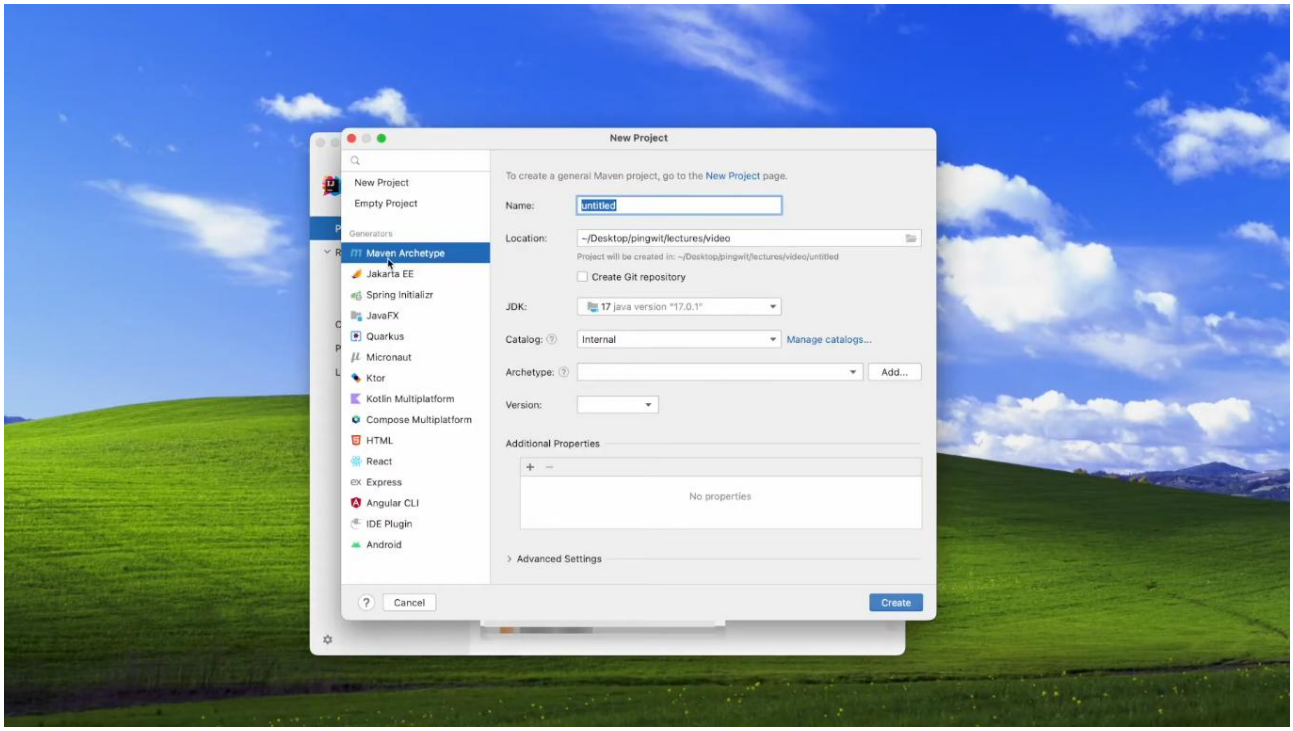



Рисунок 28 – Параметры нового проекта

В левом окне среды разработки, находится дерево файлов проекта, где находятся все основные папки и файлы проекта (рисунок 29).

В дереве файлов проекта заключены следующие модули:

– в папке com.example.metringsystem расположены основные классы, содержащие логику приложения;

– в папке resources расположены файлы настроек проекта, содержащие параметры подключения БД;

– в папке test находятся файлы для проведения тестирования проекта;

– файл pom.xml хранящийся в корне проекта содержит информацию о проекте и различных деталях конфигурации, которые используются Maven для создания проекта.

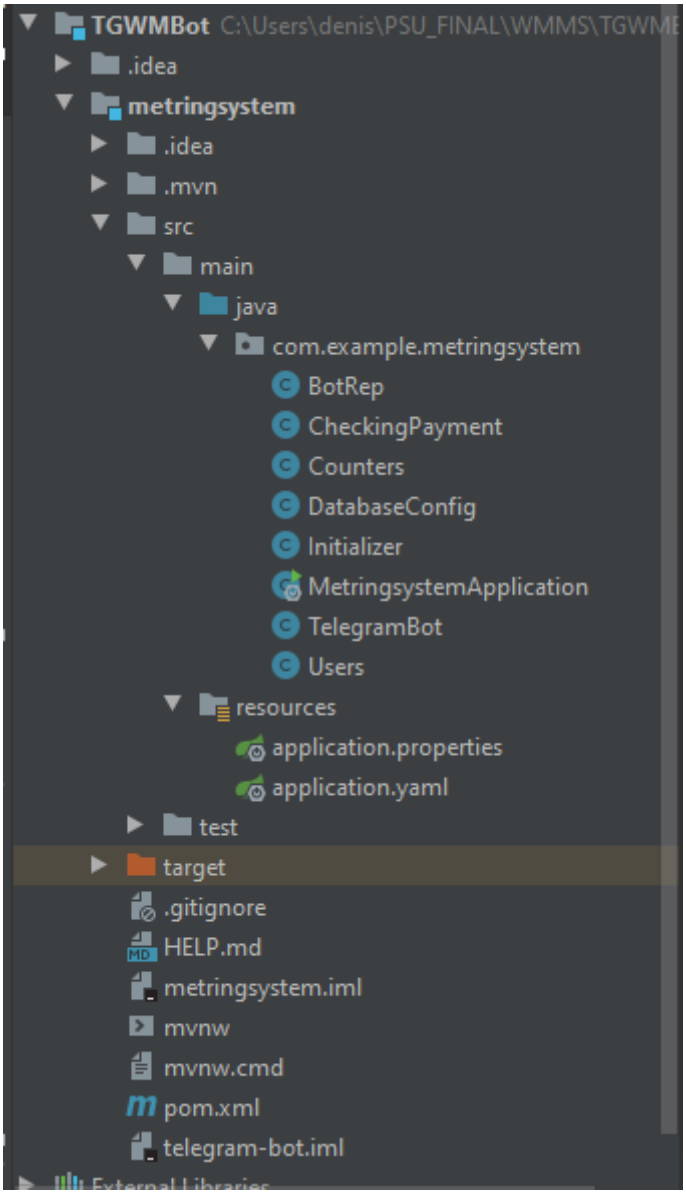



Рисунок 29 – Дерево файлов проекта

Окно среды разработки содержит следующие компоненты (рисунок 30):

– в верхней части меню навигации и настройки среды разработки;

– слева менеджер ресурсов, структуры и дерева проекта;

– в нижней части располагаются окна панель логов компиляции;

– справа окна настройки maven и БД;

– в центре основное рабочее пространства проекта.

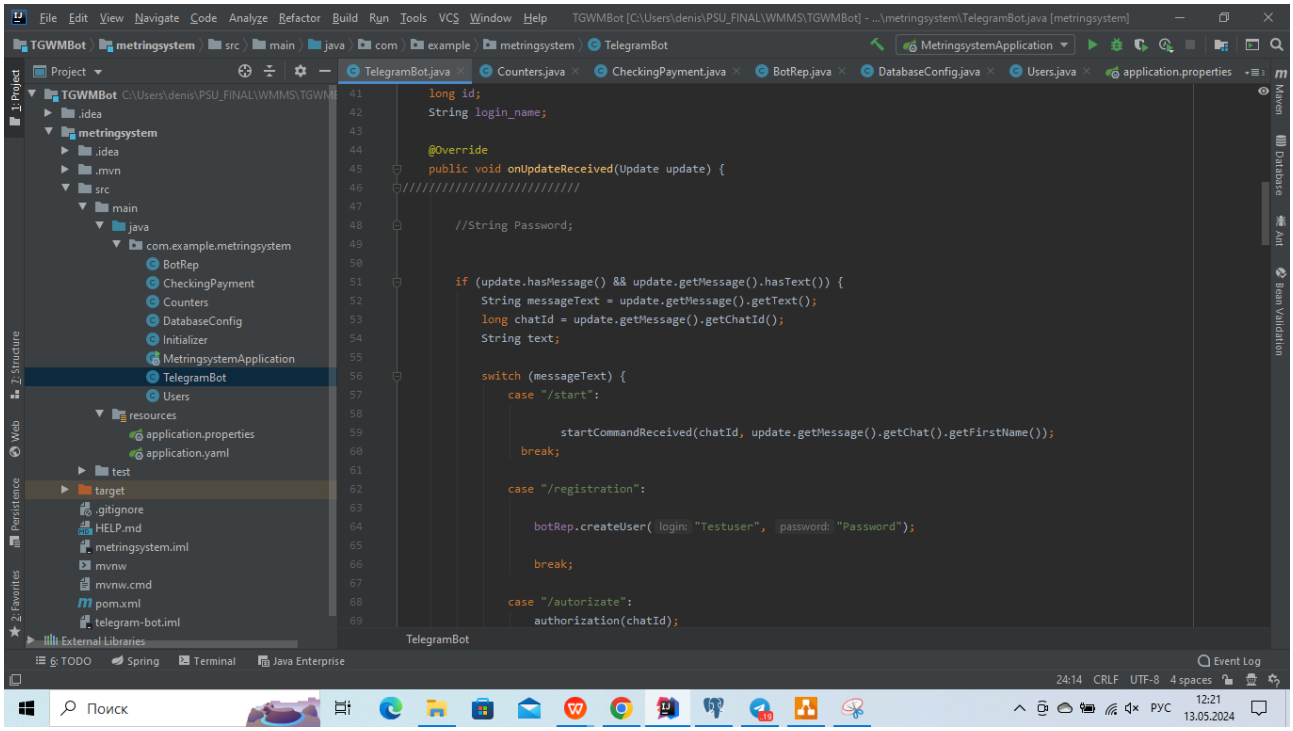



Рисунок 30 – Рабочее окно среды разработки

Таким образом, IntelliJ IDEA обеспечивает удобное и эффективное разработку программного обеспечения благодаря своим продвинутым инструментам, интуитивно понятному интерфейсу и широким возможностям настройки проектов

## 2.2 Разработка БД

Для удобства работы с базой данных будет использоваться приложение с графической оболочкой pgadmin4.

pgAdmin 4 – многофункциональное приложение, предназначенное для управления и разработки базы данных PostgreSQL. С помощью него нами будет управляться БД, выполняться запросы и проводится проверка данных при тестировке. Также приложение имеет ряд возможностей для взаимодействия с базами данных PostgreSQL.

Для создания БД открываем приложение pgadmin4 и подключаемся к серверу (рисунок 31).

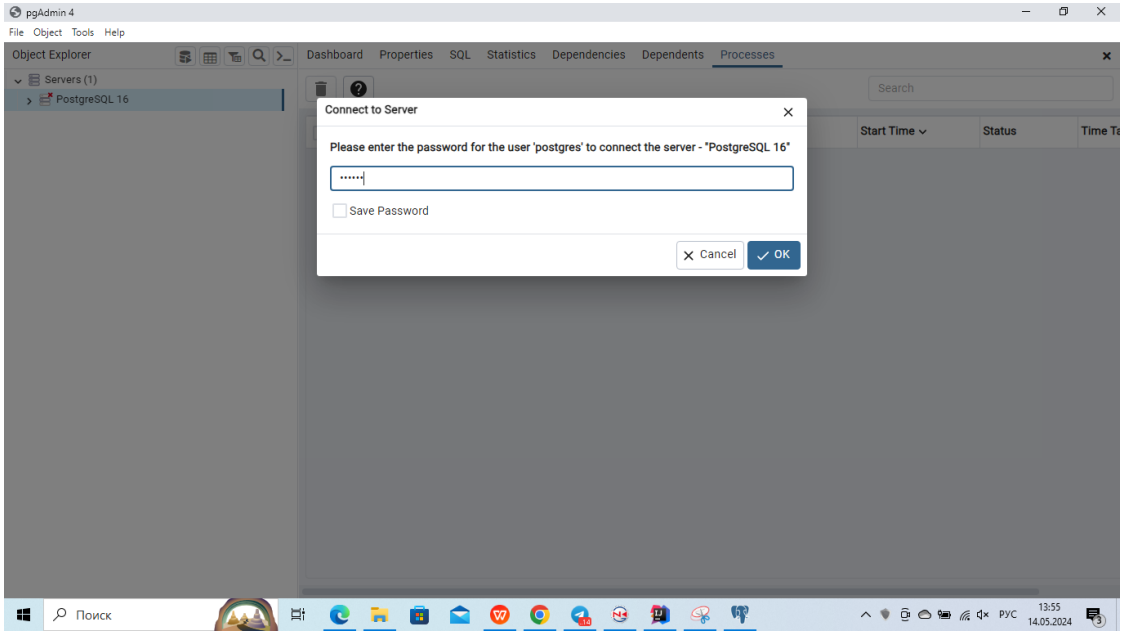



Рисунок 31 – Подключение к серверу

Вводим пароль и создаём базу данных на сервере (рисунок 32).

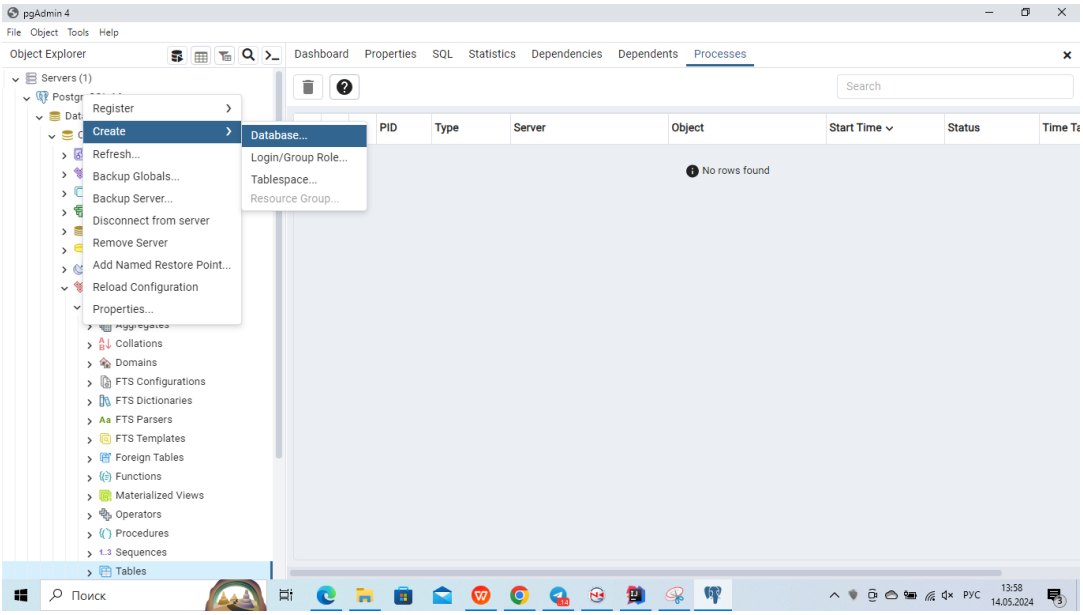
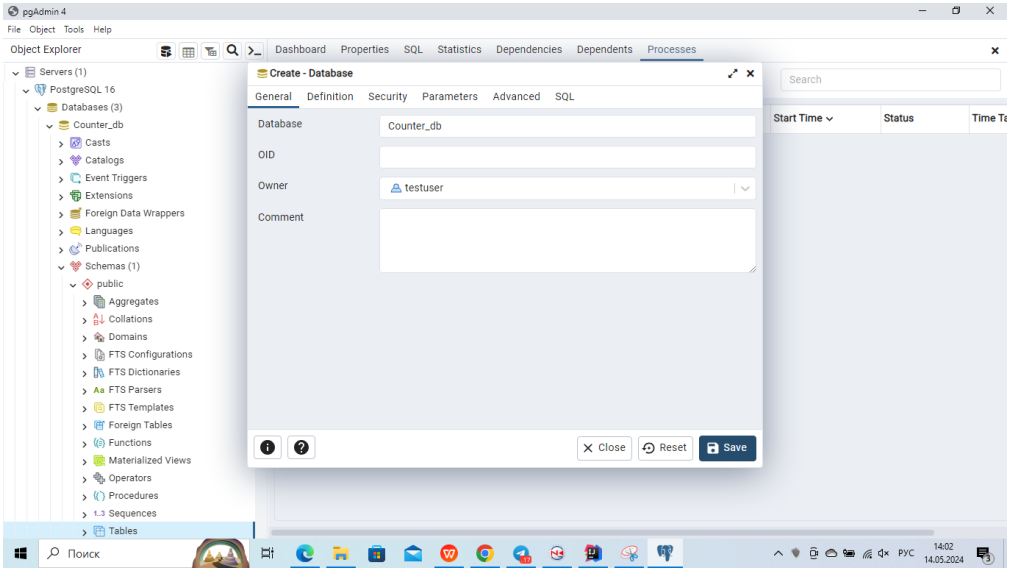



Рисунок 32 – Создание базы данных

Вводим параметры базы (Название, имя владельца) и нажимаем «save» (рисунок 33).




Изображение 33 – настройка бд

После этого созданная нами БД появится в структуре БД слева (рисунок 34).

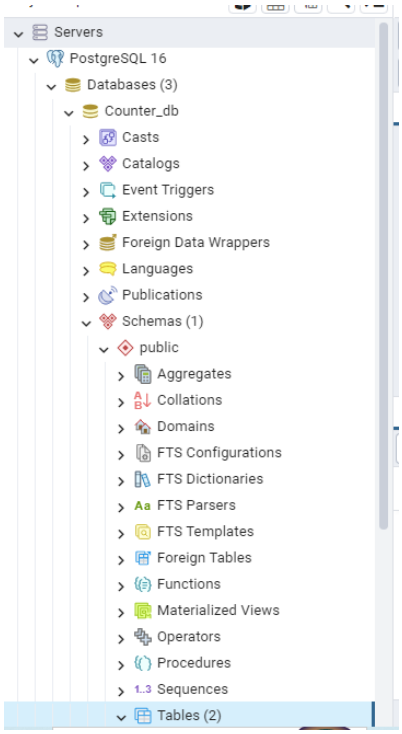



Рисунок 34 – Структура базы данных

В соответствии с диаграммой сущности (рисунки 3 – 5) требуется создать 2 таблицы: users и counters.

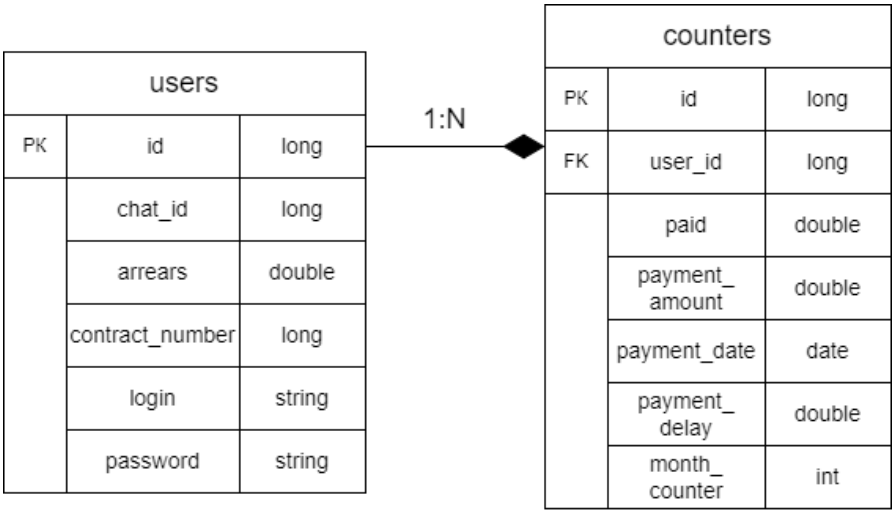



Рисунок 35 – Диаграмма сущностей

Для создания требуемых таблиц используем следующие команды.

Для создания таблицы users.

```
create table users
(
    id bigserial  primary key,
    arrears  double precision,
    chat_id  bigint  constraint uk_nr2rmfhq6wfp39vcduy7iketb
unique,
    contract_number bigint,
    login    varchar(255)  constraint
uk_ow0gan20590jrb00upg3va2fn         unique,
    password    varchar(255)
);
```

Для создания таблицы counters.

```
create table counters
(
    id bigserial
    primary key,
    month_counter  integer,
    paid           double precision,
    payment_amount double precision,
    payment_date   date,
    payment_delay  double precision,
    user_id        bigint
);
```

В результате готовая БД имеет следующую структуру (рисунок 36):

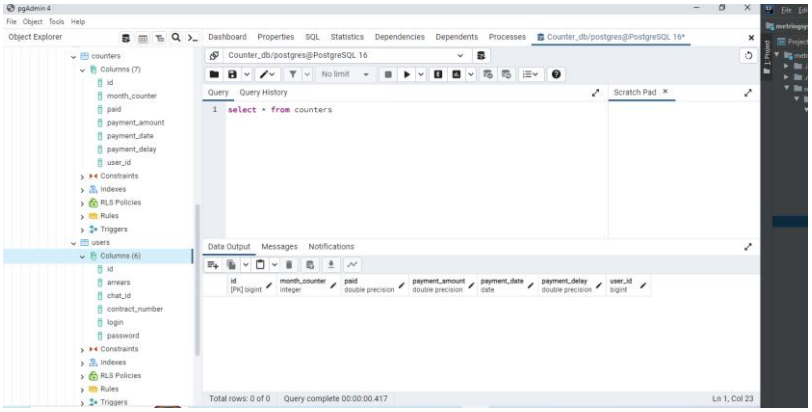



Рисунок 36 – Структура готовой базы данных

## 2.3 Реализация телеграм бота

Для получения команд от пользователя через телеграм-приложение в логике используются телеграмный API telegramBotsApi [5]. API предоставляет библиотека методов для работы в телеграм ботом – telegrambots. Для работы с библиотекой telegrambots подключаем зависимость telegrambots (рисунок 37).

```
<!-- TelegramBots API dependency -->
<dependency>
    <groupId>org.telegram</groupId>
    <artifactId>telegrambots</artifactId>
    <version>5.6.0</version>
</dependency>
```

Рисунок 37 – Подключение зависимости telegrambots

После подключения зависимости [6] создали класс TelegramBot для получения сообщений из telegram. Класс[7] наследует методы TelegramLongPollingBot. В нём содержаться методы для получения токена и имени бота, получения команды для требуемого события, а также методы для получения сообщений пользователя через telegram API.

Метод onUpdateReceived принимает объект update, в котором находятся такие параметры как: идентификатор чата, имя пользователя, сообщение пользователя. В данном методе приложение принимает команды пользователя в конструкции switch case и в каждом кейсе вызывает требуемый метод.

Для регистрации используется команда registration, в кейсе вызывается метод registrationInit, который создаёт в БД новую запись о созданном пользователе:

```
 case "/registration":

 registrationAction.registrationInit(update.getMessage());

 break;
```

registrationInit инициализирует процесс регистрации, создаёт объект UserDTO и выводит информацию о регистрации:

```
public void registrationInit(Message msg){

UserDTO newUser = cacheMapRegistration.computeIfAbsent(msg.getChatId(), u-> new UserDTO());

displayRegister(msg, newUser);

}
```

displayRegister – выводит информацию о регистрации пользователю и проверяет получение данных от пользователя.

```
 private void displayRegister(Message msg, UserDTO user) {
        StringBuffer sb = buildRegistrationInfo(user);
        // check if a user is fully filled
        boolean complete = (
                Strings.isNotEmpty(user.getLogin()) &&
                Strings.isNotEmpty(user.getPassword()) &&
                user.getContractNumber() > 0
        );

        InlineKeyboardMarkup markup = registrationButtonMarkup(complete);
        sendTextWithMarkup(msg.getChatId(), sb.toString(), markup);
```

```
}
```

В классе BaseAction находятся методы для основных действий: отправка сообщений, редактирование сообщений, удаляет сообщение из чата.

Метод sendOnlyTextToUser проверяет, зарагистрирован ли пользователь и если зарегистрирован, то присылает сообщение с результатом действия.

```
    public boolean sendOnlyTextToUser(Long userId, String text) {
        if(userId == null)
            return false;

        SendMessage s = new SendMessage();
        s.setChatId(userId.toString());
        s.disableWebPagePreview();
        s.setParseMode("Markdown");
        s.setText(text);

        try {
            getBot().execute(s);
            System.out.println(userId+": " +text);
            return true;
        } catch (TelegramApiException e) {
            e.printStackTrace();
        }

        return false;
    }
```

Метод sendTextWithMarkup используется для выбора сортировки показаний на текущие и все прошлые. Метод выводит сообщение с клавиатурой, где пользователь может выбрать дальнейшие действие.

```
    public void sendTextWithMarkup(Long chatId, String text,
InlineKeyboardMarkup replyMarkup) {
        SendMessage s = new SendMessage();
        s.setChatId(chatId.toString());
        s.enableMarkdown(true);
```

```
        s.setText(text);
        s.setReplyMarkup(replyMarkup);

        try {
            getBot().execute(s);
            System.out.println(chatId+": " +text);
        } catch (TelegramApiException tae) {
            tae.printStackTrace();
        }
    }
```

Метод sendEditTextWithMarkup используется для редактирования сообщения, содержащее данные регистрации. При успешном заполнении логина или пароля метод вызывается и изменяет статус заполнения данных (отмечает заполненое поле галочкой).

```
    public void sendEditTextWithMarkup(Long chatId, String text,
Integer messageId, InlineKeyboardMarkup replyMarkup) {
        EditMessageText em = new EditMessageText();
        em.setChatId(chatId.toString());
        em.enableMarkdown(true);
        em.setMessageId(messageId);
        em.setText(text);
        em.setReplyMarkup(replyMarkup);

        try {
            getBot().execute(em);
        } catch (TelegramApiException tae) {
            tae.printStackTrace();
        }
```

```
}
```

Для упрощения работы с командами используется класс ActionCommand. Он используется для парсинга и управления командами. Данный класс позволяет извлекать и проверять части команды. Метод commandRaw принимает строку. Метод Parse разбивает строку на части, геттеры и сетттеры предоставляют доступ к данным частям команды.

```
public class ActionCommand {

    private String commandRaw;
    private String command;
    private String action;
    private String params;
    private String replyTo;

    public ActionCommand(String commandRaw) {
        this.commandRaw = commandRaw;
    }

    public ActionCommand parse(){

        if(StringUtils.hasText(this.commandRaw)){
            String[] parts = this.commandRaw.split("-");

            if(parts.length > 0){
                this.command = parts[0];

                if(parts.length > 1){
                    this.action = parts[1];
                }

                if(parts.length > 2){
                    this.params = parts[2];
                }

                if(parts.length > 3){
                    this.replyTo = parts[3];
                }
```

```
                }
            }

            return this;
        }

        public String getCommand() {
            return command;
        }

        public boolean isCommand(String cmd) {
            return (command != null && command.equals(cmd));
        }

        public String getAction() {
            return action;
        }

        public boolean isAction(String act) {
            return (action !=  null &&  action.equals(act));
        }

        public String getParams() {
            return params;
        }

        public String getReplyTo() {
            return replyTo;
        }

        public Integer getReplyToInt() {
            return (replyTo!= null)? Integer.valueOf(replyTo):
null;
        }

        public boolean isCommandAndAction(String cmd, String act)
{
            return isCommand(cmd) && isAction(act);
        }

        public boolean isReplyTo() {
```

```
            return StringUtils.hasText(replyTo);
        }
    }
```

Метод startCommandReceived предназанчен для вывода начального сообщения с информацией о возможных командах бота. Он отправляет в чат команды и активирует клавиатуру для пользователя.

```
    private void startCommandReceived(long chatId, String name) {

            String answer = "Это бот Система учёта счётчиков, вот список команд, которые я могу предложить:" +

                    "\n /autorizate - авторизация"+

                    "\n /getresult - получение результата"+

                    "\n /registration - регистрация"+

                    "\n /invoice - выставить счет пользователю";

            //log.info("Replied to user " + name);

            System.out.println("Replied to user " + name);

            //sendMessage(chatId, answer);

            sendKeyboardReply(chatId, answer);

        }
```

Метод sendMessage вызывается для отправки сообщения в чат пользователя.

```
    private void sendMessage(long chatId, String textToSend) {
            SendMessage message = new SendMessage();
            message.enableMarkdown(true);
            message.setChatId(String.valueOf(chatId));
            //log.info("message: "+ textToSend);
            System.out.println("message: "+ textToSend);
            message.setText(textToSend);
```

```
            try{
                execute(message);
            }
            catch (TelegramApiException e){
                //log.error(e.getMessage());
                System.out.println(e.getMessage());
            }
        }
```

Функция sendKeyboardReply вызывается для предоставления пользователю клавиатуры и взаиможействия с ней. keyboardRow содержит информацию о клавишах и содержимом действии.

```
    public void sendKeyboardReply(Long chatId, String text){

            SendMessage s = new SendMessage();
            s.setChatId(chatId.toString());
            s.enableMarkdown(true);
            s.setText(text);
            ReplyKeyboardMarkup replyKeyboardMarkup = new
ReplyKeyboardMarkup();
            s.setReplyMarkup(replyKeyboardMarkup);
            replyKeyboardMarkup.setSelective(true);
            replyKeyboardMarkup.setResizeKeyboard(true);
            replyKeyboardMarkup.setOneTimeKeyboard(false);

            List<KeyboardRow> keyboard = new
ArrayList<KeyboardRow>();

            KeyboardRow keyboardRow = new KeyboardRow();
            keyboardRow.add(new KeyboardButton("Сумма к оплате"));
            keyboardRow.add(new KeyboardButton("Задолженность"));
            keyboardRow.add(new KeyboardButton("Показания"));

            keyboard.add(keyboardRow);

            keyboardRow = new KeyboardRow();
            keyboardRow.add(new KeyboardButton("Добавить текущий
платеж"));

            keyboard.add(keyboardRow);
```

```
            replyKeyboardMarkup.setKeyboard(keyboard);

            try{
                execute(s);
            }
            catch (TelegramApiException e){}
        }
```

Для добавления платежа обрабатывается команда /invoice. Если messageText содержит данный текст, то условием проверяется сообщение от пользователя задданым форматом дд.мм.гггг сумма платежа. Далее в блоке программы обрабатывается несколько случаев: не верная дата заполнения, не найден пользователь, и успешное выполнение операции.

```
    if(messageText.startsWith("/invoice")){
                    String[] parts = messageText.split(" ");

                    if(parts.length>2){
                        LocalDate date =
DateUtil.parseDate(parts[1]);

                        Double paymentAmount = null;
                        try{
                            paymentAmount =
Double.valueOf(parts[2]);
                        } catch (NumberFormatException nfe){
                            sendMessage(chatId,
                                    "Не коректный формат
комманды\n\n"+
                                            "*формат*: /invoice
{payment date: dd.mm.yyyy} {payment amount: number} {user id:
number (опционально)}\n"+
                                            "*пример*: /invoice
01.05.2024 1200.30 1"
                            );
                            return;
                        }

                        Long userId = null;
                        try{
```

```
                            userId = (parts.length>3)?
Long.valueOf(parts[3]): null;
                        } catch (NumberFormatException nfe){
/*Nothing to do*/}

                        if(userId == null){
                            Optional<UserDTO> user =
botRep.getUserByChatId(chatId);

                            if(!user.isPresent()){
                                sendMessage(chatId, "Пользователь
не зарегистрирован! Пройти регистрацию /registration");
                                return;
                            }
                            userId = user.get().getId();
                        }

                        if(botRep.addInvoice(date, paymentAmount,
userId)){
                            sendMessage(chatId, "Счет выставлен
для пользователя: " + userId+ " на сумму " + paymentAmount + " на
дату "+date.toString());
                        }
                    } else {
                        sendMessage(chatId,
                                "*формат*: /invoice {payment date:
dd.mm.yyyy} {payment amount: number} {user id: number
(опционально)}\n"+
                                        "*пример*: /invoice
01.05.2024 1200.30 1"
                        );
                    }

                    return;
                }
```

## 2.4 Тестирование и отладка

Для проверки работоспособности программы было проведено тестирование программы. Был составлен и выполнен сценарий тестирования программы (таблица 1) на наличие ошибок работы приложения:

Результаты проверок функционала бота представлен на рисунках 38 – 58.

Таблица 1 – Сценарий тестирования бота

| Действие | Ожидаемый результат | Фактический результат |
|---|---|---|
| 1. Отправка команды /start | Бот пришлёт сообщение со списком команд | Совпадает с ожидаемым результат |
| 2. Нажатие на клавиши клавиатуры «сумма к оплате», «Задолженность», «показания», «Добавить текущий платёж» | Бот пришлёт сообщение с требованием о регистрации | Совпадает с ожидаемым результат |
| 3. Отправить команду /registration | Бот пришлёт сообщение с текстом «Регистрация нового ользователя» , а также текст: Login: no login<br>Password: no password Contract Number: no contract number | Совпадает с ожидаемым результат |
| 4. Отправить команду /registration | Бот пришлёт сообщение с текстом «Регистрация нового ользователя» , а также текст: Login: no login<br>Password: no password Contract Number: no contract number | Совпадает с ожидаемым результат |
| 4.1 Кликнуть по клавише Login в тексте «Регистрация пользователя» и ввести логин user<br>Предыдуший шаг№4 | Бот пришлёт сообщение с текстом Введите @логин, после ввода логина изменит статус логина в сообщении «Регистрация нового пользователя» | Совпадает с ожидаемым результат |
| 4.2 Кликнуть по клавише Login в тексте «Регистрация пользователя» и ввести логин user2 | Бот пришлёт сообщение с текстом Введите @логин, после ввода логина изменит статус логина в сообщении | Совпадает с ожидаемым результат |

| | | |
|---|---|---|
| Предыдущий шаг№4.1 | «Регистрация нового пользователя» | |
| 4.3 Кликнуть по клавише password в тексте «Регистрация пользователя» и ввести пароль user2<br>Предыдущий шаг№4 | Бот пришлёт сообщение с текстом Введите @пароль, после ввода пароля изменит статус пароля в сообщении «Регистрация нового пользователя» | Совпадает с ожидаемым результат |
| 4.4 Кликнуть по клавише password в тексте «Регистрация пользователя» и ввести пароль user2<br>Предыдущий шаг№4.3 | Бот пришлёт сообщение с текстом Введите @пароль, после ввода пароля изменит статус пароля в сообщении «Регистрация нового пользователя» | Совпадает с ожидаемым результат |
| 4.5 Кликнуть по клавише contract number в тексте «Регистрация пользователя» и ввести номер контракта user2<br>Предыдущий шаг№4 | Бот пришлёт сообщение с текстом Введите @номер контракта, после ввода номер контракта изменит статус пароля в сообщении «Регистрация нового пользователя» | Совпадает с ожидаемым результат |
| 4.6 Кликнуть по клавише contract number в тексте «Регистрация пользователя» и ввести номер контракта user2<br>Предыдущий шаг№4.5 | Бот пришлёт сообщение с текстом Введите @номер контракта, после ввода номер контракта изменит статус пароля в сообщении «Регистрация нового пользователя» | Совпадает с ожидаемым результат |
| 5. После шага №4.5 или 4.6 нажать на клавишу register | Бот пришлёт сообщение с текстом «Пользователь успешно создан» | Совпадает с ожидаемым результат |
| 6. После шага №5 выполнить повторную регистрацию пользователя | Бот пришлёт сообщение с текстом «Пользователь успешно создан» | Совпадает с ожидаемым результат |
| 7. После шага №5 выполнить повторную | Бот проигнорирует сообщение | Совпадает с ожидаемым результат |

| | | |
|---|---|---|
| ввод данных пользователя | | |
| 8. После шага №4.5 или 4.6 нажать на клавишу register | Бот пришлёт сообщение с текстом «Пользователь успешно создан» | Совпадает с ожидаемым результат |
| 9. После шага №5 выполнить повторную регистрацию пользователя | Бот пришлёт сообщение с текстом «Пользователь успешно создан» | Совпадает с ожидаемым результат |
| 10. После шага №5 выполнить повторную ввод данных пользователя | Бот проигнорирует сообщение | Совпадает с ожидаемым результат |
| 11. Нажать на клавишу «Сумма к оплате», с предварительно нулевым значением в БД. | Бот выдаст сообщение: «Записи показания за текущий месяц не найдено» | Совпадает с ожидаемым результат |
| 12. Нажать на клавишу «Сумма к оплате», с предварительно заполненным значением поля показания в БД. | Бот выдаст сообщение: «Сумма к оплате» с указанной суммой, которую требуется оплатить | Совпадает с ожидаемым результат |
| 13. Нажать на клавишу «Добавить текущий платёж», с предварительно заполненным значением поля показания в БД. | Бот выдаст сообщение: с предложением ввести сумму за текущий месяц | Совпадает с ожидаемым результат |
| 14. Ввести значение | Бот пришлёт сообщение о том, что платёж добавлен | Совпадает с ожидаемым результат |
| 15. Нажать на клавишу «Показания», с предварительно заполненным значением поля показания в БД. И выбрать Текущее, а затем предыдущее щначение из бд. | Бот выдаст сообщение: с имеющимися показаниями | Совпадает с ожидаемым результат |
| 16. Нажать на клавишу «Задолженность», с | Бот выдаст сообщение: с имеющейся | Совпадает с ожидаемым |

| | | |
|---|---|---|
| предварительно заполненным значением поля показания в БД. | задолженностью | результат |
| 17. Нажать на клавишу «Сумма к оплате», с предварительно заполненным значением поля показания в БД. | Бот выдаст сообщение: «Сумма к оплате» с указанной суммой, которую требуется оплатить | Совпадает с ожидаемым результат |
| 18. Нажать на клавишу «Показания», с предварительно заполненным значением поля показания в БД. И выбрать Текущее, а затем предыдущее щначение из бд. | Бот выдаст сообщение: с имеющимися показаниями | Совпадает с ожидаемым результат |

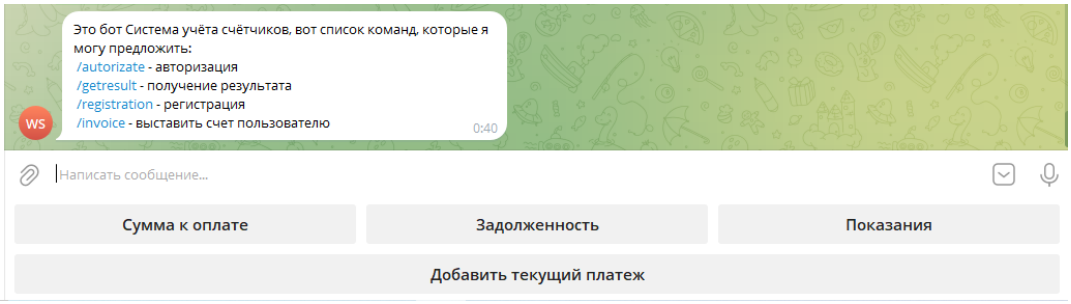
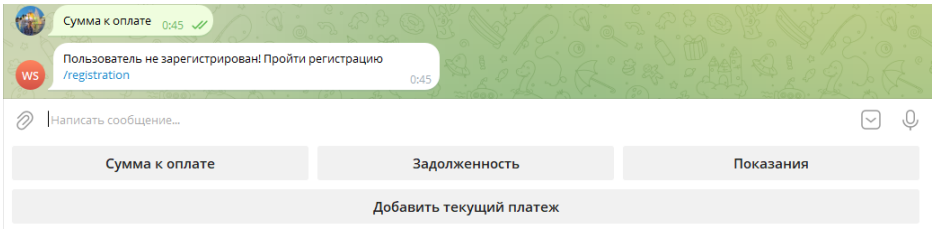



Рисунок 38 – Тестирование команды /start




Изображение 39 – Результат работы с ботом незарегистрированного пользователя

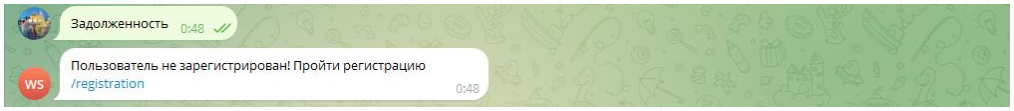



Рисунок 40 – Результат работы с ботом незарегистрированного пользователя

Рисунок 41 – Результат работы с ботом незарегистрированного пользователя

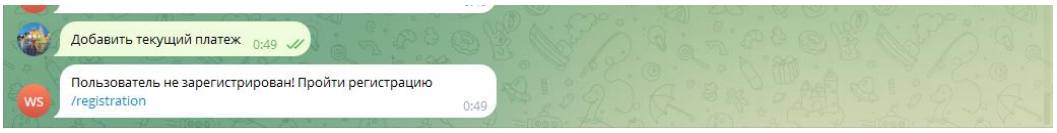



Рисунок 42 – Результат работы с ботом незарегистрированного пользователя

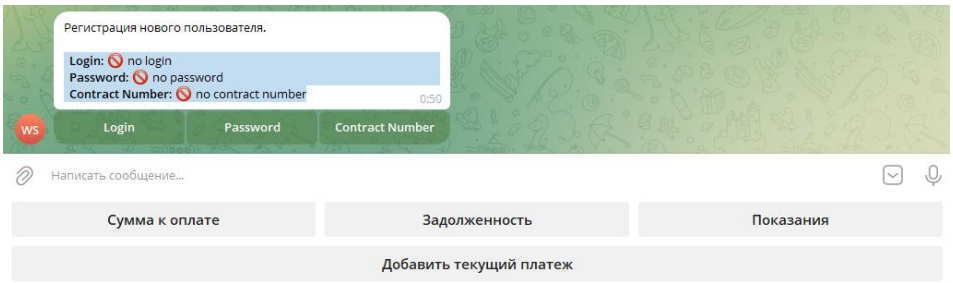



Рисунок 43 –Тестирование команды /registration

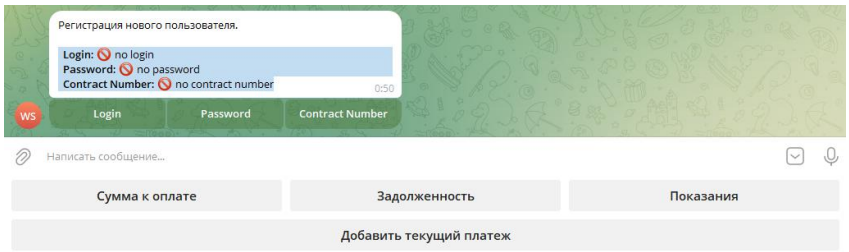



Рисунок 44 – Тестирование команды /registration

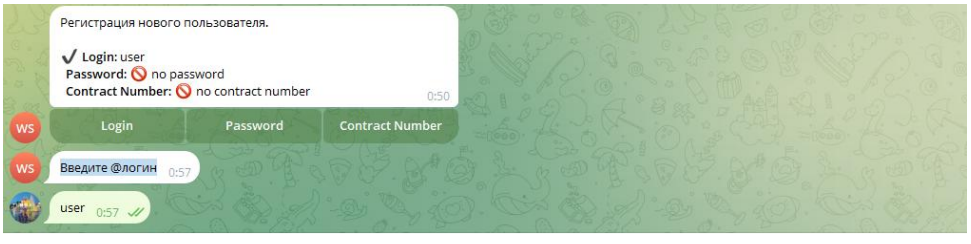



Рисунок 45 – Ввод логина пользователем

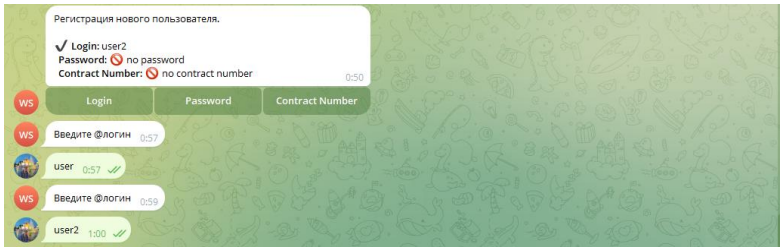



Рисунок 46 – Ввод логина пользователем

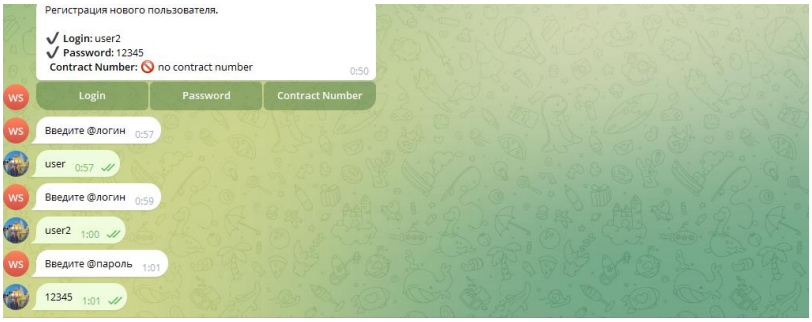



Рисунок 47 – Ввод пароля пользователем

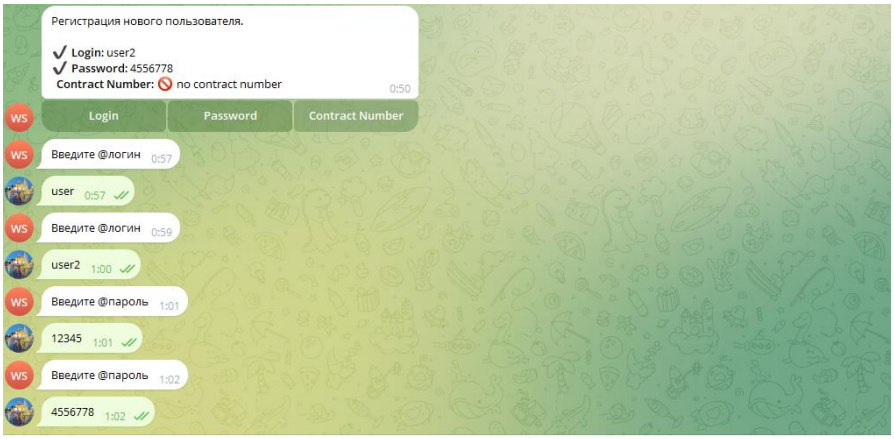



Рисунок 48 – Ввод пароля пользователем

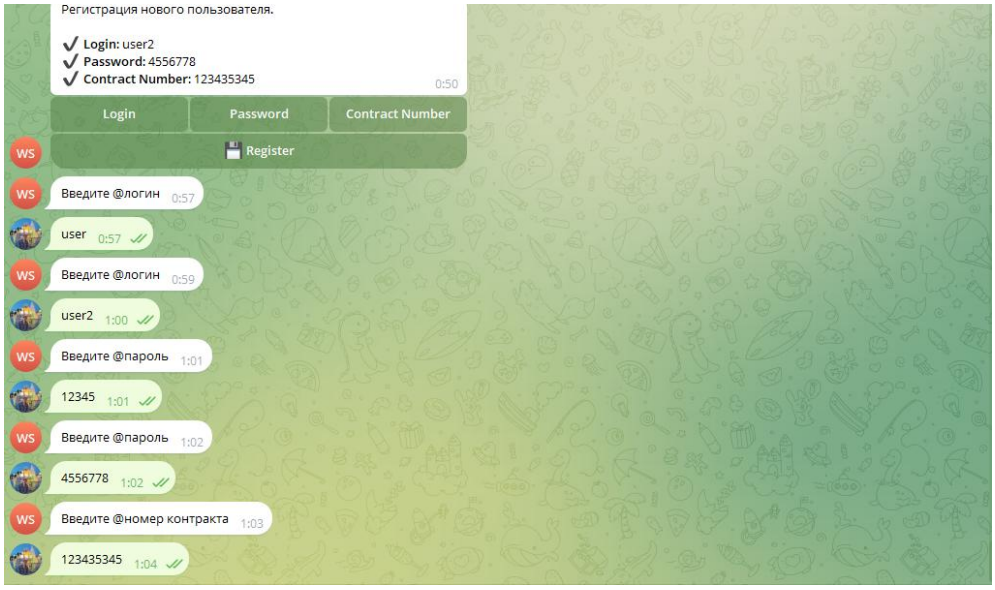



Рисунок 49 – Ввод контракта пользователем

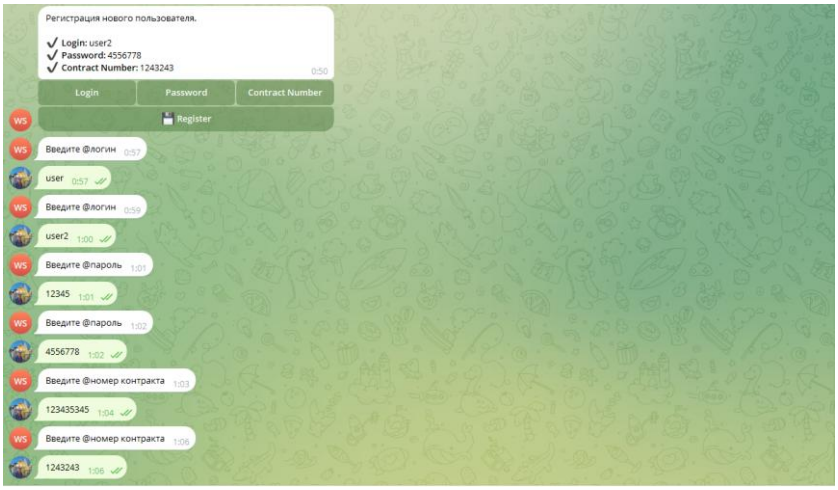



Рисунок 50 – Ввод контракта пользователем

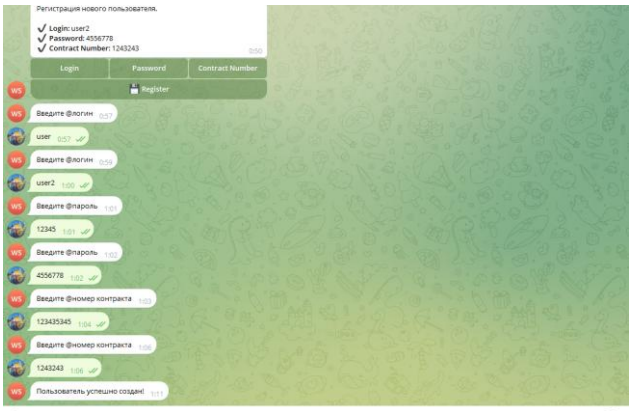



Рисунок 51 – Тестирование кнопки регистрации

Рисунок 52 – Тестирование кнопки регистрации

Результат тестирования представлен на рисунке 53:

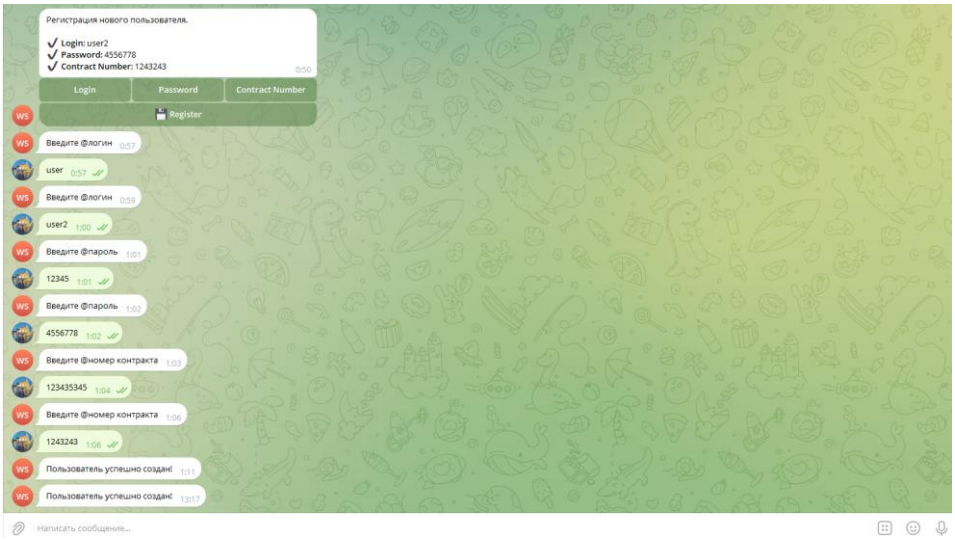



Рисунок 53 – Проверка клика кнопок сообщения после регистрации

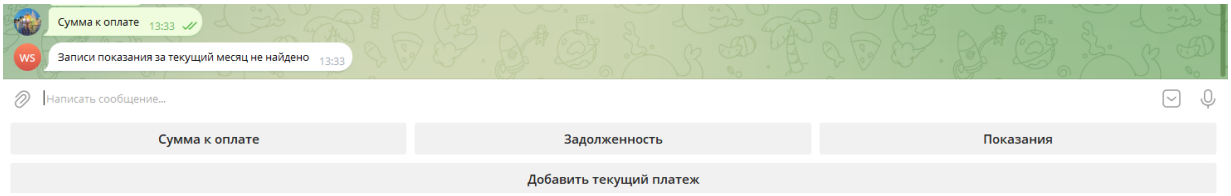



Рисунок 54 – Проверка клика кнопок клавиатуре телеграма

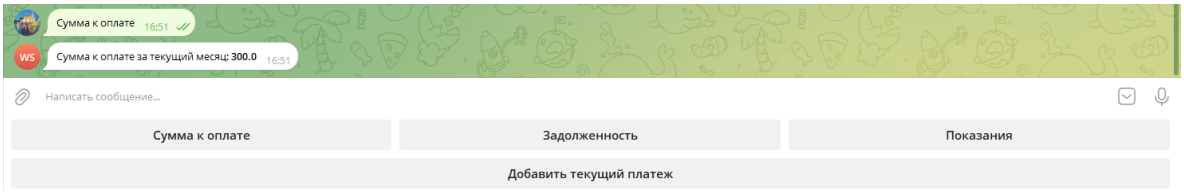



Рисунок 55 – Проверка клика кнопок клавиатуре телеграма

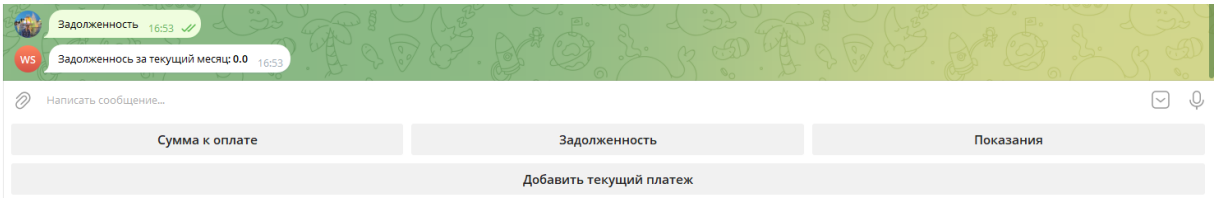



Рисунок 56 – Проверка клика кнопок клавиатуре телеграма

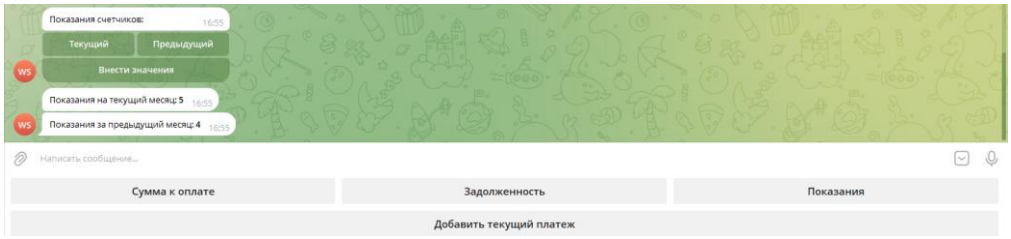



Рисунок 57 – Результат получения показаний

Рисунок 58 – Результат получения суммы платежа

В результате проведения тестирования ошибок выявлено не было. Программа успешно выполняет свой функционал в соответствии с поставленной задачей.

# 3 Руководство пользователя

Бот предназначен для учёта показаний счётчиков. Его основные команды:

– регистрация;

– внесение показаний;

– получение показаний;

– внести сумму платежа;

– проверка суммы долга;

Для получения доступа к клавиатуре нужно ввести команду «/start». После этого пользователю станет доступна клавиатура (рисунок 59).

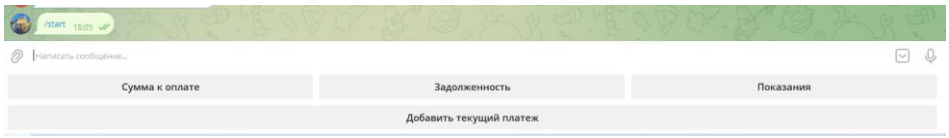



Рисунок 59 – Тестирование запуска бота

После этого бот готов к работе: В рабочей области находится экран для обмена сообщениями и клавиатура для взаимодействия с ботом. Рабочая область представлена на рисунке 60.

Рисунок 60 - Рабочее окно приложения

Для того, чтобы пользователю стали доступны команды из клавиатуры нужно зарегистрироваться: для этого вводим или кликаем на команду /registration (Рисунок 61).

Рисунок 61 – Вызов команды регистрации

Для регистрации пользователя нужно кликнуть на клавишу логин, пароль или номер контракта и после этого ввести нужные данные(Рисунок 62).

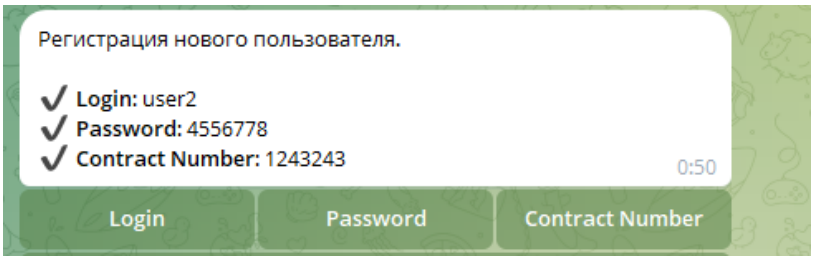



Рисунок 62 – Окно регистрации пользователя

После ввода данных в клавиатуре сообщения появится клавиша «Register»(Рисунок 63).

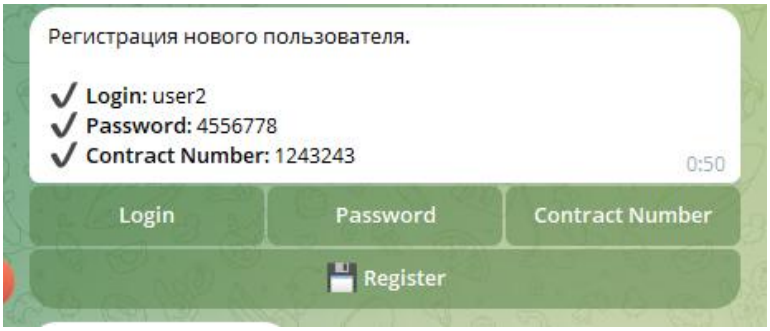



Рисунок 63 – Сообщение регистрации с клавишей «Register»

Чтобы зарегистрироваться нужно нажать на эту клавишу и появится окно об успешной регистрации(Рисунок 64).

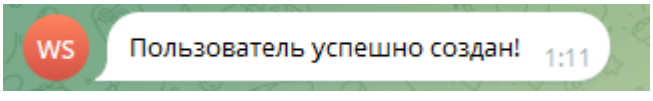



Рисунок 64 – Сообщение об успешной регистрации

Теперь пользователю доступна клавиатура для работы с ботом. Для того чтобы начать работу с показаниями нужно нажать на клавишу «Показания» и

бот выведет сообщение, в котором можно выбрать получить текущие показания счётчиков(показания за текущий месяц), прошлые показания или внести показания за текущий месяц (Рисунок 65).

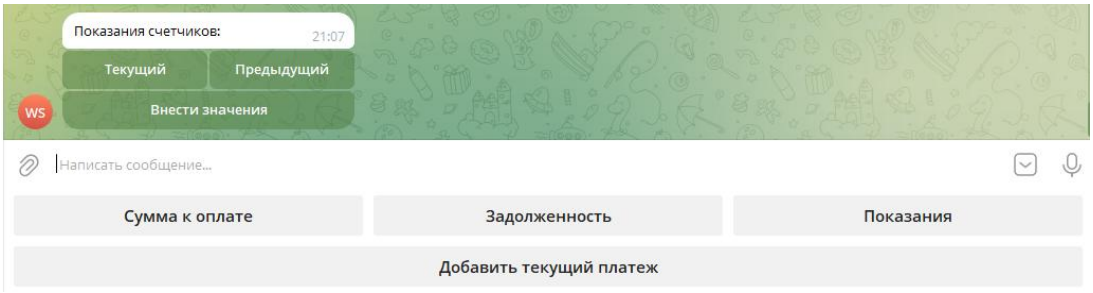



Рисунок 65 – Сообщение с выбором работы с показаниями

Для получения показаний нужно кликнуть на клавишу с требуемыми показаниями и бот пришлёт сообщение с ними(Рисунок 66).

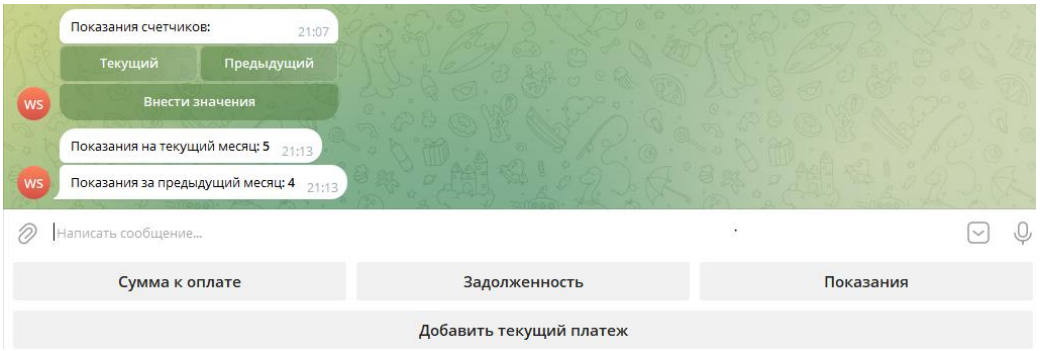



Рисунок 66 – Вывод сообщений с показаниями за текущий/предыдущие месяцы

Для того, чтобы внести значения за текущий месяц, нужно нажать клавишу «Внести значения» и ввести значение за текущий месяц (рисунок 67).

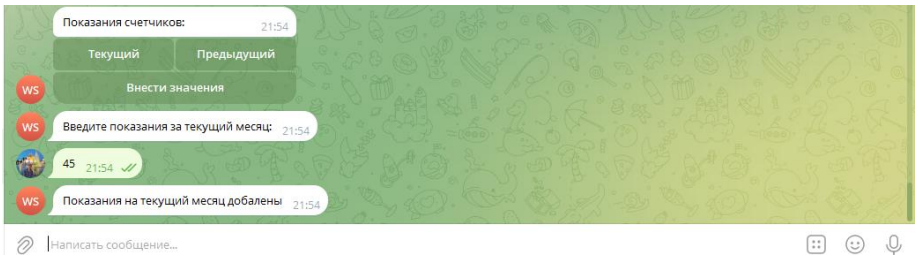



Рисунок 67 – Ввод значений за текущий месяц

Для получения суммы, которую нужно внести в текущем месяце пользователю нужно нажать на клавишу «Сумма к оплате». Бот отправит сообщение с суммой, которую нужно оплатить (рисунок 68).

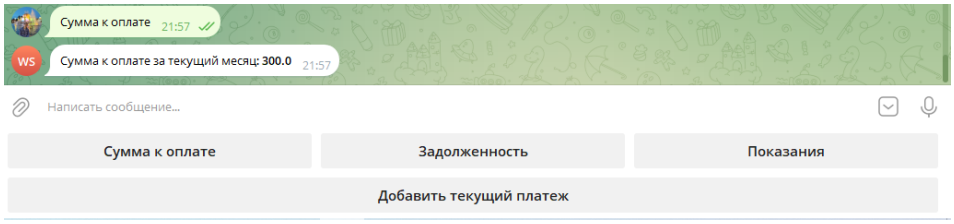



Рисунок 68 – Вывод суммы, требуемой к оплате

Для получения суммы задолженности требуется нажать на клавишу «Задолженность». Бот пришлёт сообщение с суммой задолженности (рисунок 69).

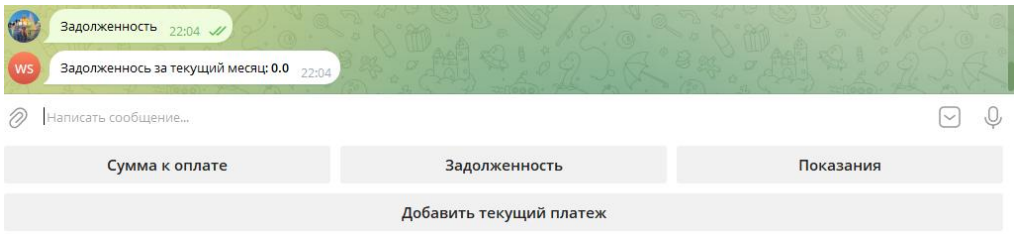



Рисунок 69 – Получения сообщения с сумой задолженности

Клавиша «Добавить текущий платёж» служит для внесения суммы, оплаченной в текущем месяце. Для внесения суммы нужно нажать клавишу и ввести сумму. После этого бот пришлёт сообщение об успешном добавлении платежа (рисунок 70).

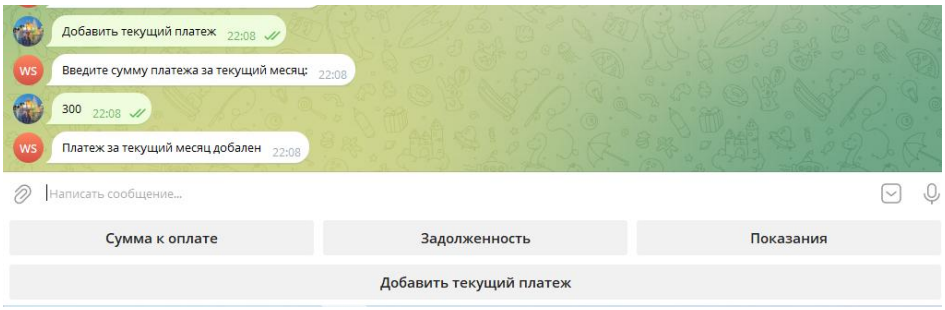



Рисунок 70 – Добавление текущего платежа

# 4 Безопасность жизнедеятельности

Безопасность жизнедеятельности представляет собой совокупность действий, которые направлены на обеспечение безопасности человека в окружающей среде, сохранение его здоровья, разработку способов и средств защиты, которые уменьшают воздействие вредных факторов на здоровье человека, а также на разработку мер по снижению ущерба и ликвидации последствий чрезвычайных ситуаций.

Для того, чтобы уменьшить пагубное воздействия на здоровье человека при использовании компьютера следует правильно организовать рабочее место. Рабочий стол и стул должны быть регулируемыми по высоте, чтобы обеспечить правильное положение тела. Спинка стула должна поддерживать естественный изгиб позвоночника. Стул должен иметь подлокотники для поддержки рук. Верхний край экрана монитора должен находиться на уровне глаз или немного ниже. Расстояние от глаз до экрана должно быть 50-70 см. Монитор должен быть наклонен под углом 10-20 градусов для предотвращения бликов. Рабочее место должно быть равномерно освещено, предпочтительно с использованием естественного света. Искусственное освещение должно быть расположено таким образом, чтобы не создавать бликов на экране монитора. Регулярные перерывы должны составлять каждые 20-30 минут для отдыха глаз. Использование фильтров синего света на экране или специальных очков для работы за компьютером.

Важным фактором является положение человека во время работы. Сидеть нужно прямо, ноги должны полностью стоять на полу или на подставке. Угол между бедрами и туловищем должен быть примерно 90-100 градусов. Руки должны быть согнуты в локтях под углом около 90 градусов, запястья должны находиться в нейтральном положении. Рекомендуется выполнять регулярные перерывы и упражнения: Делать короткие перерывы каждые 30-60 минут для разминки и растяжки. Выполнять простые упражнения для шеи, плеч, рук и спины.

Немаловажным фактором является психологический комфорт и организация рабочего процесса. Организация рабочего пространства, минимизирующая отвлекающие факторы. Использование методов тайм-менеджмента для эффективного распределения задач и времени. Организация рабочего процесса: Четкое планирование рабочего времени с учетом перерывов. Использование удобного и функционального программного обеспечения для минимизации нагрузки и оптимизации рабочего процесса.

Для соблюдения безопасных условий эксплуатации ВМ предъявляются следующие требования к помещениям, в которых они используются:

– температура в помещении ниже 30 градусов Цельсия;

– отсутствие токопроводящей пыли;

– влажность не более 60 процентов;

– хорошая вентиляция;

– полы из непроводящего материала;

– заполнение площади помещения металлическими предметами менее 20 процентов.

Система электробезопасности включает защиту от прямого и косвенного прикосновения к токоведущим и токопроводящим частям оборудования, применение защитного заземления, использование автоматического отключения питания. Работники проходят обучение безопасным методам и приемам работы с токопроводящим оборудованием и проверку знаний требований к организации и выполнению работ.

## Заключение

В ходе выполнения данной работы был проведен анализ современных систем учёта показаний счётчиков, выявлены их особенности и основные характеристики, удовлетворяющие потребности в планировании и организации работы предприятия. Получены навыки проектирования телеграм-ботов.

Были изучены архитектуры приложений, их типовые примеры и используемые инструменты. Получены навыки разработки клиент-серверной архитектуры.

Приобретены знания по созданию архитектуры хранения данных. Созданы основные UML-диаграммы: диаграммы классов, диаграммы деятельности, диаграмма развертывания и диаграмма вариантов использования.

В результате работы была спроектирована, разработана и отлажена система учёта показаний счётчиков на базе телеграм-бота. Приложение создано в INTELEJ IDEA на языке программирования java с использованием фреймворка spring boot.

## Список использованных источников

1 Сервис жилищно-коммунальных услуг «Квартплата онлайн» [электронный ресурс]. – URL: https://dom.gosuslugi.ru/#!/main (Дата последнего обращения 29.04.2024).

2 Сервис жилищно-коммунальных услуг «ГИС ЖКХ» [электронный ресурс]. – URL: https://xn--80aaaf3bi1ahsd.xn--80asehdb/ (Дата последнего обращения 29.04.2024).

3 Статья «Подробная инструкция телеграм бота на java» [электронный ресурс]. – URL: https://rukovodstvorus.ru/podrobnaya-instrukcziya-telegram-bot-na-java/ (Дата последнего обращения 2.05.2024).

4 Статья «Настройка Spring Boot для телеграм-бота» [электронный ресурс]. – URL:https://javarush.com/groups/posts/2966-sozdaem-telegram-bota-s-ispoljhzovaniem-spring-boot-pt2-quiz-bot (Дата последнего обращения 5.05.2024).

5 Документация по Telegram Bot API [электронный ресурс]. – URL: https://core.telegram.org/bots/api (Дата последнего обращения 12.05.2024).

6 Spring Framework Documentation [электронный ресурс]. – URL: https://docs.spring.io/spring-framework/reference/index.html (Дата последнего обращения 12.05.2024).

7 Руководство по языку программирования java [Электронный ресурс]. – URL: https://metanit.com/java/tutorial/ (Дата последнего обращения 10.05.2024).

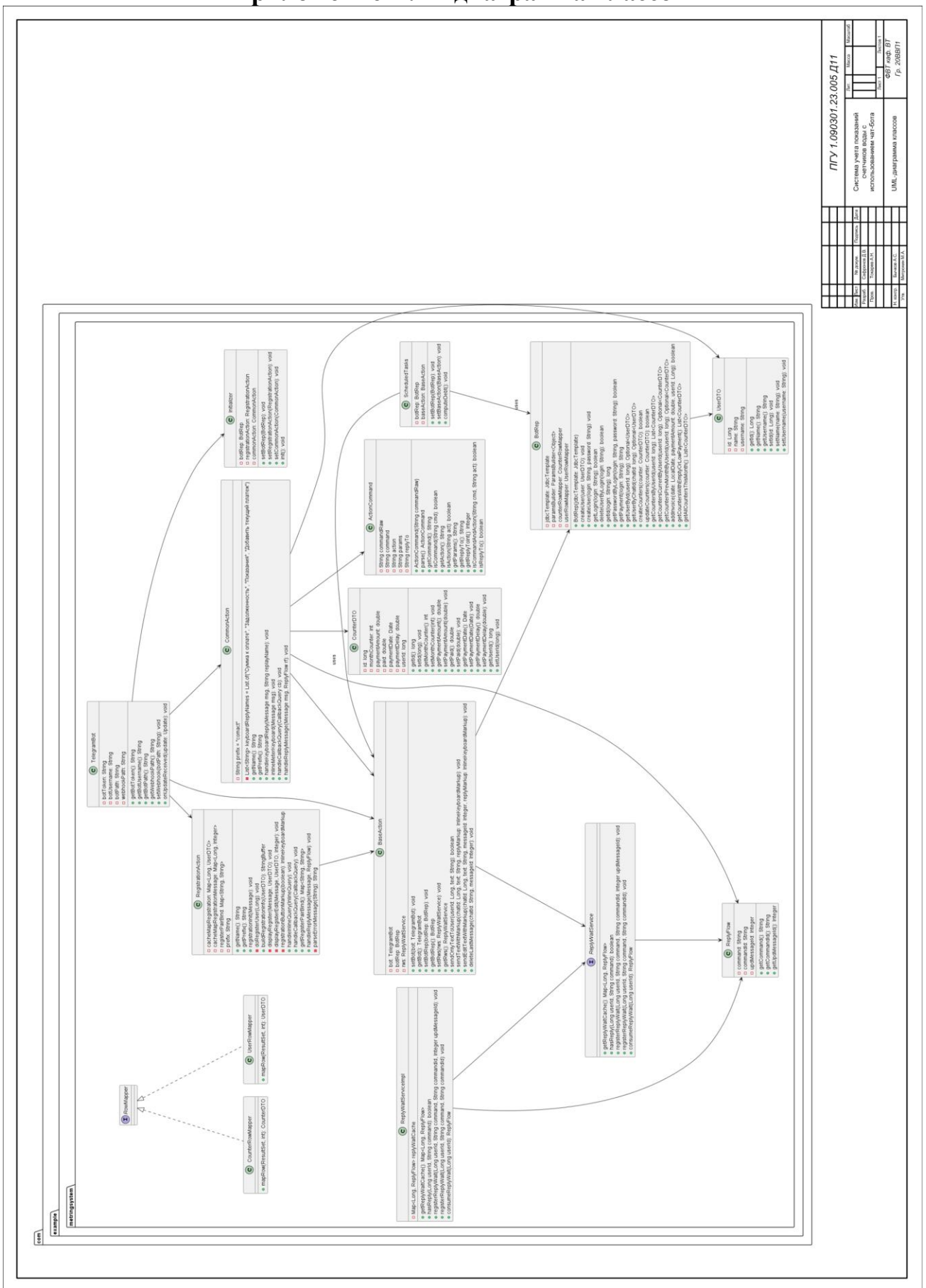
## Приложение А

(Обязательное)

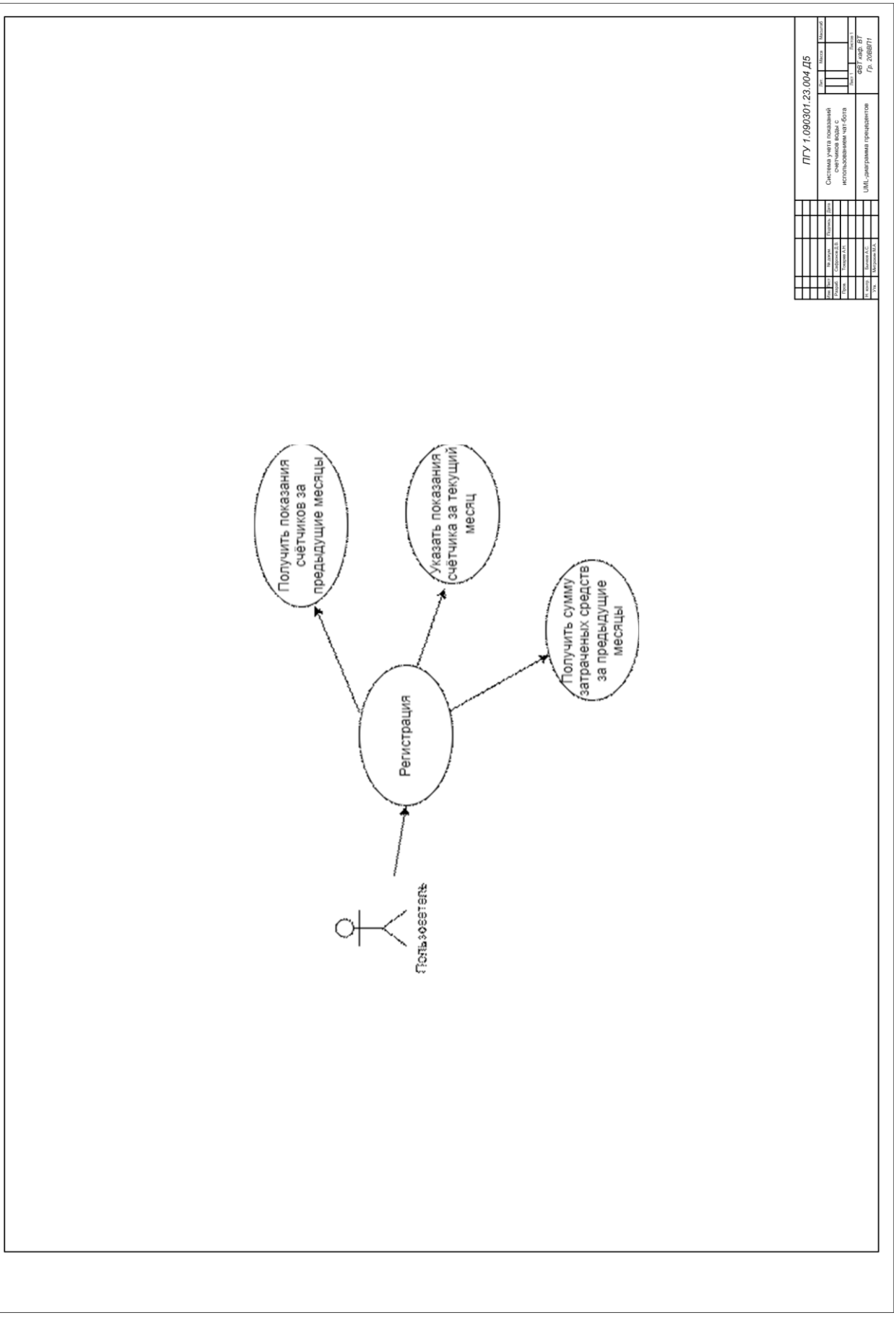
Система учёта показаний счётчиков воды с использованием чат-бота

UML-диаграммы

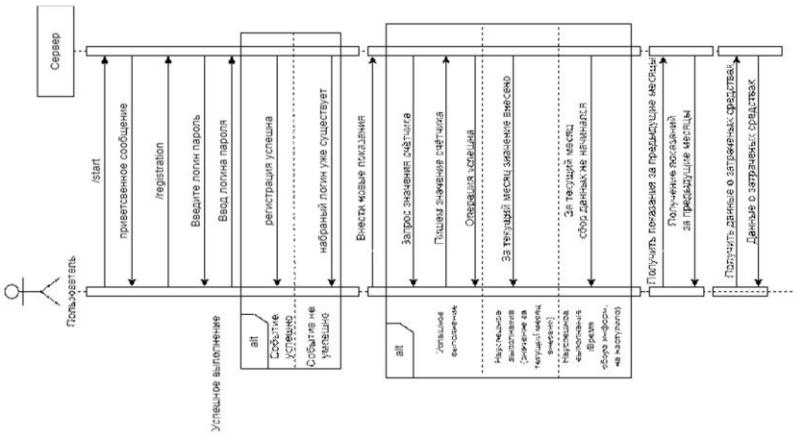
# Приложение А.1 – диаграмма классов



# Приложение А.2 – Диаграмма прецедентов



# Приложение А.3 – Диаграмма последовательности




ПГУ 1.090301.23.004 Д14

Система учета показаний счетчиков воды с использованием чат-бота

UML-диаграмма последовательности

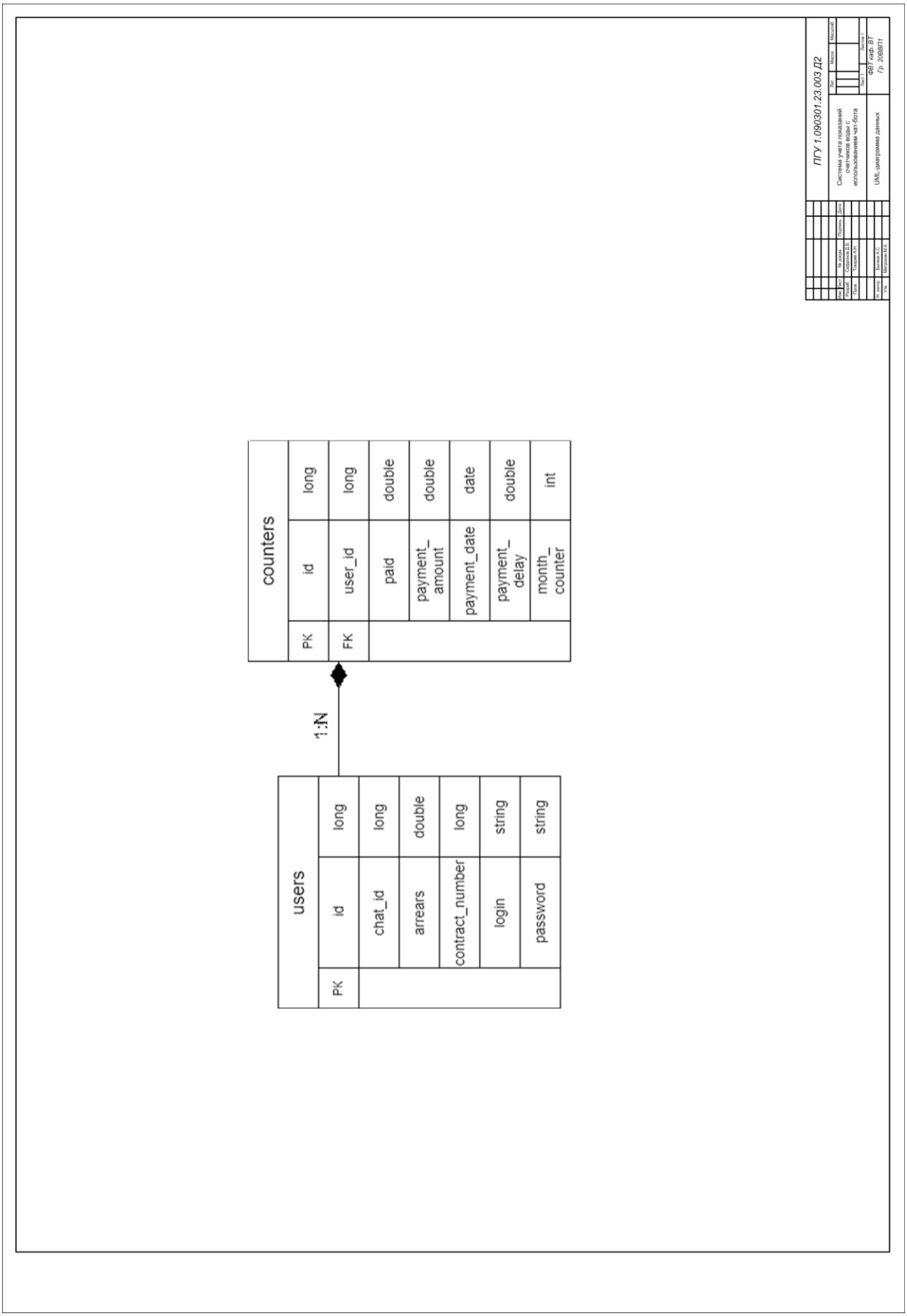
Лит. | Масса | Масштаб

Лист 1 | Листов 1

ФБГ гиф. ВТ Гр. 208871

| Изм. | Лист | № докум. | Подпись | Дата | ПГУ 1.090301.23.001 ПЗ | Лист 66 |
|---|---|---|---|---|---|---|

# Приложение А.4 – Диаграмма сущностей



**Приложение Б**

(Обязательное)

Система учёта показаний счётчиков воды с использованием чат-бота

Листинг программы

```
package com.example.metringsystem;


    import com.example.metringsystem.system.actions.CommonAction;
    import
com.example.metringsystem.system.actions.RegistrationAction;
    import
com.example.metringsystem.system.actions.reply.ReplyFlow;
    import com.example.metringsystem.system.dao.domain.UserDTO;
    import com.example.metringsystem.system.dao.utils.DateUtil;
    //import lombok.Getter;
    //import lombok.extern.slf4j.Slf4j;
    import org.springframework.stereotype.Component;
    import org.telegram.telegrambots.bots.TelegramLongPollingBot;
    import
org.telegram.telegrambots.meta.api.methods.send.SendMessage;
    import
org.telegram.telegrambots.meta.api.methods.updates.GetUpdates;
    import
org.telegram.telegrambots.meta.api.objects.CallbackQuery;
    import org.telegram.telegrambots.meta.api.objects.Update;
    import org.telegram.telegrambots.meta.api.objects.User;
    import
org.telegram.telegrambots.meta.api.objects.inlinequery.InlineQuery
;
    import
org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyb
oardMarkup;
    import
org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.K
eyboardButton;
    import
org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.K
eyboardRow;
    import
org.telegram.telegrambots.meta.exceptions.TelegramApiException;

    import java.time.LocalDate;
    import java.util.ArrayList;
    import java.util.List;
    import java.util.Optional;

    //@Slf4j
    //@Getter
    @Component
    public class TelegramBot extends TelegramLongPollingBot {

        private BotRep botRep;

        public void setBotRep(BotRep botRep) {
            this.botRep = botRep;
        }
```

```
        private RegistrationAction registrationAction;

        public    void    setRegistrationAction(RegistrationAction
registrationAction) {
            this.registrationAction = registrationAction;
        }

        private CommonAction commonAction;

        public void setCommonAction(CommonAction commonAction) {
            this.commonAction = commonAction;
        }

        @Override
        public String getBotUsername() {
            return "water_metering_system_bot";
        }

        @Override
        public String getBotToken() {
            return               "6849774414:AAGbpSHMjqmihiQoqVzC-
yagUupGysX5o2A";
        }

        boolean autorization_password;
        boolean autorization_login;
        long id;
        String login_name;

        @Override
        public void onUpdateReceived(Update update) {
    ////////////////////////////

            //String Password;


            if               (update.hasMessage()               &&
update.getMessage().hasText()) {
                String                messageText                =
update.getMessage().getText();
                long chatId = update.getMessage().getChatId();
                User u = update.getMessage().getFrom();
                String text;

                // keyboard reply handler

if(commonAction.keyboardReplyNames.contains(messageText)){

commonAction.handleKeyboardReply(update.getMessage(),
messageText);
                    return;
                }
```

```
                // handle of waiting reply

if(registrationAction.getRws().hasReply(u.getId(),
registrationAction.getPrefix()))  {

                    ReplyFlow                rf                =
registrationAction.getRws().consumeReplyWait(u.getId());

                    if(rf != null) {

registrationAction.handleReplyMessage(update.getMessage(), rf);
                    }
                    return;
                }                                          else
if(commonAction.getRws().hasReply(u.getId(),
commonAction.getPrefix()))  {

                    ReplyFlow                rf                =
commonAction.getRws().consumeReplyWait(u.getId());

                    if(rf != null) {

commonAction.handleReplyMessage(update.getMessage(), rf);
                    }
                    return;
                }



                if(messageText.startsWith("/invoice")){
                    String[] parts = messageText.split(" ");

                    if(parts.length>2){
                        LocalDate             date              =
DateUtil.parseDate(parts[1]);

                        Double paymentAmount = null;
                        try{
                            paymentAmount                        =
Double.valueOf(parts[2]);
                        } catch (NumberFormatException nfe){
                            sendMessage(chatId,
                                    "Не      коректный      формат
комманды\n\n"+
                                            "*формат*:    /invoice
{payment  date:  dd.mm.yyyy}  {payment  amount:  number}  {user  id:
number (опционально)}\n"+
                                            "*пример*:    /invoice
01.05.2024 1200.30 1"
                            );
                            return;
                        }
```

```
                        Long userId = null;
                        try{
                            userId       =       (parts.length>3)?
Long.valueOf(parts[3]): null;
                        }   catch   (NumberFormatException   nfe){
/*Nothing to do*/}

                        if(userId == null){
                            Optional<UserDTO>        user        =
botRep.getUserByChatId(chatId);

                            if(!user.isPresent()){
                                sendMessage(chatId,  "Пользователь
не зарегистрирован! Пройти регистрацию /registration");
                                return;
                            }
                            userId = user.get().getId();
                        }

                        if(botRep.addInvoice(date,  paymentAmount,
userId)){
                            sendMessage(chatId,   "Счет   выставлен
для пользователя: " + userId+ " на сумму " + paymentAmount + " на
дату "+date.toString());
                        }
                    } else {
                        sendMessage(chatId,
                                "*формат*:    /invoice    {payment
date: dd.mm.yyyy} {payment amount: number} {user id: number }\n"+
                                        "*пример*:        /invoice
01.05.2024 1200.30 1"
                        );
                    }

                    return;
                }

                switch (messageText) {
                    case "/start":

                        startCommandReceived(chatId,
update.getMessage().getChat().getFirstName());
                      break;

                    case "/registration":

registrationAction.registrationInit(update.getMessage());
                        //botRep.createUser("Testuser",
"Password");
                        break;

                    /*case "/autorizate":
```

```
                        authorization(chatId);

                        break;
                    case "/getresult":

                        String            paymentList            =
botRep.getPayment(update.getMessage().getChat().getFirstName());
                        sendMessage(chatId,
paymentList.toString());
                        //getResult(chatId);

                    break;*/

                    default:
                        /*if (autorization_login == true)
                        {

                            boolean            login            =
botRep.getLogin(messageText);
                            //sendMessage(chatId, password);
                            if(login == true)
                            {
                                sendMessage(chatId, "успешно");
                                login_name = messageText;
                                sendMessage(chatId,       "Введите
пароль");
                                autorization_login = false;
                                autorization_password = true;
                                break;

                            }
                            else {
                                sendMessage(chatId,         "Логин
неверный");
                                sendMessage(chatId,       "Введите
Логин");
                                autorization_login = true;
                            }
                        }


                        if (autorization_password == true)
                        {

                            boolean         password         =
botRep.getPasswordByLogin(login_name, messageText);
                            //sendMessage(chatId, password);
                            if(password == true)
                            {
                                sendMessage(chatId,
"успешно");
                                id                               =
botRep.getId(login_name);
```

```
                                        autorization_password        =
false;
                                       //          sendMessage(chatId,
String.valueOf(id));
                                        break;

                                    }
                                    else {
                                        sendMessage(chatId,        "не
верно");
                                        sendMessage(chatId,   "Введите
пароль");
                                        autorization_password = true;
                                    }

                                }*/
                                //sendMessage(chatId,   "command   was
not recognized");

                        }

                } else if(update.hasCallbackQuery()) {
                    CallbackQuery               cb               =
update.getCallbackQuery();


if(cb.getData().startsWith(RegistrationAction.prefix)){

registrationAction.handleCallbackQuery(cb);
                        }                                          else
if(cb.getData().startsWith(CommonAction.prefix)){
                            commonAction.handleCallbackQuery(cb);
                        }

                        //actionController.handleCallbackQuery(cb);
                } else if(update.hasInlineQuery()){
                    InlineQuery iq = update.getInlineQuery();

                    //actionController.handleInlineQuery(iq);
                }
        }

        private void getResult(long chatId, String name) {
            String paymentList = botRep.getPayment(name);
            sendMessage(chatId, paymentList.toString());
        }


        private void authorization(long chatId) {

            autorization_login = true;
            sendMessage(chatId, "Введите логин");
        }
```

```
        private List<Update> executeApiMethod(GetUpdates method)
throws TelegramApiException {
            return execute(method);
        }


        private void startCommandReceived(long chatId, String
name) {


            String answer = "Это бот Система учёта счётчиков, вот
список команд, которые я могу предложить:" +
                    /* "\n /autorizate - авторизация"+*/
                    /* "\n /getresult - получение результата" +*/
                     "\n /registration - регистрация"+
                     "\n /invoice - выставить счет пользователю";


            //log.info("Replied to user " + name);
            System.out.println("Replied to user " + name);

            //sendMessage(chatId, answer);
            sendKeyboardReply(chatId, answer);
        }

        private void sendMessage(long chatId, String textToSend)
{
            SendMessage message = new SendMessage();
            message.enableMarkdown(true);
            message.setChatId(String.valueOf(chatId));
            //log.info("message: "+ textToSend);
            System.out.println("message: "+ textToSend);
            message.setText(textToSend);

            try{
                execute(message);
            }
            catch (TelegramApiException e){
                //log.error(e.getMessage());
                System.out.println(e.getMessage());
            }
        }
    ////////////////////////

        public void sendKeyboardReply(Long chatId, String text){

            SendMessage s = new SendMessage();
            s.setChatId(chatId.toString());
            s.enableMarkdown(true);
            s.setText(text);
```

```
            ReplyKeyboardMarkup    replyKeyboardMarkup    =    new
ReplyKeyboardMarkup();
            s.setReplyMarkup(replyKeyboardMarkup);
            replyKeyboardMarkup.setSelective(true);
            replyKeyboardMarkup.setResizeKeyboard(true);
            replyKeyboardMarkup.setOneTimeKeyboard(false);

            List<KeyboardRow>        keyboard        =        new
ArrayList<KeyboardRow>();

            KeyboardRow keyboardRow = new KeyboardRow();
            keyboardRow.add(new      KeyboardButton("Сумма      к
оплате"));
            keyboardRow.add(new KeyboardButton("Задолженность"));
            keyboardRow.add(new KeyboardButton("Показания"));

            keyboard.add(keyboardRow);

            keyboardRow = new KeyboardRow();
            keyboardRow.add(new  KeyboardButton("Добавить  текущий
платеж"));

            keyboard.add(keyboardRow);

            replyKeyboardMarkup.setKeyboard(keyboard);

            try{
                execute(s);
            }
            catch (TelegramApiException e){}
        }

    }


    package com.example.metringsystem;


    //import lombok.extern.slf4j.Slf4j;
    import org.springframework.boot.SpringApplication;
    import
org.springframework.boot.autoconfigure.SpringBootApplication;
    import org.springframework.stereotype.Component;

    //@Slf4j
    @Component
    @SpringBootApplication
    public class MetringsystemApplication {
        public     static     void     main(String[]     args){
    SpringApplication.run(MetringsystemApplication.class, args);}

        /*@Autowired
        TelegramBot bot;
```

```
          @EventListener({ContextRefreshedEvent.class})
          public void init() throws TelegramApiException {
               TelegramBotsApi      telegramBotsApi      =      new
TelegramBotsApi(DefaultBotSession.class);
               try {
                    telegramBotsApi.registerBot(bot);
               }
               catch (TelegramApiException e) {
                    log.error("Error occurred: " + e.getMessage());
               }
          }*/
     }


     package com.example.metringsystem;

     import com.example.metringsystem.system.actions.CommonAction;
     import
com.example.metringsystem.system.actions.RegistrationAction;
     import
org.springframework.beans.factory.annotation.Autowired;
     import
org.springframework.context.event.ContextRefreshedEvent;
     import org.springframework.context.event.EventListener;
     import
org.springframework.scheduling.annotation.EnableScheduling;
     import org.springframework.stereotype.Component;
     import org.telegram.telegrambots.meta.TelegramBotsApi;
     import
org.telegram.telegrambots.meta.exceptions.TelegramApiException;
     import
org.telegram.telegrambots.updatesreceivers.DefaultBotSession;
     //import lombok.extern.slf4j.Slf4j;
     import com.example.metringsystem.TelegramBot;

     //@Slf4j
     @Component
     public class Initializer {

          private BotRep botRep;

          @Autowired
          public void setBotRep(BotRep botRep) {
              this.botRep = botRep;
          }

          private RegistrationAction registrationAction;

          @Autowired
          public    void    setRegistrationAction(RegistrationAction
registrationAction) {
              this.registrationAction = registrationAction;
```

```
        }

        private CommonAction commonAction;

        @Autowired
        public void setCommonAction(CommonAction commonAction) {
            this.commonAction = commonAction;
        }

        @EventListener({ContextRefreshedEvent.class})
        public void init() throws TelegramApiException {
            TelegramBotsApi      telegramBotsApi      =      new
TelegramBotsApi(DefaultBotSession.class);
            try {

                TelegramBot bot = new TelegramBot();
                bot.setBotRep(botRep);
                bot.setRegistrationAction(registrationAction);
                bot.setCommonAction(commonAction);

                telegramBotsApi.registerBot(bot);
            }
            catch (TelegramApiException e) {
                //log.error("Error occurred: " + e.getMessage());
                System.out.println("Error    occurred:    "    +
e.getMessage());
            }
        }
    }

    package com.example.metringsystem;
    import
com.example.metringsystem.system.dao.domain.CounterDTO;
    import com.example.metringsystem.system.dao.domain.UserDTO;
    import
com.example.metringsystem.system.dao.rowMapers.CounterRowMapper;
    import
com.example.metringsystem.system.dao.rowMapers.UserRowMapper;
    import com.example.metringsystem.system.dao.utils.DateUtil;
    import
com.example.metringsystem.system.dao.utils.ParamsBuilder;
    import
org.springframework.beans.factory.annotation.Autowired;
    import
org.springframework.dao.EmptyResultDataAccessException;
    import org.springframework.jdbc.core.JdbcTemplate;
    import org.springframework.stereotype.Repository;
    import
org.springframework.transaction.annotation.Transactional;

    import java.sql.Date;
    import java.time.LocalDate;
    import java.time.ZoneId;
```

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;

@Repository
public class BotRep {

    private final JdbcTemplate jdbcTemplate;

    private final ParamsBuilder paramsBuilder = new
ParamsBuilder<Object>();
    private final CounterRowMapper counterRowMapper = new
CounterRowMapper();
    private final UserRowMapper userRowMapper = new
UserRowMapper();

    @Autowired
    public BotRep(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    @Transactional
    public void createUser(UserDTO user) {
        List<Object> parameters = paramsBuilder.newMap()
                .add(user.getChatId())
                .add(user.getArrears())
                .add(user.getContractNumber())
                .add(user.getLogin())
                .add(user.getPassword())
                .build();

        String sql = "INSERT INTO users (chat_id, arrears,
contract_number, login, password) VALUES (?, ?, ?, ?, ?)";
        jdbcTemplate.update(sql, parameters.toArray());
    }

    @Deprecated
    public void createUser(String login, String password) {
        String sql = "INSERT INTO users (login, password)
VALUES (?, ?)";
        jdbcTemplate.update(sql, login, password);
    }


    /*public boolean getPassword(String password) {
        String sql = "SELECT user_id FROM users WHERE login =
?";
        return jdbcTemplate.queryForObject(sql, String.class,
password);
    }*/

    public boolean getLogin(String login) {
```

```
            String sql = "SELECT COUNT(*) FROM users WHERE login
= ?";
            int    count    =    jdbcTemplate.queryForObject(sql,
Integer.class, login);
            return count > 0;
        }

        @Transactional
        public boolean deleteUserByLogin(String login) {
            String sql = "DELETE FROM users WHERE login = ?";
            return jdbcTemplate.update(sql, login) > 0;
        }

        public long getId(String login) {
            String sql = "SELECT id FROM users WHERE login = ?";
            return  jdbcTemplate.queryForObject(sql,  long.class,
login);
        }

        public  boolean  getPasswordByLogin(String  login,  String
password) {
            String sql = "SELECT COUNT(*) FROM users WHERE login
= ? AND password = ?";
            int    count    =    jdbcTemplate.queryForObject(sql,
Integer.class, login, password);
            return count > 0;
        }


        /*public List<String> getAllLogins() {
            String sql = "SELECT login FROM users";
            return jdbcTemplate.queryForList(sql, String.class);
        }

        public double getPaymentByCounterId(long counterId) {
            String  sql  =  "SELECT  payment  FROM  users  WHERE
counter_id = ?";
            return jdbcTemplate.queryForObject(sql, Double.class,
counterId);
        }

        public List<Double> getAllPayment() {
            String sql = "SELECT payment FROM users";
            return jdbcTemplate.queryForList(sql, Double.class);
        }

        public    void    updateCounters(long    counterId,    int
newCounters) {
            String  sql  =  "UPDATE  users  SET  counters  =  ?  WHERE
counter_id = ?";
            jdbcTemplate.update(sql, newCounters, counterId);
        }*/
```

```
        public String getPayment(String login) {
            String sql = "SELECT payment FROM users WHERE login =
?";
            return  jdbcTemplate.queryForObject(sql,  String.class,
login);
        }

        public Optional<UserDTO> getUserById(long userId){
            String sql = "SELECT * FROM users c WHERE id = ?";

            try {
                return
Optional.of(jdbcTemplate.queryForObject(sql,        userRowMapper,
userId));
            } catch (EmptyResultDataAccessException erdae){
                return Optional.empty();
            }
        }

        public Optional<UserDTO> getUserByChatId(long chatId){
            String  sql  =  "SELECT  *  FROM  users  c  WHERE  chat_id  =
?";

            try {
                return
Optional.of(jdbcTemplate.queryForObject(sql,        userRowMapper,
chatId));
            } catch (EmptyResultDataAccessException erdae){
                return Optional.empty();
            }
        }


        @Transactional
        public boolean createCounters(CounterDTO counter){

            List<Object> parameters = paramsBuilder.newMap()
                    .add(counter.getUserId())
                    .add(counter.getMonthCounter())
                    .add(counter.getPaymentAmount())
                    .add(counter.getPaid())
                    .add(counter.getPaymentDate())
                    .add(counter.getPaymentDelay())
                    .build();

            String   sql   =   "INSERT   INTO   counters(user_id,
month_counter,  payment_amount,  paid,  payment_date,  payment_delay)
VALUES (?,?,?,?,?,?)";

            return                          jdbcTemplate.update(sql,
parameters.toArray())>0;
        }
```

```
        @Transactional
        public boolean updateCounters(CounterDTO counter){

            List<Object> parameters = paramsBuilder.newMap()
                    .add(counter.getMonthCounter())
                    .add(counter.getPaymentAmount())
                    .add(counter.getPaid())
                    .add(counter.getPaymentDate())
                    .add(counter.getPaymentDelay())
                    .add(counter.getId())
                    .build();

            String sql = "UPDATE counters SET month_counter=?,
payment_amount=?, paid=?, payment_date=?, payment_delay=?   WHERE
id = ?";

            return                          jdbcTemplate.update(sql,
parameters.toArray())>0;
        }

        // получения ежемесячных показаний с датой платежа и
оплаченной суммой
        public List<CounterDTO> getCountersByUserId(long userId){
            String sql = "SELECT * FROM counters c WHERE user_id
= ? order by payment_date desc ";

            return    jdbcTemplate.query(sql,    counterRowMapper,
userId);
        }


        // получения показаний текущий месяц
        public                                  Optional<CounterDTO>
getCountersCurrentByUserId(long userId){
            String sql = "SELECT * FROM counters where user_id =
?  and  date_trunc('month',  payment_date)  =  date_trunc('month',
CURRENT_DATE)";

            try {
                return
Optional.of(jdbcTemplate.queryForObject(sql,      counterRowMapper,
userId));
            } catch (EmptyResultDataAccessException erdae){
                return Optional.empty();
            }
        }


        // получения показаний за предыдущий месяц
        public                                  Optional<CounterDTO>
getCountersPrevMonthByUserId(long userId){
```

```
            String sql = "SELECT * FROM counters where user_id =
? and date_trunc('month', payment_date) = date_trunc('month',
CURRENT_DATE - interval '1 month')";

            try {
                return
Optional.of(jdbcTemplate.queryForObject(sql,      counterRowMapper,
userId));
            } catch (EmptyResultDataAccessException erdae){
                return Optional.empty();
            }
        }

        public boolean addInvoice(LocalDate date, double
paymentAmount, Long userId) {
            CounterDTO counter = new CounterDTO();
            counter.setUserId(userId);
            counter.setPaymentAmount(paymentAmount);

counter.setPaymentDate(DateUtil.localDateToDate(date));
            return createCounters(counter);
        }

        // получения списка показаний где платеж не был
осуществлен
        public                                    List<CounterDTO>
getCountersWithEmptyOrLowPayment(){
            String sql = "SELECT * FROM counters c WHERE paid = 0
or paid < payment_amount order by payment_date desc ";

            return jdbcTemplate.query(sql, counterRowMapper);
        }

        // получения списка показаний за текущий месяц
        public List<CounterDTO> getAllCountersThisMonth(){
            String sql = "SELECT * FROM counters c WHERE
date_trunc('month',     payment_date)     =     date_trunc('month',
CURRENT_DATE)";
            return jdbcTemplate.query(sql, counterRowMapper);
        }
    }



    package com.example.metringsystem.system.tasks;

    import com.example.metringsystem.BotRep;
    import com.example.metringsystem.system.actions.BaseAction;
    import
com.example.metringsystem.system.dao.domain.CounterDTO;
    import com.example.metringsystem.system.dao.domain.UserDTO;
    import
org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import java.time.LocalDate;
import java.util.Date;
import java.util.List;
import java.util.Optional;
import java.util.Random;

@Component
public class ScheduledTasks {

    private BotRep botRep;

    @Autowired
    public void setBotRep(BotRep botRep) {
        this.botRep = botRep;
    }

    private BaseAction baseAction;

    @Autowired
    public void setBaseAction(BaseAction baseAction) {
        this.baseAction = baseAction;
    }

    //@Schduled(cron = "00 10 28 * *")
    @Scheduled(fixedDelay = 120000)
    public void computeDebt(){


        if(LocalDate.now().getDayOfMonth() != 28)
            return;

        List<CounterDTO>        listEmptyPaidCounters        =
botRep.getAllCountersThisMonth();

        listEmptyPaidCounters.stream().forEach(c->{

            //Если За текущий месяц сумма платежа не указана
на 28 число, то добавить в задолжность среднюю сумму
            if(c.getPaid() == 0d){
                c.setPaymentDelay(300);

                botRep.updateCounters(c);
            } else

            //если показания за текущий месяц указаны, но
оплаченная сумма не соответсвует показаниям на 28 число(из расчёта
куб воды = 30.руб) установить соответсв. Сумму в задолжность и
оповестить пользователя.
            if(c.getPaid()     >0     &&     (c.getPaid()     <
c.getMonthCounter() * 30)){
```

```
                    c.setPaymentDelay((c.getMonthCounter() * 30 -
c.getPaid()));

                    Optional<UserDTO>            user            =
botRep.getUserById(c.getUserId());

                    user.ifPresent(u->{

baseAction.sendOnlyTextToUser(u.getChatId(),  "Сумма  задолженности
в этом месяце: *"+c.getPaymentDelay()+"*");
                    });

                    botRep.updateCounters(c);
                }

            });
        }

    }
```

```
    package com.example.metringsystem.system.dao.utils;

    import java.util.ArrayList;
    import java.util.HashMap;
    import java.util.List;
    import java.util.Map;

    public class ParamsBuilder<E> {

        public ParamsBuilder() {
        }

        protected List<E> newContainer() {
            return new ArrayList<>();
        }

        public final ParamsBuilder.Builder newMap() {
            ParamsBuilder.Builder       builder       =       new
ParamsBuilder.Builder();
            this.preInit(builder);
            return builder;
        }

        protected void preInit(ParamsBuilder.Builder builder) {
        }
        public final class Builder {
            private final List<E> list;

            private Builder() {
                this.list = ParamsBuilder.this.newContainer();
            }

            public final ParamsBuilder.Builder add(E value) {
```

```
                this.list.add(value);
                return this;
            }

            public final List<E> build() {
                return this.list;
            }
        }
    }


    package com.example.metringsystem.system.dao.utils;

    import java.time.LocalDate;
    import java.time.ZoneId;
    import java.time.format.DateTimeFormatter;
    import java.util.Date;

    public class DateUtil {

        public static LocalDate parseDate(String date){
            DateTimeFormatter             formatter             =
DateTimeFormatter.ofPattern("dd.MM.yyyy");
            LocalDate     dateTime     =     LocalDate.parse(date,
formatter);

            return dateTime;
        }

        public static Date localDateToDate(LocalDate date){
            return Date.from(

date.atStartOfDay(ZoneId.systemDefault()).toInstant()
            );
        }
    }

    package com.example.metringsystem.system.dao.rowMapers;

    import
com.example.metringsystem.system.dao.domain.CounterDTO;
    import com.example.metringsystem.system.dao.domain.UserDTO;
    import org.springframework.jdbc.core.RowMapper;

    import java.sql.ResultSet;
    import java.sql.SQLException;

    public class UserRowMapper implements RowMapper<UserDTO> {
        @Override
        public  UserDTO  mapRow(ResultSet  rs,  int  rowNum)  throws
SQLException {
            UserDTO userDTO = new UserDTO();
            userDTO.setId(rs.getLong("id"));
```

```
            userDTO.setArrears(rs.getDouble("arrears"));

userDTO.setContractNumber(rs.getLong("contract_number"));
            userDTO.setLogin(rs.getString("login"));
            userDTO.setPassword(rs.getString("password"));
            userDTO.setChatId(rs.getLong("chat_id"));

            return userDTO;
        }
    }


    package com.example.metringsystem.system.dao.rowMapers;

    import
com.example.metringsystem.system.dao.domain.CounterDTO;
    import org.springframework.jdbc.core.RowMapper;

    import java.sql.ResultSet;
    import java.sql.SQLException;

    public        class        CounterRowMapper        implements
RowMapper<CounterDTO> {

        @Override
        public CounterDTO mapRow(ResultSet rs, int rowNum) throws
SQLException {

            CounterDTO counterDTO = new CounterDTO();
            counterDTO.setId(rs.getLong("id"));

counterDTO.setMonthCounter(rs.getInt("month_counter"));

counterDTO.setPaymentAmount(rs.getDouble("payment_amount"));
            counterDTO.setPaid(rs.getDouble("paid"));

counterDTO.setPaymentDate(rs.getDate("payment_date"));

counterDTO.setPaymentDelay(rs.getDouble("payment_delay"));
            counterDTO.setUserId(rs.getLong("user_id"));

            return counterDTO;
        }
    }



    package com.example.metringsystem.system.dao.domain;

    //import lombok.Data;

    import javax.persistence.*;
    import java.util.ArrayList;
```

```
import java.util.Date;
import java.util.List;

@Entity
@Table(name = "users")
//@Data
public class UserDTO {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private long id;

    //задолженность
    @Column(name = "arrears")
    private double arrears;

    //Номер договора
    @Column(name = "contract_number")
    private long contractNumber;

    @Column(name = "login", unique = true)
    private String login;

    @Column(name = "password")
    private String password;

    @Column(name="chat_id", unique = true)
    private long chatId;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public double getArrears() {
        return arrears;
    }

    public void setArrears(double arrears) {
        this.arrears = arrears;
    }

    public long getContractNumber() {
        return contractNumber;
    }

    public void setContractNumber(long contractNumber) {
        this.contractNumber = contractNumber;
    }
```

```
    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public long getChatId() {
        return chatId;
    }

    public void setChatId(long chatId) {
        this.chatId = chatId;
    }
}


package com.example.metringsystem.system.dao.domain;

//import lombok.Data;

import javax.persistence.*;
import java.util.Date;

@Entity
@Table(name="counters")
//@Data
public class CounterDTO  {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    // Ежемесячные показания
    @Column(name="month_counter")
    private int monthCounter;

    //Сумма к оплате за месяц
    @Column(name="payment_amount")
    private double paymentAmount;

    //Оплаченная сумма за месяц
    @Column(name="paid")
```

```
    private double paid;

    //Дата платежа
    @Column(name="payment_date", columnDefinition = "DATE")
    private Date paymentDate;

    // Просрочка платежа
    @Column(name="payment_delay")
    private double paymentDelay;

    @Column(name="user_id")
    private long userId;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public int getMonthCounter() {
        return monthCounter;
    }

    public void setMonthCounter(int monthCounter) {
        this.monthCounter = monthCounter;
    }

    public double getPaymentAmount() {
        return paymentAmount;
    }

    public void setPaymentAmount(double paymentAmount) {
        this.paymentAmount = paymentAmount;
    }

    public double getPaid() {
        return paid;
    }

    public void setPaid(double paid) {
        this.paid = paid;
    }

    public Date getPaymentDate() {
        return paymentDate;
    }

    public void setPaymentDate(Date paymentDate) {
        this.paymentDate = paymentDate;
    }
```

```
        public double getPaymentDelay() {
            return paymentDelay;
        }

        public void setPaymentDelay(double paymentDelay) {
            this.paymentDelay = paymentDelay;
        }

        public long getUserId() {
            return userId;
        }

        public void setUserId(long userId) {
            this.userId = userId;
        }
    }
```

```
    package com.example.metringsystem.system.actions;

    import
com.example.metringsystem.system.actions.reply.ReplyFlow;
    import com.example.metringsystem.system.dao.domain.UserDTO;
    import org.apache.logging.log4j.util.Strings;
    import org.springframework.dao.DuplicateKeyException;
    import org.springframework.stereotype.Component;
    import
org.telegram.telegrambots.meta.api.objects.CallbackQuery;
    import org.telegram.telegrambots.meta.api.objects.Message;
    import
org.telegram.telegrambots.meta.api.objects.inlinequery.InlineQuery
;
    import
org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKey
boardMarkup;
    import
org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.I
nlineKeyboardButton;

    import java.util.*;
    import java.util.concurrent.ConcurrentHashMap;

    @Component
    public class RegistrationAction extends BaseAction implements
IActions {

        //map to store registration for new users
        private Map<Long, UserDTO> cacheMapRegistration = new
ConcurrentHashMap<>();
        private Map<Long, Integer> cacheMapRegistrationMessage =
new ConcurrentHashMap<>();

        private Map<String, String> registerPairBind = null;
        public static final String prefix = "regUser";
```

```
        @Override
        public String getName() {
            return "registration";
        }

        @Override
        public String getPrefix() {
            return prefix;
        }

        public void registrationInit(Message msg){
            UserDTO                    newUser                    =
cacheMapRegistration.computeIfAbsent(msg.getChatId(),    u->    new
UserDTO());
            displayRegister(msg, newUser);
        }

        private void doRegisterUser(Long id) {
            cacheMapRegistration.computeIfPresent(id,     (charId,
userDTO)->{
                userDTO.setChatId(charId);
                getBotRep().createUser(userDTO);
                cacheMapRegistration.computeIfPresent(charId,  (k,
v) -> { cacheMapRegistration.remove(k); return v; });

cacheMapRegistrationMessage.computeIfPresent(charId,  (k,  v)  ->  {
cacheMapRegistrationMessage.remove(k); return v; });
                return userDTO;
            });
        }

        public StringBuffer buildRegistrationInfo(UserDTO user){
            StringBuffer sb = new StringBuffer(0);
            sb.append("Регистрация                              нового
пользователя.").append(newLine).append(newLine);

            sb

.append((Strings.isNotEmpty(user.getLogin()))?checked:" ")
                    .append(" *Login*: ")

.append((Strings.isEmpty(user.getLogin()))?noData+"              no
login":user.getLogin())
                    .append(newLine);
            sb

.append((Strings.isNotEmpty(user.getPassword()))?checked:" ")
                    .append(" *Password*: ")

.append((Strings.isEmpty(user.getPassword()))?noData+"           no
password":user.getPassword())
                    .append(newLine);
```

```
            sb

.append((user.getContractNumber()>0)?checked:" ")
                    .append(" *Contract Number*: ")
                    .append((user.getContractNumber()          ==
0)?noData+" no contract number":user.getContractNumber())
                    .append(newLine)
                    .append(newLine);
            return sb;
        }


        private void displayRegister(Message msg, UserDTO user) {
            StringBuffer sb = buildRegistrationInfo(user);

            // check if a user is fully filled
            boolean complete = (
                    Strings.isNotEmpty(user.getLogin()) &&
                    Strings.isNotEmpty(user.getPassword()) &&
                    user.getContractNumber() > 0
            );

            InlineKeyboardMarkup             markup             =
registrationButtonMarkup(complete);

            sendTextWithMarkup(msg.getChatId(),     sb.toString(),
markup);
        }

        private  void  displayRegisterEdit(Message  msg,  UserDTO
user, Integer messageId) {

            StringBuffer sb = buildRegistrationInfo(user);

            // check if a user is fully filled
            boolean complete = (
                    Strings.isNotEmpty(user.getLogin()) &&

Strings.isNotEmpty(user.getPassword()) &&
                            user.getContractNumber() > 0
            );

            InlineKeyboardMarkup             markup             =
registrationButtonMarkup(complete);

            sendEditTextWithMarkup(msg.getChatId(),
sb.toString(), messageId, markup);
        }

        private                                  InlineKeyboardMarkup
registrationButtonMarkup(boolean complete){

            String prefix = getPrefix();
```

```
            List<List<InlineKeyboardButton>>    buttons    =    new
ArrayList<>();

            List<InlineKeyboardButton>    buttonsLine1    =    new
ArrayList<>();

buttonsLine1.add(InlineKeyboardButton.builder().text("Login").call
backData(prefix+"-login").build());

buttonsLine1.add(InlineKeyboardButton.builder().text("Password").c
allbackData(prefix+"-password").build());

buttonsLine1.add(InlineKeyboardButton.builder().text("Contract
Number").callbackData(prefix+"-contractNumber").build());
            buttons.add(buttonsLine1);

            if(complete) {
                List<InlineKeyboardButton>   buttonsLine2   =   new
ArrayList<>();

buttonsLine2.add(InlineKeyboardButton.builder().text("\uD83D\uDCBE
Register").callbackData(prefix + "-doregister").build());
                buttons.add(buttonsLine2);
            }

            InlineKeyboardMarkup     markupKeyboard     =     new
InlineKeyboardMarkup();

            markupKeyboard.setKeyboard(buttons);

            return markupKeyboard;
        }


        public void handleInlineQuery(InlineQuery iq){

        }

        public void handleCallbackQuery(CallbackQuery cb) {
            ActionCommand            ac            =            new
ActionCommand(cb.getData()).parse();

cacheMapRegistrationMessage.putIfAbsent(cb.getMessage().getChatId(
), cb.getMessage().getMessageId());

            if  (ac.isAction("login")  ||  ac.isAction("password")
|| ac.isAction("contractNumber")) {
                sendOnlyTextToUser(cb.getMessage().getChatId(),
getRegisterPairBind().get(ac.getAction()));

getRws().registerReplyWait(cb.getMessage().getChatId(),     prefix,
prefix+"-"+ac.getAction(), cb.getMessage().getMessageId());
```

```
            } else if (ac.isAction("doregister")) {
                try {
                    doRegisterUser(cb.getMessage().getChatId());
                } catch (DuplicateKeyException dke){
                    sendOnlyTextToUser(
                            cb.getMessage().getChatId(),
                            "Пользователь с таким логином или с
таким Chat ID существует: "+

parseErrorMessage(dke.getMessage()));
                    return;
                } catch (Exception pe){

sendOnlyTextToUser(cb.getMessage().getChatId(),   "Ошибка!   Что-то
пошло не так: "+ pe.getMessage());
                    return;
                }
                sendOnlyTextToUser(cb.getMessage().getChatId(),
"Пользователь успешно создан!");
            }
        }

        public  Map<String, String> getRegisterPairBind() {
            if(registerPairBind == null){
                registerPairBind = new HashMap<>();

                registerPairBind.put("login", "Введите @логин");
                registerPairBind.put("password",          "Введите
@пароль");
                registerPairBind.put("contractNumber",    "Введите
@номер контракта");
            }

            return registerPairBind;
        }

        public void handleReplyMessage(Message msg, ReplyFlow rf)
{
            ActionCommand           ac           =           new
ActionCommand(rf.getCommandId()).parse();

            if(!(ac.isAction("login")  ||  ac.isAction("password")
|| ac.isAction("contractNumber"))){
                return;
            }

            UserDTO                   userDTO                   =
cacheMapRegistration.get(msg.getChatId());

            if(ac.isAction("login")){
                userDTO.setLogin(msg.getText());
            } else if(ac.isAction("password")){
                userDTO.setPassword(msg.getText());
```

```
            } else if(ac.isAction("contractNumber")){
                try {

userDTO.setContractNumber(Integer.valueOf(msg.getText()));
                } catch (NumberFormatException nfe){

sendOnlyTextToUser(msg.getChatId(),"Некоректный  формат,  Contract
Number должен быть числом");
                }
            }

            getRws().consumeReplyWait(msg.getChatId());
            //deleteLastMessage(msg.getChatId().toString(),
rf.getUpdMessageId()+1);
            //deleteLastMessage(msg.getChatId().toString(),
rf.getUpdMessageId()+2);
            Integer                editMessageId                =
cacheMapRegistrationMessage.get(msg.getChatId());

            if(editMessageId != null)
                displayRegisterEdit(msg, userDTO, editMessageId);
        }

        private String parseErrorMessage(String message){
            if(Strings.isNotEmpty(message)){
                String[] lines = message.split("\n");

                return              Arrays.stream(lines).filter(l-
>l.startsWith("Detail:")).findFirst().orElse("");

            } else {
                return message;
            }
        }
    }


    package com.example.metringsystem.system.actions;

    public interface IActions {

        public final String  newLine = "\n";
        public final String  tabLine = "\t";
        public final String  noData = "\uD83D\uDEAB";
        public final String  checked = "\u2714";

        String getName();
        String getPrefix();
    }


    package com.example.metringsystem.system.actions;
```

```
import
com.example.metringsystem.system.actions.reply.ReplyFlow;
import
com.example.metringsystem.system.dao.domain.CounterDTO;
import com.example.metringsystem.system.dao.domain.UserDTO;
import org.apache.logging.log4j.util.Strings;
import org.springframework.stereotype.Component;
import
org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import
org.telegram.telegrambots.meta.api.objects.CallbackQuery;
import org.telegram.telegrambots.meta.api.objects.Message;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKey
boardMarkup;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.I
nlineKeyboardButton;
import
org.telegram.telegrambots.meta.exceptions.TelegramApiException;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Component
public class CommonAction extends BaseAction implements
IActions{

    public static final String prefix = "comact";

    public final List<String> keyboardReplyNames =
List.of("Сумма к оплате", "Задолженность", "Показания", "Добавить
текущий платеж");


    @Override
    public String getName() {
        return "common";
    }

    @Override
    public String getPrefix() {
        return prefix;
    }

    public void handleKeyboardReply(Message msg, String
replayName){
        if(Strings.isEmpty(replayName))
            return;

        Optional<UserDTO> user =
getBotRep().getUserByChatId(msg.getChatId());
```

```
            if(!user.isPresent()){
                sendOnlyTextToUser(msg.getChatId(),  "Пользователь
не зарегистрирован! Пройти регистрацию /registration");
                return;
            }

            Long chatId = msg.getChatId();

            if(replayName.equals("Сумма к оплате")){
                user.ifPresent(u->{
                    Optional<CounterDTO>         counter         =
getBotRep().getCountersCurrentByUserId(u.getId());

                    counter.ifPresentOrElse((c)->{

                        StringBuffer sb = new StringBuffer();
                        sb
                                .append("Сумма    к    оплате    за
текущий месяц: *")
                                .append(c.getPaymentAmount())
                                .append("*");

                        sendOnlyTextToUser(chatId,
sb.toString());
                    }, ()->{
                        sendOnlyTextToUser(chatId,         "Записи
показания за текущий месяц не найдено");
                    });
                });

            } else if(replayName.equals("Задолженность")){

                user.ifPresent(u->{
                    Optional<CounterDTO>         counter         =
getBotRep().getCountersCurrentByUserId(u.getId());

                    counter.ifPresentOrElse((c)->{

                        StringBuffer sb = new StringBuffer();
                        sb
                                .append("Задолженнось  за  текущий
месяц: *")
                                .append(c.getPaymentDelay())
                                .append("*");

                        sendOnlyTextToUser(chatId,
sb.toString());
                    }, ()->{
                        sendOnlyTextToUser(chatId,         "Записи
показания за текущий месяц не найдено");
                    });
                });
```

```
            } else if(replayName.equals("Показания")){
                inlineMeterKeyboard(msg);
            }    else    if(replayName.equals("Добавить    текущий
платеж")){
                sendOnlyTextToUser(chatId, "Введите сумму платежа
за текущий месяц:");
                getRws().registerReplyWait(chatId,         prefix,
prefix+"-currentpayment", msg.getMessageId());
            }

        }

        public void inlineMeterKeyboard(Message msg){
            SendMessage s = new SendMessage();
            s.setChatId(msg.getChatId().toString());    //    Боту
может писать не один человек, и поэтому чтобы отправить сообщение,
грубо говоря нужно узнать куда его отправлять
            s.enableMarkdown(true);

            StringBuffer sb = new StringBuffer();
            sb
                    .append("Показания счетчиков:");
            s.setText(sb.toString());

            List<List<InlineKeyboardButton>>    buttons    =    new
ArrayList<>();

            List<InlineKeyboardButton>     buttonRow     =     new
ArrayList<>();

buttonRow.add(InlineKeyboardButton.builder().text("Текущий").callb
ackData(getPrefix()+"-metercur").build());

buttonRow.add(InlineKeyboardButton.builder().text("Предыдущий").ca
llbackData(getPrefix()+"-meterprew").build());

//buttonRow.add(InlineKeyboardButton.builder().text("Все").callbac
kData(getPrefix()+"-meterall").build());
            buttons.add(buttonRow);

            buttonRow = new ArrayList<>();

buttonRow.add(InlineKeyboardButton.builder().text("Внести
значения").callbackData(getPrefix()+"-meterenter").build());
            buttons.add(buttonRow);

            InlineKeyboardMarkup     markupKeyboard     =     new
InlineKeyboardMarkup();

            markupKeyboard.setKeyboard(buttons);
            s.setReplyMarkup(markupKeyboard);
```

```
            try {
                getBot().execute(s);
            } catch (TelegramApiException tae) {
                tae.printStackTrace();
            }
        }

        public void handleCallbackQuery(CallbackQuery cb) {

            if(cb.getData().startsWith(getPrefix())){

                ActionCommand ac = new
ActionCommand(cb.getData()).parse();
                Optional<UserDTO> user =
getBotRep().getUserByChatId(cb.getFrom().getId());

                Long chatId = cb.getMessage().getChatId();
                if (ac.isAction("metercur")) {
                    user.ifPresent(u->{
                        Optional<CounterDTO> counter =
getBotRep().getCountersCurrentByUserId(u.getId());

                        counter.ifPresentOrElse((c)->{

                            StringBuffer sb = new StringBuffer();
                            sb
                                    .append("Показания на текущий
месяц: *")
                                    .append(c.getMonthCounter())
                                    .append("*");

                            sendOnlyTextToUser(chatId,
sb.toString());
                        }, ()->{
                            sendOnlyTextToUser(chatId,     "Записи
показания за текущий месяц не найдено");
                        });
                    });
                } else if (ac.isAction("meterprew")) {
                    user.ifPresent(u->{
                        Optional<CounterDTO> counter =
getBotRep().getCountersPrevMonthByUserId(u.getId());

                        counter.ifPresentOrElse((c)->{

                            StringBuffer sb = new StringBuffer();
                            sb
                                    .append("Показания за
предыдущий месяц: *")
                                    .append(c.getMonthCounter())
                                    .append("*");
```

```
                                sendOnlyTextToUser(chatId,
sb.toString());
                            }, ()->{
                                sendOnlyTextToUser(chatId,     "Записи
показания за предыдущий месяц не найдено");
                            });
                        });
                    } else if (ac.isAction("meterenter")) {

//action.sendMessageToAllUser(cb.getMessage(), "delete");
                        sendOnlyTextToUser(chatId, "Введите показания
за текущий месяц:");
                        getRws().registerReplyWait(chatId,     prefix,
prefix+"-"+ac.getAction(), cb.getMessage().getMessageId());
                    }
                }
            }

            public void handleReplyMessage(Message msg, ReplyFlow rf)
{
                ActionCommand            ac            =            new
ActionCommand(rf.getCommandId()).parse();

                if(!(ac.isAction("meterenter")                          ||
ac.isAction("currentpayment"))){
                    return;
                }

                Long chatId = msg.getChatId();
                Optional<UserDTO>                user                =
getBotRep().getUserByChatId(chatId);

                if(ac.isAction("meterenter")){
                    try {
                        Integer                meter                =
Integer.valueOf(msg.getText());

                        user.ifPresent(u->{
                            Optional<CounterDTO>       counter       =
getBotRep().getCountersCurrentByUserId(u.getId());

                            counter.ifPresentOrElse((c)->{

                                c.setMonthCounter(meter);

                                getBotRep().updateCounters(c);

                                StringBuffer sb = new StringBuffer();
                                sb
                                        .append("Показания на текущий
месяц добалены");
```

```
                                sendOnlyTextToUser(chatId,
sb.toString());

getRws().consumeReplyWait(msg.getChatId());
                            }, ()->{
                                sendOnlyTextToUser(chatId,     "Записи
показания за текущий месяц не найдено");
                            });
                        });

                    } catch (NumberFormatException nfe){

sendOnlyTextToUser(msg.getChatId(),"Некоректный  формат,  Показания
должен быть числом");
                    }
                } else if(ac.isAction("currentpayment")){
                    try {
                        Double                paidSumm                =
Double.valueOf(msg.getText());

                        user.ifPresent(u->{
                            Optional<CounterDTO>       counter       =
getBotRep().getCountersCurrentByUserId(u.getId());

                            counter.ifPresentOrElse((c)->{

                                c.setPaid(paidSumm);

                                getBotRep().updateCounters(c);

                                StringBuffer sb = new StringBuffer();
                                sb.append("Платеж  за  текущий  месяц
добален");

                                sendOnlyTextToUser(chatId,
sb.toString());

getRws().consumeReplyWait(msg.getChatId());
                            }, ()->{
                                sendOnlyTextToUser(chatId,     "Записи
показания за текущий месяц не найдено");
                            });
                        });

                    } catch (NumberFormatException nfe){

sendOnlyTextToUser(msg.getChatId(),"Некоректный    формат,    Сумма
платежа должен быть числом");
                    }
                }
            }
        }
```

```
package com.example.metringsystem.system.actions;

import com.example.metringsystem.BotRep;
import com.example.metringsystem.TelegramBot;
import
com.example.metringsystem.system.actions.reply.ReplyWaitService;
//import lombok.extern.slf4j.Slf4j;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import
org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import
org.telegram.telegrambots.meta.api.methods.updatingmessages.Delete
Message;
import
org.telegram.telegrambots.meta.api.methods.updatingmessages.EditMe
ssageText;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKey
boardMarkup;
import
org.telegram.telegrambots.meta.exceptions.TelegramApiException;

//@Slf4j
@Component
public class BaseAction {
    private TelegramBot bot;

    @Autowired
    public void setBot(TelegramBot bot) {
        this.bot = bot;
    }

    public TelegramBot getBot() {
        return bot;
    }


    private BotRep botRep;

    @Autowired
    public void setBotRep(BotRep botRep) {
        this.botRep = botRep;
    }

    public BotRep getBotRep() {
        return botRep;
    }

    private ReplyWaitService rws;
```

```
    public ReplyWaitService getRws() {
        return rws;
    }

    @Autowired
    public void setRws(ReplyWaitService rws) {
        this.rws = rws;
    }

    public boolean sendOnlyTextToUser(Long userId, String
text) {
        if(userId == null)
            return false;

        SendMessage s = new SendMessage();
        s.setChatId(userId.toString());
        s.disableWebPagePreview();
        s.setParseMode("Markdown");
        s.setText(text);

        try {
            getBot().execute(s);
            //log.info(userId+": " +text);
            System.out.println(userId+": " +text);
            return true;
        } catch (TelegramApiException e) {
            e.printStackTrace();
        }

        return false;
    }

    public void sendTextWithMarkup(Long chatId, String text,
InlineKeyboardMarkup replyMarkup) {

        SendMessage s = new SendMessage();
        s.setChatId(chatId.toString());
        s.enableMarkdown(true);

        s.setText(text);
        s.setReplyMarkup(replyMarkup);

        try {
            getBot().execute(s);
            //log.info(chatId+": " +text);
            System.out.println(chatId+": " +text);
        } catch (TelegramApiException tae) {
            tae.printStackTrace();
        }
    }

    public void sendEditTextWithMarkup(Long chatId, String
text, Integer messageId, InlineKeyboardMarkup replyMarkup) {
```

```
            EditMessageText em = new EditMessageText();

            em.setChatId(chatId.toString());
            em.enableMarkdown(true);
            em.setMessageId(messageId);

            em.setText(text);
            em.setReplyMarkup(replyMarkup);

            try {
                getBot().execute(em);
            } catch (TelegramApiException tae) {
                tae.printStackTrace();
            }
        }

        public synchronized void deleteLastMessage(String chatId,
Integer messageId){
            DeleteMessage   dm   =   new   DeleteMessage(chatId,
messageId);
            try {
                getBot().execute(dm);
            } catch (TelegramApiException e) {
                e.printStackTrace();
            }
        }
    }


    package com.example.metringsystem.system.actions;

    import com.example.metringsystem.BotRep;
    import com.example.metringsystem.TelegramBot;
    import
com.example.metringsystem.system.actions.reply.ReplyWaitService;
    //import lombok.extern.slf4j.Slf4j;
    import
org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.stereotype.Component;
    import
org.telegram.telegrambots.meta.api.methods.send.SendMessage;
    import
org.telegram.telegrambots.meta.api.methods.updatingmessages.Delete
Message;
    import
org.telegram.telegrambots.meta.api.methods.updatingmessages.EditMe
ssageText;
    import
org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKey
boardMarkup;
    import
org.telegram.telegrambots.meta.exceptions.TelegramApiException;
```

```
//@Slf4j
@Component
public class BaseAction {
    private TelegramBot bot;

    @Autowired
    public void setBot(TelegramBot bot) {
        this.bot = bot;
    }

    public TelegramBot getBot() {
        return bot;
    }


    private BotRep botRep;

    @Autowired
    public void setBotRep(BotRep botRep) {
        this.botRep = botRep;
    }

    public BotRep getBotRep() {
        return botRep;
    }

    private ReplyWaitService rws;

    public ReplyWaitService getRws() {
        return rws;
    }

    @Autowired
    public void setRws(ReplyWaitService rws) {
        this.rws = rws;
    }

    public  boolean  sendOnlyTextToUser(Long  userId,  String
text) {
        if(userId == null)
            return false;

        SendMessage s = new SendMessage();
        s.setChatId(userId.toString());
        s.disableWebPagePreview();
        s.setParseMode("Markdown");
        s.setText(text);

        try {
            getBot().execute(s);
            //log.info(userId+": " +text);
            System.out.println(userId+": " +text);
```

```
                return true;
            } catch (TelegramApiException e) {
                e.printStackTrace();
            }

            return false;
        }

        public void sendTextWithMarkup(Long chatId, String text,
InlineKeyboardMarkup replyMarkup) {

            SendMessage s = new SendMessage();
            s.setChatId(chatId.toString());
            s.enableMarkdown(true);

            s.setText(text);
            s.setReplyMarkup(replyMarkup);

            try {
                getBot().execute(s);
                //log.info(chatId+": " +text);
                System.out.println(chatId+": " +text);
            } catch (TelegramApiException tae) {
                tae.printStackTrace();
            }
        }

        public void sendEditTextWithMarkup(Long chatId, String
text, Integer messageId, InlineKeyboardMarkup replyMarkup) {

            EditMessageText em = new EditMessageText();

            em.setChatId(chatId.toString());
            em.enableMarkdown(true);
            em.setMessageId(messageId);

            em.setText(text);
            em.setReplyMarkup(replyMarkup);

            try {
                getBot().execute(em);
            } catch (TelegramApiException tae) {
                tae.printStackTrace();
            }
        }

        public synchronized void deleteLastMessage(String chatId,
Integer messageId){
            DeleteMessage dm = new DeleteMessage(chatId,
messageId);
            try {
                getBot().execute(dm);
            } catch (TelegramApiException e) {
```

```
                e.printStackTrace();
            }
        }
    }


    package com.example.metringsystem.system.actions.reply;

    import org.springframework.stereotype.Service;
    import
org.springframework.transaction.annotation.Transactional;

    import java.util.HashMap;
    import java.util.Map;

    @Service
    @Transactional
    public class ReplyWaitServiceImpl implements ReplyWaitService
{

        private final Map<Long, ReplyFlow> replyWaitCache = new
HashMap<>();

        @Override
        public Map<Long, ReplyFlow> getReplyWaitCache() {
            return replyWaitCache;
        }

        @Override
        public boolean hasReply(Long userId, String command) {

            ReplyFlow rf = replyWaitCache.get(userId);

            return rf != null && (command != null &&
command.equals(rf.getCommand()));
        }

        @Override
        public void registerReplyWait(Long userId, String
command, String commandId, Integer updMessageId) {
            replyWaitCache.put(userId, new ReplyFlow(command,
commandId, updMessageId));
        }

        @Override
        public void registerReplyWait(Long userId, String
command, String commandId) {
            replyWaitCache.put(userId, new ReplyFlow(command,
commandId, null));
        }

        @Override
        public ReplyFlow consumeReplyWait(Long userId) {
```

```
            ReplyFlow rf = replyWaitCache.get(userId);

            if(rf != null){
                replyWaitCache.remove(userId);
            }

            return rf;
        }

    }
    package com.example.metringsystem.system.actions.reply;

    import java.util.Map;

    public interface ReplyWaitService {

        Map<Long, ReplyFlow> getReplyWaitCache();

        boolean hasReply(Long userId, String commandId);

        void registerReplyWait(Long userId, String command,
String commandId, Integer updMessageId);

        void registerReplyWait(Long userId, String command,
String commandId);

        ReplyFlow consumeReplyWait(Long userId);
    }


    package com.example.metringsystem.system.actions.reply;

    public class ReplyFlow {

        private String command;
        private String commandId;
        private Integer updMessageId;


        public ReplyFlow(String command, String commandId,
Integer updMessageId) {
            this.command = command;
            this.commandId = commandId;
            this.updMessageId = updMessageId;
        }

        public String getCommand() {
            return command;
        }

        public String getCommandId() {
            return commandId;
        }
```

```
    public Integer getUpdMessageId() {
        return updMessageId;
    }
}
```

## Приложение В

(Обязательное)

Система учёта показаний счётчиков воды с использованием чат-бота

Презентация

# Система учёта показаний счётчиков воды с использованием чат-бота

Автор работы:
студент 4 курса гр.20ВВП1 Сафронов Денис
Научный руководитель:
к.т.н., доцент Токарев Андрей Николаевич

## Актуальность проекта

- Доступ к базе показаний со своего смартофна
- Учёт расхода ресурсов и средств
- Своевременное оповещение об оплате или внесении показаний

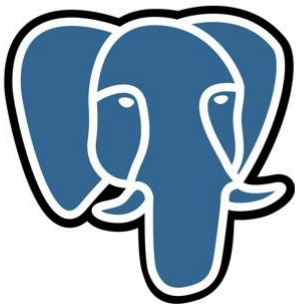
# Цель работы:

Изучение систем учёта показаний счётчиков и разработка системы для сбора и обработки показаний с использованием популярной социальной сети - телеграм

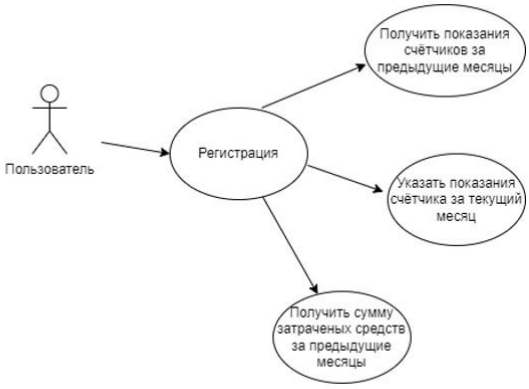
## Анализ аналогов

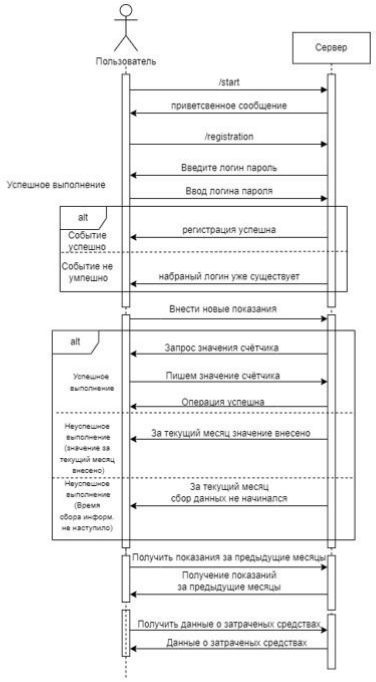
| Функционал | mos.ru | квартплата онлайн | ГИС ЖКХ |
|---|---|---|---|
| Авторизация | + | + | - |
| Возможность просмотра показаний за прошедшие месяцы | + | + | + |
| Возможность просмотра суммы оплаты за прошедшие месяцы | + | + | - |
| Возможность указать расход за текущий месяц в любой месяц | - | - | - |
| Доступ для всех жителей РФ | - | + | + |

# Выбор технологий

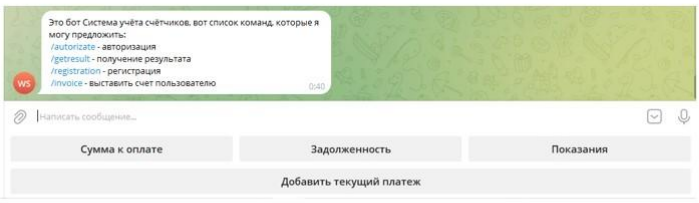


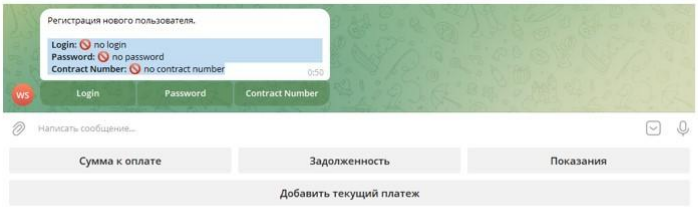
# UML диаграммы:



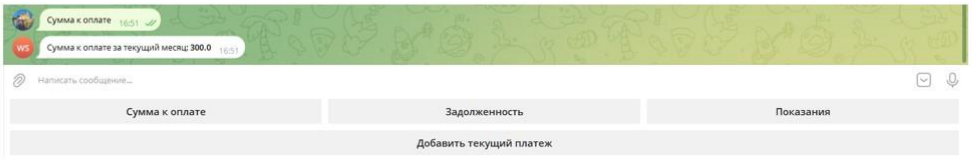



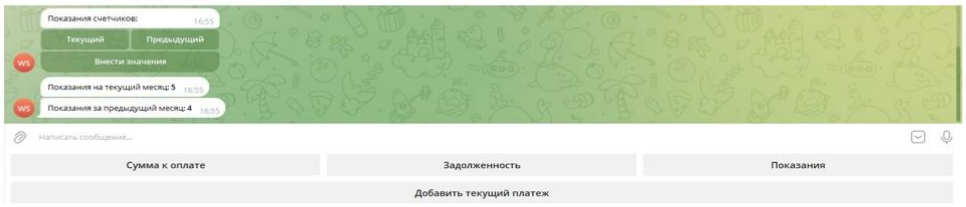
# Запуск бота и регистрация:




# Проверка основных функций бота:



# Заключение:

В ходе выполнения данной работы был проведен анализ современных систем учёта показаний счётчиков, выявлены их особенности и основные характеристики, удовлетворяющие потребности в планировании и организации работы предприятия. Получены навыки проектирования телеграм-ботов.

В результате работы была спроектирована, разработана и отлажена система учёта показаний счётчиков на базе телеграм-бота.