МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ “ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”

Факультет компьютерных наук
Кафедра программирования и информационных технологий

Система управления сбором показаний индивидуальных приборов учёта в многоквартирных домах и выставление счетов за потребленные услуги.

Курсовая работа по дисциплине «Технологии программирования»

09.03.02 *Информационные системы и технологии*
Программная инженерия в информационных системах

Преподаватель _____В.С. Тарасов, ст. преподаватель __.__20__
Обучающийся _____Д.И. Белашков, 3 курс, д/о
Обучающийся _____С.С. Крупенин, 3 курс, д/о
Обучающийся _____Д.В. Макушин, 3 курс, д/о
Руководитель _____К.В Сиволапов, преподаватель

Воронеж 2021

Содержание

Введение

В современном мире практически все люди пользуются услугами, оказываемыми генерирующими компаниями – это предоставление доступа к горячему и холодному водоснабжению, электричеству, отоплению. В конце каждого месяца происходит передача показаний по данным услугам и их оплата. Чаще всего каждая компания присылает квитанции на каждый лицевой счет, причем существуют различные способы оплаты и регистрации показаний.

Можно выделить три основных варианта работы с квитанцией:

Первый вариант – рукописное заполнение квитанций и посещение местного отделения банка или почты. В этом случае заполнение квитанции не представляет проблемы, но часто возникают трудности в виде траты времени на посещение банка, особенно если приходится долго стоять в очереди. При этом, когда сотрудник оформляет квитанцию, есть вероятность ошибки, из-за которой оплата может пройти на другой счет.

Второй вариант – заполнение квитанции и оплата с использованием приложения того или иного банка. Данный способ имеет преимущество в виде удобства заполнения и оплаты квитанций, не выходя из дома. Но при этом есть недостатки, из которых главными являются поиск нужной компании, отдельная оплата каждой из квитанций, и ввод лицевых счетов, которые представляют из себя длинные числовые последовательности, в которых легко ошибиться, из-за чего можно оплатить совершенно другой счет.

Третий вариант – совершать учет данных счетчика и производить оплату через унифицированное приложение коммунальных услуг, предоставляемых генерирующей компанией, которое дает пользователю информацию только о своих счетчиках и позволяет совершать их контроль и оплату показаний.

Таким образом целью, нашей работы является создание такой системы, которая позволит легко оплачивать свои счета в несколько кликов и полностью избавит пользователей от ошибки оплаты другого счета.

В свою очередь компания сможет легко отслеживать потребление каждого пользователя, позволяя перейти полностью на систему электронной оплаты без использования бумажных квитанций и посредника в виде почтальона.

1. Используемые определения

Таблица 1 - Используемые определения

Генерирующая компания	Компания-поставщик коммунальных услуг
Коммунальные услуги	Услуги, предоставляемые пользователем приложения генерирующей компанией
Лицевой счет	Бухгалтерский счет для ведения расчетов с физическими и юридическими лицами. На лицевой счет приписано несколько (в том числе и ни одного) счетчиков
Личный кабинет	Панель управления лицевым счетом для пользователя
Тарифный план	Стоимость единицы услуги за месяц

2. Анализ предметной области

2.1.Постановка задачи

Целью данного курсового проекта является разработка самостоятельной системы управления сбором показаний индивидуальных приборов учёта в многоквартирных домах и выставление счетов за потребленные услуги, которая позволит упростить процесс учета счетчиков, и оплаты их как по отдельности, так и группой. При этом в данной системе предусмотрено предоставление возможности оплаты за тот или иной счет другому пользователю.

Данная система разделена на две составные части: приложение-личный кабинет и управляющая система генерирующей компании. Приложение предназначено для управления личным кабинетом генерирующей компании, управляющая система предназначена для автоматизации начисления платежей по коммунальным услугам, предоставляемых генерирующей компанией и управления лицевыми счетами пользователей.

Система решает следующие задачи:

— Передача показаний счетчиков пользователем.

— Формирование платежной квитанции на лицевой счет.

— Сохранение истории платежей.

— Сбор статистики потребления коммунальных услуг.

2.2.Анализ существующих решений

Аналогом системы можно считать приложение-личный кабинет Квадры, в котором возможна только оплата сразу всех счетчиков. Однако, в этом приложении можно выделить простой и понятный интерфейс, возможность добавления нескольких лицевых счетов и возможность работы с приложением без авторизации.

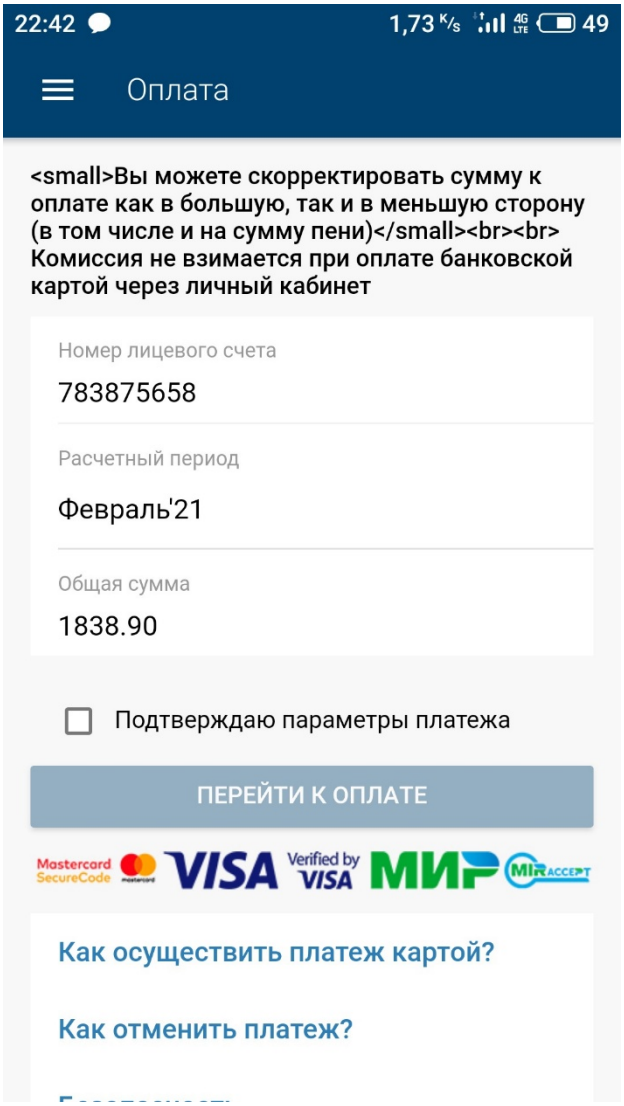

Рисунок 1 - Оплата задолженности в «Квадра»

Исходя из достоинств и недостатков данного приложения, можно выделить, что приложение системы управления сбором показаний индивидуальных приборов учёта в многоквартирных домах и выставление счетов за потребленные услуги должно иметь возможность выбирать счетчики для оплаты и передачи показаний, но при этом сохранять простоту интерфейса и достаточную функциональность.

2.3.Пользователи системы

Авторизованный пользователь – пользователь, успешно прошедший регистрацию и вошедший в личный кабинет. Возможности:

— привязать несколько лицевых счетов к аккаунту пользователя;

— передать показания счетчика с лицевых счетов, привязанных к аккаунту пользователя;

— получить квитанцию на оплату лицевого счета, привязанного к аккаунту пользователя;

— просматривать статистику/историю оплат.

Неавторизованный пользователь – пользователь, не пожелавший регистрировать личный кабинет. Возможности:

— получить квитанцию на оплату по номеру лицевого счета;

— передать показания по счетчикам по номеру лицевого счета.

Администратор – сотрудник компании, который имеет доступ к базе данных системы. Возможности:

— создавать новые лицевые счета;

— изменять и удалять лицевые счета;

— привязать к лицевым счетам новые счетчики и удалять их.

2.4.Продуктовые воронки

Рассмотрим количество шагов, которые необходимо пройти до основных функций приложения – оплата и передача показаний. Пользователь переходит по кнопке «Оплатить» на экран, где выбирает нужные для оплаты счетчики, после чего переходит на экран, где заполняет текущие показания (это можно сделать и без оплаты задолженности), после чего формируется квитанции для оплаты и осуществляется переход на экран выбора платежной системы. Таким

образом, всего за 3 экрана из бокового меню можно совершить оплату или передачу показаний.

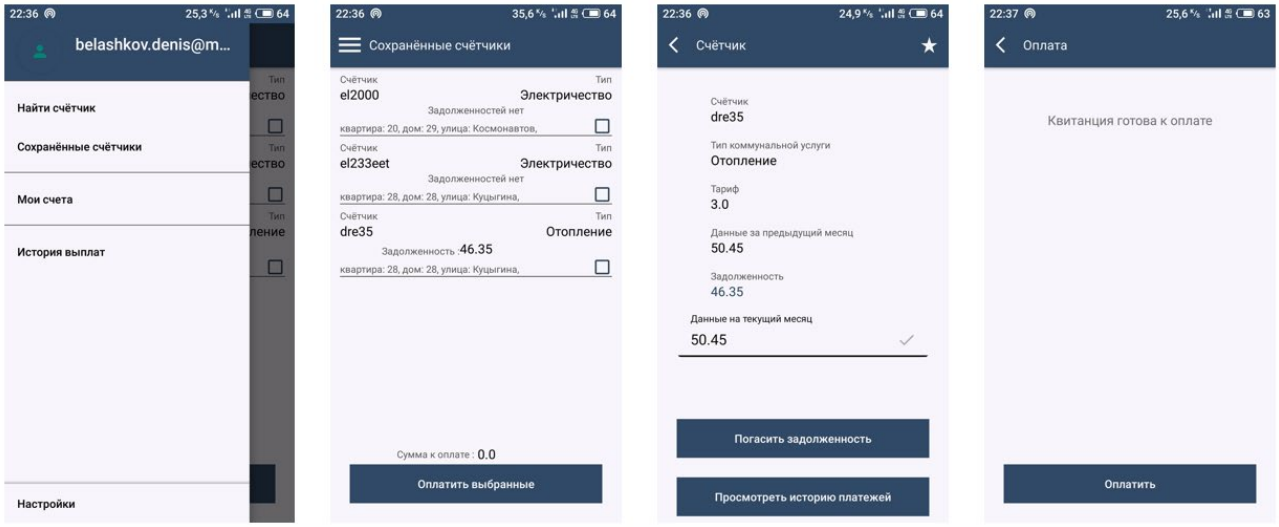

Рисунок 1 - Продуктовая воронка оплаты и передачи показаний счетчиков

2.5.Графическое описание работы системы

IDEF0 диаграмма

Рассмотрим основной бизнес-процесс на примере контекстной диаграммы. Данная диаграмма является общим видением процесса работы Системы.

Работу системы регулируют Правила оплаты, Законы РФ, Пользовательское соглашение и Уровни доступа к Системе, определяемые Администратором сервиса.

Работу системы обеспечивают Администратор Сервиса и Исходная БД.

На вход системы поступает Пользователь (авторизованный или нет), на выходе система выдает Квитанции об оплате и Измененное состояние счетчика.

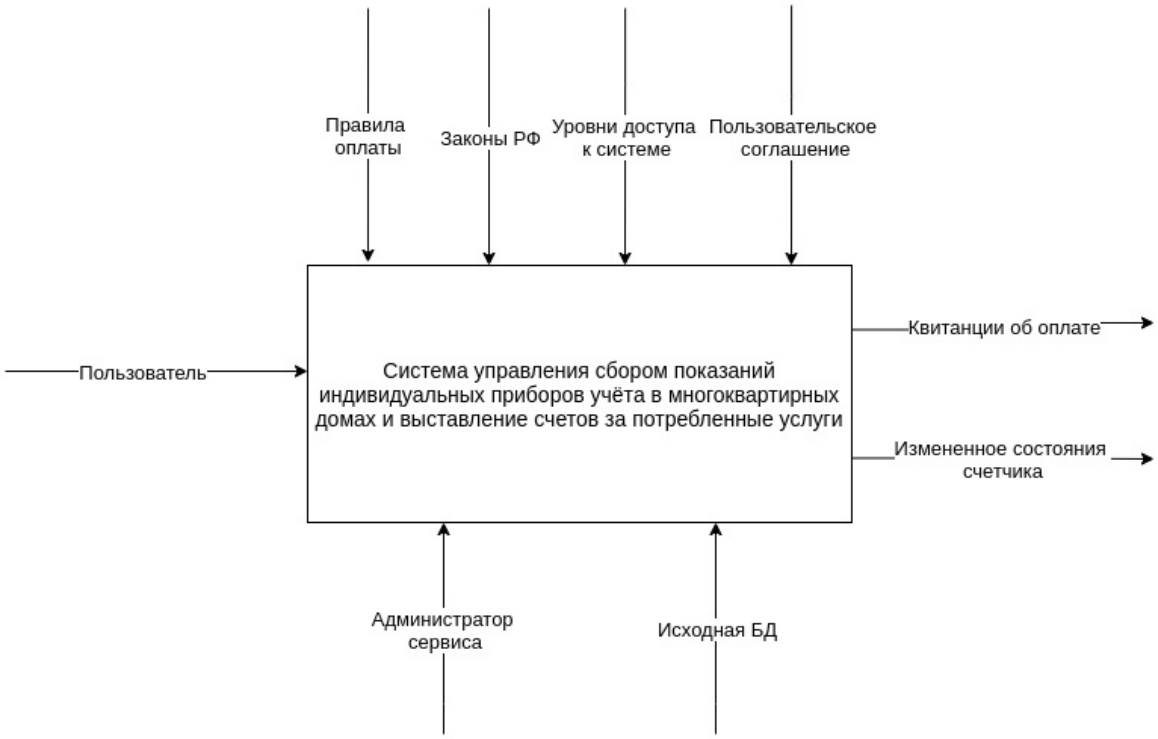

Рисунок 2 - IDEF0

Диаграмма прецедентов

Диаграммы прецедентов определяют действия и отношения актеров (действующих лиц системы) между собой и их действия по отношению к системе.

Функции неавторизованного пользователя включаются в себя: Регистрация, Оплата, Ввод показаний, Получение квитанции, Просмотр счетчиков. Авторизованный пользователь обладает теми же функциями, что и неавторизованный, а также имеет возможность просматривать истории платежей и счетов.

Администратор обладает функциями Добавление и редактирование информации и Редактирования пользовательской информации.

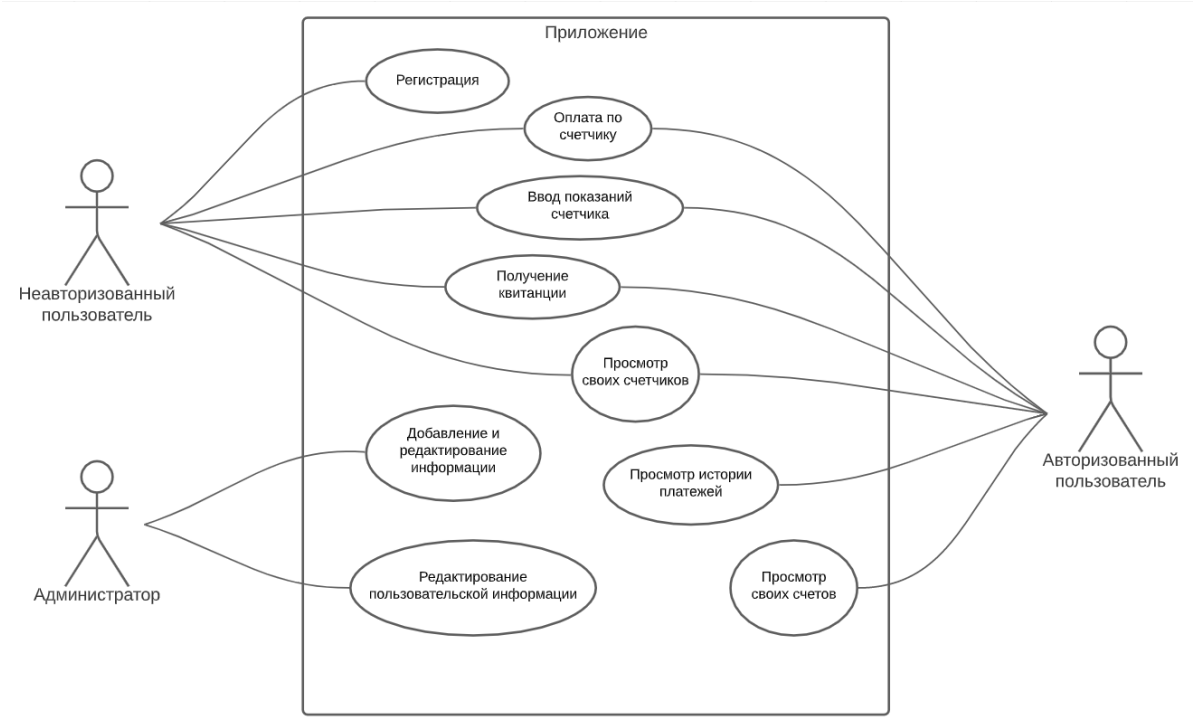

Рисунок 3 - Диаграмма прецедентов

Диаграмма классов

Диаграмма классов определяет классы системы, их атрибуты, методы и взаимодействия. В данной системе используется несколько основных классов-модулей и контроллер для взаимодействия между ними.

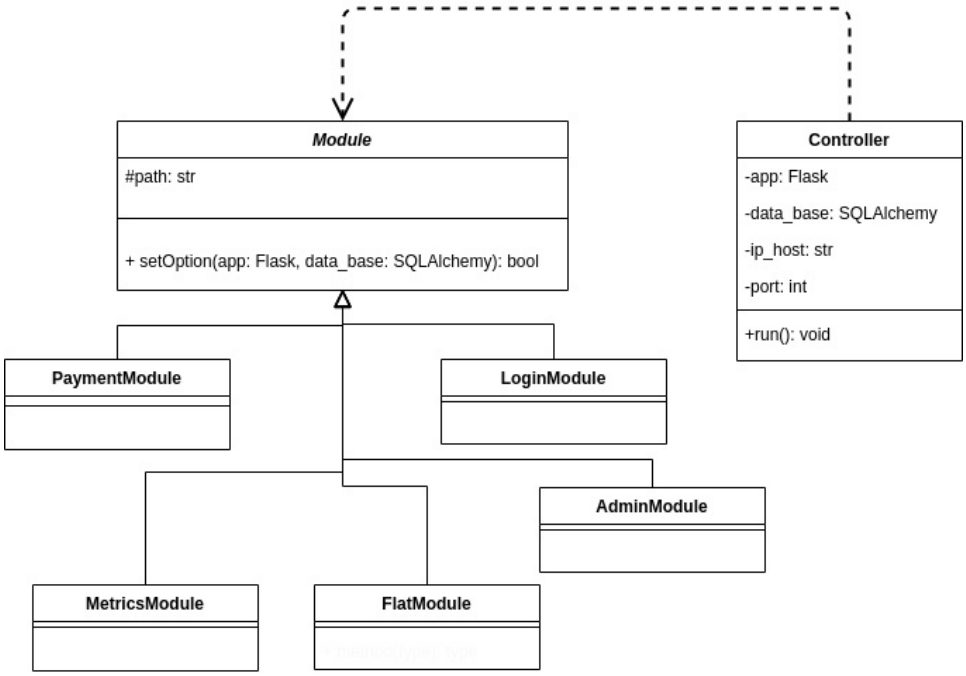

Рисунок 4 - Диаграмма классов

Диаграмма объектов

Диаграмма объектов определяет множество объектов-экземпляров классов и отношений в некоторый момент времени. Данные модули будут находиться на сервере, к которому будет обращаться пользователь через приложение.

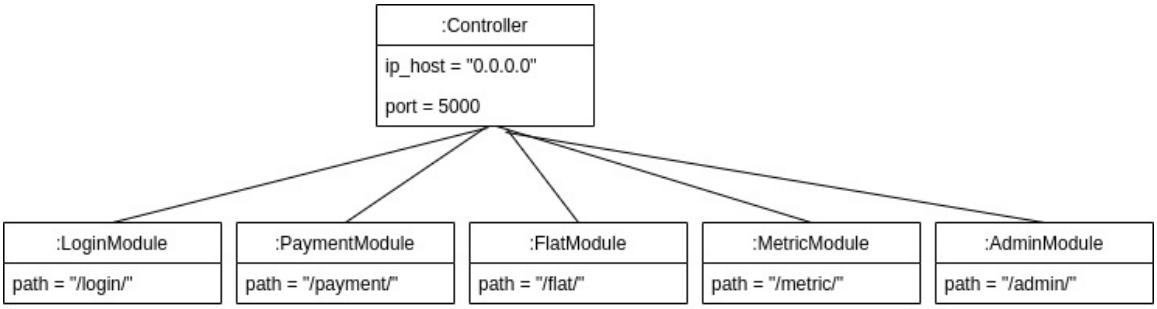

Рисунок 5 - Диаграмма объектов

Диаграмма активностей

Диаграммы активностей для передачи показаний и оплаты задолженностей поясняют диаграмму последовательности, моделируя поведение системы и раскрывая детали алгоритмической реализации операций.

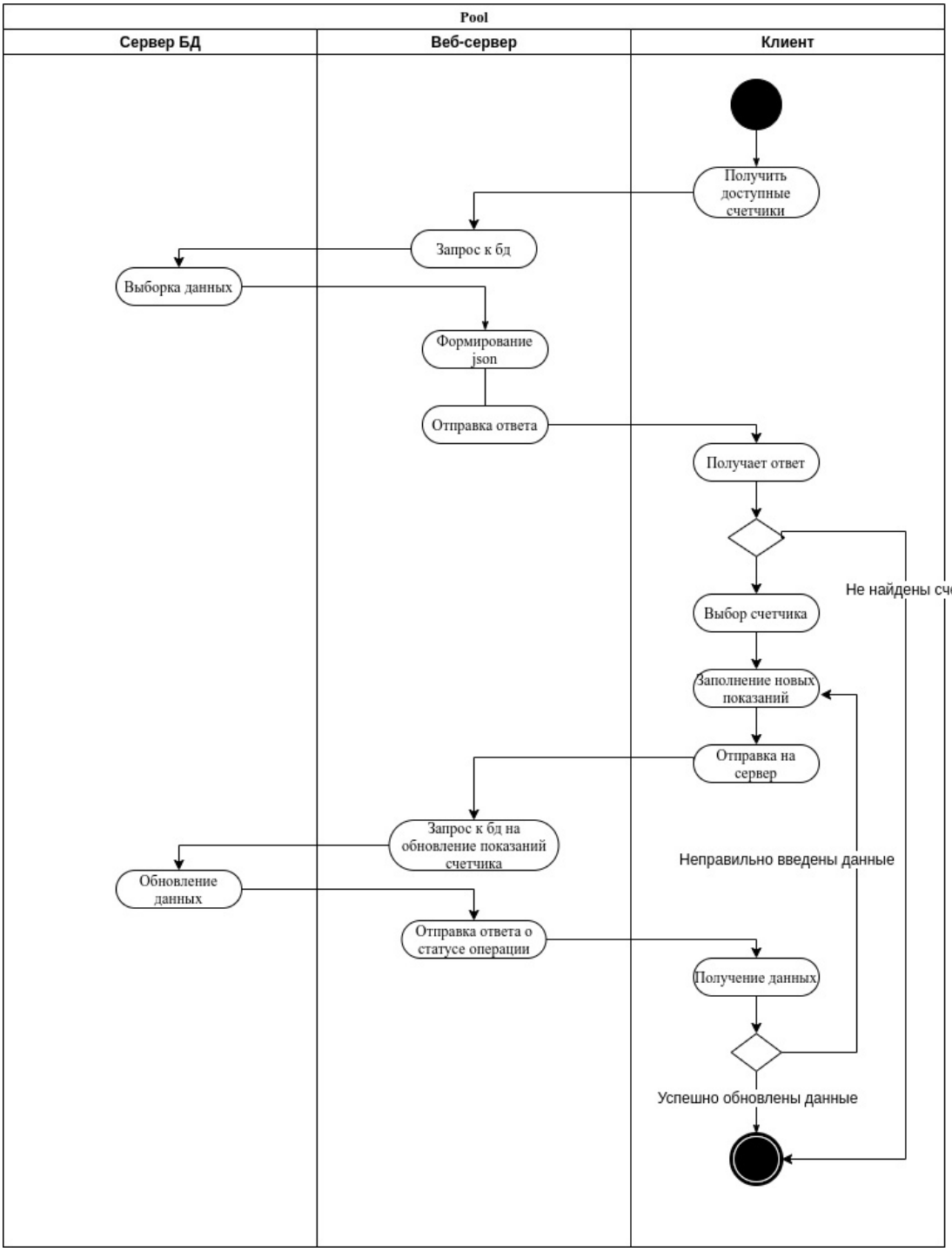

Рисунок 6 – Диаграмма активностей. Передача показаний.

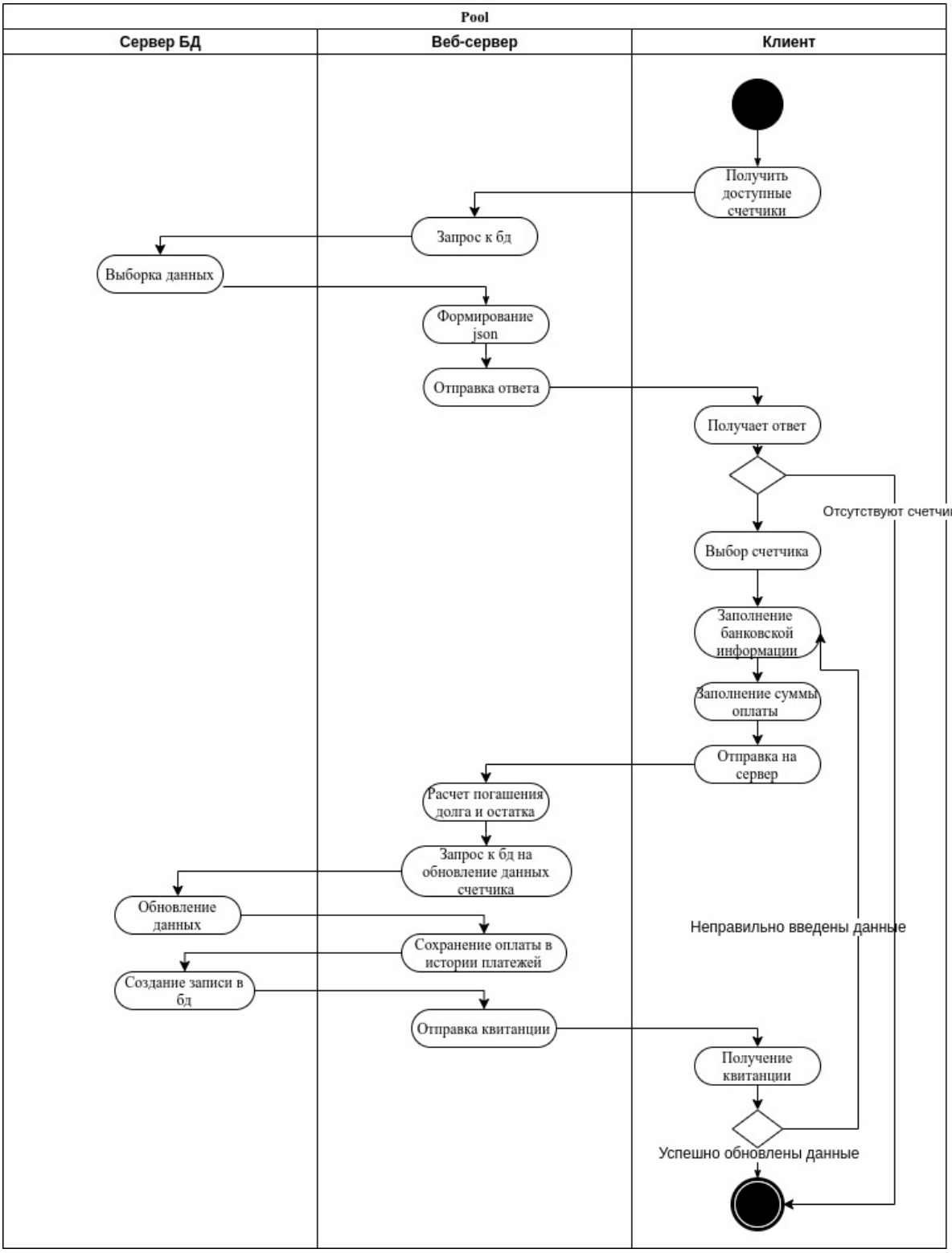

Рисунок 7 – Диаграмма активностей. Оплата задолженностей.

Диаграмма развертывания

Диаграмма развертывания определяет аппаратные компоненты, как осуществляется их работы на узлах и как части системы соединяются друг с другом. Application – клиент-приложение для пользователя.

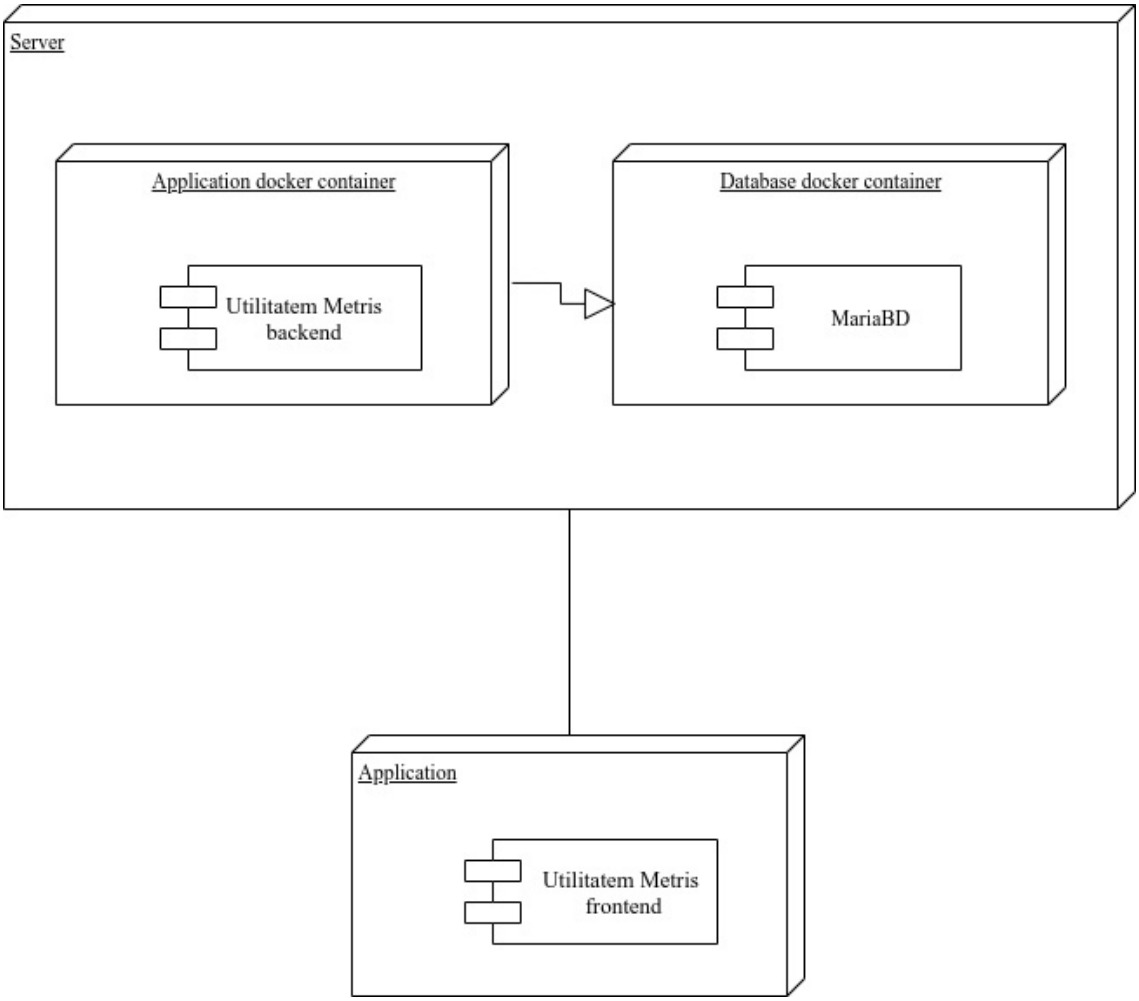

Рисунок 8 – Диаграмма развертывания

Диаграмма последовательности

Диаграмма последовательности нужна для отображения взаимодействий объектов и субъектов в динамике, отображая временные особенности передачи и приема сообщений между объектами и субъектами.

Пользователь отправляет показания счетчиков Системе, она отправляет ему результат подтверждения показаний. После этого, если есть необходимость в оплате, Пользователь заполняет форму для оплаты, которую создала Систему, Пользователь подтверждает факт оплаты, и форма удаляется. Так же у Пользователя есть возможность управления личным кабинетом. Администратор имеет право на создание новых личных кабинетов в Системе, и он получает от нее запросы от пользователей.

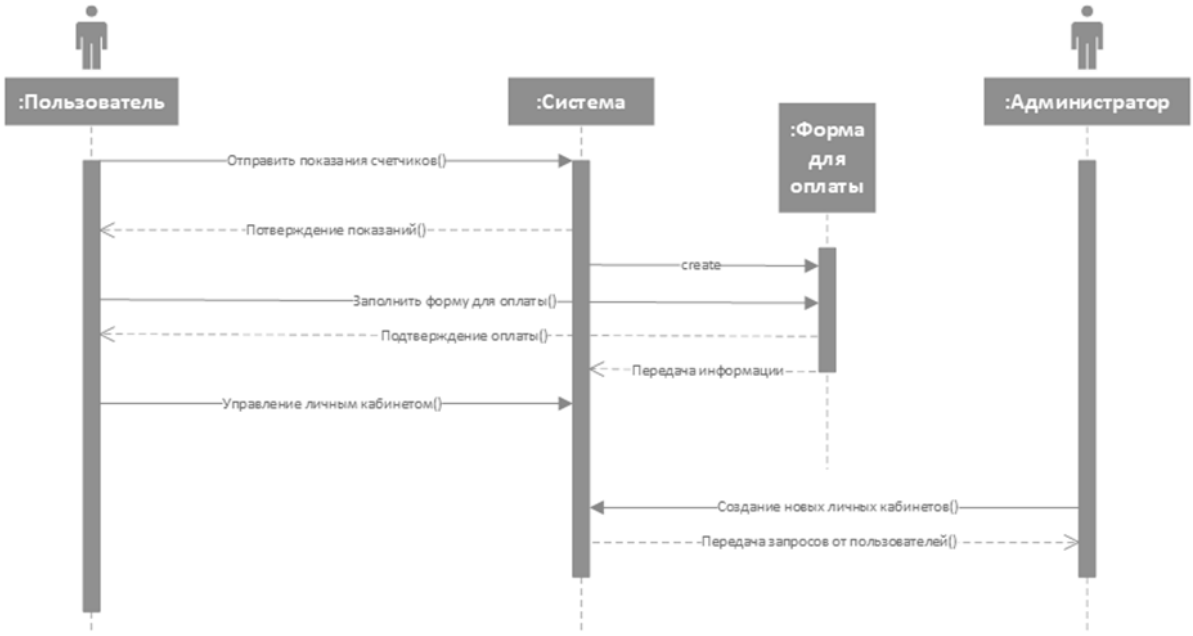

Рисунок 9 - Диаграмма последовательности

Диаграмма взаимодействий

Диаграмма взаимодействий определяет какие способы взаимодействия с системой есть в распоряжении пользователей. Она позволяет видеть все взаимодействия запросов в системе и дополняет диаграмму последовательностей

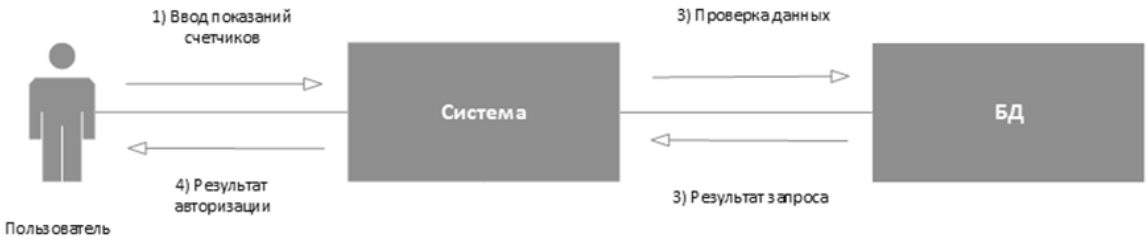

Рисунок 10 - Диаграмма взаимодействий. Авторизация.

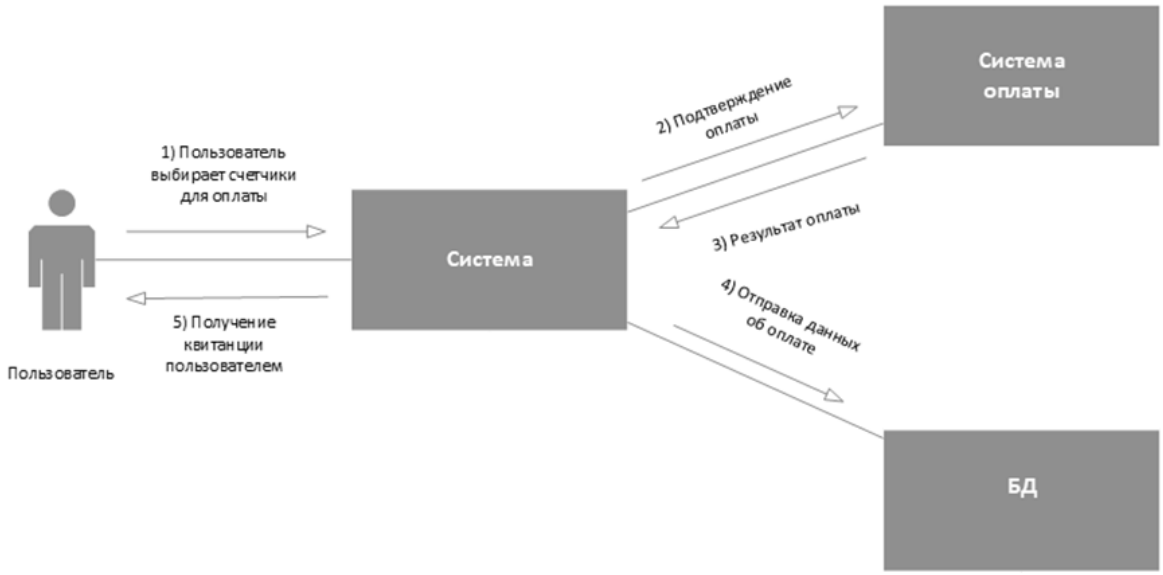

Рисунок 11 - Диаграмма взаимодействий. Оплата задолженностей.

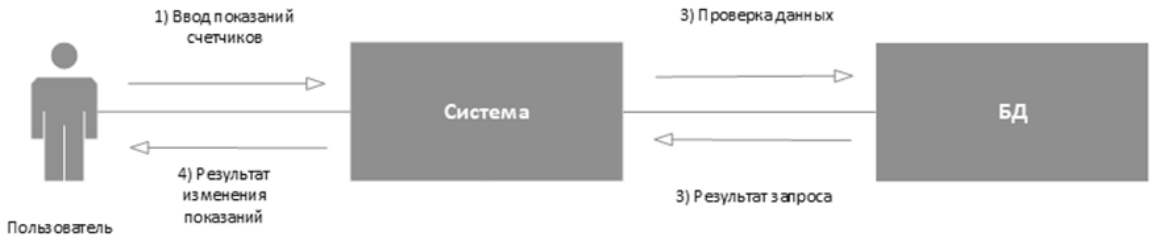

Рисунок 12 - Диаграмма взаимодействий. Передача показаний.

Диаграмма состояний

Диаграмма состояний определяет, как объект переходит из одного состояния в другое. Рассмотри диаграмму состояний показаний счетчиков.

После состояния «Введенные показатели», форма с показаниями счетчиков может, в случае корректности введенных данных, перейти в состояние «Измененные показатели». В случае некорректных данных показания не изменяются.

Аналогично и с формой оплаты, если после состояния «Оплата задолженности» средств на счету хватает, то форма переходит в состояние «Погашение задолженности», иначе же оплата не проходит и задолженность остается прежней.

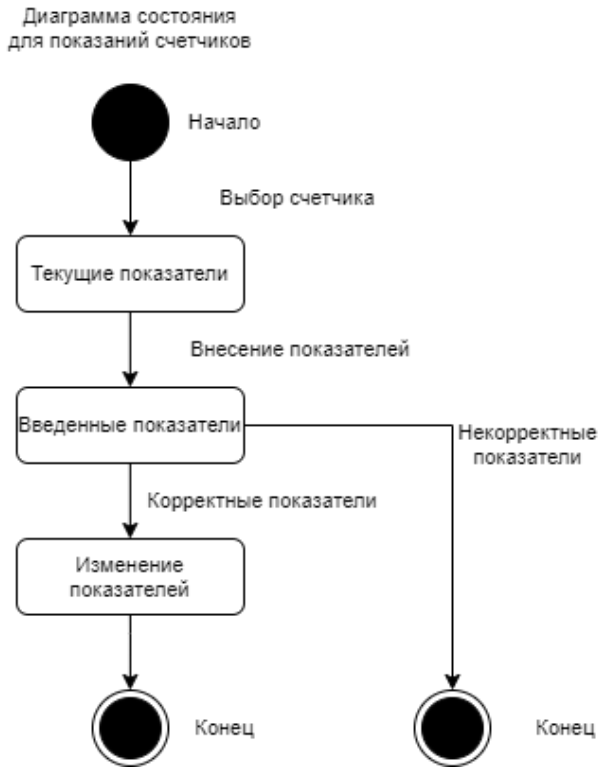

Рисунок 13 - Диаграмма состояний. Передача показаний.

Рисунок 14 - Диаграмма состояний. Оплата задолженностей.

ER-диаграмма

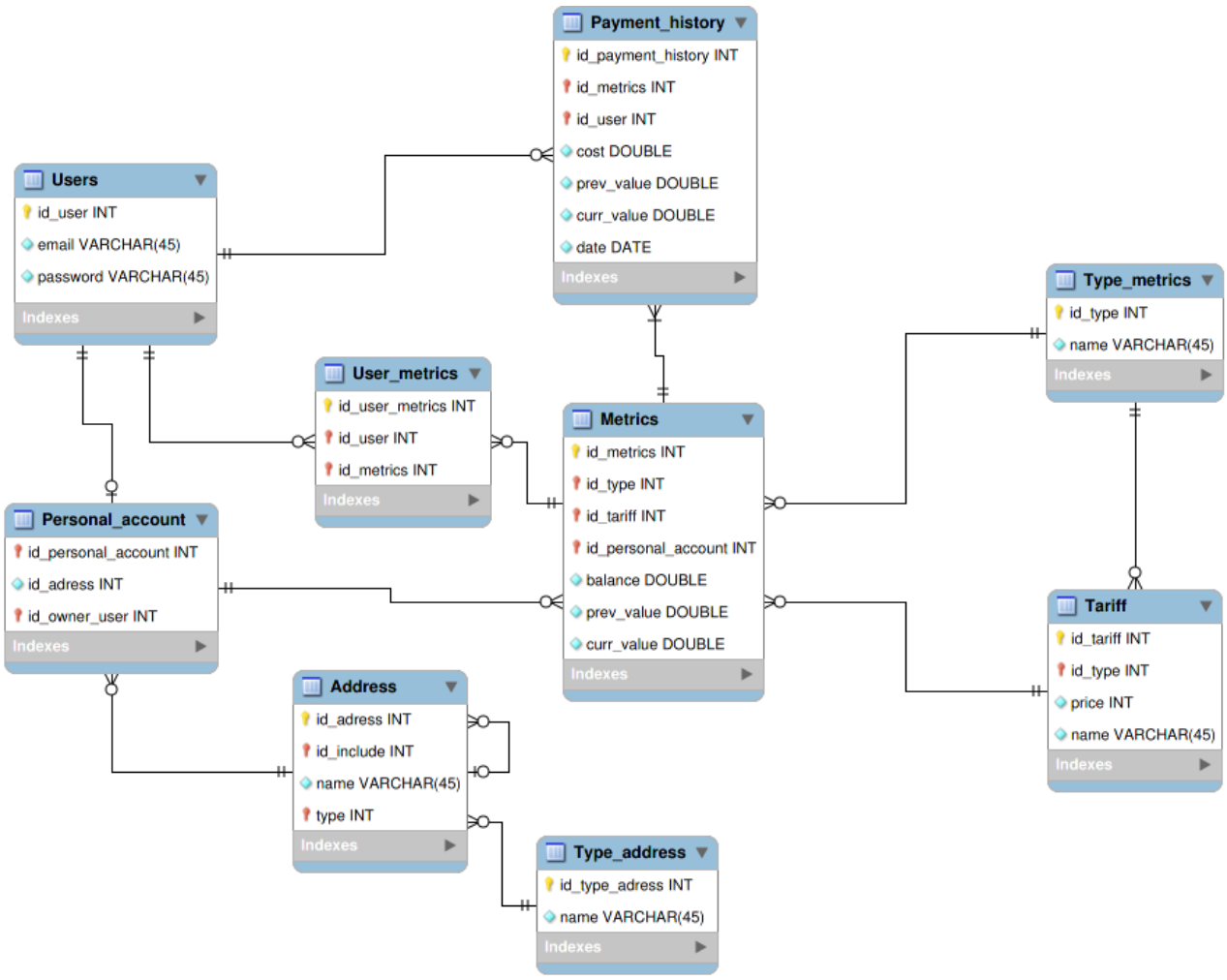

Рисунок 15 - ER-диаграмма

3. Реализация приложения

3.1.Анализ средств реализации

Для разработки клиентского приложения для Android выбран следующий стэк технологий:

— Android SDK.

— Kotlin.

Для серверной части:

— Python 3.9.

— Flask.

— SQLAlchemy.

Язык программирования Kotlin отлично подходит для разработки мобильных приложений для устройств под системой Android, поскольку имеет относительно большое сообщество, хорошо написанную документацию и поддержку от Google.

Язык программирования Python поддерживает огромное количество библиотек, позволяющих использовать наиболее подходящие инструменты для написания серверной части приложения. Также для этого языка характерна гибкость разработки, легкость в написании и поддерживании кода и дальнейшие улучшения проекта.

Фреймворк Flask, относящийся к категории микрофреймворков, является удобным инструментом для разработки, так как он поддерживает расширения для добавления новой функциональности через сторонние библиотеки и защищает от межсайтового скриптинга.

Для связи клиента и сервера используется библиотека Retrofit2, упрощающая работу с REST API сайта и являющаяся практически стандартом для сетевого взаимодействия

SQLAlchemy – набор инструментов Python SQL, который предоставляет разработчикам множество возможностей для работы с базами данных при помощи языка SQL.

В качестве системы управления базы данными используется MariaDB, ответвление MySQL. От него Maria отличается повышенной производительностью, меньшим количеством ошибок и большей функциональностью. Например, оптимизатор запросов, более быстрые индексы для механизма хранения данных и так далее.

Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в различных средах. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер.

Heroku — это платформа, позволяющая создавать, запускать и управлять приложениями полностью в облаке.

3.2.Разработка Frontend

Получение и передача данных с серверной частью приложения осуществления при помощи библиотеки Retrofit2.

Приложение для Android делится на несколько основных частей по архитектуре Model-View-ViewModel (MVVM), для разделения модели и ее представления.

Представление основных экранов состоит из MainActivity и фрагментов:

— AccountFragment: аккаунт пользователя.

— AddMeterFragment: добавление счетчика.

— HistoryFragment: история оплат.

— MeterFragment: счетчик.

— MyAccountsFragment: счета пользователя.

— PaymentFragment: квитанция.

— SavedMetersFragment: сохраненные счетчики пользователя.

— SettingsFragment: настройки.

Через контроллеры DrawerController вызывается отрисовка основных экранов. Для взаимодействия между моделями и представлениями используются ViewModel:

— AccountViewModel.

— GeneralButtonViewModel.

— HistoryMeterItemViewModel.

— LoginViewModel.

— MeterItemViewModel.

— MeterViewModel.

Взаимодействие с сервером осуществляется через модели и интерфейсы для взаимодействия. ApiWorker — базовый класс всех Worker'ов. Его задача — выполнять асинхронный запрос в сеть на определённом CoroutineDispatcher (на определённом потоке), и отдавать результат в виде ApiResult<ResponseBody>.

GeneralWorker — его наследник. В нём происходят все запросы в сеть. Он вызывает метод родителя для обработки запроса и передаёт ему Dispatchers.IO (рекомендуемый для сетевых и БД взаимодействий) и метод для выполнения, взятый из интерфейсов.

ApiResult имеет трех наследников. Success, содержащий тело ответа сервера, GenericError, содержащий html код ошибки, NetworkError, отвечающий за ошибки подключения к интернету.

Модели:

— ContinueRegisterUser.

— CurrentMetric.

— Flat.

— InformationAboutPayment.

— ItemPaymentHistory.

— LoginUser.

— Metric.

— MetricWithSavedField.

— Payment.

— QuickLoginUser.

— RegisterUser.

— SuccessfulLoginUser.

3.2.1. Навигация по приложению

Открывая приложения, пользователь может выбрать желаемый тип входа, без авторизации (Быстрая оплата) или с авторизаций (Вход).

Рисунок 16 - Экран входа

Далее пользователя встречает экран с сохраненными счетчиками, где он может выбрать счетчики для оплаты или долгим нажатием на счетчик перейти на экран с его данными. Сохраненные счетчики могут принадлежать различным лицевым счетам, например, если пользователь не является владельцев квартиры, но должен регулярно оплачивать счетчики После нажатия кнопки «Оплатить выбранные» происходит переход на экран с подтверждением оплаты. После успешной оплаты происходит переход на экран, уведомляющий об этом.

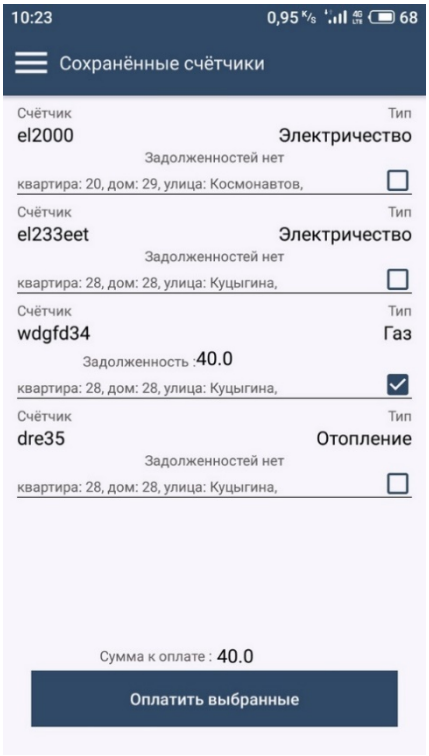

Рисунок 17 – Экран сохраненных счетчиков

Рисунок 18 – Экран подтверждения оплаты

При долгом нажатии на поле с названием счетчика можно перейти в меню, где есть возможность изменить показания счетчика и просмотреть историю оплат по этому счетчику.

Рисунок 19 – Экран истории оплат

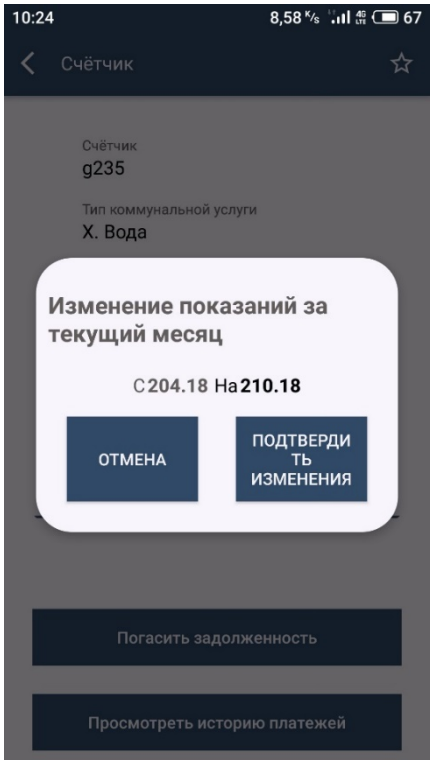

Рисунок 20 – Экран подтверждения изменений показания

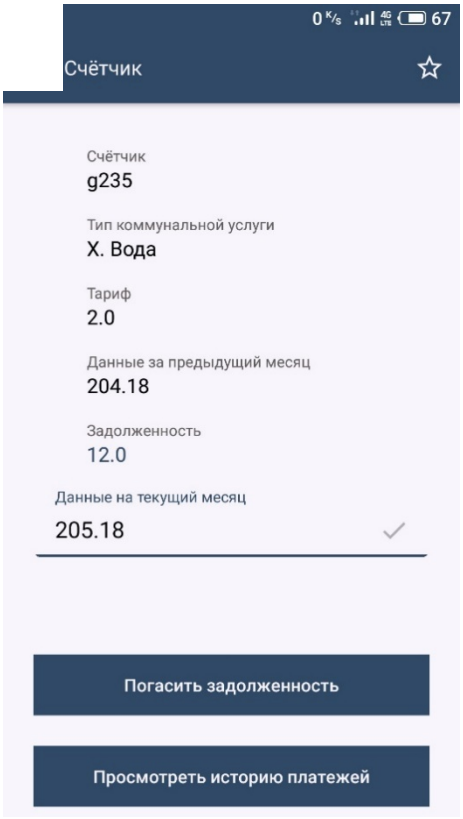

Рисунок 21 – Экран счетчика

3.3. Разработка Backend

Серверная часть приложения написана на языке Python с использованием фреймворка Flask. Сложная архитектура в виде прослоек логики, данных и репозиториев не подходит для функционального кода на Python и его динамической компиляции.

Архитектура серверной части включает следующие компоненты:

— Контроллер.

— Модули.

— Сущности (Entity).

Контроллер используется для инициализации и привязки всех модулей и запуска всего приложения.

Модули используются для настройки принятия, обработки запросов и передач ответа. Каждый модуль имеет свой уникальный URL. Все конечные точки, за исключением публичных, имеют защиту собственной реализации с использованием технологии JWT. Для взаимодействия с сущностями базы данных используется ORM SQLAlchemy.

Entity являются сущностями базы данных, которые были автоматически ассоциированы после подключения к базе данных.

Для загрузки проекта на Heroku использовался docker контейнер. Данный способ обладает таким преимуществом, как кроссплатформенность для размещения проекта, что позволяет разрабатывать проект без опоры на ОС разработчика или сервера. Данный способ собирает образ приложения с нужными зависимости и выбранном окружении, после чего его можно будет запускать на другом устройстве независимо от его особенностей.

3.4. Тестирование

В приложении есть инструментальные и модульные тесты. В листингах представлены примеры тестов на валидацию email пользователя и конвертации строки, пришедшей из сервера, в объект класса Date. В листинге 1 показано

тестирование email пользователя. Условию удовлетворяет только строка "name@email.com".

Листинг 1 – Тест на валидацию emal.

```
@RunWith(MockitoJUnitRunner::class)
class EmailValidatorTest {
    val incorrect_email = "ie"
    val empty_field = "ef"

    @Mock
    private lateinit var mockContext: Context

    @Test
    fun emailValidator_CorrectEmailSimple_ReturnsTrue() {
        // Given a mocked Context injected into the object under test...
        Mockito.`when`(mockContext.getString(R.string.incorrect_email))
            .thenReturn(incorrect_email)
        Mockito.`when`(mockContext.getString(R.string.empty_field))
            .thenReturn(empty_field)

        Assert.assertEquals(
            EmailValidator.validate("name@email.com", mockContext), ""
        )
        Assert.assertEquals(
            EmailValidator.validate("name@email", mockContext), incorrect_email
        )
        Assert.assertEquals(
            EmailValidator.validate("", mockContext), empty_field
        )
    }
}
```

Заключение

В результате выполнения курсовой работы было разработано мобильное приложение для передачи показаний счетчиков и выставления счетов за потребленные услуги, серверное приложение, на котором происходят все основные операции, а также веб-приложение для администраторов, которые могут управлять аккаунтами пользователей.

У авторизованных пользователей реализованы функции привязки лицевых счетов, добавление отдельных счетчиков в сохраненные, просмотр истории оплат, передача показаний и сама оплата показаний.

У неавторизованных пользователей есть только возможность передавать показания счетчиков и оплачивать их.

Администраторы могут добавлять, изменять и удалять лицевые счета и их счетчики.

Сервис имеет следующие перспективы развития:

- использование нейросети для обработки фотографий счетчика и внесения показаний;
- интеграция оплаты с основными платежными системами;
- добавление английского языка в приложение;
- улучшения серверной части приложения для обработки большого количества пользователей;
- добавление дополнительной функциональности в мобильное приложение – автоматическая оплата и передача показаний, система уведомлений для пользователя.

Список используемой литературы

1. Grinberg M. Flask Web Development. Developing web applications with Python — O'Reilly Media, 2014—258 p. — ISBN 9781449372613, ISBN 9781449372620.
2. Copperwaite M., Leifer C. Learning Flask Framework. Build dynamic, data driven websites and modern web applications with Flask. — Packt Publishing, 2015. — 250 p. — ISBN 9781783983360.
3. Bartholomew, Daniel. Getting Started with MariaDB — 2013. — ISBN 9781782168096.
4. Bartholomew, Daniel. MariaDB Cookbook — 2014. — ISBN 978-1-78328-440-5.
5. Forta, Ben MariaDB Crash Course— Addison Wesley, 2011. — ISBN 0-321-79994-1.