МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГУ»)

Факультет компьютерных наук

Кафедра информационных технологий управления

Сбор показаний индивидуальных приборов учёта в многоквартирных домах и выставление счетов за потреблённые услуги

Курсовой проект

09.03.02 Информационные системы и технологии

Информационные системы и технологии в управлении предприятием

Преподаватель_____В.С. Тарасов, ст. преподаватель

Обучающийся_____А.А. Сарайкин, 3 курс, д/о

Обучающийся_____Я.В. Солодовникова, 3 курс, д/о

Воронеж 2023

СОДЕРЖАНИЕ

Введение

В обязанности практически всех людей входит оплата коммунальных услуг. Для этого необходимо каждый месяц передавать показания счетчиков в компании, отвечающие за предоставление таких услуг, и производить оплату в соответствии с квитанциями.

Раньше был доступен только один способ оплаты: через отделение банка или почты. Данный способ можно считать достаточно затратным по времени и усилиям, так как необходимо добраться до отделения банка, после чего дождаться своей очереди. Также стоит учитывать, что график работы отделений может быть неудобным и ограничивать возможности клиента. Например, большинство банков и почтовых отделений не работают по воскресеньям.

С распространением смартфонов появился еще один способ оплаты: через мобильное приложение банка. Клиенту не надо тратить время на очереди, ходить куда-либо и подстраиваться под график работы. Несмотря на эти преимущества, есть ряд недостатков. Например, нельзя в одном месте просмотреть историю передаваемых показаний, для внесения оплаты по нескольким квитанциям необходимо потратить много усилий, при заполнении реквизитов можно допустить ошибку и оплатить другой счет.

Возможен еще один способ оплаты: через специализированное приложение. Его преимущество, что в одном месте можно передать показания всех счетчиков и в одном месте произвести оплату для всех видов коммунальных платежей. Также можно просматривать историю и статистику, собранную по всем видам потребляемых услуг.

Данный проект направлен на создание приложения, которое позволит сократить время и количество действий на передачу показаний ИПУ и их оплату.

1 Постановка задачи

1.1 Постановка задачи

Целью данного курсового проекта является создание приложения для сбора показаний индивидуальных приборов учёта в многоквартирных домах и выставление счетов за потреблённые услуги.

Помимо этого, будут также доступны функции:

— Расчёт статистики по потреблённым услугам;

— Составление прогноза на потребление коммунальных услуг в следующем периоде.

1.2 Обзор аналогов

1.2.1 Квартплата+

Мобильное приложение, направленное на оплату ЖКУ, домофона, связи и других услуг, а также передачу показаний ИПУ.

Данное приложение не имеет онбординг экрана, сразу после входа пользователь видит экран авторизации. С экрана авторизации можно попасть на экран «О приложении» с помощью кнопки в правом верхнем углу.

После авторизации осуществляется переход на экран «Мои услуги». Сразу после этого осуществляется поиск платежей по адресу пользователя. По нажатию кнопки «Оплатить», можно произвести оплату выбранных счетов. Приложение не сообщает никакой информации, которая должна помочь в навигации. Интуитивно не понятно, куда нужно нажать пользователю, чтобы передать показания счетчиков. Это можно узнать только в чате поддержки, если задать соответствующий вопрос.

На главном экране, в строке адреса находится кнопка «Счетчики». Если нажать на нее, осуществляется переход на экран со списком счетчиков. При нажатии на конкретный счетчик открывается экран, на котором можно посмотреть информацию о счетчике и передать показания.

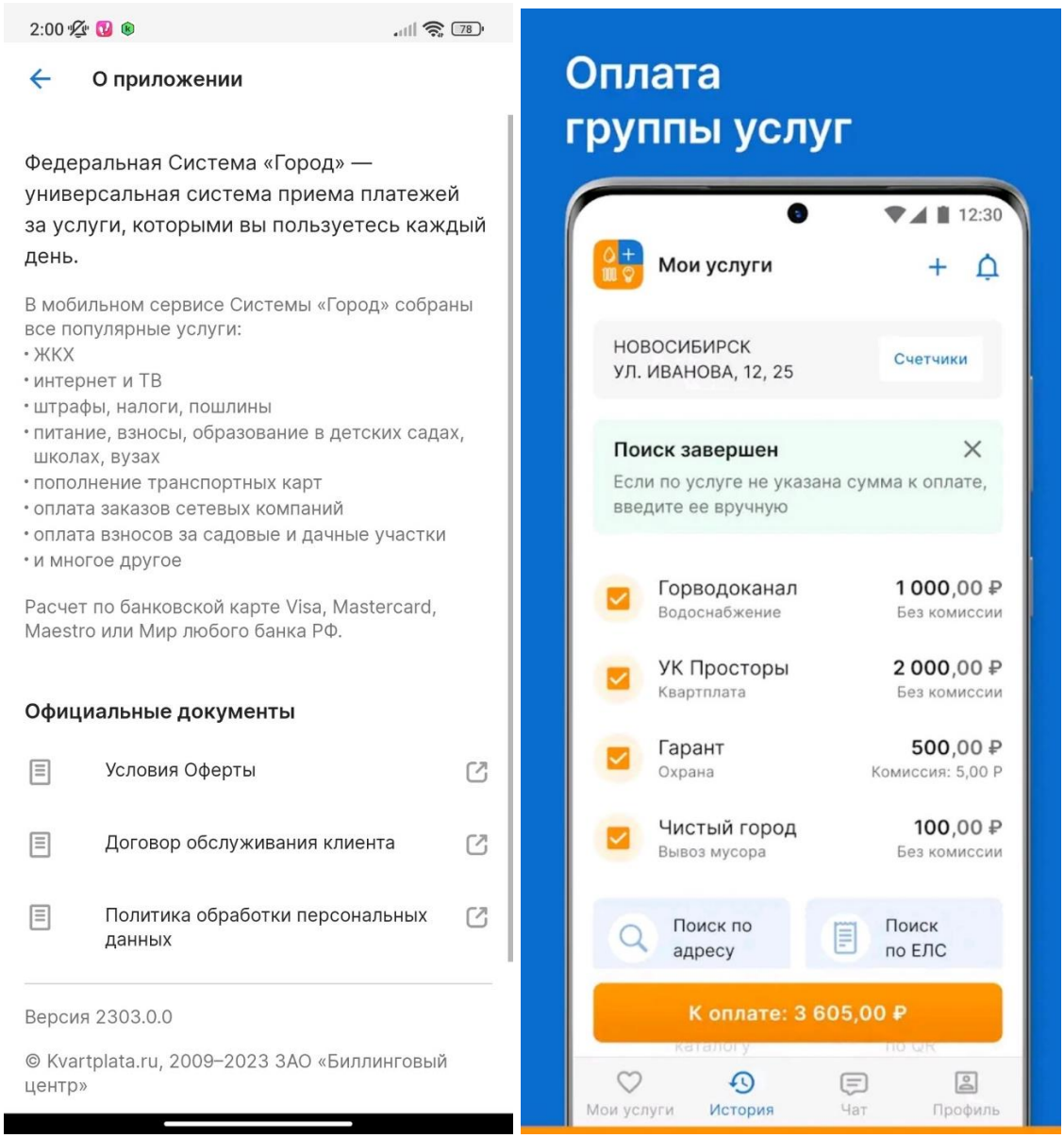

Рисунок 1 - Информационный и главный экраны

Для передачи показаний необходимо пройти через 3 экрана: Главный → Счетчики → Счетчик. От экрана «Счетчики» можно перейти только на экран одного конкретного счетчика. Соответственно, если пользователь передал показания счетчика, чтобы передать показания следующего, ему нужно вернуться на экран назад и затем снова перейти на экран необходимого счетчика.

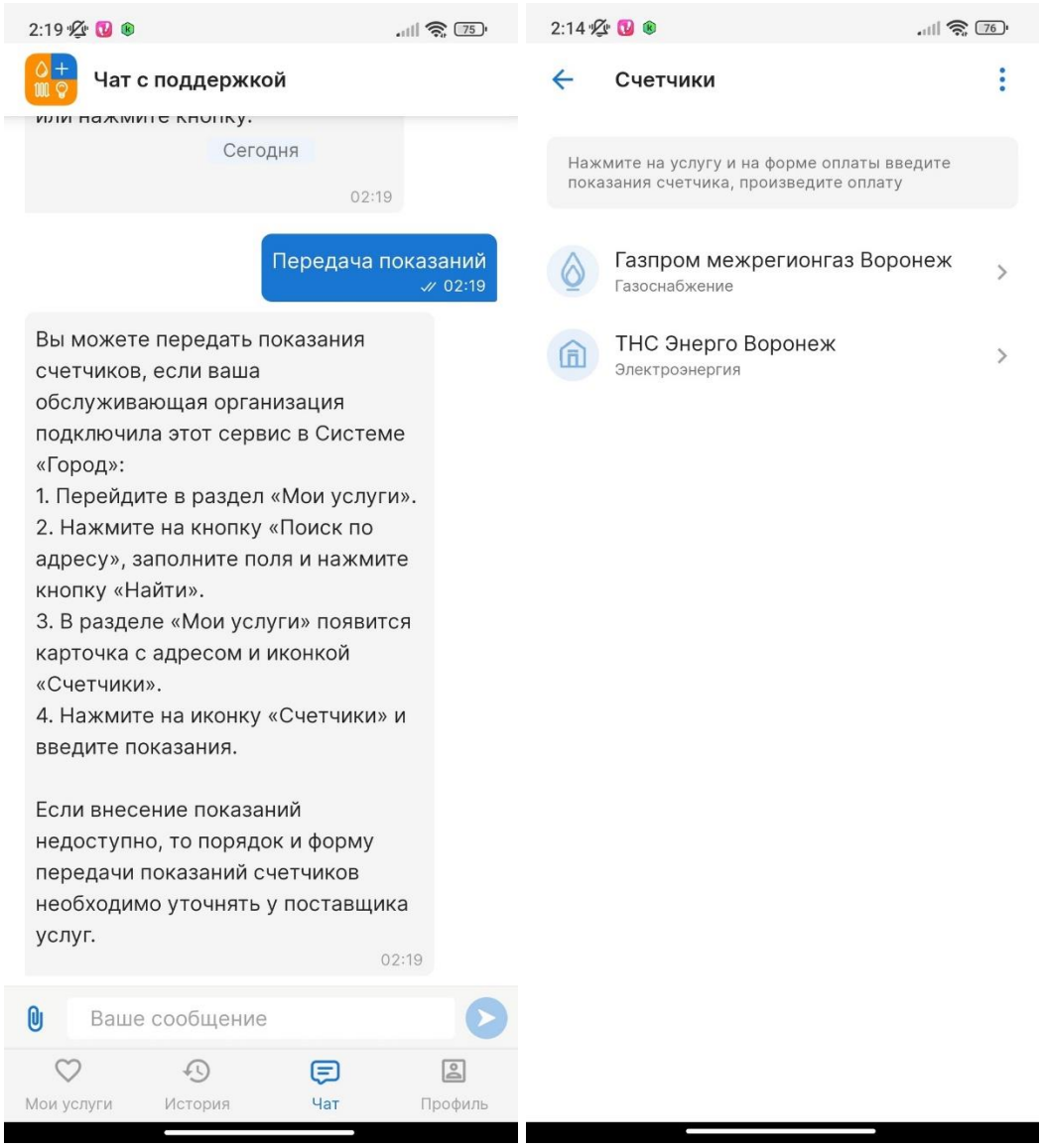

Рисунок 2 - Чат с поддержкой и экран со списком счетчиков

Преимущества приложения:

— Автоматический поиск счетов на оплату по адресу пользователя;

— Чат с поддержкой.

Недостатки приложения:

— Отсутствие информации о навигации в приложении даже при первом открытии;

— Для передачи показаний ИПУ необходимо совершить много действий.

В отличие от реализуемого проекта, Квартплата+ не обладает функциями просмотра статистики потребления коммунальных услуг или просмотра прогноза потребления на следующий период.

1.2.2 РВК.Услуги

Мобильное приложение для клиентов ГК «Росводоканал». Оно позволяет оплачивать ЖКУ и передавать показания ИПУ.

Данное приложение имеет онбординг экран, после которого пользователь переходит на экран авторизации. Предлагается выбор: войти как клиент или как подрядчик.

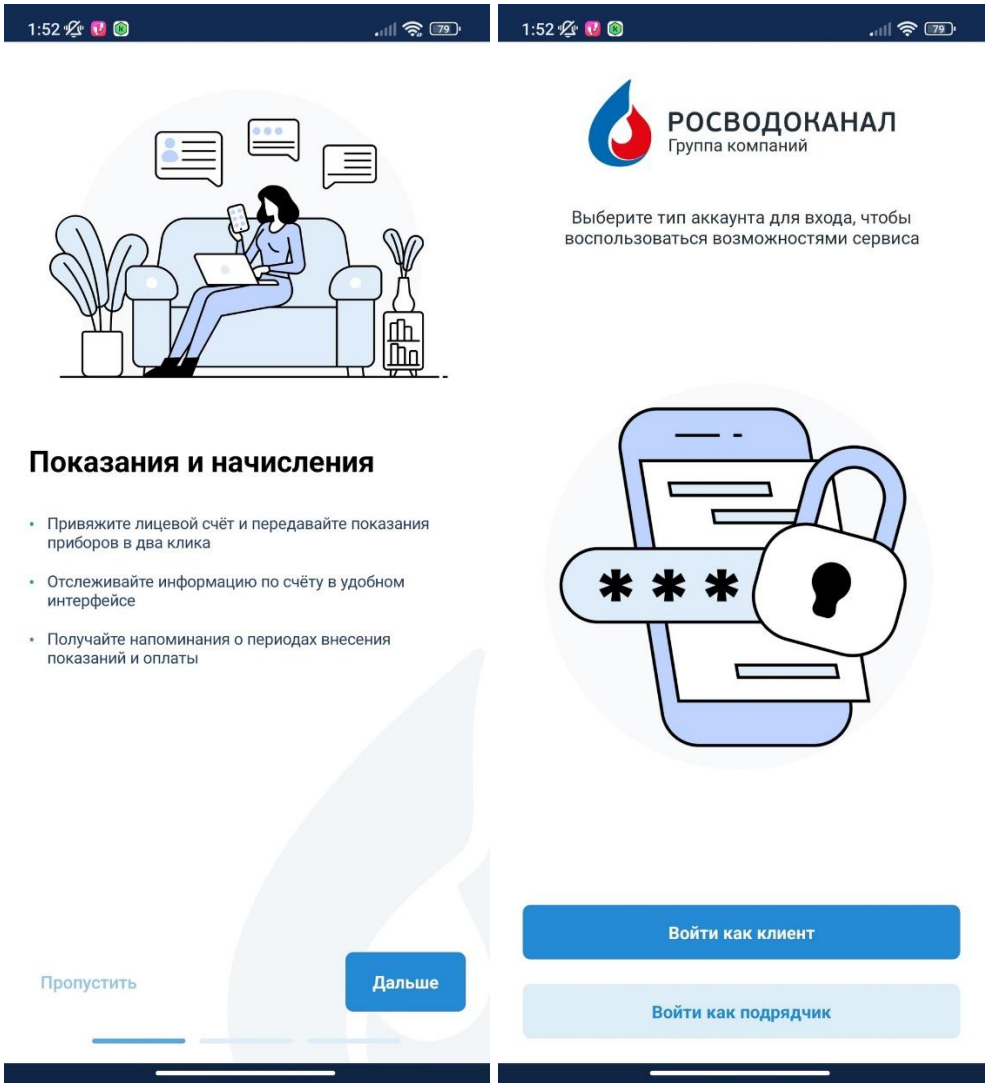

Рисунок 3 - Онбординг экран и экран авторизации

После авторизации осуществляется переход на экран «Счета». На нем отображаются все лицевые счета, которые привязаны к пользователю. При

нажатии на кнопку «Управлять лицевым счетом» осуществляется переход на экран «Лицевой счет».

Даже если у пользователя всего 1 лицевой счет, ему все равно необходимо на экране «Счета» выбирать конкретный лицевой счет каждый раз при запуске приложения. Таким образом, чтобы передать показания необходимо пройти 3 экрана: «Счетчики» → «Лицевой счет» → «Передать показания».

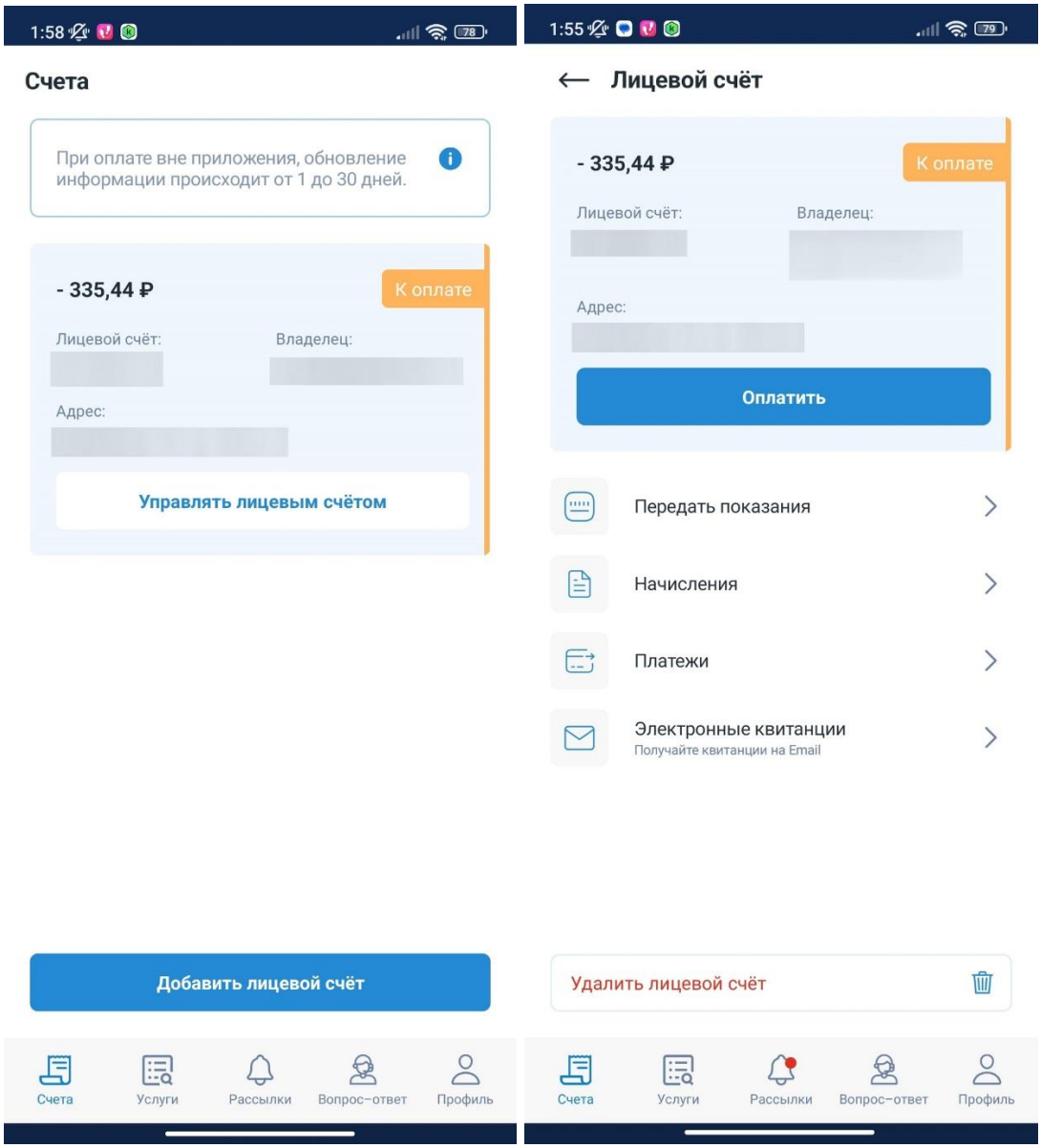

Рисунок 4 - Экраны «Счета» и «Лицевой счет»

С экрана «Лицевой счет» можно выполнить все доступные по нему операции: передача показаний, просмотр истории начислений, просмотр истории платежей, управление электронными квитанциями.

На экране «Услуги» можно посмотреть список доступных услуг и оформить необходимые. На экране «Рассылки» можно посмотреть новости, рассылки и уведомления.

Преимущества приложения:

— Возможность оформления заявок на определенные услуги;

— Возможность получения электронных квитанций на Email;

— Чат с поддержкой.

Недостатки приложения:

— Для передачи показаний ИПУ необходимо совершить много действий.

В отличие от реализуемого проекта, РВК.Услуги не обладает функциями просмотра статистики потребления коммунальных услуг или просмотра прогноза потребления на следующий период.

Подводя итог, можно выделить следующие недостатки, присущие многим мобильным приложениям для оплаты ЖКУ и передачи показаний ИПУ: для передачи показаний нужно сделать много действий. В реализуемом приложении будет необходимо совершить меньшее количество шагов для передачи показаний.

2 Анализ предметной области

2.1 Глоссарий

База данных (БД) — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе [1].

Индивидуальные приборы учета (ИПУ) — это приборы, которые учитывают личное потребление коммунальных ресурсов клиентом.

Жилищно-коммунальные услуги (ЖКУ) — услуги по поддержанию и восстановлению надлежащего технического и санитарно-гигиенического состояния зданий, сооружений, оборудования, коммуникаций и объектов коммунального назначения [2].

Личный кабинет — это персональная страница пользователя, доступная после авторизации [3].

Лицевой счёт — бухгалтерский счёт для ведения учёта расчетов с физическими и юридическими лицами, на котором отражаются все финансово-кредитные операции с определенным клиентом [4].

Тариф – установленная стоимость за единицу ресурса [5].

2.2 Сценарии пользователей

1. Пользователь: Лидия Короткова, 58 лет.

   Описание: владелица двухкомнатной квартиры в жилом комплексе, обладает общими навыками использования телефона.

   Пользовательская история: Лидии необходимо передать показания ИПУ и произвести оплату счетов, за предоставляемые ЖКУ, с целью своевременного исполнения обязанностей потребителя ЖКУ в соответствии с договором.

2. Пользователь: Иван Моисеев, 22 года.

Описание: владелец однокомнатной квартиры в жилом комплексе, обладает продвинутыми навыками в использовании мобильного телефона.

Пользовательская история: Ивану необходимо узнать прогноз потребления ЖКУ на следующий период, чтобы заранее планировать свой бюджет и траты на месяц.

3. Пользователь: Мария Трунова, 30 лет.

   Описание: владелица двухкомнатной квартиры, обладает продвинутыми навыками использования мобильного телефона.

   Пользовательская история: Марии необходимо узнать свою статистику потребления ЖКУ, чтобы оценить объемы потребления и внести при необходимости коррективы.

2.3 Продуктовые воронки

Оценим количество шагов, которое необходимо авторизованному пользователю для осуществления передачи показаний ИПУ и платы ЖКУ.

При открытии приложения уже авторизованному пользователю будет показан сплэш экран, сразу после которого осуществится переход на экран «Передача показаний», на котором можно передать показания всех ИПУ пользователя. Для оплаты ЖКУ необходимо в нижнем меню выбрать «Оплата показаний».

Таким образом, необходимо всего 3 шага для выполнения основных функций приложения.

2.4 Диаграммы, описывающие работу системы

Диаграммы, описывающие работу системы, играют важную роль в процессе разработки. Они помогают разработчикам и аналитикам лучше понять требования к системе, ее структуру, функциональность и

взаимодействие между компонентами. Диаграммы полезны в процессе разработки приложений по следующим причинам:

— Визуализация системы: Диаграммы предоставляют наглядное представление о системе и ее компонентах;

— Уточнение требований: Диаграммы позволяют детализировать требования к системе и ее функциональности;

— Разработка архитектуры: Диаграммы позволяют разработчикам спроектировать архитектуру приложения, разделить его на модули, определить функциональные блоки и определить, как они будут взаимодействовать друг с другом;

— Коммуникация и согласование: Диаграммы служат средством коммуникации между участниками проекта;

— Документация: Диаграммы являются частью документации проекта.

В целом, диаграммы способствуют более эффективной разработке, улучшению коммуникации и обеспечивают надежную основу для проектирования и создания высококачественного приложения.

2.4.1 Диаграмма прецедентов (Use-case diagram)

Диаграмма прецедентов (use-case diagram) является одним из видов диаграмм UML (Unified Modeling Language), используемых для моделирования функциональных требований системы, показывая взаимодействие между актерами (users) и прецедентами (use cases).

Основная цель диаграммы прецедентов состоит в идентификации функциональных требований системы и визуализации взаимодействия между актерами и прецедентами. Она помогает понять, как система будет использоваться пользователями и какие функциональности они ожидают от системы.

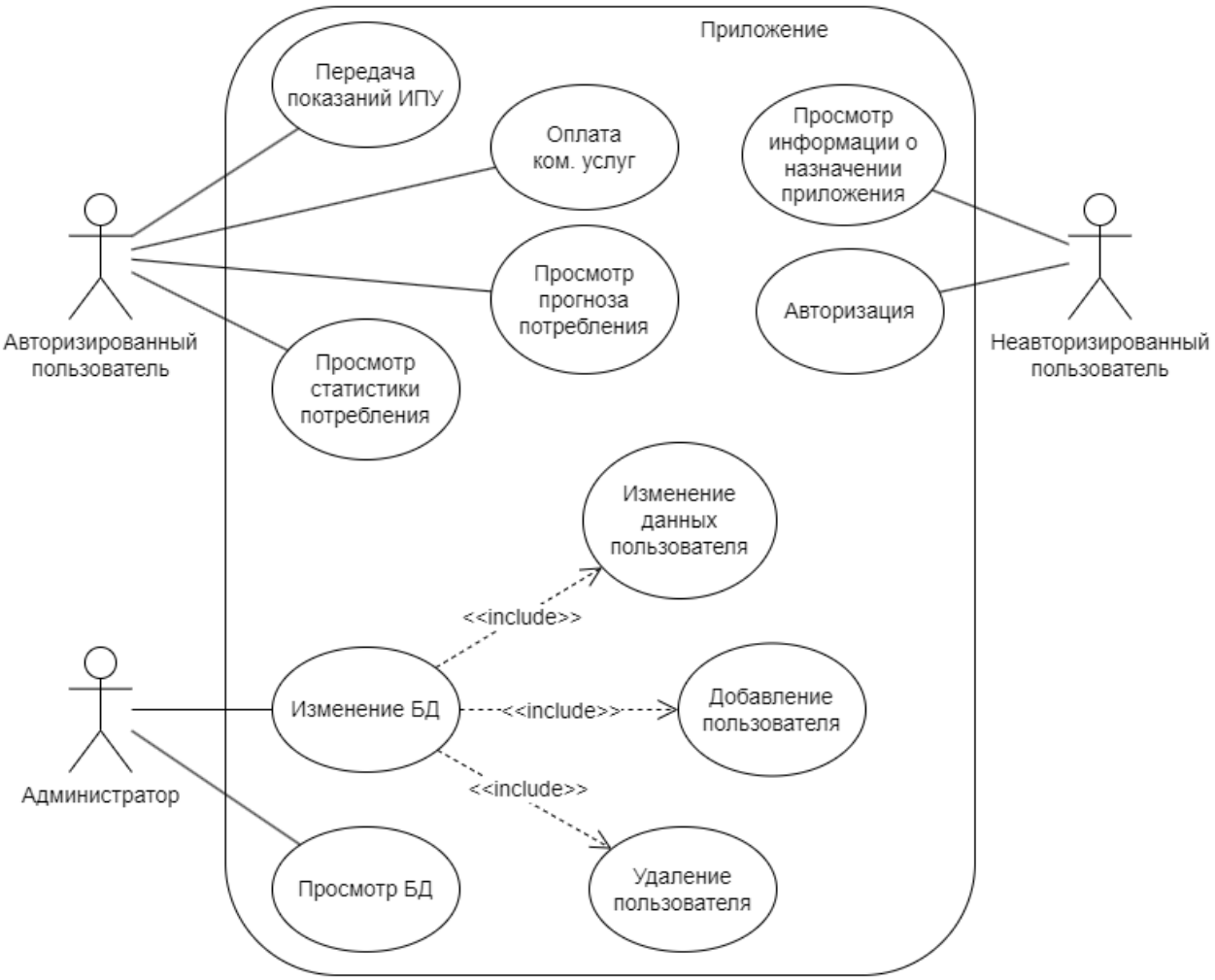

Рисунок 5 - Use-case diagram

Неавторизированный пользователь — пользователь, не прошедший авторизацию или не зарегистрированный в системе. Данный пользователь может:

— Просматривать информацию о назначении приложения;

— Авторизоваться.

Авторизированный пользователь — пользователь, прошедший авторизацию в системе. Помимо функций, доступных неавторизированному пользователю, данный пользователь может:

— Передавать показания ИПУ;

— Получить счет на оплату коммунальных услуг;

— Просматривать статистику потребления за определенный период времени;

— Просматривать прогноз потребления услуг на следующий период.

Администратор — сотрудник компании, который может:

— Производить регистрацию клиентов в системе;

— Просматривать БД;

— Вносить изменения в БД (например, удалять или изменять данные клиента).

2.4.2 Диаграмма классов (Class diagram)

Диаграмма классов (class diagram) является одним из основных видов диаграмм UML, используемых для моделирования статической структуры системы, показывая классы, их атрибуты, методы и взаимосвязи между ними.

Основная цель диаграммы классов состоит в визуализации статической структуры системы и описании классов, их атрибутов и методов. Она помогает понять, как объекты системы организованы, какие свойства у них есть и как они взаимодействуют друг с другом.

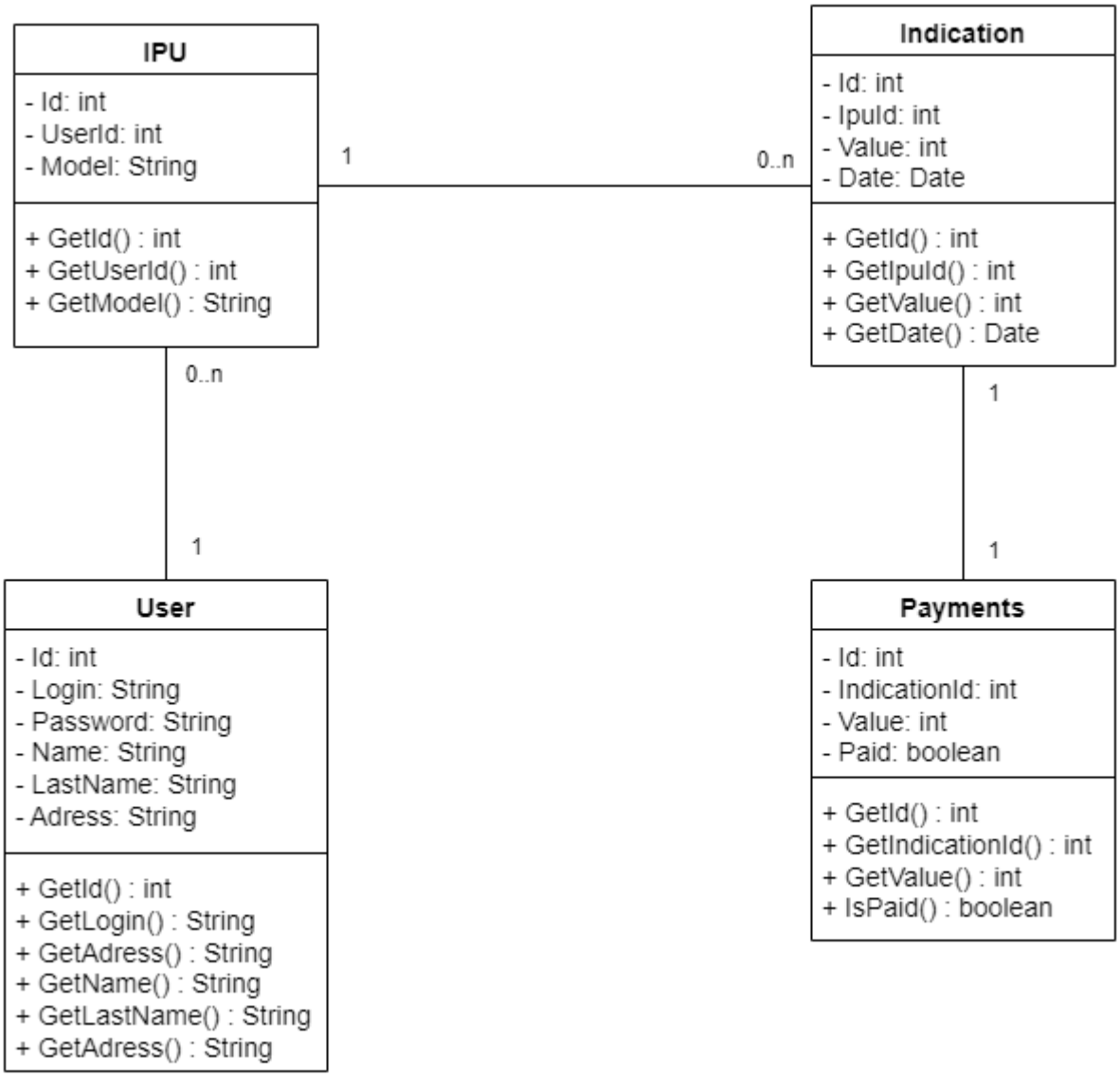

Рисунок 6 - Class diagram

2.4.3 Диаграмма активностей (Activity diagram)

Диаграмма активностей является одним из видов диаграмм UML, которые используются для визуализации и описания поведения системы или процесса. Она позволяет моделировать последовательность и взаимодействие различных действий, состояний и решений, которые выполняются в рамках системы.

Основная цель диаграммы активностей состоит в том, чтобы показать, какие активности выполняются в системе, какие действия происходят одновременно, какие условия приводят к разным ветвлениям или

объединениям активностей, и как данные или управление передаются между активностями.

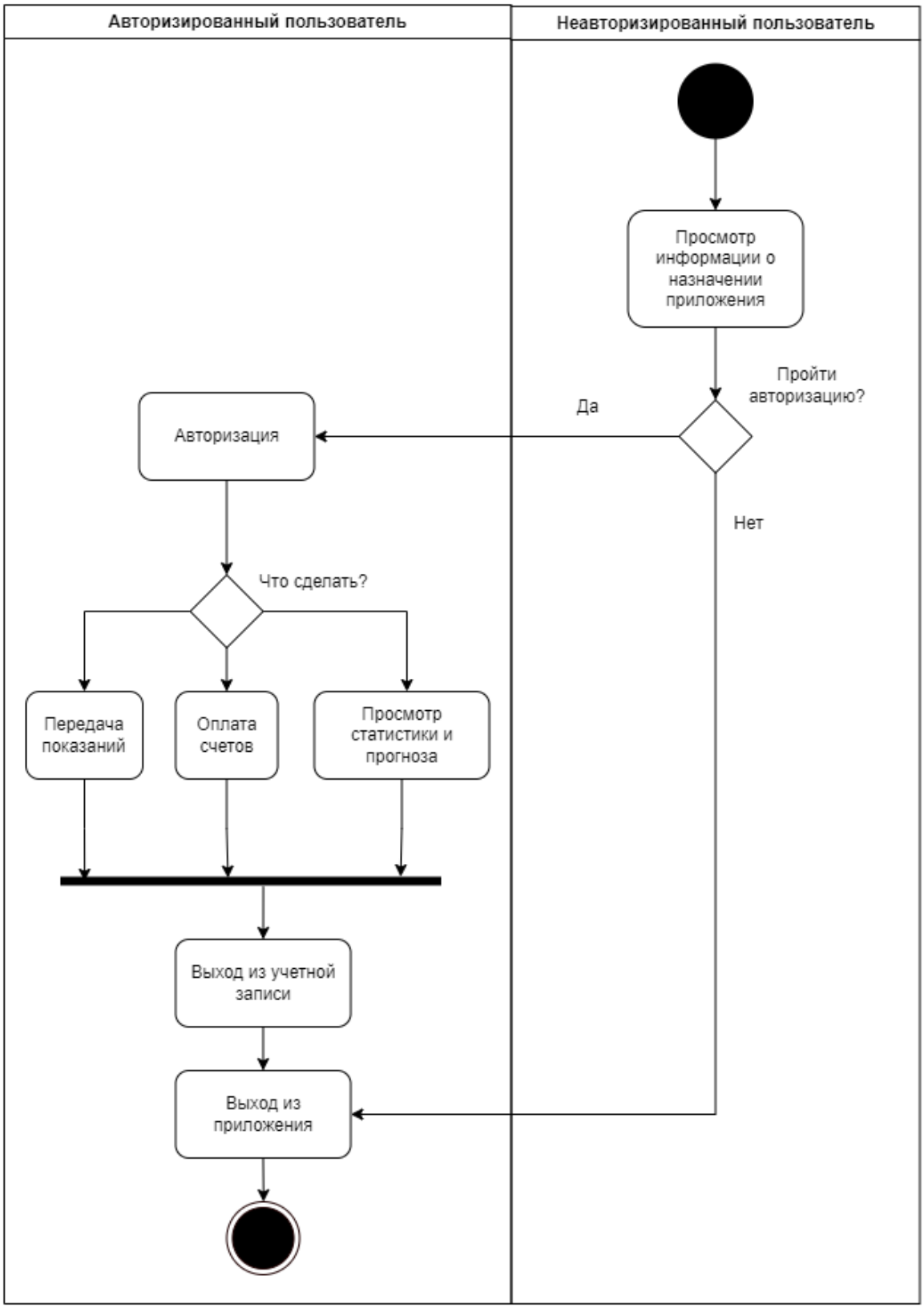

Рисунок 7 - Activity diagram

На рисунке 7 приведена диаграмма активностей для разрабатываемой системы. При входе в систему неавторизованный пользователь видит онбординг экраны с информацией о приложении и его возможностях. После просмотра всех экранов осуществляется переход на экран авторизации. Пользователь может пройти авторизацию или выйти из приложения. После авторизации пользователь может: передать показания, оплатить счета, посмотреть статистику или прогноз.

2.4.4 Диаграмма последовательности (Sequence diagram)

Диаграмма последовательности (sequence diagram) является одним из видов диаграмм UML, используемых для визуализации взаимодействия между объектами или компонентами системы в определенной последовательности.

Основная цель диаграммы последовательности заключается в визуализации динамического поведения системы или процесса. Она помогает понять, как объекты взаимодействуют друг с другом, какие сообщения они обмениваются и в каком порядке, а также какие операции вызываются и как они влияют на состояние системы.

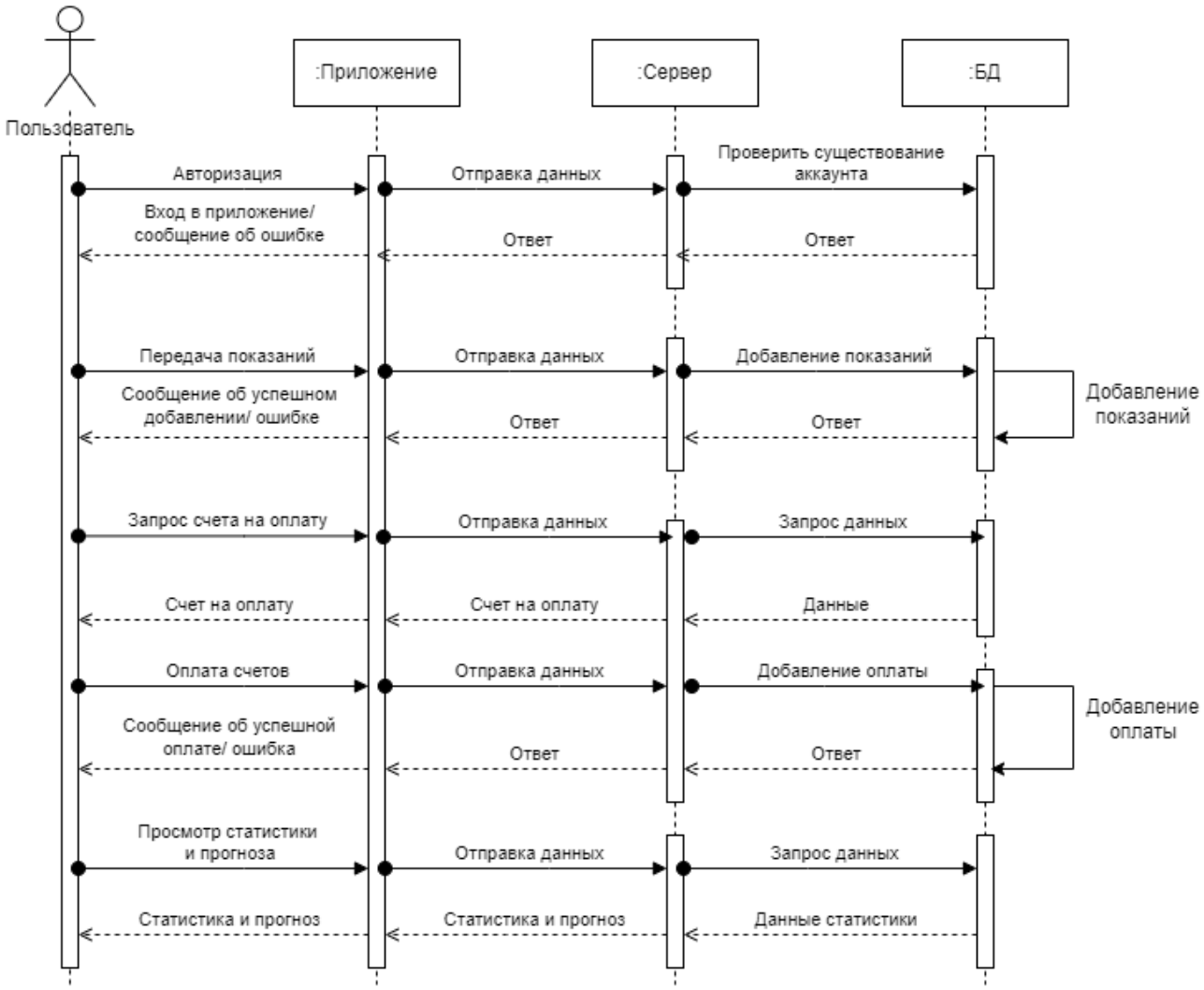

Рисунок 8 - Sequence diagram

2.4.5 Диаграмма развёртывания (Deployment diagram)

Диаграмма развёртывания (deployment diagram) является одним из видов диаграмм UML, используемых для моделирования физической конфигурации и развёртывания компонентов системы на аппаратном и программном обеспечении.

Основная цель диаграммы развёртывания состоит в визуализации физического размещения компонентов системы и их взаимодействия на различных устройствах. Она помогает понять, как компоненты развертываются на аппаратных устройствах, какие связи и зависимости между ними существуют, а также какие ресурсы (процессор, память, хранилище и т.д.) требуются для работы системы.

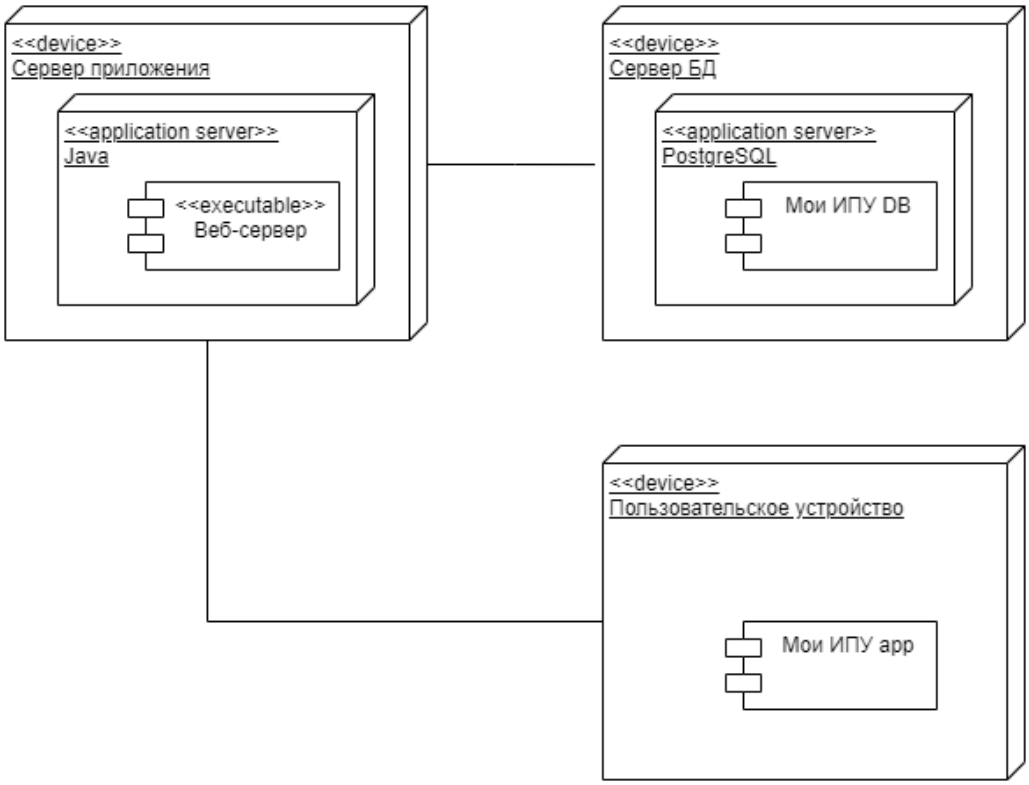

Рисунок 9 - Deployment diagram

2.4.6 Диаграмма сотрудничества (Collaboration diagram)

Диаграмма сотрудничества (collaboration diagram), также известная как диаграмма коммуникации, является одним из видов диаграмм UML, используемых для визуализации взаимодействия между объектами или компонентами системы.

Основная цель диаграммы сотрудничества заключается в визуализации взаимодействия и коммуникации между объектами или компонентами системы. Она помогает понять, как объекты сотрудничают друг с другом, какие сообщения они обмениваются и каким образом взаимодействие влияет на поведение системы.

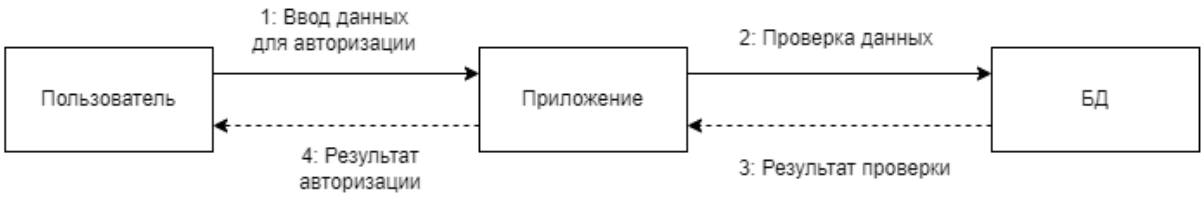

Рисунок 10 - Авторизация

Рисунок 11 - Передача показаний

Рисунок 12 - Оплата счета

2.4.7 Диаграмма объектов (Object diagram)

Диаграмма объектов (object diagram) является одним из видов диаграмм UML, используемых для визуализации статической структуры системы, отображая объекты и их отношения в конкретном моменте времени.

Основная цель диаграммы объектов заключается в визуализации статической структуры системы и отношений между объектами. Она помогает увидеть, какие объекты существуют в системе, какие значения у них есть для атрибутов, и как они связаны друг с другом.

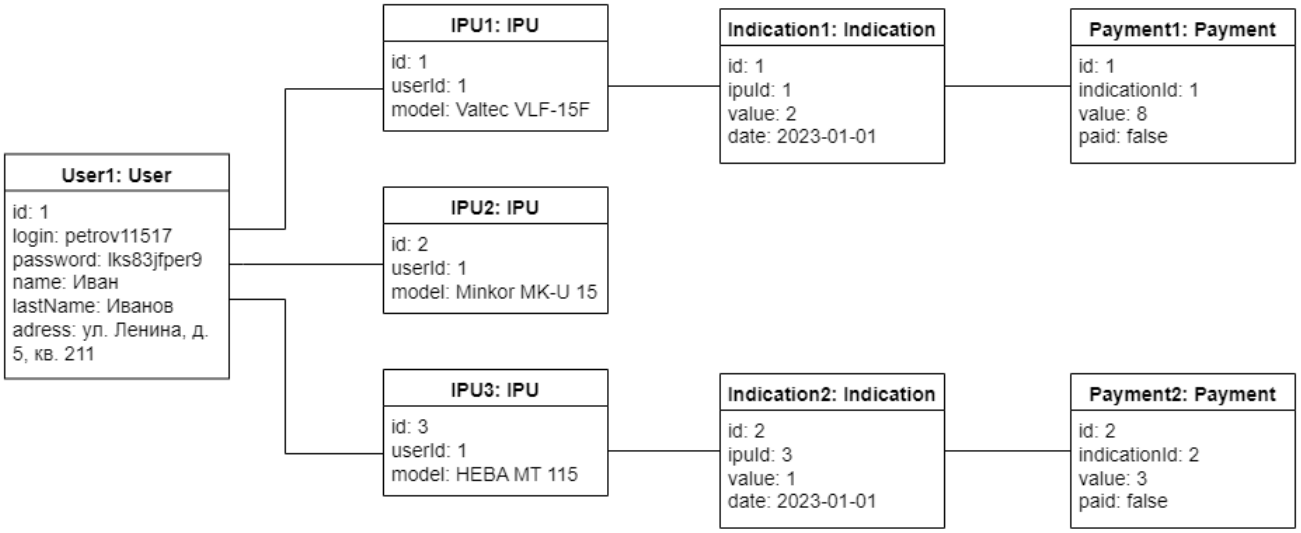

Рисунок 13 - Object diagram

2.4.8 Диаграмма состояний (Statechart diagram)

Диаграмма состояний (statechart diagram) является одним из видов диаграмм UML, используемых для моделирования динамического поведения системы или объекта, показывая его возможные состояния и переходы между ними.

Основная цель диаграммы состояний состоит в визуализации и моделировании динамического поведения системы или объекта. Она помогает понять, как объект или система реагируют на события и какие действия выполняются в каждом состоянии. Также диаграмма состояний позволяет выявить возможные состояния и переходы, идентифицировать поведенческие правила и ограничения, а также моделировать сложные процессы или алгоритмы.

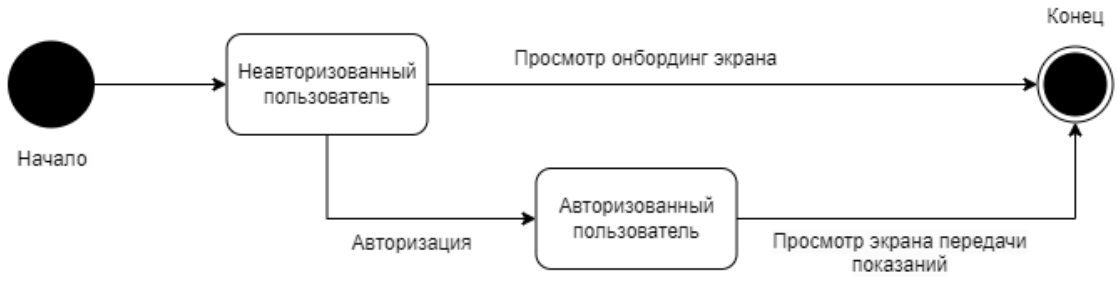

Рисунок 14 - Диаграмма состояний пользователя

На рисунке 14 отображены возможные состояния пользователя. При входе в приложение он неавторизованный, после прохождения авторизации — авторизованный.

Рисунок 15 - Диаграмма состояний показаний

На рисунке 15 отображены возможные состояния показаний. Изначально они не переданы, после передачи пользователем — переданы, после повторной передачи — изменены.

Рисунок 16 - Диаграмма состояний платежа

На рисунке 16 отображены возможные состояния платежа. Пока не переданы показания он не сформирована, после передачи показаний — сформирована, после оплаты пользователем — оплачены.

2.4.9 IDEF0 диаграмма

IDEF0 (Integration Definition for Function Modeling) — это методология и язык моделирования, используемые для описания функциональных аспектов системы или организации. IDEF0 диаграмма — это один из инструментов, используемых в рамках IDEF0 методологии для визуализации функциональной модели.

IDEF0 диаграмма представляет собой блок-схему, состоящую из блоков и стрелок, где каждый блок представляет функцию, выполняемую в системе, а стрелки показывают потоки данных или управления между функциями. Блоки могут быть иерархически организованы, что позволяет строить детализированные модели функций.

Основная цель IDEF0 диаграммы состоит в предоставлении наглядной модели функциональной структуры системы или организации.

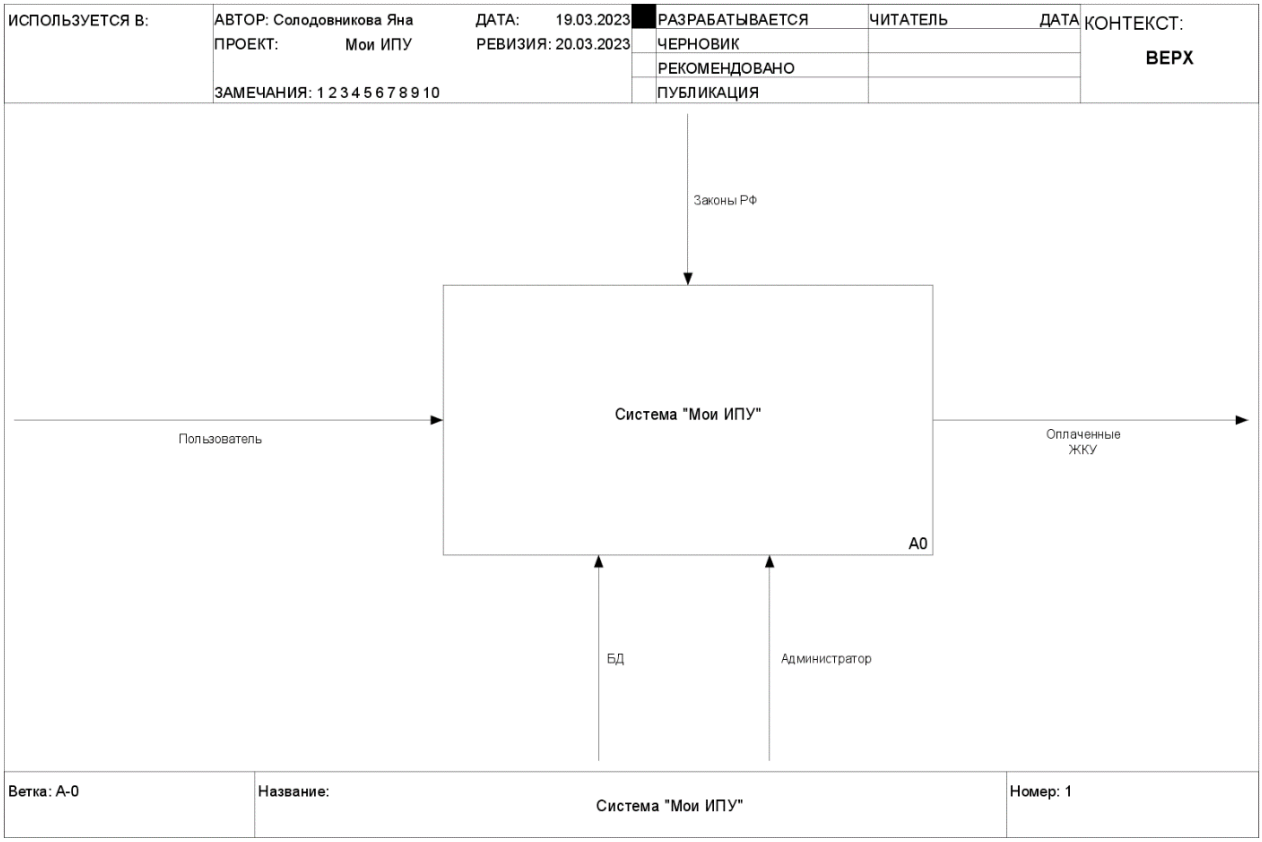

Рисунок 17 - Система «Мои ИПУ» (0 уровень)

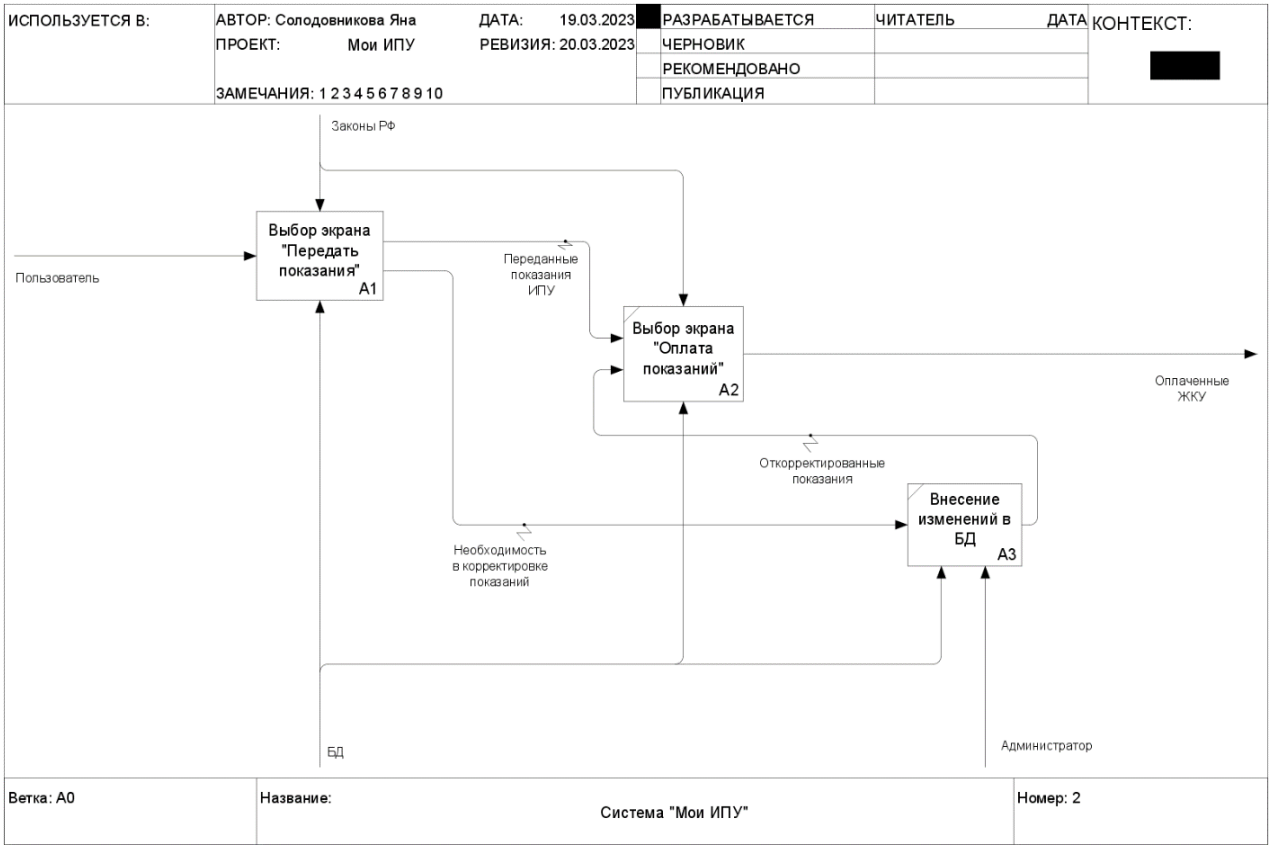

Рисунок 18 - Система «Мои ИПУ» (1 уровень)

3 Реализация

3.1 Средства реализации

В качестве языка программирования для разработки серверной части был выбран Java — это широко используемый объектно-ориентированный язык программирования, предлагающий ряд преимуществ, среди которых:

— Переносимость: Программы на Java могут работать на разных платформах без изменений;

— Большая библиотека: Java предлагает обширную стандартную библиотеку классов для разработки приложений;

— Объектно-ориентированный подход: Java поддерживает полностью объектно-ориентированное программирование, что упрощает создание гибких и расширяемых приложений;

— Безопасность: Java предоставляет встроенные механизмы безопасности, защищающие приложения от угроз;

— Многопоточность: Java поддерживает разработку многопоточных приложений, что позволяет эффективно использовать ресурсы системы.

В качестве фреймворка для разработки серверной части приложения был выбран Spring — популярный фреймворк для разработки приложений на языке Java, предлагающий ряд преимуществ, среди которых:

— Легкость использования: Spring предоставляет простую и интуитивно понятную модель программирования;

— Инверсия управления (IoC) и внедрение зависимостей (DI): Spring упрощает разработку приложений и обеспечивает гибкую связь между компонентами;

— Аспектно-ориентированное программирование (AOP): Spring поддерживает разделение перекрестно-резонирующих аспектов приложения;

— Модульность: Spring предлагает модульную архитектуру, что позволяет выбирать необходимые компоненты и расширять функциональность;

— Расширяемость: Spring поддерживает сторонние библиотеки и имеет активное сообщество разработчиков.

В качестве СУБД была выбрана PostgreSQL — мощная и расширяемая система управления базами данных, предлагающий ряд преимуществ, среди которых:

— Надежность: PostgreSQL обеспечивает высокую надежность и устойчивость, благодаря встроенным механизмам сохранности данных и поддержке транзакций;

— Расширяемость: PostgreSQL предлагает обширный набор встроенных и пользовательских расширений, что позволяет адаптировать СУБД под конкретные потребности и расширять ее функциональность;

— Гибкость модели данных: PostgreSQL поддерживает различные типы данных, включая географические, JSON, XML и другие, что обеспечивает гибкость при моделировании данных;

— Масштабируемость: PostgreSQL позволяет горизонтальное и вертикальное масштабирование, что позволяет увеличивать производительность и обрабатывать большие объемы данных;

— Поддержка SQL-стандартов: PostgreSQL активно следует стандартам SQL и предоставляет обширный набор функций и операторов, что упрощает разработку и миграцию приложений.

3.2 Разработка Frontend

3.3 Разработка Backend

Разработка серверной части приложения проходила в соответствии со следующими этапами:

— инициализация сервера без привязки к БД;

— проектирование БД;

— создание БД;

— связь сервера с БД;

— реализация REST запросов.

3.3.1 Инициализация сервера без привязки к БД

На первом этапе необходимо было создать и настроить связь между классами: Controller, Service, Entity, DTO, Storage. В качестве entity выступают Indication, Ipu и Person.

Для обработки запросов, связанных с пользователем необходим PersonController. В нем есть POST метод authorization, который используется для авторизации пользователя. Он принимает объект класса PersonAuthDto, содержащий лицевой счет и пароль, и возвращает объект класса ResponseEntity<PersonDto>.

Для реализации логики запросов необходим PersonService. В нем реализован метод authorization, который осуществляет процесс авторизации пользователя. Он принимает объект класса PersonAuthDto в качестве параметра и возвращает объект класса PersonDto, содержащий полную информацию о пользователе.

Для хранения и обработки информации в PersonService используется PersonStorage. На данном этапе PersonStorage является Map, где ключом является personalAccount, а значением — объект класса PersonEntity.

Для обработки запросов, связанных с показаниями необходим IndicationController. В нем есть POST метод addIndications, который используется для передачи показаний. Он принимает IndicationWrapper в качестве параметра и ничего не возвращает. Данный оберточный класс обрабатывает входящий List<IndicationCraeteDto>. Также есть GET метод getCurrentIndications, который используется для получения текущих показаний. Он принимает personalAccount в качестве параметра и возвращает объект класса ResponseEntity< List<IndicationDto>>.

Для реализации логики запросов необходим IndicationService. В нем реализован метод addIndications, который осуществляет процесс добавления новых показаний. Он принимает List<IndicationCraeteDto> в качестве параметра и ничего не возвращает. Также в нем реализован метод getCurrentIndications, который осуществляет процесс получения текущих показаний. Он принимает personalAccount в качестве параметра и возвращает List<IndicationDto>.

Для хранения и обработки информации о показаниях в IndicationService используется IndicationStorage. На данном этапе IndicationStorage является Map, где ключом является id, а значением — объект класса IndicationEntity. Для хранения информации об ИПУ IndicationService используется IpuStorage. На данном этапе IpuStorage является Map, где ключом является id, а значением — объект класса IpuEntity.

Для обработки запросов, связанных с платежами необходим PaymentController. В нем есть POST метод pay, который используется для оплаты счетов. Он принимает объект класса PersonGetDto в качестве параметра и ничего не возвращает. GET метод getPayments, который используется для получения текущих счетов на оплату. Он принимает personalAccount в качестве параметра и возвращает объект класса ResponseEntity<List<PaymentDto>>. Также есть GET метод getExpectedPayment, который используется для получения ожидаемого

размера платежа в следующем месяце. Он принимает personalAccount в качестве параметра и возвращает объект класса ResponseEntity<Integer>.

Для реализации логики запросов необходим PaymentService. В нем реализован метод pay, который осуществляет процесс оплаты счетов. Он принимает объект класса PersonGetDto в качестве параметра и ничего не возвращает. Реализован метод getPayments, который осуществляет процесс получения текущих счетов на оплату. Он принимает personalAccount в качестве параметра и возвращает List< PaymentDto>. Также реализован метод getExpectedPayment, который осуществляет процесс получения ожидаемого размера платежа на следующий расчетный период. Он принимает personalAccount в качестве параметра и возвращает int значение.

Для хранения и обработки информации о платежах в PaymentService используется IndicationStorage. На данном этапе IndicationStorage является Map, где ключом является id, а значением — объект класса IndicationEntity. Для хранения информации об ИПУ — IpuStorage, для хранения информации о пользователях — PersonStorage.

Таким образом был проинициализирован сервер, созданы классы и настроена связь между ними.

3.3.2 Проектирование БД

На этапе проектирования БД были созданы ER-диаграмма и физическая модель БД.

ER-диаграмма (сущность-связь) — это графическое представление структуры и взаимосвязей данных в базе данных. Она моделирует сущности (объекты) в системе, их атрибуты (характеристики) и связи между ними.

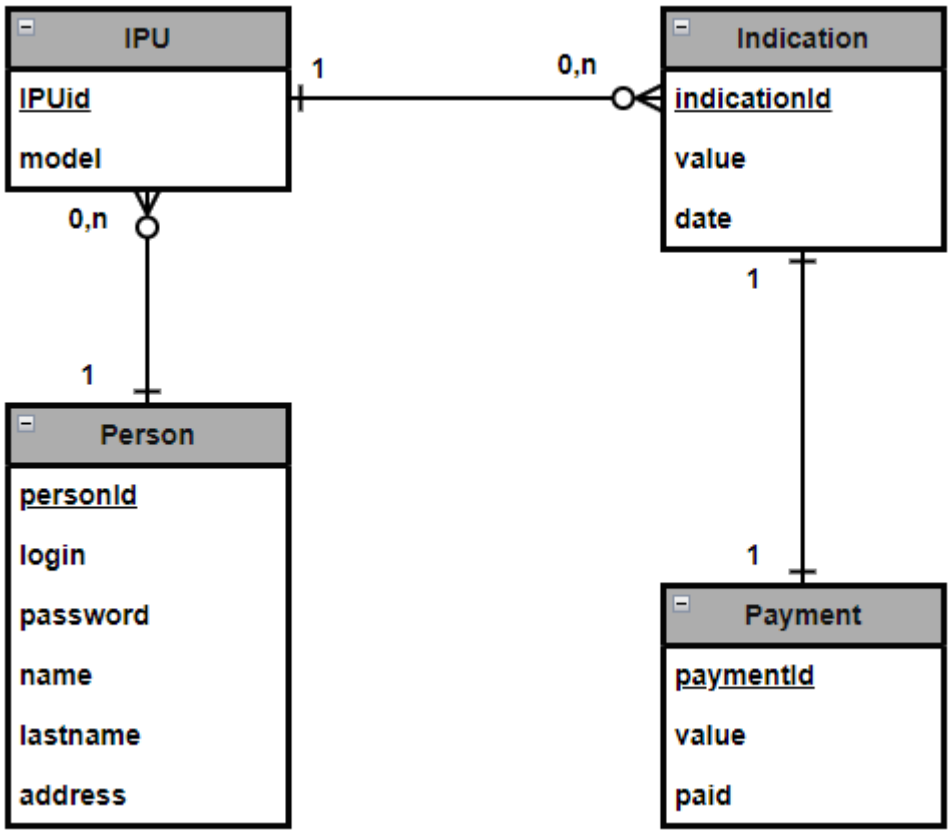

Рисунок 19 - ER диаграмма

На рисунке 19 приведена ER-диаграмма, на которой есть сущности Person, IPU, Indication, Payment, а также настроена связь между ними. По данной диаграмме составлена физическая модель БД (рисунок 20).

Физическая модель базы данных представляет собой конкретное описание структуры базы данных, которое определяет, как данные будут храниться на физическом уровне. Она определяет способ организации таблиц, индексов, ограничений, связей и других аспектов базы данных, с учетом особенностей конкретной системы управления базами данных (СУБД).

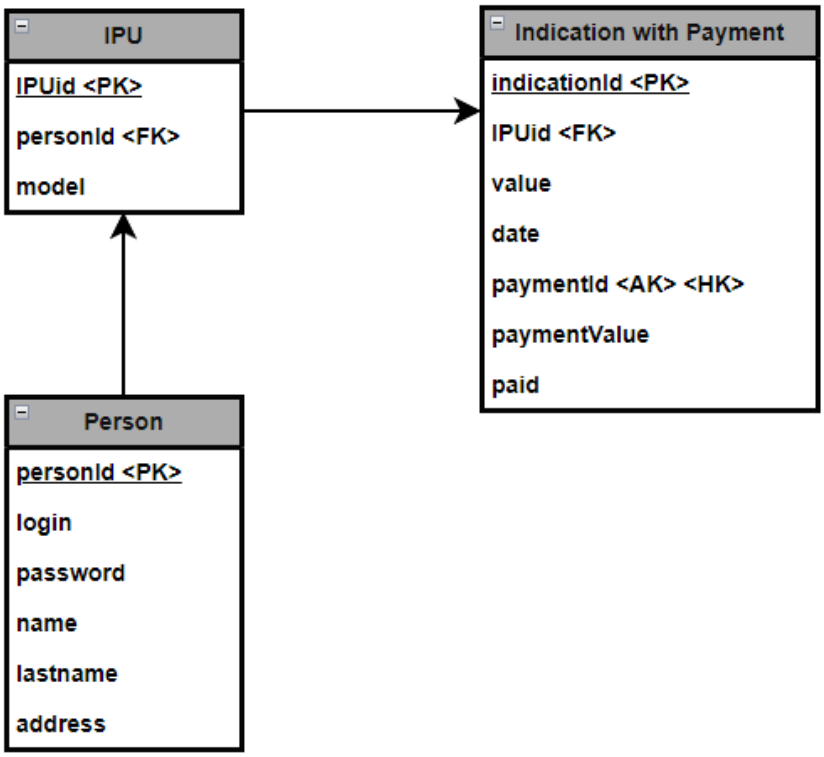

Рисунок 20 - Физическая модель БД

3.3.3 Создание БД

На данном этапе необходимо создать локальную БД. В ней не нужно создавать таблицы, поскольку это будет реализовано самим сервером на следующем этапе.

Во фреймворке Spring существует механизм ORM (Object-Relational Mapping), который используется для создания таблиц БД на основе сущностей (Entity). Он включает в себя компоненты Hibernate/JPA.

Hibernate/JPA — это фреймворк и API для работы с базами данных в Java. Они позволяют упростить взаимодействие с базой данных, оперируя объектами и связями между ними, а не таблицами и столбцами. Hibernate является реализацией JPA и предоставляет удобные инструменты для маппинга объектов на таблицы, выполнения запросов и управления транзакциями.

3.3.4 Связь сервера и БД

На данном этапе для создания связи между сервером и БД необходимо было задать параметры подключения к БД в файле application.properties. Также необходимо было заменить все Storage классы на интерфейсы, наследующие JpaRepository. В них реализованы все необходимые методы для взаимодействия с БД.

Для создания таблиц по Entity, в них необходимо было добавить аннотации @Entity, @Table и @Column. В соответствии с физической моделью БД между Entity существуют различные связи, для реализации которых использовались аннотации @ManyToOne и @OneToMany.

3.3.5 Реализация REST запросов

На первом этапе был проинициализирован сервер, на данном этапе необходимо реализовать логику REST запросов.

Метод authorization в PersonService находит по логину и паролю данные пользователя в таблице Person и возвращает их. В случае, если пользователя в таблице найти не удалось, возвращается null значение.

Метод getCurrentIndications в IndicationService по лицевому счету находит какие ИПУ принадлежат пользователю, перебирает каждый ИПУ и для него из таблицы Indication получает последние переданные показания. Все показания добавляются в список, который в конце и возвращает метод.

Метод addIndications в IndicationService перебирает все переданные показания, для каждого из них находит ИПУ, к которому оно относится, и рассчитывает размер платежа за потребленные услуги в соответствии с тарифом. После этого формирует запись и добавляет ее в таблицу Indication.

Метод getPayments в PaymentService по лицевому счету находит какие ИПУ принадлежат пользователю, перебирает каждый ИПУ и для него из таблицы Indication получает сумму всех неоплаченных счетов. Полученные платежи добавляются в список, который в конце и возвращает метод.

Метод getExpectedPayment в PaymentService по лицевому счету находит какие ИПУ принадлежат пользователю, перебирает каждый ИПУ и

для него из таблицы Indication получает размер последних 5 платежей. Они суммируются и в конце полученное число делится на количество учтенных платежей, то есть данный метод рассчитывает средней значение последних 5 платежей.

Метод pay в PaymentService по лицевому счету находит какие ИПУ принадлежат пользователю, перебирает каждый ИПУ и для него в таблице Indication устанавливает значение «оплачено» для всех неоплаченных показаний, относящихся к нему.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое база данных? // Oracle — URL: https://www.oracle.com/cis/database/what-is-database/ (дата обращения: 12.03.2023). — Текст: электронный.

2. ГОСТ Р 51929-2002. Услуги жилищно-коммунальные. Термины и определения: принят и введен в действие Постановлением Госстандарта России от 20 августа 2002 г. N 307-ст: дата введения 2003-01-01. — URL: https://docs.cntd.ru/document/1200030456 (дата обращения: 12.03.2023). — Текст: электронный.

3. Зачем нужен личный кабинет // Inostudio — URL: https://inostudio.com/blog/articles-managment/zachem-nuzhen-lichnyy-kabinet/ (дата обращения: 12.03.2023). — Текст: электронный.

4. Лицевые счёта // Большая советская энциклопедия : [в 30 т.] / гл. ред. А. М. Прохоров. — 3-е изд. — М. : Советская энциклопедия, 1969—1978.

5. Как формируются тарифы ЖКХ // VGKH — URL: https://vgkh.ru/articles/kak-formiruyutsya-tarify-zhkkh/ (дата обращения: 12.03.2023). — Текст: электронный.