МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Мобильные и сетевые технологии
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему: «Разработка и администрирование веб-приложения управления коммунальными платежами»

Обучающийся А.А. Тятюшев
(И.О. Фамилия) (личная подпись)

Руководитель к.т.н., Н.В. Хрипунов
(ученая степень, звание, И.О. Фамилия)

Консультант к.п.н., доцент С.А. Гудкова
(ученая степень, звание, И.О. Фамилия)

Тольятти 2025

Аннотация

Тема бакалаврской работы: «Разработка и администрирование веб-приложения управления коммунальными платежами».

Цель данной работы — создать универсальное программное решение, обеспечивающее эффективное управление коммунальными платежами пользователей. Приложение позволит пользователям отслеживать начисления, оплачивать счета онлайн, получать уведомления о задолженностях и анализировать потребление коммунальных услуг по разным категориям.

Актуальность темы обусловлена растущими требованиями к автоматизации процессов учета платежей и улучшению взаимодействия между поставщиками коммунальных услуг и потребителями. Современное цифровое пространство требует удобных инструментов, позволяющих упростить процесс оплаты и повысить прозрачность расчетов. Структура работы включает введение, три основных раздела, заключение и приложение.

В первом разделе проведен анализ предметной области, рассмотрены ключевые термины, исследованы возможности применения технологий в сфере жилищно-коммунального хозяйства, а также рассмотрены существующие решения, реализованные в данной области.

Второй раздел представляет проектирование веб-приложения. Описывается архитектура программного продукта.

Третий раздел посвящен процессу реализации и администрирования приложения. В ней подробно описывается выбор инструментов реализации,программная реализация и тестирования.

Abstract

Bachelor's thesis topic: "Development and administration of a web-based utility payment management application".

The purpose of this work is to create a universal software solution that provides effective management of utility bills for users. The application will allow users to track charges, pay bills online, receive debt notifications, and analyze utility consumption by different categories.

The relevance of the topic is due to the growing requirements for automating payment accounting processes and improving interaction between utility service providers and consumers. The modern digital space requires convenient tools to simplify the payment process and increase the transparency of payments. The structure of the work includes an introduction, three main sections, a conclusion and an appendix.

In the first section, the subject area is analyzed, key terms are considered, the possibilities of using technologies in the field of housing and communal services are explored, and existing solutions implemented in this area are considered.

The second section presents the design of a web application. The architecture of the software product is described.

The third section is devoted to the process of application implementation and administration. It describes in detail the choice of implementation tools, software implementation and testing.

Содержание

Введение

Актуальность темы дипломной работы обусловлена возрастающей потребностью в автоматизации процессов управления коммунальными платежами[4]. В условиях современного мира, где время является одним из самых ценных ресурсов, автоматизация рутинных операций позволяет значительно повысить эффективность работы организаций, сократить издержки и улучшить качество обслуживания клиентов. Компания "Квартплата24", занимающаяся управлением коммунальными платежами, сталкивается с необходимостью оптимизации своих процессов, что делает разработку специализированного веб-приложения крайне востребованной.

Целью данной работы является проектирование и реализация веб-приложения для управления коммунальными платежами, которое позволит автоматизировать ключевые процессы, такие как начисление платежей, их оплата, ведение истории платежей и предоставление отчетности, а также обеспечить стабильную и безопасную работу системы[7].

Достижение этой цели предполагает решение следующих задач:

- проведение анализа деятельности компании "Квартплата24" и выявление ключевых процессов, требующих автоматизации;
- разработка требований к веб-приложению на основе анализа существующих решений и потребностей компании;
- проектирование логической и физической структуры приложения, включая модели данных, пользовательские сценарии и архитектуру системы;
- реализация веб-приложения с использованием современных инструментов и технологий;
- обеспечение администрирования приложения, включая настройку серверов, мониторинг производительности, резервное копирование данных и обеспечение безопасности;

– тестирование приложения для обеспечения его корректной работы и соответствия предъявляемым требованиям.

Объектом исследования является процесс управления коммунальными платежами в компании "Квартплата24". Предметом исследования выступает разработка и администрирование веб-приложения, автоматизирующего данный процесс. Практическая значимость работы заключается в том, что внедрение разработанного решения позволит:

- сократить время обработки платежей на 40-60%;
- уменьшить количество ошибок при начислении платежей;
- повысить удовлетворённость клиентов за счёт прозрачности расчетов;
- обеспечить соответствие требованиям законодательства в сфере ЖКХ.

Результаты проекта могут быть адаптированы для других управляющих компаний, что делает исследование особенно ценным для всей отрасли жилищно-коммунального хозяйства.

1 Характеристика объекта автоматизации

В данной главе будет проведен анализ предметной области, рассмотрены ключевые термины, исследованы возможности применения технологий в сфере жилищно-коммунального хозяйства, а также рассмотрены существующие решения, реализованные в данной области.

1.1 Характеристика компании «ООО Квартплата 24»

Компания «ООО Квартплата 24» является специализированной организацией, занимающейся управлением и обработкой коммунальных платежей. Основная деятельность компании заключается в предоставлении услуг по начислению, учету и обработке платежей за жилищно-коммунальные услуги (ЖКУ), таких как электроэнергия, водоснабжение, газ, отопление и другие. Компания сотрудничает с управляющими компаниями, товариществами собственников жилья (ТСЖ) и непосредственно с жильцами многоквартирных и частных домов.

Основные направления деятельности компании:

- начисление коммунальных платежей: расчет сумм к оплате на основе тарифов, показаний счетчиков и нормативов потребления;
- обработка платежей: прием и учет платежей от физических и юридических лиц через различные каналы (банковские переводы, онлайн-платежи, наличные расчеты);
- ведение базы данных: учет абонентов, управление лицевыми счетами, хранение истории платежей и начислений;
- формирование отчетности: подготовка отчетов для управляющих компаний, ТСЖ и контролирующих органов;
- консультации и поддержка клиентов: оказание помощи абонентам по вопросам начислений, оплаты и других аспектов коммунальных услуг.

Компания «ООО Квартплата 24» имеет четкую организационную структуру, которая приведена на рисунке 1.

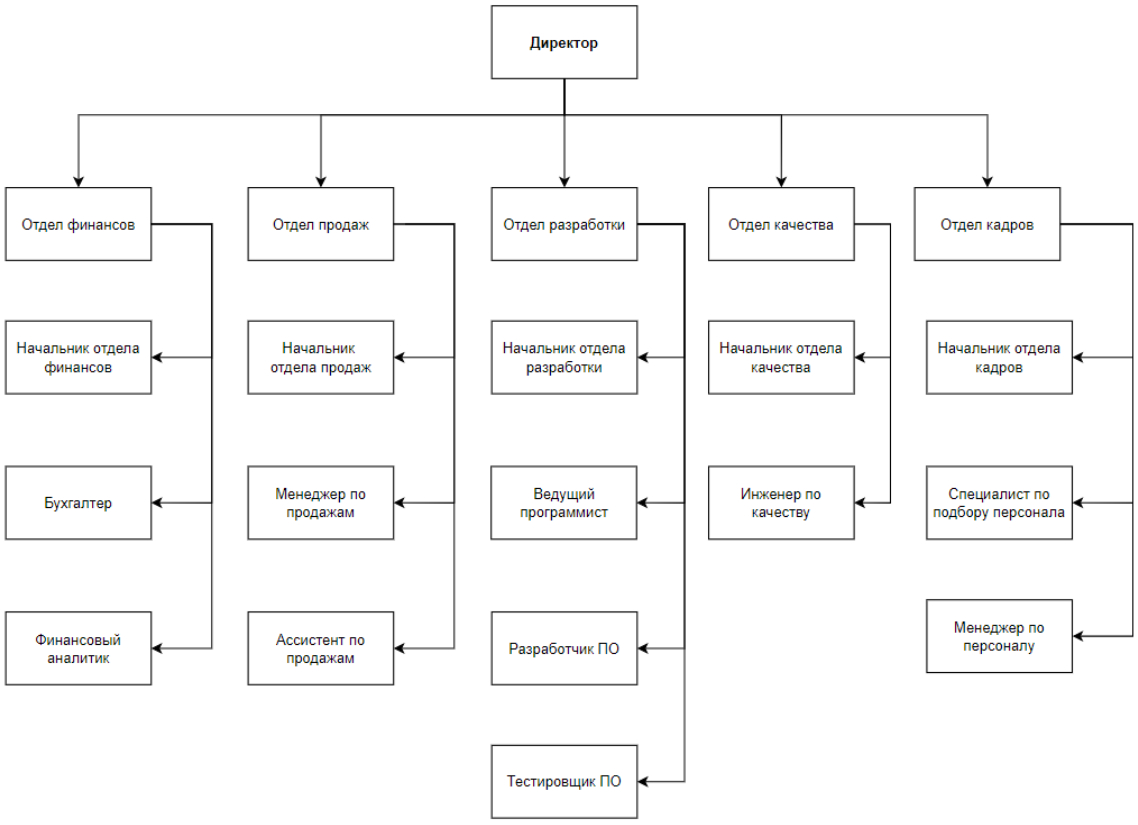


Рисунок 1 – Организационная структура компании «Квартплата 24»

Представленная на рисунке 1 организационная структура компании «Квартплата 24» подразумевает, что директору подчиняются начальники отделов. В свою очередь начальникам отделов подчиняются сотрудники соответствующих отделов. «Квартплата 24» предоставляет сервисы по расчету и приему платежей ЖКУ [5]. Экосистема сервисов «Квартплата 24» представляет собой более 10 тесно интегрированных между собой облачных сервисов. На рисунке 2 представлена диаграмма сервисов организации «Квартплата 24».

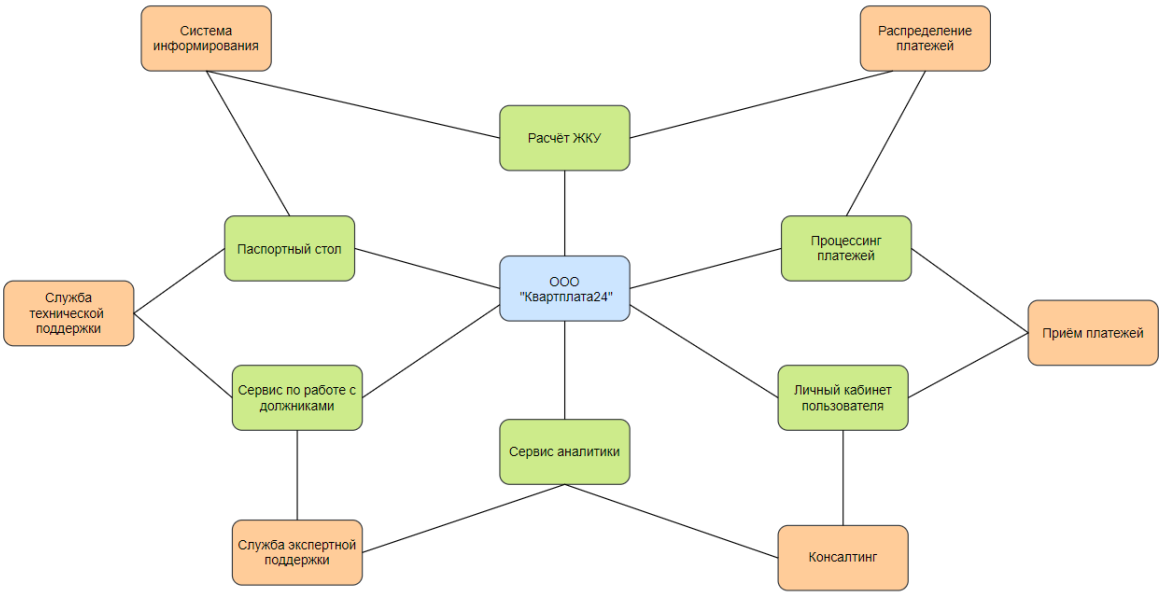


Рисунок 2 – Диаграмма сервисов «Квартплата 24»

В основе работы сервисов «Квартплата 24», которые представлены на рисунке 2, лежат высокотехнологичные IT-решения, которые позволяют оптимизировать процессы учета и расчетов между потребителями и поставщиками услуг. Это включает в себя использование машинного обучения для анализа платежных потоков[5] и управления дебиторской задолженностью. Пользовательский интерфейс и мобильное приложение. «Квартплата 24» предлагает удобный пользовательский интерфейс и мобильное приложение, которые делают процесс оплаты ЖКУ простым и доступным с любого устройства. Пользователи могут управлять своими платежами, просматривать историю транзакций и получать уведомления о счетах и платежах. Консультационные и поддерживающие услуги. Сервис предоставляет круглосуточную поддержку клиентов, что включает телефонные звонки, чаты в приложении и электронные запросы. Команда поддержки помогает решать вопросы, связанные с платежами и обслуживанием счетов, а также консультирует по вопросам экономии на коммунальных услугах. Широкая сеть партнеров. «Квартплата 24» сотрудничает с широким кругом финансовых учреждений, интеграторами

систем безопасности и поставщиками коммунальных услуг, чтобы предложить потребителям лучшие условия и сервис.Экологическая инициатива. Сервис активно внедряет экологические инициативы, направленные на снижение использования бумаги за счет электронных платежей и выписок, что способствует защите окружающей среды. Безопасность и конфиденциальность. «Квартплата 24» придает особое значение защите данных клиентов. Все платежные транзакции и пользовательские данные защищены современными методами шифрования и соответствуют международным стандартам безопасности[8].

1.2 Анализ бизнес-процессов

Бизнес-процессы – это последовательность действий, выполняемых в организации для достижения ее целей. Они являются фундаментом функционирования любой компании и могут включать в себя такие процессы, как производство товаров, обработка заказов, управление финансами, маркетинг и другие.

Анализ бизнес-процессов помогает выявить сильные и слабые стороны компании, а также определить пути повышения эффективности и снижения затрат.

IDEF0 (Integration Definition for Function Modeling) — метод моделирования бизнес-процессов. Он используется для описания функций, действий и взаимосвязей внутри организации, а также для обозначения входов и выходов каждого процесса. Модели IDEF0 применяются для анализа и улучшения существующих процессов, а также для создания новых. Диаграммы IDEF0 используют прямоугольники, стрелки и другие символы для представления различных элементов процесса, таких как входы, выходы, элементы управления и механизмы.

Контекстная диаграмма процесса управления коммунальными платежами отражена на рисунке 3.

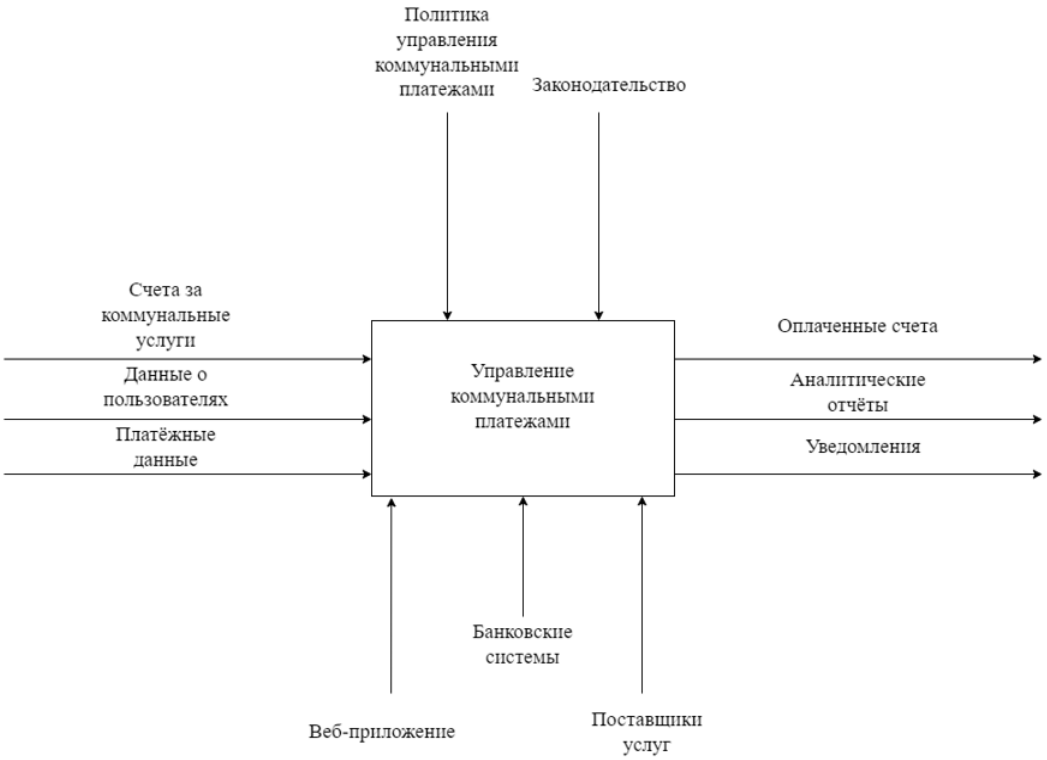


Рисунок 3 – Контекстная диаграмма IDEF0

На рисунке 4 представлена диаграмма декомпозиции первого уровня IDEF0.

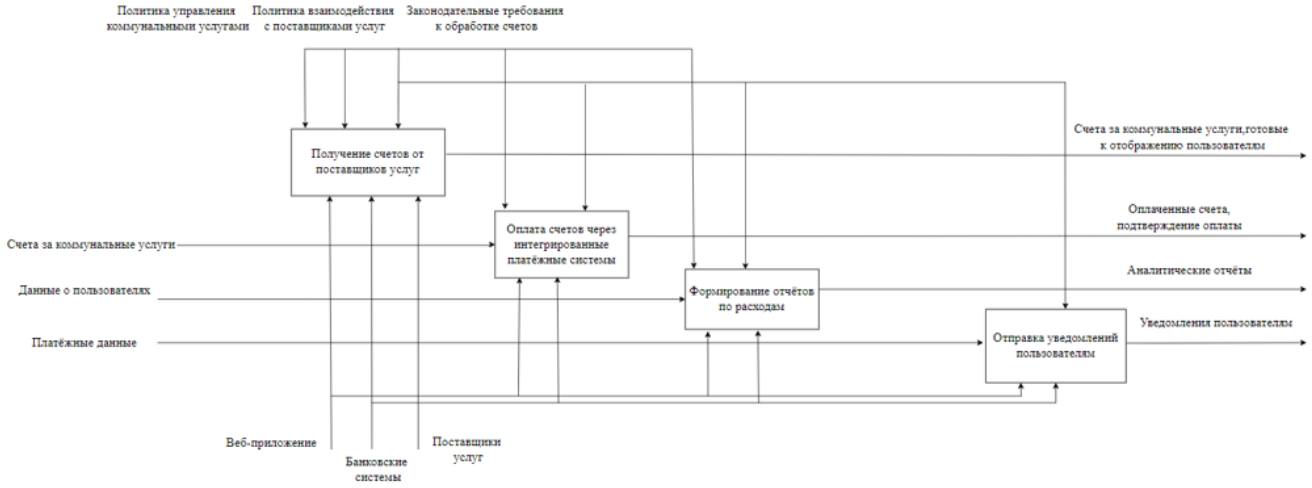


Рисунок 4 – Диаграмма декомпозиции первого уровня IDEF0

Представленная на рисунке 4 диаграмма показывает движение информации между различными сущностями, системами и процессами в рамках обработки коммунальных платежей. Основные элементы диаграммы:

- входные данные (политики, счета, данные пользователей);
- процессы (получение счетов, оплата, формирование отчетов, отправка уведомлений);
- выходные данные (готовые к отображению счета, аналитические отчёты, уведомления);
- источники и приемники данных (веб-приложение, банковские системы, поставщики услуг).

1.3 Разработка требований

Определение требований к веб-приложению для управления коммунальными платежами играет достаточно важную роль, так как от этого зависит его функциональность, надежность и соответствие ожиданиям пользователей. Требования должны учитывать интересы как жителей, так и административного персонала управляющих организаций и поставщиков ресурсов.

Таблица 1 – Функциональные требования для веб-приложения

Аспекты системы	Характеристика и требование
Регистрация и вход пользователей	безопасная регистрация и аутентификация пользователей с применением современных методов защиты данных
Персональный кабинет	доступ к информации о начислениях, подаче показаний счетчиков, оплате услуг и получению уведомлений
Автоматизация оплаты	автоматический расчет сумм платежей с учетом тарифов, скидок и других параметров
Интеграция с внешними системами	взаимодействие с ГИС ЖКХ, банковскими и платежными системами

Таблица 2 – Нефункциональные требования для веб-приложения

Аспекты системы	Характеристика и требование
Безопасность	соответствие стандартам защиты персональных данных, предотвращение несанкционированного доступа
Производительность	обеспечение стабильной работы при высокой нагрузке, возможность масштабирования
Удобство интерфейса	интуитивно понятный и комфортный интерфейс, минимизирующий необходимость обучения
Доступность	кросс-платформенная совместимость, включая мобильные устройства, круглосуточный доступ

Сочетание указанных функциональных и нефункциональных требований, которые представлены в таблице 1 и таблице 2, формирует сбалансированную систему, которая не только эффективно автоматизирует ключевые бизнес-процессы компании, но и обеспечивает высокий уровень безопасности, надежности и удобства использования. Реализация такого решения позволит значительно повысить эффективность работы с коммунальными платежами, сократить операционные издержки и улучшить качество обслуживания клиентов. Более того, применение современных методов аутентификации и защиты информации способствует усилению безопасности и снижению рисков утечки данных.

Тем самым, детальное формулирование как функциональных, так и нефункциональных требований, с учетом передового опыта и научных рекомендаций, является основой успешной разработки и внедрения веб-приложения для администрирования коммунальных платежей.

Перспективы дальнейшего развития системы включают возможность интеграции с системами "умного дома" для автоматического сбора показаний счетчиков, внедрение чат-ботов для обработки типовых запросов пользователей, а также использование технологий больших данных для аналитики потребления ресурсов. Особое внимание при реализации проекта

следует уделить этапу тестирования, чтобы гарантировать корректность финансовых расчетов и устойчивость системы к различным видам нагрузок. Успешное внедрение данного решения создаст прочную технологическую основу для цифровой трансформации процессов управления жилищно-коммунальным хозяйством.

В данном разделе проведен анализ деятельности компании «Квартплата 24», включая её бизнес-процессы и существующие IT-решения. На основе исследования сформулированы функциональные и нефункциональные требования к системе. Разработанные диаграммы IDEF0 и вариантов использования позволили визуализировать ключевые процессы и взаимодействие участников. Результаты анализа показали, что автоматизация процессов управления платежами является критически важной для повышения эффективности работы компании.

2 Проектирование веб-приложения управления коммунальными платежами

Раздел 2 содержит комплексное описание проектирования веб-приложения, начиная от архитектурных решений и логики взаимодействия компонентов, заканчивая моделированием структуры данных. Это позволяет создать надежную, масштабируемую и удобную систему для управления коммунальными платежами.

Проектирование веб-приложения управления коммунальными платежами включает в себя разработку архитектуры системы, проектирование базы данных, определение основных модулей и их взаимодействия. В качестве основного языка программирования для реализации приложения выбран Python, благодаря его надежности, производительности и широким возможностям для создания масштабируемых веб-приложений. Для разработки используются современные фреймворки и технологии, такие как Flask для серверной части и HTML и CSS для фронтенда.

2.1 Архитектурное проектирование

Веб-приложение строится на основе многослойной архитектуры, которая состоит из презентационного уровня (Frontend), бизнес-логики (Backend) и уровня доступа к данным (Data Access Layer).

Frontend

Презентационный уровень отвечает за пользовательский интерфейс и взаимодействие с конечными пользователями. Он реализован на базе современных веб-технологий, включая HTML для структуры страниц, CSS для стилизации и JavaScript для интерактивности. Для повышения эффективности разработки используются популярные фреймворки, такие как Thymeleaf для традиционного подхода или Angular для создания одностраничных

приложений (SPA). Структура презентационного уровня представлена на рисунке 5.

Основные функции:

– отображение личного кабинета пользователя;

– формы для регистрации, авторизации и оплаты счетов;

– отображение истории платежей и начислений.

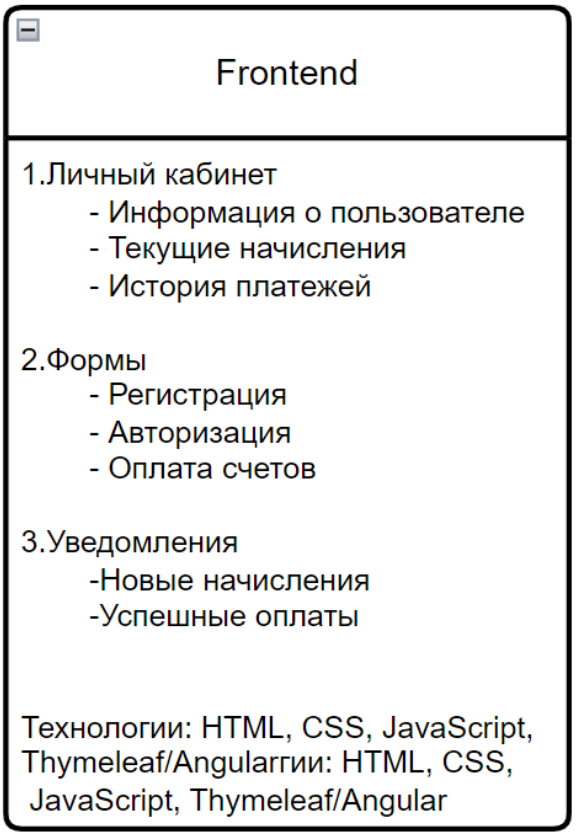


Рисунок 5 – Структура презентационного уровня (Frontend)

Основной функционал фронтенд-части включает отображение личного кабинета пользователя с персональными данными и информацией о платежах, реализацию интуитивно понятных форм для регистрации новых пользователей, авторизации в системе и проведения платежных операций, а также предоставление наглядной истории платежей и начислений. Все элементы интерфейса разрабатываются с учетом принципов юзабилити и

доступности, чтобы обеспечить комфортное взаимодействие для пользователей с разным уровнем технической подготовки.

Backend

Серверная часть приложения (Backend) реализована на языке Python с использованием микрофреймворка Flask, который обеспечивает гибкость и производительность при обработке бизнес-логики. Архитектура backend-компонентов включает несколько специализированных модулей, каждый из которых отвечает за определенный аспект функционирования системы.

Модуль аутентификации и авторизации обеспечивает безопасный доступ пользователей к системе, реализуя механизмы проверки учетных данных и управления правами доступа в соответствии с ролевой моделью[8]. Модуль расчетов выполняет автоматическое начисление платежей, используя актуальные тарифы и данные о потреблении коммунальных ресурсов, что гарантирует точность и прозрачность формируемых квитанций.

Для обработки финансовых операций разработан специализированный платежный модуль, который обеспечивает интеграцию с внешними платежными системами и банковскими сервисами, а также своевременное обновление статусов проведенных платежей. Дополнительный модуль уведомлений автоматически проверяет наличие задолженностей при каждом входе пользователя в систему и оперативно информирует его о текущем состоянии расчетов за коммунальные услуги.

Такая модульная структура backend-части позволяет эффективно масштабировать систему, добавляя новые функции без необходимости существенной переработки существующего кода, а также обеспечивает надежную и бесперебойную работу всех критически важных компонентов приложения.

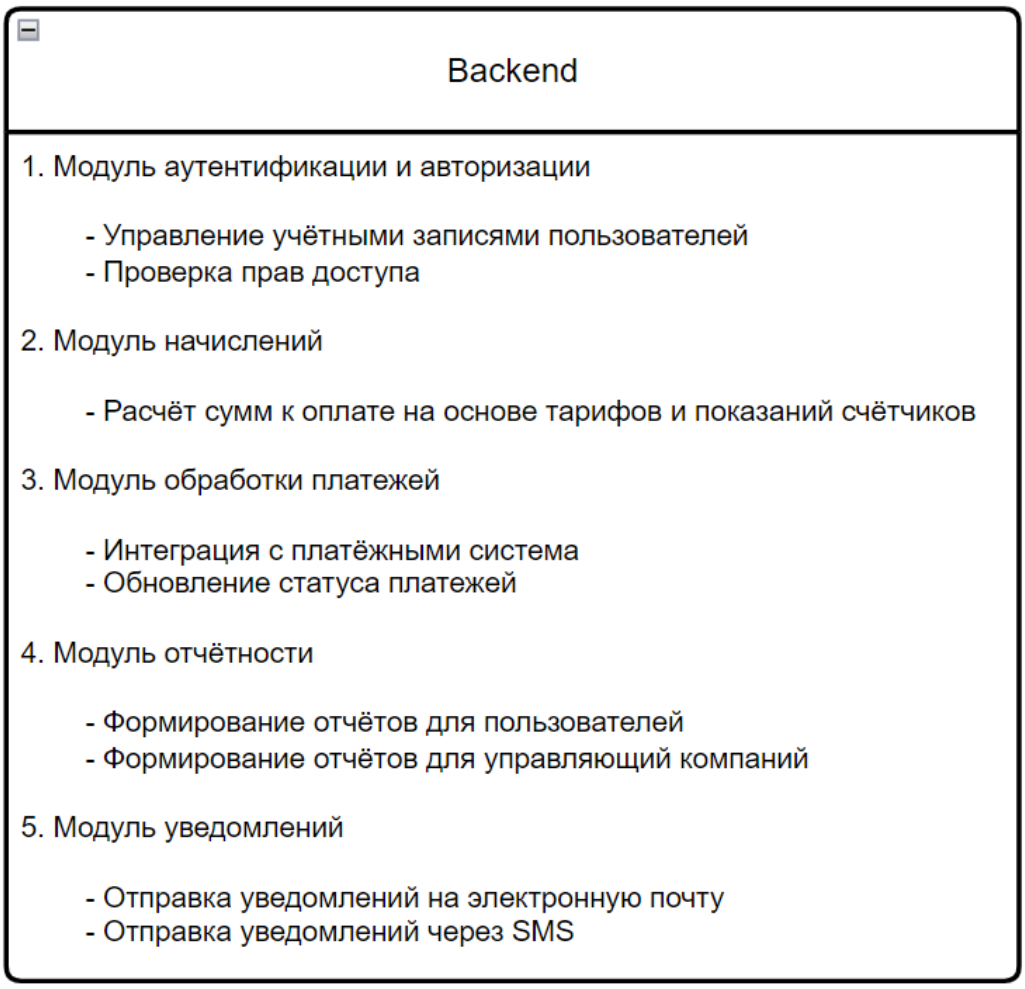


Рисунок 6 – Структура бизнес-логики (Backend)

Data Access Layer

Уровень доступа к данным (Data Access Layer) играет ключевую роль в архитектуре приложения, обеспечивая надежное взаимодействие между бизнес-логикой и системой хранения информации. Для реализации этого слоя используется ORM-библиотека SQLAlchemy, которая значительно упрощает работу с базой данных за счет абстрагирования от специфики SQL-запросов и предоставления объектно-ориентированного интерфейса для манипуляции данными[10],[18].

В системе определены основные сущности данных, включающие пользователей системы, привязанные к ним лицевые счета, историю начислений, информацию о произведенных платежах и актуальные тарифы на

коммунальные услуги. Каждая сущность представлена в виде модели данных с четко определенными атрибутами и связями между ними.

В качестве системы управления базами данных выбрана реляционная СУБД SQLite, которая идеально подходит для данного проекта благодаря своей простоте, надежности и отсутствию необходимости в отдельном сервере баз данных[19]. Структура базы включает нормализованные таблицы для хранения всех типов данных, с продуманными связями между ними, что обеспечивает целостность информации и эффективность выполнения запросов[9],[19]. SQLAlchemy выступает в роли промежуточного слоя, преобразуя объекты Python в реляционные структуры и обратно, что значительно ускоряет разработку и повышает безопасность работы с данными[18].

2.2 Логическое проектирование

Логическая модель работы веб-приложения для управления коммунальными платежами представляет собой основу, на которой строится вся система. Эта модель определяет, как взаимодействуют между собой различные компоненты приложения, а также как осуществляется обработка данных и выполнение ключевых операций. Приложение должно обеспечивать пользователю возможность эффективно и безопасно контролировать свои коммунальные расходы, а также оплачивать счета через удобный интерфейс.

В первую очередь, основная задача приложения — предоставить пользователю доступ к актуальной информации о его коммунальных платежах. Это включает в себя возможность войти в систему с помощью безопасной аутентификации, такой как логин и пароль, а также двухфакторной аутентификации для дополнительной защиты. После того как пользователь вошел в систему, он может просматривать текущие счета за коммунальные услуги, а также получать доступ к истории предыдущих платежей. Система должна работать с базой данных, где хранятся все данные о счетах, платежах

и других деталях. Для удобства пользователя важно, чтобы приложение позволяло сортировать и фильтровать информацию, чтобы ему было легче ориентироваться в большом объеме данных. Например, можно предусмотреть фильтры по дате, по типу услуги или по состоянию счета (оплачен/не оплачен).

Другая важная функция — это возможность создания и редактирования счетов, которая должна быть доступна только администраторам системы. Для этого в логической модели работы приложения должны быть предусмотрены соответствующие алгоритмы, которые обеспечат проверку данных перед созданием счета, а также возможность корректировать информацию в случае ошибок. Создание счета может включать в себя расчет суммы для оплаты с учетом всех тарифов и возможных скидок. Также важно предусмотреть, чтобы все данные обновлялись в реальном времени, и пользователи всегда получали актуальную информацию о своих платежах.

Еще одной ключевой функцией является возможность оплаты счетов через интеграцию с внешними платежными системами. Веб-приложение должно быть связано с несколькими платёжными шлюзами, такими как банковские системы или специализированные сервисы, которые обеспечат выполнение финансовых транзакций. Важно, чтобы процесс оплаты был максимально простым и понятным для пользователя, а также обеспечивал безопасность всех передаваемых данных. Во время оплаты система должна предоставлять пользователю подтверждение успешной операции или, в случае ошибок, выводить сообщение о проблеме (например, недостаточный баланс на счете или сбой в платежной системе).

Говоря о функционале, нельзя забывать и об обработке ошибок, особенно когда речь идет о финансовых операциях. Например, если во время попытки провести онлайн-платеж возникла ошибка (платеж не прошел или произошел сбой в соединении), система должна адекватно отреагировать. Пользователь должен быть информирован о том, что операция не завершена, а также получить рекомендации по дальнейшим действиям — например,

повторить попытку или проверить правильность введенных данных. Логика обработки ошибок должна быть достаточно гибкой, чтобы предусмотреть множество сценариев, от временных сбоев в интернет-соединении до ошибок в данных пользователя. Важно, чтобы приложение не оставляло пользователя в неведении и обеспечивало его информированность о текущем статусе операции.

Безопасность данных является еще одним важным аспектом работы веб-приложения для управления коммунальными платежами. Платежные системы и персональная информация пользователей требуют защиты от различных угроз. Приложение должно обеспечивать защиту данных на всех уровнях[8]. Например, передача данных между клиентом и сервером должна происходить через защищенные протоколы, такие как HTTPS, что гарантирует шифрование данных и защиту от перехвата информации[8]. Все персональные данные пользователя, включая адреса, платежные реквизиты и историю платежей, должны быть зашифрованы как в процессе передачи, так и при хранении на сервере, чтобы исключить возможность их несанкционированного доступа. Для этого можно использовать современные методы шифрования, такие как алгоритм AES для данных на сервере и SSL/TLS для передачи данных по сети[8].

Также важным аспектом является защита от SQL-инъекций и других типов атак, направленных на базу данных. Для этого необходимо использовать подготовленные запросы и избегать прямого включения пользовательских данных в SQL-запросы. Кроме того, приложение должно обеспечивать защиту от атак типа "перебора паролей" (brute force), ограничивая количество попыток ввода неправильного пароля и используя капчу. Все эти меры направлены на минимизацию рисков утечек данных и несанкционированного доступа.

Важно также учитывать, что система должна обеспечивать управление доступом, что предполагает наличие различных ролей с различными правами. Например, пользователи могут иметь доступ только к своим личным данным и платежам, а администраторы смогут изменять тарифы, редактировать счета

и просматривать данные о всех пользователях. Такое разделение функций и прав доступа помогает не только организовать эффективное взаимодействие с системой, но и минимизировать риски утечек или ошибок, связанных с некорректным использованием данных.

Диаграмма вариантов использования наглядно демонстрирует и описывает, какой функционал будет доступен каждой группе пользователей в разрабатываемой программной среде.

В случае с личным кабинетом, основной группой пользователей будут клиенты, которые взаимодействуют с коммунальными услугами внутри личного кабинета с помощью определенных вариантов использования. Но также может быть введен актер «Администратор», который отвечает за управление и администрирование личный кабинетом потребителей при возникновении каких-либо проблем. Администратор имеет расширенные права и работает в определенной среде, позволяющей ему управлять пользователями. Далее представлено описание диаграммы вариантов использования для веб-приложения управления коммунальными платежами.

Основные участники системы:

- Клиент (пользователь) - физическое лицо, использующее приложение для управления своими коммунальными платежами. Основной функционал включает: процедуру регистрации и авторизации, управление объектами недвижимости, просмотр и оплату счетов, а также создание и отслеживание заявок;
- Администратор - сотрудник управляющей компании, ответственный за администрирование системы. В его обязанности входит: управление тарифной политикой, формирование и корректировка счетов, обработка поступающих от пользователей заявок;
- Платежная система - внешний сервис, обеспечивающий обработку финансовых операций. Осуществляет прием платежей и при необходимости - возврат средств;

- Система уведомлений - внешний сервис коммуникации, отвечающий за своевременное информирование пользователей посредством email- и SMS-рассылок о задолженностях.

Система предназначена для автоматизации процессов управления платежами за коммунальные услуги и включает несколько ключевых модулей, взаимодействующих между собой.

Для пользователей система предоставляет удобный личный кабинет, где можно:

- зарегистрироваться и авторизоваться в систем;
- добавлять и просматривать свои объекты недвижимости (квартиры, дома);
- видеть актуальные счета с возможностью фильтрации по дате, сумме и статусу;
- оплачивать счета различными способами (банковской картой, электронным кошельком);
- создавать заявки (например, на перерасчет) и отслеживать их статус.

Администраторы управляющей компании работают с отдельным интерфейсом, позволяющим:

- настраивать и изменять тарифы на услуги;
- формировать и корректировать счета для пользователей;
- обрабатывать поступающие от жильцов заявки и обращения.

Интеграция с внешними системами обеспечивает:

- безопасное проведение платежей через подключенные банки и платежные сервисы;
- автоматическую отправку уведомлений пользователям по email и SMS о новых счетах, напоминаниях об оплате и изменениях статуса заявок.

Вся информация надежно хранится в базе данных, где учтены:

- данные пользователей и их объектов недвижимости;
- история начислений и платежей;

– действующие тарифы;

– заявки и обращения.

Система разработана с учетом удобства использования как для жильцов, так и для сотрудников управляющей компании, что позволяет сделать процесс оплаты коммунальных услуг прозрачным и максимально комфортным для всех участников.

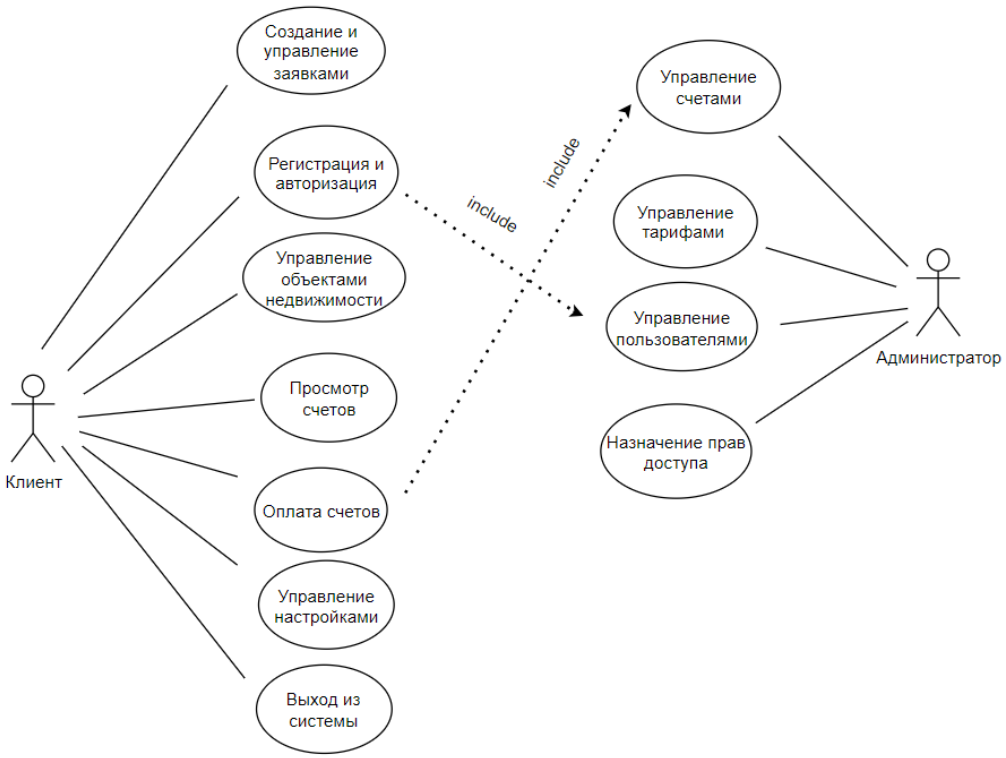


Рисунок 7 – Диаграмма вариантов использования веб-приложения

Диаграмма вариантов использования, представленная на рисунке 7, наглядно демонстрирует взаимодействие основных пользователей с веб-приложением для управления коммунальными платежами. Она охватывает ключевые функции системы, которые обеспечивают удобство и эффективность работы как для пользователей, так и для администраторов.

Также реализована диаграмма последовательности, демонстрирующая обработку REST-запросов:

– пользователь заполняет форму регистрации;

- данная форма проверяется на корректность заполнения (проверка почты,проверка сложности пароля);
- создаётся идентификатор нового пользователя;
- пользователь успешно зарегистрирован.

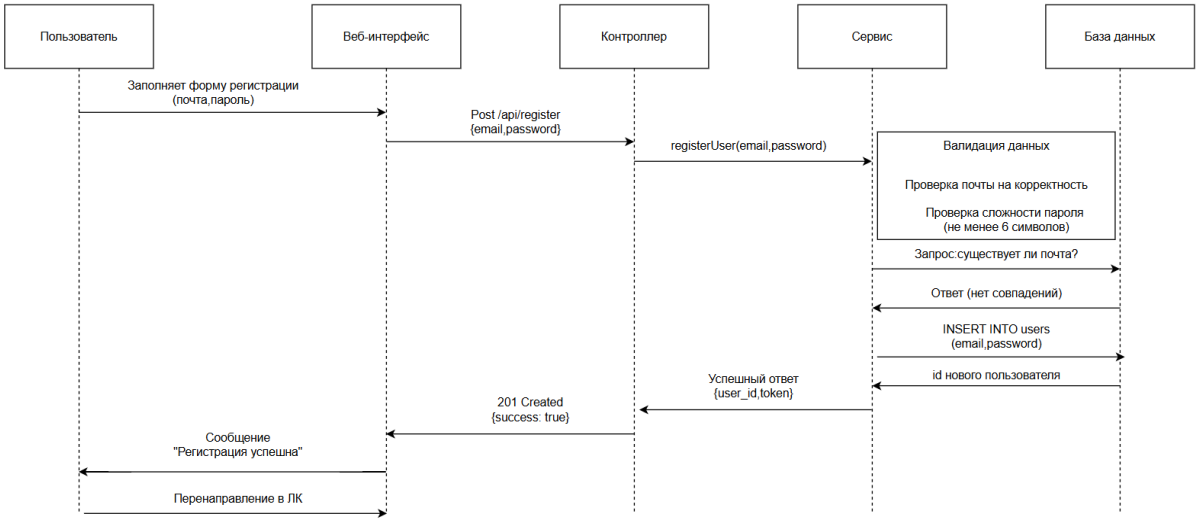


Рисунок 8 – Диаграмма последовательностей

На рисунке 8 представлена диаграмма последовательности,которая визуализирует взаимодействие между Frontend, Backend и Database при выполнении различных REST API запросов.

2.3 Проектирование данных

В данном подразделе рассматривается построение ER-диаграммы, позволяющей наглядно представить структуру базы данных веб-приложения для администрирования коммунальных платежей, а также детально описываются основные модели данных, их атрибуты и взаимосвязи. Правильное моделирование данных является основой для создания надежной, масштабируемой и легко поддерживаемой информационной системы, способной эффективно обрабатывать большие объемы транзакций и обеспечивать прозрачность всех операций.

Проектирование данных основано на принципах нормализации и реляционной модели. В системе выделены ключевые сущности:

- пользователь (User): id, логин, хеш пароля, email, телефон;
- лицевой счёт (Account): id, адрес, id пользователя;
- начисление (Charge): id, id счёта, сумма, услуга, период;
- платёж (Payment): id, id начисления, сумма, дата, статус;
- тариф (Tariff): id, тип услуги, ставка.

Связи реализованы через внешние ключи. Используются отношения "один ко многим": пользователь – лицевые счета, лицевой счёт – начисления, начисления – платежи.

ER-диаграмма базы данных визуализирует структуру и взаимосвязи моделей.

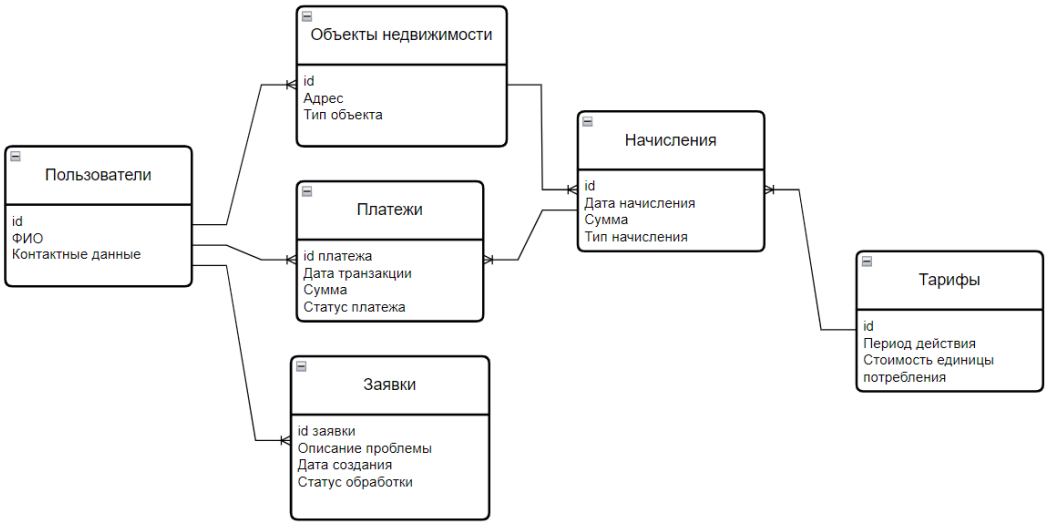


Рисунок 9 – Логическая модель базы данных (ER-диаграмма)

Построение ER-диаграммы и детальное описание моделей данных для веб-приложения управления коммунальными платежами являются важными этапами проектирования, которые закладывают основу для создания надежной, масштабируемой и удобной в эксплуатации базы данных. Этот подход обеспечивает прозрачность расчетов, контроль транзакций и

возможность быстрого реагирования на изменяющиеся требования пользователей, что является критически важным в условиях современной цифровой трансформации ЖКХ.

Раздел посвящен проектированию архитектуры приложения, включая frontend, backend и уровень доступа к данным. Построены диаграммы компонентов, последовательностей и ER-диаграмма базы данных, что обеспечило четкое понимание структуры системы. Выбранная многослойная архитектура и реляционная модель данных позволяют создать масштабируемое и надежное решение. Проектирование логики приложения и взаимодействия пользователей подтвердило соответствие системы поставленным требованиям.

Особое внимание было уделено обеспечению безопасности данных при передаче между уровнями системы. Разработанные механизмы аутентификации и авторизации гарантируют защиту персональной информации пользователей. Оптимизированная структура базы данных обеспечивает высокую производительность при обработке транзакций и формировании отчетности.

3 Реализация веб-приложения управления коммунальными платежами

Реализация веб-приложения для управления коммунальными платежами — это этап, на котором все идеи и концепции, разработанные на предыдущих стадиях проектирования, начинают воплощаться в конкретный продукт. Этот этап является решающим, поскольку на нем реализуется вся функциональность приложения, обеспечивается его интеграция с внешними системами и серверами, а также осуществляется настройка интерфейсов и элементов взаимодействия с пользователем. Реализация должна отвечать не только техническим требованиям, но и обеспечивать высокое качество работы приложения, его удобство для конечных пользователей и безопасность данных. Для этого потребуется правильный выбор технологий, инструментов и платформ, грамотная организация процесса разработки и тестирования, а также внимание к каждому этапу работы.В данном разделе описываются ключевые этапы реализации, используемые технологии и подходы к разработке.

3.1 Выбор инструментов реализации

Выбор инструментов и технологий для разработки веб-приложения — это ключевой этап, который оказывает непосредственное влияние на успешность всего проекта. Технологии должны быть не только функциональными, но и производительными, масштабируемыми и безопасными[7],[8],[9]. Кроме того, они должны обеспечивать удобство разработки и поддержки, позволяя команде сосредоточиться на реализации функционала и удовлетворении потребностей пользователей, а не на технических ограничениях[1]. Важно выбрать такие инструменты, которые гармонично интегрируются друг с другом и обеспечивают надежную основу

для приложения. Ниже прописаны основные технологии и инструменты,которые будут использоваться для реализации веб-приложения:

- python: основной язык программирования для разработки серверной части;
- flask: фреймворк для создания автономных и масштабируемых веб-приложений;
- session: для обеспечения безопасности и управления доступом;
- SQLAlchemy: для работы с базой данных;
- jinja2: для реализации пользовательского интерфейса;
- SQL Lite: реляционная база данных для хранения информации;
- REST API: для взаимодействия между фронтендом и бэкендом;
- maven/gradle: для управления зависимостями и сборки проекта.

Язык программирования Python

Python был выбран для разработки веб-приложения управления коммунальными платежами благодаря своей простоте, читаемости, высокой производительности и богатой экосистеме библиотек и фреймворков [1]. Эти особенности делают Python идеальным выбором для создания масштабируемых, безопасных и удобных в поддержке веб-приложений.

Преимущества выбора python:

- простота и читаемость кода: python обладает чистым и понятным синтаксисом, что ускоряет разработку и облегчает поддержку кода. Это особенно важно для проектов, где требуется быстрая итерация и адаптация к изменяющимся требованиям;
- богатая экосистема библиотек и фреймворков: python предлагает множество специализированных библиотек и фреймворков для веб-разработки, работы с данными, интеграции с внешними сервисами и обеспечения безопасности. Например, Django и Flask предоставляют готовые решения для создания веб-приложений;

– высокая производительность:
благодаря оптимизированным библиотекам (например, asyncio для асинхронного программирования) python обеспечивает высокую производительность даже при работе с большими объемами данных;
– кроссплатформенность:
python работает на различных операционных системах, что позволяет легко развертывать приложение на разных платформах без дополнительных усилий;
– поддержка современных технологий:
python активно используется в машинном обучении, анализе данных и автоматизации, что может быть полезно для расширения функционала приложения в будущем (например, для прогнозирования платежей или анализа потребления ресурсов);
– большое сообщество и документация:
python имеет активное сообщество разработчиков и обширную документацию, что упрощает поиск решений для любых проблем, возникающих в процессе разработки [1],[7].

Flask (для серверной части)

Flask был выбран в качестве фреймворка для разработки веб-приложения управления коммунальными платежами благодаря своей гибкости, простоте и минималистичному подходу [1,11,15]. В отличие от более тяжеловесных решений, Flask не навязывает жесткую структуру проекта, что позволяет адаптировать архитектуру под конкретные требования системы. Это особенно важно для проекта, где ключевыми задачами являются обработка платежей, интеграция с внешними API банков и платежных систем, а также работа с базой данных [12]. Flask дает свободу в выборе компонентов: для работы с базой данных можно использовать SQLAlchemy или Psycopg2, для аутентификации - Flask-Login или JWT, а для построения API - Flask-RESTful. Такой модульный подход упрощает разработку и позволяет использовать только те инструменты, которые действительно необходимы.

Еще одним преимуществом Flask является его легковесность и быстрый старт проекта, что важно для создания MVP. При этом фреймворк легко масштабируется: при необходимости можно добавить Celery для фоновых задач, Redis для кэширования или Socket.IO для уведомлений в реальном времени. Flask также поддерживает асинхронность (начиная с версии 2.0), что полезно для обработки платежных транзакций и отправки уведомлений без блокировки основного потока.

Frontend: HTML и CSS

HTML и CSS были выбраны в качестве фронтенд-технологий для веб-приложения управления коммунальными платежами по ряду важных причин, которые идеально соответствуют целям и масштабу данного проекта[3]. HTML (HyperText Markup Language) обеспечивает базовую структуру веб-страниц, позволяя организовать и отображать контент, такой как формы для ввода данных, таблицы с платежами и информационные блоки, в то время как CSS (Cascading Style Sheets) отвечает за визуальное оформление, включая макет, цвета, шрифты и адаптивность интерфейса [3],[10],[20]. Их использование обусловлено прежде всего простотой и скоростью разработки: в отличие от сложных JavaScript-фреймворков, HTML и CSS не требуют дополнительных инструментов сборки или глубоких знаний программирования, что позволяет сосредоточиться на реализации ключевых функций бэкенда, таких как обработка платежей и работа с базой данных.

Еще одним ключевым преимуществом является их полная совместимость с Flask, который идеально работает с серверным рендерингом через шаблонизатор Jinja2[2],[14]. Это позволяет динамически встраивать данные из Python-кода (например, списки платежей или баланс пользователя) прямо в HTML-разметку, а также использовать наследование шаблонов для создания единого дизайна всех страниц (например, через базовый layout.html). Формы, созданные на HTML, могут напрямую взаимодействовать с Flask-роутами, отправляя данные методом POST или GET без необходимости настройки дополнительного API, что значительно упрощает процесс

разработки и тестирования. Например, форма для оплаты может быть описана в HTML, а ее обработка — во Flask, что делает код более прозрачным и удобным для отладки.

База данных SQL Lite

SQLite был выбран в качестве базы данных для веб-приложения управления коммунальными платежами благодаря своей простоте, надежности и минимальным требованиям к настройке. Эта встраиваемая СУБД идеально подходит для учебного и небольшого коммерческого проекта, так как не требует отдельного серверного процесса и хранит все данные в одном файле, что значительно упрощает развертывание и перенос между окружениями. SQLite полностью поддерживает транзакции и обеспечивает целостность данных, что критически важно для финансовых операций. Хотя она не предназначена для высоконагруженных систем, её производительности вполне достаточно для MVP и проектов с числом пользователей до нескольких тысяч [3]. Использование SQLite вместе с ORM SQLAlchemy позволяет в будущем, при необходимости, легко перейти на более мощные СУБД, такие как PostgreSQL, практически без изменений в коде приложения [7]. Для дипломного проекта, где важны простота разработки и демонстрация основных функций, SQLite является оптимальным выбором, позволяя сосредоточиться на реализации бизнес-логики, а не на администрировании базы данных.

SQLAlchemy (для работы с базой данных)

SQLAlchemy был выбран в качестве ORM (Object-Relational Mapping) для работы с базой данных SQLite, поскольку он предоставляет удобный и безопасный способ взаимодействия с базой данных через объекты Python [12]. Вместо написания сырых SQL-запросов, которые могут быть подвержены ошибкам и уязвимостям (например, SQL-инъекциям), SQLAlchemy позволяет работать с данными как с обычными объектами Python. Это значительно ускоряет разработку, делает код более читаемым и поддерживаемым. Кроме того, SQLAlchemy поддерживает транзакции, связи между таблицами

(relationships) и миграции, что критически важно для приложения, работающего с финансовыми данными, такими как коммунальные платежи. Еще одно ключевое преимущество — это возможность легко сменить базу данных (например, с SQLite на PostgreSQL) в будущем без переписывания большей части кода, так как SQLAlchemy абстрагирует работу с разными СУБД[6],[18].

Flask-SQLAlchemy

Был выбран как расширение для Flask, которое упрощает интеграцию SQLAlchemy в веб-приложение. Это расширение добавляет удобные функции, такие как автоматическое управление сессиями базы данных в рамках HTTP-запросов, что предотвращает утечки памяти и другие проблемы [6],[12]. Flask-SQLAlchemy также предоставляет готовые решения для создания моделей данных и выполнения запросов, что ускоряет разработку. Например, оно позволяет объявлять модели данных в стиле Flask (через db.Model), что делает код более согласованным и удобным для работы в контексте веб-приложения[14]. Кроме того, Flask-SQLAlchemy интегрируется с другими популярными расширениями Flask, такими как Flask-Migrate (для миграций), что делает его идеальным выбором для проекта на Flask.

Db Browser for SQLite

Был выбран в качестве графического интерфейса для работы с базой данных, так как он позволяет визуально просматривать, редактировать и анализировать данные без необходимости писать SQL-запросы вручную. Это особенно полезно на этапе разработки и отладки, когда нужно быстро проверить содержимое таблиц, выполнить простые запросы или исправить данные. DB Browser поддерживает все основные функции SQLite, включая создание и изменение таблиц, импорт/экспорт данных, а также выполнение сложных SQL-запросов [6]. Его простой и интуитивно понятный интерфейс делает его отличным инструментом для разработчиков, которые хотят сэкономить время на рутинных операциях с базой данных. В контексте

учебного проекта это также помогает лучше понять структуру данных и проверить корректность работы ORM.

Аутентификация и авторизация Session

Session — встроенный объект Flask для хранения данных между запросами был выбран для реализации аутентификации и авторизации в веб-приложении управления коммунальными платежами благодаря своей мощной функциональности, гибкости и интеграции с экосистемой Flask[2],[8]. Session— это один из самых популярных фреймворков для обеспечения безопасности в моём веб-приложении, который предоставляет комплексные решения для защиты веб-приложений от различных угроз, таких как несанкционированный доступ, атаки CSRF, XSS и другие.

Одной из ключевых причин выбора Session является его тесная интеграция с Flask. Session легко настраивается и автоматически конфигурируется в технологии Flask, что позволяет быстро добавить безопасность в приложение без необходимости написания большого количества boilerplate-кода.

Встроенный объект Flask - session для хранения данных между запросами предоставляет мощные механизмы для аутентификации и авторизации. Аутентификация — это процесс проверки подлинности пользователя, например, через логин и пароль, а авторизация — это процесс проверки прав доступа пользователя к определенным ресурсам. Spring Security поддерживает различные методы аутентификации, такие как базовая аутентификация, формальная аутентификация, OAuth2, JWT (JSON Web Tokens) и LDAP. Для веб-приложения управления коммунальными платежами можно использовать декоратор @login_required, который проверяет аутентификацию пользователя в Flask-приложении. Это позволяет обеспечить безопасный доступ к данным и функциям приложения.

Встроенный объект session во Flask предоставляет удобный и безопасный механизм для хранения данных между HTTP-запросами, что особенно важно для реализации функционала аутентификации пользователей

и персонификации работы веб-приложений. Одним из ключевых преимуществ session является его простота использования - разработчику не требуется настраивать дополнительные системы хранения данных или реализовывать сложную логику работы с cookies, так как Flask предоставляет готовый интерфейс, работающий по принципу словаря Python. Это значительно ускоряет процесс разработки и снижает вероятность ошибок.

Важным достоинством session является его безопасность. Все данные, хранящиеся в сессии, автоматически подписываются с использованием секретного ключа приложения (SECRET_KEY), что предотвращает возможность их подделки злоумышленниками[8]. Дополнительно Flask позволяет настроить передачу сессионных cookies только по защищенному HTTPS-соединению, что исключает их перехват при передаче по сети. Встроенная защита от CSRF-атак делает этот механизм надежным решением для большинства веб-приложений.

Гибкость объекта session проявляется в возможности хранения различных типов данных, включая числа, строки, списки и словари, при условии их JSON-совместимости. Разработчик может легко контролировать время жизни сессии через настройку параметра PERMANENT_SESSION_LIFETIME, что особенно полезно для реализации функционала "запомнить меня" в системах аутентификации. При этом данные сессии сохраняются между запросами без необходимости явного сохранения на сервере.

С точки зрения производительности, использование session имеет преимущество перед серверными решениями для хранения сессий, так как не требует обращений к базе данных или внешним хранилищам при каждом запросе. Это особенно важно для приложений с высокой нагрузкой, где критична скорость обработки запросов. Однако важно отметить, что данный механизм оптимально подходит для хранения относительно небольших объемов данных из-за ограничений на размер cookies (обычно до 4 КБ).

Интеграция session с другими компонентами Flask, такими как система сообщений flash или расширение Flask-Login, обеспечивает согласованную работу всех частей приложения и упрощает реализацию сложной бизнес-логики [1],[8],[15]. Это делает session универсальным инструментом для управления состоянием пользователя в веб-приложении.

3.2 Программная реализация приложения

Программная реализация веб-приложения для управления коммунальными платежами является основным этапом, в ходе которого реализуются все компоненты системы: от серверной части до интерфейса пользователя. Это ключевая стадия, где все теоретические концепции и архитектурные решения, разработанные на этапе проектирования, превращаются в работающий продукт. В данном подразделе описываются ключевые аспекты программной реализации, включая структуру проекта, основные компоненты и примеры кода.

Установка необходимых инструментов:

- Visual Studio Code;
- расширения для VS Code;
- SQL Lite;
- python.

Модели (Model) представлены классами SQLAlchemy для работы с базой данных на рисунках 10-12:

```python
# Модели
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(100), unique=True, nullable=False)
    password = db.Column(db.String(200), nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    payments = db.relationship('Payment', backref='payer', lazy=True)
    charges = db.relationship('Charge', backref='user', lazy=True)
```

Рисунок 10 – Модель User

Модель User на рисунке 10 представляет таблицу в базе данных, которая хранит информацию о пользователях.

```python
class Charge(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    type = db.Column(db.String(50), nullable=False)
    amount = db.Column(db.Float, nullable=False)
    month = db.Column(db.Integer, nullable=False)
    year = db.Column(db.Integer, nullable=False)
    is_paid = db.Column(db.Boolean, default=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
```

Рисунок 11 – Модель Charge

Модель Charge на рисунке 11 представляет таблицу, которая хранит информацию о начислениях за коммунальные услуги.

```python
class Payment(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    amount = db.Column(db.Float, nullable=False)
    type = db.Column(db.String(50), nullable=False)
    date = db.Column(db.Date, nullable=False)
    comment = db.Column(db.String(200))
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    method = db.Column(db.String(50))
    status = db.Column(db.String(20))
```

Рисунок 12 – Модель Payment

Модель Payment на рисунке 12 представляет таблицу в базе данных для хранения информации о платежах в вашем веб-приложении для управления коммунальными платежами.Модель создана с использованием Flask-SQLAlchemy, что видно по использованию db.Column и db.Model. Поля marked как nullable=False являются обязательными для заполнения при создании записи о платеже.

```
class PaymentForm(FlaskForm):
    amount = FloatField('Сумма (руб)', validators=[DataRequired()])
    type = SelectField('Тип платежа', choices=[
        ('electricity', 'Электричество'),
        ('water', 'Вода'),
        ('gas', 'Газ'),
        ('rent', 'Квартплата'),
        ('other', 'Другое')
    ], validators=[DataRequired()])
    date = DateField('Дата платежа', default=datetime.now, validators=[DataRequired()])
    comment = StringField('Комментарий')
```

Рисунок 13 – Форма для добавления платежей

```
class ChargeForm(FlaskForm):
    type = SelectField('Тип начисления', choices=[
        ('electricity', 'Электричество'),
        ('water', 'Вода'),
        ('gas', 'Газ'),
        ('rent', 'Квартплата'),
        ('other', 'Другое')
    ], validators=[DataRequired()])
    amount = FloatField('Сумма (руб)', validators=[DataRequired()])
    month = SelectField('Месяц', choices=[(i, str(i)) for i in range(1, 13)], validators=[DataRequired()])
    year = IntegerField('Год', validators=[DataRequired()])
```

Рисунок 14 – Форма для создания начислений

```python
class LoginForm(FlaskForm):
    email = EmailField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Пароль', validators=[DataRequired()])

class RegisterForm(FlaskForm):
    email = EmailField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Пароль', [
        DataRequired(),
        Length(min=6, message='Пароль должен быть не менее 6 символов')
    ])
    confirm = PasswordField('Повторите пароль', validators=[
        DataRequired(),
        EqualTo('password', message='Пароли должны совпадать')
    ])
```

Рисунок 15 – Форма для авторизации пользователя

```python
class ChargePaymentForm(FlaskForm):
    amount = FloatField('Сумма (руб)', validators=[DataRequired()])
    payment_method = RadioField('Способ оплаты', choices=[
        ('СБП', 'СБП (Система быстрых платежей)'),
        ('Банковская карта', 'Банковская карта'),
        ('Электронный кошелек', 'Электронный кошелек')
    ], validators=[DataRequired()], default='СБП')
```

Рисунок 16 – Форма для оплаты начислений

На рисунках 13-16 представлены формы,валидация данных которых реализована через классы FlaskForm.

Контроллеры (Controller) реализованы через Flask-роуты:

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if 'user_id' in session:
        return redirect(url_for('dashboard'))

    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user and check_password_hash(user.password, form.password.data):
            session['user_id'] = user.id
            flash('Вы успешно вошли в систему!', 'success')
            return redirect(url_for('dashboard'))
        flash('Неверный email или пароль', 'danger')

    return render_template('login.html', form=form)
```

Рисунок 17 – Контроллер /login

Задача контроллера на рисунке 17 заключается в аутентификации пользователей.

Логика:

– проверяет email и пароль (сравнивает хэш через check_password_hash);

– при успехе создает сессию (session['user_id']);

– при ошибке выводит сообщение через flash().

```python
@app.route('/register', methods=['GET', 'POST'])
def register():
    if 'user_id' in session:
        return redirect(url_for('dashboard'))

    form = RegisterForm()
    if form.validate_on_submit():
        try:
            hashed_password = generate_password_hash(form.password.data)
            user = User(email=form.email.data, password=hashed_password)
            db.session.add(user)
            db.session.commit()
            flash('Регистрация прошла успешно! Теперь вы можете войти.', 'success')
            return redirect(url_for('login'))
        except IntegrityError:
            db.session.rollback()
            flash('Пользователь с таким email уже существует', 'danger')

    return render_template('register.html', form=form)
```

Рисунок 18 – Контроллер /register

Контроллер на рисунке 18 отвечает за регистрацию новых пользователей.

Логика:

– хэширует пароль (generate_password_hash);

– проверяет уникальность email (ловит IntegrityError);

– создает запись в таблице User.

Использует: RegisterForm с валидацией пароля и email.

```
@app.route('/logout')
def logout():
    session.pop('user_id', None)
    flash('Вы успешно вышли из системы', 'info')
    return redirect(url_for('index'))
```

Рисунок 19 – Контроллер /logout

Контроллер на рисунке 19 отвечает за выход из системы.

Логика: удаляет user_id из сессии (session.pop).

На рисунках 17-19 представлены контроллеры,которые отвечают за авторизацию и регистрацию.

```
@app.route('/payments')
@login_required
def payments():
    user = User.query.get(session['user_id'])
    payments_list = Payment.query.filter_by(user_id=user.id).order_by(Payment.date.desc()).all()
    total = sum(p.amount for p in payments_list)

    return render_template('payments.html',
                           payments=payments_list,
                           total=total)
```

Рисунок 20 – Контроллер /payments

Контроллер на рисунке 20 отвечает за отображение списка платежей пользователя.

Логика:

– запрашивает платежи из Payment для текущего пользователя;

– считает общую сумму (sum(p.amount for p in payments));

– передает данные в шаблон payments.html.

```python
@app.route('/payment/new', methods=['GET', 'POST'])
@login_required
def new_payment():
    form = PaymentForm()
    if form.validate_on_submit():
        payment = Payment(
            amount=form.amount.data,
            type=form.type.data,
            date=form.date.data,
            comment=form.comment.data,
            user_id=session['user_id'],
            method="Ручной ввод",
            status="Завершен"
        )
        db.session.add(payment)
        db.session.commit()
        flash('Платёж успешно добавлен!', 'success')
        return redirect(url_for('payments'))

    return render_template('payment.html', form=form)
```

Рисунок 21 – Контроллер /payments/new

На рисунке 21 представлен контроллер,задача которого заключается в добавлении нового платежа.

Логика:

– валидирует данные через PaymentForm;

– создает запись с типом "Ручной ввод" и статусом "Завершен";

– привязывает платеж к текущему пользователю (user_id=session['user_id']).

```python
@app.route('/payment/<int:id>/edit', methods=['GET', 'POST'])
@login_required
def edit_payment(id):
    payment = Payment.query.get_or_404(id)
    if payment.user_id != session['user_id']:
        flash('У вас нет прав для редактирования этого платежа', 'danger')
        return redirect(url_for('payments'))

    form = PaymentForm(obj=payment)
    if form.validate_on_submit():
        form.populate_obj(payment)
        db.session.commit()
        flash('Платёж успешно обновлён!', 'success')
        return redirect(url_for('payments'))

    return render_template('payment.html', form=form, payment=payment)
```

Рисунок 22 – Контроллер /payments/<int:id>/edit

Контроллер на рисунке 22 отвечает за редактирование существующего платежа.

Логика:

– проверяет, принадлежит ли платеж текущему пользователю;

– заполняет форму PaymentForm данными из БД (form = PaymentForm(obj=payment));

– обновляет запись через form.populate_obj(payment).

```python
@app.route('/payment/<int:id>/delete', methods=['POST'])
@login_required
def delete_payment(id):
    payment = Payment.query.get_or_404(id)
    if payment.user_id != session['user_id']:
        flash('У вас нет прав для удаления этого платежа', 'danger')
        return redirect(url_for('payments'))

    db.session.delete(payment)
    db.session.commit()
    flash('Платёж успешно удалён!', 'success')
    return redirect(url_for('payments'))
```

Рисунок 23 – Контроллер /payments/<int:id>/delete

Задача контроллера,который представлен на рисунке 23,заключается в удалении платежа.

На рисунках 20-23 представлены контроллеры,которые отображают работу с платежами.

Логика:

– проверяет права доступа;

– удаляет запись через db.session.delete(payment).

```python
@app.route('/charges')
@login_required
def charges():
    user = User.query.get(session['user_id'])
    charges_list = Charge.query.filter_by(user_id=user.id).order_by(Charge.year.desc(), Charge.month.desc()).all()

    return render_template('charges.html', charges=charges_list)
```

Рисунок 24 – Контроллер /charges

Назначение: просмотр списка начислений.

Логика:

– фильтрует начисления по user_id и сортирует по дате;

– передает данные в шаблон charges.html.

```python
@app.route('/charge/new', methods=['GET', 'POST'])
@login_required
def new_charge():
    form = ChargeForm()
    if form.validate_on_submit():
        charge = Charge(
            type=form.type.data,
            amount=form.amount.data,
            month=form.month.data,
            year=form.year.data,
            user_id=session['user_id']
        )
        db.session.add(charge)
        db.session.commit()
        flash('Начисление успешно добавлено!', 'success')
        return redirect(url_for('charges'))

    return render_template('new_charge.html', form=form)
```

Рисунок 25 – Контроллер /charge/new

Назначение: создание нового начисления.

Логика:

– валидирует ChargeForm (тип услуги, сумма, месяц/год);

– устанавливает is_paid=False по умолчанию.

```
@app.route('/charge/<int:id>/mark_paid', methods=['POST'])
@login_required
def mark_charge_paid(id):
    charge = Charge.query.get_or_404(id)
    if charge.user_id != session['user_id']:
        flash('У вас нет прав для изменения этого начисления', 'danger')
        return redirect(url_for('charges'))

    charge.is_paid = True
    db.session.commit()
    flash('Начисление отмечено как оплаченное', 'success')
    return redirect(url_for('charges'))
```

Рисунок 26 – Контроллер /charge/<int:id>/mark_paid

Назначение: отметка начисления как оплаченного.

Логика:

– меняет статус is_paid на True;

– не создает автоматически платеж (это делает отдельный метод /process_charge_payment).

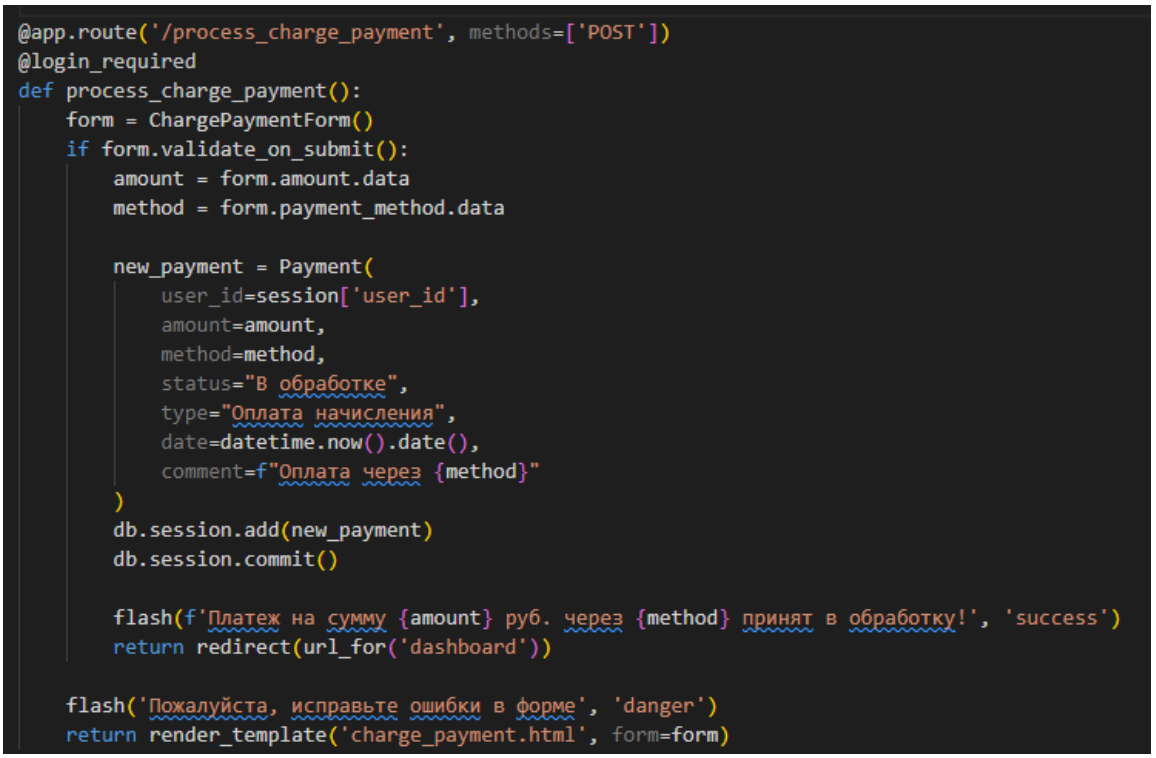
На рисунках 24-26 представлены контроллеры,которые отвечают за управление начислениями.

```
@app.route('/charge_payment')
@login_required
def charge_payment():
    form = ChargePaymentForm()
    return render_template('charge_payment.html', form=form)
```

Рисунок 27 – Контроллер /charge_payment

Назначение: страница выбора способа оплаты.

Логика: отображает форму ChargePaymentForm (сумма, метод оплаты).



```python
@app.route('/process_charge_payment', methods=['POST'])
@login_required
def process_charge_payment():
    form = ChargePaymentForm()
    if form.validate_on_submit():
        amount = form.amount.data
        method = form.payment_method.data

        new_payment = Payment(
            user_id=session['user_id'],
            amount=amount,
            method=method,
            status="В обработке",
            type="Оплата начисления",
            date=datetime.now().date(),
            comment=f"Оплата через {method}"
        )
        db.session.add(new_payment)
        db.session.commit()

        flash(f'Платеж на сумму {amount} руб. через {method} принят в обработку!', 'success')
        return redirect(url_for('dashboard'))

    flash('Пожалуйста, исправьте ошибки в форме', 'danger')
    return render_template('charge_payment.html', form=form)
```

Рисунок 28 – Контроллер /process_charge_payment

Назначение: обработка оплаты.

Логика:

– создает запись в Payment со статусом "В обработке";

– добавляет комментарий с методом оплаты (например, "СБП").

На рисунках 27-28 представлены контроллеры,которые отвечают за оплату начислений.

3.3 Тестирование

Тестирование является критически важным этапом разработки веб-приложения для управления коммунальными платежами. Основная цель тестирования — верификация корректности работы всех функций системы, выявление потенциальных ошибок и обеспечение надежности приложения в

эксплуатационных условиях [8]. Для комплексной проверки системы было проведено функциональное тестирование, позволяющее оценить взаимодействие компонентов приложения и соответствие заявленным требованиям. Результаты тестирования представлены в Таблице 3.

Таблица 3 – Результаты тестирования функциональных модулей приложения

№ теста	Наименование теста	Ожидаемый результат	Фактический результат	Статус
1	Регистрация нового пользователя	Успешная регистрация,переход на главную страницу	Пользователь успешно зарегистрирован	Успешно
2	Ввод некорректного email при регистрации	Отображается сообщение об ошибке валидации	Ошибка отображается корректно	Успешно
3	Успешная оплата начисления	Статус начисления изменяется на оплачено,создаётся запись в истории платежей	Платёж проведён,статус изменён	Успешно
4	Добавление начисления	Появление начисления в ЛК жильца	Начисление отображается в списке	Успешно
5	Просмотр истории платежей	Отображение полной истории платежей	Список отображён с корректными данными	Успешно

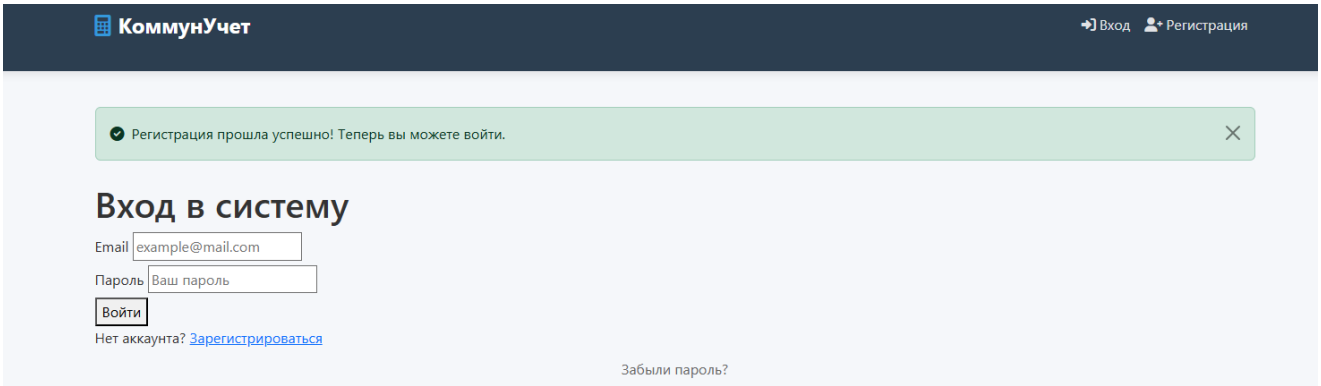


Рисунок 30 – Тест-кейс 1 Успешная регистрация

Регистрация нового пользователя осуществляется через форму, в которой пользователь указывает логин, email и пароль. После отправки формы

данные сохраняются в базе, а пользователь перенаправляется на главную страницу. Поведение системы соответствует ожидаемому

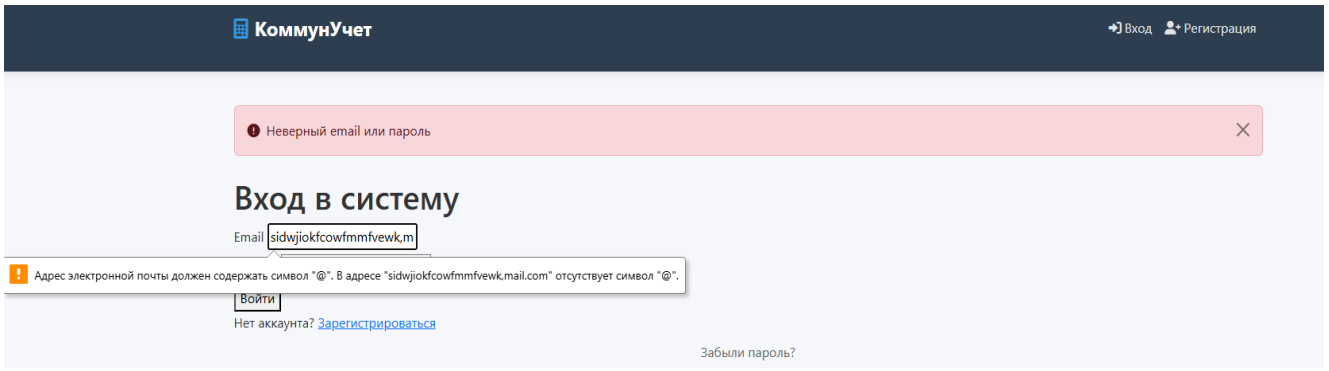


Рисунок 31 – Тест-кейс 2 Ввод некорректного email при регистрации

При вводе email без символа "@" система должна отклонить форму. В тесте проверено наличие валидационного сообщения. Ранее валидация отсутствовала — была добавлена через wtforms.validators.Email.

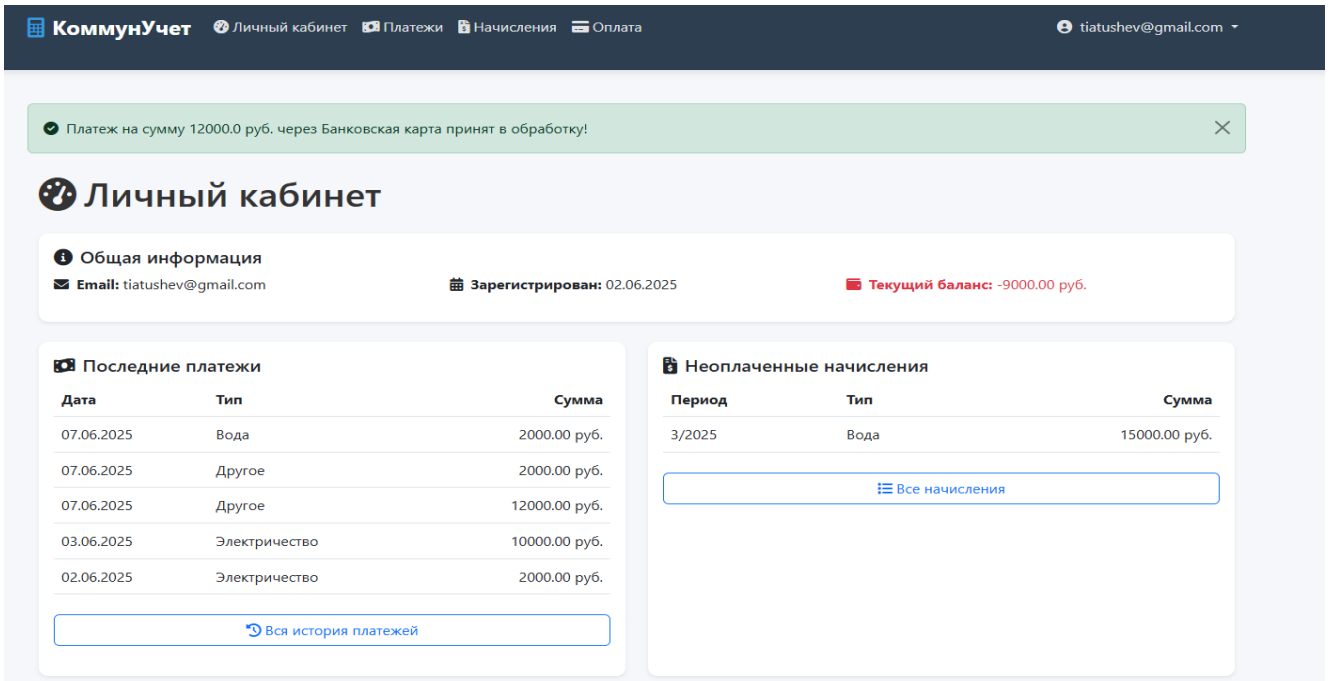


Рисунок 32 – Тест-кейс 3 Успешная оплата начисления

Оплата начисления производится через выбор услуги, подтверждение суммы и сохранение платежа. После оплаты начисление автоматически получает статус "оплачено", а в таблицу платежей добавляется новая запись.

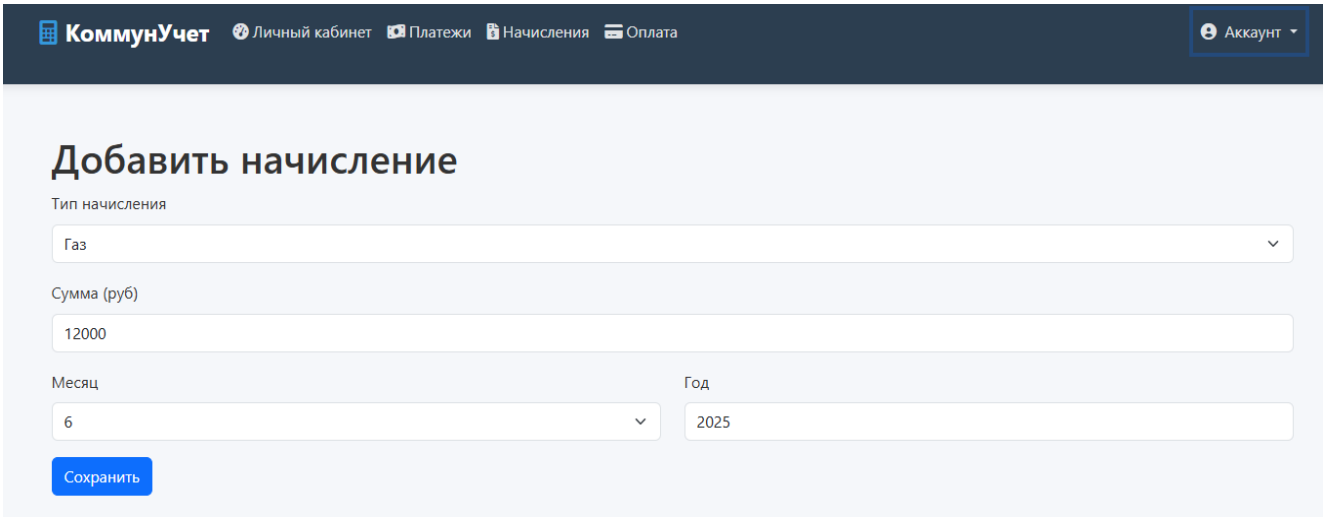


Рисунок 33 – Тест-кейс 4 Добавление начисления

Добавление начисления осуществляется администратором. Система сохраняет тип услуги, сумму и дату. При этом сумма проверяется на положительное значение, исключая ошибки некорректного ввода.

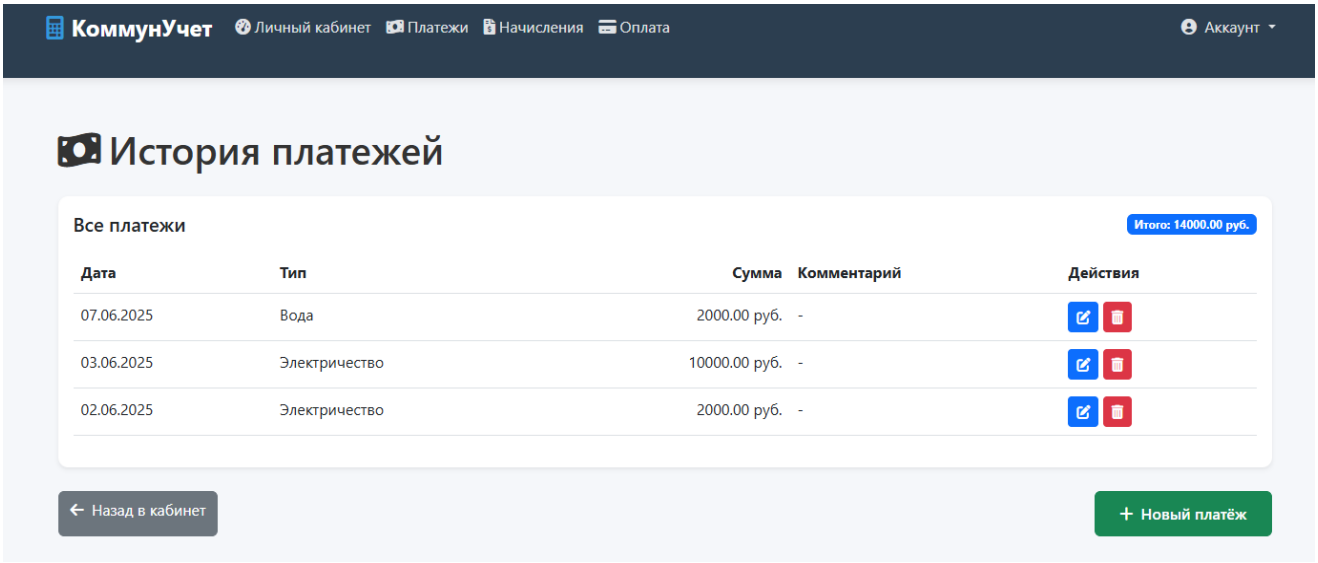


Рисунок 34 – Тест-кейс 5 Просмотр истории платежей

История платежей представлена в виде таблицы в личном кабинете пользователя. Тест проверял корректность отображения всех совершённых транзакций, соответствие дат и сумм.

В данном разделе представлены типовые ошибки, обнаруженные в процессе разработки, а также способы их устранения. Примеры включают проблемы с валидацией данных, отображением уведомлений и другие аспекты, влияющие на функциональность и пользовательский опыт[8].

Ошибка 1 – Отсутствие валидации email

Описание: форма регистрации принимала адреса без символа "@".

```
email = StringField("Email")
```

Рисунок 35 – Отсутствие валидации email (до)

```
from wtforms.validators import Email
email = StringField("Email", validators=[Email(message="Неверный email")])
```

Рисунок 36 – Валидация email с сообщением об ошибке (после)

Ошибка 2 – Неотображаемое уведомление об ошибке входа

Описание: при неправильном вводе логина/пароля уведомление не отображалось.

Решение: добавлена flash-сообщение во Flask-контроллер:

if not user or not check_password_hash(user.password, form.password.data):

flash("Неверный логин или пароль", "danger")

```
if not user or not check_password_hash(user.password, form.password.data):
    return redirect(url_for("login"))
```

Рисунок 37 – Ошибка авторизации без уведомления (до)

```
if not user or not check_password_hash(user.password, form.password.data):
    flash("Неверный логин или пароль", "danger")
    return redirect(url_for("login"))
```

Рисунок 38 – Корректное уведомление при ошибке входа (после)

Ошибка 3 – Отрицательная сумма платежа

Описание: пользователь мог ввести отрицательное значение суммы.

Решение: добавлен валидатор:

from wtforms.validators import NumberRange

amount = FloatField("Сумма", validators=[NumberRange(min=0.01, message="Сумма должна быть положительной")])

```
amount = FloatField("Сумма")
```

Рисунок 39 – Ввод отрицательной суммы (до)

```
from wtforms.validators import NumberRange
amount = FloatField("Сумма", validators=[NumberRange(min=0.01, message="Сумма должна
```

Рисунок 40 – Ошибка при попытке ввести отрицательную суммы (после)

В процессе тестирования были выявлены и устранены некоторые незначительные ошибки, связанные с валидацией входных данных и отображением уведомлений. Проведенные тесты подтвердили, что веб-приложение полностью соответствует функциональным требованиям и предоставляет пользователям удобный инструмент для управления коммунальными платежами[8]. Все ключевые сценарии, включая регистрацию, внесение платежей, контроль начислений и формирование отчетов, работают корректно.

Интерфейс приложения прошел проверку на удобство использования с участием фокус-группы, включающей пользователей разных возрастных категорий. Результаты показали, что система интуитивно понятна и не требует

специального обучения. Была отмечена четкость отображения информации о платежах и простота навигации между разделами.

Разработанное решение демонстрирует хороший потенциал для масштабирования - архитектура позволяет легко добавлять новые модули и функциональные возможности.

Данные результаты подтверждают, что разработанное веб-приложение является надежным решением для автоматизации учета коммунальных платежей.

В разделе описан процесс реализации приложения с использованием технологий Python, Flask, SQLite и SQLAlchemy. Приведены примеры кода, формы интерфейса и контроллеры, обеспечивающие основные функции системы. Тестирование подтвердило корректность работы всех модулей, включая регистрацию, оплату и формирование отчетов. Устранение выявленных ошибок, таких как валидация email и отображение уведомлений, повысило надежность и удобство использования приложения.

Заключение

В рамках выполнения выпускной квалификационной работы была разработана и реализована информационная система — веб-приложение для управления коммунальными платежами, предназначенное для автоматизации процессов учёта, начислений и оплаты услуг ЖКХ. Проведённый анализ предметной области показал высокую актуальность цифровых решений в жилищно-коммунальном секторе[4]. Существующие программные продукты обладают либо ограниченным функционалом, либо высокой стоимостью обслуживания, что делает предложенное решение значимым с точки зрения доступности и адаптируемости под конкретные нужды.

В первой главе была подробно рассмотрена структура деятельности компании «Квартплата 24», выполнен анализ её бизнес-процессов, построены диаграммы IDEF0 и вариантов использования. Это позволило выделить ключевые функции и проблемы, которые должны быть решены при помощи информационной системы. Также сформулированы функциональные и нефункциональные требования, соответствующие реальным условиям эксплуатации.

Во второй главе была проведена разработка архитектуры приложения с использованием многослойной модели. Подробно описаны уровни приложения: презентационный, бизнес-логики и доступа к данным. Были построены диаграммы компонентов, прецедентов, последовательностей, а также логическая модель базы данных (ER-диаграмма). Такой подход обеспечил чёткость в проектировании и позволил формализовать структуру системы до начала её непосредственной реализации.

На этапе реализации, представленном в третьем разделе, выбран стек технологий, включающий Python, Flask, SQLite и SQLAlchemy. Была осуществлена реализация основных модулей: регистрации, авторизации, создания начислений, их оплаты и просмотра истории платежей. Контроллеры реализованы с соблюдением принципов безопасности и архитектурной

целостности. Приложение имеет интуитивно понятный интерфейс, соответствующий современным требованиям UX-дизайна[3].

Особое внимание было уделено тестированию, результаты которого отражены в соответствующем подразделе. Проведены ручные и функциональные тесты ключевых пользовательских сценариев, оформлены скриншоты и фрагменты кода до и после исправлений[5]. Были выявлены и устранены ошибки, связанные с валидацией данных и корректным отображением сообщений для пользователей[5].

Разработанное приложение позволяет пользователям отслеживать начисления, оплачивать услуги онлайн, просматривать историю платежей, а управляющим организациям — управлять начислениями и тарифами. Оно способствует сокращению времени обслуживания, снижению числа ошибок в расчётах, улучшению взаимодействия между пользователями и поставщиками услуг[2],[4],[7],[11].

Список используемой литературы и используемых источников

1. Бизли, Д. Python. Подробный справочник. — СПб.: Символ-Плюс, 2020. — 864 с.
2. Гринберг, М. Flask. Разработка веб-приложений на Python. — СПб.: Питер, 2022. — 352 с.
3. Жуков, Р.Н. UX/UI дизайн интерфейсов. — СПб.: БХВ-Петербург, 2020. — 240 с.
4. Иванов, А.В. Цифровизация жилищно-коммунального хозяйства. — М.: Инфра-М, 2021. — 256 с.
5. Козлов, Д.Ю. Тестирование программного обеспечения. — СПб.: Питер, 2021. — 288 с.
6. Майерс, Дж. SQLAlchemy. Гибкая работа с базами данных на Python. — М.: ДМК Пресс, 2020. — 288 с.
7. Маттес, Э. Изучаем Python. Программирование игр, визуализация данных, веб-приложения. — СПб.: Питер, 2021. — 704 с.
8. Петров, К.С. Безопасность веб-приложений. — СПб.: БХВ-Петербург, 2022. — 416 с.
9. Смирнов, В.А. Проектирование баз данных. — М.: ДМК Пресс, 2019. — 320 с.
10. Федоров, М.Л. Современные фреймворки для веб-разработки. — М.: Альфа-книга, 2022. — 352 с.
11. Brown M. Web Development with Python and Flask. — O'Reilly, 2022. — 398 p.
12. Chen L. Modern Payment Systems. — Wiley, 2023. — 356 p.
13. Davis K. User Experience Design. — Apress, 2022. — 284 p.
14. Flask Documentation. — [Электронный ресурс]. — Режим доступа: https://flask.palletsprojects.com/en/2.3.x/ (дата обращения: 12.05.2025).

15. Johnson S. Database Design for Mere Mortals. — Addison-Wesley, 2020. — 672 p.
16. Python Documentation. — [Электронный ресурс]. — Режим доступа: https://docs.python.org/3/ (дата обращения: 12.05.2025).
17. Real Python: Flask Tutorials. — [Электронный ресурс]. — Режим доступа: https://realpython.com/tutorials/flask/ (дата обращения: 12.05.2025).
18. SQLAlchemy Documentation. — [Электронный ресурс]. — Режим доступа: https://www.sqlalchemy.org/ (дата обращения: 12.05.2025).
19. SQLite Documentation. — [Электронный ресурс]. — Режим доступа: https://www.sqlite.org/docs.html (дата обращения: 12.05.2025).
20. Wilson G. Software Engineering for Web Applications. — Springer, 2021. — 412 p.