Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

Дисциплина: Системы управления базами данных

Пояснительная записка

к курсовому проекту

на тему

**база данных общежития**

БГУИР КП 1-53 01 02 418 ПЗ

Студент гр. 120604 Михейка Н.М.

Руководитель Трофимович А.Ф.

Минск 2023

Учреждение образования

«Белорусский государственный университет информатики   
и радиоэлектроники»

Факультет информационных технологий и управления

УТВЕРЖДАЮ

Заведующий кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023г.

ЗАДАНИЕ

по курсовому проектированию

Студенту *Михейка Никите Максимовичу–––––––––––––––––––*

1. Тема проекта *База данных общежития*

2. Срок сдачи студентом законченного проекта2 *декабря 2023г.––––*

3. Исходные данные к проекту: *Разработка и реализация базы данных общежития для хранения информации о пользователях, сотрудниках, жильцах, журнале ОПТ, журнале дежурств, инвентаря, комнат и выговорах. Создание функционала для просмотра, редактирования и удаления хранящихся в БД данных.* *Создание соответствующего интерфейса. Реализация механизма для определения статуса заселения на основе выбранных параметров.* *Для пользователей предоставляется разные уровни доступа, а также регистрация и авторизация по логину и паролю. У администратора имеется возможность блокировать пользователей и просматривать историю. Доступ к базе данных осуществляется через веб-приложение. Система валидирует все входные данные на стороне сервера, а также не позволяет пользователям выполнять операции, доступные администраторам. Система позволяет выполнить импорт данных в БД или их экспорт.*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке):

*Введение*

1. *Обзор исследуемой области*
2. *Проектирование системы*
3. *Программная реализация системы*

*Заключение*

*Список использованных источников*

*Приложение*

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*1. ER-диаграмма в нотации Питера Чена (ПД, формат А3)*

6. Консультант по проекту (с обозначением разделов проекта) *Трофимович А.Ф.*

7. Дата выдачи задания –*8*–*сентября 2023 г.––––––––––––––––––––––  –*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

*раздел 1 к 15.10 – 15%; –– –––––––––––––– –––––––––––––––– –*

*раздел 2 к 15.11 – 50%;–––––––––––––––– ––––––––––––––––––  ––––*

*раздел 3 к 15.12 – 15%;––––––––––––––––––––– –––––––––––––––––*

*оформление пояснительной записки и графического   
материала к 20.12 – 20 %*

*Защита курсового проекта с 08.12 по 21.12.2023 г.*

Руководитель*– –––––––­­­­­----––– А.Ф.Трофимович*

(подпись)

Задание принял к исполнению *–––––––\_\_\_\_\_\_\_\_\_\_\_–– Н.М.Михейка*

(дата и подпись студента)

**Содержание**

[1 Обзор исследуемой области 6](#_Toc154029053)

[1.1 Описание предметной области 6](#_Toc154029054)

[1.2 Обзор аналогов 7](#_Toc154029055)

[1.3 Постановка задачи 10](#_Toc154029056)

[2 Проектирование базы данных общежития 12](#_Toc154029057)

[2.1 Проектирование базы данных 12](#_Toc154029058)

[2.2 Проектирование приложения 18](#_Toc154029059)

[3 Разработка базы данных общежития 23](#_Toc154029060)

[3.1 Выбор средств реализации 23](#_Toc154029061)

[3.2 Реализация базы данных 27](#_Toc154029062)

[3.3 Программная реализация приложения 30](#_Toc154029063)

[3.4 Инструкция пользователя 32](#_Toc154029064)

[Заключение 38](#_Toc154029065)

[Список использованных источников 39](#_Toc154029066)

[Приложение А 40](#_Toc154029067)

[Ведомость курсового проекта 45](#_Toc154029068)

**ВВЕДЕНИЕ**

В современном образовательном процессе управление информацией играет ключевую роль, особенно в контексте организации студенческой жизни. Студенческие общежития являются неотъемлемой частью университетской инфраструктуры, обеспечивая студентам комфортные условия проживания и способствуя формированию общности. Целью данного курсового проекта является разработка и реализация базы данных для управления студенческим общежитием. Проект направлен на оптимизацию процессов администрирования и повышение эффективности управления жизнью общежития, что, в свою очередь, способствует улучшению общего студенческого опыта.

Актуальность данной работы обусловлена необходимостью внедрения современных технологий в управление студенческим жильем, что позволит повысить прозрачность процессов, улучшить обслуживание студентов и обеспечить безопасность проживания. Кроме того, в условиях постоянного изменения потребностей студенческой общественности важно создать гибкую и масштабируемую систему, способную адаптироваться к различным сценариям использования.

Разрабатываемая база данных содержит в себе как функциональную часть, совершающую основную работу, так и пользовательский интерфейс, позволяющий эффективно пользоваться веб-приложением.

В результате выполнения проекта ожидается получить надежную базу данных, которая позволит эффективно управлять студенческим общежитием, сократить время и усилия, затрачиваемые на ручное ведение документации, а также повысить качество обслуживания студентов, проживающих в общежитии.

Задачами курсового проекта являются:

– освоение, углубление и обобщение знаний, полученных во время обучения;

– изучение подходов к проектированию баз данных, а также к созданию готового продукта;

– формирование умений использовать справочную литературу, нормативную, нормативно-техническую документацию;

– приобретение навыков по оформлению текстовой и графической документации согласно требованиям государственных стандартов и стандарта предприятия Дипломные проекты (работы).

# **ОБЗОР ИССЛЕДУЕМОЙ ОБЛАСТИ**

## **Описание предметной области**

Предметной областью данного курсового проекта является управление информацией о студенческом общежитии. Общежития – это неотъемлемая часть студенческой жизни, предоставляя студентам не только место для проживания, но и создавая уникальное социокультурное пространство. Управление общежитием включает в себя решение разнообразных задач, таких как ведение учета проживающих, обеспечение безопасности, контроль доступа, а также организация быта и удобства для студентов.

История управления студенческими общежитиями тесно связана с развитием систем образования. В начале своего существования общежития представляли собой скромные обители, предоставляющие студентам простое жилье. С течением времени, с увеличением числа студентов и изменением требований, общежития стали становиться более сложными структурами, где кроме проживания обеспечивается инфраструктура для активного студенческого общения и обучения.

В течение многих лет управление общежитием осуществлялось вручную с использованием традиционных методов. Регистрация и учет проживающих осуществлялись на бумажных носителях, а контроль доступа и безопасность обеспечивались при помощи физических замков и ключей. Администрация общежития взаимодействовала с жильцами напрямую, решая текущие вопросы вручную.

Коммуникация и информационный обмен в общежитии строились на личных встречах и объявлениях на информационных досках. Организация событий и контроль за состоянием помещений также осуществлялись традиционными методами, не требующими применения современных технологий.

Такой подход, хотя и обеспечивал базовые функции, часто сопровождался рядом трудностей, таких как неэффективное использование времени, потеря данных, а также ограниченные возможности для обеспечения безопасности и комфорта проживания. В условиях современного быстрого развития технологий становится ясной необходимость внедрения автоматизированных систем управления, чтобы оптимизировать процессы и повысить качество обслуживания студентов в общежитии.

## **Обзор аналогов**

Для более полного понимания проблематики и эффективного проектирования базы данных студенческого общежития необходимо провести анализ существующих аналогичных систем. Рассмотрение аналогов представляет собой важный этап, поскольку позволяет выявить успешные практики, учесть ошибки и инновационные подходы, применяемые в данной области.

В современном мире существует ряд информационных систем, предназначенных для управления общежитиями и обеспечения комфортного проживания студентов. Примерами подобных решений могут служить системы электронного учета проживания, системы безопасности, а также программные продукты, обеспечивающие взаимодействие между администрацией общежития, персоналом и студентами.

Одним из успешных аналогов может быть система, которая эффективно решает задачи учета проживающих, предоставления отчетности, автоматизации процессов выдачи и учета ключей, контроля доступа и обеспечения общего уровня безопасности. Важным аспектом является также удобство использования для конечных пользователей – студентов и администрации.

Проведение сравнительного анализа позволит выявить особенности успешных решений, а также определить те аспекты, которые следует учесть в процессе проектирования базы данных для студенческого общежития. Такой подход обеспечит более глубокое понимание требований к проекту и способствует созданию более совершенной и адаптивной системы управления студенческим жильем.

Одним из первых результатов на поисковой странице располагается сайт *www.hse.ru.* На главной странице пользователя встречает довольно современный и минималистичный интерфейс (рисунок 1.1). Акцент внимания приходится на навигационную панель – одну из самых важных составляющих любого сайта или веб-приложения. Из неё пользователь может получить доступ к другим страницам сайта: профиль, поиск, учебная информация, переключатель языка сайта и версия для слабовидящих.

Не меньше внимания приходится на кнопку «Онлайн-оплата проживания», при нажатии на которую происходит перенаправление к выбору способа оплаты (рисунок 1.2). Данная опция очень сильно упрощает процесс оплаты проживания и исключает физическое взаимодействие.

На этом функционал данного сайта заканчивается, что негативно сказывается на комфорте проживания из-за отсутствия остальных требуемых возможностей.

Изображение выглядит как человек, Человеческое лицо, одежда, в помещении

Автоматически созданное описание

Рисунок 1.1 – Интерфейс сайта *www.hse.ru*

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.2 – Оплата проживания на сайте *www.hse.ru*

Следующим рассматриваемым аналогом является сайт ИИС, функционал которого ориентирован на организацию учебного процесса. Он также включает возможности взаимодействия с системой общежития. Для доступа к сайту требуется пройти авторизацию с помощью номера студенческого билета, что исключает несанкционированный доступ (рисунок 1.3). Стоит отметить возможность восстановления пароля.

После успешной авторизации пользователю предоставляется доступ ко всем страницам сайта. На рисунке 1.4 проиллюстрирован интерфейс для организации процесса проживания в общежитии. В общих чертах интерфейс не является перегруженным и интуитивно понятен благодаря иконкам в контекстном меню. Имеется возможность подать заявку на заселение, отслеживать её статус с течением времени и просматривать предоставленные льготы. Такого набора функций вполне достаточно для автоматизации процесса заселения, но недостаточно для последующего проживания.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 1.3 – Страница авторизации ИИС

Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Автоматически созданное описание

Рисунок 1.4 – Основной интерфейс ИИС

Благодаря рассмотренным аналогам, их плюсам и минусам получено понимание наиболее востребованных функций проектируемой системы. Полученные знания возможно использовать в будущих проектах, а также учитывать опыт предшественников.

## **Постановка задачи**

Разработать систему студенческого общежития. Пользователь входит в аккаунт или регистрируется. Новый пользователь заполняет контактную информацию о себе. Пользователь открывает представление базы данных. Пользователь выбирает таблицу и получает визуальное отображение. Во время просмотра таблицы пользователь может сортировать, искать и сохранять данные из таблицы. Пользователь в любой момент может выйти из аккаунта или при наличии доступа администрировать систему. Администратор имеет право заблокировать любого пользователя, просматривать историю запросов. Администратор управляет таблицами, может блокировать и разблокировать пользователя, регистрировать сотрудников в качестве новых администраторов, добавлять новые данные и изменять старые данные.

Требования к БД:

* информация о предметной области должна храниться в БД;
* если данные в базе хранятся на кириллице, рекомендуется применять кодировку *UTF*-8;
* при проектировании с БД рекомендуется использовать не менее 6 таблиц;
* реализовать защиту от *SQL injection*;
* при проектировании БД использовать правила нормализации таблиц;
* при реализации БД разрешается использовать любое СУБД.

Базовые требования к приложению:

* приложение должно корректно обрабатывать возникающие исключительные ситуации;
* классы и другие сущности приложения должны быть грамотно структурированы по пакетам и иметь отражающую их функциональность название;
* при реализации бизнес-логики приложения нужно использовать шаблоны проектирования;
* для хранения пользовательской информации между запросами использовать сессию;
* при реализации пользовательского интерфейса разрешается использовать любые *front-end* технологии;
* реализовать защиту от *cross site scripting (XSS)* и от повторного выполнения запроса нажатием *F5*;
* валидацию входных данных производить на сервере.

Минимальные требования к функциональности приложения:

* авторизация (*sign in*) и выход (*sign out*) в/из системы;
* регистрация пользователя и/или добавление объекта предметной области системы;
* просмотр информации;
* удаление информации;
* добавление и модификация информации;
* работа с файлами (*xml, json*).

Системные требования:

* операционная система *Windows* 7, 8, 10, 11;
* процессор: 1 гигагерц (ГГц) или более быстрый процессор или *SoC*;
* оперативная память: 1 гигабайт (ГБ) для 32-разрядных или 2 ГБ для 64-разрядных;
* место на жестком диске: 16 ГБ для 32-разрядной ОС или 20 ГБ для 64-разрядной ОС;
* монитор: 800 x 600;
* сеть: не менее 0,5 Мбит/с;
* реляционная база данных;
* объектно-ориентированный язык программирования;
* интерфейс.

Для разработки веб-приложения можно использовать следующие технологии:

* языки программирования, разметки, стили (*HTML, CSS, JavaScript*);
* библиотеки, фреймворки (*React, NodeJS, Express*);
* СУБД (*PostgreSQL*).

# **2 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ ОБЩЕЖИТИЯ**

## **Проектирование базы данных**

В данном разделе представлено проектирование базы данных для студенческого общежития. База данных является фундаментальным элементом любой информационной системы, и эффективное её проектирование является ключевым для обеспечения надёжности, гибкости и эффективности работы системы.

Целью этого раздела является представление описания сущностей, атрибутов и доменов базы данных, а также обсуждение процесса нормализации, который поможет устранить избыточность и аномалии данных.

Для начала рассмотрим сущности, их атрибуты и домены, которые будут присутствовать в нашей базе данных. Атрибуты являются характеристиками сущности, которые хранятся в базе данных. Домен определяет множество значений, которые может принимать атрибут.

1. Сущность "Пользователь":

Атрибуты:

* Идентификатор: представляет идентификатор пользователя.
* Логин: представляет адрес электронной почты пользователя.
* Пароль: представляет пароль для авторизации пользователя.
* Роль: представляет роль пользователя.

Домены:

* Идентификатор пользователя представляет собой уникальное число.
* Электронная почта имеет домен типа строка с проверкой на соответствие формату электронной почты.
* Пароль может иметь домен типа строка и хранится в зашифрованном виде для обеспечения безопасности.
* Роль имеет домен типа строка (по умолчанию значение «*default*» – пользователь, «*admin*» - администратор, «*ban*» - заблокированный пользователь)

1. Сущность "Сотрудник":

Атрибуты:

* Идентификатор сотрудника: представляет идентификатор сотрудника.
* ФИО: представляет фамилию, имя, отчество сотрудника.
* Должность: представляет должность, занимаемую сотрудником в общежитии.
* Телефон: представляет номер мобильного телефона.
* Адрес: представляет город проживания сотрудника.
* Зарплата: представляет заработную плату сотрудника.
* Идентификатор пользователя: представляет идентификатор пользователя, соответствующего сотруднику.

Домены:

* Идентификатор сотрудника представляет уникальное число.
* ФИО имеет строковый тип.
* Должность имеет строковый тип.
* Телефон имеет строковый тип.
* Адрес имеет строковый тип.
* Зарплата имеет числовой тип.
* Идентификатор пользователя представляет уникальное число.

1. Сущность "Жилец":

Атрибуты:

* Идентификатор жильца: представляет идентификатор жильца.
* ФИО: представляет фамилию, имя, отчество сотрудника.
* Должность: представляет должность, занимаемую сотрудником в общежитии.
* Телефон: представляет номер мобильного телефона.
* Регистрация: представляет город проживания жильца вне общежития.
* Идентификатор заявления: представляет идентификатор заявление, которое подал жилец для заселения.
* Номер комнаты: представляет номер комнаты, в котором проживает жилец.
* Идентификатор дежурств: представляет идентификатор отработанных дежурств.
* Идентификатор ОПТ: представляет идентификатор отработанных часов дежурств.

Домены:

* Идентификатор жильца представляет уникальное число.
* ФИО имеет строковый тип.
* Должность имеет строковый тип.
* Телефон имеет строковый тип.
* Регистрация имеет строковый тип.
* Зарплата имеет числовой тип.
* Идентификатор заявления представляет уникальное число.
* Идентификатор дежурств представляет уникальное число.
* Идентификатор ОПТ представляет уникальное число.

1. Сущность "Комната":

Атрибуты:

* Идентификатор комнаты: представляет идентификатор комнаты и её номер.
* Статус: представляет статус, в котором находится комната.
* Количество жильцов: представляет количество жильцов, проживающих в комнате.
* Количество мебели: представляет количество мебели, находящейся в комнате.

Домены:

* Идентификатор комнаты представляет уникальное число.
* Статус имеет строковый тип.
* Количество жильцов имеет числовой тип, значение по умолчанию 0.
* Количество мебели имеет числовой тип, значение по умолчанию 0.

1. Сущность "Заявление на заселение":

Атрибуты:

* Идентификатор заявления: представляет идентификатор заявления на заселение.
* Статус: представляет статус, в котором находится заявление.
* Дата подачи: представляет дату подачи и принятия заявления в систему.
* Дата заселения: представляет дату подтверждения статуса заявления «К заселению».

Домены:

* Идентификатор заявления представляет уникальное число.
* Статус имеет строковый тип.
* Дата подачи имеет тип дата.
* Дата заселения имеет тип дата.

1. Сущность "Заявка на ремонт":

Атрибуты:

* Идентификатор заявки: представляет идентификатор заявки на произведение ремонтных работ.
* Статус: представляет статус, в котором находится заявка.
* Идентификатор заявившего: представляет идентификатор жильца, подавшего заявку.
* Идентификатор ремонтника: представляет идентификатор сотрудника, устраняющего или устранившего неполадку.

Домены:

* Идентификатор заявки представляет уникальное число.
* Статус имеет строковый тип.
* Идентификатор заявившего представляет уникальное число.
* Идентификатор ремонтника представляет уникальное число.

1. Сущность "Выговор":

Атрибуты:

* Идентификатор выговора: представляет идентификатор выговора, вынесенного сотрудником жильцу.
* Статус: представляет статус, в котором находится выговор.
* Суть: представляет суть выписанного выговора.
* Идентификатор жильца: представляет идентификатор жильца, получившего выговор.
* Идентификатор сотрудника: представляет идентификатор сотрудника, выписавшего выговор.

Домены:

* Идентификатор выговора представляет уникальное число.
* Статус имеет строковый тип, по умолчанию активен.
* Суть имеет строковый тип.
* Идентификатор жильца представляет уникальное число.
* Идентификатор сотрудника представляет уникальное число.

1. Сущность "Мебель":

Атрибуты:

* Идентификатор мебели: представляет идентификатор мебели, находящейся в комнате.
* Статус: представляет статус, в котором находится выговор.
* Наименование: представляет полное наименование мебели.
* Идентификатор комнаты: представляет идентификатор комнаты, в которой находится мебель.

Домены:

* Идентификатор мебели представляет уникальное число.
* Статус имеет строковый тип, по умолчанию активен.
* Наименование имеет строковый тип.
* Идентификатор комнаты представляет уникальное число.

1. Сущность "ОПТ":

Атрибуты:

* Идентификатор ОПТ: представляет идентификатор записи в журнале ОПТ.
* Количество: представляет количество часов, которые жилец отработал.

Домены:

* Идентификатор ОПТ представляет уникальное число.
* Количество имеет числовой тип, по умолчанию 0.

1. Сущность "Дежурства":

Атрибуты:

* Идентификатор дежурств: представляет идентификатор записи в журнале дежурств.
* Количество: представляет количество смен, которые жилец отработал.

Домены:

* Идентификатор дежурств представляет уникальное число.
* Количество имеет числовой тип, по умолчанию 0.

Каждая сущность представляет конкретный объект в системе и имеет набор связанных с ней атрибутов, которые описывают ее свойства. Также все сущности имеют некоторые связи между друг другом, к примеру, жилец может отрабатывать ОПТ, сотрудник выписывать выговор, а комната иметь мебель.

Логическая диаграмма базы данных является важным инструментом в проектировании информационных систем, предоставляя абстрактное представление о структуре и взаимосвязях данных. На такой диаграмме могут быть изображены сущности, представляющие объекты в системе и их атрибуты, описывающие характеристики этих объектов. Отношения между сущностями отражают связи и зависимости между ними, такие как "один ко многим" или "многие ко многим". Таблицы базы данных, их поля и типы данных также могут быть представлены на логической диаграмме. Такая визуализация облегчает понимание структуры данных, что полезно при разработке и оптимизации баз данных для эффективного хранения и управления информацией.

Изображение логической модели базы данных студенческого общежития представлено на рисунке 2.1. Наиболее важными таблицами в этой системе являются *Users*, *Residents* и *Employees*. Уже на них могут ссылаться внешние ключи из других таблиц, например *FK* *user\_id* ссылается на первичный ключ *PK* *user\_id*, тем самым образуя связь между ними.

Изображение выглядит как снимок экрана, дизайн

Автоматически созданное описание

Рисунок 2.1 – Логическая диаграмма базы данных

Ещё одним способом проектирования является *ER*-диаграмма. *ER*-диаграмма (*Entity-Relationship Diagram*) в нотации Питера Чена является графическим средством моделирования баз данных, предложенным Питером Ченом. Эта нотация предоставляет инструмент для визуального представления сущностей и их отношений в информационной системе. ER-диаграммы в нотации Питера Чена помогают лучше понять структуру базы данных, их ключевые сущности и отношения, что важно для проектирования и создания эффективных баз данных.

На рисунке 2.2 изображены все сущности, их атрибуты и отношения. Сущности изображаются прямоугольниками, атрибуты – овалами, а отношения – ромбами. Также имеются обычные линия – это линии связи. Всего на диаграмме имеется 10 сущностей и 10 отношений между ними. Наиболее важными отношениями являются: сущность «Жилец» имеет отношение «Живёт в» с сущностью «Комната», сущность «Сотрудник» имеет отношение «Выносит» с сущностью «Выговор».

Изображение выглядит как снимок экрана, шаблон, Графика, графический дизайн

Автоматически созданное описание

Рисунок 2.2 – ER-диаграмма в нотации Питера Чена

## **Проектирование приложения**

Проектирование приложения является неотъемлемой частью разработки программного продукта, обеспечивая техническую основу для последующей реализации и тестирования. Этот этап направлен на достижение целей, учтенных в начальном этапе проекта, и обеспечивает рамки для успешной разработки и внедрения приложения в реальную среду использования.

Раздел курсового проекта, посвященный проектированию приложения, играет ключевую роль в описании технической архитектуры и функциональности будущего программного продукта. Цель этого раздела заключается в том, чтобы предоставить подробный обзор структуры, компонентов и взаимосвязей, которые будут реализованы в приложении. Проектирование приложения — это этап, на котором определяется, как система будет реализована на практике на основе предварительных анализов и требований.

Первым пунктом проектирования приложения является диаграмма вариантов использования. Диаграмма вариантов использования (*Use Case Diagram*) представляет собой графическое изображение функциональности системы и взаимодействия её актеров (пользователей) с этой функциональностью. Этот инструмент анализа системы широко применяется в процессе проектирования информационных систем.

На диаграмме вариантов использования выделяются два основных элемента: актёры и варианты использования.

Актёры (*Actors*) представляют собой внешних пользователей или другие системы, взаимодействующие с разрабатываемой системой. Актёр может быть человеком, группой пользователей или внешней системой, которая взаимодействует с системой.

Варианты использования (*Use Cases*). Варианты использования представляют собой конкретные сценарии взаимодействия между актёрами и системой. Они описывают, как система реагирует на действия актёров для достижения конкретных целей.

Диаграмма вариантов использования помогает разработчикам и заказчикам лучше понять функциональные требования к системе и обеспечивает основу для дальнейшего проектирования и реализации программного продукта.

На рисунке 2.3 изображены два действующих лица: «Пользователь» и «Администратор». Основными вариантами использования являются: «Авторизоваться», «Выйти», «Выбрать таблицу» и «Выбрать пользователя».

Диаграмма алгоритма — это визуальное представление последовательности шагов, выполняемых для достижения определенной цели или решения конкретной задачи. Она позволяет легко воспринимать и анализировать логику выполнения алгоритма. Существует несколько типов диаграмм алгоритмов, однако одним из наиболее распространенных является блок-схема.

Каждый прямоугольник представляет собой определенный шаг алгоритма. Внутри блока обычно записывается описание выполняемой операции или инструкции.

Стрелки соединяют прямоугольники и указывают направление выполнения алгоритма. Они показывают порядок выполнения шагов от начала к концу.

Ромбы используются для представления условных операторов. Внутри ромба записывается условие, и в зависимости от результата выполнения условия алгоритм может перейти к определенному блоку.

Овалы обозначают начало и конец выполнения алгоритма. Внутри овала обычно записывают заголовок, указывающий на цель алгоритма.

Диаграммы алгоритмов широко применяются при проектировании и программировании для понимания, анализа и документирования логики выполнения различных операций. Они облегчают восприятие и взаимодействие с кодом, а также улучшают коммуникацию между членами команды разработки.

Изображение выглядит как снимок экрана, круг, текст, Шрифт

Автоматически созданное описание

Рисунок 2.3 – Диаграмма вариантов использования

На рисунках 2.4 и 2.5 изображены алгоритм регистрации и алгоритм блокировки пользователя соответственно. Эти алгоритмы иллюстрируют последовательность действий именно в приложении для студенческого общежития, а также соответствующие проверки.

Алгоритм регистрации создан исключительно для новых пользователей, ещё не имеющих собственного профиля в системе. Начальным шагом является ввод собственной почты, которая по совместительству является логином, и пароля. Стоит заметить, что пароль вводится два раза. После ввода производятся две проверки: является ли введённый логин незанятым и совпадает пароль из пункта «Подтверждение пароля» с изначально введённым паролем. Если проверки успешно пройдены пользователь переадресуется на страницу с авторизацией. На этом выполнение алгоритма регистрации завершается.

Изображение выглядит как диаграмма, текст, линия, шаблон

Автоматически созданное описание

Рисунок 2.4 – Алгоритм регистрации

Алгоритм блокировки пользователя реализован для пользователей с уровнем доступа «Администратор», которые обладают расширенным списком возможностей. Подразумевается, что перед началом алгоритма блокировки администратор уже выбрал пользователя, которого требуется заблокировать. Администратор нажимает кнопку с получением данных и проверяет тем самым, заблокирован ли выбранный пользователь. Если выбранный пользователь заблокирован, то алгоритм завершается, иначе происходит следующий шаг. Администратор вводит идентификатор блокируемого пользователя и нажимает кнопку. В случае правильно выбранного идентификатора соответствующий пользователь блокируется. Если после последующей проверки администратор получил удовлетворительный результат алгоритм завершается успешно.

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рисунок 2.5 – Алгоритм блокировки пользователя

В данном подразделе были рассмотрены три важных инструмента в области информационных технологий: логическая диаграмма базы данных, диаграмма вариантов использования и диаграмма алгоритма. Использование таких диаграмм способствует более эффективному взаимодействию между членами команды, а также повышает понимание требований и структуры разрабатываемых систем.

# **РАЗРАБОТКА БАЗЫ ДАННЫХ ОБЩЕЖИТИЯ**

## **Выбор средств реализации**

Данный подраздел в проектной документации является ключевым этапом, определяющим технологический стек и инструменты, которые будут использоваться для реализации системы. Этот раздел включает в себя несколько пунктов: выбор системы управления базами данных (СУБД), выбор языка программирования для веб-приложения, определение языка программирования, который будет использоваться для разработки серверной и клиентской частей веб-приложения, выбор веб-фреймворка, выбор инструментов разработки и управления кодом.

Каждое из этих решений следует тщательно обосновать, учитывая требования проекта, опыт разработки и другие факторы. Важно подчеркнуть, что выбор технологического стека должен соответствовать целям проекта и обеспечивать его эффективное и устойчивое развитие.

Системы управления базами данных (СУБД) представляют собой программное обеспечение для создания и управления базами данных. Существует множество СУБД, и они могут быть разделены на различные типы в зависимости от их структуры, модели данных, способа хранения информации и других параметров. Далее представлены основные типы СУБД и примеры каждого:

Реляционные СУБД (*RDBMS*): основаны на модели данных в виде таблиц (отношений). Используют язык структурированных запросов *SQL*. Обеспечивают целостность данных и связи между таблицами. Примеры: *MySQL*, *PostgreSQL*, *Oracle* *Database*, *Microsoft* *SQL* *Server*.

*NoSQL* СУБД: различные модели данных (ключ-значение, документоориентированные, столбцовые, графовые), гибкая схема данных, подходят для работы с большими объемами неструктурированных данных. Примеры: *MongoDB* (документоориентированная), *Cassandra* (столбцовая), *Redis* (ключ-значение), *Neo4j* (графовая).

Иерархические СУБД: данные организованы в виде древовидной структуры, которая используются для моделирования иерархий. Примеры: *IBM* *IMS* (*Information* *Management* *System*), *XML*-базы данных.

Объектно-ориентированные СУБД: поддерживают понятие объектов и классов, предоставляют инструменты для хранения и манипулирования объектами. Примеры: *db4o*, *ObjectDB.*

Встраиваемые СУБД: интегрированы в приложение и работают в контексте этого приложения, обычно легки и быстры в использовании. Примеры: *SQLite*, *H2* *Database*.

Каждая СУБД имеет свои преимущества и недостатки в зависимости от конкретных потребностей проекта. Выбор подходящей СУБД зависит от требований к производительности, масштабируемости, структуры данных и других факторов.

*PostgreSQL*, часто называемая *Postgres*, является мощной и открытой реляционной системой управления базами данных (*RDBMS*). Вот несколько ключевых характеристик и особенностей *PostgreSQL*:

- *PostgreSQL* распространяется под лицензией *PostgreSQL*, которая является открытой лицензией с открытым исходным кодом. Это позволяет пользователям свободно использовать, изменять и распространять систему.

- Поддерживает широкий набор функций структурированных запросов *SQL*, включая сложные запросы, триггеры, хранимые процедуры и многое другое.

- Позволяет создавать пользовательские типы данных, операторы, функции и агрегатные функции, что обеспечивает высокую степень гибкости и адаптации под конкретные потребности.

- Поддерживает *ACID*-свойства (*Atomicity*, *Consistency*, *Isolation*, *Durability*), что обеспечивает надежность и целостность данных в случае сбоев или ошибок.

- Поддерживает одновременную работу нескольких пользователей с возможностью управления уровнем изоляции транзакций.

- Предоставляет встроенную поддержку для геопространственных данных, что делает *PostgreSQL* подходящим выбором для географических информационных систем (ГИС). Также в нем есть поддержка временных типов данных.

- *PostgreSQL* предоставляет механизмы для создания и использования расширений (*extensions*), а также поддерживает хранение и запросы *JSON*-данных.

- Позволяет реализовать репликацию данных для обеспечения отказоустойчивости и масштабируемости.

- *PostgreSQL* имеет большое активное сообщество пользователей и разработчиков, что обеспечивает поддержку, обмен опытом и разработку новых возможностей.

Данная СУБД широко используется в различных проектах и приложениях, включая веб-приложения, геоинформационные системы, аналитические системы и другие области, где требуется надежное и масштабируемое хранилище данных.

*JavaScript* является высокоуровневым, интерпретируемым языком программирования, который широко используется для создания динамических веб-страниц. Он является основным языком сценариев для веб-браузеров и позволяет создавать интерактивные пользовательские интерфейсы. *JavaScript* использует объектную модель, что позволяет разработчикам создавать объекты, содержащие методы и свойства, обеспечивая таким образом структурный подход к программированию. Код *JavaScript* выполняется интерпретатором во время выполнения, что облегчает быстрое тестирование и разработку. *JavaScript* часто используется для обработки событий, таких как щелчки мыши, отправка форм, загрузка страницы и другие, что делает его подходящим для создания интерактивных веб-приложений. Встроен в большинство современных веб-браузеров, таких как *Chrome*, *Firefox*, *Safari* и др. Переменные в *JavaScript* не требуют явного указания типа данных и могут изменять свой тип в процессе выполнения программы.

*Node.js* — это среда выполнения *JavaScript*, построенная на движке *V8* от *Google* *Chrome*. Она позволяет выполнять код *JavaScript* на стороне сервера, вне браузера. Вот несколько ключевых характеристик *Node.js*: асинхронное выполнение, высокая производительность, модульность, создание веб-серверов, работа с сетью и вводом-выводом, расширенные возможности.

Эта платформа стала популярным выбором для разработки серверных приложений, особенно в области веб-разработки, благодаря своей эффективности и асинхронному характеру, что делает его подходящим для обработки большого количества одновременных соединений.

*React* — это библиотека *JavaScript*, разработанная компанией *Facebook*, предназначенная для создания пользовательских интерфейсов (*UI*) веб-приложений. *React* позволяет разработчикам строить многокомпонентные приложения, где каждый компонент представляет собой независимую часть интерфейса.

Вот несколько ключевых концепций *React*:

* Основная идея React заключается в создании переиспользуемых компонентов. Компоненты могут быть простыми, такими как кнопка или текстовое поле, или сложными, такими как форма или целый блок контента. Компоненты могут вкладываться друг в друга для создания более сложных интерфейсов.
* *React* использует виртуальный *DOM* для оптимизации производительности. Вместо многократного обновления реального *DOM* при каждом изменении данных, *React* сначала обновляет виртуальный *DOM*, сравнивает его с текущим состоянием и затем обновляет только те части реального *DOM*, которые изменились.
* *React* использует *JSX (JavaScript XML)* для описания структуры пользовательского интерфейса. *JSX* представляет собой синтаксический сахар, который позволяет писать код, напоминающий *HTML*, но при этом остается чистым *JavaScript*.
* Компоненты могут иметь состояние, которое позволяет им хранить и обновлять данные в течение жизненного цикла. Свойства передаются компонентам сверху вниз и используются для передачи данных от родительских к дочерним компонентам.
* Компоненты в *React* имеют жизненный цикл, который включает в себя методы, вызываемые на различных этапах их существования. Например, *componentDidMount* вызывается после того, как компонент был добавлен в *DOM*, а *componentWillUnmount* вызывается перед удалением компонента из *DOM*.

Фреймворк *React* стал популярным инструментом для разработки веб-приложений благодаря своей эффективной системе компонентов, управлению состоянием и виртуальному *DOM*, что обеспечивает высокую производительность даже в больших и сложных приложениях.

*Express.js* — это минималистичный и гибкий веб-фреймворк для приложений на языке *JavaScript*, основанный на *Node.js*. Он предоставляет набор инструментов для разработки веб-приложений и *API*, упрощая создание серверов и маршрутизацию запросов.

Ниже приведены несколько ключевых аспектов *Express.js*:

* *Express* предоставляет простой и интуитивно понятный способ определения обработчиков запросов для различных *URL*-маршрутов. Маршруты определяются с использованием *HTTP*-методов (например, *GET*, *POST*) и шаблонов *URL*.
* *Express* использует *middleware* для выполнения операций в процессе обработки запроса. *Middleware* — это функции, которые имеют доступ к объектам запроса (*req*), ответа (*res*) и следующей функции *middleware* в стеке. *Middleware* может использоваться, например, для логирования, обработки ошибок, аутентификации и других задач.
* *Express* не включает в себя средства шаблонизации из коробки, но разработчики могут легко интегрировать свои любимые шаблонизаторы (например, *EJS*, *Pug*, *Handlebars*) для удобного создания *HTML*-страниц.
* *Express* упрощает предоставление статических файлов, таких как изображения, стили и скрипты, с использованием встроенного *middleware express.static*.
* *Express* часто используется для создания *RESTful* *API*, благодаря простоте маршрутизации и поддержке различных *HTTP*-методов.

После тщательного анализа требований проекта и основных характеристик различных средств, выбраны следующие технологии и средства для реализации базы данных и веб-приложения: *PostgreSQL*, *JavaScript*, *ReactJS*, *NodeJs* и *ExpressJS*. Весь процесс реализации кода программы производится в *IDE* от Майкрасофт *Visual Studio Code*.

*VSCode* — это бесплатный редактор кода, разработанный и поддерживаемый *Microsoft*. Он предоставляет множество функций, которые делают процесс разработки приложений более эффективным и приятным. Имеет открытый исходный код.

Таким образом, результирующий набор средств реализации проекта представлен в таблице 3.1.

Таблица 3.1 – Средства реализации веб-приложения

|  |  |
| --- | --- |
| Характеристика | Средство реализации |
| Языки программирования | *JavaScript* |
| СУБД | *PostgreSQL* |
| Фреймворк | *React* |
| Интерфейс | *HTML/CSS* |
| Среда разработки | *Visual Studio Code* |
| Библиотеки | *Express, Router, XLSX, Chart* |

## **Реализация базы данных**

Физическая модель базы данных описывает, как данные фактически хранятся и организованы на уровне хранилища данных. Эта модель конкретизирует структуры и отношения между данными, представленными в логической модели, на уровне, близком к аппаратному обеспечению и операционной системе. Основные компоненты физической модели базы данных включают в себя:

1. Таблицы. В физической модели таблицы преобразуются в реальные файлы или структуры хранения данных. Каждая таблица имеет свою собственную структуру файлов, определяющую распределение данных на диске.

2. Столбцы и поля. Каждый столбец таблицы преобразуется в соответствующее поле в файле данных. Здесь определяются типы данных, длина полей, форматы хранения и другие атрибуты, связанные с конкретным хранилищем.

3. Индексы. Физическая модель включает в себя описание индексов, которые оптимизируют процессы поиска и доступа к данным. Индексы могут быть представлены в виде деревьев или других структур данных, ускоряющих выполнение запросов.

4. Кластеризация и разделение данных. Физическая модель также определяет, как данные разделяются между различными уровнями хранилища (например, разные диски, таблицы пространств) и как они кластеризуются для оптимизации производительности.

5. Ограничения целостности. В физической модели описываются ограничения целостности на уровне базы данных, такие как ограничения уникальности, внешние ключи и другие правила, которые должны соблюдаться на уровне хранилища данных.

6. Хранимые процедуры и триггеры. Если используются хранимые процедуры или триггеры, они также включаются в физическую модель, определяя способы их хранения и выполнения.

7. Файловые группы и пространства таблиц. Физическая модель может также включать концепцию файловых групп и пространств таблиц для эффективного управления физическими файлами базы данных.

8. Физические зависимости и оптимизации. Модель описывает различные физические аспекты, такие как распределение данных по разным дискам, размеры блоков данных и другие оптимизации, направленные на улучшение производительности системы.

Физическая модель базы данных тесно связана с конкретной СУБД и инфраструктурой, на которой она работает, и может отличаться в зависимости от конкретных требований и характеристик системы.

В данном курсовом проекте для хранения данных используется реляционная база данных, которая представляет собой множество взаимосвязанных таблиц, приведенных к третьей нормальной форме. Схема реляционной базы данных представлена на рисунке 3.1.

Разработанная база данных содержит 13 таблиц: *suw, floor\_duty, rooms, furniture, residents, applications, repair, employees, reprimands, usesrs, history*.

Таблица «*users*» - содержит информацию о пользователе: идентификатор, логин, пароль в захешированном виде и роль пользователя. Первичным ключом является *user\_id*.

Таблица «*history*» - содержит информацию о истории: идентификатор, название таблицы, название операции и дату операции. Первичным ключом является *id*.

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание

Рисунок 3.1 – Физическая схема базы данных

Таблица «*rooms*» - содержит информацию о комнатах: идентификатор, количество мебели, количество жильцов и статус. Первичным ключом является *room\_id*.

Таблица «*furniture*» - содержит информацию о мебели: идентификатор, наименование, статус и номер комнаты. Первичным ключом является *room\_id*. Внешним ключом является *room\_id*, ссылающийся на таблицу rooms.

Таблица «*employees*» - содержит информацию о сотруднике: идентификатор, должность, номер телефона, адрес и зарплату. Первичным ключом является *employee\_id*.

Таблица «*suw*» - содержит информацию об ОПТ: идентификатор, количество часов по месяцам и суммарное количество часов. Первичным ключом является *suw\_id*.

Таблица «*floor\_duty*» - содержит информацию о дежурствах: идентификатор, количество дежурств по месяцам и суммарное количество дежурств. Первичным ключом является *duty\_id*.

Таблица «*residents*» - содержит информацию о жильце: идентификатор, ФИО, номер телефона, регистрацию, номер заявления, номер комнаты, идентификатор ОПТ и идентификатор дежурств. Первичным ключом является *resident\_id*. Внешними ключами являются *room\_id*, *application\_id*, *suw\_id*, *duty\_id*, ссылающиеся на таблицы *rooms*, *applications*, *suw*, *floor\_duty*.

Таблица «*applications*» - содержит информацию о заявлении на заселение: идентификатор, номер комнаты, дату подачи и дату заселения. Первичным ключом является *application\_id*. Внешним ключом является *room\_id*, ссылающийся на таблицу *rooms*.

Таблица «*repair*» - содержит информацию о выговорах: идентификатор, статус, суть поломки, номер комнаты, идентификатор жильца и идентификатор сотрудника. Первичным ключом является *repair\_id*. Внешними ключами являются *room\_id*, *resident\_id*, *employee\_id*, ссылающиеся на таблицы *rooms*, *residents*, *employees*.

Таблица «*reprimands*» - содержит информацию о выговорах: идентификатор, статус, суть поломки, идентификатор жильца и идентификатор сотрудника. Первичным ключом является *reprimand\_id*. Внешними ключами являются *room\_id*, *resident\_id*, *employee\_id*, ссылающиеся на таблицы *residents*, *employees*.

Все идентификаторы и счетчики имеют числовой тип, даты имеют тип дата, а остальные поля имеют текстовый тип.

## **Программная реализация приложения**

Структура веб-приложения для взаимодействия с базой данных студенческого общежития состоит из трёх частей: клиентская, серверная и база данных.

Ниже приведён код из клиентской части веб-приложения. Этот код выполняет асинхронную проверку роли пользователя, используя данные, хранящиеся в локальном хранилище браузера.

*const checkUserRole = async () => {*

*const userLogin = localStorage.getItem('userLogin');*

*try {*

*const response = await fetch(`http://localhost:8080/api/user/role/${userLogin}`);*

*const responseData = await response.json();*

*if (!response.ok) {*

*throw new Error(`HTTP error! Status: ${response.status}`);*

*}*

*if (!responseData || !responseData.role) {*

*throw new Error('Role information not available in the response.');*

*}*

*const userRole = responseData.role;*

*if (userRole === 'ban') {*

*console.log('Вы забанены!');*

*alert('Вы забанены! Свяжитесь с администратором...');*

*navigate('/login');*

*}*

*} catch (error) {*

*console.error('Error checking user role:', error.message);*

*} };*

Сначала объявление функции *checkUserRole*, которая асинхронна, то есть она может выполнять асинхронные операции, такие как запросы к серверу. Получение значения *userLogin* из локального хранилища браузера и сохранение его в переменной *userLogin*. Предполагается, что это имя пользователя или какой-то уникальный идентификатор. Блок *try* используется для обработки возможных ошибок, которые могут произойти в процессе выполнения кода. Если происходит ошибка, управление передается блоку *catch*, где можно обработать ошибку. Использование функции *fetch* для отправки асинхронного запроса к серверу. *URL* запроса формируется с использованием значения *userLogin*. Ожидание ответа с использованием ключевого слова *await*. Ожидание преобразования ответа сервера в формат *JSON*. Полученные данные сохраняются в переменной *responseData*. Проверка, был ли успешен *HTTP*-запрос. Если статус ответа не в пределах 200-299, выбрасывается ошибка с сообщением об ошибке *HTTP*. Проверка, содержат ли данные роль пользователя. Если нет, выбрасывается ошибка с сообщением о том, что информация о роли недоступна в ответе. Извлечение роли пользователя из данных ответа. Проверка, является ли роль пользователя "*ban*". Если да, выводится сообщение в консоль, выводится предупреждение с использованием *alert*, и пользователь перенаправляется на страницу входа с использованием функции *navigate('/login').* Обработка возможных ошибок. Если что-то пошло не так при выполнении кода в блоке *try*, сообщение об ошибке выводится в консоль.

Далее приведён код из серверной части приложения. Этот код представляет собой обработчик запроса (*handler*) для бана пользователя.

*async banUser(req, res) {*

*const { id } = req.body;*

*try {*

*const banedUser = await db.query(`UPDATE users*

*SET user\_role = 'ban'*

*WHERE users.id = $1;`, [id]);*

*res.json({ success: true, message: 'Пользователь успешно забанен' });*

*} catch (error) {*

*console.error('Ошибка при бане пользователя:', error);*

*res.status(500).json({ error: 'Внутренняя ошибка сервера' });*

*} }*

Сначала объявление асинхронной функции *banUser*, которая принимает объекты запроса *req* и ответа *res* в качестве параметров. Обычно это используется в контексте веб-приложений, где *req* содержит информацию о запросе, а *res* используется для отправки ответа клиенту. Извлечение значения *id* из тела запроса (*req.body*). Предполагается, что клиент отправляет объект, в котором есть свойство *id*, и это значение используется для идентификации пользователя, которого нужно забанить. Здесь используется блок *try-catch* для обработки возможных ошибок, которые могут возникнуть в процессе выполнения кода. Использование асинхронного запроса к базе данных для обновления записи в таблице пользователей (*users*). Запрос устанавливает роль пользователя в значение *ban*, где идентификатор пользователя равен переданному значению *id*. Если операция бана выполнена успешно, возвращается *JSON*-ответ с информацией об успешном выполнении и сообщением. В случае возникновения ошибки выводится сообщение об ошибке в консоль, и сервер возвращает клиенту статус 500 (внутренняя ошибка сервера) вместе с *JSON*-ответом, указывающим на возникшую ошибку.

## **Инструкция пользователя**

При первом посещении сайта перед пользователем открывается страница регистрации (рисунок 3.2). На этой странице имеется форма, которую нужно заполнить: в первом поле для ввода требуется ввести почту, далее пароль, а в третьем поле повторить пароль, чтобы исключить ситуации с неверным запоминанием пароля. Если пароль введён некорректно, пользователь получит уведомление «Пароли не совпадают». Под формой имеется ссылка для пользователей, уже имеющих собственный профиль. Перейдя по этой ссылке либо успешно пройдя процесс регистрации, пользователь переадресуется на страницу авторизации. На рисунке 3.3 изображён интерфейс авторизацию, который практически полностью аналогичен интерфейсу регистрации.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3.2 – Интерфейс регистрации пользователя

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рисунок 3.3 – Интерфейс авторизации пользователя

При прохождении процесса авторизации введённые пользователем данные проходят проверку: пароль хешируется, логин проверяется на существование, а роль пользователя сверяется. Если какой-то пункт нарушается пользователь не получается доступа к дальнейшему взаимодействию с системой.

На рисунке 3.4 виден интерфейс страницы пользователя после успешной авторизации. В самом верху расположена навигационная панель, с помощью которой можно совершить переход на другие страницы приложения. Под навигационной панелью расположены 3 блока: блок с информацией о пользователе, блок объявлений и блок изменения контактных данных.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 3.4 – Интерфейс страницы пользователя

Блок изменения контактных данных имеет форму для заполнения и соответствующую кнопку, при нажатии которой все введённые данные отправяться на сервер и сразу же отобразятся в блоке информацией о пользователе.

В самом низу страницы имеется панель с информацией о техподдержке: мобильный номер телефона и адрес электронной почты. Эта панель имеет информационный характер.

Если с помощью навигационной панели перейти на страницу просмотра содержимого базы-данных, то пользователю отбразиться интерфейс с рисунка 3.5. Можно заметить, что навигационная панель и нижняя панель остались.  
 Далее от пользователя требуется выбрать нужную таблицу и нажать кнопку «Получить данные таблицы». После чего интерфейс страницы примет вид как на рисунке 3.6. Имеется два блока с формами и кнопками и одно табличное представление. В левом блоке есть поле для ввода, при любом изменении в котором происходит фильтрация данных в табличном представлении. Две кнопки в этом блоке позволяют экспортировать данные из табличного представления в нужном формате. В самой таблице данные можно сортировать при нажатии на заголовок столбца.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 3.5 – Интерфейс просмотра содержимого БД до выбора таблицы

В правом блоке имеется форма для изменения данных в таблице, которые заносятся сразу после нажатия кнопки «Обновить». Последняя кнопка в блоке позволяет отобразить над таблицей диаграмму, построенную на основе данных из таблицы (рисунок 3.7).

Если пользователь находясь на своей странице через навигационную панель решит посетить «Админ-панель», то произойдёт проверка его роли. Если роль «*admin*», то пользователю предоставиться доступ к админ-панели, иначе он получит уведомление «Вы не админ!». Аналогичная проверка происходит при авторизации, но отказ произойдёт, если значение роли «ban». Оповещение о блокировке происходит с помощью уведомления в окне браузера, а также через консоль браузера. По умолчанию у всех пользователей роль «*default*».

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание

Рисунок 3.6 – Интерфейс просмотра содержимого БД после выбора таблицы

Изображение выглядит как текст, снимок экрана, музыка

Автоматически созданное описание

Рисунок 3.7 – Диаграмма на основе содержимого таблицы

В результате перехода в админ-панель администратор может наблюдать интерфейс как на рисунке 3.8. Отличительным является доступ к таблицам: *users* и *history*. Их всё также можно экспортировать. При этом для табличного представления с пользователями имется изменённый функционал. Для введённого в поле для ввода идентификатора имеются три кнопки: «Забанить», «Разбанить» и «Удалить». Блокировка и разблокировка меняют роль пользователя, а удаление полность стирает данные пользователя из базы данных.

Для выхода из системы с любой страницы достаточно нажать кнопку «Выйти», после чего из памяти браузера удалиться логин, под которым вошёл пользователь.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 3.8 – Интерфейс админ-панели

В данном подразделе рассмотрены основные аспекты взаимодействия пользователя с веб-приложением. Можно выделить интуитивно понятный интерфейс, обратную связь, визуальные подсказки и доступность. С учетом этих аспектов веб-приложение может предоставить более удовлетворительный и эффективный опыт для пользователей, что является ключевым фактором в успехе любого веб-приложения.

# **ЗАКЛЮЧЕНИЕ**

В процессе выполнения курсового проекта проведен анализ изучаемой области, включающий описание исследуемого объекта, обзор сопоставимых аналогов и постановку задачи. Следующим этапом было проектирование системы студенческого общежития на основе сформулированных задач, включая создание базы данных с логической схемой и разработку приложения с определением основного функционала и структуры.

В ходе проектирования была определена логика, а также созданы *UML*-диаграммы вариантов использования для пользователей и администраторов, а также схемы алгоритма для процесса регистрации пользователя и блокировки.

Система управления базами данных студенческого общежития была разработана на основе полученных данных. При реализации курсового проекта применялись современные технологии создания веб-систем, такие как язык программирования *JavaScript*, фреймворк *React*, СУБД *PostgreSQL*, а также средства визуализации интерфейса *HTML*/*CSS*.

Дополнительно предоставлено подробное руководство по использованию системы для конечного пользователя.

Пояснительная записка оформлена в соответствии с требованиями стандарта СТП 01-2017 [10].

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Ивасенко А.Г. Информационные технологии в экономике и управлении: учебное пособие [Текст] / А.Г. Ивасенко, А.Ю. Гридасов, В.А. Павленко. - М.: КНОРУС. 2015. - 160 с.

[2] ГОСТ 33707-2016 (ISO/IEC 2382:2015) Информационные технологии (ИТ). Словарь.

[3] [Дейт К. Дж.](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D1%82,_%D0%9A%D1%80%D0%B8%D1%81%D1%82%D0%BE%D1%84%D0%B5%D1%80) [Введение в системы баз данных](https://ru.wikipedia.org/wiki/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B2_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B_%D0%B1%D0%B0%D0%B7_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) = *Introduction to Database Systems*. – 8-е изд. – М.: [Вильямс](https://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2005. – 1328 с.

[4] Крэг Ларман. Применение *UML* 2.0 и шаблонов проектирования = *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.* – 3-е изд. – М.: [Вильямс](https://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2006. – 736 с.

[5] Г. Буч, Д. Рамбо, И. Якобсон. Язык *UML*. Руководство пользователя = *The Unified Modeling Language Usere Guide*. – 2-е. – М.: ДМК Пресс, 2006. – 120 с.

[6] [Когаловский М.Р.](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B3%D0%B0%D0%BB%D0%BE%D0%B2%D1%81%D0%BA%D0%B8%D0%B9,_%D0%9C%D0%B8%D1%85%D0%B0%D0%B8%D0%BB_%D0%A0%D1%83%D0%B2%D0%B8%D0%BC%D0%BE%D0%B2%D0%B8%D1%87) Энциклопедия технологий баз данных. – М.: [Финансы и статистика](https://ru.wikipedia.org/w/index.php?title=%D0%A4%D0%B8%D0%BD%D0%B0%D0%BD%D1%81%D1%8B_%D0%B8_%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0&action=edit&redlink=1), 2002. – 800 с.

[7] В. Васвани. *MySQL*: использование и администрирование = *MySQL Database Usage & Administration*. – М.: [«Питер»](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D1%82%D0%B5%D1%80_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)), 2011. – 368 с.

[8] Джесс Чедвик и др. *ASP.NET MVC* 4: разработка реальных веб-приложений с помощью *ASP.NET MVC = Programming ASP.NET MVC 4: Developing Real-World Web Applications with ASP.NET MVC.* – М.: [«Вильямс»](https://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2013. – 432 с.

[9] К. Бауэр, Г. Грегори, Г. Кинг. *Java Persistence API и Hibernate = Java Persistence with Hibernate.* – 2-е. – [ДМК Пресс](https://ru.wikipedia.org/w/index.php?title=%D0%94%D0%9C%D0%9A_%D0%9F%D1%80%D0%B5%D1%81%D1%81&action=edit&redlink=1), 2017. – 632 с.

[10] СТП 01-2017. Стандарт предприятия. Дипломные проекты (работы). Общие требования. – Минск: БГУИР, 2017. – 169 с.

# **Приложение А**

**(обязательное)**

**Реализация *Databse.js***

*import { Link } from "react-router-dom";*

*import { useNavigate, useParams } from "react-router-dom";*

*import React, { useState, useEffect } from 'react';*

*import "./App.css";*

*import Employees from './components/Employees.js';*

*import Residents from './components/Residents.js';*

*import Rooms from './components/Rooms.js';*

*import Inventory from './components/Inventory.js';*

*import SUW from './components/SUW.js';*

*import Duty from './components/Duty.js';*

*import Applications from './components/Applications.js';*

*import Reprimands from './components/Reprimands.js';*

*import Repair from './components/Repair.js';*

*const NavigationBar = () => {*

*const navigate = useNavigate();*

*const handleLogout = () => {*

*localStorage.removeItem("userLogin");*

*navigate('/login');*

*};*

*return (*

*<div className="navigation-bar">*

*<div className="logo">*

*<img src="/images/logo.png" alt="Logo" />*

*<span>Общежитие</span>*

*</div>*

*<div className="nav-buttons">*

*<Link to="/login" onClick={handleLogout} className="nav-button">Выйти</Link>*

*<Link to="/page" className="nav-button">Профиль</Link>*

*<Link to="/admin" className="nav-button">Админ-панель</Link>*

*</div>*

*</div>*

*);*

*};*

*const DataBase = () => {*

*const [tableName, setTableName] = useState('');*

*const [data, setData] = useState([]);*

*const [selectedTable, setSelectedTable] = useState('');*

*const navigate = useNavigate();*

*const checkUserRole = async () => {*

*const userLogin = localStorage.getItem('userLogin');*

*try {*

*const response = await fetch(`http://localhost:8080/api/user/role/${userLogin}`);*

*const responseData = await response.json();*

*if (!response.ok) {*

*throw new Error(`HTTP error! Status: ${response.status}`);*

*}*

*if (!responseData || !responseData.role) {*

*throw new Error('Role information not available in the response.');*

*}*

*const userRole = responseData.role;*

*if (userRole === 'ban') {*

*console.log('Вы забанены!');*

*alert('Вы забанены! Свяжитесь с администратором...');*

*navigate('/login');*

*}*

*} catch (error) {*

*console.error('Error checking user role:', error.message);*

*}*

*};*

*useEffect(() => {*

*checkUserRole();*

*}, []);*

*const handleFetchData = async () => {*

*try {*

*const userLogin = localStorage.getItem("userLogin");*

*const response = await fetch(`http://localhost:8080/api/table/${selectedTable}`);*

*if (!response.ok) {*

*throw new Error(`HTTP error! Status: ${response.status}`);*

*}*

*const result = await response.json();*

*setData(result);*

*} catch (error) {*

*console.error('Error fetching data:', error);*

*}*

*};*

*const handleTableChange = (e) => {*

*setSelectedTable(e.target.value);*

*};*

*const handleSubmit = (e) => {*

*e.preventDefault();*

*handleFetchData();*

*};*

*const renderTableComponent = () => {*

*switch (selectedTable) {*

*case 'employees':*

*return <Employees data={data} />;*

*case 'residents':*

*return <Residents data={data} />;*

*case 'rooms':*

*return <Rooms data={data} />;*

*case 'furniture':*

*return <Inventory data={data} />;*

*case 'suw':*

*return <SUW data={data} />;*

*case 'floor\_duty':*

*return <Duty data={data} />;*

*case 'applications':*

*return <Applications data={data} />;*

*case 'reprimands':*

*return <Reprimands data={data} />;*

*case 'repair':*

*return <Repair data={data} />;*

*default:*

*return null;*

*}*

*};*

*return (*

*<div className="database">*

*<div className="database-main">*

*<NavigationBar />*

*<div className="db-frame">*

*<form onSubmit={handleSubmit} className="database-frame">*

*<select value={selectedTable} onChange={handleTableChange} className="login-input">*

*<option value="">Выберите таблицу</option>*

*<option value="employees">Работники</option>*

*<option value="residents">Жильцы</option>*

*<option value="rooms">Комнаты</option>*

*<option value="furniture">Инвентарь</option>*

*<option value="suw">ОПТ</option>*

*<option value="floor\_duty">Дежурства [Импорт]</option>*

*<option value="applications">Заявления на заселение</option>*

*<option value="reprimands">Выговоры</option>*

*<option value="repair">Неполадки</option>*

*</select>*

*<button type="submit" className="database-button" onClick={handleFetchData}>*

*<span className="login-button-text">*

*Получить данные таблицы*

*</span>*

*</button>*

*</form>*

*</div>*

*<div className="data-table">*

*{renderTableComponent()}*

*</div>*

*</div>*

*<div className="about">*

*<div className="texter">*

*УО "Белорусский государственный университет информатики и радиоэлектроники" <br></br>*

*© 2021-2023 buterkot. Все права защищены.*

*</div>*

*<div className="support">*

*<span>Техподдержка:</span>*

*<span><a href="tel:+375295173162" className="phone">+375 (29) 517-31-62</a></span>*

*<span className="mail">miheynikita@gmail.com</span>*

*</div>*

*</div>*

*</div>*

*);*

*};*

*export default DataBase;*

# **ВЕДОМОСТЬ КУРСОВОГО ПРОЕКТА**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | *Дополнительные сведения* | | | | |
|  | | | | | *Текстовые документы* |  | | | | |
|  | | | | |  |  | | | | |
| *БГУИР КП 1-53 01 02 418* *ПЗ* | | | | | *Пояснительная записка* | *45c.* | | | | |
|  | | | | |  |  | | | | |
|  | | | | | *Графический документ* |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | | *ER-диаграмма в нотации Питера Чена* | *Формат А3* | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  |  |  |  |  | *БГУИР КП 1-53 01 02 418* *Д1* | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| *Изм.* | *Л* | *№ докум.* | *Подп* | *Дата* | *База данных общежития*  *Ведомость курсового проекта* | *Лит* | | | *Лист* | *Листов* |
| *Разраб.* | | *Михейка* |  |  |  |  |  | *45* | *45* |
| *Провер.* | | *Трофимович* |  |  | *Кафедра ИТАС*  *гр. 120604* | | | | |
| *Т.контр.* | |  |  |  |
| *Н.контр.* | |  |  |  |
| *Утв.* | |  |  |  |
| *Реценз.* | |  |  |  |