Министерство науки и высшего образования Российской Федерации

ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»

Институт новых материалов и технологий

Кафедра «Теплофизика и информатика в металлургии»

Оценка работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель от УрФУ К.А.Щипанов

Тема задания на практику

«Разработка веб-приложения для системы контроля доступа»

ОТЧЕТ

Вид практики Производственная практика

Тип практики Преддипломная практика

Студент: Мазлова Елена Юрьевна

Код, направление подготовки: 09.03.02 – «Информационные системы и технологии»

Учебная группа: НМТ-493907

Екатеринбург 2023

Министерство науки и высшего образования Российской Федерации

ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»

|  |  |
| --- | --- |
|  | СОГЛАСОВАНО:  УрФУ  «10» апреля 2023 г. |
|  | Зав. кафедрой «Теплофизика и информатика в металлургии» ИНМТ  \_\_\_\_\_\_\_\_\_\_ Н.А. Спирин  Подпись расшифровка подписи |

Институт ИНМТ. Группа НМТ-493907. Кафедра «Теплофизика и информатика в металлургии»

Код, наименование направления 09.03.02 – Информационные системы и технологии (СУОС)

Наименование образовательной программы 09.03.02/33.11 – Информационные системы и технологии в металлургии

**ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ**

на преддипломную практику студента

Мазлова Елена Юрьевна

(фамилия, имя, отчество)

1. Тема задания на практику «Разработка веб-приложения для системы контроля доступа»

2. Срок практики с 10.04.2023 г. по 21.05.2023 г. Срок сдачи студентом отчета c 22.05.2023 г. по 28.05.2023 г.

3. Место прохождения практики Кафедра «Теплофизика и информатика в металлургии».

4. Вид практики: Производственная практика.

5. Тип практики: Преддипломная практика

6. Содержание отчета:

* Описание предметной области
* Обзор существующих программных продуктов
* Функциональные требования к программному обеспечению
* Описание структуры базы данных
* Описание разработанного программного обеспечения
* Фрагменты программного кода

**Рабочий график (план) проведения практики**

|  |  |  |  |
| --- | --- | --- | --- |
| **Этапы практики** | **Наименование работ студента** | **Срок** | **Примечание** |
| организационный | Ознакомление с рабочей программой практики; изучение методических рекомендаций по практике; согласование индивидуального задания с РП от УрФУ; усвоения правил техники безопасности и охраны труда. | 10.04.2023 г. –  17.04.2023 г. |  |
| основной | Проектирование и реализация функций программного обеспечения.  Документирование программного обеспечения. | 18.04.2023 г. –  07.05.2023 г. |  |
| заключительный | Подведение итогов и составление отчета: систематизация, анализ, обработка собранного в ходе практики материала, предоставление отчета. | 08.05.2023 г. –  21.05.2023 г. |  |

Содержание практики и планируемые результаты практики согласованы с руководителем:

Руководитель от УрФУ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К.А. Щипанов

Подпись расшифровка подписи

Задание принял к исполнению (студент) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.Ю. Мазлова

Подпись расшифровка подписи

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc135093780)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc135093781)

[1.1 Актуальность работы 5](#_Toc135093782)

[1.2 Описание работы СКУД 5](#_Toc135093783)

[1.3 Обзор существующих программных продуктов 7](#_Toc135093784)

[2 ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ 11](#_Toc135093785)

[2.1 Постановка задачи 11](#_Toc135093786)

[2.2 Определение требований пользователей к информационной системе 11](#_Toc135093787)

[3 РАЗРАБОТКА И ОПИСАНИЕ БАЗЫ ДАННЫХ 13](#_Toc135093788)

[3.1 Проектирование концептуальной модели базы данных в сервисе ERDPlus 13](#_Toc135093789)

[3.2 Разработка и описание даталогической модели базы данных на платформе Microsoft SQL Server 2014 13](#_Toc135093790)

[3.3 Автоматизация наполнения базы тестовыми данными с использованием сервиса Microsoft SQL Server Integration Services 20](#_Toc135093791)

[4 РАЗРАБОТКА И ОПИСАНИЕ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ 22](#_Toc135093792)

[4.1 Управление версиями проекта на основе сервиса GitHub 22](#_Toc135093793)

[4.2 Разработка клиентского приложения 23](#_Toc135093794)

[4.3 Описание работы приложения 32](#_Toc135093795)

[ЗАКЛЮЧЕНИЕ 37](#_Toc135093796)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 38](#_Toc135093797)

[ПРИЛОЖЕНИЕ А ФРАГМЕНТ ЛИСТИНГА ПРОГРАММНОГО КОДА КОНТРОЛЛЕРА HOMECONTROLLER.CS 39](#_Toc135093798)

[ПРИЛОЖЕНИЕ Б РУКОВОДСТВО АДМИНИСТРАТОРА СИСТЕМЫ 65](#_Toc135093799)

ВВЕДЕНИЕ

Данный отчет составлен по результатам прохождения производственной преддипломной практики в период с 10.04.2023 по 21.05.2023.

Место прохождения практики – ФГАОУ ВО УрФУ, кафедра «Теплофизика и информатика в металлургии», г. Екатеринбург.

Целью прохождения преддипломной практики является получение теоретических знаний, умений и навыков, а также их применений в практической деятельности.

Во время прохождения практики была реализована информационная система контроля передвижения сотрудников предприятия. Информационная система предполагает разработку схемы базы данных с помощью сервиса ERDPlus и ее последующую реализацию в пакете SQL Management Studio 2014. Далее было разработано пользовательское WEB-приложение в среде разработки Microsoft Visual Studio 2019 с использованием патча ASP.NET Core Web Application (MVC) на платформе .Net 5.0. Основными конечными пользователями приложения являются сотрудники охранного предприятия, их непосредственное руководство, а также администратор системы.

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

* 1. **Актуальность работы**

В настоящее время наиболее актуальным вопросом всех современных предприятий является безопасность сотрудников и охрана труда. Одним из важнейших составляющих безопасности является контроль передвижения сотрудников по средствам электронно-пропускной системы, а также видеонаблюдения на территории предприятия.

Система контроля и управления доступом, СКУД — это совокупность программно-аппаратных технических средств контроля и средств управления, имеющих целью ограничение и регистрацию входа-выхода объектов (людей, транспорта) на заданной территории через «точки прохода»: двери, ворота, КПП. Основная задача систем – управление доступом на заданную территорию, включая также:

• ограничение доступа на заданную территорию;

• идентификацию лица, имеющего доступ на заданную территорию.

Дополнительными задачами являются учёт рабочего времени, ведение базы персонала / посетителей, интеграция с системой безопасности (с системой видеонаблюдения, пожарно-охранной сигнализацией и системами оповещения).

На данный момент рынок программных обеспечений для СКУД-систем достаточно богат, однако не все программы удовлетворяют функциональным требованиям заказчиков, либо наоборот содержат множество излишних функций, замедляющих работу системы. Поэтому вопрос разработки простой, но при этом функциональной системы остается открытым.

* 1. **Описание работы СКУД-систем**

В общем виде все СКУД-системы работают следующим образом: у входа на объект устанавливается устройство аутентификации, например, считыватель карты доступа, и устройство, которое может по электрическому сигналу блокировать/разблокировать вход (дверь с электромагнитным замком, ворота, шлагбаум и т.д.). Контроллер, к которому они подключены, по локальной сети обменивается данными с управляющим компьютером и сервером, на котором хранится база данных пользователей (рисунок 1.1).

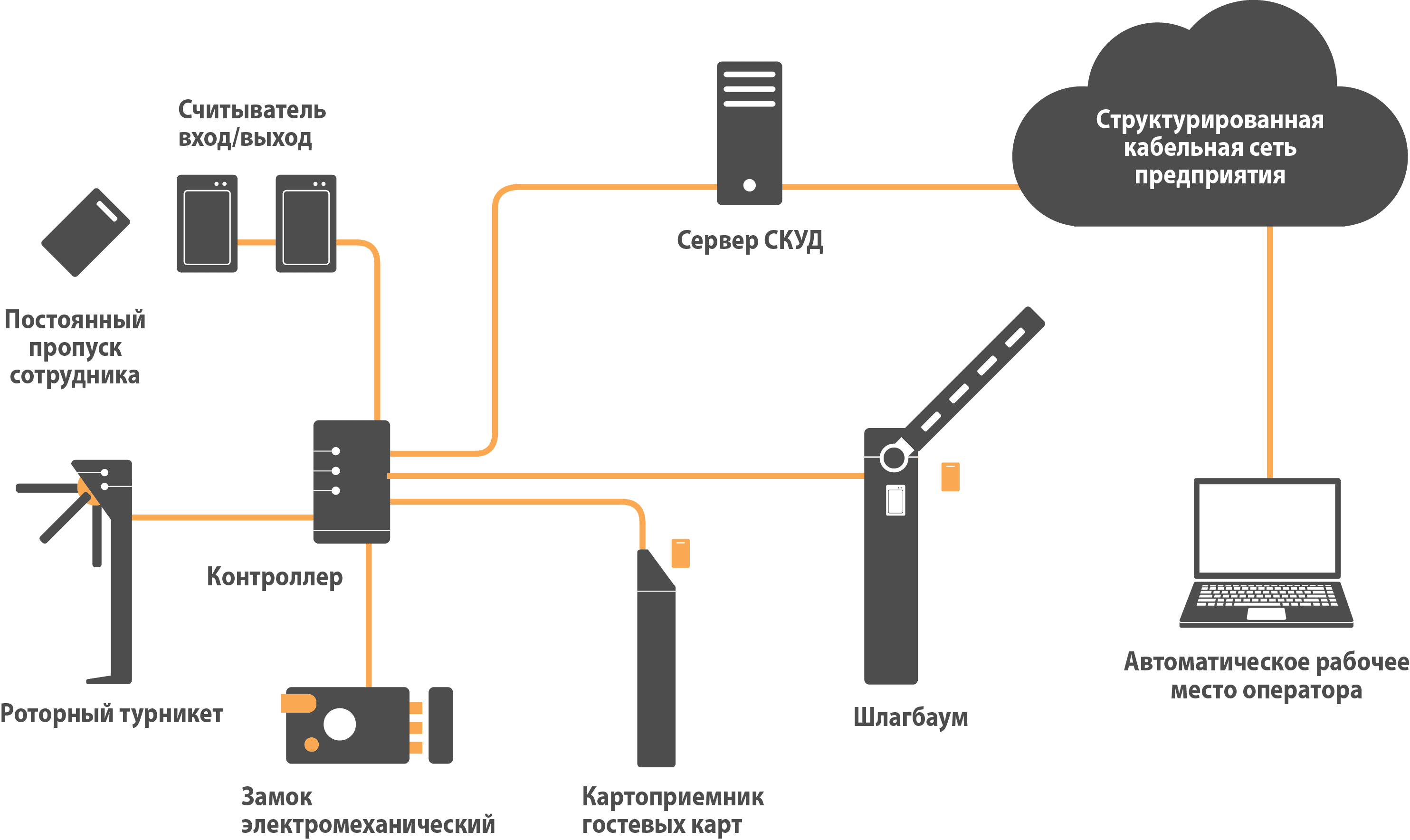


Рисунок 1.1 – Схема работы СКУД-системы

Пользователь подносит свою карту к считывателю, который передает его идентификационные данные на контроллер. Контроллер сверяет полученные идентификационные данные с теми, что хранятся в базе данных, и проверяет права доступа к данному объекту. В зависимости от результатов проверки, владельцу карты либо разрешается доступ на объект – на электромагнитный замок или турникет подается сигнал на открытие, или запрещается – замок остается закрытым. Шлагбаумы и электромеханические замки, автоматические ворота и турникеты, ростовые кабины и приводы ворот предназначены для ограничения доступа к охраняемому объекту. Эти устройства открываются или блокируются по сигналу, приходящему с контроллера.

Для регистрации нового пользователя в биометрической СКУД используется или один из считывателей системы, или сканер, подключенный к рабочему месту сотрудника, отвечающего за ввод новых пользователей. Данные нового пользователя и информация о том, к каким помещениям ему разрешен доступ, сразу же попадают в базу данных СКУД.

Каждый контроллер определяет логику работы одной или нескольких дверей (ворот, шлагбаумов). На вход контроллера поступает сигнал с устройства аутентификации, а также (возможно) сигналы с датчиков охранно-пожарной сигнализации и детекторов движения видеокамер. В соответствии с логикой, которая заложена в контроллер при настройке, он дает исполнительным устройствам сигнал на открытие/блокировку.

В зависимости от сложности системы и требуемого уровня безопасности, можно заложить в контроллер различные алгоритмы работы. Например, можно ввести для определенной группы пользователей ограничение доступа по времени: доступ в помещение будет разрешен только в определенный промежуток времени. Еще один распространенный алгоритм – доступ в помещение только в присутствии одного или более лиц с более высоким уровнем доступа.

* 1. **Обзор существующих программных продуктов**

В обзоре представлены следующие программные продукты (топ-10 приложений для СКУД по версии пользователей Google):

* ZkTeco
* ControlGate
* PERCo
* RusGuard
* Parsec
* HID Global
* Sigur

**«ZkTeco»**

Китайская компания «ZkTeco Co., Ltd» широко известна своей направленностью на биометрические методы верификации и предоставляет услуги для крупного и среднего бизнеса. Предприятие владеет более 90 филиалами по всему миру, в том числе России и странах СНГ.

Программное обеспечение базируется на облачной платформе ZKTeco + Smart Office. Оно поддерживает биометрическую идентификацию, видеонаблюдение, а также интеллектуальную систему входа и выхода «Человек + автомобиль + объект». ПО ZKTime.Net V3.0 предназначено для работы с системой учета рабочего времени компании, включает в себя возможности распознавания лиц, отпечатков и гибридную биометрию. На основе получаемой информации, программа способна создавать отчеты и платежные ведомости. Кроме этого, учитывая реалии времени, в компании разрабатывают методы бесконтактной идентификации COVID-19, что также интегрируется в программное обеспечение через соответствующие модули.

**«ControlGate»**

Российская компания «КонтролГейт» специализируется на программировании и разработке систем безопасности. ПО ControlGate включает в себя полный набор модулей для проведения наблюдения, контроля оборудования, управления доступом, а также анализа и сбора информации. Система безопасности объекта под управлением ПО ControlGate способна работать с широким спектром идентификаторов, в том числе и биометрических. При необходимости, есть возможности комбинировать между собой несколько типов идентификаторов. Например, биометрический терминал и считыватель ПИН-кода. Масштабирование системы осуществляется за счет подключения дополнительных модулей.

Основными плюсом софта является его кроссплатформенность, т.е. возможность работать на различных операционных системах (Windows, Linux, MacOs). Это позволяет интегрироваться с оборудованием таких производителей как: Hikvision, Dahua, ZKTeco, IronLogik.

СКУД ControlGate построена на принципах модульности. Существует основной базовый блок, к которому при необходимости подключаются дополнительные (информатор, бюро пропусков, СУРВ, обмен данными). Ярким плюсом служит то, что в базовом модуле предустановлены следующие компоненты: непосредственно контроль доступа, видеосистема, генератор отчетов, среда выполнения пользовательских сценариев, планировщик задач.

**«PERCo»**

Компания «PERCo» входит в число ведущих отечественных производителей оборудования в области безопасности. Комплексные ПО СКУД представлено системами PERCo-Web, PERCo-S-20 и PERCo- S-20 «Школа».

PERCo-Web устанавливается на отдельный компьютер и позволяет работать другим пользователям через веб-интерфейс. Система контролирует доступ лиц в здания, разграничение прав входа в помещения и учет времени работы персонала.

Основным назначением ПО PERCo-S-20 является организация комплексной работы систем безопасности предприятия. ПО координирует работу СКУД с системами пожарно-охранной сигнализации и видеонаблюдения.

Система PERCo-S-20 «Школа» разработана специально для установки в учебных заведениях. Она предназначена для недопущения прохода посторонних, ведет учет посещаемости учащихся с одновременным уведомлением родителей о времени прихода и ухода ребенка.

**«RusGuard»**

Компания «РусГард» специализируется на производстве оборудования для СКУД и охранной сигнализации. Программное обеспечение данной компании – софт для полноценной работы систем доступа. Выделяют три версии софта «Standart», «Premium» и «Enterprise». ПО способно решать задачи, как крупного предприятия, так и небольшого офиса. Связь осуществляется по защищенному протоколу HTTPS.

Важной особенностью является то, что покупателю оборудования передается комплект ПО бесплатно. В дальнейшем осуществляется бесплатная техподдержка. В то же время специалисты, отмечают достаточно высокую стоимость самого оборудования, что нивелирует плюсы от бесплатного программного обеспечения.

**«Parsec»**

Разработчиком и производителем данного программного обеспечения является ООО «Диамант групп». Компания занимается реализацией решений в сфере безопасности и идентификации объектов.

Программное обеспечение компании представлено программными модулями PNSoft и PNOffice, которые имеют множество вариаций для различных комбинаций оборудования. ПО PNSoft применяется на СКУД, устанавливаемых на предприятиях, в то время как PNSoft используется в системах доступа небольших объектов и больше ориентирован на учет и контроль персонала.

**«HID Global»**

Компания «HID Global» являет мировым лидером по производству СКУД. Головной офис компании расположен в г.Остин, штат Техас (США).

Программное обеспечение компании имеет весьма широкую направленность и используется в СКУД для физического и логического доступа персонала, а также в сфере выпуска финансовых инструментов. Софт от HID Global используется для регистрации биометрических параметров и аутентификации, выпуска идентификационных карт для банковского и финансового секторов. Также он широко применяется в операционных системах для микрочипов, а также в сфере Internet of Things (IoT).

Вместе с тем при всей разнонаправленности и плюсах разработок HID, в ПО имеется существенный минус – его цена. Впрочем, высокая стоимость программного обеспечения характерна практически для всех продуктов зарубежного производства.

**«Sigur»**

Владельцем данного бренда является российская компания «ПромАвтоматика», которая работает в сфере создания СКУД с 2006 года. За это время, бренд «Sigur» стал весьма известен на российском рынке систем для безопасности.

Программное обеспечение компании имеет модульную архитектуру и позволяет при необходимости увеличивать возможности системы через установку дополнительных модулей. Доступ ко всем модулям, установленным на сервере, осуществляется удаленно через веб-интерфейс. Решения, созданные компанией, позволяют вести учет рабочего времени, осуществлять защищенную идентификацию по смартфону, вести наблюдение за событиями на охраняемом объекте.

2 ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ

**2.1 Постановка задачи**

Разрабатываемое ПО «Система контроля доступа» предназначено для повышения уровня безопасности организации за счет автоматизации контроля передвижения сотрудников предприятия по средствам веб приложения с непосредственной интеграцией с базой данных.

Требуется создать программное обеспечение на основе базы данных, которое позволит отслеживать передвижения работников некой организации, а также предоставит пользователю возможность просмотра и редактирования информации таблиц базы данных с графическим отображением информации и последующим сохранением данных во внешней файл.

В процессе создания информационной системы были выделены следующие основные задачи:

1. анализ уже существующих программных продуктов;
2. разработка архитектуры информационной системы;
3. разработка структуры базы данных и автоматизированное наполнение ее тестовыми данными;
4. реализация многопользовательского интерфейса web-приложения с возможностью авторизации пользователей;
5. реализация функции вывода данных в табличном виде с функциями сортировки и фильтрации данных;
6. реализация функции добавления, редактирования и удаления записей базы данных;
7. реализация системы учета рабочего времени;
8. реализация функции вывода данных в графическом виде;
9. реализация функции сохранения данных во внешнем файле;
10. разработка сопроводительной документации.

**2.2 Определение требований пользователей к информационной системе**

Программное обеспечение позволяет работать со всеми таблицами базы данных, синхронизируя изменения в реальном времени. У пользователей системы есть три роли – сотрудник охраны, начальник охраны и системный администратор. Для авторизации в системе пользователь должен ввести свою электронную почту и пароль, авторизация производится по средствам проверки подлинности логина и пароля, а также роли пользователя на сайте. Функцию регистрации не целесообразно предусматривать в данной системе из соображений политики безопасности пользовательских данных.

У роли «сотрудник охраны» есть возможность просмотра таблицы «Доступ» (данных о передвижении сотрудников).

У роли «начальник охраны» есть возможность просмотра и редактирования записей всех таблиц базы данных, активация/деактивация пропусков, просмотр диаграммы и формирование отчетного документа с возможностью фильтрации таблиц, а также просмотр авторизованных пользователей с ролью «сотрудник охраны» со временем авторизации, с возможностью формирования отчетного документа.

У роли «системный администратор» есть возможность просмотра, редактирования, добавления и удаления записей всех таблиц БД, активация/деактивация пропусков, просмотр диаграммы и формирование отчетного документа с возможностью фильтрации таблиц, а также получение сведений о рабочем времени сотрудников организации и просмотр авторизованных пользователей сайта со временем авторизации и возможностью формирования отчетного документа.

Диаграмма носит статистический характер. Она показывает пользователю количество посетителей за каждый день месяца, а также количество посещений каждого отдельного сотрудника за выбранный период. Программное обеспечение позволяет сохранять следующие отчетные документы: файл таблиц базы данных, отчет о пользователях системы, отчет о рабочем времени. Отчетные документы представляют собой Excel-файлы. Данный формат был выбран для удобства дальнейшей работы с данными.

3 РАЗРАБОТКА И ОПИСАНИЕ БАЗЫ ДАННЫХ

**3.1 Проектирование концептуальной модели базы данных в сервисе ERDPlus**

ERDPlus – современный удобный сервис моделирования баз данных с простым и понятным пользователю интерфейсом и достаточно широким функционалом. Он позволяет создавать диаграммы связей сущностей, реляционные схемы, звездных схемы и инструкции SQL DDL. [13].

Для создания концептуальной модели БД необходимо выбрать тип диаграммы «Реляционная схема», после чего в конструкторе создать все необходимые таблицы, продумать названия столбцов таблиц с указанием типа данных и первичных ключей. Далее установить связи между таблицами и указать внешние ключи. После чего сохранить диаграмму. Результат разработки концептуальной схемы представлен на рисунке 3.1.

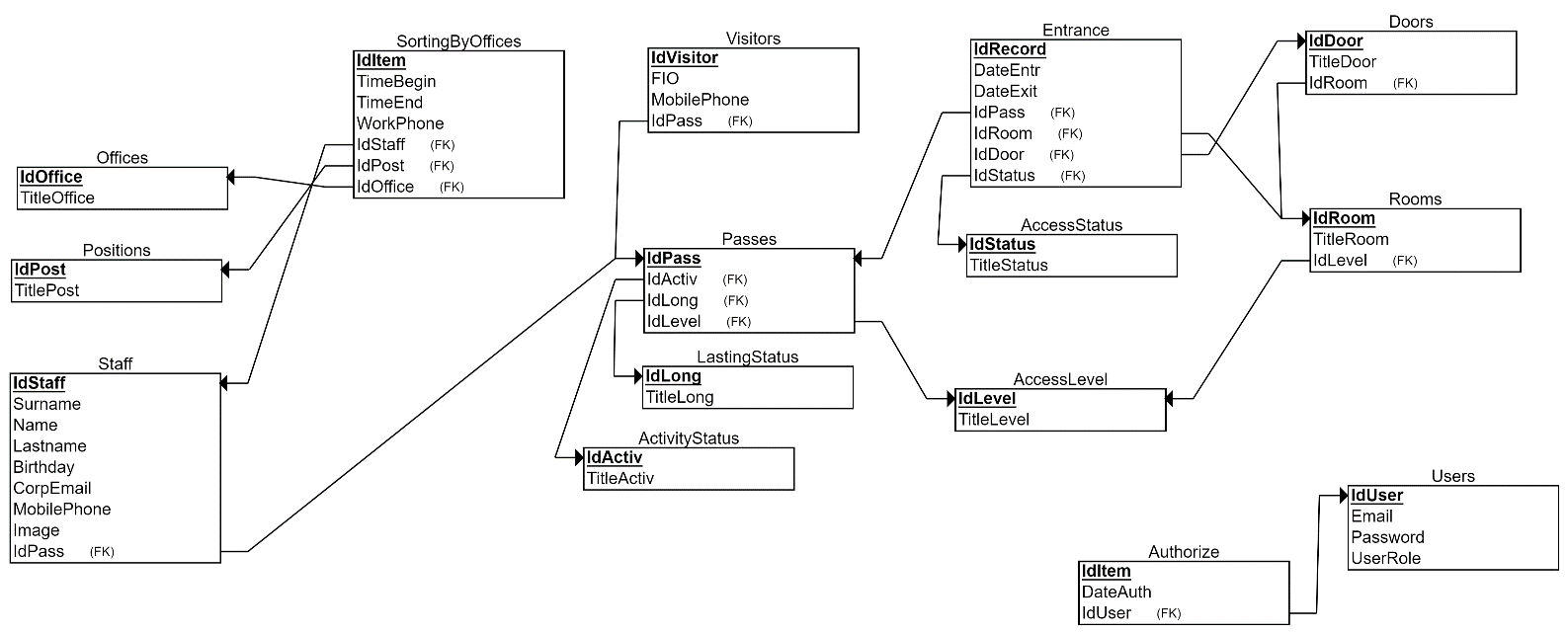


Рисунок 3.1 – Концептуальная схема базы данных

Сервис ERDPlus удобен тем, что автоматически генерирует код SQL запроса для создания разработанной БД. После выполнения такого запроса в SQL Management Studio автоматически создается база данных с заданными в схеме типами данных, первичными и внешними ключами и связями между таблицами.

**3.2 Разработка и описание даталогической модели базы данных на платформе Microsoft SQL Server 2014**

SQL Server Management Studio — утилита Microsoft SQL Server 2005 и более поздних версий для конфигурирования, управления и администрирования всех компонентов Microsoft SQL Server. Утилита включает скриптовый редактор и графическую программу, которая работает с объектами и настройками сервера. [12].

Даталогическая схема базы данных представляет собой модель, отражающую логические взаимосвязи между элементами данных безотносительно их содержания и физической организации.

Даталогическая модель формируется непосредственно в SQL Management Studio с помощью функции создания диаграммы. Так как база данных была создана автоматически (см. пункт 3.1), схема базы данных так же сформирована автоматически. Даталогическая схема БД представлена на рисунке 3.2.

Разработанная база данных «EntranceControlWeb» содержит в себе 12 справочных и 3 сводных (дочерних) таблицы данных. Структуры всех таблиц БД представлены в таблицах 3.1 – 3.15. В каждой таблице предусмотрен первичный ключ, как правило это ID записи. Данные из справочных таблиц в дочерние передаются по средствам внешних ключей и связей.

Таблица 3.1 – Структура справочной таблицы «AccessLevel»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Примечание |
| ID\_Level | int | PK |
| TitleLevel | nvarchar(20) |  |

Справочная таблица «AccessLevel» (Уровни доступа) содержит информацию обо всех возможных уровнях доступа сотрудников и помещений, первичный ключ – «ID\_Level».

Таблица 3.2 – Структура справочной таблицы «AccessStatus»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Примечание |
| ID\_Status | int | PK |
| TitleStatus | nvarchar(20) |  |

Справочная таблица «AccessStatus» (Статусы доступа) содержит в себе информацию о возможных статусах доступа сотрудников, первичный ключ – «ID\_Status».

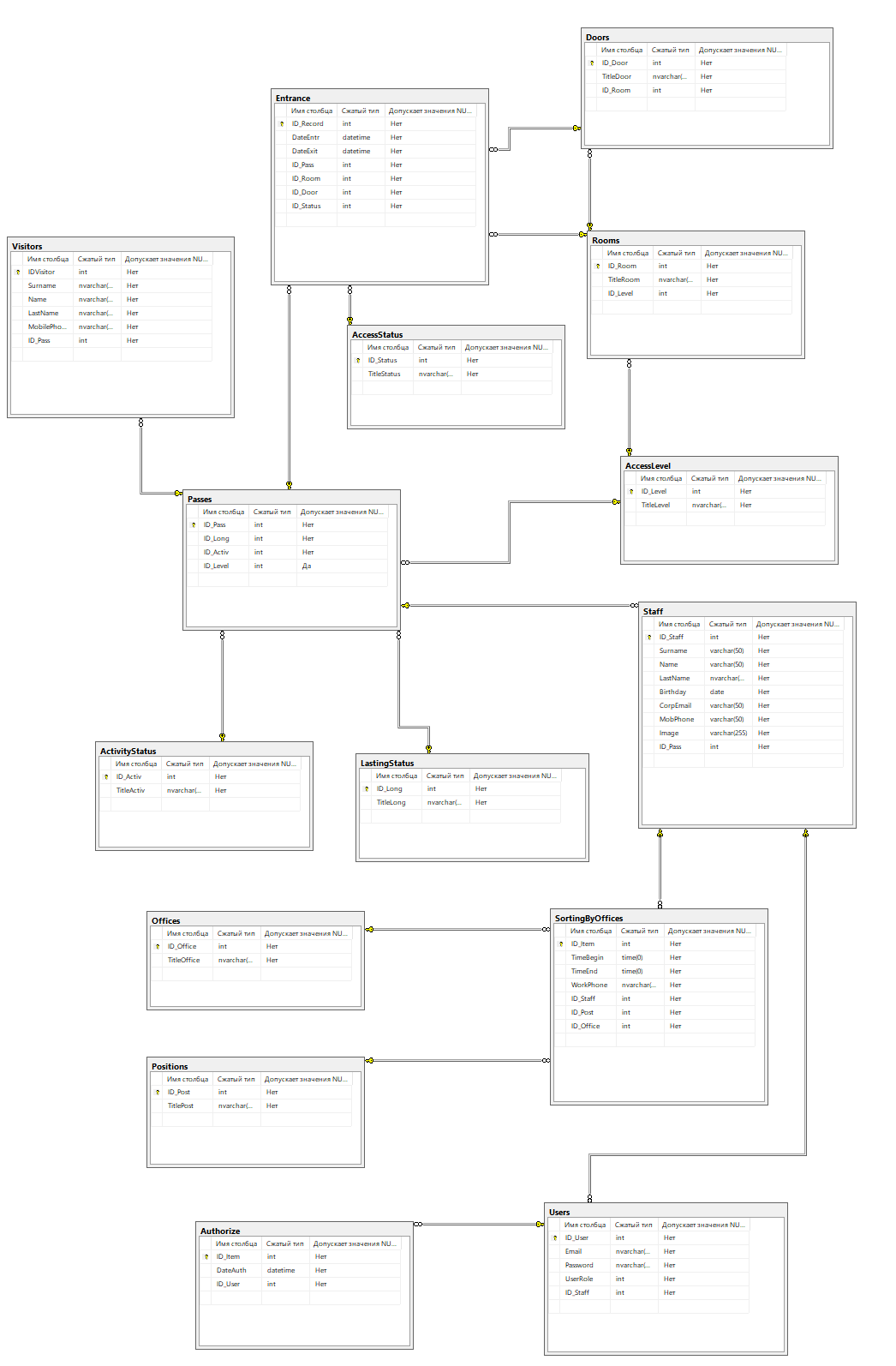


Рисунок 3.2 – Даталогическая схема базы данных

Таблица 3.3 – Структура справочной таблицы «ActivityStatus»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Примечание |
| ID\_Activ | int | PK |
| TitleActiv | nvarchar(20) |  |

Справочная таблица «ActivityStatus» (Статусы активности) содержит в себе информацию о возможных статусах активности пропусков, первичный ключ – «ID\_Activ».

Таблица 3.4 – Структура справочной таблицы «Doors»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Примечание |
| ID\_Door | int | PK |
| TitleDoor | nvarchar(20) |  |
| ID\_Room | int | FK |

Справочная таблица «Doors» (Типы турникетов) содержит информацию обо всех типах турникетов на входах в организацию, первичный ключ – «ID\_Door». Внешний ключ таблицы – «ID\_Room» – несет информацию в каком именно помещении установлен данных турникет.

Таблица 3.5 – Структура справочной таблицы «LastingStatus»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Примечание |
| ID\_Long | int | PK |
| TitleLong | nvarchar(20) |  |

Справочная таблица «LastingStatus» (Статусы длительности) содержит в себе информацию о возможных статусах длительности пропусков, первичный ключ – «ID\_Long».

Таблица 3.6 – Структура справочной таблицы «Offices»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Примечание |
| ID\_Office | int | PK |
| TitleOffice | nvarchar(20) |  |

Справочная таблица «Offices» (Отделы) содержит в себе информацию о всех отдлеах компании, первичный ключ – «ID\_Office».

Таблица 3.7 – Структура справочной таблицы «Positions»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Примечание |
| ID\_Post | int | PK |
| TitlePost | nvarchar(20) |  |

Справочная таблица «Positions» (Должности) содержит в себе информацию о всех должностях предприятия, первичный ключ – «ID\_Post».

Таблица 3.8 – Структура справочной таблицы «Rooms»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Примечание |
| ID\_Room | int | PK |
| Наименование столбца | Тип данных | Примечание |
| TitleRoom | nvarchar(20) |  |
| ID\_Level | int | FK |

Справочная таблица «Rooms» (Помещения) содержит в себе информацию обо всех помещениях на территории компании, первичный ключ – «ID\_Room». Внешний ключ таблицы («ID\_Level») несет информацию об уровне доступа соответствующего помещения.

Таблица 3.9 – Структура справочной таблицы «Staff»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Примечание |
| ID\_Staff | int | PK |
| Surname | nvarchar(20) |  |
| Name | nvarchar(20) |  |
| LastName | nvarchar(20) |  |
| Birthday | date |  |
| CorpEmail | nvarchar(20) |  |
| MobPhone | char(15) |  |
| Image | varchar(255) |  |
| ID\_Pass | int | FK |

Таблица «Staff» (Сотрудники) содержит в себе личную информацию о сотрудниках организации (Фамилия, Имя, Отчество, дата рождения), фото сотрудника, а также контактную информацию – корпоративную почту и номер мобильного телефона. Внешний ключ таблицы («ID\_Pass») несет информацию о номере пропуска сотрудника. Первичным ключом таблицы является столбец «ID\_Staff».

Таблица 3.10 – Структура справочной таблицы «Visitors»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Примечание |
| IDVisitor | int | PK |
| Surname | nvarchar(20) |  |
| Наименование столбца | Тип данных | Примечание |
| Name | nvarchar(20) |  |
| LastName | nvarchar(20) |  |
| MobilePhone | nvarchar(15) |  |
| ID\_Pass | int | FK |

Таблица «Visitors» (Посетители) содержит в себе личную информацию о посетителях организации (Фамилия, Имя, Отчество), а также контактную информацию – номер мобильного телефона. Внешний ключ таблицы («ID\_Pass») несет информацию о номере пропуска посетителя. Первичным ключом таблицы является столбец «IDVisitor».

Таблица 3.11 – Структура дочерней таблицы «Entrance»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Примечание |
| ID\_Record | int | PК |
| DateEntr | date |  |
| DateExit | date |  |
| ID\_Pass | int | FK |
| ID\_Room | int | FK |
| ID\_Door | int | FK |
| ID\_Status | int | FK |

Дочерняя таблица «Entrance» (Досуп) содержит информацию о времени входа и выхода сотрудника, помещении, типе турникета на входе в помещение и статусе доступа. Первичный ключ таблицы – «ID\_Record», внешние ключи: «ID\_Pass», «ID\_Room», «ID\_Door», «ID\_Status».

Таблица 3.12 – Структура дочерней таблицы «Passes»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Примечание |
| ID\_Pass | int | PК |
| ID\_Long | int | FK |
| ID\_Activ | int | FK |
| Наименование столбца | Тип данных | Примечание |
| ID\_Level | int | FK |

Дочерняя таблица «Passes» (Пропуска) содержит информацию о номере, длительности, активности и уровне доступа пропуска. Первичный ключ таблицы – «ID\_Pass», внешние ключи: «ID\_Long», «ID\_Activ», «ID\_

Таблица 3.13 – Структура дочерней таблицы «SortingByOffices»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Примечание |
| ID\_Item | int | PK |
| TimeBegin | time(0) |  |
| TimeEnd | time(0) |  |
| WorkPhone | nvarchar(15) |  |
| ID\_Staff | int | FK |
| ID\_Post | int | FK |
| ID\_Office | int | FK |

Дочерняя таблица «SortingByOffices» (Распределение по отделам) содержит информацию о времени начала и завершения работы сотрудника (в дальнейшем будет добавлена функция редактирования рабочего времени), его ФИО, рабочий телефон, должность и отдел. Первичный ключ таблицы отсутствует, внешние ключи: «ID\_Staff», «ID\_Post», «ID\_Office».

База данных содержит две дополнительные таблицы «Users» и «Authorize» (структуры представлены в таблицах 3.14 – 3.15). Таблица «Users» нужна для хранения логинов, паролей и роли пользователя на сайте, а таблица «Authorize» – для вывода информации о дате и времени авторизации пользователя на сайте.

Таблица 3.14– Структура таблицы «Users»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Примечание |
| ID\_User | int | PK |
| Email | nvarchar(50) |  |
| Password | nvarchar(50) |  |
| Наименование столбца | Тип данных | Примечание |
| UserRole | int |  |

Таблица «Users» (Пользователи) содержит информацию об Email-адресе зарегистрированного пользователя, его пароль и роль на сайте. Первичный ключ таблицы – «ID\_User».

Таблица 3.15 – Структура таблицы «Authorize»

|  |  |  |
| --- | --- | --- |
| Наименование столбца | Тип данных | Примечание |
| DateAuth | date |  |
| ID\_User | int | FK |

Таблица «Authorize» содержит информацию о дате входа на сайт конкретного пользователя. Таблица не содержит первичных ключей, внешний ключ таблицы – «ID\_User».

**3.3 Автоматизация наполнения базы тестовыми данными   
с использованием сервиса Microsoft SQL Server Integration Services**

Для заполнения БД тестовыми данными был создан Excel-файл версии Microsoft Excel 97-2003 (для лучшей совместимости с Microsoft SQL Server Integration Services и SQL Management Studio 2014), в который с помощью задачи экспорта данных были перенесены шаблоны всех таблиц БД. Затем файл был вручную наполнен тестовыми данными. Пример наполнения тестовыми данными таблицы «Распределение по отделам» представлен на рисунке 3.3.



Рисунок 3.3 – Наполнение таблицы «Распределение по отделам» тестовыми данными

Затем в среду разработки Visual Studio 2019 был установлен пакет Microsoft SQL Server Integration Service и создан проект Integration Service Project. Работа в сервисе осуществляется по средствам добавления в окно конструктора соответствующих блоков задач из раздела «Инструменты». Первый блок – «Задача выполнение SQL» – выполняет функцию подключения к серверу и базе данных, а также очистку таблиц базы с помощью специального SQL-скрипта от возможных не нужных записей во избежание возникновения ошибок при переносе данных. Последующие блоки – «Задача потока данных» – отвечают за заполнение данными соответствующих таблиц БД. Блоки «Задача потока данных» состоят из функциональных модулей «Источник Excel» и «Назначение ADO NET». В модуле источника происходит подключение к Excel-файлу с тестовыми данными и выбор необходимых столбцов, а модуль «назначение» отвечает за подключение к соответствующей таблице базы данных и корректный перенос данных.

При наличии столбцов с типом данных DateTime, необходимо выполнить промежуточную операцию преобразования данных для корректного переноса их в таблицу базы данных. Данная операция выполняется с помощью дополнительного модуля «Преобразование данных», в котором формат данных файла Excel конвертируется в формат данных SQL Management Studio. Результат работы в сервисе Microsoft SQL Server Integration Service представлен на рисунке 3.4.



Рисунок 3.4 – Успешное выполнение операции наполнения данными   
в среде Visual Studio 2019

4 РАЗРАБОТКА И ОПИСАНИЕ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

**4.1 Управление версиями проекта на основе сервиса GitHub**

Системы контроля версий позволяют разработчикам сохранять все изменения, внесённые в код. Поэтому в случае возникновения неисправных ошибок кода разработчик может откатить код до рабочего состояния. Системы контроля версий также дают возможность нескольким разработчикам работать над одним проектом и сохранять внесённые изменения.

Данные возможности полностью предоставляет система управления версиями GitHub – сервис онлайн-хостинга репозиториев, обладающий всеми функциями распределённого контроля версий и функциональностью управления исходным кодом. Данный сервис даёт разработчикам возможность сохранять их код онлайн, а затем взаимодействовать с другими разработчиками в разных проектах. [14]. Работа с системой осуществлялась посредствам графического интерфейса настольного приложения GitHub Desktop (рисунок 4.1).

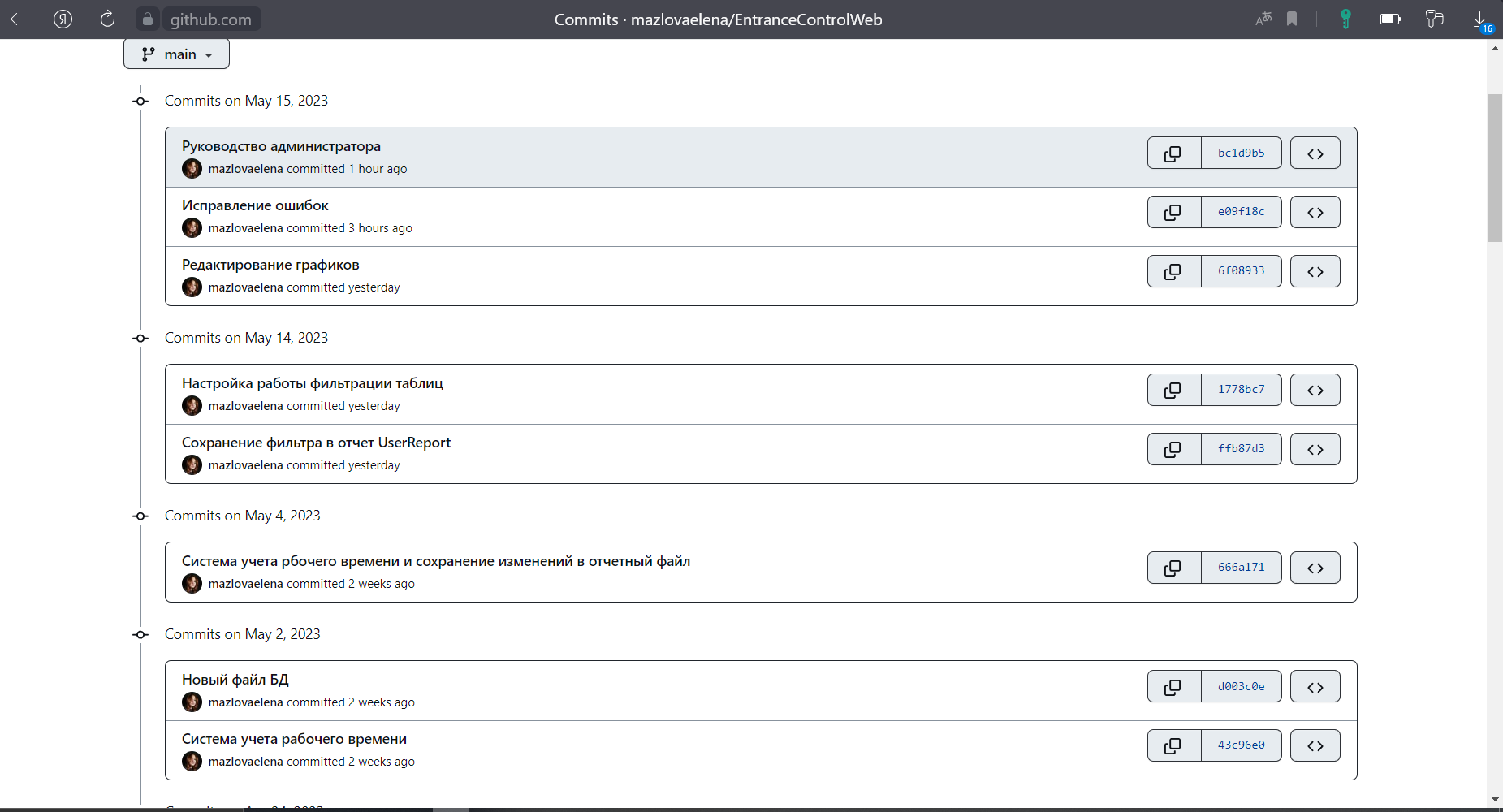


Рисунок 4.1 – Репозиторий проекта в сервисе GitHub

Ссылка на репозиторий в системе GitHub: <https://github.com/mazlovaelena/EntranceControlWeb>

**4.2 Разработка клиентского приложения**

Один из важнейших начальных этапов работы – разработка архитектуры системы, так как грамотно построенная архитектура исключает возникновение множества ошибок в процессе работы и даёт правильное представление внутренних процессов системы и взаимодействия с ней конечного пользователя.

Взаимодействие конечного пользователя с системой осуществляется посредствам web-интерфейса. В соответствии с требованиями, в нём предусмотрены функция авторизации пользователя, возможность просмотра, редактирования, добавления и удаления данных, функции фильтрации и сортировки данных таблиц, просмотр статистической диаграммы и сведений о рабочем времени сотрудников, а также формирование отчёта с функцией сохранения во внешний файл. К web-интерфейсу подключена база данных, что гарантирует автоматическое обновление данных как на сервере БД, так и в клиентской части системы. Общий вид архитектуры системы представлен на рисунке 4.2.

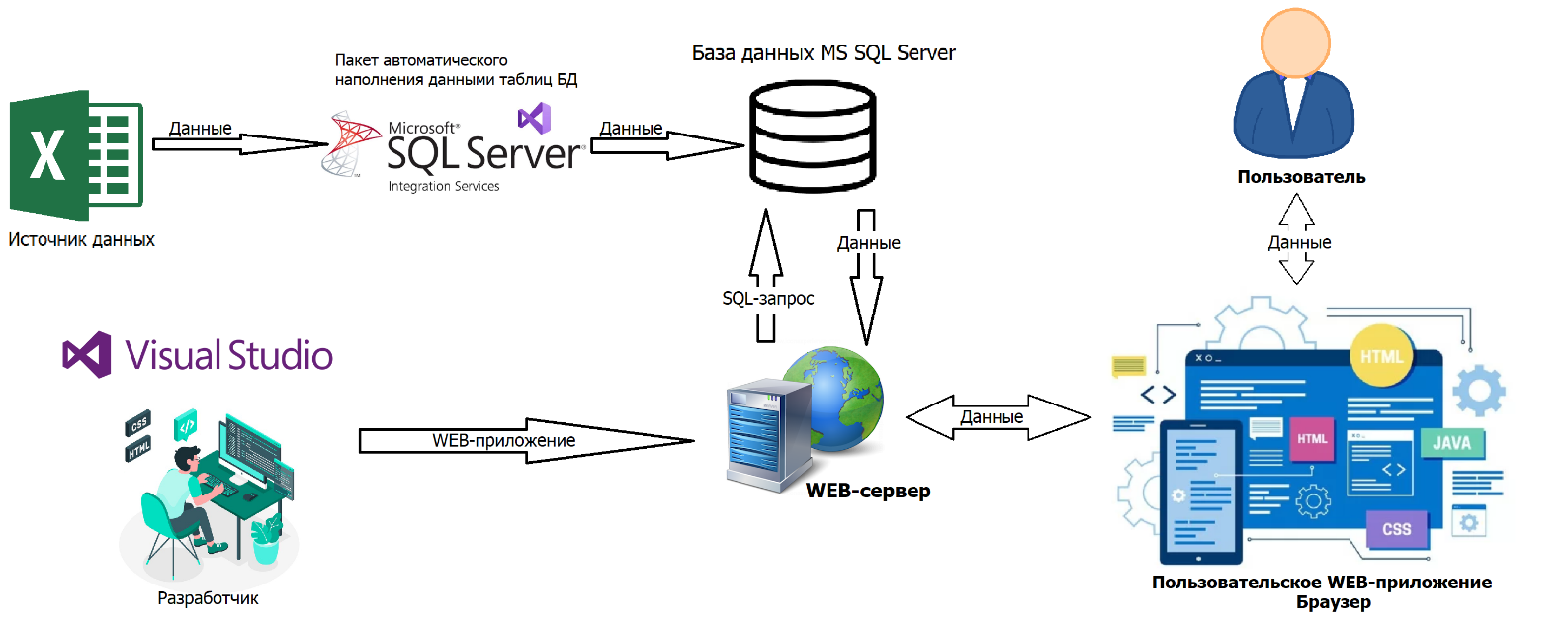


Рисунок 4.2 – Архитектура информационной системы

Для разработки клиентского приложения был создан проект ASP.NET Core Web Application (MVC) в среде разработки Visual Studio 2019 на платформе .Net 5.0. Проект подразумевает разделение написания приложения на 3 функциональные составляющие: Модель-Представление-Контроллер. Модель (Model) отвечает за данные, а также определяет структуру приложения. Представление (View) отвечает за взаимодействие с пользователем, то есть код компонента определяет внешний вид приложения и способы его использования. Контроллер (Controller) отвечает за связь между model и view – определяет, как сайт реагирует на действия пользователя.

Процесс разработки приложения начался с создания дизайна пользовательских страниц. Для этого были использованы дополнительные языки программирования CSS, HTML и JavaScript. CSS отвечает за дизайн страницы, HTML определяет разметку страницы и расположение элементов, а JavaScript необходим для отображения динамических элементов страницы, например, всплывающих окон.

Сначала была реализована функция авторизации пользователей по ролям. С этой целью необходимо подключить базу данных к проекту для корректного обмена данными. Для этого были использованы расширения Microsoft.EntityFrameworkCore.SqlServer и Microsoft.EntityFrameworkCore.SqlServer.Tools. Подключение базы данных выполнялось с помощью ввода в консоль диспетчера пакетов строки запроса подключения и создания строк подключения в файле конфигурации appsettings.json и файле запуска программы Startup.cs. После подключения БД в проекте появились классы для каждой из таблиц базы (рисунок 4.3) и класс-контекст, через который происходит обращение к таблицам БД.

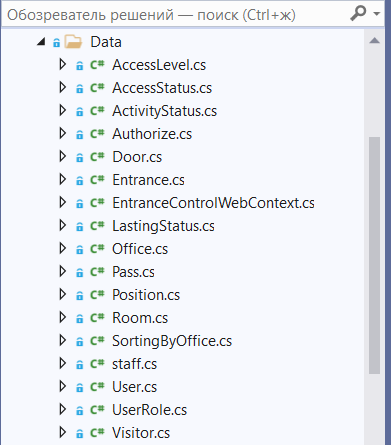


Рисунок 4.3 – Классы таблиц базы данных

Наличие ролей пользователей на сайте подразумевает создание дополнительного базового класса UserRole.cs, содержащего в себе определение значений для каждой роли на сайте.

Далее был создан отдельный контроллер, в котором прописан сам функционал авторизации пользователей. Авторизация происходит по средствам куки (небольшого фрагмента данных, отправляемого веб-сервером и хранимого на компьютере пользователя) и claims – набора данных, которые шифруются и добавляются в куки. Когда пользователь вводит данные для авторизации в системе, контроллер проверяет наличие такого пользователя в базе данных, после чего отправляет ему аутентификационные куки, которые при последующих запросах будут передаваться обратно на сервер, десериализоваться и использоваться для аутентификации пользователя.

Для корректной работы приложения необходимо преобразовать все записи таблиц БД в листы моделей представления и затем вывести их на форму. Преобразование данных осуществлялось в контроллере, а функция вывода строк была написана в представлениях с использованием цикла foreach. Фрагменты листинга кода модели и контроллера представлены на рисунке 4.4.

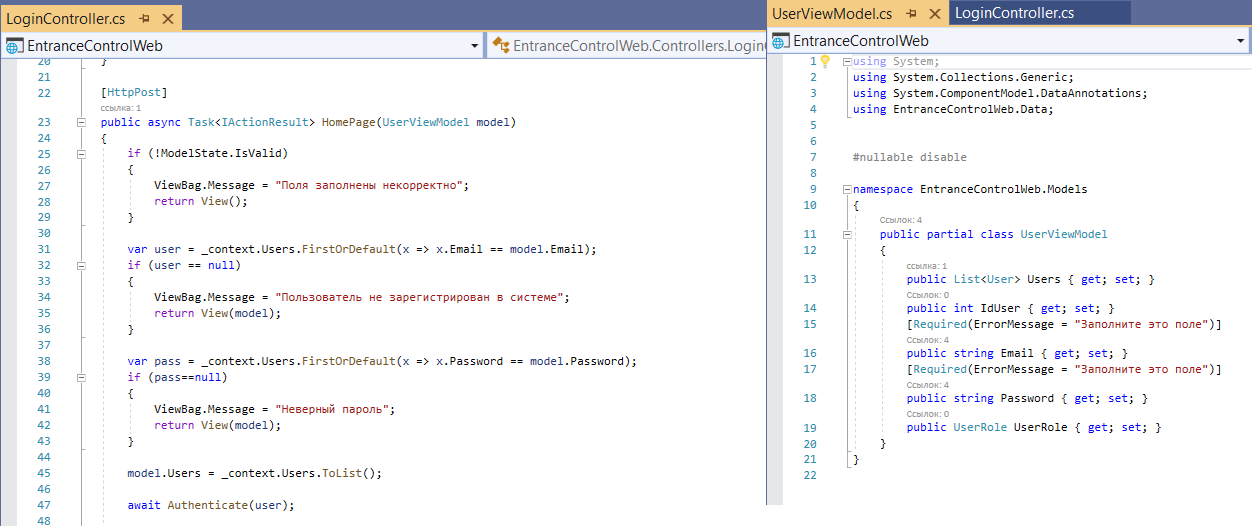


Рисунок 4.4 – Фрагменты кода приложения (слева контроллер LoginController.cs, справа модель преставления UserViewModel.cs)

Во избежание возможных ошибок ввода данных была предусмотрена валидация данных, предупреждающая пользователя о некорректном вводе или об отправке пустой формы. Для этого в модель представления подключается дополнительное пространство имен «System.ComponentModel.DataAnnotations». Для каждого поля, где нужна проверка вводимых значений, указывается соответствующий атрибут, например, обязательное заполнение поля или маска ввода, ограничивающая ввод букв в поля ввода цифр или проверяющая корректность ввода email-адреса. Далее в контроллере прописывается условие проверки валидности модели. Если модель не прошла проверку, пользователю выводится сообщение об ошибке. Пример работы валидации данных представлен на рисунке 4.5.

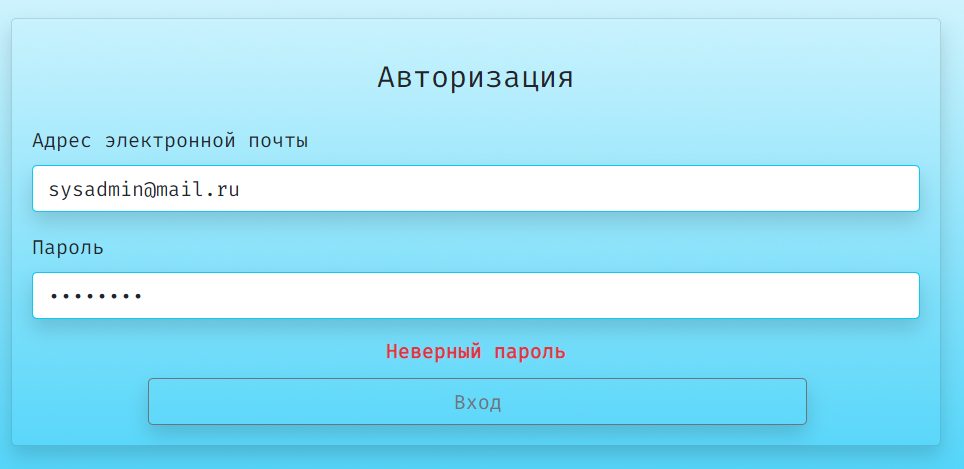


Рисунок 4.5 – Пример работы валидации данных при вводе пользователем неправильного пароля

Самой удобной и понятной для пользователя формой представления данных из базы является табличная форма визуализации данных. Для вывода всех таблиц на пользовательские формы также необходимы представление, модель и контроллер. Для пользователей с ролями «начальник охраны» и «системный администратор» предусмотрены возможности редактирования, добавления и удаления информации. Данные функции подразумевают создание дополнительных форм ввода данных.

Логика для сохранения изменений в БД прописана в контроллере и связана с пользовательскими страницами посредствам моделей. В случае создания или изменения записи, в переменную базы данных передаются значения соответствующих полей модели, в которых содержатся измененные пользователем данные. При удалении записи сначала происходит поиск выбранной записи в таблице базы, а затем ее удаление. Данные функции реализованы с помощью LINQ-запросов. Удаление данных справочных таблиц может повлечь за собой ошибки в данных дочерних таблиц. Во избежание подобных ситуаций, необходимо настроить спецификацию каскадного удаления записей в SQL Management Studio. Листинг программного кода контроллера HomeController.cs представлен в приложении А.

Для удобства работы с данными в системе были предусмотрены сортировка и фильтрация данных таблиц. Реализация функции сортировки подразумевает создание строковой переменной в методе отображения страницы. Далее создается переменная поиска значений в таблице базы данных с помощью select запроса. Затем, в конструкции switch-case прописывается логика сортировки записей. Например, если нужно отсортировать записи в порядке возрастания, прописывается соответствующий LINQ-запрос. Завершающий этап – присвоение ViewBag значений сортировки и добавление тэг-хелпера на представление страницы. Фрагменты листинга кода контроллера и представления представлены на рисунке 4.6.



Рисунок 4.6 – Фрагмент листинга программного кода: слева – контроллер HomeController.cs, справа – представление Rooms.cshtml

Реализация функции фильтрации записей подразумевает создание дополнительных раскрывающихся списков с параметрами поиска. Для этого в моделях представлений были созданы дополнительные листы интерфейса IEnumerable<SelectListItem>, данные в которые передаются напрямую из базы данных в контроллере. Листы имеют 2 свойства: значение и подпись данных. В качестве значений выступают ID передаваемых записей. Затем в каждом представлении был создан раскрывающийся список DropDownListFor и в методе отображения страницы каждому раскрывающемуся списку была присвоена числовая переменная. Далее для каждой такой переменной было прописано условие проверки переменной, которое проверяет ее валидность (она не должна быть равна нулю или пустому значению). После чего реализуется поиск с помощью LINQ-запроса и функция вывода найденной информации на представление. То есть, когда пользователь выбирает некий параметр для фильтрации, в переменную контроллера передается выбранное значение, происходит фильтрация данных и пользователю выводится результат фильтрации. Для некоторых параметров фильтрации (например, поиск по дате или фамилии) необходимы строковые переменные и поля текстового ввода. В таком случае, условие поиска сначала проверяет валидность строки, то есть отсутствие пустых строк, а затем ищет и выводит на пользовательскую форму данные, содержащие искомый запрос. Фрагменты листинга кода контроллера и представления представлены на рисунке 4.7.



Рисунок 4.7 – Фрагмент листинга программного кода: слева – контроллер HomeController.cs, справа – представление Entrance.cshtml

Для пользователей системы с ролями «начальник охраны» и «системный администратор» предусмотрен личный кабинет. Навигация по функциям личного кабинета реализована с помощью простого кнопочного интерфейса. В личном кабинете «начальнику охраны» доступен просмотр диаграммы посещаемости, пользователей сайта с ролью «сотрудник» и возможность сохранения файла базы данных. У роли «системный администратор» функционал работы шире. Помимо аналогичных функций «начальника охраны», ему так же доступны редактирование данных справочных таблиц, просмотр и сохранение отчета о рабочем времени сотрудников и просмотр всех пользователей сайта.

Одна из ключевых функций данной системы – учет рабочего времени сотрудников. Для ее реализации была создана дополнительная модель представления WorkTimeViewModel, содержащая в себе 3 класса параметров и 4 листа для хранения полученных данных. Расчет фактического рабочего времени сотрудников производится исходя из записей в таблице «Доступ». Для каждого ID сотрудника выбираются все даты входа и выхода, вычитаются и передаются в лист. Получается, что для каждого ID сотрудника создается лист со значениями времени работы по дням. Далее данные этих листов суммируются для получения итогового значения рабочих часов за период и записываются в другой итоговый лист. Для вычисления времени работы по производственному плану, для каждого ID сотрудника берется время начала и завершения смены из таблицы «Распределение по отделам», определяется их разность (количество часов в рабочей смене) и итог умножается на 20 рабочих дней в месяце. После чего данные записываются в лист. Для корректного вывода данных на представления, два итоговых листа со временем работы по факту и плановым временем работы объединяются в один конечный лист и выводятся в виде таблицы на пользовательской странице. Весь вышеописанный функционал был написан с помощью LINQ-запросов и цикла перебора записей foreach. Фрагмент листинга кода контроллера представлен на рисунке 4.8.

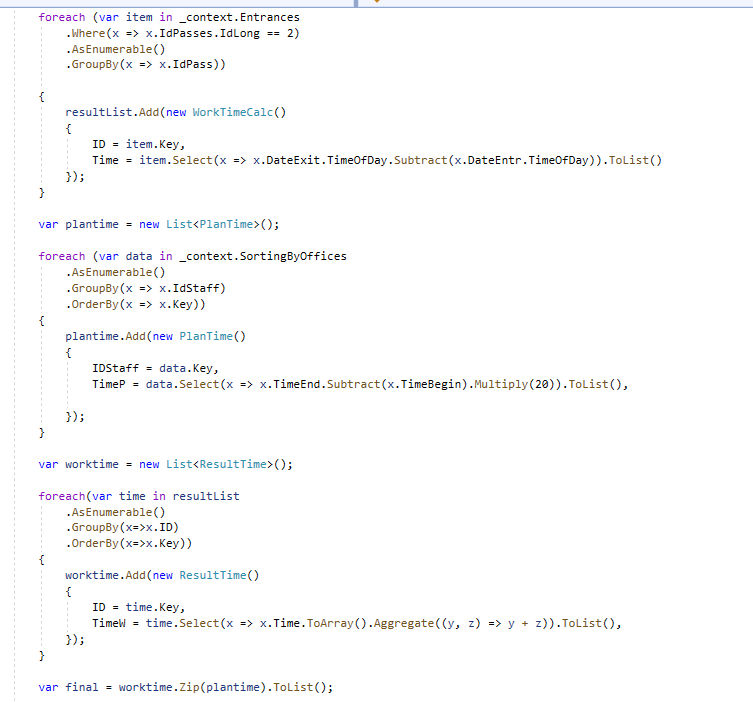


Рисунок 4.8 – Фрагмент листинга кода контроллера CabinetController.cs

Система содержит 2 статистических графика количества посетителей за каждый день месяца и количество посещений каждого сотрудника за выбранный период. График возможно отфильтровать по каждому сотруднику отдельно. С этой целью в контроллере было написано 2 LINQ-запроса, подсчитывающих количество записей за каждое число месяца и количество записей для каждого сотрудника отдельно, после чего данные сохраняются в лист для удобства дальнейшей работы с ними. Функция фильтрации данных реализована подобно фильтрации в таблицах. Для создания диаграммы на пользовательской странице была использована библиотека Chart.js. На пользовательскую форму был добавлен элемент canvas и прописан скрипт, содержащий тип диаграммы, её дизайн, подписи осей и название. Данные из листов, созданных ранее, передаются с помощью Json сериализации. Фрагмент кода страницы «Chart.cshtml» представлен на рисунке 4.9.

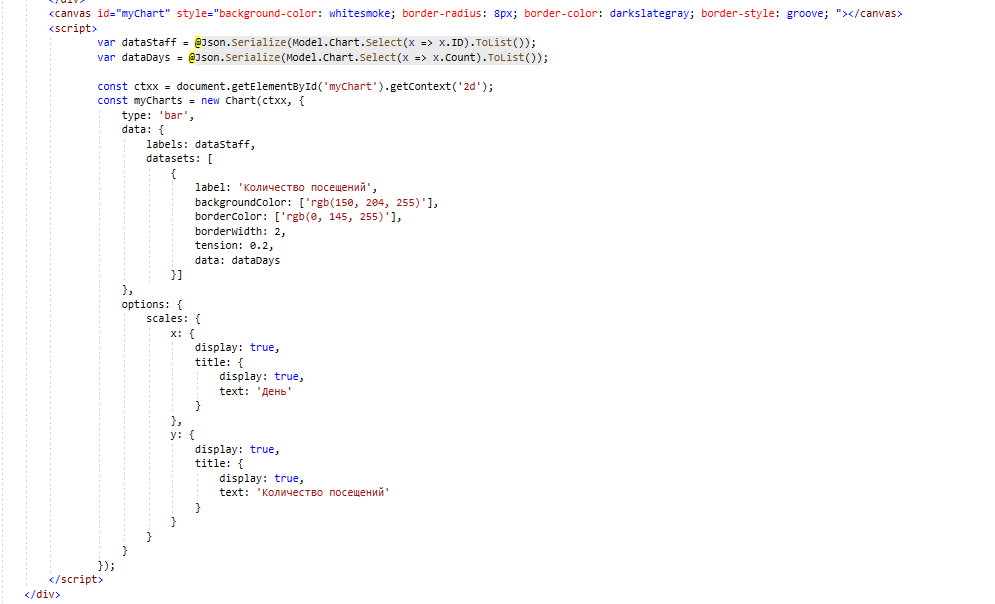


Рисунок 4.9 – Фрагмент кода страницы вывода диаграммы «Chart.cshtml»

Одной из функций данной системы является экспорт данных таблиц базы во внешний файл. Табличные данные наиболее удобно экспортировать в Excel-файл. Для этого в проект было подключено расширение EPPlus, позволяющее переносить данные непосредственно из базы и сохранять их во внешнем файле. Так же была предусмотрена возможность выбора сохраняемых таблиц. То есть, пользователь может сохранить как полностью все таблицы базы, так и выбрать только необходимые. Для этих целей на пользовательскую страницу были добавлены checkbox’ы, а в контроллере присвоены соответствующие переменные для каждого checkbox’а. Когда пользователь выбирает галочкой нужную таблицу, в контроллере выполняется условие добавления выбранной таблицы в лист Excel. После нажатия кнопки сохранения файл скачивается на ПК пользователя. Для случая сохранения данных всех таблиц БД был создан отдельный метод, сохраняющий все таблицы. Фрагмент кода контроллера представлен на рисунке 4.10.

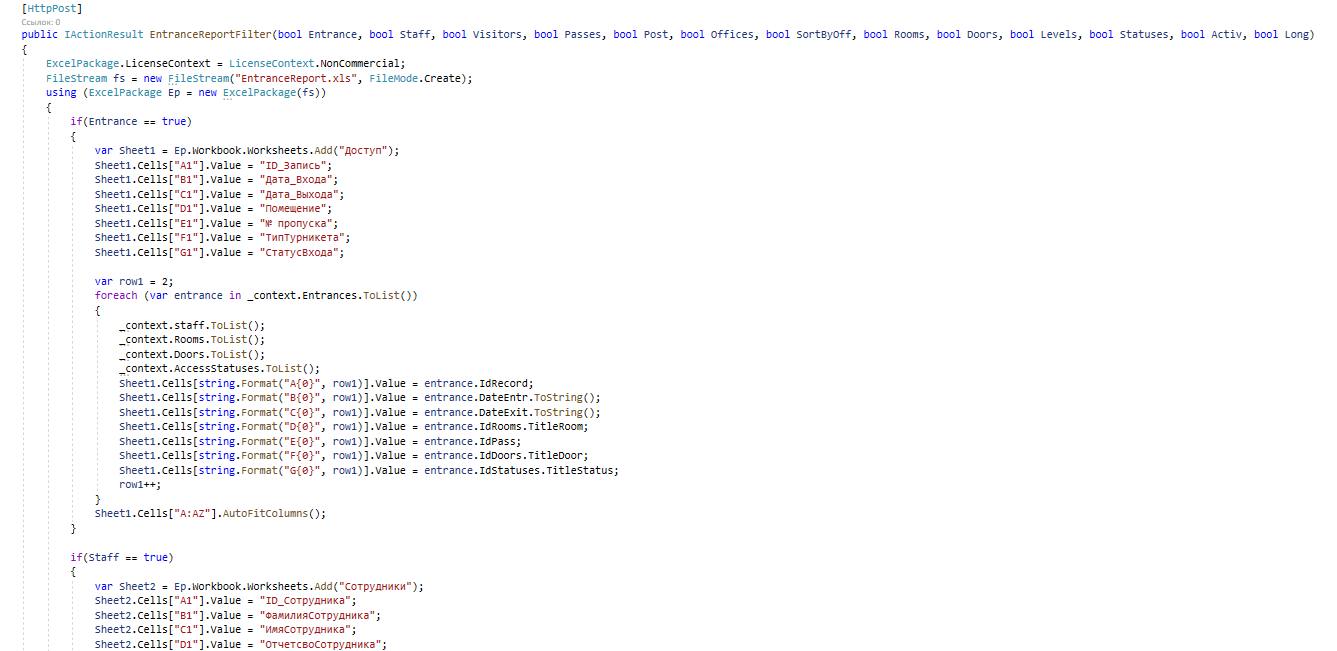


Рисунок 4.10 – Фрагмент кода контроллера CabinetContoller.cs

Просмотр авторизованных пользователей сайта реализован аналогично выводу остальных табличных данных системы, в данном случае также предусмотрена функция поиска записей по email-адресу. Пользователь может сохранить как всю таблицу целиком, так и только отфильтрованную версию. Реализация функции сохранения фильтра записей таблицы во внешний файл была выполнена с использованием файлов куки. После ввода поискового запроса на странице, результаты фильтрации записываются в куки, которые хранятся на пользовательском компьютере до завершения сессии. А при нажатии на кнопку сохранения отчета с фильтром, в ячейки Excel-файла записываются значения из куки текущей сессии. Фрагмент листинга кода котроллера представлен на рисунке 4.11.

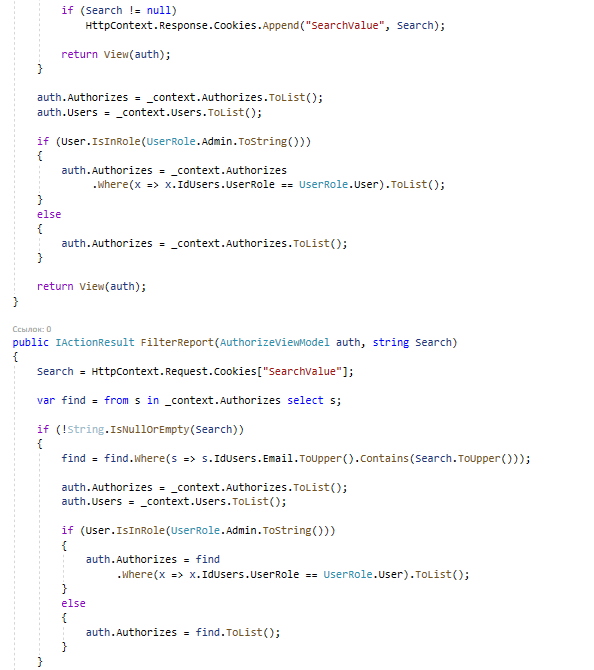


Рисунок 4.11 – Фрагмент листинга кода контроллера CabinetContoller.cs

**4.3 Описание работы приложения**

При запуске приложения пользователя встречает главная страница (рисунок 4.12) с формой авторизации пользователя. Для дальнейшей работы в системе обязательно авторизацироваться.

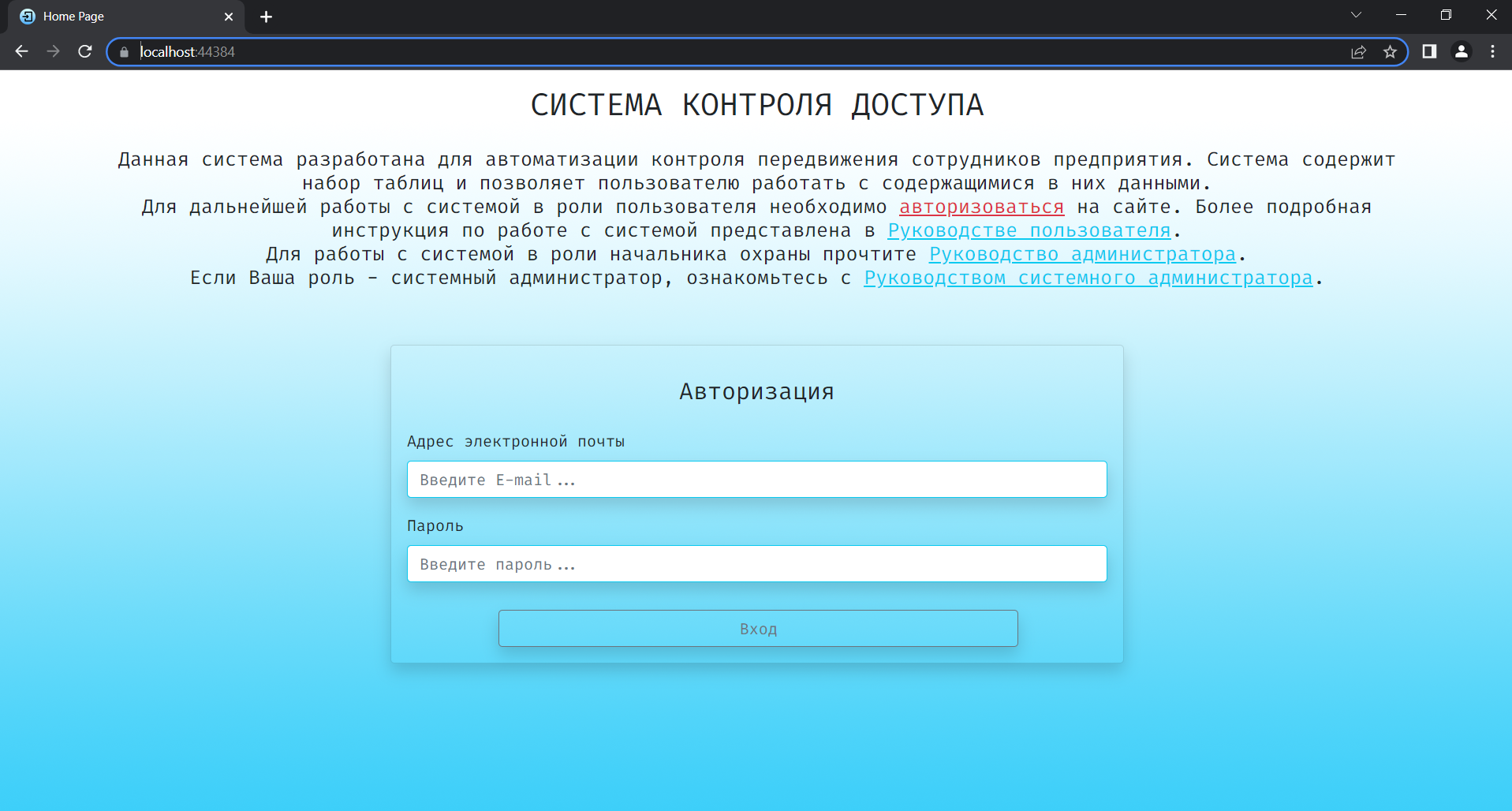


Рисунок 4.12 – Главная страница системы

На главной странице есть ссылки на руководства по использованию системы для всех пользователей системы. Пример руководства «начальника охраны» представлен на рисунке 4.13. Руководство по использованию системы пользователя с ролью «системный администратор» представлено в приложении Б.

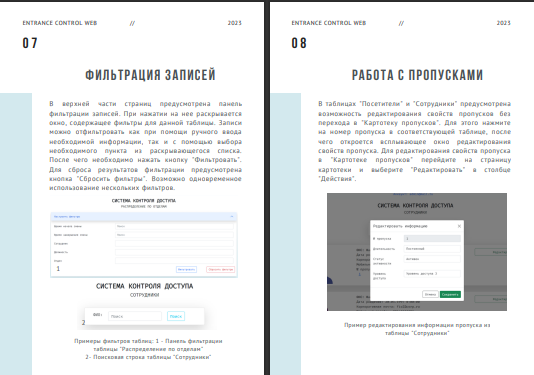


Рисунок 4.13 – Руководство начальника охраны

После авторизации в системе на панели навигации вверху страницы появится email, под которым пользователь вошел на сайт. Навигация в системе реализована с помощью удобной динамической навигационной панели (рисунок 4.14). Ее содержимое меняется в зависимости от роли пользователя.

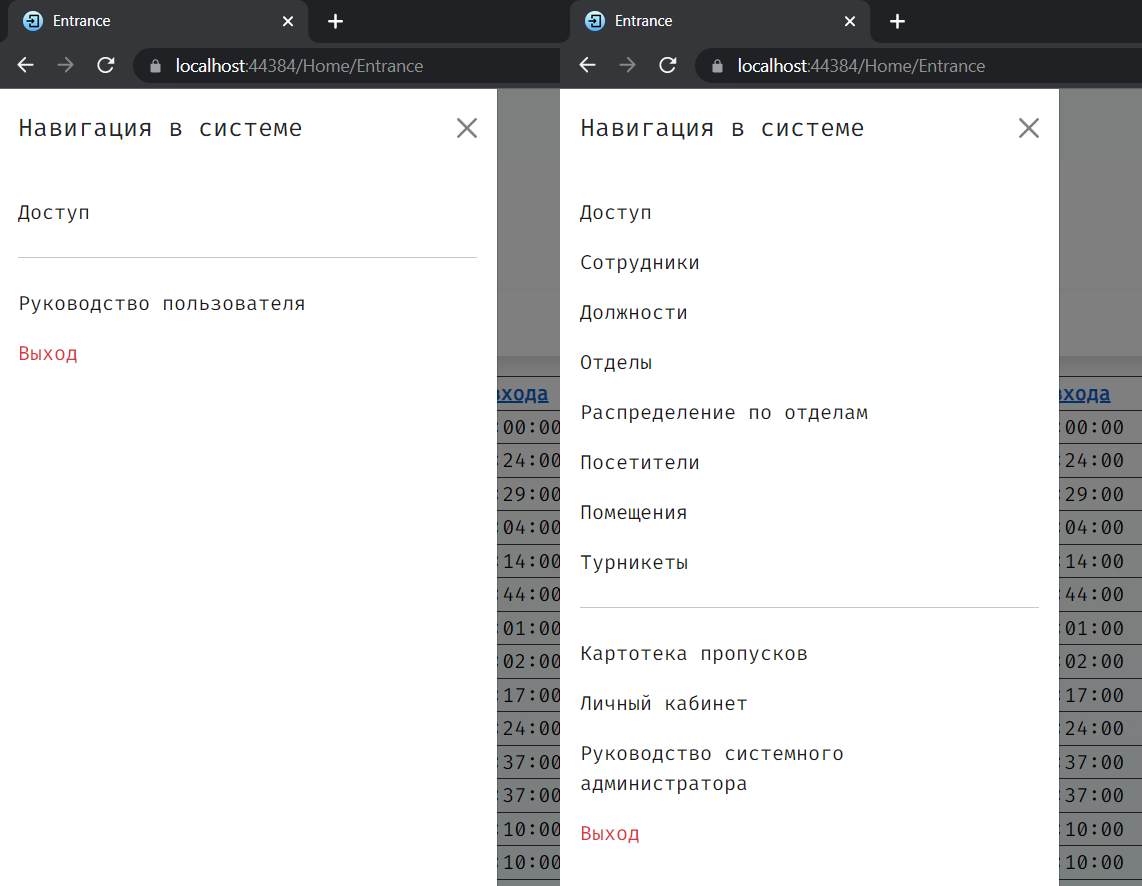


Рисунок 4.14 – Навигационная панель системы (справа – для пользователя с ролью «сотрудник охраны», слева – для пользователя с ролью «начальник охраны»)

У пользователя с ролью «сотрудник охраны» ограниченные права на сайте: он может только просматривать таблицу «Доступ».

При авторизации в системе с ролью «начальник охраны» у пользователя появляется доступ ко всем таблицам, с возможностью фильтрации (рисунок 4.15) и сортировки записей (рисунок 4.16), а также возможностью редактирования информации (рисунок 4.17). У пользователя дополнительно есть доступ к личному кабинету (рисунок 4.18) с функциями просмотра диаграммы, просмотра пользователей сайта с ролью «сотрудник охраны» и сохранения данных во внешний файл.

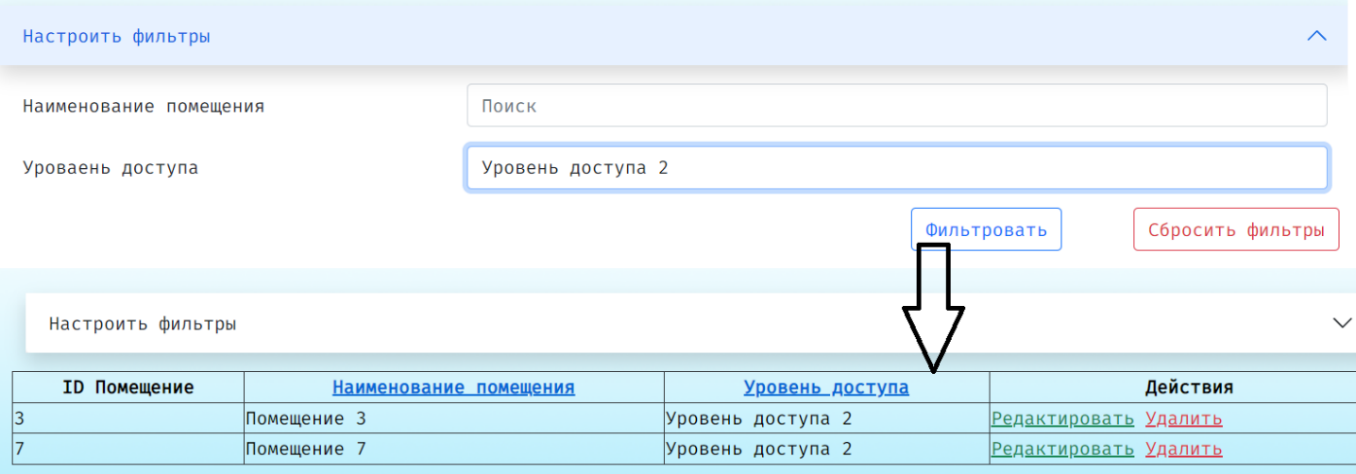


Рисунок 4.15 – Результат фильтрации записей таблицы «Помещения»

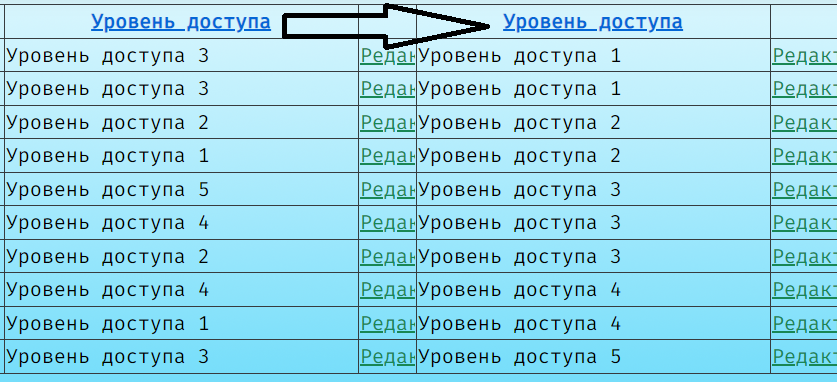


Рисунок 4.16 – Результат сортировки записей в столбце «Уровень доступа» таблицы «Помещения»

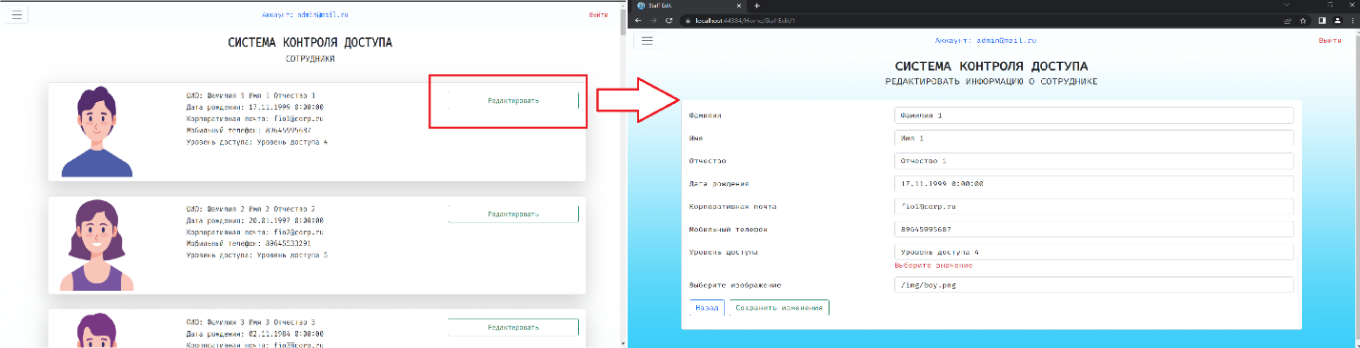


Рисунок 4.17 – Редактирование записи таблицы «Сотрудники»

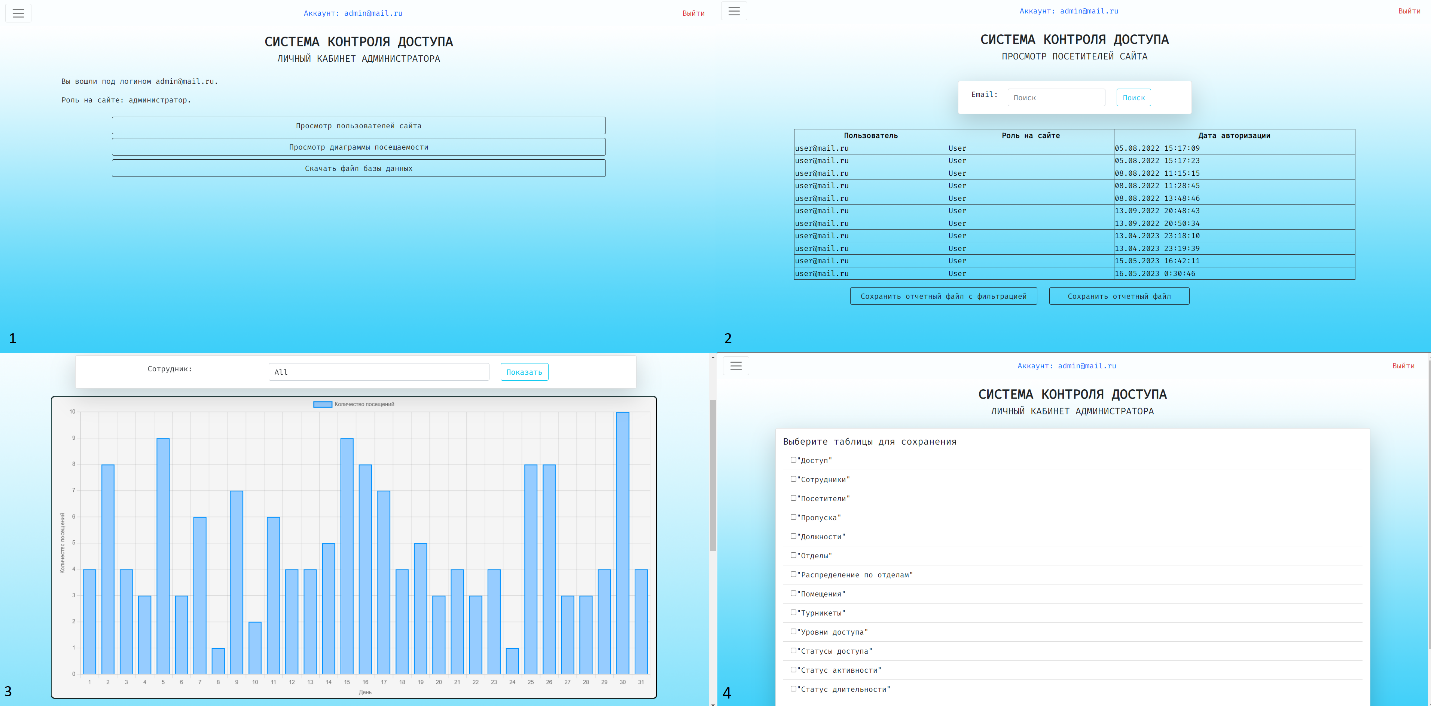


Рисунок 4.18 – Личный кабинет начальника охраны: 1 – функции личного кабинета; 2 – просмотр посетителей сайта; 3 – просмотр диаграммы; 4 – сохранение отчетного файла

Для пользователя с ролью «системный администратор» в приложении помимо функции редактирования, предусмотрены функции добавления и удаления новых записей во все таблицы (рисунок 4.19), а также доступ к личному кабинету с функциями просмотра диаграммы, сохранение файла базы данных, редактирование данных справочных таблиц, просмотр и сохранение отчета о рабочем времени сотрудников (рисунок 4.20) и просмотр всех пользователей сайта.

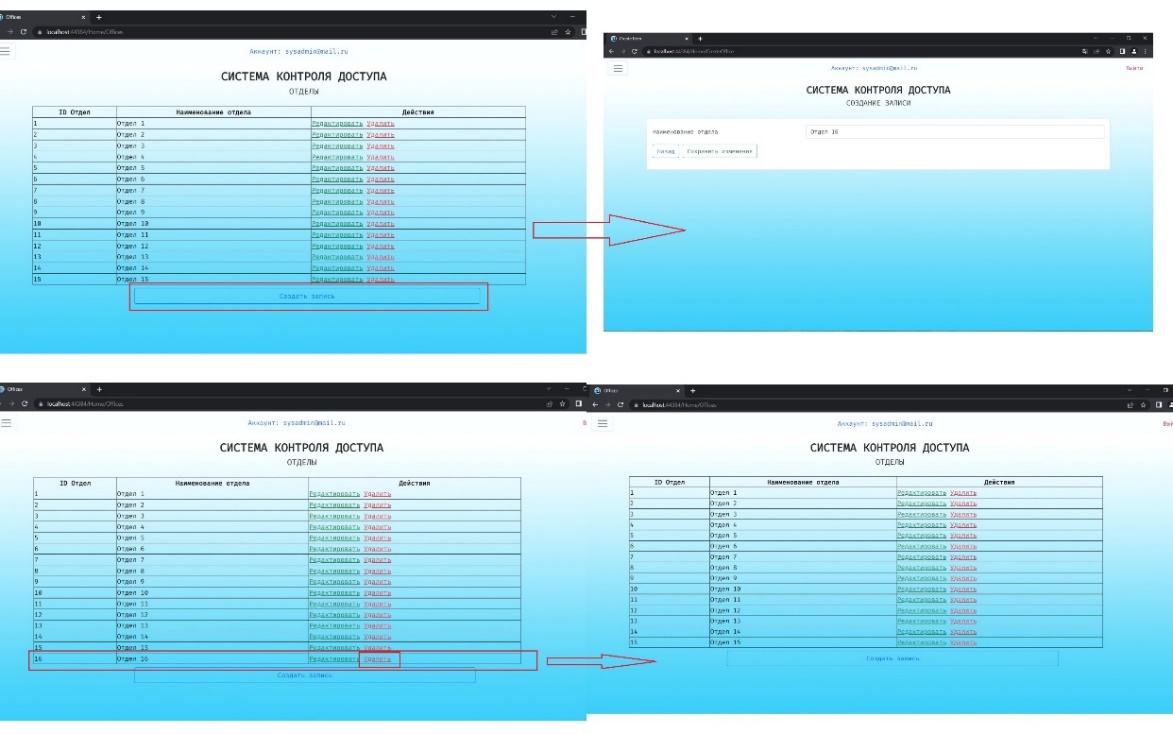


Рисунок 4.19 – Функции добавления записей (сверху) и удаления записей (снизу)



Рисунок 4.20 – Просмотр сведений о рабочем времени сотрудников

ЗАКЛЮЧЕНИЕ

В ходе прохождения производственной преддипломной практики был разработан программный продукт для системы контроля доступа, интегрированный с базой данных MS SQL Server 2014 в режиме реального времени. В ходе создания веб-сервиса были выполнены основные этапы создания программного обеспечения:

1. анализ уже существующих ПО;
2. разработка архитектуры информационной системы;
3. разработка структуры базы данных и автоматизированное наполнение ее тестовыми данными;
4. реализация многопользовательского интерфейса web-приложения с возможностью авторизации и регистрации пользователей;
5. реализация функции вывода данных в табличном виде с функциями сортировки и фильтрации данных;
6. реализация функции добавления новых записей, редактирования и удаления существующих;
7. реализация системы учета рабочего времени;
8. реализация функции вывода данных в графическом виде;
9. реализация функции сохранения данных во внешнем файле;
10. разработка сопроводительной документации.

В процессе разработки проекта была использована система контроля и управления версиями программного обеспечения GitHub, организовано централизованное хранение программных версий в удаленном репозитории и настроем обмен с локальным репозиторием на рабочем компьютере.

На данный момент приложение содержит весь необходимый для работы функционал, простой и понятный интерфейс. Ожидается, что программный продукт позволит значительно упростить контроль доступа на любое предприятие, тем самым повысив уровень безопасности организации.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. ГОСТ 24104-85 ЕСС АСУ «Автоматизированные системы управления. Общие требования».
2. ГОСТ 24104-86 ЕСС АСУ «Автоматизированные системы управления. Стадии создания».
3. ГОСТ 34.201-89 «Информационная технология. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначение документов при создании автоматизированных систем».
4. ГОСТ 34.603-92 «Информационная технология. Комплекс стандартов на автоматизированные системы. Виды и порядок освоения».
5. Гущин А. Н. Базы данных: учебник. – М.: Директ-Медиа, 2014. – 266 с.   
   <http://biblioclub.ru/index.php?page=book_red&id=222149&sr=1>
6. Гущин А.Н. Базы данных: учебно-методическое пособие. – М., Берлин: Директ-Медиа, 2015. – 311 с.   
   <http://biblioclub.ru/index.php?page=book_red&id=278093&sr=1>
7. Указания к оформлению дипломных и курсовых проектов и работ. Методические указания / Н.Б. Лошкарев, А.Н. Лошкарев, Л.А. Зайнуллин. – Екатеринбург: УГТУ–УПИ, 2007. – 49 с.
8. Википедия: SQL Server Management Studio. Полный текст: <https://ru.wikipedia.org/wiki/SQL_Server_Management_Studio>
9. ERDPlus – официальный сайт. [ЭРДПлюс (erdplus.com)](https://erdplus.com/)
10. Википедия: Azure DevOps Server. Полный текст: <https://ru.wikipedia.org/wiki/Azure_DevOps_Server>
11. Википедия: GitHub. Полный текст: <https://ru.wikipedia.org/wiki/GitHub>
12. Назарова О.Б., Мекешкин Е.Т. Анализ систем контроля и управления доступом для использования в муниципальных организациях // Научное обозрение. Технические науки. – 2019. – № 4. – С. 50-54; Полный текст: <https://science-engineering.ru/ru/article/view?id=1256>
13. Алгоритм и основные принципы работы СКУД. Полный текст: <http://asupro.com/building/control/algorithm-basic-principles-skud.html>
14. Метанит: Отправка файлов. Полный текст: <https://metanit.com/sharp/aspnet5/5.7.php>

ПРИЛОЖЕНИЕ А ФРАГМЕНТ ЛИСТИНГА ПРОГРАММНОГО КОДА   
КОНТРОЛЛЕРА HOMECONTROLLER.CS

using EntranceControlWeb.Models;

using EntranceControlWeb.Data;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Logging;

using System;

using System.Diagnostics;

using System.LINQ;

using Microsoft.AspNetCore.Mvc.Rendering;

namespace EntranceControlWeb.Controllers

{

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

private EntranceControlWebContext \_context;

public HomeController(ILogger<HomeController> logger, EntranceControlWebContext context)

{

\_logger = logger;

\_context = context;

}

#region ДЕЙСТВИЯ С ТАБЛИЦЕЙ "ТУРНИКЕТЫ"

//Отображение страницы

public IActionResult Doors(DoorViewModel door, string sort, string Search, int? IdRoom)

{

door.RoomSelect = new SelectList(\_context.Rooms, "IdRoom", "TitleRoom");

ViewBag.RoomSort = sort == "Room" ? "Room dsc" : "Room";

ViewBag.DoorSort = String.IsNullOrEmpty(sort) ? "Door dsc" : "Door";

var find = from s in \_context.Doors select s;

if (!String.IsNullOrEmpty(Search))

{

find = find.Where(s => s.TitleDoor.ToUpper().Contains(Search.ToUpper()));

}

if(IdRoom != 0 && IdRoom != null)

{

find = find.Where(s => s.IdRoom == IdRoom);

}

switch (sort)

{

case "Room":

find = find.OrderBy(s => s.IdRoom);

break;

case "Room dsc":

find = find.OrderByDescending(s => s.IdRoom);

break;

case "Door":

find = find.OrderBy(s => s.TitleDoor);

break; case "Door dsc":

find = find.OrderByDescending(s => s.TitleDoor);

break;

}

door.Doors = find.ToList();

door.Rooms = \_context.Rooms.ToList();

return View(door);

}

public IActionResult ClearDoor()

{

return RedirectToAction(nameof(Doors));

}

//Удаление записи

public IActionResult DelDoor(int id)

{

var data = \_context.Doors.FirstOrDefault(x => x.IdDoor == id);

var entr = \_context.Entrances.FirstOrDefault(x => x.IdDoor == id);

if (entr != null)

{

\_context.Entrances.Remove(entr);

\_context.Doors.Remove(data);

}

\_context.SaveChanges();

return RedirectToAction(nameof(Doors));

}

//Редактирование записи

[HttpPost]

public IActionResult DoorEdit(DoorViewModel door)

{

var edit = \_context.Doors.FirstOrDefault(x => x.IdDoor == door.IdDoor);

edit.IdDoor = door.IdDoor;

edit.TitleDoor = door.TitleDoor;

edit.IdRoom = door.IdRoom;

\_context.SaveChanges();

return RedirectToAction(nameof(Doors));

}

public IActionResult DoorEdit(int id)

{

var viewmodel = new DoorViewModel();

var view = \_context.Doors.FirstOrDefault(x => x.IdDoor == id);

if (view != null)

{

viewmodel.IdDoor = view.IdDoor;

viewmodel.TitleDoor = view.TitleDoor;

viewmodel.IdRoom = view.IdRoom;

}

return Json(viewmodel);

}

//Создание записи

[HttpPost]

public IActionResult CreateDoor(DoorViewModel door)

{

var create = new Door { TitleDoor = door.TitleDoor, IdRoom = door.IdRoom };

\_context.Doors.Add(create);

\_context.SaveChanges();

return RedirectToAction(nameof(Doors));

}

public IActionResult CreateDoor()

{

var door = new DoorViewModel();

ViewBag.Room = new SelectList(\_context.Rooms, "IdRoom", "TitleRoom");

door.Doors = \_context.Doors.ToList();

door.Rooms = \_context.Rooms.ToList();

return View(door);

}

#endregion

#region ДЕЙСТВИЯ С ТАБЛИЦЕЙ "ОТДЕЛЫ"

//Отображение страницы

public IActionResult Offices(OfficeViewModel off, string sort, string Search)

{

ViewBag.OffSort = String.IsNullOrEmpty(sort) ? "Off dsc" : "Off";

var find = from s in \_context.Offices select s;

if (!String.IsNullOrEmpty(Search))

{

find = find.Where(s => s.TitleOffice.ToUpper().Contains(Search.ToUpper()));

}

switch (sort)

{

case "Off":

find = find.OrderBy(s => s.IdOffice);

break;

case "Off dsc":

find = find.OrderByDescending(s => s.IdOffice);

break;

}

off.Offices = find.ToList();

return View(off);

}

//Удаление записи

public IActionResult DelOffice(int id)

{

var data = \_context.Offices.FirstOrDefault(x => x.IdOffice == id);

var office = \_context.SortingByOffices.FirstOrDefault(x => x.IdOffice == id);

if (data != null)

{

\_context.SortingByOffices.Remove(office);

\_context.Offices.Remove(data);

}

\_context.SaveChanges();

return RedirectToAction(nameof(Offices));

}

//Редактирование записи

[HttpPost]

public IActionResult OfficeEdit(OfficeViewModel off)

{

var edit = \_context.Offices.FirstOrDefault(x => x.IdOffice == off.IdOffice);

edit.IdOffice = off.IdOffice;

edit.TitleOffice = off.TitleOffice;

\_context.SaveChanges();

return RedirectToAction(nameof(Offices));

}

public IActionResult OfficeEdit(int id)

{

var viewmodel = new OfficeViewModel();

var view = \_context.Offices.FirstOrDefault(x => x.IdOffice == id);

if (view != null)

{

viewmodel.IdOffice = view.IdOffice;

viewmodel.TitleOffice = view.TitleOffice;

}

return Json(viewmodel);

}

//Создание записи

[HttpPost]

public IActionResult CreateOffice(OfficeViewModel off)

{

var create = new Office { TitleOffice = off.TitleOffice };

\_context.Offices.Add(create);

\_context.SaveChanges();

return RedirectToAction(nameof(Offices));

}

public IActionResult CreateOffice()

{

return View();

}

#endregion

#region ДЕЙСТВИЯ С ТАБЛИЦЕЙ "ДОЛЖНОСТИ"

//Отображение страницы

public IActionResult Positions(PositionViewModel pos, string sort, string Search)

{

ViewBag.PostSort = String.IsNullOrEmpty(sort) ? "Post dsc" : "";

var find = from s in \_context.Positions select s;

if (!String.IsNullOrEmpty(Search))

{

find = find.Where(s => s.TitlePost.ToUpper().Contains(Search.ToUpper()));

}

switch (sort)

{

default:

find = find.OrderBy(s => s.IdPost);

break;

case "Post dsc":

find = find.OrderByDescending(s => s.IdPost);

break;

}

pos.Positions = find.ToList();

return View(pos);

}

//Удаление записи

public IActionResult DelPos(int id)

{

var data = \_context.Positions.FirstOrDefault(x => x.IdPost == id);

var office = \_context.SortingByOffices.FirstOrDefault(x => x.IdPost == id);

if (data != null)

{

\_context.SortingByOffices.Remove(office);

\_context.Positions.Remove(data);

}

\_context.SaveChanges();

return RedirectToAction(nameof(Positions));

}

//Редактирование записи

[HttpPost]

public IActionResult PosEdit(PositionViewModel pos)

{

var edit = \_context.Positions.FirstOrDefault(x => x.IdPost == pos.IdPost);

edit.IdPost = pos.IdPost;

edit.TitlePost = pos.TitlePost;

\_context.SaveChanges();

return RedirectToAction(nameof(Positions));

}

public IActionResult PosEdit(int id)

{

var viewmodel = new PositionViewModel();

var view = \_context.Positions.FirstOrDefault(x => x.IdPost == id);

if (view != null)

{

viewmodel.IdPost = view.IdPost;

viewmodel.TitlePost = view.TitlePost;

}

return Json(viewmodel);

}

//Создание записи

[HttpPost]

public IActionResult CreatePost(Position pos, int id)

{

var create = new Position { TitlePost = pos.TitlePost };

\_context.Positions.Add(create);

\_context.SaveChanges();

return RedirectToAction(nameof(Positions));

}

public IActionResult CreatePost()

{

return View();

}

#endregion

#region ДЕЙСТВИЯ С ТАБЛИЦЕЙ "ПОМЕЩЕНИЯ"

//Отображение данных

public IActionResult Rooms(RoomViewModel room, string sort, string Search, int? IdLevel)

{

room.LevelSelect = new SelectList(\_context.AccessLevels, "IdLevel", "TitleLevel");

ViewBag.RoomSort = sort == "Room" ? "Room dsc" : "Room";

ViewBag.LevelSort = sort == "Level" ? "Level dsc" : "Level";

var find = from s in \_context.Rooms select s;

if (!String.IsNullOrEmpty(Search))

{

find = find.Where(s => s.TitleRoom.ToUpper().Contains(Search.ToUpper()));

}

if (IdLevel != 0 && IdLevel != null)

{

find = find.Where(s => s.IdLevel == IdLevel);

}

switch (sort)

{

case "Room":

find = find.OrderBy(s => s.IdRoom);

break;

case "Room dsc":

find = find.OrderByDescending(s => s.IdRoom);

break;

case "Level":

find = find.OrderBy(s => s.IdLevel);

break;

case "Level dsc":

find = find.OrderByDescending(s => s.IdLevel);

break;

}

room.Rooms = find.ToList();

room.Levels = \_context.AccessLevels.ToList();

return View(room);

}

public IActionResult ClearRoom()

{

return RedirectToAction(nameof(Rooms));

}

//Удаление записи

public IActionResult DelRoom(int id)

{

var data = \_context.Rooms.FirstOrDefault(x => x.IdRoom == id);

var entr = \_context.Entrances.FirstOrDefault(x => x.IdRoom == id);

var door = \_context.Doors.FirstOrDefault(x => x.IdRoom == id);

if (data != null)

{

\_context.Entrances.Remove(entr);

\_context.Doors.Remove(door);

\_context.Rooms.Remove(data);

}

\_context.SaveChanges();

return RedirectToAction(nameof(Rooms));

}

//Редактирование записи

[HttpPost]

public IActionResult RoomEdit(RoomViewModel room)

{

var edit = \_context.Rooms.FirstOrDefault(x => x.IdRoom == room.IdRoom);

edit.IdRoom = room.IdRoom;

edit.TitleRoom = room.TitleRoom;

edit.IdLevel = room.IdLevel;

\_context.SaveChanges();

return RedirectToAction(nameof(Rooms));

}

public IActionResult RoomEdit(int id)

{

var viewmodel = new RoomViewModel();

var view = \_context.Rooms.FirstOrDefault(x => x.IdRoom == id);

if (view != null)

{

viewmodel.IdRoom = view.IdRoom;

viewmodel.TitleRoom = view.TitleRoom;

viewmodel.IdLevel = view.IdLevel;

}

return Json(viewmodel);

}

//Создание записи

[HttpPost]

public IActionResult CreateRoom(RoomViewModel room)

{

var create = new Room { TitleRoom = room.TitleRoom, IdLevel = room.IdLevel };

\_context.Rooms.Add(create);

\_context.SaveChanges();

return RedirectToAction(nameof(Rooms));

}

public IActionResult CreateRoom()

{

var room = new RoomViewModel();

SelectList lev = new SelectList(\_context.AccessLevels, "IdLevel", "TitleLevel");

ViewBag.Levels = lev;

room.Rooms = \_context.Rooms.ToList();

room.Levels = \_context.AccessLevels.ToList();

return View(room);

}

#endregion

#region ДЕЙСТВИЯ С ТАБЛИЦЕЙ "РАСПРЕДЕЛЕНИЕ ПО ОТДЕЛАМ"

//Отображение страницы

public IActionResult SortByOff(SortingByOfficeViewModel sort, string order, string Search1, string Search2, int? IdStaff, int? IdPost, int? IdOffice)

{

sort.StaffSelect = new SelectList(\_context.staff, "IdStaff", "Surname");

sort.PostSelect = new SelectList(\_context.Positions, "IdPost", "TitlePost");

sort.OfficeSelect = new SelectList(\_context.Offices, "IdOffice", "TitleOffice");

ViewBag.SortStaff = order == "Staff" ? "Staff dsc" : "Staff";

ViewBag.SortPost = order == "Post" ? "Post dsc" : "Post";

ViewBag.SortOffice = order == "Office" ? "Office dsc" : "Office";

ViewBag.IdSort = order == "Id" ? "Id desc" : "Id";

var find = from s in \_context.SortingByOffices select s;

if (!String.IsNullOrEmpty(Search1))

{

find = find.Where(s => s.TimeBegin.ToString().ToUpper().Contains(Search1.ToUpper()));

}

if (!String.IsNullOrEmpty(Search2))

{

find = find.Where(s => s.TimeEnd.ToString().ToUpper().Contains(Search2.ToUpper()));

}

if (IdStaff != 0 && IdStaff != null)

{

find = find.Where(s => s.IdStaff == IdStaff);

}

if(IdPost != 0 && IdPost != null)

{

find = find.Where(s => s.IdPost == IdPost);

}

if (IdOffice != 0 && IdOffice != null)

{

find = find.Where(s => s.IdOffice == IdOffice);

}

switch (order)

{

case "Staff":

find = find.OrderBy(s => s.IdStaff);

break;

case "Staff dsc":

find = find.OrderByDescending(s => s.IdStaff);

break;

case "Post":

find = find.OrderBy(s => s.IdPost);

break;

case "Post dsc":

find = find.OrderByDescending(s => s.IdPost);

break;

case "Office":

find = find.OrderBy(s => s.IdOffice);

break;

case "Office dsc":

find = find.OrderByDescending(s => s.IdOffice);

break;

case "Id":

find = find.OrderBy(s => s.IdItem);

break;

case "Id desc":

find = find.OrderByDescending(s => s.IdItem);

break;

}

sort.Sortings = find.ToList();

sort.Staffs = \_context.staff.ToList();

sort.Positions = \_context.Positions.ToList();

sort.Offices = \_context.Offices.ToList();

return View(sort);

}

public IActionResult ClearSort()

{

return RedirectToAction(nameof(SortByOff));

}

//Удаление записи

public IActionResult DelSort(int id)

{

var data = \_context.SortingByOffices.FirstOrDefault(x => x.IdItem == id);

if (data != null)

{

\_context.SortingByOffices.Remove(data);

\_context.SaveChanges();

}

return RedirectToAction(nameof(SortByOff));

}

//Редактирование записи

[HttpPost]

public IActionResult SortEdit(SortingByOfficeViewModel sort)

{

var edit = \_context.SortingByOffices.FirstOrDefault(x => x.IdItem == sort.IdItem);

edit.IdItem = sort.IdItem;

edit.TimeBegin = sort.TimeBegin;

edit.TimeEnd = sort.TimeEnd;

edit.WorkPhone = sort.WorkPhone;

edit.IdStaff = sort.IdStaff;

edit.IdPost = sort.IdPost;

edit.IdOffice = sort.IdOffice;

\_context.SaveChanges();

return RedirectToAction(nameof(SortByOff));

}

public IActionResult SortEdit(SortingByOfficeViewModel sort, int id)

{

sort.StaffSelect = new SelectList(\_context.staff, "IdStaff", "Surname");

sort.PostSelect = new SelectList(\_context.Positions, "IdPost", "TitlePost");

sort.OfficeSelect = new SelectList(\_context.Offices, "IdOffice", "TitleOffice");

sort.Sortings = \_context.SortingByOffices.ToList();

sort.Staffs = \_context.staff.ToList();

sort.Positions = \_context.Positions.ToList();

sort.Offices = \_context.Offices.ToList();

var view = \_context.SortingByOffices.FirstOrDefault(x => x.IdItem == id);

if (id != 0)

{

var edit = \_context.SortingByOffices.FirstOrDefault(x => x.IdItem == id);

sort.IdItem = edit.IdItem;

sort.TimeBegin = edit.TimeBegin;

sort.TimeEnd = edit.TimeEnd;

sort.WorkPhone = edit.WorkPhone;

sort.IdStaff = edit.IdStaff;

sort.IdPost = edit.IdPost;

sort.IdOffice = edit.IdOffice;

}

return View(sort);

}

//Создание записи

[HttpPost]

public IActionResult CreateSort(SortingByOfficeViewModel sort)

{

var create = new SortingByOffice

{

TimeBegin = sort.TimeBegin,

TimeEnd = sort.TimeEnd,

WorkPhone = sort.WorkPhone,

IdStaff = sort.IdStaff,

IdPost = sort.IdPost,

IdOffice = sort.IdOffice,

};

\_context.SortingByOffices.Add(create);

\_context.SaveChanges();

return RedirectToAction(nameof(SortByOff));

}

public IActionResult CreateSort()

{

var sort = new SortingByOfficeViewModel();

SelectList staff = new SelectList(\_context.staff, "IdStaff", "Surname");

ViewBag.Staff = staff;

SelectList pos = new SelectList(\_context.Positions, "IdPost", "TitlePost");

ViewBag.Position = pos;

SelectList of = new SelectList(\_context.Offices, "IdOffice", "TitleOffice");

ViewBag.Office = of;

sort.Sortings = \_context.SortingByOffices.ToList();

sort.Staffs = \_context.staff.ToList();

sort.Positions = \_context.Positions.ToList();

sort.Offices = \_context.Offices.ToList();

return View(sort);

}

#endregion

#region ДЕЙСТВИЯ С ТАБЛИЦЕЙ "СОТРУДНИКИ"

//Отображение данных

public IActionResult Staff(StaffViewModel staff, string Search)

{

var find = from s in \_context.staff select s;

if (!String.IsNullOrEmpty(Search))

{

find = find.Where(s => s.Surname.ToUpper().Contains(Search.ToUpper())

|| s.Name.ToUpper().Contains(Search.ToUpper())

|| s.LastName.ToUpper().Contains(Search.ToUpper()));

}

staff.ActivSelect = new SelectList(\_context.ActivityStatuses, "IdActiv", "TitleActiv");

staff.LongSelect = new SelectList(\_context.LastingStatuses, "IdLong", "TitleLong");

staff.LevelSelect = new SelectList(\_context.AccessLevels, "IdLevel", "TitleLevel");

staff.Passes = \_context.Passes.ToList();

staff.Lastings = \_context.LastingStatuses.ToList();

staff.Activities = \_context.ActivityStatuses.ToList();

staff.Levels = \_context.AccessLevels.ToList();

staff.Staffs = find.ToList();

return View(staff);

}

//Редактирование данных

[HttpPost]

public IActionResult StaffEdit(StaffViewModel staff)

{

var edit = \_context.staff.FirstOrDefault(x => x.IdStaff == staff.IdStaff);

staff.Staffs = \_context.staff.ToList();

edit.IdStaff = staff.IdStaff;

edit.Surname = staff.Surname;

edit.Name = staff.Name;

edit.LastName = staff.LastName;

edit.Birthday = staff.Birthday;

edit.CorpEmail = staff.CorpEmail;

edit.MobPhone = staff.MobPhone;

edit.Image = staff.Image;

edit.IdPass = staff.IdPass;

\_context.SaveChanges();

return RedirectToAction(nameof(Staff));

}

public IActionResult StaffEdit(StaffViewModel staff, int id)

{

staff.Staffs = \_context.staff.ToList();

var view = \_context.staff.FirstOrDefault(x => x.IdStaff == id);

if (id != 0)

{

var edit = \_context.staff.FirstOrDefault(x => x.IdStaff == id);

staff.Staffs = \_context.staff.ToList();

staff.IdStaff = edit.IdStaff;

staff.Surname = edit.Surname;

staff.Name = edit.Name;

staff.LastName = edit.LastName;

staff.Birthday = edit.Birthday;

staff.CorpEmail = edit.CorpEmail;

staff.MobPhone = edit.MobPhone;

staff.Image = edit.Image;

staff.IdPass = edit.IdPass;

}

return View(staff);

}

//Редактирование записи

[HttpPost]

public IActionResult PassEditStaff(PassesViewModel pass)

{

var edit = \_context.Passes.FirstOrDefault(x => x.IdPass == pass.IdPass); edit.IdPass = pass.IdPass;

edit.IdActiv = pass.IdActiv;

edit.IdLong = pass.IdLong;

edit.IdLevel = pass.IdLevel;

\_context.SaveChanges();

return RedirectToAction(nameof(Staff));

}

public IActionResult PassEditStaff(int id)

{

var pass = new PassesViewModel();

var edit = \_context.Passes.FirstOrDefault(x => x.IdPass == id);

if (edit != null)

{

pass.IdPass = edit.IdPass;

pass.IdActiv = edit.IdActiv;

pass.IdLong = edit.IdLong;

pass.IdLevel = edit.IdLevel;

}

return Json(pass);

}

//Удаление записи

public IActionResult DelStaff(int id)

{

var data = \_context.staff.FirstOrDefault(x => x.IdStaff == id);

var sort = \_context.SortingByOffices.FirstOrDefault(x => x.IdStaff == id);

if (data != null)

{

\_context.SortingByOffices.Remove(sort);

\_context.staff.Remove(data);

}

\_context.SaveChanges();

return RedirectToAction(nameof(Staff));

}

//Создание записи

[HttpPost]

public IActionResult CreateStaff(StaffViewModel staff)

{

var create = new staff

{

Surname = staff.Surname,

Name = staff.Name,

LastName = staff.LastName,

Birthday = staff.Birthday,

CorpEmail = staff.CorpEmail,

MobPhone = staff.MobPhone,

Image = staff.Image,

IdPass = staff.IdPass,

};

\_context.staff.Add(create);

\_context.SaveChanges();

return RedirectToAction(nameof(Staff));

}

public IActionResult CreateStaff()

{

var staff = new StaffViewModel();

staff.Staffs = \_context.staff.ToList();

return View(staff);

}

#endregion

#region ДЕЙСТВИЯ С ТАБЛИЦЕЙ "ДОСТУП"

//Отображение данных

public IActionResult Entrance(EntranceViewModel entr, string sort, string Search1, string Search2, int? IdDoor, int? IdPass, int? IdRoom, int? IdStatus)

{

entr.DoorSelect = new SelectList(\_context.Doors, "IdDoor", "TitleDoor");

entr.PassSelect = new SelectList(\_context.Passes, "IdPass", "IdPass");

entr.RoomSelect = new SelectList(\_context.Rooms, "IdRoom", "TitleRoom");

entr.StatSelect = new SelectList(\_context.AccessStatuses, "IdStatus", "TitleStatus");

ViewBag.DateEntrSort = sort == "DateEntr" ? "DateEntr dsc" : "DateEntr";

ViewBag.DoorSort = sort == "Door" ? "Door desc" : "Door";

ViewBag.RoomSort = sort == "Room" ? "Room desc" : "Room";

ViewBag.IdSort = sort == "Id" ? "Id desc" : "Id";

var find = from s in \_context.Entrances select s;

if (!String.IsNullOrEmpty(Search1))

{

find = find.Where(s => s.DateEntr > DateTime.Parse(Search1));

}

if (!String.IsNullOrEmpty(Search2))

{

find = find.Where(s => s.DateExit < DateTime.Parse(Search2));

}

if (IdRoom != 0 && IdRoom != null)

{

find = find.Where(s => s.IdRoom == IdRoom);

}

if (IdPass != 0 && IdPass != null)

{

find = find.Where(s => s.IdPass == IdPass);

}

if (IdDoor != 0 && IdDoor != null)

{

find = find.Where(s => s.IdDoor == IdDoor);

}

if (IdStatus != 0 && IdStatus != null)

{

find = find.Where(s => s.IdStatus == IdStatus);

}

switch (sort)

{

case "DateEntr":

find = find.OrderBy(s => s.DateEntr);

break;

case "DateEntr dsc":

find = find.OrderByDescending(s => s.DateEntr);

break;

case "Door desc":

find = find.OrderByDescending(s => s.IdDoor);

break;

case "Door":

find = find.OrderBy(s => s.IdDoor);

break;

case "Room":

find = find.OrderBy(s => s.IdRoom);

break;

case "Room desc":

find = find.OrderByDescending(s => s.IdRoom);

break;

case "Id":

find = find.OrderBy(s => s.IdRecord);

break;

case "Id desc":

find = find.OrderByDescending(s => s.IdRecord);

break;

}

entr.Entrances = find.ToList();

entr.Passes = \_context.Passes.ToList();

entr.Rooms = \_context.Rooms.ToList();

entr.Doors = \_context.Doors.ToList();

entr.Statuses = \_context.AccessStatuses.ToList();

return View(entr);

}

public IActionResult ClearEntr()

{

return RedirectToAction(nameof(Entrance));

}

#endregion #region ДЕЙСТВИЯ С ТАБЛИЦЕЙ "ПОСЕТИТЕЛИ"

//Отображение страницы

public IActionResult Visitors(VisitorViewModel vis, string sort, string Search)

{

vis.ActivSelect = new SelectList(\_context.ActivityStatuses, "IdActiv", "TitleActiv");

vis.LongSelect = new SelectList(\_context.LastingStatuses, "IdLong", "TitleLong");

vis.LevelSelect = new SelectList(\_context.AccessLevels, "IdLevel", "TitleLevel");

ViewBag.PassSort = sort == "Pass" ? "Pass dsc" : "Pass";

var find = from s in \_context.Visitors select s;

if (!String.IsNullOrEmpty(Search))

{

find = find.Where(s => s.Surname.ToUpper().Contains(Search.ToUpper())

|| s.Name.ToUpper().Contains(Search.ToUpper())

|| s.LastName.ToUpper().Contains(Search.ToUpper()));

}

switch (sort)

{

case "Pass":

find = find.OrderBy(s => s.IdPass);

break;

case "Pass dsc":

find = find.OrderByDescending(s => s.IdPass);

break;

}

vis.Visitors = find.ToList();

vis.Passes = \_context.Passes.ToList();

vis.Lastings = \_context.LastingStatuses.ToList();

vis.Activities = \_context.ActivityStatuses.ToList();

vis.Levels = \_context.AccessLevels.ToList();

return View(vis);

}

//Редактирование записи

[HttpPost]

public IActionResult VisEdit(VisitorViewModel vis)

{

var edit = \_context.Visitors.FirstOrDefault(x => x.Idvisitor == vis.Idvisitor);

edit.Idvisitor = vis.Idvisitor;

edit.Surname = vis.Surname;

edit.Name = vis.Name;

edit.LastName = vis.LastName;

edit.MobilePhone = vis.MobilePhone;

edit.IdPass = vis.IdPass;

\_context.SaveChanges();

return RedirectToAction(nameof(Visitors));

}

public IActionResult VisEdit(int id)

{

var vis = new VisitorViewModel();

var edit = \_context.Visitors.FirstOrDefault(x => x.Idvisitor == id);

if (edit != null)

{

vis.Idvisitor = edit.Idvisitor;

vis.Surname = edit.Surname;

vis.Name = edit.Name;

vis.LastName = edit.LastName;

vis.MobilePhone = edit.MobilePhone;

vis.IdPass = edit.IdPass;

}

return Json(vis);

} //Удаление записи

public IActionResult DelVis(int id)

{

var vis = \_context.Visitors.FirstOrDefault(x => x.Idvisitor == id);

if (vis != null)

{

\_context.Visitors.Remove(vis);

}

\_context.SaveChanges();

return RedirectToAction(nameof(Visitors));

}

//Создание записи

[HttpPost]

public IActionResult CreateVisit(VisitorViewModel vis)

{

var create = new Visitor { Surname = vis.Surname, Name = vis.Name, LastName = vis.LastName, MobilePhone = vis.MobilePhone, IdPass = vis.IdPass };

\_context.Add(create);

\_context.SaveChanges();

return RedirectToAction(nameof(Visitors));

}

public IActionResult CreateVisit()

{

var vis = new VisitorViewModel();

vis.Visitors = \_context.Visitors.ToList();

vis.Passes = \_context.Passes.ToList();

return View(vis);

}

//Редактирование записи пропуска

[HttpPost]

public IActionResult PassEditVis(PassesViewModel pass)

{

var edit = \_context.Passes.FirstOrDefault(x => x.IdPass == pass.IdPass);

edit.IdPass = pass.IdPass;

edit.IdActiv = pass.IdActiv;

edit.IdLong = pass.IdLong;

edit.IdLevel = pass.IdLevel;

\_context.SaveChanges();

return RedirectToAction(nameof(Visitors));

}

public IActionResult PassEditVis(int id)

{

var pass = new PassesViewModel();

var edit = \_context.Passes.FirstOrDefault(x => x.IdPass == id);

if (edit != null)

{

pass.IdPass = edit.IdPass;

pass.IdActiv = edit.IdActiv;

pass.IdLong = edit.IdLong;

pass.IdLevel = edit.IdLevel;

}

return Json(pass);

}

#endregion

#region ДЕЙСТВИЯ С ТАБЛИЦЕЙ "ПРОПУСКА"

//Отображение страницы

public IActionResult Passes(PassesViewModel pass, string Search, int? IdActiv, int? IdLong, int? IdLevel)

{

var find = from s in \_context.Passes select s;

if (!String.IsNullOrEmpty(Search))

{

find = find.Where(s => s.IdPass.ToString().ToUpper().Contains(Search.ToUpper()));

}

if (IdActiv != 0 && IdActiv != null)

{

find = find.Where(s => s.IdActiv == IdActiv);

}

if (IdLong != 0 && IdLong != null)

{

find = find.Where(s => s.IdLong == IdLong);

}

if (IdLevel != 0 && IdLevel != null)

{

find = find.Where(s => s.IdLevel == IdLevel);

}

pass.ActivSelect = new SelectList(\_context.ActivityStatuses, "IdActiv", "TitleActiv");

pass.LongSelect = new SelectList(\_context.LastingStatuses, "IdLong", "TitleLong");

pass.LevelSelect = new SelectList(\_context.AccessLevels, "IdLevel", "TitleLevel");

pass.Passes = find.ToList();

pass.Lastings = \_context.LastingStatuses.ToList();

pass.Activities = \_context.ActivityStatuses.ToList();

pass.Levels = \_context.AccessLevels.ToList();

return View(pass);

}

public IActionResult ClearPass()

{

return RedirectToAction(nameof(Passes));

} //Создание записи

[HttpPost]

public IActionResult CreatePass(PassesViewModel pass)

{

var create = new Pass { IdPass = pass.IdPass, IdLong = pass.IdLong, IdActiv = pass.IdActiv, IdLevel = pass.IdLevel };

\_context.Add(create);

\_context.SaveChanges();

return RedirectToAction(nameof(Passes));

}

public IActionResult CreatePass()

{

var pass = new PassesViewModel();

SelectList last = new SelectList(\_context.LastingStatuses, "IdLong", "TitleLong");

ViewBag.Lasting = last;

SelectList act = new SelectList(\_context.ActivityStatuses, "IdActiv", "TitleActiv");

ViewBag.Activ = act;

SelectList lev = new SelectList(\_context.AccessLevels, "IdLevel", "TitleLevel");

ViewBag.Level = lev;

pass.Activities = \_context.ActivityStatuses.ToList();

pass.Lastings = \_context.LastingStatuses.ToList();

pass.Levels = \_context.AccessLevels.ToList();

pass.Passes = \_context.Passes.ToList();

return View(pass);

}

//Редактирование записи

[HttpPost]

public IActionResult PassEdit(PassesViewModel pass)

{

var edit = \_context.Passes.FirstOrDefault(x => x.IdPass == pass.IdPass);

edit.IdPass = pass.IdPass;

edit.IdActiv = pass.IdActiv;

edit.IdLong = pass.IdLong;

edit.IdLevel = pass.IdLevel;

\_context.SaveChanges();

return RedirectToAction(nameof(Passes));

}

public IActionResult PassEdit(int id)

{

var pass = new PassesViewModel();

var edit = \_context.Passes.FirstOrDefault(x => x.IdPass == id);

if(edit != null)

{

pass.IdPass = edit.IdPass;

pass.IdActiv = edit.IdActiv;

pass.IdLong = edit.IdLong;

pass.IdLevel = edit.IdLevel;

}

return Json(pass);

}

//Удаление записи

public IActionResult DelPass(int id)

{

var pass = \_context.Passes.FirstOrDefault(x => x.IdPass == id);

if (pass != null)

{

\_context.Passes.Remove(pass);

}

\_context.SaveChanges();

return RedirectToAction(nameof(Passes));

} #endregion

#region ДЕЙСТВИЯ СО СТПРАВОЧНЫМИ ТАБЛИЦАМИ

public IActionResult Dictionary(DictionaryViewModel dict)

{

dict.Levels = \_context.AccessLevels.ToList();

dict.Lastings = \_context.LastingStatuses.ToList();

dict.Statuses = \_context.AccessStatuses.ToList();

dict.Activities = \_context.ActivityStatuses.ToList();

return View(dict);

}

//Удаление записи таблицы "Уровни доступа"

public IActionResult DelLevel(int id)

{

var data = \_context.AccessLevels.FirstOrDefault(x => x.IdLevel == id);

var room = \_context.Rooms.FirstOrDefault(x => x.IdLevel == id);

var pass = \_context.Passes.FirstOrDefault(x => x.IdLevel == id);

if (data != null)

{

\_context.Passes.Remove(pass);

\_context.Rooms.Remove(room);

\_context.AccessLevels.Remove(data);

}

\_context.SaveChanges();

return RedirectToAction(nameof(Dictionary));

}

//Редактирование записи таблицы "Уровни доступа"

[HttpPost]

public IActionResult LevelEdit(DictionaryViewModel acclev)

{

var edit = \_context.AccessLevels.FirstOrDefault(x => x.IdLevel == acclev.IdLevel);

edit.IdLevel = acclev.IdLevel;

edit.TitleLevel = acclev.TitleLevel;

\_context.SaveChanges();

return RedirectToAction(nameof(Dictionary));

}

public IActionResult LevelEdit(int id)

{

var viewmodel = new DictionaryViewModel();

var view = \_context.AccessLevels.FirstOrDefault(x => x.IdLevel == id);

if (view != null)

{

viewmodel.IdLevel = view.IdLevel;

viewmodel.TitleLevel = view.TitleLevel;

}

return Json(viewmodel);

}

//Создание записи таблицы "Уровни доступа"

[HttpPost]

public IActionResult CreateLevel(DictionaryViewModel lev)

{

var create = new AccessLevel { TitleLevel = lev.TitleLevel };

\_context.AccessLevels.Add(create);

\_context.SaveChanges();

return RedirectToAction(nameof(Dictionary));

}

public IActionResult CreateLevel()

{

return View();

}

//Удаление записи таблицы "Статус доступа"

public IActionResult DelStatus(int id)

{

var data = \_context.AccessStatuses.FirstOrDefault(x => x.IdStatus == id);

var entr = \_context.Entrances.FirstOrDefault(x => x.IdStatus == id);

if (data != null)

{

\_context.Entrances.Remove(entr);

\_context.AccessStatuses.Remove(data);

}

\_context.SaveChanges();

return RedirectToAction(nameof(Dictionary));

}

//Редактирование записи таблицы "Статус доступа"

[HttpPost]

public IActionResult StatusEdit(DictionaryViewModel accstat)

{

var edit = \_context.AccessStatuses.FirstOrDefault(x => x.IdStatus == accstat.IdStatus);

edit.IdStatus = accstat.IdStatus;

edit.TitleStatus = accstat.TitleStatus;

\_context.SaveChanges();

return RedirectToAction(nameof(Dictionary));

}

public IActionResult StatusEdit(int id)

{

var viewmodel = new DictionaryViewModel();

var view = \_context.AccessStatuses.FirstOrDefault(x => x.IdStatus == id);

if (view != null)

{

viewmodel.IdStatus = view.IdStatus;

viewmodel.TitleStatus = view.TitleStatus;

}

return Json(viewmodel);

}

//Создание записи таблицы "Статус доступа"

[HttpPost]

public IActionResult CreateStatus(DictionaryViewModel stat)

{

var create = new AccessStatus { TitleStatus = stat.TitleStatus };

\_context.AccessStatuses.Add(create);

\_context.SaveChanges();

return RedirectToAction(nameof(Dictionary));

}

public IActionResult CreateStatus()

{

return View();

}

//Удаление записи таблицы "Статус Активности"

public IActionResult DelActiv(int id)

{

var data = \_context.ActivityStatuses.FirstOrDefault(x => x.IdActiv == id);

var pass = \_context.Passes.FirstOrDefault(x => x.IdActiv == id);

if (data != null)

{

\_context.Passes.Remove(pass);

\_context.ActivityStatuses.Remove(data);

}

\_context.SaveChanges();

return RedirectToAction(nameof(Dictionary));

}

//Редактирование записи таблицы "Статус Активности"

[HttpPost]

public IActionResult ActivEdit(DictionaryViewModel act)

{

var edit = \_context.ActivityStatuses.FirstOrDefault(x => x.IdActiv == act.IdActiv);

edit.IdActiv = act.IdActiv;

edit.TitleActiv = act.TitleActiv;

\_context.SaveChanges();

return RedirectToAction(nameof(Dictionary));

}

public IActionResult ActivEdit(int id)

{

var viewmodel = new DictionaryViewModel();

var view = \_context.ActivityStatuses.FirstOrDefault(x => x.IdActiv == id);

if (view != null)

{

viewmodel.IdActiv = view.IdActiv;

viewmodel.TitleActiv = view.TitleActiv;

}

return Json(viewmodel);

}

//Создание записи таблицы "Статус Активности"

[HttpPost]

public IActionResult CreateActiv(DictionaryViewModel act)

{

var create = new ActivityStatus { TitleActiv = act.TitleActiv };

\_context.ActivityStatuses.Add(create);

\_context.SaveChanges();

return RedirectToAction(nameof(Dictionary));

}

public IActionResult CreateActiv()

{

return View();

}

//Удаление записи таблицы "Статус длительности"

public IActionResult DelLong(int id)

{

var data = \_context.LastingStatuses.FirstOrDefault(x => x.IdLong == id);

var pass = \_context.Passes.FirstOrDefault(x => x.IdLong == id);

if (data != null)

{

\_context.Passes.Remove(pass);

\_context.LastingStatuses.Remove(data); }

\_context.SaveChanges();

return RedirectToAction(nameof(Dictionary));

}

//Редактирование записи таблицы "Статус длительности"

[HttpPost]

public IActionResult LongEdit(DictionaryViewModel last)

{

var edit = \_context.LastingStatuses.FirstOrDefault(x => x.IdLong == last.IdLong);

edit.IdLong = last.IdLong;

edit.TitleLong = last.TitleLong;

\_context.SaveChanges();

return RedirectToAction(nameof(Dictionary));

}

public IActionResult LongEdit(int id)

{

var viewmodel = new DictionaryViewModel();

var view = \_context.LastingStatuses.FirstOrDefault(x => x.IdLong == id);

if (view != null)

{

viewmodel.IdLong = view.IdLong;

viewmodel.TitleLong = view.TitleLong;

}

return Json(viewmodel);

}

//Создание записи таблицы "Статус длительности"

[HttpPost]

public IActionResult CreateLong(DictionaryViewModel last)

{

var create = new LastingStatus { TitleLong = last.TitleLong };

\_context.LastingStatuses.Add(create);

\_context.SaveChanges();

return RedirectToAction(nameof(Dictionary));

}

public IActionResult CreateLong()

{

return View();

}

#endregion

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error(int code)

{

if (code == 404)

{

return View("Error");

}

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

ПРИЛОЖЕНИЕ Б РУКОВОДСТВО АДМИНИСТРАТОРА СИСТЕМЫ

