СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Технологическая практика обеспечивает последовательность и непрерывность в формировании у студентов-практикантов общепрофессиональных и профессиональных навыков.

Ее ценность заключается в приобретении новых и закреплении уже полученных знаний. В период прохождения технологической практики, обучающиеся закрепляют теоретический материал, приобретают практические навыки.

Технологическая практика проводится под контролем руководителя практики от предприятия (организации) на которую прибыл студент-практикант.

Во время технологической практики студенты-практиканты соблюдают график предприятия, не нарушают правила внутреннего распорядка, выполняют поручения своего руководителя назначеного на предприятии.

Целью технологической практики является сбор данных для оформления отчета, получение реального опыта работы на предприятии, изучение ее структуры, оборудования, программ, а также научиться использовать навыки, приобретенные за время обучения в университете.

Местом прохождения практики стал медицинский центр “Горизонт” по специальности «инженер-программист».

Руководителем практики стал инженер-програмист Артюшкевич Андрей Александрович.

Заданием на технологическую практику является:

– разработка приложения для автоматизации деятельности медицинского центра;

– разработка структуры *web*-приложения базы данных медицинского центра “Горизонт”;

– разработка графического интерфейса для базы данных медицинского центра “Горизонт”;.

Данное *Web*-приложение будет использовать архитектуру «клиент-сервер». Архитектура «клиент-сервер» на сегодня получила широкое распространение. Данный тип системы представляет собой взаимодействие структурных компонентов, где структурными компонентами являются сервер и узлы-поставщики определённых сервисов, а также клиенты – пользователи, которые пользуются данным сервисом. Запросы на получение и изменение информации из базы данных отправляют клиенты. Сервер обрабатывает запросы и возвращает клиенту результат (ответ).

Для разработки приложения будет использоваться технология *ASP.NET Core MVC*, база данных будет сгенерирована в системе управления базами данных *MS SQL Server*.

1 ОБЩИЕ СВЕДЕНИЯ О МЕДИЦИНСКОМ ЦЕНТРЕ “ГОРИЗОНТ”

Предприятие введено в эксплуатацию в 2012 году. Созданный первоначально в 1986 году в качестве медико-санитарной части, в составе ОАО «Горизонт», с января 2012 медицинский центр «Горизонт» реорганизован в Частное предприятие «Медицинский центр «Горизонт» в составе ОАО «Управляющая компания холдинга «Горизонт». С февраля 2013 года Предприятие переименовано в Медицинское унитарное предприятие «Медицинский центр «Горизонт».

Деятельность Медицинского центра осуществляется на основании лицензии № 02040/ 7104 от 25 января 2012года, сроком действия лицензии – до 25.01.2022 года (продлена на неопределенный срок).

Виды медицинской деятельности, оказываемые в центре: гинекология, биохимические методы исследования, гематологическая лабораторная диагностика, общеклинические (неинвазивные) методы исследования, вакцинация, кардиология, лечебный массаж, неврология, первичная медицинская помощь, рентгенологическая диагностика, стоматология терапевтическая, хирургическая и ортопедическая, ультразвуковая диагностика, физиотерапия, хирургия, терапия, оториноларингология, офтальмология, эндокринология, урология, предварительные(при поступлении на работу) и переодические медицинские осмотры, медицинский осмотр водителей, освидетельствование на пригодность к управлению автотранспортом, функциональная диагностика.

На предприятии разработана «Программа проведения производственного контроля в учреждении здравоохранения» по соблюдению и выполнению санитарных норм и правил, утвержденная 01.02.2022г.

В центре представлены более 25 направлений медицинской деятельности: от диагностики до оперативных вмешательств. Более 110 000 пациентов ежегодно получают квалифицированную медицинскую помощь на базе нашего центра. 150 компаний Беларуси доверили нам заботу о здоровье своих сотрудников.

Деятельностью медицинского центра руководит главврач, который назначается приказом из главного офиса предприятия «Горизонт». Главврач издает приказы и дает указания, обязательные для лиц, относящихся к персоналу медицинского центра, и других лиц, состоящих с ним в трудовых отношениях. Также он несет полную ответственность за работу медцентра, соблюдение работниками трудовой дисциплины, правил по охране труда и технике безопасности. Право подписи финансовых документов предоставляется директору и главному бухгалтеру.

В непосредственной подчинённости у главврача находятся:
– главная медсестра;
– главный бухгалтер;
– инспектор отдела кадров;
– ведущий инженер-программист;
– инженер по ОТ;

В подчинении главного бухгалтера находятся остальные бухгалтеры и экономист.

Основной задачей бухгалтерии является своевременный и достоверный учет хозяйственных операций и результатов финансово-хозяйственной деятельности, правильный и своевременный учет денежных средств, основных фондов, товарно-материальных ценностей, учет всех финансовых, кредитных и расчетных операций. Бухгалтерия обеспечивает предоставление бухгалтерской отчетности на основе первичных документов.

Главная медсестра отвечает за соблюдение нормативных актов, соблюдение правил санитарной и трудовой дисциплины, а также за работу персонала.

Инспектор отдела кадров отвечает за поиск нового персонала, их трудоустройство в организацию, оформление нужных бумаг, подписание приказов о переводе в другие отделения.

Инженер по охране труда отвечает за осуществление контроля за состоянием охраны труда в организации; оказание методической помощи структурным подразделениям предприятия в налаживании работы по охране труда; оказывает помощь в организации обучения работников по вопросам охраны труда; контролирует обеспечение работников специальной одеждой, специальной обувью и другими средствами индивидуальной защиты, своевременность и правильность ухода за этими изделиями; составляет отчетность об охране труда.

Ведущий инженер-программист ответственнен за исправность оборудования внутри медцентра, за проведение своевременной диагностики и замены устаревшего оборудования.

Каждый работник при исполнении служебных обязанностей руководствуется Уставом организации и должностной инструкцией, в которой чётко и точно должны быть указаны, каким критериям должен удовлетворять работник, занимающий данную должность, его права, обязанности и ответственность.

Организационная структура предприятия приведена в приложении Б на рисунке Б.1.

Инструкция по охране труда при работе с персональным компьютером:

Инженер-программист при осуществлении своих должностных обязанностей осуществляет работу на персональном компьютере (ПК). При работе с ПК обязательным требованием является неукоснительное соблюдение

правил «Инструкции по охране труда при работе на персональном компьютере». Данная инструкция содержит следующие разделы и представлена в приложении В:

– общие требования безопасности;
– требования безопасности перед началом работы;
– требования безопасности при выполнении работы;
– требования безопасности в аварийных ситуациях;
– требования безопасности по окончанию работы;
– приложения.

Должностная инструкция инженера-программиста (приложение Г) содержит следующие разделы:

– общие положения;
– должностные обязанности;
– права инженера-программиста;
– взаимоотношения (связи на должности);
– оценка работы и ответственности.

2 АНАЛИТИЧЕСКИЙ ОБЗОР НА СУЩЕСТВУЮЩИЕ ПРОГРАММНЫЕ ПРОДУКТЫ АВТОМАТИЗАЦИИ РАБОЧЕГО МЕСТА ИНЖЕНЕРА-ПРОГРАММИСТА

К существующим программным продуктам относятся:

– Пакет *Microsoft Office* (*Word, Excel, Outlook*), *Microsoft Word* – текстовый процессор, предназначенный для создания, просмотра, редактирования и форматирования текстов статей, деловых бумаг, а также иных документов, с локальным применением простейших форм таблично-матричных алгоритмов.

– Программа *Microsoft Excel* используется для работы с электронными таблицами, для подсчета данных, построения диаграмм, графиков.

– Программа *Microsoft Outlook* используется для внутренней почты по предприятию, в ней возможно создавать аккаунты для пользователей, предоставлять им доступ к различным частям работы программы, а также ограничивать их возможности использования программы.

В бухгалтерии используют конфигурацию «1С:Бухгалтерия», которая позволяет ввести бухгалтерский учет, отчетность, налоговый учет, управленческий учет, а также позволяет использовать Многопользовательский режим работы, в том числе осуществляется поддержка клиент-серверного и удаленного (через браузер) варианта работы.

Инженер-программист занимается разработкой, производством, ремонтом и эксплуатацией электронных изделий различного назначения. Такие специалисты осуществляют деятельность по разработке схем электронных узлов промышленного и бытового назначения в конструкторских бюро и исследовательских центрах. Инженер-програмист обеспечивает производственные процессы по контролю за оборудованием, обслуживает, налаживает, тестирует и ремонтирует электронные узлы и агрегаты: от бытовой электроники до сложных систем управления на транспорте.

Инженер-програмист проводит ежедневную проверку оборудования в организации. Для этого ему необходимо хранить информацию о данных оборудованиях:

– где находится оборудование;

– в каком оно состоянии;

– проводился ли его ремонт;

– какие детали нужны для ремонта;

– учет прихода деталей для ремонта;

– где хранятся детали, заказанные для ремонта оборудования;

– дата проведения ремонта оборудования;

Для автоматизации деятельности инженера-програмист в медицинском центре “Горизонт”, появилась необходимость разработки программного продукта: веб-приложения базы данных для учёта записей о приёме клиентов. Разработанное приложение с базой данных, позволяет:

– хранить информацию о клиентах медицинского центра;

– хранить информацию о врачах, работающих в медицинском центре;

– хранить информацию о записях на приём клиента к врачу;

– редактировать имеющиеся данные;

– добавлять новые данные;

– удалять данные;

Для выполнения практической работы в качестве *СУБД* используется *MS SQL Server* и язык программирования *C#*. В частности, упрощенная версия ядра *СУБД SQL Server Express– SQL Server LocalDB* входящая в пакет *Visual Studio 2022*. В качестве технологии для разработки веб-приложения используется платформа *ASP.NET Core 5 MVC*. Данная платформа является хорошим решением для создания веб-приложений с помощью шаблона проектирования *Model-View-Controller* (модель-контроллер-представление). Структура *MVC* предполагает разделение приложения на: модель, представление и контроллер. Для доступа к данным используется технология *Entity Framework Core*. Данная технология является *ORM (object-relational mapping* – отображение данных на реальные объекты) инструментом, т.е. она позволяет работать с реляционными базами данных, используя классы и их иерархии.

В приложении используются следующие данные о работе медцентра:

– информация о пациентах (ФИО, дата рождения, название организации, в которой работает пациент);

– информация о врачах (ФИО, дата рождения, занимаемая должность);

– талоны, содержащие информацию о врачах, пациентах, датах записи на приём, а также нюансов приёма и заключение врача;

Эти данные позволят ускорить и систематизировать процесс записи пациентов на приём, а также вести учёт пациентов, рабочих часов врачей и количество обращений в медицинский центр.

3 СТРУКТУРА БАЗЫ ДАННЫХ И ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

Для работы с таблицами базы данных в приложении необходимы классы, которые описывают каждую таблицу. Также в данных классах используются аннотации – специальные атрибуты, которые определяют различные правила для отображения свойств модели.

В каталоге Data находится класс DbCreation, который предназначен для первичного заполнения базы данных при первом запуске приложения, необходимыми для работы сведениями о клиентах, сотрудниках, сделках, и дополнительных данных. HorizontContext, являющийся контекстом базы данных — это основной класс, который координирует функциональные возможности Entity Framework для заданной модели данных. Этот класс создается путем наследования от класса Microsoft.EntityFrameworkCore.DbContext. В коде указываются сущности, которые включаются в модель данных.

Каталог *bin* - сюда помещается скомпилированная сборка приложения *MVC* вместе со всеми ссылаемыми сборками библиотеками и кодом. Каталог wwwroot предназначен для хранения библиотек *JavaScript,* используемых в приложении, стилей css, bootstrap, все что отвечает за визуальную часть приложения, его внешнее оформление.

В корне каталога приложения находятся файл *appsettings.json,* отвечающий за подключение к базе данных, которое передается в контекст путем вызова метода для объекта *DbContextOptionsBuilder*. При локальной разработке система конфигурации *ASP.NET Core* считывает строку подключения из этого файла. *ASP.NET Core* по умолчанию реализует технологию внедрения зависимостей. С помощью внедрения зависимостей службы (контекст базы данных *Entity Framework*) регистрируются во время запуска приложения. Затем компоненты, которые используют эти службы (контроллеры *MVC*), обращаются к ним через параметры конструктора. Регистрация *HorizontContext* происходит в файле *Startup.* В файле *Program* изменено поведение приложения при первичном запуске: получение экземпляра контекста базы данных из контейнера внедрения зависимостей, вызов метода инициализации с передачей ему контекста, освобождение контекста после завершения метода заполнения базы.

Были созданы следующие сущности, каждая из которых содержит свои атрибуты, данные и описание:

– «Клиенты» – *Clients*;

– «Врачи» – *Doctors*;

– «Талоны» – *Talons*;

Для сущности *«Cleint»* было создана таблица с атрибутами: *«ClientId», «ClientName», «ClientBirthday», «ClientWorkPlace»*.

Подробное описание отношения и атрибутов приведено в таблице 3.1.

Таблица 3.1 – Отношения описывающие сущность «*Clients*»

Атрибуты	Описание	Тип данных
ClientID	Уникальный идентификатор содержащий информацию по каждому клиенту. Является первичным ключом.	*int*
ClientName	Содержит информацию о ФИО пациента.	*nvchar*
ClientBirthday	Содержит информацию о дате рождения пациента.	*datetime*
ClientWorkPlace	Содержит информацию об организации, где работает пациент.	*nvchar*

В сущности *«Doctors»* для выполнения поставленной задачи были созданы атрибуты *«DoctorID», «DoctorName», «DoctorBirthday», «DoctorPosition»*.

Подробное описание отношения и атрибутов приведено в таблице 3.2.

Таблица 3.2 – Отношения описывающие сущность «*Doctors*»

Атрибуты	Описание	Тип данных
DoctorId	Уникальный идентификатор содержащий информацию о врачах медцентра. Является первичным ключом.	*int*
DoctorName	Содержит информацию о ФИО врача.	*nvchar*
DoctorBirthday	Содержит информацию о возрасте врача.	*datetime*
DoctorPosition	Содержит информацию о должности.	*nvchar*

В сущности *«Tickets»* были созданы следующие атрибуты: *«TicketId», «VisitDate», «Cost», «WhatHappened», «ResultsOfVisit»*. *«ClientID»* и *«DoctorID»* используются для связи сущности *«Tickets»* с *«Clients»* и *«Doctors»*. Подробное описание отношения и атрибутов приведено в таблице 3.3.

Таблица 3.3 – Отношения описывающие сущность «*Tickets*»

Атрибуты	Описание	Тип данных
TicketId	Уникальный инкрементируемый идентификатор для каждого талона. Является первичным ключом.	*int*
VisitDate	Содержит дату визита к врачу.	*DateTime*
Cost	Содержит информацию о цене услуги.	*int*
WhatHappened	Содержит информацию о проблеме пациента.	*nvchar*
ResultsOfVisit	Содержит информацию о результатах обследования или лечения пациента, а также рекомендации от врача.	*nvchar*
DoctorId	Содержит информацию о ФИО врача, используется для связи с сущностью *Doctors*	*int*
ClientId	Содержит информацию о ФИО пациента, используется для связи с сущностью *Clients*	*int*

Созданная база данных *MS SQL Server* имеет два основных рабочих системных файла, содержащих данные и объекты, такие как таблицы, индексы, хранимые процедуры и представления. Файлы журнала содержат сведения, необходимые для восстановления всех транзакций в базе данных.

С помощью библиотеки Entity Framework было осуществлено взаимодействия языка программирования C# с физической моделью данных, который произвёл соотношения классов и таблиц, с помощью которых можно осуществлять доступ к данным непосредственно при работе приложения.

Физическая модель базы данных – это совокупность методов и средств размещения данных во внешней памяти и созданная на их основе внутренняя (физическая) модель данных.

В процессе проектирования базы данных «*Медицинский центр Горизонт*» были разработаны 3 таблицы: врачи, пациенты, талоны. Каждая из разработанных таблиц взаимосвязана с другой таблицей связью «один ко многим».

На основании данных таблиц и установленных между ними связей была разработана база данных «*Медицинский центр Горизонт*», физическая модель которой представлена на рисунке 3.1.

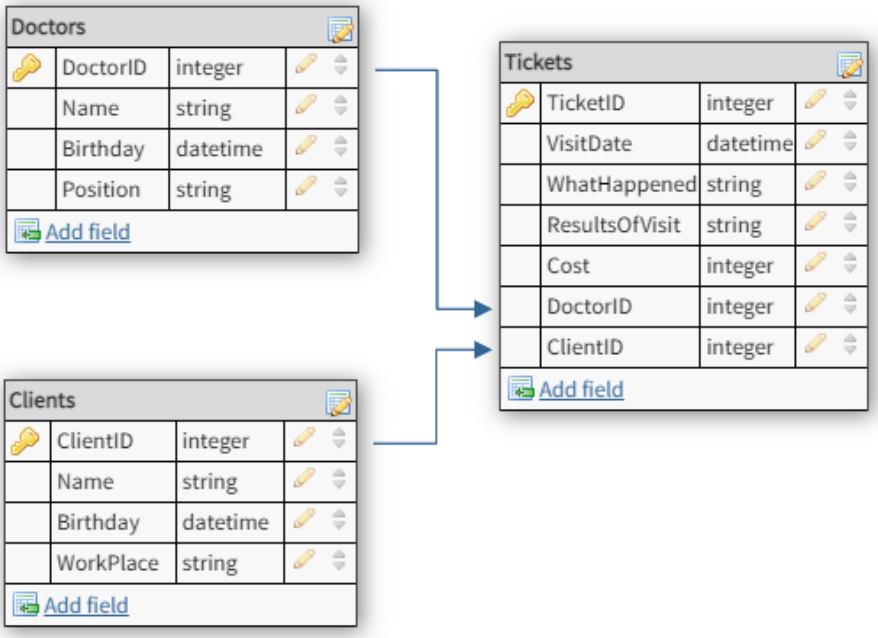

Рисунок 3.1 – Физическая модель базы данных

Таблицы *Doctors* и *Clients*, связана с таблицей *Tickets*, связью один ко многим.

В состав данного веб-приложения входят три основных компонента: модель, представление и контроллер.

Интерфейс программы – это реализация диалога между программой и пользователем. Он создаётся с помощью сущностей, описанных в классах Client, Doctor, Ticket (моделей, которые содержат данные и хранят логику обработки этих данных, но не содержат логику взаимодействия с пользователем) и связывается с представлением (интерфейсом) с помощью контроллера, который обеспечивает связь между пользователем и приложением, хранилищем данных и представляет собой центральный компонент *MVC* для управления взаимодействием с пользователем, работы с моделью и выбора представления для отображения.

Интерфейс определяет, в какой форме будет идти обмен информацией, то есть как программа будет получать данные и команды пользователя и в каком виде представлять ему информацию. Интерфейс составляют все видимые и невидимые компоненты программы, с помощью которых пользователь вводит запрашиваемые данные, управляет режимами ее работы и видит получаемые результаты.

После запуска программы во вкладке браузера появляется главное меню, на нем расположены вкладки, которые переходят на нужную таблицу базы данных (рисунок 3.2).

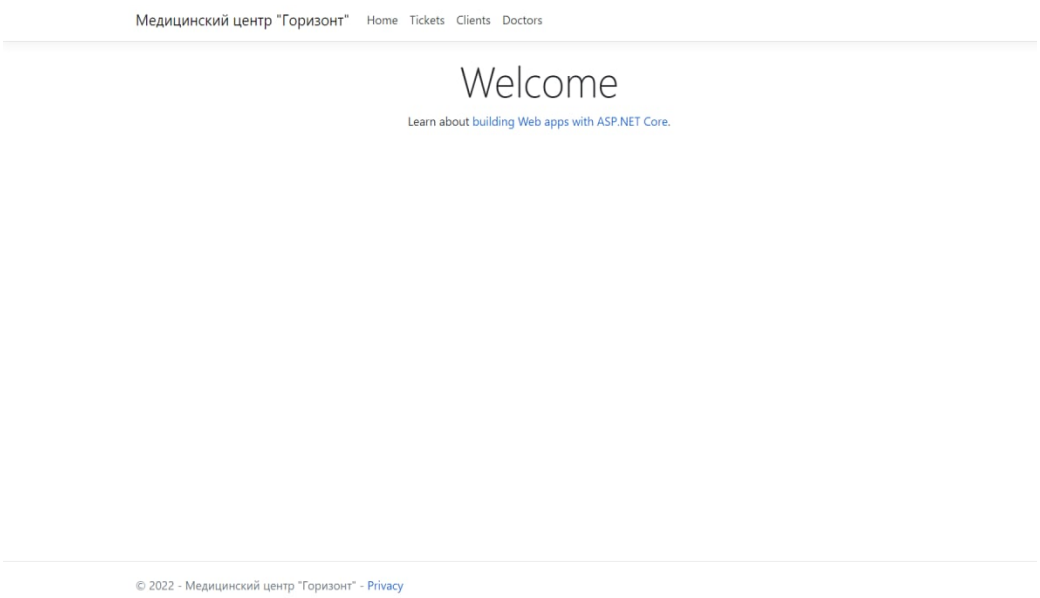

Рисунок 3.2 – Интерфейс главного меню программы

Рассмотрим интерфейс каждой таблицы, созданной в базе данных. При переходе на вкладку Clients появляется окно с таблицей, в которой находится вся основная информация о клиентах медицинского центра. Таблица представлена на рисунке 3.3.

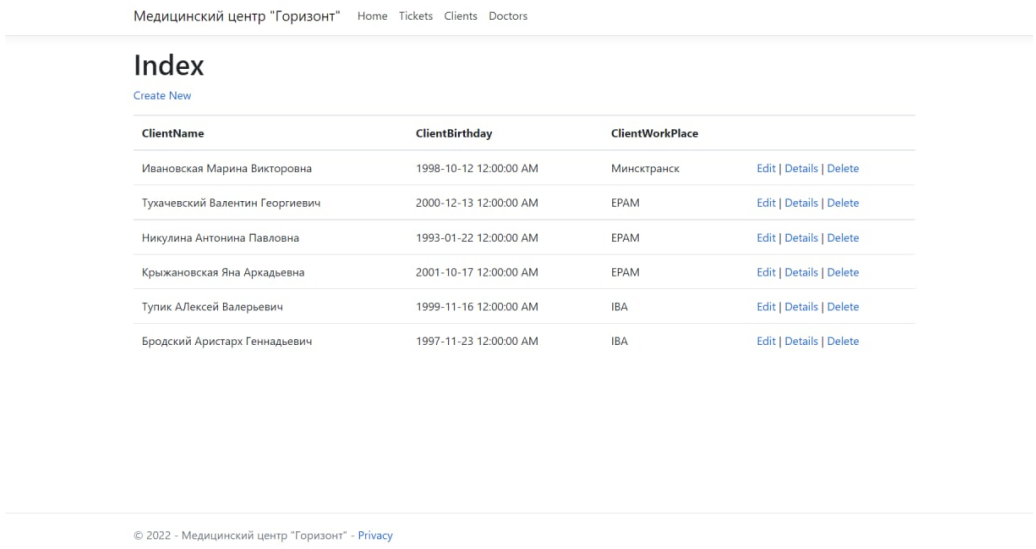

Рисунок 3.3 – Интерфейс таблицы Clients

Также в интерфейсе есть возможность редактирования и удаления данных и создание новых данных.

Изображения добавления, удаления, изменения данных представлен соответственно на рисунке 3.4.

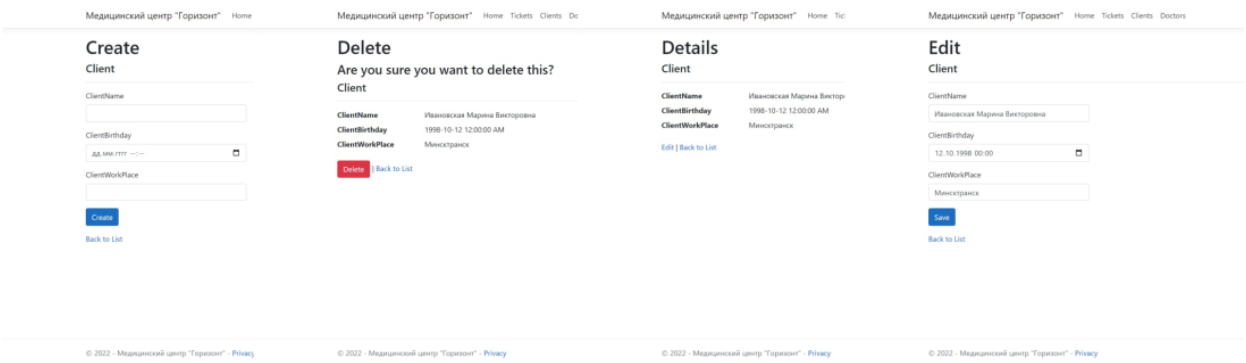

Рисунок 3.4 – Интерфейс добавления, удаления, изменения данных

Интерфейс изменения данных содержит поля для редактирования всех данных по выбранной строке таблицы, а также кнопки Save (для сохранения изменений) и Back to list (для возврата к таблице без изменений).

Интерфейс удаления данных содержит информацию по выбранному полю таблицы и 2 кнопки: Delete (удаление) и Back to list (для возврата к таблице без изменений).

Интерфейс добавления содержит тектовые поля для внесения новых значений вручную, а также 2 кнопки: Create (создать) и Back to list (для возврата к таблице без изменений).

Интерфейс добавления, удаления, изменения данных на этой и всех последующих вкладках идентичен друг другу.

При переходе на вкладку Doctors появляется окно, изображенное на рисунке 3.5.

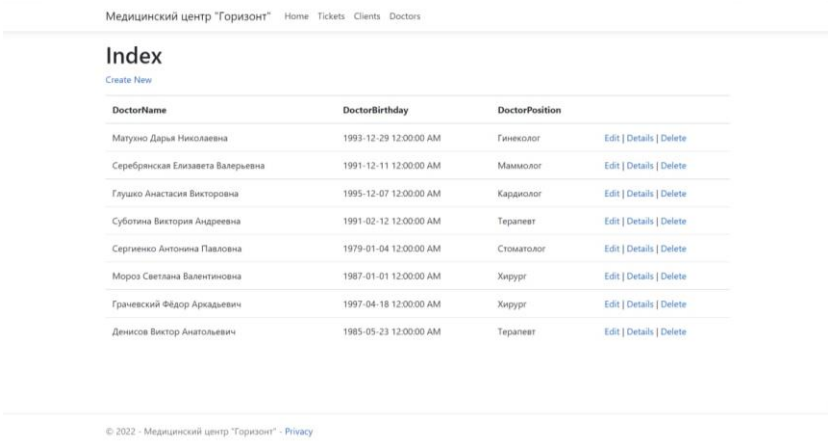

Рисунок 3.5 – Интерфейс таблицы Doctors

Интерфейс добавления, удаления, изменения данных во вкладке Doctor представлен соответственно на рисунке 3.6.

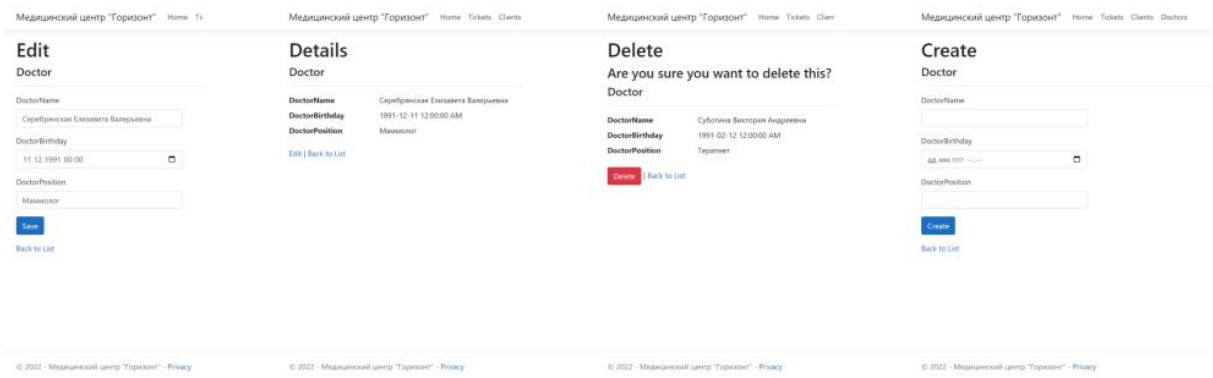

Рисунок 3.6 – Интерфейс добавления, удаления, изменения данных таблицы Doctors

На вкладке Tickets изображено окно, изображенное на рисунке 3.7.

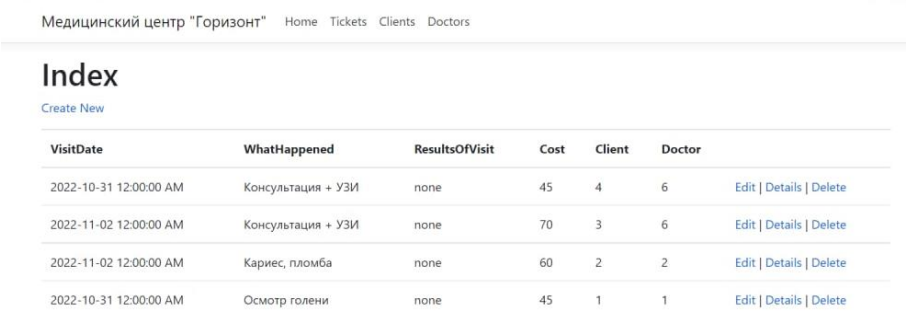

Рисунок 3.7 – Интерфейс таблицы Tickets

Интерфейс добавления, удаления, изменения данных в данной вкладке представлен соответственно на рисунке 3.8.

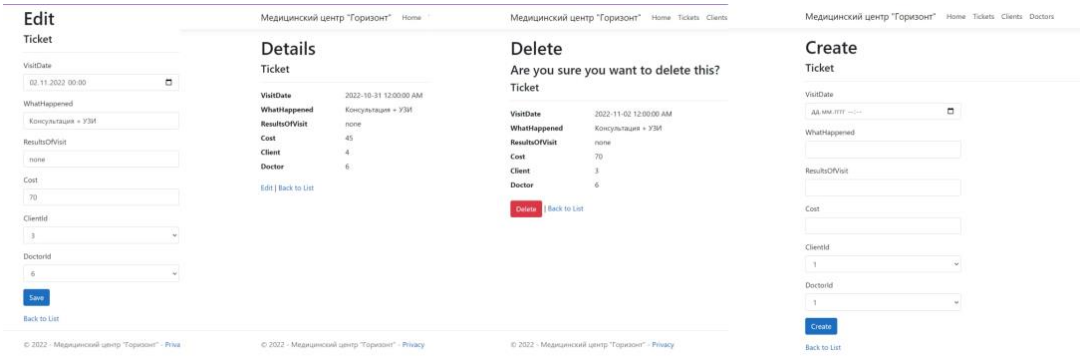

Рисунок 3.8 – Интерфейс добавления, удаления, изменения данных таблицы Tickets.

ЗАКЛЮЧЕНИЕ

При прохождении практики в медицинском центре “Горизонт”, который специализируется на проведении консультаций, диагностики заболеваний и их лечении, были закреплены знания по разработке баз данных, полученные в процессе обучения, а так же подробно изучен материал, необходимый для выполнения индивидуального задания.

Было проведено ознакомление с организационно-правовой формой, структурой управления, предметом деятельности организации. В ходе работы были развиты коммуникативные и организационные навыки, инициативность. Был получен опыт работы в коллективе, а также практические навыки, которые играют определяющую роль в профессиональной деятельности любого специалиста.

Цель технологической практики была достигнута, получены данные для оформления отчета, получен реальный опыт работы на предприятии, изучена его структура, оборудование, программы. Использованы навыки, приобретенные за время обучения в университете.

В процессе прохождении практики было подробно изучено рабочее место инженера-программиста: специфика его работы, данные, необходимые для обслуживания электронного оборудования, непосредствпенно сам процес ремонта оборудования и его тестирование, отчетные документы.

Была разработана структура базы данных с использованием технологии *ASP.NET Core MVC,* и сгенерирована в системе управления базами данных *MS SQL Server*.

Структура базы данных наглядно отображает перечень данных, необходимых для работы инженера-программиста.

Также была разработана структура графического интерфейса пользователя главного окна программы и окна работы со справочниками с учетом предьявленных требований.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *horizont-med.by* [Электронный ресурс] / Информация о предприятии. – Режим доступа: *https://horizont-med.by*/О-нас/. – Дата доступа: 17.10.2022.

2.Хомопепко, А.Д. Базы данных: Учебник для высших учебных заведений / А.Д. Хомопепко, В.М.Цыганков, М.Г. Мальцев. – СПб:.КОРОНА-Век, 2009. – 736 с.

3.Чамберс Д., *ASP.NET Core*. Разработка приложений / Д. Пэкетт, С. Тим. – Спб.: Питер, 2018. – 464 с.

4. *Refdb* [Электронный ресурс] / Графический интерфейс пользователя. – Режим доступа: *https://refdb.ru/look*/2648372.*html*. – Дата доступа: 18.10.2022.

5.Фримен, А. *ASP.NET Core MVC* 2 с примерами на *C#* для профессионалов – ЯрГУ:. Яросл. гос. ун-т им. П. Г. Демидова, 2019. – 468 с.

6.Россум, Г. Язык программирования C# / Д.С. Откидач, М. Задка. – СПб:.КОРОНА-Век, 2001. – 454 с.

7. Зафиевский, А. В. Базы данных: учебное пособие / А. В. Зафиевский, А. А. Короткин, А. Н. Лататуев. – ЯрГУ:. Яросл. гос. ун-т им. П. Г. Демидова, 2012. – 164 с.

ПРИЛОЖЕНИЕ А
Листинг программы
(обязательное)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using BD_Horizont.Data;
using BD_Horizont.Models;

namespace BD_Horizont.Controllers
{
    public class TicketsController : Controller
    {
        private readonly HorizontContext _context;

        public TicketsController(HorizontContext context)
        {
            _context = context;
        }

        // GET: Tickets
        public async Task<IActionResult> Index()
        {
            var horizontContext = _context.Tickets.Include(t => t.Client).Include(t => t.Doctor);
            return View(await horizontContext.ToListAsync());
        }

        // GET: Tickets/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var ticket = await _context.Tickets
                .Include(t => t.Client)
                .Include(t => t.Doctor)
                .FirstOrDefaultAsync(m => m.TicketId == id);
            if (ticket == null)
            {
                return NotFound();
            }

            return View(ticket);
        }

        // GET: Tickets/Create
        public IActionResult Create()
        {
            ViewData["ClientId"] = new SelectList(_context.Clients, "ClientId", "ClientId");
            ViewData["DoctorId"] = new SelectList(_context.Doctors, "DoctorId", "DoctorId");
            return View();
        }

        // POST: Tickets/Create
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
```

```
        public async Task<IActionResult>
Create([Bind("TicketId,VisitDate,WhatHappened,ResultsOfVisit,Cost,ClientId,DoctorId")] Ticket ticket)
        {
            if (ModelState.IsValid)
            {
                _context.Add(ticket);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            ViewData["ClientId"] = new SelectList(_context.Clients, "ClientId", "ClientId", ticket.ClientId);
            ViewData["DoctorId"] = new SelectList(_context.Doctors, "DoctorId", "DoctorId", ticket.DoctorId);
            return View(ticket);
        }

        // GET: Tickets/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var ticket = await _context.Tickets.FindAsync(id);
            if (ticket == null)
            {
                return NotFound();
            }
            ViewData["ClientId"] = new SelectList(_context.Clients, "ClientId", "ClientId", ticket.ClientId);
            ViewData["DoctorId"] = new SelectList(_context.Doctors, "DoctorId", "DoctorId", ticket.DoctorId);
            return View(ticket);
        }

        // POST: Tickets/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("TicketId,VisitDate,WhatHappened,ResultsOfVisit,Cost,ClientId,DoctorId")] Ticket ticket)
        {
            if (id != ticket.TicketId)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(ticket);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!TicketExists(ticket.TicketId))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            ViewData["ClientId"] = new SelectList(_context.Clients, "ClientId", "ClientId", ticket.ClientId);
```

```
            ViewData["DoctorId"] = new SelectList(_context.Doctors, "DoctorId", "DoctorId", ticket.DoctorId);
            return View(ticket);
        }

        // GET: Tickets/Delete/5
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var ticket = await _context.Tickets
                .Include(t => t.Client)
                .Include(t => t.Doctor)
                .FirstOrDefaultAsync(m => m.TicketId == id);
            if (ticket == null)
            {
                return NotFound();
            }

            return View(ticket);
        }

        // POST: Tickets/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var ticket = await _context.Tickets.FindAsync(id);
            _context.Tickets.Remove(ticket);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private bool TicketExists(int id)
        {
            return _context.Tickets.Any(e => e.TicketId == id);
        }
    }
}



using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using BD_Horizont.Data;
using BD_Horizont.Models;

namespace BD_Horizont.Controllers
{
    public class DoctorsController : Controller
    {
        private readonly HorizontContext _context;

        public DoctorsController(HorizontContext context)
        {
            _context = context;
        }

        // GET: Doctors
```

```
        public async Task<IActionResult> Index()
        {
            return View(await _context.Doctors.ToListAsync());
        }

        // GET: Doctors/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var doctor = await _context.Doctors
                .FirstOrDefaultAsync(m => m.DoctorId == id);
            if (doctor == null)
            {
                return NotFound();
            }

            return View(doctor);
        }

        // GET: Doctors/Create
        public IActionResult Create()
        {
            return View();
        }

        // POST: Doctors/Create
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("DoctorId,DoctorName,DoctorBirthday,DoctorPosition")] Doctor
doctor)
        {
            if (ModelState.IsValid)
            {
                _context.Add(doctor);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            return View(doctor);
        }

        // GET: Doctors/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var doctor = await _context.Doctors.FindAsync(id);
            if (doctor == null)
            {
                return NotFound();
            }
            return View(doctor);
        }

        // POST: Doctors/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
```

```
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id, [Bind("DoctorId,DoctorName,DoctorBirthday,DoctorPosition")] Doctor
doctor)
        {
            if (id != doctor.DoctorId)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(doctor);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!DoctorExists(doctor.DoctorId))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            return View(doctor);
        }

        // GET: Doctors/Delete/5
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var doctor = await _context.Doctors
                .FirstOrDefaultAsync(m => m.DoctorId == id);
            if (doctor == null)
            {
                return NotFound();
            }

            return View(doctor);
        }

        // POST: Doctors/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var doctor = await _context.Doctors.FindAsync(id);
            _context.Doctors.Remove(doctor);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private bool DoctorExists(int id)
        {
            return _context.Doctors.Any(e => e.DoctorId == id);
        }
    }
```

```
}
 using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using BD_Horizont.Data;
using BD_Horizont.Models;

namespace BD_Horizont.Controllers
{
    public class ClientsController : Controller
    {
        private readonly HorizontContext _context;

        public ClientsController(HorizontContext context)
        {
            _context = context;
        }

        // GET: Clients
        public async Task<IActionResult> Index()
        {
            return View(await _context.Clients.ToListAsync());
        }

        // GET: Clients/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var client = await _context.Clients
                .FirstOrDefaultAsync(m => m.ClientId == id);
            if (client == null)
            {
                return NotFound();
            }

            return View(client);
        }

        // GET: Clients/Create
        public IActionResult Create()
        {
            return View();
        }

        // POST: Clients/Create
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("ClientId,ClientName,ClientBirthday,ClientWorkPlace")] Client client)
        {
            if (ModelState.IsValid)
            {
                _context.Add(client);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            return View(client);
```

```
        }

        // GET: Clients/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var client = await _context.Clients.FindAsync(id);
            if (client == null)
            {
                return NotFound();
            }
            return View(client);
        }

        // POST: Clients/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id, [Bind("ClientId,ClientName,ClientBirthday,ClientWorkPlace")] Client
client)
        {
            if (id != client.ClientId)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(client);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!ClientExists(client.ClientId))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            return View(client);
        }

        // GET: Clients/Delete/5
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var client = await _context.Clients
                .FirstOrDefaultAsync(m => m.ClientId == id);
            if (client == null)
            {
```

```
            return NotFound();
        }

        return View(client);
    }

    // POST: Clients/Delete/5
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var client = await _context.Clients.FindAsync(id);
        _context.Clients.Remove(client);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }

    private bool ClientExists(int id)
    {
        return _context.Clients.Any(e => e.ClientId == id);
    }
  }
}
 using BD_Horizont.Models;
using System;
using System.Linq;
namespace BD_Horizont.Data

{
  public class DataInitializer
  {
    public static void Initialize(HorizontContext context)
    {
      context.Database.EnsureCreated();

      // Look for any students.
      if (context.Tickets.Any())
      {
        return;   // DB has been seeded
      }



      var clients = new Client[]
      {
      new Client{ClientName="Maryna", ClientBirthday=DateTime.Parse("2022-10-17"), ClientWorkPlace="OAO
Minsktrans"},
      new Client{ClientName="Maryna", ClientBirthday=DateTime.Parse("2022-10-17"), ClientWorkPlace="OAO
Minsktrans"},
      new Client{ClientName="Maryna", ClientBirthday=DateTime.Parse("2022-10-17"), ClientWorkPlace="OAO
Minsktrans"}
      };
      foreach (Client c in clients)
      {
        context.Clients.Add(c);
      }
      context.SaveChanges();

      var doctors = new Doctor[]
      {
      new Doctor{DoctorName="Sergey", DoctorBirthday=DateTime.Parse("2022-10-12"), DoctorPosition="surgion"},
      new Doctor{DoctorName="Sergey", DoctorBirthday=DateTime.Parse("2022-10-12"), DoctorPosition="surgion"},
      new Doctor{DoctorName="Sergey", DoctorBirthday=DateTime.Parse("2022-10-12"), DoctorPosition="surgion"},
      new Doctor{DoctorName="Sergey", DoctorBirthday=DateTime.Parse("2022-10-12"), DoctorPosition="surgion"}
      };
```

```
            foreach (Doctor d in doctors)
            {
                context.Doctors.Add(d);
            }
            context.SaveChanges();
            var tickets = new Ticket[]
            {
            new Ticket{ Cost=120, WhatHappened="something happend", ClientId=1, DoctorId=1, ResultsOfVisit="text",
VisitDate=DateTime.Parse("2022-02-02")},
            new Ticket{ Cost=120, WhatHappened="something happend", ClientId=2, DoctorId=2, ResultsOfVisit="text",
VisitDate=DateTime.Parse("2022-02-02")},
            new Ticket{ Cost=120, WhatHappened="something happend", ClientId=3, DoctorId=2, ResultsOfVisit="text",
VisitDate=DateTime.Parse("2022-02-02")},
            new Ticket{ Cost=120, WhatHappened="something happend", ClientId=2, DoctorId=2, ResultsOfVisit="text",
VisitDate=DateTime.Parse("2022-02-02")}
            };
            foreach (Ticket t in tickets)
            {
                context.Tickets.Add(t);
            }
            context.SaveChanges();
        }
    }
}
 using BD_Horizont.Models;
using Microsoft.EntityFrameworkCore;

namespace BD_Horizont.Data
{
    public class HorizontContext: DbContext
    {
        public HorizontContext(DbContextOptions<HorizontContext> options) : base(options)
        {
        }

        public DbSet<Ticket> Tickets { get; set; }
        public DbSet<Client> Clients { get; set; }
        public DbSet<Doctor> Doctors { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Ticket>().ToTable("Ticket");
            modelBuilder.Entity<Client>().ToTable("Client");
            modelBuilder.Entity<Doctor>().ToTable("Doctor");
        }
    }
}
 using System;
using System.Collections.Generic;

namespace BD_Horizont.Models
{
    public class Client
    {
        public int ClientId { get; set; }
        public string ClientName { get; set; }
        public DateTime ClientBirthday { get; set; }
        public string ClientWorkPlace { get; set; }

        public ICollection<Ticket> Tickets { get; set; }

    }
}
 using System;
using System.ComponentModel.DataAnnotations;
```

```
namespace BD_Horizont.Models
{
    public class Ticket
    {

        public int TicketId { get; set; }
        public DateTime VisitDate { get; set; }
        public string WhatHappened { get; set; } //что случилось
        public string ResultsOfVisit { get; set; }
        public int Cost { get; set; } // стоимость

        public int ClientId { get; set; } //имя пациента связь
        public Client Client { get; set; }

        public int DoctorId { get; set; } //должность доктора связь
        public Doctor Doctor { get; set; } //имя доктора связь
    }
}
```

ПРИЛОЖЕНИЕ Б
Организационная структура предприятия

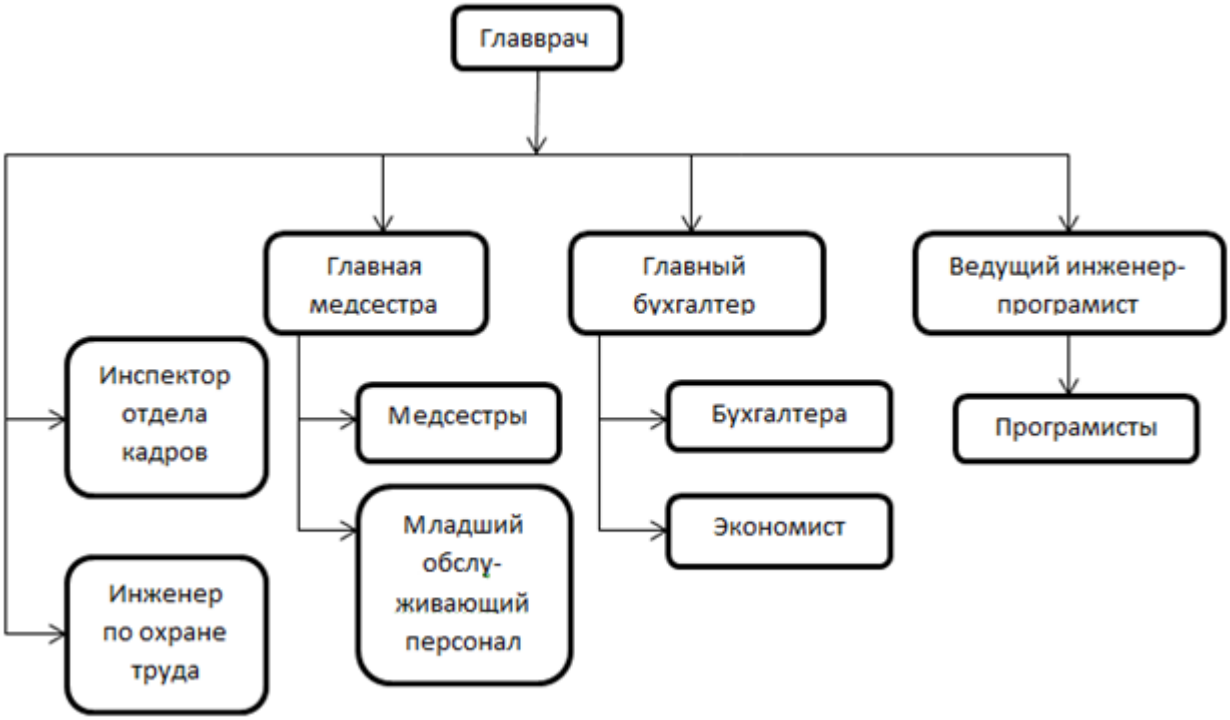

Рисунок Б.1 – Организационная структура предприятия

ПРИЛОЖЕНИЕ В
Инструкция по охране труда при работе с персональным компьютером
(обязательное)

ИНСТРУКЦИЯ ПО ОХРАНЕ ТРУДА ПРИ РАБОТЕ НА ПЕРСОНАЛЬНОМ КОМПЬЮТЕРЕ ИНЖЕНЕРА-ПРОГРАММИСТА

1 ОБЩИЕ ТРЕБОВАНИЯ БЕЗОПАСНОСТИ

1.1 Инструкция по охране труда при работе с персональными компьютерами устанавливает общие требования безопасности для работников, использующих в работе персональные компьютеры (ПК).

1.2 К работе с ПК допускаются работники, не имеющие медицинских противопоказаний, прошедшие инструктаж по вопросам охраны труда, с группой по электробезопасности не ниже 1.

Женщины со времени установления беременности и в период кормления грудью к выполнению всех видов работ, связанных с использованием ПК, не допускаются.

1.3 При работе с ПК на работников могут оказывать неблагоприятное воздействие следующие опасные и вредные производственные факторы:

– повышенный уровень электромагнитных излучений;

– повышенный уровень ионизирующих излучений;

– повышенный уровень статического электричества;

– повышенная напряженность электростатического поля;

– повышенная или пониженная ионизация воздуха;

– повышенная яркость света;

– прямая и отраженная блескость; повышенное значение напряжения в электрической цепи, замыкание которой может произойти через тело человека;

– статические перегрузки костно-мышечного аппарата и динамические локальные перегрузки мышц кистей рук;

– перенапряжение зрительного анализатора;

– умственное перенапряжение;

– эмоциональные перегрузки;

– монотонность труда.

В зависимости от условий труда, в которых применяются ПК, и характера работы на работников могут воздействовать также другие опасные и вредные производственные факторы.

1.4 Организация рабочего места с ПК должна учитывать требования безопасности, удобство положения, движений и действий работника.

Рабочий стол с учетом характера выполняемой работы должен иметь достаточный размер для рационального размещения монитора (дисплея), клавиатуры, другого используемого оборудования и документов, поверхность, обладающую низкой отражающей способностью.

Клавиатура располагается на поверхности стола таким образом, чтобы пространство перед клавиатурой было достаточным для опоры рук работника (на расстоянии не менее чем 300 мм от края, обращенного к работнику).

Чтобы обеспечивалось удобство зрительного наблюдения, быстрое и точное считывание информации, плоскость экрана монитора располагается ниже уровня глаз работника предпочтительно перпендикулярно к нормальной линии взгляда работника (нормальная линия взгляда – 15° вниз от горизонтали).

Для исключения воздействия повышенных уровней электромагнитных излучений расстояние между экраном монитора и работником должно составлять не менее 500 мм (оптимальное 600-700 мм).

Применяемые подвижные подставки для документов (пюпитры) размещаются в одной плоскости и на одной высоте с экраном.

Рабочий стул (кресло) должен быть устойчивым, место сидения должно регулироваться по высоте, а спинка сиденья – по высоте, углам наклона, а также расстоянию спинки от переднего края сиденья. Регулировка каждого параметра должна быть независимой, легко осуществляемой и иметь надежную фиксацию.

Для тех, кому это удобно, предусматривается подставка для ног.

1.5 Рабочее место размещается таким образом, чтобы естественный свет падал сбоку (желательно слева).

Для снижения яркости в поле зрения при естественном освещении применяются регулируемые жалюзи, плотные шторы.

Светильники общего и местного освещения должны создавать нормальные условия освещенности и соответствующий контраст между экраном и окружающей обстановкой с учетом вида работы и требований видимости со стороны работника. Освещенность на поверхности стола в зоне размещения рабочего документа должна составлять 300-500 люкс.

Возможные мешающие отражения и отблески на экране монитора и другом оборудовании устраняются путем соответствующего размещения экрана, оборудования, расположения светильников местного освещения.

При рядном размещении рабочих столов расположение экранов видеомониторов навстречу друг другу из-за их взаимного отражения не допускается.

Для обеспечения безопасности работников на соседних рабочих местах расстояние между рабочими столами с мониторами (в направление тыла поверхности одного монитора и экрана другого монитора) должно быть не менее 2,0 м, а расстояние между боковыми поверхностями мониторов – не менее 1,2 м.

1.6 Для снижения уровня напряженности электростатического поля при необходимости применяются экранные защитные фильтры.

При эксплуатации защитный фильтр должен быть плотно установлен на экране монитора и заземлен.

1.7 Для обеспечения оптимальных параметров микроклимата проводятся регулярное в течение дня проветривание и ежедневная влажная уборка помещений, используются увлажнители воздуха.

1.8 При работе с ПК обеспечивается доступ работников к первичным средствам пожаротушения, аптечкам первой медицинской помощи.

1.9 Работники при работе с ПК с учетом воздействующих на них опасных и вредных производственных факторов обеспечиваются средствами индивидуальной защиты в соответствии с типовыми отраслевыми нормами для соответствующих профессий и должностей.

1.10 При работе с ПК работники обязаны:

– соблюдать режим труда и отдыха, установленный законодательством, правилами внутреннего трудового распорядка организации, трудовую дисциплину, выполнять требования охраны труда, правил личной гигиены;

– выполнять требования пожарной безопасности, знать порядок действий при пожаре, уметь применять первичные средства пожаротушения;

– курить только в специально предназначенных для курения местах;

– знать приемы оказания первой помощи при несчастных случаях на производстве;

– о неисправности оборудования и других замечаниях по работе с ПК сообщать непосредственному руководителю или лицам, осуществляющим техническое обслуживание оборудования.

1.11 Не допускается:

– выполнять работу, находясь в состоянии алкогольного опьянения либо в состоянии, вызванном употреблением наркотических средств, психотропных или токсических веществ, а также распивать спиртные напитки, употреблять наркотические средства, психотропные или токсические вещества на рабочем месте или в рабочее время;

– устанавливать системный блок в закрытых объемах мебели, непосредственно на полу;

– использовать для подключения ПК розетки, удлинители, не оснащенные заземляющим контактом (шиной).

1.12 Работники, не выполняющие требования настоящей Инструкции, привлекаются к ответственности согласно законодательству.

2 ТРЕБОВАНИЯ БЕЗОПАСНОСТИ ПЕРЕД НАЧАЛОМ РАБОТЫ

2.1 Перед началом работы с ПК работник обязан:

2.1.1 Проветрить рабочее помещение;

2.1.2 Проверить:

– устойчивость положения оборудования на рабочем столе;

– отсутствие видимых повреждений оборудования, дискет в дисководе системного блока;

– исправность и целостность питающих и соединительных кабелей, разъемов и штепсельных соединений, защитного заземления (зануления);

– исправность мебели.

2.1.3 Отрегулировать:

– положение стола, стула (кресла), подставки для ног, клавиатуры, экрана монитора;

– освещенность на рабочем месте. При необходимости включить местное освещение;

2.1.4 Протереть поверхность экрана монитора, защитного фильтра (при его наличии) сухой мягкой тканевой салфеткой;

2.1.5 Убедиться в отсутствии отражений на экране монитора, встречного светового потока;

2.1.6 Включить оборудование ПК в электрическую сеть, соблюдая следующую последовательность: стабилизатор напряжения (если он используется), блок бесперебойного питания, периферийные устройства (принтер, монитор, сканер и другие устройства), системный блок.

2.2 Запрещается приступать к работе при:

– выраженном дрожании изображения на мониторе;

– обнаружении неисправности оборудования;

– наличии поврежденных кабелей или проводов, разъемов, штепсельных соединений;

– отсутствии или неисправности защитного заземления (зануления) оборудования.

3 ТРЕБОВАНИЯ БЕЗОПАСНОСТИ ПРИ ВЫПОЛНЕНИИ РАБОТ

3.1 Во время работы с ПК работник обязан:

– соблюдать требования охраны труда, установленные настоящей Инструкцией;

– содержать в порядке и чистоте свое рабочее место;

– держать открытыми вентиляционные отверстия оборудования;

– соблюдать оптимальное расстояние от экрана монитора до глаз.

3.2 Работу за экраном монитора следует периодически прерывать на регламентированные перерывы, которые устанавливаются для обеспечения работоспособности и сохранения здоровья, или заменять другой работой с целью сокращения рабочей нагрузки у экрана.

3.3 Время регламентированных перерывов в течение рабочего дня (смены) устанавливается в зависимости от его (ее) продолжительности, вида и категории трудовой деятельности согласно приложению 1 к настоящей Инструкции.

При 8-часовой рабочей смене и работе с ПК регламентированные перерывы устанавливаются:

– для I категории работ через 2 часа от начала рабочей смены и через 2 часа после обеденного перерыва продолжительностью 15 минут каждый;

– для II категории работ через 2 часа от начала рабочей смены и через 1,5-2 часа после обеденного перерыва продолжительностью 15 минут каждый или продолжительностью 10 минут каждый час работы;

– для III категории работ через 1,5-2 часа от начала рабочей смены и через 1,5-2 часа после обеденного перерыва продолжительностью 20 минут каждый или продолжительностью 15 минут через каждый час работы.

3.4 Продолжительность непрерывной работы с ПК без регламентированного перерыва не должна превышать 2 часов.

3.5 Во время регламентированных перерывов для снижения нервно-эмоционального напряжения, утомления зрительного анализатора, улучшения функционального состояния, нервной, сердечно-сосудистой, дыхательной систем, а также мышц плечевого пояса, рук, спины, шеи и ног целесообразно выполнять комплексы упражнений согласно приложению 2 к настоящей Инструкции.

Работникам с высоким уровнем напряженности труда во время регламентированных перерывов и в конце рабочего дня показана психологическая разгрузка в специально оборудованных комнатах психологической разгрузки.

3.6 С целью уменьшения отрицательного влияния монотонности необходимо применять чередование операций.

При работе с текстовой информацией следует отдавать предпочтение физиологически наиболее оптимальному режиму представления черных символов на белом фоне.

3.7 Не следует оставлять оборудование включенным без наблюдения. При необходимости прекращения на некоторое время работы корректно закрываются все активные задачи и оборудование выключается.

3.8 При работе с ПК не разрешается:

– при включенном питании прикасаться к панелям с разъемами оборудования, разъемами питающих и соединительных кабелей, экрану монитора;

– загромождать верхние панели оборудования, рабочее место бумагами, посторонними предметами;

– производить переключения, отключение питания во время выполнения активной задачи;

– допускать попадание влаги на поверхность оборудования;

– включать сильно охлажденное (принесенное с улицы в зимнее время) оборудование;

– производить самостоятельно вскрытие и ремонт оборудования;

– вытирать пыль на включенном оборудовании;

– допускать нахождение вблизи оборудования посторонних лиц.

4 ТРЕБОВАНИЯ БЕЗОПАСНОСТИ В АВАРИЙНЫХ СИТУАЦИЯХ

4.1 В аварийных (экстремальных) ситуациях необходимо:

– при повреждении оборудования, кабелей, проводов, неисправности заземления, появлении запаха гари, возникновении необычного шума и других неисправностях немедленно отключить электропитание оборудования и сообщить о случившемся непосредственному руководителю и лицу, осуществляющему техническое обслуживание оборудования;

– в случае сбоя в работе оборудования ПК или программного обеспечения вызвать специалиста организации, осуществляющего техническое обслуживание данного оборудования, для устранения неполадок;

– при возгорании электропроводки, оборудования и тому подобных происшествиях отключить электропитание и принять меры по тушению пожара с помощью имеющихся первичных средств пожаротушения, сообщить о происшедшем непосредственному руководителю. Применение воды и пенных огнетушителей для тушения находящегося под напряжением электрооборудования недопустимо. Для этих целей используются углекислотные огнетушители.

В случае внезапного ухудшения здоровья (усиления сердцебиения, появления головной боли и других) прекратить работу, выключить оборудование, сообщить об этом руководителю и при необходимости обратиться к врачу.

4.2 При несчастном случае на производстве необходимо:

– быстро применять меры по предотвращению воздействия на потерпевшего травмирующих факторов, оказанию потерпевшему первой помощи, вызову на место происшествия медицинских работников или доставке потерпевшего в организацию здравоохранения;

– сообщить о происшествии руководителю.

5 ТРЕБОВАНИЯ БЕЗОПАСНОСТИ ПО ОКОНЧАНИИ РАБОТЫ

По окончании работы работник обязан:

– корректно закрыть все активные задачи;

– при наличии дискеты в дисководе извлечь ее;

– выключить питание системного блока;

– выключить питание всех периферийных устройств;

– отключить блок бесперебойного питания;

– отключить стабилизатор напряжения (если он используется);

– отключить питающий кабель от сети;

– осмотреть и привести в порядок рабочее место;

– о неисправности оборудования и других замечаний по работе с ПК сообщить непосредственному руководителю или лицам, осуществляющим техническое обслуживание оборудования;

– при необходимости вымыть с мылом руки.

ПРИЛОЖЕНИЕ Г
Должностная инструкция инженера-программиста
(обязательное)

1 ОБЩИЕ ПОЛОЖЕНИЯ

1.1. Инженер-программист (программист) относится к категории специалистов.

1.2. Инженер-программист (программист) назначается на должность и освобождается от нее приказом руководителя организации по представлению руководителя структурного подразделения (иного должностного лица).

1.3. На должность:

– инженера-программиста (программиста) назначается лицо, имеющее высшее профессиональное (техническое или инженерно-математическое, математическое) образование без предъявления требований к стажу работы или среднее специальное (техническое или инженерно-математическое, математическое) образование и стаж работы в должности техника-программиста I квалификационной категории не менее 3 лет либо на других должностях, замещаемых специалистами со средним специальным образованием, не менее 5 лет;

– инженера-программиста (программиста) II квалификационной категории назначается лицо, имеющее высшее профессиональное (техническое или инженерно-математическое, математическое) образование и стаж работы в должности инженера-программиста (программиста) или на других инженерно-технических должностях, замещаемых специалистами с высшим профессиональным образованием, не менее 3 лет;

– инженера-программиста (программиста) I квалификационной категории назначается лицо, имеющее высшее профессиональное (техническое или инженерно-математическое, математическое) образование и стаж работы в должности инженера-программиста (программиста) II квалификационной категории не менее 3 лет.

1.4. В своей деятельности инженер-программист (программист) руководствуется:

– нормативными правовыми актами, другими руководящими и методическими материалами, регламентирующими разработку алгоритмов, программ и использование вычислительной техники при обработке информации;

– уставом организации;

– приказами, распоряжениями руководителя организации (непосредственного руководителя);

– настоящей должностной инструкцией.

1.5. Инженер-программист (программист) должен знать:

– руководящие и нормативные материалы, регламентирующие методы разработки алгоритмов, программ и использования вычислительной техники при обработке информации;

– основные принципы структурного программирования;

– виды программного обеспечения;

– технико-эксплуатационные характеристики, конструктивные особенности, назначение и режимы работы ЭВМ, правила их технической эксплуатации;

– технологию автоматизированной обработки информации;

– виды технических носителей информации;

– методы классификации и кодирования информации;

– формализованные языки программирования;

– действующие стандарты, системы счислений, шифров и кодов;

– порядок оформления технической документации;

– передовой отечественный и зарубежный опыт программирования и использования вычислительной техники;

– основы экономики, организации производства, труда и управления;

– основы трудового законодательства;

– правила и нормы охраны труда и пожарной безопасности.

1.6. В случае временного отсутствия инженера-программиста (программиста) его обязанности исполняет лицо, назначенное приказом руководителя организации, которое несет ответственность за надлежащее их исполнение.

2 ДОЛЖНОСТНЫЕ ОБЯЗАННОСТИ

Инженер-программист (программист) осуществляет следующие обязанности:

– на основе анализа математических моделей и алгоритмов решения научных, прикладных экономических и других задач разрабатывает программы, обеспечивающие возможность выполнения алгоритмов и соответственно поставленных задач средствами вычислительной техники, проводит их отладку и тестирование;

– разрабатывает технологию решения задач на всех этапах обработки информации;

– осуществляет выбор языка программирования для описания алгоритмов и структур данных;

– определяет информацию, подлежащую обработке средствами вычислительной техники, ее объемы, структуру, макеты и схемы ввода, обработки, хранения и вывода, методы ее контроля;

– выполняет работу по подготовке программ к отладке и проводит отладку;

– определяет объем и содержание данных контрольных примеров, обеспечивающих наиболее полную проверку соответствия программ их функциональному назначению;

– осуществляет запуск отлаженных программ и ввод исходных данных, определяемых условиями поставленных задач;

– проводит тестирование и корректировку разработанной программы на основе анализа выходных данных;

– разрабатывает инструкции по работе с программами, оформляет необходимую техническую документацию;

– осваивает и применяет в работе новые компьютерные технологии;

– определяет возможность использования готовых программных продуктов;

– осуществляет сопровождение внедренных программ и программных средств;

– разрабатывает и внедряет системы автоматической проверки правильности программ, типовые и стандартные программные средства, составляет технологию обработки информации;

– выполняет работу по унификации и типизации вычислительных процессов;

– принимает участие в создании каталогов и картотек стандартных программ, в разработке форм документов, подлежащих машинной обработке, в проектировании программ, позволяющих расширить область применения вычислительной техники.

3 ПРАВА ИНЖЕНЕРА-ПРОГРАММИСТА

Инженер-программист (программист) имеет право:

– знакомиться с проектами решений руководства организации, касающихся его деятельности;

– вносить предложения по совершенствованию работы, связанной с предусмотренными настоящей должностной инструкцией обязанностями;

– в пределах своей компетенции сообщать непосредственному руководителю обо всех недостатках в деятельности организации (структурного подразделения, отдельных работников), выявленных в процессе исполнения своих должностных обязанностей, и вносить предложения по их устранению;

– запрашивать лично или по поручению руководства организации от подразделений организации и иных специалистов информацию и документы, необходимые для выполнения своих должностных обязанностей;

– требовать от руководства организации оказания содействия в исполнении своих должностных обязанностей.

4 ВЗАИМООТНОШЕНИЯ (СВЯЗИ НА ДОЛЖНОСТИ)

4.1. Инженер-программист (программист) подчиняется непосредственно руководителю структурного подразделения (иному должностному лицу).

4.2. Инженер-программист (программист) осуществляет взаимодействие с работниками структурных подразделений организации по вопросам, входящим в его компетенцию:

– получает технические задания на разработку программного обеспечения;

– предоставляет программное обеспечение, техническую документацию и инструкции по работе с разработанными программами.

5 ОЦЕНКА РАБОТЫ И ОТВЕТСТВЕННОСТЬ

5.1. Результаты работы инженера-программиста (программиста) оценивает руководитель структурного подразделения (иное должностное лицо).

5.2. Инженер-программист (программист) несет ответственность за:

– неисполнение (ненадлежащее исполнение) своих должностных обязанностей;

– несоблюдение правил внутреннего трудового распорядка, правил и норм охраны труда и пожарной безопасности;

– причинение материального ущерба организации - в соответствии с действующим законодательством.