Министерство просвещения Приднестровской Молдавской Республики

Государственное образовательное учреждение   
среднего профессионального образования   
«Тираспольский техникум информатики и права»

КУРСОВАЯ РАБОТА

**на тему «Разработать программную систему для ветеринарной клиники»**

по учебной дисциплине «Разработка программных модулей»

по специальности 09.02.06 «Информационные системы и программирование»

Выполнил

Белько Владимир Иванович

обучающийся II курса

специальность 09.02.06 «Сетевое и системное администрирование»

Руководитель

Подсекина Татьяна Сергеевна

преподаватель информационных дисциплин

Дата защиты проекта

«\_\_»\_\_\_\_\_\_\_2025 г.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Тирасполь 2025

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc198400908)

[ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧ 6](#_Toc198400909)

[1.1. История развития автоматизации ветеринарных клиник 6](#_Toc198400910)

[1.2. Организационная структура ветеринарной клиники 7](#_Toc198400911)

[1.3. Функции, выполняемые информационной системой ветеринарной клиники 8](#_Toc198400912)

[1.4. Постановка задачи 9](#_Toc198400913)

[ГЛАВА 2. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ 11](#_Toc198400914)

[2.1. Требования к программе 11](#_Toc198400915)

[2.2. Руководство пользователя 11](#_Toc198400916)

[2.3. Руководство программиста 15](#_Toc198400917)

[ЗАКЛЮЧЕНИЕ 17](#_Toc198400918)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 18](#_Toc198400919)

[ПРИЛОЖЕНИЕ А 19](#_Toc198400920)

# ВВЕДЕНИЕ

В современном мире информационные технологии оказывают значительное влияние на развитие различных сфер, включая здравоохранение. Ветеринарная медицина не является исключением, и автоматизация процессов управления в ветеринарных клиниках становится важным шагом к повышению качества обслуживания, улучшению эффективности работы и оптимизации процессов. Программа для хранения базы данных будет всегда актуальной темой для современного мира. Разработка программных систем для ветеринарных клиник позволяет упростить процессы ведения медицинской документации, учета животных, планирования приёмов и улучшения взаимодействия с клиентами.

Целью курсовой работы является разработка программной системы для ветеринарной клиники на языке программирования C#. Язык программирования C# выбран для реализации проекта, так как он является одним из самых популярных и мощных инструментов для создания различных приложений, включая решения для автоматизации бизнес-процессов. C# обладает множеством библиотек и инструментов, позволяющих эффективно создавать как декстопные приложения, так и системы с графическими интерфейсами, что идеально подходит для разработки программного обеспечения для ветеринарных клиник.

Программная система, предназначенная для ветеринарной клиники, должна обеспечивать удобное ведение базы данных пациентов (животных), учет истории болезней, запись на приёмы, управление расписанием врачей, а также предоставление отчетности для анализа работы клиники. Кроме того, система должна обеспечивать возможность взаимодействия с клиентами, например, через отправку уведомлений о назначенных визитах или обновлениях состояния здоровья животных.

Задачи, которые необходимо решить для достижения изначально поставленной нам цели:

1. Изучить теоретические основы работы ветеринарных клиник, процессы, которые могут быть автоматизированы с помощью программного обеспечения, и требования к таким системам.
2. Разработать структуру и базу данных для хранения информации о пациентах, их заболеваниях, назначениях, приёмах и сотрудниках клиники.
3. Создать графический интерфейс пользователя, интуитивно понятный для сотрудников клиники, с возможностью добавления, редактирования и поиска данных.
4. Реализовать функционал управления расписанием врачей, записей на приёмы, а также систему уведомлений для клиентов.
5. Провести тестирование системы и обеспечить её стабильную работу.

Объектом исследования является процесс разработки, проектировка и реализации программной системы для ветеринарной клиники с использованием языка программирования C#. Предметом исследования являются методы программирования, проектирования и реализации программных решений, а также подходы к автоматизации процессов управления в ветеринарной практике.

В ходе работы будут использованы методы: анализ (для изучения процессов и потребностей клиники), синтез (для создания программного кода), а также моделирование (для тестирования функционала системы и её взаимодействия с пользователями).

Разработка программной системы для ветеринарной клиники на языке C# позволит продемонстрировать возможности программирования в сфере здравоохранения и автоматизации, а также продвинуться в создании эффективных и удобных инструментов для пользователей. Такие системы могут значительно облегчить работу ветеринарных специалистов и повысить уровень обслуживания клиентов, а также служат основой для разработки более сложных решений в области медицинской автоматизации.

Курсовая работа состоит из введения, двух глав, заключения и списка использованных источников. В первой главе рассматриваются теоретические основы разработки программных систем для ветеринарных клиник, а во второй главе представлена практическая реализация системы с графическим интерфейсом и функционалом для автоматизации процессов работы клиники.

# ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧ

# 1.1. История развития автоматизации ветеринарных клиник

Автоматизация ветеринарных клиник – относительно новое, но активно развивающееся направление. Изначально ветеринарные клиники, как и многие другие медицинские учреждения, вели учет пациентов и документацию вручную. Это было трудоемким процессом, подверженным ошибкам и требующим больших затрат времени.

С развитием компьютерных технологий в 20 веке начались попытки автоматизировать некоторые процессы, такие как ведение картотеки пациентов и финансовый учет. Однако, широкое распространение специализированные программные системы для ветеринарных клиник получили лишь в последние десятилетия.

Этому способствовали несколько факторов:

Рост числа ветеринарных клиник и увеличение объема данных. С увеличением количества домашних животных и развитием ветеринарной медицины, клиникам стало сложно эффективно обрабатывать и хранить информацию о пациентах, владельцах, лечении и финансах вручную.

Потребность в повышении эффективности работы. Автоматизация позволяет сократить время на выполнение рутинных задач, таких как заполнение документов, поиск информации и составление отчетов, что дает возможность ветеринарным врачам уделять больше времени пациентам.

Необходимость улучшения качества обслуживания клиентов. Современные программные системы предоставляют инструменты для автоматизации записи на прием, рассылки напоминаний, ведения истории болезни и обеспечения более удобного взаимодействия с владельцами животных.

Развитие информационных технологий. Появление доступных и мощных компьютеров, развитие сетей Интернет, технологий и разработка удобного программного обеспечения сделали автоматизацию ветеринарных клиник реальностью.

Современные программные системы для ветеринарных клиник предлагают широкий спектр функций, включая:

* Ведение электронной медицинской карты животного.
* Учет владельцев животных.
* Планирование и управление расписанием приемов.
* Управление финансами и выставление счетов.
* Генерация отчетов и статистики.
* Интеграция с лабораторным оборудованием.
* Автоматизация рассылки напоминаний и уведомлений.

Автоматизация ветеринарных клиник продолжает развиваться, и в будущем можно ожидать появления еще более совершенных и удобных систем, которые помогут ветеринарным врачам оказывать качественную медицинскую помощь животным.

# 1.2. Организационная структура ветеринарной клиники

Организационная структура ветеринарной клиники может варьироваться в зависимости от ее размера, специализации и формы собственности. Однако, существуют общие элементы, характерные для большинства клиник.

1. Руководство. Как правило, во главе клиники стоит главный врач или директор. Он отвечает за общее управление клиникой, разработку стратегии развития, принятие ключевых решений и контроль за работой персонала.
2. Ветеринарные врачи. Это специалисты, которые занимаются диагностикой, лечением и профилактикой заболеваний животных. В зависимости от размера клиники, могут быть врачи общей практики и узкие специалисты (хирурги, кардиологи, дерматологи и др.).
3. Ассистенты ветеринарных врачей (фельдшеры). Они помогают врачам в проведении процедур, уходе за животными, подготовке инструментов и ведении документации.
4. Администраторы (регистраторы). Они отвечают за прием и регистрацию пациентов, ведение записи на прием, ответы на телефонные звонки, оформление документов и расчет с клиентами.
5. Обслуживающий персонал. Включает санитаров, уборщиков и других сотрудников, обеспечивающих чистоту и порядок в клинике.

В крупных клиниках могут быть и другие подразделения, такие как:

* Лаборатория. Для проведения анализов и исследований.
* Аптека. Для продажи ветеринарных препаратов.
* Стационар. Для содержания животных, нуждающихся в длительном лечении или наблюдении.

Эффективная организационная структура является важным фактором успешной работы ветеринарной клиники. Она обеспечивает четкое распределение обязанностей, координацию действий персонала и создание благоприятной атмосферы для работы и обслуживания клиентов.

# 1.3. Функции, выполняемые информационной системой ветеринарной клиники

Информационная система ветеринарной клиники выполняет множество важных функций, направленных на автоматизацию и оптимизацию рабочих процессов.

Основные функции:

Управление данными о пациентах. Система позволяет хранить и обрабатывать информацию о животных (кличка, вид, порода, возраст, история болезни) и их владельцах (ФИО, контактные данные). Это обеспечивает быстрый доступ к необходимой информации и уменьшает вероятность потери части данных.

Управление приемами и расписанием. Система автоматизирует процесс записи на прием, позволяет вести расписание врачей, отправлять напоминания о предстоящих визитах и оптимизировать загрузку клиники.

Ведение медицинской документации. Система позволяет создавать и хранить электронные медицинские карты, протоколы осмотров, результаты анализов и другие документы, что облегчает работу врачей и обеспечивает сохранность информации.

Управление финансами. Система автоматизирует учет оказанных услуг, выставление счетов, прием оплаты и ведение финансовой отчетности, что снижает вероятность ошибок и упрощает финансовое управление всей клиникой в целом.

Формирование отчетов и статистики. Система позволяет генерировать различные отчеты (отчет о посещаемости, отчет о доходах, отчет о наиболее распространенных заболеваниях) и анализировать данные, что помогает руководству клиники принимать обоснованные управленческие решения.

Взаимодействие с клиентами. Система может включать функции онлайн-записи на прием, рассылки напоминаний, отправки результатов анализов по электронной почте и другие инструменты, направленные на повышение удобства и лояльности клиентов.

Внедрение информационной системы в ветеринарной клинике позволяет:

* Повысить эффективность работы персонала.
* Улучшить качество обслуживания пациентов и клиентов.
* Снизить затраты времени и ресурсов на выполнение рутинных задач.
* Уменьшить количество ошибок и потерь данных.
* Обеспечить более прозрачное и эффективное управление клиникой.

# 1.4. Постановка задачи

Целью данной курсовой работы является разработка программной системы для ветеринарной клиники, которая автоматизирует основные процессы учета и управления.

Система должна обеспечивать выполнение следующих задач:

1. Управление данными о животных. Система должна позволять добавлять, редактировать, удалять и просматривать информацию о животных, включая кличку, вид, породу, возраст и данные о владельце.
2. Управление данными о владельцах. Система должна обеспечивать хранение и обработку информации о владельцах животных, включая ФИО, контактные данные и список принадлежащих им животных.
3. Управление данными о ветеринарах. Система должна позволять добавлять, редактировать, удалять и просматривать информацию о ветеринарных врачах, работающих в клинике, включая специализацию.
4. Управление приемами. Система должна обеспечивать возможность записи животных на прием, ведение расписания врачей, учет оказанных услуг, выставление счетов и формирование истории посещений.
5. Формирование отчетов. Система должна генерировать отчеты о посещаемости, доходах и другие статистические данные.
6. Сохранение и загрузка данных. Система должна обеспечивать возможность сохранения данных в файл и загрузки данных из файла.

Разрабатываемая система должна быть удобной в использовании, надежной и обеспечивать защиту данных.

# ГЛАВА 2. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

# 2.1. Требования к программе

Разрабатываемая программа должна соответствовать следующим требованиям:

1. Функциональные требования:

Реализация функций управления данными о животных, владельцах, ветеринарах и приемах в соответствии с поставленными задачами;

Обеспечение возможности формирования отчетов и статистики;

Реализация функций сохранения и загрузки данных из файла.

Также необходимо реализовать поставленные нам функции:

Регистрация клиента; Регистрация животного; Запись на приём; Ведение истории приёмов; Ведение медицинских записей (История болезни); Фиксация диагноза; Назначение лечения; Результаты обследований; Вакцинации и дегельминтизации; Управление услугами и прейскурантом; Справочник услуг; Формирование счёта; Управление пользователями и ролями; Отчетность.

1. Требования к интерфейсу:

Интуитивно понятный и удобный пользовательский интерфейс;

Обеспечение логичной навигации между функциями программы;

Отображение информации в четком и структурированном виде.

1. Технические требования:

Программа должна быть разработана на языке программирования C#;

Обеспечение корректной работы программы в операционной системе Windows 7–11;

Обеспечение надежности и стабильности работы программы.

1. Требования к данным:

Обеспечение целостности и сохранности данных;

Обеспечение возможности резервного копирования и восстановления потерянных данных.

# 2.2. Руководство пользователя

Данный раздел содержит описание основных функций программы и инструкции по их использованию.

1. Главное меню. После запуска программы открывается главное меню, которое предоставляет доступ ко всем основным функциям системы.

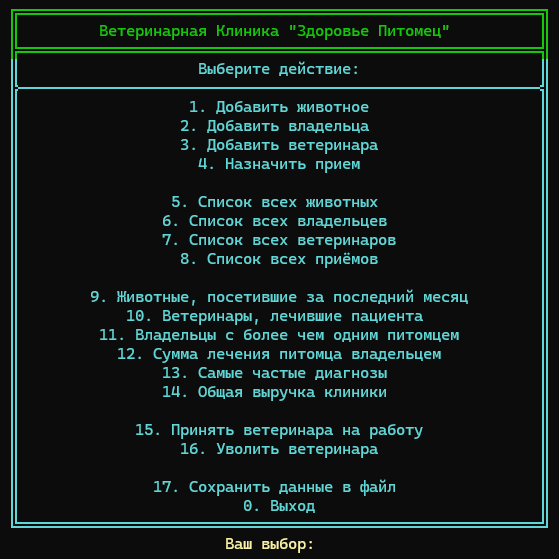


Рис. 1 Меню программы.

1. Управление владельцами. Данная функция позволяет добавлять, редактировать, удалять и просматривать информацию о владельцах животных.

Добавление владельца. Для добавления нового владельца необходимо выбрать пункт "Добавить владельца" в главном меню. Затем программа запросит следующую информацию:

ФИО владельца;

Телефон владельца;

После ввода всех данных владелец будет добавлен в систему.

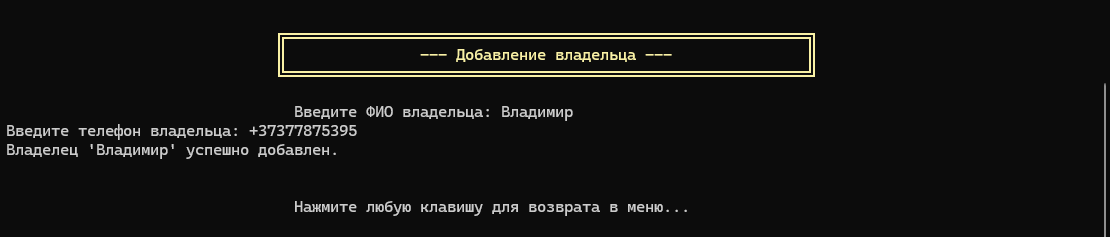


Рис. 2 Ввод данных о владельце.

Просмотр информации о владельцах. Для просмотра информации обо всех владельцах необходимо выбрать пункт "Просмотреть список владельцев" в главном меню. Программа выведет на экран список всех владельцев с указанием их основных данных и списка принадлежащих им животных.

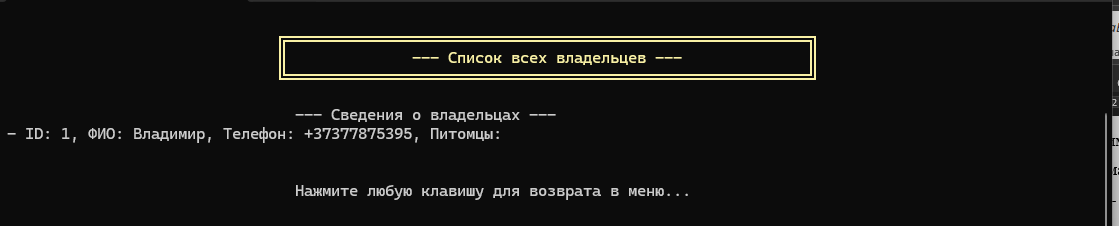


Рис. 3 Вывод информации о всех владельцах.

1. Управление животными. Данная функция позволяет добавлять, редактировать, удалять и просматривать информацию о животных.

Добавление животного.

Для добавления нового животного необходимо выбрать пункт "Добавить животное" в главном меню. Затем программа запросит следующую информацию:

Имя питомца; Вид питомца; Порода питомца; Возраст питомца; Кличка питомца; ФИО владельца.

После ввода всех данных животное будет добавлено в систему.

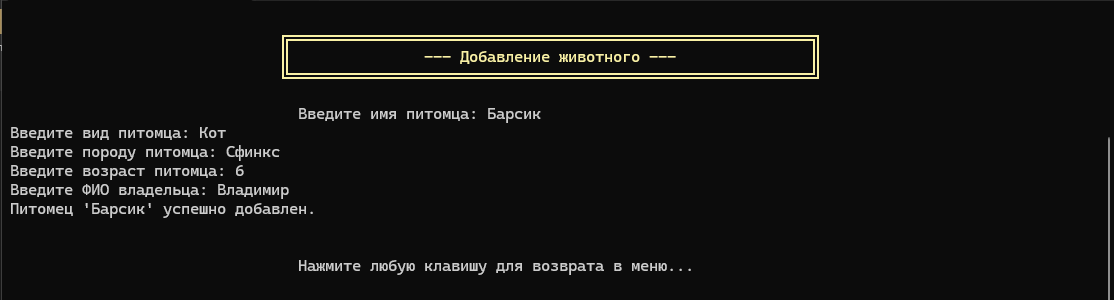


Рис. 4 Ввод информации о питомце и указание владельца.

Просмотр информации о животных. Для просмотра информации обо всех животных необходимо выбрать пункт "Просмотреть список животных" в главном меню. Программа выведет на экран список всех животных с указанием их основных данных.

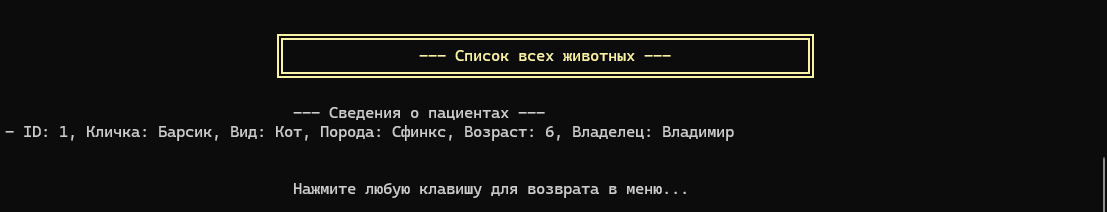


Рис 5. Вывод информации о всех питомцах.

1. Управление ветеринарами. Данная функция позволяет добавлять, редактировать, удалять и просматривать информацию о ветеринарах.

Добавление ветеринара. Для добавления нового ветеринара необходимо выбрать пункт "Добавить ветеринара" в главном меню. Затем программа запросит следующую информацию:

ФИО ветеринара.

Специализация ветеринара.

После ввода всех данных ветеринар будет добавлен в систему.

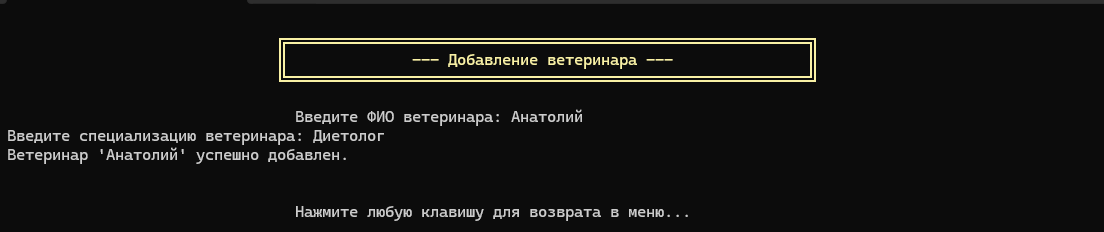


Рис 6. Назначение ветеринара и указание информации о нем.

Удаление ветеринара. Для удаления ветеринара необходимо выбрать пункт "Удалить ветеринара" в главном меню. Затем программа запросит ID ветеринара, которого необходимо удалить, и запросит подтверждение операции.

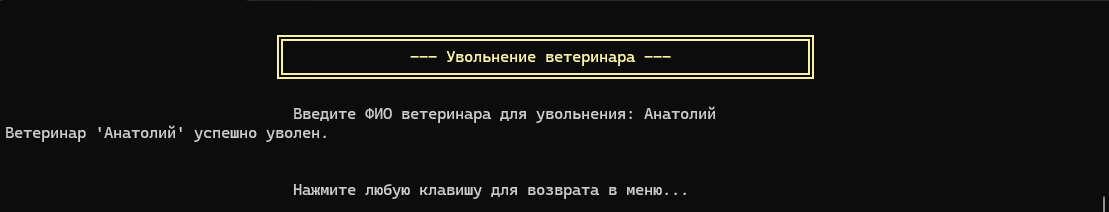


Рис. 7 Увольнение ветеринара из клиники.

Просмотр информации о ветеринарах. Для просмотра информации обо всех ветеринарах необходимо выбрать пункт "Просмотреть список ветеринаров" в главном меню. Программа выведет на экран список всех ветеринаров с указанием их основных данных.

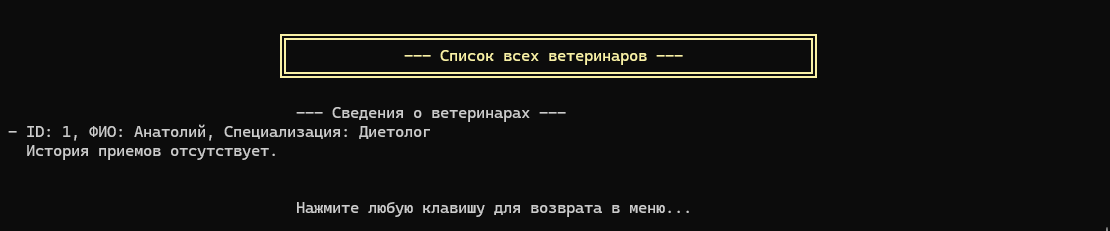


Рис. 8 Получение информации о всех ветеринарах клиники.

1. Управление приемами. Данная функция позволяет добавлять, редактировать, удалять и просматривать информацию о приемах животных.

Добавление приема. Для добавления нового приема необходимо выбрать пункт "Добавить прием" в главном меню. Затем программа запросит следующую информацию: Кличка животного; ФИО ветеринара; Дата приема; Диагноз; Назначенное лечение; Стоимость приема.

После ввода всех данных прием будет добавлен в систему.

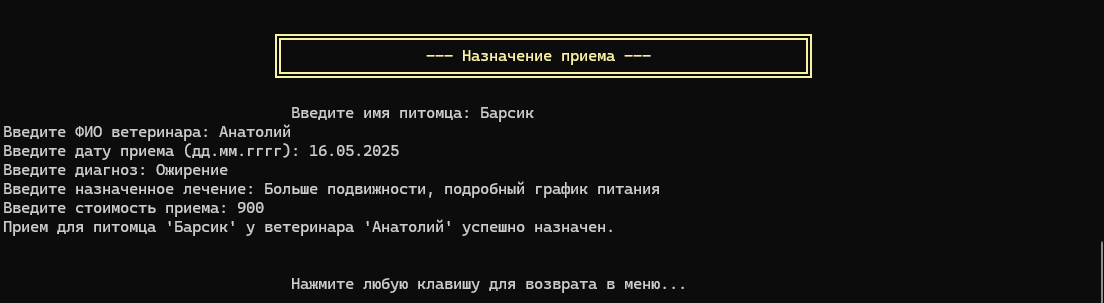


Рис. 9 Назначение приема и указание дополнительных данных.

Просмотр информации о приемах. Для просмотра информации обо всех приемах необходимо выбрать пункт "Просмотреть список приемов" в главном меню. Программа выведет на экран список всех приемов с указанием их основных данных.

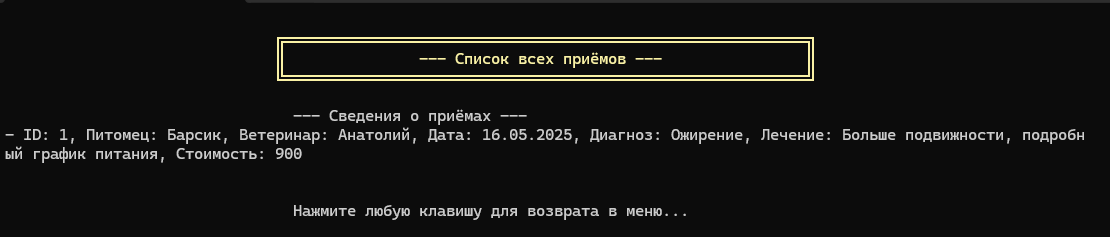


Рис. 10 Вывод информации о всех приемах.

1. Формирование отчетов. Данная функция позволяет генерировать различные отчеты на основе данных, хранящихся в системе.

Отчет о посещаемости. Данный отчет содержит информацию о количестве посещений клиники за последний месяц.

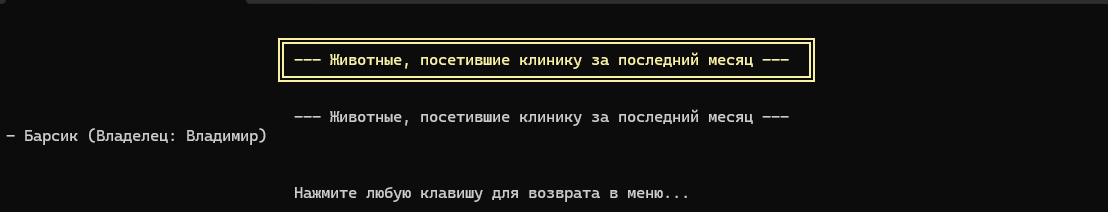


Рис. 11 Вывод информации о животных посетивших клинику за последний месяц.

Отчет о доходах. Данный отчет содержит информацию об общих доходах клиники.

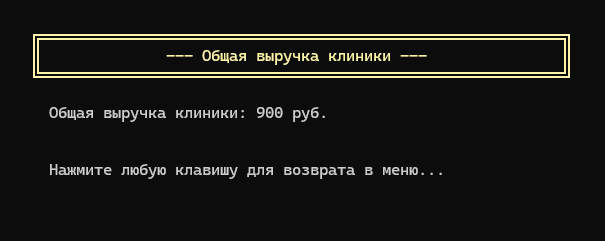


Рис. 12 Получение данных об общей выручке клиники.

Отсчет о затратах на питомца. Данный отчет содержит затраты на одного питомца учитывая все приёмы, которые он посетил.

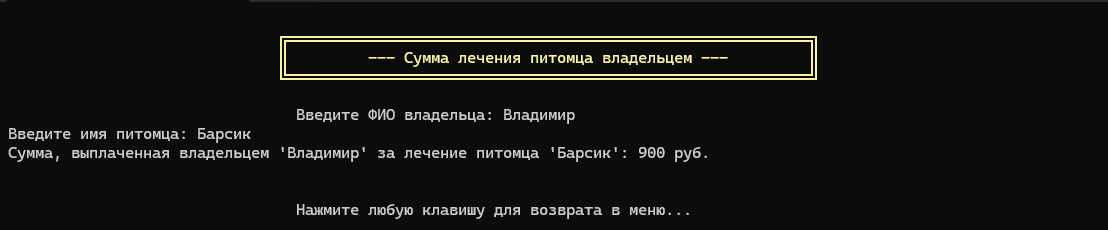


Рис. 13 Вывод информации о затратах на указанного питомца.

Отсчет о посещении. Указывает на ветеринаров, которых посетил указанный пациент.

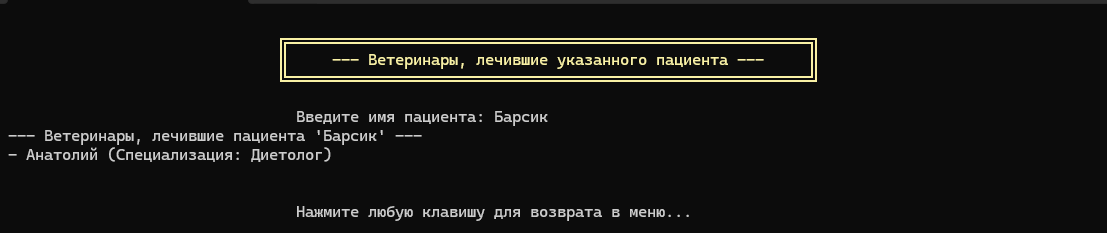


Рис. 14 Вывод информации о ветеринарах, лечивших указанного пациента.

Отсчет о диагнозах. Подразумевает выявление самых частых диагнозов.

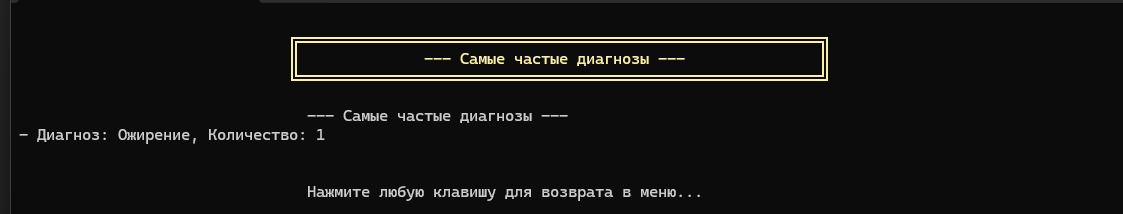


Рис. 15 Получение информации о самых частых диагнозах в клинике.

Отсчет о владении. Указывает на владельцев, имеющих более 1 питомца.

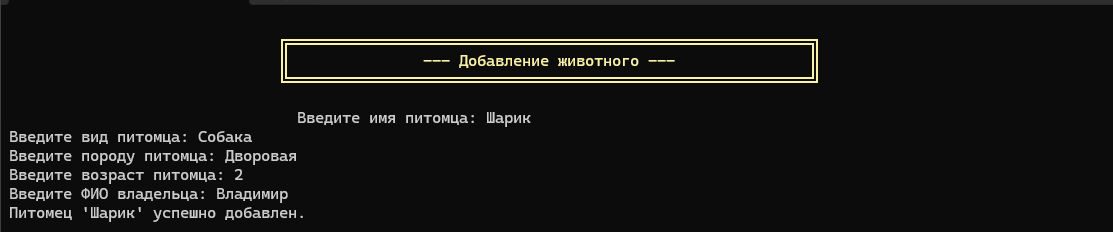


Рис. 16.1 Добавление дополнительного питомца для вывода информации о владении.

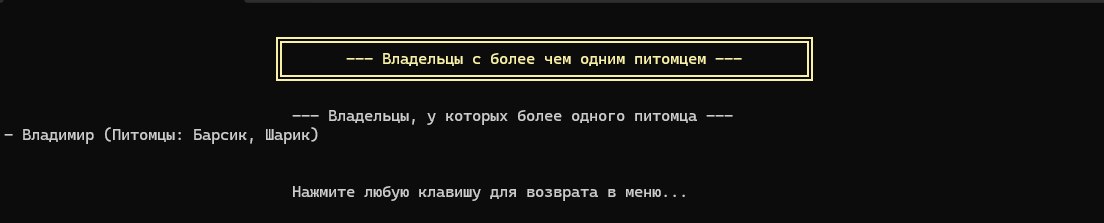


Рис. 16.2 Вывод информации о владельцах, имевших более одного питомца.

1. Сохранение и загрузка данных. Данные могут быть сохранены в файл для резервного копирования или переноса на другой компьютер, а также загружены из файла.

Сохранение данных. Для сохранения данных необходимо выбрать пункт "Сохранить данные" в главном меню и указать имя файла.

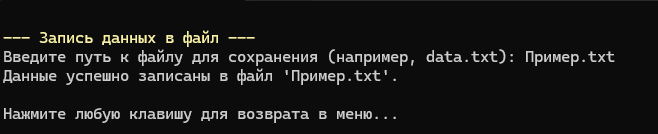


Рис. 17.1 Указание названия текстового файла при сохранении.

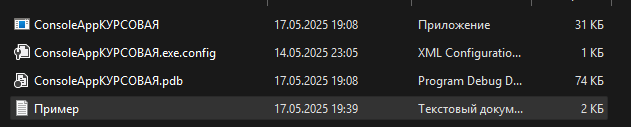


Рис. 17.2 Сохраненный файл успешно создан в дериктории.

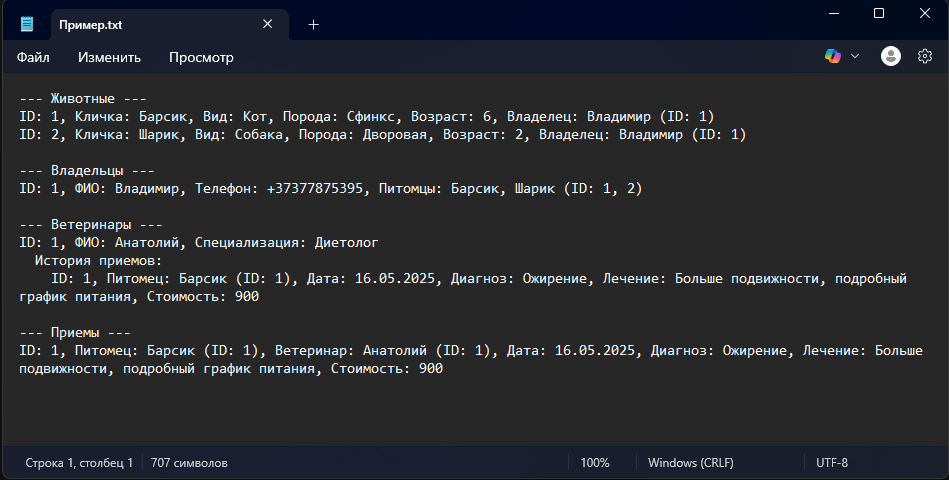


Рис. 17.3 Сохраненная пользователем информация в текстовом файле.

# 2.3. Руководство программиста

Данный раздел содержит описание структуры программы, используемых классов и методов, а также инструкции по расширению и модификации программы.

1. Структура программы. Программа разработана на языке C# и представляет собой консольное приложение. Основные классы программы:

Animal: класс, представляющий информацию о животном.

Owner: класс, представляющий информацию о владельце животного.

Veterinarian: класс, представляющий информацию о ветеринарном враче и его специальности.

Appointment: класс, представляющий информацию о приеме указанного животного.

Actions: класс, содержащий методы для выполнения основных операций над данными (добавление, редактирование, удаление, просмотр).

1. Описание классов.

Класс Animal; Класс Owner; Класс Veterinarian; Класс Appointment; Класс Actions.

1. Расширение и модификация программы.

Для добавления новых функций необходимо добавить соответствующие методы в класс Actions и вызвать их из главного меню.

Для изменения структуры данных необходимо изменить классы Animal, Owner, Veterinarian, Appointment.

Для реализации графического интерфейса необходимо использовать Windows Forms или WPF.

# ЗАКЛЮЧЕНИЕ

В процессе выполнения данной курсовой работы была успешно разработана и реализована программная система, предназначенная для автоматизации ключевых бизнес-процессов ветеринарной клиники. Это достижение является результатом комплексного подхода, включающего в себя глубокий анализ предметной области, тщательное проектирование архитектуры системы и последовательную имплементацию функциональных требований.

1. Детальный анализ предметной области и ее специфики: На начальном этапе работы было проведено всестороннее исследование ветеринарной отрасли, ее истории и современного состояния в контексте информационных технологий. Мы изучили типовую организационную структуру ветеринарных клиник, выявили основные потоки данных и ключевые операции, такие как запись на прием, ведение амбулаторных карт, учет оказанных услуг и выписанных медикаментов. Особое внимание было уделено специфике работы с животными-пациентами, их уникальными идентификаторами и потребностями в долгосрочном наблюдении. Это позволило сформировать комплексное понимание реальных потребностей пользователей и определить оптимальный набор функций для разрабатываемой системы. Понимание специфики позволило избежать типичных ошибок при автоматизации, связанных с недостаточным учетом отраслевых нюансов.

2. Формулирование и детализация функциональных и нефункциональных требований: На основе глубокого анализа предметной области были четко сформулированы и детализированы все необходимые требования к программной системе. Функциональные требования включали в себя такие аспекты, как управление базой данных клиентов и пациентов, планирование расписания приемов, регистрация оказанных ветеринарных услуг, ведение истории болезни животных, а также генерация различных отчетов. Нефункциональные требования охватывали аспекты производительности, надежности, удобства использования, безопасности данных и масштабируемости. Применение методологии сбора требований позволило обеспечить полноту и непротиворечивость спецификаций, что стало прочным фундаментом для последующих этапов разработки.

3. Разработка гибкого и масштабируемого архитектурного решения: Выбранная архитектура системы основывается на принципах модульности и расширяемости. Это означает, что система состоит из независимых, но взаимодействующих компонентов, что значительно упрощает ее дальнейшую модификацию и адаптацию к изменяющимся требованиям. Использование объектно-ориентированного подхода позволило создать легко поддерживаемый и тестируемый код. Архитектура предусматривает возможность интеграции с внешними системами в будущем, например, с лабораторным оборудованием или платежными шлюзами, что обеспечивает долгосрочную актуальность разработанного решения.

4. Реализация всеобъемлющего функционала для эффективного управления клиникой: В ходе реализации был создан полноценный инструментарий для автоматизации ключевых операций ветеринарной клиники. Система позволяет:

Вести полную базу данных: хранить информацию о владельцах животных (контактные данные), а также подробные сведения о каждом пациенте (вид, порода, кличка, дата рождения, индивидуальные особенности).

Организовывать расписание приемов: эффективно управлять записью пациентов, отслеживать занятость врачей, предотвращать накладки и оптимизировать загрузку клиники.

Документировать оказанные услуги: фиксировать все проведенные манипуляции, назначенные лекарства, диагностические процедуры и их стоимость, что обеспечивает прозрачность и точность финансового учета.

Вести электронную историю болезни: создавать и хранить подробные записи о каждом посещении животного, включая анамнез, результаты осмотров, диагнозы, назначения и ход лечения. Это значительно повышает качество медицинского обслуживания и позволяет врачам оперативно получать всю необходимую информацию о пациенте.

Формировать аналитические отчеты: генерировать различные отчеты, такие как отчеты о приеме пациентов, об оказанных услугах, финансовые отчеты. Эти отчеты предоставляют ценную информацию для анализа эффективности работы клиники и принятия обоснованных управленческих решений.

5. Разработка интуитивно понятного пользовательского интерфейса: Особое внимание было уделено созданию максимально удобного и интуитивно понятного пользовательского интерфейса. Проектирование UI/UX осуществлялось с учетом потребностей конечных пользователей – ветеринарных врачей, администраторов и другого персонала клиники. Навигация по системе проста и логична, что минимизирует время на обучение и позволяет персоналу быстро освоить все функции, повышая их производительность и снижая вероятность ошибок.

6. Подготовка исчерпывающей сопроводительной документации: В рамках проекта были разработаны два ключевых документа: руководство пользователя и руководство программиста. Руководство пользователя предоставляет подробные инструкции по работе с системой для конечных пользователей, позволяя им самостоятельно освоить ее функционал. Руководство программиста содержит детальное описание архитектуры, кода и логики работы системы, что критически важно для ее дальнейшей поддержки, модификации и развития другими разработчиками. Это обеспечивает преемственность и устойчивость проекта.

Практическая значимость разработанной программной системы трудно переоценить. Она позволяет ветеринарной клинике значительно оптимизировать свои операционные процессы, сократить объем ручной работы и связанные с ней ошибки, улучшить качество ведения записей и повысить прозрачность деятельности. В конечном итоге, это приводит к повышению эффективности работы персонала, улучшению обслуживания клиентов и созданию более комфортной и контролируемой рабочей среды. Система служит надежным инструментом для формирования достоверной базы данных, что является основой для принятия стратегических решений.

Практическая значимость разработанной программной системы заключается в ее способности существенно повысить эффективность работы ветеринарной клиники за счет автоматизации рутинных операций, минимизации ошибок, улучшения качества обслуживания клиентов и предоставления актуальной информации для принятия управленческих решений.

В перспективе, разработанная система может быть расширена дополнительным функционалом, таким как интеграция с лабораторным оборудованием, модули для управления складскими запасами медикаментов, а также возможность удаленного доступа для клиентов и врачей. Дальнейшее развитие системы может также включать улучшение аналитических отчетов для более глубокого анализа данных и поддержку мобильных платформ для повышения доступности.

Таким образом, выполненная курсовая работа подтверждает актуальность выбранной темы и демонстрирует практические навыки в области разработки программных систем, способных решать реальные задачи в условиях современной ветеринарной практики.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Книги и учебники:

1. Дейт К., Дарвин Х. Разработка и реализация информационных систем.
2. Joseph Albahari, Ben Albahari. C# 7.0 in a Nutshell.
3. Карло Ц., Браун П. Основы баз данных.

2. Статьи и публикации:

1. PubMed: https://pubmed.ncbi.nlm.nih.gov/
2. Google Scholar: https://scholar.google.com/
3. AVMA: https://www.avma.org/

3. Интернет-ресурсы:

1. C# Guide: https://docs.microsoft.com/en-us/dotnet/csharp/
2. .NET Documentation: https://docs.microsoft.com/en-us/dotnet/
3. Coursera: https://www.coursera.org/
4. Udemy: https://www.udemy.com/
5. edX: https://www.edx.org/
6. Stack Overflow: https://stackoverflow.com/
7. GitHub: https://github.com/

# ПРИЛОЖЕНИЕ А

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace ConsoleAppКУРСОВАЯ

{

class Animal

{

public int IdAnimal { get; set; }

public string Name { get; set; }

public string Species { get; set; }

public string Breed { get; set; }

public int Age { get; set; }

public Owner Owner { get; set; }

}

class Owner

{

public int IdOwner { get; set; }

public string FioOwner { get; set; }

public string Phone { get; set; }

public List<Animal> Animals { get; set; } = new List<Animal>();

}

class Veterinarian

{

public int IdVeter { get; set; }

public string FioVeter { get; set; }

public string Specialisation { get; set; }

public List<Appointment> HistoryAppointments { get; set; } = new List<Appointment>();

}

class Appointment

{

public int IdAppointment { get; set; }

public Animal Animal { get; set; }

public Veterinarian Veterinarian { get; set; }

public DateTime Data { get; set; }

public string Diagnosis { get; set; }

public string Medication { get; set; }

public decimal Cost { get; set; }

}

class Actions

{

private List<Animal> animals;

private List<Owner> owners;

private List<Veterinarian> veterinarians;

private List<Appointment> appointments;

private int nextAnimalId = 1;

private int nextOwnerId = 1;

private int nextVeterinarianId = 1;

private int nextAppointmentId = 1;

public Actions(List<Animal> animalsList, List<Owner> ownersList, List<Veterinarian> veterinariansList, List<Appointment> appointmentsList)

{

animals = animalsList;

owners = ownersList;

veterinarians = veterinariansList;

appointments = appointmentsList;

}

public void AddAnimal()

{

Console.Write("Введите вид питомца: ");

string species = Console.ReadLine();

Console.Write("Введите породу питомца: ");

string breed = Console.ReadLine();

Console.Write("Введите возраст питомца: ");

if (int.TryParse(Console.ReadLine(), out int age))

{

Console.Write("Введите кличку питомца: ");

string name = Console.ReadLine();

Console.Write("Введите ФИО владельца: ");

string ownerFio = Console.ReadLine();

Owner owner = owners.FirstOrDefault(o => o.FioOwner.ToLower() == ownerFio.ToLower());

if (owner != null)

{

Animal newAnimal = new Animal { IdAnimal = nextAnimalId++, Name = name, Species = species, Breed = breed, Age = age, Owner = owner };

animals.Add(newAnimal);

owner.Animals.Add(newAnimal);

Console.WriteLine($"Питомец '{name}' успешно добавлен.");

}

else

{

Console.WriteLine($"Владелец с ФИО '{ownerFio}' не найден. Пожалуйста, сначала добавьте владельца.");

}

}

else

{

Console.WriteLine("Некорректный ввод возраста.");

}

}

public void AddOwner()

{

Console.Write("Введите ФИО владельца: ");

string fio = Console.ReadLine();

Console.Write("Введите телефон владельца: ");

string phone = Console.ReadLine();

Owner newOwner = new Owner { IdOwner = nextOwnerId++, FioOwner = fio, Phone = phone };

owners.Add(newOwner);

Console.WriteLine($"Владелец '{fio}' успешно добавлен.");

}

public void AddVeterinarian()

{

Console.Write("Введите ФИО ветеринара: ");

string fio = Console.ReadLine();

Console.Write("Введите специализацию ветеринара: ");

string specialisation = Console.ReadLine();

Veterinarian newVeterinarian = new Veterinarian { IdVeter = nextVeterinarianId++, FioVeter = fio, Specialisation = specialisation };

veterinarians.Add(newVeterinarian);

Console.WriteLine($"Ветеринар '{fio}' успешно добавлен.");

}

public void AddAppointment()

{

Console.Write("Введите кличку питомца для записи на прием: ");

string animalName = Console.ReadLine();

Animal animal = animals.FirstOrDefault(a => a.Name.ToLower() == animalName.ToLower());

if (animal != null)

{

Console.Write("Введите ФИО ветеринара, к которому хотите записаться: ");

string vetFio = Console.ReadLine();

Veterinarian veterinarian = veterinarians.FirstOrDefault(v => v.FioVeter.ToLower() == vetFio.ToLower());

if (veterinarian != null)

{

Console.Write("Введите дату приема (гггг-мм-дд): ");

if (DateTime.TryParse(Console.ReadLine(), out DateTime date))

{

Console.Write("Введите диагноз: ");

string diagnosis = Console.ReadLine();

Console.Write("Введите назначенное лечение: ");

string medication = Console.ReadLine();

Console.Write("Введите стоимость приема: ");

if (decimal.TryParse(Console.ReadLine(), out decimal cost))

{

Appointment newAppointment = new Appointment

{

IdAppointment = nextAppointmentId++,

Animal = animal,

Veterinarian = veterinarian,

Data = date,

Diagnosis = diagnosis,

Medication = medication,

Cost = cost

};

appointments.Add(newAppointment);

veterinarian.HistoryAppointments.Add(newAppointment);

Console.WriteLine($"Прием для '{animal.Name}' у ветеринара '{veterinarian.FioVeter}' на {date.ToShortDateString()} успешно назначен.");

}

else

{

Console.WriteLine("Некорректный ввод стоимости.");

}

}

else

{

Console.WriteLine("Некорректный формат даты.");

}

}

else

{

Console.WriteLine($"Ветеринар с ФИО '{vetFio}' не найден.");

}

}

else

{

Console.WriteLine($"Питомец с кличкой '{animalName}' не найден.");

}

}

public void PrintAnimalDetails()

{

if (animals.Any())

{

Console.WriteLine("--- Список всех животных ---");

foreach (var animal in animals)

{

Console.WriteLine($"ID: {animal.IdAnimal}, Кличка: {animal.Name}, Вид: {animal.Species}, Порода: {animal.Breed}, Возраст: {animal.Age}, Владелец: {animal.Owner.FioOwner} (ID: {animal.Owner.IdOwner})");

}

}

else

{

Console.WriteLine("Список животных пуст.");

}

}

public void PrintOwnerDetails()

{

if (owners.Any())

{

Console.WriteLine("--- Список всех владельцев ---");

foreach (var owner in owners)

{

Console.WriteLine($"ID: {owner.IdOwner}, ФИО: {owner.FioOwner}, Телефон: {owner.Phone}, Питомцы: {string.Join(", ", owner.Animals.Select(a => a.Name))} (ID: {string.Join(", ", owner.Animals.Select(a => a.IdAnimal))})");

}

}

else

{

Console.WriteLine("Список владельцев пуст.");

}

}

public void PrintVeterinarianDetails()

{

if (veterinarians.Any())

{

Console.WriteLine("--- Список всех ветеринаров ---");

foreach (var veterinarian in veterinarians)

{

Console.WriteLine($"ID: {veterinarian.IdVeter}, ФИО: {veterinarian.FioVeter}, Специализация: {veterinarian.Specialisation}");

}

}

else

{

Console.WriteLine("Список ветеринаров пуст.");

}

}

public void PrintAppointmentDetails()

{

if (appointments.Any())

{

Console.WriteLine("--- Список всех приёмов ---");

foreach (var appointment in appointments)

{

Console.WriteLine($"ID: {appointment.IdAppointment}, Питомец: {appointment.Animal.Name} (ID: {appointment.Animal.IdAnimal}), Ветеринар: {appointment.Veterinarian.FioVeter} (ID: {appointment.Veterinarian.IdVeter}), Дата: {appointment.Data.ToShortDateString()}, Диагноз: {appointment.Diagnosis}, Лечение: {appointment.Medication}, Стоимость: {appointment.Cost}");

}

}

else

{

Console.WriteLine("Список приёмов пуст.");

}

}

public void GetAnimalsVisitedLastMonth()

{

DateTime lastMonthStart = DateTime.Now.AddMonths(-1);

var lastMonthVisits = appointments.Where(a => a.Data >= lastMonthStart);

if (lastMonthVisits.Any())

{

Console.WriteLine("--- Животные, посетившие клинику за последний месяц ---");

foreach (var visit in lastMonthVisits.Select(v => v.Animal).Distinct())

{

Console.WriteLine($"- {visit.Name} (Владелец: {visit.Owner.FioOwner})");

}

}

else

{

Console.WriteLine("За последний месяц клинику никто не посещал.");

}

}

public void GetVeterinariansByPatient()

{

Console.Write("Введите кличку питомца: ");

string animalName = Console.ReadLine();

Animal animal = animals.FirstOrDefault(a => a.Name.ToLower() == animalName.ToLower());

if (animal != null)

{

var veterinariansTreated = appointments

.Where(a => a.Animal == animal)

.Select(a => a.Veterinarian)

.Distinct();

if (veterinariansTreated.Any())

{

Console.WriteLine($"--- Ветеринары, лечившие питомца '{animal.Name}' ---");

foreach (var vet in veterinariansTreated)

{

Console.WriteLine($"- {vet.FioVeter} (Специализация: {vet.Specialisation})");

}

}

else

{

Console.WriteLine($"Питомец '{animal.Name}' не посещал ветеринаров.");

}

}

else

{

Console.WriteLine($"Питомец с кличкой '{animalName}' не найден.");

}

}

public void GetOwnersWithMultiplePets()

{

var ownersWithMultiple = owners.Where(o => o.Animals.Count > 1);

if (ownersWithMultiple.Any())

{

Console.WriteLine("--- Владельцы с более чем одним питомцем ---");

foreach (var owner in ownersWithMultiple)

{

Console.WriteLine($"- {owner.FioOwner} (Количество питомцев: {owner.Animals.Count}, Питомцы: {string.Join(", ", owner.Animals.Select(a => a.Name))})");

}

}

else

{

Console.WriteLine("Нет владельцев с более чем одним питомцем.");

}

}

public void GetTotalCostByOwnerAndPet()

{

Console.Write("Введите ФИО владельца: ");

string ownerFio = Console.ReadLine();

Owner owner = owners.FirstOrDefault(o => o.FioOwner.ToLower() == ownerFio.ToLower());

if (owner != null)

{

Console.Write("Введите кличку питомца: ");

string animalName = Console.ReadLine();

Animal animal = owner.Animals.FirstOrDefault(a => a.Name.ToLower() == animalName.ToLower());

if (animal != null)

{

decimal totalCost = appointments

.Where(a => a.Animal == animal)

.Sum(a => a.Cost);

Console.WriteLine($"Общая сумма, выплаченная владельцем '{owner.FioOwner}' за лечение питомца '{animal.Name}': {totalCost} руб.");

}

else

{

Console.WriteLine($"Питомец с кличкой '{animalName}' не найден у владельца '{owner.FioOwner}'.");

}

}

else

{

Console.WriteLine($"Владелец с ФИО '{ownerFio}' не найден.");

}

}

public void GetMostFrequentDiagnoses()

{

var frequentDiagnoses = appointments

.GroupBy(a => a.Diagnosis)

.OrderByDescending(g => g.Count())

.Take(5); // Выведем топ-5 самых частых диагнозов

if (frequentDiagnoses.Any())

{

Console.WriteLine("--- Самые частые диагнозы ---");

foreach (var diagnosisGroup in frequentDiagnoses)

{

Console.WriteLine($"- Диагноз: '{diagnosisGroup.Key}' (Количество: {diagnosisGroup.Count()})");

}

}

else

{

Console.WriteLine("Нет данных о диагнозах.");

}

}

public void GetTotalRevenue()

{

decimal totalRevenue = appointments.Sum(a => a.Cost);

Console.WriteLine($"Общая выручка клиники: {totalRevenue} руб.");

}

public void HireVeterinarian()

{

AddVeterinarian();

}

public void FireVeterinarian()

{

Console.Write("Введите ФИО ветеринара для увольнения: ");

string fio = Console.ReadLine();

Veterinarian veterinarianToRemove = veterinarians.FirstOrDefault(v => v.FioVeter.ToLower() == fio.ToLower());

if (veterinarianToRemove != null)

{

veterinarians.Remove(veterinarianToRemove);

Console.WriteLine($"Ветеринар '{fio}' успешно уволен.");

}

else

{

Console.WriteLine($"Ветеринар с ФИО '{fio}' не найден.");

}

}

public void SaveDataToFile(string filePath)

{

try

{

using (StreamWriter writer = new StreamWriter(filePath))

{

writer.WriteLine("--- Животные ---");

foreach (var animal in animals)

{

writer.WriteLine($"ID: {animal.IdAnimal}, Кличка: {animal.Name}, Вид: {animal.Species}, Порода: {animal.Breed}, Возраст: {animal.Age}, Владелец: {animal.Owner.FioOwner} (ID: {animal.Owner.IdOwner})");

}

writer.WriteLine();

writer.WriteLine("--- Владельцы ---");

foreach (var owner in owners)

{

writer.WriteLine($"ID: {owner.IdOwner}, ФИО: {owner.FioOwner}, Телефон: {owner.Phone}, Питомцы: {string.Join(", ", owner.Animals.Select(a => a.Name))} (ID: {string.Join(", ", owner.Animals.Select(a => a.IdAnimal))})");

}

writer.WriteLine();

writer.WriteLine("--- Ветеринары ---");

foreach (var veterinarian in veterinarians)

{

writer.WriteLine($"ID: {veterinarian.IdVeter}, ФИО: {veterinarian.FioVeter}, Специализация: {veterinarian.Specialisation}");

if (veterinarian.HistoryAppointments.Any())

{

writer.WriteLine(" История приемов:");

foreach (var appointment in veterinarian.HistoryAppointments)

{

writer.WriteLine($" ID: {appointment.IdAppointment}, Питомец: {appointment.Animal.Name} (ID: {appointment.Animal.IdAnimal}), Дата: {appointment.Data.ToShortDateString()}, Диагноз: {appointment.Diagnosis}, Лечение: {appointment.Medication}, Стоимость: {appointment.Cost}");

}

}

else

{

writer.WriteLine(" История приемов отсутствует.");

}

}

writer.WriteLine();

writer.WriteLine("--- Приемы ---");

foreach (var appointment in appointments)

{

writer.WriteLine($"ID: {appointment.IdAppointment}, Питомец: {appointment.Animal.Name} (ID: {appointment.Animal.IdAnimal}), Ветеринар: {appointment.Veterinarian.FioVeter} (ID: {appointment.Veterinarian.IdVeter}), Дата: {appointment.Data.ToShortDateString()}, Диагноз: {appointment.Diagnosis}, Лечение: {appointment.Medication}, Стоимость: {appointment.Cost}");

}

Console.WriteLine($"Данные успешно записаны в файл '{filePath}'.");

}

}

catch (Exception ex)

{

Console.WriteLine($"Произошла ошибка при записи в файл: {ex.Message}");

}

}

}

public static class StringExtensions

{

public static string PadCenter(this string str, int totalLength)

{

int padding = totalLength - str.Length;

if (padding <= 0)

return str;

int leftPadding = padding / 2;

int rightPadding = padding - leftPadding;

return str.PadLeft(leftPadding).PadRight(totalLength);

}

}

class Program

{

static List<Animal> animals = new List<Animal>();

static List<Owner> owners = new List<Owner>();

static List<Veterinarian> veterinarians = new List<Veterinarian>();

static List<Appointment> appointments = new List<Appointment>();

static void Main(string[] args)

{

Console.Title = "Ветеринарная Клиника \"Здоровье Питомец\"";

Console.OutputEncoding = System.Text.Encoding.UTF8;

Console.CursorVisible = false; // Скроем курсор

Actions clinicActions = new Actions(animals, owners, veterinarians, appointments);

int menuWidth = 60;

int consoleWidth = Console.WindowWidth;

bool flag = true;

while (flag)

{

Console.Clear(); // Очистка экрана в начале каждой итерации

PrintHeaderFancy(menuWidth, consoleWidth);

PrintCenteredMenuFancy(menuWidth, consoleWidth);

Console.SetCursorPosition((consoleWidth - "Ваш выбор: ".Length) / 2, Console.CursorTop);

Console.ForegroundColor = ConsoleColor.Yellow;

Console.Write("Ваш выбор: ");

Console.ResetColor();

string s = Console.ReadLine();

Console.WriteLine();

switch (s)

{

case "1":

RunActionFancy("Добавление животного", clinicActions.AddAnimal, menuWidth, consoleWidth);

break;

case "2":

RunActionFancy("Добавление владельца", clinicActions.AddOwner, menuWidth, consoleWidth);

break;

case "3":

RunActionFancy("Добавление ветеринара", clinicActions.AddVeterinarian, menuWidth, consoleWidth);

break;

case "4":

RunActionFancy("Назначение приема", clinicActions.AddAppointment, menuWidth, consoleWidth);

break;

case "5":

RunActionFancy("Список всех животных", clinicActions.PrintAnimalDetails, menuWidth, consoleWidth);

break;

case "6":

RunActionFancy("Список всех владельцев", clinicActions.PrintOwnerDetails, menuWidth, consoleWidth);

break;

case "7":

RunActionFancy("Список всех ветеринаров", clinicActions.PrintVeterinarianDetails, menuWidth, consoleWidth);

break;

case "8":

RunActionFancy("Список всех приёмов", clinicActions.PrintAppointmentDetails, menuWidth, consoleWidth);

break;

case "9":

RunActionFancy("Животные, посетившие клинику за последний месяц", clinicActions.GetAnimalsVisitedLastMonth, menuWidth, consoleWidth);

break;

case "10":

RunActionFancy("Ветеринары, лечившие указанного пациента", clinicActions.GetVeterinariansByPatient, menuWidth, consoleWidth);

break;

case "11":

RunActionFancy("Владельцы с более чем одним питомцем", clinicActions.GetOwnersWithMultiplePets, menuWidth, consoleWidth);

break;

case "12":

RunActionFancy("Сумма лечения питомца владельцем", clinicActions.GetTotalCostByOwnerAndPet, menuWidth, consoleWidth);

break;

case "13":

RunActionFancy("Самые частые диагнозы", clinicActions.GetMostFrequentDiagnoses, menuWidth, consoleWidth);

break;

case "14":

RunActionFancy("Общая выручка клиники", clinicActions.GetTotalRevenue, menuWidth, consoleWidth);

break;

case "15":

RunActionFancy("Прием ветеринара на работу", clinicActions.HireVeterinarian, menuWidth, consoleWidth);

break;

case "16":

RunActionFancy("Увольнение ветеринара", clinicActions.FireVeterinarian, menuWidth, consoleWidth);

break;

case "17":

Console.Clear();

Console.ForegroundColor = ConsoleColor.Yellow;

Console.WriteLine($"\n--- Запись данных в файл ---");

Console.ResetColor();

Console.Write("Введите путь к файлу для сохранения (например, data.txt): ");

string filePath = Console.ReadLine();

clinicActions.SaveDataToFile(filePath);

Console.ForegroundColor = ConsoleColor.Gray;

Console.WriteLine("\nНажмите любую клавишу для возврата в меню...");

Console.ResetColor();

Console.ReadKey(true);

break;

case "0":

Console.Clear();

PrintGoodbyeFancy(menuWidth, consoleWidth);

flag = false;

break;

default:

Console.Clear();

PrintErrorFancy("Некорректный ввод. Пожалуйста, повторите.", menuWidth, consoleWidth);

Console.ReadKey(true);

break;

}

}

Console.CursorVisible = true; // Вернем курсор перед выходом

}

static void PrintHeaderFancy(int menuWidth, int consoleWidth)

{

Console.ForegroundColor = ConsoleColor.Green;

string headerText = "Ветеринарная Клиника \"Здоровье Питомец\"";

string topBorder = "╔" + new string('═', menuWidth - 2) + "╗";

string middleLine = "╠" + new string('═', menuWidth - 2) + "╣";

string bottomBorder = "╚" + new string('═', menuWidth - 2) + "╝";

string paddedHeader = "║" + headerText.PadCenter(menuWidth - 2) + "║";

int horizontalPadding = (consoleWidth - menuWidth) / 2;

int topOffset = 1; // Смещение от верхнего края

Console.SetCursorPosition(horizontalPadding, topOffset);

Console.WriteLine(topBorder);

Console.SetCursorPosition(horizontalPadding, topOffset + 1);

Console.WriteLine(paddedHeader);

Console.SetCursorPosition(horizontalPadding, topOffset + 2);

Console.WriteLine(middleLine);

Console.ResetColor();

}

static void PrintCenteredMenuFancy(int menuWidth, int consoleWidth)

{

Console.ForegroundColor = ConsoleColor.Cyan;

string title = "Выберите действие:";

string[] menuItems = new string[]

{

"1. Добавить животное",

"2. Добавить владельца",

"3. Добавить ветеринара",

"4. Назначить прием",

"",

"5. Список всех животных",

"6. Список всех владельцев",

"7. Список всех ветеринаров",

"8. Список всех приёмов",

"",

"9. Животные, посетившие за последний месяц",

"10. Ветеринары, лечившие пациента",

"11. Владельцы с более чем одним питомцем",

"12. Сумма лечения питомца владельцем",

"13. Самые частые диагнозы",

"14. Общая выручка клиники",

"",

"15. Принять ветеринара на работу",

"16. Уволить ветеринара",

"",

"17. Сохранить данные в файл", // Новый пункт меню

"0. Выход"

};

int horizontalPadding = (consoleWidth - menuWidth) / 2;

string verticalLine = "║";

int startLine = 4; // Начальная строка для меню

int consoleHeight = Console.WindowHeight;

Console.SetCursorPosition(horizontalPadding, startLine);

Console.WriteLine(verticalLine + title.PadCenter(menuWidth - 2) + verticalLine);

Console.SetCursorPosition(horizontalPadding, startLine + 1);

Console.WriteLine("╠" + new string('─', menuWidth - 2) + "╣");

int currentLine = startLine + 2;

foreach (string item in menuItems)

{

if (currentLine < consoleHeight - 2) // Проверка, не выходим ли за нижнюю границу

{

Console.SetCursorPosition(horizontalPadding, currentLine++);

Console.WriteLine(verticalLine + item.PadLeft((menuWidth - 2) / 2 + item.Length / 2).PadRight(menuWidth - 2) + verticalLine);

}

}

if (currentLine < consoleHeight - 1) // Проверка перед выводом нижней границы

{

Console.SetCursorPosition(horizontalPadding, currentLine);

Console.WriteLine("╚" + new string('═', menuWidth - 2) + "╝");

Console.ResetColor();

}

else

{

Console.WriteLine("\nМеню слишком длинное для отображения полностью.");

Console.ResetColor();

}

}

static void RunActionFancy(string title, Action action, int menuWidth, int consoleWidth)

{

Console.Clear(); // Очистка перед выполнением действия

Console.ForegroundColor = ConsoleColor.Yellow;

string topBorder = "╔" + new string('═', menuWidth - 2) + "╗";

string bottomBorder = "╚" + new string('═', menuWidth - 2) + "╝";

string paddedTitle = "║" + $"--- {title} ---".PadCenter(menuWidth - 2) + "║";

int horizontalPadding = (consoleWidth - menuWidth) / 2;

int leftOffset = horizontalPadding + 2; // Отступ слева для содержимого действия

int topOffset = 1;

Console.SetCursorPosition(horizontalPadding, topOffset);

Console.WriteLine(topBorder);

Console.SetCursorPosition(horizontalPadding, topOffset + 1);

Console.WriteLine(paddedTitle);

Console.SetCursorPosition(horizontalPadding, topOffset + 2);

Console.WriteLine(bottomBorder);

Console.ResetColor();

Console.SetCursorPosition(leftOffset, topOffset + 4);

action?.Invoke();

Console.ForegroundColor = ConsoleColor.Gray;

Console.SetCursorPosition(leftOffset, Console.CursorTop + 2);

Console.WriteLine("Нажмите любую клавишу для возврата в меню...");

Console.ResetColor();

Console.ReadKey(true);

}

static void PrintErrorFancy(string message, int menuWidth, int consoleWidth)

{

Console.Clear(); // Очистка перед выводом ошибки

Console.ForegroundColor = ConsoleColor.Red;

string topBorder = "╔" + new string('═', menuWidth - 2) + "╗";

string bottomBorder = "╚" + new string('═', menuWidth - 2) + "╝";

string paddedMessage = "║" + $"Ошибка: {message}".PadCenter(menuWidth - 2) + "║";

int horizontalPadding = (consoleWidth - menuWidth) / 2;

int topOffset = 1;

Console.SetCursorPosition(horizontalPadding, topOffset);

Console.WriteLine(topBorder);

Console.SetCursorPosition(horizontalPadding, topOffset + 1);

Console.WriteLine(paddedMessage);

Console.SetCursorPosition(horizontalPadding, topOffset + 2);

Console.WriteLine(bottomBorder);

Console.ResetColor();

Console.SetCursorPosition(horizontalPadding + 2, topOffset + 4);

}

static void PrintGoodbyeFancy(int menuWidth, int consoleWidth)

{

Console.Clear(); // Очистка перед прощанием

Console.ForegroundColor = ConsoleColor.Green;

string goodbyeText = "Спасибо за работу!";

string subText = "До свидания!";

string topBorder = "╔" + new string('═', menuWidth - 2) + "╗";

string bottomBorder = "╚" + new string('═', menuWidth - 2) + "╝";

string paddedGoodbye = "║" + goodbyeText.PadCenter(menuWidth - 2) + "║";

string paddedSub = "║" + subText.PadCenter(menuWidth - 2) + "║";

int horizontalPadding = (consoleWidth - menuWidth) / 2;

int topOffset = 1;

Console.SetCursorPosition(horizontalPadding, topOffset);

Console.WriteLine(topBorder);

Console.SetCursorPosition(horizontalPadding, topOffset + 1);

Console.WriteLine(paddedGoodbye);

Console.SetCursorPosition(horizontalPadding, topOffset + 2);

Console.WriteLine(paddedSub);

Console.SetCursorPosition(horizontalPadding, topOffset + 3);

Console.WriteLine(bottomBorder);

Console.ResetColor();

}

}

}