Содержание

Введение

С самого начала существования человеческого вида животные играли исключительную роль в жизни человека: обеспечивали его пищей и другими материалами, средствами передвижения, а также были неотъемлемой частью культуры и религии. Практика содержания домашних животных известна человечеству на протяжении тысячелетий. Животные играют важную роль в жизни человека и сегодня.

Так, по данным ВЦИОМ[1], в 2019 году у 68% россиян в семье есть домашние животные, в основном — кошки и собаки. Россия на 2019 год являлась страной с наибольшим количеством семей, в которых есть домашние животные.

Животные нуждаются в медецинской помощи. Необходима защита животных от инфекционных[2], протозойных[3], гельминтозных[4], арахноэнтомозных[5] и незаразных болезней. Особую важность имеет борьба с болезнями, общими человеку и животным, с вирусными[6] болезнями животных. Насущной задачей является ликвидация гельминтозов. Оздоровление внешней среды от возбудителей и переносчиков болезней и разработка более современных методов и способов ветеринарно-санитарной оценки продуктов животноводства имеют важное значение для животноводства и гигиены. Ветеринария, ветеринарная медицина – это система наук, изучающих болезни животных, вопросы повышения их продуктивности, методы защиты людей от зоонозов.

Учреждениями, оказывающими ветеринарную помощь,являются ветеринарные клиники. Крупные ветеринарные клиники ежедневно оказывают услуги большому количеству клиентов, имеют большое количество персонала и оборудования, оказывают широкий спектр услуг. Учет деятельности крупной клиники без использования автоматизированных систем является затруднительным. Автоматизация учета деятельности будет удобна не только крупным, но и небольшим и средним клиникам.

В связи с тем, что число домашних животных постоянно растет, увеличивает-

[1]Всероссийский центр изучения общественного мнения

[2]Инфекционные заболевания — группа заболеваний, вызываемых проникновением в организм патогенных (болезнетворных) микроорганизмов, вирусов и прионов.

[3]Протозооз, или протозойные заболевания – это инфекции, вызываемые паразитическими простейшими.

[4]Гельминтозы – болезни, вызываемые гельминтами — паразитическими червями.

[5]Арахноэнтомозы — паразитарные болезни животных, культурных растений и человека, вызванные членистоногими.

[6]Вирусное заболевание возникает, когда организм заражается патогенными вирусами, а частицы инфекционного вируса прикрепляются к чувствительным клеткам и попадают в них.

ся и число пациэнтов ветеринарных клиник. Так, по данным Интерфакс, с 2015 по 2018 год число домашних животных в РФ выросло на 14% [1]. Современные ветеринарные клиники накапливают огромные объемы данных. Эффективность использования этой информации работниками клиники влияет на качество оказания ветеринарной медицинской помощи. Таким образом, создание информационной системы ветеринарной клиники является актуальной задачей.

Целью данной работы является создание информационной системы ветеринарной клиники. Под информационной системой понимается совокупность программных средств, предназначенная для сбора, обработки, хранения и выдачи информации и принятия управленческих решений.

Для достижения поставленной цели необходимо решить следующие задачи.

1. Провести анализ предметной области.
2. Определить необходимый функционал и определить постановку задачи.
3. Провести анализ существующих на рынке решений.
4. Спроектировать информационную систему.
5. Выбрать технологический стек.
6. Разработать программное обеспечение.

1 Аналитический раздел

В данном разделе проводится анализ предметной области, формализация требований к программе, а также рассматриваются различные способы хранения данных в приложениях.

1.1 Анализ предметной области

Целями деятельности учреждений ветеринарии являются:

- обеспечение эпизоотического и ветеринарно-санитарного благополучия на обслуживаемой территории;
- предупреждение болезней животных, их лечение, обеспечение полноценности и безопасности выпускаемой продукции животного происхождения в ветеринарно-санитарном отношении, защита населения от болезней, общих для животных и человека, и пищевых отравлений.

Основными задачами ветеринарных клиник являются:

- предупреждение и ликвидация карантинных и особо опасных болезней животных и осуществление региональных планов ветеринарного обслуживания животноводства;
- организация диагностической работы и проведение лабораторных исследований всеми современными методами.

Учреждения ветеринарии осуществляют следующие виды деятельности:

- выявление и установление причин и условий возникновения и распространения заразных и массовых незаразных болезней животных;
- организация и проведение противоэпизоотических и ветеринарно-санитарных мероприятий, направленных на обеспечение эпизоотического и ветеринарно-санитарного благополучия на обслуживаемой территории;
- организация и проведение ветеринарных профилактических и лечебных мероприятий;

- осуществление специальных мероприятий по защите животных от поражающего воздействия экстремальных факторов, природных и техногенных катастроф;

- проведение с целью диагностики заболеваний животных, в том числе птиц, пушных зверей, пчел и рыб, бактериологических, вирусологических, токсикологических, копрологических и других лабораторных исследований соответствующих материалов; сообщение результатов исследования и выдача соответствующих заключений;

- установление лабораторного диагноза болезней животных; выявление животных, больных заразными болезнями или болезнями, связанными с нарушениями обмена веществ и другими отклонениями в жизнедеятельности организма;

- осуществление организационно-методической и консультативной помощи.

В штате ветеринарной клиники с электронной информационной системой должно быть не менее пяти человек:

- администратор клиники (главврач);

- два ветеринарных врача;

- уборщик(ца);

- системный администратор.

1.2 Постановка задачи

В рамках курсовой работы необходимо спроектировать и разработать информационную систему для автоматизации деятельности ветеринарных клиник. Система должна поддерживать следующий функционал:

- возможность добавления новых пользователей и аутентификация;

- электронная медицинская документация, необходимая для работы специалистов клиники (форма осмотра врача, электронная медицинская карта животного, идентификация животного – сведения о чипировании);

- электронная юридическая документация (сведения о договорах с книентами, паспортные данные клиентов и работников);
- электронная работа с персоналом и расписанием;
- просмотр информации о всех животных и клиентах (хозяевах животных), состоящих на учете в ветеринарной клинике;
- просмотр медицинских отчетов о проведенных осмотрах;
- редактирование, удаление, добавление сведений;
- выделение разных ролей среди пользователей и наделение их определенными правами в разрабатываемой системе.

Система должна представлять собой приложение по модели клиент-сервер, состоящее из серверной части, отвечающую за хранение данных и обрабтку запросов клиентов, а также клиентского desktop-приложения, отвечающего за взаимодействие пользователей с данными. Десктопное приложение – это клиентское программное обеспечение, устанавливаемое на рабочую станцию пользователя.

1.3 БД и СУБД

База данных (БД) – это совокупность данных, организованных по определённым правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ. Эти данные относятся к определённой предметной области и организованы таким образом, что могут быть использованы для решения многих задач многими пользователями.

Система управления базами данных (СУБД) – это совокупность программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия её с прикладными программами [2].

Программы, с помощью которых пользователи работают с базой данных, называются приложениями. В общем случае с одной базой данных могут работать множество различных приложений.

При рассмотрении приложении, работающих с одной базой данных, предполагается, чго они могут работать параллельно и независимо друг от друга, и именно

СУБД призвана обеспечить работу множества приложений с единой базой данных таким образом, чтобы каждое из них выполнялось корректно, по учитывало все изменения в базе данных, вносимые другими приложениями.

1.4 Модели данных

Модель данных — это некоторая абстракция, которая, будучи приложима к конкретным данным, позволяет пользователям и разработчикам трактовать их уже как информацию, то есть сведения, содержащие не только данные, но и взаимосвязь между ними.

На рисунке 1 представлена классификация моделей данных.

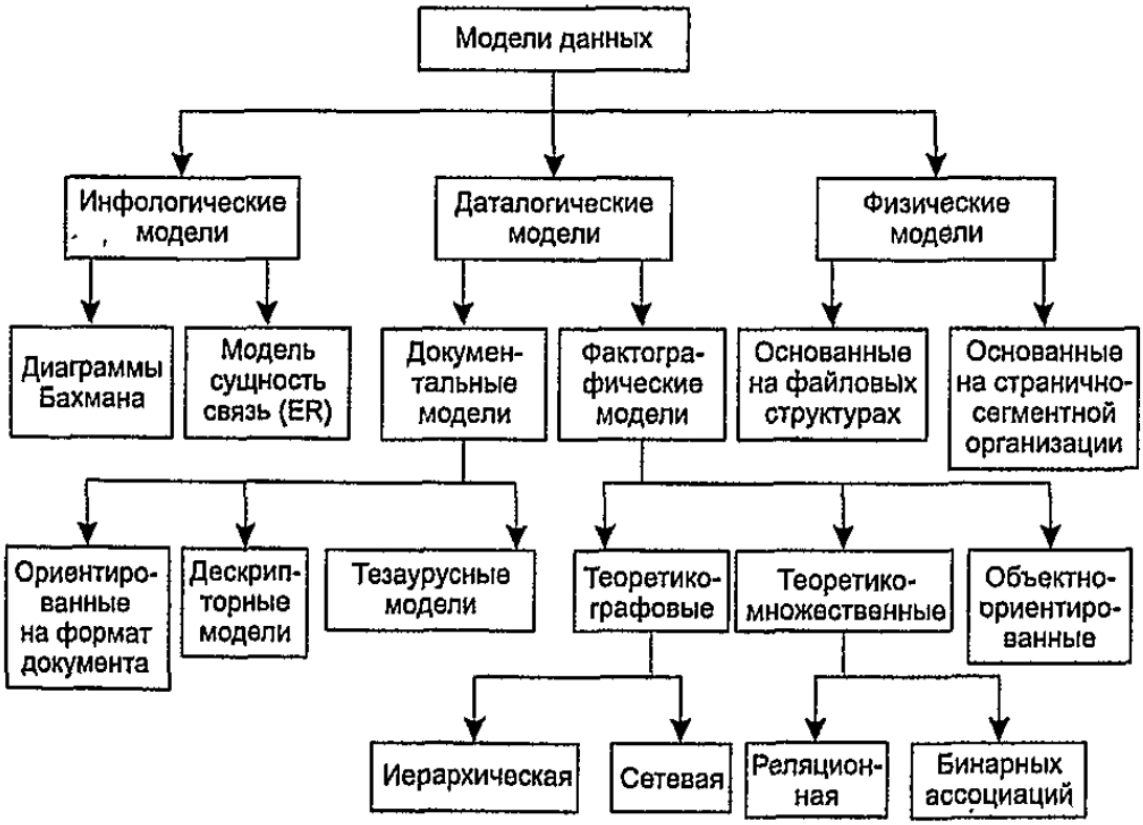

Рис. 1: Классификация моделей данных [2].

1.5 Типы хранилищ

Выбор наиболее подходящего хранилища данных является ключевым для проекта. Хранилища данных делятся на разные группы в зависимости от методов

структурирования данных и поддерживаемых операций. По используемой модели данных СУБД делятся на реляционные и нереляционные. По способу доступа к БД, СУБД делятся на файл-серверные, клиент-серверные и встраиваемые. Рассмотрим некоторые наиболее часто встречающиеся их типы.

1.5.1 Реляционные базы данных

Реляционные БД основаны на реляционной модели данных (РМД). РМД построена на таких разделах математики, как теория множеств и логика первого порядка. Реляционные БД являются наиболее широко распространенными, так как РМБ имеет под собой развитый математический аппарат.

Теоретической основой этой модели является теория отношений. Основной структурой данных в модели является отношение, именно поэтому модель получила название реляционной. Отношение имеет простую графическую интерпретацию, оно может быть представлено в виде таблицы.

Главным преимуществом РМД является простота представления и формирования баз данных. Управление данными в реляционных БД осуществляется с помощью декларативного языка запросов SQL (Structured Query Language), который основан на реляционной алгебре.

С другой стороны, РМД имеет следующие недостатки:

- отсутствие механизма отображения связи типа «многие ко многим»;
- отсутствие возможности указания типов связей;
- отсутствие специальных механизмов навигации;
- потеря соответствия (impedance mismatch) – структуры данных, хранимые в памяти, и РМД не соответствуют друг другу, из-за чего, в частности, цена создания проекта с использованием реляционной модели данных увеличивается.

Перечисленные недостатки, с одной стороны, ведет к упрощению модели, что и сделало ее такой популярной, но с другой стороны, приводит к снижению скорости доступа к данным, увеличивает объем памяти, занимаемой БД. Кроме того, не каждую предметную область возможно представить в виде отношений – основного понятия РМД.

Примерами реляционных БД являются PostgreSQL, Oracle, Microsoft SQL Server.

1.5.2 Нереляционные базы данных

К нереляционным системам хранения данных относятся четыре типа систем: хранилище пар «ключ – значение», база данных документов (документоориентированная БД), семейство столбцов (БД столбцов) и база данных графов.

Хранилище пар «ключ-значение» представляет собой хэш-таблицу, то есть доступ к БД осуществляется через ключ. Обладает всеми преимуществами и недостатками этой структуры данных. Такие хранилища легко масштабируемы и имеют высокую производительность, однако не позволяют делать запросы сразу к нескольким хранилищам и не поддерживают запросы по значению. Хранилища пар «ключ – значение» чаще всего используются для хранения изображений и в качестве кэшей. Примеры хранилищ пар «ключ – значение»: Berkley DB, Redis.

Базы данных документов предназначены для хранения иерархических структур данных. В основе таких БД лежат хранилища, имеющие структуру дерева. Основной концепцией в таких БД является документ. Документы – это «самоописываемые иерархические структуры данных» [3] (XML, JSON, BSON и т. д.). Примеры документоориентированных БД: MongoDB, CouchDB.

БД столбцов позволяют группировать значения в семейства столбцов, каждое из которых является ассоциативным массивом данных [3]. Примеры: Apache HBase, Apache Cassandra.

Графовые БД предоставляют возможность хранить сущности и отношения между ними. Такие БД могут не справиться с большим объемом данным, если, например, необходимо изменить свойства всех сущностей, то есть выполнить операцию, затрагивающую весь граф. Примеры графовых БД: Neo4j, OrientDB.

В целом, можно отметить следующие преимущества нереляционных БД (NoSQL).

- Возможность эффективно обрабатывать большие объемы данных на кластерах. Реляционные БД не предназначены для этого.
- Более удобный способ обмена данными, что повышает производительность разработки приложений.
- БД NoSQL работают без схемы, «позволяя свободно добавлять поля в базу данных без предварительного изменения структуры» [3]. БД NoSQL не имеют ограничений на тип хранимых данных.
- NoSQL-базы лучше масштабируются (по сравнению с реляционными БД).

- Разработка NoSQL-базы требует меньше времени, чем разработка реляционной БД.

1.5.3 Файл-серверные, клиент-серверные и встраиваемые БД

В файл-серверных СУБД файлы данных расположены на файл-сервере, а СУБД расположена на каждой клиентской рабочей станции. Сейчас эта технология считается устаревшей и не рекомендуется к использованию. Примеры: Microsoft Access, Paradox.

В клиент-серверной СУБД и файлы данных и сама СУБД располагаются на сервере. Все запросы от клиентов выполняются централизованно. Такие СУБД предъявляют высокие требования к серверу. Примеры: PostgreSQL, Oracle, Microsoft SQL Server, MongoDB.

Встраиваемая БД является составной частью программного продукта. Она локально хранит данные конкретного приложения и не рассчитана на совместное использование. Примеры: SQLite, BerkleyDB, RocksDB, Firebird Embedded, Microsoft SQL Server Compact.

1.6 Выбор типа хранилища

Реляционная модель данных, с одной стороны, обладает простотой и наглядностью для пользователей-непрограммистов, а с другой – серьезное теоретическое обоснование. Это определило большую популярность модели. Кроме того, развитие формального аппарата представления и манипулирования данными в рамках реляционной модели сделали се наиболее перспективной для использования в системах представления знаний, что обеспечивает качественно иной подход к обработке данных в больших информационных системах.

По этим причинам в данной работе будет использоваться реляционная модель данных. С точки зрения доступа к БД, будет использоваться клиент-серверная СУБД, так как файл-серверные СУБД считаются устаревшими, а встраиваемые БД, хранящие данные локально, не подходят для централизованного хранения данных информационной системы ветеринарной клиники, которая будет иметь большое количество пользователей.

В качестве СУБД в данной работе будет использоваться PostgreSQL. PostgreSQL – это мощная объектно-реляционная система баз данных с открытым исходным

кодом, активная разработка которой насчитывает более 30 лет, что принесло ей прочную репутацию благодаря надежности, функциональной устойчивости и производительности. Кроме того, PostgreSQL имеет подробную официальную документацию и большое сообщество разработчиков.

1.7 Диаграмма вариантов использования

Исходя из анализа предментой области, необходимо выделить группы пользователей разабатываемой системы.

Диаграмма вариантов использования (диаграмма прецедентов или use-case диаграмма) описывает функциональное назначение системы. На диаграмме вариантов использования проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью вариантов использования.

Диаграмма вариантов использования представлена на рисунке 2.

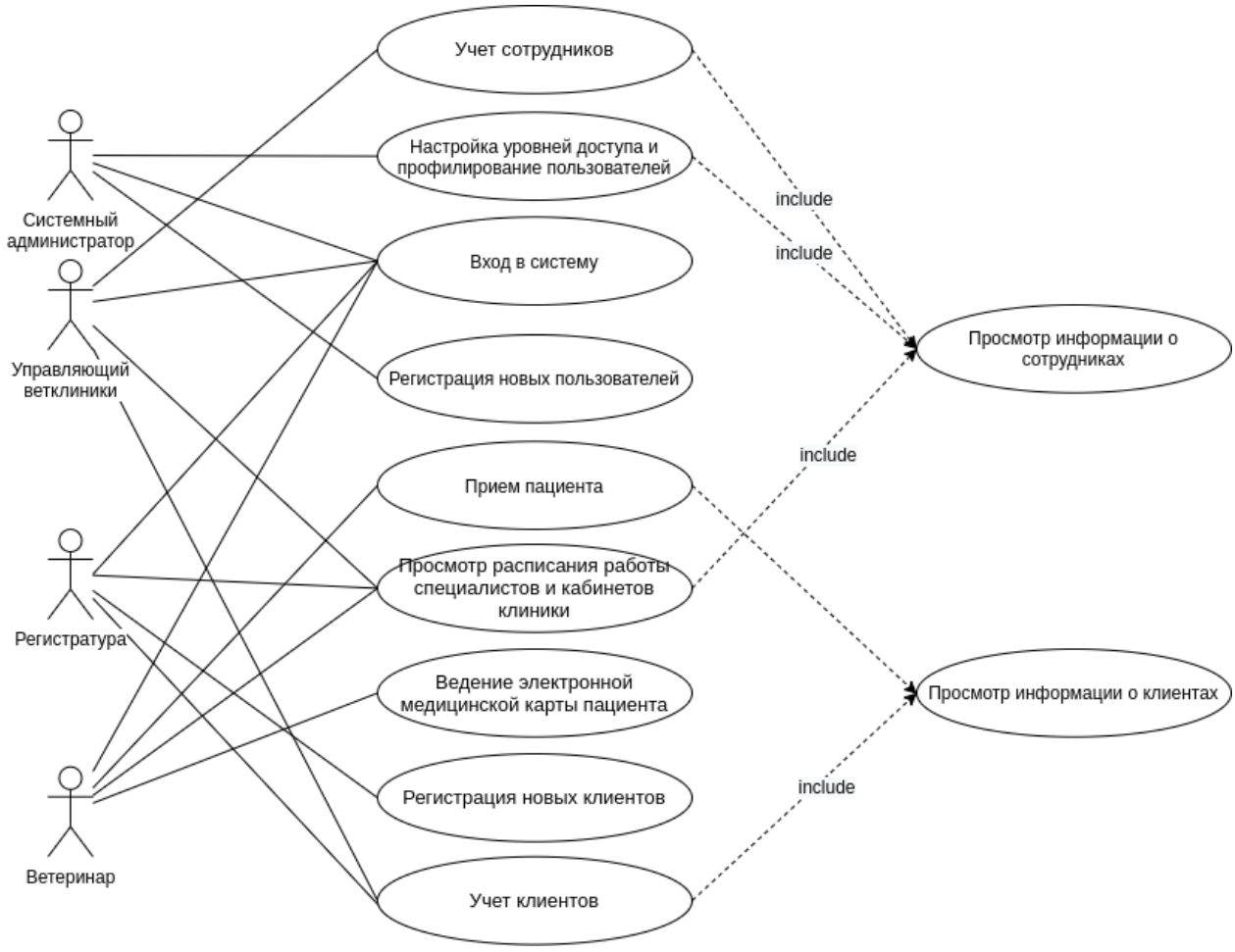

Рис. 2: Use-case диаграмма.

Выделяется четыре типа пользователей: системный администратор, управляющий ветеринарной клиники, сотрудник регистратуры и ветеринар. Пользователи взаимодействуют с системой и используют ее функциональные возможности для достижения определенных целей. Для каждого типа пользователей предусмотрен свой набор возможностей.

Системный администратор:

- настройка уровней доступа и профилирование пользователей;
- вход в систему;
- регистрация новых пользователей.

Управляющей ветеринарной клиники:

- вход в систему;
- просмотр расписания работы специалистов и кабинетов клиники;
- учет клиентов.

Регистратура:

- вход в систему;
- просмотр расписания работы специалистов и кабинетов клиники;
- регистрация новых клиентов;
- учет клиентов.

Ветеринар:

- вход в систему;
- прием пациента;
- просмотр расписани работы специалистов и кабинетов клиники;
- ведение электронной медицинской карты пациента.

1.8 Определение требований к структуре базы данных

Для проектирования базы данных необходимо определить природу данных, с которыми придется работать.

Исходя из анализа предметной области, можно выделить следующие категории данных:

- паспорт;
- адрес;
- клиент клиники;
- договор с клиентом;
- медицинская карта животного;
- чип животного;
- осмотр ветеринара;
- состояние животного;
- персонал клиники;
- должность в клинике;
- расписание;
- пользователи системы.

Каждая категория данных описывается информацией, описанной в таблице 1.

Таблица 1: Категории данных и информация, которую они содержат.

Категория	Информация
Паспорт	Фамилия; имя; отчество; пол; дата рождения; серия-номер паспорта; дата выдачи паспорта; национальность
Адрес	страна; город; улица; дом; квартира
Клиент клиники	контакты; адрес; паспорт
Договор с клиентом	номер договора; дата заключения; дата последнего обновления; клиент, с которым заключен договор; дата истечения срока договора
Чип животного	номер чипа, дата имплантации; страна имплантации; расположение чипа
Медицинская карта животного	имя; порода; вид; пол; отметка о кастрации; дата рождения; окрас; особые приметы; дата постановки на учет; чип; фотография; договор
Осмотр ветеринара	врач; животное-пациент; отметка об амбулаторном приеме; дата; динамика состояния животного со слов владельца; анамнез; текущее состояние; диагноз; рекоммендации; дата повторного осмотра (если необходимо); назначения; примечение; отметка о первичном осмотре
Состояние животного	общее состояние; пульс; вес; артериальное давление; температура; скорость наполнения капилляров; частота дыхательных движений
Расписание	сотрудник; день недели; время начала; время окончания; кабинет
Должность в клинике	название; зарплата
Персонал клиники	паспорт; должность; уровень образования; дата наема; дата увольнения
Пользователи системы	сотрудник; логин; пароль; уровень доступа.

1.9 Разработка логической модели данных

Концептуальную схему предметной области позволяет описать ER-модель (модель сущность-связь). С её помощью можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями.

Диаграмма сущность-связь, описывающая выбранную предметную область, представлена на рисунке 3.

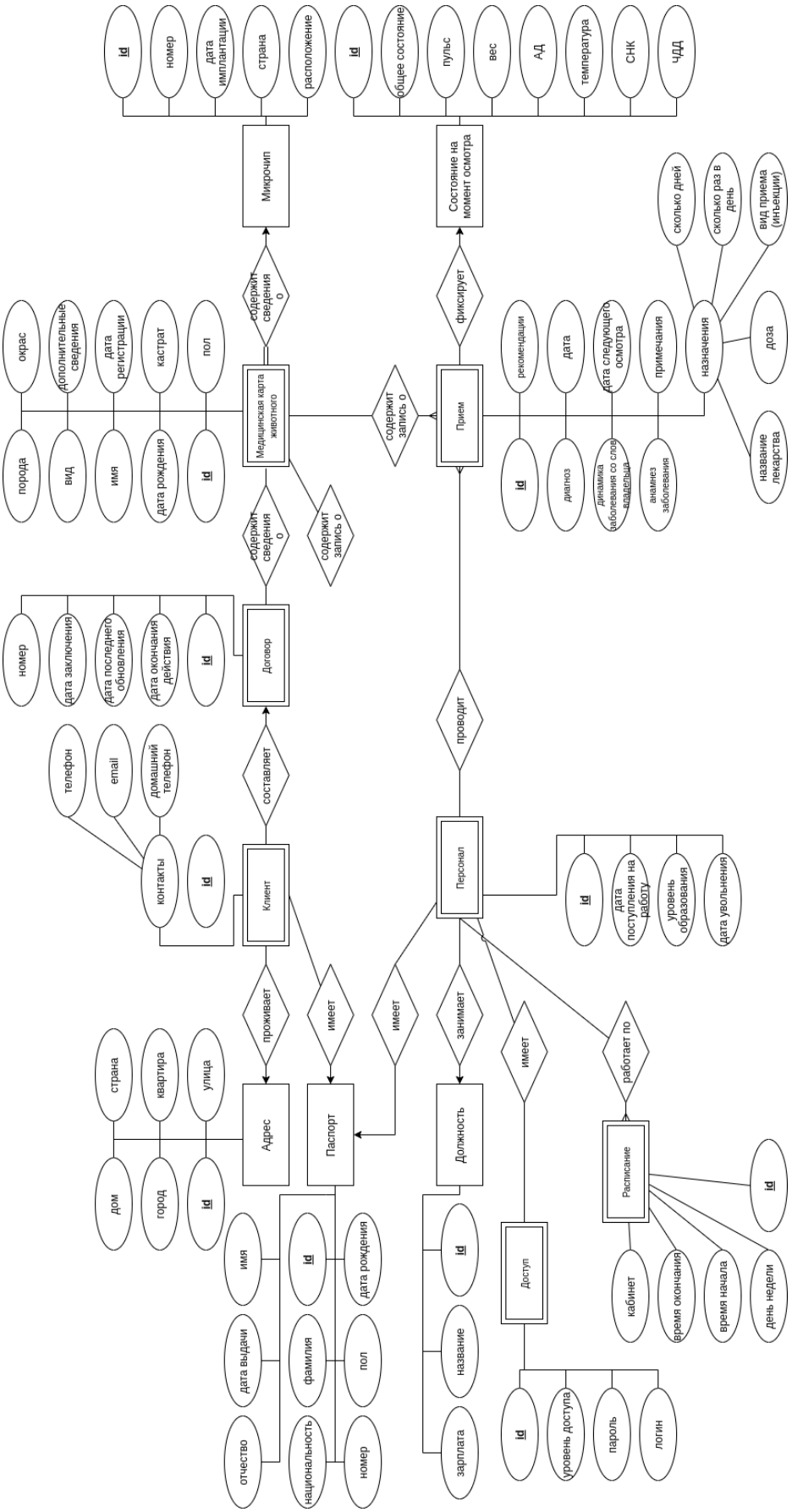

Рис. 3: ER-диаграмма.

1.10 Архитектура приложения

Архитектура программного обеспечения – совокупность важнейших решений об организации программной системы. Главной целью архитектуры ПО является упрощение разработки, развертывания и сопровождение программной системы.

В модели клиент-сервер роли определены следующим образом: сервер предоставляет ресурсы и службы одному или нескольким клиентам, которые обращаются к серверу за обслуживанием. Большинство серверов могут устанавливать отношение "один ко многим"с клиентами, что означает, что один сервер может предоставлять ресурсы нескольким клиентам одновременно.

Когда клиент запрашивает соединение с сервером, сервер может либо принять, либо отклонить это соединение. Если соединение принято, сервер устанавливает и поддерживает соединение с клиентом по определенному протоколу .

Часто клиенты и серверы взаимодействуют через компьютерную сеть на разных аппаратных средствах, но и клиент и сервер могут находиться в одной и той же системе. Хост сервера запускает одну или несколько серверных программ, которые совместно используют свои ресурсы с клиентами.

Клиент не предоставляет общий доступ ни к одному из своих ресурсов, но запрашивает данные или службу у сервера. Поэтому клиенты инициируют сеансы связи с серверами, которые ожидают входящих запросов. Клиенту не знает о том, как работает сервер при выполнении запроса и доставке ответа. Клиент должен только понимать ответ, основанный на хорошо известном прикладном протоколе, т. е. содержание и форматирование данных для запрашиваемой службы. Клиенты и серверы обмениваются сообщениями в шаблоне обмена сообщениями запрос-ответ . Клиент отправляет запрос, а сервер возвращает ответ.

Преимуществом модели взаимодействия клиент-сервер является то, что программный код клиентского приложения и серверного разделен. Кроме того, к машинам клиентов предъявляются пониженные требования, так как большая часть вычислительных операций будет производиться на сервере.

Схема взаимодействия по модели клиент-сервер представлена на рисунке 4.

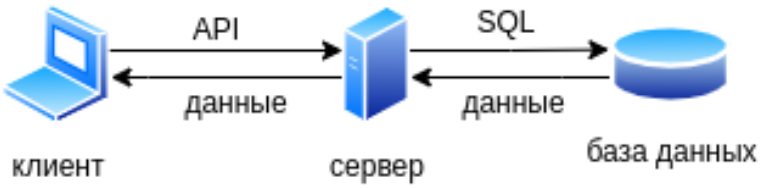

Рис. 4: Модель клиент-сервер.

Вывод из аналитического раздела

Таким образом, в данном разделе был проведен анализ предметной области, формализация требований к программе, а также рассмотрены различные способы хранения данных в приложениях. Был произведен выбор типа хранилища и выбор СУБД. В качестве СУБД была выбрана PostgreSQL. Были выделены различные категории пользователей системы и построена диаграмма вариантов использования. Кроме того, была разработана логическая модель данных и построена ER-диаграмма предметной области.

2 Конструкторский раздел

В данном разделе будет выполнено проектирование базы данных с учетом выбранной СУБД.

2.1 Проектирование таблиц базы данных

В предыдущем разделе были выделены ключевые сущности и обозначены связи между ними в рассматриваемой предметной области. В соответствии с выделенными выделенными сущностями, база данных должна содержать следующие таблицы.

- Таблица адресов – addresses.
- Таблица паспортов – passports.
- Таблица клиентов – clients.
- Таблица договоров – contracts.
- Таблица микрочипов – microchips.
- Таблица должностей – position.
- Таблица состояний животного – animal_states.
- Таблица медицинских карт животных – animal_medical_records.
- Таблица персонала – staff.
- Таблица осмотров – visits.
- Таблица пользователей системы – access.
- Таблица расписания – schedule.

Рассмотрим подробнее каждую из таблиц. Названия столбцов, их тип, ограничения и пояснения к столбцам каждой из 12 перечисленных таблиц БД представлены в таблицах 2-13. Типы выбраны в соответствии с теми тем, какие типы предоставляет СУБД PostgreSQL.

Таблица 2: Столбцы таблицы адресов addresses.

Название	Тип	Ограничения	Пояснение
addr_id	SERIAL	NOT NULL, PRIMARY KEY	Идентификатор адреса
country	VARCHAR(64)	NOT NULL	Наззвание страны
city	VARCHAR(255)	NOT NULL	Название города
street	VARCHAR(255)	NOT NULL	Название улицы
house	VARCHAR(10)	NOT NULL	Номер дома
flat	VARCHAR(10)	-	Номер квартиры (если есть)

Таблица 3: Столбцы таблицы паспортов passports.

Название	Тип	Ограничения	Пояснение
pass_id	SERIAL	NOT NULL, PRIMARY KEY	Идентификатор паспорта
surname	VARCHAR(64)	NOT NULL	Фамилия
name	VARCHAR(64)	NOT NULL	Имя
patronymic	VARCHAR(64)	-	Отчество (если есть)
sex	sex_type	NOT NULL	Пол (необходимо определить пользовательский тип)
birth	DATE	NOT NULL	Дата рождения
num	VARCHAR(10)	NOT NULL	Серия-номер паспорта
issue_date	DATE	NOT NULL	Дата выдачи паспорта
nationality	VARCHAR(128)	-	Национальность (если указана)

Таблица 4: Столбцы таблицы клиентов clients.

Название	Тип	Ограничения	Пояснение
cli_id	SERIAL	NOT NULL PRIMARY KEY	Идентификатор клиента
contacts	JSON	NOT NULL	Контакты
address	INT	NOT NULL	Идентификатор адреса клиента
passport	INT	NOT NULL	Идентификатор паспорта клиента

Таблица 5: Столбцы таблицы договоров contracts.

Название	Тип	Ограничения	Пояснение
contr_id	SERIAL	NOT NULL PRIMARY KEY	Идентификатор договора
code	VARCHAR(20)	NOT NULL	Номер договора
conclusion_date	DATE	NOT NULL	Дата заключения договора
last_update_date	DATE	NOT NULL	
owner	INT	NOT NULL	Дата последнего обновления договора (повторное заключение)
valid_until	DATE	NOT NULL	Дата истечения договора

Таблица 6: Столбцы таблицы микрочипов microchips.

Название	Тип	Ограничения	Пояснение
chip_id	SERIAL	NOT NULL PRIMARY KEY	Идентификатор чипа
chip_num	VARCHAR(15)	NOT NULL	Номер чипа
impl_date	DATE	NOT NULL	Дата имплантации
country	VARCHAR(3)	NOT NULL	Трехбуквенный код страны, где чип был имплантирован
location	TEXT	NOT NULL	Расположение чипа

Таблица 7: Столбцы таблицы должностей position.

Название	Тип	Ограничения	Пояснение
pos_id	SERIAL	NOT NULL PRIMARY KEY	Идентификатор должности
title	VARCHAR(256)	NOT NULL	Название должности
salary salary	INT INT	NOT NULL NOT NULL	Заработная плата (в рублях)

Таблица 8: Столбцы таблицы состояний животного animal_states.

Название	Тип	Ограничения	Пояснение
state_id	SERIAL	NOT NULL PRIMARY KEY	Идентификатор состояния
general	general_type	NOT NULL	Общее состояние (необходим пользовательский тип)
pulse	INT	NOT NULL	Пульс
weight	REAL	NOT NULL	Вес в кг
ap	VARCHAR(10)	NOT NULL	Артериальное давление
temperature	REAL	NOT NULL	Температура тела в градусах по Цельсию
cfr	INT	NOT NULL	Скорость наполнения каппиляров (в секундах)
resp_rate	INT	NOT NULL	Частота дыхательных движений

Таблица 9: Столбцы таблицы медицинских карт animal_medical_records.

Название	Тип	Ограничения	Пояснение
anim_id	SERIAL	NOT NULL PRIMARY KEY	Идентификатор карты
name	VARCHAR(64)	NOT NULL	Имя животного
breed	TEXT	NOT NULL	Порода
species	TEXT	NOT NULL	вид
sex	sex_type	NOT NULL	Пол
castrated	BOOLEAN	NOT NULL DEFAULT('f')	Отметка о стерилизации
birth	DATE	NOT NULL	Дата рождения
other_data	TEXT	NOT NULL	Дополнительные сведения
color	TEXT	NOT NULL	Окрас
special_signs	TEXT	NOT NULL	Особые приметы
registr_date	DATE	NOT NULL	Дата регистрации в клинике
chip_id	INT	NOT NULL	Идентификатор чипа
contract	INT	NOT NULL	Идентификатор
			договора
rel_path_to_photo	VARCHAR(255)	NOT NULL	Относительный путь до фотографии

Таблица 10: Столбцы таблицы персонала staff.

Название	Тип	Ограничения	Пояснение
staff_id	SERIAL	NOT NULL PRIMARY KEY	Идентификатор сотрудника
passport	INT	NOT NULL	Идентификатор паспорта
position	INT	NOT NULL	Идентификатор должности
edu_level	edu_level_type	NOT NULL	Уровень образования
fire_date	DATE	-	Дата увольнения (если уволен)
employ_date	DATE	NOT NULL	Дата найма

Таблица 11: Столбцы таблицы осмотров visits.

Название	Тип	Ограничения	Пояснение
vis_id	SERIAL	NOT NULL PRIMARY KEY	Идентификатор осмотра
doctor	INT	NOT NULL	Идентификатор ветеринара
animal	INT	NOT NULL	Идентификатор карты животного
visit_date	DATE	NOT NULL	Дата осмотра
owner_dynamics	owner_dynamics_type	NOT NULL	Динамика состояния со слов владельца
history_disease	TEXT	NOT NULL	Анамнез
cur_state	INT	NOT NULL	Идентификатор текущего состояния
diagnosis	TEXT	NOT NULL	Диагноз
recommendations	TEXT	NOT NULL	Рекоммендации
next_visit	DATE	-	Дата повторного осмотра
prescribings	JSON	-	Назначения
note	TEXT	-	Примечания

Таблица 12: Столбцы таблицы пользователей системы access.

Название	Тип	Ограничения	Пояснение
acc_id	SERIAL	NOT NULL PRIMARY KEY	Идентификатор аккаунта
employee	INT	NOT NULL	Идентификатор сотрудника
login	VARCHAR(64)	NOT NULL	Логин
password	BYTEA	NOT NULL	Пароль
access_level	access_level_type	NOT NULL	Уровень доступа (пользовательский тип)

Таблица 13: Столбцы таблицы расписания schedule.

Название	Тип	Ограничения	Пояснение
shed_id	SERIAL	NOT NULL PRIMARY KEY	Идентификатор расписания
employee_id	INT	NOT NULL	Идентификатор сотрудника
day_of_week	day_of_week_type	NOT NULL	День недели
start	TIMESTAMP	NOT NULL	Время начала рабочего дня
end	TIMESTAMP	NOT NULL	Время окончания рабочего дня
cabinet	VARCHAR(10)	-	кабинет (если есть)

2.2 Сценарий создания базы данных

Сцернарий создания базы данных приведен в листинге 1.

Листинг 1: Сценарий создания БД.

```
#!/bin/bash
# Required package: jq

ps_user=`jq '.username' dist.conf | tr -d \"`
db_name=`jq '.db_name' dist.conf | tr -d \"`
```

```
pwd=`pwd`

createdb -U $ps_user $db_name

cd ../scripts/update

files=`ls -r update_*.sql | sort`
for VAR in $files
do
    psql -U $ps_user -d $db_name -f $VAR
done

cd $pwd
```

Конфигурационный файл для этого сценария приведен в листинге 2.

Листинг 2: Конфигурационный файл.

```
{
    "username" : "postgres",
    "db_name" : "vet"
}
```

Сценарий первой ревизии БД представлен в листинге 3.

Листинг 3: Первая ревизия.

```
---- first revision

\i ../install/create_passport.sql
\i ../install/create_access.sql
\i ../install/create_addresses.sql
\i ../install/create_animal_medical_records.sql
\i ../install/create_animal_states.sql
\i ../install/create_client.sql
\i ../install/create_contract.sql
\i ../install/create_microchips.sql
\i ../install/create_position.sql
\i ../install/create_schedule.sql
\i ../install/create_staff.sql
\i ../install/create_visits.sql
```

2.3 Определение пользовательских типов в БД

Как видно из предыдущего подраздела, для создания таблиц БД необходимо определить несколько пользовательских типов. Сценарий создания пользовательских типов приведен в листинге 4.

Листинг 4: Определение прользовательских типов.

```
CREATE TYPE access_level_type AS ENUM('admin', 'main', 'registry', 'vet');

CREATE TYPE general_type AS ENUM('middle', 'good', 'bad');

CREATE TYPE sex_type AS ENUM ('m', 'f', 'other');

CREATE TYPE day_of_week_type AS ENUM('Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat');

CREATE TYPE edu_level_type AS ENUM('resident', 'middle', 'postgraduate', 'specialist', 'bachelor');

CREATE TYPE owner_dynamics_type AS ENUM('stably', 'worse', 'better');
```

2.4 Сценарий создания таблиц БД

Сценарий создания таблиц БД представлен в листинге 5.

Листинг 5: Создание таблиц БД.

```
CREATE TABLE access(
    acc_id SERIAL NOT NULL PRIMARY KEY,
    employee INT NOT NULL,
    login VARCHAR(64) NOT NULL,
    password BYTEA NOT NULL,
    access_level access_level_type NOT NULL
);

CREATE TABLE addresses(
    addr_id SERIAL NOT NULL PRIMARY KEY,
    country VARCHAR(64) NOT NULL,
    city VARCHAR(255) NOT NULL,
    street VARCHAR(255) NOT NULL,
    house VARCHAR(10) NOT NULL,
```

```
15      flat VARCHAR(10)
16  );
17
18  CREATE TABLE animals_medical_records(
19      anim_id SERIAL NOT NULL PRIMARY KEY,
20      name VARCHAR(64) NOT NULL,
21      breed TEXT NOT NULL,
22      species TEXT NOT NULL,
23      sex sex_type NOT NULL,
24      castrated BOOLEAN NOT NULL DEFAULT('f'),
25      birth DATE NOT NULL,
26      other_data TEXT NOT NULL,
27      color TEXT NOT NULL,
28      special_signs TEXT NOT NULL,
29      registr_date DATE NOT NULL,
30      chip_id INT NOT NULL,
31      contract INT NOT NULL,
32      rel_path_to_photo VARCHAR(255) NOT NULL
33  );
34
35  CREATE TABLE animal_states(
36      state_id SERIAL NOT NULL PRIMARY KEY,
37      general general_type NOT NULL,
38      pulse INT NOT NULL,
39      weight REAL NOT NULL,
40      ap VARCHAR(10) NOT NULL,
41      temperature REAL NOT NULL,
42      cfr INT NOT NULL,
43      resp_rate INT NOT NULL
44  );
45
46  CREATE TABLE clients(
47      cli_id SERIAL NOT NULL PRIMARY KEY,
48      contacts JSON NOT NULL,
49      address INT NOT NULL,
50      passport INT NOT NULL
51  );
52
53  CREATE TABLE contract(
54      contr_id SERIAL NOT NULL PRIMARY KEY,
55      code VARCHAR(20) NOT NULL,
```

```
56      conclusion_date DATE NOT NULL,
57      last_update_date DATE NOT NULL,
58      "owner" INT NOT NULL,
59      valid_until DATE NOT NULL
60  );
61
62  CREATE TABLE microchips(
63      chip_id SERIAL NOT NULL PRIMARY KEY,
64      chip_num VARCHAR(15) NOT NULL,
65      impl_date DATE NOT NULL,
66      country VARCHAR(3) NOT NULL,
67      location TEXT NOT NULL
68  );
69
70  CREATE TABLE passports(
71      pass_id SERIAL NOT NULL PRIMARY KEY,
72      surname VARCHAR(64) NOT NULL,
73      name VARCHAR(64) NOT NULL,
74      patronymic VARCHAR(64),
75      sex sex_type NOT NULL,
76      birth DATE NOT NULL,
77      num VARCHAR(10) NOT NULL,
78      issue_date DATE NOT NULL,
79      nationality VARCHAR(128)
80  );
81
82  CREATE TABLE position(
83      pos_id SERIAL NOT NULL PRIMARY KEY,
84      title VARCHAR(256) NOT NULL,
85      salary INT NOT NULL
86  );
87
88  CREATE TABLE schedule(
89      shed_id SERIAL NOT NULL PRIMARY KEY,
90      employee_id INT NOT NULL,
91      day_of_week day_of_week_type NOT NULL,
92      "start" TIME NOT NULL,
93      "end" TIME NOT NULL,
94      cabinet VARCHAR(10)
95  );
96
```

```
97 CREATE TABLE staff(
98     staff_id SERIAL NOT NULL PRIMARY KEY,
99     passport INT NOT NULL,
100    position INT NOT NULL,
101    edu_level edu_level_type NOT NULL,
102    fire_date DATE,
103    employ_date DATE NOT NULL
104 );
105
106 CREATE TABLE visits(
107    vis_id SERIAL NOT NULL PRIMARY KEY,
108    doctor INT NOT NULL,
109    animal INT NOT NULL,
110    visit_date DATE NOT NULL,
111    owner_dynamics owner_dynamics_type NOT NULL,
112    history_disease TEXT NOT NULL,
113    cur_state INT NOT NULL,
114    diagnosis TEXT NOT NULL,
115    recommendations TEXT NOT NULL,
116    next_visit DATE,
117    prescribings JSON,
118    note TEXT
119 );
```

Вторая ревизия базы данных – создание ограничений в таблицах – представлена в листинге 6.

Листинг 6: Создание органичений.

```
1 ---- add constraint
2
3 ALTER TABLE clients ADD CONSTRAINT addr_client_fk
4     FOREIGN KEY (address) REFERENCES addresses(addr_id)
5     ON DELETE CASCADE;
6
7 ALTER TABLE clients ADD CONSTRAINT pass_client_fk
8     FOREIGN KEY (passport) REFERENCES passports(pass_id)
9     ON DELETE CASCADE;
10
11 ALTER TABLE staff ADD CONSTRAINT pass_staff_fk
12    FOREIGN KEY (passport) REFERENCES passports(pass_id)
13    ON DELETE CASCADE;
14
```

```
15 ALTER TABLE staff ADD CONSTRAINT pos_staff_fk
16     FOREIGN KEY (position) REFERENCES position(pos_id)
17     ON DELETE CASCADE;
18
19 ALTER TABLE access ADD CONSTRAINT stf_empl_fk
20     FOREIGN KEY (employee) REFERENCES staff(staff_id)
21     ON DELETE CASCADE;
22
23 ALTER TABLE contract ADD CONSTRAINT own_cntr_fk
24     FOREIGN KEY ("owner") REFERENCES clients(cli_id)
25     ON DELETE CASCADE;
26
27 ALTER TABLE schedule ADD CONSTRAINT empl_schdl_fk
28     FOREIGN KEY (employee_id) REFERENCES staff(staff_id)
29     ON DELETE CASCADE;
30
31 ALTER TABLE animals_medical_records ADD CONSTRAINT anmr_ctrc_fk
32     FOREIGN KEY (contract) REFERENCES contract(contr_id)
33     ON DELETE CASCADE;
34
35 ALTER TABLE animals_medical_records ADD CONSTRAINT anmr_mchp_fk
36     FOREIGN KEY (chip_id) REFERENCES microchips(chip_id)
37     ON DELETE CASCADE;
38
39 ALTER TABLE visits ADD CONSTRAINT vis_schdl_fk
40     FOREIGN KEY (doctor) REFERENCES staff(staff_id)
41     ON DELETE CASCADE;
42
43 ALTER TABLE visits ADD CONSTRAINT vis_anmr_fk
44     FOREIGN KEY (animal) REFERENCES animals_medical_records(anim_id)
45     ON DELETE CASCADE;
46
47 ALTER TABLE visits ADD CONSTRAINT vsts_asts_fk
48     FOREIGN KEY (cur_state) REFERENCES animal_states(state_id)
49     ON DELETE CASCADE;
50
51 ALTER TABLE passports ADD CONSTRAINT unique_pass_num UNIQUE(num);
52
53 ALTER TABLE microchips ADD CONSTRAINT unique_mchip_num UNIQUE(chip_num);
```

2.5 Диаграмма базы данных

Диаграмма базы данных представлена на рисунке 5.

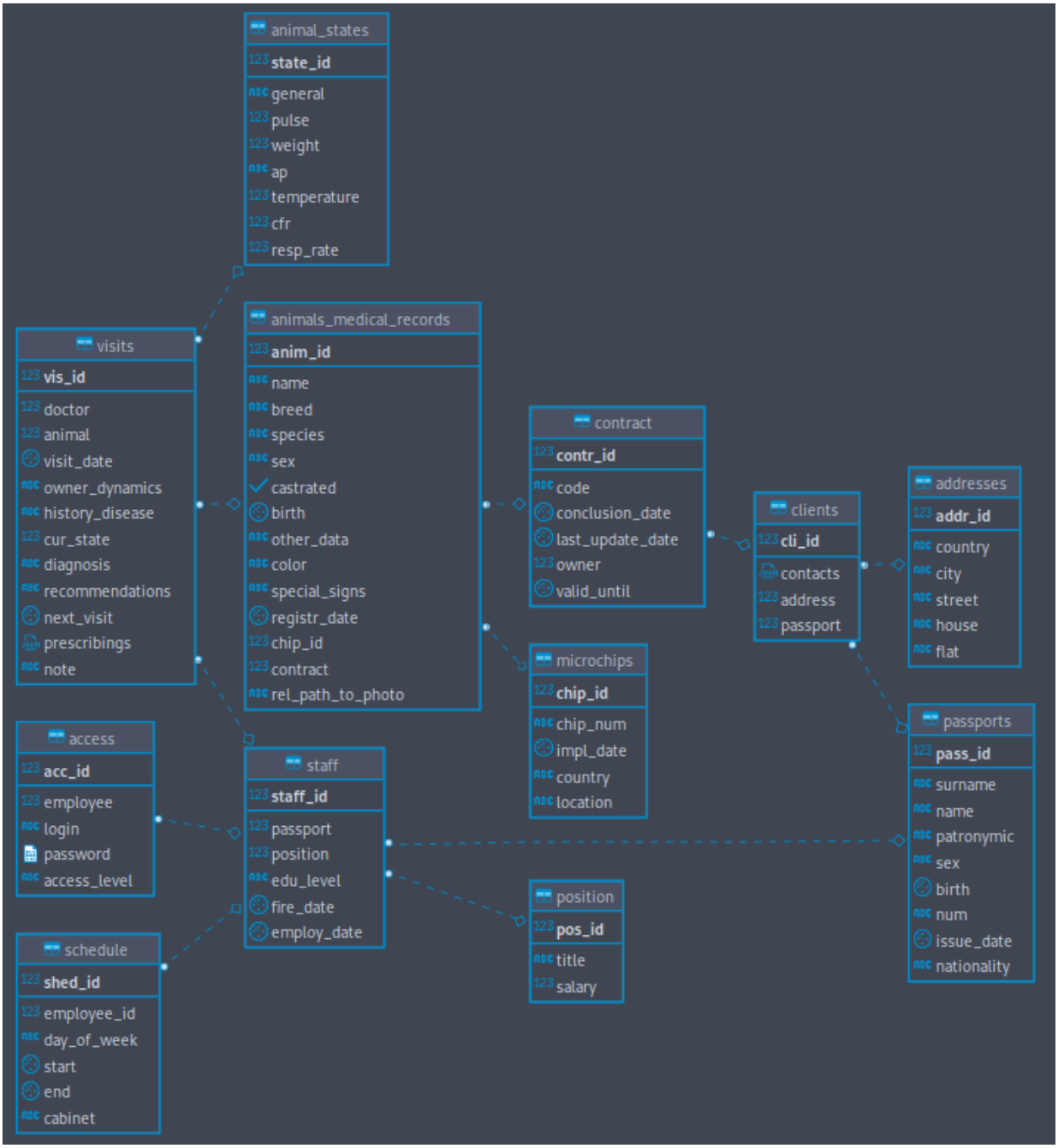

Рис. 5: Диаграмма базы данных.

2.6 Паттерны проектирования

В настоящее время при создании приложений с графическими пользовательскими интерфейсами применяются специальные паттерны проектирования, позволяющие синхронизировать разные функциональные области программного про-

дукта между собой. Такие паттерны предназначены для уменьшения трудозатрат (и, как следствие, финансовых затрат) на разработку сложного программного обеспечения.

Фундаментальным паттерном, который нашел применение во многих технологиях и дал развитие новым, является паттерн MVC (Model-View-Controller). Этот паттерн позволяет отделить графический пользовательский интерфейс от логики программирования. Это позволяет разрабатывать логику независимо от GUI.

MVC состоит из трех компонент: View (представление, пользовательский интерфейс), Model (модель, бизнес логика) и Controller (контроллер, содержит логику на изменение модели при определенных действиях пользователя, реализует Use Case). Основная идея этого паттерна в том, что и контроллер и представление зависят от модели, но модель никак не зависит от этих двух компонент. Это позволяет разрабатывать и тестировать модель, ничего не зная о представлениях и контроллерах. Контроллер так же ничего не должен знать о представлении (в иделе, а на практике это не всегда так).

Модель предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления (не знает как данные визуализировать) и контроллера (не имеет точек взаимодействия с пользователем), просто предоставляя доступ к данным и управлению ими.

Модель строится таким образом, чтобы отвечать на запросы, изменяя своё состояние. Модель, за счёт независимости от визуального представления, может иметь несколько различных представлений.

Представление отвечает за получение необходимых данных из модели и отправляет их пользователю. Представление не обрабатывает введённые данные пользователя.

Контроллер обеспечивает связь между пользователем и системой, контролирует и направляет данные от пользователя к системе и наоборот. Контроллер использует модель и представление для реализации необходимого действия.

2.7 Архитектура REST

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web. REST является очень простым интерфейсом управления информацией без использования

каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы.

Для каждой единицы информации (info) определяется 5 действий.

- GET /info/(Index) – получает список всех объектов. Как правило, это упрощенный список, то есть содержащий только поля идентификатора и названия объекта, без остальных данных.
- GET /info/id(View) – получает полную информацию об объекте.
- PUT /info/илиPOST /info/(Create) – создает новый объект. Данные передаются в теле запроса без применения кодирования, даже urlencode.
- POST /info/idилиPUT /info/id(Edit) – изменяет данные с идентификатором id, возможно, заменяетих. Данные также передаются в теле запроса, но, в отличие от PUT, здесь есть некоторый нюанс. Дело в том, что POST-запрос подразумевает наличие urldecoded-post-data. Т.е. если не применять кодирования – это нарушение стандарта.
- DELETE /info/id(Delete) – удаляет данные с идентификатором id.

Существует шесть обязательных ограничений для построения распределённых REST-приложений.

1. Модель клиент-сервер. Необходимо привести архитектуру приложения к модели клиент-сервер.
2. Отсутствие состояния. В период между запросами клиента никакая информация о состоянии клиента на сервере не хранится. То есть при выполнении запроса клиент передает серверу всю необходимую для выполнения запроса информацию.
3. Кэширование. Клиенты могут выполнять кэширование ответов сервера.
4. Единообразие интерфейса. К унифицированным интерфейсам предъявляются следующие требования.

- Идентификация ресурсов. Все ресурсы идентифицируются в запросах.
- Манипуляция ресурсами через представление.
- «Самоописываемые» сообщения. Каждое сообщение содержит достаточно информации, чтобы понять, каким образом его обрабатывать.

5. Слои. Клиент обычно не способен точно определить, взаимодействует он напрямую с сервером или же с промежуточным узлом, в связи с иерархической структурой сетей.

6. Код по требованию. REST может позволить расширить функциональность клиента за счёт загрузки кода с сервера в виде апплетов или сценариев.

Вывод из конструкторского раздела

Таким образом, в данном разделе была спроектирована база данных, были разработаны сценарии создания базы данных и приведена ее окончательная схема. Кроме того, был рассмотрен фундаментальный паттерн проектирования, который будет применяться в настоящей работе.

3 Технологический раздел

В данном разделе будут рассмотрены средства реализации проекта, приведены листинги и интерфейс приложения.

3.1 Выбор языка программирования и среды разработки

Выберем языки программирования и фреймворки для написания серверной и клиентской частей приложения.

3.1.1 Серверная часть

В качетсве языка программирования для написания серверной части был выбран Python и библиотека Flask. Flask – это легковесный фреймворк для создания веб-приложений на Python, использующий набор инструментов Werkzeug.

Flask можно считать лучшим веб-фреймворком для создания веб-приложений на Python по следующим причинам.

- Flask практически не зависит от внешних библиотек.
- Flask легок для понимания, имеет простую структуру и интуитивно понятный синтаксис.
- Flask позволяет довольно быстро создать веб-приложение, в отличие, от, например, Django.
- Гибкость. Большую часть инструментов фреймворка можно редактировать под свои нужды.
- Наличие хороших инструментов для тестирования, встроенного сервера разработки.

Недостатком Flask является то, что он не поддерживает асинхронность и обрабатывает все запросы в один поток. Однако в рамках учебного проекта этот недостаток незначителен.

Взаимодействие с базой данных в Python

Для взаимодействия с базой данных в Python был выбран модуль pyodbc. pyodbc - это модуль Python с открытым исходным кодом, который упрощает доступ к базам данных ODBC. Он реализует спецификацию DB API 2.0. Python DB

API определяет независимый от базы данных интерфейс для данных, хранящихся в реляционных базах данных.

3.1.2 Клиентская часть

Для разработки клиентского приложения был выбран язык C++ и фреймворк Qt. Главным преимуществом Qt является платформонезависимость. Qt предоставляет под держку большого числа операционных систем: Microsoft Windows, Mae OS X, Linux, FreeBSD и других клонов UNIX с X 11, а также и для мобильных операционных систем IOS, Android, Windows Phone, Windows RT и BlackBerry.

Qt использует интерфейс API низкого уровня, что позволяет кроссплатформенным приложениям работать не менее эффективно, чем приложениям, разработанным под конкретную платформу. Qt не только является средством создания интерфейса пользователя, но и предоставляет полный инструментарий для программирования, который включает в себя поддержку двух- и трехмерной графики, возможность интернационализации, использование форматов JSON и XML, STL-совместимую библиотеку контейнеров, поддержку стандартных протоколов ввода/вывода, классы для работы с сетью, поддержку программирования баз данных, включая Oracle, Microsoft SQL Server, IBM 082, MySQL, SQLite, Sybase, PostgreSQL и т.д.

Qt является полностью объектно-ориентированной библиотекой, использующей новую концепцию межобъектных коммуникаций «сигналы и слоты». Qt имеет подробную официальную документацию, что значительно упрощает ее изучение.

Сетевое взаимодействие на стороне клиента

Первоначально для отправки сетевых запросов и получения ответов использовался класс QNetworkAccessManager. Однако он приводил к крахам всего приложения, поэтому был написанный собственный класс NetworkFetcher для общения с сервером. Он был написан с использованием библиотеки libcurl.

libcurl – это бесплатная и простая в использовании библиотека для передачи URL-адресов на стороне клиента. libcurl является легко переносимой библиотекой, она одинаково работает на многих платформах. Кроме того, libcurl является хорошо документированной.

Заголовочный файл класса NetworkFetcher приведен в листинге 7, а реализация – в листинге 8.

Листинг 7: network_fetcher.h.

```
#pragma once

#include <memory>
#include <curl/curl.h>
#include <QUrl>
#include <QVector>
#include <QNetworkRequest>

class Multipart final
{
    void addPart(const QString&, const QByteArray&);
    void addFilePart(const QString&, const QString&);

private:
    using Part = std::tuple<QByteArray, bool, QString>;
    QMap<QString, Part> parts;
};

class NetworkFetcher final
{
    struct RequestInfo;
    struct Info;
public:
    using Header = std::tuple<QString, QByteArray>;
    using AuthData = std::tuple<QString, QString>;
    using Response = std::tuple<int32_t, double, QByteArray, QVector<
        Header>>;

public:
    NetworkFetcher();
    ~NetworkFetcher() noexcept;

    Response httpGet(const QNetworkRequest&, std::chrono::milliseconds);
    Response httpPost(const QNetworkRequest&, const QByteArray&, std::
        chrono::milliseconds);
    Response httpPost(const QNetworkRequest&, Multipart&&, std::chrono::
        milliseconds);

private:
    Response performRequest(const QUrl&);
```

```
38      void setCurlOptions();
39      void appendContentLenght(const QNetworkRequest &, size_t);
40      QString getUserAgent();
41      void setRequest(const QNetworkRequest&, std::chrono::milliseconds,
            bool = false);
42
43  private:
44      CURL* curl;
45      std::unique_ptr<RequestInfo> req_info;
46      std::unique_ptr<Info> info;
47  };
```

Листинг 8: network_fetcher.cpp.

```
1   #include <chrono>
2   #include <QDebug>
3   #include "network_fetcher.h"
4
5   static auto setopt = &curl_easy_setopt;
6   static auto getinfo = &curl_easy_getinfo;
7
8   struct NetworkFetcher::RequestInfo
9   {
10      double totalTime;
11      double nameLookupTime;
12      double connectTime;
13      double appConnectTime;
14      double preTransferTime;
15      double startTransferTime;
16      double redirectTime;
17      int8_t redirectCount;
18  };
19
20  struct NetworkFetcher::Info
21  {
22      QUrl baseUrl;
23      QVector<Header> headers;
24      std::chrono::milliseconds timeout;
25      bool followRedirects;
26      int16_t maxRedirects;
27      bool noSignal;
28      AuthData basicAuth;
```

```
29      QString certPath;
30      QString certType;
31      QString keyPath;
32      QString keyPassword;
33      QString customUserAgent;
34      QString uriProxy;
35      QString proxyAuth;
36      RequestInfo lastRequest;
37  };
38
39  NetworkFetcher::NetworkFetcher():
40      req_info(std::make_unique<RequestInfo>()),
41      info(std::make_unique<Info>())
42  {
43      curl = curl_easy_init();
44      info->timeout= std::chrono::milliseconds(0);
45      info->followRedirects = false;
46      info->noSignal = false;
47      info->maxRedirects = -1l;
48  }
49
50  NetworkFetcher::~NetworkFetcher() noexcept
51  {
52      curl_free(curl);
53  }
54
55  NetworkFetcher::Response NetworkFetcher::httpGet(const QNetworkRequest &
        req, std::chrono::milliseconds timeout)
56  {
57      setRequest(req, timeout);
58      return performRequest(req.url());
59  }
60
61  NetworkFetcher::Response NetworkFetcher::httpPost(const QNetworkRequest
        & req, const QByteArray & data, std::chrono::milliseconds tout)
62  {
63      size_t size = static_cast<size_t>(data.size());
64      setRequest(req, tout, true);
65      appendContentLenght(req, size);
66
67      const char* __data = data.data();
```

```
68      setopt(curl, CURLOPT_POST, 1L);
69      setopt(curl, CURLOPT_POSTFIELDS, __data);
70      setopt(curl, CURLOPT_POSTFIELDSIZE, size);
71
72      return performRequest(req.url());
73  }
74
75  extern "C"
76  {
77      size_t writeCallback(void *data, size_t size, size_t nmemb, void *
            userdata)
78      {
79          NetworkFetcher::Response* r;
80          r = static_cast<NetworkFetcher::Response*>(userdata);
81          size_t sz = size * nmemb;
82          std::get<2>(*r).append(QByteArray(static_cast<char*>(data),
                static_cast<int>(sz)));
83          return (sz);
84      }
85
86      size_t headerCallback(void *data, size_t size, size_t nmemb, void *
            userdata)
87      {
88          NetworkFetcher::Response* r;
89          r = reinterpret_cast<NetworkFetcher::Response*>(userdata);
90          size_t _sz = size*nmemb;
91          QByteArray header(static_cast<char*>(data), static_cast<int>(_sz
                ));
92          int idx = header.indexOf(':');
93          if ( idx == -1)
94          {
95              header = header.trimmed();
96              if (header.isEmpty())
97              {
98                  return _sz;
99              }
100             std::get<3>(*r).append({header, "present"});
101         }
102         else
103         {
104             QByteArray key = header.left(idx).trimmed();
```

```
105             QByteArray value = header.mid(idx + 1).trimmed();
106             std::get<3>(*r).append({key, value});
107         }
108         return _sz;
109     }
110 }
111
112 NetworkFetcher::Response NetworkFetcher::performRequest(const QUrl &url)
113 {
114     Response ret;
115
116     CURLcode res = CURLE_OK;
117     curl_slist* headerList = nullptr;
118
119     char* url_data = url.toString().toLatin1().data();
120     setopt(curl, CURLOPT_URL, url_data);
121     setopt(curl, CURLOPT_WRITEFUNCTION, writeCallback);
122     setopt(curl, CURLOPT_WRITEDATA, &ret);
123     setopt(curl, CURLOPT_HEADERFUNCTION, headerCallback);
124     setopt(curl, CURLOPT_HEADERDATA, &ret);
125
126     for (const auto& header : info->headers)
127     {
128         char* hdr_str = std::get<0>(header).toLatin1().append(": ").
               append(std::get<1>(header)).data();
129         headerList = curl_slist_append(headerList, hdr_str);
130     }
131
132     setopt(curl, CURLOPT_HTTPHEADER, headerList);
133     setCurlOptions();
134
135     res = curl_easy_perform(curl);
136     qDebug() << curl_easy_strerror(res);
137     if (res != CURLE_OK)
138     {
139         switch (res)
140         {
141             case CURLE_OPERATION_TIMEDOUT:
142                 std::get<0>(ret) = res;
143                 std::get<2>(ret) = "Operation Timeout.";
144                 break;
```

```
145             case CURLE_SSL_CERTPROBLEM:
146                 std::get<0>(ret) = res;
147                 std::get<2>(ret) = curl_easy_strerror(res);
148                 break;
149         default:
150                 std::get<0>(ret) = -1;
151                 std::get<2>(ret) = "Failed to query.";
152         }
153     }
154     else
155     {
156         int32_t http_code = 0;
157         getinfo(curl, CURLINFO_RESPONSE_CODE, &http_code);
158         std::get<0>(ret) = static_cast<int32_t>(http_code);
159     }
160 
161     getinfo(curl, CURLINFO_TOTAL_TIME, &info->lastRequest.totalTime);
162     getinfo(curl, CURLINFO_NAMELOOKUP_TIME, &info->lastRequest.
        nameLookupTime);
163     getinfo(curl, CURLINFO_CONNECT_TIME, info->lastRequest.connectTime);
164     getinfo(curl, CURLINFO_APPCONNECT_TIME, &info->lastRequest.
        appConnectTime);
165     getinfo(curl, CURLINFO_PRETRANSFER_TIME, &info->lastRequest.
        preTransferTime);
166     getinfo(curl, CURLINFO_STARTTRANSFER_TIME, &info->lastRequest.
        startTransferTime);
167     getinfo(curl, CURLINFO_REDIRECT_TIME, &info->lastRequest.
        redirectTime);
168     getinfo(curl, CURLINFO_REDIRECT_COUNT, &info->lastRequest.
        redirectCount);
169 
170     curl_slist_free_all(headerList);
171     curl_easy_reset(curl);
172 
173     std::get<1>(ret) = info->lastRequest.totalTime;
174     return ret;
175 
176 }
177 
178 void NetworkFetcher::setCurlOptions()
179 {
```

```
180     auto& auth = info->basicAuth;
181     if (std::get<0>(auth).isEmpty() == false)
182     {
183         char* auth_str = QString("%1:%2").arg(std::get<0>(auth), std::
                get<1>(auth)).toLatin1().data();
184         setopt(curl, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
185         setopt(curl, CURLOPT_USERPWD, auth_str);
186     }
187     setopt(curl, CURLOPT_USERAGENT, getUserAgent().data());
188     if (info->timeout > std::chrono::milliseconds(0))
189     {
190         setopt(curl, CURLOPT_TIMEOUT_MS, info->timeout.count());
191         setopt(curl, CURLOPT_NOSIGNAL, 1);
192     }
193     else
194     {
195         setopt(curl, CURLOPT_TIMEOUT, 0);
196         setopt(curl, CURLOPT_NOSIGNAL, 1);
197     }
198
199     if (info->followRedirects == true)
200     {
201         setopt(curl, CURLOPT_FOLLOWLOCATION, 1L);
202         setopt(curl, CURLOPT_MAXREDIRS, static_cast<int64_t>(info->
                maxRedirects));
203     }
204
205     if (info->noSignal)
206     {
207         setopt(curl, CURLOPT_NOSIGNAL, 1);
208     }
209
210     if (info->certPath.isEmpty() == false)
211     {
212         char* data = info->certPath.toLatin1().data();
213         setopt(curl, CURLOPT_SSLCERT, data);
214     }
215
216     if (info->certType.isEmpty() == false)
217     {
218         char* data = info->certType.toLatin1().data();
```

```
219         setopt(curl, CURLOPT_SSLCERTTYPE, data);
220     }
221 
222     if ( info->keyPath.isEmpty() == false)
223     {
224         char* data = info->keyPath.toLatin1().data();
225         setopt(curl, CURLOPT_SSLKEY, data);
226     }
227 
228     if (info->keyPassword.isEmpty() == false)
229     {
230         char* data = info->keyPassword.toLatin1().data();
231         setopt(curl, CURLOPT_KEYPASSWD, data);
232     }
233 
234     if (info->uriProxy.isEmpty() == false)
235     {
236         char* data = info->uriProxy.toLatin1().data();
237         setopt(curl, CURLOPT_PROXY, data);
238         setopt(curl, CURLOPT_HTTPPROXYTUNNEL, 1L);
239 
240         if (info->proxyAuth.isEmpty() == false)
241         {
242             char* data = info->proxyAuth.toLatin1().data();
243             setopt(curl, CURLOPT_PROXYAUTH, CURLAUTH_ANY);
244             setopt(curl, CURLOPT_PROXYUSERPWD, data);
245         }
246     }
247 
248     setopt(curl, CURLOPT_NOPROGRESS, 1L);
249 }
250 
251 void NetworkFetcher::appendContentLenght(const QNetworkRequest &request,
        size_t uploadSize)
252 {
253     QList<QByteArray> raw_headers = request.rawHeaderList();
254     for (const QByteArray & raw_header : raw_headers)
255     {
256         if (raw_header.toLower() == "content-length")
257         {
258             return;
```

```
259         }
260     }
261
262     qulonglong sz = uploadSize;
263     info->headers.push_back({"content-length", QByteArray::number(sz)});
264 }
265
266 QString NetworkFetcher::getUserAgent()
267 {
268     return QString("vetclinic/0.0.1");
269 }
270
271 void NetworkFetcher::setRequest(const QNetworkRequest &request,
272                                 std::chrono::milliseconds timeout,
273                                 bool unsetExpect)
274 {
275     info->timeout = timeout;
276
277     QVector<Header> headers;
278     QList<QByteArray> rawHeaders = request.rawHeaderList();
279     for (const QByteArray & rawHeader : rawHeaders)
280     {
281         const QByteArray & raw_header_data = request.rawHeader(rawHeader
            );
282         headers.push_back({rawHeader, raw_header_data});
283     }
284
285     if (unsetExpect == true)
286     {
287         headers.push_back({"Expect", ""});
288     }
289
290     info->headers = std::move(headers);
291 }
292
293 void Multipart::addPart(const QString & name, const QByteArray & body)
294 {
295     parts[name] = { body, false, "" };
296 }
```

3.2 Реализация SQL-запросов

Для реализации необходимого функционала были написаны SQL-запросы, приведенные в листинге 9 (запросы записаны как строка форматирования Python, в процессе обработки запроса она дополняется данными, полученными по сети).

Листинг 9: SQL-запросы.

```
SELECT anim_id, name, species, birth FROM animals_medical_records;

SELECT * FROM animals_medical_records a
    LEFT JOIN contract ON scontract=contr_id
    LEFT JOIN microchips m ON a.chip_id=m.chip_id
    WHERE anim_id={};

SELECT * FROM clients c
    LEFT JOIN passports ON passport=pass_id
    LEFT JOIN addresses ON address=addr_id
    WHERE cli_id={};

SELECT a.login, a.access_level, s.*, p.*, ps.* FROM staff s
    LEFT JOIN access a ON s.staff_id=a.employee
    JOIN position p ON position=p.pos_id
    JOIN passports ps ON s.passport=ps.pass_id
    WHERE a.login='{}' AND a.password='{}';

INSERT INTO visits (doctor, animal, visit_date, owner_dynamics,
   history_disease, cur_state, diagnosis, recommendations, next_visit,
   prescribings, note) VALUES ({}, {}, '{}', '{}', '{}', '{}', '{}', '{}
   ', {}, '{}', '{}');

INSERT INTO animal_states (general, pulse, weight, ap, temperature, cfr,
    resp_rate) VALUES ({}, {}, {}, {}, {}, {}, {}) RETURNING state_id;

SELECT day_of_week, start, "end", cabinet FROM schedule s WHERE s.
   employee_id={};

SELECT acc_id, login, surname, name,
        patronymic, access_level, title, employ_date, fire_date
        FROM access a JOIN staff s ON a.employee=s.staff_id
        JOIN position p ON s.position=p.pos_id
        JOIN passports pas ON s.passport=pas.pass_id;

```

```
SELECT a.name, a.species
    FROM visits v JOIN animals_medical_records a
    ON v.animal = a.anim_id
    WHERE v.next_visit = '{}';
```

3.3 Регистрация, авторизация и аутентификация пользователей

Регистрация новых пользователей в системе производится только системным администратором, только он обладает правами для совершения данного действия.

При авторизации уже зарегестрированных пользователей клиенту выдается сессионный ключ (генерируется uuid). Полученный сессионный ключ клиент впоследствии передает серверу при запросах. Сессионный ключ позволяет совершать запросы только авторизованным пользователям. Срок, в течение которого действителе ключ, определяется в когфигурационном файле. После истечения ключа необходима повторная авторизация.

Подобную схему использует протокол сетевой аутентификации Kerberos, который предлагает механизм взаимной аутентификации клиента и сервера перед установлением связи между ними, причём в протоколе учтён тот факт, что начальный обмен информацией между клиентом и сервером происходит в незащищённой среде, а передаваемые пакеты могут быть перехвачены и модифицированы.

3.4 Реализация сервера

Диаграмма классов сервера представлена на рисунке 6.

Пример реализации обработчика запросов (дочернего класса AbstractHandler) приведен в листинге 10. Класс AuthHandler обрабатывает запрос на авторизацию.

Листинг 10: AuthHandler.py.

```
from .abstract_handler import AbstractHandler
from database.dbconn import DBConn
from database.dbquery import DBQuery
from database.dbaccess_manager import access_manager
from server.key_data_checker import valid_key_checker
import json
import uuid
```

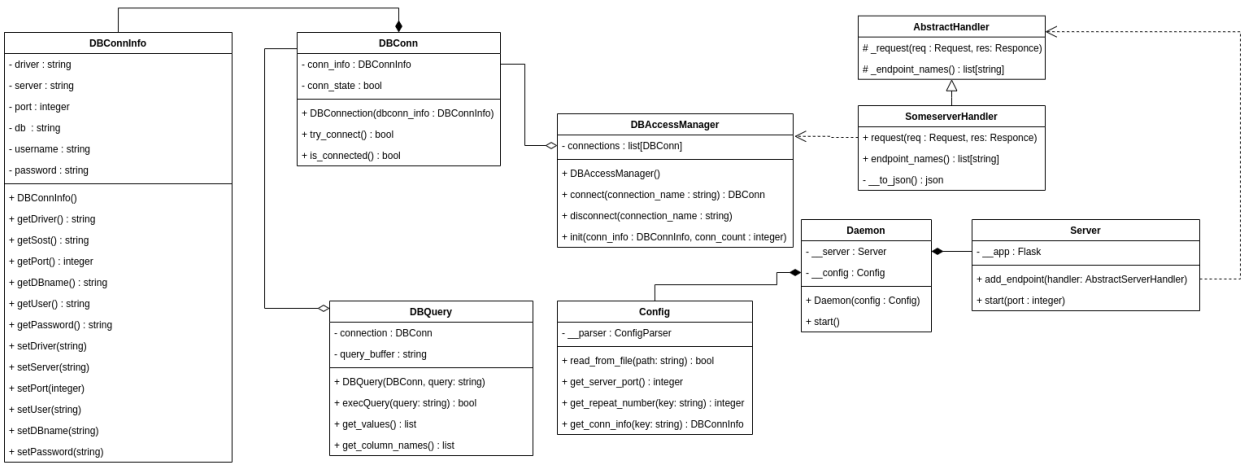

Рис. 6: Диаграмма классов сервера.

```
8
9  class AuthHandler(AbstractHandler):
10     def request(self, req, res):
11         res.content_type = "Application/Json"
12
13         try:
14             auth_data = json.loads(req.data)
15         except json.decoder.JSONDecodeError:
16             res.status_code=403
17             res.data = json.dumps({"error" : "Empty fields"})
18             return
19
20         if not auth_data.__contains__('login') or \
21             not auth_data.__contains__('password'):
22             res.status_code=403
23             res.data = json.dumps({"error" : "Expected values was not
                   received"})
24             return
25
26         state, data = self.__query_staff_data_from_db(auth_data['login'
               ], auth_data['password'])
27         if (not state):
28             res.status_code=403
29             res.data = json.dumps({"error" : "invalid fields"})
30             return
31
32         key = str(uuid.uuid4())
33         data["password"] = key
34         staff_id=data["employee"]["staff_id"]
```

```
35
36          state, schedule = self.__query_staff_schedule(staff_id)
37          if (not state):
38              res.status_code=500
39              return
40
41          data["employee"]["schedule"] = schedule
42
43          json_data = json.dumps(data)
44          valid_key_checker.register_key(key, staff_id)
45
46          res.data = json_data
47
48      def __to_json_schedule(self, rows, column_names):
49          arr = []
50          l = len(column_names)
51          for i in rows:
52              d = {}
53              d[column_names[0]] = str(i[0])
54              for j in range (2, l):
55                  d[column_names[j]] = str(i[j])
56              arr.append(d)
57          return arr
58
59      def __to_json_staff_schedule(self, rows, column_names):
60          out_json_obj = {}
61          row = rows[0]
62          for access in range(0, 2):
63              out_json_obj[column_names[access]] = str(row[access])
64
65          passport_json={}
66          for ps in range(11, len(row)):
67              passport_json[column_names[ps]] = str(row[ps])
68
69          positions_json={}
70          for pos in range(8,11):
71                  positions_json[column_names[pos]] = str(row[pos])
72
73          staff_json={}
74          for staff in range(2, 8):
75              staff_json[column_names[staff]] = str(row[staff])
```

```
76
77          staff_json["passport"] = passport_json
78          staff_json["position"] = positions_json
79          out_json_obj["employee"] = staff_json
80          return out_json_obj
81
82      def __query_staff_schedule(self, uid):
83          conn_name = str(uuid.uuid4())
84          conn = access_manager.connect(conn_name)
85
86          str_query = \
87              """SELECT * FROM schedule WHERE employee_id={}""".format(uid
                    )
88          query = DBQuery(conn, str_query)
89          if not query.execQuery():
90              return False, None
91          else:
92              schedule = query.get_values()
93          access_manager.disconnect(conn_name)
94          return True, self.__to_json_schedule(schedule, query.
                get_column_names())
95
96      def __query_staff_data_from_db(self, login, passw):
97          conn_name = str(uuid.uuid4())
98          conn = access_manager.connect(conn_name)
99          str_query = \
100             """SELECT a.login, a.access_level, s.*, p.*, ps.*
101                 FROM staff s LEFT JOIN access a ON s.staff_id=a.employee
102                     JOIN position p ON position=p.pos_id
103                     JOIN passports ps ON s.passport=ps.pass_id
104                         WHERE a.login='{}' AND a.password='{}';""".
                                format(login, passw)
105         query = DBQuery(conn, str_query)
106         if not query.execQuery():
107             return False, None
108         else:
109             result = query.get_values()
110
111         if len(result) == 0:
112             return False, None
113
```

```
114         access_manager.disconnect(conn_name)
115         return True, self.__to_json_staff_schedule(result, query.
               get_column_names())
```

3.5 Реализация клиентской части

В приложении для хранения, сериализации и десериализации данных, хранящихся в базе данных, были реализованы специальные классы, наследующиеся от класса ISerializable (листинги 11).

Листинг 11: ISerializable.h.

```
1  #pragma once
2
3  #include <QByteArray>
4  #include <QDebug>
5
6  template <typename T>
7  struct ISerializable
8  {
9      virtual ~ISerializable() noexcept = default;
10     virtual T serialize() const { qFatal("Use undefined function"); }
11     virtual bool deserialize(const T&) noexcept { qFatal("Use undefined
          function"); }
12 };
```

Пример такого класса (для таблицы access) приведен в листингах 12-13.

Листинг 12: access_data.h.

```
1      #pragma once
2
3  #include "QJsonHeaders.h"
4  #include "core/iserializable.h"
5  #include "types/staff.h"
6
7  class AccessLevel : public ISerializable<QJsonValue>
8  {
9  public:
10     enum class AccessLevelEnum
11     {
12         Admin,
13         Main,
```

```
14          Registry,
15          Vet
16      };
17
18      virtual bool deserialize(const QJsonValue &value) noexcept override;
19      virtual QJsonValue serialize() const override;
20      QString toString();
21
22  private:
23      AccessLevelEnum current;
24  };
25
26  class AccessData final : public ISerializable<QJsonObject>
27  {
28  public:
29      virtual bool deserialize(const QJsonObject &) noexcept override;
30      virtual QJsonObject serialize() const override;
31
32      QString getLogin() const;
33      uint64_t getUid() const;
34      QByteArray getPassword() const;
35      AccessLevel getLevel() const;
36      const Staff &getOwner() const;
37      void setLogin(const QString &value);
38      void setPassword(const QByteArray &value);
39      void setLevel(const AccessLevel &value);
40      void setOwner(const Staff &value);
41
42  private:
43      uint64_t uid;
44      QString login;
45      QByteArray password;
46      AccessLevel level;
47      Staff employee;
48  };
```

Листинг 13: access_data.cpp.

```
1  #include <QVariant>
2
3  #include "user_data.h"
4  #include "json_fields.h"
```

```
5
6  bool AccessLevel::deserialize(const QJsonValue & v) noexcept
7  {
8      QString value = v.toString();
9      bool is_ok = false;
10     if (value == AccessLevelType::access_vet)
11     {
12         current = AccessLevelEnum::Vet;
13         is_ok = true;
14     }
15     else if (value == AccessLevelType::access_main)
16     {
17         current = AccessLevelEnum::Main;
18         is_ok = true;
19     }
20     else if (value == AccessLevelType::access_admin)
21     {
22         current = AccessLevelEnum::Admin;
23         is_ok = true;
24     }
25     else if (value == AccessLevelType::access_registry)
26     {
27         current = AccessLevelEnum::Registry;
28         is_ok = true;
29     }
30
31     return is_ok;
32 }
33
34 QJsonValue AccessLevel::serialize() const
35 {
36     QJsonValue value;
37     switch (current)
38     {
39         case AccessLevelEnum::Vet:
40             value = QJsonValue(AccessLevelType::access_vet);
41             break;
42         case AccessLevelEnum::Main:
43             value = QJsonValue(AccessLevelType::access_main);
44             break;
45         case AccessLevelEnum::Admin:
```

```
46              value = QJsonValue(AccessLevelType::access_admin);
47              break;
48          case AccessLevelEnum::Registry:
49              value = QJsonValue(AccessLevelType::access_registry);
50              break;
51      }
52
53      return value;
54  }
55
56  QString AccessLevel::toString()
57  {
58      QString value;
59      switch (current)
60      {
61          case AccessLevelEnum::Vet:
62              value = AccessLevelType::access_vet;
63              break;
64          case AccessLevelEnum::Main:
65              value = AccessLevelType::access_main;
66              break;
67          case AccessLevelEnum::Admin:
68              value = AccessLevelType::access_admin;
69              break;
70          case AccessLevelEnum::Registry:
71              value = AccessLevelType::access_registry;
72              break;
73      }
74      return value;
75  }
76
77  bool AccessData::deserialize(const QJsonObject &document) noexcept
78  {
79      bool ok = true;
80      uid = document.value(AccessJson::field_acc_id).toVariant().
            toULongLong(&ok);
81      login = document.value(AccessJson::field_acc_login).toString();
82      password = document.value(AccessJson::field_acc_password).toVariant
            ().toByteArray();
83      ok &= employee.deserialize(document.value(AccessJson::
            field_acc_employee).toObject());
```

```
84      ok &= level.deserialize(document.value(AccessJson::
            field_acc_access_level));
85      return ok;
86  }
87
88  QJsonObject AccessData::serialize() const
89  {
90      QJsonObject root;
91      root.insert(AccessJson::field_acc_id, QJsonValue::fromVariant(
            QVariant::fromValue(uid)));
92      root.insert(AccessJson::field_acc_employee, employee.serialize());
93      root.insert(AccessJson::field_acc_login, login);
94      root.insert(AccessJson::field_acc_password, QJsonValue::fromVariant(
            password));
95      root.insert(AccessJson::field_acc_access_level, level.serialize());
96      return root;
97  }
98
99  QString AccessData::getLogin() const
100 {
101     return login;
102 }
103
104 void AccessData::setLogin(const QString &value)
105 {
106     login = value;
107 }
108
109 uint64_t AccessData::getUid() const
110 {
111     return uid;
112 }
113
114 QByteArray AccessData::getPassword() const
115 {
116     return password;
117 }
118
119 void AccessData::setPassword(const QByteArray &value)
120 {
121     password = value;
```

```
122 }
123
124 AccessLevel AccessData::getLevel() const
125 {
126     return level;
127 }
128
129 void AccessData::setLevel(const AccessLevel &value)
130 {
131     level = value;
132 }
133
134 const Staff& AccessData::getOwner() const
135 {
136     return employee;
137 }
138
139 void AccessData::setOwner(const Staff &value)
140 {
141     employee = value;
142 }
```

3.6 Примеры работы программы

Рассмотрим некоторые примеры работы программы и ее интерфейсы. На рисунке 7 представлено окно авторизации.

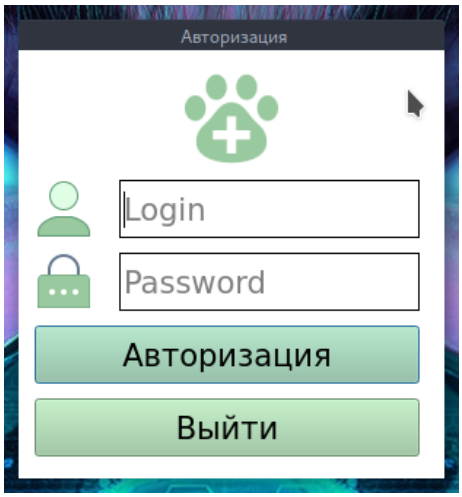

Рис. 7: Окно авторизации.

На рисунке 8 представлен профиль администратора. На рисунке 9 представлена вкладка и ифнормацией об аккаунте.

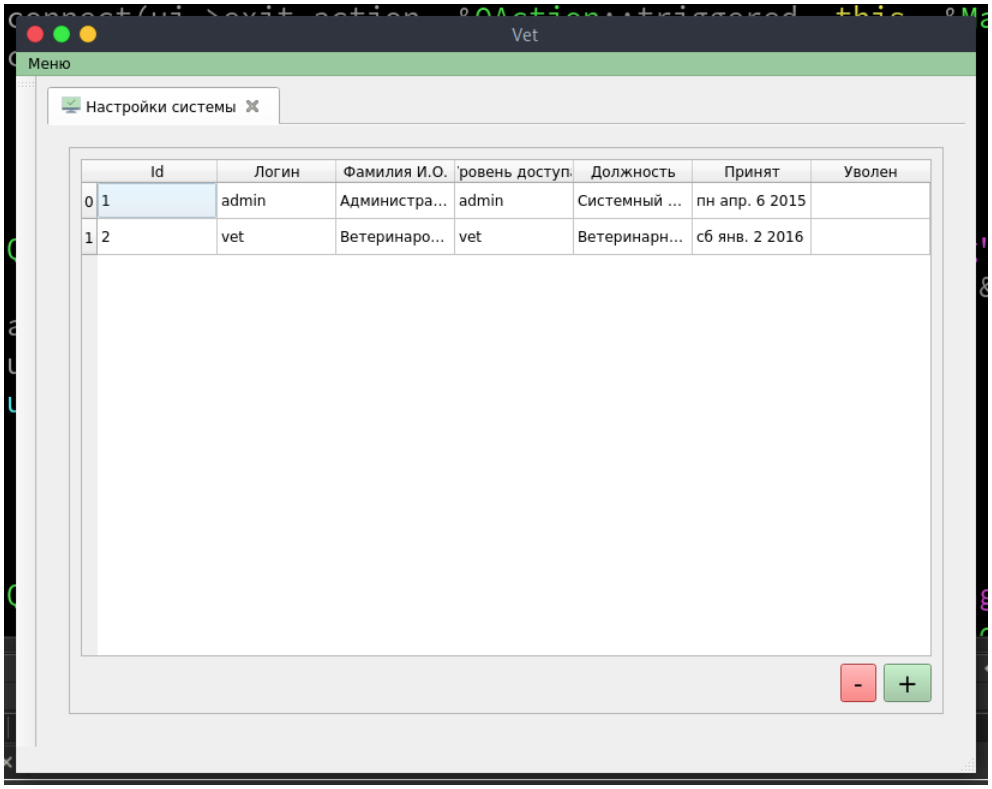

Рис. 8: Профиль «администратор».

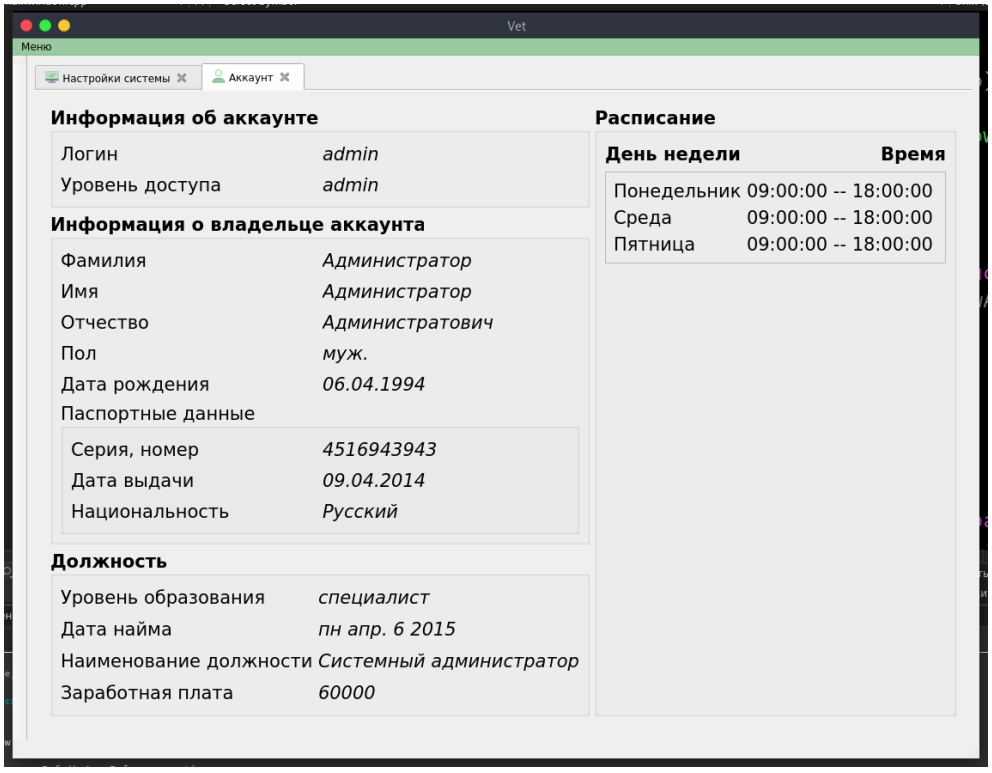

Рис. 9: Информация об аккаунте.

На рисунке 10 прелставлен профиль врача-ветеринара. На рисунке 11 прелставлена форма заполнения нового осмотра ветеринара. На рисунке 12 прелставлена форма заполнения нового назначения.

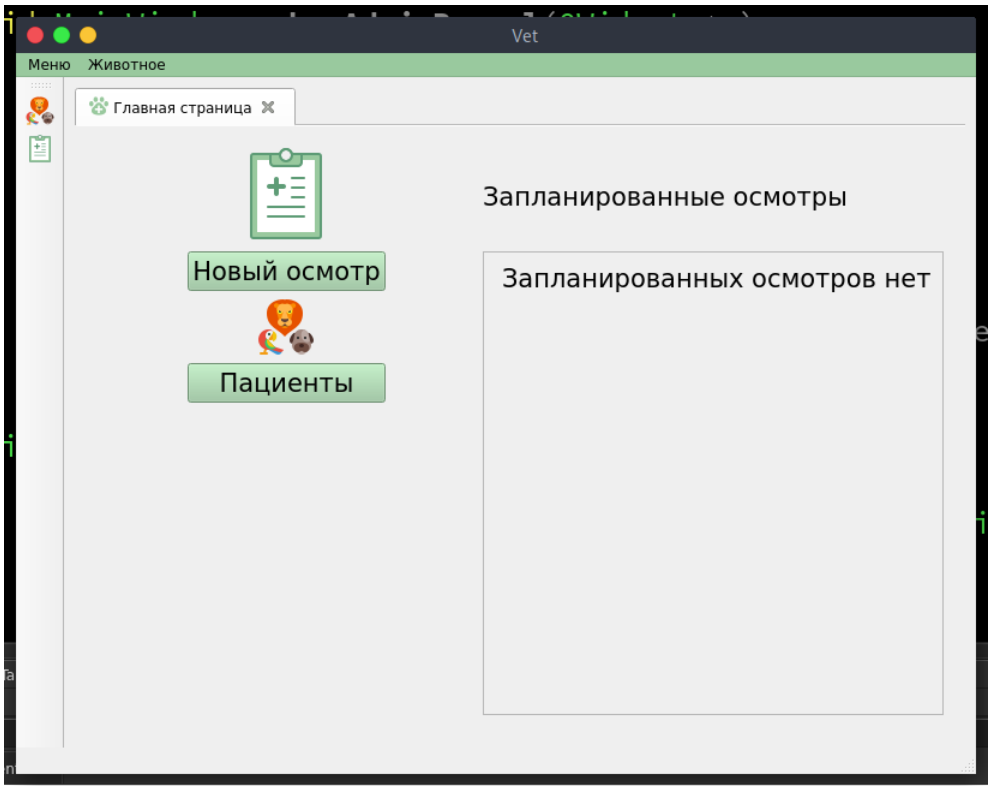

Рис. 10: Профиль «ветеринар».

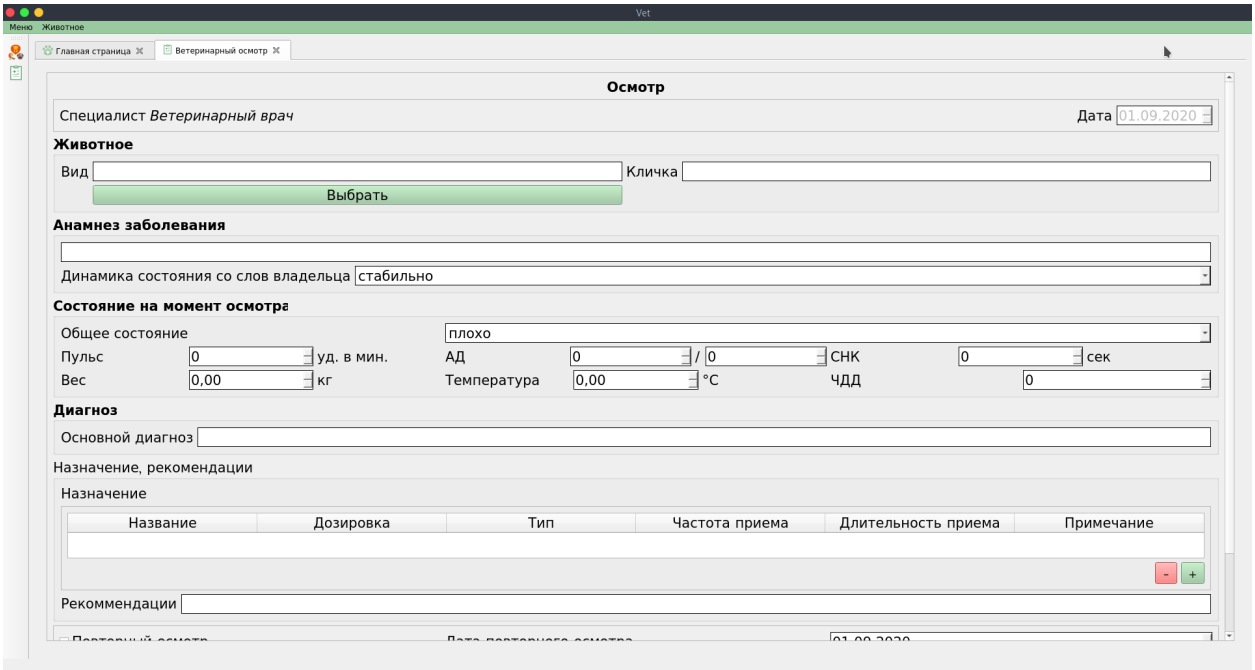

Рис. 11: Новый осмотр.

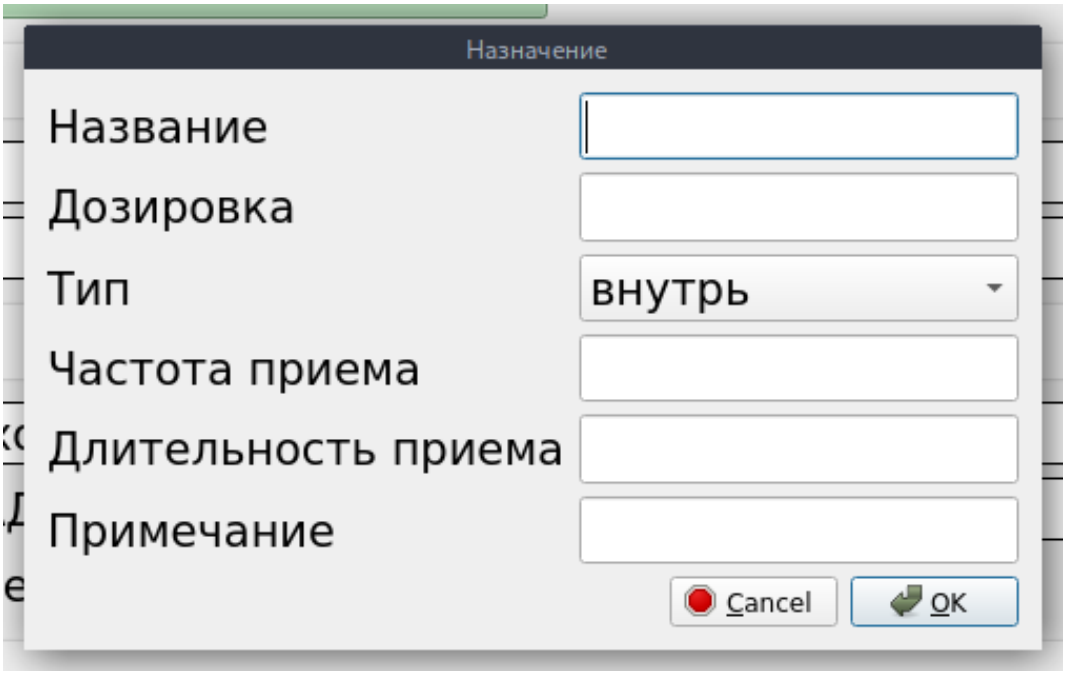

Рис. 12: Новое назначение.

Выводы из технологического раздела

Таким образом, в данном разделе были выбраны средства реализации проекта, приведены листинги и показаны примеры работы программы.

Заключение

В ходе данной работы было реализована информационная система ветеринарной клиники, предназначенная для сбора, обработки, хранения и выдачи информации и принятия управленческих решений.

В процессе работы были получены знания, касающиеся организации межсетевого взаимодействия, создания и настройки сервера, а также закреплены навыки проектирования и реализации базы данных.

Список литературы

[1] Число домашних животных в РФ выросло на 14% за три года [Электронный ресурс]. – Режим доступа: https://www.interfax.ru/russia/631927, свободный – (07.07.2020).

[2] Карпова, И. П. Базы данных. Учебное пособие. / И. П. Карпова. – М.: Московский государственный институт электроники и математики (Технический университет), 2009.

[3] Фулер, Мартин. NoSQL: новая методология разработки нереляционных баз данных. / Мартин Фулер, Прамодкумар Дж. Садаладж. Пер. с англ. – М. : ООО «И. Д. Вильямс», 2013. – 192 с.

[4] Гинзбург А. Г., Иванов А. Д., Организация ветеринарного дела, 2 изд., М., 1970.

[5] Ветеринарная энциклопедия, т. 1, М., 1968.

[6] Инфекционные заболевания [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Infection, свободный – (07.07.2020).

[7] Протозойные инфекции [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Protozoan_infection, свободный – (07.07.2020).

[8] Гельминтозы [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Helminthiasis, свободный – (07.07.2020).

[9] Вирусные заболевания [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Viral_disease, свободный – (07.07.2020).

[10] Цели, задачи и предмет деятельности учреждений ветеринарии [Электронный ресурс]. – Режим доступа: http://www.chelagro.ru/about/subordinated/goals_and_objectives_of_veterinary_in свободный – (08.07.2020).

[11] Карпова, И. П. Введение в базы данных. Учебное пособие. / И. П. Карпова. – Спб.: Питер, 2020. – 240 с.

[12] Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения. / Р. Мартин. — СПб.: Питер, 2018. — 352 с.

[13] Архитектура REST. [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/38730/, свободный – (20.08.2020).