Содержание

Введение

Сфера здравоохранения является одной из наиболее важных и социально значимых. В условиях постоянного роста объёма информации, связанной с пациентами, медицинскими записями, расписанием врачей и управлением ресурсами, использование современных систем управления базами данных (СУБД) становится необходимостью. Актуальность данной работы обусловлена необходимостью оптимизации работы больничных учреждений, что позволяет сократить время обработки данных, снизить число ошибок, улучшить качество обслуживания пациентов и повысить общий уровень медицинских услуг.

Цель исследования – разработка и анализ системы управления базами данных для автоматизации информационных процессов в больнице для улучшения работы персонала и качества обслуживания пациентов.

Для достижения поставленной цели в рамках курсовой работы были сформулированы следующие задачи:

1. Провести анализ требований к базе данных, используемой в больничных учреждениях.

2. Разработать структуру базы данных, включая основные сущности, их атрибуты и взаимосвязи.

3. Описать процедуры использования базы данных для управления приёмами, медицинскими записями и другими процессами.

Объект исследования – информационные процессы в больнице, связанные с регистрацией, хранением и обработкой медицинских данных.

Предмет исследования – система управления базами данных, предназначенная для автоматизации информационных процессов в больничных учреждениях.

1 Анализ технического задания

Разработка информационной системы для ведения больничного каталога требует тщательного анализа технического задания, поскольку данный проект должен учитывать специфические потребности медицинских учреждений. В условиях стремительного развития технологий и роста объема данных, связанных с медицинскими услугами, важно обеспечить надежность, удобство и безопасность работы системы. Выбор технологий и инструментов разработки играет ключевую роль, так как от этого зависит эффективность работы конечных пользователей. Для больниц необходимо всё самое современное и мощное, чтобы система могла поддерживать высокие объемы данных и обеспечить быстрый доступ к информации.

При анализе выбора языка программирования, необходимо сопоставить такие популярные языки как C# и C++. Каждый из них имеет свои преимущества и недостатки. C++ предлагает разработчикам высокий уровень управления памятью и эффективные механизмы работы с низкоуровневыми ресурсами, что может быть полезно в приложениях, требующих высокой производительности. Эта гибкость позволяет создавать высоконагруженные системы, однако требует от программистов глубокой подготовки и опыта в управлении памятью, что в медицине может оказаться критичным.

С другой стороны, язык C# обладает более высокой абстракцией и включает в себя многофункциональные библиотеки и фреймворки, которые значительно упрощают процесс разработки. В отличие от C++, C# наиболее оптимально подходит для разработки приложений, связанных с пользовательскими интерфейсами и интеграцией с базами данных. C# интегрирован с платформой .NET, которая имеет обширные возможности для работы с веб-технологиями и создания многоуровневых приложений, что для больниц является важным аспектом.

Что касается среды разработки, Visual Studio является одним из наиболее мощных инструментов для разработки на C#. Она предлагает широкий функционал, включая отладку, интеграцию с системами контроля версий и

множество сторонних плагинов, что упрощает процесс создания программного обеспечения. Кроме того, Visual Studio поддерживает разработку приложений для Windows, что особенно актуально для многих медицинских учреждений, использующих системы Windows в своей инфраструктуре.

В конечном счете, выбор языка программирования стоит делать в пользу C# и среды разработки Visual Studio. Это решение можно обосновать не только удобством разработки и поддержки, но и тем, что C# предлагает оптимальные инструменты для создания масштабируемых и надежных систем, идеально подходящих для нужд больниц. Простота работы с базами данных и возможность быстрой интеграции новых функций делают C# идеальным выбором для создания информационной системы, отвечающей самым высоким требованиям современного здравоохранения.

1.1 Обзор аналогов

В современных российских больницах используются различные информационные системы и базы данных, обеспечивающие эффективное управление медицинскими услугами и данными пациентов. Такие системы, как правило, разрабатываются с целью оптимизации процессов, связанных с ведением медицинских записей, управлением закупками, расчетом материалов и улучшением взаимодействия между врачами и пациентами. Обзор аналогов информационной системы для ведения больничного каталога позволяет выделить несколько значимых решений, которые уже используются в медицинских учреждениях РФ.

Одним из наиболее распространенных решений является система "1С:Медицинская организация". Это инструмент, который включает в себя не только возможность вести учет пациентов, но и позволяет автоматизировать многие процессы, такие как планирование рабочего времени медперсонала, учет медицинских услуг и их стоимости, создание отчетов и интеграция с

другими системами. За счет своих модулей, система "1С" предоставляет широкие возможности для кастомизации под конкретные нужды больницы.

Еще одним известным аналогом является система "Мед.Эксперт", которая предназначена для работы в здравоохранении и предлагает функционал для ведения электронной медицинской документации, управления потоком пациентов и хирургических процедур. Это решение активно используется в ряде крупных медицинских учреждений и позволяет объединять данные из различных источников, обеспечивая целостный подход к ведению медкарты.

Система "Гемотест" также заслуживает внимания. Она применяется в многих лабораторных центрах и поликлиниках, обеспечивая автоматизацию всех этапов диагностики и ведения отчетности. Эта система позволяет быстро и эффективно обрабатывать результаты анализов и хранить историю пациентов, что улучшает качество оказания медицинских услуг.

Несмотря на развитие существующих решений, в большинстве систем наблюдается необходимость в улучшении интеграционных процессов и доступности данных для различных пользователей. Это открывает возможности для внедрения новых информационных систем, таких как рассматриваемая нами информационная система для ведения больничного каталога, которая сможет более эффективно учитывать специфические требования и нужды российских больниц.

Таким образом, обзор аналогов показывает, что в Российской Федерации существует множество информационных систем для медицинских учреждений, однако каждая из них имеет свои ограничения. Это создает потребность в разработке более гибких и адаптивных решений, которые были бы способны учитывать все аспекты работы больницы, обеспечивать высокую степень доступности и безопасности данных пациентов, а также интегрироваться с уже существующими системами. Создание такой информационной системы для ведения больничного каталога может

значительно повысить эффективность работы медицинских учреждений и улучшить качество обслуживания пациентов.

Программа для ведения больничного каталога, разработанная в рамках данной курсовой работы, превосходит аналоги благодаря интуитивно понятному интерфейсу, гибким настройкам и интеграции с существующими системами. Мы обеспечиваем регулярные обновления и круглосуточную поддержку, а также высокий уровень безопасности данных пациентов. Уникальные инструменты для аналитики и отчетности помогают улучшить управление учреждения и обеспечивают высокое качество обслуживания. Все эти преимущества делают нашу программу более эффективной и удобной в работе.

2 Проектирование базы данных

2.1 Концептуальная модель

Программа "Больница" представляет собой интегрированную информационную систему для управления всеми аспектами больничной деятельности, включая взаимодействие с пациентами, медицинским персоналом, данными о лечении, а также административными и финансовыми процессами. Система построена как централизованная база данных с пользователями, работающими через интерфейс приложения. Это могут быть сотрудники больницы разных уровней: администраторы, врачи, бухгалтеры, фармацевты и другие специалисты. Система организована так, чтобы предоставить им доступ только к необходимым данным, гарантируя защиту личной и медицинской информации.

В ядре системы лежат тесно взаимосвязанные сущности. Таблица Patients является центральной для программы, так как все процессы обслуживания пациента зависят от информации, связанной с ним. Каждый пациент проходит регистрацию, при этом фиксируется его личная информация, возраст, контактные данные и данные об имеющемся страховом полисе. После регистрации пациент становится доступным для назначения встреч (Appointments), что осуществляют либо администраторы больницы, либо врачи через пользовательский интерфейс.

Назначенные встречи хранятся в таблице Appointments, где фиксируется время визита, причина обращения, а также статус встречи (например, запланирована, завершена или отменена). В момент встречи врач, связанный через таблицу Doctors, вносит данные в медицинскую карту пациента (MedicalRecords). Медицинская карта содержит важнейшую информацию о диагнозах, плане лечения, предписаниях и комментариях специалиста. Она облегчает отслеживание состояния пациента и помогает врачам принимать решения при дальнейших визитах.

Если пациенту требуется использование медикаментов, врач оформляет рецепт, данные о котором фиксируются в таблице Prescriptions. Эта таблица связана с таблицей Medications, где содержится информация о соответствующих лекарственных препаратах, их дозировке и побочных эффектах. Рецепт содержит чёткие указания по применению медикамента, что помогает снизить риски ошибок и недоразумений в процессе лечения. Выписанные лекарства могут быть проверены и выданы через аптеку больницы.

Финансовый аспект программы охватывается таблицей Payments, которая отслеживает все платёжные операции, связанные с пациентами. После предоставления услуг (например, консультаций, лечения или госпитализации), администраторы могут фиксировать поступление платежей. В таблицу заносятся такие данные, как сумма платежа, метод оплаты и дата. Эти сведения полезны для выполнения бухгалтерских операций и составления отчётов.

Для пациентов, нуждающихся в госпитализации, назначаются помещения, описанные в таблице Rooms. Администраторы и врачи могут проверять наличие свободных мест, распределять комнаты в зависимости от их типа и статуса занятости. Это упрощает систематизацию размещения и позволяет избегать конфликтов при распределении ресурсов.

Ключевую роль в системе играет персонал, информация о котором хранится в таблице Staff. Административный и медицинский персонал фиксируется в системе, включая их личные данные, должности и роли. Это необходимо для управления кадровым составом и распределения обязанностей. К примеру, администраторы отвечают за регистрацию пациентов, оформление документов и обработку платежей, в то время как врачи и медсёстры сосредотачиваются на их медицинском обслуживании.

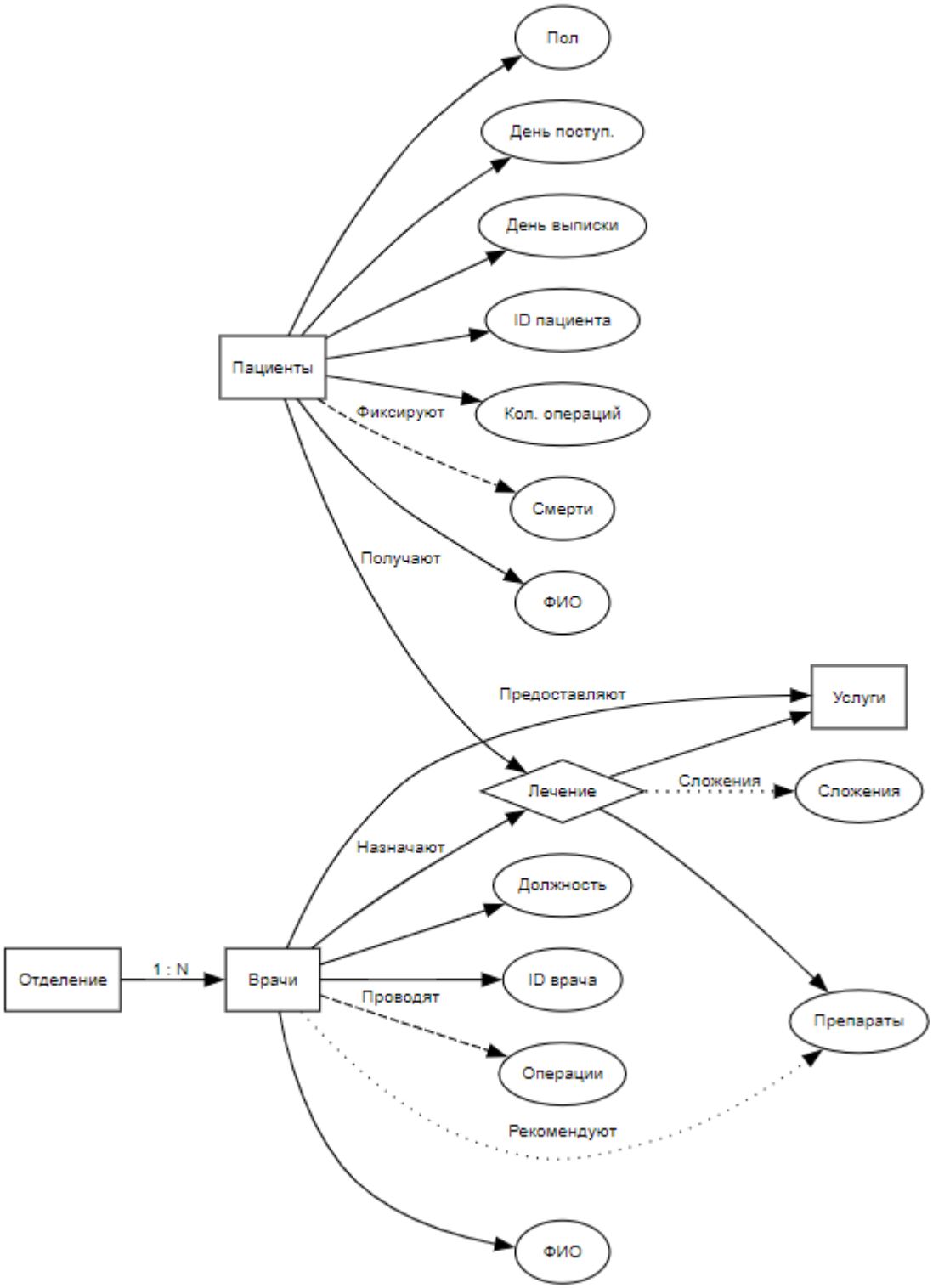

Рисунок 1 – ER-диаграмма концептуальной модели

Программа создаёт интегрированную среду, где каждая сущность выполняет свою роль в общей структуре оказания услуг. Пациент поступает в качестве центрального элемента системы, и вокруг него разворачивается обработка данных: регистрация, приём, диагностика, назначение лекарств или

процедур, обработка финансовых операций и обеспечение палатами. Всё это взаимодействует через связные таблицы, поддерживая целостность данных и упрощая выполнение задач сотрудниками.

С функциональной точки зрения программа предусматривает мультиролевой доступ. Каждый тип пользователя, будь то врач, администратор либо бухгалтер, видит только те разделы и функции приложения, которые необходимы ему для работы. Кроме того, программа предусматривает дополнительные возможности, такие как построение отчётов, статистику посещений, уровни загруженности персонала и ресурсов больницы. С точки зрения безопасности система требует авторизации и предполагает использование современных методов шифрования для хранения чувствительных данных пациентов.

Реализация концепции способствует автоматизации всех основных процессов в больнице, минимизации человеческих ошибок, сокращению времени на обработку данных и повышению общего уровня удовлетворённости пациентов качеством обслуживания.

2.2 Атрибуты, сущности и целлостность данных

Описание сущностей:

1. Patients (Пациенты):

- Описание: Хранит данные пациентов, включая личные и страховые данные.

- Ключевые атрибуты:

  - patient_id: Уникальный идентификатор.

  - first_name, last_name, gender, date_of_birth: Личные данные пациента.

  - insurance_info: Страховая информация.

2. Doctors (Доктора):

- Описание: Информация о докторах, их специализациях и контактах.

- Ключевые атрибуты:

- doctor_id: Уникальный идентификатор.
- specialization: Специализация доктора.
- phone_number, email: Контакты.

3. Appointments (Назначения):
- Описание: Организует встречи между пациентами и докторами.
- Ключевые атрибуты:
  - appointment_id: Уникальный идентификатор.
  - patient_id, doctor_id: Связь с пациентом и врачом.
  - appointment_date, reason_for_visit, status: Данные по встрече.

4. MedicalRecords (Медицинские записи):
- Описание: История лечения пациентов.
- Ключевые атрибуты:
  - record_id: Уникальный идентификатор записи.
  - patient_id, doctor_id: Связь с пациентом и врачом.
  - diagnosis, treatment_plan, visit_date, notes: Медицинские данные.

5. Prescriptions (Рецепты):
- Описание: Хранение информации о выписанных лекарствах.
- Ключевые атрибуты:
  - prescription_id: Уникальный идентификатор.
  - record_id: Ссылка на медицинскую запись.
  - medication_id: Ссылка на таблицу медикаментов.
  - dosage_instructions: Данные по дозировке.

6. Medications (Медикаменты):
- Описание: Каталог медикаментов, которые есть в больнице.
- Ключевые атрибуты:
  - medication_id: Уникальный идентификатор.

- dosage, side_effects: Информация о лекарствах.

- photo: Для хранения изображения медикамента.

7. Rooms (Комнаты):

- Описание: Управление комнатами для пациентов.

- Ключевые атрибуты:

- room_id: Уникальный идентификатор.

- room_number: Номер комнаты.

- type: Тип комнаты (например, VIP, стандарт).

- availability_status: Статус доступности комнаты (свободна, занята).

8. Staff (Персонал):

- Описание: Информация о других сотрудниках больницы (регистраторы, лаборанты и т.д.).

- Ключевые атрибуты:

- staff_id: Уникальный идентификатор.

- position: Должность (например, администратор, уборщик).

9. Payments (Платежи):

- Описание: Управление финансовыми операциями пациентов.

- Ключевые атрибуты:

- payment_id: Уникальный идентификатор.

- patient_id: Связь с пациентом.

- amount: Сумма.

- payment_method: Метод оплаты (наличные, карта).

10. Departments (Отделы):

- Описание: Организация больничных отделов.

- Ключевые атрибуты:

- department_id: Уникальный идентификатор.

- name: Название (например, терапия, хирургия).

Связи между сущностями:

1. Пациенты — Назначения:
   - Один пациент может иметь несколько назначений.
   - Связь: patient_id в Appointments.

2. Доктора — Назначения:
- Один доктор может проводить несколько встреч.
   - Связь: doctor_id в Appointments.

3. Назначения — Медицинские записи:
   - Каждая медицинская запись связана с одной встречей.
   - Связь: appointment_id.

4. Медицинские записи — Рецепты:
   - Каждая медицинская запись может иметь несколько рецептов.
   - Связь: record_id в Prescriptions.

5. Рецепты — Медикаменты:
   - Один рецепт привязан к одному медикаменту.
   - Связь: medication_id в Prescriptions.

6. Пациенты — Платежи:
   - Один пациент может совершать несколько платежей.
   - Связь: patient_id в Payments.

7. Доктора — Отделы:
   - Один доктор принадлежит к одному отделу.

- Связь: department_id.

8. Пациенты — Комнаты:

- Пациента можно назначить в комнату.

- Связь: patient_id и room_id (можно добавить таблицу размещения).

Сущность	Атрибут	Описание
Отделение	Наименование	Название отделения больницы
		Уникальный идентификатор отделения
	Адрес	Адрес, где расположено отделение,
	Телефон	Контактный номер для отделения
Врачи	ID врача	Уникальный идентификатор врача
	ФИО	Полное имя врача
	Должиость	Роль, занимаемая врачом (хирург, терапевт и т,д,)
	Отделение	Ссылка на отделение, в котором работакуг врач
Пациенты	ID пациента	Уникальный идентификатор пациен-
	ФИО	Полное имя пациента
	пол	Пол пациента (мужской/женский)
	День поступления	Дата поступления пациента
	День выписки	Дата выписки пациента
	Количество операций	Общее число операций у пациента
Лечение	П) операции	Уникальный идентификатор операции
	Тип лечения	Услуга или операция, назначенные, при лечении
	Препараты	Нтзвание медикаментов, выдаваемых пациенту
Услуги	Цена услуги	Стоимость услуги
	Название услу-	Тип предоставляемой услуги (например, рентген, УЗИ и т.п.)
Препараты	Наименование	Название препарата
	Производитель	Компания - протводитель лекарств
Смерти	Дата	Дата наступления смерти пациента
	Причина	Причина смерти
	ФИО врача	Врач, который зафиксировал смерть

Таблица 2 - Таблица атрибутов сущностей базы данных

2.3 Построение физической модели

Физическое проектирование - это процедура создания описания конкретной реализации БД с описанием структуры хранения данных, методов доступа к данным. В данном случае реализация была осуществлена в MS SQL.

Скрипт создания базы данных:

SQL-код для создания базы данных с таблицами:

```
CREATE DATABASE Hospital;
USE Hospital;
CREATE TABLE Patients (
    patient_id INTEGER PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    date_of_birth DATE,
    gender VARCHAR(10),
    address VARCHAR(255),
    phone_number VARCHAR(15),
    email VARCHAR(100),
    insurance_info VARCHAR(255)
);
CREATE TABLE Appointments (
    appointment_id INTEGER PRIMARY KEY AUTO_INCREMENT,
    patient_id INTEGER,
    doctor_id INTEGER,
    appointment_date DATETIME,
    reason_for_visit VARCHAR(255),
    status VARCHAR(50),
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),
    FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id)
);
CREATE TABLE Doctors (
```

```
    doctor_id INTEGER PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    specialization VARCHAR(100),
    phone_number VARCHAR(15),
    email VARCHAR(100)
);
CREATE TABLE MedicalRecords (
    record_id INTEGER PRIMARY KEY AUTO_INCREMENT,
    patient_id INTEGER,
    doctor_id INTEGER,
    visit_date DATETIME,
    diagnosis VARCHAR(255),
    treatment_plan VARCHAR(255),
    notes TEXT,
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),
    FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id)
);
CREATE TABLE Prescriptions (
    prescription_id INTEGER PRIMARY KEY AUTO_INCREMENT,
    record_id INTEGER,
    medication_id INTEGER,
    dosage_instructions VARCHAR(255),
    FOREIGN KEY (record_id) REFERENCES MedicalRecords(record_id),
    FOREIGN KEY (medication_id) REFERENCES
Medications(medication_id)
);
CREATE TABLE Medications (
    medication_id INTEGER PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
```

```
    dosage VARCHAR(50),
    side_effects VARCHAR(255),
    photo BLOB
);
CREATE TABLE Payments (
    payment_id INTEGER PRIMARY KEY AUTO_INCREMENT,
    patient_id INTEGER,
    amount REAL,
    payment_date DATETIME,
    payment_method VARCHAR(50),
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id)
);
CREATE TABLE Rooms (
    room_id INTEGER PRIMARY KEY AUTO_INCREMENT,
    room_number INTEGER,
    type VARCHAR(100),
    availability_status VARCHAR(50)
);
CREATE TABLE Staff (
    staff_id INTEGER PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    position VARCHAR(100)
);
CREATE TABLE Departments (
    department_id INTEGER PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100)
);
ALTER TABLE Doctors ADD department_id INTEGER;
```

ALTER TABLE Doctors ADD FOREIGN KEY (department_id) REFERENCES Departments(department_id);

Appointments		CREATE TABLE [Appointments]([appointment_id]
appointment_id	INTEGER	"appointment_id" INTEGER
patient_id	INTEGER	"patient_id" INTEGER NOT NULL
doctor_id	INTEGER	"doctor_id" INTEGER NOT NULL
appointment_date	DATETEXT	"appointment_date" DATETEXT NOT NULL
reason_for_visit	VARCHAR	"reason_for_visit" VARCHAR
status	VARCHAR	"status" VARCHAR

Рисунок 3 – Таблица Appointments

Departments		CREATE TABLE Departments (department_id I
department_id	INTEGER	"department_id" INTEGER
name	VARCHAR	"name" VARCHAR NOT NULL

Рисунок 4 – Таблица Departments

MedicalRecords		CREATE TABLE [MedicalRecords]([record_id] INTEGER PRIMARY KEY AUTOINCREMENT,
record_id	INTEGER	"record_id" INTEGER
patient_id	INTEGER	"patient_id" INTEGER NOT NULL
doctor_id	INTEGER	"doctor_id" INTEGER NOT NULL
visit_date	DATETEXT	"visit_date" DATETEXT NOT NULL
diagnosis	VARCHAR	"diagnosis" VARCHAR
treatment_plan	VARCHAR	"treatment_plan" VARCHAR
notes	VARCHAR	"notes" VARCHAR

Рисунок 5 – Таблица MedicalRecords

Medications		CREATE TABLE Medications (medication_id INTEGER
medication_id	INTEGER	"medication_id" INTEGER
name	VARCHAR	"name" VARCHAR NOT NULL
dosage	VARCHAR	"dosage" VARCHAR NOT NULL
side_effects	VARCHAR	"side_effects" VARCHAR
photo	BLOB	"photo" BLOB

Рисунок 6 – Таблица Medications

Patients		CREATE TABLE [Patients]([patient_id] INTEGER PRIMARY
patient_id	INTEGER	"patient_id" INTEGER
first_name	VARCHAR	"first_name" VARCHAR NOT NULL
last_name	VARCHAR	"last_name" VARCHAR NOT NULL
date_of_birth	DATETEXT	"date_of_birth" DATETEXT NOT NULL
gender	VARCHAR	"gender" VARCHAR NOT NULL
address	VARCHAR	"address" VARCHAR
phone_number	VARCHAR	"phone_number" VARCHAR
email	VARCHAR	"email" VARCHAR
insurance_info	VARCHAR	"insurance_info" VARCHAR

Рисунок 7 – Таблица Patients

Payments		CREATE TABLE [Payments]([payment_id]
payment_id	INTEGER	"payment_id" INTEGER
patient_id	INTEGER	"patient_id" INTEGER NOT NULL
amount	REAL	"amount" REAL NOT NULL
payment_date	DATETEXT	"payment_date" DATETEXT NOT NULL
payment_method	VARCHAR	"payment_method" VARCHAR

Рисунок 8 – Таблица Payments

Prescriptions		CREATE TABLE Prescriptions (prescription_id
prescription_id	INTEGER	"prescription_id" INTEGER
record_id	INTEGER	"record_id" INTEGER NOT NULL
medication_id	INTEGER	"medication_id" INTEGER NOT NULL
dosage_instructions	VARCHAR	"dosage_instructions" VARCHAR NOT NULL

Рисунок 9 – Таблица Prescriptions

Rooms		CREATE TABLE Rooms (room_id INTEGER PRIMARY KEY AUTOINCREMENT, room_number
room_id	INTEGER	"room_id" INTEGER
room_number	VARCHAR	"room_number" VARCHAR NOT NULL
type	VARCHAR	"type" VARCHAR NOT NULL CHECK("type" IN ('Одиночная', 'Общая'))
availability_status	VARCHAR	"availability_status" VARCHAR CHECK("availability_status" IN ('Доступна', 'Занята'))

Рисунок 10 – Таблица Rooms

Staff		CREATE TABLE Staff (staff_id INTEGER PRIMARY KEY AUTOINCREMENT,
staff_id	INTEGER	"staff_id" INTEGER
first_name	VARCHAR	"first_name" VARCHAR NOT NULL
last_name	VARCHAR	"last_name" VARCHAR NOT NULL
position	VARCHAR	"position" VARCHAR NOT NULL
department	VARCHAR	"department" VARCHAR

Рисунок 11 – Таблица Staff

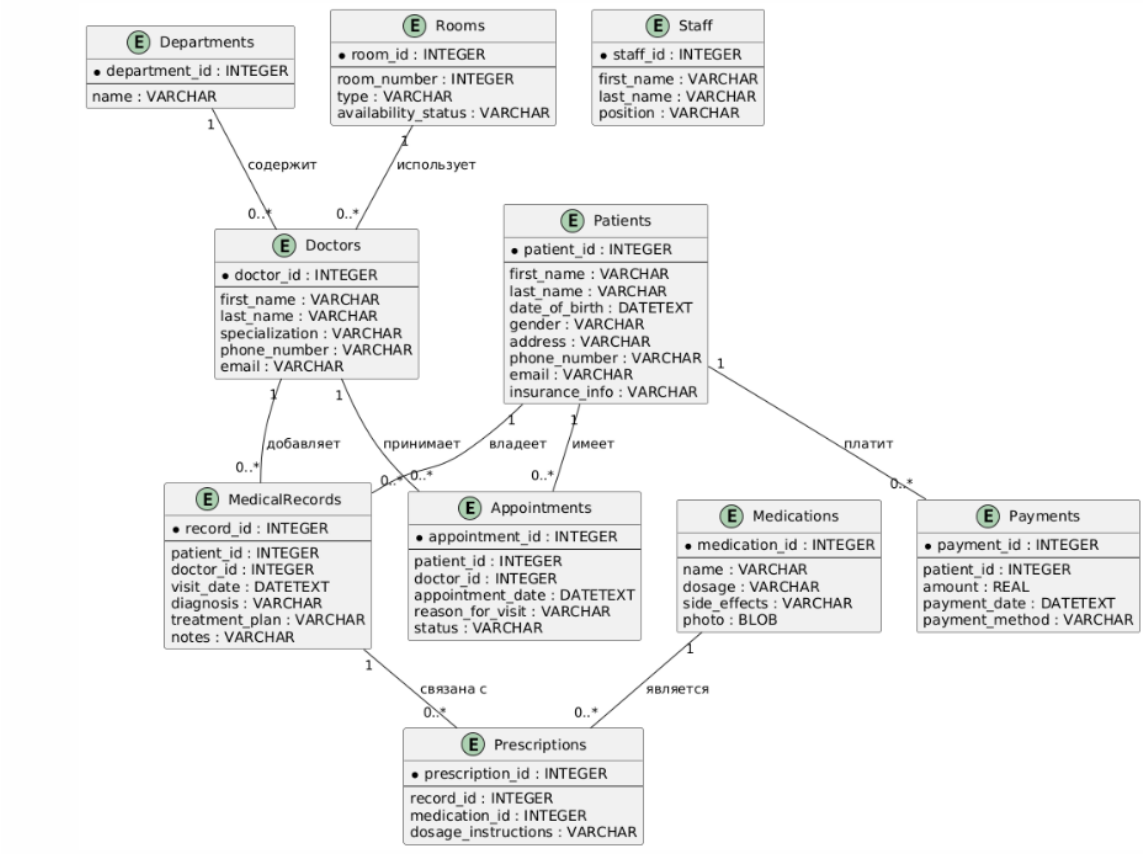

Рисунок 12 – UML-диаграмма базы данных

Запросы SELECT с использованием JOIN:

1. Вывести список всех пациентов вместе с их врачами

```
SELECT
    Patients.first_name AS PatientFirstName,
    Patients.last_name AS PatientLastName,
    Doctors.first_name AS DoctorFirstName,
    Doctors.last_name AS DoctorLastName,
    Doctors.specialization AS DoctorSpecialization
FROM
    Patients
JOIN
    Appointments ON Patients.patient_id = Appointments.patient_id
JOIN
    Doctors ON Appointments.doctor_id = Doctors.doctor_id;
```

2. Найти список всех назначений для конкретного пациента

```
SELECT
   Appointments.appointment_date AS AppointmentDate,
   Appointments.reason_for_visit AS Reason,

   Doctors.first_name AS DoctorFirstName,
   Doctors.last_name AS DoctorLastName
FROM
   Appointments
JOIN
   Doctors ON Appointments.doctor_id = Doctors.doctor_id
WHERE
   Appointments.patient_id = 1; -- Замените `1` на ID нужного пациента
```

3. Получить медицинские записи пациента вместе с выписанными рецептами

```
SELECT
   Patients.first_name AS PatientFirstName,
   Patients.last_name AS PatientLastName,
   MedicalRecords.diagnosis AS Diagnosis,
   MedicalRecords.treatment_plan AS TreatmentPlan,
   Medications.name AS MedicationName,
   Prescriptions.dosage_instructions AS DosageInstructions
FROM
   Patients
JOIN
   MedicalRecords ON Patients.patient_id = MedicalRecords.patient_id
JOIN
   Prescriptions ON MedicalRecords.record_id = Prescriptions.record_id
JOIN
   Medications ON Prescriptions.medication_id = Medications.medication_id
```

```
WHERE
    Patients.patient_id = 1; -- Замените `1` на ID нужного пациента
```

3 Разработка приложения по работе с СУБД

3.1 Назначение,основные функции, структура приложения

Для разработки программного обеспечения в рамках выполнения данной задачи применяются различные инструменты и технологии, которые обеспечивают эффективное создание как клиентской, так и серверной части с базой данных. В качестве системы управления базами данных была выбрана MySQL. Данная СУБД широко используется благодаря своей производительности, устойчивости и возможности обработки реляционных данных. Основным преимуществом MySQL является её открытость и наличие большого количества встроенных функций, необходимых для работы с большими объемами информации. Для локального развертывания базы данных и администрирования использовалась утилита MySQL Workbench, которая позволяет визуализировать создаваемую структуру данных, осуществлять проектирование таблиц и выполнение SQL-запросов. В случае необходимости веб-ориентированного управления базой данных может быть применен PhpMyAdmin.

Среда разработки Visual Studio 2022 выбрана для написания основного кода приложения. Данный программный комплекс является универсальной интегрированной средой разработки, которая поддерживает множество языков программирования, включая C#. Использование C# в сочетании с технологией ASP.NET Core позволяет создавать как клиентские, так и серверные части программного обеспечения, обеспечивая взаимодействие с базой данных через встроенные библиотеки и пакеты, например, такие как MySQL Data Connector или Entity Framework. Удобство Visual Studio также заключается в интеграции с системами контроля версий и автоматизацией процесса сборки приложений.

Backend-часть приложения, отвечающая за обработку запросов и взаимодействие с базой данных, реализуется с помощью стандартных технологий, принятых в промышленной разработке.

Для обеспечения доступности данных используются операции CRUD (создание, чтение, обновление и удаление записей), которые реализуются посредством SQL-запросов. Кодом на языке C# обеспечивается подключение к базе данных MySQL, обработка SQL-запросов и возвращение результата клиенту. Это достигается за счет использования стандартных библиотек .NET Framework или .NET Core.

Если в рамках реализации требуется создание графического пользовательского интерфейса, возможно использование технологий WPF (Windows Presentation Foundation). WPF позволяет проектировать современный, интуитивно понятный фронтенд для настольных приложений с поддержкой сложных графических интерфейсов. Для создания веб-интерфейса могут применяться такие инструменты, как Blazor или React.js, обеспечивающие взаимодействие с серверной частью посредством API.

На этапе тестирования программы предусматривается использование локального сервера. Для этого могут быть задействованы программные среды, такие как XAMPP или WAMP, которые обеспечивают локальный запуск базы данных MySQL и обработку запросов. Это дает возможность отлаживать приложение в изолированной среде без необходимости разворачивать его на реальном сервере.

Основные функции приложения включают в себя добавление, обновление и удаление записей, связанных с пациентами и врачами, с использованием команд базы данных SQLite. При загрузке формы приложения происходит инициализация базы данных, после чего загружаются данные о записях назначений, а также списки пациентов и врачей, которые отображаются в соответствующих комбобоксах. Это позволяет пользователю выбирать существующих пациентов и врачей для создания или редактирования записей о назначениях. При добавлении новой записи приложение проверяет, заполнены ли все необходимые поля, и валидирует введенные данные с помощью регулярных выражений. Это обеспечивает корректность введенной информации, такой как имена пациентов и врачей,

дата визита и причина обращения. Если все данные введены правильно, запись добавляется в базу данных, и пользователю отображается сообщение об успешном добавлении. Функция обновления записи работает аналогично, но вместо добавления новой записи происходит изменение существующей. Пользователь может выбрать запись из таблицы, после чего данные загружаются в поля ввода, где их можно редактировать. После внесения изменений и подтверждения, обновленная информация сохраняется в базе данных. Удаление записей также реализовано через интерфейс, где пользователь может выбрать запись и удалить ее из базы данных. Приложение уведомляет пользователя о результате операции, будь то успешное удаление или необходимость выбора записи для удаления. Таким образом, приложение обеспечивает полный цикл работы с медицинскими записями, начиная от их создания и заканчивая редактированием и удалением, при этом акцентируя внимание на валидации данных и удобстве взаимодействия с пользователем.

Приложение для управления медицинскими записями в больнице имеет четкую и логичную структуру, которая обеспечивает эффективное взаимодействие между пользователем и базой данных. В центре приложения находится форма, которая служит интерфейсом для выполнения различных операций, таких как добавление, обновление и удаление записей о пациентах и врачах.

Основные компоненты приложения включают в себя классы, отвечающие за различные аспекты функциональности. Класс `Appointments` управляет записями о приёмах, обеспечивая загрузку данных из базы, отображение их в интерфейсе и обработку действий пользователя, таких как добавление новых записей или удаление существующих. Этот класс использует экземпляр `ClassProvide`, который реализует паттерн Singleton для управления соединением с базой данных SQLite. Это позволяет избежать создания нескольких соединений и обеспечивает централизованный доступ к базе данных. Класс `ClassProvide` отвечает за установление и управление соединением с базой данных.

Он инициализирует соединение при создании экземпляра и предоставляет методы для получения этого соединения, а также для его закрытия. Это обеспечивает надежное управление ресурсами и предотвращает утечки памяти. В дополнение к классу `Appointments`, приложение включает другие классы, такие как `Doctors`, `Departments`, и `Payments`, каждый из которых имеет свою собственную логику и интерфейс для управления соответствующими данными.

Эти классы взаимодействуют с базой данных аналогичным образом, используя SQL-запросы для выполнения операций CRUD (создание, чтение, обновление, удаление). Интерфейс пользователя состоит из различных элементов управления, таких как текстовые поля, комбобоксы и таблицы данных, которые позволяют пользователю вводить информацию и получать обратную связь. Валидация данных осуществляется с помощью регулярных выражений, что гарантирует, что введенные данные соответствуют ожидаемым форматам, прежде чем они будут отправлены в базу данных. Таким образом, структура приложения организована вокруг классов, которые инкапсулируют логику работы с данными и интерфейсом, обеспечивая четкое разделение ответственности.

Это позволяет легко расширять функциональность приложения и поддерживать его в будущем.

Для визуализации структуры приложения можно использовать UML-диаграммы объектов и компонентов. UML-диаграмма объектов будет представлять классы и их взаимосвязи, показывая, как они взаимодействуют друг с другом, в то время как UML-диаграмма компонентов отразит архитектуру приложения, включая основные модули и их интерфейсы. Эти диаграммы помогут лучше понять архитектурные решения, принятые при разработке приложения, и упростят процесс его дальнейшего развития.

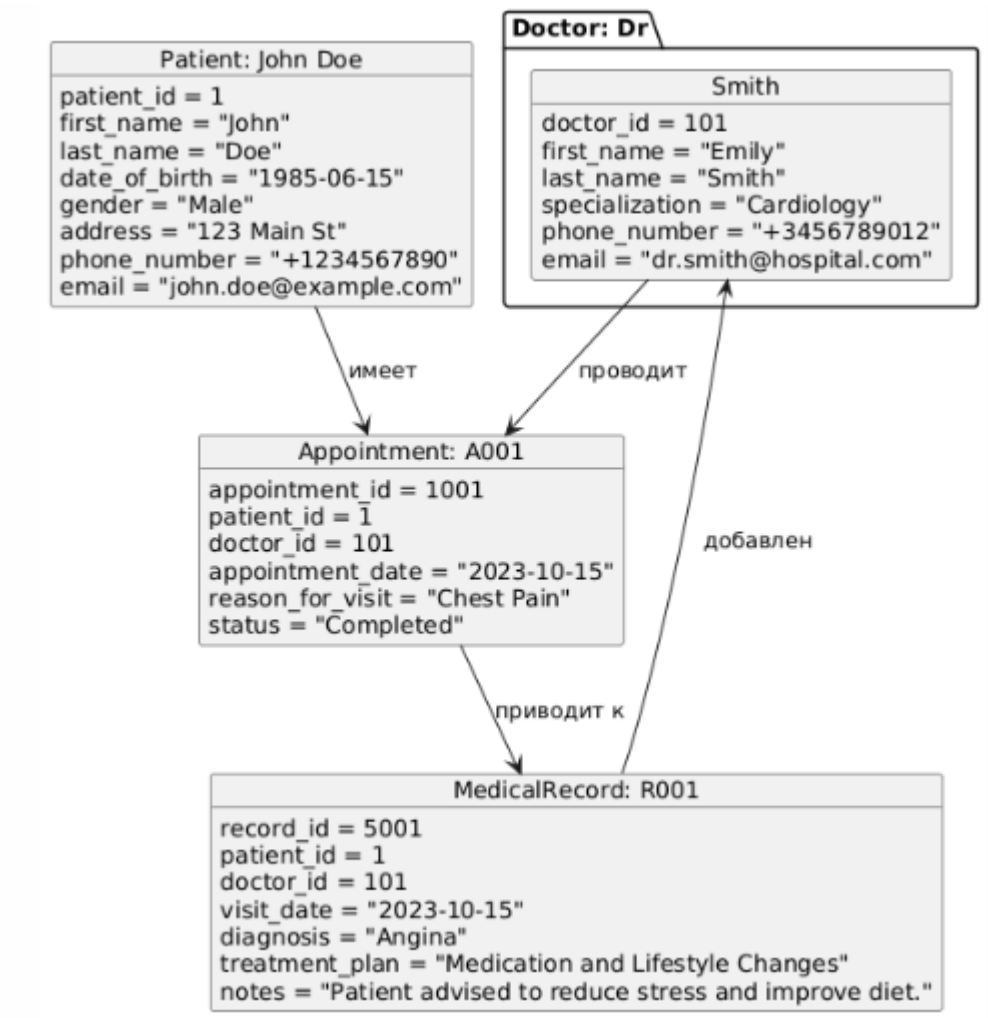

Рисунок 13 – UML-диаграмма объектов

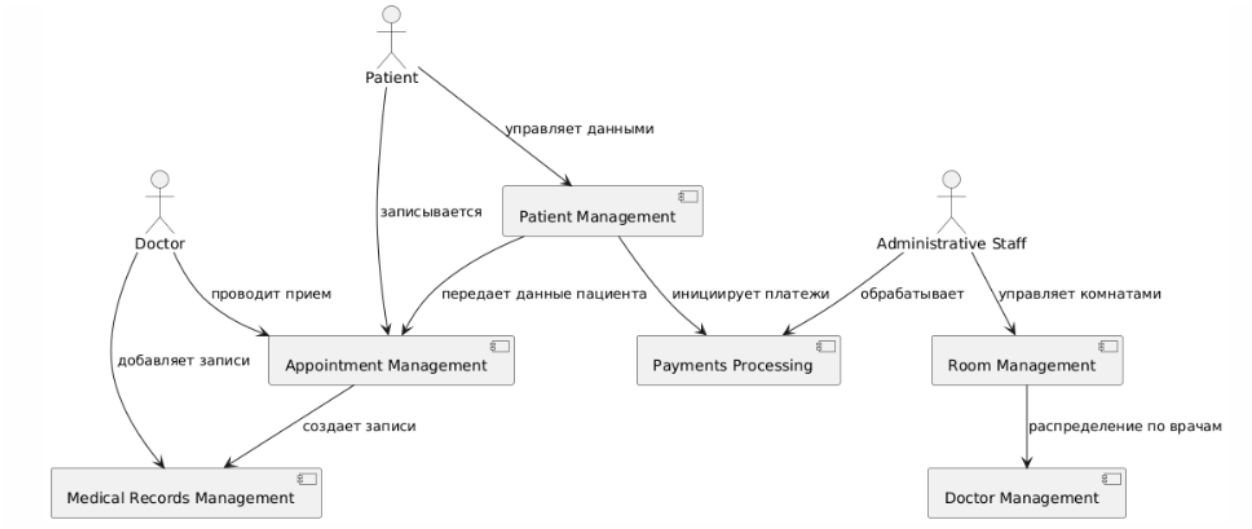

Рисунок 14 – UML-диаграмма компонентов

3.2 Создание SQL-запросов

SQL-запросы являются основным механизмом взаимодействия с реляционными базами данных. Они позволяют эффективно управлять данными, обеспечивая их хранение, извлечение, модификацию и удаление.

Результатом запроса является представление — виртуальная таблица, представляющая собой поименованный запрос, который будет подставлен как подзапрос при использовании представления.

В отличие от обычных таблиц реляционных баз данных, представление не является самостоятельной частью набора данных, хранящегося в базе.

Создание таблицы для пациентов

```
CREATE TABLE Patients (
    patient_id INTEGER PRIMARY KEY,
    first_name VARCHAR,
    last_name VARCHAR,
    date_of_birth DATETEXT,
    gender VARCHAR,
    address VARCHAR,
    phone_number VARCHAR,
    email VARCHAR,
    insurance_info VARCHAR
);
```

Создание таблицы для докторов

```
CREATE TABLE Doctors (
    doctor_id INTEGER PRIMARY KEY,
    first_name VARCHAR,
    last_name VARCHAR,
    specialization VARCHAR,
    phone_number VARCHAR,
    email VARCHAR
);
```

```
Создание таблицы для отделов
CREATE TABLE Departments (
    department_id INTEGER PRIMARY KEY,
    name VARCHAR
);
Создание таблицы для записей приема
CREATE TABLE Appointments (
    appointment_id INTEGER PRIMARY KEY,
    patient_id INTEGER,
    doctor_id INTEGER,
    appointment_date DATETEXT,
    reason_for_visit VARCHAR,
    status VARCHAR,
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),
    FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id)
);
Создание таблицы для медицинских записей
CREATE TABLE MedicalRecords (
    record_id INTEGER PRIMARY KEY,
    patient_id INTEGER,
    doctor_id INTEGER,
    visit_date DATETEXT,
    diagnosis VARCHAR,
    treatment_plan VARCHAR,
    notes VARCHAR,
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),
    FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id)
);
Создание таблицы для лекарств
CREATE TABLE Medications (
```

```
    medication_id INTEGER PRIMARY KEY,
    name VARCHAR,
    dosage VARCHAR,
    side_effects VARCHAR,
    photo BLOB
);
Создание таблицы для платежей
CREATE TABLE Payments (
    payment_id INTEGER PRIMARY KEY,
    patient_id INTEGER,
    amount REAL,
    payment_date DATETEXT,
    payment_method VARCHAR,
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id)
);
Создание таблицы для рецептов
CREATE TABLE Prescriptions (
    prescription_id INTEGER PRIMARY KEY,
    record_id INTEGER,
    medication_id INTEGER,
    dosage_instructions VARCHAR,
    FOREIGN KEY (record_id) REFERENCES MedicalRecords(record_id),
    FOREIGN KEY (medication_id) REFERENCES Medications(medication_id)
);
Создание таблицы для кабинетов
CREATE TABLE Rooms (
    room_id INTEGER PRIMARY KEY,
    room_number INTEGER,
    type VARCHAR,
    availability_status VARCHAR
```

);

Эти SQL запросы позволяют создать готовую структуру базы данных для системы медицинских записей, включающую все необходимые таблицы и связи между ними.

3.3 Руководство пользователя

Системные требования:

Операционная система Windows 10. Приложение не зависит от типа применяемого процессора.

Для того, чтобы приложение, описанное в курсовой работе, функционировало, необходимо подключить базу данных MS SQL Server, а так же подключить базу данных hospital.db.

Разработанный интерфейс прост в использовании. Для того, чтобы вводить или редактировать данные в приложении, нужно выбрать таблицу из списка таблиц, нажать кнопку «Показать», выполнить необходимые изменения и нажать кнопку «Редактировать». Чтобы выполнить специальные запросы и увидеть их результат, нужно зайти в Меню-> «выбираем интересующий запрос».

3.4 Руководство программиста

Приложение создано на языке программирования C# в среде Visual Studio с использованием базы данных SQLite. Оно включает в себя несколько форм, таких как форма авторизации, рабочая форма администратора, рабочая форма менеджера по аренде и форма архива. Для работы приложения необходимы следующие компоненты: Visual Studio 2022 или новее, .NET Framework 4.7.2 или новее, а также СУБД SQLite. Установка приложения состоит из следующих шагов: откройте Visual Studio, запустите проект и установите пакет SQLite через пункт управления NuGet пакетами.

Проект состоит из следующих файлов:

1) Appointments.cs

2) ClassProvide.cs

3) Departments.cs

4) Doctors.cs

5) Form1.cs

6) Patients.cs

7) Payments.cs

Описание форм:

1) Appointments.cs

При загрузке формы в методе Appointments_Load осуществляется подключение к базе данных SQLite через объект класса ClassProvide, извлечение списка записей о назначениях и их вывод в компонент dataGridView1. Каждая запись включает в себя такие данные, как идентификатор назначения, фамилии пациента и доктора, дата назначения, причина визита и статус. Также происходит заполнение двух выпадающих списков comboBox1 и comboBox2 фамилиями пациентов и докторов соответственно, с выборкой уникальных значений из базы данных. Третий выпадающий список comboBox3 заполняется фиксированными значениями, представляющими возможные статусы назначения: "Запланированный", "Отмененный", "Завершенный".

Метод button2_Click обрабатывает событие нажатия кнопки для удаления записи. Если выделена строка в dataGridView1, то идентификатор записи извлекается, и соответствующая запись удаляется из базы данных с помощью SQL-команды DELETE, после чего отображается сообщение об успешном удалении. Если запись для удаления не выбрана, отображается сообщение об ошибке.

Метод button1_Click (начало которого представлено) обрабатывает ввод пользователя, собирая данные из элемента управления формы, таких как выбранные пациент, доктор, дата назначения, причина визита и статус, для последующего добавления записи в базу данных.

2) ClassProvide.cs

Данный код реализует класс ClassProvide, который представляет собой синглтон для управления соединением с базой данных SQLite. При первом запросе экземпляра класса через метод GetInstance создаётся объект ClassProvide, который устанавливает соединение с базой данных, находящейся по указанному пути hospital.db. Метод GetConnection возвращает текущий объект соединения SQLiteConnection, предоставляя доступ к нему из других частей программы. Метод CloseConnection закрывает соединение с базой данных, если оно было открыто. Таким образом, этот класс отвечает за создание и управление единственным экземпляром подключения к базе данных.

3) Departments.cs

В классе Departments создается подключение к базе данных SQLite при загрузке формы. В методе Departments_Load используется экземпляр класса ClassProvide для получения соединения с базой данных. Далее выполняется SQL-запрос, который выбирает идентификатор и название департамента из таблицы Departments базы данных. Полученные данные считываются через SQLiteDataReader, и для каждого департамента добавляется новая строка в DataGridView, отображающий список департаментов.

4) Doctors.cs

При загрузке формы код создает экземпляр подключения к базе данных и отправляет запрос на получение информации о врачах, включая их ID, имя,

фамилию, специализацию, номер телефона и адрес электронной почты. Полученные данные отображаются в таблице dataGridView1.

Удаление записи осуществляется через событие кнопки, проверяя, выбрана ли строка в таблице. Если строка выбрана, она удаляется из базы данных по ID врача, после чего таблица обновляется, и выводится уведомление об успешном удалении. Если строка не выбрана, отображается сообщение об ошибке.

Добавление записи происходит через форму ввода, где заполняются поля имени, фамилии, специализации, номера телефона и электронной почты. Код проверяет корректность введенных данных с помощью регулярных выражений.

5) Form1.cs

При запуске формы в методе Form1_Load происходит инициализация подключения к базе данных и загрузка данных для отображения в различных элементах интерфейса. Сначала выполняется запрос к базе данных для извлечения информации о медицинских записях, включая идентификатор записи, фамилии пациентов и врачей, дату визита, диагноз, план лечения и заметки. Эти данные затем добавляются в таблицу (dataGridView1).

Кроме того, выполняются два дополнительных запроса: один для получения уникальных фамилий пациентов, а второй для фамилий врачей. Эти данные заполняют выпадающие списки (comboBox1 и comboBox2), что позволяет пользователю выбрать пациента и врача из существующих записей.

В форме также предусмотрены обработчики событий для различных элементов меню. Каждый элемент меню вызывает соответствующую форму для работы с различными сущностями, такими как отделения, врачи, пациенты, персонал, медикаменты, палаты, платежи, рецепты и записи на приём. Например, при клике на пункт меню для работы с врачами открывается форма для работы с врачами (Doctors), и так далее для других сущностей. Все эти формы открываются в модальном режиме с использованием ShowDialog.

Каждое действие в форме связано с отображением данных или навигацией по разным частям системы через модальные окна, которые позволяют пользователю управлять информацией в базе данных.

В методе dataGridView1_CellDoubleClick при двойном клике на ячейку таблицы проверяется, выбрана ли строка. Если строка выбрана, создается экземпляр подключения к базе данных и выполняется SQL-запрос для получения данных о медицинской записи, включая имя пациента, фамилию доктора, дату визита, диагноз, план лечения и заметки. Эти данные затем заполняют соответствующие поля интерфейса: comboBox1, comboBox2, dateTimePicker1, textBox1, textBox2, textBox3.

Метод button3_Click отвечает за удаление записи из базы данных. Если строка в таблице выбрана, то создается подключение к базе данных, и выполняется SQL-запрос для удаления записи по ее ID. После успешного удаления таблица обновляется, и отображается уведомление об успешном удалении. Если строка не выбрана, показывается сообщение об ошибке.

Часть метода button1_Click подготавливает данные для вставки новой записи. Она считывает значения из элементов формы (comboBox1, comboBox2, dateTimePicker1, textBox1, textBox2, textBox3) и сохраняет их в соответствующих переменных. Остальная часть метода, судя по всему, будет включать проверку правильности введенных данных и добавление новой записи в базу данных, что аналогично описанному ранее методу добавления новой записи.

6) Patients.cs

Метод Patients_Load выполняется при загрузке формы и выполняет два основных действия:

1. Извлекает из базы данных информацию о пациентах (идентификатор, имя, фамилия, дата рождения, пол, адрес, номер телефона, электронная почта, информация о страховке) и добавляет её в таблицу dataGridView1.

2. Загружает уникальные значения пола пациентов и добавляет их в выпадающий список (comboBox1) для фильтрации или выбора.

Метод button3_Click выполняется при нажатии соответствующей кнопки и отвечает за удаление выделенной строки (записи пациента) из таблицы и базы данных:

1. Проверяет, выбрана ли строка в dataGridView1. Если строка не выбрана, отображается сообщение об ошибке.

2. Если строка выбрана, извлекает идентификатор выделенного пациента.

3. Удаляет запись с соответствующим идентификатором из базы данных, выполняя SQL-запрос DELETE.

4. Обновляет таблицу и выводит уведомление об успешном удалении.

Метод button1_Click (начало кода) отвечает за добавление данных нового пациента:

1. Считывает поля пользовательского ввода, такие как имя, фамилия, дата рождения, пол, адрес, телефон, email и информацию о страховке из текстовых полей (textBox1, textBox6, и т. д.), выпадающих списков (comboBox1) и элемента выбора даты (dateTimePicker1).

2. Логика для добавления записи в базу данных, вероятно, находится дальше, но её нет в предоставленной части кода.

7) Payments.cs

1. Загрузка данных о пациентах:

При загрузке формы (Patients_Load) создается подключение к базе данных и выполняется запрос для получения информации о пациентах (имя, фамилия, дата рождения, пол, адрес, номер телефона, email и страховка).

Эти данные добавляются в таблицу dataGridView1 на форме.

Дополнительно, выполняется запрос для получения уникальных значений пола пациентов (мужской, женский) и они добавляются в выпадающий список comboBox1.

2. Удаление записи о пациенте:

Когда нажимается кнопка удаления (button3_Click), проверяется, была ли выбрана строка в dataGridView1. Если да, выполняется удаление записи о пациенте из базы данных по идентификатору пациента.

После удаления строки таблица обновляется, и появляется сообщение "Запись удалена из базы!"

3. Добавление записи о платеже:

Когда нажимается кнопка добавления платежа (button1_Click), выполняется проверка введенных данных:

Все поля должны быть заполнены.

Проводится валидация данных с использованием регулярных выражений для проверки правильности ввода:

Имя пациента (поле pc) должно состоять только из букв и пробела.

Сумма платежа (поле am) должна содержать только цифры и пробел.

Дата платежа (поле pd) должна быть корректно отформатирована.

Способ оплаты (поле pm) должен быть введен корректно.

Если все данные правильные, создается SQL-запрос для добавления новой записи о платеже в таблицу Payments с использованием данных о пациенте, суммы, даты и способа оплаты.

4. Дополнительная валидация и добавление платежа через вторую кнопку:

Для второго сценария, связанного с добавлением платежа (button2_Click), также выполняется проверка:

Все обязательные поля должны быть заполнены.

Для каждого поля выполняется проверка на правильность формата данных с использованием регулярных выражений, аналогично предыдущему случаю.

Если данные введены правильно, будет выполнен SQL-запрос для добавления записи о платеже в базу данных.

4 Тестирование программы

Тестирование включает в себя проверку всех функций, таких как добавление, обновление и удаление записей, а также валидацию пользовательского ввода.

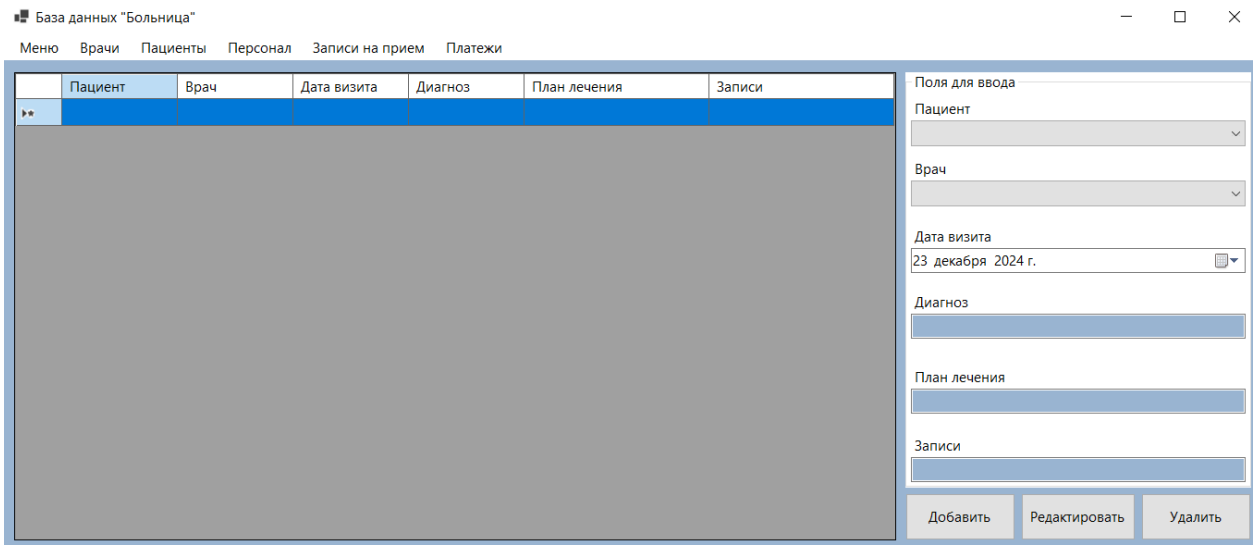

Рисунок 15 - Главное окно

На рисунке 15 представлено главное окно приложения. Оно служит центральной точкой взаимодействия пользователя с системой. В этом окне отображаются основные функции, такие как добавление, редактирование и удаление записей о пациентах и врачах.

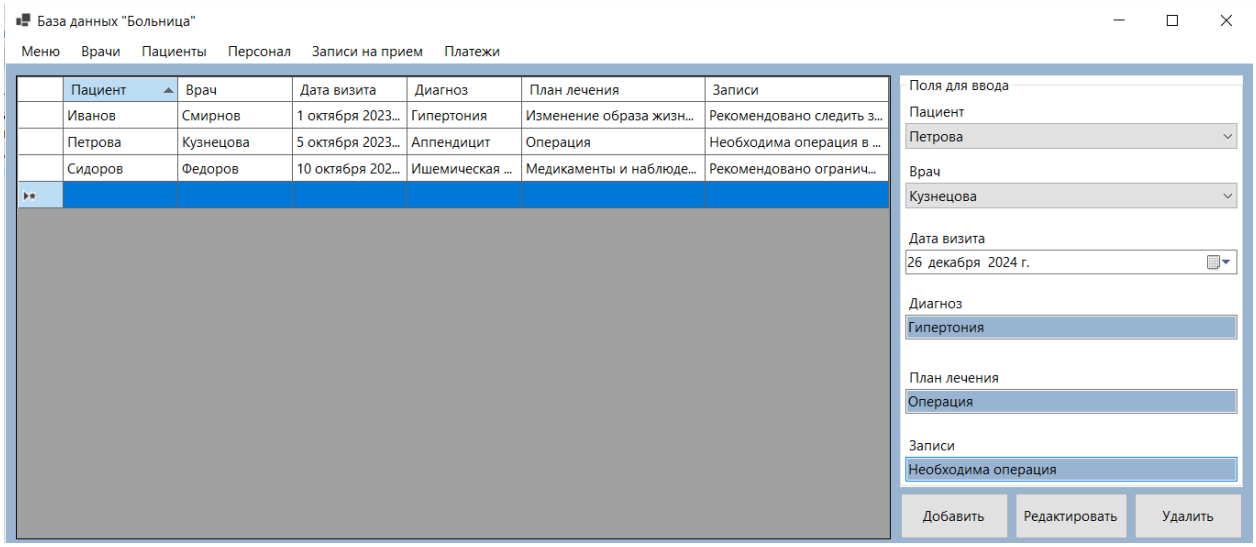

Рисунок 16 - Главное окно с заполненными данными

Главное окно также включает в себя элементы управления, такие как комбобоксы для выбора пациентов и врачей, текстовые поля для ввода

информации и таблицу для отображения записей. Это окно вызывается при запуске приложения и обеспечивает пользователю доступ ко всем необходимым функциям.

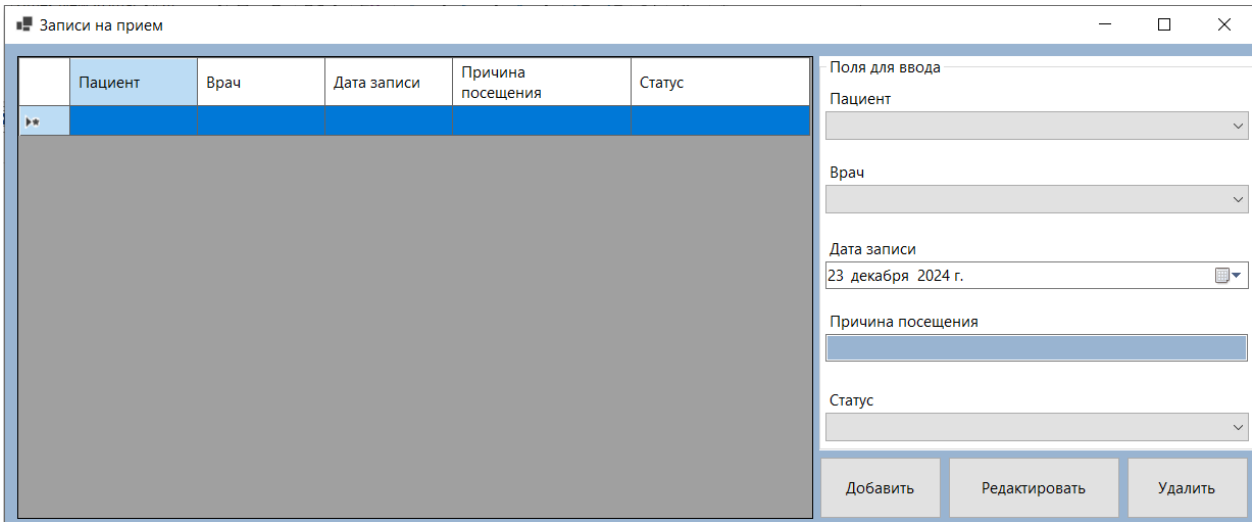

Рисунок 17 - Окно добавления записи

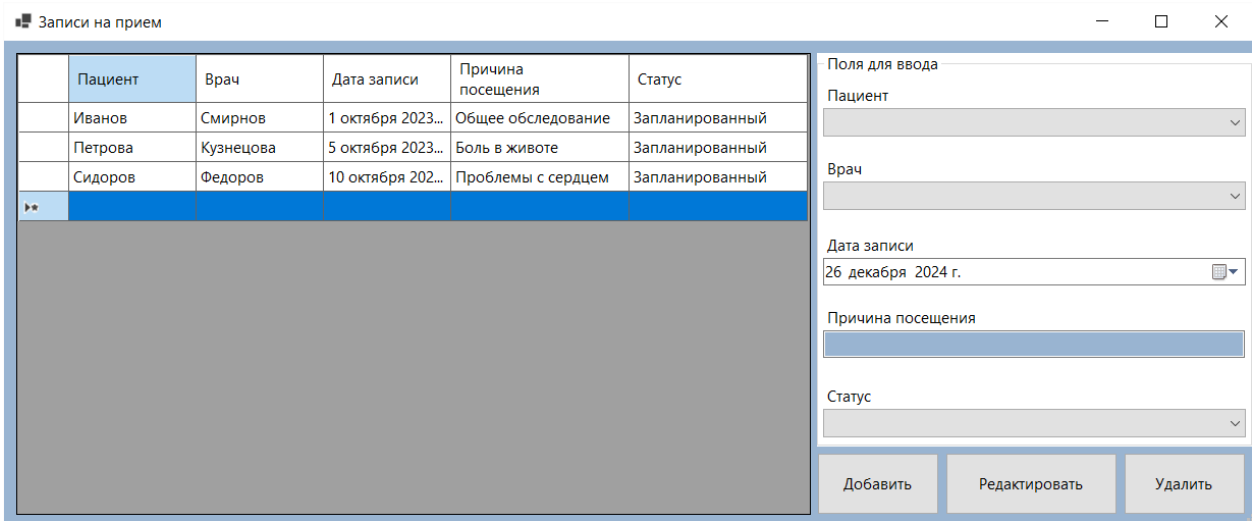

Рисунок 18 - Окно добавления записи с заполненными данными.

Это окно позволяет пользователю вводить данные о пациенте, враче, дате приема и причине визита. Оно вызывается при нажатии кнопки "Добавить" на главном окне и обеспечивает удобный интерфейс для ввода информации.

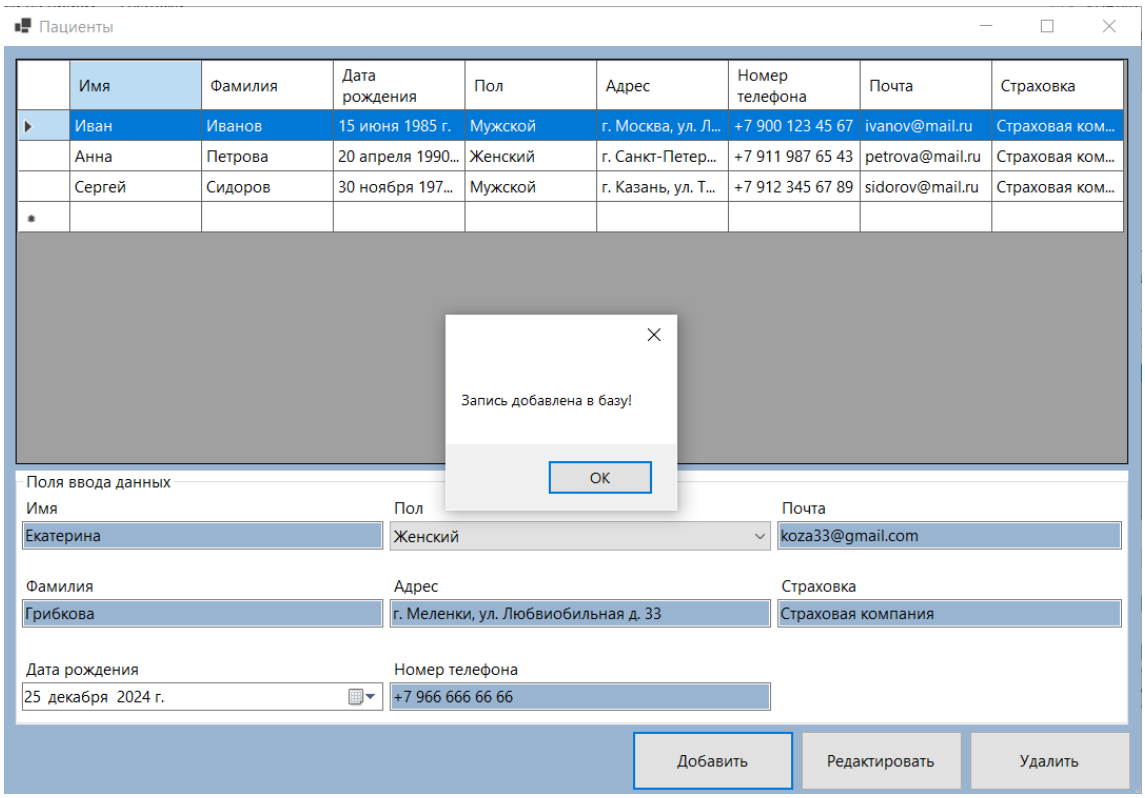

Рисунок 19 – Окно добавления пациентов

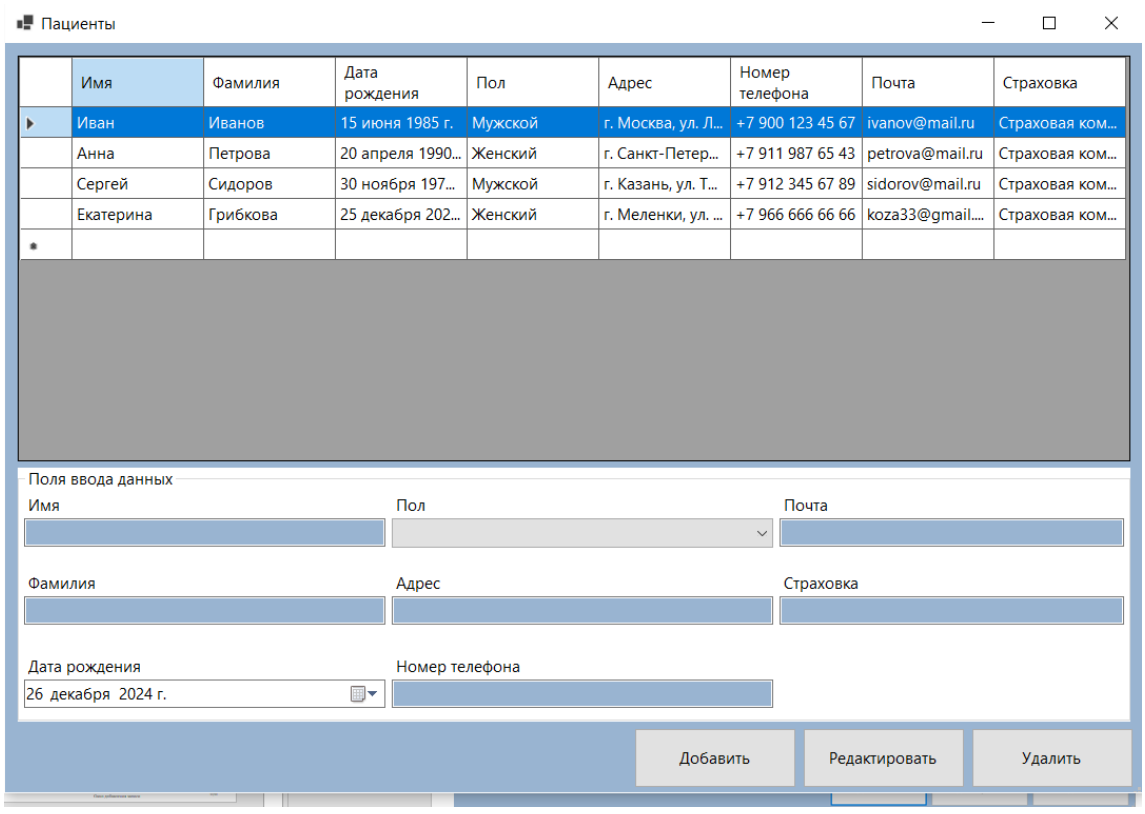

Рисунок 19 – Окно добавления пациентов с новым пациентом

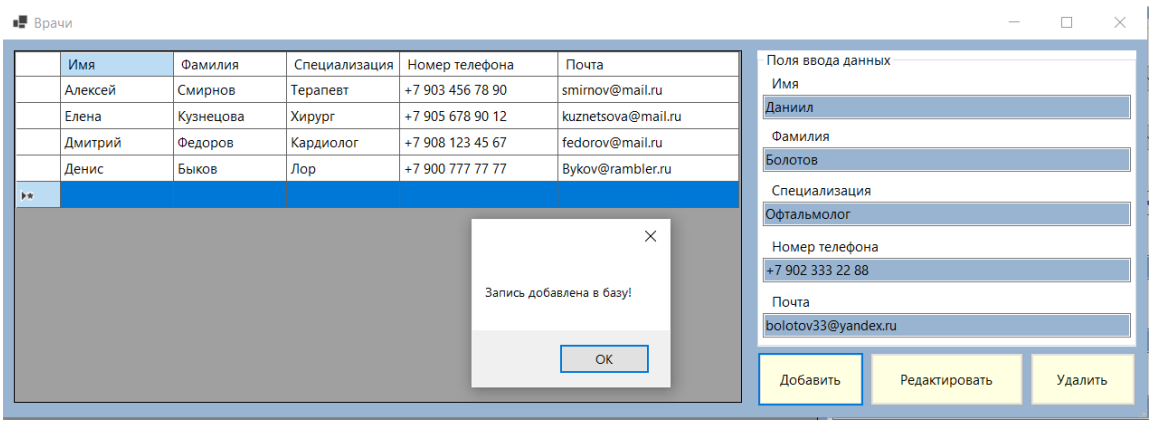

Рисунок 19 – Окно добавления врачей

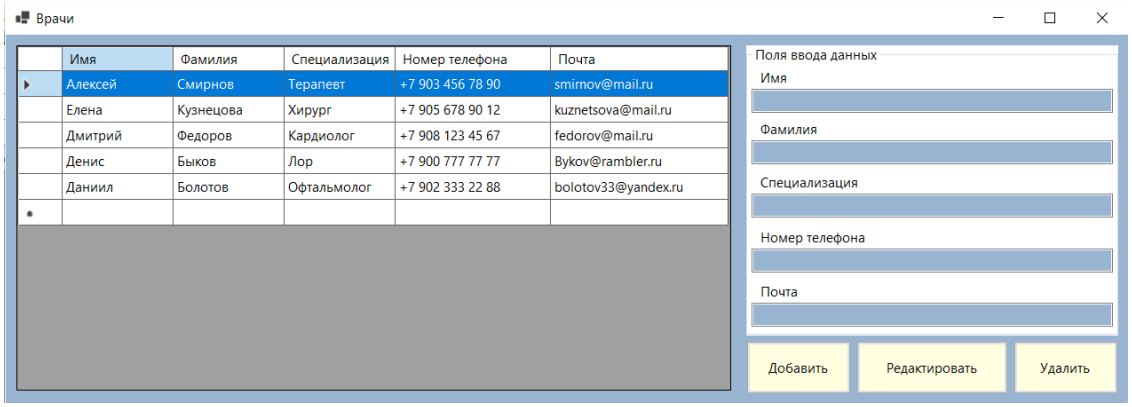

Рисунок 19 – Окно добавления врачей с новым врачом

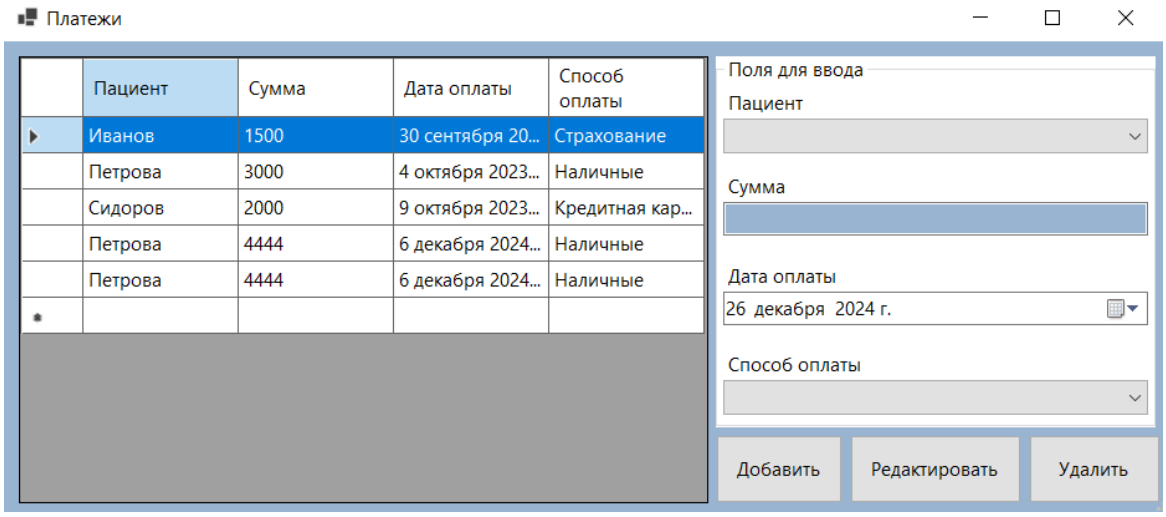

Рисунок 20 – Окно добавления платежей

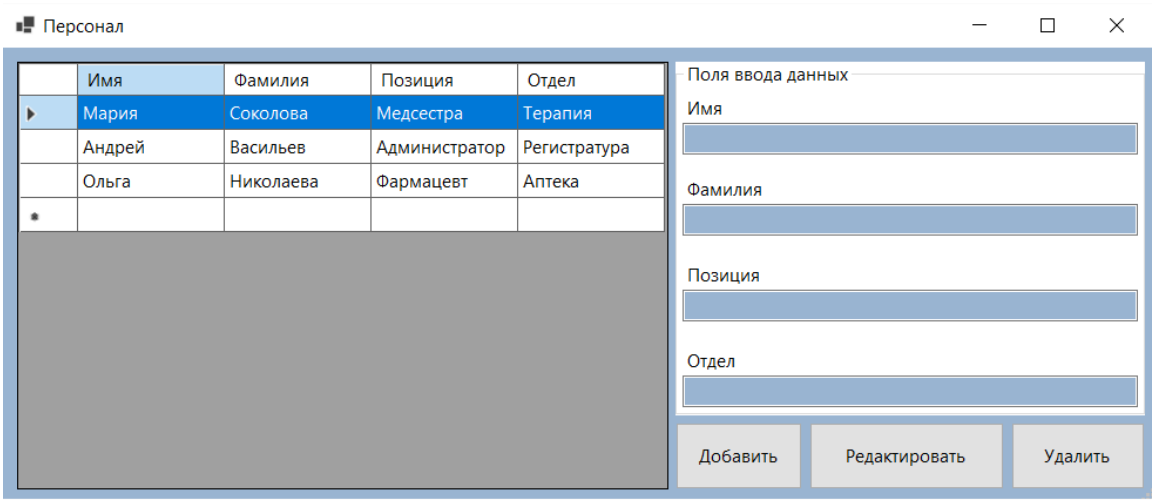

Рисунок 21 – Окно добавления персонала

Во вкладке меню содержатся: отделения, палаты, лекарства, предписания к лекарствам

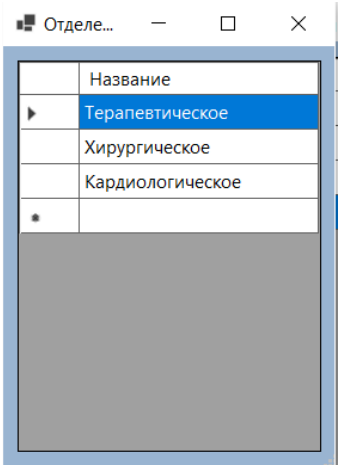

Рисунок 22 – Окно со списком отделений

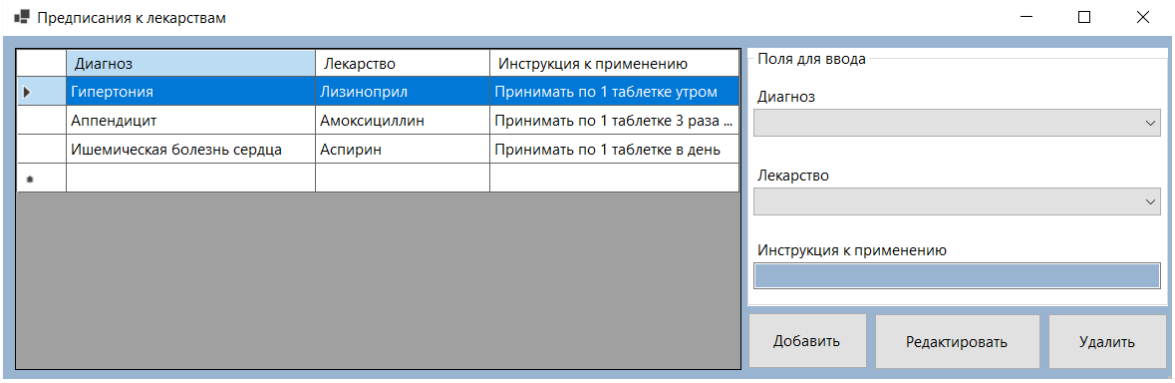

Рисунок 23 – Окно добавления предписаний к лекарствам

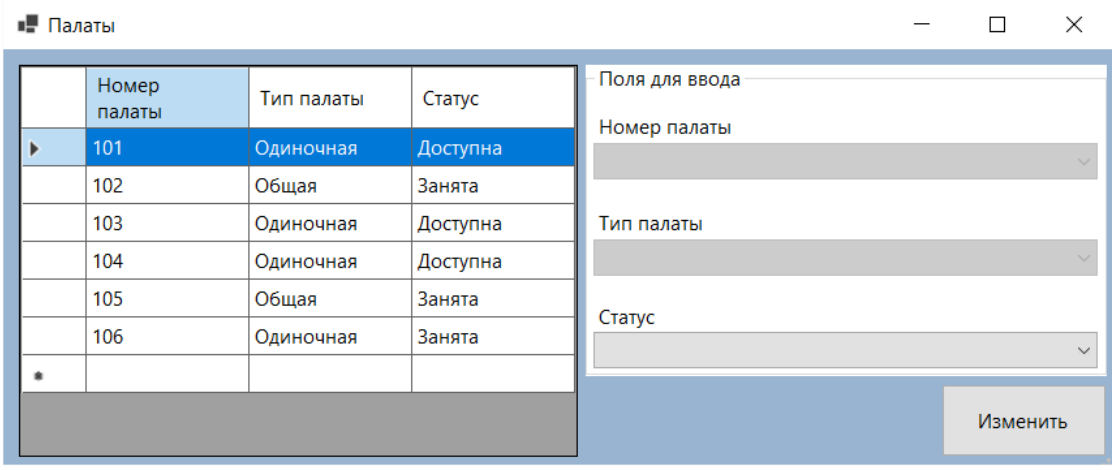

Рисунок 24 – Окно со списком палат

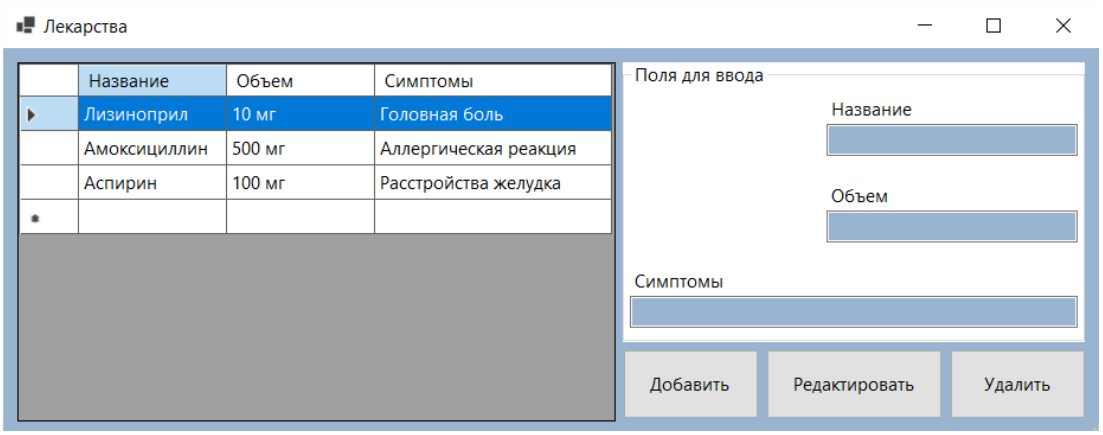

Рисунок 25 – Окно со списком лекарств

Заключение

В рамках данной курсовой работы была исследована необходимость внедрения системы управления базами данных для автоматизации процессов в больнице. В ходе выполнения работы был проведён анализ требований, разработана структура базы данных и описаны основные сценарии работы с системой.

Результатом исследования стало создание базовой концепции СУБД, позволяющей эффективно управлять данными пациентов, записями на приём, медицинскими картами и прочими аспектами работы больницы. Реализация подобной системы способствует оптимизации времени, позволяет избежать ошибок в работе медицинского персонала и повышает качество оказываемых услуг.

Таким образом, поставленная цель работы была достигнута, а предъявленные задачи выполнены, что подтверждает практическую и теоретическую значимость разработанной базы данных для больничных учреждений.

Список литературы

1) Бьерн Страуструп, Язык программирования C++. Специальное издание. Пер. с англ. – М.: Издательство Бином, 2011 г. – 1136 с.: ил.

2) Казарин С.А., Клишин А.П. К 143 Среда разработки Java-приложений Eclipse: (ПО для объектно-ориентированного программирования и разработки приложений на языке Java): Учебное пособие. Москва 2008. — 77 с.

3) Медведев, М. А. М42 Программирование на СИ# : учеб. пособие / М. А. Медведев, А. Н. Медведев. — Екатеринбург : Изд-во Урал. ун-та, 2015. — 64 с.

4) Мотев А. А. М85 Уроки MySQL. Самоучитель. — СПб.: БХВ-Петербург, 2006. — 208 с.: ил.

5) Рындина С. В.. – Пенза : Изд-во ПГУ, 2023. – 82 с. , Цифровые технологии управления получением, хранением, передачей и обработкой больших данных: SQLite : учеб.-метод. пособие.

6) Сапаров А.Ю., Разработка Windows Forms приложений на языке программирования C#: учебно-методическое пособие / Сост.: А.Ю. Сапаров, Ижевск, 2020. 61 с.

1) Сергеева Т.И. Базы данных: модели данных, проектирование, язык SQL: учеб. пособие / Т.И. Сергеева, М.Ю. Сергеев. Воронеж: ФГБОУ ВПО «Воронежский государственный технический университет», 2012. 233 с.

Приложение А. Модули данных

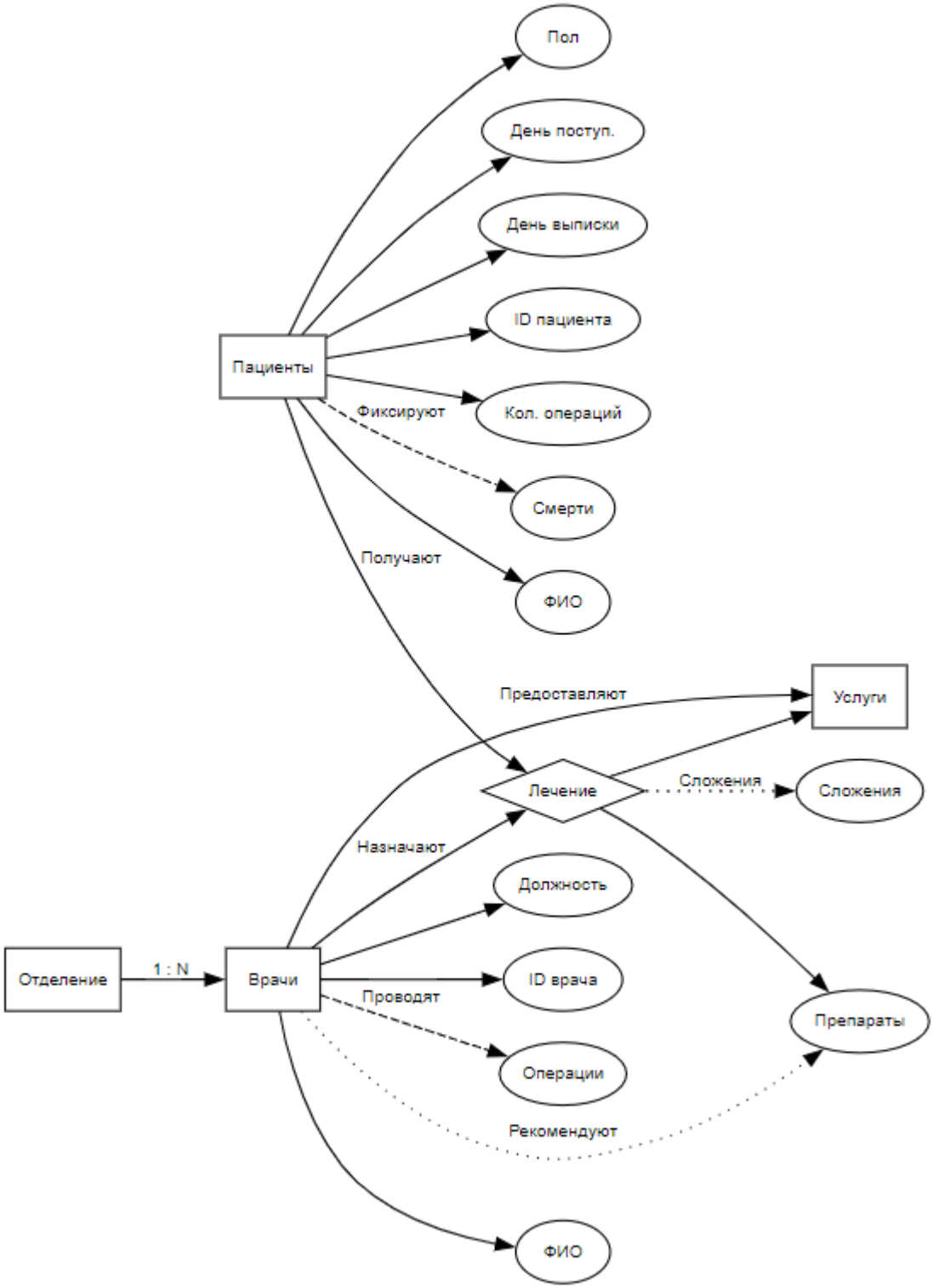

Рисунок 1 – ER-диаграмма концептуальной модели

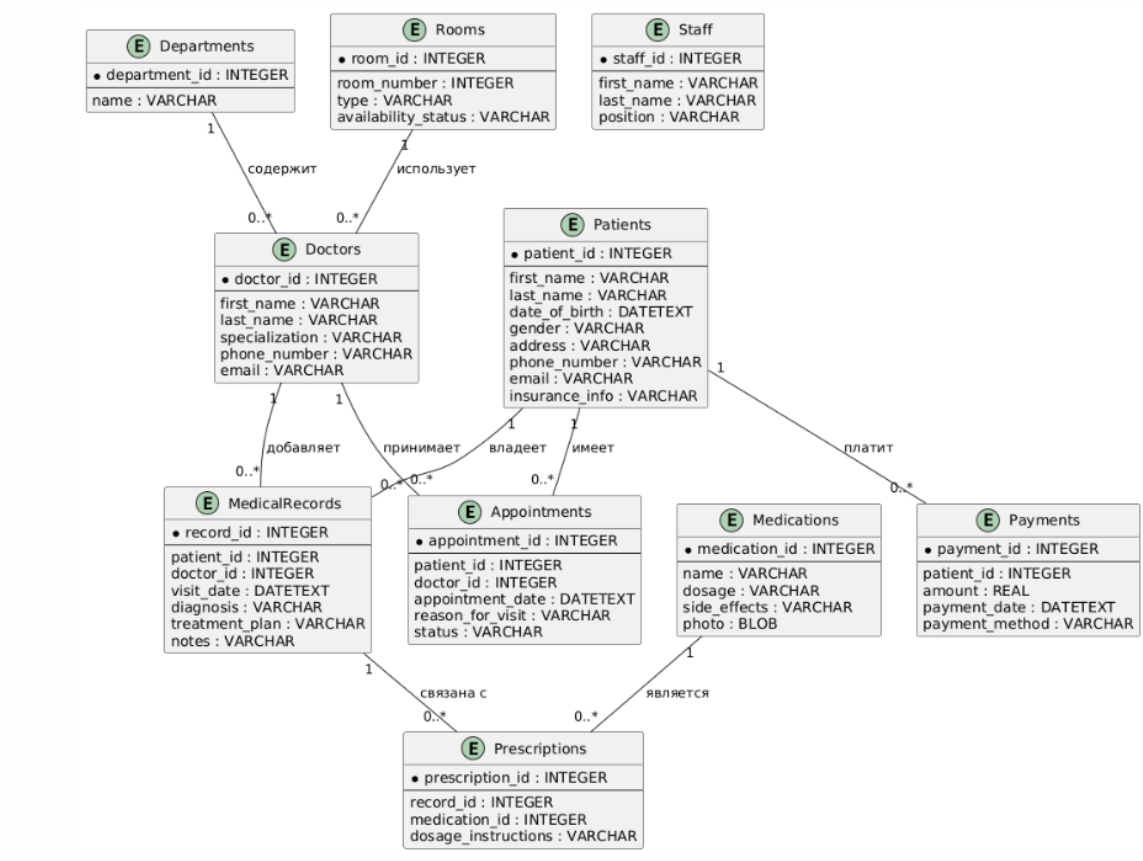

Рисунок 2 – UML-диаграмма базы данных

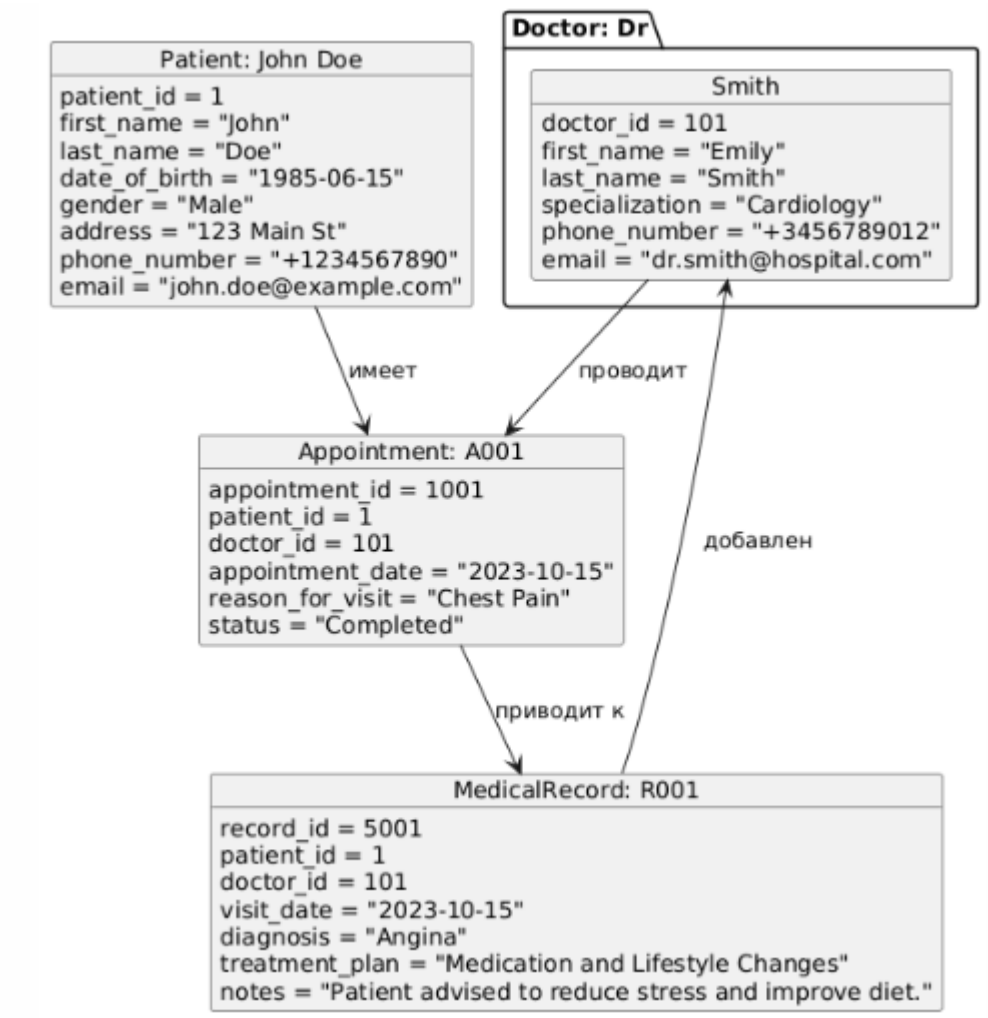

Рисунок 3 – UML-диаграмма объектов

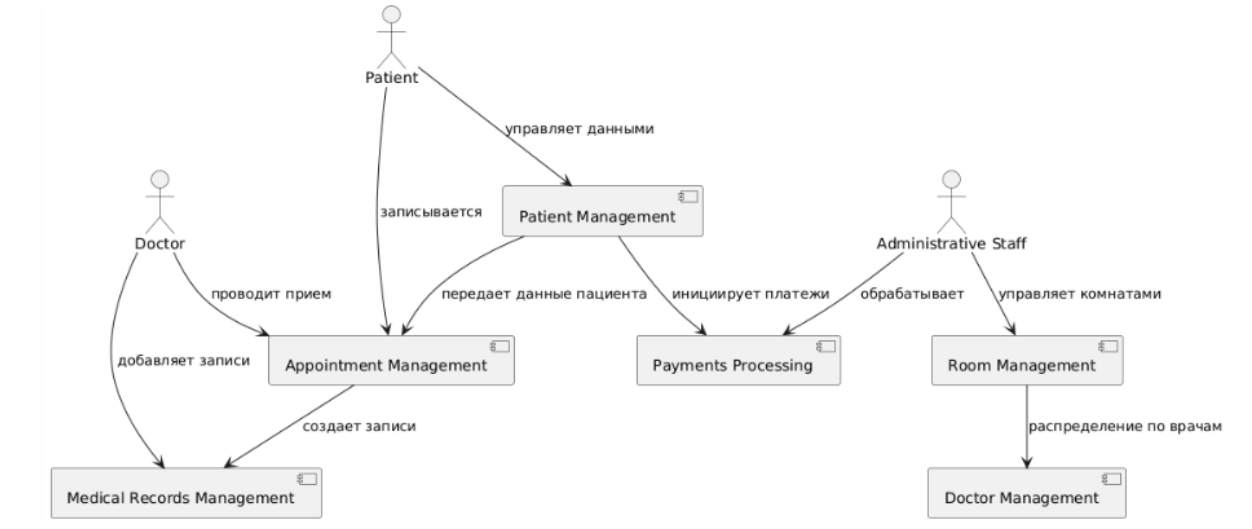

Рисунок 4 – UML-диаграмма компонентов

Приложение Б. Текст кода

Весь исходный код представлен по ссылке:

https://github.com/Banchilla/Hospital