



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Разработка программного обеспечения для
хранения и аналитики данных медицинской
компании.

Студент ИУ7-66Б
(Группа)

П.Ю. Сироткина
(Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы

Ю.М. Гаврилова
(Подпись, дата) (И.О.Фамилия)

2022 г.

РЕФЕРАТ

Расчетно-пояснительная записка 54 с., 26 рис., 1 табл., 21 источн., 2 прил.

Ключевые слова: Базы Данных, PostgreSQL, Web-приложение, Golang, медицинская информационная система, реляционная модель данных.

Объектом разработки является медицинская информационная система.

Целью данной курсовой работы является разработка базы данных для хранения и аналитики данных медицинской компании.

Для достижения цели были выполнены следующие задачи:

- формализована поставленная задача;
- описана структура базы данных;
- рассмотрены модели данных и выбрана реляционная модель;
- спроектирована база данных, реализованы соответствующие запросы;
- реализован интерфейс для доступа к базе данных;
- реализовано программное обеспечение для работы с базой данных.

В результате выполнения работы была спроектирована и разработана заявленная база данных, а также соответствующее приложение.

СОДЕРЖАНИЕ

РЕФЕРАТ	3
ВВЕДЕНИЕ	5
1 Аналитический раздел	7
1.1 Формализация поставленной задачи	7
1.2 Системы управления базами данных	10
1.3 Методы защиты информации	12
1.4 Обзор существующих реализаций	12
2 Конструкторский раздел	17
2.1 Ролевая модель	17
2.2 Проектирование базы данных	19
2.3 Диаграммы последовательностей	24
2.4 Описание хранимой функции	27
3 Технологический раздел	28
3.1 Выбор средств разработки	28
3.2 Детали разработки приложения	29
3.3 Хранимая функция	35
3.4 Тестирование	35
3.5 Описание презентационного уровня приложения	38
ЗАКЛЮЧЕНИЕ	44
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	46
ПРИЛОЖЕНИЕ А Сценарий создания базы данных	47
ПРИЛОЖЕНИЕ Б Запросы к базы данных	51
ПРИЛОЖЕНИЕ В Детали реализации	53

ВВЕДЕНИЕ

В современном мире люди генерируют огромное количество данных. С 2012 года и по настоящее время ежедневно генерируется около $2.5 \cdot 10^{18}$ байтов информации[1]. Объемы производимых данных постоянно увеличиваются как следствие автоматизации многих процессов жизнедеятельности человека, также рост объема данных обусловлен тем фактом, что население планеты постоянно растет.

В данной курсовой работе будет рассмотрена сфера здравоохранения человека и генерируемая информация, связанная с ней.

Целью курсовой работы является разработка и реализация медицинской информационнои системы (далее МИС), прикладные команды которой используют данные, хранящиеся в базе данных, для автоматизации и анализа деятельности клиники с целью дальнейшего планирования деятельности учреждения. Исходными данными для разработки являются данные о пациентах, врачах, заболеваниях, методах их лечения и др.

Выбор предметной области для выполнения курсовой работы был обусловлен следующими причинами:

1. Согласно Федеральному закону от 29 июля 2017 г. №242-ФЗ «О внесении изменений в отдельные законодательные акты РФ по вопросам применения информационных технологий в сфере охраны здоровья» было приказано «создавать медицинские информационные системы, содержащие данные о пациентах, об оказываемой им медицинской помощи, о медицинской деятельности медицинских организаций с соблюдением требований, установленных законодательством РФ в области персональных данных, и соблюдением врачебной тайны». Таким образом, создание МИС теперь обусловлено на законодательном уровне.
2. Одним из важных последствий развития пандемии COVID-19 стал резкий рост обращений в больницы и оказания соответствующих услуг, в том числе вакцинации населения, возникла острая необходимость в еще более стремительной автоматизации данного процесса.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- формализовать поставленную задачу;
- описать структуру базы данных: сущности, связи между ними, основные понятия;
- провести анализ систем управления базами данных и выбрать подходящий вариант для решения задачи;
- спроектировать и заполнить базу данных, реализовать соответствующие запросы;
- реализовать интерфейс для доступа к базе данных;
- реализовать программное обеспечение, которое позволит пользователю создавать, получать и изменять сведения из разработанной базы данных.

1 Аналитический раздел

1.1 Формализация поставленной задачи

Задачей данной курсовой является разработка базы данных для хранения и анализа данных некой медицинской компании, а также соответствующего приложения, предоставляющего интерфейс для работы с базой.

Предметной областью поставленной задачи является деятельность некоторой медицинской клиники. Для автоматизации учета информации разрабатывается т.н. медицинская информационная система. Под этим термином можно понимать любую информационную систему, которая хранит и обрабатывает информацию, связанную со здоровьем пациентов и деятельностью учреждений здравоохранения[2].

Основными трудностями при разработке МИС являются:

- трудности, связанные с государственным документооборотом, конфиденциальностью персональных данных и врачебной тайной[3];
- в рамках развития интегрированных медицинских информационных систем возрастает роль готовности медицинского персонала и пациентов к участию в этих процессах[4]. Большинство врачей практически не владеют компьютерными технологиями, поэтому им проще вести бумажные ведомости, чем использовать программное обеспечение, что приводит к существенному снижению эффективности работы клиники.

Общие требования к разрабатываемому программному обеспечению

Предметная область поставленной задачи является обширной и включает в себя множество понятий и связей между ними, в связи с этим были сформулированы следующие требования к разрабатываемой программе в рамках курсовой работы:

- должен быть предоставлен функционал для регистрации и аутентификации пользователей в системе, также должен быть создан личный кабинет с основной информацией о пользователе;
- должен быть предоставлен функционал для записи пациента на прием к врачу, а также отслеживания как активных, так и обработанных заявок. Должна формироваться медицинская карта с возможностью ее просмотра;
- должна быть предоставлена статистика по заболеваемости среди зарегистрированных случаев в данной клинике.

Также были сформулированы следующие допущения:

- в рамках курсовой работы не затрагивается тема контроля врачебной тайны и конфиденциальности персональных данных пациента. Таким образом, любой доктор может просмотреть все данные из медицинской карты пациента;
- в разрабатываемой МИС отсутствует платежная система, т.е. нельзя проводить и фиксировать финансовые транзакции;
- МИС не ведет складской учет (инвентаризация, маркировка медикаментов и т.д.);
- не предусматривается возможность отмены заявки на прием. В конечном итоге заявка должна быть одобрена специалистом;
- разрабатываемая МИС предоставляет функционал только для администрирования приемов клиентов докторами клиники, для администрирования сведений в медицинской карте пациента, а также предоставляет возможность сбора и отображения статистики обращений в клинику.

Формализация данных

На рисунке 1.1 представлена диаграмма сущность-связь в нотации Чена[5], описывающая сущности предметной области и их взаимодействие.

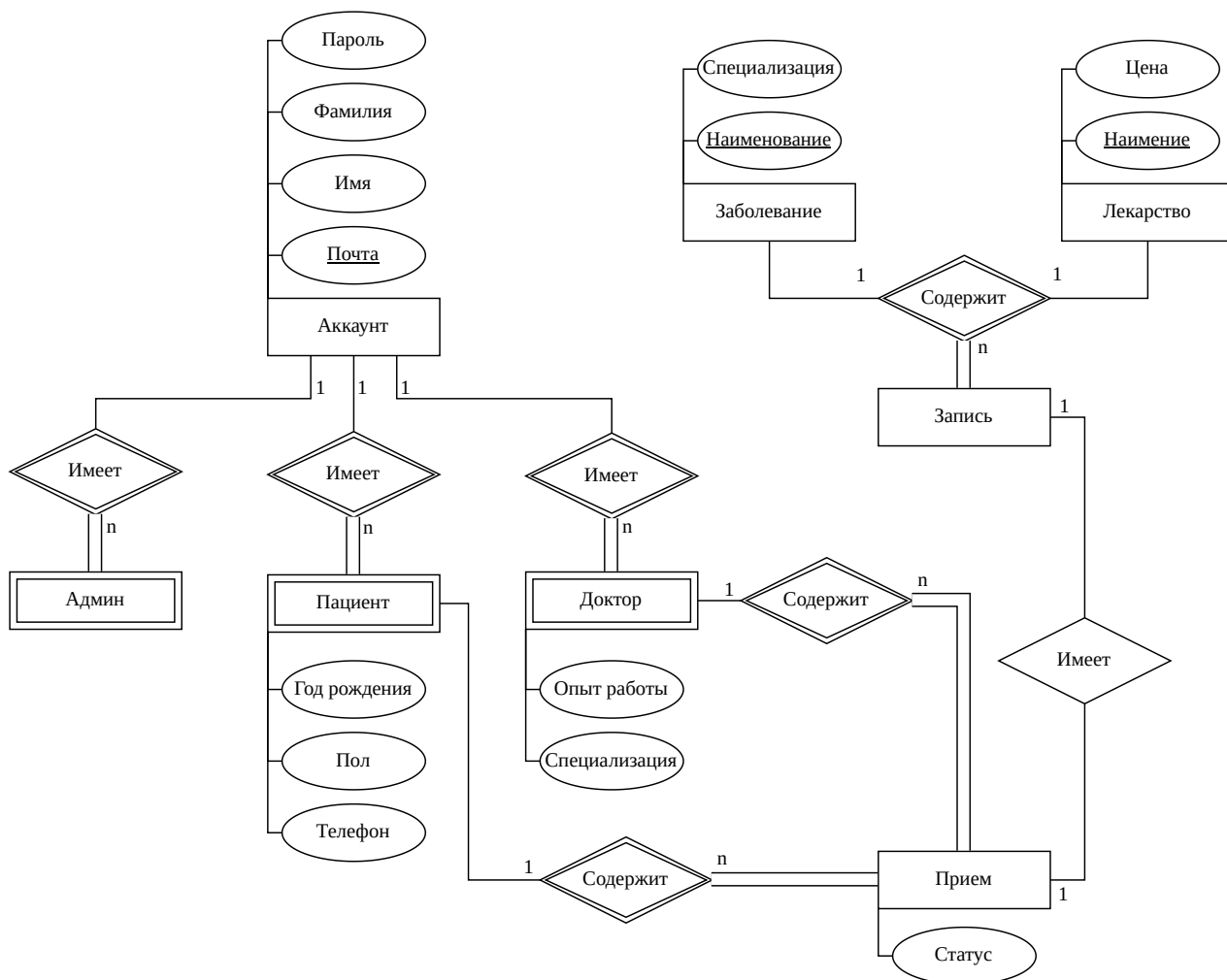


Рисунок 1.1 – Диаграмма сущность-связь в нотации Чена, описывающая сущности предметной области и их взаимодействие

Медицинская информационная система, проектируемая в ходе выполнения курсовой работы, включает в себя информацию о следующих объектах:

1. Запись медицинской карты - сущность, описывающая конкретный прием, поставленный диагноз и лекарство для лечения пациента. Набор таких сущностей образуют медицинскую карту пациента.
2. Прием - сущность, характеризующая поход пациента к врачу. Прием имеет статус, который может принимать одно из двух возможных значений: «Активный» (запланированный, еще не совершившийся) и «Обработанный» (совершившийся прием).

3. Диагноз - сущность, описывающая диагноз, который может быть поставлен пациенту. Каждому диагнозу поставлена в соответствие определенная специальность доктора, характеризующая сферу болезни.
4. Лекарство - сущность, описывающая медикаментозные средства, которые могут быть назначены пациенту.
5. Аккаунт - сущность общего вида, описывающая каждый из трех возможных подтипов сущностей-пользователей в системе: пациент, доктор или администратор.

1.2 Системы управления базами данных

Система управления базами данных — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных[6].

Классифицировать СУБД можно, используя различные признаки классификации, например, по степени распространенности, по способу доступа к БД и т.д. Важнейшим классификационным признаком СУБД является тип модели данных, поддерживаемый СУБД.

Классификация СУБД на основе типа модели данных

Дореляционная модель данных

Иерархические модели имеют древовидную структуру, где каждому узлу соответствует один сегмент, представляющий собой поименованный линейный кортеж полей данных. Каждому сегменту соответствует один входной и несколько выходных сегментов.

Каждый элемент структуры лежит на единственном иерархическом пути, начинающемся от корневого. Модель допускает только два типа связей между сущностями: «один к одному» и «один ко многим».

Сетевая модель данных является расширением иерархической и призвана устранить ограничения, связанные с ней. В иерархических структурах запись-потомок должна иметь в точности одного предка; в сетевой структуре данных потомок может иметь любое число предков.

Сетевая и иерархическая модели данных тесно связаны с физическим размещением информации.

Реляционная модель данных

Реляционная модель данных представляет собой совокупность данных, состоящую из набора двумерных таблиц. В теории множеств таблице соответствует термин отношение (relation), физическим представлением которого и является таблица.

Таблица состоит из строк, называемых записями, и столбцов, называемых полями. На пересечении строк и столбцов находятся конкретные значения данных. В таблице не должно быть одинаковых строк, каждый столбец должен иметь уникальное значение.

Эта модель является логической, в отличие от иерархической и сетевой. Она опирается на такие разделы математики, как теория множеств и логика первого порядка.

Постреляционная модель данных

Данная модель является расширением реляционной модели. Она снимает ограничение неделимости данных, допуская многозначные поля, и набор значений воспринимается как самостоятельная таблица, встроенная в главную таблицу[7].

Многомерная модель представляет данные как многомерные массивы (гиперкубы), использование которых позволяет получать различные срезы при аналитической обработке данных. Осями многомерной системы координат служат основные атрибуты анализируемого бизнес-процесса. На пересечениях осей измерений находятся данные, количественно характеризующие процесс, — меры[8].

Объектно-ориентированная модель представляет из себя структуру, которую графически можно изобразить в виде дерева, узлами которого являются объекты. Базовыми понятиями являются: объекты, классы, методы, наследование, полиморфизм, инкапсуляция.

Выбор модели данных

Для решения поставленной задачи была выбрана реляционная модель данных, т.к. разрабатываемая база данных характеризуется большим набором отношений между сущностями, и в связи с этим схема базы данных,

основанной на этой модели, будет более наглядна и проста, чем аналогичная схема сетевых и иерархических моделей, т.к. в данном случае описание базы данных основывается только на естественной структуре данных без введения какого-либо дополнительного уровня абстракции для их представления. Это свойство обусловлено использованием теории множеств.

1.3 Методы защиты информации

Выделяют несколько основных уровней защиты информации в базах данных:

1. Защита на уровне хранилища. Одним из методов защиты на этом уровне является шифрование - преобразование некоего осмысленного текста в неосмысленный набор символов посредством применения некоторой шифрующей функции[9].
2. Защита на уровне базы данных. Одним из методов защиты на этом уровне является установление паролей. Пароли устанавливаются пользователями или администраторами системы. Учет и управление паролями выполняется самой СУБД[10][11].
3. Защита на уровне приложения. Процесс предоставления доступа к БД осуществляется путем выполнения регистрации или аутентификации, а затем авторизации в системе. Вход в систему открывает доступ к БД и соответствующим правам авторизованного пользователя.

1.4 Обзор существующих реализаций

Medesk

Данная МИС позволяет проводить мониторинг работы медицинского центра. Система поддерживает складской учет, имеет внутреннюю платежную систему, представляет механизм для тайм-менеджмента, колл-центра и др. Как и многих систем подобного рода, у Medesk отсутствует бесплатная версия программного обеспечения. Средняя стоимость подписки составляет 3500 рублей в месяц по состоянию на 2022 год.

Данная система представлена на следующих платформах: Web-приложение, Windows, MacOS, Linux.

На рисунке 1.2 представлен пользовательский интерфейс МИС Medesk.

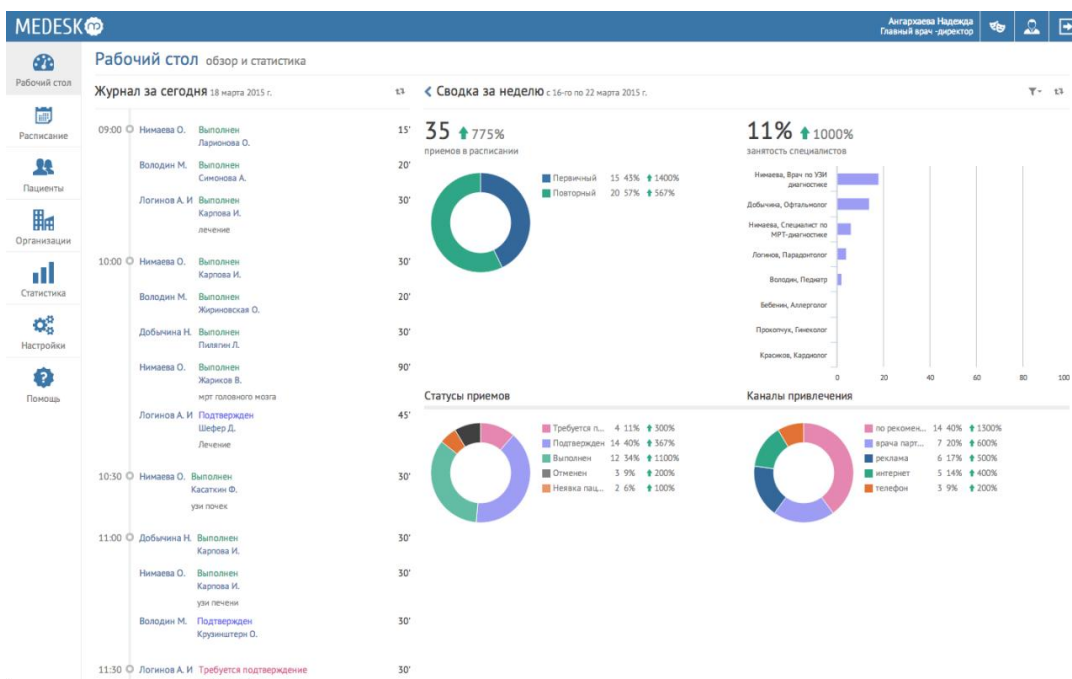


Рисунок 1.2 – Интерфейс МИС Medesk

Medods

Программное обеспечение специализировано на стоматологических клиниках, но в общем случае может предоставлять базовый функционал для любых типов медицинских клиник.

На рисунке 1.3 представлен пользовательский интерфейс МИС Medods.

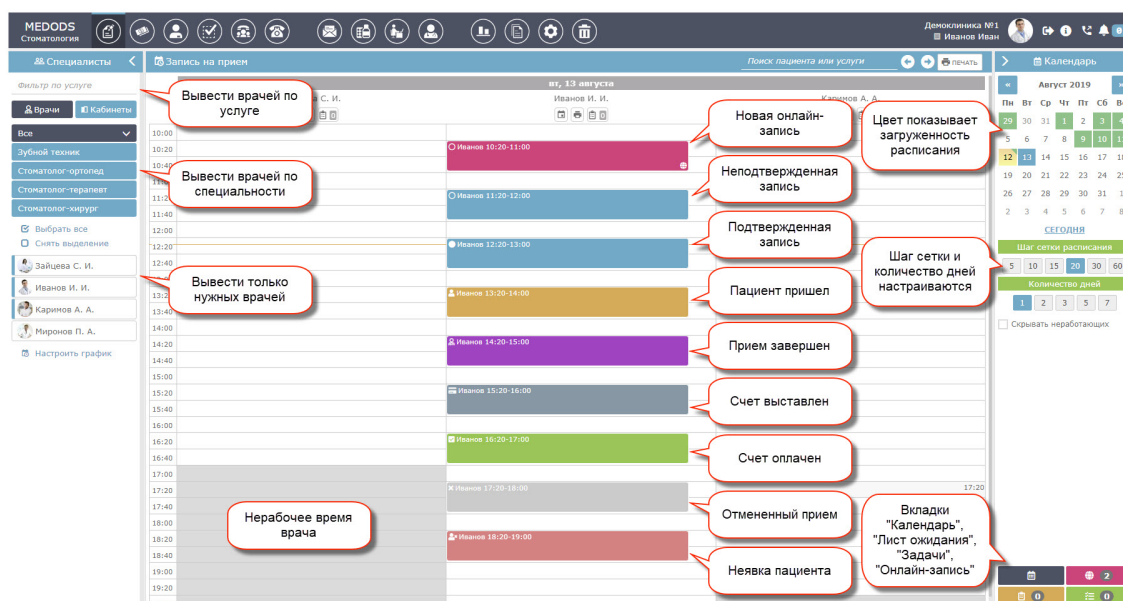


Рисунок 1.3 – Интерфейс МИС Medods

ПО предоставляет функционал для управления регистратурой, имеет кабинет врача и руководителя, также предоставляет механизм для онлайн-записи на прием и технической поддержки. Система также имеет платежную систему.

У данного продукта также отсутствует бесплатная версия. Средняя стоимость подписки составляет 4900 рублей в месяц по состоянию на 2022 год.

Данная система представлена на следующих платформах: Web-приложение, Windows, MacOS, Linux.

Инфоклиника

Данная МИС предоставляет функционал для многопрофильных клиник. В зависимости от потребностей и особенностей клиники, компания конфигурирует ПО для каждой компании индивидуально и предоставляет его заказчиком. Соответственно, у этой МИС также нет бесплатной версии, а средняя стоимость подписки зависит от типа поставляемого ПО.

На рисунке 1.4 представлен пользовательский интерфейс МИС Инфоклиника.

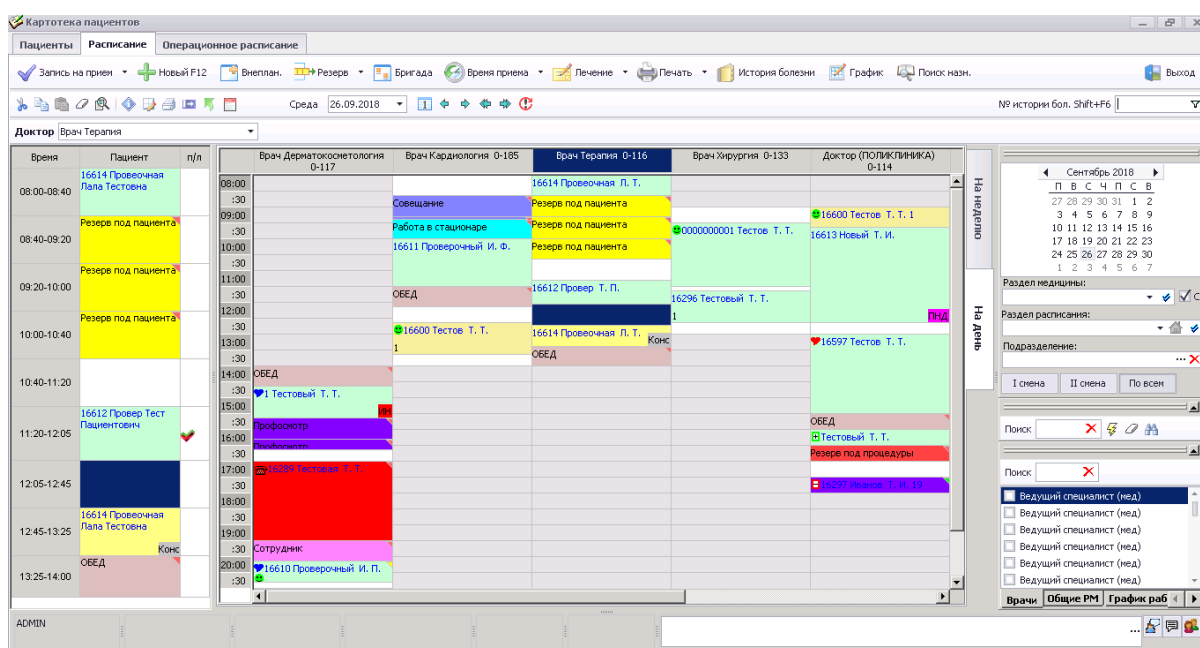


Рисунок 1.4 – Интерфейс МИС Инфоклиника

Инфоклиника предоставляет широкий функционал: особое внимание уделено созданию мессенджера внутри системы, предоставляется механизм для ведения складского учета, проведения платежей, система предоставляет функционал для страхования.

Данная система представлена на следующих платформах: Web-приложение, Windows.

Сравнительный анализ существующих реализаций

В таблице 1.1 представлен сравнительный анализ существующих реализаций. Анализ был проведен в соответствии со следующими критериями:

- стоимость - критерий, определяющий является ли ПО бесплатным;
- профиль - является ли приложение многопрофильным с точки зрения предоставляемых услуг;
- кабинет - возможность создать аккаунт в системе пациентом самостоятельно;
- заявка - возможность оставить заявку на прием самостоятельно;
- карта - возможность ведения медицинской амбулаторной карты пациента.

Таблица 1.1 – Сравнительный анализ существующих реализаций

Реализация	Стоимость	Профиль	Кабинет	Заявка	Карта
Medesk	-	+	+	+	+
Medods	-	-	+	-	+
Инфоклиника	-	+	-	-	-

Обычно ПО подобного рода ориентировано больше на администраторский функционал, чем на пользовательский. Разрабатываемое ПО будет ориентировано на функционал пациента в большей степени, чем это сделано в рассмотренных реализациях: пациент будет иметь возможность самостоятельно создать аккаунт в системе, самостоятельно оставить заявку на прием, посмотреть заявки и медицинскую историю (карту), будет сформирован личный кабинет. Также разрабатываемое ПО будет распространяться бесплатно и будет многопрофильным.

Вывод

В данном разделе была формализована поставленная задача, сформулированы общие требования и допущения к разрабатываемому ПО, проведен обзор существующих реализаций. Были рассмотрены модели хранения данных и для решения поставленной задачи была выбрана реляционная модель.

2 Конструкторский раздел

2.1 Ролевая модель

Ролевая модель используется для реализации системы безопасности сервера базы данных и позволяет разрешать или запрещать тем или иным группам пользователей работу с объектами базы данных.

В рамках поставленной задачи выделены следующие роли:

1. Пациент - роль, которой соответствует функционал записи на прием к врачу, просмотра медицинской карты, личного кабинета и активных заявок на прием.
2. Доктор - роль, которой соответствует функционал обработки заявок на прием, назначения диагнозов и лекарств, а также просмотр личного кабинета.
3. Администратор - роль, которой соответствует функционал регистрации новых пользователей в системе с ролями «Доктор» и «Администратор», функционал просмотра зарегистрированных пользователей в системе с любой ролью, а также просмотр статистики о заболеваемости.

Варианты использования системы

На рисунках 2.1-2.4 представлены диаграммы вариантов использования системы в соответствии с выделенными типами пользователей.



Рисунок 2.1 – Диаграмма вариантов использования системы для незарегистрированного пользователя

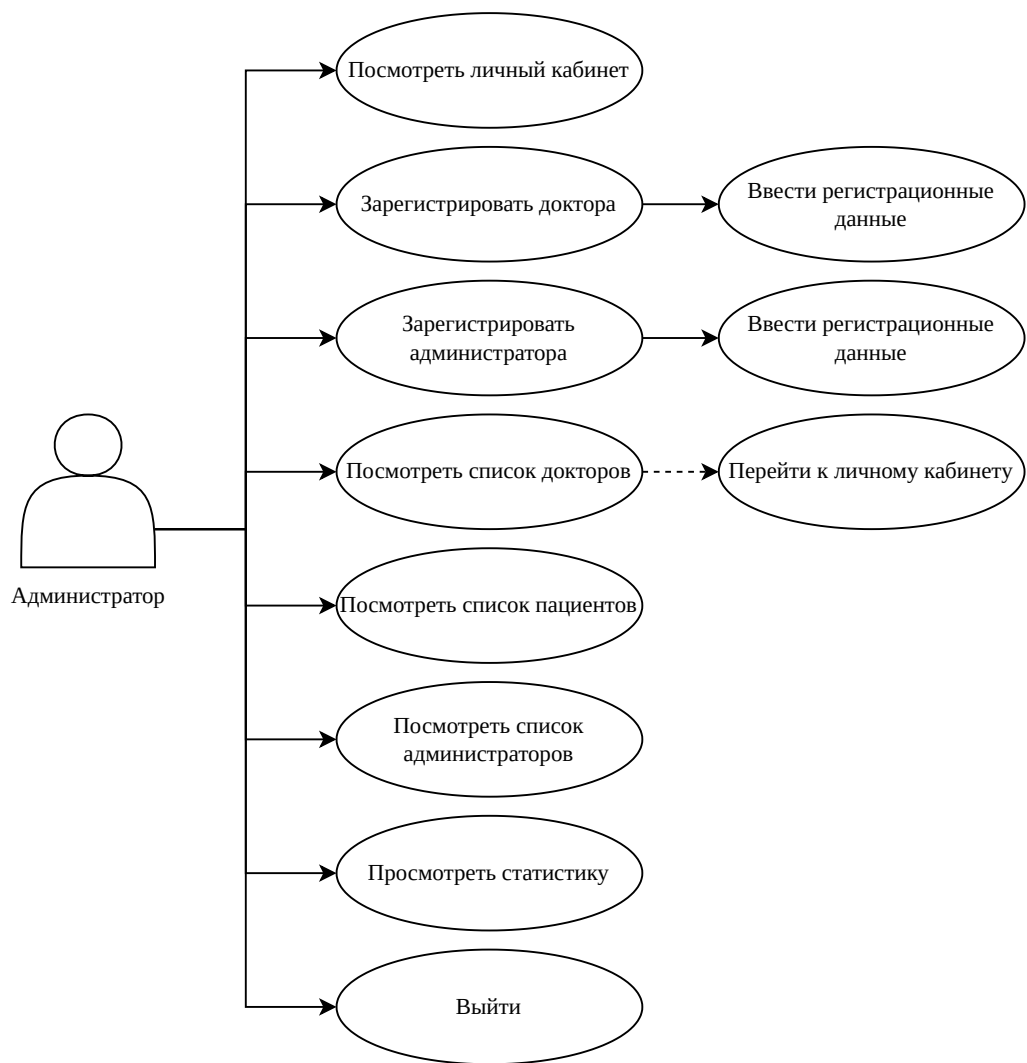


Рисунок 2.2 – Диаграмма вариантов использования системы для администратора

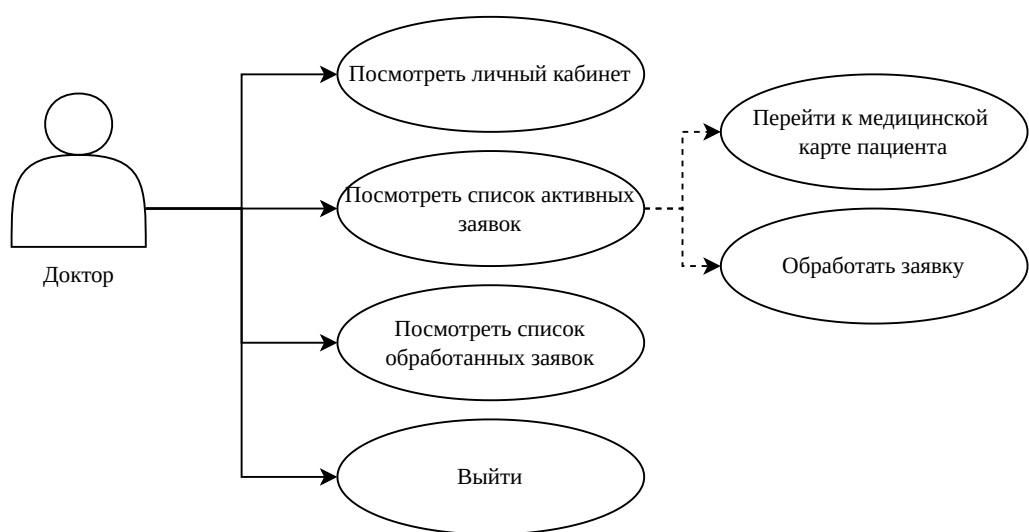


Рисунок 2.3 – Диаграмма вариантов использования системы для доктора

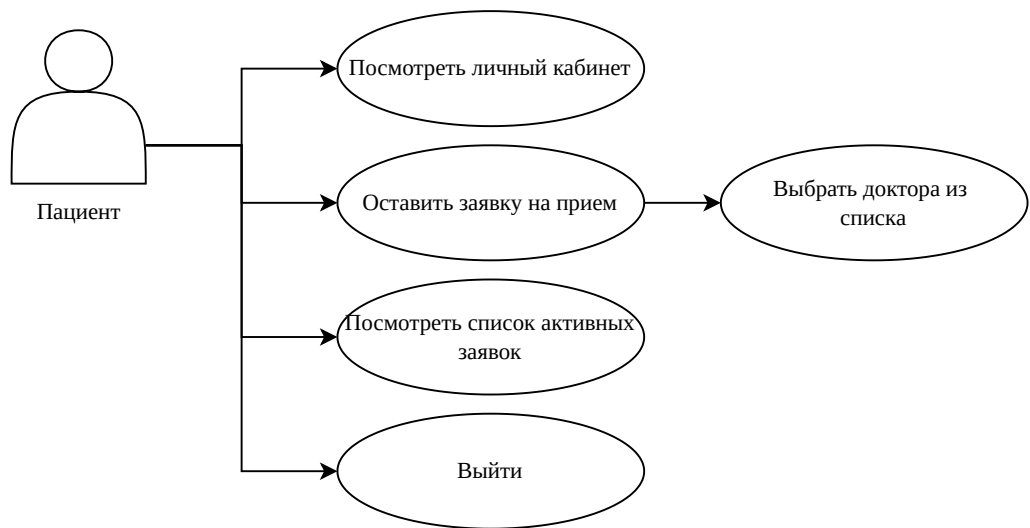


Рисунок 2.4 – Диаграмма вариантов использования системы для пациента

2.2 Проектирование базы данных

На рисунке 2.5 представлена ER-диаграмма разрабатываемой базы данных, спроектированная на основе выделенных сущностей и их свойств из предыдущего раздела.

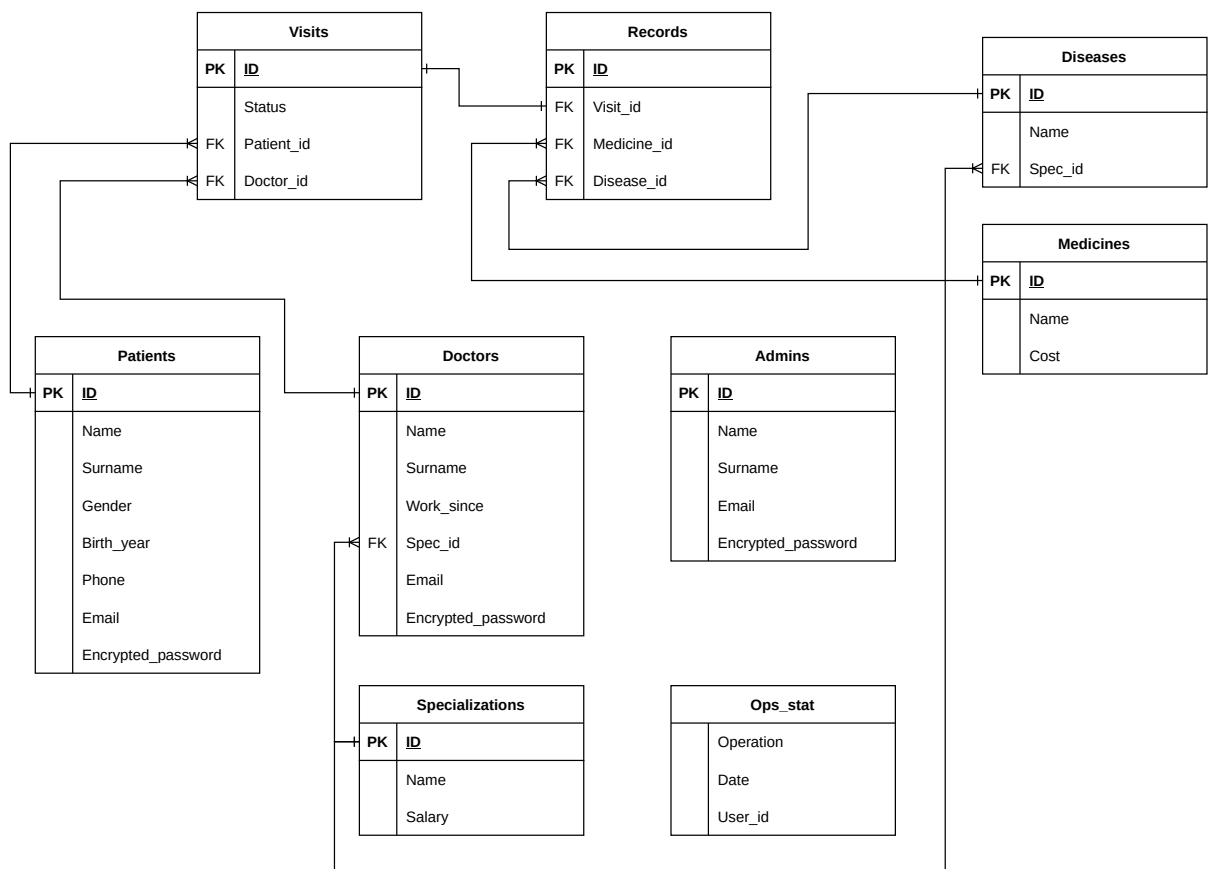


Рисунок 2.5 – ERD-диаграмма разрабатываемой базы данных

Далее приведено описание полей каждой таблицы из базы данных. Соответствующий скрипт можно посмотреть в приложении А.

Таблица Medicines

Данная таблица используется для описания назначаемого пациенту медицинского препарата.

Значения полей:

- ID - уникальный идентификатор медицинского препарата, целочисленный тип.
- Name - наименование, строковый тип.
- Cost - цена (в российских рублях), целочисленный тип.

Таблица Specializations

Данная таблица используется для описания врачебной специализации.

Значения полей:

- ID - уникальный идентификатор специализации, целочисленный тип.
- Name - наименование, строковый тип.
- Salary - зарплата доктора с соответствующей специализацией (в российских рублях), целочисленный тип.

Таблица Diseases

Данная таблица используется для описания диагностируемого заболевания.

Значения полей:

- ID - уникальный идентификатор заболевания, целочисленный тип.
- Name - наименование, строковый тип.
- Spec_id - идентификатор специализации, к которой относится заболевание. Является внешним ключом относительно поля ID таблицы Specializations, целочисленный тип.

Таблица Admins

Данная таблица используется для описания типа пользователя «Администратор».

Значения полей:

- ID - уникальный идентификатор пользователя «Администратор», целочисленный тип.
- Name - имя, строковый тип.
- Surname - фамилия, строковый тип.
- Email - адрес электронной почты, строковый тип.
- Encrypted_password - зашифрованный пароль, строковый тип.

Таблица Doctors

Данная таблица используется для описания типа пользователя «Доктор».

Значения полей:

- ID - уникальный идентификатор пользователя «Доктор», целочисленный тип.
- Name - имя, строковый тип.
- Surname - фамилия, строковый тип.
- Work_since - год начала врачебной практики, целочисленный тип.
- Spec_id - идентификатор специализации доктора. Является внешним ключом относительно поля ID таблицы Specializations, целочисленный тип.
- Email - адрес электронной почты, строковый тип.
- Encrypted_password - зашифрованный пароль, строковый тип.

Таблица Patients

Данная таблица используется для описания типа пользователя «Пациент».

Значения полей:

- ID - уникальный идентификатор пользователя «Доктор», целочисленный тип.
- Name - имя, строковый тип.
- Surname - фамилия, строковый тип.
- Gender - пол, строковый тип.
- Birth_year - год рождения, целочисленный тип.
- Phone - номер телефона, строковый тип.
- Email - адрес электронной почты, строковый тип.
- Encrypted_password - зашифрованный пароль, строковый тип.

Таблица Visits

Данная таблица используется для описания приема пациента.

Значения полей:

- ID - уникальный идентификатор приема, целочисленный тип.
- Status - статус приема. Может принимать одно из двух возможных значений: «Активный» (запланированный, еще не совершившийся) и «Обработанный» (совершившийся прием), строковый тип.
- Patient_id - идентификатор пациента. Является внешним ключом относительно поля ID таблицы Patients, целочисленный тип.
- Doctor_id - идентификатор доктора. Является внешним ключом относительно поля ID таблицы Doctors, целочисленный тип.

Таблица Records

Данная таблица используется для описания записи из медицинской карты пациента.

Значения полей:

- ID - уникальный идентификатор приема, целочисленный тип.
- Visit_id - идентификатор приема. Является внешним ключом относительно поля ID таблицы Visits, целочисленный тип.
- Medicine_id - идентификатор медицинского препарата. Является внешним ключом относительно поля ID таблицы Medicines, целочисленный тип.
- Disease_id - идентификатор заболевания. Является внешним ключом относительно поля ID таблицы Diseases, целочисленный тип.

Таблица Ops_stat

Данная таблица используется для логирования совершаемых в базе данных операций.

Значения полей:

- Operation - наименование совершившейся операции в базе данных (вставка, чтение, удаление, изменение), строковый тип.
- Date - дата совершения операции, тип для хранения даты.
- User_id - идентификатор пользователя, совершившего операцию, целочисленный тип.

2.3 Диаграммы последовательностей

На рисунке 2.6 представлена диаграмма последовательностей для сценария авторизации пользователей.

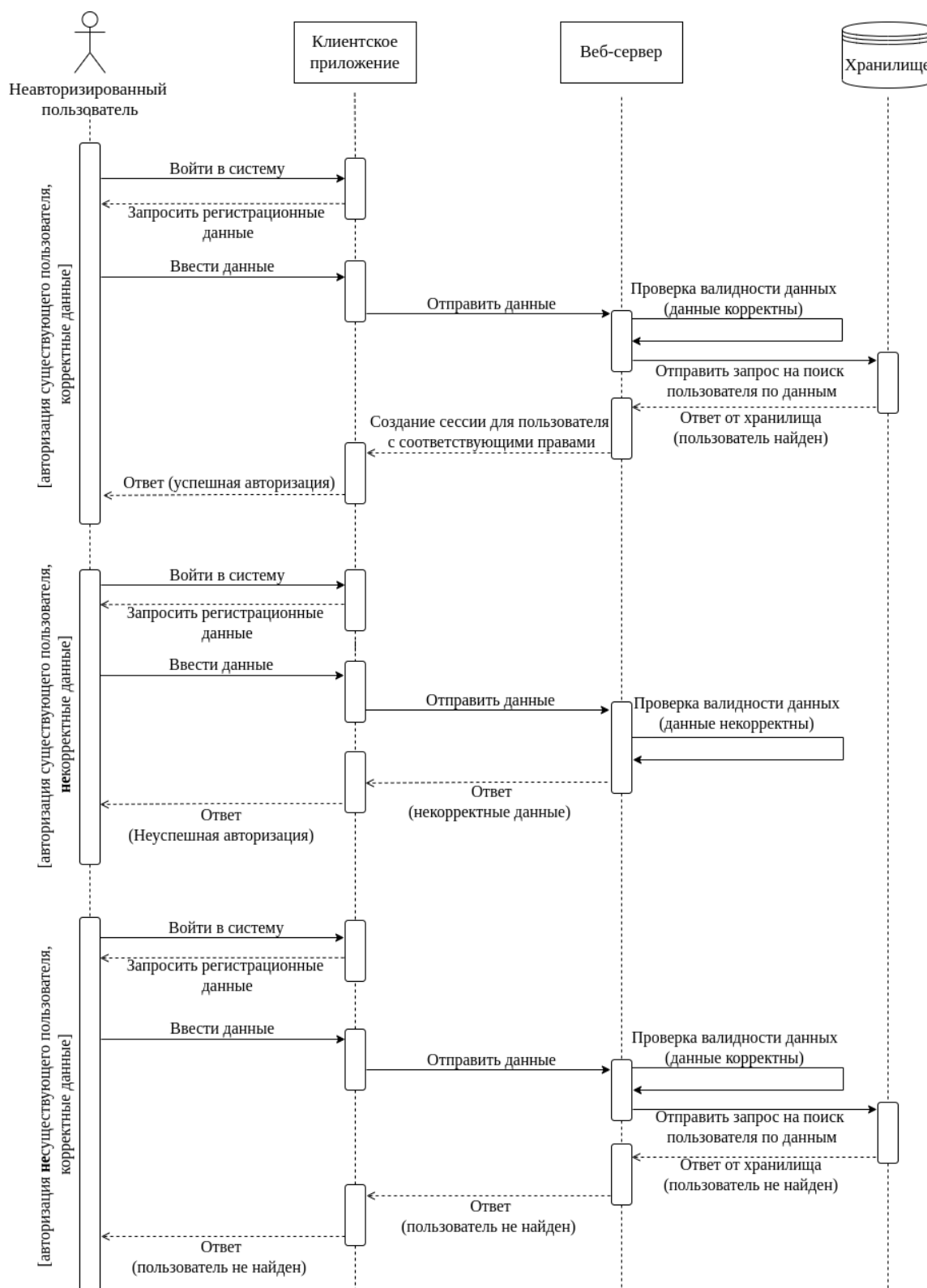


Рисунок 2.6 – Диаграмма последовательности для сценария авторизации пользователя

На рисунке 2.7 представлена диаграмма последовательности для сценария создания пациентом заявки на прием.

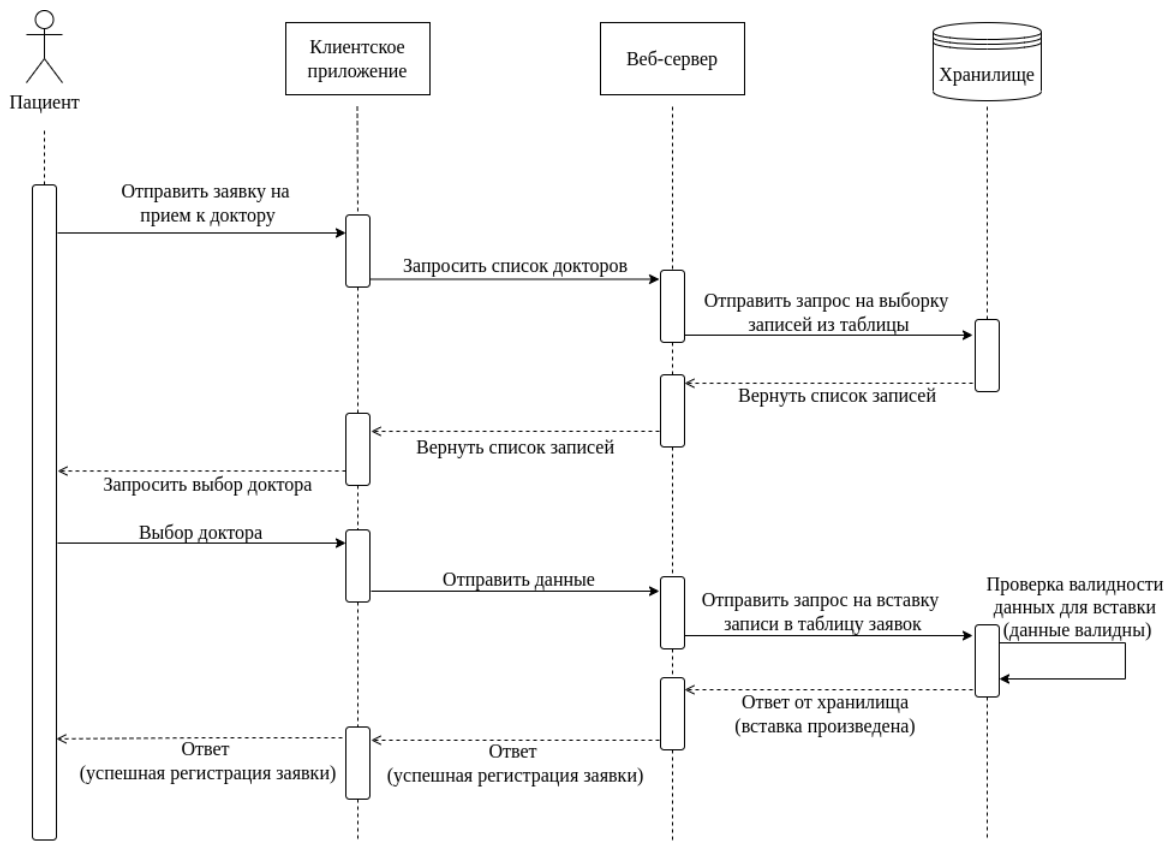


Рисунок 2.7 – Диаграмма последовательности для сценария создания заявки на прием

На рисунке 2.8 представлена диаграмма последовательности для сценария просмотра администратором списка докторов в системе.

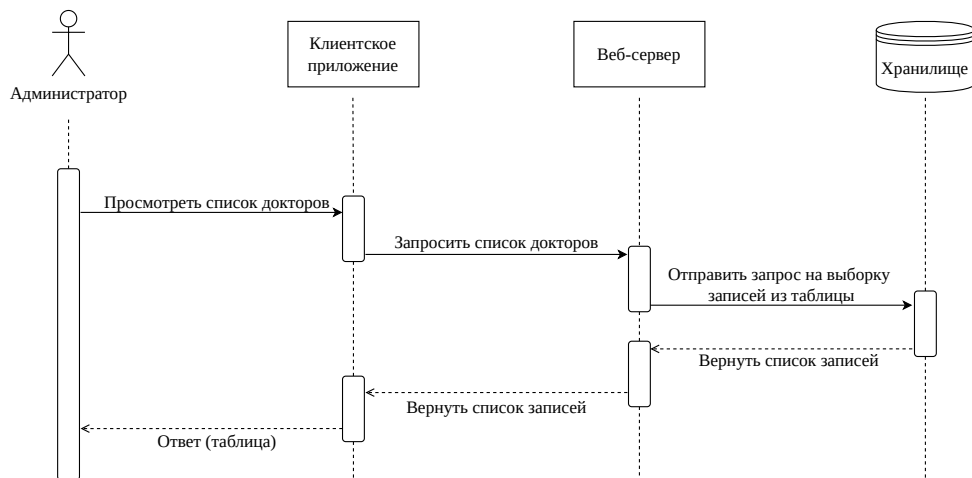


Рисунок 2.8 – Диаграмма последовательности для сценария просмотра списка докторов

На рисунке 2.9 представлена диаграмма последовательности для сценария обработки доктором заявки пациента на прием.

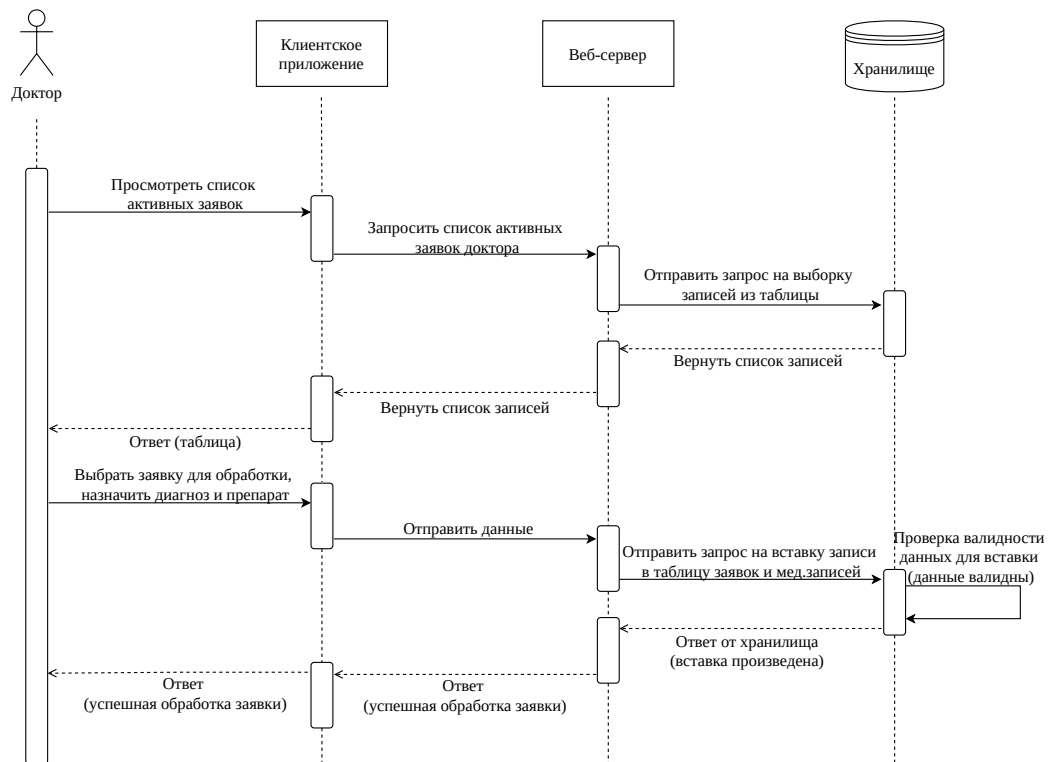


Рисунок 2.9 – Диаграмма последовательности для сценария обработки доктором заявки на прием

2.4 Описание хранимой функции

На рисунке 2.10 представлен алгоритм работы хранимой функции, которая подсчитывает статистику заболеваемости (в процентах) для конкретной специализации. Функция возвращает таблицу, содержащую два поля - процент заболеваемости (целое число от 0 до 100) и идентификатор заболевания (целое положительное число).

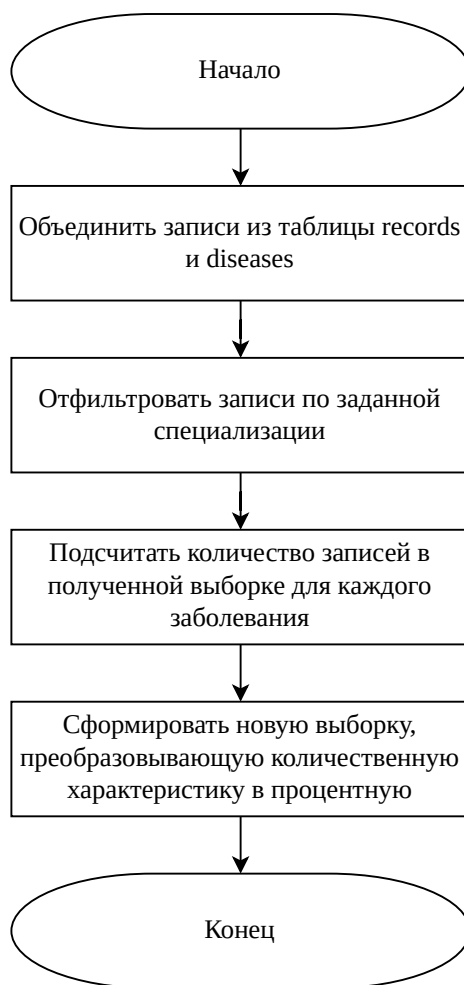


Рисунок 2.10 – Алгоритм работы хранимой функции подсчета статистики заболеваемости

Вывод

В данном разделе была спроектирована база данных: выделено три типа пользователей, приведены варианты использования системы в соответствии с ролями и представлена диаграмма разрабатываемой системы. В результате разработки должен быть спроектирован сервер, предоставляющий описанный в этом разделе функционал.

3 Технологический раздел

3.1 Выбор средств разработки

В качестве системы управления базами данных была выбрана PostgreSQL[12], поскольку данная СУБД поддерживает необходимую модель данных, выбор которой был обоснован ранее, также эта СУБД является свободным программным обеспечением (open source) и является бесплатной.

В качестве языка программирования для написания серверной части приложения был выбран Golang[13], поскольку он поддерживает все необходимые инструменты для решения поставленной задачи: предоставляется функционал для написания простых одностраничных серверов и http-клиентов, также он предоставляет необходимые инструменты для взаимодействия с БД, созданной с использованием PostgreSQL.

В ходе решения поставленной задачи были использованы дополнительные инструменты, расширяющие базовый функционал языка Golang: gorilla/mux[13] для настройки маршрутизации, http[14] для обработки входящих и исходящих запросов, sessions[15] для управления сессией пользователя, пакет rq[16] для взаимодействия с БД. Для шифрования данных был использован пакет x/crypto[17], для валидации данных - go-ozzo/validation[18].

Для оформления клиентских страниц веб-сервера был использован язык разметки HTML/CSS[19].

3.2 Детали разработки приложения

Описание структур, соответствующих таблицам базы данных

В данном разделе представлены структуры, описывающие таблицы базы данных. Структурам Admin, Doctor и Patient дополнительно создано поле Encrypted_password с целью хранения зашифрованного пароля.

На листингах 3.1-3.2 представлены структуры, соответствующие таблицам из базы данных.

Листинг 3.1 – Структуры, описывающие таблицы из базы данных

```
1  type Admin struct {
2      ID          int      'json:"id"'
3      Name        string   'json:"name"'
4      Surname     string   'json:"surname"'
5      Email       string   'json:"email"'
6      Password    string   'json:"password,omitempty"'
7      EncryptedPassword string 'json:"-"'
8  }
9
10 type Doctor struct {
11     ID          int      'json:"id"'
12     Name        string   'json:"name"'
13     Surname     string   'json:"surname"'
14     Work_since  int      'json:"work_since"'
15     Spec_id     int      'json:"spec_id"'
16     Email       string   'json:"email"'
17     Password    string   'json:"password,omitempty"'
18     EncryptedPassword string 'json:"-"'
19 }
20
21 type Patient struct {
22     ID          int      'json:"id"'
23     Name        string   'json:"name"'
24     Surname     string   'json:"surname"'
25     Gender      string   'json:"gender"'
26     Birth_year  int      'json:"birth_year"'
27     Phone       string   'json:"phone"'
28     Email       string   'json:"email"'
29     Password    string   'json:"password,omitempty"'
30     EncryptedPassword string 'json:"-"'
31 }
```

Листинг 3.2 – Структуры, описывающие таблицы из базы данных (продолжение)

```
1 type Disease struct {
2     ID          int    'json:"id"'
3     Name        string 'json:"name"'
4     Spec_id     int    'json:"spec_id"'
5 }
6
7 type Specialization struct {
8     ID          int    'json:"id"'
9     Name        string 'json:"name"'
10    Salary      int    'json:"salary"'
11 }
12
13 type Medicine struct {
14     ID          int    'json:"id"'
15     Name        string 'json:"name"'
16     Cost        int    'json:"cost"'
17 }
18
19 type Visit struct {
20     ID          int    'json:"id"'
21     Status      string 'json:"status"'
22     Doctor_id   int    'json:"doctor_id"'
23     Patient_id  int    'json:"patient_id"'
24 }
25
26 type Record struct {
27     ID          int    'json:"id"'
28     Visit_id    int    'json:"visit_id"'
29     Disease_id  int    'json:"disease_id"'
30     Medicine_id int    'json:"medicine_id"'
31 }
```

Установка соединения и отправка запросов

Для того, чтобы сервер мог принимать запросы и отправлять ответы, необходимо настроить роутер.

В приложении В на листингах В.1-В.2 представлена функция, настраивающая роутер сервера. Выделены закрытые ссылки, требующие соответствующие права доступа, для администратора, доктора и пациента.

На листинге 3.3 приведены функции, необходимые для настройки и запуска сервера.

Листинг 3.3 – Настройка и запуск сервера

```
1 func newServer(store store.Store, sessionStore sessions.Store) *server {
2     s := &server{
3         router:      mux.NewRouter(),
4         logger:      logrus.New(),
5         store:        store,
6         sessionStore: sessionStore,
7     }
8
9     s.configureRouter()
10
11    return s
12 }
13
14 func Start(config *Config) error {
15    db, err := newDB(config.DatabaseURL)
16    if err != nil {
17        return err
18    }
19
20    defer db.Close()
21    store := sqlstore.New(db)
22    sessionStore := sessions.NewCookieStore([]byte(config.SessionKey))
23    srv := newServer(store, sessionStore)
24
25    return http.ListenAndServe(config.BindAddr, srv)
26 }
27
28 func newDB(dbURL string) (*sql.DB, error) {
29    db, err := sql.Open("postgres", dbURL)
30    if err != nil {
31        return nil, err
32    }
33    if err := db.Ping(); err != nil {
34        return nil, err
35    }
36    return db, nil
37 }
38
39 func (s *server) ServeHTTP(w http.ResponseWriter, r *http.Request) {
40    s.router.ServeHTTP(w, r)
41 }
```

На листинге 3.4 представлена функция обработки запроса, аутентификацию пациента в системе. Для других ролей функция аутентификации имеет аналогичный вид.

Листинг 3.4 – Аутентификация пользователя

```
1 func (s *server) authenticatePatient(next http.Handler) http.Handler {
2     return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
3
4         session, err := s.sessionStore.Get(r, sessionName)
5
6         if err != nil {
7             s.error(w, r, http.StatusInternalServerError, err)
8             return
9         }
10
11         id, ok := session.Values["patient_id"]
12
13         if !ok {
14             s.error(w, r, http.StatusUnauthorized, errNotAuthenticated)
15             return
16         }
17
18         p, err := s.store.Patient().Find(id.(int))
19
20         if err != nil {
21             s.error(w, r, http.StatusUnauthorized, errNotAuthenticated)
22             return
23         }
24
25         next.ServeHTTP(w, r.WithContext(context.WithValue(r.Context(),
26                                                         ctxKeyUser, p)))
27     })
28 }
```

Запросы к базе данных

В данном разделе приведены примеры функций, осуществляющих запросы к базе данных. Для примера рассмотрим сущность пользователя с ролью «Пациент».

На листинге 3.5 представлена функция создания записи в базе данных о новом пациенте.

Листинг 3.5 – Функция создания записи

```
1 func (r *PatientRepository) Create(p *model.Patient) error {
2     if err := p.Validate(); err != nil {
3         return err
4     }
5
6     if err := p.BeforeCreate(); err != nil {
7         return err
8     }
9
10    return r.store.db.QueryRow(
11        "INSERT INTO patients (name, surname, birth_year, gender, phone,
12            email, encrypted_password)
13            VALUES ($1, $2, $3, $4, $5, $6, $7)
14            RETURNING id",
15        &p.Name,
16        &p.Surname,
17        &p.Birth_year,
18        &p.Gender,
19        &p.Phone,
20        &p.Email,
21        &p.EncryptedPassword,
22        ).Scan(&p.ID)
23 }
```

Шифрование данных

Для авторизации в системе необходимо ввести адрес электронной почты и пароль. Исходный пароль, вводимый пользователем, хранится в базе данных в зашифрованном виде. При регистрации перед вставкой соответствующей записи в таблицу происходит шифрование пароля.

На листинге 3.6 представлена функция, шифрующая введенный пароль нового пользователя. Для примера рассмотрена роль «Пациент», для остальных ролей приведенные ниже функции имеют аналогичный вид.

Листинг 3.6 – Функция, шифрующая пароль

```
1      func (p *Patient) BeforeCreate() error {
2          if len(p.Password) > 0 {
3              enc, err := encryptString(p.Password)
4              if err != nil {
5                  return err
6              }
7              p.EncryptedPassword = enc
8          }
9          return nil
10     }
```

На листинге 3.7 представлена функция, очищающая исходный пароль пользователя после успешного создания учетной записи в системе.

Листинг 3.7 – Функция, очищающая исходный пароль

```
1      func (p *Patient) Sanitize() {p.Password = ""}
```

На листинге 3.8 представлена функция, сравнивающая вводимый пользователем пароль при попытке авторизации в системе с паролем, хранящемся в зашифрованном виде в базе данных.

Листинг 3.8 – Функция, сравнивающая пароли

```
1      func (p *Patient) ComparePassword(password string) bool {
2          return bcrypt.CompareHashAndPassword(
3              []byte(p.EncryptedPassword), []byte(password)) == nil
4      }
```

На листинге 3.9 представлена функция шифрования строки.

Листинг 3.9 – Функция шифрования строки

```
1      func encryptString(s string) (string, error) {
2          b, err := bcrypt.GenerateFromPassword([]byte(s), bcrypt.MinCost)
3          if err != nil {
4              return "", err
5          }
6          return string(b), nil
7      }
```

3.3 Хранимая функция

На листинге 3.10 представлен сценарий создания хранимой функции для подсчета статистики о заболеваемости.

Листинг 3.10 – Сценарий создания хранимой функции

```
1 CREATE OR REPLACE FUNCTION percent(integer)
2 RETURNS TABLE
3 (
4     disease_id integer ,
5     percent     numeric
6 )
7 AS
8 $code$
9 WITH cte AS
10 (
11     SELECT r.disease_id , count(r.id)
12     FROM records AS r JOIN diseases AS d ON r.disease_id = d.id
13     WHERE d.spec_id = $1
14     GROUP BY r.disease_id
15 )
16 SELECT disease_id ,
17     round((count::float/(select count(id) from records)*100)::numeric,2)
18 FROM cte;
19 $code$
20 LANGUAGE SQL;
```

3.4 Тестирование

Тестирование разработанного приложения выполнялось с помощью пакета Testing[20].

На листингах 3.11-3.12 приведен пример тестирования функции запроса всех записей из таблицы приемов, которые закреплены за конкретным доктором. Суммарно было написано около 70 тестирующих функций.

Листинг 3.11 – Пример тестирования

```
1 func TestVisitRepository_GetAllVisitsByDoctor(t *testing.T) {
2     db, teardown := sqlstore.TestDB(t, databaseURL)
3     defer teardown("visits", "doctors", "patients")
4     s := sqlstore.New(db)
5
6     p1 := model.TestPatient(t)
7     s.Patient().Create(p1)
8     d1 := model.TestDoctor(t)
9     s.Doctor().Create(d1)
```

Листинг 3.12 – Пример тестирования (продолжение)

```
1      p2 := model.TestPatient(t)
2      p2.Email = "patient@mail.ru"
3      s.Patient().Create(p2)
4      d2 := model.TestDoctor(t)
5      d2.Email = "doctor@mail.ru"
6      s.Doctor().Create(d2)
7
8      v := model.TestVisit(t)
9      v.Doctor_id = d1.ID
10     v.Patient_id = p1.ID
11     s.Visit().Create(v)
12
13     v = model.TestVisit(t)
14     v.Doctor_id = d1.ID
15     v.Patient_id = p2.ID
16     s.Visit().Create(v)
17     s.Visit().CommitVisit(v.ID)
18     v, _ = s.Visit().Find(v.ID)
19
20     v = model.TestVisit(t)
21     v.Doctor_id = d2.ID
22     v.Patient_id = p2.ID
23     s.Visit().Create(v)
24
25     v = model.TestVisit(t)
26     v.Doctor_id = d2.ID
27     v.Patient_id = p2.ID
28     s.Visit().Create(v)
29
30     u, err := s.Visit().GetAllVisitsByDoctor(d1.ID)
31     assert.NoError(t, err)
32     assert.NotNil(t, u)
33     assert.Equal(t, len(u), 2)
34 }
```

Наполнение базы данных

База данных заполнялась с помощью написания скрипта на языке Golang с использованием пакета go-randomdata[21]. В результате было создано 1000 пациентов, 100 докторов, 10 администраторов. Данные о заболеваниях, специализациях докторов и медикаментах заполнялись вручную, т.к. эти данные носят специфический характер.

На листинге 3.13 приведен пример генерации данных для сущности «Администратор».

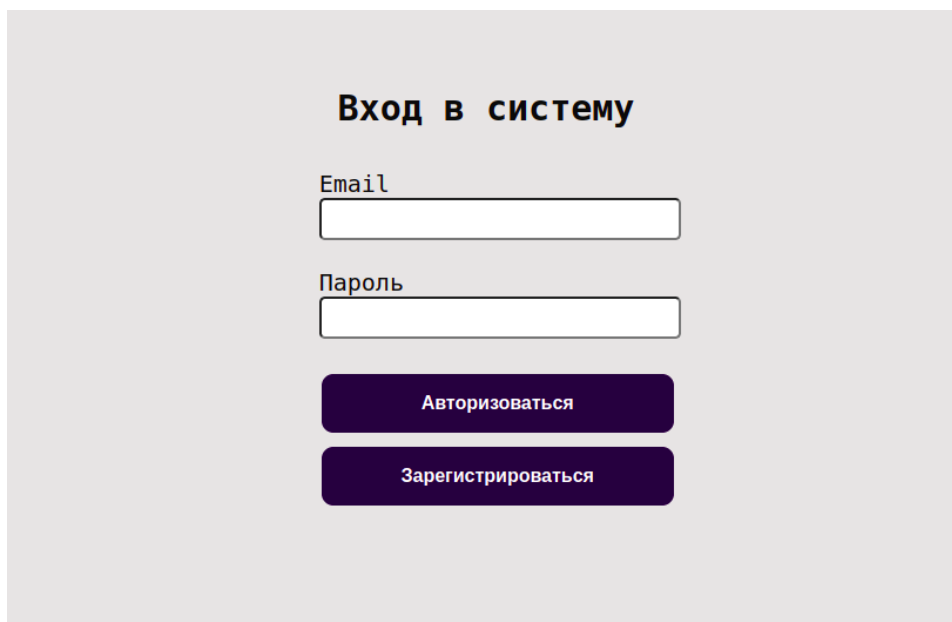
Листинг 3.13 – Пример генерации данных для администратора

```
1 package main
2
3 import (
4     "fmt"
5     "math/rand"
6     "os"
7     "strconv"
8     "github.com/Pallinder/go-randomdata"
9     "golang.org/x/crypto/bcrypt"
10 )
11
12 func encryptString(s string) (string, error) {
13     b, err := bcrypt.GenerateFromPassword([]byte(s), bcrypt.MinCost)
14     if err != nil { return "", err }
15     return string(b), nil
16 }
17
18 const del = ","
19
20 func main() {
21     file, err := os.Create("admins.txt")
22     if err != nil {
23         fmt.Println("Unable to create file:", err)
24         os.Exit(1)
25     }
26     defer file.Close()
27     password := "123456"
28     var text_gender string
29
30     for i := 0; i < n; i++ {
31         gender := rand.Intn(2) + 1
32         enc, err := encryptString(password)
33         if err != nil {
34             fmt.Println("Unable to encrypt", err)
35             os.Exit(1)
36         }
37         file.WriteString(randomdata.FirstName(gender) + del +
38             randomdata.LastName() +
39             del + randomdata.Email() + del + enc + "\n")
40     }
```

3.5 Описание презентационного уровня приложения

В данном разделе проведен краткий обзор пользовательского интерфейса приложения.

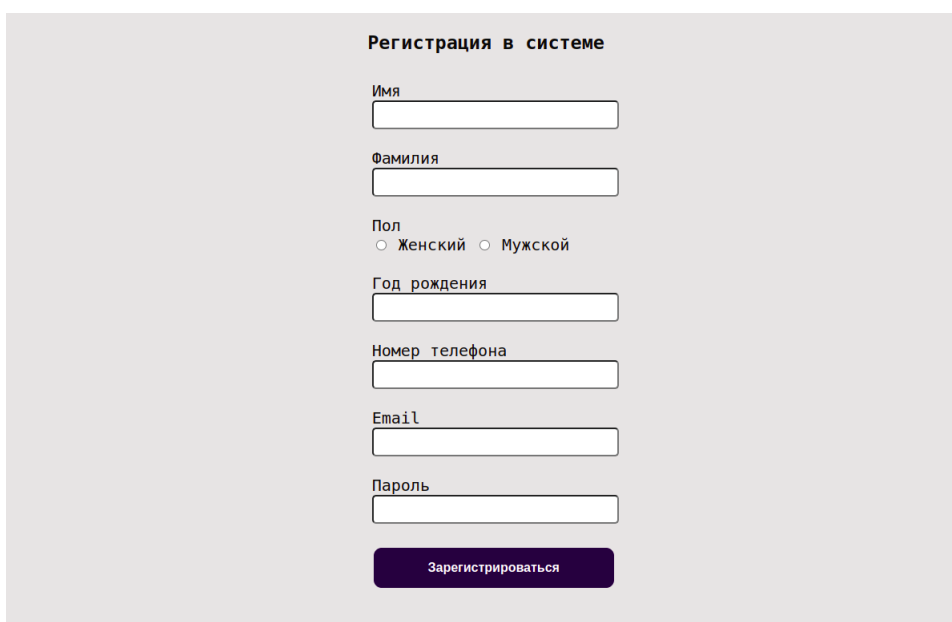
На рисунке 3.1 представлена стартовая страница веб-приложения.



The screenshot shows a login interface with a light gray background. At the top, the title "Вход в систему" is centered in bold black text. Below the title are two input fields: "Email" and "Пароль", each with a white border and a light gray background. Under the "Пароль" field is a small eye icon for toggling password visibility. Below the input fields are two dark purple buttons with white text: "Авторизоваться" and "Зарегистрироваться".

Рисунок 3.1 – Стартовая страница приложения

На рисунке 3.2 представлена форма регистрации для пациента. Формы регистрации, соответствующие ролям «Доктор» и «Администратор», будут иметь аналогичный вид.



The screenshot shows a registration form with a light gray background. The title "Регистрация в системе" is centered at the top in bold black text. The form contains several input fields: "Имя", "Фамилия", "Год рождения", "Номер телефона", "Email", and "Пароль". The "Пароль" field has a small eye icon for toggling password visibility. Between the "Фамилия" and "Год рождения" fields, there is a "Пол" section with two radio buttons: "Женский" and "Мужской". At the bottom of the form is a dark purple button with white text labeled "Зарегистрироваться".

Рисунок 3.2 – Форма регистрации для пациента

Для регистрации администратора нужно перейти по специальной ссылке: `admin_start`. Зарегистрировать доктора может только администратор.

На рисунке 3.3 представлена страница, демонстрирующая меню авторизованного пользователя с ролью «Пациент».

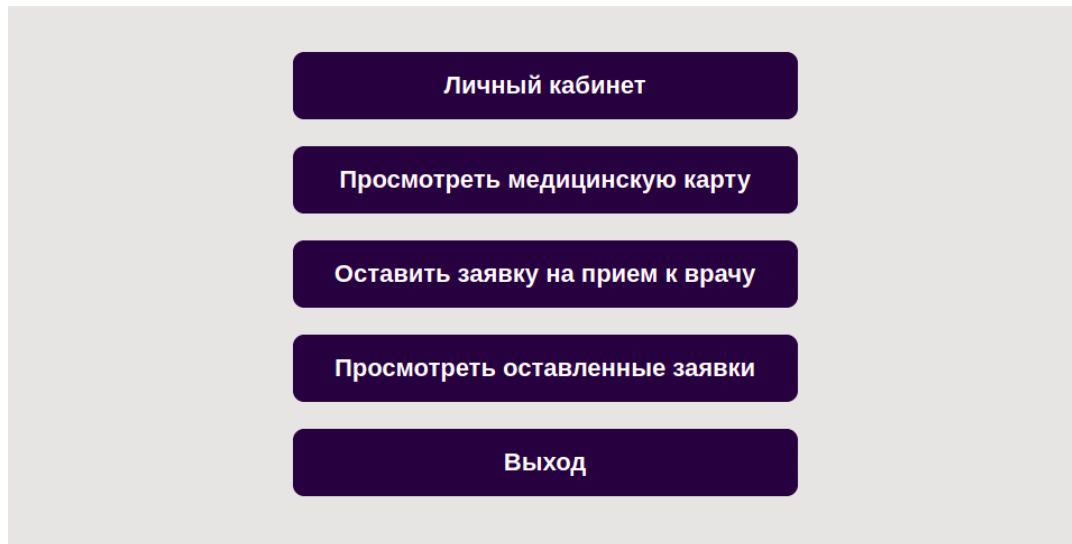


Рисунок 3.3 – Главная страница пациента

На рисунке 3.4 представлена страница, демонстрирующая меню авторизованного пользователя с ролью «Администратор».

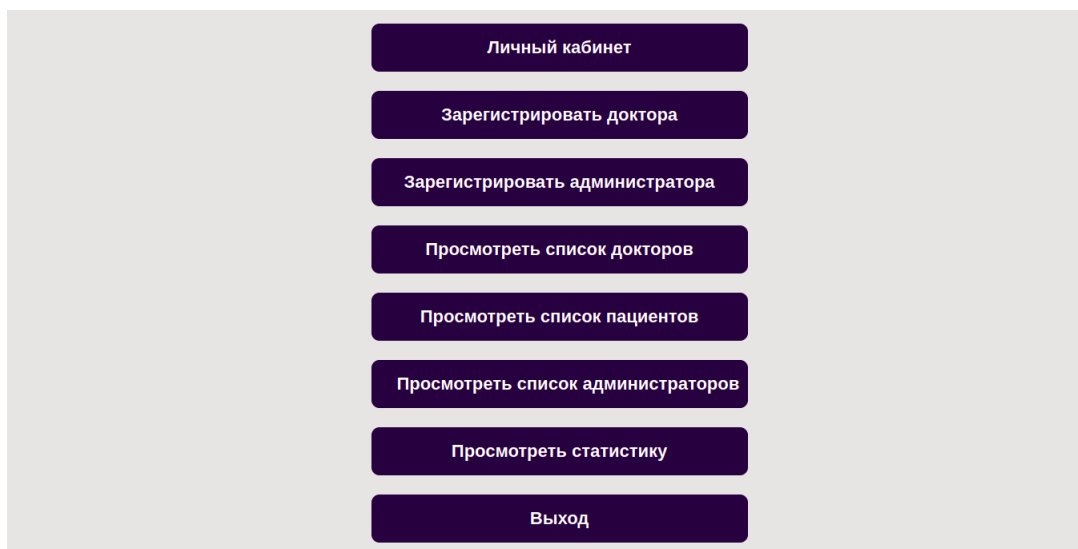


Рисунок 3.4 – Главная страница администратора

На рисунке 3.5 представлена страница, демонстрирующая меню авторизованного пользователя с ролью «Доктор».

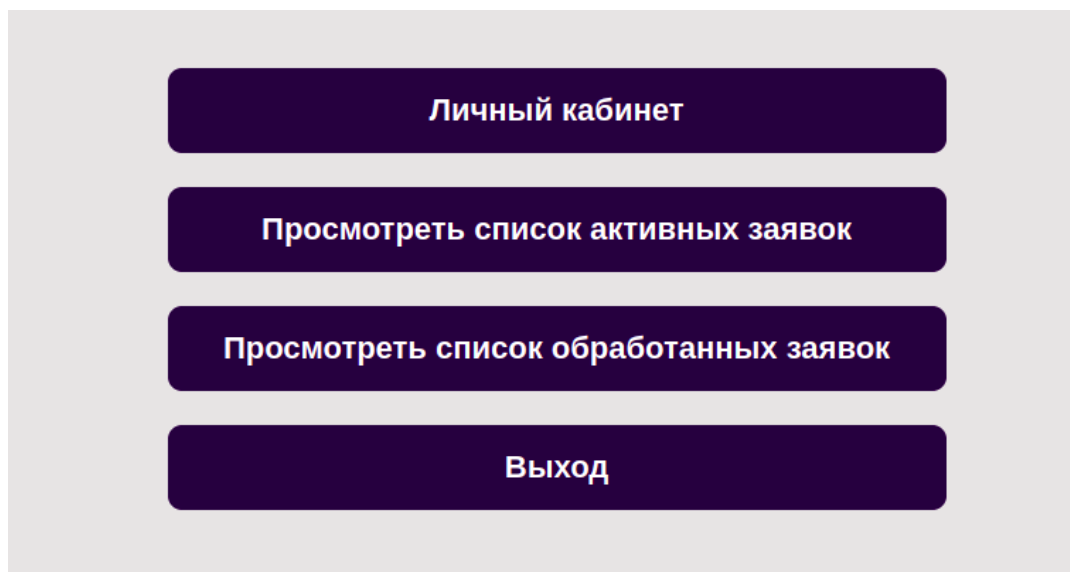


Рисунок 3.5 – Главная страница доктора

На рисунке 3.6 представлена страница, демонстрирующая личный кабинет пользователя с ролью «Пациент».

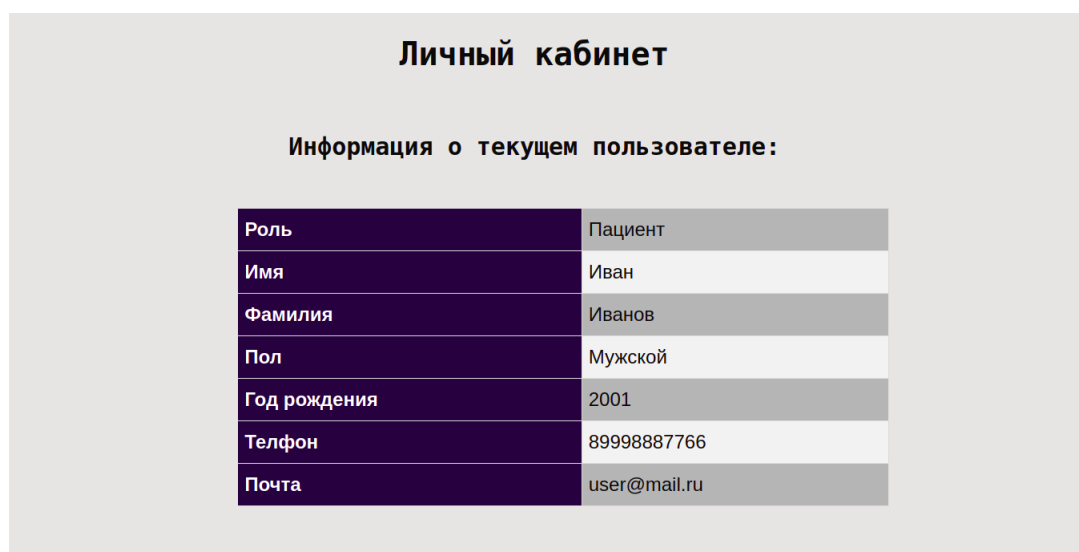


Рисунок 3.6 – Личный кабинет пациента

Для пользователей с ролями «Доктор» и «Администратор» личный кабинет имеет аналогичный вид.

На рисунке 3.7 представлена страница, демонстрирующая медицинскую карту пациента.

Медицинская карта пациента:				
№ заявки	Доктор	Специализация	Диагноз	Препарат
1	Александр Александрович	Психиатр	Невроз	Лазолван
5	Федор Федорович	Отоларинголог	Гайморит	Анальгин
4	Петр Петрович	Хирург	Перелом	Аспирин
8	Александр Александрович	Психиатр	Депрессия	Нурофен
7	Иван Иванов	Онколог	Рак легких	Ношпа
6	Тимофей Тимофей	Кардиолог	Ишемия	Нурофен
3	Василий Васильев	Терапевт	ОРВИ	Ношпа
9	Василий Васильев	Терапевт	ОРВИ	Ношпа

Рисунок 3.7 – Медицинская карта пациента

На рисунке 3.8 представлена страница, демонстрирующая форму заявки на прием к доктору.

Список доступных врачей:

№	Имя	Фамилия	Год начала практики	Специализация
1	Василий	Васильев	2000	Терапевт
2	Петр	Петров	2001	Хирург
3	Иван	Иванов	1999	Отоларинголог
4	Федор	Федоров	2000	Кардиолог

Выберите номер доктора, подходящего Вам:

3 ▾

Оставить заявку

Вернуться назад

Рисунок 3.8 – Форма заявки на прием к доктору

На рисунке 3.9 представлена страница, демонстрирующая список докторов в системе, доступный только администратору. Страница просмотра списка администраторов и пациентов имеет аналогичный вид.

Список докторов, зарегистрированных в системе:

Имя	Фамилия	Обработанные заявки	Личный кабинет	Год начала практики	Специализация	Зарплата	Email
Иван	Иванов	Ссылка	Ссылка	2000	Терапевт	15000	ivan@mail.ru
Федор	Федоров	Ссылка	Ссылка	2005	Кардиолог	30000	fedor@mail.ru
Василий	Васильев	Ссылка	Ссылка	2001	Психиатр	40000	vasily@mail.ru
Петр	Петров	Ссылка	Ссылка	2002	Офтальмолог	45000	petr@mail.ru
Анна	Федорова	Ссылка	Ссылка	1999	Хирург	20000	anna@mail.ru
Алексей	Алексеев	Ссылка	Ссылка	1998	Кардиолог	30000	alexey@mail.ru

[Вернуться на главную страницу](#)

Рисунок 3.9 – Просмотр списка докторов с системе

На рисунке 3.10 представлена страница, демонстрирующая список активных заявок доктора.

Список активных заявок:

№ заявки	Имя	Фамилия	Мед. карта	Пол	Год рождения	Телефон
3	Анна	Федорова	Ссылка	Женский	2003	89998887766
4	Марго	Иванова	Ссылка	Женский	2005	89998887766
5	Семен	Тимофеев	Ссылка	Мужской	1999	89998887766

Выберите номер заявки, которую Вы хотите зафиксировать:

4

Выберите диагноз:

ОРВИ

Выберите препарат:

Аспирин

Рисунок 3.10 – Просмотр списка активных заявок доктора

На рисунке 3.11 представлена страница, демонстрирующая список обработанных заявок доктора.

Список обработанных заявок:

№ заявки	Имя	Фамилия	Пол	Год рождения	Телефон	Диагноз	Препарат
4	Марго	Иванова	Женский	2005	89998887766	ОРВИ	Аспирин
2	Полина	Сироткина	Женский	2001	89049598821	ОРВИ	Аскорбиновая кислота
3	Анна	Федорова	Женский	2003	89998887766	ОРЗ	Анальгин
5	Семен	Тимофеев	Мужской	1999	89998887766	ОРВИ	Нурофен

[Вернуться на главную](#)

Рисунок 3.11 – Просмотр списка обработанных заявок доктора

На рисунке 3.12 представлена страница, демонстрирующая статистику заболеваемости.

Специализация	Болезнь	Заболеваемость, %
Терапевт	ОРВИ	22.22%
Хирург	Перелом	11.11%
Отоларинголог	Гайморит	11.11%
Кардиолог	Ишемия	11.11%
Онколог	Рак легких	11.11%

[Выход](#)

Рисунок 3.12 – Просмотр статистики заболеваемости

Вывод

В данном разделе был обоснован выбор средств разработки приложения, представлены детали разработки и дано описание презентационного уровня приложения. Также проведено тестирование приложения.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была достигнута поставленная цель: реализована база данных, а также соответствующее приложение для хранения и анализа данных медицинской компании.

Для достижения цели были выполнены следующие задачи:

- формализована поставленная задача;
- описана структура базы данных;
- рассмотрены модели данных и выбрана реляционная модель;
- спроектирована база данных, реализованы соответствующие запросы;
- реализован интерфейс для доступа к базе данных;
- реализовано программное обеспечение для работы с базой данных.

В процессе выполнения курсовой работы были получены знания в области медицины и здравоохранения, проектирования баз данных и разработки соответствующих веб-серверов. Был получен опыт работы с соответствующими инструментами разработки: PostgreSQL, Golang и язык разметки HTML.

В качестве перспектив разработки программного обеспечения можно рассмотреть такие задачи, как масштабирование существующей реализации для достаточно большого количества пользователей и повышение уровня выдерживаемой нагрузки и безопасности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Гордиенко Е. П., Паненко Н. С.* Современные технологии обработки и анализа больших данных в научных исследованиях. — Филиал федерального государственного бюджетного образовательного учреждения высшего образования «Ростовский государственный университет путей сообщения» в г. Воронеж, 2018. — С. 44—48.
2. *Буданова А. С.* Понятие и назначение медицинской информационной системы. — Финансовый университет при Правительстве Российской Федерации, г. Москва, 2016. — С. 40—41.
3. О проектировании медицинских баз данных и информационных систем для организации и управления лечебно-диагностических процессов / А. А. Абдуманов [и др.]. — Телекоммуникации и транспорт, 2016. — С. 45—53.
4. *Соколова А. А.* Медицинские информационные системы в работе медицинского персонала. — Вестник магистратуры, 2021.
5. *Пин-Шен Ч. П.* Модель «сущность-связь» - шаг к единому представлению данных. — Системы управления базами данных, 1995.
6. *Заушина А. С., Жуковская А. Н.* Основные функции и типовая организация СУБД. — Сибирский государственный аэрокосмический университет имени академика М.Ф. Решетнева, 2021. — С. 177—179.
7. Постреляционная, многомерная и объектно-ориентированная модели данных. — URL: http://bseu.by/it/tohod/lekcii2_4.htm.
8. Многомерная модель данных. — URL: https://studme.org/117583/informatika/mnogomernaya_model_dannyh.
9. Медицинские информационные системы в работе медицинского персонала. — URL: http://bseu.by/it/tohod/lekcii9_2.htm.
10. *Кучкин В. П.* Методы защиты баз данных. — Военная академия Ракетных войск стратегического назначения им. Петра Великого, г. Серпухов, Московская область, 2021.
11. Администрирование баз данных. — URL: http://bseu.by/it/tohod/lekcii9_2.htm.

12. PostgreSQL Documentation. — URL: <https://www.postgresql.org/docs/>.
13. Golang Documentation. — URL: <https://go.dev/doc/>.
14. Net/Http package documentation. — URL: <https://pkg.go.dev/net/http>.
15. Sessions package documentation. — URL: <https://pkg.go.dev/github.com/gorilla/sessions>.
16. Lib/Pq package documentation. — URL: <https://pkg.go.dev/github.com/lib/pq>.
17. X/Crypto package documentation. — URL: <https://pkg.go.dev/golang.org/x/crypto>.
18. Validation package documentation. — URL: <https://pkg.go.dev/github.com/go-ozzo/ozzo-validation/v4>.
19. HTML documentation. — URL: <https://devdocs.io/html/>.
20. Testing package documentation. — URL: <https://pkg.go.dev/testing>.
21. Randomdata package documentation. — Дата обновления: 09.05.2022. — URL: <https://pkg.go.dev/github.com/pallinder/go-randomdata>.

ПРИЛОЖЕНИЕ А

Сценарий создания базы данных

На листингах А.1-А.2 представлен сценарий создания базы данных.

Листинг А.1 – Сценарий создания базы данных

```
1 CREATE TABLE patients (
2     id                serial    not null primary key,
3     name              varchar   not null ,
4     surname           varchar   not null ,
5     gender            varchar   not null ,
6     birth_year        integer   not null ,
7     phone             varchar   not null ,
8     email             varchar   not null unique ,
9     encrypted_password varchar   not null
10 );
11
12 CREATE TABLE specializations (
13     id      serial not null primary key,
14     name    varchar not null ,
15     salary  integer not null
16 );
17
18 CREATE TABLE diseases (
19     id      serial not null primary key,
20     name    varchar not null ,
21     spec_id integer  references specializations (id)
22 );
23
24 CREATE TABLE medicines (
25     id      serial not null primary key,
26     name    varchar not null ,
27     cost    integer not null
28 );
29
30 CREATE TABLE doctors (
31     id                serial    not null primary key,
32     name              varchar   not null ,
33     surname           varchar   not null ,
34     work_since        integer   not null ,
35     spec_id           integer   references specializations (id),
36     email             varchar   not null unique ,
37     encrypted_password varchar   not null
38 );
```

Листинг А.2 – Сценарий создания базы данных(продолжение)

```
1 CREATE TABLE visits (
2     id          serial not null primary key,
3     status      varchar not null,
4     patient_id  integer references patients (id),
5     doctor_id   integer references doctors (id)
6 );
7
8 CREATE TABLE records (
9     id          serial not null primary key,
10    visit_id     integer references visits (id),
11    disease_id   integer references diseases (id),
12    medicine_id  integer references medicines (id)
13 );
14
15 CREATE TABLE admins (
16     id          serial not null primary key,
17     name        varchar not null,
18     surname     varchar not null,
19     email       varchar not null unique,
20     encrypted_password varchar not null
21 );
22
23 CREATE TABLE ops_stat
24 (
25     operation char(1) not null,
26     date timestamp not null,
27     user_id text not null
28 );
```

На листингах А.3-А.4 представлен сценарий создания ролей и выделения прав.

Листинг А.3 – Создание ролей и выделение прав

```
1 create role patient with login password '111';
2 grant select, insert on table patients to patient;
3 grant select, insert on table visits to patient;
4 grant select on table records to patient;
5
6 create role doctor with login password '222';
7 grant select, insert on table doctors to doctor;
8 grant select on table medicines to doctor;
9 grant select on table diseases to doctor;
10 grant select, update on table visits to doctor;
11 grant select, insert on table records to doctor;
```

Листинг А.4 – Создание ролей и выделение прав(продолжение)

```
1 create role admin with login password '333';
2 grant select , insert on table admins to admin;
3 grant select on table patient to admin;
4 grant select on table medicines to doctor;
5 grant select on table diseases to doctor;
6 grant select , insert on table doctors to admin;
7 grant select , insert on table records to admin;
```


На листинге А.5 представлен сценарий создания логирующего триггера.

Листинг А.5 – Сценарий создания логирующего триггера

```
1 CREATE OR REPLACE FUNCTION logging()
2 RETURNS TRIGGER
3 AS
4 $code$
5     BEGIN
6     IF (TG_OP = 'INSERT') THEN
7         INSERT INTO ops_stat SELECT 'I', now(), user;
8     ELSIF (TG_OP = 'DELETE') THEN
9         INSERT INTO ops_stat SELECT 'S', now(), user;
10    ELSIF (TG_OP = 'UPDATE') THEN
11        INSERT INTO ops_stat SELECT 'S', now(), user;
12    END IF;
13    RETURN NULL;
14    END;
15 $code$
16 LANGUAGE PLPGSQL;
17
18 CREATE TRIGGER log AFTER INSERT OR DELETE OR UPDATE ON patients
19 FOR ROW EXECUTE FUNCTION logging();
20
21 CREATE TRIGGER log AFTER INSERT OR DELETE OR UPDATE ON doctors
22 FOR ROW EXECUTE FUNCTION logging();
23
24 CREATE TRIGGER log AFTER INSERT OR DELETE OR UPDATE ON admins
25 FOR ROW EXECUTE FUNCTION logging();
26
27 CREATE TRIGGER log AFTER INSERT OR DELETE OR UPDATE ON diseases
28 FOR ROW EXECUTE FUNCTION logging();
29
30 CREATE TRIGGER log AFTER INSERT OR DELETE OR UPDATE ON specializations
31 FOR ROW EXECUTE FUNCTION logging();
32
33 CREATE TRIGGER log AFTER INSERT OR DELETE OR UPDATE ON medicines
34 FOR ROW EXECUTE FUNCTION logging();
35
36 CREATE TRIGGER log AFTER INSERT OR DELETE OR UPDATE ON visits
37 FOR ROW EXECUTE FUNCTION logging();
38
39 CREATE TRIGGER log AFTER INSERT OR DELETE OR UPDATE ON records
40 FOR ROW EXECUTE FUNCTION logging();
```

ПРИЛОЖЕНИЕ Б

Запросы к базы данных

На листингах Б.1-Б.2 представлены запросы, реализованные в процессе выполнения курсовой работы.

Листинг Б.1 – Запросы к базе данных

```
1  INSERT INTO admins (name, surname, email, encrypted_password)
2  VALUES ($1, $2, $3, $4) RETURNING id
3
4  SELECT id, name, surname, email, encrypted_password
5  FROM admins WHERE id = $1
6
7  SELECT id, name, surname, email, encrypted_password
8  FROM admins WHERE email = $1
9
10 SELECT id, name, surname, email, encrypted_password FROM admins
11
12 INSERT INTO doctors (name, surname, work_since, spec_id, email,
13 encrypted_password) VALUES ($1, $2, $3, $4, $5, $6) RETURNING id
14
15 SELECT id, name, surname, work_since, spec_id, email, encrypted_password
16 FROM doctors WHERE id = $1
17
18 SELECT id, name, surname, work_since, spec_id, email, encrypted_password
19 FROM doctors WHERE email = $1
20
21 SELECT id, name, surname, work_since, spec_id, email, encrypted_password
22 FROM doctors
23
24 INSERT INTO patients (name, surname, birth_year, gender, phone, email,
25 encrypted_password) VALUES ($1, $2, $3, $4, $5, $6, $7) RETURNING id
26
27 SELECT id, name, surname, birth_year, gender, phone, email,
28 encrypted_password FROM patients WHERE id = $1
29
30 SELECT id, name, surname, birth_year, gender, phone, email,
31 encrypted_password FROM patients WHERE email = $1
32
33 SELECT id, name, surname, gender, birth_year, phone, email,
34 encrypted_password FROM patients
35
36 SELECT disease_id, percent FROM percent($1)
37
38 SELECT name, spec_id FROM diseases WHERE id = $1
```

Листинг Б.2 – Запросы к базе данных(продолжение)

```
1 SELECT id, name, spec_id FROM diseases
2
3 SELECT id, name FROM diseases WHERE spec_id = $1
4
5 SELECT name, cost FROM medicines WHERE id = $1
6
7 SELECT id, name, cost FROM medicines
8
9 INSERT INTO records (visit_id, disease_id, medicine_id)
10 VALUES ($1, $2, $3) RETURNING id
11
12 SELECT records.id, records.visit_id, records.disease_id,
13 records.medicine_id FROM records JOIN visits ON records.visit_id=visits.id
14 WHERE visits.patient_id = $1
15
16 SELECT records.id, records.visit_id, records.disease_id,
17 records.medicine_id FROM records JOIN visits ON records.visit_id=visits.id
18 WHERE visits.doctor_id = $1
19
20 SELECT id, name, salary FROM specializations WHERE id = $1
21
22 SELECT id, name, salary FROM specializations WHERE name = $1
23
24 SELECT id, name, salary FROM specializations
25
26 SELECT id, status, patient_id, doctor_id FROM visits WHERE id = $1
27
28 INSERT INTO visits (status, patient_id, doctor_id)
29 VALUES ('Active', $1, $2) RETURNING id
30
31 SELECT id, patient_id FROM visits WHERE doctor_id=$1 AND status = 'Active'
32
33 SELECT id, patient_id FROM visits WHERE doctor_id=$1 AND status = 'Done'
34
35 SELECT id, patient_id FROM visits WHERE doctor_id=$1
36
37 SELECT id, doctor_id FROM visits WHERE patient_id=$1 AND status = 'Active'
38
39 UPDATE visits SET status = 'Done' WHERE id = $1;
```

ПРИЛОЖЕНИЕ В

Детали реализации

На листингах В.1-В.2 представлена функция, настраивающая роутер сервера.

Листинг В.1 – Настройка роутера

```
1 func (s *server) configureRouter() {
2     s.router.Use(s.setRequestID)
3     s.router.Use(s.logRequest)
4     s.router.Use(handlers.CORS(handlers.AllowedOrigins([] string {"*"})))
5
6     //general routes
7     s.router.HandleFunc("/start", s.handleStart())
8     s.router.HandleFunc("/login", s.handleLogin())
9     s.router.HandleFunc("/after", s.handleAfter())
10
11    //patient routes
12    s.router.HandleFunc("/create_patient", s.handlePatientCreate())
13    s.router.HandleFunc("/commit_create", s.handlePatientCommitCreate())
14
15    //patient private routes
16    patient_root := s.router.PathPrefix("/patient").Subrouter()
17    patient_root.Use(s.authenticatePatient)
18    patient_root.HandleFunc("/main", s.handlePatientMainPage())
19    patient_root.HandleFunc("/cabinet", s.handlePatientCabinet())
20    patient_root.HandleFunc("/create_visit", s.handlePatientCreateVisit())
21    patient_root.HandleFunc("/commit_visit",
22                                s.handlePatientCommitCreateVisit())
23    patient_root.HandleFunc("/record", s.handlePatientRecord())
24    patient_root.HandleFunc("/show_active_visits",
25                                s.handlePatientShowActiveVisits())
26
27    //doctor private routes
28    doctor_root := s.router.PathPrefix("/doctor").Subrouter()
29    doctor_root.Use(s.authenticateDoctor)
30    doctor_root.HandleFunc("/cabinet", s.handleDoctorCabinet())
31    doctor_root.HandleFunc("/main", s.handleDoctorMainPage())
32    doctor_root.HandleFunc("/show_active_visits",
33                                s.handleDoctorShowActiveVisits())
34    doctor_root.HandleFunc("/show_done_visits",
35                                s.handleDoctorShowDoneVisits())
36    doctor_root.HandleFunc("/commit_visit", s.handleDoctorCommitVisit())
37    doctor_root.HandleFunc("/record", s.handleDoctorRecord())
```

Листинг В.2 – Настройка роутера(продолжение)

```
1      //admin private routes
2      admin_root := s.router.PathPrefix("/admin").Subrouter()
3      admin_root.Use(s.authenticateAdmin)
4
5      admin_root.HandleFunc("/main", s.handleAdminMainPage())
6      admin_root.HandleFunc("/cabinet", s.handleAdminCabinet())
7      admin_root.HandleFunc("/create_doctor", s.handleAdminCreateDoctor())
8      admin_root.HandleFunc("/commit_doctor",
9                              s.handleAdminCommitCreateDoctor())
10     admin_root.HandleFunc("/create_admin", s.handleAdminCreateAdmin())
11     admin_root.HandleFunc("/commit_admin",
12                             s.handleAdminCommitCreateAdmin())
13     admin_root.HandleFunc("/show_doctors", s.handleAdminShowDoctors())
14     admin_root.HandleFunc("/show_patients", s.handleAdminShowPatients())
15     admin_root.HandleFunc("/show_stat", s.handleAdminShowStat())
16     admin_root.HandleFunc("/show_admins", s.handleAdminShowAdmins())
17     admin_root.HandleFunc("/doctor_done_visits",
18                             s.handleAdminShowDoctorDoneVisits())
19     admin_root.HandleFunc("/doctor_cabinet",
20                             s.handleAdminShowDoctorCabinet())
21
22     //admin general routes
23     s.router.HandleFunc("/admin_start", s.handleAdminStart())
24     s.router.HandleFunc("/admin_pass", s.handleAdminGeneralPass())
25
26     //admin general private routes
27     admin_general_root:=s.router.PathPrefix("/admin_general").Subrouter()
28     admin_general_root.Use(s.authenticateAdminGeneral)
29     admin_general_root.HandleFunc("/admin_create",
30                                   s.handleAdminGeneralCreate())
31     admin_general_root.HandleFunc("/commit_create",
32                                   s.handleAdminGeneralCommitCreate())
33     admin_general_root.HandleFunc("/admin_login",
34                                   s.handleAdminGeneralLogin())
35     admin_general_root.HandleFunc("/commit_login",
36                                   s.handleAdminGeneralCommitLogin())
37 }
```