СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

БД – База данных

Кэш – промежуточный буфер с быстрым доступом, содержащий информацию, которая может быть запрошена с наибольшей вероятностью.

ПО – Программное обеспечение

ОС – Операционная система

СУБД – Система управления базами данных

ORM – Object-Relational Mapping – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных»

ООП – Объектно-ориентированное программирование - методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования

JSON – JavaScript Object Notation – простой формат обмена данными, удобный для чтения и написания

SQL – Structured Query Language – «структурированный язык запросов»

JS – язык программирования JavaScript

СУ – Система управления

АСУ – Автоматизированная система управления

ВВЕДЕНИЕ

Разрабатываемая система предназначена для использования в медицинских учреждениях типа районной поликлиники, городской больницы или иных медицинских организациях общелечебного назначения. Система не претендует на полноценную замену приема у личного врача, а скорее обеспечивает информационную поддержку во всех отношениях, возникающих между пациентом и представителями медицинских учреждений. Разрабатываемая система улучшает способы структурирования, хранения, управления и доставки медицинской информации, предоставляя персонализированную, безопасную и легкодоступную среду взаимодействия для всех участников и заинтересованных сторон органов здравоохранения.

Объективная сложность информационных объектов в рамках здравоохранения и необходимость интерактивного взаимодействия обуславливают те технологии, бизнес-модели, среды разработка и реализации, которые далее будут рассмотрены более подробно. В частности, приложение основано на использовании реляционных баз данных и веб-сервисов ASP.NET. Основной проблемой разработки является создание гибкой модели базы данных, которая отражает потребности предметной области здравоохранения и его многочисленных участников. Эта задача требует четкого видения моделируемых объектов и того, как система будет предлагать информационно-алгоритмические решения для заинтересованных сторон. Также необходимо учитывать, что решение ряда проблем требуют возможности управления и передачи данных в режиме реального времени. Веб-службы используются для облегчения взаимодействия с базами данных и интеграции распределенных источников. Эти службы могут легко ссылаться и вызываться из различных типов клиентских приложений, поощряя повторное использование кода и обеспечивая доступ к данным с различными уровнями доступа и представления.

Пояснительная записка состоит из семи разделов.

В первом разделе производится анализ отрасли здравоохранения.

Во втором разделе рассматриваются используемые технологии для разработки системы.

Третий раздел посвящён проектированию системы.

Четвёртый раздел посвящён разработке базы данных.

Пятый раздел посвящён разработке веб-сервисов.

Шестой раздел посвящён разработке веб-приложений.

В седьмом разделе приводится технико-экономическое обоснование проекта.

Дипломный проект выполнен самостоятельно, проверен в системе «Антиплагиат»

Процент оригинальности соответствует норме, установленной кафедрой систем управления. Цитирования обозначены ссылками на публикации, указанные в «Списке использованных источников».

1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

Информационные технологии в медицине и здравоохранении помогают решить следующие задачи:

- вести учет пациентов клиник;
- наблюдать дистанционно за их состоянием;
- сохранять и передавать результаты диагностических обследований;
- контролировать правильность назначенного лечения;
- проводить удаленную консультацию с пациентом;
- давать консультации малоопытным сотрудникам.

Информационные технологии в медицине дают возможность проводить качественное наблюдение за состоянием пациентов. Ведение электронных медицинских карт позволяет сократить время сотрудников клиник, потраченное на оформление различных бланков. Вся информация о пациенте представлена в одном документе, доступном медицинскому персоналу учреждения. Все данные об обследованиях и результатах процедур также вводятся непосредственно в электронную медицинскую карту. Это дает возможность другим специалистам оценить качество назначенного лечения, обнаружить неточности диагностики.

Применение ИТ в медицине позволяет врачам проводить онлайн-консультации в любое удобное время. При этом повышается доступность медицинских услуг. Люди могут получить квалифицированную помощь от опытных врачей удаленно. Это особенно необходимо людям:

- проживающим в географически удаленных районах;
- с ограниченными физическими возможностями;
- попавшим в чрезвычайную ситуацию.

Таким образом, пациентам или докторам не нужно преодолевать большие расстояния, чтобы получить консультацию. Врач может с помощью современных информационных технологий оценить состояние пациента, провести его осмотр и ознакомиться со всеми результатами его обследований.

Такие консультации необходимы не только пациентам с физиологическими проблемами. Беседы также позволяют людям, которые нуждаются в психиатрической или психологической помощи. Аудиовизуальное общение позволяет наладить контакт врачу с пациентом и оказать ему необходимую поддержку.

Экспоненциально растущая индустрия здравоохранения демонстрирует значительные тенденции, которые изменяют будущее отрасли. Ключевые тенденции включают растущую волну расширения прав и возможностей пациентов, а также внедрение современных информационных технологий (ИТ), которые улучшают диагностику и лечение. Пациенты желают более активно участвовать в медицинских взаимодействиях и решениях. Таким образом, они активно ищут в Интернете информацию, связанную со здоровьем. Кроме того, пациенты очень восприимчивы к новым технологиям, которые облегчают взаимодействие с профессионалами здравоохранения. Эта тенденция способствует внедрению такой веб-системы, такой как и разрабатываемая[1].

Существующие веб-службы также решают проблему расширения полномочий пациентов. Тем не менее, Интернет полон ненадежных источников, которые пытаются заменить личного врача дистанционными онлайн-советами. Эта тенденция, безусловно, неприемлема, поскольку ничто не должно подменять богатство взаимоотношений между пациентом и врачом. Ценная система не должна пытаться подменить отношения, но должна поощрять их[2].

Некоторым традиционным и современным участникам здравоохранения удалось добиться снижения затрат и улучшения качества медицинской помощи. Любое решение, которое пытается предложить полную платформу возможностей, должно успешно справляться со сложностью практики здравоохранения и последних тенденциями.

Система предлагает всестороннее представление о практике здравоохранения ключевым заинтересованным сторонам. Это улучшает способ структурирования медицинской информации, ее хранения, управления и доставки. Кроме того, она поощряет санитарное просвещение и более активное взаимодействие между пациентами и лицами, осуществляющими уход. В частности, система предлагает персонализированные веб-порталы для укрепления взаимоотношений между пациентом и врачом. Это достигается за счет обеспечения безопасной, легкодоступной и богатой данными среды связи. Персонализированные порталы позволяют заинтересованным сторонам легко управлять информацией, необходимой им для выполнения своих ролей. При этом каждый пользователь видит определённый фрагмент информации в зависимости от своей роли. Роль пользователя определяется посредством процесса входа в систему.

Вся информация, доступная через систему, хранится и извлекается с серверов в режиме реального времени. Это означает, что любые изменения моментально отражаются в базе данных. Таким образом, все пользователи могут получать самую свежую и точную информацию, доступную в любое время без задержек. Самое главное, что этот процесс не связан с поддержкой определённой платформы, и пользователи могут получить доступ к приложению с любого устройства, поддерживающего Веб, включая беспроводные КПК и мобильные телефоны[3].

Через свои персональные порталы различные группы пользователей могут выполнять различные действия, предназначенные для их конкретных ролей. Например, пациенты могут электронным образом осуществлять:

– ввод и отслеживание различных измерений состояния здоровья;
– просмотр информации о выписанных рецептах;
– просмотр личных медицинских / лабораторных тестов;
– заказывать встречи с первичным врачом;
– отправку писем врачу;
– просмотр историю болезни;
– обновление контактной информации;
– получение предупреждений, если состояния определённого параметра здоровья превышает установленное критическое значение.

В случае с врачами они могут электронным образом:

– получать доступ к записям пациентов, в том числе к информации от других врачей, если у них есть разрешение;
– выполнять заказ рецептов, лабораторных и медицинских тестов через простой интерфейс, который подтверждает подачу заявки;
– получать оповещения о значительных изменениях в жизненно важных показателях пациента и других измерениях здоровья;
– управлять встречами с пациентами.

Из своего дома или откуда-либо еще с помощью беспроводного устройства пациенты и врачи могут получить доступ к системе. Мгновенный доступ к информации из любой точки исключает время поиска, налагаемое традиционными бумажными документами. Электронные записи предотвращают ошибки при расшифровке, повышают эффективность и улучшают рабочий процесс. Кроме того, система обеспечивает точную и актуальную картину состояния каждого пациента и уровня риска.

Предоставляя пациентам возможность непосредственно вводить данные о состоянии здоровья в систему, записи пациентов содержат информацию, которая, возможно, не была собрана иным образом. Эти преимущества позволяют врачам своевременно предлагать медицинские услуги.

Поддерживая гибкость, необходимую в мире здравоохранения, система помогает стандартизировать практику медицины, предоставляя общую среду для организации и передачи данных в том же формате. Благодаря функциональным возможностям, предлагаемых системой, учреждения здравоохранения могут, бесспорно, ощущать определённые преимущества, снижения затрат и более высокой уровень удовлетворенности пациентов.

2 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

Создание подобной системы возможно благодаря достижениям в области интеграционных технологий для распределенных систем. Платформа .NET является примером таких достижений. Она обеспечивает бесшовную интеграцию многочисленных языков программирования, классов, фреймов и протоколов, ориентированных на интернет, включая C#, XML, ADO.NET, Web Services, SOAP, ASP.NET и WSDL. Эти технологии обсуждаются в следующих разделах.

2.1 Фреймворк

Платформа Microsoft .NET - это инновационная среда разработки программного обеспечения. Она предоставляет новый интерфейс прикладного программирования для создания, развертывания и запуска масштабируемых приложений и сервисов, ориентированных на любой браузер или устройство. Платформа включает в себя одну из самых больших доступных библиотек классов, известную как библиотека классов Framework. Библиотека помогает создавать мощные приложения для настольных компьютеров, клиент / сервер и веб-сервисы. Visual Studio .NET является центральным инструментом платформы, который предлагает среду разработки и интерфейс программирования, в которых может быть создано такое разнообразие приложений[4].

2.2 Объектно-ориентированный язык программирования

C# - уже не новый объектно-ориентированный язык, разработанный Microsoft для облегчения ориентированных на Интернет высокопроизводительных распределенных приложений .NET. Он был построен на других языках, таких как C ++ и Java, поэтому имеет очевидное сходство с этими языками. C# также предоставляет среду разработки Rapid Application Development, похожую на Visual Basic, способствующую более быстрой реализации. .NET Framework поддерживает множество существующих и новых языков программирования. Тем не менее, язык C#, вероятно, является лучшим выбором для разработок .NET. C# был логичным выбором для разработки данной системы, поскольку он должен был быть полностью разработан в среде .NET.

2.3 Расширяемый язык разметки

Расширяемый язык разметки (XML) находится в центре новых технологий баз данных .NET и веб-служб, поэтому знание XML необходимо для развития системы. XML уже давно стал доминирующим стандартом для структурирования, передачи и обмена данными во всемирной сети через протокол передачи гипертекста (HTTP). Это не отдельная технология, а часть семейства растущих технологий и структур для обмена данными между организациями.

Поскольку XML не зависит от технологии и платформы, это делает возможным обмен данными между разрозненными системами. Таким образом, он особо популярен, позволяя интегрироваться с унаследованными системами. Это означает, что любая система может принимать и интерпретировать XML-документ, не заботясь о платформе источника, и наоборот. Например, источник может быть основан на .NET, а принимающий клиент может быть настольным приложением, клиентом на базе Java, мобильным устройством. IPAQ, базы данных Oracle или системы ASP.NET, например веб-приложения разрабатываемой системы.

Подобно языку гипертекстовой разметки (HTML), XML является языком разметки на основе тэгов. Тем не менее, в отличие от HTML, он позволяет использовать пользовательские теги, которые делают XML-документы более гибкими. Основная цель заключается в организации информации таким образом, чтобы ее могли понять люди, и компьютеры могли ее технически интерпретировать.

2.4 Технология баз данных

Базы данных используются для хранения и управления данными организаций. Хотя технологии хранения и манипулирования данными существуют уже много лет, перенос данных из одной организации в другую не был простой задачей. С появлением XML существует общая среда для обмена данными между организациями независимо от исходной и целевой платформ.

Библиотека классов .NET Framework включает коллекцию классов для управления взаимодействиями с базами данных, которые известны как ADO.NET. ADO.NET основывается на методологиях обработки данных, таких как DataSet. DataSet - это класс, который имеет коллекцию методов для обработки данных на основе XML. Объекты DataSet могут хранить сложную

информацию и отношения в одном переносимом и структурированном объекте. Это возможно, потому что объекты заключают в себе несколько объектов DataTable и DataRelation.

Наборы данных могут быть заполнены динамически с результатами запроса к базе данных или из документов XML. DataSet может быть локально изменен, а позднее синхронизирован с базой данных back-end, используя операции обновления. Синхронизация возможна, поскольку DataSet были разработаны как разобщённые объекты. DataSet также может быть преобразован обратно в XML-формат. Это преобразование возможно, потому что DataSet основан на XML и, таким образом, сохраняет его структуру.

XML настолько тесно интегрирован в платформу .NET Framework, что передача и манипуляция данными HTTP с использованием объектов .NET на базе .NET проста. Эти характеристики делают основанные на XML DataSets отличным выбором для инкапсуляции и передачи данных между клиентами и веб-службами.

2.5 Веб-сервисы

Корпорации могут думать о веб-сервисах, как о бизнес-функциях, передаваемых через Интернет. Технически Веб-сервисы - это объекты и методы, которые могут быть вызваны с любого клиента через HTTP. Несомненно, это центральный элемент .NET Framework, и опять же поддержка XML является фундаментальной в технологиях веб-сервисов. Эта инфраструктура обеспечивает встроенную поддержку для вызова Веб-сервисов без использования дополнительных инструментов. В отличие от предыдущих технологий распределенных систем, веб-службы построены на протоколе простого доступа к объектам (SOAP). SOAP использует стандарт XML для описания данных и позволяет отправлять и получать сообщения через HTTP.

Поскольку он основан на отраслевых стандартах, различные клиентские платформы могут обращаться к веб-службам, вызывая удаленные методы с использованием SOAP-сообщений на основе XML. Веб-методы возвращают результирующие типы данных и объекты, представленные как другое сообщение SOAP. Это позволяет передавать примитивные типы, такие как целые числа и строки, а также сложные DataSet”ы через Веб как XML. В случае DataSets возвращаемые данные содержат два раздела. Первая - это встроенная схема XML, описывающая конкретную таблицу базы данных, а вторая содержит все извлеченные записи таблицы. Возможность возвращать

DataSets является одним из самых мощных видов использования веб-сервисов. Фактически, большинство методов веб-службы разрабатываемой системы возвращают DataSets.

Веб-метод должен иметь определение, как и любой другой не-Веб-метод. Это включает имя метода и его параметры с соответствующими типами. В определении метода также должен указываться модификатор доступа метода. Для веб-методов модификатор доступа должен быть общедоступным, чтобы внешние классы могли обращаться к ним. Тип возвращаемого метода также должен быть указан в определении. Наконец, чтобы сделать метод доступным как Веб-метод, ему нужен атрибут [Web Method]. Веб-методы, которые обращаются к базам данных, должны создать объект подключения к базе данных и задать строку подключения базы данных, к которой необходимо получить доступ.

Многие фирмы изучают веб-службы для выполнения целого ряда задач - от обмена информацией с внутренними компьютерными системами до ужесточения связей с деловыми партнерами. Данный веб-сервис может выполнять функции в нескольких приложениях, освобождая время и ресурсы, способствуя повторному использованию кода и эффективному развитию. Распределенный характер веб-служб делает это возможным, позволяя отделить интерфейс от вычислений и функций базы данных. Как будет видно в следующих разделах, веб-службы также способствуют созданию мощных новых бизнес-моделей.

2.6 Веб-приложения

ASP.NET - это среда программирования, которая позволяет быстро разрабатывать мощные приложения для разных клиентов. Он основан на оригинальной структуре Microsoft Active Server Page (ASP). Говоря простыми словами, ASP - это спецификация динамически создаваемой веб-страницы. Когда браузер запрашивает страницу ASP, веб-сервер создает страницу с кодом HTML и отправляет ее обратно в браузер.

Все веб-страницы ASP.NET имеют расширение .aspx. Файл .aspx представляет собой графический интерфейс приложения. Фактический код приложения находится в файле .aspx.cs. Расширение .cs представляет собой документ C#. Другие языки, совместимые с ASP.NET, имеют другое расширение файла после спецификации .aspx.

Первоначальная модель ASP претерпела значительные изменения, включив новые функциональные возможности, которые являются основой ASP.NET. Возможность разработки и доступа к удаленным веб-сервисам, вероятно, является ключевой особенностью новой инфраструктуры ASP. Веб-службы могут выполнять функции, необходимые веб-приложениям ASP.NET, освобождая приложение для поддержки всех функций локально. В среде ASP.NET веб-службы закодированы в файле .cs.asmx. Файл .asmx предоставляет динамически созданный интерфейс для тестирования и доступа к службам через браузер.

ASP.NET позволяет легко интегрировать веб-интерфейсы и сервисы. Эти два модуля необходимо скомпилировать вместе, но сначала веб-приложение должно ссылаться на службы. Для этого необходимо создать прокси-класс веб-службы. Если веб-адрес службы известен, как это имеет место, этот процесс так же прост, как ввод URL-адреса веб-службы в специальном диалоговом окне. Также можно выполнить поиск в каталоге Universal Description, Discovery and Integration (UDDI) для других служб. UDDI облегчает поиск веб-сервисов, предлагаемых другими компаниями, которые могут соответствовать вашим потребностям.

Когда URL-адрес отправлен, в диалоговом окне отображается страница служебной документации, включая контракт с языком описания веб-сервиса (WSDL). WSDL - это XML-документ, который определяет методы, параметры и типы данных веб-методов, доступных в веб-службе. Принимая контракт, создается прокси-сервер WSDL. Прокси-класс представляет собой сервис на стороне клиента и упрощает использование веб-службы. Клиентское приложение будет взаимодействовать с этим прокси, а не с сервисом напрямую, путем упаковки запросов и возврата сообщений в SOAP. При использовании прокси методы Веб вызывают так же, как и любые другие методы локальных классов.

ASP.NET также предлагает ряд инновационных серверных средств управления. Одним из примеров является элемент управления DataGrid, который может быть легко заполнен информацией, содержащейся в объекте DataSet. Когда Веб-метод обеспечивает извлечение базы данных, информация извлекается из сервера базы данных и передается как XML через HTTP. ASP.NET получает объект DataSet, который является представлением базы данных XML, включая указанные записи таблицы. Приложение распознает, что XML-файл представляет объект DataSet. Таким образом, файл можно легко загрузить в DataGrid, потому что этот элемент управления подключается

к объекту DataSet. На самом деле DataGrid привязан к определенному DataTable в DataSet, возвращаемому Веб-методом. Наконец, элемент управления ASP.NET отображает эту информацию как HTML[4].

Помимо элемента управления DataGrid, другие веб-элементы управления позволяют привязку DataSet и позволяют указывать столбцы, отображаемые приложением. Эта функциональность очень важна для эффективности разработки данной системы. Например, если приложение должно отображать имя пациента и его адрес в двух разных элементах управления, нет необходимости выполнять два специализированных запроса и вызовы веб-методов (т.е. тот, который возвращает имя, а другой, который возвращает адрес). Требуется только один запрос, который возвращает всю информацию о пациенте в объекте DataSet. После того, как DataSet будет получено приложением, каждый элемент управления может выбрать различные столбцы в DataTable.

Наконец, ASP.NET предлагает отличные инструменты для работы с событиями. События запускаются, когда пользователь нажимает кнопку или, например, делает выбор из списка. Приложение может обрабатывать события локально или через вызовы веб-служб. Приложение в первую очередь будет использовать последнее для обработки своих событий.

3 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

3.1 Задачи проектирования

Технологии, описанные в предыдущих разделах, должны быть организованы разумно, чтобы предоставить техническую инфраструктуру, необходимую для поддержки любой бизнес-модели, включая Сеть. На рисунке 4.1 показан дизайн инфраструктуры.

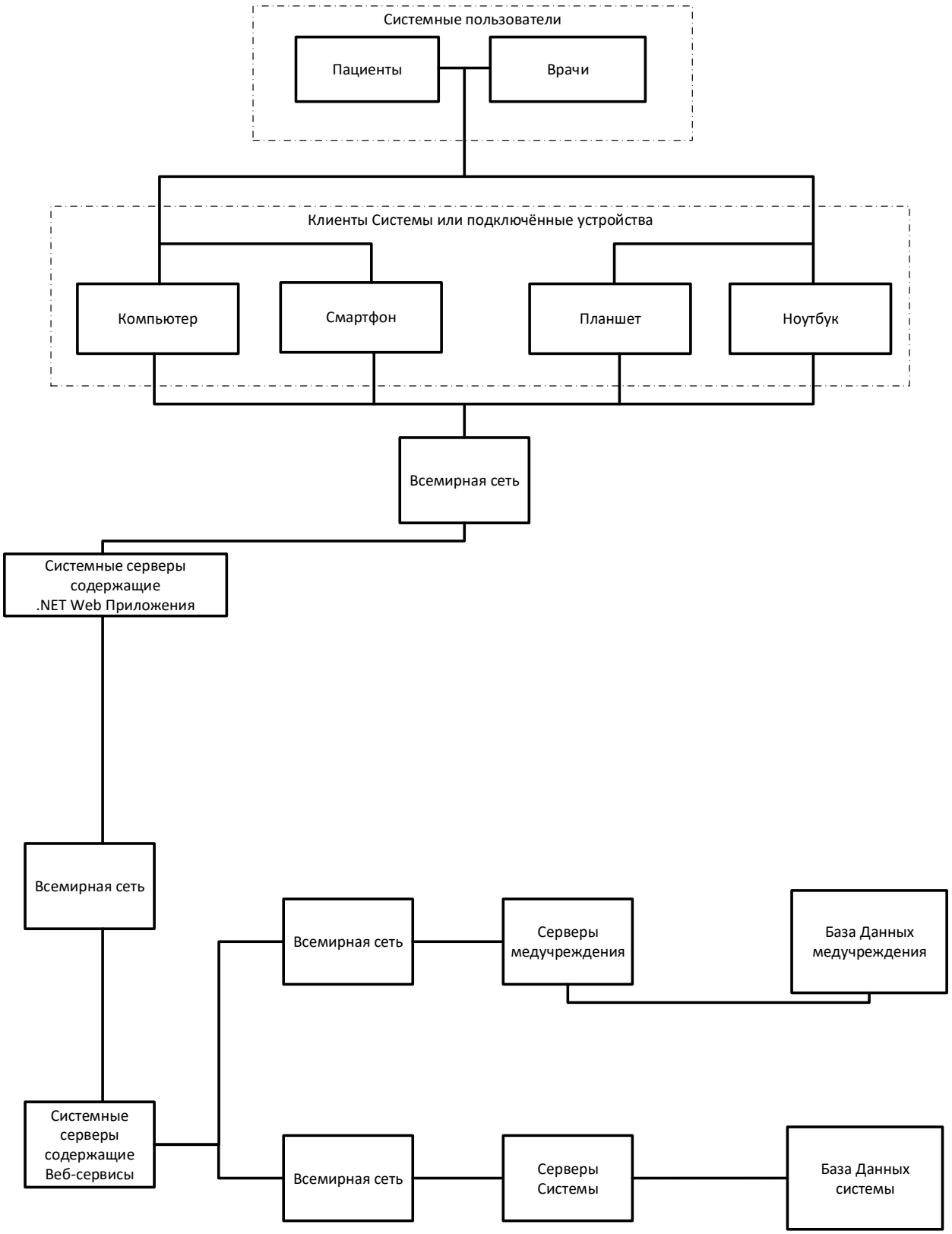

Рисунок 3.1 – Инфраструктура технологии электронного здравоохранения

Хотя прототип системы ориентирован на пациентов и врачей, дизайн инфраструктуры позволяет широкому кругу групп пользователей получать доступ к системе. К ним относятся пациенты, врачи, медсестры, фельдшеры,

лаборанты, фармацевты, администраторы больниц и страховые компании. Эти пользователи могут использовать различные клиентские устройства для подключения к приложению. Например, они могут использовать браузер на ПК / рабочей станции, ноутбуке, беспроводном КПК или мобильном телефоне. Все устройства могут подключаться к приложению через общедоступную и существующую сетевую инфраструктуру Интернета. Веб-сервер принимает и управляет запросами для разных клиентов и направляет их в нужное место. Приложение ASP.NET, расположенное на веб-сервере, вызывает методы сервера веб-служб, когда это необходимо. Вся передача данных между этими двумя серверами осуществляется исключительно через HTTPS посредством Интернета. Веб-службы могут получать доступ к локальной базе данных системы или другим распределенным источникам данных. Взаимодействие с базой данных осуществляется через объекты соединения данных, которые также используют HTTPS в качестве среды обмена.

Очевидно, что инфраструктура технологии построена поверх Интернета и его протоколов передачи. Интернет является основой, которая склеивает и поддерживает связь между различными техническими компонентами системы. Архитектуры, основанные на использовании Интернета, обладают множеством преимуществ по сравнению с собственными инфраструктурами, включая более низкие затраты на обслуживание и более высокую системную совместимость.

Чтобы создать прототип системы, пришлось разработать и создать:

– базу данных для хранения всей системной информации;
– SQL-запросы для всех операций добавления, обновления и извлечения;
– веб-службы, которые будут обрабатывать взаимодействия с базами данных и другие функции;
– веб-приложение, которое служит интерфейсом к системе;
– ссылки на веб-службы для связывания сервисов с приложением.

В остальной части этого документа обсуждаются эти шаги, которые концентрируются на разработке и интеграции технологий баз данных и веб-сервисов. Структура базы данных и веб-службы, представленная в этом документе, отличается от представленного прототипа.

Модель базы данных, в свою очередь, для сложной информационной системы должна быть комплексной и гибкой. Это особенно верно для медицинских учреждений, в которых субъективность практикующего врача и уникальность дела каждого пациента являются центральными аспектами этой

практики. Проектирование базы данных для такой системы требует глубокого понимания бизнес-процессов и правил моделируемого мира. Исследование рынка помогает понять потребности и отобразить их в функциях системы.

Захват процессов и правил в модели реляционной базы данных, безусловно, является проблемой. Более сложной задачей является включение в базу данных свободы выражения, которая так необходима для врачей. При обследовании пациента большинство врачей записывают свои наблюдения и диагноз на чистый лист бумаги, что дает им свободу и гибкость для организации их мыслей любым возможным способом. Большинство врачей могут рассмотреть некоторые общие моменты при осмотре пациента, но информация, включенная в запись пациента, в основном субъективно выбрана. Врачи могут включать любую и всю информацию, которую они считают необходимой, без ограничений и без необходимости категоризации данных.

Кроме того, каждое обследование пациента может проходить по совершенно разным путям в соответствии с конкретной ситуацией данного пациента, знаниями врача и ресурсами больницы. Когда в рабочем процессе имеется так много возможных путей, задача разработки базы данных, а также веб-приложения - задача, которая не будет ограничивать работу врача или лица, принимающего решения.

Наконец, индустрия здравоохранения быстро меняется. На рынок регулярно выводятся новые лекарства, а другие изымаются. Инновационные медицинские тесты и оборудование доступны, в то время как другие становятся устаревшими. Биологические открытия изменяют способ оценки и лечения некоторых заболеваний. Хороший дизайн базы данных для системы должен гарантировать расширяемость базы данных и возможность частой записи изменений. Более того, эти изменения не должны влиять на текущую структуру базы данных, взаимосвязь базы данных со стороной приложения и, самое главное, данными, которые уже сохранены.

3.2 Моделирование реляционной базы данных

Помимо детального понимания моделируемого мира, разработчику необходимо понимать концепции моделирования баз данных. Вся информация в модели реляционной базы данных представляется в виде

таблиц, где каждая таблица представляет собой набор записей или строк. Все записи в определенной таблице имеют одинаковое количество столбцов. Столбцы также называются полями или атрибутами. Набор связанных таблиц формирует базу данных. В следующих разделах будут рассмотрены методы и правила создания базы данных разрабатываемой системы.

3.2.1 Выбор сущностей и атрибутов

Первым шагом в моделировании баз данных является выявление сущностей, в данном случае объектов системы здравоохранения. Сущности могут рассматриваться как системные роли, объекты или другие объекты с независимым существованием, относительно которых информация будет храниться в системе. Пациент, врач, посещение больницы, рецепт и лабораторный тест - все это примеры сущностей. Объекты представляют собой таблицы базы данных. Каждая строка в таблице представляет собой вхождение сущности, и каждый столбец представляет другой атрибут объекта, такой как имя пациента или название лекарства по рецепту.

Хорошей практикой является разработка соглашения об именах сущностей и атрибутов. Для идентификации объектов база данных системы представляет все объекты с первой буквой каждого слова заглавной и всех слов, написанных без пробелов между ними. Атрибуты будут следовать одному и тому же соглашению. Для целей этого документа они будут заключены в кавычки, чтобы иметь возможность отличать их от сущностей. Например, HospitalVisit является сущностью, а атрибут «VisitId» является атрибутом.

Каждый объект должен иметь один атрибут или сочетание атрибутов, которые однозначно идентифицируют каждую строку в таблице. Это гарантирует, что можно различать конкретную запись при выполнении различных операций над таблицей. Уникальные атрибуты называются первичными ключами. Первичный ключ не может иметь нулевое значение, что означает, что значение для этого поля должно быть указано для каждой строки.

В случае объектов, связанных с людьми, имя и фамилия обычно не являются лучшим выбором для первичного ключа. Многие пациенты могут иметь одно и то же имя. Вместо этого уникальные идентификаторы могут генерироваться системой пошагово, случайным образом или в соответствии с некоторыми другими факторами. Идентификаторы могут также содержать

уникальные по определению числа, такие как номер социального страхования. В некоторых случаях также полезно разработать соглашение об идентификаторе. Например, при создании прототипа системы были сделаны некоторые соглашения. Все идентификаторы пациентов начинаются с буквы P, за которой следует случайным образом сгенерированное 9-значное число. То же самое касается врачей, но вместо этого их идентификаторы начинаются с буквы D.

3.2.2 Сущность-связь и диаграммы

Центральным элементом моделирования реляционной базы данных является установление отношений между объектами. Внешние ключи являются важным понятием в отношениях сущностей. Внешний ключ - это столбец или комбинация столбцов в таблице, соответствующей первичному ключу связанной таблицы. Например, если «PatientId» является первичным ключом в таблице «Patient», это будет внешний ключ в любой другой таблице, в который он включен.

Существует три возможных типа отношений: один-к-одному, одно-ко-многим и многое-ко-многим. Отношения «один к одному» означают, что ровно одна строка соответствует одной строке связанной таблицы. Например, может быть одна таблица, в которой хранятся имена пациентов и другая таблица, в которой хранятся адреса и контактная информация. Обе таблицы относятся к одному и тому же пациенту. «PatientId» в одной таблице соответствует точно одному «PatientId» в другой таблице. Имеет смысл объединить эти две таблицы, но в некоторых случаях разделение может быть более эффективным.

В отношениях «один-ко-многим» одна строка соответствует множеству строк соответствующей таблицы. Например, один пациент может запланировать много встреч, и каждая встреча соответствует одному пациенту. Это идеальные отношения типов. Соотношения «многие-ко-многим» не рекомендуется, поскольку они вынуждают повторять данные. В этом типе отношений многие строки соответствуют многим строкам связанной таблицы. Например, одна медицинская процедура или тест могут быть выполнены многими врачами, и один и тот же врач может выполнять многие другие тесты. Если есть тест, включающий трех врачей, вся тестовая информация будет повторяться три раза. Лучше создать два отношения «один ко многим», чтобы каждая комбинация тест-врач была уникальной. Это

возможно путем создания промежуточной таблицы TestTester. Этот пример будет рассмотрен в следующем разделе.

Создание эскизов объектов и их взаимоотношений является фундаментальной частью процесса проектирования. Это дает разработчику БД и другим разработчикам ценный визуальный инструмент. Эти эскизы называются диаграммами сущность-связь. Диаграммы, которые появятся в последующих разделах, будут идентифицировать первичные и внешние ключи с кодом PK и FK, соответственно. Связующая стрелка между объектами представляет отношения[5].

При установлении отношений важно проверить дизайн с правилами нормализации данных. Эти правила представляют собой набор стандартов проектирования, которые сводят к минимуму избыточность данных, чтобы предотвратить введение несогласованной информации. Несоответствия могут привести к повреждению данных и ошибкам. В сущности, правила можно суммировать, как включающие только неключевые столбцы, которые являются фактами столбца первичного ключа таблицы. Например, объект «Patient» должен содержать идентификатор, который ссылается на своего основного врача. Он не должен включать какую-либо другую связанную с врачом информацию, такую как его имя или специализация. Эти неключевые столбцы - это не факты пациента, а врача и должны быть включены только как столбцы сущности Врача.

3.3 Манипулирование данными и их извлечение

SQL, который обозначает Структурированный Язык Запросов, является языком высокого уровня, используемым для манипулирования данными. С помощью команд SQL можно извлекать и изменять данные из таблиц. Веб-службы системы, представленные ниже в этом документе, широко используют команды и предложения SQL. Фактически, большинство Веб-методов системы предназначены исключительно для выполнения поиска и манипулирования данными с помощью SQL.

Операции извлечения, обычно называемые запросами, используют инструкцию SQL SELECT для отображения запрашиваемой информации. Чтобы указать строки, которые вы хотите получить в запросе, предложение WHERE позволяет определить критерии, которым должна удовлетворять строка. Команды модификации включают INSERT, UPDATE и DELETE. Ключевой частью манипулирования реляционными базами данных является

операция JOIN. Объединение двух или более таблиц объединяет их так, как если бы они были единым во время действия операции, путем сравнения значений в указанных столбцах. В этом случае предложение WHERE также используется для сравнения значений.

3.4 Преимущества и ограничения реляционных баз данных

Технологии реляционных баз данных были разработаны для точного и надежного учета деловых операций. Тем не менее, когда необходимо манипулировать огромными объемами данных в режиме реального времени, реляционная база данных может отставать. Эта технология может больше не удовлетворять потребности бизнеса, требующего быстрого анализа данных. Существенной проблемой реляционной модели является то, что по мере роста размера базы данных снижается производительность запросов, увеличивается стоимость владения и невозможность поддержания системы реального времени. В системах реального времени, таких как разрабатываемая, это, безусловно, вызывает беспокойство, особенно когда ожидается, что база данных будет значительно увеличиваться с каждым днем. Например, врачи выполняют несколько приёмов каждый день, и каждый приём добавляет многочисленные строки в различные таблицы базы данных. Многие пациенты также вводят несколько ежедневных измерений состояния здоровья, добавляя сотни или тысячи новых записей.

В дополнение к проблеме размера добавляются дополнительные ограничения, когда база данных сложна. Реляционная модель проста в использовании, но только если она хорошо отображается в структурах данных приложения. Если структура сложная, сопоставление информации с таблицами аналогично форсированию квадратного прямоугольника в круглое отверстие. Отличной альтернативой является объектно-ориентированная модель базы данных. Объектно-ориентированная модель обеспечивает полнофункциональные возможности программирования баз данных в сочетании с элементами объектно-ориентированных языков программирования. Свойства и методы объекта заключают в себе сложность данных. Эта модель также имеет объектно-ориентированный аспект класса, который поддерживает инкапсуляцию, множественное наследование и абстрактные типы данных.

Объектно-ориентированная парадигма имеет то преимущество, что она является более естественным способом моделирования реального мира. Он обеспечивает более естественные структуры данных, лучшую

ремонтопригодность и возможность повторного использования кода. Они также гораздо более подходят для приложений, которые должны обрабатывать сложные отношения. Объектные базы данных являются расширяемыми, что упрощает определение и поддержку новых типов данных. Они также обеспечивают расширенную поддержку мультимедийных типов данных, таких как изображения, аудио и видео, которые понадобятся для будущих циклов разработки системы. Например, эти типы позволяют хранить изображения рентгеновских и других снимков, аудиофайлы сердца, а также видеоролики различных процедур.

Другой альтернативной моделью является XML-база данных. Эти типы баз данных обеспечивают высокую производительность с возможностью масштабирования. Эта модель включает в себя различные уровни сложности данных. XML-базы данных - идеальные решения для приложений, которым необходимо управлять постоянно меняющимися данными, такими как разрабатываемая. Переработка базы данных не требуется при введении или изменении существующих структур данных. Для приложений, в которых реализовано объектно-ориентированное программирование и передача данных на основе XML, эти альтернативные модели являются естественным выбором. Будущие разработки в области электронного здравоохранения должны тщательно анализировать переход к модели объекта и / или модели базы данных XML.

4 РАЗРАБОТКА БАЗЫ ДАННЫХ СИСТЕМЫ

Прототип разрабатываемой системы фокусируется на фундаментальных взаимодействиях между пациентами и практиками здравоохранения, в частности врачами. Таким образом, дизайн базы данных, представленный в этом документе, в основном ограничивается этой областью фокусировки. В ходе обсуждения особое внимание уделяется разработке таблиц базы данных, включая выбор сущностей и атрибутов, а также диаграммы сущность-связь. Чтобы облегчить процесс моделирования базы данных и обсуждения, дизайн был поделен на группы. Каждая группа представляет собой категорию информации, которая необходима для обеспечения предлагаемой функциональности всей системы.

На рисунке 4.1 представлен обзор основных участников и групп данных системы электронного здравоохранения и их взаимосвязей

Рисунок 4.1 – Обзор дизайна базы данных

Центральным звеном диаграммы является объект Посещение, соединяющий все элементы и участников вместе в продуктивных отношениях. В следующих подразделах подробно рассматриваются таблицы, приведенные в каждой группе на рисунке 2.

4.1 Пациенты и врачи

Каждый медицинский центр должен хранить и управлять контактной и демографической информацией о пациентах и врачах. Эта информация является центральным элементом разрабатываемой системы. Взаимодействие между сущностями Patient и Physician управляет большинством других отношений базы данных. В этом разделе обсуждаются эти объекты и их характеристики.

Patient. Сущность пациента является фундаментальной для базы данных, потому что большинство других таблиц напрямую связаны с этой сущностью. Его атрибуты включают «Patientld», «Physicianld», «FirstName», «MiddleName»,LastName»,«SSN»,«DateOfBirth»,«Age»,«Sex»,«Ethnicity»,«We ight»,«Height», «MarialStatus» и контактную информацию, такую как «Adress», «PhoneNumber» и «Email». Первичный ключ для этой сущности - это случайно заданое «Patientld». Несмотря на то, что номер социального страхования (SSN) является кандидатом на первичный ключ, эта информация остается конфиденциальной. Наконец, у каждого пациента есть один основной врач, и, следовательно, есть поле, в котором хранится «PhysicianId» этого врача. Кроме того, каждому врачу может быть назначено много пациентов, что создает взаимосвязь один-ко-многим.

Psycisian – субъект-терапевт также занимает центральное место в базе, потому что пациенты могут получать медицинскую помощь, необходимую им через врачей. К атрибутам врача относятся «FirstName», «MiddleName», «LastName», «Specialization» и контактная информация, такая как «Address», «PhoneNumber», «Pager» и «Email». Первичный ключ таблицы - «Physicianld».

PhysicianVisitHour – система предполагает, что каждый врач имеет одно и то же рабочее время каждую неделю. Например, каждую среду каждый врач исследует пациентов с 8 утра до 10 утра каждые 30 минут. Таким образом, этот объект хранит «Physicianld», «DayOfWeek» и «Time». Три атрибута составляют первичный ключ таблицы. «DayOfWeek» представляет один из семи возможных дней недели. «Time» указывает время начала каждой встречи. Для упомянутого выше врача было бы четыре записи базы данных с

«DayOfWeek», равной «среде». В этом примере столбцы «Время» этих четырех записей содержат «8:00», «8:30», «9:00» и «9:30». Каждый врач может иметь много записей в таблице PhysicianVisitHour, подразумевая связь один-ко-многим.

Рисунок 4.2 иллюстрирует взаимосвязь между пациентами и врачами и их доступными часами посещения.

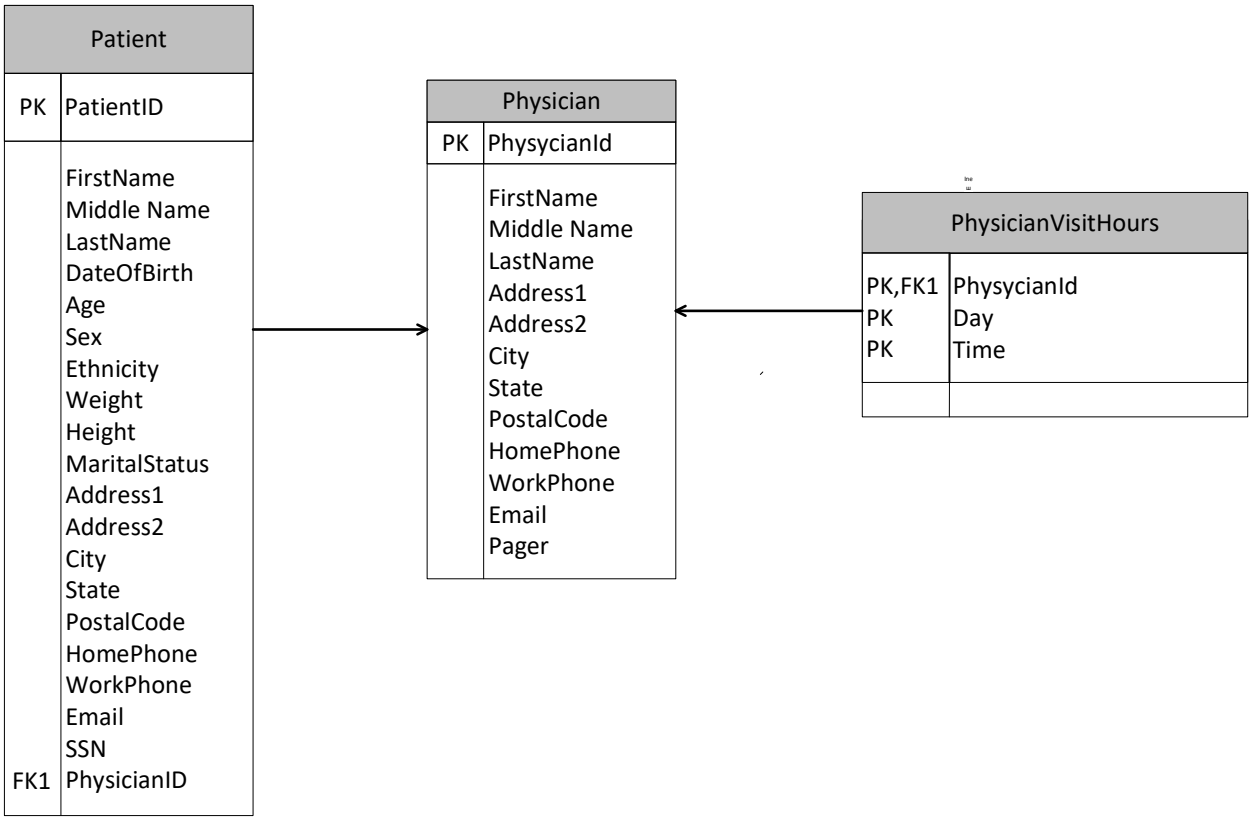

Рисунок 4.2 – Отношения между пациентом и врачом

4.2. Медицинская история пациентов

Медицинская история пациентов, включая аллергии и состояния здоровья, необходима для лучшего понимания потребностей пациента. По этой причине база данных включает в себя группу таблиц для управления медицинской историей.

4.2.1 Аллергии

Аллергии можно разделить на две основные группы: общие аллергии и связанные с медикаментами аллергии. Каждая группа представляет объект базы данных.

GeneralAllergy - атрибутами объекта GeneralAllergy являются «PatientID», «Allergy», «Year» и «Reaction». «Patientld» относится к пациенту в субъекте Patient. «Allergy» - это предмет, вызывающий аллергическую реакцию. «Year» указывает год, когда была обнаружена аллергия. «Reaction» указывает на симптомы, которые испытывает указанный пациент при воздействии с объектом аллергии. Между Patient и GeneralAllergy существует взаимосвязь «один ко многим» (то есть любой конкретный пациент может иметь много аллергий). Поэтому необходимо иметь составной ключ, чтобы однозначно идентифицировать каждую запись. Первичный ключ объединяет «PatientId» и «Allergy».

DrugAllergy - атрибутами объекта DrugAllergy являются «Patientld», «Drugld», «Year» и «Reaction». «DrugId» - это код, который представляет собой упорядоченное лекарство. Информация о каждом препарате хранится в сущности «Drug». Этот объект обсуждается в другом разделе. Первичный ключ - это комбинация «Patientld» и «Drugld». Оба столбца также являются внешними ключами, которые относятся к компоненту «PatientId» и «DrugId» соответственно. Пациент может иметь много аллергий на препараты, и у многих пациентов может быть аллергия на одно и то же лекарство. Таким образом, существует связь «один ко многим» между Patient и DrugAllergy, а также между Drug и DrugAllergy.

Рисунок 7.3 иллюстрирует обсуждаемые отношения. Для простоты объект пациента на рисунке включает поле «Patientld».

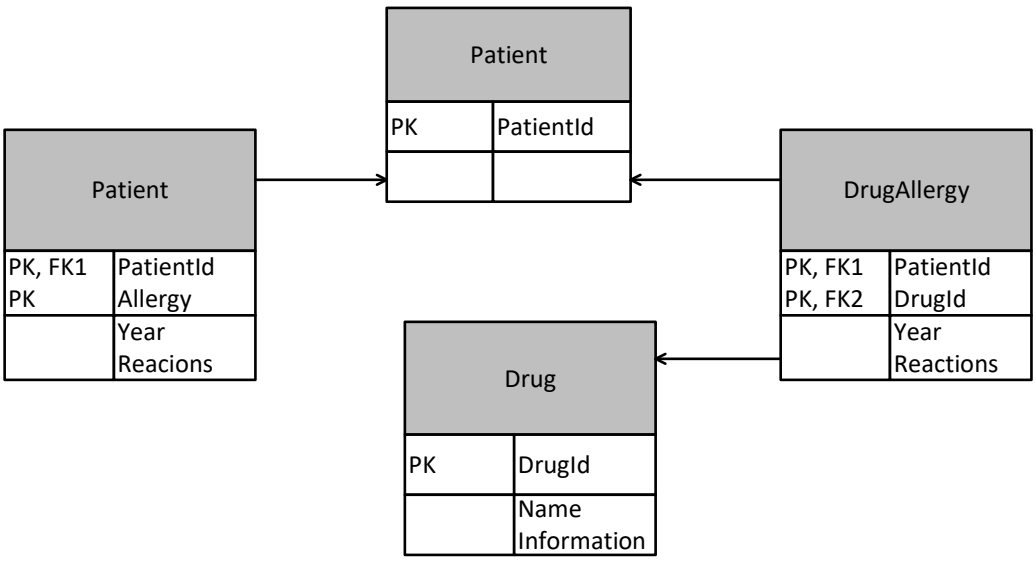

Рисунок 4.3 – Отношения между пациентом и аллергией

4.2.2 Состояние здоровья

Медицинские записи хранят состояние здоровья пациентов и историю болезни их членов семьи. Для этого в базу данных входят два объекта. Рисунок 7.4 иллюстрирует отношения между объектами.

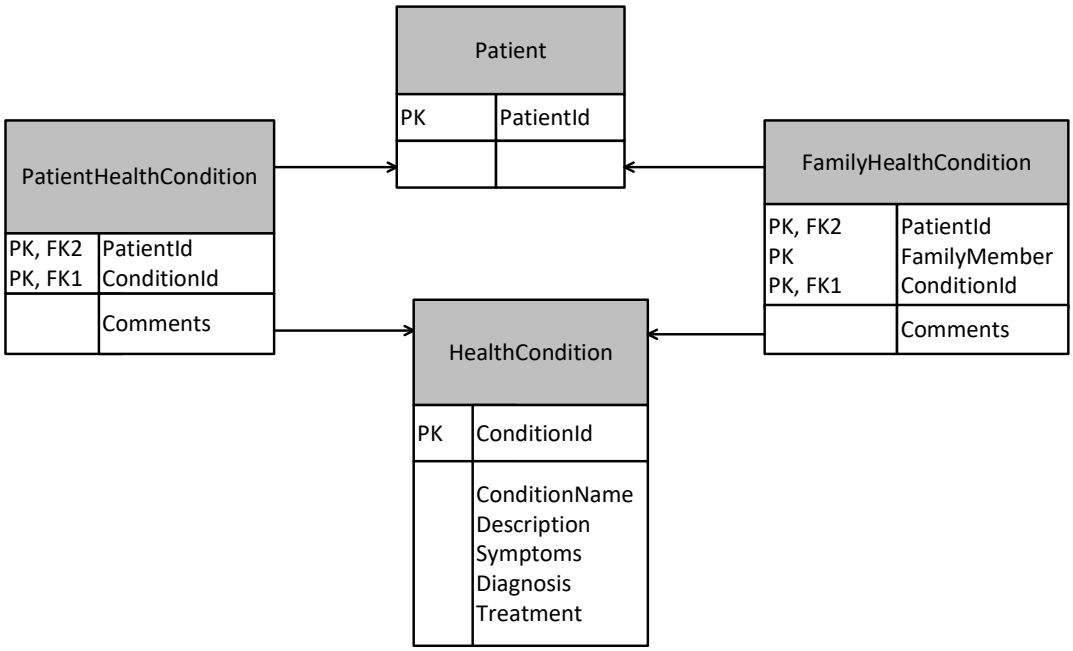

Рисунок 7.4 – Отношения между пациентами и состояниями здоровья

PatientHealthCondition - атрибутами PatientHealthCondition являются «Patientld», «Conditionld» и «Comments». «Conditionld» - это код, который представляет состояние здоровья, которое имеет пациент. Таблица HealthCondition содержит информацию о каждом состоянии. Эта таблица обсуждается в ближайшее время. «Comments» содержат дополнительную информацию об определенном состоянии пациента. Первичные ключи: «Patientld» и «Conditionld». Между этой сущностью и пациентом существует взаимосвязь «один ко многим», потому что у пациента может быть много условий.

FamilyHealthCondition - объект FamilyHealthCondition содержит те же атрибуты таблицы PatientHealthCondition плюс дополнительное поле с именем FamilyMember. В этом поле указывается член семьи пациента, связанный с конкретным состоянием записи. Первичные ключи: «Patientld», «Conditionld» и «FamilyMember». Пациент может иметь много членов семьи с тем же заболеванием, а у конкретного члена семьи может быть несколько заболеваний.

HealthCondition - атрибутами объекта HealthCondition являются «Conditionld», «ConditionName», «Description», «Symptoms», «Diagnosis» и «Treatment». Первичный ключ - это «Conditionld», который представляет собой код, который идентифицирует каждое состояние здоровья при записи. Другие атрибуты являются самоочевидными. Отношения между PatientHealthCondition и HealthCondition являются отношениями один-ко-многим. То же самое относится и к FamilyHealthCondition. Каждая запись в HealthCondition может быть связана с несколькими записями в PatientHealthCondition и FamilyHealthCondition.

4.3 Медицинское страхование

Медицинское страхование является центральной, но очень сложной частью системы здравоохранения из-за ее многочисленных участников и правил. В большинстве случаев пациент должен иметь медицинскую страховку, чтобы иметь право на получение медицинской помощи от врача. В качестве преимущества для больниц разрабатываемая система интегрирует информацию о медицинском страховании в базу данных.

HealthInsuranceCompany - компания HealthInsuranceCompany хранит информацию о страховых компаниях. Атрибутами этой таблицы являются «InsuranceCompanyld», «CompanyName», «Address», «PhoneNumber» и «ContactPerson». Каждый пациент может подписаться на план медицинского обслуживания более чем одной компании, и каждая компания может иметь тысячи подписчиков. Это создает отношения «многие ко многим» между HealthInsuranceCompany и Patient, обосновывая необходимость промежуточной таблицы.

PatientInsurance - PatientInsurance - это промежуточный объект. Он содержит четыре атрибута: «InsuranceCompanyld», «PatientId», «InsurancePolicyNumber» и «GroupPolicyNumber». «InsuranceCompanyld» - это внешний ключ, который относится к компании в таблице HealthInsuranceCompany. Номера полиса определяют план медицинского обслуживания пациента в соответствующей страховой компании. «InsuranceCompanyld» и «Patientld» представляют составной первичный ключ таблицы. PatientInsurance имеет отношение «один ко многим» с HealthInsuranceCompany и Patient. Таким образом, каждая запись PatientInsurance принадлежит одному конкретному пациенту и компании.

На рисунке 4.5 показаны взаимосвязи между этими таблицами.

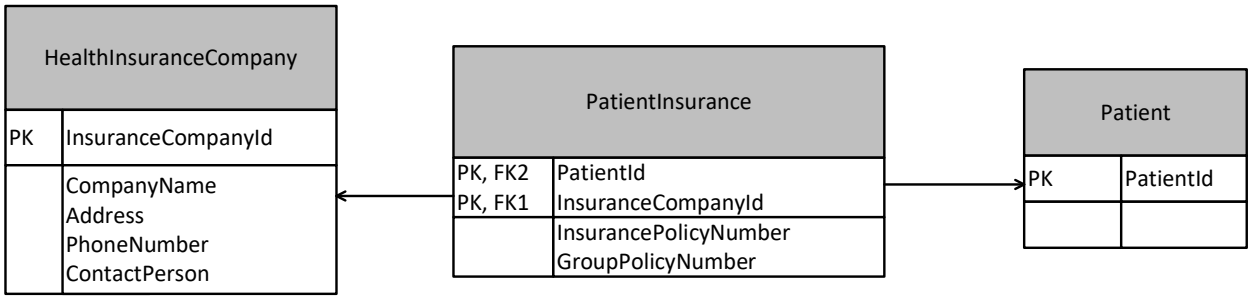

Рисунок 4.5 – Отношение пациентов к медицинскому страхованию

Отношения между пациентом и страховой компанией гораздо сложнее, чем изображенные на рисунке. Страховые компании предлагают несколько планов с различными льготами и ограничениями. Эти простые таблицы нуждаются в дальнейшей доработке на последующих этапах. Кроме того, необходимо будет смоделировать взаимодействие между страховыми компаниями и медицинскими учреждениями.

4.4 Жизненно важные показатели и другие параметры здоровья

Проведение регулярных медицинских измерений и мониторинг прогресса пациента являются жизненно важными элементами полного режима здоровья. Система понимает важность управления измерениями. Таким образом, она обеспечивает функции хранения и отслеживания параметров работоспособности.

4.4.1 Параметры здоровья

База данных содержит семь объектов, соответствующих различным параметрам здоровья. Идея этих таблиц состоит в том, чтобы отслеживать определенный параметр здоровья для конкретного пациента во времени. Все таблицы следуют аналогичной логике и имеют отношение «один ко многим» с сущностью Patient. На рисунке 4.6 показаны взаимосвязи между журналами работоспособности и объектом пациента.

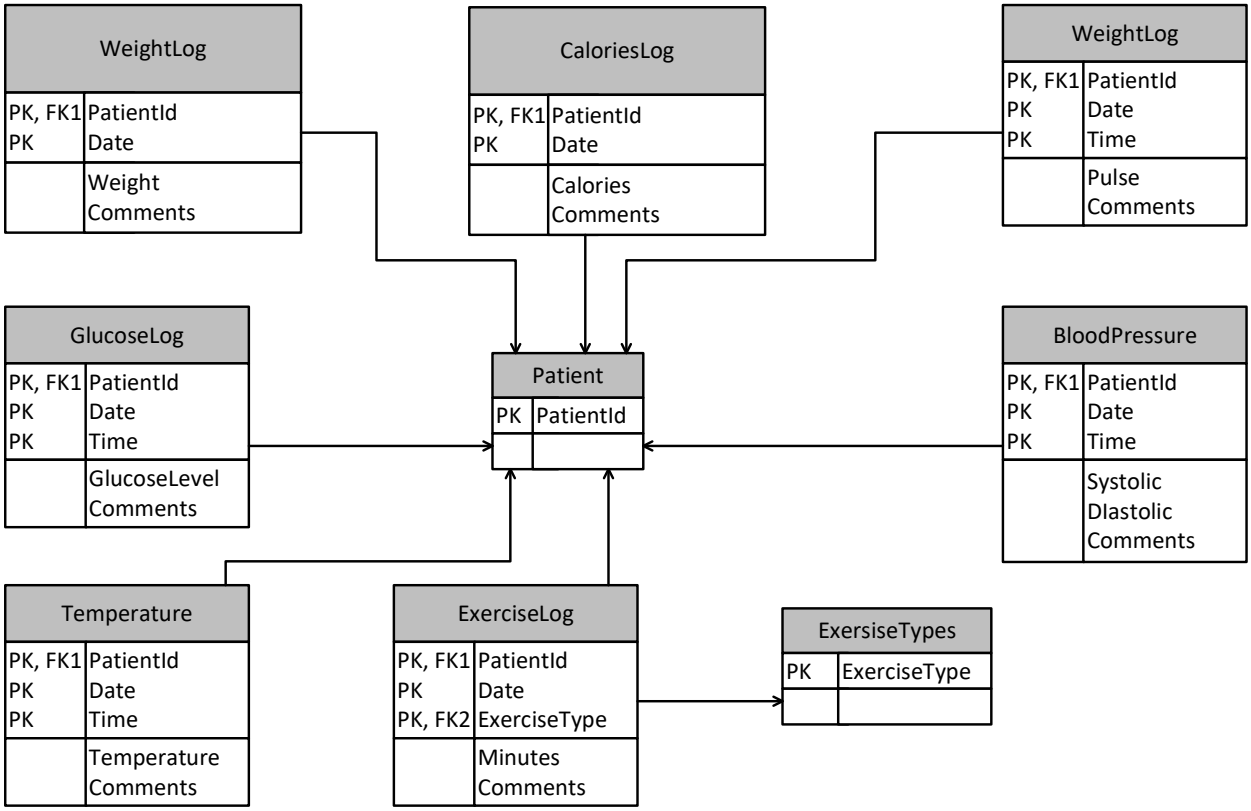

Рисунок 4.6 – Связь между пациентами и журналом работоспособности

Каждая таблица имеет три общих поля, которые являются полями «PatientId», «Date» и «Comments». «Date» хранит дату, когда было выполнено измерение. «Comments» предоставляют гибкость для включения любой связанной информации о данном измерении. Кроме того, каждое измерение имеет уникальные характеристики, и, следовательно, каждая таблица должна иметь разные атрибуты. Например, каждый объект должен хранить фактическое значение измерения.

WeightLog - таблица содержит поле “Weight”. Первичные ключи - «PatientId» и «Date». Это позволяет пациентам вводить свой вес один раз в день.

CaloriesLog - таблица содержит поле «Calories». Первичные ключи - «PatientId» и «Date». Это позволяет пациентам вводить ежедневное потребление калорий.

Некоторые параметры измеряются более одного раза в день, и важно связать каждое измерение с определенным временем. Таким образом, следующие таблицы содержат атрибут «Time». «Time» вместе с «PatientId» и «Date» составляют первичный ключ этих таблиц, что позволяет записывать несколько измерений в разное время в течение одного и того же дня.

GlucoseLog - таблица содержит поле «GlucoseLevel».
BloodPressureLog - таблица содержит поля «Systolic» и «Diastotic».
PulseLog - таблица содержит поле «Pulse».
TemperatureLog - Таблица содержит поле «Temperature».

Наконец, в другой таблице хранится упражнение. ExerciseLog отличается от других журналов, потому что нет конкретного параметра для измерения.

ExerciseLog - Помимо «Patientld» и «Date», ExerciseLog имеет два дополнительных атрибута, которые являются «ExerciseType» и «Minutes». «ExerciseType» указывает тип упражнения, выполняемого пациентом. Это поле относится к записи в таблице ExerciseTypes. «Minutes» записывают время в минутах, которое пациент потратил на выполнение упражнений. Пациент может участвовать в более чем одном типе активности в день. Таким образом, «PatientId», «Date» и «ExerciseType» составляют первичный ключ.

ExerciseTypes - Таблица содержит разнообразный список упражнений для выбора пациентов. Его основной и единственный столбец - «ExerciseType».

4.4.2 Оповещения о пациентах

Помимо отслеживания параметров здоровья, система также управляет предупреждениями, когда данное измерение превышает установленное критическое значение для конкретного пациента. Врачи устанавливают критические значения. Различные врачи могут устанавливать разные значения для одного и того же пациента на основе их личного суждения.

ParameterAlertCriteria - объект ParameterAlertCriteria хранит критические значения для каждого пациента. Он содержит четыре атрибута, которые являются «PatientId», «Physicianld», «HealthParameter» и «CriticalValue». «Physicianld» относится к врачу, который создал каждое предупреждение в записи. «HealthParameter» указывает параметр (например, уровень глюкозы), для которого было установлено предупреждение. «CriticalValue» - это значение, которое врач считает критическим для данного пациента. «PatientId», «Physicianld» и «HealthParameter» составляют первичный ключ таблицы. Эта комбинация ключей позволяет врачам устанавливать множественные критерии оповещения для нескольких пациентов.

HealthParmeterAlert - HealthParameterAlert - это объект, который хранит предупреждения при превышении критических значений. Каждое оповещение

однозначно идентифицируется «Alertld». Другие поля - «PatientId», «Physicianld», «HealthParameter», «MeasurementValue» и «Date». «MeasurementValue» сохраняет запись измерения, которая превышает установленное значение. Например, если уровень глюкозы у пациента равен 130 и превышает его критическое значение, поле «MeasurementValue» будет хранить 130. «Date» хранит дату, когда измерялось превышение. Каждая запись в HealthParameterAlert соответствует ровно одному установленному критерию предупреждения, и каждое установленное предупреждение может иметь много записей в HealthParameterAlert. Это подразумевает одноточечное отношение между двумя таблицами.

Рисунок 4.7 иллюстрирует взаимосвязь между пациентами, врачами и предупреждениями.

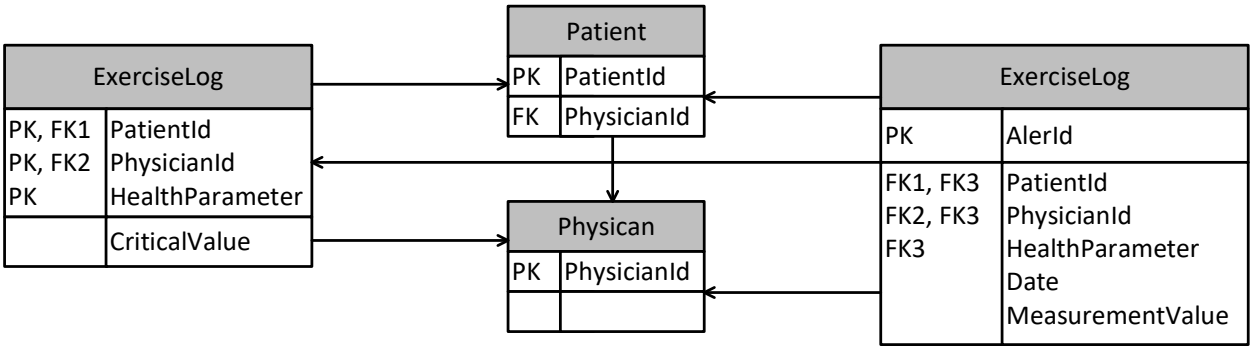

Рисунок 4.7 Отношения между пациентами и оповещениями

Одна и та же основная идея этих предупреждений может быть применена к результатам лабораторных или медицинских испытаний. Например, если измерение пробы крови, такое как гемоглобин, опускается ниже определенного предела, врач может быть немедленно уведомлен о результатах лабораторных исследований.

4.5 Посещения больниц и осмотры

Фундаментальная часть медико-санитарной помощи инкапсулируется при посещении пациентом своего врача. Из-за субъективного характера медицинской практики врачи нуждаются в большой гибкости во время экзаменов, чтобы соответствующим образом адаптироваться к каждому случаю.

4.5.1 Посещения больницы

Существуют три основных типа посещения, каждый из которых имеет уникальные характеристики; Поэтому база данных содержит три объекта для моделирования каждого типа. Этими объектами являются HospitalizationVisit, EmergencyVisit и ScheduledVisit. Три таблицы содержат «Patientld», «Visitld» и «Comments». «Visitld» хранит случайно сгенерированный идентификатор, который уникально идентифицирует каждое посещение записи. «Комментарии» позволяют хранить дополнительную информацию, связанную с конкретным посещением, включая цель посещения. Кроме того, каждое посещение имеет уникальные характеристики и, таким образом, содержит разные столбцы. На рисунке 4.8 показаны объекты посещения и их взаимосвязи.

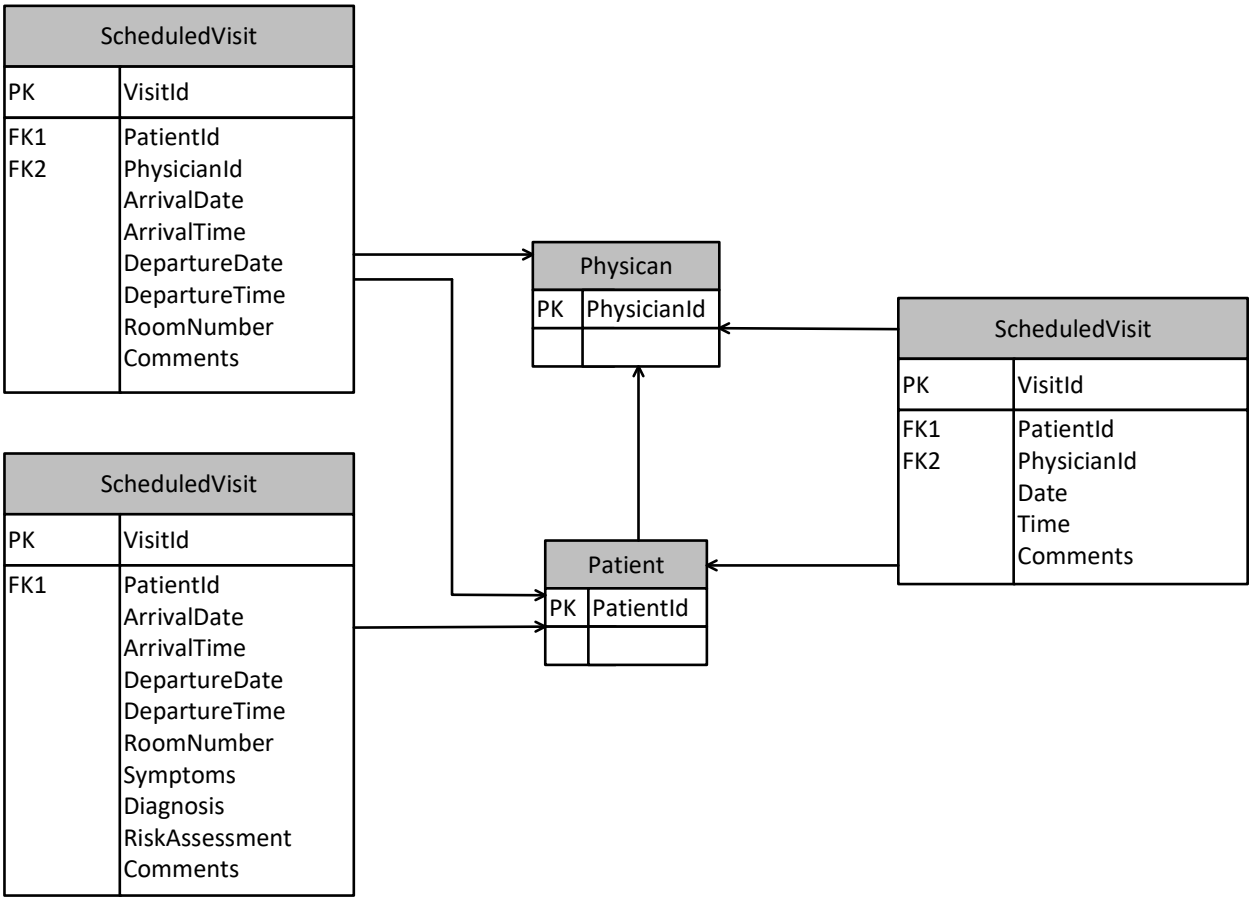

Рисунок 4.8 Параметры работоспособности-предупреждения

ScheduledVisit - Содержит посещение «Date» и «Time» и «PhysicianId» посещаемого врача. Веб-службы и приложения ориентированы на ScheduledVisit.

Объекты HospitalizationVisit и EmergencyVisit имеют другие общие области. Это «ArrivalDate», «ArrivalTime», «DepartureDate» и «DepartureTime» пациента в больницу или отделение неотложной помощи.

HospitalizationVisit - Содержит «RoomNumber», где пациент находится в больнице, и «Physicianld» врача, ответственного за госпитализацию.

EmergencyVisits - при экстренных посещениях необходимо включить информацию о «Symptoms» пациента, предварительном «Diagnosis» и «RiskAssssment».

4.5.2 Обследования

Во время посещения больницы пациент может пройти несколько видов обследования, каждый из которых заполняется другим лицом, осуществляющим уход.

Examination - единица обследования обеспечивает гибкость хранения информации о различных типах осмотров. Атрибутами таблицы являются «ExaminationNumber», «Visitld», «Date», «Examinerld», «ExaminationTypeld», «Observations» и «Comments». Поле «ExaminationNumber» начинается со значения «1», когда врач заказывает первое предписание для указанного посещения. Это значение увеличивается последовательно с каждым дополнительным рецептом, заказанным в данном посещении. «Visitld» связывает каждый осмотр с визитом, когда он был. «Date» хранит дату завершения экспертизы. Во время запланированного посещения дата обследования будет совпадать с датой посещения. В случае госпитализации и срочных визитов, дата находится между датами прибытия и отъезда. «Examinerld» представляет идентификатор врача, медсестры или фельдшера, который завершил исследование. «ExaminationTypeld» указывает тип проверки, которая была завершена. Информация о каждом типе исследования хранится в объекте ExaminationType. «Observations» хранят важные наблюдения, сделанные во время обследования. Например, если врач исследует глаза пациента, он может написать о состоянии глаз в этой области. Поле «Comments» содержит дополнительные замечания, сделанные практикующим. Каждая запись однозначно идентифицируется с помощью «ExaminationNumber» и «VisitId». Запись в таблице экзаменов соответствует одному визиту, но каждое посещение может проходить несколько экзаменов. Это создает отношение «один ко многим» между ScheduledVisit и Examination.

ExaminationType - ExaminationType содержит список всех возможных обследований. Он содержит «ExaminationTypeId», который является первичным ключом, «ExaminationName» и «ProcedureDescription».

Рисунок 4.9 иллюстрирует взаимосвязи таблицы Examination. Для простоты таблица Examiner включает врачей, медсестер и фельдшеров.

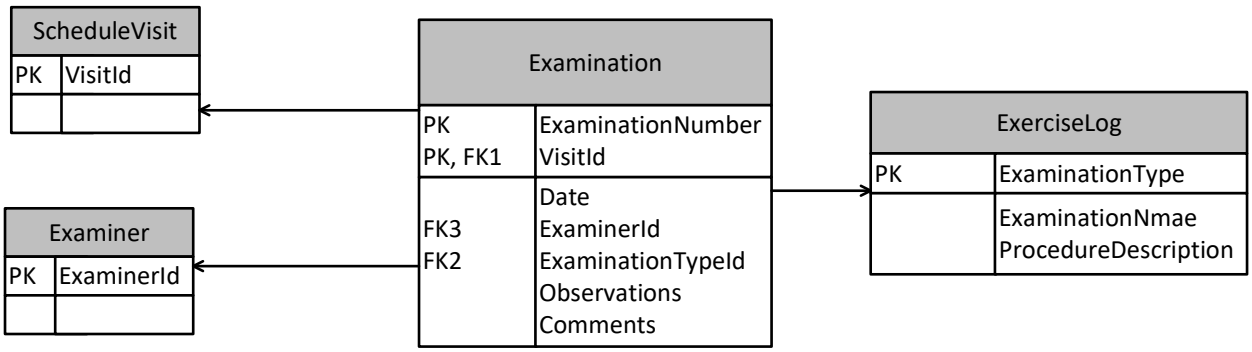

Рисунок 4.9 – Отношения обследований

4.6 Заказы предписания для пациентов

Фундаментальной частью любого медицинского визита являются предписания, составленные врачом. База данных системы содержит две таблицы, которые управляют информацией, относящейся к рецептам.

Prescriptions Order - Объект PrescriptionsOrder хранит необходимую информацию для завершения процесса заказа рецепта. Врач может заказать несколько рецептов для пациента в данном визите, создав отношение один-к-одному между PrescriptionOrder и любым из объектов посещения. Атрибутами таблицы являются «PrescriptionNumber», «Visitld», «OrderDate», «Physicianld», «Drugld», «DrugQuantity», «Dose», «Refills», «PrescriptionPurpose» и «Special instructions». «Prescription Number» является уникальным идентификатором для каждого рецепта, заказанного во время данного посещения. Поле «PrescriptionNumber» начинается со значения «1», когда врач заказывает первое предписание для указанного посещения. Это значение увеличивается последовательно с каждым дополнительным рецептом, заказанным в данном посещении. «VisitId» - это внешний ключ, относящийся к визиту, где был предписан рецепт. «OrderDate» хранит дату, когда заказ был отправлен врачом. «Physicianld» относится к врачу, который представил заказ. «DrugId» представляет собой предписанный препарат. Это поле относится к записи объекта «Drug». «DrugQuantity» - это общее количество таблеток, капсул или жидкости, которые должны быть предоставлены пациенту. «Dose» - это

количество препарата, которое пациент должен принимать в каждый указанный промежуток времени. «Refills» указывают, сколько раз пациенту разрешено изменять порядок рецепта. «PrescriptionPurpose» позволяет врачу определить назначение предписанного препарата. «SpecialInstructions» предоставляет гибкость для добавления дополнительных спецификаций или комментариев об использовании лекарств. Первичный ключ таблицы состоит из «Visitld» и «PrescriptionNumber».

Drug - таблица Drug хранит «Drugld», «DrugName», «ChemicalName», «CommonUses», «Directions», «Предупреждения», «SideEffects» и «Дополнительная информация». «Drugld» - это первичный ключ. Другие поля используются для описания цели применения препарата и его использования. Между PrescriptionOrder и Drug существует отношение «один ко многим».

Patient Prescriptions - объект PatientPrescription хранит дополнительную информацию о конкретном заказе и его последующих шагах. Запись в объекте PrescriptionOrder может иметь более одной связанной записи в таблице PatientPrescription, создавая отношение «один ко многим». Это объясняется тем, что определенный рецепт может повторяться несколько раз. Предел повторения накладывается значением в поле «Refills» таблицы PrescriptionOrder. Атрибутами таблицы являются «VisitId», «PrescriptionNumber», «RefillNumber», «RequestDate», «CompletedDate», «PickupDate», «Pharmacyld», «Pharmacistld» и «Comments». Первые два атрибута идентифицируют конкретную запись в PrescriptionOrder. «RefillNumber» определяет количество раз, когда конкретный заказ был пополнен. Это поле начинается со значения «0», когда рецепт завершается в первый раз. Значение увеличивается последовательно с каждым пополнением для данного рецепта. «RequestDate» хранит дату, когда пациент попросил лекарство из аптеки. «CompletedDate» указывает дату, когда заказ был заполнен аптекой. “PickupDate” - дата, когда пациент взял лекарство. «Pharmacyld» относится к аптеке, где было заказано лекарство. «Pharmacistld» связывает заказ с конкретным фармацевтом, который его завершил. «Comments» позволяют фармацевту включать любые необходимые комментарии о заказе. Первичными ключами PatientPrescription являются «VisitId», «PrescriptionNumber» и «RefillNumber».

Рисунок 4.10 иллюстрирует взаимосвязи между различными таблицами предписания.

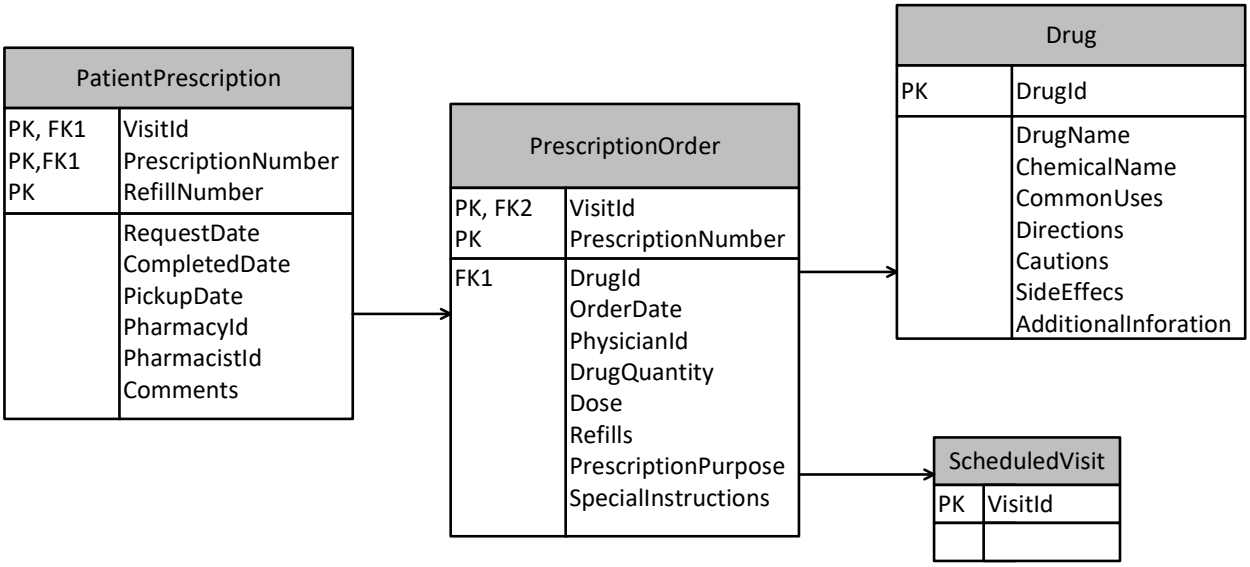

Рисунок 4.10 Отношения с заказами по рецепту

4.7 Лабораторные тесты

Лабораторные исследования важны для диагностики и адекватного лечения заболеваний. База данных разрабатываемой системы хранит заказы лабораторных тестов, представленные врачами, и соответствующие результаты теста. Врачи могут заказать несколько лабораторных анализов за посещение. Можно даже заказать один и тот же лабораторный тест более одного раза во время длительного посещения, например, госпитализации. Это создает отношение «один ко многим».

LabTestOrders - объект LabTestOrder хранит необходимую информацию для завершения процесса упорядочивания теста. Атрибутами таблицы являются «VisitId», «LabTestNumber», «LabTestld», «OrderDate», «Physicianld» и «Комментарии». «Visitld» относится к конкретному визиту, на котором был заказан тест. «LabTestNumber» - уникальный идентификатор для каждого теста, заказанного во время данного посещения. Поле «LabTestNumber» начинается со значения «1», когда врач заказывает первый тест для указанного посещения. Это значение увеличивается последовательно с каждым дополнительным лабораторным тестом, заказанным в данном посещении. «LabTestld» представляет собой лабораторный тест, который вы заказываете. Это поле относится к записи в таблице

LabTestType. «OrderDate» указывает дату, когда тест был заказан врачом, указанным в поле «Physicianld». «Comments» содержат специальные инструкции или комментарии о конкретном тесте. Первичные ключи таблицы - это «VisitId» и «LabTestNumber».

LabTestType - таблица содержит «LabTestld», «TestName», «Description», «Target», «Procedure» и «Preparation”. «LabTestld» является первичным ключом. Другие поля используются для описания теста. Существует отношение одно-ко-многим между LabTestOrder и LabTestType.

База данных также предоставляет таблицы, в которых хранятся результаты лабораторных тестов. Учитывая, что каждый лабораторный тест имеет уникальные атрибуты, для каждого типа теста существуют разные таблицы. Например, в таблице GlucoseToleranceTest хранятся уровни глюкозы для регулярных интервалов времени. В таблице CellBloodCountTest хранятся значения различных типов ячеек в заданное время. Тем не менее, есть шесть общих полей среди таблиц всех типов тестов. Обычными полями являются «VisitId», «LabTestNumber», «TestDate», «Laboratoryld» «LabTechnicianld» и «Comments». «VisitId» и «LabTestNumber» являются первичными ключами всех таблиц результатов тестирования. «TestDate» указывает дату завершения теста. «LabTechnicianld» относится к технику, который оценил тестовый образец. «Laboratoryld» относится к лабораторному объекту, в котором оценивался тест. «Comments» предоставляет пространство для добавления дополнительных наблюдений.

Каждая запись в LabTestOrder соответствует ровно одной записи в одной из таблиц результатов теста, устанавливая взаимное отношение между таблицами. Это один из редких случаев, когда используется этот тип отношений. Необходимо иметь отношение один к одному, потому что каждый тест следует за другим форматом и не может быть сгруппирован в одну общую таблицу. На рисунке 4.11 показаны взаимосвязи между посещениями пациентов, заказами лабораторных тестов и их результатами.

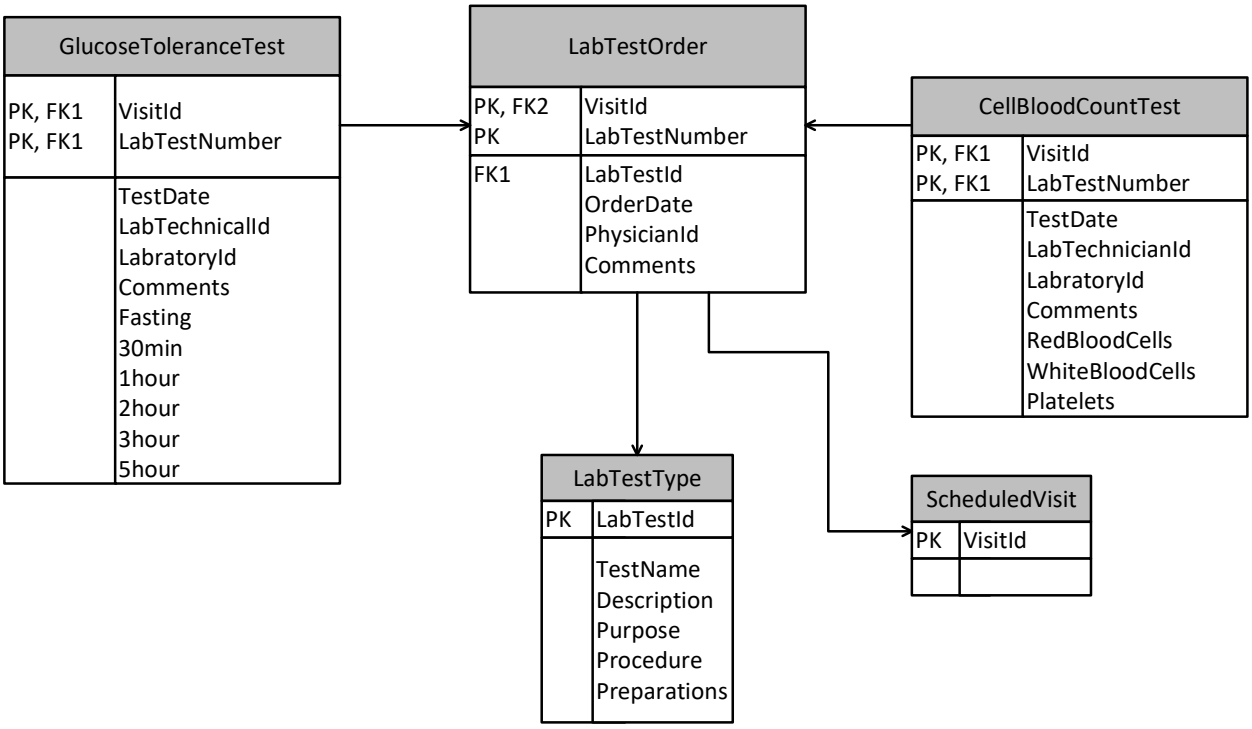

Рисунок 4.11 – Отношения с лабораториями

4.8 Медицинские тесты

Медицинские тесты включают простые и сложные процедуры, которые помогают в диагностике состояния или заболевания и помогают определить его тяжесть. Примеры тестов включают в себя сонограммы, электрокардиограммы, стресс-тесты и рентген.

MedicalTestOrder - таблица MedicalTestOrder похожа на таблицу LabTestOrder. Он имеет двойной первичный ключ, состоящий из «VisitId» и «TestNumber». «VisitId» относится к конкретному посещению, где был заказан тест, а «TestNumber» является уникальным идентификатором для каждого теста, заказанного во время данного посещения. Поле «TestNumber» начинается со значения «1», когда врач заказывает первый тест для указанного посещения. Это значение увеличивается последовательно с каждым дополнительным тестовым заказом, представленным в данном посещении. Как и в таблице LabTestOrder, в таблице MedicalTestOrder также хранятся «OrderDate», «Physicianld» и «Comments». Он также хранит «MedicalTestId», который представляет собой медицинский тест, который вы заказываете. Это поле относится к определенной записи в таблице MedicalTestType.

MedicalTestType - эта таблица содержит «MedicalTestld», «TestName», «Description», «Purpose», «Procedure» и «Preparation». «MedicalTestld» является первичным ключом. Другие поля используются для описания теста.

Существует множество взаимосвязей между MedicalTestOrder и MedicalTestType.

База данных предоставляет таблицы, в которых хранятся результаты медицинского обследования. Учитывая, что каждый медицинский тест имеет уникальные атрибуты, несколько таблиц хранят разные порции информации о каждом тесте.

TestObservationsResult - TestObservationsResults хранит результаты наблюдений и результаты в виде текста. Каждая запись в MedicalTestOrder соответствует ровно одной записи в этой таблице, устанавливая отношение «один к одному». Поля «Visitld», «MedicalTestNumber», «TestDate», «Observations» и «Results». Первичные ключи таблицы - это «Visitld» и «MedicalTestNumber». «TestDate» указывает дату завершения теста.

TestImage - таблица TestImage предоставляет гибкость для хранения изображений для разных тестов независимо от их типа. Например, он может хранить изображения рентгеновских лучей и сонограмм. В таблице хранятся «Visitld», «MedicalTestNumber», «ImageNumber», «ImageFileName» и «Obserrvations». Первые два поля связывают запись с определенным тестовым заказом. Поле «ImageNumber» начинается со значения «1» при сохранении первого изображения. Значение увеличивается последовательно с каждым дополнительным изображением. «ImageFileName» хранит имя файла, который содержит тестовое изображение. «Obserrvations» позволяет комментировать любые изображения. Первичный ключ состоит из «Visitld», «MedicalTestNumber» и «ImageNumber». Многие изображения могут быть созданы в конкретном тесте, поэтому каждая запись в MedicalTestOrder может иметь много связанных записей на TestImage. Это устанавливает отношение «один ко многим» между таблицами.

TestTester - таблица TestTester хранит врачей и техников, которые были вовлечены в данный тест. В отличие от лабораторного теста, когда один специалист оценивает образец крови, медицинский тест может проводиться несколькими лицами, ухаживающими за больными. Поля таблицы - это «Visitld», «MedicalTestNumber» и «Testerld». Первые два поля идентифицируют конкретный тестовый заказ. «Testerld» представляет техник или врач, участвующих в процедуре. Между TestTester и объектом MedicalTestOrder и Tester существует взаимосвязь «один ко многим». Для простоты объект Tester используется в этом случае для представления врачей, медсестер и технических специалистов.

На рисунке 4.12 показаны взаимосвязи между посещениями пациентов, заказами на медицинские испытания и их результатами.

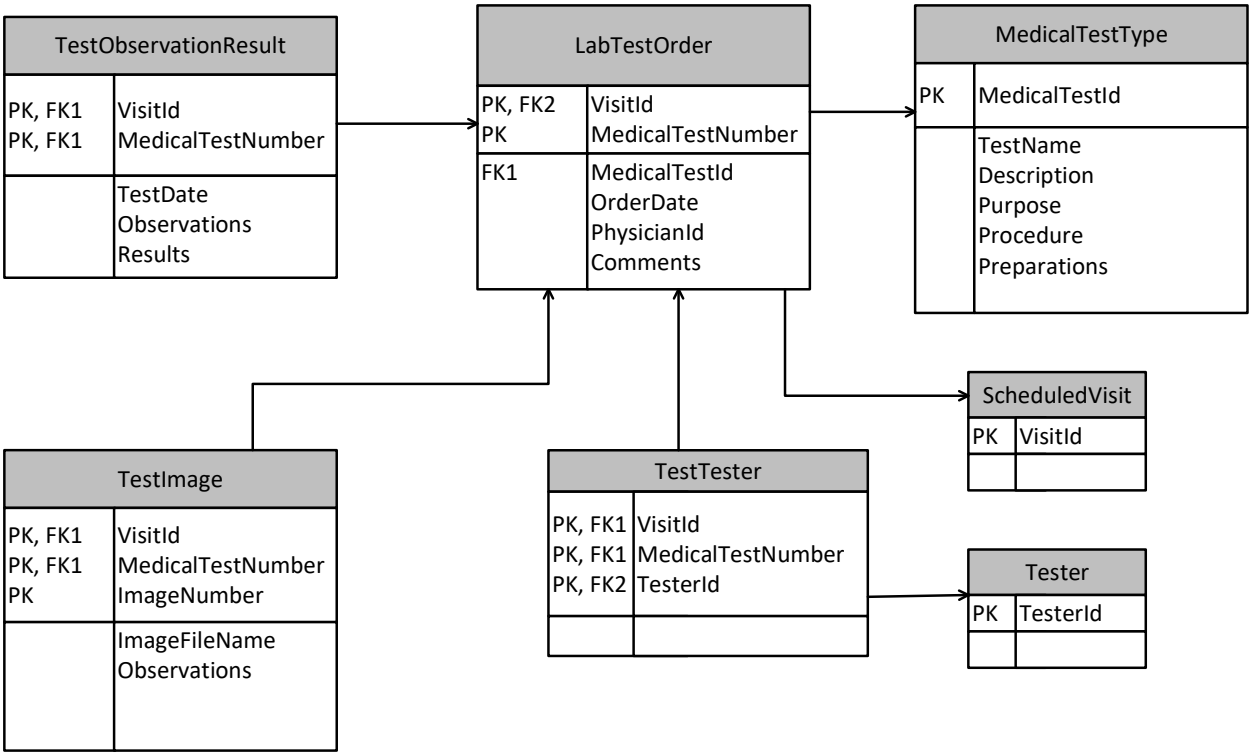

Рисунок 4.12 – Медицинские тестовые отношения

5 ВЕБ-СЕРВИСЫ

Веб-сервисы являются жизненно важным элементом решения. Они предназначены для выполнения различных функций удаленного вычисления, в частности, манипуляций с базами данных. Каждый веб-метод связан с конкретным запросом SQL. Некоторые методы содержат простые однотабличные запросы. Другие методы выполняют сложные операции, которые включают в себя несколько запросов с участием объединений, сравнение DataSet и вызовы внутренних методов. У Веб-методов есть два типа возвращаемых данных. Методы, которые извлекают данные, имеют тип возврата DataSet. Методы, которые включают вставку данных, возвращают логическое значение, указывающее успешность или неудачу операции[6].

Существует компромисс между общей спецификой веб-методов. Большинство методов были разработаны для оптимального использования с Веб и мобильными приложениями, потому что это была конечная цель разработки. Хотя уровень представления отделен от манипуляций с данными, возник вопрос о том, как конечные пользователи хотят просматривать и упорядочивать свою информацию. Таким образом, некоторые методы были разработаны специально для интерфейсов системы. Тем не менее, большинство методов также являются достаточно общими для использования с другими приложениями. Некоторые примеры такого компромисса будут обсуждены позже.

В разрабатываемой системе есть семь веб-сервисов: PatientInfo, HealthTools, Scheduling, Examination, Prescription, Laboratory и MedicalTest. База данных служила моделью для структурирования различных услуг. Таким образом, каждая служба управляет различными группами баз данных, которые были обсуждены. Каждая служба и ее методы рассматриваются в следующих разделах.

5.1 Служба информации о пациенте

Методы в веб-службе PatientInfo управляют личной и медицинской информацией пациентов. Все методы заключают в себе простой SQL-запрос, который извлекает записи из разных таблиц, связанных с объектом Patient. Основные функции каждого метода объясняются ниже.

PatientInformation - принимает «PatientId» и извлекает DataSet из таблицы Patient, которая содержит одну запись с информацией об указанном пациенте.

PatientInsurancePlans - принимает «PatientId» и извлекает DataSet из таблицы PatientInsurance, которая содержит одну или несколько записей с информацией медицинского страхования указанного пациента.

PatientPrimaryPhysician - принимает «PatientId» и извлекает DataSet из таблицы Physician, которая содержит одну запись с информацией о главном враче указанного пациента. Во-первых, метод ищет в таблице Patient значение «Physicianld», а затем использует его для извлечения записи из таблицы Physician.

PhysicianPatients - принимает «Physicianld» и извлекает DataSet из таблицы Patient, которая содержит ноль или более записей, с информацией о пациентах, имеющих в качестве основного врача того, у кого указан указанный идентификатор.

PatientGeneralAllergies - принимает «PatientId» и извлекает DataSet из таблицы GeneralAllergy, которая содержит ноль или более записей об общей аллергии указанного пациента.

PatientDrugAllergies - принимает «PatientId» и извлекает DataSet из таблицы DrugAllergy, которая содержит ноль или более записей о лекарственной аллергии указанного пациента.

PatientHealthConditions - принимает «PatientId» и извлекает DataSet из таблицы PatientHealthCondition, которая содержит ноль или более записей о состоянии здоровья указанного пациента.

PatientFamilyHealthConditions - принимает «Patientld» и извлекает DataSet из таблицы FamilyHealthCondition, которая содержит ноль или более записей о состоянии здоровья семьи указанного пациента.

5.2 Служба средств управления таблицами

В веб-службе HealthTools имеется 14 веб-методов, которые относятся к одной из семи таблиц параметров работоспособности. Каждая таблица имеет один соответствующий метод для просмотра существующих записей, а другой для добавления новых записей. Кроме того, существует четыре метода управления параметрическими предупреждениями.

Методы, которые добавляют новые записи, имеют сходную логику реализации. После вставки нового измерения метод вызывает частный метод для поиска предупреждений пациента, которые находятся в таблице ParameterAlert Criteria. Единичное измерение может превышать

установленные критерии нескольких врачей. Частный метод возвращает DataSet, содержащий записи о любых превышениях критериев предупреждения.

Метод обновления проверяет, является ли DataSet пустым или нет. Если он не пуст, он вызывает другой закрытый метод для вставки новых записей в таблицу Health ParameterAlert, соответствующую каждому из превышений оповещений. Частный метод генерирует случайный «AlertId» для каждой записи как первичный ключ HealthParameterAlert. Ниже приводится описание каждого метода.

ViewBloodPressureLog - принимает «Patientld» и извлекает DataSet из таблицы BloodPressureLog, которая содержит ноль или более записей со всеми записями артериального давления указанного пациента.

UpdateBloodPressureLog - принимает «Patientld», «Date», «Time», «BloodPressure» и «Comments», чтобы ввести новую запись кровяного давления. Инструкция SQL INSERT создает новую запись, в которой хранятся необходимые поля в BloodPressureLog. Метод возвращает логический тип, который указывает, была ли вставка успешной или нет.

ViewPulseLog - принимает «Patientld» и извлекает DataSet из таблицы PulseLog, которая содержит ноль или более записей со всеми импульсными записями указанного пациента.

UpdatePulseLog - принимает «Patientld», «Date», «Time», «Pulse» и «Comments», чтобы ввести новый импульс. Инструкция SQL INSERT создает новую запись, в которой хранятся необходимые поля в PulseLog. Метод возвращает логический тип, который указывает, была ли вставка успешной или нет.

ViewWeightLog - принимает «Patientld» и извлекает DataSet из таблицы WeightLog, которая содержит ноль или более записей со всеми записями веса указанного пациента.

UpdateWeightLog - принимает «Patientld», «Дата», «Вес» и «Комментарии», чтобы ввести новую запись веса. Инструкция SQL INSERT создает новую запись, в которой хранятся необходимые поля в WeightLog. Метод использует новую запись веса, чтобы обновить таблицу Patient с помощью другого SQL-запроса. В этом столе хранится самый последний вес для пациента. Метод возвращает логический тип, который указывает, была ли вставка успешной или нет.

ViewGlucoseLog - принимает «Patientld» и извлекает DataSet из таблицы GlucoseLog, которая содержит ноль или более записей со всеми записями уровня глюкозы указанного пациента.

UpdateGlucoseLog – принимает «Patientld», «Date», «Time», «GlucoseLevel» и «Comments», чтобы ввести новую запись глюкозы. Оператор SQL INSERT создает новую запись, в которой хранятся необходимые поля в GlucoseLog. Возвращает логический тип, указывающий, была ли вставка успешной или нет.

ViewCaloriesLog - принимает «Patientld» и извлекает DataSet из таблицы CaloriesLog, которая содержит ноль или более записей с ежедневными количествами калорий указанного пациента.

UpdateCaloriesLog - принимает «Patientld», «Дата», «Калории» и «Комментарии» для ввода новой записи калории. Инструкция SQL INSERT создает новую запись, в которой хранятся необходимые поля в CaloriesLog. Метод возвращает логический тип, который указывает, была ли вставка успешной или нет.

ViewTemperatureLog - принимает «Patientld» и извлекает DataSet из таблицы TemperatureLog, которая содержит ноль или более записей со всеми температурными записями указанного пациента.

UpdateTemperatureLog - Принимает «Patientld», «Date», «Time», «Temperature» и «Comments» для ввода новой записи температуры. Оператор SQL INSERT создает новую запись, в которой хранятся необходимые поля в температурном журнале. Метод возвращает логический тип, который указывает, была ли вставка успешной или нет.

ViewExerciseLog – принимает «Patientld» и извлекает DataSet из таблицы ExerciseLog, которая содержит ноль или более записей со всеми сообщениями об активности упражнений указанного пациента.

UpdateExerciseLog - принимает «Patientld», «Date», «ExerciseType», «Minutes» и «Comments», чтобы ввести новую запись упражнения. Инструкция SQL INSERT создает новую запись, в которой хранятся необходимые поля в ExerciseLog. Метод возвращает логический тип, который указывает, была ли вставка успешной или нет.

EstablishPatientAlertCriteria – принимает «Patientld», «Physicianld», «HealthParameter» и «CriticalValue». Оператор SQL INSERT создает запись, которая хранит эти поля в таблице ParameterAlertCriteria. Метод возвращает

логический тип, который указывает, была ли вставка выполнена успешно или нет.

DeletePatientAlertCriteria - Принимает “Patientld”, “Physicianld” и “HealthParameter”. Инструкция SQL DELETE устраняет запись, на которую ссылаются эти поля. Метод возвращает логический тип, который указывает, была ли операция успешной или нет.

ViewPatientAlerts - принимает «Patientld» и извлекает DataSet из таблицы HealthParmeterAlert, которая содержит ноль или более записей со всеми сообщаемыми предупреждениями для указанного пациента.

ViewPhysicianPatientAlerts - принимает «Physicianld» и извлекает DataSet из таблицы HealthParmeterAlert, которая содержит ноль или более записей со всеми сообщаемыми предупреждениями, которые были установлены указанным врачом.

5.3 Служба Планирования

Услуга «Планирование» позволяет пациентам и врачам организовывать и управлять своими медицинскими встречами. В дополнение к манипуляции с базой данных эти методы включают серию табличных объединений, сравнений и дополнительных вычислений.

PhysicianAvailableTimes - принимает «Physicianld» и «Date» и возвращает DataSet, который содержит ноль или более записей с доступными временами врача в указанную дату. Возвращаемые данные не извлекаются непосредственно из существующей таблицы базы данных. Сначала, запрос возвращает рабочее время врача в указанную дату. Эти данные находятся в таблице PhysicianVisitHours. Затем другой запрос возвращает время, когда врач назначил встречи на указанную дату. Эти данные находятся в таблице ScheduledVisit, где записываются все запланированные посещения. Веб-метод сравнивает время, возвращенное обоими запросами, и определяет время, когда врач еще доступен. DataSet создается для хранения этих времен, а затем возвращается в вызывающую функцию.

CreateAppointment - принимает «PatientId», «Date», «Time» и «Comment» для назначения новой встречи. Прототип позволяет только назначать встречи с первичными врачами. «Physicianld» не требуется в качестве входных данных для этого метода. Он определяет, кто является основным врачом пациента, вызвав метод PatientPrimaryPhysician веб-службы PatientInfo. Метод также проверяет, доступен ли пациент в указанные дату и время. Это делается путем

сравнения данной даты и времени с текущими запланированными встречами, которые хранятся в таблице ScheduledVisit. Если пациент доступен, для уникальной идентификации записи генерируется случайный «Вирус». Частный метод с именем VisitIdGenerator создает эту идентификацию. Этот метод проверяет, что сгенерированный идентификатор не существует в базе данных. Наконец, инструкция SQL INSERT создает новую запись, в которой хранятся необходимые поля в таблице ScheduledVisit. Метод возвращает логический тип, который указывает, была ли вставка успешной или нет.

PhysicianAppointments - принимает «Physicianld» и «Date» для извлечения DataSet из таблицы ScheduledVisit, которая содержит ноль или более записей с информацией о запланированных встречах врача на указанную дату.

PatientFutureAppointments - принимает «Patientld» и извлекает DataSet из таблицы ScheduledVisits, которая содержит ноль или более записей с информацией обо всех запланированных назначениях для данного пациента. Столбец «Дата» для выбранных записей содержит текущую дату или будущую дату.

PatientPastAppointments - принимает «PatientId» и извлекает DataSet из таблицы ScheduledVisit, которая содержит ноль или более записей с информацией обо всех предыдущих назначениях для данного пациента. Столбец «Дата» для выбранных записей содержит текущую дату или прошедшую дату.

Интересно заметить разницу в дизайне между тремя методами, которые возвращают встречи. Каждый день у врачей много назначений, и лучший способ организовать их - по дате. Таким образом, существует способ получения назначений по дате. Напротив, у пациентов может быть только один или два запланированных встречи, и более эффективно показать им полный список всех будущих назначений. Назначение метода PatientPastAppointments показано ниже.

Эта разница в дизайне показывает компромисс между общностью и специфичностью методов. Потребности пользователей играют значительную роль в принятии решения о том, где установить компромисс. Эти три метода взаимодействуют с одной таблицей, но предназначены для разных функций. Если бы использовался общий метод для всех трех функций, это подразумевало бы дальнейшие манипуляции SQL в приложении ASP.NET.

Веб-службы были разработаны таким образом, чтобы не приходилось беспокоиться о манипулировании данными.

5.4 Экспертиза

Служба экспертизы содержит ряд методов для записи и просмотра данных, относящихся к обследованиям пациентов.

ExaminationTypes - не принимает аргументов и возвращает DataSet, который содержит список всех типов проверки, хранящихся в таблице ExaminationType.

CreateExamination - принимает значение «VisitId», «ExaminationTypeld», «Examinerld», «Date», «Observations» и «Comments». Этот метод вызывает другой частный не-веб-метод с именем ExaminationNumberGenerator для создания уникального «ExaminationNumber». Во-первых, частный метод проверяет, есть ли другие проверки, связанные с указанным посещением. Затем он генерирует «ExaminationNumber» на основе количества предыдущих экзаменов. Например, если в данном визите было два предыдущих экзамена, номер новой записи будет «3». Инструкция SQL INSERT создает запись, в которой хранятся эти поля в таблице Examination. Метод возвращает логический тип, который указывает, была ли вставка успешной или нет.

ViewVisitExaminations - принимает значение “Visitld” и возвращает DataSet из таблицы Examination, которая содержит ноль или более строк с информацией обо всех проверках, выполненных в указанное посещение.

5.5 Предписания

Служба предписания предоставляет методы для подачи и просмотра рецептов.

Drugs – не принимает аргументов и возвращает DataSet из таблицы Drug, которая содержит информацию обо всех лекарствах, находящихся в настоящее время на рынке.

OrderPrescription – принимает «Visitld», «Drugld», «OrderDate», «Physicianld», «Drug Quantity», « Dose», « Refills», «PrescriptionPurpose» и «SpecialInstructions». Метод вызывает частный не-веб-метод под названием PrescriptionNumberGenerator для создания уникального «PrescriptionNumber». Во-первых, частный метод проверяет, были ли заказаны другие рецепты во время указанного посещения. Затем он генерирует «PrescriptionNumber» в зависимости от количества предыдущих заказов. Оператор SQL INSERT

создает новую запись с указанными аргументами в таблице PrescriptionOrder. Метод возвращает логический тип, который указывает, была ли вставка успешной или нет.

ViewVisitPrescriptionOrders – принимает значение «VisitId» и возвращает DataSet из таблицы PrescriptionOrder, которая содержит информацию обо всех рецептах, упорядоченных во время указанного посещения.

PatientPrescriptions – принимает «Patientld» и возвращает DataSet из таблицы PrescriptionOrder, которая содержит информацию о предписаниях по рецепту для указанного пациента.

Опять же, интересно заметить, что есть два разных метода, которые возвращают информацию о предписании. Первый метод предназначен специально для врачей, чтобы они могли организовывать рецептурные заказы путем посещения. Второй метод может быть использован как пациентами, так и врачами, поскольку он возвращает сводку всех рецептов, заказанных для конкретного пациента.

5.6 Лабораторная служба

Служба Лаборатории содержит методы отправки и просмотра заказов лабораторных тестов.

LabTestTypes - не принимает аргументов и возвращает DataSet, который содержит список всех типов лабораторных тестов, перечисленных в таблице LabTestType.

OrderLabTest – принимает «VisitId», «LabTestld», «OrderDate», «PhysicianId» и «Комментарии». Метод вызывает частный не-веб-метод LabTestNumberGenerator для создания уникального «LabTestNumber». Сначала частный метод проверяет, В ходе указанного посещения были заказаны другие лабораторные тесты, затем генерируется «LabTestNumber» на основе количества предыдущих заказов. Наконец, инструкция SQL INSERT создает новую запись с указанными аргументами в таблице LabTestOrder.

ViewVisitLabTestOrders - принимает «VisitId» и возвращает DataSet из таблицы LabTestOrder, который содержит информацию обо всех лабораторных тестах, которые были заказаны во время указанного посещения.

PatientLabTestOrders - принимает «Patientld» и возвращает DataSet из таблицы LabTestOrder, который содержит информацию обо всех лабораторных тестах, которые были заказаны для указанного пациента.

5.7 Служба медицинских тестов

Служба MedicalTest имеет методы подачи и просмотра заказов на медицинские испытания.

MedicalTestTypes - не принимает аргументов и возвращает DataSet, который содержит список всех типов медицинских тестов, перечисленных в таблице MedicalTestType.

OrderMedicalTest - принимает значение «Visitld», «MedicalTestld», «OrderDate», «Physicianld» и «Comments». Метод вызывает частный не-веб-метод под названием MedicalTestNumberGenerator для создания уникального «MedicalTesNumber». Во-первых, частный метод проверяет, были ли заказаны другие медицинские тесты во время указанного посещения. Затем он генерирует «MedicalTestNumber» на основании количества предыдущих заказов. Наконец, инструкция SQL INSERT создает новую запись с указанными аргументами в таблице MedicalTestOrder.

ViewVisitMedicalTestOrders - принимает значение «Visitld» и возвращает DataSet из таблицы MedicalTestOrder, которая содержит информацию обо всех медицинских тестах, которые были заказаны во время указанного посещения.

PatientMedicalTestOrders - принимает «Patientld» и возвращает DataSet из таблицы MedicalTestOrder, которая содержит информацию обо всех медицинских тестах, заказанных для указанного пациента.

6 РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЙ

Последним шагом после проектирования и кодирования веб-служб является создание веб-приложения. Приложение связывает пользовательский интерфейс с веб-службами[7].

Приложение построено в основном из страниц .aspx. Он также содержит другие страницы, которые были разработаны как стандартные .html-страницы с JavaScript. Фрагмент кода приведён в Приложении А. Страницы .html являются в основном учебными и информативными и не требуют взаимодействия с пользователем, веб-службами или базой данных. Цель этого раздела - представить обзор дизайна страниц .aspx, в частности страниц, взаимодействующих с веб-службами. Обсуждение будет сосредоточено на том, какие методы веб-служб вызывается и как данные, возвращаемые службами, отображаются в приложении.

На рисунке 6.13 показано, как приложение-прототип разветвляется на два вспомогательных приложения для стороны пациента и стороны врача.

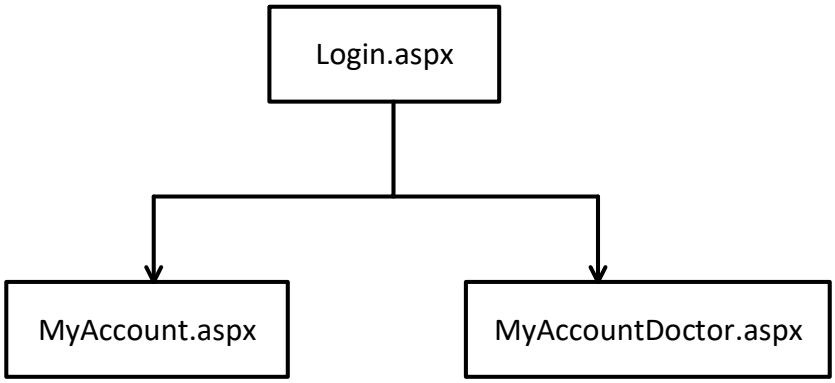

Рисунок 6.13 – Структура входа в систему электронного здравоохранения

Система предлагает одинаковый интерфейс входа для всех пользователей, и разумно распознавать роль пользователя через идентификатор входа. После успешной авторизации Login.aspx перенаправляет пользователя на страницу приветствия в соответствии с его ролью. После входа в систему приложение определяет, кто является пользователем и на каких страницах он может получить доступ.

Изображение интерфейса для входа для персонального компьютера (слева) и для мобильных устройств (справа) можно увидеть на изображении 6.14.

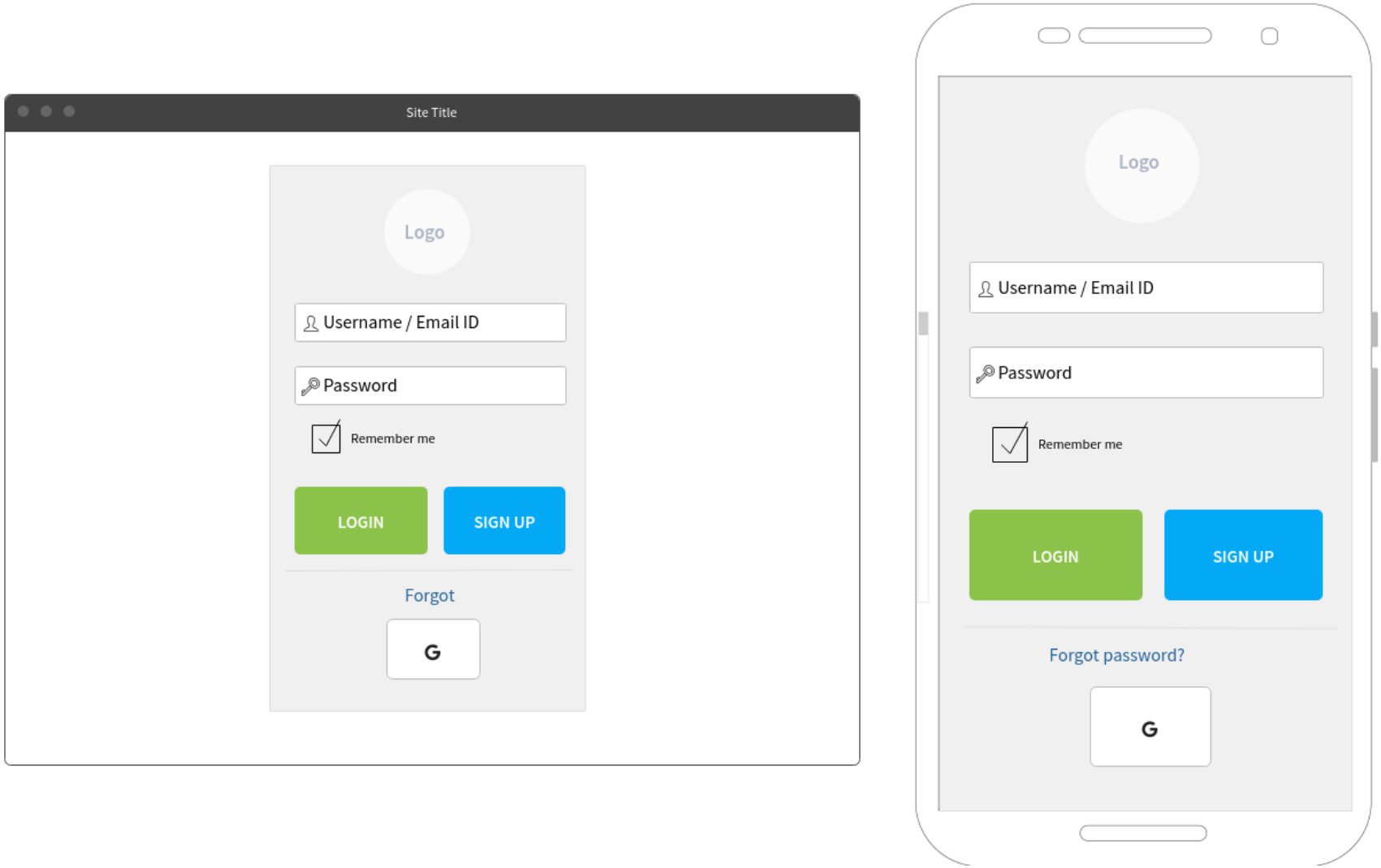

Рисунок 6.14 – Архитектура приложения для пациентов

6.1 Приложение для пациентов

Архитектура приложения на стороне пациента показана на рисунке 6.15.

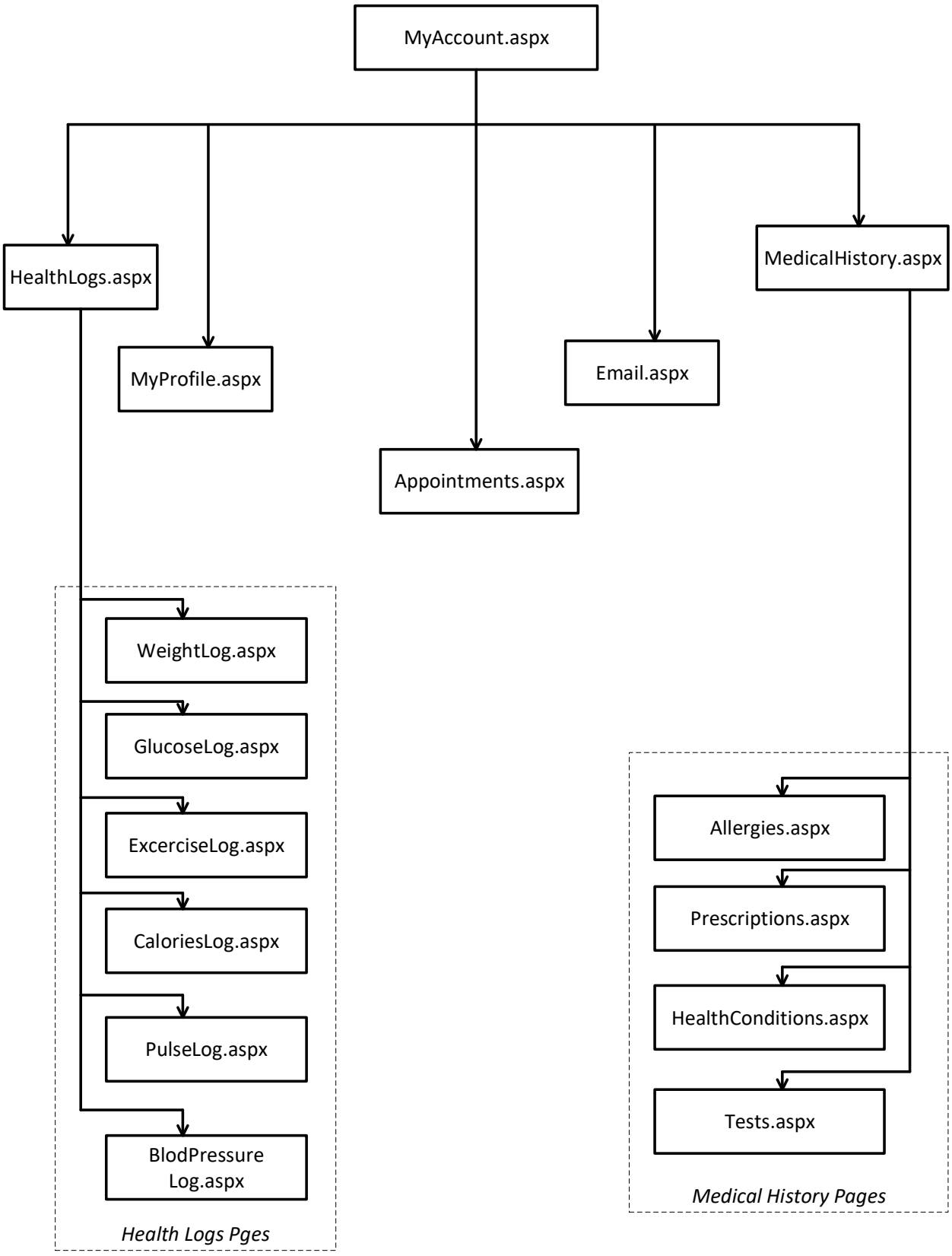

Рисунок 6.15 – Архитектура приложения для пациентов

На странице приветствия для пациента отображается MyAccount.aspx.

MyAccount.aspx – вызывает метод ViewPatientAlerts и связывает возвращенный DataSet в DataGrid для пациентов, чтобы просматривать их личные предупреждения.

В MyAccount.aspx пациенты могут перейти к пяти другим страницам: HealthLogs.aspx, MyProfile.aspx, Appointments.aspx, Email.aspx и MedicalHistory.aspx. HealthLogs.aspx и MedicalHistory.aspx не используют веб-службы. Использование веб-сервиса других страниц описано ниже.

MyProfile.aspx - вызывает PatientInformation и PatientInsurancePlans и связывает возвращенные столбцы с несколькими текстовыми полями, отображающими всю информацию о пациенте.

Appointments.aspx - вызывает PatientFutureAppointments, чтобы показать пациентам запланированные посещения. Возвращенный DataSet ограничен элементом управления DataGrid. На этой странице также предусмотрен календарь для выбора даты новых встреч. Каждый раз, когда выбрана дата, страница вызывает PhysicianAvailableTimes. Доступное время ограничено раскрывающимся списком. Когда пациент выбирает дату и нажимает кнопку отправки, вызывается метод CreateAppointment. PatientFutureAppointments вызывается снова, чтобы отобразить новое запланированное назначение.

Email.aspx - включает текстовые поля, которые напоминают формат электронной почты для пациентов, чтобы отправлять электронные сообщения своим основным врачам. Когда пациент нажимает кнопку отправки, вызывается PatientPrimaryPhysician. Электронный адрес врача содержится в возвращаемом DataSet. Сообщение электронной почты отправляется на этот адрес.

Страница HealthLogs.aspx содержит ссылки на шесть страниц, которые позволяют пациентам вводить и отслеживать параметры здоровья.

WeightLog.aspx - вызывает ViewWeightLog при загрузке страницы. Используя информацию в возвращаемом DataSet, страница создает график, который показывает траекторию всех измерений веса. На этой странице отображаются текстовые поля для ввода новых данных пациентом. При нажатии кнопки отправки введенные параметры передаются методу UpdateWeightLog. ViewWeightLog снова вызывается для отображения вновь введенных данных.

ExerciseLog.aspx - вызывает ViewExerciseLog, когда страница загружается. Используя информацию в возвращенном DataSet, страница создает граф, который показывает траекторию всех упражнений. На этой странице отображаются текстовые поля для ввода новых данных пациентом. Когда пользователь нажимает кнопку отправки, введенные параметры передаются методу UpdateExerciseLog. ViewExerciseLog снова вызывается для отображения вновь введенных данных.

CaloriesLog.aspx - вызывает ViewCaloriesLog, когда страница загружается. Используя информацию в возвращаемом DataSet, метод создает график, который показывает траекторию всех записей калорий. На странице есть текстовые поля для ввода новых данных пациентом. Пользователь нажимает кнопку отправки, введенные параметры передаются в метод UpdateCaloriesLog. ViewCaloriesLog снова вызывается для отображения вновь введенных данных.

GlucoseLog.aspx - вызывает ViewGlucoseLog при загрузке страницы. Используя информацию в возвращаемом DataSet, метод создает график, который показывает траекторию всех измерений глюкозы. На этой странице отображаются текстовые поля для ввода новых данных пациентом. Когда пользователь нажимает кнопку отправки, введенные параметры передаются методу UpdateGlucoseLog. ViewGlucosetLog снова вызывается для отображения вновь введенных данных.

PressureLog.aspx - вызывает ViewBloodPressureLog при загрузке страницы. Используя информацию в возвращаемом DataSet, метод создает график, который показывает траекторию всех измерений кровяного давления. На этой странице отображаются текстовые поля для ввода новых данных пациентом. Когда пользователь нажимает кнопку отправки, введенные параметры передаются методу UpdateBlood PressureLog. ViewBloodPressureLog снова вызывается для отображения вновь введенных данных.

PulseLog.aspx - вызывает ViewPulseLog, когда страница загружается. Используя информацию в возвращаемом DataSet, метод создает график, который показывает траекторию всех импульсных измерений. На этой странице отображаются текстовые поля для ввода новых данных пациентом. Когда пользователь нажимает кнопку отправки, введенные параметры передаются методу UpdatePulseLog. ViewPulseLog снова вызывается для отображения вновь введенных данных.

Страница MedicalHistory.aspx содержит ссылки на четыре страницы, которые позволяют пациентам просматривать их историю болезни, включая условия, аллергию, а также заказываемые тесты и рецепты.

1. HealthConditions.aspx - вызывает методы PatientHealthConditions и PatientFamilyHealthConditions и отображает возвращенные DataSet в двух отдельных DataGrid.

2. Allergies.aspx - вызывает методы PatientGeneralAllergies и PatientDrugAllergies и отображает возвращенные DataSet в двух отдельных DataGrids.

3. Prescriptions.aspx - вызывает метод PatientPrescriptions и отображает возвращенный DataSet в DataGrid.

4. Tests.aspx - вызывает методы PatientLabTestOrders и PatientMedicalTestOrders и отображает возвращенные DataSet в двух отдельных DataGrids.

Интерфейс главной страницы приложения пациента можно увидеть на рисунке 6.16

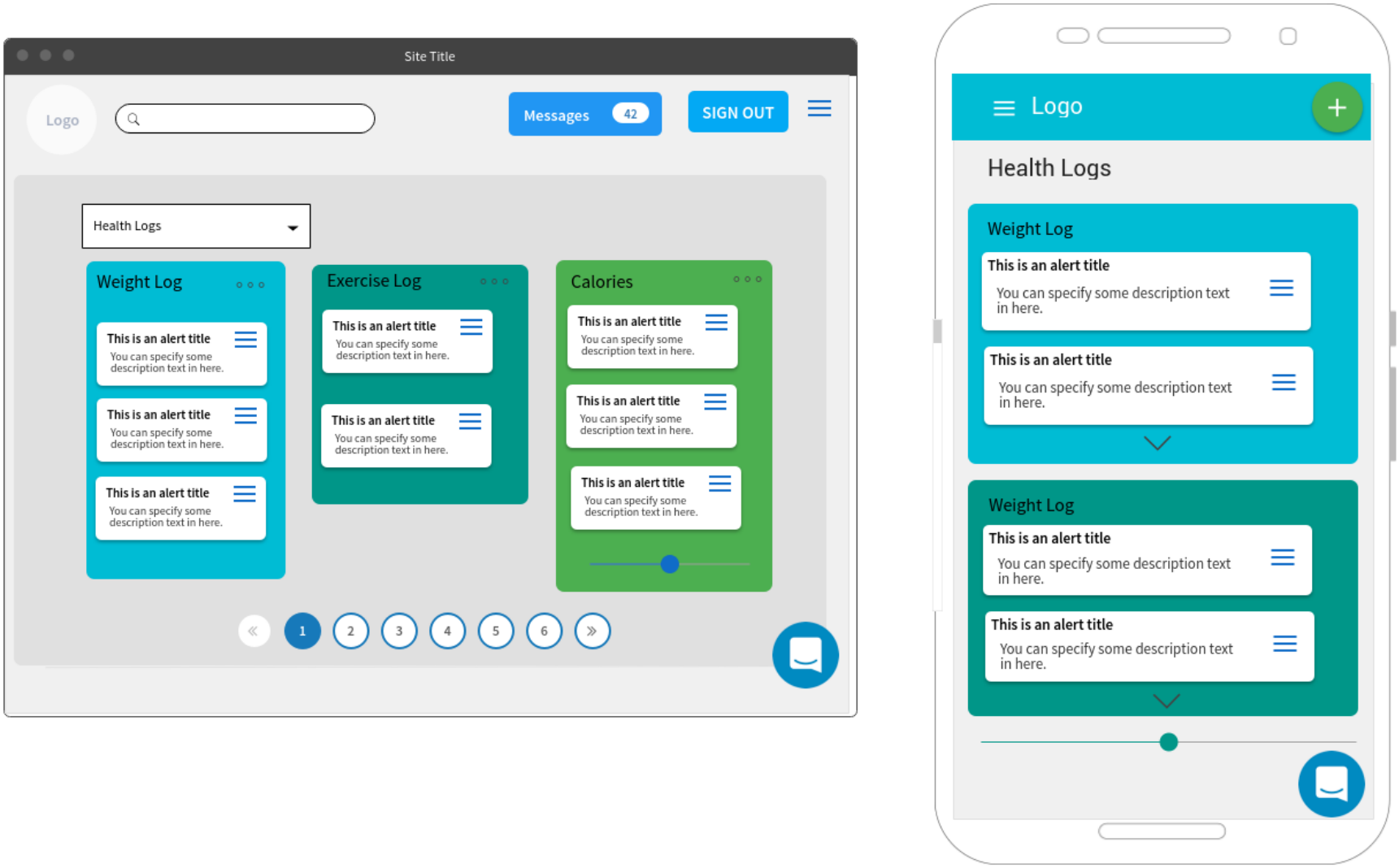

Рисунок 6.16 – Интерфейс главной страницы приложения пациента

6.2 Приложение врача

Архитектура приложения на стороне врача показана на рисунке 6.17.

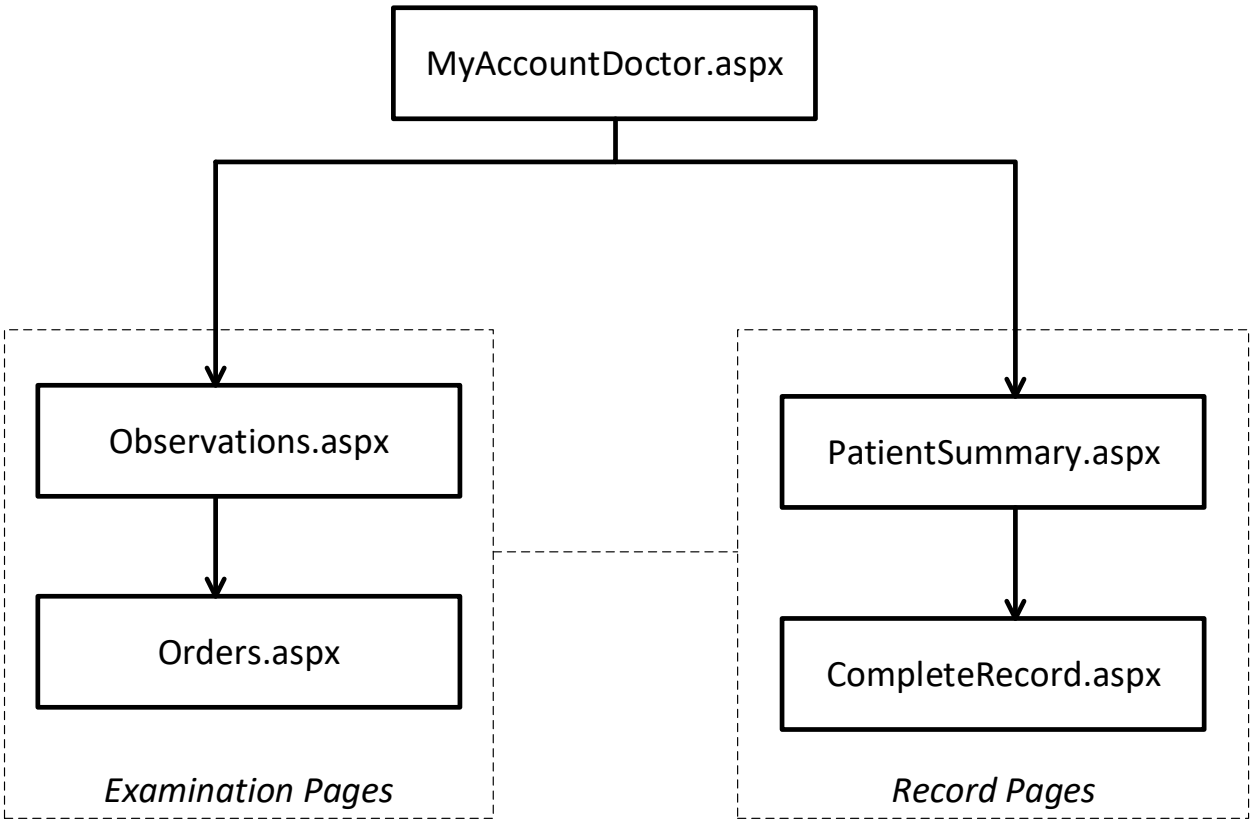

Рисунок 6.17 – Архитектура приложения врача.

На странице приветствия для пациента MyAccountDoctor.aspx находятся:

MyAccountDoctor.aspx - вызывает метод ViewPhysicianPatientAlerts и связывает возвращенный DataSet в DataGrid. Метод PhysicianPatients также вызывается и ограничен выпадающим меню, которое отображает список пациентов врача. На этой странице также предусмотрен календарь для выбора даты для просмотра встреч. Каждый раз, когда вы выбираете дату, страница вызывает пункт PhysicianAppointments. Возвращенный DataSet с назначениями для выбранной даты привязан к DataGrid.

В MyAccountDoctor.aspx, у врачей есть возможность проверить пациента, запланированного на текущую дату. Это делается путем выбора одного из пациентов в элементе управления DataGrid назначения. Две страницы позволяют врачам регистрировать информацию, связанную с посещением.

1. Examination.aspx - страница вызывает методы ExaminationTypes и связывает DataSet с раскрывающимся списком, который покажет врачам все доступные типы проверки. Врач может сделать выбор, ввести данные в различные текстовые поля, а когда нажата кнопка отправки, страница вызывает CreateExamination. На странице отображается подтверждающее сообщение в соответствии с возвращаемым логическим типом.

2. Orders.aspx - страница вызывает Drugs, LabTestTypes, MedicalTestTypes и связывает три возвращенных DataSet в три раскрывающихся списка. Врачи могут выбрать лекарство из списка и представить рецепт. Страница вызывает OrderPrescription, когда врач делает выбор. То же самое относится к лабораторным испытаниям и медицинским испытаниям. В этих случаях вызываются соответственно OrderLabTest и OrderMedicalTest. На странице отображается подтверждающее сообщение во всех трех случаях в соответствии с возвращаемым логическим типом.

На MyAccountDoctor.aspx врачи также имеют возможность просматривать записи пациента без необходимости обследовать этого пациента. Это делается путем выбора одного из пациентов в списке выпадающих пациентов. Две страницы содержат информацию, относящуюся к записи.

1. Record.aspx - Когда страница загружается, вызывается метод PatientPastAppointments. Набор данных привязан к выпадающему списку, который показывает дату всех предыдущих встреч. Врачи могут выбрать любое из прошлых посещений из списка, чтобы просмотреть подробную информацию о посещении. Когда выбран конкретный визит, страница вызывает ViewVisitExamination, ViewVisitLabTestOrders, ViewVisitMedicalTestOrders и ViewVisitPrescriptionOrders. DataSets, возвращаемые четырьмя методами, ограничены четырьмя различными DataGrid.

2. PatientSummary.aspx - страница вызывает семь методов, все связанные с таблицами журналов работоспособности. Это ViewBloodPressureLog, ViewGlucoseLog, ViewTemperatureLog, ViewCaloriesLog, ViewPulseLog, ViewExerciseLog и ViewWeightLog. Страница использует возвращенные DataSets для отображения графиков с траекторией каждого измерения здоровья.

Интерфейс главной страницы приложения врача можно увидеть на рисунке 6.18

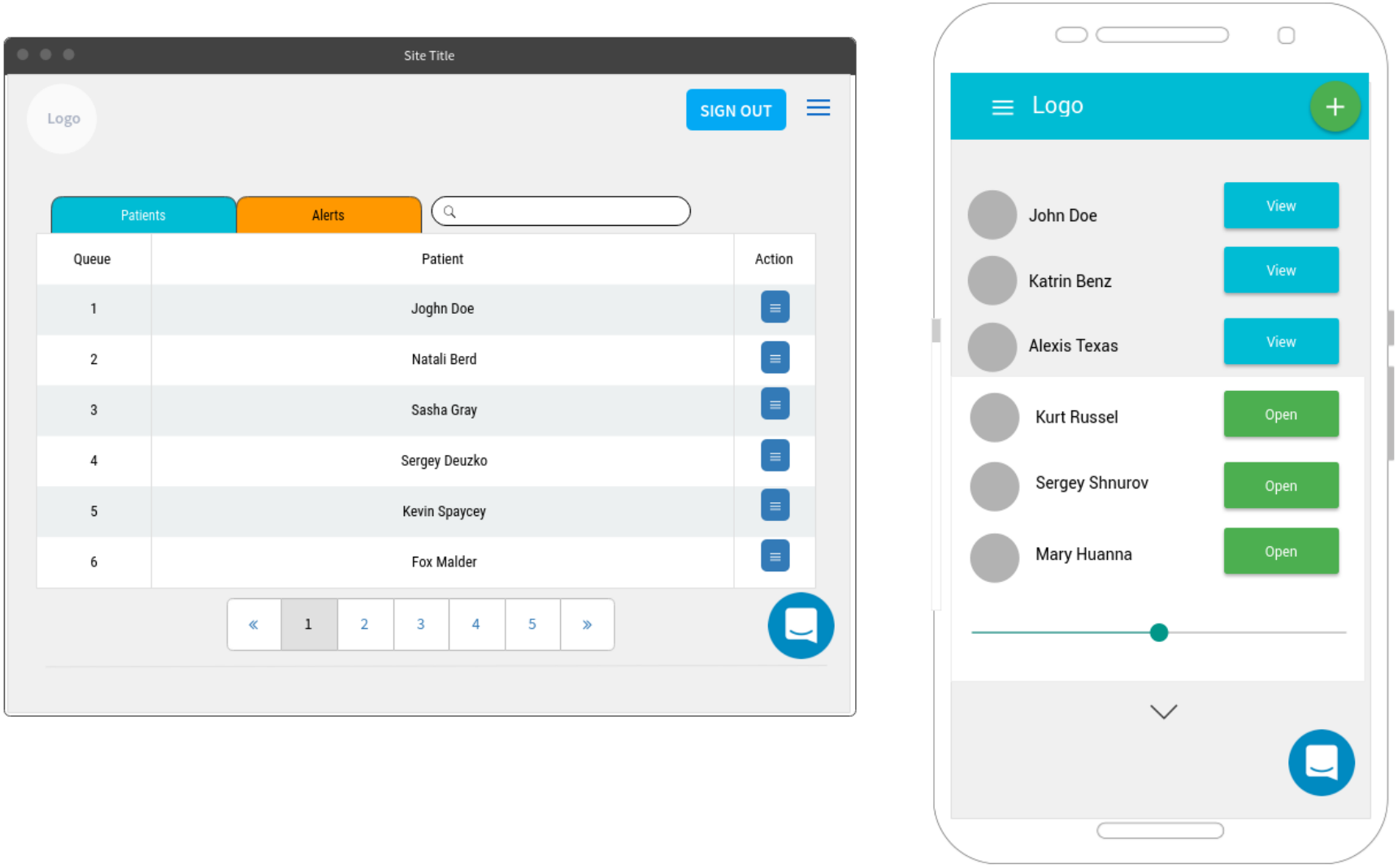

Рисунок 6.18 – Интерфейс главной страницы приложения врача

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ

7.1 Характеристика разрабатываемого продукта

Темой дипломного проекта является разработка программного обеспечения для составления медицинского учреждения, предоставляющего возможность отчётности и анализа пациентов.

Приложение должно предоставлять пользователю графический интерфейс. В данном разделе производится расчет экономической эффективности от разработки и внедрения системы.

Разрабатываемое приложение относится к категории программного обеспечения общего назначения. Данное программное средство относится к 1-й категории сложности. По степени новизны программное средство относится к группе «В»[8].

7.2 Расчёт сметы затрат и цены программного продукта

Основная заработная плата исполнителей проекта определяется по формуле.

$$З_о = \sum_{i=1}^{n} Т_{чi} \cdot Т_ч \cdot Ф_{эi} \cdot К, \quad (1)$$

где n – количество исполнителей, занятых разработкой ПС;

$Т_{чi}$ – часовая тарифная ставка i-го исполнителя (руб.);

$Ф_{Эi}$ – эффективный фонд рабочего времени i-го исполнителя (дней)

$Т_ч$ – количество часов работы в день (ч);

К – коэффициент премирования (1,5).

Примем тарифную ставку 1-го разряда равной 190 руб. Среднемесячная норма рабочего времени составляет 168 часов. Часовой тарифный оклад руководителя проекта составляет 190*4,64/168= 5,25 руб. Часовой тарифный оклад инженера программиста составляет 190*4,06/168= 4,59 руб. Часовой тарифный оклад тестировщика составляет 190*2,27/168 = 2,57 руб.

В таблице 7.1 представлен расчет основной заработной платы.

Таблица 7.1 - Расчет основной заработной платы

Исполнитель	Раз-ряд	Тарифный коэффи-циент	Месячная тарифная ставка, руб.	Часовая тарифная ставка, руб.	Плановый фонд рабочего времени, дн.	Заработная плата, руб.
Руководитель-проекта	12	4,64	881,6	5,25	30	1202,182
Инженер-программист	10	4,06	771,4	4,59	75	2629,773
Тестировщик	5	2,27	431,3	2,57	60	1176,273
Основная заработная плата						7512,35

Дополнительная заработная плата рассчитывается по формуле:

$$З_д = \frac{З_о \cdot Н_д}{100}, \quad (2)$$

где $Н_д$ – норматив дополнительной заработной платы, % ($Н_д$ = 15%).

$$З_д = 7512{,}35 \cdot \frac{15}{100} = 1126{,}85 \text{ руб.}$$

Отчисления на социальные нужды включают в предусмотренные законодательством отчисления в фонд социальной защиты (34%) и фонд обязательного страхования (0,6%) в процентах от основной и дополнительной заработной платы и рассчитываются по формуле:

$$Р_{соц} = \frac{З_о + З_д}{100} \cdot Н_{соц} \quad (3)$$

$$Р_{соц} = \frac{(7512{,}35 + 1126{,}85)}{100} \cdot 34{,}6 = 2989{,}16 \text{ руб.}$$

Расходы по статье «Машинное время» ($Р_{мв}$) включают оплату машинного времени, необходимого для разработки и отладки ПС, и определяются по формуле:

$$Р_{мв} = Ц_м \cdot Т_ч \cdot С_р \quad (4)$$

где Цм – цена одного машино-часа;
Тч – количество часов работы в день;
Ср – длительность проекта.

Стоимость машино-часа на предприятии составляет 1,5 руб. Разработка проекта займет 75 дней. Определим затраты по статье «Машинное время»:

$$Р_{Мв} = 1{,}5 \cdot 8 \cdot 75 = 900{,}00 \text{ руб.}$$

Расходы по статье «Прочие затраты» включают затраты на приобретение специальной научно-технической информации и специальной литературы. Определяются в процентах к основной заработной плате:

$$P_{Пз} = \frac{З_о \cdot Н_{Пз}}{100}, \quad (5)$$

где $Н_{пз}$ – норматив прочих затрат в целом по организации.

$$Р_{Пз} = \frac{7512{,}35 \cdot 50}{100} = 3756{,}18 \text{ руб.}$$

Общая сумма расходов по всем статьям на ПО представляет полную себестоимость ПО:

$$С_п = Р_{мв} + З_о + З_д + Р_{соц} + Р_{пз}. \quad (6)$$

$С_п$ = 900,00+ 7512,35+1126,85+2989,16 +3756,18= 16284,54 руб.

Для определения цены ПО необходимо рассчитать плановую прибыль.

Прибыль рассчитывается по формуле:

$$П_о = \frac{С_п \cdot У_р}{100}, \quad (7)$$

где $П_о$ – плановая прибыль от реализации ПО, руб;

$У_р$ – уровень рентабельности ПО.

$$П_о = \frac{16284{,}54 \cdot 20}{100} = 3256{,}91 \text{ руб.}$$

Прогнозируемая цена ПО без налогов $Ц_п = С_п + П_о = 16284{,}54 + 3256{,}91 = 19541{,}45$ руб.

Отпускная цена (цена реализации) ПО включает налог на добавленную стоимость:

$$Ц_{от} = С_п + П_о + НДС \quad (8)$$

$$НДС = \frac{Ц_п \cdot Н_{дс}}{100} \quad (9)$$

где $Н_{дс}$ – ставка налога на добавленную стоимость.

$$НДС = \frac{19541{,}45 \cdot 20}{100} = 3908{,}29 \text{ руб.}$$

$$Ц_{от} = 16284{,}54 + 3256{,}91 + 3908{,}29 = 23449{,}74 \text{ руб.}$$

Прибыль от реализации ПС за вычетом налога на прибыль ($Н_п$) остается организации разработчику и представляет собой экономический эффект от создания нового программного средства (чистая прибыль):

$$П_ч = П_о \cdot \left(1 - \frac{Н_п}{100}\right) \quad (10)$$

где $Н_п$ – ставка налога на прибыль ($Н_п$ = 18%).

$$П_ч = 3256{,}91 \cdot \left(1 - \frac{18}{100}\right) = 2670{,}67 \text{ руб.}$$

Все расчеты себестоимости и прибыли можно свести в таблицу 7.2.

Таблица 7.2 - Расчет себестоимости и прибыли ПО

Наименование статей	Усл. обозн.	Значение (руб)	Методика расчёта
1	2	3	4
Основная заработная плата исполнителей	$З_о$	7512,35	Определяется на основании расчетов
Дополнительная заработная плата исполнителей	$З_д$	1126,85	$З_д = \frac{З_о \cdot Н_д}{100}$
Отчисления в фонд социальной защиты населения	$Р_{соц}$	2989,16	$Р_{соц} = \frac{З_о + З_д}{100} \cdot Н_{соц}$
Машинное время	$Р_{мв}$	900,00	Определяется на основании расчета.
Прочие прямые расходы	$Р_{Пз}$	3756,18	$Р_{Пз} = \frac{З_о \cdot Н_{Пз}}{100}$
Полная себестоимость	$С_п$	16284,54	$С_п = Р_{мв} + З_о + З_д + Р_{соц} + Р_{Пз}$
Прогнозируемая прибыль	$П_о$	3256,91	$П_о = \frac{С_п \cdot У_р}{100}$
Прогнозируемая цена без налогов (цена предприятия)	$Ц_п$	19541,45	$Ц_п = С_п + П_о$
Налог на добавленную стоимость (НДС)	НДС	3908,29	$НДС = \frac{Ц_п \cdot Н_{дс}}{100}$
Прогнозируемая отпускная цена	$Ц_{от}$	23449,74	$Ц_{от} = С_п + П_о + НДС$

7.3 Расчет экономического эффекта ПО для свободной реализации на рынке

Расчёт цены на одну копию(лицензию) ПО. Цена формируется на основе затрат на разработку и реализацию ПО (затраты на реализацию можно принять в пределах 5-10% от затрат на разработку) и запланированного уровня рентабельности (Ур).

$$Ц = \frac{З_р}{N} + Пед + НДС, \quad (11)$$

где Ц – цена реализации одной копии (лицензии) ПО (руб.);

Зр – сумма расходов на разработку и реализацию (руб.);

N – количество копий (лицензий) ПО, которое будет куплено клиентами за год; N=5

Пед – прибыль, получаемая организацией-разработчиком от реализации одной копии программного продукта (руб.);

НДС – сумма налога на добавленную стоимость (руб.).

$$П_{ед} = \frac{Сп \cdot Ур}{N \cdot 100\%}, \quad (12)$$

$$Пед = \frac{162845{,}40 * 0{,}2}{5} = 6513{,}80 \text{ руб.}$$

Расчёт цены на одну копию(лицензию) ПО:

$$Ц = \frac{16284{,}54 + 3908{,}29}{5} + 6513{,}80 = 10552{,}37$$

Общие капитальные вложения ($К_о$) потребителя, связанные с приобретением, внедрением и использованием ПО рассчитываются по формуле:

$$К_о = К_{пр} + К_с + К_{ос} \quad (13)$$

где $К_{пр}$ – затраты пользователя на приобретение по цене на одну копию руб;

$К_с$ – затраты пользователя на оплату услуг по сопровождению ПО (10% от $К_{пр}$), руб;

$К_{ос}$ – затраты пользователя на освоение ПО (10% от $К_{пр}$), руб;

$$К_{о} = 10552{,}37 \cdot (1 + 0{,}10 + 0{,}1) = 12662{,}84 \text{ руб}$$

А суммарная годовая прибыль по проекту в целом будет равна:

$$П = Пед \text{ x } N. \quad (14)$$

$$П = 6513{,}80 * 5 = 32569{,}0 \text{ руб}$$

Для пользователя в качестве экономического эффекта выступает лишь чистая прибыль, которая остается в распоряжении предприятия:

$$\Delta П_{ч} = П(1 - \frac{Н_{п}}{100}) \quad (15)$$

где Н_п – налог на прибыль, Н_п=18%.

$$\Delta П_{ч} = 32569{,}0 \ * (1 - 0{,}18) = 26706{,}70 \text{ руб.}$$

Расчет экономического эффекта за весь период использования ПО (4 года) целесообразно представить в таблице (Таблица 7.3).

Полученные суммы результата (чистой прибыли) и затрат (капительных вложений) по годам необходимо привести к единому моменту времени – расчетному году (2017) путем умножения результатов и затрат на коэффициент дисконтирования α_t, который рассчитывается по формуле:

$$\alpha_t = \frac{1}{(1 + E_H)^{t_i - t_p}} \quad (16)$$

где E_H – норма дисконта в долях единицы;

t_i – порядковый номер года, результаты и затраты которого приводятся к расчетному году;

t_p – расчетный год (t_p= 1).

$$\alpha_{t1} = \frac{1}{(1 + 0{,}15)^{1-1}} = 1$$

$$\alpha_{t2} = \frac{1}{(1+0{,}15)^{2-1}} = 0{,}870$$

$$\alpha_{t3} = \frac{1}{(1+0{,}15)^{3-1}} = 0{,}756$$

$$\alpha_{t4} = \frac{1}{(1+0{,}15)^{4-1}} = 0{,}658$$

Таблица 7.3 – Расчет экономического эффекта от использования нового ПО

Показатели	Усл. обозн.	Ед. изм.	Годы			
			2017	2018	2019	2020
1	2	3	4	5	6	7
Результаты:						
Прирост чистой прибыли	$\Delta П_ч$	руб.	26706,70	26706,70	26706,70	26706,70
С учетом фактора времени	$\Delta П_ч \cdot \alpha_t$	руб.	26706,70	23234,8	20190,3	17573,0
Затраты						
Приобретение ПО	$К_{пр}$	руб.	10552,37	-	-	-
Освоение ПО	$К_{ос}$	руб.	1055,23	-	-	-
Сопровождение ПО	$К_с$	руб.	1055,23	-	-	-
Всего затрат	$К_о$	руб.	12662,84	-	-	-
Тот же с учетом фактора времени	$К_о \cdot \alpha_t$	руб.	12662,84	-	-	-
Превышение результата над затратами	$P_t \cdot \alpha_t - З_t$	руб.	14043,86	23234,8	20190,3	17573,0
Тот же с нарастающим итогом			14043,86	32278,66	57468,96	75041,96
Коэффициент приведения	α_t		1	0,870	0,756	0,658

Так как чистый дисконтированный доход больше нуля, то проект эффективен, т.е. инвестиции в разработку данного ПО экономически целесообразны.

Рассчитаем рентабельность инвестиций в разработку и внедрение программного продукта ($Р_И$) по формуле

$$Р_и = \frac{П_{чср}}{З} \cdot 100\% \quad (17)$$

где $П_{чср}$ - среднегодовая величина чистой прибыли за расчетный период, руб., которая определяется по формуле

$$П_{чср} = \frac{\sum_{i=1}^{n} П_{чt}}{n} \quad (18)$$

где $П_{чt}$ - чистая прибыль, полученная в году t, руб.

$$Пчср = \frac{26706{,}70 + 23234{,}80 + 20190{,}30 + 17573{,}0}{4} = 21925{,}50 \text{ руб}$$

$$Ри = \frac{21925{,}50}{12662{,}84} * 100\% = 173{,}1\ \%$$

7.4 Выводы

В результате технико-экономического обоснования применения программного продукта были получены следующие значения показателей их эффективности:

– чистый дисконтированный доход за четыре года работы программы составит 21925,5 руб;

– затраты на разработку программного продукта окупятся на первый же год его использования;

– рентабельность инвестиций составляет 173, 1%.

Таким образом, применение программного продукта является эффективным и инвестиции в его разработку целесообразно осуществлять.

ЗАКЛЮЧЕНИЕ

Целью дипломного проектирования была разработка автоматизированной системы управления медицинского учреждения. Разработанная система способна поддерживать не менее 20 000 пользователей на круглосуточной основе. При этом передача всех личных данных происходит в зашифрованном виде.

Разработанная автоматизированная система полностью отвечает всем требованиям задания на разработку проекта. При разработке были рассмотрены и проанализированы существующие решения автоматизации, необходимые литературные и интернет-источники, на основе которых были выбраны средства проектирования.

Далее была разработана модель базы данных системы и приложения, которые смогли бы осуществлять взаимодействие с ней, предоставляя пользователям полноценный доступ к системе.

На конечной стадии были разработаны интерфейсы пользователей для двух ролей: персонала медучреждения и пациентов.

Предложенные в проекте концепции имеют целью создать достаточно полную модель информационных взаимодействий в рамках выбранной предметной области (здравоохранения), несмотря на объективные временные ограничения по реализации самого проекта. В реальной среде и условиях промышленной эксплуатации могут потребоваться дополнительные временные и другие затраты для адаптации выбранных моделей к условиям реальных потребностей.

Тем не менее, в стартовой версии проекта в соответствии с заданием были разработаны алгоритмы, модели данных и интерфейсы, которые, в конечном итоге, дали на выходе функционально законченное приложение, имеющее прикладное значение.

Характеристики практического внедрения проекта отражены в разделе технико-экономического обоснования разработанной системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Медицински новости [Электронный ресурс]. – Режим доступа : http://www.mednovosti.by/.

[2] Cyberleninka [Электронный ресурс]. – Режим доступа : http:// http://cyberleninka.ru/

[3] Хабрахабр [Электронный ресурс]. – Режим доступа : http:// habrahabr.ru /.

[4] MSDN [Электронный ресурс]. – Режим доступа : https://msdn.microsoft.com/ru-ru/.

[5] Tproger [Электронный ресурс]. – Режим доступа : https://tproger.ru/.

[6] W3School [Электронный ресурс]. – Режим доступа : https://www.w3schools.com/.

[7] Википедия [Электронный ресурс]. – Режим доступа : http://www.ru.wikipedia.org/.

[8] Носенко, А. Технико-экономическое обоснование дипломных проектов. Ч.2: методическое пособие / А. А. Носенко, А. В. Грицай – Минск: БГУИР, 2002. – 57 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода (к пункту 6)

```
var ESR =
/******/ (function(modules) { // webpackBootstrap
/******/  // The module cache
/******/  var installedModules = {};
/******/  // The require function
/******/  function __webpack_require__(moduleId) {
/******/       // Check if module is in cache
/******/       if(installedModules[moduleId])
/******/            return installedModules[moduleId].exports;
/******/       // Create a new module (and put it into the
cache)
/******/       var module = installedModules[moduleId] = {
/******/            exports: {},
/******/            id: moduleId,
/******/            loaded: false
/******/       };
/******/       // Execute the module function
/******/       modules[moduleId].call(module.exports, module,
module.exports, __webpack_require__);
/******/       // Flag the module as loaded
/******/       module.loaded = true;
/******/       // Return the exports of the module
/******/       return module.exports;
/******/  }
/******/  // expose the modules object (__webpack_modules__)
/******/  __webpack_require__.m = modules;
/******/  // expose the module cache
/******/  __webpack_require__.c = installedModules;
/******/  // __webpack_public_path__
```

```
/******/  __webpack_require__.p = "";
/******/  // Load entry module and return exports
/******/  return __webpack_require__(0);
/******/ })
/************************************************************************
*********/
/******/ ([
/* 0 */
/***/ function(module, exports, __webpack_require__) {

var esrLiveView = __webpack_require__(1);
var liveviewUI = __webpack_require__(2);
var styles = __webpack_require__(12);
var utils = __webpack_require__(17);
var tracker = __webpack_require__(18);

module.exports = {
esrLiveView: esrLiveView,
liveviewUI: liveviewUI,
utils: utils,
tracker: tracker
};



/***/ },
/* 1 */
/***/ function(module, exports, __webpack_require__) {

'use strict';

var liveviewUI = __webpack_require__(2);
var saver = __webpack_require__(8);
```

```
var mobiles = __webpack_require__(11);

var classhelper = __webpack_require__(6);

module.exports = function (opts) {

var liveDoc = opts.liveDoc;

var context = opts.context;

var viewId = opts.viewId;

//var invitationId = opts.invitationId;

var invites = opts.invites;

var viewOnlineButtonId = 'liveDoc_button_online_' + viewId;

var viewOfflineButtonId = 'liveDoc_button_offline_' + viewId;

//var viewInviteButtonId = 'liveDoc_invite_button_' +
invitationId;

var internalViewContainerId = 'esr_view_container';

var liveDocHostUrl = opts.liveDocHostUrl;

var isViewPopupped = false;

var invited = false;

var LiveDocInviteEventHandler = function (invitationId,
connector, liveviewUI) {

var self = this;

self.connector = connector;

self.liveviewUI = liveviewUI;

self.invitationId = invitationId;

self.handle = function (args) {

console.log(args);

if (args == liveDoc.BUTTON_EVENT.BUTTON_AVAILABLE) {

if (self.connector.isOnline) {

console.log('Triggered Invite : ' + invitationId);

invited = true;

if (liveviewUI.getState() === liveviewUI.UI_STATES.NOT_STARTED
&& !liveviewUI.isInviteVisible()) {

liveviewUI.setTriggeredInvite(invitationId);

// liveviewUI.hideLiveviewButton();
```

```
// liveviewUI.showLiveviewInvite();
var buttons = document.getElementsByClassName('liveview-
button');
if (buttons && buttons[0]) {
classhelper.addClass(buttons[0], 'esr-liveview-
blink_animation');
}
}
}
}
if (args == liveDoc.BUTTON_EVENT.BUTTON_REJECTED) {
liveviewUI.hideLiveviewInvite();
liveviewUI.invitationRejected();
liveviewUI.showLiveviewButton();
}
}
}
var LiveDocConnector = function (liveDoc, opts) {
var self = this;
var context = opts.context;
var eventHandlers = {};
var state = {};
function b64toBlob(b64Data, contentType, sliceSize, fileName) {
contentType = contentType || '';
sliceSize = sliceSize || 512;
var byteCharacters = atob(b64Data);
var byteArrays = [];
for (var offset = 0; offset < byteCharacters.length; offset +=
sliceSize) {
var slice = byteCharacters.slice(offset, offset + sliceSize);
var byteNumbers = new Array(slice.length);
for (var i = 0; i < slice.length; i++) {
```

```
byteNumbers[i] = slice.charCodeAt(i);
}
var byteArray = new Uint8Array(byteNumbers);
byteArrays.push(byteArray);
}
var blob = new Blob(byteArrays, { type: contentType });
saver.saveAs(blob, fileName);
self.postEvent('deleteLiveViewLog', {});
//deleteLogLiveView();
}
var messageListnener = {
load: function (data) {
console.log('LOAD');
},
preViewLoad: function (data) {
console.log('PRE_VIEW_LOAD');
if (!opts.newWindow) {
liveviewUI.showIframe();
}
},
preViewCancel: function (data) {
console.log('PRE_VIEW_CANCEL');
if (!opts.newWindow) {
liveviewUI.setState(liveviewUI.UI_STATES.NOT_STARTED);
}
},
chasitorDocViewEnded: function (data) {
console.log('DOC_VIEW_ENDED');
if (data) {
console.log(data);
}
```

```
},
chasitorDocViewMessage: function (data) {
console.log('DOC_VIEW_MESSAGE');
if (data) {
console.log(data);
}

if (!opts.newWindow) {
liveviewUI.incrementUnread();
}
},

chasitorDocViewTransferred: function (data) {
console.log('DOC_VIEW_TRANSFERRED');
if (data) {
console.log(data);
}
},

transferToButtonInitiated: function (data) {
console.log('DOC_VIEW_TRANSFER_TO_BUTTON_INITIATED');
if (data) {
console.log(data);
}
},
chasitorDocDisconnected: function (data) {
console.log('DOC_DISCONNECTED');
if (data) {
console.log(data);
}
},
```

```
chasitorDocJoinedConference: function (data) {
console.log('DOC_JOINED_CONFERENCE');
if (data) {
console.log(data);
}
},
chasitorDocLeftConference: function (data) {
console.log('DOC_LEFT_CONFERENCE');
if (data) {
console.log(data);
}
},
chasitorDocTypingUpdate: function (data) {
console.log('DOC_TYPING_UPDATE');
if (data) {
console.log(data);
}
},
chasitorChasitorViewCanceled: function (data) {
console.log('CHASITOR_VIEW_CANCELED');
if (data) {
console.log(data);
}
},
chasitorChasitorViewEnded: function (data) {
console.log('CHASITOR_VIEW_ENDED');
if (data) {
console.log(data);
}
},
chasitorChasitorViewMessage: function (data) {
```

```
console.log('CHASITOR_VIEW_MESSAGE');
if (data) {
console.log(data);
}
},
chasitorIdleTimeout: function (data) {
console.log('CHASITOR_IDLE_TIMEOUT');
if (data) {
console.log(data);
}
},
chasitorViewEstablished: function (data) {
console.log('VIEW_ESTABLISHED');
if (!opts.newWindow) {
liveviewUI.setState(liveviewUI.UI_STATES.ON_VIEW);
if (!liveviewUI.isViewVisible()) {
liveviewUI.hideLiveviewButton();
liveviewUI.showView();
}
}
},
chasitorViewRequestFailed: function (data) {
console.log('VIEW_REQUEST_FAILED');
if (data) {
console.log(data);
}
},
chasitorViewRequestSuccessful: function (data) {
console.log('VIEW_REQUEST_SUCCESSFUL');
if (data) {
console.log(data);
```

```
}
if (!opts.newWindow) {
state.isViewRequestSuccessfull = true;
}
},
clearChasitorIdleTimeout: function (data) {
console.log('CLEAR_CHASITOR_IDLE_TIMEOUT');
if (data) {
console.log(data);
}
},
chasitorConnectionError: function (data) {
console.log('CONNECTION_ERROR');
if (data) {
console.log(data);
}
},deploymentSettings: function (data) {
console.log('DEPLOYMENT_SETTINGS');
if (data) {
console.log(data);
}
},
fileTransferCanceled: function (data) {
console.log('FILE_TRANSFER_CANCELED');
if (data) {
console.log(data);
}
},
fileTransferFailure: function (data) {
console.log('FILE_TRANSFER_FAILURE');
if (data) {
```

```
console.log(data);
}
},
fileTransferRequested: function (data) {
console.log('FILE_TRANSFER_REQUESTED');
if (data) {
console.log(data);
}
},
fileTransferSuccess: function (data) {
console.log('FILE_TRANSFER_SUCCESS');
if (data) {
console.log(data);
}
},
postView: function (data) {
console.log('POST_VIEW');
if (data) {
console.log(data);
liveDocHostUrl = data.url;
}
if (!opts.newWindow) {
liveviewUI.setState(liveviewUI.UI_STATES.ON_POST_VIEW);
setTimeout(function() {
liveviewUI.showView();
liveviewUI.hideLiveviewButton();
}, 500);
}
},
postViewFinished: function (data) {
console.log('POST_VIEW_FINISHED');
```

```
if (!opts.newWindow) {
liveviewUI.setState(liveviewUI.UI_STATES.ON_POST_VIEW_FINISHED);
}
},
chasitorQueueUpdate: function (data) {
console.log('QUEUE_UPDATE');
if (data) {
console.log(data);
}
},
chasitorReconnecting: function (data) {
console.log('RECONNECTING');
if (data) {
console.log(data);
}
},
resetViewMessages: function (data) {
console.log('RESET_VIEW_MESSAGES');
if (data) {
console.log(data);
}
},
startChasitorIdleWarning: function (data) {
console.log('START_CHASITOR_IDLE_TIMEOUT_WARNING');
if (data) {
console.log(data);
}
},
updateChasitorIdleWarning: function (data) {
console.log('UPDATE_CHASITOR_IDLE_TIMEOUT_WARNING');
if (data) {
```

```
console.log(data);
}
},
viewLogSaved: function (fileContentStr) {
console.log('viewLogSaved');
var contentType = 'text/pdf';
b64toBlob(fileContentStr, contentType, null, 'Eurostar
View.pdf');
},
viewWindowMinimized: function (arg) {
console.log('viewWindowMinimized');
},
viewWindowFocused: function (arg) {
console.log('viewWindowFocused');
if (opts.newWindow) {
liveviewUI.showView();
}
},
viewAlreadyOpen: function () {
console.log('viewAlreadyOpen');
liveviewUI.closeAlreadyOpen();
},
dynamicFAQStatistics: function (data) {
console.log('DYNAMIC_FAQ_STATISTICS');
if (data) {
console.log(data);
}
},
resetView: function() {
console.log('resetView');
if (!opts.newWindow) {
liveviewUI.resetView();
```

Перечень оборудования

Ведомость дипломного проекта