# Архитектура

Программа будет создаваться на языке C# с использованием среды разработки Microsoft Visual Studio 2019 на платформе .NET Framework 4.7.2 с использованием интерфейса программирования приложений Windows Forms.

Выбор языка обусловлен объектно-ориентированным подходом к написанию программы. На данный момент язык C# является одним из самых популярных, удобных и полноценных среди объектно-ориентированных языков программирования.

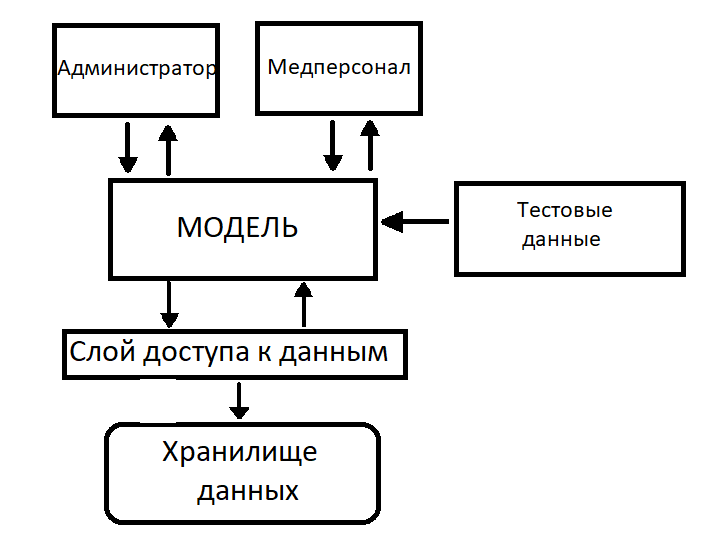
В процессе проектирования рассматривались 2 платформы, с которыми язык С# имеет наилучшее взаимодействие, а именно .NET Core и .NET Framework. Наш выбор пал на последнюю по причине увеличенного количества управляющих элементов и более высокого уровня развития платформы. К сожалению, это означает что приложение будет работать исключительно на машинах с операционной системой Windows.

Для реализации пользовательского интерфейса было решено использовать графический подход, так как он простой для восприятия и интуитивно понятен. Мы будем использовать Windows Forms API который является частью платформы .NET Framework. С его помощью у нас появиться возможность создать полноценный интерфейс для пользователей приложения.

# Структура проекта

Будет разумно поделить проект, описанный в спецификации, на два отдельных приложения: для администратора и для врачей, поскольку функции этих типов пользователей между собой не пересекаются. Кроме того, для тестирования программы на этапе программирования уместно создать несколько тестовых данных для проверки правильности работы программы.

На этом основании можем составить следующею структуру проекта.



# Объектная модель

## Предисловие

В этом пункте мы опишем основные объекты этого проекта. Жирным шрифтом будем выделять классы и их колекции, через двоеточие записывать их поля, словами «список» и «база» будем характеризовать коллекцию указанных классов. Множественные типы будут записаны через тире с указанием коллекции неких классов.

## Определение основных типов объектов

В **базе медикаментов** имеются **препараты.**

**Администратор** дополняет **список пользователей**, добавляет **препараты**, регистрирует **поставки**.

**Врач** имеет доступ **к базе пациентов**, **базе медикаментов, базе болезней**. Способен редактировать информацию о **пациенте**.

## Определение множественных пользовательских типов

**База медикаментов** — коллекция **препаратов.**

**Список пользователей —** коллекция **пользователей.**

**База пациентов** — коллекция **пациентов.**

**База болезней —** коллекция **болезней.**

**Список порций —** коллекция **порций.**

**База поставок —** коллекция **поставок**.

## Определение единичных пользовательских типов

**Препарат**: идентификатор, название, изображение, описание.

**Пользователь**: должность, логин, пароль.

**Пациент**: имя, адрес, дата рождения, пол, история болезни, назначенный рецепт.

**Болезнь**: коллекция симптомов, описание, способы лечения, информация о распространении.

**Порция: препарат**, количество.

**Поставка:** дата, **список порций**.

**Медучреждение**: **список пользователей**, **база медикаментов**,  **база пациентов**, **база болезней**, **база поставок.**

## Методы:

На данном этапе к главным методам мы можем отнести:

1. Процедура приема пациента ()
2. Регистрация новых пользователей программы ()
3. Учёт поставок медикаментов ()
4. Просмотр и изменение информации о пациенте ()

Более подробное описание методов можно посмотреть в спецификации к проекту.

## Диаграмма классов

В диаграмме использовались отношения обобщения(стрелка), агрегации(ромб) и ассоциации(просто линия).

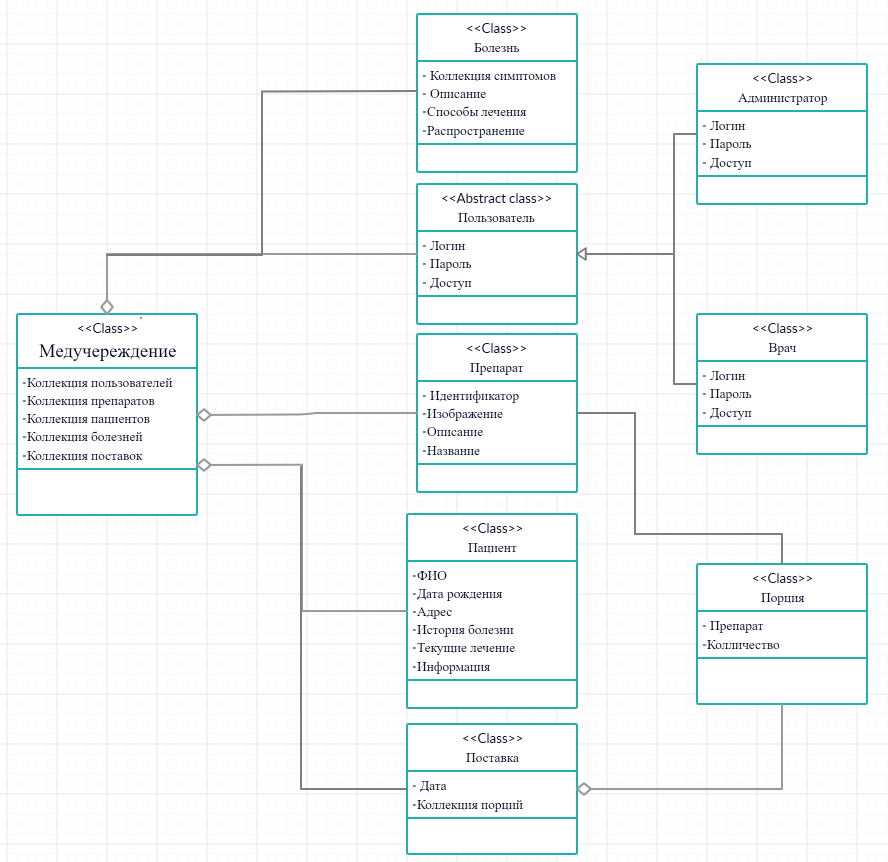


Диаграмма 1. Диаграмма классов объектной модели

# Хранилище и доступ к данным

При выходе из программы происходит сохранение всех множественных пользовательских объектов для дальнейшего возобновления работы. Поэтому, необходимо выбрать способ хранения данных в системе.

Мы остановились на сохранении в виде бинарных файлах, так как оно создает файлы приемлемых размеров и не позволяет рядовым пользователям прочитать содержимое файла. В проекте не рассматриваются вопросы сохранности данных, следственно этот способ нам подходит. Данные будут переводиться из текстовых в поток байтов при помощи сериализации с использованием встроенной библиотеки BinaryFormatter и сохранятся в корневой папке проекта.

# План выполнения проекта

1. Составление спецификации (до 12.04)
2. Проектирование (до 20.04)
3. Программирование модели (до 26.04)
4. Программирование пользовательского интерфейса (до 17.05)
5. Написание пояснительной записки (до 31.05)

Тестирование провидится параллельно с программированием пользовательского интерфейса