МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Факультет вычислительной математики и кибернетики**

**Кафедра: МОЭВМ**

Направление подготовки: «Фундаментальная информатика и информационные технологии»

Магистерская программа: «Компьютерная графика»

**КУРСОВАЯ РАБОТА**

**Тема:**

**«Система обмена и хранения файлов стандарта DICOM»**

Допущена к защите Выполнил:

Заведующий кафедрой: студент группы 85м22

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Реутов Игорь Олегович

ученая. степень, ученое звание, ф.и.о.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_одпись подпись

Научный руководитель:

Доцент К.Т.Н, Борисов Н.А.

ученая степень, ученое звание, ф.и.о.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

Нижний Новгород  
2016

Оглавление

[Введение 3](#_Toc441736403)

[1. Постановка задачи 5](#_Toc441736404)

[2.Технологии 6](#_Toc441736405)

[2.1 Клиент-Сервер 6](#_Toc441736406)

[2.2 Облачные вычисления 7](#_Toc441736407)

[3. Шифрование 12](#_Toc441736408)

[3.1 Методы шифрования 13](#_Toc441736409)

[3.2 MD5 14](#_Toc441736410)

[4. DICOM 15](#_Toc441736411)

[4.1 Стандарт DICOM 15](#_Toc441736412)

[4.2 DICOM сервисы 17](#_Toc441736413)

[4.3 DICOM информационные уровни 18](#_Toc441736414)

[5. План работы 21](#_Toc441736415)

[6. Результаты 23](#_Toc441736416)

[7. Вывод 28](#_Toc441736417)

[8. Литература 29](#_Toc441736418)

[Приложение 30](#_Toc441736419)

# Введение

В современном мире существует тенденция к упрощению способов безопасной передачи данных из одной точки в другую или сразу в несколько. При этом пользователь не всегда желает, чтобы переданные им данные мог видеть каждый. Для таких операций существует множество файло-обменников, специальных клиент-серверных приложений с регистрацией, облачные сервисы.

Файлообменник, файлхостинг или файловый хостинг — сервис, предоставляющий пользователю место под его файлы и круглосуточный доступ к ним через web, как правило по протоколу http. Такой сервис позволяет удобно «обмениваться» файлами. На специальной странице файлообменника (чаще всего на главной) пользователь загружает файл на сервер файлообменника, а файлообменник отдает пользователю постоянную ссылку, которую он может рассылать по e-mail, публиковать в блогах, на форумах или пересылать через системы IM. Неудобства файлообменника состоят в том, что пользователь должен разослать ссылку на залитые им данные.

Технология «Клиент-Сервер» широко используется в наше время. Она описывает взаимодействие между двумя компьютерами, согласно которому клиент запрашивает у сервера некоторые услуги, а сервер обслуживает запрос. Архитектура «клиент-сервер» определяет общие принципы организации взаимодействия в сети, где имеются серверы, узлы-поставщики некоторых специфичных функций (сервисов) и клиенты, потребители этих функций. В любой сети (даже одно-ранговой), построенной на современных сетевых технологиях, присутствуют элементы клиент-серверного взаимодействия, чаще всего на основе двухзвенной архитектуры. Двухзвенной (2-tier) она называется из-за необходимости распределения трех базовых компонентов между двумя узлами (клиентом и сервером). Также выделяют трехзвенные (3-tier) и многозвенные архитектуры (Multi-tier). Ограниченность данной технологии состоит в том, что для работы с сервером необходимо клиентское приложение, а также необходимо быть зарегистрированным на сервере, иначе сервер откажется обрабатывать клиентские запросы без аутентификации. Также пользователь может разослать свои данные только тем, кто зарегистрированы на сервере. При всем этом, сервер может иметь специальные методы обработки данных, перед тем, как разослать их выбранным пользователям. Иными словами, сервер имеет некоторый функционал, в отличии от файлообменника, на который можно только загрузить файл.

«Облачные технологии» в последнее время активно разрабатываются и внедряются в нашу жизнь. Суть концепции облачных технологий заключается в предоставлении конечным пользователям удаленного динамического доступа к услугам, вычислительным ресурсам и приложениям (включая операционные системы и инфраструктуру) через интернет. Развитие сферы хостинга было обусловлено возникшей потребностью в программном обеспечении и цифровых услугах, которыми можно было бы управлять изнутри, но которые были бы при этом более экономичными и эффективными за счет экономии на масштабе. Большинство сервис-провайдеров предлагают облачные вычисления в форме VPS-хостинга, виртуального хостинга, и ПО-как-услуга(SaaS). Облачные услуги долгое время предоставлялись в форме SaaS, например, Microsoft Hosted Exchange и SharePoint. Нельзя не признать, что технологии облачных вычислений имеют огромнейший потенциал, потому что все современные компьютерные продукты постоянно увеличивают свои требования к техническому оснащению компьютера пользователя, что неизбежно ведет к значительным затратам на апгрейд. Особенно требовательной к системным ресурсам становится игровая индустрия. Так что данная технология позволяет решить проблему чрезмерной требовательности приложений к ресурсам конечного пользователя.

В данной работе будет рассмотрена задача построении клиент-серверного приложения для передачи файлов стандарта DICOM. Данный сервис предназначен для хранения файлов данного стандарта, а также для их передачи, по запросу пользователя, имеющий право на данную операцию. С помощью данного сервиса пациент может просмотреть кардиограммы, рентген снимки и другие медицинские данные. Также будет возможность передать выбранные данные врачам из других медицинских учреждений, для предоставления сторонних заключений. Также некоторые данные пользователь может разрешить для просмотра в качестве учебного материала.

# 1. Постановка задачи

Задача данный работы состоит в написании клиент-серверного приложения для работы и пересылке файлов стандарта DICOM. Пользователь должен иметь возможность просматривать медицинские заключения и анализы, предоставленные врачами, а также иметь возможность отправить медицинские анализы для предоставления заключения врачей из других медицинских организаций. Врачи должны иметь право добавлять новые записи, после анализа предоставленных данных. Приложение должно автоматически показывать содержимое пакеты данных для просто проведения анализа врачами или просмотра заключения пациентом. Медицинские данные могут быть предоставлены студентам в качестве учебного материала, только по согласию владельца данных. Остальные данные должны быть доступны только пользователю, либо врачу, которому они были отосланы. Для реализации данной задачи необходимо написать серверное и клиентское приложение. Для написания будет использован язык C# предоставляемы MS Visual Studio. Для хранения данных будет использован PostrgreSQL, к которому можно подключиться, используя NPGSQL - NET Data Provider, позволяющий приложению, подключиться к Базе Данных PostreSQL при написании кода на C#. Для работы с файлами стандарта DICOM будет использована open-source библиотека Grassroot.

# 2.Технологии

## 2.1 Клиент-Сервер

Клиент-сервер (client-server architecture/topology) — архитектура или организация построения сети (в том числе локальной и распределенной), в которой производится разделение вычислительной нагрузки между включенными в ее состав компьютера, выполняющими функции клиентов, и одной мощной центральной ЭВМ — сервером. Процесс наблюдения за данными отделен от программ, использующих эти данные. Сервер может поддерживать центральную базу данных, расположенную на большом компьютере, зарезервированном для этой цели. Клиентом будет обычная программа, расположенная на любой ЭВМ, включенной в сеть, а также сама ЭВМ, которая по мере необходимости запрашивает данные с сервера. Производительность при использовании клиент-серверной архитектуры выше обычной, поскольку как клиент, так и сервер делят между собой нагрузку по обработке данных. Достоинствами клиент-серверной архитектуры являются большой объем памяти и ее пригодность для решения разнородных задач, возможность подключения большого количества рабочих станций, включая ПЭВМ и пассивные терминалы, а также установки средств защиты от несанкционированного доступа. Основные понятия клиент-серверной архитектуры:

Клиент (client) — сторона (ЭВМ, программа или пользователь), запрашивающая и использующая информацию и/или ресурсы у сервера в среде клиент-сервер.

Тонкий клиент (thin client) — терминал сети без жестких дисков и флоппи-дисководов, вычислительная мощность которого и объем памяти определяются задачами пользователя. Все программы и приложения, хранящиеся на сервере, становятся доступными для пользователя при включении его устройства и выполнении процедуры регистрации на сервере. Тонким клиентом называют также ПК (в том числе и мобильный) с минимизированной мощностью процессора, оперативной и внешней памяти, позволяющий пользователю осуществлять ввод и отображение данных за счет выполнения вычислений и хранения данных на более мощном ПК или сервере, с которыми он может осуществлять связь, при помощи каналов средней пропускной способности. К тонкому клиенту могут подключаться внешние устройства ввода/вывода данных (сканеры, мониторы, принтеры и проекторы).

Брокеры объектных запросов (ORB, Object Request Brokers) — форма промежуточного программного обеспечения для разработки систем клиент-сервер.

Стандартная архитектура брокера объектных запросов (COBRA, Common Object Request Broker Architecture) — стандарт, разработанный группой OMG (Object Management Group), который определяет интерфейсы между объектами сети, позволяющими им работать совместно.

Двухзвенная модель (two-tier model) — архитектура построения сети, предусматривающая один сервер и несколько клиентов, является наиболее простой и распространенной. Недостаток — ограниченное число клиентских рабочих мест.

Трехзвенная модель (three-tier model) — архитектура построения системы клиент-сервер, в которой предусмотрено промежуточное звено (дополнительный компьютер), расположенное между сервером и клиентом двухзвенной модели. Промежуточное звено работает как монитор обработки транзакций или брокер объектных запросов. Трехзвенные модели обеспечивают работу существенно большего числа клиентов, чем двухзвенные модели.

N-звенная модель (N-tier model) — архитектура построения сети, предусматривающая наличие нескольких серверов приложений, число которых определяется необходимым уровнем нагрузки сети. При многозвенной модели построения системы количество возможных клиентских мест значительно больше, чем при использовании двухзвенной модели.

Промежуточное ПО (middleware) — программное обеспечение, выполняющее функцию связи клиента и сервера и предназначенное для содействия обмена данными, в том числе при взаимодействии клиентов с удаленным сервером. Для поддержки промежуточного программного обеспечения может использоваться дополнительный сервер, которому присваивается наименование, связанное с выполняемыми им функциями, например, сервер приложений, сервер баз данных.

Промежуточное ПО, ориентированное на обработку сообщений (MOM, Message-Oriented Middleware) — промежуточное программное обеспечение, которое для обмена данными использует сообщения и очереди. MOM позволяет прикладным программам продолжить обработку прежде, чем завершится обращение к удаленным службам. MOM работает в относительно медленных вычислительных сетях, таких, как PBC (региональные вычислительные сети) и Интернет. Ассоциация MOMA (Message-Oriented Middleware Association) производит разработку стандартов для этого класса средств программного обеспечения.

Разделение программ (application partitioning) — процесс разбиения прикладных программ «клиент-сервер» на части, которые должны выполняться на клиенте, сервере и в некоторых случаях на сервере приложения.

Сервер баз данных (сервер СУБД, database server) — сервер, состоящий из ЭВМ, операционной системы и СУБД. В зависимости от архитектуры построения сети сервер баз данных может являться основным ее сервером или сервером, поддерживающим промежуточное программное обеспечение.

Супервизор (supervisor) — часть операционной системы, координирующей использование ресурсов вычислительной сети и поддерживающей потоки операций, выполняемых центральным процессором. Супервизором называют функциональные обязанности и/или должность системного программиста, ответственного за поддержку работы программного обеспечения локальной сети, распределение доступа к ее информационным ресурсам (базам данных, файлам), а также их защиту от несанкционированного доступа, повреждения и разрушения.

## 2.2 Облачные вычисления

Облачные вычисления — технология распределённой обработки данных, в которой компьютерные ресурсы и мощности предоставляются пользователю как Интернет-сервис. Облако – это новая технология использования серверных ресурсов, помогающая задействовать всю доступную мощность процессоров и объем оперативной памяти, разделяя их между различными независимыми задачами. Иными словами, в облаке одновременно запущено большое количество виртуальных серверов, не мешающих друг другу. И если один или несколько из них перестанет работать из-за каких-либо программных ошибок или иных проблем, остальных серверов это никак не коснется. Как правило, современное облако состоит из большого количества высокопроизводительных серверов, на которых запущены виртуальные машины (серверы), свои для каждого пользователя. Одно из главных преимуществ облака, помимо независимости каждого пользователя от остальных, является возможность плавно регулировать объем используемых ресурсов и, соответственно, оплачивать только те ресурсы, которые действительно требуются для решения задачи. Без переплаты.

### 2.2.1 Характеристики облачных вычислений:

* Самообслуживание по требованию

Потребитель, когда это ему необходимо, может самостоятельно задействовать вычислительные возможности, такие как серверное время или сетевое хранилище данных, в автоматическом режиме, без взаимодействий с персоналом поставщика услуг.

* Широкая доступность через сеть (Интернет)

Возможности доступны через сеть; доступ к ним осуществляется на основе стандартных механизмов, что обеспечивает использование разнородных тонких и толстых клиентских платформ (например, мобильных телефонов, ноутбуков, КПК).

* Объединение ресурсов в пул

Поставщик объединяет свои вычислительные ресурсы в пул для обслуживания большого числа потребителей, используя принцип множественной аренды (Multi-tenancy). Различные физические и виртуальные ресурсы динамически распределяются и перераспределяются в соответствии с потребностями пользователей. Возникает ощущение независимости от местоположения, когда заказчик не знает и не контролирует, где конкретно находятся вычислительные ресурсы, которыми он пользуется, но, возможно, может определить их расположение на более абстрактном уровне (например, страна, регион или дата-центр). Примером ресурсов могут быть хранилище данных, вычислительная мощность, оперативная память, пропускная способность, виртуальные машины.

* Способность к быстрой адаптации

Вычислительные возможности могут быстро и гибко резервироваться (часто автоматически) для оперативного масштабирования под задачи заказчика, и также быстро освобождаться. С точки зрения потребителя доступные возможности часто выглядят ничем не ограниченными и могут быть приобретены в любом количестве в любое время.

* Измеримая услуга

Облачные системы автоматически контролируют и оптимизируют использование ресурсов через измерение некоторых абстрактных параметров. Параметры варьируются в зависимости от типа услуги. Например, это могут быть: размер хранилища данных, вычислительная мощность, пропускная способность и/или число активных пользовательских записей. Использование ресурсов отслеживается, контролируется; формируются отчеты. Таким образом и поставщик, и потребитель получают прозрачную информацию об объеме оказанных (потребленных) услуг.

### 2.2.2Модели обслуживания

* Программное обеспечение как услуга (SaaS, англ. Software-as-a-Service) — модель, в которой потребителю предоставляется возможность использования прикладного программного обеспечения провайдера, работающего в облачной инфраструктуре и доступного из различных клиентских устройств или посредством тонкого клиента, например, из браузера (например, веб-почта) или интерфейс программы. Контроль и управление основной физической и виртуальной инфраструктурой облака, в том числе сети, серверов, операционных систем, хранения, или даже индивидуальных возможностей приложения (за исключением ограниченного набора пользовательских настроек конфигурации приложения) осуществляется облачным провайдером.
* Платформа как услуга (PaaS, англ. Platform-as-a-Service) — модель, когда потребителю предоставляется возможность использования облачной инфраструктуры для размещения базового программного обеспечения для последующего размещения на нём новых или существующих приложений (собственных, разработанных на заказ или приобретённых тиражируемых приложений). В состав таких платформ входят инструментальные средства создания, тестирования и выполнения прикладного программного обеспечения — системы управления базами данных, связующее программное обеспечение, среды исполнения языков программирования — предоставляемые облачным провайдером. Контроль и управление основной физической и виртуальной инфраструктурой облака, в том числе сети, серверов, операционных систем, хранения осуществляется облачным провайдером, за исключением разработанных или установленных приложений, а также, по возможности, параметров конфигурации среды (платформы).
* Инфраструктура как услуга (IaaS, англ. IaaS or Infrastructure-as-a-Service) предоставляется как возможность использования облачной инфраструктуры для самостоятельного управления ресурсами обработки, хранения, сетей и другими фундаментальными вычислительными ресурсами, например, потребитель может устанавливать и запускать произвольное программное обеспечение, которое может включать в себя операционные системы, платформенное и прикладное программное обеспечение. Потребитель может контролировать операционные системы, виртуальные системы хранения данных и установленные приложения, а также ограниченный контроль набора доступных сервисов (например, межсетевой экран, DNS). Контроль и управление основной физической и виртуальной инфраструктурой облака, в том числе сети, серверов, типов используемых операционных систем, систем хранения осуществляется облачным провайдером.

### 2.2.3 Модели развертывания

* Частное облако (англ. private cloud) — инфраструктура, предназначенная для использования одной организацией, включающей несколько потребителей (например, подразделений одной организации), возможно также клиентами и подрядчиками данной организации. Частное облако может находиться в собственности, управлении и эксплуатации как самой организации, так и третьей стороны (или какой-либо их комбинации), и оно может физически существовать как внутри, так и вне юрисдикции владельца.
* Публичное облако (англ. public cloud) — инфраструктура, предназначенная для свободного использования широкой публикой. Публичное облако может находиться в собственности, управлении и эксплуатации коммерческих, научных и правительственных организаций (или какой-либо их комбинации). Публичное облако физически существует в юрисдикции владельца — поставщика услуг.
* Гибридное облако (англ. hybrid cloud) — это комбинация из двух или более различных облачных инфраструктур (частных, публичных или общественных), остающихся уникальными объектами, но связанных между собой стандартизованными или частными технологиями передачи данных и приложений.
* Общественное облако (англ. community cloud) — вид инфраструктуры, предназначенный для использования конкретным сообществом потребителей из организаций, имеющих общие задачи (например, миссии, требований безопасности, политики, и соответствия различным требованиям). Общественное облако может находиться в кооперативной (совместной) собственности, управлении и эксплуатации одной или более из организаций сообщества или третьей стороны (или какой-либо их комбинации), и оно может физически существовать как внутри, так и вне юрисдикции владельца.

### 2.2.4 Экономические аспекты

При использовании облачных вычислений потребители информационных технологий могут существенно снизить капитальные расходы — на построение центров обработки данных, закупку серверного и сетевого оборудования, аппаратных и программных решений по обеспечению непрерывности и работоспособности — так как эти расходы поглощаются провайдером облачных услуг. Кроме того, длительное время построения и ввода в эксплуатацию крупных объектов инфраструктуры информационных технологий и высокая их начальная стоимость ограничивают способность потребителей гибко реагировать на требования рынка, тогда как облачные технологии обеспечивают возможность практически мгновенно реагировать на увеличение спроса на вычислительные мощности.

При использовании облачных вычислений затраты потребителя смещаются в сторону операционных — таким образом классифицируются расходы на оплату услуг облачных провайдеров.

Для объяснения экономической составляющей облачных подходов к вычислениям часто используется аналогия с услугами водо или электроснабжения, предоставляемыми в развитых инфраструктурах по соответствующим коммунальным сетям, легкодоступными и оплачиваемыми по мере потребления, в сравнении с разработкой каждым потребителем собственного водозабора или монтированием собственной электроустановки.

### 2.2.5 Преимущества облаков

Переход к облачным вычислениям дает существенные экономические преимущества:

* Размещение ИТ-инфраструктуры в облаке позволяет достичь экономии в совокупной стоимости владения от 30% до 70%.
* Снижение капитальных затрат до 70% при закупке оборудования.
* Повышение ресурсов использования и обслуживания оборудования до 70% ежемесячно.
* Экономия ресурсов ЦОД (электричество, охлаждение, площади) до 50% ежемесячно.
* Сокращение затрат на резервирование оборудования на 50-70% при аналогичном уровне доступности ежемесячно.
* Сокращение затрат на лицензирование на 30% ежемесячно.
* Сокращение времени развертывания новых сервисов до 90%.

Основные преимущества облачных технологий по сравнению с физическими серверами:

* Доступность – облака доступны всем из любой точки, где есть Интернет.
* Мобильность - сотрудники Компаний становятся более мобильными так, как могут получить доступ к своему рабочему месту из любой точки земного шара, используя ноутбук, нетбук, планшетник или смартфон.
* Круглогодичная безотказно устойчивая работа Компании.
* Повышение безопасности за счет консолидации вычислительных ресурсов, сведение до минимума «человеческого фактора» и подотчетность пользователей к несанкционированному доступу в систему и скачиванию данных.
* Шифрование данных, криптография и защита данных в соответствии со ст. 152-ФЗ.
* Повышение качества предоставляемых ИТ-услуг при меньшем количестве высококвалифицированных специалистов.
* Отсутствие первоначальных капитальных затрат или их существенное сокращение.
* Десятикратное сокращение временных затрат на внедрение и оперативное перераспределение ресурсов.
* Оперативное выборочное наращивание мощности.

# 3. Шифрование

Шифрование — обратимое преобразование информации в целях сокрытия от неавторизованных лиц, с предоставлением, в это же время, авторизованным пользователям доступа к ней. Главным образом, шифрование служит задачей соблюдения конфиденциальности передаваемой информации. Важной особенностью любого алгоритма шифрования является использование ключа, который утверждает выбор конкретного преобразования из совокупности возможных для данного алгоритма. Пользователи являются авторизованными, если они обладают определенным аутентичным ключом. Вся сложность и, собственно, задача шифрования состоит в том, как именно реализован этот процесс. В целом, шифрование состоит из двух составляющих — зашифрование и расшифрование. С помощью шифрования обеспечиваются три состояния безопасности информации:

* Конфиденциальность.

Шифрование используется для скрытия информации от неавторизованных пользователей при передаче или при хранении.

* Целостность.

Шифрование используется для предотвращения изменения информации при передаче или хранении.

* Идентифицируемость.

Шифрование используется для аутентификации источника информации и предотвращения отказа отправителя информации от того факта, что данные были отправлены именно им.

Для того, чтобы прочитать зашифрованную информацию, принимающей стороне необходимы ключ и дешифратор (устройство, реализующее алгоритм расшифровывания). Идея шифрования состоит в том, что злоумышленник, перехватив зашифрованные данные и не имея к ним ключа, не может ни прочитать, ни изменить передаваемую информацию. Кроме того, в современных криптосистемах (с открытым ключом) для шифрования, расшифрования данных могут использоваться разные ключи. Однако, с развитием криптоанализа, появились методики, позволяющие дешифровать закрытый текст без ключа. Они основаны на математическом анализе переданных данных.

## 3.1 Методы шифрования

### 3.1.1 Симметричное шифрование

Симметричное шифрование - способ шифрования, в котором для шифрования и расшифрования применяется один и тот же криптографический ключ.   
До изобретения схемы асимметричного шифрования единственным существовавшим способом являлось симметричное шифрование. Ключ алгоритма должен сохраняться в секрете обеими сторонами. Ключ алгоритма выбирается сторонами до начала обмена сообщениями.

В настоящее время симметричные шифры — это:

* блочные шифры. Обрабатывают информацию блоками определённой длины (обычно 64, 128 бит), применяя к блоку ключ в установленном порядке, как правило, несколькими циклами перемешивания и подстановки, называемыми раундами. Результатом повторения раундов является лавинный эффект — нарастающая потеря соответствия битов между блоками открытых и зашифрованных данных.
* поточные шифры, в которых шифрование проводится над каждым битом либо байтом исходного (открытого) текста с использованием гаммирования.Поточный шифр может быть легко создан на основе блочного (например, ГОСТ 28147-86 в режиме гаммирования), запущенного в специальном режиме.

### 3.1.2 Ассиметричное шифрование

Алгоритм асимметричного шифрования требует использовать один ключ для шифрования данных и другой, но взаимосвязанный с ним ключ — для дешифрования. Один из ключей в такой схеме доступен любому, кто его запрашивает. Такой ключ называется открытым. Другой ключ известен только владельцу и называется личным.

Алгоритмы асимметричного шифрования возникли в связи с необходимостью передавать секретные ключи по незащищенным каналам. Первую систему такого рода разработал Ральф Меркл (Ralph Merkle) в 1974 году . Первым алгоритмом, завоевавшим широкую популярность, был алгоритм ДиффиХеллмана, созданный Уитфилдом Диффи (Whitfield Diffie) и Мартином Хеллманом (Martin Hellman) в 1976 году. В 1977 году Рон Ривест (Ron Rivest), Ади Шамир (Adi Shamir) и Лен Эйдельман (Len Adleman) разработали схожий алгоритм RSA.

Алгоритмы асимметричного шифрования можно применять по прямому назначению (обеспечение конфиденциальности), а также для создания цифровых подписей (аутентификация). Но по своей надежности они не соперники алгоритмам симметричного шифрования. В связи с этим асимметричные алгоритмы чаще всего применяют для шифрования секретных ключей, передаваемых по незащищенным каналам, и для создания цифровых подписей.

## 3.2 MD5

Алгоритм MD5 является алгоритмом вычисления "хэш-функции” (message digest) для различных целей - шифрование паролей, проверка целостности файлов, и т.д. На вход подается поток данных произвольной длины, а на выходе получаем хэш длиной 128 бит. Сила этого алгоритма заключается в том, что практически очень сложно, почти невозможно, найти две строки, дающие одинаковый хэш. Однако, при определенных условиях, возможно получение исходного текста. Но этот метод основан на определенном выборе начальных значений, поэтому не представляет практической ценности. Также MD5 алгоритм используется в приложениях криптографии и электронно-цифровых подписей для генерации ключа шифрования. При разработке данного алгоритма принималось во внимание скорость работы на современных 32-разрядных системах, а также требование минимизации используемой памяти.

# 4. DICOM

DICOM (Digital Imaging and COmmunications in Medicine, цифровые изображения и обмен ими в медицине) - это индустриальный стандарт для передачи радиологических изображений и другой медицинской информации между компьютерами, опирающийся на стандарт Open System Interconnection (OSI), разработанный Международной организацией по стандартам (International Standards Organization, ISO). Стандарт DICOM описывает "паспортные" данные пациента, условия проведения исследования, положение пациента в момент получения изображения и т.п., для того чтобы в последствии было возможно провести медицинскую интерпретацию данного изображения. Первая версия данного стандарта была разработана Американским Колледжем Радиологии (American College of Radiology, ACR) и Национальной ассоциацией производителей электронного оборудования (National Electrical Manufacturers Association, NEMA) в 1985 году.

Стандарт позволяет организовать цифровую связь между различным диагностическим и терапевтическим оборудованием, использующимся в системах различных производителей. Рабочие станции, Компьютерные (КТ) и Магнитно-резонансные Томографы (МРТ), микроскопы, УЗ-сканеры, общие архивы, хост-компьютеры и мэйнфреймы от разных производителей, расположенные в одном городе или нескольких городах, могут "общаться" друг с другом на основе DICOM с использованием открытых сетей по стандартным протоколам, например TCP/IP.

С использованием DICOM'а можно проводить различные медицинские исследования в территориально-распределенных диагностических центрах с возможностью сбора и обработки информации в нужном месте.

Стандарт DICOM версии 3.0 (окончательно выпущена в 1993 году) предназначен для передачи медицинских изображений, получаемых с помощью различных методов лучевой и иной диагностики, в его описании перечислено 29 диагностических методов. Данный стандарт получил широкое распространение в США, Японии, Германии и других странах.

## 4.1 Стандарт DICOM

Стандарт DICOM 3 распространяется на передачу растровых медицинских изображений, получаемых с помощью различных методов лучевой диагностики (рентгенография, ультразвуковая диагностика, эндоскопия, компьютерная и магнитно-резонансная томография и др., всего в нем перечислены 29 методов). Он получил широкое признание не только в США, но и во многих других странах, в том числе европейских. Стандарт DICOM был взят за основу разработки европейского стандарта MEDICOM, работа над которым велась рабочей группой WG4 технического комитета TC 251 Европейского института стандартизации CEN. Стандарт состоит из 13 частей, из которых в текущей версии (DICOM 3.0) представлены первые 9:

* Введение

Описываются история разработки стандарта, его назначение и структура.

* Соответствие стандарту

Указываются структура сертификата соответствия стандарту и критерии, которым должен удовлетворять производитель диагностического оборудования, чтобы иметь право объявить его совместимым со стандартом DICOM.

* Определение информационных объектов

Специфицируются используемые в стандарте информационные объекты. Предлагается информационная модель "реального мира", описывающая взаимоотношения между нормализованными объектами (пациент, устройство) и составными (исследования, изображения и др., наследующими некоторые атрибуты нормализованных объектов).

* Спецификации классов операций

Специфицируются классы действий или операций, которые могут выполняться над информационными объектами. Вводится понятие операция-объект SOP (service-object pair). Разработчики стандарта исходили из того, что применение операции к объекту может быть ограничено его свойствами, поэтому есть нужда в отдельном описании классов SOP. Например, для класса операций Хранение выделяются отдельные стандартные классы операций-объектов Хранение изображений цифровой радиографии, Хранение ультразвуковых изображений, Хранение наложений, Хранение таблицы преобразования пикселов и т.д. Аналогичные пары выделяются для оперяции Запрос/извлечение и пр.

* Структура и семантика данных

Описываются типы данных и правила кодирования, используемые при передаче данных из одной системы в другую. Специфицируются форматы передачи изображений. Стандарт допускает передачу исходных и уплотненных изображений; особо описывается синтаксис передачи при использовании неискажающих и искажающих алгоритмов уплотнения JPEG. Допускаются другие, не специфицируемые в стандарте алгоритмы уплотнения.

* Словарь данных

Приводится полный список элементов данных, описанных в стандарте DICOM. Каждый элемент данных идентифицируется парой целых чисел, например пара (0018,5100) идентифицирует описание положения пациента по отношению к устройству в момент проведения исследования. Кроме идентификатора, приводятся имя элемента, характеристика его значения (строка символов, число и т.д.) и допустимое число повторений элемента в сообщении.

* Обмен сообщениями

Описывается структура команд и протокола обмена сообщениями в стандарте DICOM.

* Обеспечение обмена сообщениями в сетевых средах

Определяются все необходимые компоненты системы обмена сообщениями в стандарте DICOM в сетевых средах, использующих протокол TCP/IP. Изложение этой части существенно опирается на соответствующие стандарты Модели взаимодействия открытых систем OSI (ISO 8222 и ISO 8649).

* Обеспечение обмена сообщениями при прямой связи абонентов (point-to-point)

Приводится подробное описание прямого взаимодействия двух устройств, включая назначение каждой ножки 50-контактного разъема, уровня передаваемых сигналов, их временные характеристики и т.д. Оно напоминает описание параллельного дуплексного интерфейса миникомпьютера, и, похоже, в основном переписано из других стандартов, ранее разработанных ассоциацией NEMA.

На стадии разработки и утверждения находятся еще 4 части стандарта:

* Носители данных и форматы файлов

Описываются теоретические основы хранения медицинских изображений на различных внешних носителях данных.

* Прикладные характеристики хранения данных на внешних носителях

Описываются требования к данным, которые должны храниться на внешних носителях. Описания имеют клиническую направленность, например, задают, какие данные должны храниться на внешних носителях при проведении ангиографии.

* Форматы носителей и физическая среда хранения данных

Специфицируются различные носители данных, которые могут использоваться для хранения медицинских изображений, например, дискеты 3.5", компакт-диски CD ROM, магнитооптические диски и тд.

* Управление выводом на печатающие устройства при прямом соединении

Описываются протоколы и операции, необходимые для вывода изображения на печатающее устройство. Вывод осуществляется системой-исполнителем, имеющей прямое соединение с системой-инициатором вывода.

## 4.2 DICOM сервисы

DICOM состоит из множества различных сервисов, большинство из которых подразумевает обмен данными по сети. Обмен файлами был добавлен в стандарт позже и является только небольшой частью стандарта.

* Store - DICOM сервис ‘Store’ предназначен для передачи изображений или других объектов (например, structured reports – структурированных отчетов) между двумя устройствами DICOM.
* Storage Commitment - Этот сервис используется для подтверждения того, что пересланные по store объекты успешно размещены в хранилище информации. Этим сообщением тот, кто принимает данные - PACS, либо станция сообщает передающему информацию – аппарату, либо станции, что данные успешно сохранены и их можно удалять.
* Query/Retrieve - Сервис поиска и доставки целых исследований или отдельных объектов на удаленном DICOM устройстве. Позволяет найти по определенным фильтрам (например – дата исследования, ФИО пациента и т.д.) интересующее исследование или объект (чаще всего объекты в DICOM – это изображения, но не только) и запросить его пересылку на локальный компьютер.
* Modality Worklist - Позволяет аппарату (часто аппараты в DICOM называют модальностями, однако один и тот же аппарат, например, литотрипсер, может иметь несколько модальностей – US, DX) получить список намеченных исследований. В данных о намеченных исследованиях содержится информация и о пациентах, это позволяет сократить повторный ввод одной и той же информации и сопутствующие ошибки.
* Modality Performed Procedure Step - Дополнительный к Modality Worklist сервис, который позволяет модальности посылать отчет об успешности выполнения исследования, о полученных изображениях, времени начала и конца исследования, полученной пациентом дозе и т.д. Сервис позволяет получить управлению больницы более точные данные по использованию ресурсов аппарата. Сервис, так же называемый MPPS, позволяет улучшить взаимодействие модальности и системы PACS, предоставляя системе список объектов, которые будут посланы перед самой посылкой.
* Printing - Этот сервис позволяет посылать изображения на печать на принтер DICOM, для получения твердой копии изображений, чаще всего на пленках. Существует способ стандартной калибровки (описанный в 14-й части стандарта) принтеров и мониторов, помогающий получить одинаковые изображения на разных мониторах и на твердой копии изображений.

## 4.3 DICOM информационные уровни

Стандартом DICOM определено два информационных уровня, файловый и сетевой.

### 4.3.1 DICOM FIle

DICOM File представляет собой объектно-ориентированный файл с теговой организацией. Информационная модель стандарта DICOM для DICOM файла четырёхступенчатая: пациент (patient) → исследование (study) → серия (series) → изображение (кадр или серия кадров) (image).

Файловый уровень стандарта DICOM 3.0 редакции 2008 года описывает:

* Атрибуты и демографические данные пациента.
* Модель и фирму производителя аппарата, на котором проводилось обследование.
* Атрибуты медицинского учреждения, где было проведено обследование.
* Атрибуты персонала, проводившего обследование пациента.
* Вид обследования и дата/время его проведения.
* Условия и параметры проведения исследования пациента.
* Параметры изображения или серии изображений, записанных в DICOM-файле.
* Уникальные ключи идентификации Unique Identifier (UID) групп данных, описанных в DICOM-файле.
* Изображение, серию или набор серий, полученных при обследовании пациента.
* Представление, в первую очередь, PDF-документов в DICOM-файле.
* Представление DICOM-записи на оптические носители, включая DVD формат.
* DICOM-протокол для передачи/приема по TCP/IP компьютерным сетям.

### 4.3.2 DICOM Network Protocol

DICOM Network Protocol использует TCP/IP для передачи медицинской информации от медицинского оборудования в PACS-систему (Picture Archiving and Communication System) и для связи между PACS-системами. Протокол трёхуровневый — нижний, сразу над TCP — DUL (DICOM Upper Layer); над ним — сервисы: DIMSE (DICOM Message protocol) и ACSE (Association Control protocol — standard OSI protocol); и выше DICOM Application Interface. Над ними расположено приложение — Medical Imaging Application. Стандарт DICOM позволяет производить интеграцию медицинского оборудования разных производителей, включая DICOM-сканеры, DICOM-серверы, автоматизированные рабочие места и DICOM принтеры в единую радиологическую или клиническую информационную систему (англ. Hospital information system).

Стандарт DICOM включает в себя ряд сетевых (основных) сервисов:

* DICOM Store (Storage Service Class) — запоминание (сохранение) изображений и другой информации.
* DICOM Query/Retrieve (Query/Retrieve Service Class) — запрос/получение списка пациентов и/или исследований с другого DICOM-устройства.
* DICOM Media Store (Media Storage Service Class) — сохранение данных на носителях информации для обмена данными.
* DICOM SCP (Service Class Provider) — реализует роль сервера в DICOM-сети.
* DICOM SCU (Service Class User) — реализует роль клиента в DICOM-сети.
* DICOM Modality Worklist (Basic Worklist Management — единственный не нормализованный Service Class) — «Рабочий Лист Исследований» — список требуемых для пациентов исследований, который может быть получен запросом пользователя к RIS-системе.
* DICOM Print (Print Management Service Class) — DICOM-печать, на специализированных DICOM-принтерах (плёночных высокого разрешения или полноцветных), работающих по DICOM-протоколу.

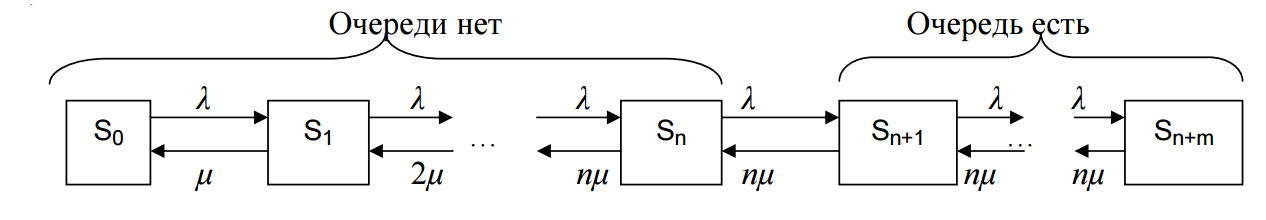
Стандарт DICOM включает в себя основные сетевые команды, каждая из которых осуществляет как запрос (request) — в основном отправляет «клиент» (Service Class User, SCU), так и ответ (response) — в основном отвечает «сервер» (Service Class Provider, SCP):

* Echo — проверяет наличие DICOM-соединения между двумя DICOM-устройствами;
* Find — осуществляет поиск DICOM-элементов и/или DICOM-файлов пациентов на выбранном DICOM-устройстве;
* Get — считывает DICOM-элементы пациентов с выбранного DICOM-устройства;
* Set — устанавливает DICOM-элементы на выбранном DICOM-устройстве;
* Store — сохраняет DICOM-элементы и/или DICOM-файлы на выбранном DICOM-устройстве;
* Move — копирует (переносит) DICOM-элементы и/или DICOM-файлы пациентов с одного DICOM-устройства на другое.

# 5. Описание реализации

Для данной задачи будет разработана модель системы массового обслуживания. Математическая модель внутреннего сервера является «Многоканальная Система массового обслуживания с ограниченной очередью» (Рис.1).

Рис.1 Схема многоканальной СМО с ограниченной очередью



Для данной задачи имеем стационарный, ординарный поток без последействия. С помощью данной математической модели можно заранее оценить требования внутреннего сервера. Для оценки требований необходимо знать интенсивность поступления заявок (λ) и интенсивность обслуживания (µ). Интенсивность поступления заявок можно получить из статистики медицинского персонала, а именно из количества пациентов у каждого врача и оборудования в день. Таким образом можно получить среднюю интенсивность в каждый день, неделю месяц или год. Интенсивность обслуживания вычисляется из серии тестов обращения к серверу. Далее, используя ниже приведенные формулы, можно узнать примерные требования внутреннего сервера.



 - вероятность того, что каналы простаивают

, где k= - вероятность того, что k каналов занято

, где  - вероятность того, что r заявок находятся в очереди

- вероятность того, что заявка будет отклонена

 - относительная пропускная способность

- абсолютная пропускная способность

Но для долгосрочной перспективы необходимо, чтобы внутренний сервер также собирал статистику интенсивностей. Эта статистика поможет определить, если оборудование нуждается в обновлении или увеличении мощности. Для написания математической модели «облачного сервиса», появились трудность прогнозирования интенсивности поступления заявок и интенсивности их обслуживания. На данном этапе, моно точно предположить количество интенсивность обращений от внутренних серверов к внешнему, для сохранения новых данных, а также обращения серверов во время тестирования новых студентов-интернов. Первое значение можно взять из средней интенсивности использования медицинского оборудования учреждений, которые подключены к данному облаку. Второе значение зависит от количества студентов-интернов. Интенсивность поступления заявок через website можно лишь предположить, проведя опрос пациентов. Но результат данного опрос не может быть достоверным. Для вычисления интенсивности обслуживания необходимо провести дополнительные тесты, в которых будет измерять время распаковки файлов стандарта DICOM для website-ов. Модель данного сервера также является многоканальной системой массового обслуживания с ограниченной очередью, как и модель внутреннего сервера.

Данная задача будет разбита на несколько подзадач в связи с объемом работы и исследований.

Первый этап

Изучение open-source библиотеки Grassroot и создание демонстрационного приложения для работы с файлами стандарта DICOM. С помощью данного приложения будут рассмотрены основные методы работы с файлами данного стандарта. Будут проведены серии тестов по сборке и разборке файлов различных размеров, для оценки времени выполнения данной операции.

Второй этап

Реализации демонстрационной версии. Внутренний сервер будет отвечать за обработку клиентских запросов и хранение их данных в локальной Базе Данных. На внутреннем сервере будет реализована система аутентификации. При успешной аутентификации пользователю будет выдан уникальный токен доступа, который аннулируется, если пользователь не производил никаких действий в течении 30 минут. После этого, пользователь должен заново пройти аутентификацию и заново получить токен доступа. Сервер будет отвечать на запросы клиента, если клиент ему пришлет уникальный токен доступа. Внутренний сервер должен обрабатывать как внутренние, запросы, так и внешние (http).

Третий этап

В данном этапе в демонстрационном внутреннем сервере тестовая СУБД будет заменена на СУБД PostGreSQL. База данных будет хранить в себе файлы стандарта DICOM, а также историю болезней клиента.

Четвертый этап

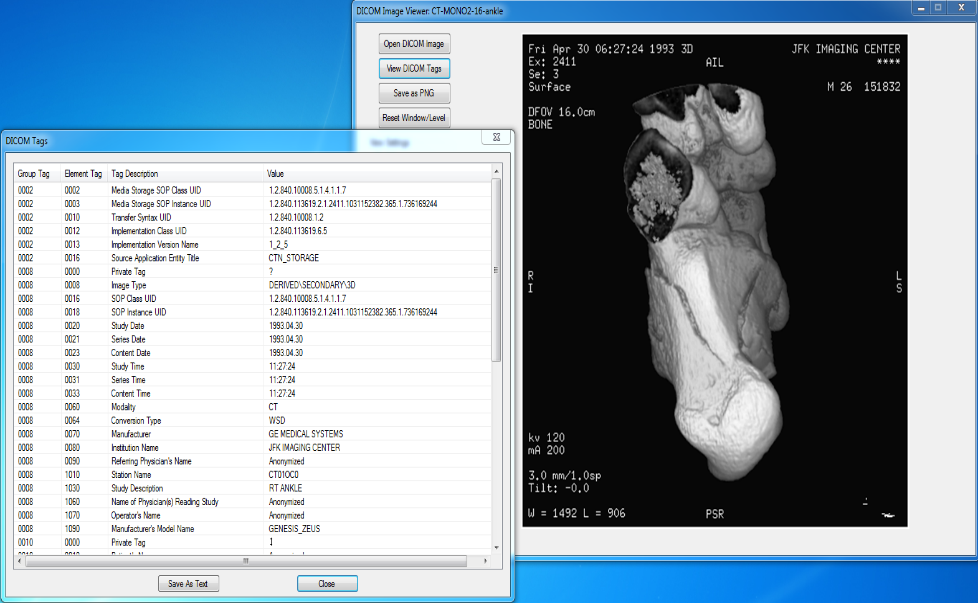
Данный этап крайне сложен в реализации и требует углубления в изучение безопасности, технологии «облачных сервисов», оценки интенсивности поступления запросов. Внешний сервер будет служить общим хранилищем, для хранения информации клиентов. К данному серверу буду обращаться внутренние сервера медицинских учреждений, для получения информации и добавлении информации определенного клиента. В эти сервера необходимо добавить кодирование информации для шифрования запросов и ответов. Данная мера необходимо для сохранности личных данных и токенов. Для работы с внешним сервером, внутренние серверы должны получить уникальный Security Token. Данный токен необходимо будет вставлять в запрос. Если данный токен не будет присутствовать в запросе или он будет не валидным, то в качестве ответа будет послан отказ. Внешний сервер также будет использовать СУБД PostgreSQL.

# 6. Результаты

Прогресс на данный момент:

* + Были реализованы методы открытия файлов стандарта DICOM, вывод изображения на экран, а также данные, которые хранятся в файле вместе с изображениями.

Рис.2 Результат открытия файла



Были рассмотрены различные тэги формат DICOM. Ниже приведены примеры данных тэгов:

{"00100010", "PNPatient's Name"}

{"00100020", "LOPatient ID"}

{"00100021", "LOIssuer of Patient ID"}

{"00100022", "CSType of Patient ID"}

{"00100030", "DAPatient's Birth Date"}

{"00100032", "TMPatient's Birth Time"}

{"00100040", "CSPatient's Sex"}

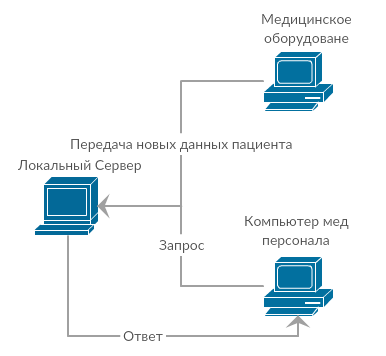
{"00100050", "SQPatient's Insurance Plan Code Sequence"}

{"00100101", "SQPatient's Primary Language Code Sequence"}

{"00100102", "SQPatient's Primary Language Modifier Code Sequence"}

* + Основываясь на математической модели, описанной в реализации, и учитываю долгосрочную перспективу был написан демонстрационный внутренний сервер для работы внутри медицинского учреждения. На (Рис.3) представлена схема работы сервера в медицинском учреждении.

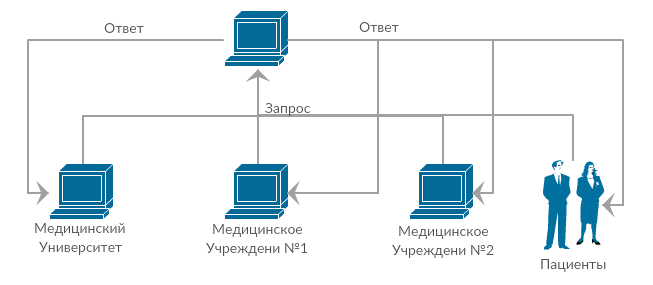
Рис.3 Схема внутреннего сервера



На данном этапе сервер находится в режиме отладки для выявления дефектов. На данном сервере хранятся все данные по пациентам, которые проходили лечение в данном отделении. Сервер содержит простую СУБД с стабовыми данными. Данная СУБД была написана для отладки работы данного сервера, в дальнейшем будет добавлена СУБД, написанная на PostgreSQL. Данные хранятся в том виде, в котором они поступили, так как каждый файл стандарта может иметь одинаковые тэги, но при этом значения их могут отличаться, Например томограмма, сделанная в разные года. Данный сервер обрабатывает как запросы по внутренней сети, так и http запросы из вне. Для внутренней сети добавлена система учетных записей, используя уникальный логин и пароль. Для каждой сессии заводится уникальный аутентификационный токен, который будет анулирован, если вошедшая учетная запись была неактивна более 30 минут. После этого учетная запись должна перезалогинится, для обновления токена. Это добавлено в локальный сервер для предотвращения несанкционированного доступа с компьютера мед персонала, если пользователь не был разлогинен. Данный сервер может отправлять запросы на внешний сервер для запроса или добавления данных клиента.

* + Написан демонстрационная версия внешнего сервер для работы с медицинскими учреждениями и запросами от клиентов. На (Рис.4) представлена работа данного сервера.

Рис.4 Схема внешнего сервера (“Облачное хранилище”)



Сервер содержит простую СУБД с стабовыми данными. Данная СУБД была написана для отладки работы данного сервера, в дальнейшем будет добавлена СУБД, написанная на PostgreSQL. Данные хранятся не в файлах стандарта DICOM, так как разные отделения медицинского учреждения могут запросить различные снимки. В связи с этим все требуемая информация и снимки будут собираться в файл стандарта DICOM после получения запроса. Каждый локальный сервер для подключения к внешнему серверу должен пройти регистрацию для получения уникального Security Token, который будет включен во все запросы локального сервера к внешнему. Все активные Security Token хранятся на внешнем сервере. Если запрос к данному серверу не содержит этого токена или он отсутствует на внешнем сервере, то запрос будет отклонен и в ответе будет указана, что в доступе отказано. Данный сервер служит для передачи данных между авторизованными локальными серверами и доступ клиента к собственным данным через различные устройства, например website. Website предоставляет следующие возможности:

* Каждый пациент медицинского учреждения мог просмотреть свою историю болезни с любого своего устройства.
* Была возможность передачи истории пациента в другое учреждение, только при личном запросе и подтверждении пациента.
* Каждый пациент мог предоставить некоторые данные (томограммные, рентгеные и т.д. снимки) в институты и университеты для обучения студентов. Данная опция позволяет медицинскому учреждению лично отслеживать прогресс студентов и выискивать себе новых кадров. Пациент имеет право не раскрывать свою личность, а предоставить только диагноз и сами снимки.

# 7. Заключение

В данной работе была разработана модель системы массового обслуживания внутреннего сервера, а также сделаны предположения насчет математической модели внешнего сервера. Рассмотрен файл стандарта DICOM, вид хранения данных, тэги. Для работы с данным стандартом были рассмотрены методы библиотеки «Grassroot». Для хранения данных была использована СУБД PostgreSQL. Также в данной работе были написаны 2 демонстрационных сервера: внутренний и внешний, основываясь на разработанной математической модели внутреннего сервера.

В демонстрационном внутреннем сервере была реализована функциональность:

* + Передачи данных от медицинского оборудования к компьютеру медицинского персонала;
  + Хранения истории болезни пациента, который лечился в данном медицинском учреждении;
  + При поступлении новых данных от медицинского оборудования или медицинского персонал, передача их не внешний сервер;

В демонстрационном внешнем сервере была реализована функциональность:

* + Передачи и получения данных между внутренним и внешними серверами;
  + Хранения истории болезни пациента, который лечился в медицинском учреждении, который имеет подключение к внешнему серверу;
  + Обработка запросов, поступивших с website;
  + Парсинг файлов стандарта DICOM для передачи их на website

# Литература

1. Официальный сайт «УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ДЛЯ СТУДЕНТОВ КАФЕДРЫ АСОИУ». Режим доступа: http://www.4stud.info/networking/lecture5.html - клиент-сервер
2. Войтиков К.Ю. Архитектура надстраиваемых приложений клиент-сервер с обобщенным протоколом передачи данных/ К.Ю. Войтиков, О.А. Змеев, А.Н. Моисеев, А.А Якушев// Информатика и программирование. – 2004 – №284 – С.166-170
3. Официальный сайт Томского Политехнического университета. Режим доступа: http://fas.aics.ru/student/lectures/aiit/cli-se.pdf
4. Официальный сайт облачного провайдера. Режим доступа: http://rentacloud.su/whatiscloud
5. Плотников А.В. Стандарт DICOM в компьютерных медицинских технологиях/ А.В. Плотников, Д.А. Прилуцкий, С.В. Селищев// Московский Институт Электронной Техники. Режим доступа: http://www.mks.ru/library/article/1997/dicom.html
6. Официальный сайт ВСА электроник Украина. Режим доступа: http://mri.com.ua/page/text/name=dicom
7. Официальный сайт Wikipedia. Режим доступа: https://ru.wikipedia.org/wiki/DICOM
8. Официальный сайт стандарта DICOM. Режим доступа: http://dicom.nema.org/standard.html - DICOM standard
9. Официальный сайт Architect интернет безопасности. Режим доступа: http://www.open-security.org/linux/tipy\_algoritmov\_simmetrichnoe\_shifrovanie
10. Официальный сайт Московского физико-технического института. Режим доступа: http://re.mipt.ru/infsec/2004/essay/2004\_MD5\_Message-Digest\_Algorithm\_\_Strelnikov.pdf
11. Nitin S. Ujgare Assistant Professor, Maharashtra (INDIA), Swati P. Baviskar Assistant Professor, Maharashtra (INDIA)/ International Journal of Computer Applications (0975 – 8887) Volume 62– No.11, January 2013. Режим доступа: http://research.ijcaonline.org/volume62/number11/pxc3884886.pdf

# Приложение

Преобразование пиксельных данных в массив байтов

void ReadPixels()

{

if (samplesPerPixel == 1 && bitsAllocated == 8)

{

if (pixels8 != null)

pixels8.Clear();

pixels8 = new List<byte>();

int numPixels = width \* height;

byte[] buf = new byte[numPixels];

file.BaseStream.Position = offset;

file.Read(buf, 0, numPixels);

for (int i = 0; i < numPixels; ++i)

{

int pixVal = (int)(buf[i] \* rescaleSlope + rescaleIntercept);

// We internally convert all 8-bit images to the range 0 - 255

//if (photoInterpretation.Equals("MONOCHROME1", StringComparison.OrdinalIgnoreCase))

// pixVal = 65535 - pixVal;

if (photoInterpretation == "MONOCHROME1")

pixVal = max8 - pixVal;

pixels8.Add((byte)(pixelRepresentation == 1 ? pixVal : (pixVal - min8)));

}

}

if (samplesPerPixel == 1 && bitsAllocated == 16)

{

if (pixels16 != null)

pixels16.Clear();

if (pixels16Int != null)

pixels16Int.Clear();

pixels16 = new List<ushort>();

pixels16Int = new List<int>();

int numPixels = width \* height;

byte[] bufByte = new byte[numPixels \* 2];

byte[] signedData = new byte[2];

file.BaseStream.Position = offset;

file.Read(bufByte, 0, numPixels \* 2);

ushort unsignedS;

int i, i1, pixVal;

byte b0, b1;

for (i = 0; i < numPixels; ++i)

{

i1 = i \* 2;

b0 = bufByte[i1];

b1 = bufByte[i1 + 1];

unsignedS = Convert.ToUInt16((b1 << 8) + b0);

if (pixelRepresentation == 0) // Unsigned

{

pixVal = (int)(unsignedS \* rescaleSlope + rescaleIntercept);

if (photoInterpretation == "MONOCHROME1")

pixVal = max16 - pixVal;

}

else // Pixel representation is 1, indicating a 2s complement image

{

signedData[0] = b0;

signedData[1] = b1;

short sVal = System.BitConverter.ToInt16(signedData, 0);

// Need to consider rescale slope and intercepts to compute the final pixel value

pixVal = (int)(sVal \* rescaleSlope + rescaleIntercept);

if (photoInterpretation == "MONOCHROME1")

pixVal = max16 - pixVal;

}

pixels16Int.Add(pixVal);

}

int minPixVal = pixels16Int.Min();

signedImage = false;

if (minPixVal < 0) signedImage = true;

// Use the above pixel data to populate the list pixels16

foreach (int pixel in pixels16Int)

{

// We internally convert all 16-bit images to the range 0 - 65535

if (signedImage)

pixels16.Add((ushort)(pixel - min16));

else

pixels16.Add((ushort)(pixel));

}

pixels16Int.Clear();

}

// 30 July 2010 - to account for Ultrasound images

if (samplesPerPixel == 3 && bitsAllocated == 8)

{

signedImage = false;

if (pixels24 != null)

pixels24.Clear();

pixels24 = new List<byte>();

int numPixels = width \* height;

int numBytes = numPixels \* samplesPerPixel;

byte[] buf = new byte[numBytes];

file.BaseStream.Position = offset;

file.Read(buf, 0, numBytes);

for (int i = 0; i < numBytes; ++i)

{

pixels24.Add(buf[i]);

}

}

}

Представление пользователю картинки

private void ReadAndDisplayDicomFile(string fileName, string fileNameOnly)

{

dd.DicomFileName = fileName;

TypeOfDicomFile typeOfDicomFile = dd.typeofDicomFile;

if (typeOfDicomFile == TypeOfDicomFile.Dicom3File ||

typeOfDicomFile == TypeOfDicomFile.DicomOldTypeFile)

{

imageWidth = dd.width;

imageHeight = dd.height;

bitDepth = dd.bitsAllocated;

winCentre = dd.windowCentre;

winWidth = dd.windowWidth;

samplesPerPixel = dd.samplesPerPixel;

signedImage = dd.signedImage;

label1.Visible = true;

label2.Visible = true;

label3.Visible = true;

label4.Visible = true;

bnSave.Enabled = true;

bnTags.Enabled = true;

bnResetWL.Enabled = true;

label2.Text = imageWidth.ToString() + " X " + imageHeight.ToString();

if (samplesPerPixel == 1)

label4.Text = bitDepth.ToString() + " bit";

else

label4.Text = bitDepth.ToString() + " bit, " + samplesPerPixel +

" samples per pixel";

imagePanelControl.NewImage = true;

Text = "DICOM Image Viewer: " + fileNameOnly;

if (samplesPerPixel == 1 && bitDepth == 8)

{

pixels8.Clear();

pixels16.Clear();

pixels24.Clear();

dd.GetPixels8(ref pixels8);

// This is primarily for debugging purposes,

// to view the pixel values as ascii data.

//if (true)

//{

// System.IO.StreamWriter file = new System.IO.StreamWriter(

// "C:\\imageSigned.txt");

// for (int ik = 0; ik < pixels8.Count; ++ik)

// file.Write(pixels8[ik] + " ");

// file.Close();

//}

minPixelValue = pixels8.Min();

maxPixelValue = pixels8.Max();

// Bug fix dated 24 Aug 2013 - for proper window/level of signed images

// Thanks to Matias Montroull from Argentina for pointing this out.

if (dd.signedImage)

{

winCentre -= char.MinValue;

}

if (Math.Abs(winWidth) < 0.001)

{

winWidth = maxPixelValue - minPixelValue;

}

if ((winCentre == 0) ||

(minPixelValue > winCentre) || (maxPixelValue < winCentre))

{

winCentre = (maxPixelValue + minPixelValue) / 2;

}

imagePanelControl.SetParameters(ref pixels8, imageWidth, imageHeight,

winWidth, winCentre, samplesPerPixel, true, this);

}

if (samplesPerPixel == 1 && bitDepth == 16)

{

pixels16.Clear();

pixels8.Clear();

pixels24.Clear();

dd.GetPixels16(ref pixels16);

minPixelValue = pixels16.Min();

maxPixelValue = pixels16.Max();

if (dd.signedImage)

{

winCentre -= short.MinValue;

}

if (Math.Abs(winWidth) < 0.001)

{

winWidth = maxPixelValue - minPixelValue;

}

if ((winCentre == 0) ||

(minPixelValue > winCentre) || (maxPixelValue < winCentre))

{

winCentre = (maxPixelValue + minPixelValue) / 2;

}

imagePanelControl.Signed16Image = dd.signedImage;

imagePanelControl.SetParameters(ref pixels16, imageWidth, imageHeight,

winWidth, winCentre, true, this);

}

if (samplesPerPixel == 3 && bitDepth == 8)

{

pixels8.Clear();

pixels16.Clear();

pixels24.Clear();

dd.GetPixels24(ref pixels24);

imagePanelControl.SetParameters(ref pixels24, imageWidth, imageHeight,

winWidth, winCentre, samplesPerPixel, true, this);

}

}

else

{

if (typeOfDicomFile == TypeOfDicomFile.DicomUnknownTransferSyntax)

{

MessageBox.Show("Sorry, I can't read a DICOM file with this Transfer Syntax.",

"Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

else

{

MessageBox.Show("Sorry, I can't open this file. " +

"This file does not appear to contain a DICOM image.",

"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

Text = "DICOM Image Viewer: ";

// Show a plain grayscale image instead

pixels8.Clear();

pixels16.Clear();

pixels24.Clear();

samplesPerPixel = 1;

imageWidth = imagePanelControl.Width - 25; // 25 is a magic number

imageHeight = imagePanelControl.Height - 25; // Same magic number

int iNoPix = imageWidth \* imageHeight;

for (int i = 0; i < iNoPix; ++i)

{

pixels8.Add(240);// 240 is the grayvalue corresponding to the Control colour

}

winWidth = 256;

winCentre = 127;

imagePanelControl.SetParameters(ref pixels8, imageWidth, imageHeight,

winWidth, winCentre, samplesPerPixel, true, this);

imagePanelControl.Invalidate();

label1.Visible = false;

label2.Visible = false;

label3.Visible = false;

label4.Visible = false;

bnSave.Enabled = false;

bnTags.Enabled = false;

bnResetWL.Enabled = false;

}

}

Пример кода сервера:

namespace ServerApplication

{

#region MesageControl

class MessageControl : HttpServer

{

private UserControl userControl;

private ConsultationControl consultationControl;

private MessageChecker messageChecker;

//private HttpRequestSender requestSender;

public MessageControl(int port) : base(port)

{

userControl = new UserControl();

consultationControl = new ConsultationControl();

messageChecker = new MessageChecker();

//requestSender = new HttpRequestSender("http://127.0.0.1:8080");

}

public override void handleGETRequest(HttpProcessor p)

{

Console.WriteLine("received GET request");

}

public override void handlePOSTRequest(HttpProcessor p, StreamReader inputData)

{

Console.WriteLine("received POST request");

var serializer = new System.Web.Script.Serialization.JavaScriptSerializer();

Dictionary<string, dynamic> jsonObject = (Dictionary<string, dynamic>) serializer.DeserializeObject(inputData.ReadToEnd());

string method = jsonObject["method"];

Dictionary<string, string>[] messageHandler = messageChecker.checkReceivedMessage(method);

if(messageHandler == null)

{

//send error response!!!!

return;

}

//add authorization check!

string str = messageHandler[0]["Authorized"];

str = str.Replace("[", "");

str = str.Replace("]", "");

str = str.Trim();

string[] authorized = str.Split(',');

if(authorized.Length > 0)

{

Dictionary<string, dynamic> Data = new Dictionary<string, dynamic>();

Data.Add("username", jsonObject["data"]["username"]);

Data.Add("table", "");

for(int i = 0; i < authorized.Length; i++)

{

if (authorized[0].Length == 0)

continue;

authorized[i] = authorized[i].Trim();

Data["table"] = authorized[i];

var response = DataBaseMessageComposer.SendRequest("check\_authorization", Data);

if (response.Count > 0)

break;

else if(i == (authorized.Length - 1))

{

Defines.Error error = new Defines.Error();

Dictionary<string, dynamic> result = new Dictionary<string, dynamic>();

result.Add("Data", "");

result.Add("Error", error);

error.No\_Permission();

this.ComposeResponse(result, p);

return;

}

}

}

switch (messageHandler[1]["Handler"])

{

case "ConsultationControl":

this.ConsultationHandler(p, method, jsonObject["data"], messageHandler[0]["Authorized"]);

break;

case "UserControl":

this.UserHandler(p, method, jsonObject["data"], messageHandler[0]["Authorized"]);

break;

}

}

private void ComposeResponse(Dictionary<string,dynamic> data, HttpProcessor p)

{

string json = (new JavaScriptSerializer()).Serialize(data);

p.writeSuccess("text/html", json);

}

private void ConsultationHandler(HttpProcessor p, string method, Dictionary<string, dynamic> methodData, string Authorized)

{

Dictionary<string, dynamic> result = new Dictionary<string, dynamic>();

Defines.Error error = new Defines.Error();

dynamic handlerResponse;

switch (method)

{

case "get\_subsidiary\_list":

result.Add("Data", this.consultationControl.GetSubsidiaryList(ref error));

break;

case "get\_specializations":

result.Add("Data", this.consultationControl.GetSpecializationList(ref error));

break;

case "get\_doctor\_list":

result.Add("Data", this.consultationControl.GetDoctorList(methodData, ref error));

break;

case "reserve\_time":

handlerResponse = this.consultationControl.ReserveTime(methodData, ref error);

result.Add("Data","");

if(handlerResponse)

result["Data"].Add("Status:", "Success");

else

result["Data"].Add("Status:", "Failed");

break;

case "create\_consultation":

handlerResponse = this.consultationControl.CreateConsultation(methodData, ref error);

result.Add("Data","");

if(handlerResponse)

result["Data"].Add("Status:", "Success");

else

result["Data"].Add("Status:", "Failed");

break;

case "add\_note":

handlerResponse = this.consultationControl.AddNote(methodData, ref error);

result.Add("Data","");

if(handlerResponse)

result["Data"].Add("Status:", "Success");

else

result["Data"].Add("Status:", "Failed");

break;

case "close\_consultation":

handlerResponse = this.consultationControl.CloseConsultation(methodData, ref error);

result.Add("Data","");

if(handlerResponse)

result["Data"].Add("Status:", "Success");

else

result["Data"].Add("Status:", "Failed");

break;

case "cancel\_consultation":

handlerResponse = this.consultationControl.CancelConsultation(methodData, ref error);

result.Add("Data","");

if(handlerResponse)

result["Data"].Add("Status:", "Success");

else

result["Data"].Add("Status:", "Failed");

break;

case "send\_message":

handlerResponse = this.consultationControl.SendMessage(methodData, ref error);

result.Add("Data","");

if(handlerResponse)

result["Data"].Add("Status:", "Success");

else

result["Data"].Add("Status:", "Failed");

break;

case "get\_consultations":

result.Add("Data", this.consultationControl.GetConsultations(methodData, ref error));

break;

case "get\_messages":

result.Add("Data", this.consultationControl.GetMessages(methodData, ref error));

break;

}

result.Add("Error", error);

this.ComposeResponse(result, p);

}

private void UserHandler(HttpProcessor p, string method, Dictionary<string, dynamic> methodData, string Authorized)

{

Defines.Error error = new Defines.Error();

Dictionary<string, dynamic> result = new Dictionary<string, dynamic>();

dynamic handlerResponse;

switch (method)

{

case "get\_pendings":

result.Add("Data", this.userControl.GetPendings(ref error));

break;

case "login":

result.Add("Data", this.userControl.Login(methodData, ref error));

break;

case "register\_user":

handlerResponse = this.userControl.RegisterUser(methodData, ref error);

result.Add("Data","");

if(handlerResponse)

result["Data"].Add("Status:", "Success");

else

result["Data"].Add("Status:", "Failed");

break;

case "approve\_user":

handlerResponse = this.userControl.ApproveUser(methodData, ref error);

result.Add("Data","");

if(handlerResponse)

result["Data"].Add("Status:", "Success");

else

result["Data"].Add("Status:", "Failed");

break;

case "get\_users":

result.Add("Data", this.userControl.GetUsers(ref error));

break;

case "add\_rights":

handlerResponse = this.userControl.AddRights(methodData, ref error);

result.Add("Data","");

if(handlerResponse)

result["Data"].Add("Status:", "Success");

else

result["Data"].Add("Status:", "Failed");

break;

case "get\_schedule":

result.Add("Data", this.userControl.GetSchedule(methodData, ref error));

break;

case "change\_schedule":

handlerResponse = this.userControl.ChangeSchedule(methodData, ref error);

result.Add("Data","");

if(handlerResponse)

result["Data"].Add("Status:", "Success");

else

result["Data"].Add("Status:", "Failed");

break;

case "vacation\_planning":

handlerResponse = this.userControl.VacationPlanning(methodData, ref error);

result.Add("Data","");

if(handlerResponse)

result["Data"].Add("Status:", "Success");

else

result["Data"].Add("Status:", "Failed");

break;

case "approve\_reject\_plan":

handlerResponse = this.userControl.Approve\_RejectPlan(methodData, ref error);

result.Add("Data","");

if(handlerResponse)

result["Data"].Add("Status:", "Success");

else

result["Data"].Add("Status:", "Failed");

break;

}

result.Add("Error", error);

this.ComposeResponse(result, p);

}

}

class MessageChecker

{

Dictionary<string, Dictionary<string, string>[]> messageTypes;

public MessageChecker()

{

DateTime date = DateTime.Now;

Console.WriteLine(date.ToString() + "Defines:: Initializing MessageChecker");

messageTypes = readFile();

}

private Dictionary<string, Dictionary<string, string>[]> readFile()

{

Dictionary<string, Dictionary<string, string>[]> result = new Dictionary<string, Dictionary<string, string>[]>();

List<string> list = new List<string>();

using (StreamReader reader = new StreamReader("MessageTypes.txt"))

{

string line;

while ((line = reader.ReadLine()) != null)

{

list.Add(line); // Add to list.

Console.WriteLine(line); // Write to console.

}

}

string[] stringSpliter = { ";" };

foreach(string el in list)

{

string message = el.Substring(0, el.IndexOf(':'));

string rest = el.Substring(el.IndexOf(':') + 2);

string[] strDescription = rest.Split(stringSpliter, StringSplitOptions.None);

Dictionary<string, string>[] dictionaryDescription = new Dictionary<string, string>[2]{

new Dictionary<string, string>(),

new Dictionary<string, string>()

};

dictionaryDescription[0].Add("Authorized", strDescription[0].Substring(strDescription[0].IndexOf(':') + 2));

dictionaryDescription[1].Add("Handler", strDescription[1].Substring(strDescription[1].IndexOf(':') + 2));

result.Add(message, dictionaryDescription);

}

return result;

}

public Dictionary<string, string>[] checkReceivedMessage(string messageType)

{

Dictionary<string, string>[] result;

if (messageTypes.ContainsKey(messageType))

result = messageTypes[messageType];

else

result = null;

return result;

}

}

#endregion

#region HttpRequestSender

class HttpRequestSender

{

string url;

public HttpClient client;

public HttpRequestSender(string url)

{

Console.WriteLine("Client:: Initializing");

this.url = url;

client = new HttpClient();

}

public string sendPostRequest(Dictionary<string,string> data)

{

Console.WriteLine("Client:: Sending data to url: " + this.url);

var content = new FormUrlEncodedContent(data);

client.PostAsync(this.url, content);

return "";

}

}

#endregion

#region HttpListener

public abstract class HttpServer

{

private int port;

private TcpListener tcpListener;

bool isActive;

public HttpServer(int port)

{

this.port = port;

isActive = true;

}

public void Listen()

{

Console.WriteLine("Server:: Initialized");

tcpListener = new TcpListener(IPAddress.Any, port);

tcpListener.Start();

Console.WriteLine("Server:: Begin Listening");

while (isActive)

{

TcpClient s = tcpListener.AcceptTcpClient();

HttpProcessor processor = new HttpProcessor(s, this);

Thread thread = new Thread(new ThreadStart(processor.process));

thread.Start();

Thread.Sleep(1);

}

}

~HttpServer()

{

if (tcpListener != null)

{

tcpListener.Stop();

isActive = false;

}

}

public abstract void handleGETRequest(HttpProcessor p);

public abstract void handlePOSTRequest(HttpProcessor p, StreamReader inputData);

}

public class HttpProcessor

{

public TcpClient socket;

public HttpServer srv;

private Stream inputStream;

public StreamWriter outputStream;

public String http\_method;

public String http\_url;

public String http\_protocol\_versionstring;

public Hashtable httpHeaders = new Hashtable();

private static int MAX\_POST\_SIZE = 10 \* 1024 \* 1024; // 10MB

private const int BUF\_SIZE = 4096;

public HttpProcessor(TcpClient s, HttpServer srv)

{

this.socket = s;

this.srv = srv;

}

private string streamReadLine(Stream inputStream)

{

int next\_char;

string data = "";

while (true)

{

next\_char = inputStream.ReadByte();

if (next\_char == '\n') { break; }

if (next\_char == '\r') { continue; }

if (next\_char == -1) { Thread.Sleep(1); continue; };

data += Convert.ToChar(next\_char);

}

return data;

}

public void process()

{

inputStream = new BufferedStream(socket.GetStream());

outputStream = new StreamWriter(new BufferedStream(socket.GetStream()));

try

{

parseRequest();

readHeaders();

if (http\_method.Equals("GET"))

{

handleGETRequest();

}

else if (http\_method.Equals("POST"))

{

handlePOSTRequest();

}

}

catch (Exception e)

{

Console.WriteLine("Exception: " + e.ToString());

writeFailure();

}

outputStream.Flush();

inputStream = null; outputStream = null; // bs = null;

socket.Close();

}

public void parseRequest()

{

String request = streamReadLine(inputStream);

string[] tokens = request.Split(' ');

if (tokens.Length != 3)

{

throw new Exception("invalid http request line");

}

http\_method = tokens[0].ToUpper();

http\_url = tokens[1];

http\_protocol\_versionstring = tokens[2];

Console.WriteLine("starting: " + request);

}

public void readHeaders()

{

Console.WriteLine("readHeaders()");

String line;

while ((line = streamReadLine(inputStream)) != null)

{

if (line.Equals(""))

{

Console.WriteLine("got headers");

return;

}

int separator = line.IndexOf(':');

if (separator == -1)

{

throw new Exception("invalid http header line: " + line);

}

String name = line.Substring(0, separator);

int pos = separator + 1;

while ((pos < line.Length) && (line[pos] == ' '))

{

pos++; // strip any spaces

}

string value = line.Substring(pos, line.Length - pos);

Console.WriteLine("header: {0}:{1}", name, value);

httpHeaders[name] = value;

}

}

public void handleGETRequest()

{

srv.handleGETRequest(this);

}

public void handlePOSTRequest()

{

Console.WriteLine("get post data start");

int content\_len = 0;

MemoryStream ms = new MemoryStream();

if (this.httpHeaders.ContainsKey("Content-Length"))

{

content\_len = Convert.ToInt32(this.httpHeaders["Content-Length"]);

if (content\_len > MAX\_POST\_SIZE)

{

throw new Exception(

String.Format("POST Content-Length({0}) too big for this simple server",

content\_len));

}

byte[] buf = new byte[BUF\_SIZE];

int to\_read = content\_len;

while (to\_read > 0)

{

Console.WriteLine("starting Read, to\_read={0}", to\_read);

int numread = this.inputStream.Read(buf, 0, Math.Min(BUF\_SIZE, to\_read));

Console.WriteLine("read finished, numread={0}", numread);

if (numread == 0)

{

if (to\_read == 0)

{

break;

}

else

{

throw new Exception("client disconnected during post");

}

}

to\_read -= numread;

ms.Write(buf, 0, numread);

}

ms.Seek(0, SeekOrigin.Begin);

}

Console.WriteLine("get post data end");

srv.handlePOSTRequest(this, new StreamReader(ms));

}

public void writeSuccess(string content\_type = "text/html", string json = " ")

{

// this is the successful HTTP response line

outputStream.WriteLine("HTTP/1.1 200 OK");

string origin = this.httpHeaders["Origin"].ToString();

outputStream.WriteLine("Access-Control-Allow-Origin:" + origin);

// these are the HTTP headers...

outputStream.WriteLine("Content-Type: " + content\_type);

outputStream.WriteLine("response:" + json);

outputStream.WriteLine("Connection: close");

// ..add your own headers here if you like

outputStream.WriteLine(""); // this terminates the HTTP headers.. everything after this is HTTP body..

outputStream.WriteLine(json);

}

public void writeFailure()

{

// this is an http 404 failure response

outputStream.WriteLine("HTTP/1.0 404 File not found");

// these are the HTTP headers

outputStream.WriteLine("Connection: close");

// ..add your own headers here

outputStream.WriteLine(""); // this terminates the HTTP headers.

}

}

#endregion

public class ServerMain

{

[STAThread]

static void Main(string[] args)

{

HttpServer httpServer;

httpServer = new MessageControl(8080);

Thread thread = new Thread(new ThreadStart(httpServer.Listen));

thread.Start();

//test application

Application.EnableVisualStyles();

Form CommandLog = new ServerCommandLog();

Application.Run(CommandLog);

}

}

}