Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные Системы и Сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему:

«Веб-приложение каталог инструкций»

БГУИР КП 1-40 01 01 05 014  ПЗ

Студент: гр. 551005 Коваленко И.А.

Руководитель: Трус В.В.

Минск 2017

Учреждение образования

«Белорусский государственный университет информатики и

радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

––––––––––––––––––––––––

(подпись)

Лапицкая Н.В. 2017 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Коваленко Илье Андреевичу

1. Тема работы Веб-приложение каталог инструкций

2. Срок сдачи студентом законченной работы 19.12.2017

3. Исходные данные к работе: требования к разрабатываемому программному средству.

4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение.

1. Аналитический обзор литературы и существующих аналогов;

2. Обоснование выбора программных средств;

3. Разработка программного средства;

4. Обоснование технических приемов программирования;

5. Проверка работы программы;

6. Руководство пользователя программы;

7. Заключение, список литературы, ведомость, приложения.

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Схема программы

6. Консультант по курсовому проекту Трус В.В.

7. Дата выдачи задания 15.09.2017 г.

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1, введение к 12.10.2017 – 10 % готовности работы;

разделы 2 к 12.10.2017 – 30 % готовности работы;

разделы 3,4 к 12.11.2017 – 60 % готовности работы;

раздел 5, 6 к 12.12.2017 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 15.12.2017 – 100 % готовности работы.

Защита курсового проекта с 15.12 по 21.12.2017 г.

РУКОВОДИТЕЛЬ В.В. Трус

(подпись)

Задание принял к исполнению –––\_\_\_\_––     15.09.2017 г.

(дата и подпись студента)

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc500902254)

[1 ОСНОВНЫЕ СВЕДЕНИЯ ОБ ИНСТРУКЦИЯ И АНАЛИЗ АНАЛОГОВ ПО 6](#_Toc500902255)

[**1.1** Инструкции и основные сведения о них 6](#_Toc500902256)

[**1.2** Анализ уже созданного ПО 7](#_Toc500902257)

[**1.4** Постановка требований к разрабатываемому продукту 8](#_Toc500902258)

[2 ВЫБОР ПРОГРАММНЫХ СРЕДСТВ 9](#_Toc500902259)

[3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 10](#_Toc500902260)

[**3.1** Создание алгоритма работы программного средства 10](#_Toc500902261)

[**3.2** Выбор шаблонов проектирования 10](#_Toc500902262)

[**3.3** Разработка программного средства 11](#_Toc500902263)

[4 ОБОСНОВАНИЕ ТЕХНИЧЕСКИХ ПРИЕМОВ ПРОГРАММИРОВАНИЯ 23](#_Toc500902264)

[5 ТЕСТИРОВАНИЕ 24](#_Toc500902265)

[6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 28](#_Toc500902266)

[ЗАКЛЮЧЕНИЕ 32](#_Toc500902267)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 33](#_Toc500902268)

[ПРИЛОЖЕНИЕ А – Исходный текст программы 34](#_Toc500902269)

#### ВВЕДЕНИЕ

Веб-приложение — клиент-серверное приложение, в котором клиент взаимодействует с сервером при помощи браузера, а за сервер отвечает — веб-сервер. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются межплатформенными службами.

В настоящее время набирает популярность новый подход к разработке веб-приложений, называемый Ajax. При использовании Ajax страницы веб-приложения не перезагружаются целиком, а лишь догружают необходимые данные с сервера, что делает их более интерактивными и производительными.

Также в последнее время набирает большую популярность технология WebSocket, которая не требует постоянных запросов от клиента к серверу, а создает двунаправленное соединение, при котором сервер может отправлять данные клиенту без запроса от последнего. Таким образом появляется возможность динамически управлять контентом в режиме реального времени.

Для создания веб-приложений на стороне сервера используются разнообразные технологии и любые языки программирования, способные осуществлять вывод в стандартную консоль.

В ходе курсового проектирования планируется создать веб-приложение каталог инструкций, которое позволит пользователям работать с инструкциями. Планируемые функции: создание, просмотр, редактирование, удаление инструкций, возможность комментировать инструкции, устанавливать рейтинг для каждой инструкции, полнотекстовой поиск по всем инструкциям, регистрация и авторизация через социальные сети, просмотр прогресса инструкций для авторизированных пользователей, выбор языка приложения (русский и английский), возможность управления профилем пользователя.

#### 1 ОСНОВНЫЕ СВЕДЕНИЯ ОБ ИНСТРУКЦИЯ И АНАЛИЗ АНАЛОГОВ ПО

* 1. Инструкции и основные сведения о них

Инструкция — документ, содержащий правила, указания или руководства, устанавливающие порядок и способ выполнения или осуществления чего-либо.

С другой стороны инструкция — это правовой акт, утверждаемый или издаваемый в целях установления правил, регулирующих организационные, научно-технические, технологические, финансовые или иные специальные стороны деятельности учреждений, организаций, предприятий (их структурных подразделений и служб), должностных лиц и граждан.

В управленческой практике используются как типовые, так и индивидуальные инструкции. Типовые обычно издают органы власти или управления для системы однотипных организаций или учреждений.

Заголовок должен четко определять те вопросы, объекты или круг лиц, на которые распространяется инструкция. Например, должностная инструкция «Секретаря-референта», Инструкция «По делопроизводству».

Как правило, первый раздел инструкции называется «Общие положения». В нем указывают цели издания, область распространения, порядок и обязательность использования, правовые, нормативные или распорядительные акты, послужившие осно­вой для разработки инструкции.

Текст инструкции состоит из разделов, подразделов, пунктов и подпунктов, которые нумеруются арабскими цифрами. Количество разделов и их логическая взаимосвязь определяются разработчиками. По стилю изложения текст инструкции носит распорядительный характер («устанавливается», «не допускается», «запрещается» и др.) и излагается от третьего лица единственного или множественного числа.

Одним из видов инструкций является инструкция по эксплуатации — описание изделия и правил пользования им. Большинство производителей включают буклеты с инструкцией по эксплуатации в комплект доставки. Такие буклеты содержат описание частей изделия, если необходимо, последовательность его сборки, рекомендации по настройке, пользованию и обслуживанию. Эти описания снабжаются иллюстрациями, схемами и чертежами. Особое внимание в правилах по эксплуатации уделяется правилам безопасности.

Для изделий, экспортируемых в разные страны, инструкции по эксплуатации часто выполняются на нескольких языках. Инструкции по эксплуатации также могут наноситься непосредственно на изделия, например в виде наклеек или надписей краской. Некоторые компании также помещают инструкции по эксплуатации на своих интернет-сайтах.

**1.2** Анализ уже созданного ПО

Из англоязычных ресурсов наиболее популярными и широко используемыми являются веб-сайты wikiHow и Instructables.

Instructables – веб-сервис, специализирующийся на пользовательских «Сделай сам» инструкциях. Пользователи размещают пошаговые инструкции, сопровождаемые различным визуальным контентом, а затем взаимодействуют между собой при помощи комментариев или тематического форума.

wikiHow – вики-проект и интернет-сообщество, состоящее из обширной базы данных практических руководств и инструкций. Веб-сайт предоставляет контент с руководствами в Интернете, позволяя всем желающим редактировать страницы. Большинство практических статей следуют схожему формату с шагами, советами, предупреждениями, списком вещей, которые могут понадобятся, и дополняются изображениями, чтобы помочь читателю узнать, как выполнить задачу.

Особенности работы сервиса Instructables:

Одна из ключевых особенностей сервиса – это создание пошаговых инструкций, которые состоят из текста и интерактивных элементов (изображения и видеоматериалы). Присутствует возможность выставлять оценки инструкциям. Для того, чтобы была возможность пользователься инструкциями без интернета в сервисе существует возможность экспорта инструкций в формат pdf.

Авторизация и регистрация на сервисе возможна как через внутреннюю систему, так и через социальные сети.

Особенности работы сервиса wikiHow:

wikiHow использует вики-метод непрерывного улучшения, позволяя редакторам добавлять, удалять или иным образом изменять контент. Когда статья создаётся, члены сообщества наблюдают за ней, чтобы улучшить её качество. Каждое редактирование проверяется во время процесса под названием «Патрулирование последних изменений», где добровольцы просматривают контент в соответствии с внутренними стандартами, исключая некачественные правки, такие как вандализм и тестовые правки, и сохраняя улучшения. Центральной в приложении является панель инструментов сообщества, которая отображает несколько динамических апплетов, которые связывают пользователей с различными инструментами редактирования, такими как программа проверки орфографии, программа автоматического подбора категории, с помощью которого статьям присваиваются категории в соответствии с их темой, и временное хранилище неопубликованных статей, куда копируются некачественные статьи, где их редактируют и переписывают по стилю и формату.

Большое внимание уделено обработке новых статей и их подготовке перед тем, как их увидит гость ресурса. По умолчанию вновь созданные статьи удаляются из индекса поисковых систем; текст статьи размыт и отображается уведомление о том, что страница невидима для читателей, но зарегистрированные пользователи могут отклонить предупреждение, чтобы просмотреть содержимое статьи.

Система под названием «Продвижение новых статей» позволяет некоторым пользователям с правом «Помощник по новым статьям» просматривать эти статьи и при необходимости доводить их до стандартов сервиса по оформлению статей. Статьи, соответствующие этим стандартам, продвигаются, что устраняет эффект размытия, а также убирает уведомление, и делает статью общедоступной и индексируемой для поиска. Если помощник по новым статьям найдёт недавно написанную высококачественную статью, он может отметить её как «Восходящая звезда». По умолчанию публикации, отмеченные как «Восходящая звезда», продвигаются и помещаются на главную страницу в течение неопределённого периода времени. Противоположно этому, статьи ниже стандартов и которые требуют слишком много работы, чтобы «быть готовы для читателей», «понижены в статусе», который удаляет их из списка продвижения новых статей и сохраняет размытие и уведомление и перемещает страницу в категорию «Статьи для проверки качества». Статьи с контентом, которые противоречат политике удаления сайта, также понижены в статусе (например, статьи, основанные на шутках, противоречащих законодательству темах, а также строго неточные или неполные инструкции).

Авторизация и регистрация, аналогично как и на wikiHow, возможна как через внутреннюю систему, так и через социальные сети.

**1.3** Постановка требований к разрабатываемому продукту

Создать веб-приложение, позволяющее управлять инструкциями.

Планируемые функции:

* ведение журнала обращений для каждого сайта;
* поддержка базовой и дайджест аутентификации;
* два уровня конфигурации: самого сервера, отдельных сайтов;
* отдача статического контента по методу GET;
* загрузка html страниц на сервер методом POST.

#### 2 ВЫБОР ПРОГРАММНЫХ СРЕДСТВ

В качестве фреймворка для создания веб-приложения был сделан выбор в пользу Ruby On Rails. Rails, написанный на языке программирования Ruby, реализует архитектурный шаблон Model-View-Controller для веб-приложений, а также обеспечивает их интеграцию с веб-сервером и сервером баз данных. Является открытым программным обеспечением и распространяется под лицензией MIT. Фреймворк был выбран потому, что он является передовым в области веб-разработки и позволяет быстро и выразительно решать поставленные задачи. Ruby on Rails использует REST-стиль построения веб-приложений.

Rails базируется на следующих принципах разработки приложений:

– максимальное использование механизмов повторного использования, позволяющих минимизировать дублирование кода в приложениях (принцип Don’t repeat yourself);

– по умолчанию используются соглашения по конфигурации, типичные для большинства приложений (принцип Convention over configuration) — явная спецификация конфигурации требуется только в нестандартных случаях.

Для разработки была выбрана операционная система macOS, так как UNIX-подобные операционные системы (которой является macOS) отлично подходят для веб-разработки.

В качестве системы контроля версий был выбран Git, ввиду его высокой популярности и распространенности. В качестве хостинга репозитория Github, из-за наличия большого (по сравнению с аналогичными сервисами) open-source сообщества.

Для обеспечения высокого качества программного кода часто используются статические анализаторы кода, позволяющие выявлять и исправлять наиболее типовые ошибки в ПО. Для Ruby наилучшим выбором будет статический анализатор Rubocop, который из коробки поддерживает все основные рекомендуемые стандарты оформления кода на языке программирования Ruby, а также имеет гибкую систему конфигурации, позволяющую добавлять в исключения некоторые правила, которые по мнению разработчика являются не особо важными для проекта. Одна из часто допустимых ошибок в rails приложениях является ошибка N+1, поэтому для автоматизации поиска и своевременного оповещения о данной ошибке будет использован инструмент gem bullet.

В качестве среды разработки был выбран редактор Atom с установленными дополнениями, так как он имеет глубокую интеграцию с Git, поддержку статического анализатора кода Rubocop, подстветку синтаксиса ruby, rails.

Для тестирования работы приложения был использован браузер Safari, который является стандартным для используемой операционной системы macOS. Браузер выбран ввиду наличия удобных средств для веб-разработки, таких как: консоль разработчика, возможность смены пользовательского агента для тестирования поведения приложения в разных браузерах, встроенные возможности для тестирования адаптивного дизайна (выбор устройства, на котором необходимо провести тестирование, выбор разрешения, ориентации экрана). Для тестирования работы приложения на мобильных устройствах был использован Android смартфон с установленным браузером Mozilla Firefox, из-за его высокой популярности и открытости (на основе него создано целое множество других браузеров).

#### 3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

**3.1** Выбор компонентов приложения

Стандартно в Rails используется RubyGems (от англ. gem, gems— драгоценный камень) — система управления пакетами для языка программирования Руби, который предоставляет стандартный формат для программ и библиотек Руби (в самодостаточном формате «gems»), инструменты, предназначенные для простого управления установкой «gems», и сервер для их распространения.

**3.2** Выбор шаблонов проектирования

Введем понятие виртуальный хост – директива в конфигурационном файле веб-сервера, предназначенная для сопоставления доступных на сервере IP-адреса, домена и директорий на сервере, а также управления доступными на сервере сайтами.

Когда на сервер приходит очередной запрос, ему будет назначен определенный виртуальный хост. Если запрошенного клиентом виртуального хоста нет, запросу будет назначен стандартный.

Для объединения обработчиков запросов и авторизации было решено использовать паттерн цепочка обязанностей. Паттерн цепочка обязанностей позволяет избежать жесткой зависимости отправителя запроса от его получателя, при этом запрос может быть обработан несколькими объектами. Объекты-обработчики связываются в цепочку. Запрос передается по этой цепочке, пока не будет обработан.

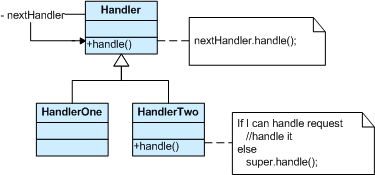


Схема 3.2 – UML диаграмма классов паттерна цепочка обязанностей

В программном средстве будет использована несколько модифицированный паттерн. Первыми в данной цепочке должны стать обработчики аутентификации, которые авторизируют запрос на дальнейшее продвижение по цепочке, содержащей обработчики методов HTTP протокола, либо, в случае неверных данных для входа, отклоняют запрос.

**3.3** Разработка программного средства

Основными структурами данных являются HTTP запрос и HTTP ответ, которые представлены в коде как классы HttpRequest и HttpResponse.

Таблица 3.1 – Основные классы и их краткое описание

|  |  |
| --- | --- |
| Название класса | Краткое описание |
| HttpRequest | Разбор запроса пользователя и его представление для дальнейшей обработки. |
| HttpResponse | Представление HTTP-ответа. |

Продолжение таблицы 3.1

|  |  |
| --- | --- |
| Название класса | Краткое описание |
| HttpResponseStatus | Получение текстовой расшифровки кода ответа HTTP. |
| HttpMimeType | Соотнесение расширения файла с mime-типом. |
| Request | Объединяет объект виртуального хоста и HTTP запроса. Для продвижения по цепочке обработчиков. |
| VirtualHost | Представление виртуальных хостов. |
| AccessLogHandler | Журналирование запросов к сайтам. |
| BasicAuthConfig | Представление информации для аутентификации пользователя по методу базовой HTTP аутентификации. |
| DigestAuthConfig | Представление информации для аутентификации пользователя по методу дайджест аутентификации. |
| VirtualHostList | Работа с виртуальными хостами. |
| VirtualHostConfigSerializer | Вспомогательный класс для сериализации списка виртуальных хостов. Стандартно используется сериализация в Xml, которая может быть заменена на любую другую. |
| VirtualHostConfigHandler | Управляет чтением конфигура-ционного файла с виртуальными хостами из файла и его десериализацией. |
| ServerConfig | Представление конфигурации сервера. |
| RequestHandler | Абстрактный класс. От него наследуются объекты-обработчики цепочки обязанностей. |

Продолжение таблицы 3.1

|  |  |
| --- | --- |
| Название класса | Краткое описание |
| BadRequest, Created, NotFound, NotImplemented, NotModified, NotSatisfiable, Ok, PartialContent, Unauthorized | Каждый класс создает объект HttpResponse с определенными параметрами в соответствии с семантикой объекта. |
| BasicAuthHandler | Проверяет, нужна ли авторизация для запрашиваемого ресурса, и, если необходимо, производит проверку. Либо отдает запрос дальше по цепочке, либо отклоняет его. |
| DigestAuthHandler | Проверка аналогична BasicAuthHandler, за исключением того, что используется другой алгоритм аутентификации. |
| GetHeadMethodHandler | Производит обработку обычного, частичного и условного метода GET/HEAD. В случае, если метод HEAD, тело заголовка не включается в ответ клиенту. |
| PostMethodHandlet | Производит загрузку текстовых html страниц на сервер во временную директорию, указанную в конфигурационном файле виртуального хоста. |
| OptionsMethodHandler | Возвращает в качестве ответа список доступных для взаимодействия методов. |
| FileSystem | Логика работы с файлами. |
| FileDescription | Декоратор над классом стандартной библиотеки FileInfo. |
| Application | Инициализация основных объектов для начала работы приложения. |

Продолжение таблицы 3.1

|  |  |
| --- | --- |
| Название класса | Краткое описание |
| HttpServer | В основном потоке Task прослушивает подключения через Server.AcceptTcpClient() прослушивает входящие соединения и на каждое новое соединение создает новый Task и в методе HandleClient происходит получение данных от клиента, их обработка при помощи объекта ControllerHandler и отправка их по Tcp протоколу. |
| ErrorHandler | Обработка критических ошибок сервера. |

Таблица 3.2 – Описание методов и свойств класса HttpRequest

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| HttpRequest(string) | Формирует из входной строки объект запроса. |
| Method | HTTP метод запроса. |
| Uri | Запрашиваемый ресурс. |
| HttpVersion | Версия HTTP протокола клиента. |
| Host | Запрашиваемый сайт. |
| KeepAlive | Нужно ли оставлять соединение открытым. |
| HttpRange | Для запроса части ресурса. |
| UserAgent | Информация о клиенте. |
| HttpAuthorization | Данные для авторизации. |
| HttpContentType | Тип передаваемого содержимого. |
| Fields | Другие поля заголовка. |
| HttpBody | Тело запроса. |

Таблица 3.3 – Описание свойств и методов класса VirtualHost

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| ServerName | Имя хоста (например, example.com). |
| ServerAlias | Второе имя хоста (например, www.example.com). |
| Directory | Директория, в которой располагаются ресурсы данного виртуального хоста. |
| UploadDirectory | Директория, в которую будут загружаться текстовые документы посредством POST метода. |
| ErrorLog | Лог ошибок виртуального хоста. |
| AccessLog | Лог всех запросов к данному виртуальному хосту. Используется объектом класса AccessLogHandler для логирования действий на ресурсами виртуального хоста (в том числе попытки получить неавторизованный доступ к ресурсу). |
| DefaultIndex | Список стандартных файлов, которые ищутся в случае обращении клиента к директории без явного указания index файла. |
| BasicAuthConfigs | Список конфигураций-объектов класса BasicAuthConfig, которые используются для авторизации пользователя на доступ к определенному ресурсу. |
| DigestAuthConfigs | Список конфигураций-объектов класса DigestAuthConfig, которые используются аналогично объектам класса BasicAuthConfig за исключением метода авторизации. |

Таблица 3.4 – Описание свойств и методов класса AccessLogHandler

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| Path | Путь к файлу с логами. |
| Ip | Адрес, с которого пришел запрос. |
| UserAgent | Информация о пользователе (если удалось определить). |
| GetTime() | Получение текущего точного времени для записи в лог. |
| WriteInfo() | Метод для записи данных в файл логов. |

Таблица 3.5 – Описание свойств и методов класса BasicAuthConfig

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| AuthDirectory | Директория, к которой ограничивается доступ. |
| Realm | Текст, который будет показан клиенту при авторизации. |
| UserName | Имя пользователя. |
| Password | Пароль. |

Таблица 3.6 – Описание свойств и методов класса ServerConfig

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| Ip | Ip адрес сервера. |
| Port | Порт сервера. |
| ServerName | Название сервера и его версия, которая будет отдаваться клиенту вместе с ответом. |
| KeepAliveTimeout | Количество секунд, которое сервер будет ожидать следующего запроса от клиента в открытом соединении. По истечении этого промежутка времени соединение между клиентом и сервером разрывается. |

Продолжение таблицы 3.6

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| DirectoryRoot | Корневая директория, где физически на диске распологается HTTP-сервер. |
| ErrorLog | Журнал для ведения ошибок, связанных с работоспособностью сервера. Обычно записываются критические ошибки, после которых HTTP-сервер не может продолжить работу. |

В ходе разработки алгоритма было решено реализовать шаблон проектирования «цепочка обязанностей» для обработки запросов.

Был определен абстрактный RequestHandler, от которого должен наследоваться каждый обработчик, а также класс ChainControllerHandler, скрывающий детали реализации цепочки.

Таблица 3.7 – Описание свойств и методов класса RequestHandler

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| NextHandler | Указатель на следующий объект-обработчик в цепочке обработчиков. |
| SetHandler() | Метод для организации правильной цепочки. |
| Handle() | Абстрактный метод, реализуемый в наследуемых классах. В нем происходит обработка запроса, возвращается результат IHttpAction. |

Таблица 3.8 – Описание свойств и методов класса ChainControllerHandler

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| RequestHandler | Cодержит указатель на первый элемент цепочки обработчиков запроса. |

Продолжение таблицы 3.8

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| Execute() | Принимает запрос и передает его на обработку первому обработчику, в итоге получает объект типа IHttpAction – результат обработки запроса произвольным количеством объектов. |
| Reg() | Вспомогательная функция для регистрации объектов в цепочке обработчиков. |

Ответы, возвращаемые цепочкой обработчиков реализуют интерфейс IHttpAction, имеющий поле HttpResponse класса HttpResponse.

Классы BadRequest, Created, NotFound, NotImplemented, NotModified, NotSatisfiable, Ok, PartialContent, Unauthorized создают объект HttpResponse с определенными параметрами в соответствии с семантикой объекта.

Таблица 3.9 – Описание свойств и методов класса HttpResponse

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| Headers | Объект класса HttpHeaders, содержащий заголовки и логику их обработки. |
| Content | Объект класса HttpContent, содержащий содержимое ответа и флаг, показывающий, нужно ли включать стандартное тело ответа на ошибочный запрос. |
| HttpVersion | Версия Http протокола, поддерживаемого серверов. |
| HttpStatusCode | Числовой код ответа сервера. |
| IsSuccessStatus | Булевое свойство, характеризующее успешность выполнения запроса. |

Продолжение таблицы 3.9

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| GetBytes() | Метод, преобразующий все поля запроса в соответствии с семантикой Http в массив байт. |
| CombineBytes() | Вспомогательная приватная функция для объединения двух массивов байт. Используется для объединения заголовков с двоичным содержимым тела ответа. |

Таблица 3.10 – Описание свойств и методов класса FileSystem

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| IndexFiles | Список содержит индексные файлы, которые будут искаться в случае, если в качестве файла была передана директория. |
| Path | Путь к файлу/директории. |
| IsDirectory() | Проверка, находится ли по заданному пути директория. |
| IsFile() | Проверка, находится ли по заданному пути файл. |
| GetFile() | Возвращает объект класса FileDescription с описанием полученного файла (может быть как и переданный файл, так и индексный файл из директории). |
| GetFileName() | Получение имени файла на основе Path. |
| CreateFile() | Создает файл с заданным содержимым. В случае, если такой файл существует, осуществляет поиск свободного имени с помощью приватного метода SearchFreeName() |

Продолжение таблицы 3.10

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| SearchFreeName() | Вспомогательный private для поиска свободного имени файла. |

Таблица 3.11 – Описание свойств и методов класса FileDescription

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| GetAllBytes() | Для получения содержимого файла как массива байт. |
| GetFileSize() | Получение размера файла. |
| GetRangeBytes() | Частичное получение содержимого файла на основе HttpRange, заданного в запросе клиента. |
| GetExtension() | Получение расширения файла. |
| GetEncoding() | Получение кодировки файла. |
| GetLastModified() | Получение даты последнего изменения файла. |

Таблица 3.12 – Описание свойств и методов класса HttpServer

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| Server | Объект класса TcpListener для прослушивания входящих запросов. |
| Task | Объект класса Task – основной цикл прослушивания и принятия входящих подключений. |
| ServerConfig | Конфигурационный файл сервера. |
| ControllerHandler | Объект, предназначенный для обработки запроса от клиента. |
| Start() | Запуск Http сервера. |
| Stop() | Остановка Http сервера. |
| AcceptBackground() | Метод принимает входящие подключения от клиентов и запускает для каждого новый Task. |

Продолжение таблицы 3.12

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| HandleClient() | Метод, содержащий цикл обработки запросов от клиента. |
| Receive() | Получение строки запроса от клиента. |
| Send() | Отправка клиенту массива байт, содержащего ответ. |

Таблица 3.13 – Описание свойств и методов класса ControllerHandler

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| ChainControllerHandler | Объект цепочки объектов-обработчиков. |
| VirtualHostList | Список виртуальных хостов. |
| ServerName | Имя сервера, отдаваемое в каждом ответе клиенту. |
| DirectoryRoot | Корневая директория сервера. |
| Execute() | Метод, получающий строку-запрос и возвращающий массив байт в качестве ответа. |
| GetErrorBody() | Вспомогательный метод получения тела ошибочного ответа (если это требуется флагом IncludeBody класса HttpContent) |

Таблица 3.14 – Описание свойств и методов класса Application

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| ErrorHandler | Объект для управления выводом критических ошибок в логи сервера, после которых работоспособность сервера остается под вопросом. |
| ExecutionPath | Директория, в которой был запущен исполняемый файл. |

Продолжение таблицы 3.14

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| HttpServer | Объект класса сервер. |
| ServerConfig | Конфигурационный файл сервера, загружаемый из файла. |
| VirtualHostList | Список всех виртуальных хостов сервера. |
| ServerConfigHandler | Управление загрузкой конфигурации сервера. |
| VirtualHostConfigHandler | Управление загрузкой конфигурации виртуальных хостов. |

Таблица 3.15 – Описание свойств и методов класса ErrorHandler

|  |  |
| --- | --- |
| Название метода или свойства | Описание |
| GetTime() | Метод для получения текущего времени (времени возникновения ошибки). |
| WriteError() | Запись ошибки в журнал без выхода из приложения. |
| WriteCriticalError() | Запись критической ошибки в журнал и выход из приложения. |

#### 4 ОБОСНОВАНИЕ ТЕХНИЧЕСКИХ ПРИЕМОВ ПРОГРАММИРОВАНИЯ

В ходе разработки логику работы с сетью по протоколу Tcp было решено вынести в класс HttpServer согласно принципу единственной ответственности. Для представления HTTP запроса и HTTP ответа были созданы классы HttpRequest и HttpResponse согласно все тому же принципу единственной ответственности.

Для обработки запроса была реализована цепочка обязанностей, которая позволяет обработать запрос один или несколькими объектами.

Изначально планировалось, что объект-обработчик будет сам создавать ответ в виде объекта типа HttpResponse, заполняя в нём необходимые поля. Но это нарушает принцип инверсии зависимостей, так как все объекты-обработчики будут зависеть от внутренней реализации HttpResponse. Было решено создать интерфейс IHttpAction, который реализуют классы Ok, NotFound и другие. Эти классы неявно создают объект типа HttpResponse, заполняя в нем нужные поля, тем самым скрывая детали его реализации от обработчиков.

Когда пользователь запрашивает статический ресурс, это может быть как файл, так и директория. Если запрошена директория, нужно определить по некоторому правилу, какой контент отдавать: индексный файл, список всех файлов или что-то другое. Чтобы скрыть детали этого определения был создан класс FileSystem, который по некоторому алгоритму находит необходимый файл либо контент и возвращает его обработчику запроса.

Для отделения логики сериализации виртуальных хостов и конфигурации сервера были созданы классы VirtualHostConfigSerializer и ServerConfigSerializer согласно принципу единственной ответственности: они отвечают за сериализацию и десериализацию соответствующих объектов. Вынесение логики работы сериализатора в отдельный класс позволяет заменить ее на любую другую.

#### 5 ТЕСТИРОВАНИЕ

Было проведено тестирование как самого приложения, так и данных, которые сервер возвращает.

После запуска приложения загружаются основной конфигурационный файл и конфигурационный файл, содержащий виртуальные хосты. В случае, если произошли какие-либо проблемы при загрузке, если возможно, то ошибка будет записана в файл и работа приложения буде завершена, если нет, то ошибка будет выведена на консоль и работа приложения будет завершена.

Проверка работы программы осуществлялась при помощи браузера Firefox и плагина HttpRequester. Проверка работы программы осуществлялась при помощи браузера Firefox с плагином HttpRequester, и приложения PostMan

Таблица 5.1 – Тестирование работы программного средства

|  |  |  |  |
| --- | --- | --- | --- |
| Запрос | Ответ | Ожидаемый ответ | Пояснение |
| HEAD / HTTP/1.1  Host: s4.localhost  Authorization: Basic YWRtaW46MTIz | HTTP/1.1 200 OK  Content-type: text; charset=windows-1251  Last-Modified: Mon, 16 Mar 2009 21:14:44 GMT  Server: Gepard /1.3  Date: Thu, 25 May 2017 20:34:54 GMT  Accept-Ranges: bytes  Content-Length: 0 | HTTP/1.1 200 OK  Content-type: text; charset=windows-1251  Last-Modified: Mon, 16 Mar 2009 21:14:44 GMT  Server: Gepard /1.3  Date: Thu, 25 May 2017 20:34:54 GMT  Accept-Ranges: bytes  Content-Length: 0 | Метод HEAD не должен включать тело ответа. Тест пройден. |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| Запрос | Ответ | Ожидаемый ответ | Пояснение |
| OPTIONS / HTTP/1.1 | HTTP/1.1 200 OK  Accept-ranges: bytes  Allow: OPTIONS, GET, HEAD, POST  Content-Length: 0  Date: Mon, 05 Jun 2017 22:04:08 GMT  Server: Gepard /1.3 | HTTP/1.1 200 OK  Accept-ranges: bytes  Allow: OPTIONS, GET, HEAD, POST  Content-Length: 0  Date: Mon, 05 Jun 2017 22:04:08 GMT  Server: Gepard /1.3 | Список всех доступных операций сервер отдаёт верно. Тест пройден. |
| GET / HTTP/1.1  Host: s4.localhost | HTTP/1.1 401 Unauthorized  Accept-ranges: bytes  Content-Length: 180  Date: Thu, 25 May 2017 21:05:07 GMT  Server: Gepard /1.3  WWW-Authenticate: Basic realm="Secure Directory Access"  <содержимое ошибки> | HTTP/1.1 401 Unauthorized  Accept-ranges: bytes  Content-Length: 180  Date: Thu, 25 May 2017 21:05:07 GMT  Server: Gepard /1.3  WWW-Authenticate: Basic realm="Secure Directory Access"  <содержимое ошибки> | Сервер отклонил запрос данных без авторизации. Тест пройден. |
| POST / HTTP/1.1  Host: s4.localhost | HTTP/1.1 201 Created  Server: Gepard /1.3  Date: Thu, 25 May 2017 21:20:38 GMT  Accept-Ranges: bytes  Content-Length: 26  Created /tmp/file3.txt<br> | HTTP/1.1 401 Unauthorized  Accept-ranges: bytes  Content-Length: 180  Date: Thu, 25 May 2017 21:20:38 GMT  Server: Gepard /1.3  WWW-Authenticate: Basic realm="Secure Directory Access"  <содержимое ошибки> | Был добавлен файла на сайт, защищенный авторизацией. Это произошло из-за добавления PostMethodHandler’а в конец цепочки. После исправления ошибки (перемещения объекта) тест был пройден. |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| Запрос | Ответ | Ожидаемый ответ | Пояснение |
| POST / HTTP/1.1  Host: s4.localhost | HTTP/1.1 201 Created  Server: Gepard /1.3  Date: Thu, 25 May 2017 21:50:58 GMT  Accept-Ranges: bytes  Content-Length: 26  Created /tmp/file3.txt<br> | HTTP/1.1 201 Created  Server: Gepard /1.3  Date: Thu, 25 May 2017 21:50:58 GMT  Accept-Ranges: bytes  Content-Length: 26  Created /tmp/file3.txt<br> | Тест пройден успешно. |
| GET / HTTP/1.1  Host: s4.localhost  Authorization: Basic YWRtaW46MTIz  Range: bytes=10000-11000 | HTTP/1.1 200 OK  Content-type: text; charset=windows-1251  Last-Modified: Mon, 16 Mar 2009 21:14:44 GMT  Server: Gepard /1.3  Date: Thu, 25 May 2017 20:34:54 GMT  Accept-Ranges: bytes  Content-Length: 5305  <содержимое> | HTTP/1.1 416 Requested Range Not Satisfiable  Server: Gepard /1.3  Date: Thu, 25 May 2017 20:34:54 GMT  Accept-Ranges: bytes  Content-Length: 0  <содержимого> | Вместо отдачи сообщения об ошибки было прислано содержимое всего ресурса. В методе GetRangeBytes() класса FileDescription отдавалось содержимое всего файла в случае, если искомого диапазона не было найдено. Это было исправлено и тест был пройден успешно. |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| Запрос | Ответ | Ожидаемый ответ | Пояснение |
| GET / HTTP/1.1  Host: s4.localhost  Authorization: Basic YWRtaW46MTIz  Range: bytes=10000-11000 | HTTP/1.1 200 OK  Content-type: text; charset=windows-1251  Last-Modified: Mon, 16 Mar 2009 21:14:44 GMT  Server: Gepard /1.3  Date: Thu, 25 May 2017 21:54:52 GMT  Accept-Ranges: bytes  Content-Length: 5305  <содержимое> | HTTP/1.1 200 OK  Content-type: text; charset=windows-1251  Last-Modified: Mon, 16 Mar 2009 21:14:44 GMT  Server: Gepard /1.3  Date: Thu, 25 May 2017 21:54:52 GMT  Accept-Ranges: bytes  Content-Length: 5305  <содержимое> | Тест пройден успешно. |

#### 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Установка:

Для установки приложения в качестве службы Windows нужно указать путь к Gepard.exe в файле Installer/install.bat и запустить последний на выполнение от имени администратора. Ход установки при помощи скрипта показан на рисунке 6.1.

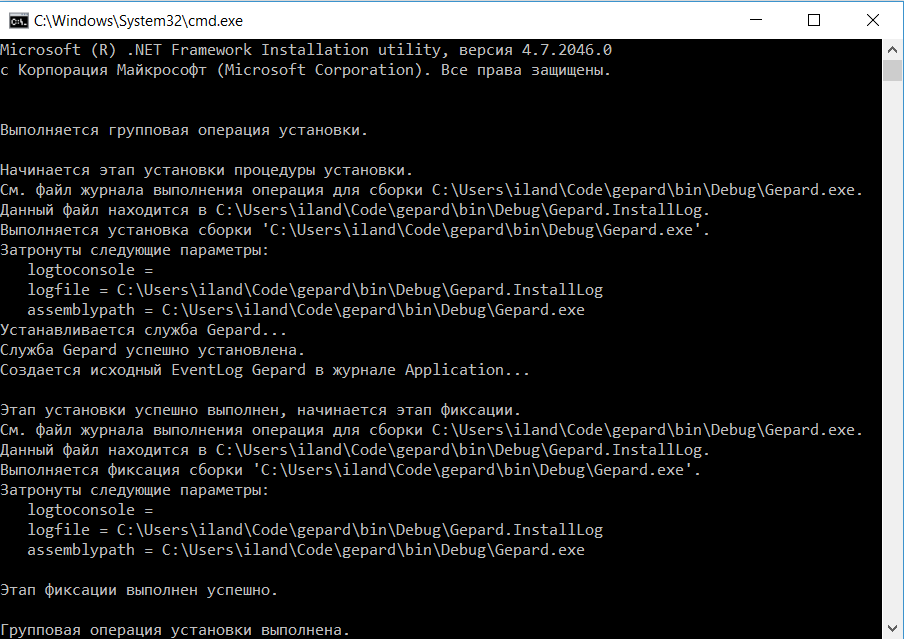


Рисунок 6.1 – Ход установки приложения

В результате в списке служб появится новая служба Gepard как показано на рисунке 6.2.

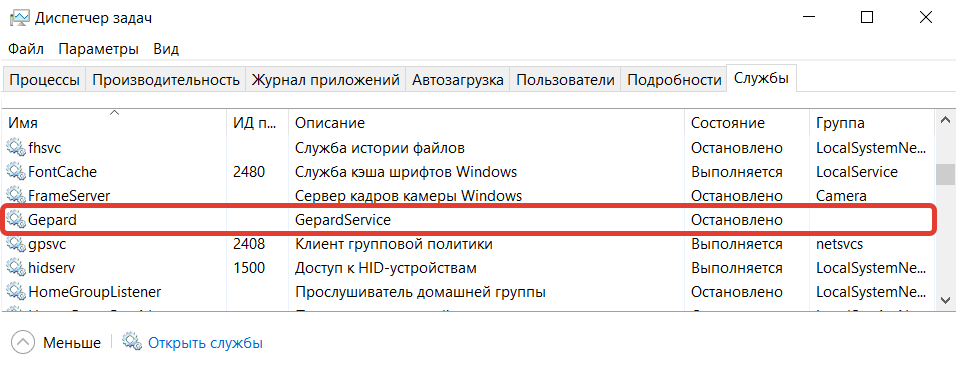


Рисунок 6.2 – Установленный сервер в списке Windows служб

Запуск сервера:

Для запуска сервера необходимо выбрать его в списке Windows служб, нажаться правой клавишей мыши в выбрать «Запустить».

Остановка сервера:

Для остановки сервера необходимо выбрать его в списке Windows служб, нажаться правой клавишей мыши в выбрать «Остановить».

Перезагрузка сервера:

Для перезагрузки сервера необходимо выбрать его в списке Windows служб, нажаться правой клавишей мыши в выбрать «Перезапустить».

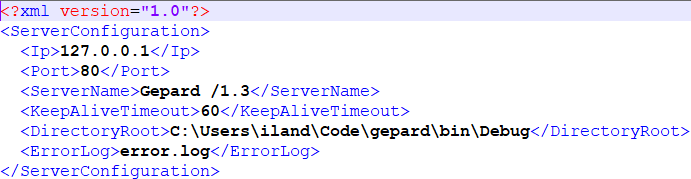


Рисунок 6.3 – Пример конфигурационного файла сервера

Конфигурирование сервера:

Основные параметры конфигурации сервера находятся в файле logs/server.xml. Он содержит следующие поля:

* Ip – адрес сервера, с которого сервер будет прослушивать входящие подключения (задаётся ip-адрес текущего устройства).
* Port – порт, который сервер будет прослушивать (по-умолчанию для HTTP задан 80).
* ServerName – имя сервера, которое сервер будет отправлять клиенту (не рекомендуется менять)
* KeepAliveTimeout – сколько (в секундах) будет ждать сервер следующего запроса от клиента в случае Keep-Alive соединения.
* DirectoryRoot – корневая директория сервера, относительно которой считаются другие системные пути.
* ErrorLog – лог, куда, в основном, будут поступать сообщения о критических ошибках.

Чтобы изменения конфигурации вступили в силу необходимо перезагрузить сервер.

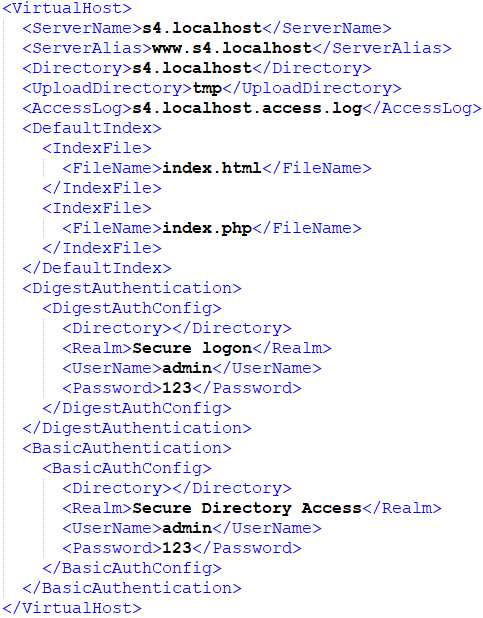


Рисунок 6.4 – Пример конфигурации сайта

Добавление/изменение существующего сайта:

Изменение конфигурации виртуальных хостов производится в файле config/vhosts.xml, где можно отредактировать любой виртуальный хост, либо добавить новый. Для нового сайта нужно создать директорию в папке www, куда и загрузить все содержимое сайта.

Существует также и стандартный виртуальный хост DefaultVirtualHost, на который будут переадресованы все обращения в случае, если запрашиваемый виртуальный хост не найден.

Список полей виртуального хоста следующий:

* ServerName – домен сайта.
* ServerAlias – зеркало основного домена сайта.
* Directory – директория, в которой располагается содержимое сайта.
* UploadDirectory – временная папка, куда попадают все загруженные текстовые и html файлы.
* AccessLog – лог всех попыток доступа к ресурсам сайта.
* DefaultIndex – массив индексных файлов IndexFile, которые будут искаться в случае, если пользователь обратится не к конкретному файлу, а к директории.
* DigestAuthentication – массив записей DigestAuthConfig, которые позволяют защитить отдельные ресурсы паролем.
* BasicAuthentication – массив записей BasicAuthConfig, которые позволяют защитить отдельные ресурсы паролем.

Описание структуры IndexFile:

* FileName – название индексного файла

Описание структуры BasicAuthConfig/DigestAuthConfig:

* Directory – директория, на которую распространяется данный блок авторизации
* Realm – сообщение, которое будет выводиться неаутентифицированному клиенту.
* UserName – имя пользователя для входа.
* Password – пароль.

Удаление:

Для удаления приложения нужно указать путь к Gepard.exe в файле Installer/uninstall.bat и запустить последний на выполнение от имени администратора. После завершения работы скрипта служба будет удалена.

#### ЗАКЛЮЧЕНИЕ

В входе курсового проектирования был создан HTTP-Сервер, позволяющий размещать статические сайты в сети интернет, либо в локальной сети.

Разработанное приложение имеет следующие функции/возможности:

* Создание сайтов.
* Поддержка гибкой системы конфигурации как самого сервера, так и отдельных виртуальных хостов (сайтов).
* Журналирование доступа к сайтам.
* Авторизация как для конкретного ресурса, так и для всего сайта.
* Поддержка метода GET для получения содержимого ресурса.
* Поддержка частичного метода GET для получения части ресурса.
* Поддержка условного метода GET для организации кеширования на стороне клиента.
* Поддержка метода HEAD для получения заголовка ответа сервера.
* Поддержка метода POST для загрузки текстовых файлов на сайт.
* Поддержка keep-alive соединений для более быстрой загрузки ресурсов в одном TCP соединении.

В настоящее время подавляющее большинство сайтов имеют динамический контент, поэтому следующим логичным шагом при дальнейшей разработке сервера станет добавление поддержки динамических страниц. Для связи внешних программ с веб-сервером можно использовать стандарт интерфейса CGI (общий интерфейс шлюза). Сам интерфейс разработан таким образом, чтобы можно было использовать любой язык программирования, который может работать со стандартными устройствами ввода-вывода. Такими возможностями обладают даже скрипты для встроенных командных интерпретаторов операционных систем, поэтому в простых случаях могут использоваться даже командные скрипты[5].

#### СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Веб-приложение - https://ru.wikipedia.org/wiki/Веб-приложение

[2] Инструкция - https://ru.wikipedia.org/wiki/Инструкция

[3] Инструкции - http://www.bibliotekar.ru/deloproizvodstvo-1/85.htm

[4] wikiHow - https://ru.wikipedia.org/wiki/WikiHow

[5] Веб-сервер - https://ru.wikipedia.org/wiki/Веб-сервер

[6] CGI - https://ru.wikipedia.org/wiki/CGI

#### ПРИЛОЖЕНИЕ А – Исходный текст программы

**Текст класса Application**

public class Application

{

private ErrorHandler ErrorHandler { get; set; }

public string ExecutionPath { get; set; }

public HttpServer HttpServer { get; set; }

public ServerConfig ServerConfig { get; set; }

public VirtualHostList VirtualHostList { get; set; }

public ServerConfigHandler ServerConfigHandler { get; set; }

public VirtualHostConfigHandler VirtualHostConfigHandler { get; set; }

public Application(string configDirectory, ServerConfigSerializer serverConfigSerializer, VirtualHostConfigSerializer virtualHostConfigSerializer)

{

ExecutionPath = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);

if (ExecutionPath == null) throw new ApplicationException("Can' get execution path.");

try

{

ServerConfigHandler = new ServerConfigHandler(serverConfigSerializer);

ServerConfig = ServerConfigHandler.LoadFromFile(Path.Combine(ExecutionPath, configDirectory, "server.xml"));

}

catch

{

Console.WriteLine("Error parsing server.xml configuration file.");

return;

}

ErrorHandler = new ErrorHandler(Path.Combine(ExecutionPath, "logs", ServerConfig.ErrorLog));

try

{

VirtualHostConfigHandler = new VirtualHostConfigHandler(virtualHostConfigSerializer);

VirtualHostList = VirtualHostConfigHandler.LoadFromFile(Path.Combine(ExecutionPath, configDirectory, "vhosts.xml"));

}

catch

{

ErrorHandler.WriteCriticalError("Error parsing vhosts.xml configuration file.");

}

var chainControllerHandler = new ChainControllerHandler();

chainControllerHandler.Reg(new DigestAuthHandler());

chainControllerHandler.Reg(new BasicAuthHandler());

chainControllerHandler.Reg(new OptionsMethodHandler());

chainControllerHandler.Reg(new PostMethodHandler(ServerConfig.DirectoryRoot));

chainControllerHandler.Reg(new GetHeadMethodHandler(ServerConfig.DirectoryRoot));

var controllerHandler = new ControllerHandler(VirtualHostList, chainControllerHandler, ServerConfig.ServerName, ServerConfig.DirectoryRoot);

HttpServer = new HttpServer(ServerConfig, controllerHandler);

}

public void Start()

{

HttpServer.Start();

}

public void Stop()

{

try

{

HttpServer.Stop();

}

catch

{

// ignored

}

}

}

**Текст класса HttpServer**

public class HttpServer

{

public static string HttpServerName { get; set; }

private TcpListener Server { get; }

private Task Task { get; }

private ServerConfig ServerConfig { get; }

private ControllerHandler ControllerHandler { get; }

public HttpServer(ServerConfig serverConfig, ControllerHandler controllerHandler)

{

HttpServerName = serverConfig.ServerName;

ServerConfig = serverConfig;

ControllerHandler = controllerHandler;

Server = new TcpListener(IPAddress.Parse(ServerConfig.Ip), ServerConfig.Port);

Task = new Task(AcceptBackground);

}

public void Start()

{

Server.Start();

Task.Start();

Console.WriteLine("Server started at " + Server.LocalEndpoint);

}

public void Stop()

{

Task.Dispose();

}

private void AcceptBackground()

{

while (true)

{

var client = Server.AcceptTcpClient();

Task.Factory.StartNew(() => HandleClient(client));

}

// ReSharper disable once FunctionNeverReturns

}

private void HandleClient(TcpClient client)

{

var stream = client.GetStream();

client.ReceiveTimeout = ServerConfig.KeepAliveTimeout \* 1000;

while (true)

{

try

{

var strRequest = Receive(stream);

var response = ControllerHandler.Execute(strRequest, client.Client.LocalEndPoint.ToString());

Send(stream, response.ArrayBytes);

if (response.ConnectionAlive) continue;

client.Close();

break;

}

catch

{

client.Close();

break;

}

}

}

private string Receive(NetworkStream stream)

{

var buffer = new byte[256];

var data = "";

while (stream.DataAvailable || data.Length == 0)

{

var bytesRead = stream.Read(buffer, 0, buffer.Length);

data += Encoding.UTF8.GetString(buffer, 0, bytesRead);

}

return data;

}

private void Send(NetworkStream stream, byte[] msg)

{

stream.Write(msg, 0, msg.Length);

}

}

**Тексты классов пространства имён Core.Main**

public class ControllerHandler

{

private ChainControllerHandler ChainControllerHandler { get; set; }

private VirtualHostList VirtualHostList { get; set; }

private string ServerName { get; set; }

private string DirectoryRoot { get; set; }

public ControllerHandler(VirtualHostList virtualHostList, ChainControllerHandler chainControllerHandler, string serverName, string directoryRoot)

{

VirtualHostList = virtualHostList;

ChainControllerHandler = chainControllerHandler;

ServerName = serverName;

DirectoryRoot = directoryRoot;

}

public ByteResponse Execute(string str, string clientIp)

{

HttpRequest requestObject = null;

HttpResponse httpResponse = null;

try

{

requestObject = new HttpRequest(str);

}

catch

{

httpResponse = new NotFound().HttpResponse;

}

var virtualHost = VirtualHostList.GetVirtualHost(requestObject != null ? requestObject.Host : "");

var request = new Request(requestObject, virtualHost);

var accessLogHandler = new AccessLogHandler(Path.Combine(DirectoryRoot, "logs", virtualHost.AccessLog), clientIp, requestObject != null ? requestObject.UserAgent : "");

if (httpResponse == null)

{

httpResponse = ChainControllerHandler.Execute(request).HttpResponse;

}

accessLogHandler.WriteInfo(request.Object.Method + " /" + request.Object.Uri.Url + " " + HttpResponseStatus.Get(httpResponse.HttpStatusCode));

httpResponse.Headers.Add("Server", HttpServer.HttpServerName);

httpResponse.Headers.Add("Date", new HttpDate(DateTime.Now).ToString());

if (httpResponse.Content.IncludeBody && httpResponse.Content.Data == null)

{

httpResponse.Content.Data = GetErrorBody(httpResponse.HttpStatusCode);

}

return new ByteResponse(httpResponse.GetBytes(), request.Object.KeepAlive);

}

private byte[] GetErrorBody(int errorCode)

{

var data = File.ReadAllText(Path.Combine(DirectoryRoot, "pages", "error.html"));

data = data.Replace("{CODE}", errorCode.ToString());

data = data.Replace("{CODE-DESCRIPTION}", HttpResponseStatus.Get(errorCode));

data = data.Replace("{SERVER}", ServerName + " / " + Environment.OSVersion);

return Encoding.UTF8.GetBytes(data);

}

}

public class ChainControllerHandler

{

private RequestHandler RequestHandler { get; set; }

public IHttpAction Execute(Requests.Request request)

{

return RequestHandler.Handle(request);

}

public void Reg(RequestHandler requestHandler)

{

if (RequestHandler != null)

{

RequestHandler.SetHandler(requestHandler);

}

else

{

RequestHandler = requestHandler;

}

}

}

public class ErrorHandler

{

public string Path { get; set; }

public ErrorHandler(string path)

{

Path = path;

}

private static string GetTime()

{

return DateTime.Now.ToString("[dd/MM/yyyy | h:mm:ss] ");

}

public void WriteError(string message)

{

File.AppendAllLines(Path, new[] { GetTime() + message });

}

public void WriteCriticalError(string message)

{

WriteError(message);

throw new Exception(message);

}

}

public interface IHttpAction

{

HttpResponse HttpResponse { get; set; }

}

public interface ILogHandler

{

string Path { get; set; }

string Ip { get; set; }

string UserAgent { get; set; }

string GetTime();

void WriteInfo(string message);

}

public abstract class RequestHandler

{

public RequestHandler NextHandler { get; set; }

public abstract IHttpAction Handle(Requests.Request request);

public void SetHandler(RequestHandler requestHandler)

{

if (NextHandler != null)

{

NextHandler.SetHandler(requestHandler);

}

else

{

NextHandler = requestHandler;

}

}

}

public class AccessLogHandler : ILogHandler

{

public string Path { get; set; }

public string Ip { get; set; }

public string UserAgent { get; set; }

public AccessLogHandler(string path, string ip, string userAgent)

{

Path = path;

Ip = ip;

UserAgent = userAgent;

}

public string GetTime()

{

return DateTime.Now.ToString("dd/MM/yyyy hh:mm:ss");

}

public void WriteInfo(string message)

{

try

{

File.AppendAllLines(Path, new[] { $"[{GetTime()}] [{Ip}] [{UserAgent}] {message}" });

}

catch

{

// ignored

}

}

}

**Тексты классов-обработчиков запроса RequestHandler**

public class BasicAuthHandler : RequestHandler

{

public override IHttpAction Handle(Request request)

{

if (request.VirtualHost.BasicAuthConfigs != null)

{

foreach (var authConfig in request.VirtualHost.BasicAuthConfigs)

{

if (request.Object.Uri.Url.StartsWith(authConfig.AuthDirectory))

{

// Need auth

var authValue = Convert.ToBase64String(Encoding.UTF8.GetBytes(authConfig.UserName + ":" + authConfig.Password));

if (request.Object.Authorization.AuthType == "Basic"

&& request.Object.Authorization["Value"] == authValue)

{

return NextHandler != null ? NextHandler.Handle(request) : new NotImplemented();

}

else

{

var httpHeaders = new HttpHeaders();

httpHeaders.Add("WWW-Authenticate", "Basic realm=\"" + authConfig.Realm + "\"");

return new Unauthorized(httpHeaders, null);

}

}

}

}

return NextHandler != null ? NextHandler.Handle(request) : new NotImplemented();

}

public class DigestAuthHandler : RequestHandler

{

private Random Random { get; }

private string Nonce { get; }

public DigestAuthHandler()

{

Random = new Random(DateTime.Now.Millisecond);

Nonce = RandomString(DateTime.Now.Second);

}

public override IHttpAction Handle(Request request)

{

if (request.VirtualHost.DigestAuthConfigs != null)

{

foreach (var authConfig in request.VirtualHost.DigestAuthConfigs)

{

if (request.Object.Uri.Url.StartsWith(authConfig.AuthDirectory))

{

// Need auth

var ha1 = Md5($"{authConfig.UserName}:{authConfig.Realm}:{authConfig.Password}");

var ha2 = Md5($"{request.Object.Method}:/{request.Object.Uri.Url}");

var response = Md5($"{ha1}:{Nonce}:{ha2}");

if (request.Object.Authorization.AuthType == "Digest"

&& request.Object.Authorization["UserName"] == authConfig.UserName

&& request.Object.Authorization["Nonce"] == Nonce

&& request.Object.Authorization["Realm"] == authConfig.Realm

&& request.Object.Authorization["Response"] == response

&& request.Object.Authorization["Uri"] == "/" + request.Object.Uri.Url)

{

return NextHandler != null ? NextHandler.Handle(request) : new NotImplemented();

}

else

{

var httpHeaders = new HttpHeaders();

httpHeaders.Add("WWW-Authenticate", $"Digest realm=\"{authConfig.Realm}\", nonce=\"{Nonce}\"");

return new Unauthorized(httpHeaders, null);

}

}

}

}

return NextHandler != null ? NextHandler.Handle(request) : new NotImplemented();

}

private static string Md5(string input)

{

var md5 = MD5.Create();

var inputBytes = Encoding.ASCII.GetBytes(input);

var hash = md5.ComputeHash(inputBytes);

var sb = new StringBuilder();

foreach (var byteHash in hash)

{

sb.Append(byteHash.ToString("X2"));

}

return sb.ToString().ToLower();

}

private string RandomString(int length)

{

const string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

return new string(Enumerable.Repeat(chars, length).Select(s => s[Random.Next(s.Length)]).ToArray()).ToLower();

}

}

public class GetHeadMethodHandler : RequestHandler

{

private string DirectoryRoot { get; set; }

public GetHeadMethodHandler(string directoryRoot)

{

DirectoryRoot = directoryRoot;

}

public override IHttpAction Handle(Request request)

{

if (request.Object.Method == "GET" || request.Object.Method == "HEAD")

{

var includeBody = request.Object.Method != "HEAD";

var httpHeaders = new HttpHeaders();

var fileSystem = new FileSystem(Path.Combine(DirectoryRoot, "www", request.VirtualHost.Directory, request.Object.Uri.Url), request.VirtualHost.DefaultIndex);

var fileDescription = fileSystem.GetFile();

if (fileDescription == null) return new NotFound(includeBody);

var dateChange = new HttpDate(fileDescription.GetLastModified());

httpHeaders.Add("Content-Type", HttpMimeType.GetByExtension(fileDescription.GetExtension()) + "; charset=" + fileDescription.GetEncoding());

httpHeaders.Add("Last-Modified", dateChange.ToString());

if (request.Object.HttpRange != null)

{

try

{

request.Object.HttpRange.Normalize(fileDescription.GetFileSize() - 1);

var bytesArray = fileDescription.GetRangeBytes(request.Object.HttpRange);

if (bytesArray.Length > 0)

{

httpHeaders.Add("Content-Range", request.Object.HttpRange.ToString());

return new PartialContent(httpHeaders, bytesArray, includeBody);

}

else

{

return new NotSatisfiable(includeBody);

}

}

catch (Exception)

{

request.Object.HttpRange = null;

}

}

if (request.Object["If-Modified-Since"] != null)

{

try

{

var requestDate = DateTime.Parse(request.Object["If-Modified-Since"].Trim());

if (requestDate >= dateChange.DateTime)

{

return new NotModified(httpHeaders, false);

}

}

catch

{

// ignored

}

}

return new Ok(httpHeaders, fileDescription.GetAllBytes(), includeBody);

}

return NextHandler != null ? NextHandler.Handle(request) : new NotImplemented();

}

}

public class OptionsMethodHandler : RequestHandler

{

public override IHttpAction Handle(Request request)

{

if (request.Object.Method == "OPTIONS")

{

var httpHeaders = new HttpHeaders();

httpHeaders.Add("Allow", "OPTIONS, GET, HEAD, POST");

return new Ok(httpHeaders, null, false);

}

return NextHandler != null ? NextHandler.Handle(request) : new NotImplemented();

}

}

public class PostMethodHandler : RequestHandler

{

public string DirectoryRoot { get; set; }

public PostMethodHandler(string directoryRoot)

{

DirectoryRoot = directoryRoot;

}

public override IHttpAction Handle(Request request)

{

if (request.Object.Method == "POST" && request.Object.HttpBody.HttpBodyObjects.Count > 0)

{

if (request.Object.ContentType.Value.Contains("x-www-form-urlencoded"))

{

var httpHeaders = new HttpHeaders();

try

{

var httpUrlEncoded = new HttpUrlEncoded(request.Object.HttpBody.HttpBodyObjects[0].Content);

var responseString = "";

foreach (var keyValue in httpUrlEncoded.Dictionary)

{

responseString += $"[\"{keyValue.Key}\" => \"{keyValue.Value}\"]<br>";

}

httpHeaders.Add("Content-Type", "text/html");

return new Ok(httpHeaders, Encoding.UTF8.GetBytes(responseString));

}

catch

{

return new BadRequest();

}

}

if (request.Object.ContentType.Value.Contains("form-data"))

{

var isCreated = false;

var content = "";

var httpHeaders = new HttpHeaders();

foreach (var bodyObject in request.Object.HttpBody.HttpBodyObjects)

{

if (bodyObject.ContentDisposition.FileName != null

&& (bodyObject.ContentType.Value.Contains("text") || bodyObject.ContentType.Value == string.Empty) )

{

isCreated = true;

var fileSystem = new FileSystem(Path.Combine(DirectoryRoot, "www", request.VirtualHost.Directory, request.VirtualHost.UploadDirectory, bodyObject.ContentDisposition.FileName));

fileSystem.CreateFile(Encoding.UTF8.GetBytes(bodyObject.Content));

content += "Created /" + request.VirtualHost.UploadDirectory + "/" + fileSystem.GetFileName() + "<br>";

}

}

if (isCreated)

{

return new Created(httpHeaders, Encoding.UTF8.GetBytes(content));

}

}

return new BadRequest();

}

return NextHandler != null ? NextHandler.Handle(request) : new NotImplemented();

}

}

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | | Текстовые документы | | | |  | | | |
|  | | | | |  | | | |  | | | |
| БГУИР КП 1–40 01 01 551005 014 ПЗ | | | | | Пояснительная записка | | | | 46 с. | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | | Графические документы | | | |  | | | |
|  | | | | |  | | | |  | | | |
| ГУИР 551005 014 СП | | | | | Схема программы | | | | Формат А1 | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  |  |  | |  |  | БГУИР КП 1-40 01 01 551005 014 Д1 | | | | | | |
|  |  |  | |  |  |
|  |  |  | |  |  |
| Изм. | Л. | № докум. | | Подп. | Дата | HTTP-Сервер  Ведомость курсового проекта |  | | | | Лист | Листов |
| Разраб. | | | Коваленко И.А. |  |  |  | Т |  | |  | 46 | 46 |
| Пров. | | | Ширай С.Ю. |  |  |  | Кафедра ПОИТ  гр. 551005 | | | | | |
|  | | |  |  |  |  |