МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М.Є. Жуковського

«Харківський авіаційний інститут»

Факультет радіоелектроніки, комп’ютерних систем та інфокомунікацій

Кафедра комп’ютерних систем, мереж і кібербезпеки

**Курсова робота**

|  |  |
| --- | --- |
| з | *Операційних систем* |
|  | (назва дисципліни) |
| на тему | «*Минимизация и форматирование файлов (HTML, CSS, JS)*.» |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Виконав: студент | | | *3* | курсу групи № | | | | *535а* |
| напряму підготовки (спеціальності) | | | | | | | | |
| *123 – комп’ютерна інженерія* | | | | | | | | |
| (шифр і назва напряму підготовки (спеціальності)) | | | | | | | | |
| *Маловічко Є.В.* | | | | | | | | |
| (прізвище й ініціали студента) | | | | | | | | |
| Керівник: | *Ст.викл. Годунов О.С.* | | | | | | | |
|  | | | | | | | | |
| (посада, науковий ступінь, прізвище й ініціали) | | | | | | | | |
| Національна шкала: | | | | |  | | | |
| Кількість балів: | | | | |  | | | |
| Оцінка ECTS: | | | | |  | | | |
| Члени комісії: | |  | | | |  |  | |
|  | | (підпис) | | | |  | (прізвище й ініціали) | |
|  | |  | | | |  |  | |
|  | | (підпис) | | | |  | (прізвище й ініціали) | |
|  | |  | | | |  |  | |
|  | | (підпис) | | | |  | (прізвище й ініціали) | |

Харків – 2020

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ](#page2) [3](#page2)

[1 ПОСТАНОВКА ЗАДАЧИ](#page3) [4](#page3)

1.1 Анализ предметной области и основные понятия 4

[2 ПРОЕКТИРОВАНИЕ](#page4) ..[9](#page4)

2.1 Выбор и обоснование архитектуры ...9

[3 РАЗРАБОТКА](#page11) [10](#page11)

[3.2 Разработка диаграммы классов](#page11) [10](#page11)

[3.3 Разработка и описание ключевых методов классов](#page12) [11](#page12)

4 ВЕРИФИКАЦИЯ 13

4.1 Верификация 13

ВЫВОДЫ 15

Приложение А. Техническое задание 16

Приложение Б. Исходные тексты программы 18

ВВЕДЕНИЕ

Работа посвящена анализу существующих методов форматирования, детальное рассмотрение минимизации кода, который передается браузеру посетителя в момент открытия сайта. Сюда относится как содержимое CSS-файлов и JS-файлов, так и HTML-код страниц. Минимизация происходит за счет удаления лишних пробелов, табуляции и пустых строк.

В наше время новые информационные технологии занимают очень важное место в специализированных сферах. С каждым днем передаваемые данные увеличиваются, что замедляет их передачу. Минимизация позволяет заметно сократить размер итогового кода, тем самым ускорить его передачу. Данная тема направлена на сферу Веб приложений.

Некоторые поисковые системы имеют ограничение по времени индексации сайта. Другими словами, если страницы не загружаются достаточно быстро, то сайт может значительно потерять в позициях поисковых систем.

1 ПОСТАНОВКА ЗАДАЧИ

В данной курсовой работе необходимо разработать программу позволяющую минимизировать и форматировать файлы (HTML, CSS, JS). Программа должна работать на операционной системе Windows на базе языка программирования C#. Интерфейс приложения должен быть консольным, но в дальнейшем может быть разработат полноценный графический интерфейс.

Программа должна выполнять следующие пункты для реализации курсовой работы: Минимизация и форматирование HTML, CSS и JS файлов.

* 1. Анализ предметной области и основные понятия

Оптимизация кода сайта - очень важный процесс, к которому нужно подойти с большой ответственностью. Рассмотрим оптимизацию HTML, CSS и JavaScript кода.

Оптимизация HTML кода сжатием пробелов. Это самый простой способ уменьшения веса кода. Уменьшение размера достигается за счет объединения пробелов и удаления переносов строк. Исходный код изображен на рис. 1.1.

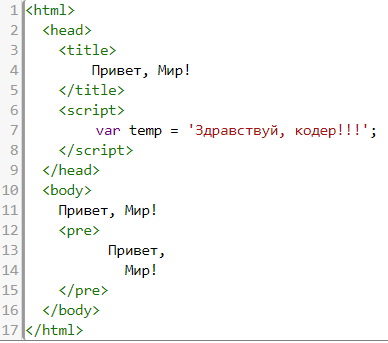


Рисунок 1.1 – Исходный код HTML файла

В таком случае исходный файл можно привести к виду (рис. 1.2).

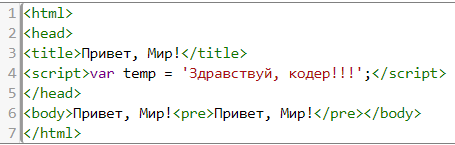


Рисунок 1.2 – Частично сжатый HTML файл

Так же код можно сжать еще сильней, записав все в одну строку, но тогда он будет совсем нечитаемым для больших HTML страниц.

Перенос подключаемых файлов ниже. Для того, чтобы ускорить загрузку страницы можно оптимизировать верхнюю часть кода страницы. Следует загрузить контент, то есть текст, а далее подключить необходимые css и js файлы. Таким образом, страница будет доступна для просмотра очень быстро, что положительно скажется на поисковых роботах, ведь им не особо важно оформление страниц, им важен контент. Пример оптимизации изображен на рис. 1.3 и на рис. 1.4.

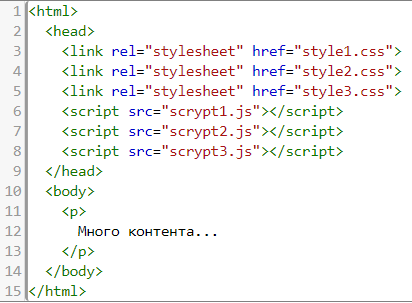


Рисунок 1.3 – Исходный HTML код

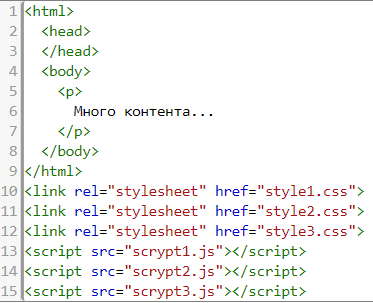


Рисунок 1.4 – Оптимизированый HTML код

Оптимизация CSS кода. За счет уменьшение количества файлов достигается уменьшение HTTP запросов к серверу. Это особенно актуально для мобильных сетей, где максимальное количество подключений ограничено. А также это хороший способ ускорить загрузку сайта. Пример оптимизации изображен на рис. 1.5 и на рис. 1.6.



Рисунок 1.5 – Исходный CSS код

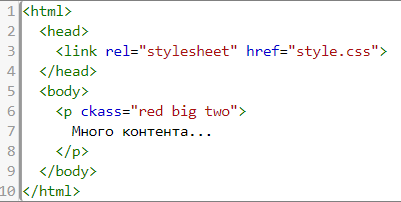


Рисунок 1.6 – Оптимизация CSS кода

Оптимизация CSS код за счет удаления форматирования. Код становится нечитабельным и не удобным для поддержки, но такие файлы занимают минимальное количество места. Пример оптимизации изображен на рис. 1.7 и на рис. 1.8.

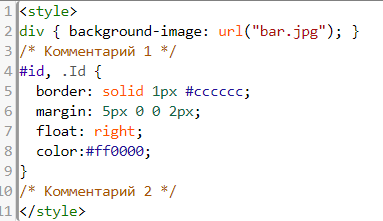


Рисунок 1.7 – Исходный CSS код



Рисунок 1.8 – Оптимизированый CSS код

Перемещение CSS кода в тег HEAD. Это действие обусловлено тем, что стили, которые прописаны в секции HEAD позволяют ускорить рендеринг и устраняют проблему перерисовки элементов, если стили прописаны после вывода элементов. Пример оптимизации изображен на рис. 1.9 и на рис. 1.10.



Рисунок 1.9 – Исходный код

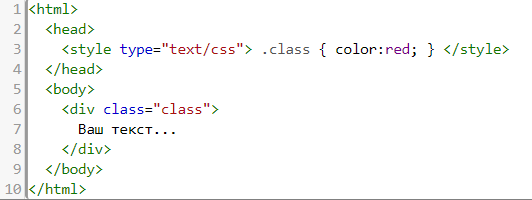


Рисунок 1.10 – Оптимизирваный код

Оптимизация JavaScript кода. Объединение JS-файлов в один. Этот способ аналогичен с объединением всех CSS-файлов в один файл. Меньше запросов к серверу - выше скорость загрузки страницы. Минимизация JavaScript кода за счет удаления форматирования. Этот прием позволяет максимально сжать файлы js.

1. ПРОЕКТИРОВАНИЕ

2.1 Выбор и обоснование архитектуры

Архитектура – это базовая организация системы, воплощенная в компонентах, их отношениях между собой и с окружением, а также принципы, определяющие проектирование и развитие системы.

В ходе разработки приложения, была разработана архитектура, демонстрирующая взаимодействие с данными пользовательского компьютера и программой.

Проведенный анализ выбираем двух уровневую архитектуру программы.

Уровни по предназначению:

– первый уровень – пользовательский интерфейс (UI);

– второй уровень – бизнес логика (BL) и взаимодействие с памятью компьютера (ST).

Данная архитектура была принята по соображениям взаимосвязи пользователя с программой, так как пользователю требуется только открыть файл конфигурации , в котором нужно добавить пути к файлам, которые нужно минимизировать и форматировать. После запуска программа сама выполнит нужные действия и сохранит минимизированое и форматированые версии файла по заданому пути который указывается в конфиг файле.

1. РАЗРАБОТКА
   1. Разработка диаграмов классов

Диаграмма классов (англ. Class diagram) - статическая представления структуры модели. Отражает статические (декларативные) элементы, такие как: классы, типы данных, их содержание и отношения. Диаграмма классов может содержать обозначение для пакетов и может содержать обозначение для вложенных пакетов. Также, диаграмма классов может содержать обозначение некоторых элементов поведения, однако их динамика раскрывается в других типах диаграмм. Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. На этой диаграмме показывают классы, интерфейсы, объекты и кооперации, а также их отношения.

При разработке диаграммы классов были использованы такие связи как: Ассоциация. Ассоциация показывает, что объекты одной сущности (класса) связаны с объектами другой сущности. Если между двумя классами определена ассоциация, то можно перемещаться от объектов одного класса к объектам другого. Вполне допустимы случаи, когда оба конца ассоциации относятся к одному и тому же классу. Это означает, что с объектом некоторого класса разрешено связать другие объекты из того же класса. Ассоциация, связывающая два класса, называется бинарной. Можно, хотя это редко бывает необходимым, создавать ассоциации, связывающие сразу несколько классов, они называются n-Арним. Графически ассоциация изображается в виде линии, соединяющей класс сам с собой или с другими классами. Отношение наследования является обычным таксономическим отношением между более общим элементом (предком) и более конкретным или специальным элементом (потомком). На диаграммах отношение обобщения обозначается сплошной линией с треугольной стрелкой на одном из концов. Стрелка указывает на общий класс (класс-предок или супер-класс), а ее отсутствие - на специальный класс (класс-потомок или подкласс). Диаграмма классов представлена ​​на рис. 3.1

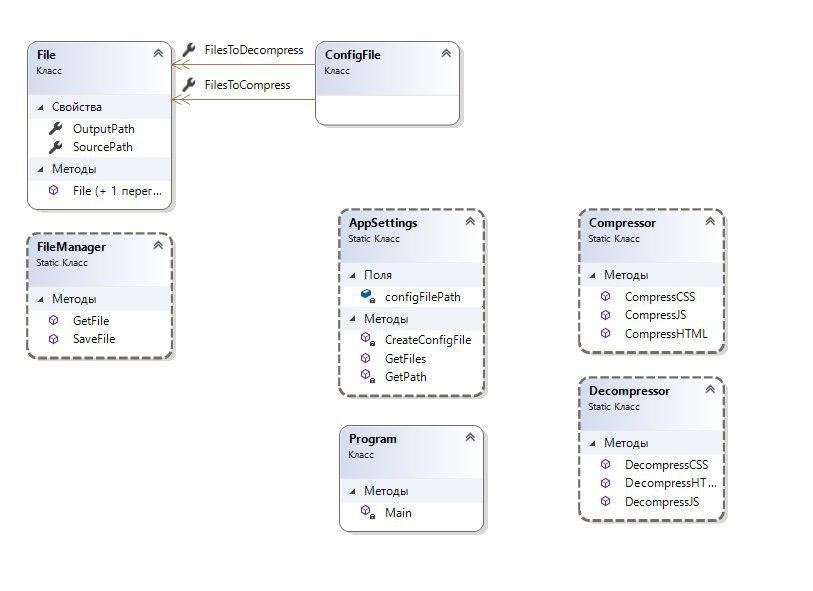


Рисунок 3.1 – Диаграмма классов

3.3 Разработка и описание ключевых методов классов

* + ходе разработки приложения было создано 7 классов, а именно: File, FileManager, ConfigFile, AppSettings, Program, Compressor, Decompressor.

Класс File – это класс, который представляет web-файл (CSS, JS и HTML) в приложении. Класс имеет такие свойства: SourcePath – путь к файлу; OutputPath – путь сохранения обработанного файла.

Класс ConfigFile – это класс, который содержит входные параметры. Класс имеет следующие свойства: FilesToCompress – список файлов для оптимизации; FilesToDecompress – список файлов для форматирования.

Класс FileManager – статический класс, который предназначен для работы с файлами (чтение и сохранение). Класс имеет следующие статические методы: GetFile – метод, чтения файла, который принимает в качестве аргумента путь к файлу, а возвращает данные в строкового типа; SaveFile – метод, сохранения файла в качестве аргументов передается путь к новому файлу и данные в виде строки.

Класс AppSettings – статический класс, который представляет конфигурацию приложения. Даный класс имеет приватное поле ConfigFilePath- путь к файлу конфигурации. И содержит следующие приватные методы: CreateConfigFile- создание файла конфигурации, вызывается, если файл конфигурации не найден; GetPath – возвращение пути файла конфигурации. Так же содержит публичный метод GetFiles – возвращает список файлов из конфиг файла.

Класс Compressor – статический класс, который содержит функционал для оптимизации файлов ( CSS, JS и HTML). Содержит публичные методы: CompressHTML – метод, который оптимизирует HTML код; CompressCSS – оптимизация CSS; CompressJS – оптимизация JS.

Класс Decompressor – статический класс, который содержит функционал для форматирования файлов ( CSS, JS и HTML). Содержит публичные методы: DecompressHTML – метод, который форматирование HTML код; DecompressCSS – форматирование CSS кода; DecompressJS – форматирование JS кода.

Класс Program – базовый класс, котрый содержит основную логику приложения.

4 ВЕРИФИКАЦИЯ И ТЕСТИРОВАНИЕ

4.1 Верификация

Верификация - проверка, способ подтверждения любых теоретических положений, алгоритмов, программ и процедур путем их сопоставления с опытными (эталонными или эмпирическими) данными, алгоритмами и программами. В частности, верификация проверяет соответствие между нормами стандартов, описанием требований (технического задания) к ПО, проектными решениями, исходным кодом, пользовательской документации и функционированием самого ПО.

Логика приложения простая, перед запуском необходимо настроить конфиг файл (добавить пути файла для сжатия и форматирования). Пример конфигурации показан на рис. 4.1

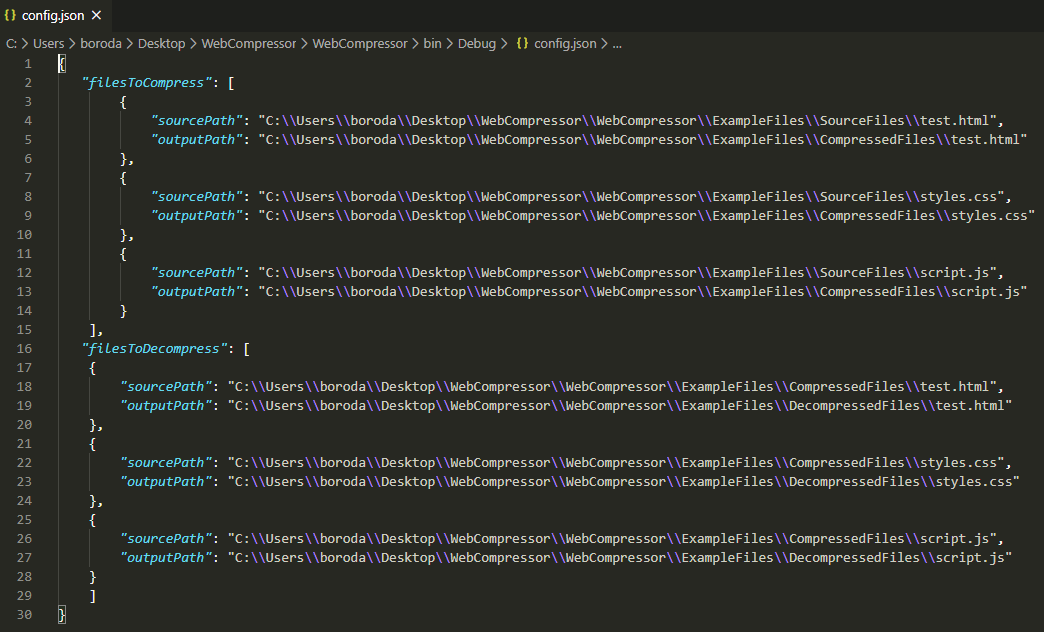


Рисунок 4.1 – Пример конфигурации

После настройки конфиг файла производим запуск приложения. Пример изображен на рис. 4.2.

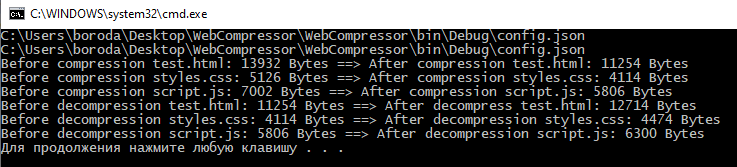


Рисунок 4.2 – Выполнение программы

Как можно заметить на рисунке 4.2 изображены действия программы, а также указано исходный размер файлов и размер полученного файла, после оптимизации и форматирования.

Если в конфиг файле указать путь на несуществующий файл, то программа остановится с ощибкой (рис. 4.3).

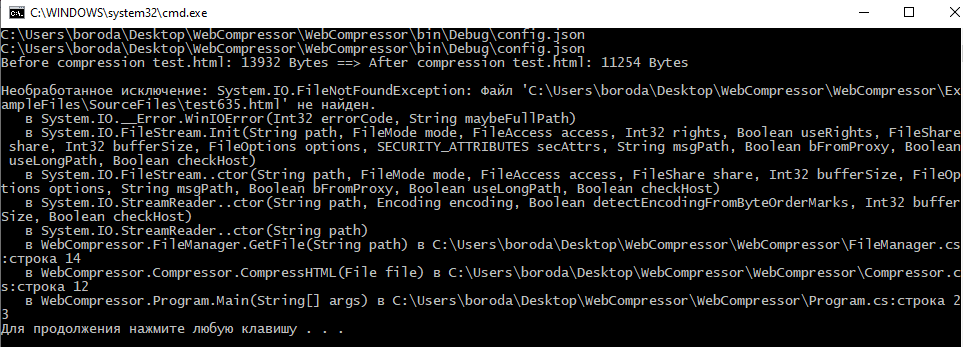


Рисунок 4.3 – Вывод ошибки

Как видно с рисунка 4.3 данный прецедент в программе не предусмотрен, что не нарушает требование ТЗ.

Если удалить конфиг файл, то программа сама создаст пустой конфиг файл(рис. 4.4)

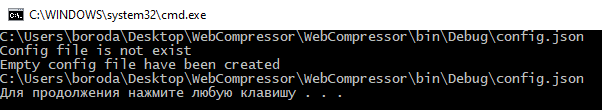


Рисунок 4.4 – Результат программы

ВЫВОДЫ

* данной курсовой работе была разработана программа минимизации и форматирование файлов ( HTML, JS, CSS). Рассмотрены основные принципы оптимизации, также
* результате анализа постановки задачи была и обоснована двухуровневая архитектура приложения. Для выполнения задания был язык программирования - C#. Разработана диаграмма классов.

Приложение, полученное в результате разработки, выполняет следующие функции:

- минимизация HTML, CSS, JS файлов;

- форматирование HTML, CSS, JS файлов;

- сохранение обработанных файлов.

Были проведены верификация приложения. Возможное дальнейшее развитие приложения:

- обработка непредусмотренных прецедентов;

- улучшение и совершенствование алгоритма.

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

1 Введение

1.1 Наиминование программы

Наиминование – “Минимизация и форматирование файлов (HTML, CSS и JS) ”

1.2 Краткая характеристика области применения

Приложение предназначено для облегчения работы и быстрой загрузки Web- страницы.

2. Основания для разработки

Основанием для разработки Программы является задание на курсовой

проект кафедры “Компьютерных систем,сетей и кибербезопасности”

503 Национального аэрокосмического университета “ХАИ” им. М.Е.

Жуковского , выданное 18.01.2020 на тему “ Минимизация и форматирование файлов (HTML, CSS и JS) ”.

3.Назначение разработки

3.1 Функциональное назначение

Приложение предназначено для облегчения работы и быстрой загрузки Web- страницы. И также минимизация и форматирование файлов.

3.2. Эксплуатационное назначение

Программу может использовать любой человек для собственных нужд.

Программа предназначена для эксплуатации на ПК.

Программа поставляется в виде исполняемых файлов ( .ехе –

файлов),устанавливается на комппьютер путем копирования в

соответствующую директорию и не требует обслуживания. Программа может конфигурироватся и запускатся с помощью других приложений.

4. Требования к программе или программному изделию

4.1 Требования к функциональным характеристикам

Программа должна выполнять следующие функции:

- Минимизация и форматирование указаных файлов.

4.2.1 Требование к организации входных данных

Указать путь к файлу и путь сохранения .

4.3 Требования к временным характеристикам

Требования к временным характеристикам приложения не

предъявляются.

4.4 Требования к надёжности

ПО относиться к классу системных программ , поэтому к надёжности

программы специальные требования не предъявляются .

4.5 Требования к информационной и программной совместимости

4.5.1 Требования к информационным структурам и методам

решений

Реализация методики ООП для информационных структур и методов

решения. Разбивка уровня логики проекта на классы.

4.5.2. Требования к исходным кодам и языкам программирования

При разработке программы должен быть использован язык

программирования С # .Среда разработки – Visual Studio 2019.

4.5.3 Требования к используемым программным средствам

В состав технических средств должны входить следующее ПО:

- ПК с операционной системой Windows 10 ,

- платформа .NET Framework 2.0 и выше,

- Microsoft Visual Studio 2019.

4.5.4 Требования к защите информации и программ

Требования к защите информации и программ не предъявляются.

5 Требования к программной документации

5.1 Предварительный состав программной документации

В результате разработки программы должна быть представлена

следующая программная документация :

1) Техническое задание.

2) Диаграммы классов.

3) Тексты программ.

4) Результаты верификации.

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЕ ТЕКСТЫ ПРОГРАММЫ

Текст программы File

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WebCompressor

{

public class File

{

public File() { }

public File(string sourcePatrh, string outputPath)

{

this.SourcePath = sourcePatrh;

this.OutputPath = outputPath;

}

public string SourcePath { get; set; }

public string OutputPath { get; set; }

}

}

Текст программы FileManage

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WebCompressor

{

public static class FileManager

{

public static string GetFile(string path)

{

using (StreamReader file = new StreamReader(path))

{

return file.ReadToEnd();

}

}

public static void SaveFile(string data, string path)

{

using (StreamWriter file = new StreamWriter(path))

{

file.Write(data);

file.Close();

}

}

}

}

Текст программы AppSettings

using Newtonsoft.Json;

using Newtonsoft.Json.Linq;

using System;

using System.Collections.Generic;

using System.IO;

namespace WebCompressor

{

static class AppSettings

{

private static string configFilePath = GetPath();

public static ConfigFile GetFiles()

{

Console.WriteLine(configFilePath);

if (System.IO.File.Exists(configFilePath))

{

using (StreamReader configFile = new StreamReader(configFilePath))

{

var config = JsonConvert.DeserializeObject<ConfigFile>(configFile.ReadToEnd());

return config;

}

}

else

{

CreateConfigFile();

Console.WriteLine("Config file is not exist");

Console.WriteLine("Empty config file have been created");

using (StreamReader configFile = new StreamReader(configFilePath))

{

var config = JsonConvert.DeserializeObject<ConfigFile>(configFile.ReadToEnd());

return config;

}

}

}

private static string GetPath()

{

var exePath = AppDomain.CurrentDomain.BaseDirectory;

return Path.Combine(exePath, "config.json");

}

private static void CreateConfigFile()

{

using (FileStream fstream = new FileStream(configFilePath, FileMode.OpenOrCreate))

{

var json = "{ \n \"filesToCompress\": [\n {\n \"sourcePath\": \"\",\n \"outputPath\": \"\"\n }\n ],\n \"filesToDecompress\": [\n {\n \"sourcePath\": \"\",\n \"outputPath\": \"\"\n }\n ]\n}";

byte[] array = System.Text.Encoding.Default.GetBytes(json);

fstream.Write(array, 0, array.Length);

}

}

}

}

Текст программы ConfigFile

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WebCompressor

{

public class ConfigFile

{

public IEnumerable<File> FilesToCompress { get; set; }

public IEnumerable<File> FilesToDecompress { get; set; }

}

}

Текст программы Compressor

using System;

using System.IO;

using System.Linq;

namespace WebCompressor

{

public static class Compressor

{

public static void СompressHTML(File file)

{

var source = FileManager.GetFile(file.SourcePath);

Console.Write("Before compression " + file.SourcePath.Split('\\').Last() + ": " + source.Length \* sizeof(char) + " Bytes");

var compressed = source;

bool isNotDone = true;

while (isNotDone)

{

isNotDone = false;

while (compressed.Contains("\n"))

{

compressed = compressed.Replace("\n", "");

isNotDone = true;

}

while (compressed.Contains("\t<"))

{

compressed = compressed.Replace("\t<", "<");

isNotDone = true;

}

while (compressed.Contains(" <"))

{

compressed = compressed.Replace(" <", "<");

isNotDone = true;

}

while (compressed.Contains("> "))

{

compressed = compressed.Replace("> ", ">");

isNotDone = true;

}

while (compressed.Contains("\" "))

{

compressed = compressed.Replace("\" ", "\" ");

isNotDone = true;

}

}

Console.WriteLine(" ==> After compression " + file.SourcePath.Split('\\').Last() + ": " + compressed.Length \* sizeof(char) + " Bytes");

FileManager.SaveFile(compressed, file.OutputPath);

}

public static void CompressCSS(File file)

{

var source = FileManager.GetFile(file.SourcePath);

Console.Write("Before compression " + file.SourcePath.Split('\\').Last() + ": " + source.Length \* sizeof(char) + " Bytes");

var compressed = source;

bool isNotDone = true;

while (isNotDone)

{

isNotDone = false;

while (compressed.Contains("{ "))

{

compressed = compressed.Replace("{ ", "{");

isNotDone = true;

}

while (compressed.Contains(" {"))

{

compressed = compressed.Replace(" {", "{");

isNotDone = true;

}

while (compressed.Contains("} "))

{

compressed = compressed.Replace("} ", "}");

isNotDone = true;

}

while (compressed.Contains(" }"))

{

compressed = compressed.Replace(" }", "}");

isNotDone = true;

}

while (compressed.Contains("; "))

{

compressed = compressed.Replace("; ", ";");

isNotDone = true;

}

while (compressed.Contains("\n"))

{

compressed = compressed.Replace("\n", "");

isNotDone = true;

}

while (compressed.Contains(", "))

{

compressed = compressed.Replace(", ", ",");

isNotDone = true;

}

}

Console.WriteLine(" ==> After compression " + file.SourcePath.Split('\\').Last() + ": " + compressed.Length \* sizeof(char) + " Bytes"); ;

FileManager.SaveFile(compressed, file.OutputPath);

}

public static void CompressJS(File file)

{

var source = FileManager.GetFile(file.SourcePath);

Console.Write("Before compression " + file.SourcePath.Split('\\').Last() + ": " + source.Length \* sizeof(char) + " Bytes");

var compressed = source;

bool isNotDone = true;

while (isNotDone)

{

isNotDone = false;

while (compressed.Contains("\n"))

{

compressed = compressed.Replace("\n", string.Empty);

isNotDone = true;

}

while (compressed.Contains("\r"))

{

compressed = compressed.Replace("\r", string.Empty);

isNotDone = true;

}

while (compressed.Contains("\t"))

{

compressed = compressed.Replace("\t", string.Empty);

isNotDone = true;

}

while (compressed.Contains("; "))

{

compressed = compressed.Replace("; ", ";");

isNotDone = true;

}

while (compressed.Contains(" return"))

{

compressed = compressed.Replace(" return", " return");

isNotDone = true;

}

while (compressed.Contains(" return"))

{

compressed = compressed.Replace(" return", ";return");

isNotDone = true;

}

while (compressed.Contains("{ "))

{

compressed = compressed.Replace("{ ", "{");

isNotDone = true;

}

while (compressed.Contains(" {"))

{

compressed = compressed.Replace(" {", "{");

isNotDone = true;

}

while (compressed.Contains("} "))

{

compressed = compressed.Replace("} ", "}");

isNotDone = true;

}

while (compressed.Contains(" }"))

{

compressed = compressed.Replace(" }", "}");

isNotDone = true;

}

while (compressed.Contains(" ("))

{

compressed = compressed.Replace(" (", "(");

isNotDone = true;

}

}

Console.WriteLine(" ==> After compression " + file.SourcePath.Split('\\').Last() + ": " + compressed.Length \* sizeof(char) + " Bytes"); ;

FileManager.SaveFile(compressed, file.OutputPath);

}

}

}

Текст программы Decompressor

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WebCompressor

{

public static class Decompressor

{

public static void DecompressHTML(File file)

{

string[] emptyTags = new string[] { "area", "base", "br", "col", "embed", "hr", "img", "input", "link", "menuitem", "meta", "param", "source", "track", "wbr" };

var source = FileManager.GetFile(file.SourcePath);

Console.Write("Before decompression " + file.SourcePath.Split('\\').Last() + ": " + source.Length \* sizeof(char) + " Bytes");

var decompressed = source;

bool isNotDone = true;

int progressIndex = 0;

int level = 0;

bool isOpenedTag = false;

while (isNotDone)

{

int startIndex = decompressed.IndexOf("<", progressIndex);

if (startIndex == -1) break;

int endIndex;

if (decompressed[startIndex + 1] != '/')

{

endIndex = decompressed.IndexOf(">", startIndex);

if (endIndex == -1) break;

while (decompressed.Substring(startIndex, endIndex - startIndex + 1).Where(x => x == '\"').Count() % 2 != 0)

{

endIndex = decompressed.IndexOf("\">", endIndex);

}

string insertStr = "\n";

level++;

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

{

insertStr = insertStr + "\t";

}

isOpenedTag = true;

if (emptyTags.Contains(decompressed.Substring(startIndex + 1, endIndex - startIndex).Split(' ', '>').First()))

{

level--;

isOpenedTag = false;

}

decompressed = decompressed.Insert(startIndex, insertStr);

}

else

{

endIndex = decompressed.IndexOf(">", startIndex);

if (endIndex == -1) break;

endIndex = endIndex + 1;

string insertStr = "\n";

if (!isOpenedTag)

{

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

{

insertStr = insertStr + "\t";

}

decompressed = decompressed.Insert(startIndex, insertStr);

endIndex = endIndex + level;

}

endIndex = endIndex - level - 1;

level--;

isOpenedTag = false;

}

progressIndex = endIndex + level + 2;

}

Console.WriteLine(" ==> After decompress " + file.SourcePath.Split('\\').Last() + ": " + decompressed.Length \* sizeof(char) + " Bytes");

FileManager.SaveFile(decompressed, file.OutputPath);

}

public static void DecompressCSS(File file)

{

var source = FileManager.GetFile(file.SourcePath);

Console.Write("Before decompression " + file.SourcePath.Split('\\').Last() + ": " + source.Length \* sizeof(char) + " Bytes");

var decompressed = source;

decompressed = decompressed.Replace(",", ", ");

decompressed = decompressed.Replace(", ", ", ");

decompressed = decompressed.Replace("{", " {\n\t");

decompressed = decompressed.Replace(" {", " {");

decompressed = decompressed.Replace(";", ";\n\t");

decompressed = decompressed.Replace("\t}", "}\n");

Console.WriteLine(" ==> After decompression " + file.SourcePath.Split('\\').Last() + ": " + decompressed.Length \* sizeof(char) + " Bytes");

FileManager.SaveFile(decompressed, file.OutputPath);

}

public static void DecompressJS(File file)

{

var source = FileManager.GetFile(file.SourcePath);

Console.Write("Before decompression " + file.SourcePath.Split('\\').Last() + ": " + source.Length \* sizeof(char) + " Bytes");

var decompressed = source;

bool isDone = false;

int progressIndex = 0;

int level = 0;

while (!isDone)

{

var index = decompressed.IndexOfAny(new char[] { '{', '}', ';', 'f' }, progressIndex);

if (index == -1)

{

break;

}

if (decompressed[index] == '{')

{

string insertStr = "\n";

level++;

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

{

insertStr = insertStr + " ";

}

decompressed = decompressed.Insert(index + 1, insertStr);

decompressed = decompressed.Insert(index, " ");

progressIndex = index + level + 1;

}

else if (decompressed[index] == ';')

{

string insertStr = "\n";

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

{

insertStr = insertStr + " ";

}

decompressed = decompressed.Insert(index + 1, insertStr);

progressIndex = index + 1;

}

else if (decompressed[index] == 'f')

{

if (decompressed.Length > 5 + index)

{

string isFor = decompressed.Substring(index, 4);

if(isFor == "for("){

progressIndex = decompressed.IndexOf("{", index);

}

else

{

progressIndex = index + 1;

}

}

else

{

progressIndex = index + 1;

}

}

else

{

string insertStr = "\n";

level--;

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

{

insertStr = insertStr + " ";

}

decompressed = decompressed.Insert(index+1, insertStr);

decompressed = decompressed.Remove(index - 2, 2);

progressIndex = index ;

}

}

Console.WriteLine(" ==> After decompression " + file.SourcePath.Split('\\').Last() + ": " + decompressed.Length \* sizeof(char) + " Bytes");

FileManager.SaveFile(decompressed, file.OutputPath);

}

}

}

Текст программы Program

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WebCompressor

{

class Program

{

static void Main(string[] args)

{

IEnumerable<File> files = AppSettings.GetFiles().FilesToCompress;

IEnumerable<File> filesToDecompress = AppSettings.GetFiles().FilesToDecompress;

foreach (var file in files)

{

var format = file.SourcePath.Split('.').Last().ToUpper();

switch (format)

{

case "HTML":

Compressor.СompressHTML(file);

break;

case "CSS":

Compressor.CompressCSS(file);

break;

case "JS":

Compressor.CompressJS(file);

break;

}

}

foreach (var file in filesToDecompress)

{

var format = file.SourcePath.Split('.').Last().ToUpper();

switch (format)

{

case "HTML":

Decompressor.DecompressHTML(file);

break;

case "CSS":

Decompressor.DecompressCSS(file);

break;

case "JS":

Decompressor.DecompressJS(file);

break;

}

}

}