API FINAL PROJECT ITS462

Sharyl Riley

Riley171@pnw.edu

https://github.com/sriley86

December 11, 2022

TABLE OF CONTENTS

STATEMENT OF WORK

Introduction

The application for this project provides a tool for a user to compare price information of various computer products from various websites. The application will be designed as a client application that uses services hosted on a server that get data from a database and provide it to the client application.

Server-Side

The server-side of the application will scrape and retrieve information from at least two websites that have computer products (desktop computers, laptops, tablets, phones, etc.). The information that must be scraped includes: 1) the site offering the products for sale, 2) the product vendor name, 3) product model, product description, product price, and optionally any product specs if available.

- A website that includes links to an example scraper test site with computer product information is https://webscraper.io/test-sites. You can find another site for the second source of computer product sales.
- the application will use a third-party tool to scrape the website to extract information. You can pick a tool of your choosing. An example of such a tool is “HTML Agility Pack”. It is available as a NuGet package, and it scrapes the webpage HTML and builds a DOM with the scraped data (XPATH and XSLT is supported).
    - Html Agility Pack website: (http://htmlagilitypack.codeplex.com/)
- The server-side of the application will store the scraped data in a database.
    - You can pick any database system to store the data.
- The server-side of the application will implement either WCF or ASMX services that can be called by the client application to retrieve the specific scraped data from the database.

Client-Side

A client will be implemented as a Windows or Web form application with the following requirements:

- A list or drop-down control that allows the user to select the type of data on which a price comparison is desired (this is a querying capability).
    - The query list must include at least: 1) computer type (desktop, laptop, tablet, phone), 2) computer vendor, 3) model, and 4) price.
    - The client application must have a control to display all the data retrieved from the database by the user selection. The database query must return and display all the information in the database about the selected product.
    - The results will be shown in a price ascending order.
- The client application will have a feature to display the retrieved data in a report format that can be printed.

Deliverables

I. A Software Requirements Specification and Software Design document will be compiled for the application.
II. A working, high-quality application.

III. An electronic copy of your project report which includes the requirements, software design, user manual, source code, database file, and screen shots. Please zip everything and submit to Brightspace. Each team member MUST individually submit the zip file to Brightspace (same file for both team members).

SOFTWARE REQUIREMENTS SPECIFICATIONS

Introduction

- The purpose of this project is to develop a tool for users to compare prices for computer products from various websites. The application will be designed as a client application that uses services hosted on a server to retrieve data from a database.
- The scope of this project includes the development of both the client and server-side of the application, as well as the scraping, storage, and retrieval of data from websites that have computer products for sale. The application will provide a user-friendly interface for users to select the type of data on which they want to perform a price comparison and will display the results of their query in a report format that can be printed.
- The background of this project is based on the need for users to have a simple and efficient way to compare prices for computer products from multiple websites. By scraping and storing data from these websites and providing a user-friendly interface for users to perform price comparisons, the application will help users make informed decisions about which products to purchase.

User Requirements

The application will be used by two primary user groups: end users who are interested in comparing prices for computer products, and administrators who are responsible for managing the data that is scraped from the various websites.

End users will need to be able to select the type of data on which they want to perform a price comparison, such as vendor, model, or price. They will also need to be able to view the results of their query, which will be displayed in ascending order by price. Finally, they will need to be able to print the results of their query in a report format.

Administrators will need to be able to access the server-side of the application to scrape and retrieve information from websites that have computer products for sale. They will also need to be able to store the scraped data in a database and implement services that can be called by the client application to retrieve the specific data from the database.

Overall, the user requirements for this project are focused on providing end users with a simple and intuitive tool for comparing prices for computer products and providing administrators with the necessary tools and features for managing the scraped data and making it available to end users.

Functional Requirements

- The application must provide a client application that uses services hosted on a server to retrieve data from a database.
- The server-side of the application must be able to scrape and retrieve information from at three websites:

  - https://webscraper.io/test-sites/e-commerce/allinone/computers/laptops
  - https://webscraper.io/test-sites/e-commerce/allinone/computers/tablets
  - https://webscraper.io/test-sites/e-commerce/allinone/phones/touch
- The scraped data must include the site offering the products for sale, the product vendor name, product description and product price.
- The application must use a third-party tool, HTML Agility Pack, to scrape the website and extract the required information.
- The server-side of the application must be able to store the scraped data in an SQLite database.
- The server-side of the application must implement ASMX services that can be called by the client application to retrieve the specific scraped data from the database.
- The client application must provide a drop-down control that allows users to select the type of data on which they want to perform a price comparison.
- The query list must include at least the options to compare prices by computer type, vendor, and price.
- The client application must have a DataGrid to display all the data retrieved from the database based on the user's selection.
- The results must be displayed in ascending order by price.
- The client application must have a feature to display the retrieved data in a report format that can be printed.

Non-Functional Requirements

- The application must be secure and protect the privacy of user data.
- The application must be user-friendly and easy to use, with a simple and intuitive user interface.
- The application must be able to handle large amounts of data efficiently, without significant performance degradation.
- The application must be scalable, and able to support a growing number of data sources over time.
- The application must be reliable, and able to provide consistent and accurate results for users.
- The application must be maintainable, with clear and well-documented code, and a modular design that allows for easy updates and modifications.
- The application must be compatible with the target platform and any other hardware or software dependencies, as specified in the Design Constraints section of the SRS.

Overall, the system requirements for this project are focused on providing a functional, user-friendly, and reliable application that meets the needs of end users and administrators and is implemented in a scalable and maintainable manner.

Design Constraints

- The application must be designed as a client application that uses services hosted on a server to retrieve data from a database.
- The server-side of the application must use a third-party tool, HTML Agility Pack, to scrape and extract information from websites that have computer products for sale.
- The server-side of the application must store the scraped data in an SQLite database and implement services that can be called by the client application to retrieve the specific data.
- The client application must be implemented as a Windows form application, with a user interface that allows users to select the type of data on which they want to perform a price comparison and display the results of their query.
- The application must be compatible with the target platform, and any other hardware or software dependencies, such as operating systems, databases, or development tools.
- The application must follow industry best practices and standards for software development, such as coding conventions, version control, and testing.

Overall, the design constraints for this project are focused on ensuring that the application is implemented in a consistent and reliable manner and is compatible with the target platform and any other dependencies. These constraints will help guide the design and implementation of the application and ensure that it meets the functional and non-functional requirements outlined in the SRS.

USER MANUAL

Overview

The application for this project provides a tool for users to compare prices for computer products from various websites. The user manual provides instructions on how to use the application including how to perform a price comparison.

The user manual provides users with the information and guidance they need to use the application effectively and efficiently. By following the instructions provided in the user manual, users should be able to easily and confidently use the application to compare prices for computer products from various websites.

Service-Side Application

Administrator manages scraped data
Precondition: Administrator is running the server-side of the application and the database is running

1. Administrator accesses the scraping service and specifies the websites to scrape
2. The scraping service retrieves the data from the specified websites and stores it in the database
3. Administrator accesses the database and verifies that the data has been successfully stored
4. Administrator accesses the services that are used by the client application to retrieve data from the database
5. Administrator verifies that the services are functioning correctly and can be called by the client application to retrieve the appropriate data

Client-Side Application

Use Case 1: End user selects type of data for price comparison
Precondition: End user has launched the client application

1. End user selects "Select Type of Price Comparison" from the dropdown menu
2. End user is presented with a list of options for the type of data on which to perform the comparison (e.g., all products, laptops, tablets, phones, by vendor)
3. End user selects the desired option from the list
4. End user is presented with a list of products matching the selected criteria, sorted in ascending order by price
5. End user selects the "Create Report" option from the main menu
6. End user is presented with a message “Report.txt Has Finished Downloading”
7. End user confirms the report.txt has been created
8. End user opens report containing the results of their price comparison
9. End user prints the report
10.

Use Case 2: End user views results of price comparison
Precondition: End user has performed a price comparison and the results are displayed in the client application

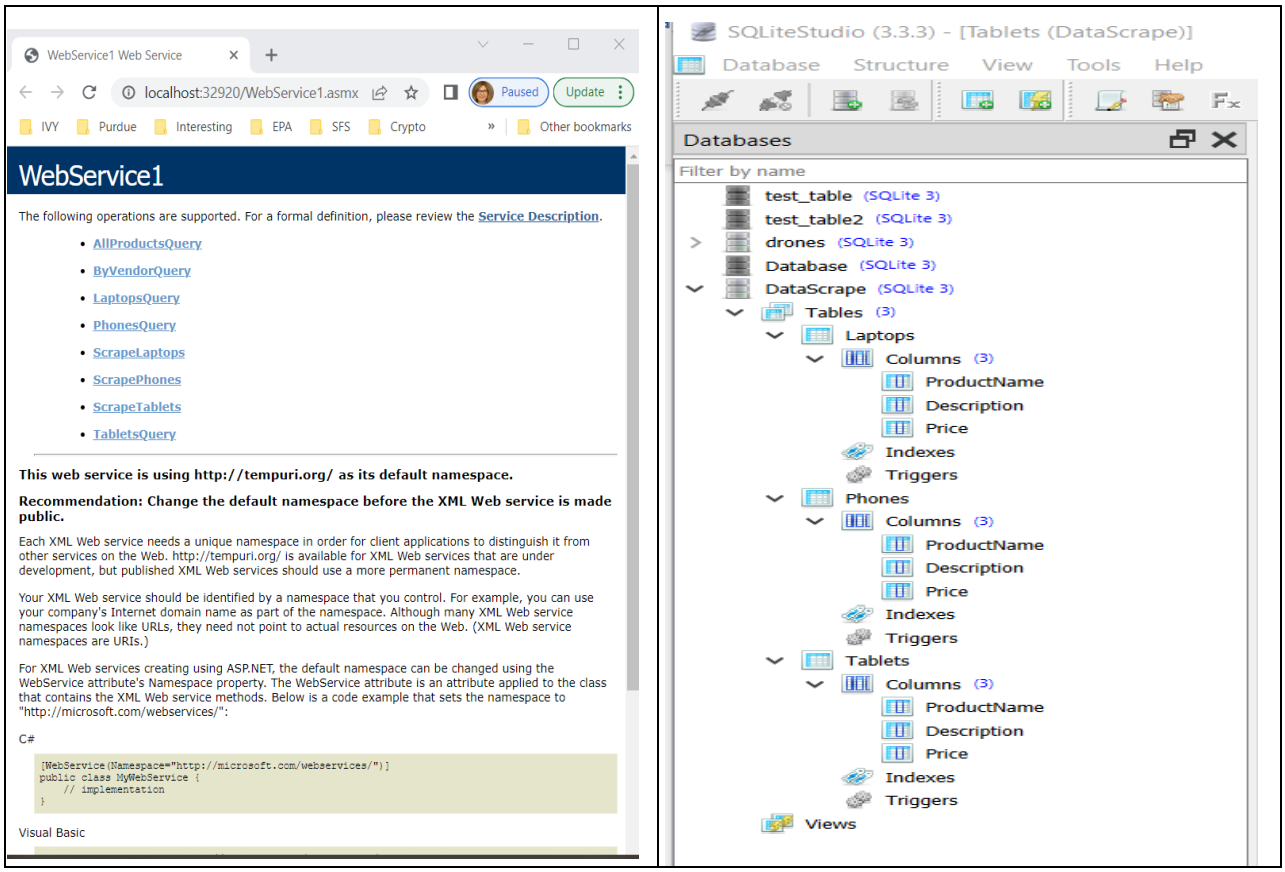
1. End user scrolls through the list of products to view the details of each product
2. End user can see the product name, description and, price for each product

SCREENSHOTS & CODE

Screenshots

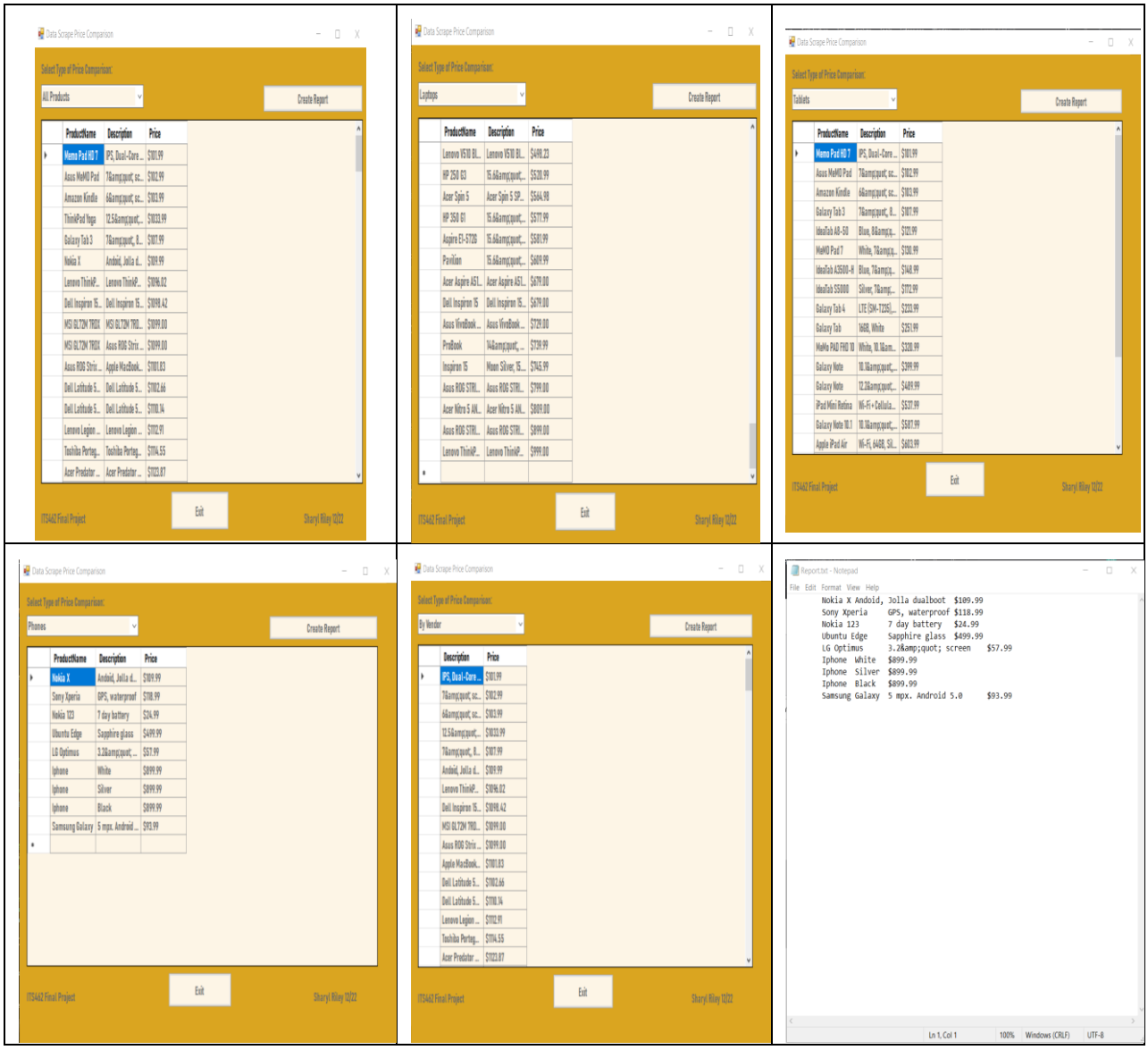
Service

- Home Page
- Database



Client

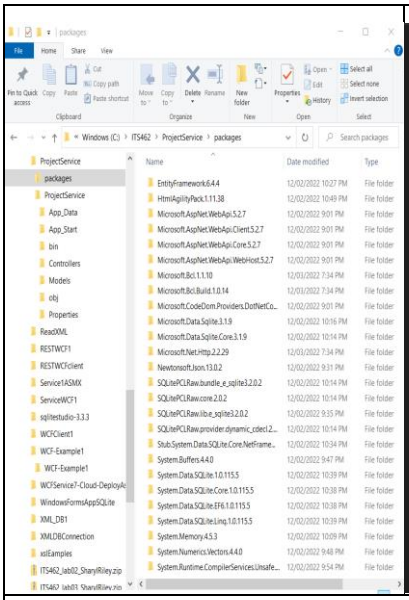
- All Products Query
- Laptops
- Tablets
- Phones
- By Vendor
- Report



Code

Service

- Packages Installed
- Products.cs
- Web.Config
- WebService1.asmx.cs
-



```
ProjectService > ProjectService > Products.cs > ...
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5
6  namespace ProjectService
7  {
       9 references
8      public class Products
9      {
10
               9 references
11             public string ProductName { get; set; }
               6 references
12             public string Description { get; set; }
               6 references
13             public string Price { get; set; }
14
15
16     }
17 }
```

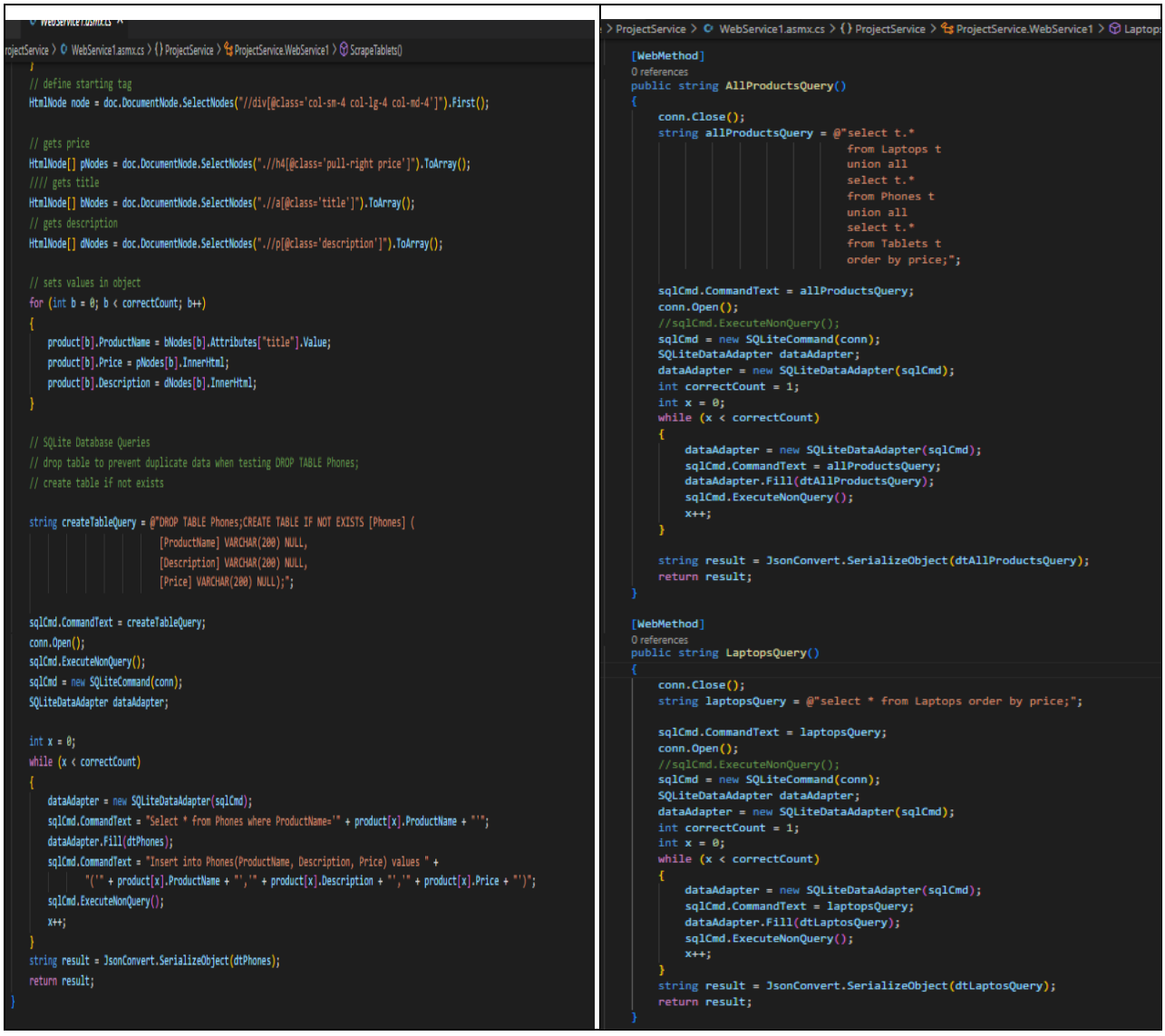
```
ProjectService > ProjectService > Web.config > configuration > system.web > webServices
1  <?xml version="1.0" encoding="utf-8"?>
2  <!--
3    For more information on how to configure your ASP.NET application, ple
4    https://go.microsoft.com/fwlink/?LinkId=301879
5    -->
6  <configuration>
7    <configSections>
8      <!-- For more information on Entity Framework configuration, visit h
9      <section name="entityFramework" type="System.Data.Entity.Internal.Co
10   </configSections>
11   <appSettings>
12   </appSettings>
13   <system.web>
14     <webServices>
15       <protocols>
16         <add name="HttpGet" />
17       </protocols>
18     </webServices>
19     <compilation debug="true" targetFramework="4.7.2" />
20     <httpRuntime targetFramework="4.7.2" />
21   </system.web>
22   <system.webServer>
```

```
ProjectService > ProjectService > WebService1.asmx.cs > ...
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Services;
6
7  //added using statements
8  using Newtonsoft.Json;
9  using System.Data;
10 using System.Xml;
11 using System.Xml.Serialization;
12 using System.IO;
13 using System.Text;
14 using System.Data.SQLite;
15 using HtmlAgilityPack;
16
17 namespace ProjectService
18 {
19     /// <summary>
20     /// Summary description for WebService1
21     /// </summary>
22     [WebService(Namespace = "http://tempuri.org/")]
23     [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
24     [System.ComponentModel.ToolboxItem(false)]
25     // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
26     // [System.Web.Script.Services.ScriptService]
       0 references
27     public class WebService1 : System.Web.Services.WebService
28     {
29         //Declare  an application path.
           1 reference
30         static string appPath = @"C:\ITS462\ProjectService\ProjectService\";
31         //Set sqllite connection
           25 references
32         static SQLiteConnection conn = new SQLiteConnection("Data Source=" + appPath + "\\DataScrape.db; Version=3;New=True;Compress=True;");
           51 references
33         SQLiteCommand sqlCmd = new SQLiteCommand(conn);
34
35         // Create DataTables for the WebMethods
           2 references
36         DataTable dtLaptops = new DataTable();
           2 references
37         DataTable dtTablets = new DataTable();
           2 references
38         DataTable dtPhones = new DataTable();
           2 references
39         DataTable dtAllProductsQuery = new DataTable();
           2 references
40         DataTable dtLaptosQuery = new DataTable();
           2 references
41         DataTable dtTabletsQuery = new DataTable();
           2 references
42         DataTable dtPhonesQuery = new DataTable();
           2 references
43         DataTable dtByVendorQuery = new DataTable();
```

```
ProjectService > WebService1.asmx.cs > ...
[WebMethod]
0 references
public string ScrapeLaptops()
{
    //close connection from previous scrape method
    conn.Close();
    //Scrape first website for LAPTOPS
    string url = "https://webscraper.io/test-sites/e-commerce/allinone/computers/laptops";
    HtmlWeb web = new HtmlWeb();
    var doc = web.Load(url);

    // gets number of items on page
    int count = doc.DocumentNode.SelectNodes("//div[@class='col-sm-4 col-lg-4 col-md-4']").Count();
    // adjusts count for the 0 start in array
    int correctCount = +count;

    // define array and initiliaze all objects within object array
    Products[] product = new Products[correctCount];
    for (int y = 0; y < correctCount; y++)
    {
        product[y] = new Products();
    }

    // define starting tag
    HtmlNode node = doc.DocumentNode.SelectNodes("//div[@class='col-sm-4 col-lg-4 col-md-4']").First();
    // Scrapes price - uses MTML code from the webpage
    HtmlNode[] pNodes = doc.DocumentNode.SelectNodes(".//h4[@class='pull-right price']").ToArray();
    // Scrapes title/ProductName - uses MTML code from the webpage
    HtmlNode[] bNodes = doc.DocumentNode.SelectNodes(".//a[@class='title']").ToArray();
    // Scrapes description - uses MTML code from the webpage
    HtmlNode[] dNodes = doc.DocumentNode.SelectNodes(".//p[@class='description']").ToArray();

    // stores values in object
    for (int b = 0; b < correctCount; b++)
    {
        product[b].ProductName = bNodes[b].Attributes["title"].Value;
        product[b].Price = pNodes[b].InnerHtml;
        product[b].Description = dNodes[b].InnerHtml;
    }

    // SQLite Database Queries
    // drop table to prevent duplicate data when testing DROP TABLE Laptops;
    // create table if not exists
    string createTableQuery = @" DROP TABLE Laptops; CREATE TABLE IF NOT EXISTS [Laptops] (
                                 [ProductName] VARCHAR(200) NULL,
                                 [Description] VARCHAR(200) NULL,
                                 [Price] VARCHAR(200) NULL);";

    sqlCmd.CommandText = createTableQuery;
    conn.Open();
    sqlCmd.ExecuteNonQuery();
    sqlCmd = new SQLiteCommand(conn);
    SQLiteDataAdapter dataAdapter;
```

```
    // SQLite query to Insert scraped data into database
    int x = 0;
    while (x < correctCount)
    {
        dataAdapter = new SQLiteDataAdapter(sqlCmd);
        sqlCmd.CommandText = "Select * from Laptops where ProductName='" + product[x].ProductName + "'";
        dataAdapter.Fill(dtLaptops);
        sqlCmd.CommandText = "Insert into Laptops(ProductName, Description, Price) values " +
                "('" + product[x].ProductName + "','" + product[x].Description + "','" + product[x].Price + "')";
        sqlCmd.ExecuteNonQuery();
        x++;
    }
    string result = JsonConvert.SerializeObject(dtLaptops);
    return result;
}

[WebMethod]
0 references
public string ScrapeTablets()
{

    //close connection from previous scrape method
    conn.Close();
    //Scrape second website for LAPTOPS
    string url = "https://webscraper.io/test-sites/e-commerce/allinone/computers/tablets";
    HtmlWeb web = new HtmlWeb();
    var doc = web.Load(url);

    // gets number of items on page
    int count = doc.DocumentNode.SelectNodes("//div[@class='col-sm-4 col-lg-4 col-md-4']").Count();
    // adjusts count for the 0 start in array
    int correctCount = +count;

    // define array and initiliaze all objects within object array
    Products[] product = new Products[correctCount];
    for (int y = 0; y < correctCount; y++)
    {
        product[y] = new Products();
    }
    // define starting tag
    HtmlNode node = doc.DocumentNode.SelectNodes("//div[@class='col-sm-4 col-lg-4 col-md-4']").First();

    // gets price
    HtmlNode[] pNodes = doc.DocumentNode.SelectNodes(".//h4[@class='pull-right price']").ToArray();
    //// gets title
    HtmlNode[] bNodes = doc.DocumentNode.SelectNodes(".//a[@class='title']").ToArray();
    // gets description
    HtmlNode[] dNodes = doc.DocumentNode.SelectNodes(".//p[@class='description']").ToArray();

    // sets values in object
    for (int b = 0; b < correctCount; b++)
    {
        product[b].ProductName = bNodes[b].Attributes["title"].Value;
        product[b].Price = pNodes[b].InnerHtml;
        product[b].Description = dNodes[b].InnerHtml;
    }
```

```
ProjectService 〉 WebService1.asmx.cs 〉 ...
    }
    // SQLite Database Queries
    // drop table to prevent duplicate data when testing DROP TABLE Tablets;
    // create table if not exists
    string createTableQuery = @"DROP TABLE Tablets; CREATE TABLE IF NOT EXISTS [Tablets] (
                              [ProductName] VARCHAR(200) NULL,
                              [Description] VARCHAR(200) NULL,
                              [Price] VARCHAR(200) NULL);";

    sqlCmd.CommandText = createTableQuery;

    conn.Open();
    sqlCmd.ExecuteNonQuery();

    // conn.Close();
    sqlCmd = new SQLiteCommand(conn);
    SQLiteDataAdapter dataAdapter;
    int x = 0;
    while (x < correctCount)
    {
        dataAdapter = new SQLiteDataAdapter(sqlCmd);
        sqlCmd.CommandText = "Select * from Tablets where ProductName='" + product[x].ProductName + "'";
        dataAdapter.Fill(dtTablets);
        sqlCmd.CommandText = "Insert into Tablets(ProductName, Description, Price) values " +
                "('" + product[x].ProductName + "','" + product[x].Description + "','" + product[x].Price + "')";
        sqlCmd.ExecuteNonQuery();
        x++;
    }
    string result = JsonConvert.SerializeObject(dtTablets);
    return result;
}

[WebMethod]
0 references
public string ScrapePhones()
{
    //Scrape third website for LAPTOPS
    //close connection from previous scrape method
    conn.Close();
    //Scrape third website for LAPTOPS
    string url = "https://webscraper.io/test-sites/e-commerce/allinone/phones/touch";
    HtmlWeb web = new HtmlWeb();
    var doc = web.Load(url);

    // gets number of items on page
    int count = doc.DocumentNode.SelectNodes("//div[@class='col-sm-4 col-lg-4 col-md-4']").Count();
    // adjusts count for the 0 start in array
    int correctCount =+ count;

    // define array and initiliaze all objects within object array
    Products[] product = new Products[correctCount];
    for (int y = 0; y < correctCount; y++)
    {
        product[y] = new Products();
    }
```



```
rojectService > WebService1.asmx.cs > { } ProjectService > ProjectService.WebService1 > ScrapeTablets()
    }
    // define starting tag
    HtmlNode node = doc.DocumentNode.SelectNodes("//div[@class='col-sm-4 col-lg-4 col-md-4']").First();

    // gets price
    HtmlNode[] pNodes = doc.DocumentNode.SelectNodes(".//h4[@class='pull-right price']").ToArray();
    //// gets title
    HtmlNode[] bNodes = doc.DocumentNode.SelectNodes(".//a[@class='title']").ToArray();
    // gets description
    HtmlNode[] dNodes = doc.DocumentNode.SelectNodes(".//p[@class='description']").ToArray();

    // sets values in object
    for (int b = 0; b < correctCount; b++)
    {
        product[b].ProductName = bNodes[b].Attributes["title"].Value;
        product[b].Price = pNodes[b].InnerHtml;
        product[b].Description = dNodes[b].InnerHtml;
    }

    // SQLite Database Queries
    // drop table to prevent duplicate data when testing DROP TABLE Phones;
    // create table if not exists

    string createTableQuery = @"DROP TABLE Phones;CREATE TABLE IF NOT EXISTS [Phones] (
                              [ProductName] VARCHAR(200) NULL,
                              [Description] VARCHAR(200) NULL,
                              [Price] VARCHAR(200) NULL);";

    sqlCmd.CommandText = createTableQuery;
    conn.Open();
    sqlCmd.ExecuteNonQuery();
    sqlCmd = new SQLiteCommand(conn);
    SQLiteDataAdapter dataAdapter;

    int x = 0;
    while (x < correctCount)
    {
        dataAdapter = new SQLiteDataAdapter(sqlCmd);
        sqlCmd.CommandText = "Select * from Phones where ProductName='" + product[x].ProductName + "'";
        dataAdapter.Fill(dtPhones);
        sqlCmd.CommandText = "Insert into Phones(ProductName, Description, Price) values " +
              "('" + product[x].ProductName + "','" + product[x].Description + "','" + product[x].Price + "')";
        sqlCmd.ExecuteNonQuery();
        x++;
    }
    string result = JsonConvert.SerializeObject(dtPhones);
    return result;
}
```

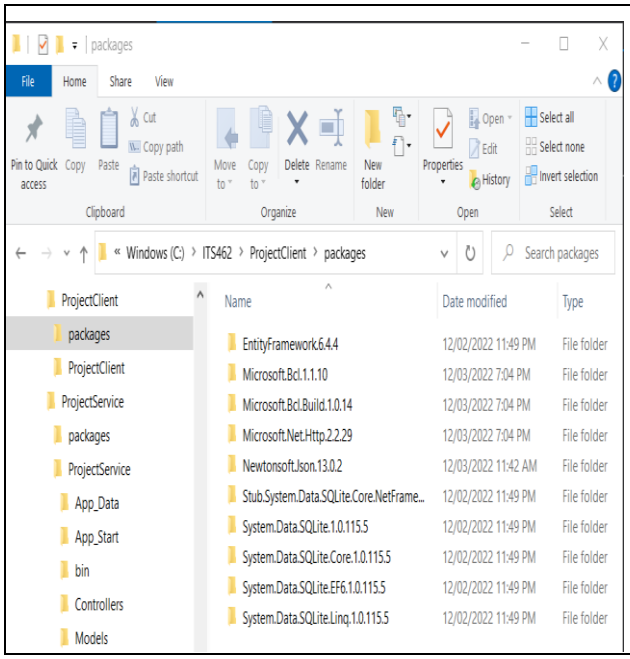
```
> ProjectService > WebService1.asmx.cs > { } ProjectService > ProjectService.WebService1 > Laptops
    [WebMethod]
    0 references
    public string AllProductsQuery()
    {
        conn.Close();
        string allProductsQuery = @"select t.*
                                    from Laptops t
                                    union all
                                    select t.*
                                    from Phones t
                                    union all
                                    select t.*
                                    from Tablets t
                                    order by price;";

        sqlCmd.CommandText = allProductsQuery;
        conn.Open();
        //sqlCmd.ExecuteNonQuery();
        sqlCmd = new SQLiteCommand(conn);
        SQLiteDataAdapter dataAdapter;
        dataAdapter = new SQLiteDataAdapter(sqlCmd);
        int correctCount = 1;
        int x = 0;
        while (x < correctCount)
        {
            dataAdapter = new SQLiteDataAdapter(sqlCmd);
            sqlCmd.CommandText = allProductsQuery;
            dataAdapter.Fill(dtAllProductsQuery);
            sqlCmd.ExecuteNonQuery();
            x++;
        }

        string result = JsonConvert.SerializeObject(dtAllProductsQuery);
        return result;
    }

    [WebMethod]
    0 references
    public string LaptopsQuery()
    {
        conn.Close();
        string laptopsQuery = @"select * from Laptops order by price;";

        sqlCmd.CommandText = laptopsQuery;
        conn.Open();
        //sqlCmd.ExecuteNonQuery();
        sqlCmd = new SQLiteCommand(conn);
        SQLiteDataAdapter dataAdapter;
        dataAdapter = new SQLiteDataAdapter(sqlCmd);
        int correctCount = 1;
        int x = 0;
        while (x < correctCount)
        {
            dataAdapter = new SQLiteDataAdapter(sqlCmd);
            sqlCmd.CommandText = laptopsQuery;
            dataAdapter.Fill(dtLaptosQuery);
            sqlCmd.ExecuteNonQuery();
            x++;
        }
        string result = JsonConvert.SerializeObject(dtLaptosQuery);
        return result;
    }
```

```
[WebMethod]
0 references
public string PhonesQuery()
{
    conn.Close();
    string phonesQuery = @"select * from Phones order by price;";

    sqlCmd.CommandText = phonesQuery;
    conn.Open();
    //sqlCmd.ExecuteNonQuery();
    sqlCmd = new SQLiteCommand(conn);
    SQLiteDataAdapter dataAdapter;
    dataAdapter = new SQLiteDataAdapter(sqlCmd);
    int correctCount = 1;
    int x = 0;
    while (x < correctCount)
    {
        dataAdapter = new SQLiteDataAdapter(sqlCmd);
        sqlCmd.CommandText = phonesQuery;
        dataAdapter.Fill(dtPhonesQuery);
        sqlCmd.ExecuteNonQuery();
        x++;
    }
    string result = JsonConvert.SerializeObject(dtPhonesQuery);
    return result;
}

[WebMethod]
0 references
public string TabletsQuery()
{
    conn.Close();
    string tabletsQuery = @"select * from Tablets order by price;";

    sqlCmd.CommandText = tabletsQuery;
    conn.Open();
    //sqlCmd.ExecuteNonQuery();
    sqlCmd = new SQLiteCommand(conn);
    SQLiteDataAdapter dataAdapter;
    dataAdapter = new SQLiteDataAdapter(sqlCmd);
    int correctCount = 1;
    int x = 0;
    while (x < correctCount)
    {
        dataAdapter = new SQLiteDataAdapter(sqlCmd);
        sqlCmd.CommandText = tabletsQuery;
        dataAdapter.Fill(dtTabletsQuery);
        sqlCmd.ExecuteNonQuery();
        x++;
    }
    string result = JsonConvert.SerializeObject(dtTabletsQuery);
    return result;
}
```

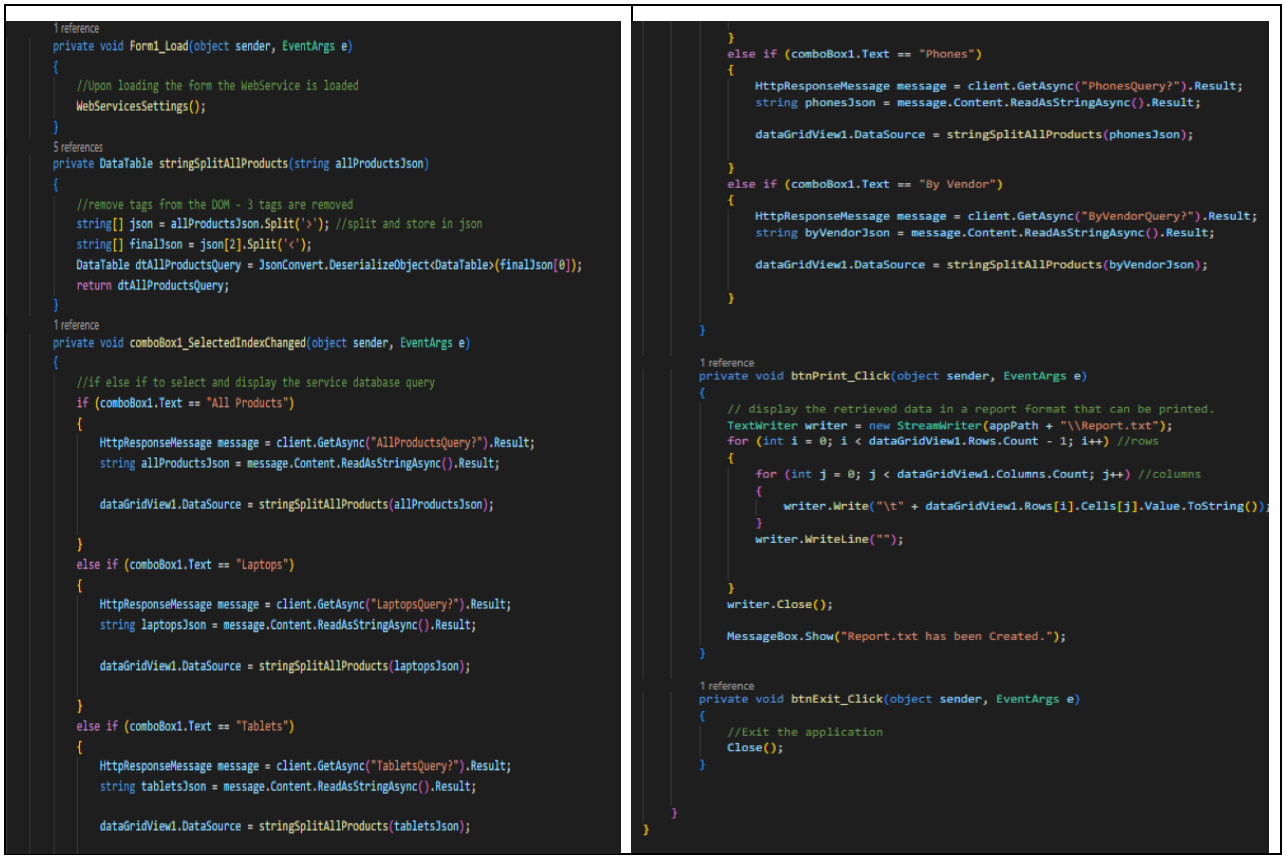
```
        [WebMethod]
        0 references
        public string ByVendorQuery()
        {
            conn.Close();
            string byVendorsQuery = @"select t.Description, t.Price
                                      from Laptops t
                                      union all
                                      select t.Description, t.Price
                                      from Phones t
                                      union all
                                      select t.Description ,t.Price
                                      from Tablets t
                                      order by Price;";

            sqlCmd.CommandText = byVendorsQuery;
            conn.Open();
            //sqlCmd.ExecuteNonQuery();
            sqlCmd = new SQLiteCommand(conn);
            SQLiteDataAdapter dataAdapter;
            dataAdapter = new SQLiteDataAdapter(sqlCmd);
            int correctCount = 1;
            int x = 0;
            while (x < correctCount)
            {
                dataAdapter = new SQLiteDataAdapter(sqlCmd);
                sqlCmd.CommandText = byVendorsQuery;
                dataAdapter.Fill(dtByVendorQuery);
                sqlCmd.ExecuteNonQuery();
                x++;
            }
            string result = JsonConvert.SerializeObject(dtByVendorQuery);
            return result;
        }
    }
}
```

Client

- Packages Installed
- Form1.cs




```
ectClient > ProjectClient > Form1.cs > ...
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
//added using statements
using Newtonsoft.Json;
using System.Net.Http;
using System.IO;

namespace ProjectClient
{
    1 reference
    public partial class Form1 : Form
    {
        //Declare  an application path.
        1 reference
        static string appPath = @"C:\ITS462\ProjectClient\ProjectClient";

        // sets the localhost webservice
        0 references
        localhost.WebService1 proxy = new localhost.WebService1();

        // create object for webservicesettings method
        6 references
        HttpClient client = new HttpClient();

        1 reference
        public Form1()
        {
            InitializeComponent();
        }
        1 reference
        private void WebServicesSettings()
        {
            //service is load through the Uri
            client.BaseAddress = new Uri("http://localhost:32920/WebService1.asmx/");
        }
        1 reference
```



```
1 reference
private void Form1_Load(object sender, EventArgs e)
{
    //Upon loading the form the WebService is loaded
    WebServicesSettings();
}
5 references
private DataTable stringSplitAllProducts(string allProductsJson)
{
    //remove tags from the DOM - 3 tags are removed
    string[] json = allProductsJson.Split('>'); //split and store in json
    string[] finalJson = json[2].Split('<');
    DataTable dtAllProductsQuery = JsonConvert.DeserializeObject<DataTable>(finalJson[0]);
    return dtAllProductsQuery;
}
1 reference
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    //if else if to select and display the service database query
    if (comboBox1.Text == "All Products")
    {
        HttpResponseMessage message = client.GetAsync("AllProductsQuery?").Result;
        string allProductsJson = message.Content.ReadAsStringAsync().Result;

        dataGridView1.DataSource = stringSplitAllProducts(allProductsJson);

    }
    else if (comboBox1.Text == "Laptops")
    {
        HttpResponseMessage message = client.GetAsync("LaptopsQuery?").Result;
        string laptopsJson = message.Content.ReadAsStringAsync().Result;

        dataGridView1.DataSource = stringSplitAllProducts(laptopsJson);

    }
    else if (comboBox1.Text == "Tablets")
    {
        HttpResponseMessage message = client.GetAsync("TabletsQuery?").Result;
        string tabletsJson = message.Content.ReadAsStringAsync().Result;

        dataGridView1.DataSource = stringSplitAllProducts(tabletsJson);
    }
    else if (comboBox1.Text == "Phones")
    {
        HttpResponseMessage message = client.GetAsync("PhonesQuery?").Result;
        string phonesJson = message.Content.ReadAsStringAsync().Result;

        dataGridView1.DataSource = stringSplitAllProducts(phonesJson);

    }
    else if (comboBox1.Text == "By Vendor")
    {
        HttpResponseMessage message = client.GetAsync("ByVendorQuery?").Result;
        string byVendorJson = message.Content.ReadAsStringAsync().Result;

        dataGridView1.DataSource = stringSplitAllProducts(byVendorJson);

    }
}

1 reference
private void btnPrint_Click(object sender, EventArgs e)
{
    // display the retrieved data in a report format that can be printed.
    TextWriter writer = new StreamWriter(appPath + "\\Report.txt");
    for (int i = 0; i < dataGridView1.Rows.Count - 1; i++) //rows
    {
        for (int j = 0; j < dataGridView1.Columns.Count; j++) //columns
        {
            writer.Write("\t" + dataGridView1.Rows[i].Cells[j].Value.ToString());
        }
        writer.WriteLine("");


    }
    writer.Close();

    MessageBox.Show("Report.txt has been Created.");
}

1 reference
private void btnExit_Click(object sender, EventArgs e)
{
    //Exit the application
    Close();
}


}
}
```