Итак, привет, кандидат в внедренцы Диадока. Чтобы попасть к нам на работу, помимо всего прочего, нужно выполнить тестовое задание. Его суть заключается в том, чтобы понять, как ты будешь справляться с нашими основными задачами.

Все, с чем мы работаем, можно разбить на три большие категории:

1. Разработка печатных форм (она ведется в инструменте FastReport). Мы не требуем знания принципов работы этого инструмента. Тем не менее, поскольку в нем используется синтаксис C#, базовые знания по C# потребуются.
2. Преобразования одних XML файлов в другие XML файлы. Для этого используется XSL-преобразования (версия 1.0).
3. Работа с публичным API. Нужно будет хоть немного понимать, как работает HTTP.

**Задания по C#**

Есть список данных (<**List**>), заполненный датами.

Например:

{

03.09.2019

11.06.1990

03.09.2019

31.12.2018

09.05.2017

}

1. Нужно узнать количество элементов в этом списке.
2. Мне нужно получить другой список, который будет содержать только уникальные даты в порядке возрастания. Причем, я хочу, чтобы дата была в формате «03 сентября 2019 года».
3. Возьмем тот же список с датами из п.1. Мне хочется узнать, сколько Ticks в каждой дате (считаем, что 03.09.2019 = 03.09.2019 00:00:00.000).

В всех случаях я хочу увидеть рабочий метод, написанный на C#, который на вход принимает список и на выходе выдает список (или целое число, отражающее количество элементов в списке).

int GetCountElements(List<DateTime> dateTimes)

{

try

{

int countDateTimes = dateTimes.Count;

return dateTimes.Count;

}

catch

{

Console.WriteLine("Список был null");

return 0;

}

}

List<string> GetListUniqueDateTime(List<DateTime> dateTimes)

{

try

{

List<string> listUniqueDateTime = dateTimes.

Distinct().

OrderBy(dateTime => dateTime.Date).ToList().

Select(dateTime => dateTime.Day < 10 ? "0" + dateTime.ToLongDateString() : dateTime.ToLongDateString()).ToList();

return listUniqueDateTime;

}

catch

{

Console.WriteLine("Список былл null");

return new List<string>();

}

}

List<DateTime> GetTicksDateTime(List<DateTime> dateTimes)

{

try

{

dateTimes.ForEach(dateTime => Console.WriteLine("для " + dateTime.ToShortDateString() + " кол-во тиков: " + dateTime.Ticks));

return dateTimes;

}

catch

{

Console.WriteLine("Список былл null");

return new List<DateTime>();

}

}

List<DateTime> dateTimes = new List<DateTime>()

{

DateTime.Now.Date.AddDays(-10).AddMonths(2).AddYears(1),

DateTime.Now.Date,

DateTime.Now.Date.AddMonths(-2).AddYears(-1)

};

int countDateTimes = GetCountElements(dateTimes);

Console.WriteLine("Кол-во элементов в списке: " + countDateTimes);

List<string> listUniqueDateTime = GetListUniqueDateTime(dateTimes);

listUniqueDateTime.ForEach(date => Console.WriteLine(date));

GetTicksDateTime(dateTimes);

**Задания по XSLT**

При работе с XSLT крайне важно понимать, как устроены XML-документы. Для этого используется синтаксис XPath-выражений, который позволяет обращаться к узлам и атрибутам XML-документа, а также проверять их значения.

Есть у меня такой XML-файл:

**<Table>**

**<Item>**

**<Type>A</Type>**

**<Product>KETTLE</Product>**

**<Quantity>31</Quantity>**

**<RepairSum>7153.07</RepairSum>**

**<DetailSum>12659.52</DetailSum>**

**<Subtotal>19812.59</Subtotal>**

**</Item>**

**<Item>**

**<Type>B</Type>**

**<Product>ESPRESSO FULL AUTOM</Product>**

**<Quantity>15</Quantity>**

**<RepairSum>19830.56</RepairSum>**

**<DetailSum>1021.12</DetailSum>**

**<Subtotal>20851.68</Subtotal>**

**</Item>**

**<Item>**

**<Type>C</Type>**

**<Product>Мясорубки</Product>**

**<Quantity>8</Quantity>**

**<RepairSum>1222.72</RepairSum>**

**<DetailSum>0</DetailSum>**

**<Subtotal>1222.72</Subtotal>**

**</Item>**

**<TotalQuantity>635</TotalQuantity>**

**<TotalRepairSum>202356.88</TotalRepairSum>**

**<TotalDetailSum>96088.55</TotalDetailSum>**

**<Total>298445.43</Total>**

**</Table>**

Я хочу получить из этого файла другой файл:

**<Result>**

**<DocumentInfo NumberOfItems="{число элементов Item в исходном файле}">**

**<Total>{содержимое узла Total в исходном файле}</Total>**

**<RealTotal>{сумма узлов Subtotal в каждом из узлов Item исходного файла}</RealTotal>**

**<IsEqual>{здесь я хочу увидеть true или false в зависимости от того, совпадает ли содержимое Total и RealTotal в преобразованном документе}</IsEqual>**

**{Ниже я хочу получить все различные значения узлов Type исходного файла, т.е. подразумевается, что преобразование будет работать, если я поменяю значение одного из этих узлов или добавлю новые}**

**<DifferentTypes>**

**<Type>A</Type>**

**<Type>B</Type>**

**<Type>C</Type>**

**</DifferentTypes>**

**</DocumentInfo>**

**</Result>**

Сложность заданий: красное – «Пешка», голубое – «Ладья», зеленое – «Ферзь».

**Ответ**

<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

<xsl:variable name="Total" select="string(Table/Total)"/>

<xsl:variable name="NumberOfItems" select="count(Table/Item)"/>

<xsl:variable name="RealTotal" select="string(sum(//Item/Subtotal))"/>

<Result>

<DocumentInfo NumberOfItems="{$NumberOfItems}">

<p><Total><xsl:value-of select="$Total"/></Total></p>

<p><RealTotal><xsl:value-of select="$RealTotal"/></RealTotal></p>

<IsEqual><xsl:value-of select="$RealTotal = $Total"/></IsEqual>

<DifferentTypes>

<xsl:for-each select="Table/Item[not(Type = following-sibling::Item/Type)]">

<p><Type><xsl:value-of select="Type"/></Type></p>

</xsl:for-each>

</DifferentTypes>

</DocumentInfo>

</Result>

</xsl:template>

</xsl:stylesheet>

**Задания с HTTP-запросами**

Иногда нам нужно консультировать клиентов по вопросам работы с API. Фактически нужно уметь самому отправлять запросы в API и понимать, что возвращается в ответ.

Не важно, каким способом, но мне нужно получить информацию, которая возвращается при простом входе из браузера на страницу <https://lenta.ru>:

1. Какой тип запроса при этом уходит на сервер? Чем этот отличается от других?

Метод запроса: GET

Запрос на извлечение данных. Страница с параметрами может быть кэширована. Отправляет данные как часть URL. Отправка файлов не поддерживается в отличие от POST. Длина запроса ограничена. Получает тело ответа в отличие от head. Не удаляет, не изменяет и не добавляет ресурсы в отличие от POST, DELETE, PUT, PATCH,

1. Какой код ответа возвращает сервер? Какие коды ты еще знаешь?

Код статуса: 200 OK

Код статуса: 304 OK при обновлении

**1xx: Информационные сообщения**

Сервер может отправить запрос вида: Expect: 100-continue - клиент ещё отправляет оставшуюся часть запроса.

**2xx: Сообщения об успехе**

Если клиент получил код из серии 2xx, то запрос ушёл успешно. Самый распространённый вариант - это 200 OK. При GET запросе, сервер отправляет ответ в теле сообщения. Также существуют и другие возможные ответы:

* **202 Accepted**: запрос принят, но может не содержать ресурс в ответе.
* **204 No Content**: в теле ответа нет сообщения.
* **205 Reset Content**: указание серверу о сбросе представления документа.
* **206 Partial Content**: ответ содержит только часть контента. В дополнительных заголовках определяется общая длина контента и другая информация.

**3xx: Перенаправление**

Своеобразное сообщение клиенту о необходимости совершить ещё одно действие.

* **301 Moved Permanently**: ресурс теперь можно найти по другому URL адресу.
* **303 See Other**: ресурс временно можно найти по другому URL адресу. Заголовок Location содержит временный URL.
* **304 Not Modified**: сервер определяет, что ресурс не был изменён и клиенту нужно задействовать закэшированную версию ответа.

**4xx: Клиентские ошибки**

Данный класс сообщений используется сервером, если он решил, что запрос был отправлен с ошибкой.

* **400 Bad Request**: вопрос был сформирован неверно.
* **401 Unauthorized**: для совершения запроса нужна аутентификация. Информация передаётся через заголовок Authorization.
* **403 Forbidden**: сервер не открыл доступ к ресурсу.
* **404 Not Found**. Это означает, что ресурс не найден на сервере
* **405 Method Not Allowed**: неверный HTTP метод был задействован для того, чтобы получить доступ к ресурсу.
* **409 Conflict**: сервер не может до конца обработать запрос, т.к. пытается изменить более новую версию ресурса.

**5xx: Ошибки сервера**

Ряд кодов, которые используются для определения ошибки сервера при обработке запроса.

* **500 Internal Server Error**
* **501 Not Implemented**: сервер не поддерживает запрашиваемую функциональность.
* **503 Service Unavailable**: это может случиться, если на сервере произошла ошибка или он перегружен.

1. Что такое cookie? Устанавливает ли сервер какие-либо cookie при входе на лента.ру?

Cookie – это небольшие строки данных, которые отправляются сервером и хранятся в браузере.

Технология хранения cookie разработана как для аутентификации пользователей на сайте без повторной идентификации, но также для хранения информации о сайте (регион, язык, корзины, и т.д.)

Да, при входе на лента.ру устанавливаются cookie. Проверим через команду document.cookie

'is\_mobile=0;

lid=vAsAAH6yTGP7ExsKAcqzIwB=;

user-id\_1.0.5\_lr\_lruid=pQ8AAEmyTGNpAX3GATvPgAA%3D;

adtech\_uid=894d28b9-d5bf-48e4-99b1-0e1a48b3ec8d%3Alenta.ru;

\_ym\_uid=1665970762852547358;

\_ym\_d=1665970762;

\_ym\_isad=1;

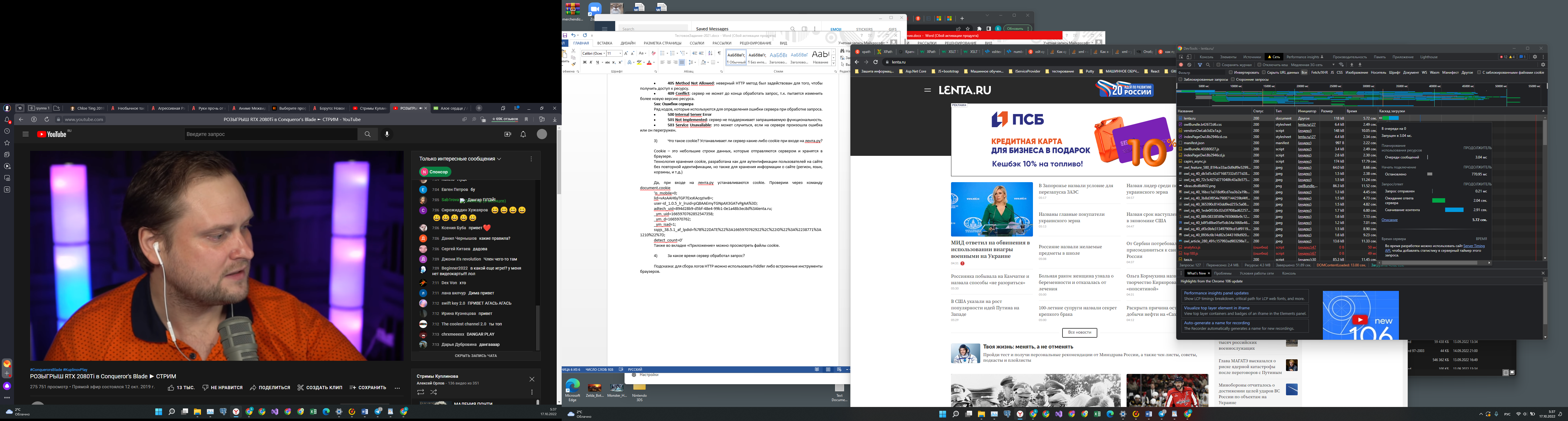
sspjs\_38.5.1\_af\_lpdid=%7B%22DATE%22%3A1665970762922%2C%22ID%22%3A%2238771%3A1210%22%7D;

detect\_count=0'

Также во вкладке «Приложение» можно просмотреть файлы cookie.

1. За какое время сервер обработал запрос?

2.04 секунды сервер обрабатывал запрос.



Подсказка: для сбора логов HTTP можно использовать Fiddler либо встроенные инструменты браузеров.