**Создание WEB приложения без хранилища данных и без реализации сложных вычислений.**

Здравствуйте дорогие читатели.

Хочу вам расказать историю одного веб приложения и о решениях которые мы довольно таки неплохо использовали.

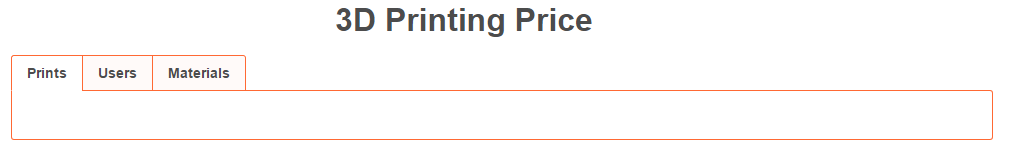
Есть у нашего рабочего коллектива одно хобби, мы любим 3Д печать и не просто любим, а активно печатаем, все и вся как говорится. Каждый у нас в комнате хоть раз, да напечатал чего нибудь, от простых игрушек на стол, до деталек которые даже в Китае не продаются. Но так-же, как простому принтеру нужны чернила, 3Д принтеру нужен пластик, изначально мы печатали пластиком, который шел в комплекте с принтером, потом скинулись(в равных долях) и купили кучу разноцветных катушек. И все бы ничего, но и эти катушки начали подходить к концу, вот тут мы и заметили что все печатают в разных объёмах, конечно же у нас сразу родилась идея создать приложение для учета расхода печатного материала, кто сколько израсходовал, тот столько и скидывается. Выбор формы не заставил себя ждать, решено было что это будет вэб приложение, так удобно обеспечить всем доступ к просмотру и редактированию информации о сеансах печати.

Начнем формирование требований к нашему приложению.

В нем мы должны редактировать информацию о сеансах печати, добавлять и изменять данные если вдруг ошиблись, редактировать информацию о пользователях, редактировать информацию о используемых материалах. Само приложение должно считать стоимость сеансов и калькулировать их по каждому пользователю, ну и хотелось бы какой нибудь график для всего этого.

Таким образом мы получаем интерфейс из трёх вкладок («Prints», «Users» и «Materials») с постоянно отображающимся чартом.

Вкладки реализуем с помощью стандартного сценария использования табов.



Теперь подумаем о расчете расходов.

При первой покупке мы заметили что цена пластика продавцами формируется за киллограм, поэтому самая первая идея была - взвешивать каждый отпечатаный объект и потом просто суммировать вес. Но для такого подхода нам нужен еще один прибор - весы, мы оставили эту идею на случай если не найдём другого способа и решили немножко изменить подход. Нами тут же было замечено что в ПО для печати отображается длина используемого прутка, а продавцами пластика в характеристиках товара предоставляется вся информация чтобы на основании длины получить массу, таким образом необходимость в весах отпадает.

Все характеристики пластика мы будем держать на вкладке «Materials», а именно: плотность, диаметр прутка и цену за киллограмм.

На вкладке «Prints»: Имя пользователя, имя пластика и израсходованная длинна, а стоимость сеанса будет уже вычисляться.

На вкладке «Users», имя пользователя и вычисляемая общая сумма расходов для каждого пользователя и именно по этой вкладке будет строится наш чарт.

Таким образом наше приложение должно содержать логику по расчету стоимости сеанса, по суммированию сеансов и по постройке чарта на базе наших данных, так-же эти данные надо где-то хранить.

Сразу же вспомнился наш же продукт SpreadsheetcloudAPI [ссылка на сервис].

Ведь он позволяет, организовать логику на уровне spreadsheet и визуализацию в виде чарта встроенными функциями, мало того при изменении логики в будущем нам будет достаточно отредактировать шаблон и не придется переписывать приложение.

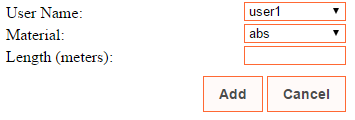
В самом же приложении, нам будет достаточно отправлять и получать данные с нашего сервиса, не реализуя никакой логики и не думая о хранении данных, все вычисляемы значения будут автоматически считаться по формулам в spreadsheet файле, данные будут храниться в нём же и нам остаётся только их показать, так же мы получаем автоматически обновляемый чарт.

В суть создания spreadsheet файла для расчётов и хранения данных, в этом посте, я вдаваться не буду, а напишу отдельный пост про это, кому интересно [вот ссылка].

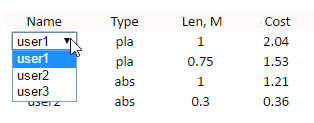
У SpreadsheetCloudAPI, есть очень хорошие функции экспорта в HTML и экспорта чартов как картинок, что очень нам на руку, так как визуализировать наши данные мы можем просто экспортируя нужные участки документа в наше приложение, на этом вопрос визуализации будет закрыт.

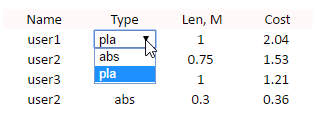
При экспорте данные будут представлять из себя таблицу и для редактирования этих данных нам достаточно будет обрабатывать клик на ячейке таблицы(td) и вешать на неё нужный нам эдитор. Для редактирования значений мы используем «Input», для выбора значений из уже имеющихся «Select».

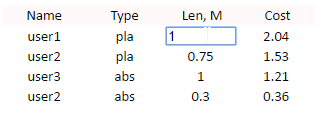
Например при создании нового сеанса печати нам понадобится выбрать имена пользователя и материала и ввести значение длины.



Для редактирования уже существующего сеанса печати мы будем использовать те-же элементы что и при создании.

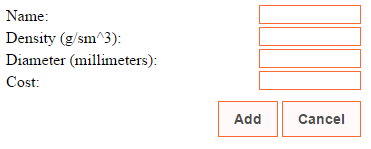


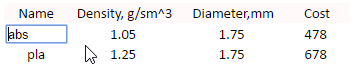


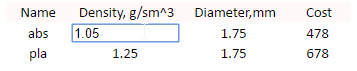


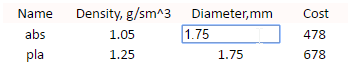
Колонка «Cost» на вкладке «Prints» редактироваться не будет, она будет вычислятся непосредственно на сервисе и вставлятся при экспорте.

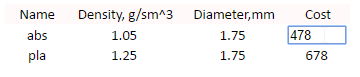
На вкладке «Materials» мы будем целиком задавать значения, то есть использовать только «Input», как при создании материала, так и при изменении его параметров.



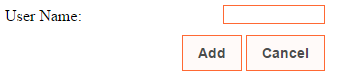


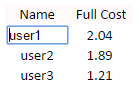






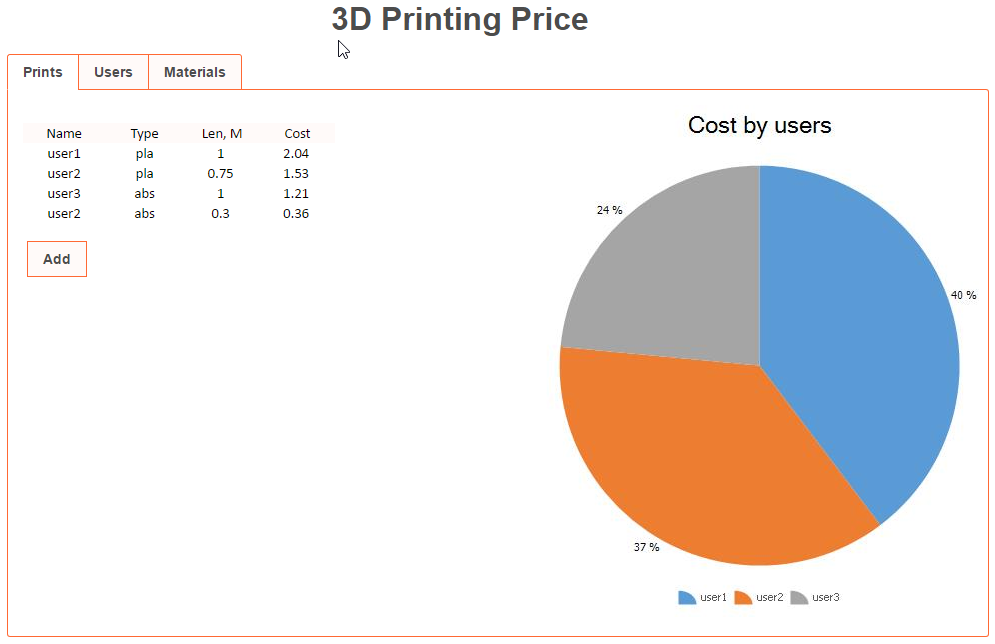
На вкладке «Users» единственный параметр который мы будем создавать и изменять это имя пользователя, используем для этого уже знакомые нам элементы «Input»





Собственно на этом всё написание нашего приложения и закончится, единственное что нам надо будет учесть это изменение имени пользователя и имени материала проводить не только на соответсвующих вкладках, но и на вкладке «Prints».

Для клиентской части приложения мы использовали JS + jQuery.



Основной код клиентской части это обработка кликов на нужных ячейках таблиц, формирование Ajax запросов для серверной части, и замена innerHTML соответсвующих элементов страницы данными экспорта.

Для серверной части нашего приложения мы использовали PHP. На серверную часть ложится функционал по обмену со SpreadsheetCloudAPI сервисом и логика изменения и добавления значений в spreadsheet файле, о чем подробно мы расскажем в посте про формирование шаблона [ссылка на пост].

Благодаря тому что все формулы присутсвуют в шаблоне, нам не нужно реализовывать логику расчётов, а только передачу и получение данных.

Для начала определим данные которые нам понадобятся для общения с сервисом и соберем их все в один класс с константами:

class PrivateConst {

const Base\_Url = 'http://spreadsheetcloudapi.azurewebsites.net/api/spreadsheet';

const API\_KEY = 'здесь должен находиться API Key полученный вами присоздании application на сервисе';

const File\_Name = '3D.xlsx';

}

Для передачи комманд сервису мы будем использовать CURL. Нам будут нужны для наших целей два ти па запросов: «PUT» и «GET», ниже приведена их реализация на PHP.

function put( $params, $url ) {

if (empty($params))

return null;

$json = json\_encode($params);

$header = generate\_header();

$request = curl\_init();

curl\_setopt\_array($request, [

CURLOPT\_URL => PrivateConst::Base\_Url.$url,

CURLOPT\_RETURNTRANSFER => true,

CURLOPT\_HTTPHEADER => $header,

CURLOPT\_CUSTOMREQUEST => 'PUT',

CURLOPT\_FOLLOWLOCATION => true,

CURLOPT\_POSTFIELDS => $json

]);

$response = curl\_exec($request);

$info = curl\_getinfo($request);

curl\_close($request);

return array('status' => $info['http\_code'], 'data' => $response);

}

function get( $params, $url ) {

if ( empty( $params ) )

return null;

$header = generate\_header();

$request = curl\_init();

curl\_setopt\_array( $request, [

CURLOPT\_URL => PrivateConst::Base\_Url.$url.'?'.http\_build\_query( $params ),

CURLOPT\_RETURNTRANSFER => true,

CURLOPT\_HTTPHEADER => $header,

CURLOPT\_FOLLOWLOCATION => true,

CURLOPT\_AUTOREFERER => true

]);

try {

$response = curl\_exec( $request );

$info = curl\_getinfo( $request );

curl\_close( $request );

} catch ( Exception $e ) {

return array( 'status' => 434, 'data' => $e );

}

return array( 'status' => $info['http\_code'], 'data' => $response );

}

function generate\_header() {

$API\_key = PrivateConst::API\_KEY;

$header = [

'Content-type: application/json',

'Authorization: amx '.$API\_key,

];

return $header;

}

Для отдачи конкректных команд мы рассмотрим пример установки значения в ячейку:

function setCellValue($sheetName, $row, $column, $filename, $value, $id){

$params = array(

'id' => $id,

'filename' => $filename,

'sheetname' => $sheetName,

'rowindex' => $row,

'columnindex' => $column,

'value' => $value,

);

return put($params, "/setcellvalue");

}

Среди параметров этого метода присутсвует $id, это ID сесси загруженного файла его мы получаем когда открываем файл на сервисе командой LoadDocument

function loadDocument($filename){

$params = array( 'filename' => $filename, );

$request = get($params, '/loaddocument')['data'];

return json\_decode($request)->Id;

}

При наличии ID и не закрытой сессии сервис не тратит каждый раз ресурсы на открытие файла и можно поочередно выполнять команды и запросы экономя время и реквесты, например перебор строк для поиска первой свободной и не заполненой ячейки.

Также приведем пример получения экспортированных данных с помощью команд «getpictures» и «exporttohtml».

function getSessionHtml($id, $sheetName, $rowLimit, $columnLimit){

$params = array(

'id' => $id,

'sheetname' => $sheetName,

);

if($rowLimit > -1){

$params['endrowindex'] = $rowLimit;

}

if($columnLimit > -1){

$params['endcolumnindex'] = $columnLimit;

}

$request = get($params, '/exporttohtml');

return $request;

}

function getPictureBytes($id) {

$params = array(

'id' => $id,

'sheetname' => SheetNames::Charts,

'picturetype' => 'Chart',

'startrowindex' => 0,

'startcolumnindex' => 0,

'endrowindex' => 19,

'endcolumnindex' => 7,

'objectindex' => 0,

'scale' => 0.2

);

return get( $params, '/getpictures' );

}

Код всего приложения можно скачать по ссылке [ссылка на приложение].

Ну и пара основных моментов работы приложения:

При создании нового материала, мы вставляем вторую строку на лист «Materials» (первая строка это шапка таблицы), а потом заполняем её данными, мы указывали этот момент при проектировании для использования встроеных нотификаций.

Так-же стоит помнить что при переименовании юзера и материала, надо менять имя и юзера, и материала не только на их листах, но и в листе «Prints».

При экспорте в html листа «Prints» мы ограничим экспортируемый рэндж колонкой «Cost»(D) оберегая пользователя от лицезрения промежуточных расчётов.

function getPrintsHtml(){

$filename = PrivateConst::File\_Name;

$sheetName = SheetNames::Prints;

$id = loadDocument($filename);

$firstEmptyRow = findFirstEmptyRow($id, $sheetName, 0);

$rowLimit = $firstEmptyRow - 1;

$result = getSessionHtml($id, $sheetName, $rowLimit, 3);

if($result['status'] == 200){

echo $result['data'];

}

}

При создании Ajax запросов с клиента на сервер мы завели параметр type в соответсвии с которым зовем нужную команду на сервисе, ниже приведен клас со списком этих команд

class Commands {

const CreatePrinting = 'createPrinting';

const CreateUser = 'createUser';

const CreateMaterial = 'createMaterial';

const GetPrintsHtml = 'getPrintsHtml';

const GetUsersHtml = 'getUsersHtml';

const GetMaterialsHtml = 'getMaterialsHtml';

const GetChart = 'getChart';

const ChangeUserName = 'changeUserName';

const ChangeMaterialName = 'changeMaterialName';

const ChangeMaterialPrice = 'changeMaterialPrice';

const ChangeMaterialDensity = 'changeMaterialDensity';

const ChangeMaterialDiameter = 'changeMaterialDiameter';

const ChangePrintingLength = 'changePrintingLength';

const ChangePrintingUser = 'changePrintingUser';

const ChangePrintingMaterial = 'changePrintingMaterial';

}

Таким образом не особо утруждаясь написанием сложной логики и работой с хранилищем данных, мы всего лишь используя SpreadsheetCloudAPI сервис, создали простое и удобное приложение которое даже не придеться переписывать, если мы захотим поменять логику, для этого будет достаточно отредактировать шаблон.

Само приложение и шаблон можно скачать по ссылке.

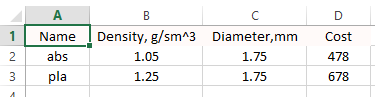
Спасибо за внимание.

**Создание spreadsheet шаблона.**

В предыдущем посте[ссылка на пост про приложение], я рассказывал про создание простого web приложения, без реализации в нем сложной логики и вычислений, благодаря возможности использования SpreadsheetCloudAPI сервиса. Для работы с этим сервисом нам нужен Spreadsheet файл в котором и будет реализована вся логика, создание этого файла мы с вами сейчас и рассмотрим.

В посте про приложение мы указавали о необходимости в трех вкладках и постоянно отображающемся чарте, для удобства работы с файлом мы заведем четыре листа: «Prints» «Users» «Materials» и «Charts».

Начнем с листа «Materials».



Где:

Name – название пластика, именно это название будет у нас стоять в столбце «Type» на вкладке «Prints»,

Density – плотность материала,

Diameter – диаметр прутка,

Cost – стоимость за киллограм материала.

Собственно на этом этот лист и закончим, так как на нем не будет производится никаких вычислений.

Использование данных с этого листа будет производится на листе «Prints».

Лист «Prints».



Где Name – имя пользователя,

Type – Тип используемого материала (пластик бывает разный),

Len, M – длина использованого материала в метрах(то что нам выдаёт ПО для печати),

Cost – стоимость печати,

Mcost – стоимость материала с листа «Materials», получать её мы будем с помощью функции VLOOKUP по имени материала, которое на этом листе отображается в столбце «Type» (B), а на листе «Materials» - «Name» (A), формула получения выглядит следующим образом: **=IF(A2="","",VLOOKUP(B2,Materials!$A$1:$D$3,4,0)).**

MDensity – плотность материала получаемая по аналогии «MCost» c листа «Materials», формула: **=IF(A2="","",VLOOKUP(B2,Materials!$A$1:$D$3,2,0))**.

Mdiameter – диматер материала, как и два предыдущих параметра получаемый с листа «Materials», формула: **=IF(A2="","",VLOOKUP(B2,Materials!$A$1:$D$3,3,0))**.

Msquare – площадь сечения материала, параметр вычисляемый по формуле из курса геометрии, «G2» - в данном случае ссылка на диаметр: **=IF(A2="","",G2\*G2\*PI()/4)**.

Mweight – вес потраченного материала, получаемый как произведение плотности на площадь, на длинну, итоговая формула: **=IF(A2="","",H2\*F2\*C2)**.

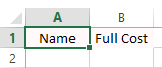
Ну и подредактируем параметр «Cost» а именно зная вес потраченного материала расчитаем его стоимость как произведение цены на вес, в формуле еще присутсвует деление на «1000», так как вес нащего материала получается в граммах, а стоимость материала указана продавцом в киллограммах, таким образом формула для столбца Cost будет выглядеть следующим образом: **=IF(A2="","",(I2\*E2)/1000)**.

Так как у нас не один сеанс печати растянем формулы например на 3000 строк.

Теперь на листе «Prints» при заполнении параметров «Type» и «Len, M» мы автоматически в нужной строчке будем получать стоимость потраченного материала.

Кстати при добавлении данных о новом материале, фрагменты формул будут автоматически менятся благодаря поддерживаемым нашим сервисам нотификациям, единственный ньюанс новый материал нужно добавлять вставкой строки в начало, это должно быть учтено при разработке приложения.

Лист «Users».



Где:

Name – имя пользователя,

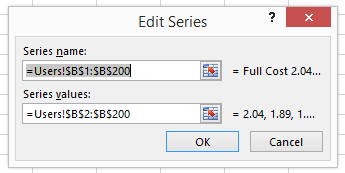
Full Cost – сумма потраченых средств этим пользователем,

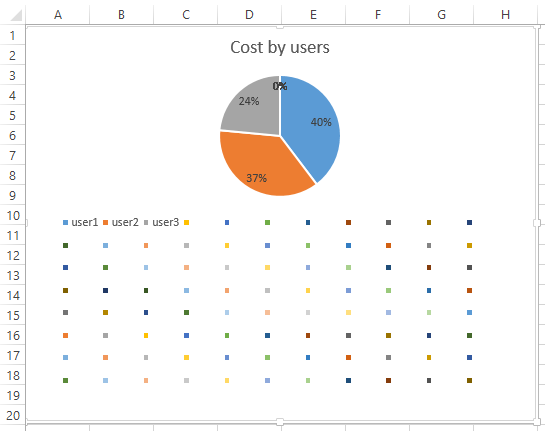
В столбце «Full Cost» создадим формулу в которой будем получать искомую сумму по имени пользователя указанного в столбце «A2», используем для этого функцию SUMIF, в итоге мы получим: **=IF(A2="","", SUMIF(Prints!$A$1:$D$3000,A2,Prints!$D$1:$D$3000))**.

Для поддержания количества пользователей больше одного растянем формулу на 200 строк.

Лист «Charts».

Здесь создадим чарт визуализирующий лист «Users».





Подведём итоги, что мы имеем в шаблоне:

Лист «Prints»: лист на котором учитываются сеансы печати и расчёты их стоимости,

Лист «Users»: список пользователей с рачетами данных по их полным тратам, на основе данного листа строится чарт визуализации,

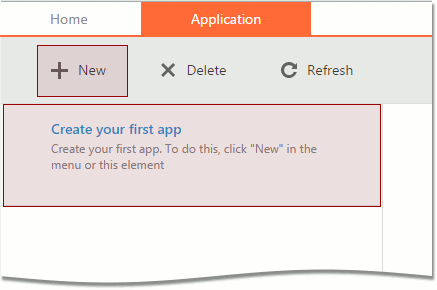
Лист «Materials»: данные материалов для печати, никаких расчетов на этом листе не производится,

Лист «Charts»: чарт визуализации листа «Users»,

Полученный шаблон можно скачать по ссылке [ссылка шаблона].

Далее для того чтобы использовать шаблон в нашем приложении нам нужно создать аккаунт на SpreadsheetCloudAPI и загрузить наш файл шаблона на сервис, далее небольшой урок по пользованию нашим сервисом.

1. Заидите на страницу админстрирования SpreadsheetCloudAPI (<http://spreadsheetadmin.azurewebsites.net/>) используя свой аккаунт или(если его нету) нажмите на «Sign Up» для создания нового.
2. Нажмите «Create your first app» если увас нет приложений или на пункт меню «New».

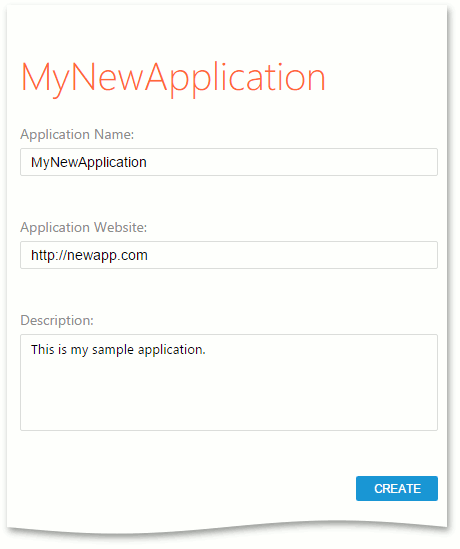


1. Заполниете требуемые значения в полях «Application Name», «Application Website» и «Application Description».

Обязательный параметр. «Application Name» указано имя, которое будет идентифицировать ваше приложение с помощью сервиса SpreadsheetCloudAPI.

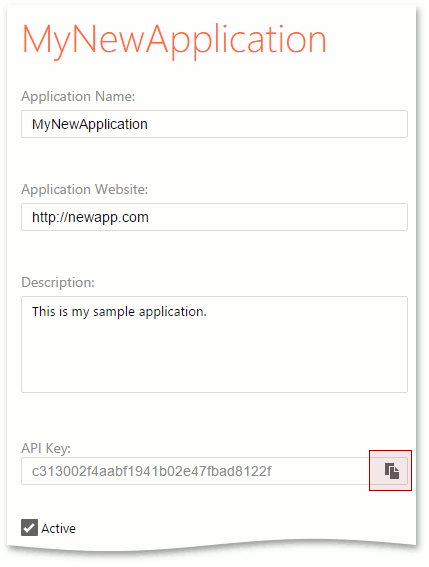
Опционально. «Application Website» указан веб-сайт вашего приложения. Это значение рассматривается как комментарий, чтобы вы могли быстрее идентифицировать свое приложение.

Опционально. «Application Description» указывается дополнительная информация (если есть) для соответствующего приложения.



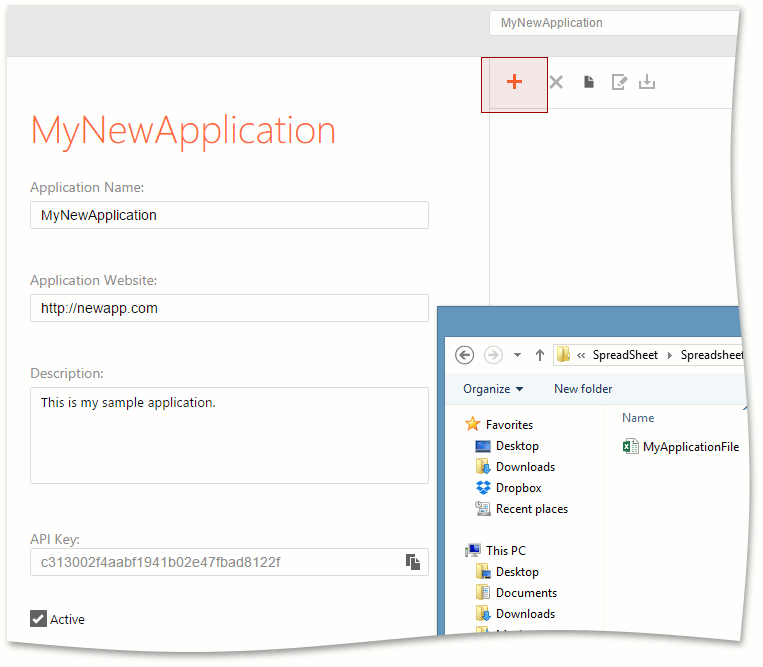
Нажмите «Create», чтобы создать приложение с указанными параметрами и создать API Key для вашего приложения.

1. После того как приложение будет создано API Key автоматически отобразится на странице приложения.



Для копирования API Key нажмите «Copy» справа.

1. Теперь остаётся только добавить файл нашего шаблона в приложение.



Для этого нажмите кнопку «+» либо воспользуйтесь функцией «Drag&Drop».

Это всё, только напомню что вам в дальнейшем нужен будет API Key.

Спасибо за внимание.

Здравствуйте дорогие гости нашего блога.

Представляем вашему вниманию новый продукт - SpreadsheetCloudAPI.

Из названия ясно, что этот продукт связан со Spreadsheet и облачными технологиями, так и есть. SpreadsheetCloudAPI это сервис который позволяет пользоваться Excel файлами расположенными в облачном хранилище используя их информационные мощности удаленно. Если конкректно, то вы используете эксель файл, не имея эксель и не задумываясь на реализацией логики для его работы. Всю логику реализовали для вас и за вас мы. Вам остается только воспользоваться нашим сервисом.

Теперь о тех, кому это нужно, раз вы это читаете, то возможно, это вы. Рассмотрим ряд сценариев, которые покажут, для чего нужен наш сервис.

1. Первый сценарий – вынесенная логика  
   Все ваши приложения изначально пишутся в расчёте на то, что на сервисе живет определённый шаблон с формулами, чартами и другим Spreadsheet контентом, ваше приложение передаёт данные в определённые ячейки взамен получая данные посчитанные из файла живущего на облаке. Пример использования этого сценария вы можете посмотреть в нашей демке <http://spreadsheetcloudapiexamples.azurewebsites.net/convert/0>  
   Вы всегда можете изменить логику вашего приложения всего лишь подменяя файл шаблона, либо редактируя его, причем наш сервис позволяет так же осуществлять и невизуальное редактирование файла, чтобы не быть голословным приведу пример из другой демки <http://spreadsheetcloudapiexamples.azurewebsites.net/demo/>  
   Самый яркий плюс этого сценария в том, что пользователю не надо обладать специальными навыками для того чтобы задать логику приложения, достаточно знаний Excel, а этими знаиями обладают не только специалисты в сфере IT. У большинства обычных пользователей PC знакомство с компьютером начинается именно с изучения основ Excel. [простой пример с кодом на C# SDK и ссылки для скачивания файлов шаблонов]
2. Второй сценарий – визуализация ваших данных,  
   <http://spreadsheetcloudapiexamples.azurewebsites.net/chartsdemo/>  
   Ваш шаблон содержит чарт, и передавая ваши данные в шаблон вы получаете их визуальное отображение в вашем приложении используя экспорт чарта в изображения.  
   Еще вы можете использовать html-экспорт чтобы отобразить, например на вашей странице, таблицу сформированную из ваших данных, либо таблицу расчетов, основанную на введенных вами формулах.  
   Так же для реализации этого сценария был создан WordPress плагин, для отображения данных в блогах созданных на базе CMS WordPress.  
   Посредством плагина в блогах WordPress просходит отображение картинок(чартов) и HTML фрагментов из excel файлов находящихся на сервисе. Также непосредственно из меню настройки вордпресс плагина, вы можете управлять файлами на вашем аккаунте в сервисе. [пример из блога на вордпрес, на медиуме и рэддите будет ссылка на вордпресс блог]
3. Сценарий третий – сбор.  
   Если вы дочитали до этого момента, то рискну предположить что вы уже посмотрели часть наших демо и заметили на многих кнопку «Download file», так вот вы можете сформировать файл на сервисе и заполнять его данными из ваших приложений, например сбор статистики, курсы торгов (данные могут быть любые) и всегда скачать эти данные в виде готового Spreadsheet файла.
4. Четвертый сценарий – конвертация.  
   Ну и так, как наш сервис может работать с несколькими файловыми форматами, вы всегда можете сконвертировать файл одного формата в любой другой, который нужен вам (из тех которые мы поддерживаем).

Естественно это не все возможные сценарии применения нашего продукта, про многие мы можем не догадываться, многие мы можем реализовать, в этом наши плюсы, мы бесплатны и будем развиваться изходя из потребностей наших пользователей, стратегия нашей монетизации такова что мы станем именно тем продуктом который вам нужен.

И если у вас возникло желание использовать наш сервис, либо вы задумались о сценарии его применения, возможно наш продукт создан для вас, если в нем чего-то не хватает то мы улучшим продукт для вас.

Что у нас есть для вас:

Сам сервис в админке также ссылка на демо <http://spreadsheetadmin.azurewebsites.net/>,

C# SDK лежит на NUGET <https://www.nuget.org/packages/SpreadsheetSDK/>,

WordPress plugin <https://wordpress.org/plugins/spreadsheetcloudapi/>.