**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.

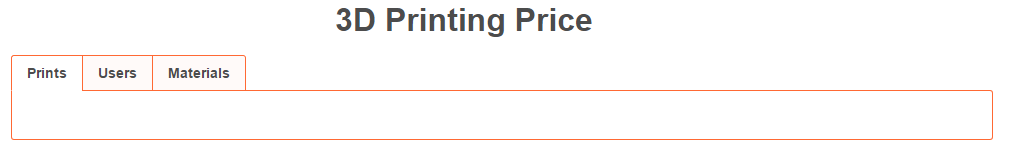
Есть у нашего рабочего коллектива одно хобби, мы любим 3Д печать и не просто любим, а активно печатаем, все и вся как говорится. Каждый у нас в комнате хоть раз, да напечатал чего нибудь, от простых игрушек на стол, до деталек которые даже в Китае не продаются. Но так-же, как простому принтеру нужны 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.

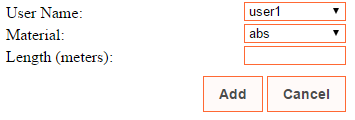
В самом же приложении, нам будет достаточно отправлять и получать данные с нашего сервиса, не реализуя никакой логики и не думая о хранении данных, все вычисляемы значения будут автоматически считаться по формулам в spreadsheet файле, данные будут храниться в нём же и нам остаётся только их показать,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

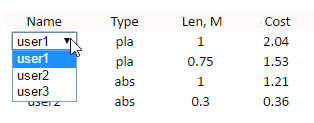
У SpreadsheetCloudAPI, есть очень полезные функции экспорта в HTML и экспорта чартов как картинок, что очень нам на руку, так как визуализировать наши данные мы можем просто экспортируя нужные участки документа в наше приложение, на этом вопрос визуализации будет закрыт.

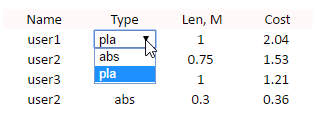
При экспорте данные будут представлять из себя таблицу и для редактирования этих данных нам достаточно будет обрабатывать клик на ячейке таблицы и вешать на неё нужный нам эдитор. Для редактирования значений мы используем «Input», для выбора значений из уже имеющихся «Select».

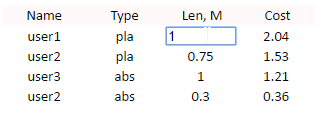
Например при создании нового сеанса печати нам понадобится выбрать имена пользователя и материала и ввести значение длины.



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

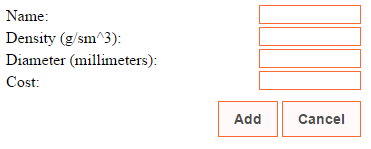


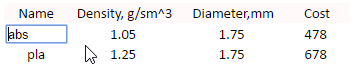


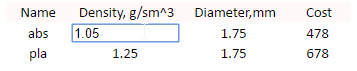


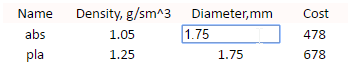
Колонка «Cost» на вкладке «Prints» редактироваться не будет, она будет вычислятся непосредственно на сервисе и вставлятся при экспорте.

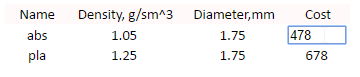
На вкладке «Materials» мы будем целиком задавать значения, то есть использовать только «Input», как при создании материала, так и при изменении его параметров.



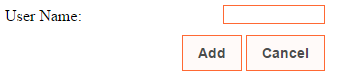


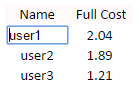






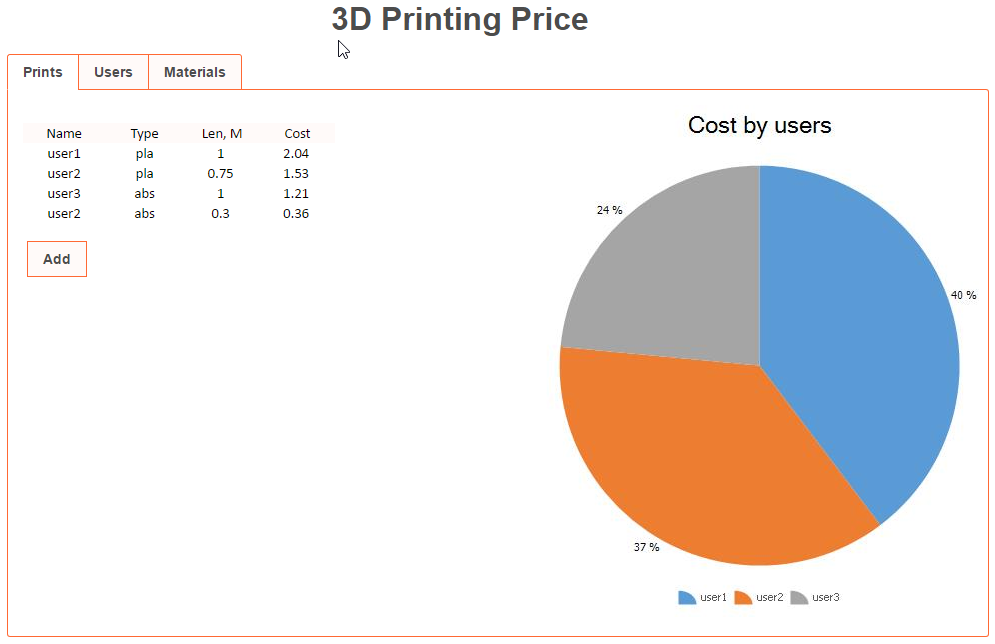
На вкладке «Users» единственный параметр который мы будем создавать и изменять это имя пользователя, используем для этого уже знакомые нам элементы «Input»





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 при создании 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.