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

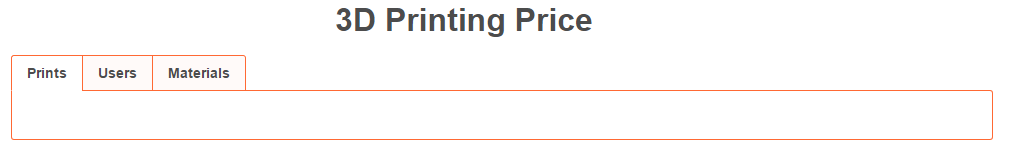
Здравствуйте дорогие читатели.

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.

Что-же мы хотим от приложения.

Мы должны редактировать информацию о 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 мы должны визуализировать информацию с помощью чарта.

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



Расчет 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 другой способ и решили немножко изменить подход. Нами 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.

Для этих целей нам идеально подходит наш продукт SpreadsheetcloudAPI [ссылка на сервис].

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.

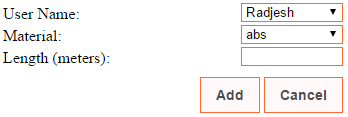
В самом же приложении, нам будет достаточно отправлять и получать данные с нашего сервиса, не реализуя никакой логики и не думая о хранении данных, все вычисляемы значения будут автоматически считаться по формулам в spreadsheet файле, данные будут храниться в нём же и нам остаётся только их показать,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

Создание spreadsheet файла для расчётов и хранения данных, описаны в отдельном посте [вот ссылка].

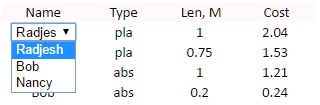
У SpreadsheetCloudAPI, есть очень полезные функции экспорта в HTML и экспорта чартов как картинок, что очень нам на руку, так как визуализировать наши данные мы можем просто экспортируя нужные участки документа в наше приложение, на этом вопрос визуализации будет закрыт.

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.

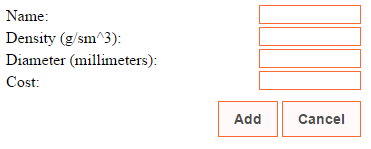


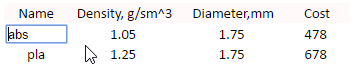
Для редактирования уже существующего сеанса печати мы будем использовать те-же элементы что и при создании.



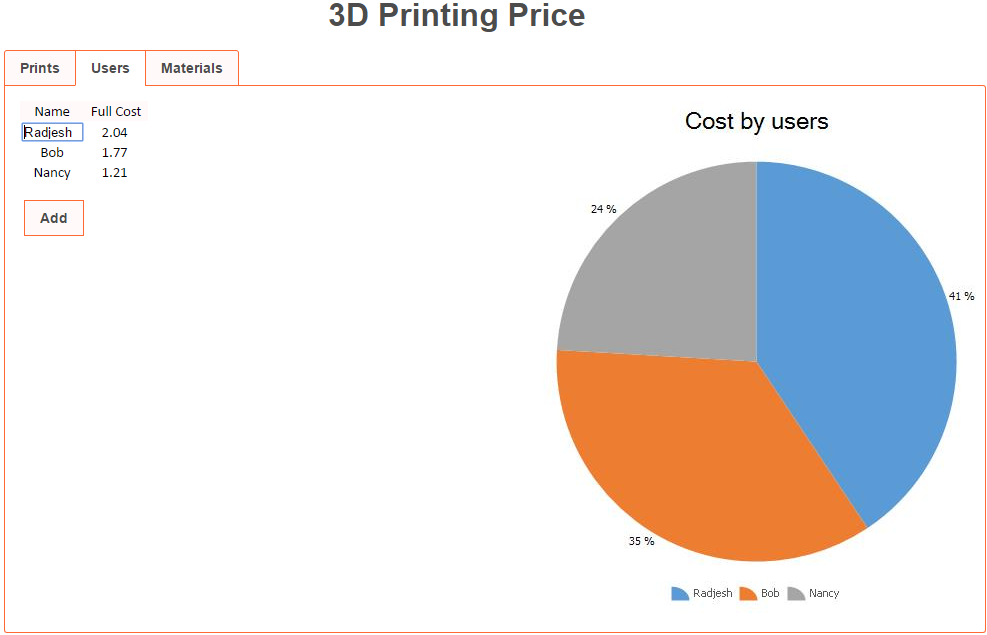
Колонка «Cost» на вкладке «Prints» редактироваться не будет, она будет вычислятся непосредственно на сервисе и вставлятся при экспорте.

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 элементы «Input»



На этом создание UI закончено. Давайте теперь обратимся к бэкэнду.

Всё общение с сервисом у нас проходит на серверной стороне. Для серверной части нашего приложения мы использовали PHP.

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

Для общения с сервисом нам нужно всего три параметра:

URL сервиса - 'http://spreadsheetcloudapi.azurewebsites.net/api/spreadsheet',

API Key – мы получаем на сервисе, более подробно [ссылка на пост про шаблон],

Имя файла.

Для удобства работы на PHP, 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 типа запросов: «PUT» и «GET», их реализация на PHP представлена в коде приложения function put( $params, $url ) и function get( $params, $url )

где: $params – параметры запроса к сервису $url – тип запроса к сервису.

Для примера мы рассмотрим методы экспорта HTML и получение ImageBytes:

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' );

}

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

$id = loadDocument($filename);

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.

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

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, будет достаточно отредактировать шаблон.

Само приложение и шаблон можно скачать по ссылке.

Спасибо за внимание.