About the issue:

When a post request to add a new row is sent to Google sheets URL, there is no effect. There is no feedback from the Google sheets API server that would help me identify the mistake.

Overview of steps I am doing to make a post request to Google sheets:

1. Create an OAuth2 client id and secret for the application with which we want to use boxes library.
2. Used google-oauth2 library to get access token and refresh token for authenticating each request to Google sheets API.
3. Used xml library to create the row in the format that google sheets API accepts.
4. Used http-conduit to construct a post request with above constructed xml bytestring as request body.

Detailed description of each of the above steps:

1. Create an OAuth 2.0 client id and secret for the application with which we want to use boxes library.

1. We must obtain OAuth 2.0 credentials from the [Google Developers Console](https://console.developers.google.com/) which can be used to obtain an access token from Google sheets API in the future.
2. After clicking the link above you will be redirected to login using your google account.
3. After login click on the left side top corner navigation button to draw out the side panel.
4. Click the API Manager Menu item.
5. In the sub menu of API manager click on the Credentials item. Here you can create a new application which gets unique client id and secret for that application.
6. Create a new application by clicking on “New Credentials” drop down to choose “OAuth Client ID”.
7. Then choose the application type as “other” and give your application a name you wish and click on create.
8. A new application gets created. Client ID and Client Secret are displayed against the application’s row. Copy these credentials for use in your application.

2. Using google-oauth2 library to make requests to the sheets API.

Before an application can access private data using any Google API, it must obtain an access token that grants access to that API. A variable parameter called scope controls the set of resources and operations that an access token permits. During the access token request, our application sends one or more values in the scope parameter.

For google sheets API the scope we must send is <https://spreadsheets.google.com/feeds>

When our application needs access to sheets data, it asks Google for above scope of access.

Google displays an OAuth dialog to the user, asking them to authorize your application to request their private data.

If the user approves, then Google gives our application a short-term access token.

Our application then requests user data, attaching the access token to the request.

If Google determines that our request and the token are valid, it returns the requested data.

[Google-oauth2](https://hackage.haskell.org/package/google-oauth2) library performs these functions for us.

The example in the above link exactly shows how to do the oauth authentication request takes place.

The library provides function called:

getAccessToken :: OAuth2Client

-> [OAuth2Scope]

-> Maybe FilePath -- ^ File in which to cache the token

-> IO OAuth2Token -- ^ Refreshed token

Which takes OAuth2Client (ClientID and Secret), scope [https://spreadsheets.google.com/feeds], a temp file for caching the credentials once obtained from the user.

When we supply these parameters to the function and if it executes for the first time it prompts the user for a verification code. It displays a url which we need to use to generate verification code from the browser by authenticating with the google account that owns the sheet. After obtaining the verification code supply to the prompt.

Then the library posts that code to the Google API for a set of tokens (access and refresh). The credentials which are obtained are stored in the file we supplied as parameter for reuse.

We then use the access token for subsequent requests until it expires.

We use the refresh token to get a new access token.

These above steps are all done by above getAccessToken function.

In our code:

In GoogleRequest.hs there are various functions that perform the above tasks by supplying correct parameters.

scopes = ["https://spreadsheets.google.com/feeds"]

createToken client = getAccessToken client scopes (Just "./key.txt")

authorize token request = request

{requestHeaders = [(hAuthorization, B8.pack $ "Bearer " ++ token)] }

The above authorize function is used to add the token header to the request to be sent to Google Sheets API for authentication.

3. Used xml library to create the row in the format that google sheets API accepts.

The google sheets api documentation clearly tells us the format of xml to be created to make a post request to create a new row in a particular google sheet.

Google sheets api considers the first row of the sheet as column headers.

If we need to send data the column headers should be considered.

The example given in the documentation is:

<entry xmlns="http://www.w3.org/2005/Atom"  
    xmlns:gsx="http://schemas.google.com/spreadsheets/2006/extended">  
  <gsx:hours>1</gsx:hours>  
  <gsx:ipm>1</gsx:ipm>  
  <gsx:items>60</gsx:items>  
  <gsx:name>Elizabeth Bennet</gsx:name>  
</entry>

In this example hours, ipm, items, name are the column headers.

To form this xml, [xml](https://hackage.haskell.org/package/xml) library is used.

In our code the above xml generation is automated by the use of below functions.



“makexml” function takes a list of tuples of column’s header name and corresponding value to convert above shown xml.

“makexmlrow” in the above function converts a column’s header name and corresponding value tuple into xml row as shown in the above example.

“unode” is an xml library function responsible to create a node such as <entry> in the above example.

“showElement” converts xml to string.

“add\_attrs” is also an xml library function used to add attributes to a particular node. In the above makexml function 2 attributes xmlns and xmlnsgsx are added to <entry> node.

4. Used http-conduit to construct a post request with above constructed xml as request body.

http-conduit is a client side library used to interact with RESTful APIs.

In our library we use http-conduit to do GET, POST, DELETE and PUT requests.

In googleRequest.hs the post function is used to create a post request to add a row to Google Sheet.

The URL to which post request is done is constructed using the key of the Google sheet.

Several headers are added to the request “Content-Length”, “Content-Type” and authorization token.

“Content-Length” is constructed dynamically according to the length of the message to be sent.

“Content-Type” is "application/atom+xml"

The string data is converted to Data.ByteString.Lazy.ByteString and is given as requestBody.

The post method gives more insight.

post url client postData = do

token <- createToken client

request <- parseUrl url

let pd = B.toLazyByteString $ B.stringUtf8 $ "<?xml version='1.0' encoding='utf-8'?>" ++ postData

let req = request{

method = "POST"

, requestHeaders = [("Content-Type", "application/atom+xml"),

("Content-Length", B8.pack $ show $ length $

"<?xml version='1.0' encoding='utf-8'?>" ++ postData)]

, requestBody = RequestBodyLBS pd

}

let postReq = authorize token postReq

res <- withManager $ httpLbs req

return $ responseStatus res

If the request is valid the Google API server updates the new row to the sheet and returns the corresponding xml of the row.

The url is for post request is formed by using this template.

urlTemplate = "https://spreadsheets.google.com/feeds/list/${key}/od6/private/full" :: String

And then replacing the key parameter based on the user input using:

formPostURL :: String -> String

formPostURL key = T.unpack $ T.replace (T.pack "${key}") (T.pack key) (T.pack urlTemplate)

addRow method in Lib.hs is responsible for calling all the above methods to process everything.

It validates the user’s row and then converts into xml string, constructs the url based on the key and then calls GoogleRequest’s post method.

