

[Get started](#)[Open in app](#)

Balakrishnan Nalin Prashanth

[Follow](#)

76 Followers

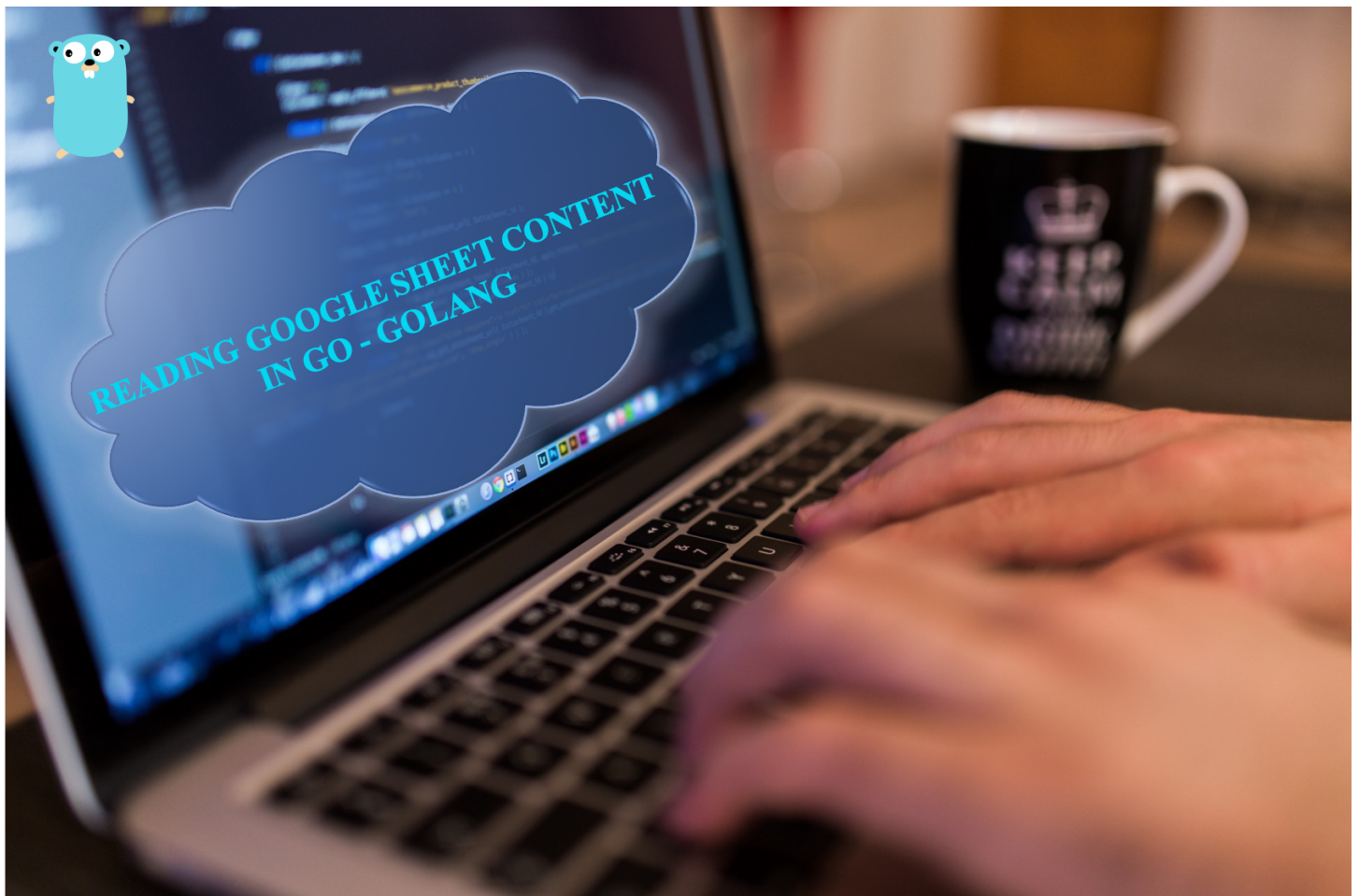
[About](#)

You have **1** free member-only story left this month. [Sign up for Medium and get an extra one](#)

Reading Google Sheet Content in GO — GOLANG



Balakrishnan Nalin Prashanth Apr 20, 2019 · 8 min read ★



Cover Image — Reading Google Sheet Content in GO

GO programming language is becoming more popular day by day and is edging out other programming languages in various aspects. It is always a challenge to learn a new programming language. Learning a technology from Google's Technology Stack can be pretty useful especially in the long term. Well, I'm not gonna explain about GO here.

Let's dive into the topic. There are many instances where an application needs to access data from a google sheet, typically in a Business Intelligent Application, where all data are maintained in google sheet and need to visualize the data programmatically or in a Machine Learning Application, where an application can host a model which can be trained given a dataset (in the form of a google sheet), and get the logs of the training result and get the trained model to do some predictions.

But the use-case I am gonna take as an example here is, maintaining a language file as a google sheet in a multi-lingual application (accommodating the concept of internationalization).

Maintaining Language File as Google Sheet, Why?? How does it even work out??

Many people would encounter questions as to why such a weird use-case (application). Well, there's purpose in doing so (it has some unique benefits).

Let me explain **“Why”** first,

- In such an application, the Google Sheet becomes a single place where the language mappings are kept. Hence, provides single point of control when bringing in changes to the language strings and also maintenance of the language file becomes much easier.
- Language strings can be modified during the run time and also update the hosted language files which are generally kept in assets/resources directory without interrupting the running servers.
- Adding a new language or removing an existing language can be easily performed, just by adding a new column with new language strings or removing an existing language string column respectively, which allows us to make very little (or in some cases no changes) to the running application.

Now let me explain **“How”**,

- Consider we have a google sheet with columns named, key, en (English strings), fr (French strings), and so on for different languages. “key” is what we use in the real application (code) to map (or identify) a specific language string.
- And also consider that we are developing a web application using a JS framework like Angular and we want to make our web application multi-lingual (internationalization), i.e. upon some criteria we want to change the language of the content which is displayed in the UI. In here, we would have a “i18n” folder inside assets/resources folder and have a language file for each of the language in json format (which is the usual practice).
- And now let’s connect both together. What if you can write a simple program which will get the language file which is a google sheet, read all the content in the google sheet, create individual json files for each language available in the google sheet in the assets folder of the web application, so that the content displayed in the web application is updated when a change takes place in the google sheet. *May be by running a **cronjob** in the hosted machine for every ‘x’ days/hours/minutes.*

*** Hope you understood what I tried to explain here, if you have any doubts, questions or suggestions please let me know in comments.*

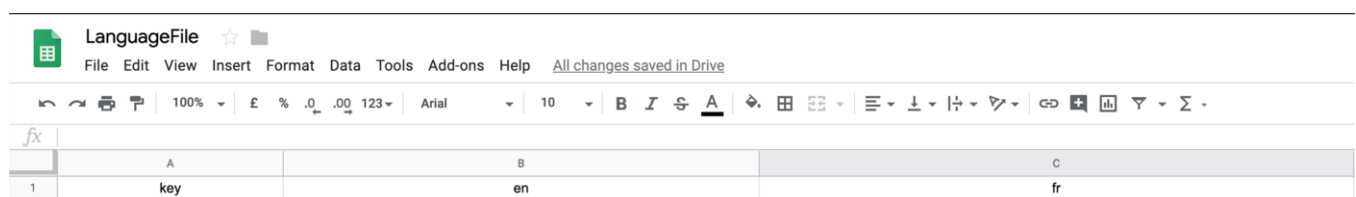
Let’s Dive in to Coding..

I am gonna have 2 approaches to address this specific problem, first approach is accessing the google sheet using the published link, downloading it as a .csv file and then, creating individual json files and the other approach is to make use of the Google Sheets API to access the google sheet and create json files using the content directly.

In this article, I am gonna show the implementation of the first approach, which is accessing the google sheet via the published url and in my next article, I’ll be showing the implementation using the Google Sheets API.

Accessing the Google Sheet via it’s Published URL

Let’s do the necessary background work first. Let’s create a new google sheet and add the relevant data so that your google sheet looks similar to the following google sheet I created:



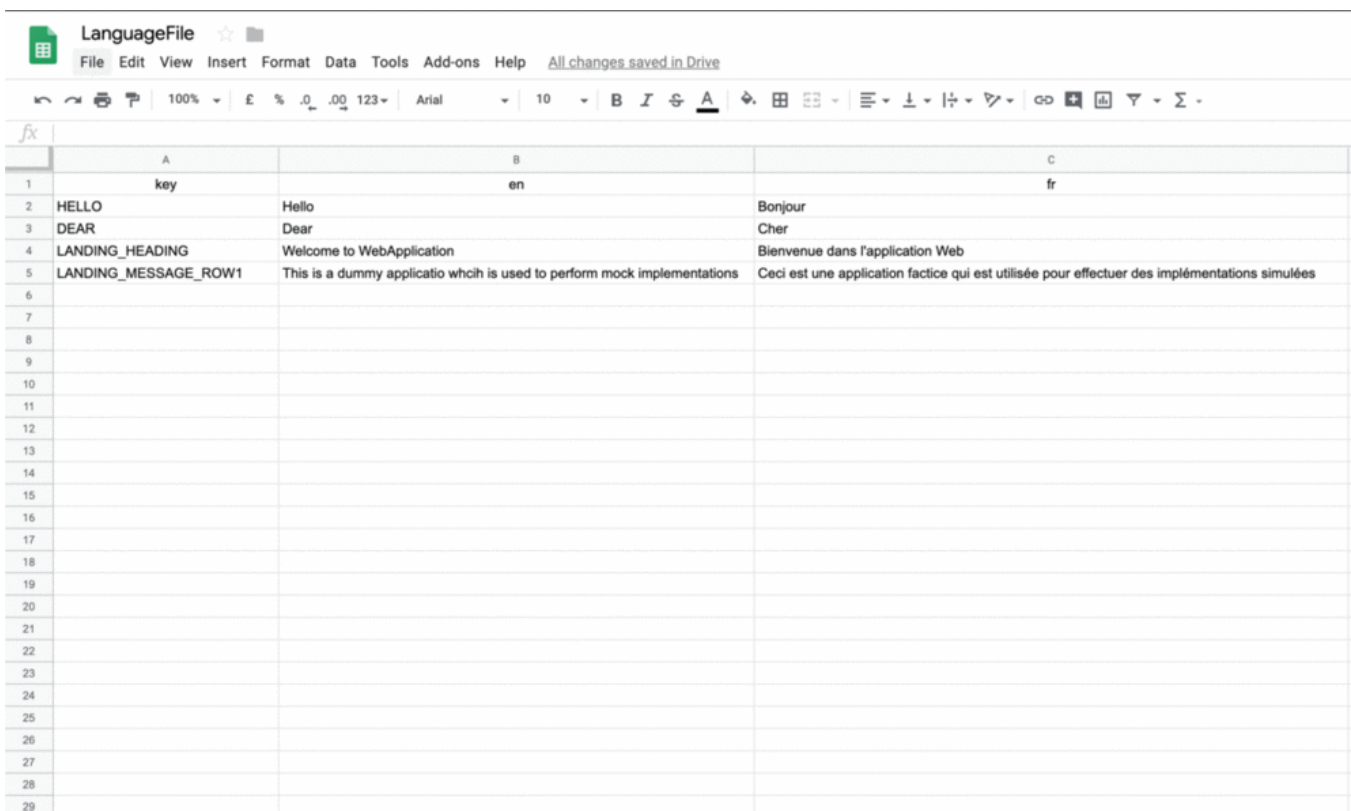
	A	B	C
1	key	en	fr

2	HELLO	Hello	Bonjour
3	DEAR	Dear	Cher
4	LANDING_HEADING	Welcome to WebApplication	Bienvenue dans l'application Web
5	LANDING_MESSAGE_ROW1	This is a dummy applicatio whcih is used to perform mock implementations	Ceci est une application factice qui est utilisée pour effectuer des implémentations simulées
6			
7			
8			
9			
10			
11			
12			
13			

Sample Google Sheet

The first column should be the key used for mapping, and the other columns should contain language strings for specific languages, and first row should contain the string used to identify the language or the name that is to be given to the json files that we will be creating eventually.

And now we will publish the google sheet to the web. Open the google sheet, click on file and select the publish to the web . And there, select the option comma-separated values (.csv) and then click publish. Then, you'll get a url, save the url for future use.



	A	B	C
1	key	en	fr
2	HELLO	Hello	Bonjour
3	DEAR	Dear	Cher
4	LANDING_HEADING	Welcome to WebApplication	Bienvenue dans l'application Web
5	LANDING_MESSAGE_ROW1	This is a dummy applicatio whcih is used to perform mock implementations	Ceci est une application factice qui est utilisée pour effectuer des implémentations simulées
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			
26			
27			
28			
29			

Steps to Publish the Google Sheet to Web

Now let's create the project's directory structure. We will be making use of the logger I created in the following article:

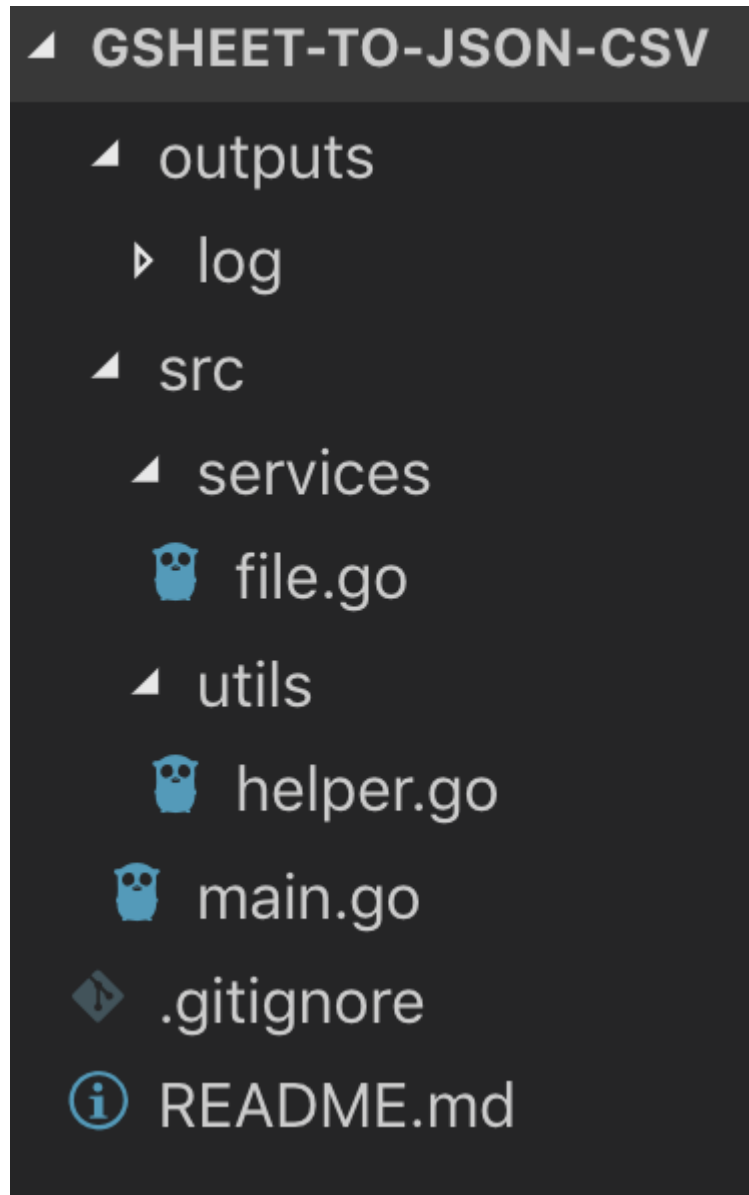
Implementing Logger in GO — GOLANG

It is very important to follow best practices when programming,

logging is one such best practice used in industry and...

medium.com

And let's create the following directories and files as shown in the following image:



Directory Structure of the Project

I am not gonna explain a lot about the `helper.go` file, since I have already explained about the `init` function and loggers in [this article](#). I have added a new struct and a function to the file, I am not gonna explain a lot here, those changes are done to handle and return errors in a better manner and the `helper.go` file should look as below:

```
1 package utils
2
3 import (
```

```

4         "fmt"
5         "log"
6         "os"
7         "path/filepath"
8     )
9
10    // ErrorResponse exported
11    type ErrorResponse struct {
12        Message string
13        Err      error
14    }
15
16    // GeneralLogger exported
17    var GeneralLogger *log.Logger
18
19    // ErrorLogger exported
20    var ErrorLogger *log.Logger
21
22    func init() {
23        absPath, err := filepath.Abs("../outputs/log")
24        if err != nil {
25            fmt.Println("Error reading given path:", err)
26        }
27
28        generalLog, err := os.OpenFile(absPath+"/general-log.log", os.O_RDWR|os.O_CREATE|os.O_APPEND, 0666)
29        if err != nil {
30            fmt.Println("Error opening file:", err)
31            os.Exit(1)
32        }
33        GeneralLogger = log.New(generalLog, "General Logger:\t", log.Ldate|log.Ltime|log.Lmicroseconds)
34        ErrorLogger = log.New(generalLog, "Error Logger:\t", log.Ldate|log.Ltime|log.Lmicroseconds)
35    }
36
37    // ReturnErrorResponse exported
38    func ReturnErrorResponse(err error, message string) *ErrorResponse {
39        return &ErrorResponse{
40            Message: message,
41            Err:      err,
42        }
43    }

```

gsheet-to-json-csv_helper.go hosted with ❤ by GitHub

[view raw](#)

The approach is quite straight forward,

- Download the published google sheet in .csv format

- Extract the content within the downloaded file to create the json files

Let's create 2 functions in the fileService.go file to perform the above 2 steps as follows:

→ A function to download the google sheet as a .csv file

```

1  func Download(url string, filename string, timeout int64) *u.ErrorResponse {
2      u.GeneralLogger.Println("Downloading", url, "...")
3      client := http.Client{
4          Timeout: time.Duration(timeout * int64(time.Second)),
5      }
6      resp, err := client.Get(url)
7      if err != nil {
8          u.ErrorLogger.Println("Cannot download file from the given url", err)
9          return u.ReturnErrorResponse(err, "Cannot download file from the given
10     }
11
12     if resp.StatusCode != 200 {
13         u.ErrorLogger.Printf("Response from the URL was %d, but expecting 200",
14             return u.ReturnErrorResponse(
15                 errors.New("Response returned with a status different from 200",
16                 "Response returned with a status different from 200",
17             )
18     }
19     if resp.Header["Content-Type"][0] != "text/csv" {
20         u.ErrorLogger.Printf("The file downloaded has content type '%s', expect
21             return u.ReturnErrorResponse(
22                 errors.New("Downloaded file didn't contain the expected content
23                 "Downloaded file didn't contain the expected content-type: 'tex
24             )
25     }
26
27     b, err := ioutil.ReadAll(resp.Body)
28     if err != nil {
29         u.ErrorLogger.Println("Cannot read Body of Response", err)
30         return u.ReturnErrorResponse(err, "Cannot read Body of Response")
31     }
32
33     err = ioutil.WriteFile(filename, b, 0644)
34     if err != nil {
35         u.ErrorLogger.Println("Cannot write to file", err)
36         return u.ReturnErrorResponse(err, "Cannot write to file")
37     }
38
39     u.GeneralLogger.Println("Doc downloaded in ", filename)
40
41     return u.ReturnErrorResponse(nil, "")

```



```

42 }

```

gsheet-to-json-csv_download.go hosted with ❤ by GitHub

[view raw](#)

The above function takes in 3 parameters, url, filename and timeout. The function first sets the timeout for the request and attempts to download the file provided via the url, then there a couple of checks: the status of the response and the content-type provided by the response, if they are of expected values, then we read the body of the response (where the content is available) and write it to the file with permission 644 (644 means, creator can modify/write, while others can only read the file), if need be, change the permissions to 666 so that everyone can read and write. The function returns an ErrorResponse (A custom struct).

→ A function to read the downloaded content and write in to json files

```

1  func WritLanguageFiles(csvFilePath string) *u.ErrorResponse {
2      csvFile, err := os.Open(csvFilePath)
3      if err != nil {
4          u.ErrorLogger.Println("Cannot open file:"+csvFilePath, err)
5          return u.ReturnErrorResponse(err, "Cannot open file:"+csvFilePath)
6      }
7
8
9      csvFileContent, err := csv.NewReader(csvFile).ReadAll()
10     for i, lang := range csvFileContent[0][1:] {
11         absPath, err := filepath.Abs(outputPath + lang + ".json")
12         if err != nil {
13             u.ErrorLogger.Println("Cannot get path specified: \""+lang+".js
14             return u.ReturnErrorResponse(err, "Cannot get path specified: \"
15         }
16
17         file, err := os.OpenFile(absPath, os.O_CREATE|os.O_WRONLY, 0644)
18         if err != nil {
19             u.ErrorLogger.Println("Cannot open file: \""+lang+".json\"", er
20             return u.ReturnErrorResponse(err, "Cannot open file: \""+lang+
21         }
22         file.Truncate(0)
23         mapLn := map[string]string{}
24         u.GeneralLogger.Println("Language:", lang, i)
25         for j, row := range csvFileContent[1:] {
26             // fmt.Println(csvFileContent[j+1][0], row[i+1])
27             mapLn[csvFileContent[j+1][0]] = row[i+1]
28         }
29         encodedJSON, _ := json.Marshal(mapLn)
30         // u.GeneralLogger.Println(string(encodedJSON))

```



```

30         // u.GenerateJSON() function (string(encodedJSON)),
31         file.Write(encodedJSON)
32         file.Close()
33     }
34     return u.ReturnErrorResponse(nil, "")
35 }

```

gsheet-to-json-csv_writefile.go hosted with ❤ by GitHub

[view raw](#)

The above function receives a single parameter which is the path to the created .csv file. First it opens the file and reads all content within, then a for loop is created to loop over the number of languages in the file:

```
for i, lang := range csvFileContent[0][1:] {...}
```

- csvFileContent[0][1:] means take the first (0th index) row from the content and from that row take all records starting from index 1 (2nd record onwards). Hence, it will create a list of languages (**Remember, our google sheet had **key** as the first column and, languages start from the second column onwards*).

Inside the first loop, we are creating/writing to the json (language) files with permission 644 (**as I mentioned in the Download function's explanation, if need be use permission 666*). And then,

```
mapLn := map[string]string{}
```

a map type is initiated to create a json object in order to write to a json file, the map is populated via another for loop which iterated over the rows in the csv file except the first row which is not row that contains language strings.

```

for j, row := range csvFileContent[1:] {
    mapLn[csvFileContent[j+1][0]] = row[i+1]
}

```

Finally the json file is written:

```
encodedJSON, _ := json.Marshal(mapLn)
```

```
file.Write(encodedJSON)
```

The created map is converted to a byte array so that it could be written to the created .json file. Now our fileService.go file should look as follows:

```
1  package services
2
3  import (
4      "encoding/csv"
5      "encoding/json"
6      "errors"
7      u "gsheet-to-json-csv/src/utils"
8      "io/ioutil"
9      "net/http"
10     "os"
11     "path/filepath"
12     "time"
13 )
14
15 // Should point to the folder where language json files are kept
16 const outputPath = "../outputs/"
17
18 // Download exported
19 func Download(url string, filename string, timeout int64) *u.ErrorResponse {
20     u.GeneralLogger.Println("Downloading", url, "...")
21     client := http.Client{
22         Timeout: time.Duration(timeout * int64(time.Second)),
23     }
24     resp, err := client.Get(url)
25     if err != nil {
26         u.ErrorLogger.Println("Cannot download file from the given url", err)
27         return u.ReturnErrorResponse(err, "Cannot download file from the given")
28     }
29
30     if resp.StatusCode != 200 {
31         u.ErrorLogger.Printf("Response from the URL was %d, but expecting 200",
32             resp.StatusCode)
33         return u.ReturnErrorResponse(
34             errors.New("Response returned with a status different from 200",
35                 "Response returned with a status different from 200",
36             )
37     }
38     if resp.Header["Content-Type"][0] != "text/csv" {
39         u.ErrorLogger.Printf("The file downloaded has content type '%s', expect
```

```

39         return u.ReturnErrorResponse(
40             errors.New("Downloaded file didn't contain the expected content
41                 \"Downloaded file didn't contain the expected content-type: 'tex
42         )
43     }
44
45     b, err := ioutil.ReadAll(resp.Body)
46     if err != nil {
47         u.ErrorLogger.Println("Cannot read Body of Response", err)
48         return u.ReturnErrorResponse(err, "Cannot read Body of Response")
49     }
50
51     err = ioutil.WriteFile(filename, b, 0644)
52     if err != nil {
53         u.ErrorLogger.Println("Cannot write to file", err)
54         return u.ReturnErrorResponse(err, "Cannot write to file")
55     }
56
57     u.GeneralLogger.Println("Doc downloaded in ", filename)
58
59     return u.ReturnErrorResponse(nil, "")
60 }
61
62 // WriteLanguageFiles exported
63 func WriteLanguageFiles(csvFilePath string) *u.ErrorResponse {
64     csvFile, err := os.Open(csvFilePath)
65     if err != nil {
66         u.ErrorLogger.Println("Cannot open file:"+csvFilePath, err)
67         return u.ReturnErrorResponse(err, "Cannot open file:"+csvFilePath)
68     }
69
70
71     csvFileContent, err := csv.NewReader(csvFile).ReadAll()
72     for i, lang := range csvFileContent[0][1:] {
73         absPath, err := filepath.Abs(outputPath + lang + ".json")
74         if err != nil {
75             u.ErrorLogger.Println("Cannot get path specified: \""+lang+".js
76             return u.ReturnErrorResponse(err, "Cannot get path specified: \
77         }
78
79         file, err := os.OpenFile(absPath, os.O_CREATE|os.O_WRONLY, 0644)
80         if err != nil {
81             u.ErrorLogger.Println("Cannot open file: \""+lang+".json\"", er
82             return u.ReturnErrorResponse(err, "Cannot open file: \""+lang+
83         }
84         file.Truncate(0)
85         mapLn := map[string]string{}
86         u.GeneralLogger.Println("Language:", lang, i)

```

```

87         for j, row := range csvFileContent[1:] {
88             // fmt.Println(csvFileContent[j+1][0], row[i+1])
89             mapLn[csvFileContent[j+1][0]] = row[i+1]
90         }
91         encodedJSON, _ := json.Marshal(mapLn)
92         // u.GeneralLogger.Println(string(encodedJSON))
93         file.Write(encodedJSON)
94         file.Close()
95     }
96     return u.ReturnErrorResponse(nil, "")
97 }

```

The main.go file performs the simplest of tasks, of calling the relevant functions. And hence, add the following code to the main.go file which makes the file very small and simple.

```

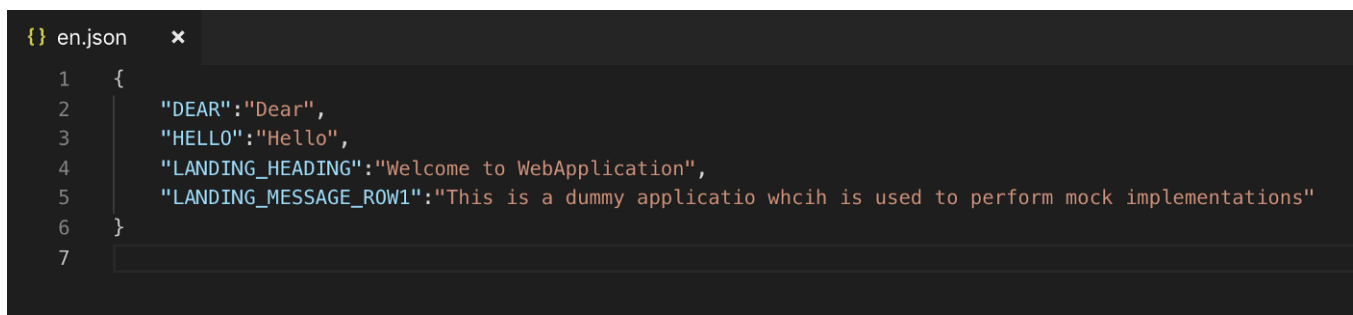
1  package main
2
3  import (
4      s "gsheet-to-json-csv/src/services"
5      u "gsheet-to-json-csv/src/utils"
6      "os"
7  )
8
9  func main() {
10     // Starting the Application
11     u.GeneralLogger.Println("Starting Extracting Language Files from GoogleSheet -")
12
13     csvFilePath := "../outputs/gsheet.csv"
14     errorResponse := s.Download(
15         "https://docs.google.com/spreadsheets/d/e/2PACX-1vQIhLNNfUKVjxMkMwdtTFr",
16         csvFilePath,
17         5000,
18     )
19     if errorResponse.Err != nil {
20         u.ErrorLogger.Println(errorResponse.Message, errorResponse.Err)
21         os.Exit(1)
22     }
23     s.WriteLanguageFiles(csvFilePath)
24     u.GeneralLogger.Println("Completed Execution..")
25 }

```

The code in that are self explanatory, on top of the file, services and utils packages are imported and the main function first calls the Download function by providing the url and filename, and then after successful download, calls the writeToFile function.

*** Before you run please make the necessary changes to the published url and file and/or directory paths mentioned in the code.*

Now let's run and see what happens, you will see 2 new files (if you have more columns in the google sheet than me, you'll see more json files being created) in the outputs directory (or the path you have specified in your code). And the files should contain the proper data in the proper structure. Example, please take a look at my en.json file:



```
{ } en.json x
1  {
2    "DEAR":"Dear",
3    "HELLO":"Hello",
4    "LANDING_HEADING":"Welcome to WebApplication",
5    "LANDING_MESSAGE_ROW1":"This is a dummy applicatio whcih is used to perform mock implementations"
6  }
7
```

Newly created en.json file

Well, it was quite a long article than I expected. Anyway, I hope I did a good job and this was helpful and you like what you just read. Please let me know your comments and suggestions.

*** The source code can be found at: <https://github.com/BNPrashanth/gsheet-to-json-csv>*

*** The idea behind the implementation shown in this article can not only be used to perform the mentioned implementation in a multi-lingual application but also in various other software applications where there is a need to read content from a google sheet and would be really useful in the context of **Business Intelligent applications**, to get visual data done faster from data available in the form of google sheets and also in **Machine Learning/Data Science applications**.*

*** In this code, some of the inputs are hardcoded in order to run the example application provided which is of-course not the best-practice when it comes to any programming language (Need to be taken from environment configurations, which is the better practice). I will have a few articles on best-practices in GO in future like the preferred directory structure, dependency management and vendoring, reading from environment configurations, etc.*