

# FINAL PROJECT: EXCEL SHEET TO WORD DOC

## Project Overview:

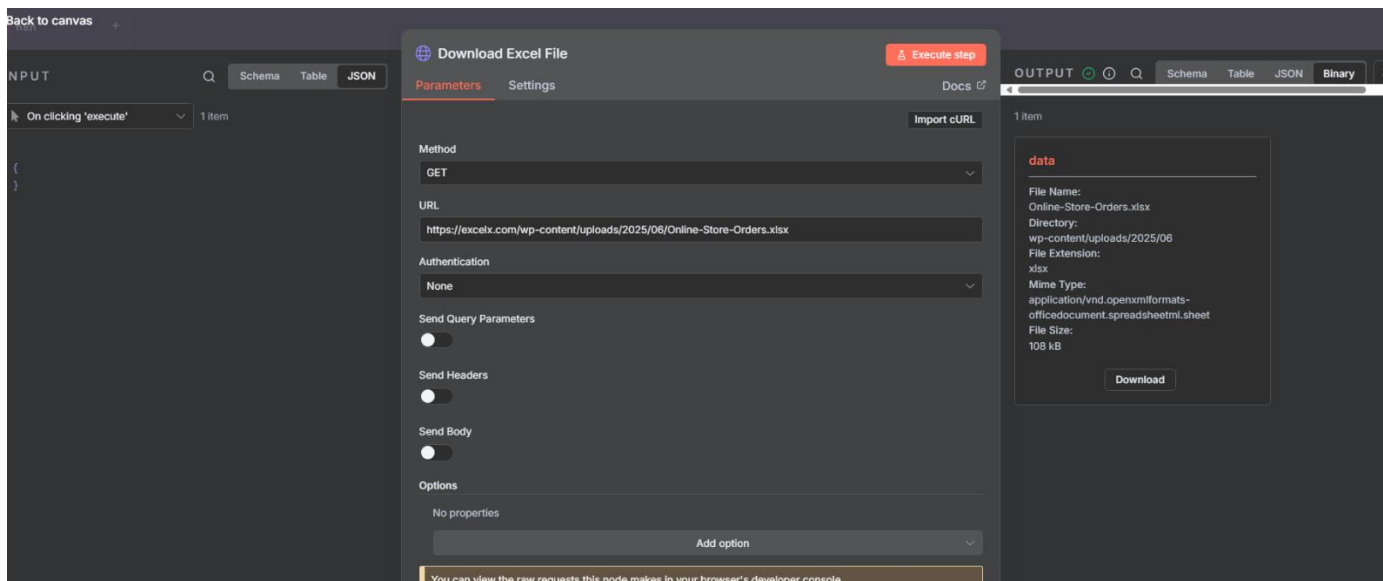
This project automates the transformation of Excel-based order data into a professionally formatted Word document using n8n and Documentero. It reads spreadsheet rows, restructures them into JSON, and generates a single .docx file with all entries displayed in a dynamic table. The workflow eliminates manual formatting and supports scalable, repeatable document generation.

## Value Proposition:

- Saves time and effort by automating the conversion of Excel data into a structured Word document, eliminating manual formatting and repetitive copy-paste tasks.
- Generates a single .docx file with all entries organized in a clean, tabular layout—ideal for summaries, reports, or invoices.
- Uses n8n's workflow automation and Documentero's templating engine to ensure consistent formatting and scalable document generation.
- Keeps your data pipeline efficient by processing all rows in bulk and saving the final output directly to your local system.
- Easy to customize and extend—you can modify templates, add filters, or integrate email and cloud storage for seamless delivery.

## Step by Step workflow:

### 1. Download Excel File



## 2. Read Spreadsheet File

The screenshot shows the 'Read Spreadsheet File' node configuration. The 'INPUT' panel on the left shows a file named 'data.xlsx' with a size of 108 KB. The 'Parameters' tab is active, showing the 'Operation' set to 'Read From File' and the 'Input Binary Field' set to 'data'. The 'Options' section has 'Header Row' checked. The 'OUTPUT' panel on the right displays 1200 items of JSON data, including order details like OrderID, Date, CustomerID, Product, Quantity, UnitPrice, ShippingAddress, PaymentMethod, OrderStatus, TrackingNumber, ItemsInCart, CouponCode, ReferralSource, and TotalPrice.

## 3. Code in JavaScript

The screenshot shows the 'Code in JavaScript' node configuration. The 'INPUT' panel on the left shows the output of the 'Read Spreadsheet File' node. The 'Parameters' tab is active, showing the 'Mode' set to 'Run Once for All Items' and the 'Language' set to 'JavaScript'. The 'OUTPUT' panel on the right displays the result of the JavaScript code, which is a filtered list of items. The JavaScript code is as follows:

```
1 return items.map((item) => {
2   return {
3     json: {
4       Order: item.json.OrderID,
5       Product: item.json.Product,
6       Quantity: item.json.Quantity,
7       Total: item.json.TotalPrice
8     }
9   };
10 });
11
12
```

## 4. Loop Over Items

The screenshot shows the 'Code in JavaScript' node configuration with a loop. The 'INPUT' panel on the left shows the output of the 'Read Spreadsheet File' node. The 'Parameters' tab is active, showing the 'Batch Size' set to 1. The 'OUTPUT' panel on the right displays the result of the JavaScript code, which is a filtered list of items. The JavaScript code is as follows:

```
1 return items.map((item) => {
2   return {
3     json: {
4       Order: item.json.OrderID,
5       Product: item.json.Product,
6       Quantity: item.json.Quantity,
7       Total: item.json.TotalPrice
8     }
9   };
10 });
11
12
```

## 5. Code Node(JavaScript)

The screenshot shows the 'Code in JavaScript1' node configuration. The 'INPUT' tab on the left displays a JSON object: 

```
{  "Order": "ORD288088",  "Product": "Monitor",  "Quantity": 5,  "Total": 2853.1}
```

. The 'Parameters' tab is active, showing 'Mode' set to 'Run Once for Each Item' and 'Language' set to 'JavaScript'. The 'JavaScript' code area contains the following code: 

```
1 return {
2   json: {
3     OrderID: $json.OrderID,
4     Product: {
5       Name: $json.Product,
6       Quantity: $json.Quantity,
7       Total: $json.Total
8     }
9   }
10 }
11
12
```

. The 'OUTPUT' tab on the right shows the resulting JSON: 

```
{  "Product": {    "Name": "Monitor",    "Quantity": 5,    "Total": 2853.1  }}
```

. A tip at the bottom states: 'Type \$ for a list of special vars/methods. Debug by using console.log() statements and viewing their output in the browser console.'

## 6. HTTPS Request

The screenshot shows the 'HTTPS Request' node configuration. The 'INPUT' tab on the left displays the same JSON object as in the previous screenshot. The 'Parameters' tab is active, showing 'Name' set to 'Content-Type' and 'Value' set to 'application/json'. The 'Send Body' toggle is turned on. The 'Body Content Type' is set to 'JSON'. The 'Specify Body' dropdown is set to 'Using JSON'. The 'JSON' body is defined as: 

```
{  document: "09XqQGRN4GcuIgQQNo08",  apiKey: "LN3CT2I-IZGEXAA-QQCUPQ-QP7DBFQ",  format: "docx",  data: {    Order: $json.Order  }}
```

. The 'Options' section is empty. A tip at the bottom states: 'You can view the raw requests this node makes in your browser's developer console'.

## 7. Write to Disk

The screenshot shows the 'Read/Write Files from Disk' node configuration. The 'INPUT' tab on the left shows 'HTTP Request' as the selected node. The 'Parameters' tab is active, showing 'Operation' set to 'Write File to Disk'. The 'File Path and Name' is set to `{{ $json.Order }}.docx`. The 'Input Binary Field' is set to 'data'. The 'Options' section is empty. A tip at the bottom states: 'Use this node to read and write files on the same computer running n8n. To handle files between different computers please use other nodes (e.g. FTP, HTTP Request, AWS)'.

COMPLETE WORKFLOW:

