

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

The screenshot shows the n8n workflow canvas with a single step named "Download Excel File". The step is configured with the following parameters:

- Method:** GET
- URL:** <https://excelx.com/wp-content/uploads/2025/06/Online-Store-Orders.xlsx>
- Authentication:** None
- Send Query Parameters:** Off
- Send Headers:** Off
- Send Body:** Off
- Options:** No properties

The "Execute step" button is highlighted in red at the top right of the step panel. To the right of the step panel, the "OUTPUT" section displays the results of the step, including the file name, directory, file extension, mime type, and file size. A "Download" button is also present in the output panel.

2. Read Spreadsheet File

The screenshot shows the Moltin Node-RED interface with the 'Read Spreadsheet File' node selected. The input section shows a file named 'Online-Store-Orders.xlsx' has been uploaded. The parameters for the node are set to 'Read From File' and 'Input Binary Field' is set to 'data'. The output section shows the raw JSON data of the spreadsheet items.

```

[{"OrderID": "ORD0200000", "Date": 44330, "CustomerID": "C72649", "Product": "Monitor", "Quantity": 5, "UnitPrice": 370.62, "ShippingAddress": "533 Main St", "PaymentMethod": "Debit Card", "OrderStatus": "Shipped", "TrackingNumber": "TRX37947983", "ItemsInCart": 7, "CouponCode": "SAV10", "ReferralSource": "Instagram", "TotalPrice": 2853.1}, {"OrderID": "ORD0200001", "Date": 45327, "CustomerID": "C57359", "Product": "Phone", "Quantity": 2, "UnitPrice": 331.35, "ShippingAddress": "323 Main St", "PaymentMethod": "Online", "OrderStatus": "Shipped", "TrackingNumber": "TRX9186779", "ItemsInCart": 1, "CouponCode": "SAV10", "ReferralSource": "Referral", "TotalPrice": 662.7}, {"Order": "ORD0200002", "Product": "Tablet", "Quantity": 1, "Total": 275.4}, {"Order": "ORD0200003", "Product": "Chair", "Quantity": 1, "Total": 373.19}, {"Order": "ORD0200004", "Product": "Printer", "Quantity": 4, "Total": 1764.84}
]

```

3. Code in JavaScript

The screenshot shows the 'Code in JavaScript' node with a loop function that maps each item from the input array to a new object containing the Order ID, Product, Quantity, and Total Price. The output shows the transformed data.

```

function (items) {
  return items.map((item) => {
    return {
      Order: item.json.OrderID,
      Product: item.json.Product,
      Quantity: item.json.Quantity,
      Total: item.json.TotalPrice
    }
  });
}

```

4. Loop Over Items

The screenshot shows the 'Loop Over Items' node. It has a 'Batch Size' of 1 and is configured to run once for each input item. The output shows the transformed data from the previous step.

5. Code Node(JavaScript)

The screenshot shows the n8n interface with a 'Code in JavaScript' node. The input is a single item from a 'Loop Over Items' node, containing an order object with fields: OrderID, Product, Quantity, and Total. The output is a JSON array where each item contains the same fields. The JavaScript code is as follows:

```
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 }
```

A tooltip at the bottom says: "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 n8n interface with an 'HTTP Request' node. The input is a 'Loop Over Items' node, specifically the 'Done Branch'. The request is configured to send a JSON body with a fixed Content-Type of 'application/json'. The body content type is set to 'JSON' and the body is specified using 'Using JSON' with the following JSON structure:

```
{
  document: "09XqDGRN4G CulgQNo08",
  apiKey: "LN3CT2I-IZGEXAA-QQCUPQQ-XP7DBFQ",
  format: "docx",
  data: [
    {
      document: "09XqDGRN4G CulgQNo08",
      apiKey: "LN3CT2I-IZGEXAA-QQCUPQQ-XP7DBFQ",
      format: "docx",
      data: ...
    }
  ]
}
```

A tooltip at the bottom says: "You can view the raw requests this node makes in your browser's developer console".

7. Write to Disk

The screenshot shows the n8n interface with a 'Read/Write Files from Disk' node. The input is an 'HTTP Request' node. The operation is set to 'Write File to Disk' and the file path and name is '\$json.Order.docx'. A note at the top says: "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)." The input binary field is set to 'data'.

COMPLETE WORKFLOW:

