****

**JC File data mapping**

**AI Agent**

**Development Document**

Prepared By: Prepared for:  
 JC

Muhammad Usman

Date of Document: Document Version:  
[2025-06-25] [1]

## [**https://www.loom.com/share/90cc208fe1dc4c7c8e5286888e49e4b9**](https://www.loom.com/share/90cc208fe1dc4c7c8e5286888e49e4b9)

## **Project planning document:**

[JC Mapping -FISTA Solutions 2025](https://docs.google.com/spreadsheets/d/1NxQ0eQ38AfVwcVw5uweUNGU35eMAdpHT8sTdA40AP1o/edit?gid=1522575117#gid=1522575117)

## **Openai API Key:**

## **sk-proj-VxEZ0RGFcnUkHklxwHdv2fM0Vpk9-dODKjArlfzmUeH2JmS\_\_o30wKQN07kMC5f8xKfTi6B9QXT3BlbkFJk95lyp9REvElvcV6RODHInfApzObt58oCsTwPTSwEAM5ndwcuDubGGSyQln6OVGDQ2mB2fgPwA**

## **Initial RFP:** [**Request for Proposal (RFP): AI-Powered Data Agent for Product Configuration**](https://docs.google.com/document/d/19psjymMLw0c8JL0dVlhCe-c6Wa-TZ5Dyn7oLSCpBlBU/edit?tab=t.0#heading=h.c75tb8u3xqil)

1. JC Data Form - Configurable Product Template.xlsx is the shape we are trying to get the data into so that it can be imported into our system easily.

2. Sample Jewelry From Vendor.CSV is a sample file of simple products that should be transformed to be configurable based on metal type, metal color, and stone carat weight.

3. Watch File.csv Is a file of watches to be turned into simple product uploads

4. All\_Attributes(5069)\_20250530.xlsx This is the attribute mapping export from JC 2.0 this is used to standardized the data found in various columns of the source data file. We would like the AI to have a try at mapping these and then let a human review and correct then save the mapping.

5. Lashbrook.csv is the data file that works with the mapping file from point 4. You can use this to better understand how the mapping works.

<https://docs.google.com/spreadsheets/d/1vPRTVphG_2QrMYuNaYynKnvir50uwB3Tz91O8hZiYKs/edit?gid=326477473#gid=326477473>

<https://docs.google.com/spreadsheets/d/19Ue5WN2nxHuw1J5VR2xDTr34Gsxhq-V-KP20B81CQYs/edit?gid=1280665202#gid=1280665202>

<https://docs.google.com/spreadsheets/d/1il6sTa2pr-bGyzq3q5BlKrpcqrwe5lfZ9KOrCW4vzAM/edit?gid=430068885#gid=430068885>

<https://docs.google.com/spreadsheets/d/1XIpnyLlkJpq_PY6G0fWFp7D3gZgd4d4Q7Sr97-ynGUw/edit?gid=20953486#gid=20953486>

<https://docs.google.com/spreadsheets/d/1_-9VqVemWseZXiEiXVuXxqIX4wOLo3l5Hhj5UnNy3rE/edit?gid=1368149032#gid=1368149032>

<https://www.figma.com/design/SBi6daYXvPtq9TXW3YRsg4/Untitled?node-id=0-1&p=f&t=pKVlGyv5JSggw3hI-0>

<https://www.loom.com/share/6261fea7fc524994b8ef4a3249765ccf>

### **🔷 PART 1: Transforming Simple Products into Configurable Products**

| **Step** | **Description** |
| --- | --- |
| **1. Identify Grouping Fields** | Based on uploaded data, we identify fields like MetalType, MetalColor, CaratWeight, etc., that are candidates for variant grouping. |
| **2. Allow Human Selection of Variant Keys** | The system will allow the user to choose which fields to use to group SKUs (e.g., Metal Type + Color). |
| **3. Group SKUs Under Parent** | Rows with the same base style (e.g., StyleNumber prefix or grouped field match) are merged into **one parent SKU** with child variants. |
| **4. Enforce Variant Rules** | We check if a group has more than 100 variants → allow human to exclude values (e.g., skip "10k" metal) or auto-separate. |
| **5. Output Configurable File** | Final export is a JC-compliant file structured to support parent-child SKU relationships, ready for JC 2.0 import. |

### 

### 

### 

### 

### **🔷 PART 2: Creating the Mapping File for Unique Value Normalization (Universal Mapping Interface)**

| **Step** | **Description** |
| --- | --- |
| **1. Column Value Scan** | After column mapping, we scan each mapped column (e.g., MetalColor, Clarity, etc.) for **unique values** found in the file. |
| **2. Compare to JC Standard Attributes** | Values are checked against **All\_Attributes(5069).xlsx**. If the value is not a direct match, the AI attempts to suggest one. |
| **3. Human Review Panel** | Users can review and override any AI-suggested mapping (e.g., "Y-Gold" → "Yellow Gold"). |
| **4. Save Mapping File** | Once confirmed, the mapping is saved and reused automatically for future uploads from the same vendor/client. |
| **5. Export Mapping for JC** | We generate a .xlsx or .csv mapping file ready to be used by JC to **standardize vendor-specific values** across the system. |

### **🧠 Additional Notes (For Discussion):**

* We’ll also cover how the system determines which products are **simple vs. configurable**
* We will go over the **logic and constraints for grouping**, including:  
  + Variant exclusions
  + Shopify Plus flag (to bypass 100-variant limit)

We need to maintain log file behind the scene, to make sure things dont get messed.

Screens:

* ~~Splash~~
* ~~Login~~
* ~~Signup~~
* ~~Forgot password~~
* ~~Side tabs~~
  + ~~Dashboard~~
  + ~~AI Mapping~~
  + ~~All files~~
  + ~~Settings~~
* ~~File upload~~
  + ~~File upload~~
  + ~~AI Mapping~~
  + ~~Manual Editing~~
  + ~~Final previeve- Save and export button~~
  + ~~Push to gemtext api~~
* All files:
* Settings:
* Dashboard
  + ~~Total vendors~~
  + ~~Total files mapped~~
  + ~~Total exported files~~

Files understanding:

## 

## 

## 

## 

## 

## **📁 1. JC Data Form - Configurable Product Template.xlsx**

### **🔹 Purpose:**

This is the **final export format** that the AI system must generate. It’s the **target schema** for all product uploads — whether from watches, Lashbrook rings, or vendor files.

### **🔹 How It Will Be Used in the Project:**

| **Use Case** | **Description** |
| --- | --- |
| ✅ **Export Structure Validation** | At export stage, ensure all fields match this template (column names, order, types) |
| ✅ **UI Template Upload (Admin Setting)** | Admins can upload this template in the **Settings Page**, so the system knows what format to align to |
| ✅ **Real-Time Column Mapping Check** | During AI column mapping, fields will be checked against this template for validity |
| ✅ **Final Download Format** | The output .xlsx or .csv will match this structure for JC import |

## **📁 2. Sample Jewelry From Vendor.csv**

### **🔹 Purpose:**

A sample of **simple product rows** (e.g., individual rings) meant to be **transformed into configurable products** based on **attributes** like:

* Metal Type
* Metal Color
* Stone Carat Weight

### **🔹 How It Will Be Used in the Project:**

| **Use Case** | **Description** |
| --- | --- |
| ✅ **AI Mapping + Grouping Logic** | The system will detect products with the same base style and group them as **variants** |
| ✅ **Used in Configurable Product Generator** | This file is the primary **test case** for building product trees under a parent SKU |
| ✅ **Human Variant Control** | Users can exclude some variants (e.g., remove “10k gold” to meet 100 variant limit) |
| ✅ **Template Compliance Test** | After grouping, the output should still match JC Data Form.xlsx |

## **📁 3. Watch File.csv**

### **🔹 Purpose:**

A real-world **non-configurable file** (e.g., standalone watch SKUs) to test **simple product workflows**.

### **🔹 How It Will Be Used in the Project:**

| **Use Case** | **Description** |
| --- | --- |
| ✅ **File Classification Test** | AI detects it's a non-groupable file (watches usually aren’t configurable) |
| ✅ **AI Field Mapping** | Map headers like MetalColor, Band Type, etc., to JC fields |
| ✅ **Optional AI Title/Desc** | If enabled, GemText will generate content; otherwise use existing values |
| ✅ **Static Column Testing** | PriceType, IsImageFromURL are already included and will be validated during export |
| ✅ **Simple Product Export Path** | Final file follows JC template but **skips configurable logic** |

## **📁 4. All\_Attributes(5069)\_20250530.xlsx**

### **🔹 Purpose:**

This is the **master attribute reference sheet** from JC 2.0, used to **normalize values** across vendor data. Think of this as the **Universal Mapping Dictionary**.

### **🔹 How It Will Be Used in the Project:**

| **Use Case** | **Description** |
| --- | --- |
| ✅ **Universal Value Mapping** | For columns like MetalColor, Clarity, StoneType, this sheet provides the **approved JC values** |
| ✅ **AI Training Reference** | AI will learn mappings like: “YG”, “Y-Gold”, “Yellow” → “Yellow Gold” |
| ✅ **Human-in-the-Loop Review Panel** | After AI suggestion, human confirms or corrects using this file’s values |
| ✅ **Memory Storage** | Confirmed mappings per client are stored and reused in future files |
| ✅ **Required in the Settings Page** | Admin can upload/update this mapping sheet centrally |

## **📁 5. Lashbrook.csv**

### **🔹 Purpose:**

This is a **real vendor file** that closely **aligns with the All\_Attributes file** — used to test **attribute normalization and AI mapping accuracy**.

### **🔹 How It Will Be Used in the Project:**

| **Use Case** | **Description** |
| --- | --- |
| ✅ **Test Case for Mapping Engine** | Map fields like MetalType, MetalColor, Gender, PriceType using All\_Attributes sheet |
| ✅ **Mapping History Demonstration** | After this file is processed and corrected, system will remember Lashbrook mappings |
| ✅ **GemText Integration Scenario** | Fields like ProductName, MetalType, Width used for title/desc generation |
| ✅ **AI + Python Code Traceability** | If deterministic AI logic is enabled, the Python code used to process this file can be reviewed/debugged by developers/admins |
| ✅ **Mapping File Import** | If a previous Lashbrook mapping exists, it can be imported and reused here |

## **📦 How These Files Work Together**

| **File Name** | **Role in Workflow** | **System Modules Involved** |
| --- | --- | --- |
| JC Data Form.xlsx | Target Format | Export Validation, Download Center |
| Sample Jewelry.csv | Grouping Logic | Configurable Product Generator, Mapping |
| Watch File.csv | Simple Product Flow | AI Mapping, Static Fields, Export |
| All\_Attributes.xlsx | Universal Value Reference | Value Normalization, AI Suggestions |
| Lashbrook.csv | Mapping Accuracy Test | Memory-Based Mapping, AI Training, GemText, Python Code Generation |

## **🧩 System Features Triggered by These Files**

| **Feature** | **Activated By** |
| --- | --- |
| ✅ Configurable Grouping | Sample Jewelry |
| ✅ Static Columns UI | Watch File |
| ✅ Universal Mapping UI | All\_Attributes |
| ✅ Human-in-the-loop Override | Lashbrook |
| ✅ AI Skipping Toggle | All vendor files |
| ✅ Python Code Generation | Lashbrook, Watch, Jewelry |
| ✅ Template Upload | JC Data Form.xlsx |
| ✅ Mapping Memory & History | Lashbrook & All Vendors |

## **✅ Conclusion**

These files cover **every use case** the system needs to support:

* Simple and complex products
* Mapping and value normalization
* Static fields, optional AI processing
* Template validation, human review
* AI transformation using Python (deterministic)

—---------------------------

## **1. Project Overview**

GemFind requires a smart AI Agent to ingest vendor-specific data files (CSV, XLS, TXT, JSON, TSV), intelligently map fields to a standardized schema, and transform flat product data into JC 2.0-ready configurable formats. A Universal Mapping Interface will ensure consistent tagging and classification of values like "metal color." This system must allow manual overrides, remember vendor preferences, integrate GemText AI for description generation, and support export in JC-compliant formats.

The platform will run on GemFind’s .NET Azure infrastructure and integrate with platform.gemfind.net.

## **2. Extended Functional Requirements**

### **✅ Task 1: File Upload**

* **Inputs**: File (.csv, .xls, .tsv, .json, .txt), Client Number
* **Validations**:  
  + Max file size: 50MB
  + Accepted formats: .csv, .xls, .json, .tsv, |-delimited .txt
  + Client number must be selected at upload
* **Behavior**:  
  + Upload via drag-and-drop UI
  + Temporarily stored and queued for AI preprocessing
* **API**: POST /upload

### **✅ Task 2: AI Column Mapping**

* **Inputs**: Vendor file headers, JC 2.0 schema, client\_number
* **Validations**:  
  + 75% match = auto-map
  + <75% = mark for human review
  + Use All\_Attributes(5069) as baseline (as seen in Mapping.xlsx)
* **Behavior**:  
  + Suggest mapping
  + Show unmapped fields for human review
  + Mandatory fields must be mapped
* **API**:  
  + POST /mapping/ai-suggested
  + GET /mapping/history/{client\_number}

### **✅ Task 3: Human-in-the-Loop Mapping Review**

* **Inputs**: Manual overrides, client\_number
* **Rules**:  
  + Manual override allowed at any confidence level
  + Edits saved for future reuse
  + Keep history of last 10 mappings per client
* **Behavior**:  
  + Editable UI table with dropdown overrides
  + Save, reset, and rollback previous mappings
* **API**: POST /mapping/save

### **✅ Task 4: Universal Mapping Interface (NEW)**

* **Purpose**: Evaluate unique column values and normalize them (e.g., YG → Yellow)
* **Output**: Cleaned .xlsx file ready for JC import
* **Behavior**:  
  + Apply mapping rules from Mapping.xlsx
  + Consistently tag values (e.g., stone types, metal colors)

### **✅ Task 5: Configurable Product Generator**

* **Inputs**: Human-selected variant fields (e.g., metal type, size)
* **Rules**:  
  + Must validate JC template compatibility
  + Variant limit: 100 (toggleable for Shopify Plus)
  + Exclude variants to meet limit
* **Output**:  
  + Configurable product tree (SKU → children)
* **API**: POST /product/configure

### **✅ Skip AI Product Title/Description Generation**

### **Users will have the option to enable or skip title/description generation for any upload session.**

### **If skipped, the process continues with all other data transformation and output preparation.**

### 

### **✅ Add Static Columns to Output**

### **Users can dynamically add one or more columns to the final export.**

### **Each column will be assigned a fixed value (e.g., "PriceType" = "Retail", "IsImageFromURL" = true) to every row.**

### 

### **✅ Deterministic AI via Python Code Generation**

### **To ensure predictable, transparent transformations:**

### **The AI Agent will generate Python code to perform all data normalization, mapping, and transformation tasks.**

### **This code will be executed within the system, logged, and optionally visible to admins.**

### **Avoids model drift or variation from prompt-only interactions.**

### 

### **✅ Task 6: GemText API Integration**

* **Inputs**: Post-mapping data, API Key, GemText Client + Prompt
* **Rules**:  
  + Reuse generated titles/descriptions for known SKUs
  + Handle rate limits & token expiry
* **Behavior**:  
  + Calls https://gemtext.jewelcloud.com/generate
  + Skips regeneration for previously uploaded SKUs
* **Preview**: Result modal showing new AI-generated content

### **✅ Task 7: Output Preview & Export**

* **Inputs**: Final processed dataset
* **Behavior**:  
  + Preview table with download option
  + Must match JC import template
* **API**: GET /export/final.csv

### **✅ Task 8: Audit Trail & Vendor Profile Management**

* **Inputs**: Client\_number, mapping logs
* **Behavior**:  
  + Track changes
  + Allow restore/delete of previous configurations
* **API**:  
  + GET /audit/{client\_number}
  + DELETE /mapping/{client\_number}

## 

## 

## 

## 

## 

## 

## 

## **3. System Architecture**

[React UI - Next.js]

|

↓

[FastAPI Backend - Python]

|

↓

[OpenAI Agent SDK] + [Pinecone]

|

↓

[PostgreSQL] & [.NET Azure Infrastructure]

|

↓

[GemText API]

* **Frontend**: React (Next.js) + Tailwind + TypeScript
* **Backend**: FastAPI Microservices → REST with .NET interoperability
* **AI Agent**: OpenAI Agents SDK + Pinecone for embeddings & learning corrections
* **Database**: PostgreSQL + Pinecone Vector DB
* **Hosting**: Azure Functions + Azure Blob + App Services

## **4. Project Deliverables**

| **Deliverable** | **Description** |
| --- | --- |
| **2 days**  **Figma Design** | Complete wireframes and UI flows for all modules including upload, mapping interface, configuration panel, and export screens |
| **2 days**  **UI/UX Design** | Fully responsive frontend using Tailwind + React (Next.js), consistent with GemFind brand guidelines |
| **2 days**  **Upload Working** | Vendor file upload with client assignment, format validation (.csv, .xls, .json, .tsv, .txt). |
| **2 days**  **Mapping Working** | AI-powered column detection, memory-based auto-mapping using OpenAI Agents SDK + Pinecone, and mapping history per client |
| **2 days**  **Manual Editing + GemText API Integration** | Human override interface with dropdowns, edit/save/rollback capability, and GemText AI integration for auto-generating titles and descriptions |

—--------------------

3-7-2025:

* Color change according to company color.
* Update logo.