**Instructions for the Freelancer: Functionality Implementation (Before UI Design)**

**Objective**

Develop the core functionality of the app before designing the user interface. The app should allow users to input an item through **scanning a barcode** or **manually entering an ITEM\_NO or VEND\_ITEM\_NO**. The app will then fetch item details from **two tables**:

1. **IM\_BARCOD** (contains all barcodes)
2. **VI\_IM\_ITEM\_WITH\_INV\_AND\_INV\_TOTS** (contains most item details, including inventory across locations)

**Step 1: Input Handling**

* **Default Focus**: Ensure the input field remains highlighted at all times so users don’t need to tap before scanning.
* **Input Methods**:
  1. **Barcode Scanner** (Automatically fills the input field)
  2. **Manual Entry** of ITEM\_NO or VEND\_ITEM\_NO
  3. **Manual Entry of Barcode**
* **Triggering Search**:
  1. If scanning, execute the search immediately.
  2. If entering manually, execute the search when the user presses the **Search button**.

**Step 2: Querying the Database**

* **Check where the input exists:**
  1. **If the input is a barcode**:
     + Look for it in IM\_BARCOD (BARCOD field).
     + If found, get the corresponding ITEM\_NO to fetch details from VI\_IM\_ITEM\_WITH\_INV\_AND\_INV\_TOTS.
  2. **If the input is an ITEM\_NO or VEND\_ITEM\_NO**:
     + Search directly in VI\_IM\_ITEM\_WITH\_INV\_AND\_INV\_TOTS (ITEM\_NO or VEND\_ITEM\_NO field).
  3. **If the item does not exist in either table**, return an error message:  
     "Item not found. Please check the barcode, ITEM\_NO, or VEND\_ITEM\_NO and try again."

**Step 3: Retrieving and Processing Data**

* **Once an ITEM\_NO is found**, retrieve its details from VI\_IM\_ITEM\_WITH\_INV\_AND\_INV\_TOTS.
* **Filter by LOC\_ID**:
  + Use only **LOC\_ID values** 1-STM, 2-ZEE, and WH1.
  + Ignore records with LOC\_ID = WH2.
* **Extract relevant item information**:
  + Some fields (e.g., DESCR, CATEGORY, SUBCATEGORY, PRC\_1) are the same across all locations.
  + Inventory-related fields (MIN\_QTY, MAX\_QTY, BIN\_1, QTY\_AVAIL, QTY\_ON\_HND) differ by location.

**Step 4: Structuring the Data for Display**

* Organize the fetched data into a **structured JSON response** for the front-end.
* Example output:

json

CopyEdit

{

"item\_no": "12345",

"description": "Sample Item",

"category": "Electronics",

"subcategory": "Mobile Phones",

"price": 99.99,

"locations": {

"1-STM": {

"min\_qty": 5,

"max\_qty": 50,

"bin": "A1-12",

"qty\_avail": 20,

"qty\_on\_hand": 25

},

"2-ZEE": {

"min\_qty": 3,

"max\_qty": 30,

"bin": "B2-10",

"qty\_avail": 15,

"qty\_on\_hand": 18

},

"WH1": {

"min\_qty": 10,

"max\_qty": 100,

"bin": "C3-5",

"qty\_avail": 50,

"qty\_on\_hand": 55

}

}

}

**Step 5: Error Handling**

* If the barcode, ITEM\_NO, or VEND\_ITEM\_NO is **not found**, return:

json

CopyEdit

{ "error": "Item not found. Please check your input and try again." }

* If the database **connection fails**, return:

json

CopyEdit

{ "error": "Database connection failed. Please try again later." }

* If the query **returns multiple matches**, return:

json

CopyEdit

{ "error": "Multiple matches found. Please refine your search." }

**Step 6: Next Steps**

* Once functionality is tested and verified, we will define which fields to display in the front-end UI.

**Additional Notes**

* Ensure the API responds quickly (<1 second) to user input.
* Keep all queries **optimized** to prevent database performance issues.
* Implement **logging** to capture any failed searches for debugging.