YHere is the **Updated Roadmap** with specific **Verification & Validation (V&V) Instructions** added to every critical step.

The "Golden Rule" for this Project

Add this instruction to your **Custom Instructions** or **System Prompt** for the Agent:

> **"After completing every coding task, you must strictly perform a 'Verification Step'. This means: 1) Run the code/server to ensure it starts without errors. 2) Create a small test script or use curl to verify the specific feature you just built works as expected. 3) Only proceed if the verification passes."**

Phase 1: Project Initialization & Infrastructure

Step 1: Scaffolding

- **Action:** Create directories and files.
- 🔍 **Validation Prompt:**

  > "Run ls -R to list the recursive directory structure. Verify that backend/, frontend/, deployment/, and config.yaml exist. Read config.yaml to ensure it exactly matches the template in the spec."

Step 2: Backend Setup

- **Action:** Install dependencies and create main.py.
- 🔍 **Validation Prompt:**

  > "Activate the python environment and run uvicorn main:app --reload.
  >
  > 1. Verify the server starts on port 8000.
  > 2. Send a curl request to http://127.0.0.1:8000/ (or your health check endpoint) and confirm you get a 200 OK response.
  > 3. Check database.py imports to ensure SQLAlchemy drivers are installed correctly."

Step 3: Frontend Setup

- **Action:** Vite init and Tailwind setup.
- 🔍 **Validation Prompt:**

"Run npm run dev in the frontend folder.

1. Use curl or a browser check to confirm http://localhost:5173 is serving the default Vite page.
2. Create a temporary Test.jsx component with a text-deep-blue class (from your config) to verify Tailwind is compiling custom colors correctly."

Phase 2: Backend Core (Database & Auth)

Step 4 & 5: Database Models & Logic

- **Action:** Create Models and CRUD functions.
- 🔍 **Validation Prompt:**

  "Create a temporary script test_db.py.

  1. In this script, call create_user('test_admin') and then get_user_by_name('test_admin').
  2. Assert that the returned object has the correct username and ID.
  3. Run this script and confirm the database file (sqlite) is created and the data is retrievable."

Step 6: Authentication System

- **Action:** JWT and Login.
- 🔍 **Validation Prompt:**

  "Create a test script test_auth.py.

  1. Register a user programmatically.
  2. Send a POST request to /token with the correct password. Verify you receive a access_token.
  3. Send a POST request with a *wrong* password. Verify you receive a 401 Unauthorized.

4. Use the received token to access a protected endpoint (create a dummy one if needed) and verify access is granted."

Phase 3: Frontend Foundation

Step 7 & 8: Login Page & State

- **Action:** React Login Page and Zustand Store.
- 🔍 **Validation Prompt:**

  "Start both Backend and Frontend.

  1. Navigate to the Login Page in the browser (or simulate it).
  2. Enter the credentials of the user created in Step 6.
  3. Click 'Log In'. Check the browser console or Zustand DevTools to verify that the token is stored in the state.
  4. Verify the user is redirected away from the login page upon success."

Phase 4: Purchase Management

Step 10: Backend Items Logic

- **Action:** Item CRUD and Friendly Name logic.
- 🔍 **Validation Prompt:**

  "Create test_items.py.

  1. Create a purchase, then add an item to it using add_item_to_purchase.

     2. Verify the friendly_name logic: Insert an item name 'Milk Frsh Alpine' and ensure the mapping logic generates/retrieves the correct friendly name as per Section 9.11.

  2. Verify that adding an item with 'Category 2' but no 'Category 1' raises a validation error or handles it as specified."

Step 12: Purchase Creation Logic (Full Stack)

- **Action:** Connect Frontend Form to Backend.
- 🔍 **Validation Prompt:**

  "Perform an End-to-End test:

  1. Login to the frontend.
  2. Go to 'Create Purchase'. Fill in 'Test Purchase', Date, and add 1 Item ('Milk', 2.00 EUR).
  3. Click Confirm.
  4. Check the backend database: Does the purchases table have the new record? Does the items table have 'Milk'?
  5. **Regression Check:** Logout and Login again to ensure Auth is still working."

Phase 5: Receipt Scanning

Step 15: AI Analysis (Mistral)

- **Action:** OCR + LLM Extraction.
- 🔍 **Validation Prompt:**

  "Create a test_ocr.py script.

  1. Use a sample receipt image (place one in tests/fixtures/).
  2. Call the analyze_information function directly with mock text or the actual API.
  3. Verify the output is valid JSON and contains a list of items with prices.
  4. Ensure the system handles API failures gracefully (e.g., returns an error message instead of crashing)."

Phase 6: Advanced Features

Step 19: User Management (Admin)

- **Action:** Admin panel and permissions.
- 🔍 **Validation Prompt:**

  > "Test Access Control2:
  >
  > 1. Login as a **Standard User**. Try to access the /users endpoint (via curl). Verify you get a 403 Forbidden.
  > 2. Login as an **Admin**. Access /users. Verify you get a list of users.
  > 3. As Admin, revoke the admin rights of another user. Verify in the DB that their administrator flag is now False."

Universal Regression Prompt

When to use: Every 3-4 steps, or when you feel the code is getting complex.

> "Run a full Regression Check.
>
> 1. **Backend:** Run all test_*.py scripts we have created so far.
> 2. **Frontend:** Ensure the build passes npm run build.
> 3. **Sanity Check:** Start the full stack (docker-compose up or manually) and verify that I can still Login and view the Dashboard. Report any broken features immediately."