Product Expert System — Windows Setup Guide for Beginners

> This guide assumes you have a Windows 10 or 11 PC and have never used Docker before.
> Every step includes exactly what to click, what to type, and what you should see.

What We're Installing (and Why)

Before we start, here's a plain-English summary of what each piece of software does:

Software	What It Does	Analogy
Docker Desktop	Runs our entire application in isolated "containers" — think of it as a virtual computer inside your computer	Like running an app inside a sealed sandbox
Git	Downloads our project code and tracks changes	Like a smart version of "Download ZIP"
Ollama	Runs AI models locally on your machine — no cloud API needed	Like having a mini ChatGPT on your laptop
VS Code (optional)	A text editor for viewing and editing project files	Like Notepad, but much better for code

Everything else (PostgreSQL database, Redis cache, Nginx web server, Python, FastAPI) **runs inside Docker automatically.** You don't install them separately.

Total cost: $0. Everything is free for our use.

PHASE 1: Install Required Software

Step 1: Check Your Windows Version

1. Press **Windows key + R** on your keyboard
2. Type `winver` and press **Enter**
3. A window pops up showing your Windows version

You need: Windows 10 version 2004 or higher, OR Windows 11.

If you're on an older version, update Windows first:
Settings → Windows Update → Check for updates.

Step 2: Enable WSL 2 (Windows Subsystem for Linux)

Docker needs WSL 2 to run Linux containers on Windows. This is a one-time setup.

1. Click the **Start** button (Windows icon, bottom-left)
2. Type **PowerShell**
3. Right-click **Windows PowerShell** and choose **Run as Administrator**
4. Click **Yes** when the "Do you want to allow this app..." popup appears
5. In the blue PowerShell window, type this command and press **Enter**:

```
powershell

wsl --install
```

6. **Wait** — this downloads and installs the Linux subsystem. It may take 5–10 minutes.
7. You'll see messages like:

```
Installing: Virtual Machine Platform
Installing: Windows Subsystem for Linux
Installing: Ubuntu
```

8. When it says **"The requested operation is successful"**, **restart your computer**.
9. After restart, a terminal window may pop up asking you to create a Linux username and password. You can set these to anything simple (e.g., username: `user`, password: `password`). This is just for the Linux subsystem, not your Windows account.
10. **Verify it worked.** Open PowerShell again (regular, not admin) and type:

```
powershell

wsl --version
```

You should see something like:

```
WSL version: 2.x.x
Kernel version: 5.15.x
```

> **If you see an error:** Go to Settings → Apps → Optional Features → More Windows Features → check both "Virtual Machine Platform" and "Windows Subsystem for Linux" → Restart.

Step 3: Install Docker Desktop

1. Open your web browser and go to: **https://www.docker.com/products/docker-desktop/**
2. Click the big blue **"Download for Windows"** button
3. Open the downloaded file: **Docker Desktop Installer.exe** (Usually in your Downloads folder — about 550 MB)
4. In the installer window:
   - ✅ Check **"Use WSL 2 instead of Hyper-V"** (should be checked by default)
   - ✅ Check **"Add shortcut to desktop"**
   - Click **OK**
5. **Wait** for the installation to complete (3–5 minutes)
6. Click **"Close and restart"** when prompted
7. After restart, **Docker Desktop** should start automatically.
   - You'll see a whale icon 🐳 in the system tray (bottom-right of your taskbar, near the clock)
   - The whale will be animated while Docker is starting up
   - **Wait until the whale stops moving** — this means Docker is ready
8. If you see a "Docker Subscription Service Agreement" popup:
   - Click **Accept** (it's free for personal and small business use)
9. You may see a "Welcome to Docker Desktop" tutorial — you can **Skip** it.
10. **Verify it worked.** Open PowerShell and type:

```
powershell

docker --version
```

You should see:

```
Docker version 27.x.x, build xxxxxxx
```

Then type:

```
powershell

docker compose version
```

You should see:

```
Docker Compose version v2.x.x
```

> **If Docker won't start:** Make sure WSL 2 is installed (Step 2). Open Docker Desktop → Settings (gear icon) → General → check "Use the WSL 2 based engine" → Apply & Restart.

Step 4: Install Git

1. Open your web browser and go to: **https://git-scm.com/download/win**
2. The download should start automatically. If not, click **"Click here to download manually"**
3. Run the installer: **Git-2.x.x-64-bit.exe**
4. Click through the installer — **use all default options.** Just keep clicking **Next → Next → Next → Install**
   Important defaults to keep:
   - ✅ "Git from the command line and also from 3rd-party software"
   - ✅ "Use bundled OpenSSH"
   - ✅ "Checkout as-is, commit Unix-style line endings" ← **change this one if you see it!** Select **"Checkout as-is, commit as-is"** to avoid line ending issues.
5. Click **Finish**
6. **Close and reopen PowerShell** (so it picks up the new Git installation)
7. **Verify it worked:**

```
powershell

git --version
```

You should see:

```
git version 2.x.x.windows.x
```

Step 5: Install Ollama (Local AI Engine)

Ollama runs the AI models that power our system's search and Q&A features.

1. Open your web browser and go to: **https://ollama.com/download/windows**
2. Click **"Download for Windows"**
3. Run the installer: **OllamaSetup.exe**
    - Follow the prompts, click **Install**, then **Finish**
4. Ollama starts automatically as a background service. You'll see a llama icon in your system tray (bottom-right).
5. **Verify it's running.** Open PowerShell and type:

```
powershell

ollama --version
```

You should see:

```
ollama version 0.x.x
```

6. **Download the AI models we need.** Type each command and wait for it to finish:

```
powershell

# This downloads the embedding model (~275 MB) — used for search
ollama pull nomic-embed-text
```

Wait for it to show `success`. Then:

```
powershell

# This downloads the language model (~4.7 GB) — used for Q&A answers
ollama pull llama3.1:8b
```

This one takes a while (5–15 minutes depending on your internet speed). Wait for `success`.

7. **Verify both models are installed:**

```
powershell
```

```
ollama list
```

You should see both models listed:

```
NAME                       SIZE
nomic-embed-text:latest    274 MB
llama3.1:8b                4.7 GB
```

8. **Quick test** — make sure the AI is responding:

```powershell
curl.exe http://localhost:11434/api/tags
```

You should see a JSON response listing the models. If you see an error, make sure the llama icon is in your system tray.

Step 6: Install VS Code (Optional but Recommended)

VS Code makes it easy to view and edit project files.

1. Go to: **https://code.visualstudio.com/**
2. Click **"Download for Windows"**
3. Run the installer with default options
4. ✅ Check **"Add to PATH"** during installation

✅ Phase 1 Complete — Checklist

Before continuing, verify everything is installed. Open PowerShell and run each:

```powershell
```

```
wsl --version              # Should show WSL version 2.x
docker --version           # Should show Docker version 27.x
docker compose version     # Should show Docker Compose v2.x
git --version              # Should show git version 2.x
ollama --version           # Should show ollama version 0.x
ollama list                # Should show nomic-embed-text and llama3.1:8b
```

If all 6 commands work, you're ready for Phase 2.

PHASE 2: Set Up the Project

Step 7: Create a Project Folder

1. Open **PowerShell** (regular, not admin)
2. Navigate to where you want the project. For example, your Documents folder:

```
powershell

cd $HOME\Documents
```

3. Create the project folder:

```
powershell

mkdir product-expert
cd product-expert
```

4. You're now at: `C:\Users\YourName\Documents\product-expert`

> **If your code is in a Git repository**, clone it instead:
>
> ```
> powershell
>
> cd $HOME\Documents
> git clone https://github.com/your-org/product-expert.git
> cd product-expert
> ```

Step 8: Place All Project Files

You need to place all the files we've built into this folder. Here's the complete list:

```
product-expert\
├── .env.example
├── Dockerfile
├── docker-compose.yml
├── requirements.txt
├── nginx.conf
├── ops.ps1
├── Makefile
├── db-schema.sql
├── seed.sql
├── models.py
├── extraction_pipeline.py
├── ingestion_orchestrator.py
├── recommendation_engine.py
├── rag_retrieval.py
├── api_layer.py
├── asyncpg_repository.py
├── config.py
├── alembic.ini
├── alembic\
│   ├── env.py
│   └── versions\
│       └── 001_initial_schema.py
└── data\
    └── samples\
        └── (your product PDF/TXT/MD files go here)
```

How to create files from the Claude artifacts:

For each artifact I created:

1. Click the **Copy** button on the artifact
2. Open **Notepad** (or VS Code)
3. Paste the content
4. Save with the exact filename shown above
5. **Important:** When saving in Notepad, change "Save as type" to **"All Files (.)"** so it doesn't add `.txt` to the end

Or in VS Code:

1. Open VS Code
2. File → Open Folder → select `product-expert`
3. Create each file in the file explorer panel on the left

> **Critical:** Make sure files like `Dockerfile` have NO file extension (not `Dockerfile.txt`). And `docker-compose.yml` ends in `.yml`, not `.yml.txt`.

Step 9: Create the Environment Configuration File

1. Open PowerShell in your project folder:

```powershell
cd $HOME\Documents\product-expert
```

2. Copy the template:

```powershell
Copy-Item .env.example .env
```

3. Open the `.env` file for editing:

```powershell
notepad .env
```

4. The defaults work for local development. The key settings are:

```env
```

```
# These are already set correctly for Ollama:
EMBEDDING_PROVIDER=ollama
EMBEDDING_API_URL=http://host.docker.internal:11434/api/embeddings
EMBEDDING_MODEL=nomic-embed-text
LLM_PROVIDER=ollama
LLM_API_URL=http://host.docker.internal:11434/api/generate
LLM_MODEL=llama3.1:8b

# Default password (change for production):
POSTGRES_PASSWORD=expert

# Default API keys for testing:
API_KEYS=dev-key-001:admin,sales-key-001:sales_engineer,pm-key-001:product_manager
```

5. Save and close Notepad (Ctrl+S, then close)

Step 10: Create the Data Folder for Documents

```powershell
# Create the folders for storing product documents
mkdir data
mkdir data\samples
```

If you have product data sheets (PDFs, text files), copy them into `data\samples\` now. If not, you can add them later.

Step 11: Allow PowerShell Scripts to Run

Windows blocks PowerShell scripts by default. We need to allow our `ops.ps1` script:

```powershell
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

Type **Y** and press Enter when asked to confirm.

✅ Phase 2 Complete — Checklist

```powershell
# Make sure you're in the project folder
cd $HOME\Documents\product-expert

# Check key files exist
Test-Path Dockerfile            # Should say True
Test-Path docker-compose.yml    # Should say True
Test-Path .env                  # Should say True
Test-Path ops.ps1               # Should say True
Test-Path db-schema.sql         # Should say True
Test-Path seed.sql              # Should say True
Test-Path api_layer.py          # Should say True
```

If all say True, you're ready for Phase 3.

PHASE 3: Build and Start the Application

Step 12: Make Sure Docker Desktop Is Running

1. Look at your system tray (bottom-right of taskbar, near the clock)
2. You should see the Docker whale icon 🐳
3. If it's not there, open Docker Desktop from the Start Menu
4. **Wait until the whale stops animating** (means Docker is ready)

Also verify Ollama is running (llama icon in system tray). If not:

```powershell
ollama serve
```

Step 13: Build the Docker Containers

This downloads base images and installs all dependencies. **First time takes 5–10 minutes.**

```powershell
```

```
cd $HOME\Documents\product-expert

docker compose build
```

What you'll see:

```
[+] Building 3/3
 => [api] Building...
 => => downloading python:3.12-slim
 => => installing pip packages
 => => copying application files
 => [api] DONE
```

If you see errors:

- `"Cannot connect to the Docker daemon"` → Docker Desktop is not running. Start it and wait for the whale to stop animating.
- `"no such file or directory: Dockerfile"` → You're not in the right folder. Run `cd $HOME\Documents\product-expert`.
- `"failed to solve: python:3.12-slim: error pulling image"` → Check your internet connection.

Step 14: Start All Services

```
powershell

docker compose up -d
```

The `-d` flag means "run in background" (detached mode).

What you'll see:

```
[+] Running 5/5
 ✓ Network product-expert_backend     Created
 ✓ Volume "product-expert_pg_data"    Created
 ✓ Container product-expert-db        Healthy
 ✓ Container product-expert-redis     Started
 ✓ Container product-expert-api       Started
 ✓ Container product-expert-proxy     Started
```

Wait about 30 seconds for everything to fully start up, then check status:

```
powershell

docker compose ps
```

You should see all 4 containers with Status "Up":

```
NAME                  STATUS            PORTS
product-expert-api    Up (healthy)      0.0.0.0:8000->8000/tcp
product-expert-db     Up (healthy)      0.0.0.0:5432->5432/tcp
product-expert-proxy  Up                0.0.0.0:80->80/tcp
product-expert-redis  Up                0.0.0.0:6379->6379/tcp
```

> **If a container shows "Restarting" or "Exit":** Check its logs:
>
> ```
> powershell
>
> docker compose logs api      # Check the API container
> docker compose logs postgres # Check the database
> ```

Step 15: Verify Everything Works

Test 1: Health Check

```
powershell

curl.exe -s http://localhost:8000/health
```

Expected output (something like):

```
json

{"status":"healthy","components":{"database":"connected","redis":"connected"}}
```

> **Important:** Use `curl.exe` (with .exe), not just `curl`. In PowerShell, `curl` is an alias for something else.

Test 2: Check the Database Was Seeded

```
powershell

docker compose exec postgres psql -U expert -d product_expert -c "SELECT code, name FROM brands;"
```

Expected output:

```
  code    |          name
----------+------------------------
 ABS      | American BioTech Supply
 LABRepCo | LABRepCo
 Corepoint | Corepoint Scientific
 Celsius  | °celsius Scientific
 CBS      | CBS / CryoSafe
```

Test 3: API Responds to Requests

```
powershell

curl.exe -s http://localhost:8000/use-cases -H "X-API-Key: dev-key-001"
```

Expected: A JSON list of use cases (vaccine_storage, laboratory_general, etc.)

Test 4: Open in Your Browser

Open your web browser and go to: **http://localhost:8000/docs**

You should see the **FastAPI Swagger UI** — an interactive API documentation page where you can test all endpoints.

🎉 THE SYSTEM IS RUNNING!

You should now have:

- **API** at http://localhost:8000
- **API docs** at http://localhost:8000/docs
- **Frontend** at http://localhost (via Nginx — needs frontend files in `frontend\dist\`)

PHASE 4: Ingest Documents and Test

Step 16: Ingest a Product Document

Place a product PDF in `data\samples\`, then:

```
powershell
```

```
# Ingest a single file
.\ops.ps1 ingest-file data\samples\ABS_Premier_26S_PDS.pdf

# Or ingest all files in the data folder
.\ops.ps1 ingest-sample
```

Expected output:

```
json

{
  "job_id": "a1b2c3d4-...",
  "status": "processing",
  "files_accepted": 1,
  "message": "1 file(s) queued for processing"
}
```

Step 17: Run the Full Smoke Test

```
powershell

.\ops.ps1 smoke-test
```

This runs 6 checks in sequence: health, stats, use-cases, product search, conflicts, and recommendation. Each should return JSON data without errors.

Step 18: Try the Q&A Feature

```
powershell

.\ops.ps1 test-ask "What vaccine refrigerators meet NSF 456?"
```

Step 19: Check for Spec Conflicts

```
powershell

.\ops.ps1 conflicts
```

If there are conflicts (from overlapping data in documents), resolve them:

```
powershell
```

```
# Via the API directly
curl.exe -X PUT http://localhost:8000/conflicts/CONFLICT-ID-HERE -H "X-API-Key: dev-key-001" -H "Content-Type: app
```

PHASE 5: Daily Operations

Starting the System (After a Reboot)

When you restart your computer:

1. **Docker Desktop** should start automatically (check for the whale icon)
   - If not, open it from the Start Menu
2. **Ollama** should start automatically (check for the llama icon)
   - If not, open PowerShell and run `ollama serve`
3. Start the application:

```powershell
cd $HOME\Documents\product-expert
.\ops.ps1 up
```

Stopping the System

```powershell
.\ops.ps1 down
```

Viewing Logs

```powershell
.\ops.ps1 logs        # API logs only
.\ops.ps1 logs-all    # All services
```

Press **Ctrl+C** to stop watching logs.

Common ops.ps1 Commands Quick Reference

```powershell
```

```
.\ops.ps1 help          # Show all commands
.\ops.ps1 up            # Start everything
.\ops.ps1 down          # Stop everything
.\ops.ps1 status        # Check what's running
.\ops.ps1 health        # Quick health check
.\ops.ps1 stats         # System statistics
.\ops.ps1 logs          # View API logs
.\ops.ps1 smoke-test    # Run all tests
.\ops.ps1 ingest-file <path>  # Ingest a document
.\ops.ps1 products      # Search all products
.\ops.ps1 conflicts     # Check spec conflicts
.\ops.ps1 db-shell      # Open database terminal
.\ops.ps1 db-dump       # Backup database
```

Troubleshooting

"docker: command not found" or "not recognized"

→ Docker Desktop is not installed or not in your PATH.
→ Restart PowerShell after installing Docker Desktop.
→ Make sure Docker Desktop is running (whale icon in system tray).

"Cannot connect to the Docker daemon"

→ Docker Desktop is installed but not running.
→ Click the Docker Desktop icon to start it.
→ Wait for the whale to stop animating before running commands.

"curl.exe: command not found"

→ You're on an older Windows version. Install curl:

```
powershell

winget install cURL.cURL
```

"port is already in use" or "address already in use"

Another program is using port 8000, 5432, 6379, or 80.

```
powershell
```

```
# Find what's using port 8000
netstat -ano | findstr :8000

# The last column is the PID. Kill it:
taskkill /PID 12345 /F
```

Common culprits: local PostgreSQL (port 5432), IIS/Apache (port 80), or another Docker project.

Container keeps restarting

```
powershell

# Check what's wrong
docker compose logs api --tail 50

# Common fixes:
# 1. Database not ready yet — wait 30 seconds and check again
# 2. Wrong .env settings — verify POSTGRES_PASSWORD matches
# 3. Port conflict — see above
```

"Ollama connection refused" in API logs

The API container can't reach Ollama on your host machine.

```
powershell

# 1. Make sure Ollama is running
curl.exe http://localhost:11434/api/tags

# 2. Make sure .env has the right URL
# It should be: http://host.docker.internal:11434/api/embeddings
# NOT: http://localhost:11434/api/embeddings
```

"no matching manifest for windows/amd64"

You need to switch Docker to Linux containers: → Right-click the Docker whale in system tray → **"Switch to Linux containers..."**

"File not found" errors during build

Make sure your files don't have hidden `.txt` extensions. In File Explorer: → View tab → check **"File name extensions"** to see real extensions.

Everything was working but now it's broken

```powershell
# Nuclear option: full reset (deletes all data!)
.\ops.ps1 reset
```

What's Running Where (Summary)

After `.\ops.ps1 up`, you have 4 containers running: