DocCompiler.ai

USER GUIDE

Open Webui User Guide

Installation, Usage & Quick Reference

A Cyber Panda Solutions LLC product

Version 3.1 2026-05-13 @ 9bd8425 Generated by DocCompiler.ai

Always-updated version at doccompiler.ai/open-webui/open-webui

Source code analysis by Claude Opus 4.6 | 1M token context

About This Document

What Is This?

This is a comprehensive, source-code-verified **User Guide** for **Open Webui**, generated by DocCompiler.ai using Claude Opus 4.6 with 1M-token context. Every command, configuration option, and behavior documented here was verified against the actual source code.

The Definitive Reference

This document is designed to be the single source of truth for Open Webui. Rather than piecing together scattered README fragments, outdated wiki pages, forum posts, and version-specific screenshots, you can rely on this as a verified, structured reference backed by automated code review.

Optimized for AI Assistants

This document is structured for consumption by AI assistants. Feed this PDF into your preferred LLM for accurate, grounded answers about Open Webui — instead of having the model guess or hallucinate from incomplete context.

Version & Updates

Generated on **2026-05-13** (version **3.1**). This document reflects the source code at commit `9bd8425`. The always-updated version is permanently available at **doccompiler.ai/open-webui/open-webui**. Do not print or download for offline use — this document is updated when the source code changes.

Our Mission

DocCompiler.ai aims to be the definitive wiki for the world's most important open-source projects. Every document is overkill by design: exhaustive, fact-checked, and built to replace the scattered tribal knowledge that slows teams down.

Table of Contents

Open WebUI User Guide

1. Quick Start
   - Prerequisites
   - Fastest path -- Ollama on the same machine
   - Fastest path -- OpenAI API only (no Ollama)
   - Fastest path -- Ollama bundled in the same container
   - First-time setup checklist
2. Installation
   - 2.1 System Requirements
   - 2.2 Docker (recommended)
   - 2.3 Docker Compose
   - 2.4 run-compose.sh convenience script
   - 2.5 pip install (Python package)
   - 2.6 Manual build (development)
   - 2.7 Updating
   - 2.8 Kubernetes / Helm
3. Configuration
   - 3.1 Environment variables
   - 3.2 Admin panel settings
   - 3.3 Authentication setup
4. Core Usage
   - 4.1 Chat interface overview
   - 4.2 Starting a chat
   - 4.3 Model selection
   - 4.4 Message controls
   - 4.5 File and image upload
   - 4.6 Voice mode (STT / TTS)
   - 4.7 Web search in chat
   - 4.8 Code interpreter
   - 4.9 Image generation
   - 4.10 Knowledge bases (RAG)
   - 4.11 Tools and Skills
   - 4.12 Channels
   - 4.13 Notes
   - 4.14 Prompt templates
   - 4.15 Memories
   - 4.16 Folders and chat organization
   - 4.17 Chat sharing and export
   - 4.18 Playground
   - 4.19 Internationalization

4.20 Theme
4.21 Keyboard shortcuts
5. Common Tasks
5.1 Connect a local Ollama instance
5.2 Connect an OpenAI-compatible API
5.3 Pull a new Ollama model
5.4 Set a default model
5.5 Create a custom model (Modelfile)
5.6 Upload a document for Q&A
5.7 Enable web search for a message
5.8 Regenerate a response
5.9 Edit a message
5.10 Continue an incomplete response
5.11 Compare models side by side
5.12 Set a system prompt for a chat
5.13 Use a prompt template
5.14 Search your chat history
5.15 Export a chat to Markdown
5.16 Share a chat publicly
5.17 Create a folder for chats
5.18 Tag a chat
5.19 Pin a chat
5.20 Archive old chats
5.21 Generate an image in chat
5.22 Run Python code in chat
5.23 Add a voice input
5.24 Listen to a response
5.25 Add a memory manually
5.26 Create a knowledge base
5.27 Create a prompt template

Open WebUI User Guide

Open WebUI is a self-hosted, feature-rich web interface for interacting with large language models. It supports Ollama backends and any OpenAI-compatible API, running entirely on your own infrastructure with no data leaving your control unless you configure it to do so. Version 0.8.12.

1. Quick Start

Get Open WebUI running in under five minutes with Docker.

Prerequisites

- Docker Engine 20.10+ and Docker Compose v2
- An Ollama instance running locally, **or** an OpenAI-compatible API key

Fastest path -- Ollama on the same machine

bash

```
docker run -d \
  -p 3000:8080 \
  --add-host=host.docker.internal:host-gateway \
  -v open-webui:/app/backend/data \
  --name open-webui \
  --restart always \
  ghcr.io/open-webui/open-webui:main
```

Open your browser at **http://localhost:3000**. The first account you create becomes the administrator.

> **Tip**
>
> If Ollama is running on `localhost` (default port 11434), the `--add-host` flag above is all you need. The container automatically discovers it at `http://host.docker.internal:11434`.

Fastest path -- OpenAI API only (no Ollama)

bash

```
docker run -d \
  -p 3000:8080 \
  -e OPENAI_API_KEY=sk-... \
  -v open-webui:/app/backend/data \
  --name open-webui \
  --restart always \
  ghcr.io/open-webui/open-webui:main
```

Fastest path -- Ollama bundled in the same container

bash

```
docker run -d \
  -p 3000:8080 \
  --gpus all \
  -v ollama:/root/.ollama \
  -v open-webui:/app/backend/data \
  --name open-webui \
  --restart always \
  ghcr.io/open-webui/open-webui:ollama
```

The `ollama` image tag bundles the Ollama runtime. Omit `--gpus all` if you have no NVIDIA GPU.

First-time setup checklist

1. Browse to `http://localhost:3000`
2. Click **Sign up** and create your admin account
3. In **Settings -> Connections**, confirm Ollama or OpenAI connection status
4. Pull or select a model from the model selector at the top of the chat screen
5. Type your first message and press **Enter**

2. Installation

2.1 System Requirements

Component	Minimum	Recommended
CPU	2 cores	4+ cores
RAM	4 GB	8 GB+
Disk	10 GB	50 GB+ (models are large)
OS	Any Docker-supported	Linux (Ubuntu 22.04+)
Docker	20.10+	Latest stable
Docker Compose	v2.0+	Latest stable
Node.js (dev only)	18.13.0	20.x LTS
npm (dev only)	6.0.0+	Latest stable

> **Important**
> Ollama model storage is separate from Open WebUI storage. A single medium-size LLM (e.g., Llama 3 8B) occupies 4-6 GB. Plan disk accordingly -- `/root/.ollama` or a bind-mounted volume fills quickly.

2.2 Docker (recommended)

Standard install (Ollama on host)

bash

```
docker run -d \
  -p 3000:8080 \
  --add-host=host.docker.internal:host-gateway \
  -e OLLAMA_BASE_URL=http://host.docker.internal:11434 \
  -v open-webui:/app/backend/data \
  --name open-webui \
  --restart always \
  ghcr.io/open-webui/open-webui:main
```

Ollama on a separate server

bash

```
docker run -d \
  -p 3000:8080 \
  -e OLLAMA_BASE_URL=http://192.168.1.50:11434 \
  -v open-webui:/app/backend/data \
  --name open-webui \
  --restart always \
  ghcr.io/open-webui/open-webui:main
```

GPU-accelerated (NVIDIA)

bash

```
docker run -d \
  -p 3000:8080 \
  --gpus all \
  --add-host=host.docker.internal:host-gateway \
  -v open-webui:/app/backend/data \
  --name open-webui \
  --restart always \
  ghcr.io/open-webui/open-webui:cuda
```

> **Warning**
>
> The `--gpus all` flag requires the NVIDIA Container Toolkit installed on the host. Without it, Docker will fail to start the container with a device error. Install it with `apt install nvidia-container-toolkit` and restart Docker.

2.3 Docker Compose

Clone the repository and use the provided compose files:

bash

```
git clone https://github.com/open-webui/open-webui.git
cd open-webui
```

Basic compose (Ollama external)

bash

```
docker compose up -d
```

The default `docker-compose.yaml` starts Open WebUI bound to port 3000 with a named volume for data persistence.

GPU-accelerated compose

bash

```
docker compose -f docker-compose.yaml -f docker-compose.gpu.yaml up -d
```

AMD GPU compose

bash

```
docker compose -f docker-compose.yaml -f docker-compose.amdgpu.yaml up -d
```

Expose Ollama API externally

bash

```
docker compose -f docker-compose.yaml -f docker-compose.api.yaml up -d
```

Bind-mount Ollama data folder

bash

```
docker compose -f docker-compose.yaml -f docker-compose.data.yaml up -d
```

Enable Playwright web scraping

bash

```
docker compose -f docker-compose.yaml -f docker-compose.playwright.yaml up -d
```

Enable OpenTelemetry observability

bash

```
docker compose -f docker-compose.yaml -f docker-compose.otel.yaml up -d
```

2.4 run-compose.sh convenience script

The repository ships `run-compose.sh` for common compose combinations:

bash

```
# GPU with 1 GPU card
./run-compose.sh --enable-gpu

# GPU with specific count
./run-compose.sh --enable-gpu[count=2]

# Expose Ollama API on default port
./run-compose.sh --enable-api

# Expose Ollama API on custom port
./run-compose.sh --enable-api[port=11435]

# Custom WebUI port
./run-compose.sh --webui[port=8080]

# Bind-mount data folder
```

```
./run-compose.sh --data[folder=/mnt/ollama-data]

# Enable Playwright
./run-compose.sh --playwright

# Build images locally instead of pulling
./run-compose.sh --build

# Tear down all containers and volumes
./run-compose.sh --drop

# Suppress output
./run-compose.sh -q --enable-gpu
```

Options can be combined:

bash

```
./run-compose.sh --enable-gpu[count=1] --enable-api[port=11434] --webui[port=3000] --data[folder=/data/olla
```

2.5 pip install (Python package)

Open WebUI is published on PyPI and can be installed directly:

bash

```
pip install open-webui
```

Start the server:

bash

```
open-webui serve
```

Start with custom host and port:

bash

```
open-webui serve --host 127.0.0.1 --port 8080
```

Check the installed version:

bash

```
open-webui --version
```

Run the development server (hot reload):

bash

```
open-webui dev
```

> **Note**
> The pip install bundles a pre-built frontend. For local development with live frontend changes, follow the manual build path in section 2.6.

2.6 Manual build (development)

Requirements: Node.js ≥18.13.0 ≤22.x.x, npm ≥6.0.0, Python 3.11+

bash

```
git clone https://github.com/open-webui/open-webui.git
cd open-webui

# Copy environment template
cp .env.example .env

# Install frontend dependencies and build
npm install
npm run build

# Install Python backend
pip install -e .

# Start
open-webui serve
```

For frontend development with hot reload:

bash

```
# Terminal 1 -- backend
open-webui serve

# Terminal 2 -- frontend dev server (proxies API to backend)
npm run dev
```

Frontend dev server runs on `http://localhost:5173` by default.

2.7 Updating

Docker named volume

bash

```
docker pull ghcr.io/open-webui/open-webui:main
docker stop open-webui
docker rm open-webui
# Re-run the original docker run command
```

Your data volume (`open-webui`) persists across container recreation.

Docker Compose

bash

```
docker compose pull
docker compose up -d
```

pip

bash

```
pip install --upgrade open-webui
```

> **Warning**
>
> Always back up `/app/backend/data` (or the Docker volume) before upgrading across major versions. Database migrations run automatically on startup but are not reversible.

2.8 Kubernetes / Helm

A community Helm chart is available. Refer to the official Open WebUI documentation for Helm values reference, as chart configuration is community-maintained and changes independently of the core application.

3. Configuration

3.1 Environment variables

Open WebUI is configured primarily through environment variables. Pass them via `-e` in `docker run`, in a `.env` file for Docker Compose, or in your shell environment for pip installs.

Core server settings

Variable	Default	Description
`WEBUI_SECRET_KEY`	auto-generated	JWT signing secret. Set explicitly in production to survive restarts.
`WEBUI_NAME`	`Open WebUI`	Instance name shown in the UI header and browser title.
`WEBUI_AUTH`	`True`	Set to `False` to disable authentication entirely (single-user, trusted network only).
`ENABLE_SIGNUP`	`True`	Allow new users to self-register. Set `False` after initial setup to lock registration.
`DEFAULT_USER_ROLE`	`pending`	Role assigned to new registrations: `pending`, `user`, or `admin`.

Backend connections

Variable	Default	Description
`OLLAMA_BASE_URL`	derived from `OLLAMA_API_BASE_URL`	Ollama server URL. Overrides the derived value.
`OLLAMA_API_BASE_URL`	`http://localhost:11434/api`	Base for Ollama API discovery.
`ENABLE_OLLAMA_API`	`True`	Toggle Ollama backend on/off.
`OPENAI_API_KEY`	--	API key for OpenAI or compatible provider.
`OPENAI_API_BASE_URL`	`https://api.openai.com/v1`	Override to point to any OpenAI-compatible endpoint.
`ENABLE_OPENAI_API`	`True`	Toggle OpenAI backend on/off.

Image generation

Variable	Default	Description
`ENABLE_IMAGE_GENERATION`	`False`	Enable the image generation feature.
`IMAGE_GENERATION_ENGINE`	`openai`	Engine: `openai`, `automatic1111`, `comfyui`, `gemini`.

Web search

Variable	Default	Description
`ENABLE_WEB_SEARCH`	`False`	Enable web search in chat.
`WEB_SEARCH_ENGINE`	--	Search engine backend (e.g., `searxng`, `google`, `bing`, `brave`, `duckduckgo`).

Code execution

Variable	Default	Description
`ENABLE_CODE_EXECUTION`	`True`	Enable in-browser code execution via Pyodide.
`ENABLE_CODE_INTERPRETER`	`True`	Enable the code interpreter feature in chat.

Speech (STT / TTS)

Variable	Default	Description
`AUDIO_STT_ENGINE`	--	Speech-to-text engine (`openai`, `whisper`, etc.).
`AUDIO_TTS_ENGINE`	--	Text-to-speech engine (`openai`, `elevenlabs`, etc.).

RAG / Knowledge

Variable	Default	Description
`RAG_EMBEDDING_MODEL`	`sentence-transformers/all-MiniLM-L6-v2`	Sentence transformer model used for document embedding.

3.2 Admin panel settings

Most runtime configuration is available through **Settings** (gear icon, top-right) when logged in as an admin. These settings persist in the database and override environment variable defaults where applicable.

Settings -> General

- Instance name override

- Default model for new chats
- System prompt defaults
- Default language and theme

Settings -> Connections

- Add, edit, remove Ollama server endpoints
- Add, edit, remove OpenAI-compatible API endpoints (API key, base URL)
- Connection status indicators (green/red dot)

Settings -> Models

- View all models available from connected backends
- Pull new models from Ollama directly in the UI
- Delete models from Ollama
- Set model visibility (hide from non-admin users)
- Model capabilities tags

Settings -> Users

- List all user accounts
- Change roles: `pending` -> `user` -> `admin`
- Suspend or delete accounts
- View API keys

Settings -> Documents (RAG)

- Embedding model selection
- Chunk size and overlap
- Top-K retrieval count
- PDF extraction engine (pdfminer, pypdf)
- Enable/disable hybrid search

Settings -> Web Search

- Enable web search globally
- Select search engine
- Set Searxng instance URL or API keys for other engines
- Result count per query

Settings -> Image Generation

- Enable/disable globally
- Engine selection
- Model / checkpoint selection
- Default image size and steps

Settings -> Audio

- STT engine and model
- TTS engine, voice, and speed
- Auto-send transcribed text

Settings -> Pipelines

- Manage pipeline (filter/action) connections
- Add pipeline server endpoints

3.3 Authentication setup

Local accounts (default)

No configuration required. The first registered user is automatically granted admin role.

Disabling registration after setup

Set `ENABLE_SIGNUP=False` or toggle in **Settings -> General -> Enable New Sign Ups**.

OAuth providers

Configure in **Settings -> General -> OAuth**:

Provider	Required env vars
Google	`GOOGLE_CLIENT_ID`, `GOOGLE_CLIENT_SECRET`
Microsoft	`MICROSOFT_CLIENT_ID`, `MICROSOFT_CLIENT_SECRET`, `MICROSOFT_CLIENT_TENANT_ID`
GitHub	`GITHUB_CLIENT_ID`, `GITHUB_CLIENT_SECRET`
OpenID Connect	`OAUTH_CLIENT_ID`, `OAUTH_CLIENT_SECRET`, `OPENID_PROVIDER_URL`

LDAP

Set `ENABLE_LDAP=True` and configure `LDAP_SERVER_HOST`, `LDAP_SERVER_PORT`, `LDAP_ATTRIBUTE_FOR_USERNAME`, `LDAP_SEARCH_BASE`, `LDAP_SEARCH_FILTER`.

Trusted header authentication

Set `WEBUI_AUTH_TRUSTED_EMAIL_HEADER` to the header name your reverse proxy injects (e.g., `X-Remote-User-Email`). The proxy must strip this header from untrusted requests.

API key authentication

Users generate API keys under **Settings -> Account -> API Keys**. Use in requests as `Authorization: Bearer <key>` or the `X-API-Key` header.

4. Core Usage

4.1 Chat interface overview

The chat interface is the central workspace. Key elements:

- **Sidebar (left):** Chat history, folders, search, new chat button
- **Model selector (top):** Dropdown to choose one or more models for the current chat
- **Chat area (center):** Conversation history with rendered markdown, code highlighting, and inline controls
- **Input bar (bottom):** Message input with attachment, voice, web search, and send controls
- **Message controls:** Per-message buttons for copy, edit, regenerate, thumbs up/down, and branch navigation

4.2 Starting a chat

1. Click **New Chat** (pencil icon or sidebar button)
2. Select a model from the dropdown at the top
3. Optionally: click the system prompt icon to set a system message for this chat
4. Type your message and press **Enter** (or **Shift+Enter** for a new line)
5. Press **Ctrl+Enter** to send without triggering chat -- useful to test input

> **Tip**
>
> Type `/` in the message bar to open the prompt template picker. Start typing the template name to filter. Press **Enter** or click to insert the template text into the input.

4.3 Model selection

Single model

Click the model name at the top of the chat window. A dropdown lists all available models from connected backends. Ollama models show their size; OpenAI models show the provider name.

Multi-model comparison

Click **+** next to the model selector to add a second (or more) model. The chat splits into parallel columns, each model responding independently to every message. Useful for side-by-side comparison of outputs, styles, or accuracy.

To remove a model from a multi-model chat, click **×** on its column header.

Model settings per chat

Click the sliders icon next to the model name to override generation parameters for this chat:

Parameter	Description
Temperature	Randomness (0 = deterministic, 2 = very creative)
Top P	Nucleus sampling threshold

Top K	Vocabulary size cap per token
Max Tokens	Maximum response length
Frequency Penalty	Penalize repeated tokens
Presence Penalty	Penalize tokens already in context
Seed	Fixed seed for reproducibility
System Prompt	Override system message for this chat

> **Tip**
> Chat-level parameter overrides persist with the chat history. If you want permanent defaults, set them in **Settings -> Models -> [Model Name] -> Advanced Parameters**.

4.4 Message controls

Each assistant message has an action row that appears on hover:

Button	Action
Copy	Copy the raw markdown text
Edit	Open the message in an inline editor; save regenerates from that point
Regenerate	Re-request the same prompt, produces a new branch
Thumbs up / down	Rate the response (stored for your review)
Branch arrows	Navigate between alternative responses at this position
Continue	Ask the model to continue an incomplete response
Read aloud	TTS playback of this message (requires TTS configured)

> **Important**
> Known issue (GitHub #21564): **Edit** and **Continue Response** may not function correctly in all configurations on v0.8.12. If editing a message does not regenerate the response, try refreshing the page and editing again. A fix is tracked upstream.

4.5 File and image upload

Click the **paperclip icon** in the message bar to attach files:

- **Documents:** PDF, DOCX, TXT, MD, CSV, XLSX -- processed through the RAG pipeline and chunked into the context
- **Images:** PNG, JPG, WEBP, GIF -- sent directly to vision-capable models; other models receive a text description

- **Code files:** Any plain-text file -- sent as raw text in context

Drag-and-drop directly onto the chat window also works.

> **Note**
> Uploaded files are extracted and chunked using the configured RAG pipeline. Very large documents (hundreds of pages) may be truncated based on the configured chunk count (Top-K setting). The model receives retrieved chunks, not the full document verbatim.

4.6 Voice mode (STT / TTS)

Speech-to-text input

Click the **microphone icon** in the message bar. Grant browser microphone permission when prompted. Speak your message -- the transcript appears in the input field. Click the microphone again to stop, then send normally.

Supported STT engines (configured by admin): OpenAI Whisper, local Whisper, browser-native Web Speech API.

Text-to-speech playback

Each assistant message has a **Read aloud** button (speaker icon). Click it to hear the response via your configured TTS engine. Click again to stop playback.

TTS engines: OpenAI TTS, ElevenLabs, browser native speech synthesis.

Voice mode (full)

Enable **Voice Mode** from the chat toolbar for a hands-free experience: the microphone activates automatically after each response, and TTS plays each reply. Press **Escape** to exit voice mode.

4.7 Web search in chat

When web search is enabled, a **globe icon** appears in the message bar. Click it to toggle web search for your next message. The search icon highlights when active.

With web search on, Open WebUI:

1. Generates a search query from your message
2. Retrieves results from the configured search engine
3. Injects relevant snippets into the model context
4. The model cites sources inline

To make web search always-on for a chat, pin it via the globe icon toolbar option.

> **Tip**
> Web search works best for factual, time-sensitive queries. For creative or reasoning tasks, disable it to avoid search result noise in the context.

4.8 Code interpreter

When code execution is enabled, the model can write Python code that runs directly in the browser via Pyodide (a Python runtime compiled to WebAssembly). Results, including plots and data tables, appear inline in the chat.

Trigger code execution by asking the model to:

- Analyze data (paste CSV or describe the dataset)
- Generate a chart or visualization
- Perform calculations
- Transform or clean data

The model writes Python, the interpreter runs it, and output displays below the code block. You can edit the code block and re-run it manually.

> **Note**
> The Pyodide sandbox has access to a subset of Python's standard library and common scientific packages (numpy, pandas, matplotlib). Network access from within the sandbox is restricted. This is a browser-side execution environment; your local filesystem is not accessible.

4.9 Image generation

When image generation is enabled, use the **image icon** in the message bar or ask the model directly:

```
Generate an image of a sunset over a mountain lake in watercolor style.
```

Or use the dedicated **Playground -> Image Generation** page for full control over:

- Prompt and negative prompt
- Model / checkpoint selection
- Image size
- Steps and CFG scale (Automatic1111 / ComfyUI)
- Seed

Generated images appear inline in the chat. Click to expand, right-click to save.

Supported engines:

Engine	Notes
`openai`	DALL-E 3 / DALL-E 2 via OpenAI API
`automatic1111`	Local Stable Diffusion via A1111 WebUI API
`comfyui`	Local Stable Diffusion via ComfyUI API
`gemini`	Google Gemini image generation

4.10 Knowledge bases (RAG)

Knowledge bases let you upload documents that models can search when answering questions. They persist across chats.

Creating a knowledge base

1. Navigate to **Workspace -> Knowledge**
2. Click **New Knowledge Base**
3. Enter a name and description
4. Upload documents (PDF, DOCX, TXT, MD, HTML, CSV, XLSX)
5. Wait for embedding to complete (progress indicator)

> **Warning**
> Known issue (GitHub #23787): Updating a file in a knowledge base may silently lose the embeddings for that file, meaning the model cannot retrieve its content. After updating a file, verify the embedding count has not dropped to zero. If it has, delete and re-upload the file.

Using a knowledge base in chat

1. In the chat input bar, click the # icon (or type #)
2. Search for and select your knowledge base
3. It attaches as a context source for that message
4. Ask your question -- retrieved chunks appear in the context

Knowledge bases can also be attached to **Models** so they are always available when using that model.

4.11 Tools and Skills

Tools extend what models can do by exposing callable functions (web scraping, calculators, APIs, etc.). Skills are reusable prompt + tool bundles.

Built-in tools

- Web search (when enabled)
- Code interpreter
- Image generation
- Knowledge base search

Custom tools

Admins and users with permission can create custom tools in **Workspace -> Tools**. Tools are written in Python and define:

- Tool name and description (the model uses these to decide when to call it)
- Input parameters (JSON schema)
- Execution logic (Python function)

Using tools in chat

When tools are attached to a model (or manually injected), the model decides autonomously when to call them based on your message. Tool calls and their results appear inline in the chat as collapsible blocks.

4.12 Channels

Channels provide a Slack/Discord-style persistent messaging experience within Open WebUI. Unlike individual chats, channels:

- Are shared spaces where multiple users can participate
- Have a persistent message history visible to all members
- Support threaded replies
- Can be used for team collaboration around shared models

Creating a channel

1. Navigate to **Workspace -> Channels** (or the megaphone icon in the sidebar)
2. Click **New Channel**
3. Set the channel name, description, and member list
4. Click **Create**

Sending in a channel

Type in the channel input bar and press **Enter**. The message is visible to all channel members. Reply to a specific message by hovering and clicking **Reply**.

4.13 Notes

The Notes feature provides a persistent scratchpad within Open WebUI. Notes support full markdown formatting and can reference content from your chats.

Access via **Workspace -> Notes** (or the document icon in the sidebar).

- Create folders to organize notes
- Use the rich markdown editor
- Reference chat history by copy-pasting
- Notes persist in the Open WebUI database

4.14 Prompt templates

Prompt templates are reusable message starters that speed up repetitive tasks.

Using templates

In the chat input bar, type `/` to open the template picker. Type to search. Press **Enter** or click to insert the template text.

Creating templates

1. Navigate to **Workspace -> Prompts**

2. Click **New Prompt**
3. Set title, description, and template content
4. Use `{{variable}}` placeholders for dynamic values -- a dialog prompts you to fill them in when you use the template
5. Save

Example template:

```
Summarize the following text in {{word_count}} words or fewer, focusing on {{focus_area}}:

{{text}}
```

4.15 Memories

Open WebUI can maintain a persistent memory about you across chats. When enabled, the model can store and recall facts, preferences, and context.

Access **Settings -> Personalization -> Memory** to:

- View stored memories
- Edit or delete individual memories
- Clear all memories
- Enable/disable automatic memory extraction

The model writes to memory when it learns something relevant about you. You can also manually add memories.

4.16 Folders and chat organization

All chats appear in the sidebar. Organize them using:

Folders:

- Right-click a chat -> **Move to Folder** -> select or create a folder
- Or drag a chat onto a folder in the sidebar
- Folders collapse/expand by clicking

Tags:

- Click the tag icon on any chat to add tags
- Filter the sidebar by tag using the tag dropdown

Search:

- Use the search bar at the top of the sidebar to search chat titles and content
- Results update as you type

Pinning:

- Right-click a chat -> **Pin** to keep it at the top of the sidebar regardless of date

Archiving:

- Right-click -> **Archive** to hide a chat without deleting it
- View archived chats via **Sidebar -> Archived Chats**

4.17 Chat sharing and export

Share a chat

Click the **share icon** (chain link) on any chat to generate a public share link. The link recipient can view the conversation read-only without logging in, provided your instance's public sharing is enabled.

To revoke a share: open the chat -> Share -> **Delete Share**.

Export a chat

Right-click a chat in the sidebar -> **Export** to download:

- **JSON** -- full conversation data including metadata
- **Markdown** -- rendered markdown text

Import chats

Settings -> Data -> Import Chat Export -- accepts JSON files in Open WebUI format. Note: ChatGPT export files (GitHub #23505) may fail to import correctly in v0.8.12 due to a known format incompatibility.

Export all data

Settings -> Data -> Export All Chats -- downloads a single JSON file containing every chat you own.

4.18 Playground

The **Playground** section (accessible from the sidebar or via `/playground`) provides direct API-level access to models outside the chat interface.

Completions

Raw text completion mode -- no chat history, just a text prompt -> model output. Useful for testing prompts and fine-tuning generation parameters.

Chat completions

Interactive chat mode with full parameter control in the sidebar panel, similar to the main chat but with all parameters exposed.

Image generation

Direct image generation interface -- see section 4.9.

4.19 Internationalization

Open WebUI supports 50+ languages. Change your interface language:

Settings -> General -> Language -> select from the dropdown.

Languages are community-contributed. If a string is not yet translated, it falls back to English.

4.20 Theme

Settings -> General -> Theme:

- **Light** -- white background
- **Dark** -- dark background (default)
- **System** -- follows your OS preference

4.21 Keyboard shortcuts

Shortcut	Action
`Ctrl+Enter`	Send message
`Shift+Enter`	New line in input
`/`	Open prompt template picker
`#`	Open knowledge base picker
`@`	Open model / tool picker
`Ctrl+Shift+O`	New chat
`Ctrl+Shift+S`	Toggle sidebar
`Ctrl+K`	Focus search
`Escape`	Cancel / close dialog
`Ctrl+C`	Copy selected message (when message is focused)

5. Common Tasks

5.1 Connect a local Ollama instance

1. Ensure Ollama is running: `ollama serve`
2. Open WebUI **Settings -> Connections -> Ollama**
3. Enter `http://localhost:11434` (or the actual hostname/IP)
4. Click **Save** -- the status dot turns green if reachable

5.2 Connect an OpenAI-compatible API

1. **Settings -> Connections -> OpenAI**
2. Set **Base URL**: `https://api.openai.com/v1` (or your provider's URL)
3. Set **API Key**: your key
4. Click **Save**

5. Compatible providers: OpenAI, Azure OpenAI, Groq, Together AI, Fireworks, Mistral AI, Anthropic (via proxy), LM Studio, vLLM, LocalAI, Ollama's `/v1` endpoint

5.3 Pull a new Ollama model

Settings -> Models -> Pull a Model -> enter the model name, e.g.:

```
llama3.2
mistral
codestral
phi3.5
gemma2:27b
```

Or from the model selector dropdown, click **Pull** next to any model name. Progress displays inline.

5.4 Set a default model

Settings -> General -> Default Model -> select from dropdown. This model pre-selects in every new chat.

5.5 Create a custom model (Modelfile)

1. **Workspace -> Models -> New Model**
2. Select a base model
3. Set name, description, and avatar
4. Write a system prompt
5. Set default generation parameters
6. Attach knowledge bases and tools
7. Save -- the model appears in the model selector

This is equivalent to creating an Ollama Modelfile but managed entirely in the UI.

5.6 Upload a document for Q&A

1. In any chat, click the **paperclip icon**
2. Select your document (PDF, DOCX, TXT, MD, etc.)
3. Once uploaded, ask questions: "What does this document say about X?"
4. For persistent access across chats, use a Knowledge Base (section 4.10) instead

5.7 Enable web search for a message

1. Click the **globe icon** in the message bar (must be enabled by admin)
2. The icon highlights to show search is on for your next message
3. Ask a question that benefits from current information: "What happened with X today?"

5.8 Regenerate a response

Click the **regenerate icon** (circular arrow) below any assistant message to request a fresh response to the same prompt. The new response creates a branch -- use the left/right arrows to navigate between versions.

5.9 Edit a message

Click the **edit icon** (pencil) on any user message. Edit the text and click **Save & Submit**. The conversation restarts from that point with the revised message.

5.10 Continue an incomplete response

If a model response was cut off, click **Continue** below the message. The model picks up where it left off.

5.11 Compare models side by side

1. Start a new chat
2. Select the first model
3. Click **+** to add a second model
4. Type your message -- both respond in parallel columns
5. Compare quality, style, or accuracy

5.12 Set a system prompt for a chat

Click the **system prompt icon** (shield or document icon near the model selector) to open the system prompt editor for the current chat. Type your instructions -- they prepend every request to the model for this chat.

5.13 Use a prompt template

1. In the message bar, type `/`
2. Start typing to filter templates
3. Click or press **Enter** to insert
4. Fill in any `{{placeholder}}` fields in the dialog that appears
5. Send the completed message

5.14 Search your chat history

Click the **search icon** (magnifying glass) at the top of the sidebar. Type keywords -- chat titles and message content are searched. Click a result to open that chat.

5.15 Export a chat to Markdown

Right-click the chat in the sidebar -> **Export -> Markdown**. The file downloads with all messages formatted as markdown.

5.16 Share a chat publicly

1. Open the chat
2. Click the **share** (chain link) button in the toolbar
3. Copy the generated URL
4. Share it -- recipients can view without logging in

5.17 Create a folder for chats

In the sidebar, click the **folder icon** or right-click an empty area -> **New Folder**. Name the folder. Drag chats into it, or right-click a chat -> **Move to Folder**.

5.18 Tag a chat

Open any chat. Click the **tag icon** in the chat header. Type a tag name and press **Enter**. Tags appear in the sidebar and can be used to filter chats.

5.19 Pin a chat

Right-click a chat in the sidebar -> **Pin**. Pinned chats appear at the very top of the sidebar above all folders.

5.20 Archive old chats

Right-click a chat -> **Archive**. The chat disappears from the main sidebar. To view or restore archived chats: **Sidebar -> Archived Chats** (clock icon at the bottom).

5.21 Generate an image in chat

With image generation enabled, type:

```
Generate an image of [your description]
```

Or click the **image icon** in the message toolbar, enter a prompt in the dialog, and click **Generate**.

5.22 Run Python code in chat

Ask the model to write and run Python:

```
Write Python code to generate a bar chart of monthly sales from this data: Jan: 120, Feb: 145, Mar: 132. Run
```

The code interpreter (Pyodide) executes the code in your browser and shows output inline.

5.23 Add a voice input

Click the **microphone icon** in the message bar. Speak your message. The transcript fills the input field. Edit if needed, then send.

5.24 Listen to a response

On any assistant message, click the **speaker icon**. The configured TTS engine reads the response aloud. Click again to stop.

5.25 Add a memory manually

Settings -> Personalization -> Memory -> Add Memory -> type the fact you want remembered, e.g., "I prefer concise answers" or "My name is Alex."

5.26 Create a knowledge base

1. **Workspace -> Knowledge -> New Knowledge Base**
2. Name it and upload documents
3. Wait for embeddings to complete
4. In future chats, type `#` and select the knowledge base

5.27 Create a prompt template

1. **Workspace -> Prompts -> New Prompt**
2. Set title: `Code Review`
3. Set content:

```
Review the following {{language}} code for bugs, performance issues, and style violations. Provide specif
```

```
4. Save -- type `/Code Review` in any chat to use it

### 5.28 View API usage / account

**Settings -&gt; Account -&gt; API Keys** -- create and manage keys. **Settings -&gt; Account -&gt; Usage**

### 5.29 Change interface language

**Settings -&gt; General -&gt; Language** -&gt; select your language. The UI updates immediately.

### 5.30 Change theme

**Settings -&gt; General -&gt; Theme** -&gt; Light, Dark, or System.

---

## 6. Troubleshooting

### Error: "Cannot connect to Ollama"

**Cause:** The Open WebUI container cannot reach the Ollama server at the configured URL.

**Solution:**
1. Verify Ollama is running: `ollama list` should return a model list without errors
2. If Ollama is on the host and Open WebUI is in Docker, use `host.docker.internal` (Linux/Mac) or `172.17.0
3. Add `--add-host=host.docker.internal:host-gateway` to your `docker run` command
4. Check `OLLAMA_BASE_URL` does not include `/api` -- it should be `http://host.docker.internal:11434` not `
5. Test connectivity: `docker exec open-webui curl http://host.docker.internal:11434`

**Prevention:** Set `OLLAMA_BASE_URL` explicitly rather than relying on default discovery.

---

### Error: "401 Unauthorized" when calling OpenAI API

**Cause:** Invalid or missing API key.

**Solution:**
1. Verify `OPENAI_API_KEY` is set and correct
2. Check the API key has not expired or been revoked
3. For non-OpenAI providers, confirm the key format matches their requirements
4. Check `OPENAI_API_BASE_URL` points to the correct endpoint (default: `https://api.openai.com/v1`)
```

```
**Prevention:** Store keys in a `.env` file and reference it in `docker-compose.yaml` with `env_file`.

---

### Error: "No models available"

**Cause:** Connected backends have no models, or backends are unreachable.

**Solution:**
1. For Ollama: run `ollama pull llama3.2` to download at least one model
2. For OpenAI: verify API key and that the account has model access
3. Check **Settings -&gt; Connections** for red status indicators
4. Check Docker logs: `docker logs open-webui`

---

### Error: "Failed to upload file"

**Cause:** File size exceeds limit, unsupported format, or storage volume full.

**Solution:**
1. Check the file is a supported format (PDF, DOCX, TXT, MD, HTML, CSV, XLSX)
2. Check available disk space: `docker exec open-webui df -h /app/backend/data`
3. If using a named volume, check host disk space
4. Very large files (100+ MB) may exceed the upload size limit -- split into smaller files

---

### Error: "Edit and Continue Response don't work" (GitHub #21564)

**Cause:** Known bug in v0.8.12 affecting message editing and continuation in certain configurations.

**Solution:**
1. Refresh the page and try again
2. Try a hard refresh (Ctrl+Shift+R) to clear any cached JavaScript state
3. If the problem persists, try in a private/incognito browser window
4. Check for updates -- a fix is tracked upstream for a subsequent release

**Prevention:** Monitor the GitHub issue for patch releases.

---

### Error: "Import of ChatGPT export file is broken" (GitHub #23505)

**Cause:** ChatGPT's export JSON format changed and v0.8.12 does not handle the new schema.

**Solution:**
1. Use a conversion script to reformat the export to Open WebUI's JSON schema before importing
2. Community-contributed conversion scripts are available in the GitHub issue thread
3. Monitor GitHub #23505 for an official fix

---

### Error: "Knowledge base search returning no results"

**Cause:** Embeddings not generated, document processing failed, or embedding count dropped to zero (GitHub

**Solution:**
1. Open the knowledge base -&gt; check the document status. If it shows a warning, re-upload the file
```

```
2. Check that the embedding model is configured and reachable: **Settings -&gt; Documents -&gt; Embedding Mo
3. If you recently updated a file in the knowledge base, delete the document and re-add it -- the update may
4. Verify ChromaDB is healthy: `docker logs open-webui | grep -i chroma`

---

### Error: "Bandwidth/memory grows exponentially with chat length" (GitHub #23733)

**Cause:** Known issue where growing chat history causes exponential growth in payload size on each request.

**Solution:**
1. Start a new chat for long-running conversations -- this resets the context window
2. Enable context summarization if available on your instance
3. Keep individual chats focused on specific topics
4. Monitor browser memory usage via browser dev tools if you notice slowdowns

**Prevention:** Adopt a practice of starting fresh chats instead of using single very long conversations.

---

### Error: "Model is stuck / no response"

**Cause:** Model is generating but very slowly, context window overflow, or backend crash.

**Solution:**
1. Wait -- long responses with large context can take several minutes on CPU
2. Click **Stop** (square icon during generation) then try again
3. Check backend health: `docker logs open-webui`
4. For Ollama: check `ollama ps` to see if the model is loaded and not stuck
5. Reduce `Max Tokens` in model parameters for faster responses

---

### Error: "CORS error in browser console"

**Cause:** Accessing Open WebUI from a domain or port not in the allowed origins.

**Solution:**
1. Access via the configured domain/port -- not a direct IP if the domain is configured
2. Check `WEBUI_URL` env var matches your actual access URL
3. If behind a reverse proxy, ensure the proxy passes `Host` and `Origin` headers correctly

---

### Error: "Registration not working / Sign up button missing"

**Cause:** `ENABLE_SIGNUP` is set to `False`, or the instance is in OAuth-only mode.

**Solution:**
1. Check if the admin deliberately disabled self-registration
2. Contact your Open WebUI admin to create an account manually
3. If you are the admin: set `ENABLE_SIGNUP=True` and restart the container

---

### Error: "PDF not rendering in knowledge base"

**Cause:** PDF extraction library not installed or PDF is malformed/protected.

**Solution:**
```

```
1. Check **Settings -&gt; Documents -&gt; PDF Extraction Engine** -- switch between `pdfminer` and `pypdf`
2. Password-protected PDFs cannot be processed -- remove the password before uploading
3. Scanned PDFs (image-only) require OCR -- Open WebUI does not include OCR by default

---

### Error: "Voice input not working"

**Cause:** Browser microphone permission denied, or STT engine not configured.

**Solution:**
1. Allow microphone access in your browser's site permissions
2. HTTPS is required for microphone access in Chrome/Firefox -- use a secure connection or `localhost`
3. Check **Settings -&gt; Audio -&gt; STT Engine** is configured
4. Try the browser-native Web Speech API option which requires no server-side config

---

### Error: "TTS not playing audio"

**Cause:** TTS engine not configured, API error, or browser audio blocked.

**Solution:**
1. Check **Settings -&gt; Audio -&gt; TTS Engine** is configured
2. If using OpenAI TTS, verify the API key has TTS access
3. Check browser audio permissions -- some browsers block audio from web apps by default
4. Test with the browser-native TTS option first to isolate whether the issue is engine config or browser

---

### Error: "Image generation failed"

**Cause:** Image generation engine not configured, API error, or local SD instance not reachable.

**Solution:**
1. Check **Settings -&gt; Images** -- verify engine is selected and connected
2. For Automatic1111: ensure the A1111 WebUI is running with `--api` flag: `python launch.py --api`
3. For ComfyUI: verify the ComfyUI server is running and the API endpoint is correct
4. For OpenAI DALL-E: verify API key and that your account has DALL-E access
5. Check Docker logs for error details: `docker logs open-webui`

---

### Error: "Code interpreter/execution not available"

**Cause:** Code execution disabled by admin, or browser compatibility issue.

**Solution:**
1. Check that `ENABLE_CODE_EXECUTION=True` (admin setting)
2. Pyodide requires a modern browser (Chrome 90+, Firefox 88+, Safari 15+)
3. Ensure Content-Security-Policy headers allow WebAssembly execution if behind a reverse proxy
4. Check the browser console for Pyodide loading errors (network issues may prevent the WASM download)

---

### Error: "Cannot log in after upgrade"

**Cause:** Database migration failure or session cookie invalidation after `WEBUI_SECRET_KEY` change.

**Solution:**
```

```
1. Clear your browser cookies for the Open WebUI domain
2. Check Docker logs for migration errors: `docker logs open-webui | head -100`
3. If `WEBUI_SECRET_KEY` changed between deployments, all existing sessions are invalidated -- log in fresh
4. If migration failed, restore from backup and check the release notes for manual migration steps

---

### Error: "LDAP login not working"

**Cause:** LDAP server connectivity, misconfigured bind DN, or attribute mismatch.

**Solution:**
1. Test LDAP connectivity from the container: `docker exec open-webui ldapsearch -H ldap://your-server -x -b
2. Verify `LDAP_ATTRIBUTE_FOR_USERNAME` matches the LDAP attribute containing usernames (often `sAMAccountNa
3. Verify `LDAP_SEARCH_FILTER` is correct for your directory schema
4. Enable debug logging and check the container logs

---

### Error: "OAuth login fails / redirect loop"

**Cause:** Redirect URI mismatch in OAuth provider settings, or `WEBUI_URL` not set correctly.

**Solution:**
1. In your OAuth provider (Google, GitHub, etc.), verify the redirect URI exactly matches `{WEBUI_URL}/oauth
2. Set `WEBUI_URL` to the exact public URL of your instance (e.g., `https://chat.example.com`)
3. Ensure no trailing slash in `WEBUI_URL`
4. Verify client ID and secret are correct

---

### Error: "Slow first response / model cold start"

**Cause:** Ollama must load the model into memory (VRAM or RAM) on first request.

**Solution:**
1. This is expected behavior -- the first request after a period of inactivity takes longer
2. Keep models preloaded with `OLLAMA_KEEP_ALIVE=-1` environment variable on the Ollama server
3. For frequently used models, send a lightweight request periodically to keep the model warm

---

### Error: "Container exits immediately after start"

**Cause:** Configuration error (missing required env vars), port conflict, or volume permission issue.

**Solution:**
1. Check logs immediately: `docker logs open-webui`
2. Verify no other service is using port 3000 (or your configured port)
3. Check volume permissions: `ls -la /var/lib/docker/volumes/open-webui/_data/`
4. Ensure required database files are accessible

---

### Error: "Data not persisting between container restarts"

**Cause:** No volume or bind mount configured -- container filesystem is ephemeral.

**Solution:**
```

```
1. Always run with a named volume: `-v open-webui:/app/backend/data`
2. Or use a bind mount: `-v /host/path:/app/backend/data`
3. Without a volume, all data (chats, settings, accounts) is lost on container removal

---

### Error: "High CPU usage / system unresponsive during generation"

**Cause:** LLM inference on CPU is inherently CPU-intensive. Very large models can saturate a system.

**Solution:**
1. Use a smaller quantized model (e.g., `llama3.2:3b-q4_0` instead of `llama3.2:8b`)
2. Enable GPU offloading if you have a compatible GPU
3. Set `num_threads` in Ollama's model parameters to limit CPU core usage
4. Deploy generation on a separate server and point Open WebUI at it

---

### Error: "Context length exceeded"

**Cause:** The conversation plus system prompt exceeds the model's maximum context window.

**Solution:**
1. Start a new chat -- this clears all accumulated context
2. Reduce the system prompt length
3. Use a model with a larger context window (e.g., `llama3.2` supports 128K tokens)
4. Enable context summarization (if available) to compress old messages

---

### Error: "Reverse proxy / SSL not working"

**Cause:** Nginx/Traefik configuration missing WebSocket support or incorrect upstream port.

**Solution:**
1. Open WebUI requires WebSocket support for streaming -- ensure your proxy passes `Upgrade` and `Connectior
2. Standard nginx config:
```

location / { proxy_pass http://localhost:3000; proxy_http_version 1.1; proxy_set_header Upgrade $http_upgrade; proxy_set_header Connection "upgrade"; proxy_set_header Host $host; }

```
3. Verify SSL certificate is valid and covers your domain

---

### Error: "Embeddings model download fails"

**Cause:** Container has no internet access, or Hugging Face Hub is blocked.

**Solution:**
1. Pre-download the embedding model: the default `sentence-transformers/all-MiniLM-L6-v2` (~80MB) downloads
2. If internet access is restricted, set `HF_HOME` to point to a pre-populated cache directory
3. Alternatively, configure a different embedding engine in **Settings -&gt; Documents**

---

### Error: "Model-level reasoning/thinking tags appearing in output" (GitHub #23339)

**Cause:** Some models emit `&lt;thinking&gt;` or reasoning tokens in their responses. v0.8.12 does not yet
```

```
**Solution:**
1. Use a system prompt to instruct the model to omit reasoning tags in final output
2. Monitor GitHub #23339 for the per-model disable toggle when it lands
3. Some Ollama model variants (non-thinking variants) omit reasoning by default -- switch to those

---

### Error: "Admin panel settings not saving"

**Cause:** Browser cache conflict, or database write permission issue.

**Solution:**
1. Hard-refresh the page (Ctrl+Shift+R) and try again
2. Check Docker logs for SQLite write errors: `docker logs open-webui | grep -i sqlite`
3. Verify the data volume has write permissions: `docker exec open-webui ls -la /app/backend/data/`
4. Check disk space -- a full disk causes silent write failures

---

### Error: "Streaming responses cut off"

**Cause:** Reverse proxy timeout, max token limit reached, or network interruption.

**Solution:**
1. If behind nginx: increase `proxy_read_timeout` (default 60s is often too short for long responses):
```

proxy_read_timeout 300s; proxy_send_timeout 300s;

```
2. Check `Max Tokens` setting -- increase or set to 0 for unlimited
3. Check Ollama's `OLLAMA_REQUEST_TIMEOUT` if using Ollama backend

---

### Error: "Multiple Ollama endpoints not load-balancing"

**Cause:** Open WebUI does not load-balance across multiple Ollama URLs by default.

**Solution:**
1. Add multiple Ollama endpoints in **Settings -&gt; Connections** -- Open WebUI lists all models from all e
2. The model selector shows which server a model is from
3. Users can select which model/server to use per chat

---

## 7. FAQ

**Q: Is Open WebUI free?**
Open WebUI itself is open source (MIT license) and free to use. You pay for any external API usage (OpenAI,

**Q: Does Open WebUI send my conversations to the cloud?**
No data leaves your server unless you configure an external API (OpenAI, etc.). When using Ollama with local

**Q: Can I use Open WebUI without Ollama?**
Yes. Open WebUI supports any OpenAI-compatible API endpoint. Set `ENABLE_OLLAMA_API=False` and configure `OP

**Q: What models can I use?**
Any model available in Ollama (llama3, mistral, phi3, gemma, codestral, qwen, and hundreds more), any OpenAI

**Q: How many users can share one Open WebUI instance?**
Open WebUI is multi-user and multi-account by design. There is no hard limit. Practical limits depend on ser
```

```
**Q: What is the difference between "pending" and "user" roles?**
`pending` users can log in but cannot interact with models -- they must be approved by an admin. `user` role

**Q: Can I run Open WebUI behind a reverse proxy?**
Yes. This is the recommended production deployment. Ensure your proxy forwards WebSocket connections (requir

**Q: How do I back up my data?**
All data lives in the Docker named volume or bind mount at `/app/backend/data`. Back up this directory regul
```

docker run --rm \ -v open-webui:/data \ -v /backup:/backup \ alpine tar czf /backup/open-webui-backup-$(date +%Y%m%d).tar.gz /data

```
**Q: Can I export my chats?**
Yes. Right-click any chat in the sidebar -&gt; **Export -&gt; JSON or Markdown**. For all chats: **Settings

**Q: How do I import ChatGPT history?**
**Settings -&gt; Data -&gt; Import Chat Export**. Note: due to GitHub issue #23505, ChatGPT export files may

**Q: Can I use Open WebUI with Azure OpenAI?**
Yes. Set `OPENAI_API_BASE_URL` to your Azure endpoint (e.g., `https://myinstance.openai.azure.com/openai/dep

**Q: Does Open WebUI support vision / image input?**
Yes, for models that support it (e.g., `llava`, `gpt-4o`, `claude-3`). Upload an image via the paperclip ico

**Q: How do I enable web search?**
Set `ENABLE_WEB_SEARCH=True` and configure a `WEB_SEARCH_ENGINE`. SearXNG is the recommended self-hosted opt

**Q: What is a Knowledge Base?**
A knowledge base is a collection of documents that are embedded (converted to vector representations) so the

**Q: Can multiple users share a knowledge base?**
Knowledge base visibility depends on your configuration. Admins can create shared knowledge bases visible to

**Q: How does RAG (Retrieval-Augmented Generation) work in Open WebUI?**
When you attach a knowledge base or upload a document, Open WebUI embeds the content using `sentence-transfo

**Q: What is the code interpreter and how is it safe?**
The code interpreter runs Python in your browser using Pyodide (Python compiled to WebAssembly). It runs in

**Q: Can I host Open WebUI on a Raspberry Pi?**
Yes, with limitations. Use a quantized 1B or 3B parameter model (e.g., `phi3.5:mini-q4_0`). CPU-only inferen

**Q: How do I create a ChatGPT-like custom GPT?**
Use **Workspace -&gt; Models** to create a custom model. Attach a system prompt, knowledge bases, and tools.

**Q: What languages does the interface support?**
50+ languages are supported. Language switching is per-user in **Settings -&gt; General -&gt; Language**. La

**Q: Can I use Open WebUI for coding assistance?**
Yes. Choose a code-specialized model like `codestral`, `qwen2.5-coder`, `deepseek-coder`, or OpenAI's `gpt-4

**Q: What is the Playground?**
The Playground (`/playground`) provides raw API-style access to models for prompt testing and experimentatio

**Q: Can I disable user registration entirely?**
Yes. Set `ENABLE_SIGNUP=False`. Existing accounts can still log in; new registrations are blocked. The admin

**Q: What ports does Open WebUI use?**
Open WebUI listens on port 8080 inside the container. The default Docker command maps this to port 3000 on t
```

```
**Q: How do I update without losing data?**
Your data is stored in a named Docker volume (`open-webui`) or bind mount, not in the container. To update:

**Q: Is there a mobile app?**
No native mobile app. Open WebUI is a responsive web application that works in mobile browsers. Add it to yo

**Q: How do I set up email notifications?**
Open WebUI does not include a built-in email notification system. For user-related emails (account creation,

**Q: Can I use Open WebUI as an API backend for my own application?**
Yes. Open WebUI exposes an OpenAI-compatible API. Use `POST /api/chat/completions` with an API key from **Se

**Q: What is the difference between Tools and Functions?**
Tools are Python functions the model can call during generation (web search, calculators, APIs). Functions i

**Q: How do Channels differ from chats?**
Chats are one-on-one conversations between you and a model -- private, and only visible to you. Channels are

**Q: How do I reset a forgotten admin password?**
Access the container shell: `docker exec -it open-webui bash`. Use the Open WebUI admin CLI or SQLite direct

**Q: Can I use Open WebUI offline?**
Yes, with local models via Ollama. After pulling models, no internet connection is required for inference. T

**Q: How do I limit which models users can see?**
Admins can hide specific models in **Settings -&gt; Models -&gt; [Model] -&gt; Visibility**. Set a model to

**Q: What is the Feishu OAuth option?**
Feishu is ByteDance's enterprise collaboration platform (similar to Slack), widely used in China. Open WebUI

**Q: How does the model parameter "seed" work?**
Setting a seed makes generation deterministic -- the same prompt + seed produces the same output every time

**Q: Can I self-host the embedding model?**
Yes. Change `RAG_EMBEDDING_ENGINE` to `ollama` and set `RAG_EMBEDDING_MODEL` to an Ollama-hosted embedding m

**Q: What happens to my data if I delete a chat?**
Deleted chats are removed from the database. Uploaded files that were part of that chat may remain in storag

---

&gt; **DocCompiler.ai** v3.1 | Upgraded source coverage | Generated 2026-05-13 | Source: `open-webui/open-we
&gt;
&gt; A Cyber Panda Solutions LLC product. This document is intentionally
&gt; comprehensive -- built to be the complete, source-verified reference
&gt; an AI or human needs without consulting other sources. **Do not print:**
&gt; the always-current version lives at [doccompiler.ai/open-webui/open-webui](https://doccompiler.ai/open-
```