Ollama ChatBot - PRD

Product overview	
🗓️ **Date**	Nov 17, 2025
🎽 **Team members**	Guy Raz & Gil Chen

🎯 Objective

Build a simple, interactive chatbot web app that communicates with a locally-installed Ollama model (e.g., Llama 3.2).

The system must demonstrate end-to-end flow between a front-end chat UI and a back-end agent that relays messages to the local LLM and streams responses.

Motivation

Show practical understanding of:

- LLM orchestration and prompt-response pipelines
- Front-end ↔ backend integration
- Local model hosting (privacy, reproducibility)

Scope

The system will implement a local LLM-powered chatbot with the following core features and capabilities:

- A single-user web-based interface built with Reflex.
- Integration with the locally installed Ollama model (default model: `llama3.2`).
- Real-time response streaming from the backend to the chat UI.
- Ability to start a new chat to clear the conversation context.
- Ability to save previous chat conversations (within the same session).
- Optional customizations of theme (System / Dark / Light) toggle or model selection.

🗒️ Requirements

Requirement	Description
User input and message sending	The user can type a prompt into the chat input field and send it to the backend.
Message forwarding to Ollama	The backend receives the prompt and forwards it to the locally running Ollama REST API.
Response streaming	The chatbot streams the model's reply token-by-token to the UI in real time.
Conversation display	The UI maintains a scrollable chat history showing both user and model messages in order.

New chat reset	The user can start a new conversation that clears previous messages.
Save previous chat conversations	The user can reach previous chat conversations by clicking the wanted one on the history bar on the left side of the screen
Optional configuration	The user can optionally choose between available local models or switch UI themes.

🏗️ System Architecture

High-Level Flow

1. User Interaction
    - The user types a message in the web interface and presses “Send.”
2. Frontend → Backend
    - The Reflex frontend sends the message to the backend through an HTTP or WebSocket request.
3. Backend → Ollama
    - The backend agent (built with `pydantic_ai`) forwards the prompt to the local Ollama REST API (`http://localhost:11434/v1`).
4. Ollama → Backend
    - The local LLM (e.g., Llama 3.2) generates tokens and streams them back to the backend.
5. Backend → Frontend
    - The backend relays the streamed response to the frontend, which renders it in real time.
6. Display & State Management
    - Reflex maintains the conversation state, message history, and allows resetting or creating a new chat.

📈 Implementation Plan

Phase	Main Tasks	Deliverables / Outcomes
Environment Setup	Install Python ≥ 3.9 and required libraries- Install and start Ollama locally- Pull default model (`ollama pull llama3.2`)	✅ Local LLM environment ready
Backend Development	Implement a `pydantic_ai.Agent` that sends user prompts to Ollama- Handle streaming responses and exceptions- Expose an API or socket endpoint for the frontend	`main.py` backend with live streaming
Frontend Development	Build the chat UI using Reflex Framework - Add message story, input field, and send button	Functional chat interface

Integration	Connect Reflex frontend to backend endpoint- Verify full data flow: UI → backend → Ollama → UI- Test conversation resets and error messages	Working end-to-end chat demo
Testing & Debugging	Unit test backend calls and error handling- Manual test user interaction and latency- Fix stability or rendering issues	Test suit & Fixes
Presentation & Submission	Deploy to Github Pages - Finalize documentation (README + PRD) - Submit GitHub repo for grading	Final deliverable ready for evaluation

🔨 Development Environment

The project is designed for reproducible, containerized development using modern tooling.

Containerization

- The entire system runs inside a Docker container, ensuring that the application behaves identically across different environments.
- The container includes:
  - Python ≥ 3.9
  - Reflex (frontend framework)
  - Pydantic AI (agent orchestration)
  - Dependencies managed by `uv` (Python package manager)
- Ollama runs locally on the remote machine (docker container).
- This approach eliminates version conflicts and simplifies onboarding for new environments.

Dev Containers

- The repository includes a `.devcontainer/devcontainer.json` configuration for VS Code Remote Containers.
- Opening the project in VS Code automatically builds the development environment inside Docker.
- The environment contains all dependencies, system tools, and ports preconfigured — making the setup seamless and reproducible.

Package Management with `uv`

- Instead of using `pip` or `venv`, this project uses `uv` — a fast, modern Python package and environment manager.
- `uv` ensures deterministic builds by maintaining locked dependency versions in `pyproject.toml` and `uv.lock`.
- It automatically isolates environments per project, improving portability and reproducibility.
- Typical workflow: `pip install uv`

- `uv sync     # Installs exact versions from lock file`
- `uv run reflex run # Runs the chatbot inside the environment`
- This guarantees that every contributor or evaluator runs the exact same dependency versions.

Benefits

- Consistent behavior across machines (no dependency drift).
- Fast environment setup using prebuilt images and `uv` caching.
- Simplified grading or peer review — the container runs identically on any system with Docker and Ollama installed.

🎨 Design

The user interface is designed to be clean, intuitive, and visually relaxing, prioritizing usability and aesthetic simplicity.

Design Principles

- **Clarity and Focus:**
  The layout presents a minimal chat area with clear message separation and intuitive controls.
  The left panel (“Chat History”) provides context without cluttering the workspace, while the main pane centers the user’s current conversation.
- **Eye-Friendly Color Palette:**
  The interface uses soft whites and gentle accent colors (lavender and sky tones) to reduce eye strain during long interactions.
  The color of the “Send” button and hover states follow accessible contrast ratios, ensuring readability on both light and dark screens.
- **Whitespace and Balance:**
  Generous spacing between elements ensures a calm, breathable layout that improves focus on content.
  The centered llama icon adds a playful but subtle identity to the design.
- **Typography:**
  The system uses a clean sans-serif font (default Reflex / system font stack) optimized for legibility across devices.
- **Responsiveness:**
  The design automatically adapts to different screen sizes (desktop or tablet), maintaining alignment and proportions.

User Experience

- The UI welcomes users with example prompts (e.g., *“Ask a question”*, *“Solve a math problem”*) to guide first interaction.
- The chat history and main conversation areas maintain visual hierarchy, ensuring users immediately recognize the primary focus.
- Subtle animations and hover effects provide responsive feedback without visual noise.

🚀 Deliverables & References

Item	Description
Chatbot Application	A fully functional web-based chatbot that interacts with a locally hosted Ollama LLM (e.g., Llama 3.2).
Source Code Repository	Complete project on GitHub, including backend (`main.py`), Reflex frontend, and setup files.
Documentation	• `README.md` with setup instructions • This PRD (Created on Confluence) • Architecture diagram & Details • Optional Github demo deployment + screen shots

Reference Links

- GitHub Repository
- Ollama Documentation
- Reflex Framework Docs
- Pydantic AI
- Python
- uv
- Dev Containers