Running LLaMA 3.2 Locally with Node.js API and Custom Chat UI

This guide explains how I set up a **local LLM environment** using **Ollama**, created a **Node.js server** to expose the model through an API, and built a **custom HTML/CSS/JavaScript UI** as a chat interface.

☑ 1. Overview of the Architecture

Flow:

```
[Chat UI (HTML, CSS, JS)]  →  [Node.js Server API]  →  [Local LLM (Ollama)]
```

- **LLM**: Running locally (LLaMA 3.2) using **Ollama**.
- **Node.js Server**: Exposes API at `http://localhost:11432/api/generate`.
- **UI**: A simple chat interface built with HTML, CSS, and JavaScript, sending user messages to the Node.js API.

☑ 2. Running the LLM Model with Ollama

Install Ollama and pull LLaMA 3.2:

```
ollama pull llama3.2
```

Run the model:

```
ollama run llama3.2
```

By default, Ollama exposes an API on **port 11434**, but we use a **custom Node.js API** for interaction.

☑ 3. Node.js Server Setup

Created a Node.js Express server to act as a middleware between the chat UI and Ollama.

server.js Example:

```
const express = require('express');
const bodyParser = require('body-parser');
const fetch = require('node-fetch');
const cors = require('cors');
```

```
const app = express();
app.use(cors());
app.use(bodyParser.json());

app.post('/api/generate', async (req, res) => {
  const { prompt } = req.body;

  const response = await fetch('http://localhost:11434/api/generate', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ model: 'llama3.2', prompt })
  });

  const reader = response.body.getReader();
  const decoder = new TextDecoder();
  let result = '';

  while (true) {
    const { done, value } = await reader.read();
    if (done) break;
    result += decoder.decode(value);
  }

  res.send(result);
});

app.listen(11432, () => console.log('Node API running on
http://localhost:11432'));
```

Start the server:

```
node server.js
```

☑ 4. Chat UI (HTML, CSS, JavaScript)

index.html Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>LLM Chat UI</title>
  <style>
    body { font-family: Arial; background: #f9f9f9; }
    .chat-container { width: 400px; margin: 50px auto; background: #fff; padding:
20px; border-radius: 10px; }
    .messages { height: 300px; overflow-y: auto; margin-bottom: 10px; }
```

```
    .input-box { display: flex; }
    .input-box input { flex: 1; padding: 10px; }
    .input-box button { padding: 10px; }
  </style>
</head>
<body>
  <div class="chat-container">
    <div class="messages" id="messages"></div>
    <div class="input-box">
      <input id="userInput" placeholder="Type your message..."/>
      <button onclick="sendMessage()">Send</button>
    </div>
  </div>

  <script>
    async function sendMessage() {
      const input = document.getElementById('userInput');
      const message = input.value;
      input.value = '';

      addMessage('You: ' + message);

      const response = await fetch('http://localhost:11432/api/generate', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ prompt: message })
      });

      const text = await response.text();
      addMessage('AI: ' + text);
    }

    function addMessage(text) {
      const messages = document.getElementById('messages');
      const div = document.createElement('div');
      div.textContent = text;
      messages.appendChild(div);
      messages.scrollTop = messages.scrollHeight;
    }
  </script>
</body>
</html>
```

Open the `index.html` file in a browser.

☑ 5. How It Works

1. **User types a message** in the chat UI.
2. **JavaScript sends a POST request** to the Node.js API at `http://localhost:11432/api/generate`.
3. The Node.js server **forwards the prompt to Ollama's local API** on `http://localhost:11434/api/generate`.

4. Ollama **generates a response using the LLaMA 3.2 model** and streams it back to the Node.js server.
5. The Node.js server **returns the response to the UI**, which displays it in the chat window.

☑ Hardware & Requirements

- **Windows 10/11 (64-bit)**
- **Ollama installed**
- **Node.js 16+**
- **RAM**: Minimum 8 GB (16 GB recommended)
- **GPU**: Optional but recommended (NVIDIA CUDA)
- **Model**: LLaMA 3.2 or any Ollama-supported model

☑ Next Steps

- Add **streaming responses** to UI for real-time effect.
- Implement **conversation history**.
- Add **authentication** if exposing the Node API.
- Explore **LangChain integration** for advanced workflows.

#AI #LLM #Ollama #NodeJS #JavaScript #WebDevelopment #MachineLearning #LocalAI