Purwadhika
Digital Technology School

Full Stack AI Software Development

AI Integration in Applications

Job Connector Program

The Modern AI App Architecture

Before writing code, you must understand how modern AI works within an application.

Presentation Layer (UI)

Captures user intent (text, voice, or screen actions) and handles streaming responses to keep the UI feeling fast.

Orchestration Layer

This is your backend (Node.js, Go, etc.). It manages the "Prompt," handles security, and decides which tools the AI needs.

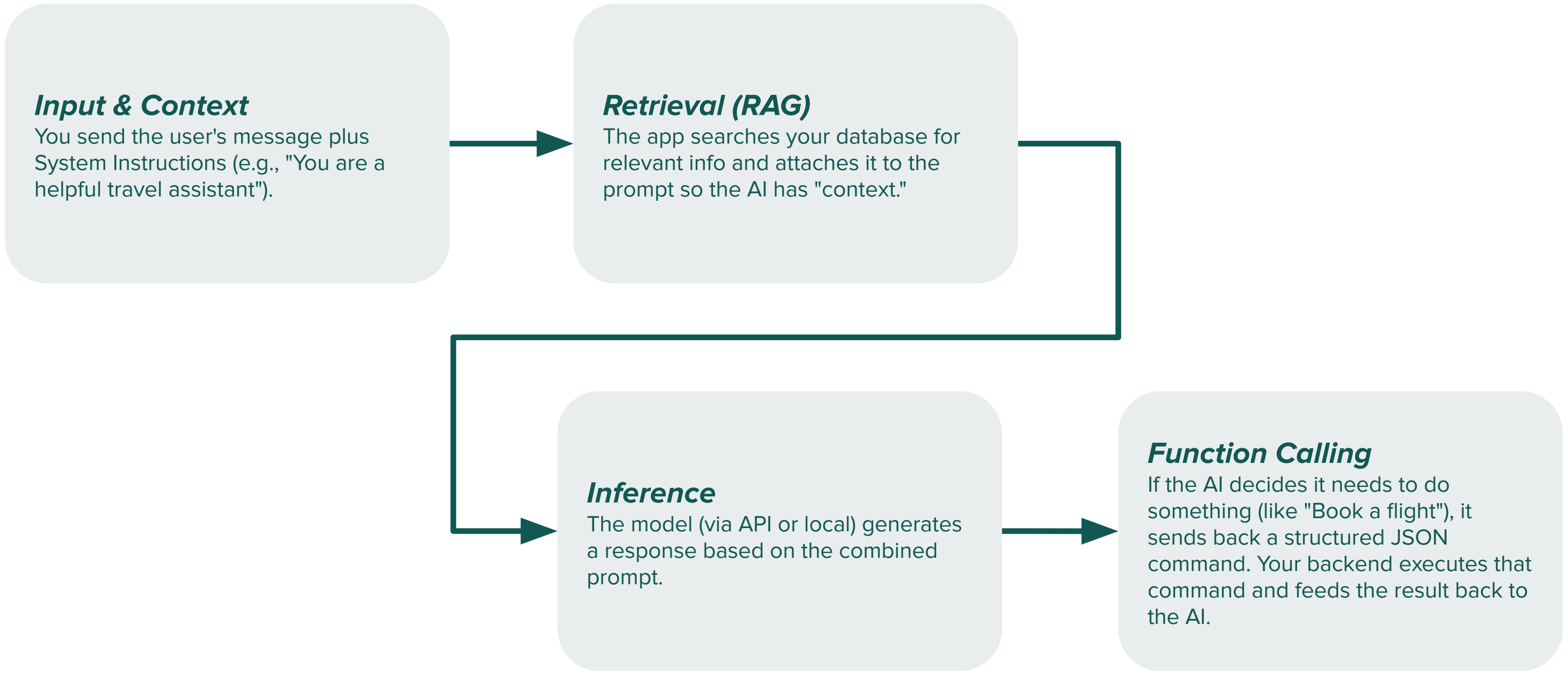
Intelligence Layer (Model)

The LLM (Large Language Model) that processes the request.

Knowledge Layer (RAG)

A Vector Database that provides the AI with specific data it wasn't originally trained on (like your private user data).

The Core Workflow: From Prompt to Action



Input & Context
You send the user's message plus System Instructions (e.g., "You are a helpful travel assistant").

Retrieval (RAG)
The app searches your database for relevant info and attaches it to the prompt so the AI has "context."

Inference
The model (via API or local) generates a response based on the combined prompt.

Function Calling
If the AI decides it needs to do something (like "Book a flight"), it sends back a structured JSON command. Your backend executes that command and feeds the result back to the AI.

Key Technical Concepts

- **Tokenization**: AI doesn't read words; it reads "tokens" (chunks of characters). You are billed and limited by these.
- **Embeddings**: Turning text into a list of numbers so the computer can calculate how "similar" two pieces of text are.
- **Streaming**: Instead of waiting 10 seconds for a full paragraph, you pipe the response bit-by-bit to the frontend for a better UX.

Creating a Chatbot

You will need **OpenRouter** to access hundreds of different AI models through a single, unified API.

Why OpenRouter?

- For a beginner, OpenRouter is often the best "all-in-one" start because:
- Unified API: You use the same code to talk to OpenAI, Anthropic, or Meta models.
- Model Variety: Switch from a heavy model (GPT-4) to a cheap, fast model (Llama 3) just by changing a string.
- No Multiple Credits: You top up one balance on OpenRouter instead of having separate bills for OpenAI, Google, and Anthropic.
- Free Models: Access "free" versions of top-tier models (like deepseek-v3:free) for testing.

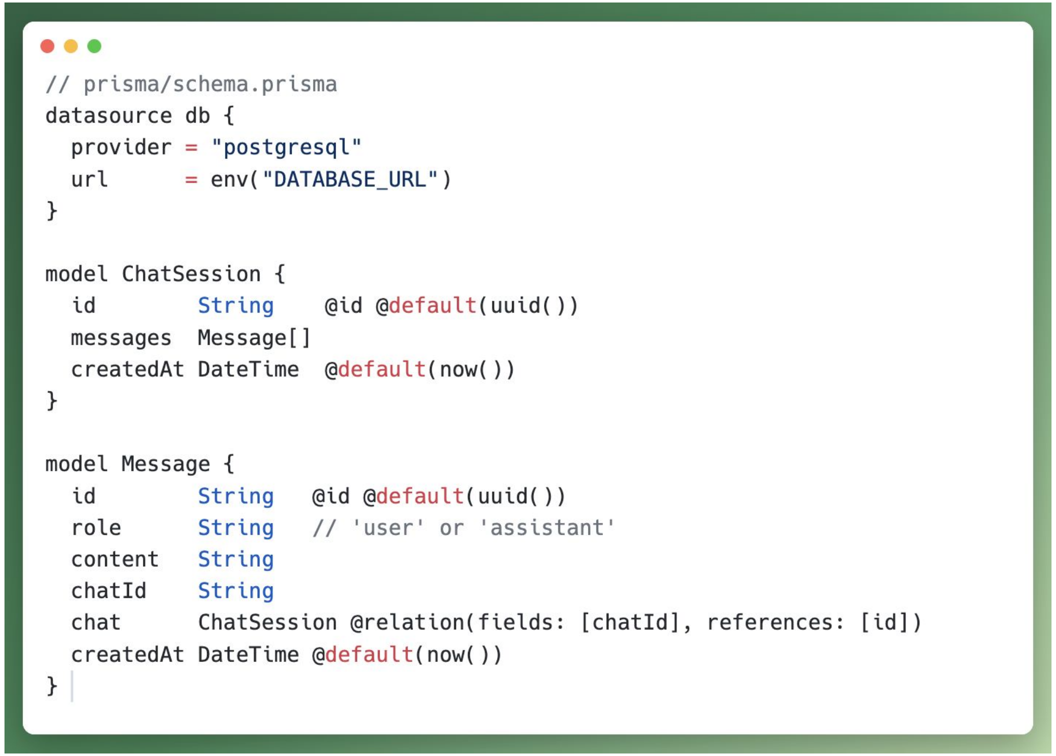
Create you OpenRotuer account here: https://openrouter.ai/

OpenRouter

Creating a Chatbot

Database Setup (PostgreSQL + Prisma)

Define your schema to store chat history and (optionally) embeddings.

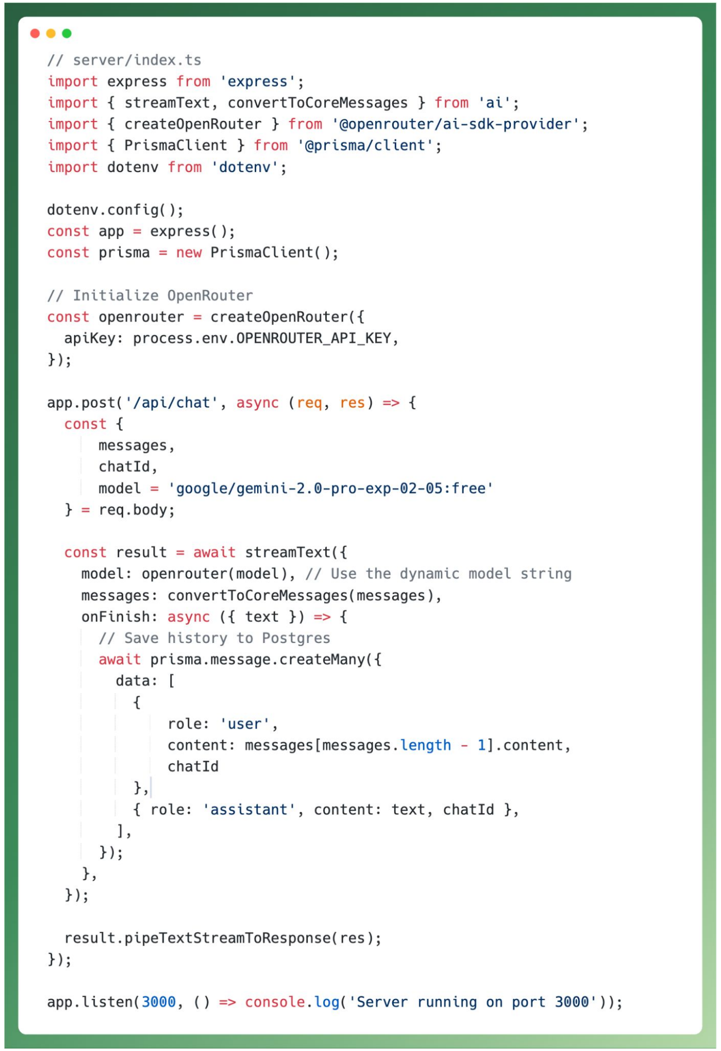


```
// prisma/schema.prisma
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model ChatSession {
  id        String    @id @default(uuid())
  messages  Message[]
  createdAt DateTime  @default(now())
}

model Message {
  id        String   @id @default(uuid())
  role      String   // 'user' or 'assistant'
  content   String
  chatId    String
  chat      ChatSession @relation(fields: [chatId], references: [id])
  createdAt DateTime @default(now())
}
```

Creating a Chatbot

Backend Logic (Express + TS + OpenRouter)

- In your Express server, you initialize the OpenRouter provider. You can now use any model available on OpenRouter by using its string identifier (e.g., anthropic/claude-3.5-sonnet).
- Use the AI SDK to stream responses and save them to the database.

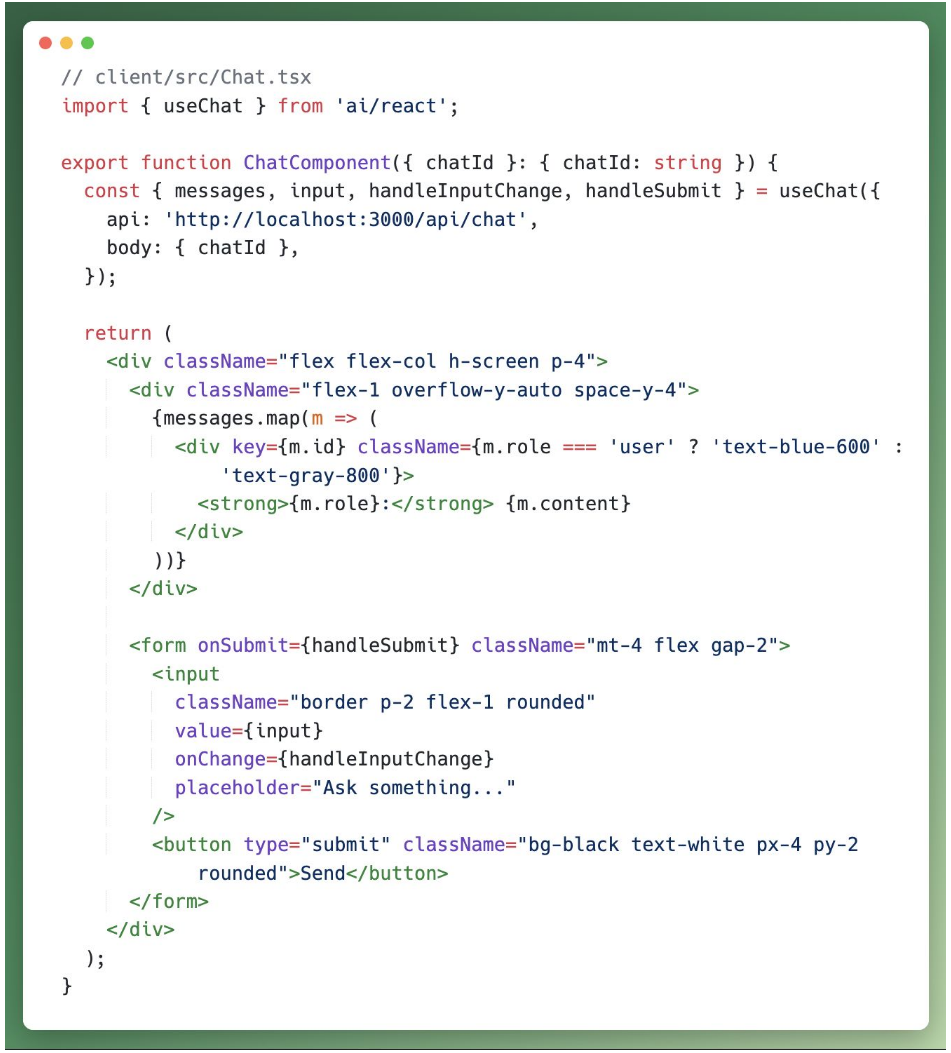


```
// server/index.ts
import express from 'express';
import { streamText, convertToCoreMessages } from 'ai';
import { createOpenRouter } from '@openrouter/ai-sdk-provider';
import { PrismaClient } from '@prisma/client';
import dotenv from 'dotenv';

dotenv.config();
const app = express();
const prisma = new PrismaClient();

// Initialize OpenRouter
const openrouter = createOpenRouter({
  apiKey: process.env.OPENROUTER_API_KEY,
});

app.post('/api/chat', async (req, res) => {
  const {
      messages,
      chatId,
      model = 'google/gemini-2.0-pro-exp-02-05:free'
  } = req.body;

  const result = await streamText({
    model: openrouter(model), // Use the dynamic model string
    messages: convertToCoreMessages(messages),
    onFinish: async ({ text }) => {
      // Save history to Postgres
      await prisma.message.createMany({
        data: [
          {
              role: 'user',
              content: messages[messages.length - 1].content,
              chatId
          },
          { role: 'assistant', content: text, chatId },
        ],
      });
    },
  });

  result.pipeTextStreamToResponse(res);
});

app.listen(3000, () => console.log('Server running on port 3000'));
```

Creating a Chatbot

Frontend Implementation (Vite + TS)

Leverage the useChat hook for instant streaming UI.



```tsx
// client/src/Chat.tsx
import { useChat } from 'ai/react';

export function ChatComponent({ chatId }: { chatId: string }) {
  const { messages, input, handleInputChange, handleSubmit } = useChat({
    api: 'http://localhost:3000/api/chat',
    body: { chatId },
  });

  return (
    <div className="flex flex-col h-screen p-4">
      <div className="flex-1 overflow-y-auto space-y-4">
        {messages.map(m => (
          <div key={m.id} className={m.role === 'user' ? 'text-blue-600' :
              'text-gray-800'}>
            <strong>{m.role}:</strong> {m.content}
          </div>
        ))}
      </div>

      <form onSubmit={handleSubmit} className="mt-4 flex gap-2">
        <input
          className="border p-2 flex-1 rounded"
          value={input}
          onChange={handleInputChange}
          placeholder="Ask something..."
        />
        <button type="submit" className="bg-black text-white px-4 py-2
            rounded">Send</button>
      </form>
    </div>
  );
}
```

Introduction to n8n

n8n is a "Fair-Code" workflow automation tool. For developers, it acts as a visual backend.

- **Why use it?** Instead of writing complex Express logic for every AI feature, you can "drag and drop" nodes to connect your Chatbot to Google Sheets, Email, or Slack.
- **Nodes:** Each block in n8n is a node (e.g., a "Postgres Node" or an "OpenAI Node").
- **Webhooks:** You can trigger n8n workflows by sending a request from your Express server to an n8n Webhook URL.

By moving the backend to n8n, you transition from a "Chatbot" to an "Autonomous Agent."

Enhance Chatbot using n8n

Instead of the Chatbot only "talking," you can use n8n to give it tools.

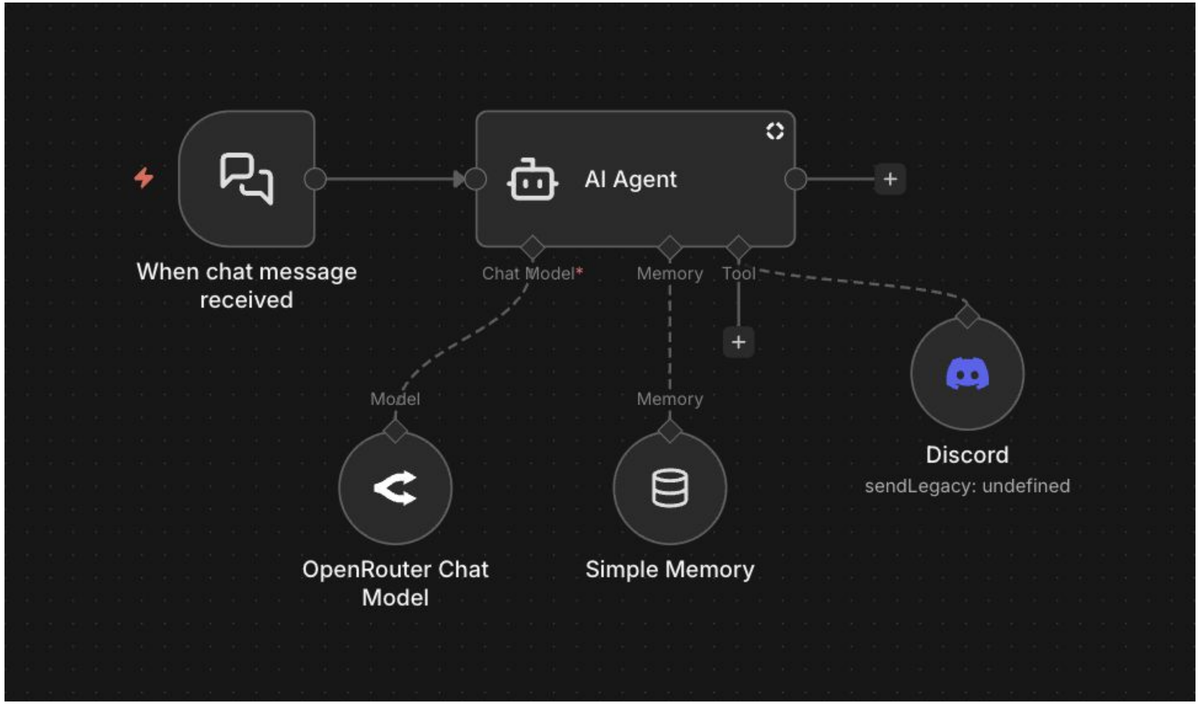
Use Case: Intelligent Lead Qualifier & CRM Automator

The n8n Workflow Architecture

- Vite Frontend (Chat Widget): Captures the initial lead inquiry (e.g., "I need a website for my real estate business").
- n8n Webhook: Receives the message and a unique leadId.
- AI Agent Node (The Sales Strategist):
    - Model: OpenRouter (openai/gpt-4o or anthropic/claude-3.7-sonnet).
    - Context: Connected to Postgres to check current service prices and availability.
    - Task: Ask qualifying questions (Budget, Timeline, Scope).
- Tool Use (Function Calling):
    - Postgres Tool: If the lead is qualified (e.g., budget > $5k), the AI automatically updates the lead status to "Qualified" in your DB.
    - Google Calendar/Calendly Node: If qualified, the AI provides a booking link to the user.
    - Slack/Email Node: Notifies your sales team: "New high-value lead qualified: [Name] - [Budget]."

n8n Workflow Example

https://drive.google.com/file/d/15beViE8ICc2aiUmhH20EXj6nW0Z3zurl/view?usp=sharing



Thank You!