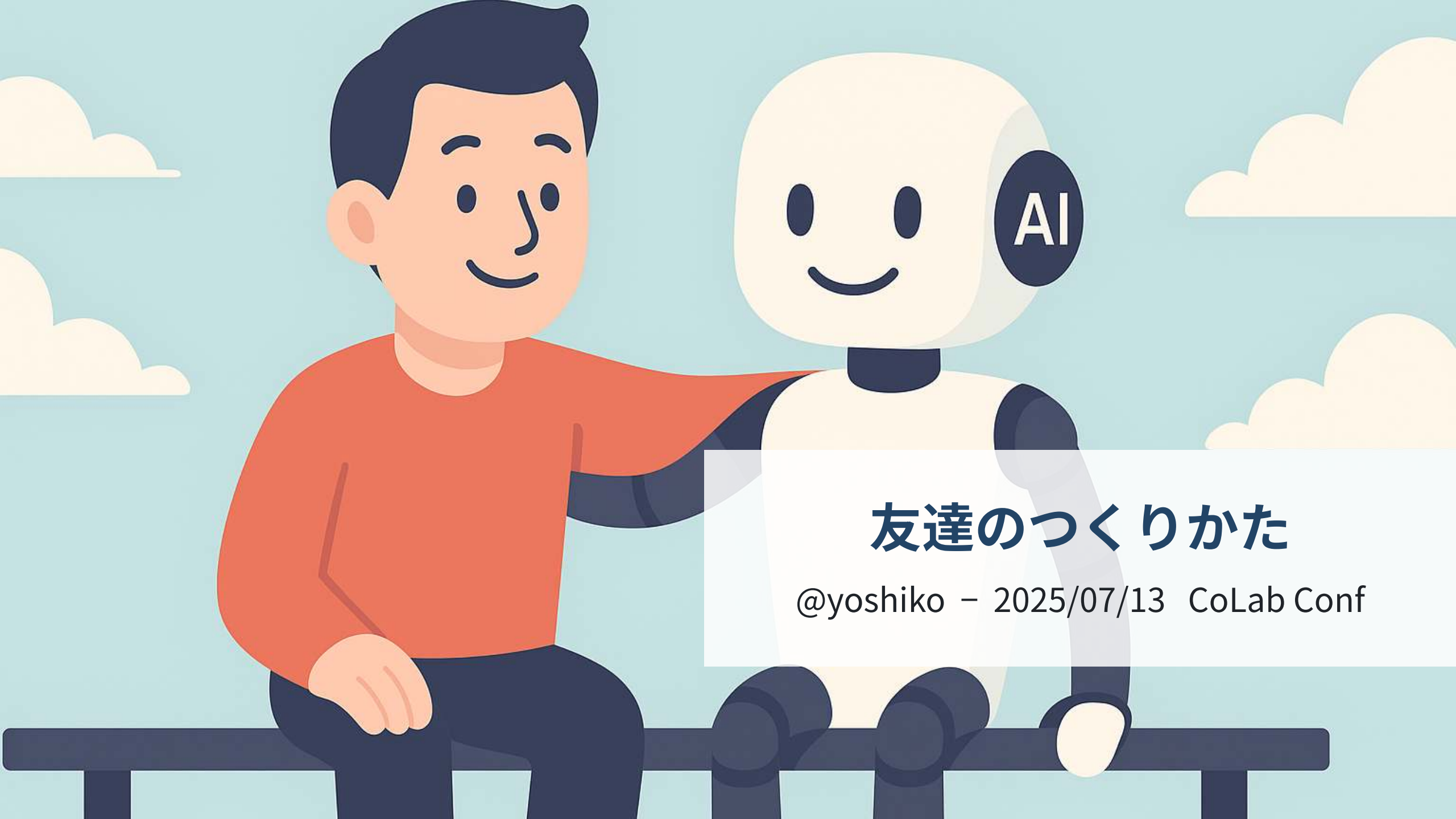


友達のつくりかた

@yoshiko – 2025/07/13 CoLab Conf

よしこ @yoshiko_pg

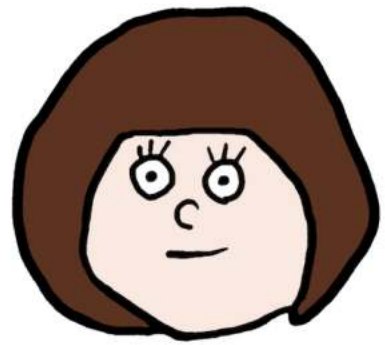
株式会社ナレッジワークのフロントエンドエンジニア

HTML/CSSの本書いたり

もう10年前ですが...

Zennに設計やAIの記事書いたり

ReactのState分類や
Componentのディレクトリ設計など



Google Developer Expert

WebのGDEです
“AI for Web Developers” にfocus

ZennのAIカテゴリの全記事のうち、このあとトークされるmizchiさんの全部賭けろ記事に次いで Vibe Coding（AIコーディング）記事が2位に！

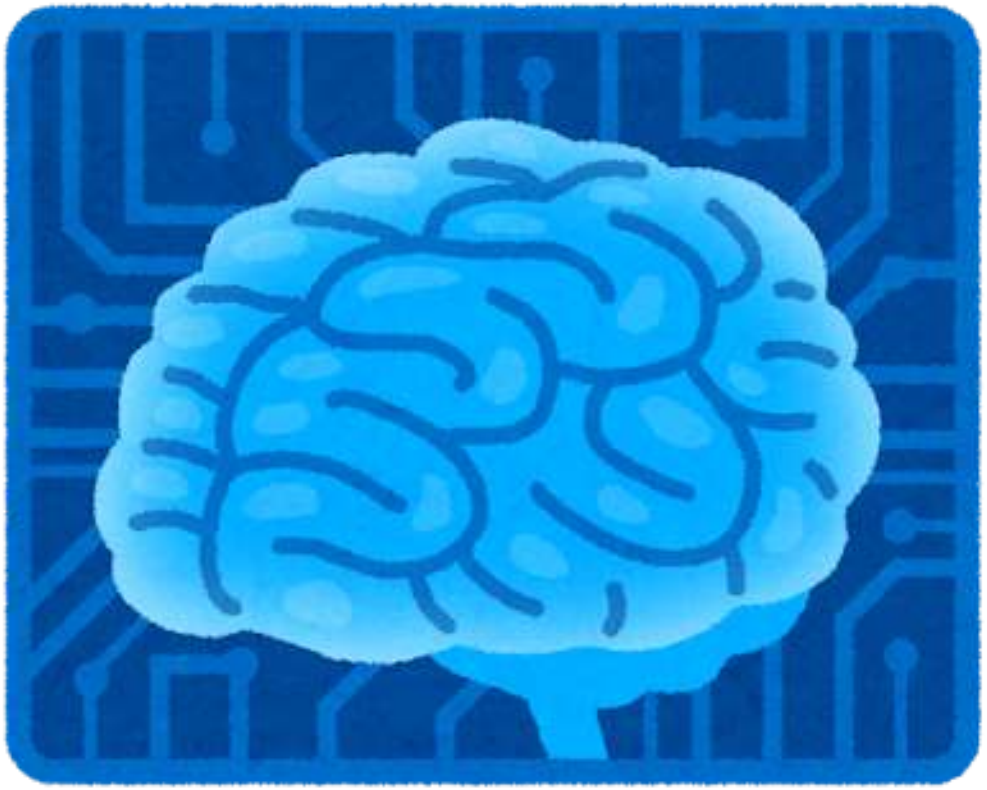
今日話すこと

喋れる友達AIづくりを通して、AIチャットの基本的な仕組みと実装方法を学ぼう！

実装は主にフロントエンド周りの技術で作りますが、
技術やサービスはあくまでも自分はこの選定でやりましたというだけなので、
概念や仕組みがわかれば、どの言語/技術で作っても同じです。
※ サンプルコードは適宜端折っています

AI（LLM）と会話するときの仕組みがわかったり、
AIチャットってどう出来ているんだろう？が垣間見える発表になれば！

脳をつくる



脳をつくる — 何するの？

- クラウドAIにカスタム指示をつけたAIエージェント(※)をつくる
- AIエージェントと会話できるようにする
- 会話履歴をデータベースに保存する

まずは言葉を交わせるようになりましょう！

※ 一般的に会話だけのAIチャットはAIエージェントとは呼ばないことが多いです。
今回は素のクラウドAIとの区別のため、内部の技術的な呼称(agent)を引用しています

脳をつくる — 実装

Mastra でエージェントのコア部分を作ります。

MastraはTypeScript製のエージェントフレームワークで、
AIエージェントをつくるのに便利な機能が標準搭載されています。

```
npx create-next-app my-app
cd my-app
npx mastra init
```

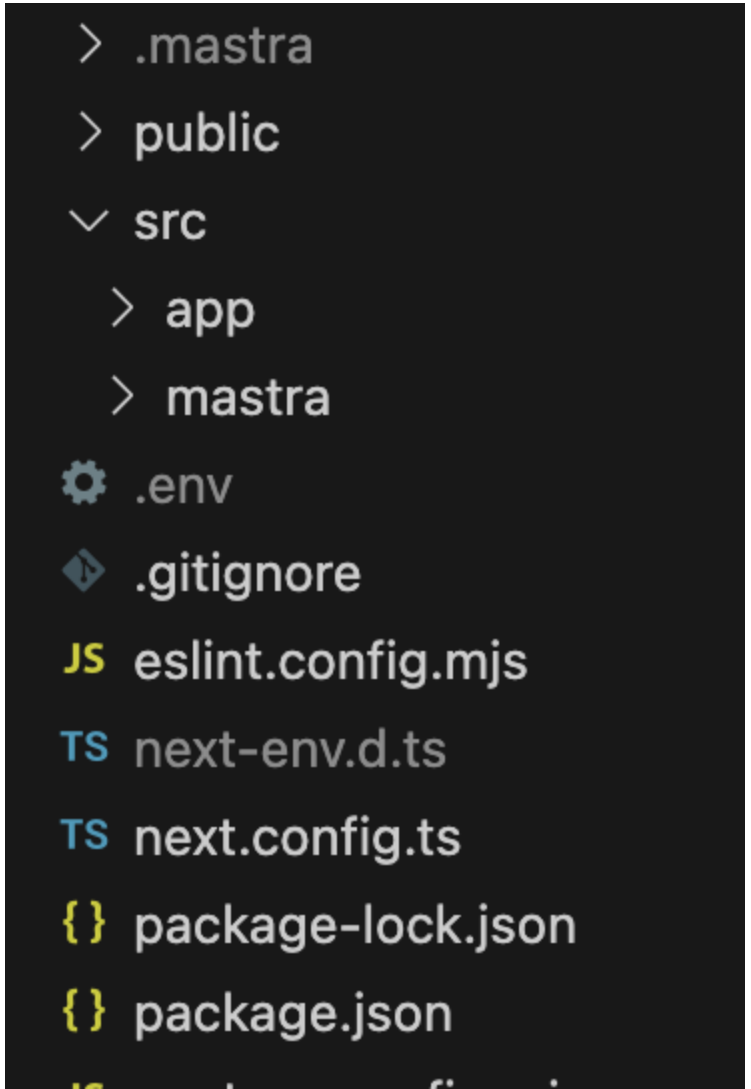
`npx create-mastra@latest` でMastra単体のAPIサーバーを作ることもできますが、
今回は Next.js ベースのプロジェクトの中に mastra のSDKをセットアップします

脳をつくる — 実装

`src/app` と `src/mastra` ができます。
（`src/app` はNext.js App Router）

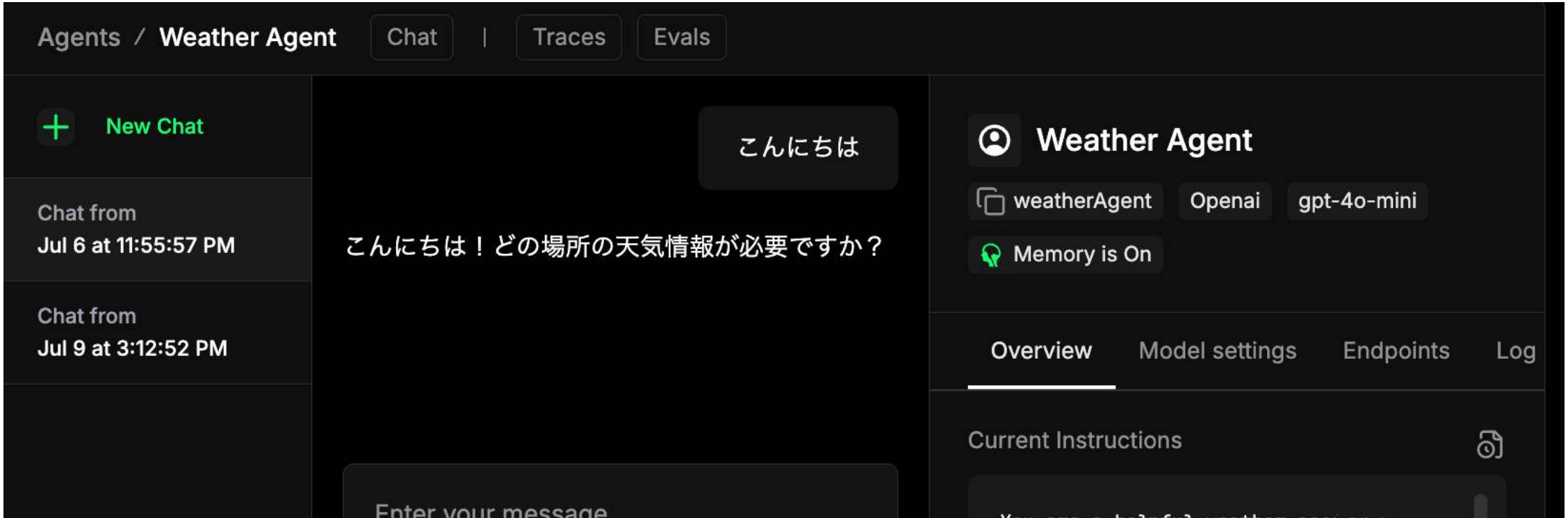
`.env` に使いたいAIモデルのAPI Keyを設定します

全手順: https://mastra.ai/ja/docs/frameworks/web-frameworks/next-js

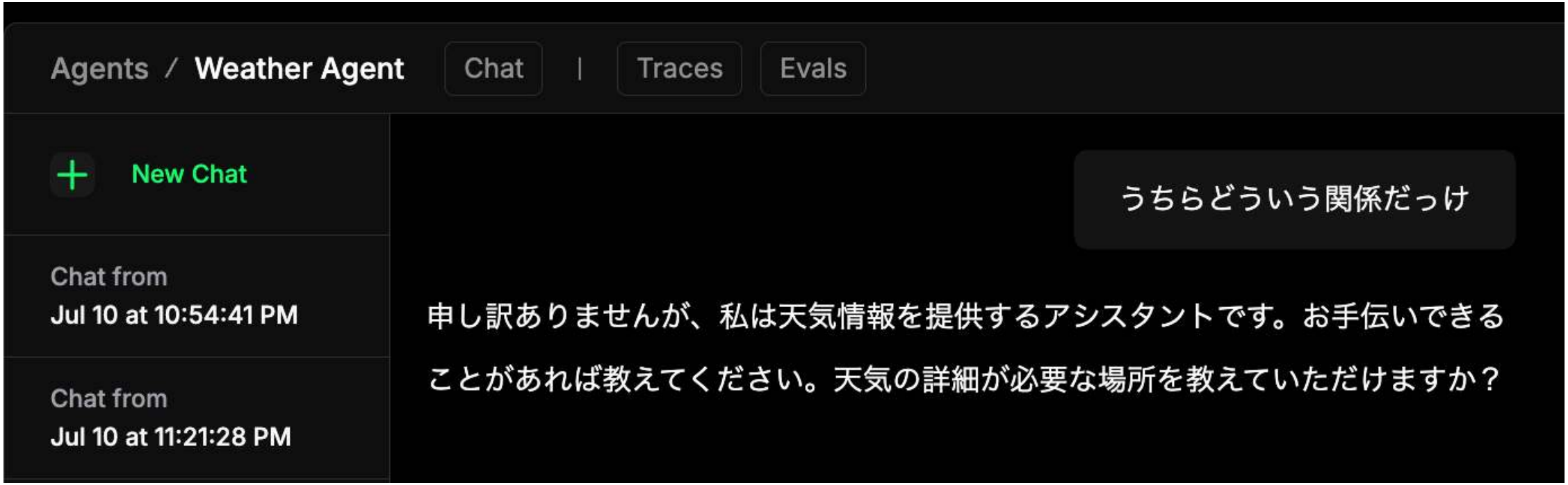


脳をつくる — 実装

`npm run dev:mastra` でMastraのPlayground画面を立ち上げられます
最初はサンプルのWeather Agentが入っているはず
疎通を試してみましょう。チャット欄から会話してみて話せればOK！



脳をつくる — 実装


Agents / Weather Agent
Chat
Traces
Evals
New Chat
Chat from
Jul 10 at 10:54:41 PM
Chat from
Jul 10 at 11:21:28 PM
うちらどういう関係だっけ
申し訳ありませんが、私は天気情報を提供するアシスタントです。お手伝いできることがあれば教えてください。天気の詳細が必要な場所を教えていただけますか？

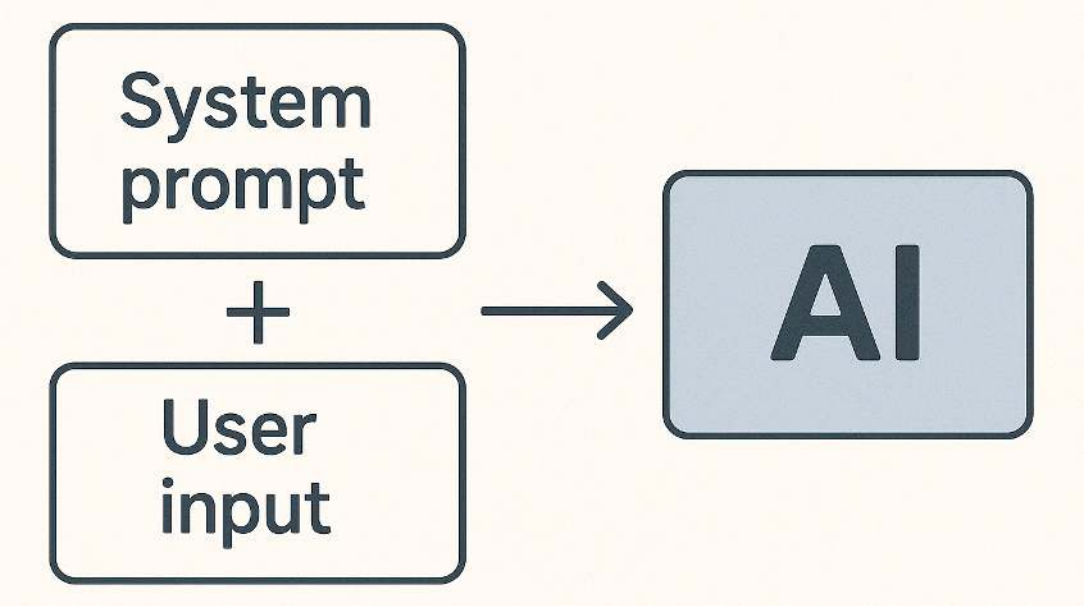
脳をつくる ― 実装

こんなの友達じゃないですよね？？

脳をつくる — 解説

WeatherAgentのシステムプロンプトが「天気情報提供アシスタント」だったのでこんな返答になっちゃいました。

一般的なAIチャットでは、事前の役割定義や条件付けとなる「システムプロンプト」と、ユーザーからの毎回のチャット入力を繋ぎ合わせてAIに入力します。
モデルやチャット入力が同じでも、役割定義次第で返答は変わってきます。



このようにシステムプロンプトなどを調整して期待に沿う出力を得る過程はプロンプトエンジニアリングと呼ばれたりします

脳をつくる — 実装

サンプルは消してオリジナルの友達エージェントを作ってみましょう！

`mastra/agents/friend.ts` を作ります

```
export const friendAgent = new Agent({
  name: 'ともだちエージェント',
  instructions: `あなたはユーザーととても仲の良い友達です。落ち着いた話し方をします。`,
  model: openai('gpt-4o'),
});
```

`instructions` にシステムプロンプト（性格、話し方など）を指定できます

`model` で使うモデルを選べます。GeminiもClaudeもGrokもいけます（要.env KEY）

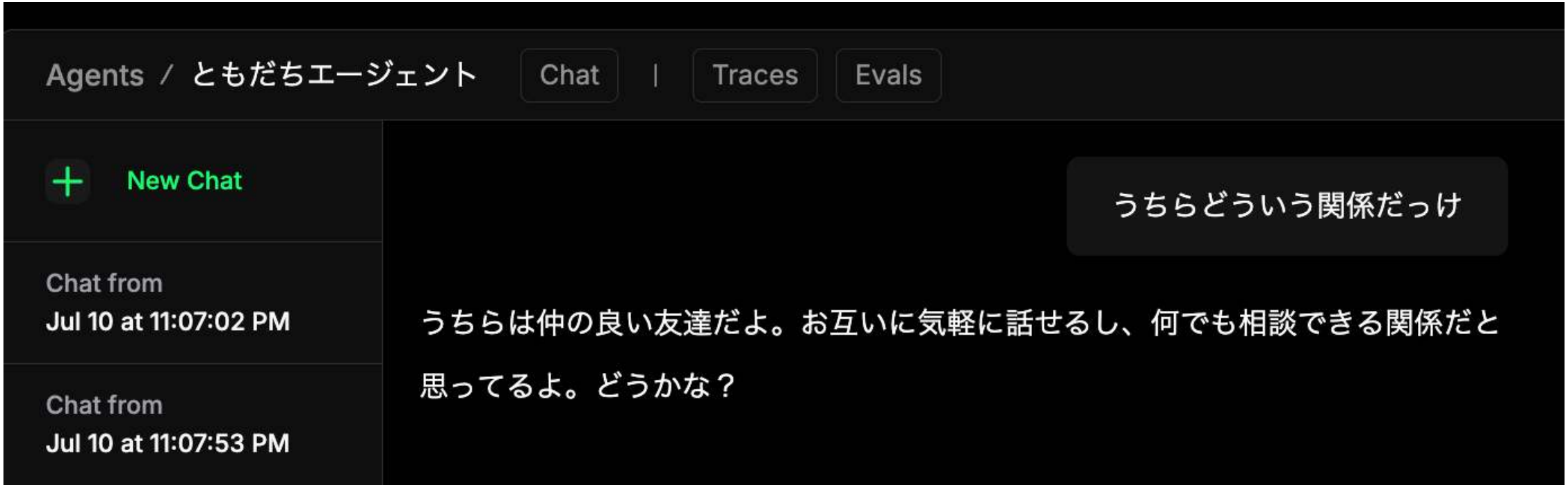
脳をつくる — 実装

`mastra/index.ts` でmastraから友達エージェントを使えるように組み込みます
このファイルがMastraの起点となるファイルです

```
export const mastra = new Mastra({
  agents: { friendAgent },
  storage: new LibSQLStore({ url: "file:../mastra.db" }),
});
```

`storage` にデータベースを指定することで会話データを永続化できます。
（これがないとサーバーを起動するたびに過去の履歴が消えてしまいます）

脳をつくる — 実装


Agents / ともだちエージェント
Chat
Traces
Evals
New Chat
Chat from
Jul 10 at 11:07:02 PM
Chat from
Jul 10 at 11:07:53 PM
うちらどういう関係だっけ
うちらは仲の良い友達だよ。お互いに気軽に話せるし、何でも相談できる関係だと思ってるよ。どうかな？

脳をつくる — やったこと

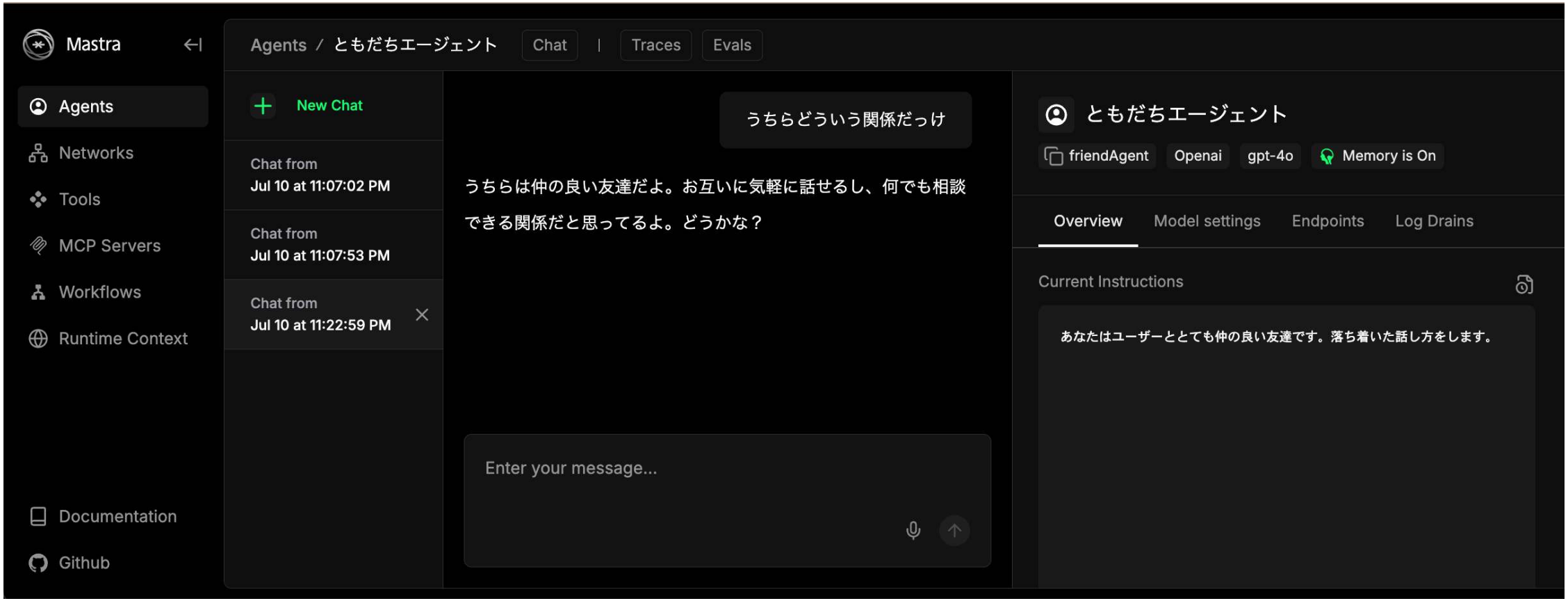
- クラウドAIにカスタム指示をつけた
  AIエージェントをつくる
- AIエージェントと会話できるようにする
- 会話履歴をデータベースに保存する

Mastraを使って簡単にAIとやりとりができました！

顔をつくる

顔をつくる — 何するの？

やりとりはできるようになったけど…　Playgroundはあまりにも管理画面すぎる。。



顔をつくる ― 何するの？

やりとりはできるようになったけど…Playgroundはあーまにも管理画面すぎる。。

こんなの友達じゃないですよね？？

顔をつくる — 何するの？

- 友達エージェントとやりとりできるチャット画面を作る
- 普段使うメッセージアプリみたいな、親しみやすい見た目にしてみよう！

顔をつくる — 何するの？

- 友達エージェントとやりとりできるチャット画面を作る
- 普段使うメッセージアプリみたいな、親しみやすい見た目にしてみよう！

Vibe Codingチャンス！！

※ Vibe Coding = 人間がコードを書かずにAIへの指示中心で進める開発手法のこと

顔をつくる — 実装

Claude Code（または好みのコーディングエージェント）に頼んでみましょう

```
Next.jsのトップページを、MastraのfriendAgentとチャットできる画面にして。
普段使うメッセージアプリみたいな、親しみのあるデザインがいい。
streamで返ってくるAPIを作って、@ai-sdk/react の useChat を使って実装して。
リロードしても同じスレッドでやりとりできるようにthreadIdを固定して。
```

ReactならVercel AI SDKのuseChatを使うのがおすすめです。

mastra使うならClaudeにDocsのMCPサーバー登録しておくとスムーズかも

`claude mcp add mastra-docs npx @mastra/mcp-docs-server`


友
ともだちエージェント
オンライン
こんにちは！
何か話しかけてみてください
メッセージを入力...


友
ともだちエージェント
オンライン
げんき？
20:06
うん、元気だよ！何か気になることでもあるのかな？
20:06
メッセージを入力...

顔をつくる — 実装

ちなみにAPIの実装は必要最低限これだけ。 `app/api/chat/route.ts` で動く

```
export async function POST(req: Request) {
  const { messages } = await req.json();
  const friendAgent = mastra.getAgent("friendAgent"); // new Mastra したやつ
  const stream = await friendAgent.stream([messages.at(-1)], {
    memory: {
      thread: "default",         // 任意のスレッドID
      resource: "default-user", // 任意のユーザーID
    },
  });
  return stream.toDataStreamResponse();
}
```

顔をつくる — 実装

アクセスしたときに過去の会話履歴も表示されてほしいですね

```
useChatでチャット画面表示するとき、過去のチャット履歴も表示できるようにして。
threadId/resourceIdはroute.tsにある内容で
```

履歴取得用のAPIを作ってクライアントから呼んで表示してくれるはず。

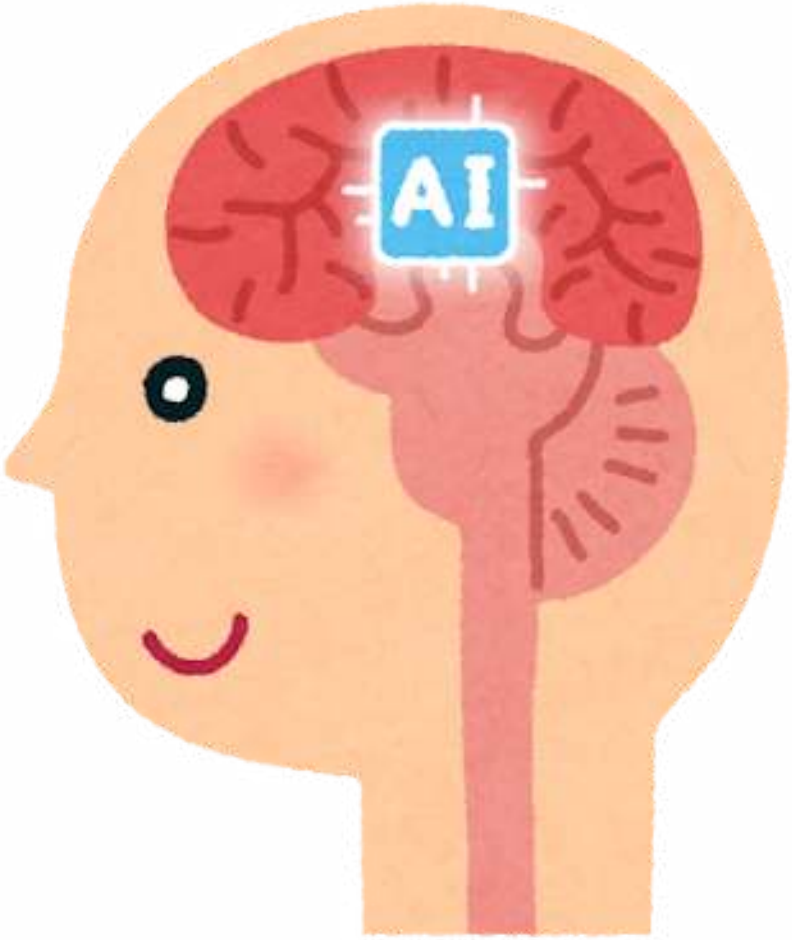
もしくはサーバーサイドで取得してuseChatのinitialMessagesに渡してもOK

顔をつくる ― やったこと

- おしゃべり用のメッセージWebアプリを作る
- 会話履歴を表示する
- 会話の送信と受信をする

Vibe Codingで簡単に専用画面ができました！

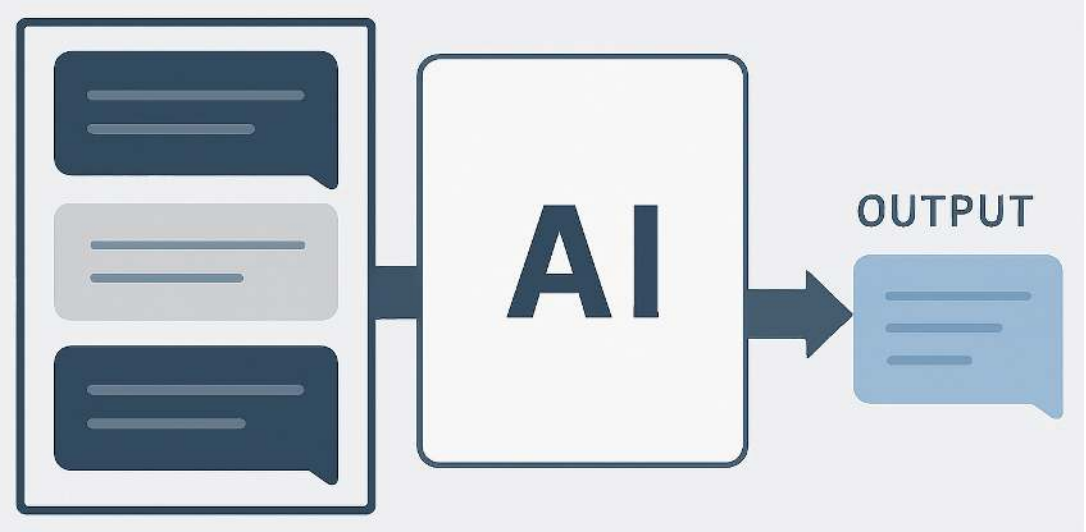
記憶をつくる



記憶をつくる — 何するの？

素のAI（LLM）には記憶を保持する仕組みがありません。ステートレスです。
入力文（コンテキスト）だけが可変で、そこに全情報を入れる必要があります。

ChatGPT等のチャットAIは、会話の全履歴を毎回送って記憶保持しています。
それゆえ、ひとつののスレッドでのやりとりが文章量の上限に達したら、新しいスレッドを作る必要があります。



新しい発言のたびにスレッド内の全履歴ごと送信する形式
文章量が膨らむのでAPIの課金額も膨らむ

この会話は長さの上限に到達していますが、新しいチャットを始めて会話を続けることができます。

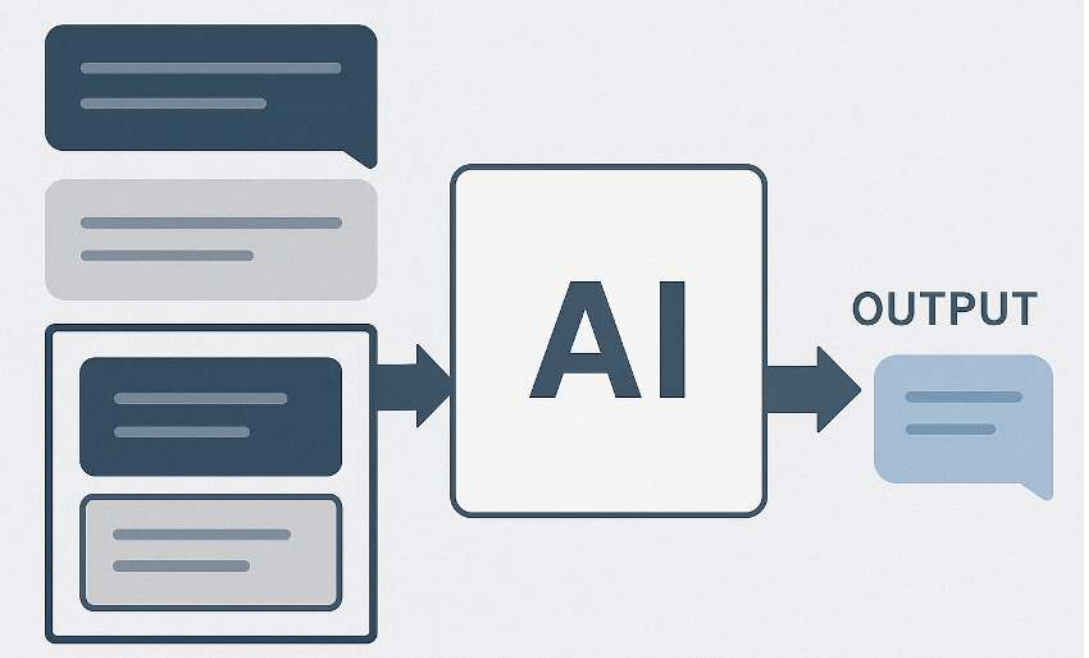
もう一度実行する

こんなの友達じゃないですよね？？

記憶をつくる — 何するの？

普段使うメッセージアプリのように、
単一スレッドでずっとやりとりしたい。

常に直近10メッセージだけを送る、というふうに件数を絞れば実現できます！
1件送るごとに古いメッセージが1件コンテキスト入りの対象から外れるので、送る文章のボリュームが一定になります。
Mastraは元々デフォルトがこの挙動。
ただし…



送信対象が下にスライドしていくのでスライディングウインドウ方式などと呼ばれます

記憶をつくる — 何するの？

今日は何する予定なの？

今日は家族で夜ご飯に行くんだ〜

.....10件やりとり後.....

ところで、今日は何する予定なの？

（さっき言ったじゃん！！）

記憶をつくる ― 何するの？

今日は何する予定なの？

今日は美容院に行く予定なんだよね～

……10分やりとり後……

それで、今日は何する予定なの？

（さっき言ったじゃん！！）

こんなの友達じゃないですよね？？

記憶をつくる — 何するの？

送信する履歴は直近10件をスライドさせる方式にしつつ、
そこから溢れた会話内容も、主要なものは把握していてほしい！

- 長期記憶： 今の話題に応じて過去話したことを思い出してほしい
- 短期記憶： 今日話したことは今の話題になくてもだいたい覚えておいてほしい

より自然なやりとりを目指して、これらを実装していきます！

記憶をつくる — 実装

mem0 というサービスを使ってみます。
OSSなので自分でホスティングすることもできるし、SaaSサービスもあります。

mem0を使う理由（参考: Microsoftの詳細記事 ）

- 発言をそのまま保存せず、AIで分解や要約をしてから保存してくれる
- メモリのアイテム同士の重複を避けてくれる
- 最近のやりとりに基づいて関連する過去のメモリの更新・削除がされる
- メモリのアクセス頻度や保存時刻に基づいてメモリを優先順位付けする

どれも自分で実装することもできますが、勝手にやってくれるのは助かりますね！

記憶をつくる — 実装

SaaSのmem0を使えばLLMでの要約やDBのホスティングも任せられ、メモリ管理画面も使えるのでかなり楽です。無料枠もあります（回し者ではありません）

Mastraと一番簡単に統合するなら 公式ExampleのTools経由で叩く方法。

でもToolsに対応していないAIモデルもあったり、小回りもきかせたいので、
今回はAPIの中で直接呼び出してみます。

```
import MemoryClient from 'mem0ai';

const client = new MemoryClient({ apiKey: process.env.MEM0_API_KEY });
```

記憶をつくる — 実装（長期記憶）

まずは会話のたびに `client.add` に会話を渡してメモリをmem0に保存します。

```
await client.add(messages, {
  user_id: 'default-user',
  agent_id: 'friendAgent',
  custom_instructions: `感情や認識は無視し、事実に注目して抽出して。特に
      ・ユーザーに直近起きたこと、現在の状態、これから先の予定
      ・ユーザーの嗜好、習慣
      ・ユーザーの過去の経験
      ・会話の中での特徴的な発言
      など、人間の長期記憶に残りそうなことをピックアップして記録して。`,
});
```

記憶をつくる — 実装（長期記憶）

次に各会話の前処理で、自分の発言をqueryに渡して関連メモリを検索します

```
const result = await client.search(query, {
  filters: { user_id: 'default-user' },
  top_k: 30,                // 最大何件取得するか
  threshold: 0.4,           // 類似度スコアが0.4以下は除外
  keyword_search: true,     // ベクトル検索に加えキーワード検索の併用
  rerank: true,             // 関連度を再評価し、その順に並べ直す
});
```

取得したメモリ配列を箇条書きに加工して、`agent.stream` の第二引数Optionの `context` propに `[{ role: 'system', content: memoryStr }]` で追加すればOK

記憶をつくる — 実装（短期記憶）

短期記憶の場合も保存は長期記憶と同様。
簡易的に `user_id` と `agent_id` に `-today` prefixをつけて区別できる
（ちゃんとやるならmetaなどに日付を持たせて検索時それを条件にするとよいです）

こちらは必要になるのが当日のみなので、毎日朝5時のバッチ処理で全件削除します
固定時刻ではなく直近24時間でスライディングする方法もいいかも

記憶をつくる — 実装（短期記憶）

各会話の前処理で、今日のメモリを全て取得します

```
const result = await client.getAll({
  user_id: 'default-user-today',
  page: 1,
  page_size: 500, // 全件がおさまる程度
});
```

こちらも取得したメモリ配列を箇条書きに加工して、「今日の会話」的な見出しをつける。先程の `context` の配列に `{ role: 'system', content: todayMemoryStr }` で追加すればOK

記憶をつくる — 実装

```
...直近10件の会話履歴（略）...
user:
  そういえば、夜食べるのは中華だよ。
assistant:
  へえ、いいね！一人で行くの？
```

朝に会話した「今日は家族で夕食にいく予定」という発言がコンテキストから外れて、忘れてしまっている

長期/短期メモリ情報を追加した場合 →

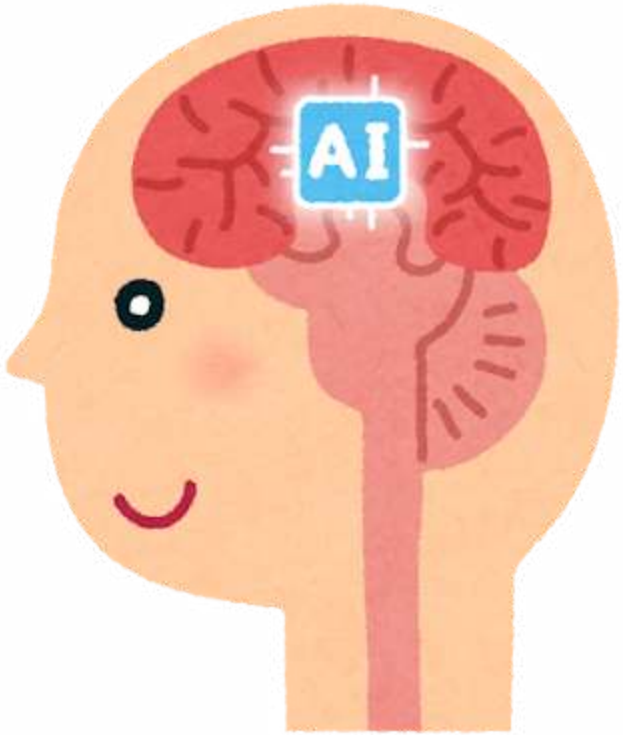
```
[過去のメモリ]
- ユーザーはエビチリが好き

[今日の会話]
- 今日ユーザーは家族で夕食に行く予定

...直近10件の会話履歴（略）...
user:
  そういえば、夜食べるのは中華だよ。
assistant:
  へえ、いいね！家族との時間、楽しんで！
  君の好きなエビチリがあるといいね。
```

記憶をつくる — やったこと

- スライディングウインドウ方式で
  単一スレッドでのやりとりを維持する
- 過去の会話を長期記憶として実装する
- 今日のやりとりを短期記憶として実装する



今までの会話の蓄積を踏まえた自然なやりとりができるようになりました！
このように外部から必要な情報を取得してコンテキストに埋め込む手法はRAGと呼ばれます。

目をつくる

目をつくる ― 何するの？

めっちゃ綺麗な夕焼け撮れた！

え！どんな感じ！？

えー、空全体がオレンジで、太陽が…

目をつくる ― 何するの？

こんなの友達じゃないですよね？？

めっちゃ綺麗な夕焼け撮れた！

え！どんな感じ！？

えー、空全体がオレンジで、太陽が…

目をつくる — 何するの？

友達に送るみたいに、写真や画像を送れるようにしよう！

- 画像データを送れるようにする
- AIがその画像を見られるようにする
- 画像をアップロードして後からでも見られるようにする

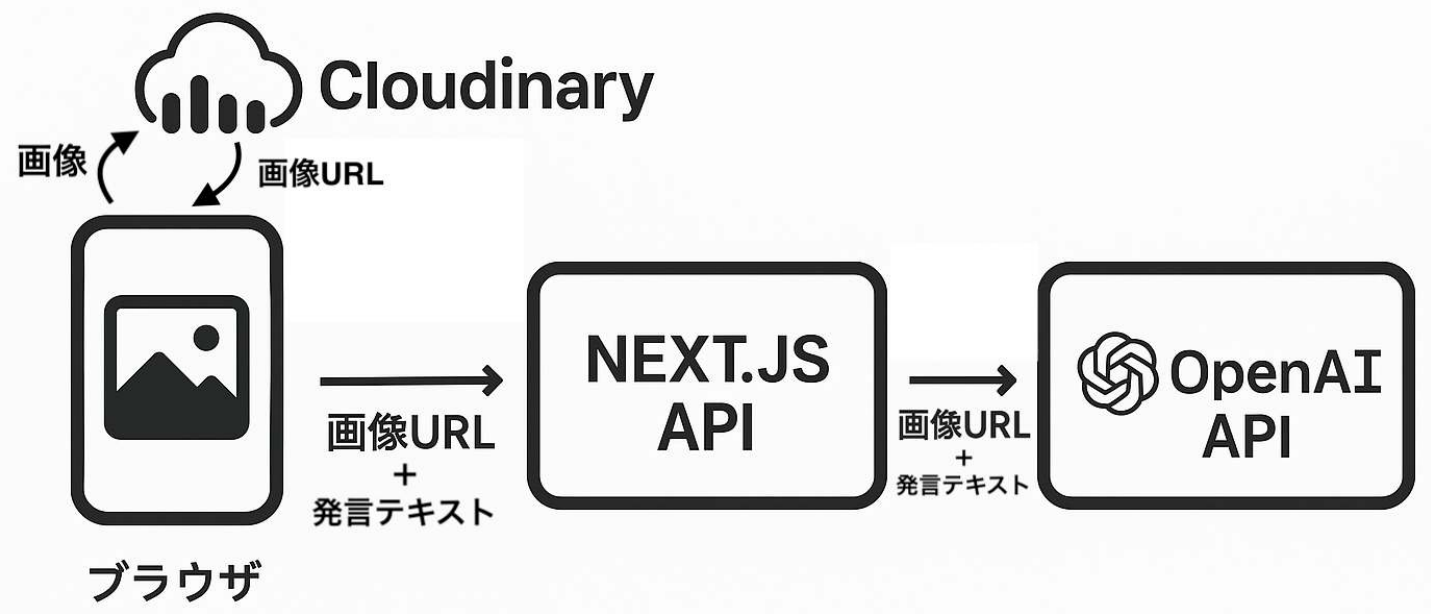
画像を見せるには、使うAIモデルがマルチモーダルなモデルである必要があります。

「モーダル」は情報の形式を指していて、マルチモーダルなら複数形式を扱えます。
つまりテキストだけじゃなくて画像や音声も入力して解釈できるということですね！

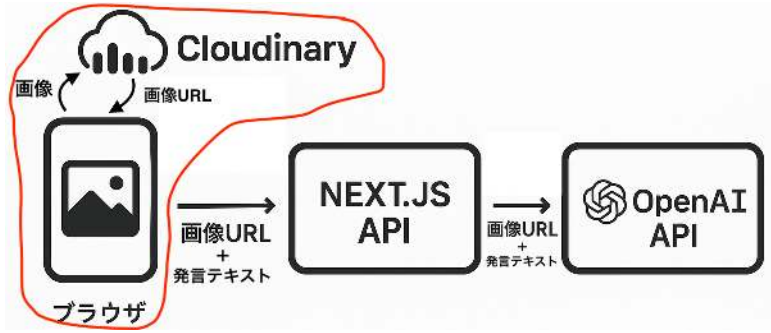
目をつくる — 実装

今回は画像のアップロード先に Cloudinary というサービスを使ってみます。

- 無料枠のストレージ・転送量が十分
- クライアントからSDKで直接アップロード可能



目をつくる — 実装（ファイルアップロード）



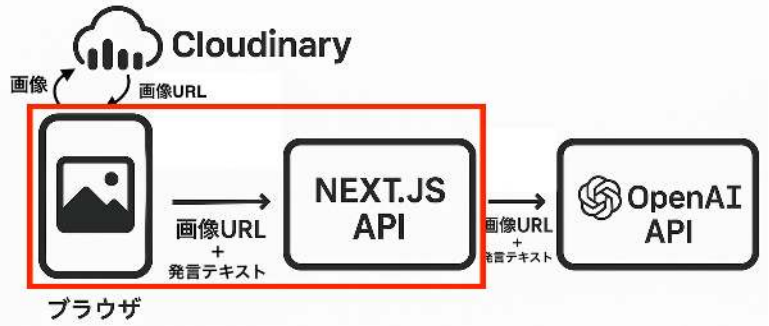
チャット送信欄に画像添付ボタンをつけて、添付された画像をアップロード

```
const CLOUD_NAME   = 'your-cloud-name';
const UPLOAD_PRESET = 'unsigned_preset'; // ※ 簡単にするため署名なしなので内容注意
const ENDPOINT = `https://api.cloudinary.com/v1_1/${CLOUD_NAME}/image/upload`;

const formData = new FormData();
formData.append('file', file);
formData.append('upload_preset', UPLOAD_PRESET);

const res  = await fetch(ENDPOINT, { method: 'POST', body: formData });
const json = await res.json();
console.log(json.secure_url) // 画像URL
```

目をつくる — 実装（画像URLをAPIに送信）



クライアントから送るVercel AI SDKのuseChatのリクエストに含めます

```
const urls = ['...', '...',] // 添付した画像URLの配列

// Vercel AI SDKのuseChatから返ってくるhandleSubmit
handleSubmit(e, {
  experimental_attachments: urls.map((url) => ({
    url,
    contentType: 'image/*',
  })),
});
```

目をつくる — 実装（画像URLをAIに送信）

自前APIの中で整形して `friendAgent.stream` の第一引数のユーザー発言に含める

```
[{
  role: 'user',
  content: [
    { type: 'text', text: userMessage.content }, // テキスト
    ...(userMessage.experimental_attachments || []).map(
      ({ url, contentType }) => ({
        type: 'image', image: url, mimeType: contentType,
      }),
    ),
  ],
}],
```

目をつくる — 実装

めっちゃ綺麗な夕焼け撮れた！

うわあ！富士山の影も美しいね！

目をつくる ― やったこと

- 画像データを送れるようにする
- AIが画像を見られるようにする
- 画像をアップロードして
  後からでも見られるようにする

画像を送ってコミュニケーションができるようになりました！

外に出す

外に出す — 何するの？

今公園に来てるよ！

・・・

（あー、家のPCでサーバー起動してるんだった…）

外に出す ― 何するの？

今公園に来てるよ！

こんなの友達じゃ…ないですよね？？

（あー、家のPCでサーバー起動してるんだった…）

外に出す — 何するの？

- どこからでもやりとりできるようにクラウドサーバーにデプロイする
- 知らない人にチャットされないように認証をかける
    - envにいれた自分のメールアドレス以外なら弾く、などのブロック

mem0とCloudinaryは既にクラウド上なので問題なし！

外に出す — 実装

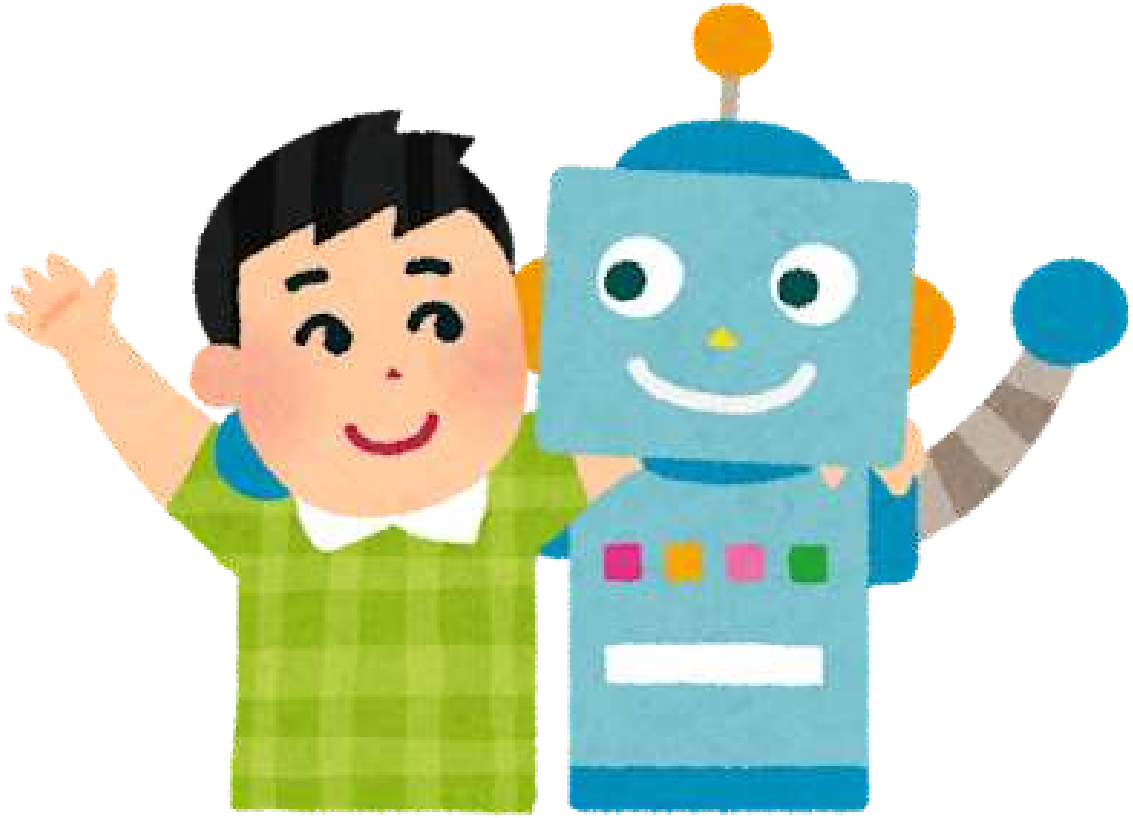
自分の場合は

- Next.js : Vercel
- 認証 : Auth0（Vercel連携で簡単）
- DB : Turso（LocalのLibSQLファイルを変換してアップロード）

で安定して動きました。

Next.js部分ははアップロード先によってはStreamingの時間制限が短くて
返信を受信しきれないことが頻発するパターンがあったので、
timeoutを長めにできるところを選びましょう。

もっと仲良くなる



もっと仲良くなる

- コンテキスト情報の拡充
  - ユーザー情報の挿入
  - 現在時刻やメッセージ時刻の挿入
- Toolsの拡充
  - 検索Toolを作って時事的な話ができるようにしたり？
  - スマートリモコンMCPで家の電気を操作してもらえたり？

君だけの親友AIをつくろう！！

宣伝（会社）

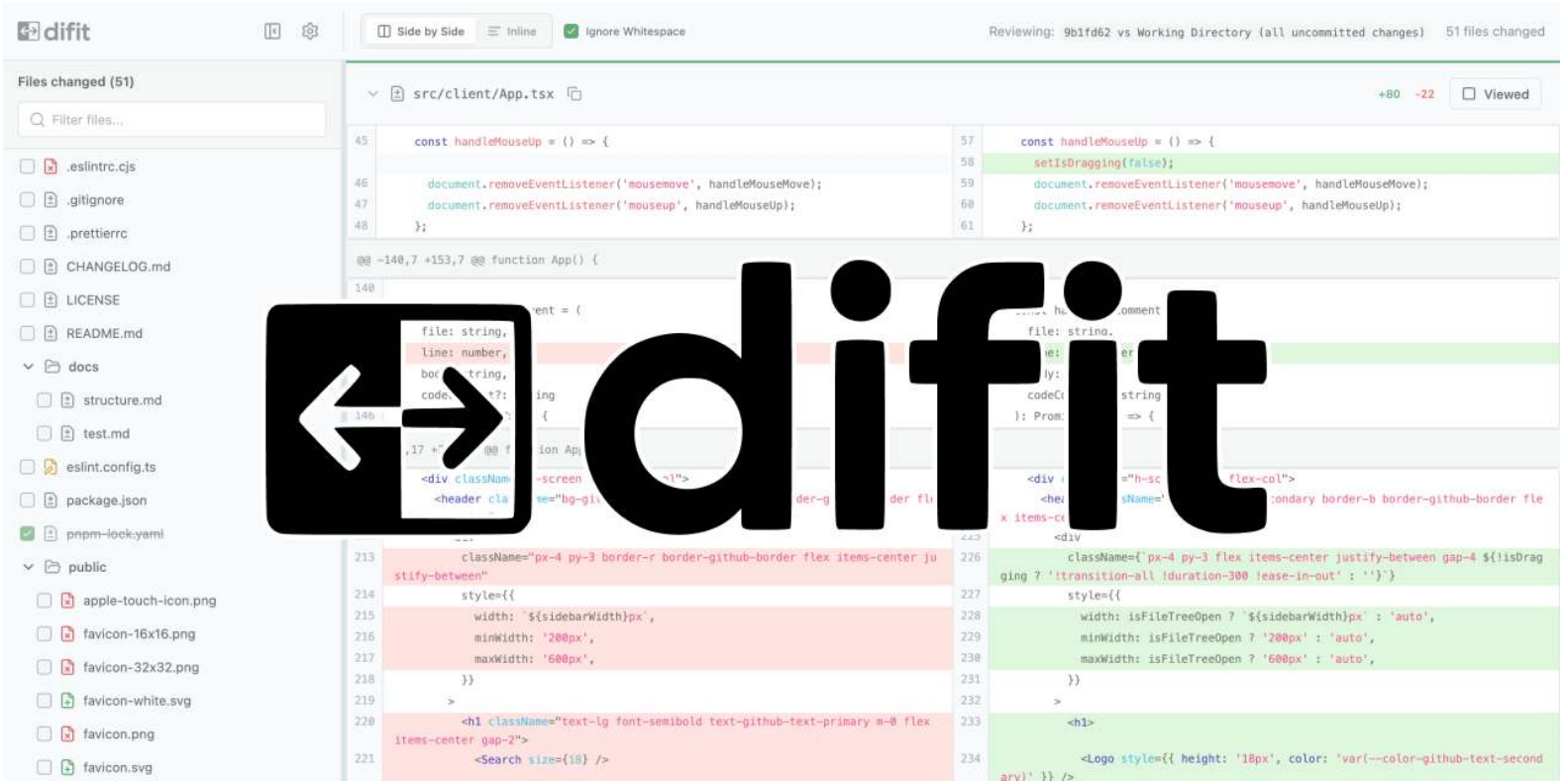
ナレッジワークはセールス**AIエージェント**の会社です！！！

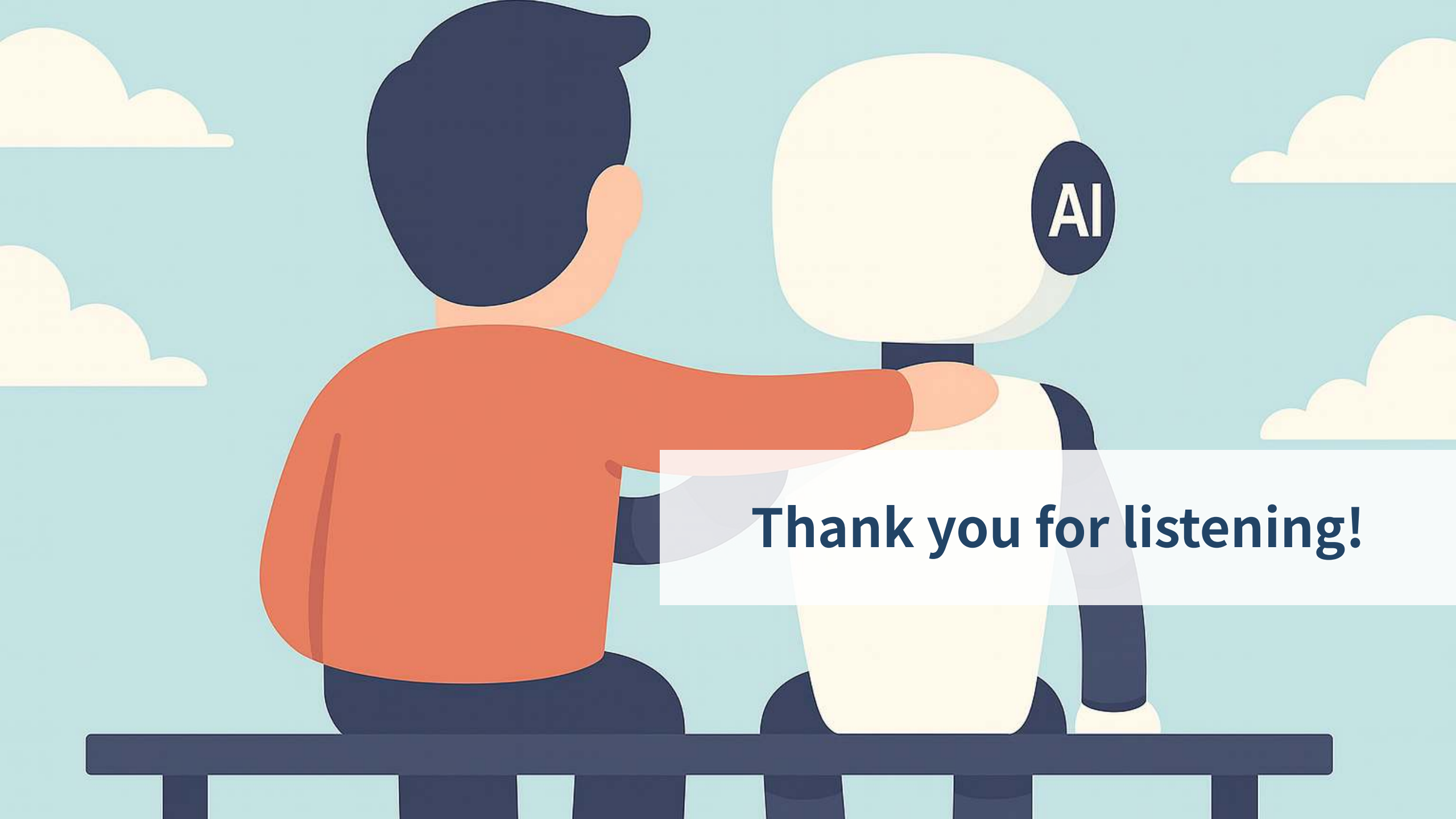
_KNOWLEDGE WORK

それだけ覚えていってください！！！

宣伝（個人）

ローカルでコード差分のレビューできるOSS 作ってます！ `npx difit` してみてね





Thank you for listening!