1/ App Flow : User auth & Chat

1. **User Account & Tokens**
    - Users can create a new account or sign in.
    - Upon authentication, the user receives an **access token** and **refresh token**.
2. **Chat Panel**
    - After login, users are redirected to the chat panel.
    - They can **start a new chat** or **browse previous conversations**.
3. **Language & Model Switching**
    - Users can switch the app language using a toggle.
        - Conversation titles and orientation update according to the selected language.
    - Users can switch between chatbot models within a conversation.
4. **Language Handling for Messages**
    - Every request includes the user's language as a query parameter.
    - When submitting a message:
        - The server detects the user's language (via Python library).
        - The message is sent to the bot.
        - The bot's response is translated to the **opposite language** using **DeepL API**.
5. **Database Storage**
    - For each message submission, the following are saved in the database:
        - Reply in Arabic
        - Reply in English
        - Reply in **detected language** (formatted in HTML)

        - Two conversation titles (tags) in both supported languages
        - Detected language of the user's message
    - Only the reply in the detected language is **HTML formatted**, since users may switch languages mid-conversation.
    - On the frontend, only the response in the **last detected language** is rendered.
    - Translated replies are used exclusively for **summary generation**.
6. **Conversation Titles**
    - The conversation title returned to the frontend matches the **language passed in query parameters**.

Summary Generation Rules

1. **Initial User Quota**
    - On signup, each user has:
        - `conversations_quota = 5`
        - `conversations_count = 0`
2. **Tracking Conversations**
    - Each new conversation increments `conversations_count`.
    - When `conversations_count` reaches `conversations_quota`, a summary is generated.
3. **Summary Details**
    - Summary is generated in **two languages** using **two models** (to be specified).
    - In the user profile, users can see:
        - General info
        - Last generated summary
4. **Summary Retrieval**

- When calling `getUserLastSummary` related api in profile section:
    - If `conversations_count` exceeds the quota and no new summary exists, a new summary is generated.
    - Otherwise, the last summary is returned.
    - If no summary exists, an appropriate message is returned.

2. Models and AI Services Used

Chat Models

The application uses three main models for chat interactions:

Gemini 2.5 Flash

- Source: AI Studio Official

DeepSeek Chat v3.1 (Free)

- Source: OpenRouter

OpenAI GPT-OSS 20B (Free)

- Source: OpenRouter

Local Models for Summary Generation and Conversation Tagging

For generating summaries of conversations and tagging them:

1. **Google FLAN-T5 Base**
    - Used for **English summary generation** and **conversation tag generation**.

- Showed very good results for both tasks.

2. **UBC-NLP AraT5 Base Title Generation**

    - Used mainly for **Arabic conversation tag generation**.
    - Model showed accepted results, but can be improved

3. **csebuetnlp mT5 Multilingual XLSum**

    - Used for **summary generation** in arabic.

    - **Limitations:** Weak performance for Arabic summaries; mostly paraphrases rather than generating meaningful summaries.the problem is related to the fact that the model shows better result on text or PDF summarising (like facebook/bart-large-cnn in english context), Got dumb in conversations.

    - Requires further research to identify a more adequate Arabic summarization model.

3. i18n (Internationalization) Implementation

Frontend

- Implemented via **Language Context**.

Translation files:

```
src/locales/ar.json

src/locales/en.json
```

-

Example in React:

```
<h1>{t("greeting")}</h1>

<p>{t("profile.title")}</p>

<button>{t("profile.edit")}</button>
```

Here, t is a function provided by the i18n library (for example, react-i18next) that:

Takes a key (like "greeting" or "profile.title").

Looks up the key in the current language JSON file (based on the language set in the context).

Returns the translated string corresponding to that key.

Automatically updates the UI if the language changes dynamically.

Backend

- i18n used in similar way as frontend , for error messages used this examples:

```
ERROR_MESSAGES = {
    "VALIDATION_ERROR": {
        "en": "Validation error occurred.",
        "ar": "حدث خطأ في التحقق من صحة البيانات."
    },
    "PROBLEM": {
        "en": "An unexpected problem occurred.",
        "ar": "حدث خطأ غير متوقع."
    },
    "CONVERSATIONS_RETURNED_SUCCESSFULLY": {
        "en": "Conversations retrieved successfully.",
        "ar": "تم استرجاع المحادثة بنجاح."
```

```
    },
    "AUTHENTICATION_ERROR": {
        "en": "Authentication failed. Please log in again.",
        "ar": "فشل التحقق من الهوية. الرجاء تسجيل الدخول مرة أخرى."
    },
    "NOT_FOUND": {
        "en": "The requested resource was not found.",
        "ar": "المورد المطلوب غير موجود."
    },
}
```

then returned on each api using the same t function.

(honestly, didn't find time to implement it in all apis , but if I did, a toast in react will be showing the translated error,

The language_code of error in backend will be detected from the query params..)

4. AI Tools used:

- Claude for frontend css optimization (Its UI capabilities are better then other bots)
- Mainly Chatgpt for technical decision makings (Suggest him my solution and ask for better…, backend aggregations detailed description for execution , than I assist and correct its output), Search of suitable models.. Also claude is used for that purpose.
- Copilot for code completions (but I have limited plan)