Design Doc

```
#TBD meaning I'm not sure yet
#Top-P for Top priority in implementation
```

Problem Statement

When I'm using linkedin, particularly since I'm in my job-searching phase, it can sometimes be inundated with a variety of messages that be be overwhelming to manage effectively. The challenges includes:

1. **Information Overload**: Lengthy, detailed messages such as job descriptions require significant time to read and understand.(sometimes it could be helpful if I can quickly grasp whether it requires citizenship/Security clearance/sponsorship)
2. **Poor Organization**: A high volume of incoming messages from diverse professional contacts makes it challenging to prioritize, categorize, and keep track of meaningful conversations.
3. **Time-Consuming 1-on-1 Communication**: Crafting individual, professional replies for various types of incoming messages can be a laborious process. It requires copy & pasting my profile information, while tailoring for the specific company.

These issues collectively results in a challenge of maintaining a high standard of professional networking

Constraints and Requirements

Language Model Integration: Must leverage a GPT-based Large Language Model for natural language understanding and responses. *For the prototype, this will be replaced by mock static responses*

User Interface: The user interface will be built using **React.js** and should follow a user-centric design approach. User-centric means the following

- Responsiveness: must adapt to various screen sizes
- Low Latency: quick message loading & smooth animation
- Context Preservation: a way to scroll back through the chat history
- Smart Input: maybe a '/command' feature for advanced users
- Accessibility: high-contrast mode, and possible keyboard shortcuts
- Feedback: a loading or typing indicator when loading respond from API

API/Backend: The backend will be built using **Express.js** and should be robust, low-latency & scalable.

- **Verification**: the backend should be able to verify the whether the request is coming from my front-end. (this is not needed if this app only runs locally, but is good to implement for possible future deployment? `#TBD`)
- **Load Chat history**: Query the database for historical messages
- **Redirection**: After verifying, the backend should be able to properly format and redirect the user-input to the AI model. Also when receiving a response from the AI model, the backend should redirect that to the front end.
- **Throttling & Rate Limiting**: protect against abuse
- **Error Handling**: including exception handling, retries
- **Concurrency**: use asynchronous programming for this? or just block the possibility of concurrency? `#TBD`
- **Advance use**: Corresponding to the Smart Input in the user interface, the backend should be able to recognize special commands and act correspondingly `#TBD`

Database: since we are assuming to already have a fine-tuned AI model. For now we need a database for storing chat history. Since I'm the only target user, no need for user ID and separate user table.

1. `ChatSessions` **table**: Stores metadata about each chat.
   - sessionID(PK), lastUpdated, etc.
2. ** `ChatMessages` : Stores information about individual Chat.
   - messageID(PK), sessionID(FK), messageText, timestamp, etc.
   For the first prototype I probably will just implement one conversation, but I guess its good to think about this Database relationship ahead.

Chosen solution

An AI chatbot that's trained on my personal data from linkedin, it can then analyze and organize my LinkedIn messages data. By engaging in conversational interactions, it can act as a one-stop solution to organize/analyze my incoming professional communications. Corresponding to the problem statement, it should:

- Provide summary on long text received
- Organize different messages, categorize and prioritize those messages
- write individual templates to help me reply in professional manner

Chatbot interface

1. **Prompt Input**: Text box for user to type in prompts or commands. #Top-p
2. **Message Display**: Area to display chat history #Top-p
3. **Loading Indicator**: Shows when the bot is fetching or processing data.

Back-End

1. **GET** `/history`: To fetch chat history from the local database
2. **POST** `/redirect`: To send user input to the AI model and return the response. #Top-p
3. **POST** `/command`: To execute special commands like `/summary`, `/categorize`
4. **POST** `/cancel`: Interrupt and cancel the request

Mock AI Model: A static list of pre-defined replies to respond to backend request

- Delay Simulation: add a small artificial delay mimicking respond from AI model
- Error Simulation: occasionally return errors or timeouts? #TBD

Test Plan

Functional Testing:
Objective: To verify the correct functionality of API endpoint.
Endpoint: Send User Prompt to Chatbot

- Method: POST
- URL: `/api/prompt`
- Test Cases:
  1. Send a valid prompt and verify a correct response.
  2. Send an invalid or empty prompt and expect proper error handling.

Performance Testing
Objective: To assess the system's responsiveness.

- **Endpoints:** All API endpoints
- **Test Cases:**
  1. Simulate multiple simultaneous requests and assess response time.

Security Testing
Objective: Ensure security

- **Test Cases:**
  1. Attempt unauthorized access to protected endpoints.

integration Testing
Objective: To test the interactions between the front-end, back-end, and the AI model.

- **Test Cases:**
    1. Verify correct interaction between front-end and back-end API.
    2. Verify correct data exchange between API and the AI model.

Alternative solutions

1. Using user's input ,associate with contextual prompts(tell AI model to form SQL queries, give it necessary Database Info) , ask the AI model to form corresponding SQL queries, send those queries to Back-End
2. Send those SQL queries to my local database, send output back to Backend
3. Backend send the output to AI Model to beautify the text, send to front-end

Reject reason: Not sure if AI model will form proper SQL queries all the time

a graph based solution - using graph data structures to model relationships between different entities #TBD Needs to explore more on this