TechBit WhatsApp Chatbot Requirement Specification

Aditi

May 15, 2025

Contents

1 Introduction **3**
1.1 Problem Statement . . . 3
1.2 Scope . . . 3

2 System Architecture **3**
2.1 Components . . . 3
2.2 Dependencies . . . 5
2.3 Environment Setup . . . 6

3 Functional Requirements **6**
3.1 Query Processing . . . 6
3.2 Predefined Questions . . . 7
3.3 Context Awareness . . . 7
3.4 Language Support . . . 7
3.5 Session Management . . . 7
3.6 Error Handling . . . 7

4 Non-Functional Requirements **8**
4.1 Performance . . . 8
4.2 Scalability . . . 8
4.3 Reliability . . . 8
4.4 Security . . . 8
4.5 Maintainability . . . 8

5 Implementation Details **8**
5.1 CLI Workflow (`app_graph.py`) . . . 8
5.2 API Workflow (`api_chat.py`) . . . 9
5.3 LangGraph Setup (`app_graph.py`) . . . 9

6 Deployment **10**
6.1 Development . . . 10
6.2 AWS EC2 Deployment . . . 10

7 Testing **11**
7.1 Unit Tests . . . 11

7.2 Integration Tests 11
7.3 Test Cases 11

8 Constraints and Assumptions **11**
8.1 Constraints 11
8.2 Assumptions 11

9 Future Enhancements **12**

10 References **12**

11 Notes on Completion **12**

1 Introduction

1.1 Problem Statement

This document outlines the requirements and design for the TechBit WhatsApp Chatbot, a multilingual support assistant that answers procedural and informational queries based on document manuals (`raw_files/docs/*.docx`) and WhatsApp chat history (`raw_files/chat_with_intent.csv`). The chatbot leverages LangGraph for workflow orchestration, Google Generative AI for natural language processing, and FAISS for efficient vector-based retrieval. It provides accurate, context-aware responses through a command-line interface (CLI) and a RESTful API, without relying on external knowledge sources, ensuring compliance with data privacy policies.

1.2 Scope

The TechBit WhatsApp Chatbot:

- Processes user queries in English, Hindi, or Hinglish, responding in the same language as the input.
- Delivers step-by-step instructions for procedural queries (e.g., "How to create new vendor?").
- Uses WhatsApp chat history to enhance response relevance with contextual understanding.
- Supports predefined questions and custom inputs via:
  - A CLI interface (`app_graph.py`) for interactive user sessions.
  - A RESTful API (`api_chat.py`) for integration with external systems, including a planned chat UI.
- Restricts responses to information in `.docx` manuals and `chat_with_intent.csv`.
- Automatically loads documents into a FAISS-based knowledge base at API startup.
- Deploys on an AWS EC2 instance for testing and user validation.

2 System Architecture

2.1 Components

The TechBit WhatsApp Chatbot consists of the following components:

- **Data Sources**:
  - **Document Manuals** (`raw_files/docs/*.docx`): `.docx` files containing procedural instructions (e.g., creating vendors, users, services). These serve as the primary knowledge source.

  - **Chat History** (`raw_files/chat_with_intent.csv`): A CSV file with WhatsApp chat data, including:
    * `Date`: Message date (e.g., 2023-01-01).
    * `Time`: Message time (e.g., 10:00).
    * `Sender`: Sender identifier.
    * `Message`: Message content.
    * `Keywords`: Extracted keywords.
    * `Intent`: Inferred intent (e.g., question, positive).
- **Knowledge Base** (`kb_docs.py`):
  - **Document Processing**: Uses `python-docx` to extract text from `.docx` files, splits text into chunks (1000 characters, 200-character overlap) using `RecursiveCharacterTextSplitter`, embeds chunks with `HuggingFaceEmb` (`sentence-transformers/all-MiniLM-L6-v2`), and stores them in a FAISS vector store (`hr_vector_store`).
  - **Chat History Parsing**: Parses `chat_with_intent.csv` using `pandas`, storing messages as dictionaries with `Date`, `Time`, `Sender`, `Message`, `Keywords`, `Intent`. Uses the last five messages for context.
  - **Automatic Loading**: Integrated into `api_chat.py`'s FastAPI lifespan event, loading documents and updating the vector store at startup.
- **LangGraph Workflow** (`app_graph.py`):
  - A state machine using LangGraph, defined by the `AgentState` typed dictionary:
    * `messages`: List of `HumanMessage` and `SystemMessage`.
    * `session_id`: Unique session identifier.
    * `docs`: List of text chunks.
    * `embeddings`: Embedding model.
    * `retriever`: FAISS-based retriever.
    * `llm`: Google Generative AI LLM (`gemini-2.0-flash`).
    * `qa_chain`: Conversational retrieval chain.
    * `whatsapp_messages`: Parsed chat history.
    * `error`: Error messages.
  - **Nodes**: `parse_whatsapp_chat`, `load_docs`, `split_text`, `embed_docs`, `setup_retriever`, `setup_llm`, `build_chain`, `qa_agent`.
  - **Edges**: `parse_whatsapp_chat` → `load_docs` → `split_text` → `embed_docs` → `setup_retriever` → `setup_llm` → `build_chain` → `qa_agent` (conditional) → END.

- **Conversational Retrieval Chain**:
  - Uses `ConversationalRetrievalChain` from LangChain, combining the FAISS retriever, LLM, and `ConversationBufferMemory` (last 5 messages).
  - Employs a `ChatPromptTemplate` with a system prompt enforcing:
    * Responses based only on `.docx` manuals and chat history.
    * Step-by-step instructions for procedural queries.
    * Language matching the user's query.
    * Context from the last five WhatsApp messages.
- **CLI Interface** (`app_graph.py`):
  - Interactive console interface allowing users to select predefined questions, enter custom queries, or exit with `quit` or `exit`.
  - Displays numbered predefined questions and processes input.
- **API Interface** (`api_chat.py`):
  - FastAPI-based RESTful API with endpoints:
    * `GET /predefined-questions`: Returns the list of predefined questions.
    * `POST /ask`: Accepts JSON with `question`, `predefined_index`, or `session_id`, returning the question, answer, and session ID.
  - Uses Pydantic models (`QuestionRequest`, `QuestionResponse`, `PredefinedQuest`
  - Automatically loads the knowledge base at startup using `kb_docs.py`.
- **Logging**:
  - Logs to `app.log` (CLI and LangGraph) and `api.log` (API).
  - Format: `[%Y-%m-%d %H:%M:%S] [%(levelname)s] - %(message)s`, with UTF-8 encoding.
  - Logs `INFO` for successful operations and `ERROR` for failures.

2.2 Dependencies

- **Python Libraries**:
  - `fastapi`: API framework.
  - `uvicorn`: ASGI server.
  - `langchain`, `langgraph`: Workflow and retrieval.
  - `langchain-google-genai`: Google Generative AI LLM/embeddings.
  - `langchain-community`: FAISS, document loaders.

  - `pandas`: CSV parsing.
  - `python-dotenv`: Environment variables.
  - `faiss-cpu`: Vector storage.
  - `python-docx`: `.docx` processing.
  - `sentence-transformers`: Embeddings.
- **External Services**:
  - Google Generative AI API (`gemini_api_key` in `.env`).

2.3 Environment Setup

- **File Structure**:

```
tech_chat/
        src/
            api_chat.py
            app_graph.py
            kb_docs.py
        raw_files/
            docs/
                doc1.docx
                ...
            chat_with_intent.csv
        hr_vector_store/
        .env
        .gitignore
        api.log
        app.log
```

- **Environment Variables**:
  - `gemini_api_key`: Google Generative AI API key, stored in `.env`.

3 Functional Requirements

3.1 Query Processing

- **Input**:
  - CLI: Number (predefined question) or text (custom query).
  - API: JSON with `question`, `predefined_index`, or `session_id`.
- **Output**:
  - Step-by-step instructions for procedural queries.
  - Concise answers for informational queries.
  - Responses in the same language as the input (English, Hindi, Hinglish).

- **Constraints**:
  - Answers derived solely from `.docx` files and `chat_with_intent.csv`.
  - Procedural queries are treated as relevant to document content.

3.2 Predefined Questions

- Fixed set of predefined questions (in `app_graph.py`):
  - What are the steps labeled as creating a new user?
  - How to create new vendor?
  - What are the steps to create new customer?
  - How do I create vendor?
  - How to create add service to the branch?
- Accessible via CLI (number) or API (`predefined_index`).

3.3 Context Awareness

- Uses the last five messages from `chat_with_intent.csv` for context.
- Context included in the prompt template:

```
WhatsApp Chat History (for context):
[{Date} {Time}] {Sender}: {Message}
```

3.4 Language Support

- Accepts queries in English, Hindi, or Hinglish.
- Detects input language and responds accordingly.

3.5 Session Management

- Maintains conversation history using a `sessions` dictionary (`app_graph.py`).
- API supports `session_id` in requests/responses for persistent sessions.

3.6 Error Handling

- **CLI**:
  - Invalid input (e.g., out-of-range number): Displays error and prompts again.
  - File not found (e.g., `raw_files/docs/`): Logs error and raises `FileNotFoundError`
- **API**:

  - Invalid `predefined_index`: HTTP 400.
  - Empty/invalid question: HTTP 400.
  - Processing errors: HTTP 500.
- Logs errors to `app.log` or `api.log`.

4 Non-Functional Requirements

4.1 Performance

- **Response Time**: <5 seconds (assuming stable Google API connectivity).
- **Startup Time**: Document loading and vector store creation (10–30 seconds, depending on `.docx` file count).

4.2 Scalability

- CLI: Single-user interactions.
- API: Handles concurrent requests via Uvicorn workers.

4.3 Reliability

- Deterministic FAISS retriever and fixed prompt ensure consistent responses.
- File validation checks `raw_files/` before processing.

4.4 Security

- `gemini_api_key` stored securely in `.env`.
- CORS: Allow all origins (development); restrict in production.
- Data privacy: Responses limited to `.docx` and `chat_with_intent.csv`.

4.5 Maintainability

- Modular design: `api_chat.py`, `app_graph.py`, `kb_docs.py`.
- Logging aids debugging.
- Type hints and clear naming improve readability.

5 Implementation Details

5.1 CLI Workflow (`app_graph.py`)

- **Entry Point**: Main function displaying predefined questions.

- **Process**:
  1. Shows numbered predefined questions.
  2. Accepts input (number or text).
  3. Validates input and constructs `AgentState`.
  4. Invokes `graph.ainvoke` to process the query.
  5. Displays answer from `SystemMessage`.
  6. Repeats until `quit` or `exit`.
- **Error Handling**:
  - Invalid input: Prompts again.
  - File/processing errors: Logs and displays.

5.2 API Workflow (`api_chat.py`)

- **Endpoints**:
  - `GET /predefined-questions`: Returns `PREDEFINED_QUESTIONS`.
  - `POST /ask`: Processes query with `question`, `predefined_index`, or `session_id`.
- **Process**:
  1. Validates request via `QuestionRequest`.
  2. Extracts question (custom or predefined).
  3. Constructs `AgentState` with `session_id`.
  4. Invokes `graph.ainvoke`.
  5. Returns `QuestionResponse` with `session_id`.
- **Startup**:
  - Lifespan event loads `.docx` files and `chat_with_intent.csv` using `kb_docs.py`, updating `hr_vector_store`.
- **Error Handling**:
  - Invalid requests: HTTP 400.
  - Server errors: HTTP 500.

5.3 LangGraph Setup (`app_graph.py`)

- **Initialization**: `compile_graph()` at module level.
- **State Management**: `AgentState` tracks data across nodes.
- **Conditional Flow**: `qa_agent` invoked based on `messages` presence.

6 Deployment

6.1 Development

- **Run CLI**:

```
python src/app_graph.py
```

- **Run API**:

```
uvicorn src.api_chat:app --host 0.0.0.0 --port 8001
```

- **Dependencies**:

```
pip install fastapi uvicorn python-docx pandas langchain
   langgraph langchain-google-genai langchain-community python-
   dotenv faiss-cpu sentence-transformers
```

6.2 AWS EC2 Deployment

- **Setup**:
  - Launch EC2 instance (e.g., `t3.medium` or `g4dn.xlarge` for Phi-3 integration).
  - Clone `debug_aditi`:

```
git clone -b debug_aditi https://username@bitbucket.org/
   username/tech_chat_repo.git
```

  - Install dependencies and run:

```
cd tech_chat
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
uvicorn src.api_chat:app --host 0.0.0.0 --port 8001
```

- **Chat UI**:
  - Integrate with a frontend (e.g., React) connecting to `http://<ec2-ip>:8001/ask`.
- **Production**:
  - Use Gunicorn:

```
gunicorn -w 4 -k uvicorn.workers.UvicornWorker src.api_chat:
   app --bind 0.0.0.0:8001
```

  - Configure Nginx with SSL.
  - Monitor with Prometheus/Grafana.

7 Testing

7.1 Unit Tests

- Test `.docx` loading and chunking.
- Test CSV parsing for required columns.
- Test FAISS retriever with sample queries.
- Test LLM language consistency.

7.2 Integration Tests

- Test CLI with predefined and custom queries.
- Test API endpoints with valid/invalid inputs.

7.3 Test Cases

- **CLI**:
  - Input: Predefined question number (e.g., 1).
  - Expected: Answer from `.docx`.
  - Input: Invalid number (e.g., 10).
  - Expected: Error message and prompt.
- **API**:
  - Request: `POST /ask` with `{"question": "How to create new vendor?", "session`$_id$" : "$test$"}. $Expected : 200 OK with answer.$
  - Request: `POST /ask` with {"predefined$_index$" : 10}. $Expected : 400 Bad Request.$

8 Constraints and Assumptions

8.1 Constraints

- Requires valid `gemini_api_key`.
- `raw_files/docs/*.docx` and `chat_with_intent.csv` must exist.
- Internet connectivity for Google API.

8.2 Assumptions

- `.docx` files contain parseable procedural instructions.
- `chat_with_intent.csv` has valid columns.
- Queries are relevant to document content.

9 Future Enhancements

- **Multimodal Support**: Process images in `.docx` files.
- **Database**: Store sessions in a database.
- **Authentication**: Add JWT for API security.
- **Caching**: Cache FAISS index for faster startup.
- **WhatsApp API**: Enable live WhatsApp integration.
- **Advanced Models**: Integrate Phi-3 or GloVe for enhanced processing.

10 References

- LangChain: `https://python.langchain.com/docs/`
- LangGraph: `https://langchain-ai.github.io/langgraph/`
- FastAPI: `https://fastapi.tiangolo.com/`
- Google Generative AI: `https://cloud.google.com/vertex-ai/docs/generative-a`
- FAISS: `https://github.com/facebookresearch/faiss`
- Sentence Transformers: `https://sbert.net/`

11 Notes on Completion

- **Alignment with Project**: Uses `api_chat.py` (FastAPI, port 8001), `app_graph.py` (CLI and LangGraph), and `kb_docs.py` (knowledge base). Excludes `chatbot_core.py` as it does not exist.
- **FileNotFoundError**: Addressed by specifying file validation (Section 3.6). Ensure `raw_files/docs/` and `raw_files/chat_with_intent.csv` exist:

```
dir "C:\Users\Aditi-51000042\Documents\whatsapp chatbot\
   tech_chat\raw_files"
```

  Update paths in `kb_docs.py` if needed:

```
docs_dir = r"C:\Users\Aditi-51000042\Documents\whatsapp chatbot\
   tech_chat\raw_files\docs"
whatsapp_chat_path = r"C:\Users\Aditi-51000042\Documents\
   whatsapp chatbot\tech_chat\raw_files\chat_with_intent.csv"
```

- **Outdated Files**: Remove from `debug_aditi`:

```
git rm src/old_script.py
git commit -m "Remove outdated script"
git push -u origin debug_aditi
```

- **AWS and UI**: Includes EC2 deployment and chat UI integration requirements. Use GPU instance (`g4dn.xlarge`) for Phi-3 if integrated.