Regulatory Bulletin Assistant: Technical Development Documentation

Table of Contents

1. Introduction

The Regulatory Bulletin Assistant is a locally developed Python-based web application designed to assist regulatory engineers in creating internal regulatory bulletins. This document provides technical details for developers working on maintaining or extending the application.

2. System Architecture

The application follows a web-based architecture:

- **Frontend**: Streamlit web interface
- **Backend**: Python scripts handling data processing and AI interactions
- **External Services**: Challenger GenAI API for language model interactions

3. Development Environment Setup

3.1 Prerequisites

- Python 3.8+

3.2 Installation

1. Ensure you have access to the local package directory.

2. Create a virtual environment:

```
python -m venv venv
source venv/bin/activate  # On Windows, use `venv\Scripts\activate`
```

3. Install dependencies from the provided `requirements.txt`:

   ```
   pip install -r requirements.txt
   ```

3.3 Configuration

1. Make sure there is a `.env` file in the project root:

   ```
   CHALLENGER_GENAI_API_KEY=your_api_key_here
   ```

2. Ensure `guided_questions.json` is present in the project root.

4. Key Components

4.1 Main Application (`app_st.py`)

- Streamlit-based web application
- Functions for handling file uploads, guided questions, and chat interactions

4.2 Keyword Extraction (`keyword_extraction.csv`)

- CSV file containing extracted keywords for matching

4.3 Requirements (`requirements.txt`)

Key dependencies include:

- `streamlit`: Web application framework
- `openai`: OpenAI API client
- `sentence-transformers`: Text embeddings
- `scikit-learn`: Machine learning utilities
- `PyPDF2`: PDF processing
- `docx2txt`: Word document processing
- `python-dotenv`: Environment variable management

5. Data Processing

5.1 Document Handling

The application supports multiple file formats:

- `.txt`: Plain text
- `.docx`: Word documents
- `.pdf`: PDF files
- `.csv`: CSV files

Document content is extracted using appropriate libraries (`docx2txt` for Word, `PyPDF2` for PDF).

5.2 Keyword Extraction

1. The application uses `keyword_extraction.csv` for keyword matching.
2. Fuzzy matching is implemented to find relevant keywords for new documents.

6. AI Integration

6.1 Language Model

- The application uses the Challenger GenAI API via the OpenAI client.
- Default model: `"llama-3-8b-instruct"`

6.2 Prompt Engineering

- Guided prompt creation based on user responses to predefined questions.
- Dynamic prompt construction combining document content, user input, and extracted keywords.

6.3 Response Processing

- Responses are processed in chunks to handle long documents.
- Parallel processing is used for efficiency.

7. User Interface

7.1 Main Interface

- Streamlit-based web interface
- Chat interface with file upload and guided question functionality

7.2 Admin Interface

- Question management
- Password changes

8. Testing

8.1 Manual Testing

- Test all UI components
- Verify file attachments for different formats
- Check AI responses for various inputs
- Ensure admin functionalities work as expected

8.2 Automated Testing (Future Implementation)

- Implement unit tests for core functions
- Create integration tests for AI interactions
- Develop UI tests using Streamlit's testing utilities

9. Deployment

9.1 Running the Application

- Use the Streamlit CLI to run the application:

  ```
  streamlit run app_st.py
  ```

9.2 Distribution

- Provide installation instructions for end-users
- Include necessary files: `guided_questions.json`, `keyword_extraction.csv`

10. Maintenance and Updates

10.1 Updating AI Models

- To use a different AI model, modify the `MODEL` constant in `app_st.py`:

  ```
  MODEL = "new-model-name"
  ```

10.2 Updating Guided Questions

- Modify `guided_questions.json` to add, remove, or update questions.
- Use the admin interface for easier management.

10.3 Keyword Database Updates

- Update `keyword_extraction.csv` with new keywords as needed.

11. Known Issues and Future Improvements

11.1 Known Issues

- Large documents may cause memory issues
- API rate limiting can affect response times

11.2 Planned Improvements

- Implement caching for API responses
- Add support for more file formats (e.g., .rtf, .odt)
- Enhance error handling and user feedback
- Implement automated testing suite
- Add multi-language support

12. Contributing

For internal developers working on this local package:

1. Obtain the latest version of the code from the designated shared directory or internal version control system.
2. Create a new branch or working copy for your changes.
3. Implement and test your changes locally.
4. Document your changes thoroughly, including updates to this technical documentation if necessary.
5. Submit your changes for review according to the internal development process.