Code Explanation

Import Statements

```python
import streamlit as st
import os
import time
from groq import Groq
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import Chroma
from langchain.text_splitter import TokenTextSplitter
from langchain.document_loaders import PyPDFLoader
from langchain.prompts import PromptTemplate
from langchain.memory import ConversationBufferMemory
```

These imports bring in necessary libraries: - `streamlit` for the web interface - `os` for environment variable handling - `time` for adding delays in response display - `groq` for interacting with the Groq API - Various `langchain` components for NLP tasks

API Key Setup

```python
os.environ["GROQ_API_KEY"] = "gsk_uod8Lw1go81ZeGWAnZz1WGdyb3FYrECUBnzBSwT7tA7
iDZzkvrzb"
```

This sets the Groq API key as an environment variable. In production, it's better to use a more secure method like loading from a .env file.

Session State Initialization

```python
def initialize_session_state():
    # ... (content of the function)
```

This function initializes various components in the Streamlit session state: - `template`: A prompt template for the chatbot - `prompt`: A PromptTemplate object using the template - `memory`: A ConversationBufferMemory for maintaining chat history - `vectorstore`: A Chroma vector store for document embeddings - `chat_history`: A list to store the conversation

Groq API Interaction

```python
def get_groq_response(client, user_query, context):
    # ... (content of the function)
```

This function sends a request to the Groq API: - It constructs a system prompt with the given context - It sends the user query along with the system prompt - It uses the "mixtral-8x7b-32768" model - It returns the generated response

Main Function

```python
def main():
    # ... (content of the function)
```

This is the main function that runs the Streamlit app:

1. It initializes the Groq client and session state.
2. It sets up a file uploader for PDF documents.
3. When a file is uploaded:
   - It saves the file if it doesn't exist.
   - It processes the PDF, splits it into chunks, and creates a vector store.
4. It displays the chat history.
5. It handles user input:
   - Retrieves relevant context from the vector store.
   - Gets a response from the Groq API.
   - Displays the response with a typing effect.
   - Updates the chat history.

Key Components Explained

1. **PDF Processing**:

   ```
   loader = PyPDFLoader(fp)
   data = loader.load()
   ```

   This loads the PDF and extracts its content.

2. **Text Splitting**:

   ```
   text_splitter = TokenTextSplitter(chunk_size=1500, chunk_overlap=200)
   all_splits = text_splitter.split_documents(data)
   ```

   This splits the document into manageable chunks for processing.

3. **Vector Store Creation**:

   ```
   st.session_state.vectorstore = Chroma.from_documents(
       documents=all_splits,
       embedding=HuggingFaceEmbeddings(model_name="sentence-transformers/a
   ll-MiniLM-L6-v2")
   )
   ```

   This creates a Chroma vector store from the document chunks, using HuggingFace embeddings.

4. **Context Retrieval**:

   ```
   context_documents = st.session_state.retriever.get_relevant_documents(u
   ser_input)
   context = " ".join([doc.page_content for doc in context_documents])
   ```

   This retrieves relevant document chunks based on the user's input.

5. **Response Generation and Display**:

```
response = get_groq_response(client, user_input, context)
# ... (code for displaying response with typing effect)
```

This gets a response from the Groq API and displays it with a typing effect.

Error Handling

The code includes try-except blocks to handle potential errors during document processing and API interactions, displaying error messages to the user when issues occur.

Streamlit Interface

The code uses Streamlit's chat interface components (`st.chat_input()`, `st.chat_message()`) to create an interactive chat experience.