AI Booking Assistant – Problem Statement

1. Objective

Design and implement an AI-driven Booking Assistant that:

- Runs as a chat-based application.
- Supports RAG using user-uploaded PDFs.
- Detects booking-related intents and collects required details.
- Confirms details before storing in a database.
- Sends email confirmations after booking.
- Includes a mandatory Admin Dashboard to view stored bookings.
- Is deployed on Streamlit Cloud with a public accessible URL.
- Booking domain is open to candidate creativity (doctor/salon/hotel/events/classes/etc.).

2. Core Requirements

2.1 RAG Chatbot

- User uploads one or more PDFs via UI.
- System must:
  - Extract text, chunk, embed, store in lightweight vector store.
  - Answer questions using RAG blending (retrieved chunks + LLM output).

2.2 Conversational Booking

Chatbot must:

- Detect intent (general query vs booking).
- Collect via multi-turn dialogue:
  - Customer name
  - Email
  - Phone
  - Booking/service type
  - Preferred date & time
- Maintain short-term memory (last 20-25 messages).
- Only after all fields:

- Summarize details
- Ask explicit confirmation
- Store data only after approval

2.3 Data Storage (SQLite / Supabase)

Must save data in:

- SQLite, OR
- Supabase (recommended for persistence)

Minimum schema:

customers

- customer_id (PK), name, email, phone

bookings

- id (PK), customer_id (FK), booking_type, date, time, status, created_at

Short-term memory may use DB or st.session_state.

2.4 Email Confirmation

After successful booking:

- Send confirmation email using:
  - SMTP (dummy Gmail/app password), OR
  - Free email API (SendGrid, etc.)

Email must include:

- Name
- Booking ID
- Date & time
- Booking type
- Other information

Must handle failures gracefully.

2.5 Tool Calling (Minimum Tools)

Must implement:

1. RAG Tool
   - Input: query → Output: retrieved answer

2. Booking Persistence Tool
    - Input: structured booking payload → Output: success + booking ID
3. Email Tool
    - Input: to_email/subject/body → Output: success/failure
4. (Optional) Web Search Tool
    - Uses third-party JSON API

Tool routing may use agent framework (e.g., LangChain, CrewAI)

2.6 Frontend & Backend

- Streamlit (recommended for simplicity and rapid development) or any alternative frontend framework of your choice, provided the deployed application is publicly accessible.
- Backend:
    - Entire logic inside Streamlit, OR
    - Streamlit as frontend calling an external FastAPI / Flask API hosted elsewhere (optional, but allowed).

Frontend must include:

- Chat interface (st.chat_message, st.chat_input)
- PDF upload
- Status messages (DB saved / email sent / errors)
- Clear user vs bot messages
- Mandatory Admin Dashboard to:
    - View all bookings
    - Filter/search by name/email/date(optional)

2.7 Short-Term Memory

- Maintain conversation context.
- Minimum: last 20–25 messages
- Used in:
    - RAG prompts
    - Booking flow continuity

2.8 Error Handling

1. Must validate and handle:

- Wrong/missing fields (email/date/time)

- Invalid/missing PDFs
- DB insert/connection errors
- Email delivery failures
- Unexpected runtime errors

2. Provide friendly messages like:

- "Email could not be sent, but booking was saved."
- "Please enter date as YYYY-MM-DD."

3. Additional Requirements (Bonus)

(Not mandatory but improves score)

- STT and/or TTS
- Booking retrieval by user
- Admin enhancements (edit/cancel/export)
- Improved UX (avatars, thinking states)

4. Suggested Code Structure

```
project_root/
app/
| ├─ main.py            # Streamlit entry
| ├─ chat_logic.py      # Intent + memory
| ├─ booking_flow.py    # Slot filling + confirmation
| ├─ rag_pipeline.py    # PDF ingest + embeddings
| ├─ tools.py           # RAG/DB/email/search tools
| ├─ admin_dashboard.py # MANDATORY admin UI
| └─ config.py
|
db/
| ├─ database.py        # SQLite/Supabase client
| └─ models.py
|
docs/ sample PDFs, diagrams
├─ requirements.txt
├─ README.md
└─ .streamlit/secrets.toml
```

5. Booking Flow (Summary)

1. Detect booking intent
2. Extract known details
3. Ask only missing fields
4. Use memory to avoid repeats
5. Summarize details
6. Ask confirmation
7. On Confirmation:
    - Save to DB
    - Send email
8. Respond with booking ID
9. Store conversation history

6. Deployment – Streamlit Cloud

6.1 Test

- PDF upload & RAG responses
- Booking flow + confirmation
- DB storage
- Email delivery
- Admin dashboard
- Input validation
- Error handling

Note: SQLite may reset on restart; acceptable for assignment.

7. Submission Requirements

All 3 mandatory and individually scored:

1. **PPT Presentation**
    - Use case
    - Solution overview & approach
    - Architecture diagram
    - Booking flow

- RAG design
- Admin dashboard
- Screenshots
- Challenges + future improvements

2. **GitHub Repository**
    - Clean readable code
    - Proper structure
    - README with instructions
    - Comments where needed
3. **Deployed Streamlit Cloud Link**
    - Public, working end-to-end demo

Please find attached the sample project structure for your reference.