High-Level System Design

1. Architecture

Architecture Style:

- **Microservices-based architecture**
- This ensures modularity, scalability, and maintainability, allowing different functionalities (chatbot, knowledge ingestion, CRM logging, etc.) to be independently managed and scaled.

Deployment Model:

- **AWS Cloud-Based Deployment**
- Utilizing AWS services for hosting, storage, and AI processing while ensuring cost efficiency.

High-Level Components:

- **Frontend:** Chatbot UI (Website) & Admin Panel
- **Backend Services:** Chatbot Processing, Email Integration, Logging & Analytics
- **Database & Storage:** Knowledge Base (Transcripts & PPTs), Logging Database
- **Integrations:** CRM, Microsoft Outlook, Microsoft Teams (for transcript ingestion)

---

2. Core Modules & Responsibilities

Module	Responsibilities
Chatbot Service	- Handles user queries from the website and email. - Generates responses using the knowledge base (transcripts & slides). - Supports multi-turn conversation.
Knowledge Ingestion Service	- Periodically extracts transcripts from Microsoft Teams. - Parses PowerPoint slides for keyword indexing. - Processes and stores structured knowledge data in the database.
Response Generation & Context Management	- Parses user queries and applies context tracking for multi-turn chats. - Queries knowledge repository to fetch relevant information. - Ensures response accuracy by citing the source (transcript/slide).
Predefined Responses Management	- Admin users manage predefined chatbot responses for common queries.
Email Query Handling Service	- Fetches customer emails from Microsoft Exchange/Outlook. - Automatically generates replies using existing knowledge base data.
User Query Logging & Analytics	- Logs chatbot queries and responses for improvement. - Generates periodic analytics reports for admin insight. - Enforces a 3-month data retention policy.
Content Moderation Service	- Scans and filters inappropriate or spam queries from users.
CRM Integration Module	- Logs anonymized customer queries into the CRM system. - Ensures compliance with data protection regulations.
Admin Module	- Admins can access chatbot logs, manage predefined responses, and generate reports.

---

3. Technology Stack Recommendation

Frontend

- **React** (User-friendly interface for chatbot & admin portal)

Backend

- **Python (FastAPI)** (for chatbot logic, API integrations, and NLP processing)

AI Model

- **Phi-2 (Small LLM model)** (used for generating responses based on structured knowledge base)

Database & Storage

- **PostgreSQL (AWS RDS)** (structured knowledge storage)
- **Elasticsearch** (for indexing and fast retrieval of text-based knowledge)
- **Amazon S3** (storage of transcripts and PowerPoint slides)

Integration Layers

- **Microsoft Azure API** (for Teams transcript extraction & Outlook email handling)
- **RESTful APIs & Webhooks** (for CRM integration)

Infrastructure & Hosting

- **AWS Lambda & ECS (Fargate)** (for scalable service execution)
- **Amazon API Gateway** (secure API exposure)
- **AWS IAM & Cognito** (admin authentication)

---

4. System Workflows

1. Knowledge Ingestion & Indexing Workflow

1. System fetches **new meeting transcripts** from Microsoft Teams monthly.
2. System extracts **text and key topics** from PowerPoint slides.
3. **Elasticsearch indexes data**, making it searchable for chatbot queries.
4. Knowledge base updates, enabling chatbot to fetch the latest information.

---

2. Website Chatbot Query Handling Workflow

1. Website visitor interacts with the chatbot (React frontend).
2. User query is **sent to the chatbot backend API (FastAPI)**.
3. The chatbot component:
    - Checks indexed transcripts/slides for relevant answers.
    - Uses Phi-2 to construct a response.
    - Provides source citation for transparency.
4. The frontend displays the response to the user.
5. The query & response are logged for analytics.

---

3. Email Query Handling Workflow

1. A customer emails a question to the company's support email.
2. The Email Handling Service fetches the email via **Microsoft Exchange API**.
3. The AI chatbot parses the query and attempts to construct a response.
4. If a relevant answer exists:
    - The AI generates a reply and responds via Outlook.
5. If no relevant answer exists:
    - The system forwards the email to human support.

---

4. Admin Predefined Response Management Workflow

1. Admin logs into the web-based management panel (React frontend, AWS Cognito authentication).
2. Admin navigates to the "Predefined Responses" section.
3. They edit, add, or delete common question responses.
4. The updated responses are **stored in PostgreSQL** for chatbot retrieval.

---

5. Logging & Analytics Workflow

1. Every chatbot query/response is logged in **PostgreSQL**.
2. The Log Management Module enforces **a 3-month retention policy**.
3. **Automated reports** summarize chatbot activity using BI tools like Amazon QuickSight.
4. Admins access reports for performance monitoring and quality improvement.

5. Scalability & Security Considerations

Scalability

 Microservices Architecture: Each module (chatbot, ingestion, email handler) can scale independently.
 AWS Lambda & Fargate: Serverless functions reduce costs while ensuring on-demand scalability.
 Elasticsearch for Fast Search: Optimized for retrieving information quickly from large transcript datasets.
 Auto-scaling Database (RDS PostgreSQL): Ensures performance is maintained under high loads.

Component	Scaling Strategy
Chatbot Service	Auto-scale using AWS Fargate
Knowledge Base	Scale PostgreSQL & Elasticsearch clusters
API Gateway	Use AWS API Gateway with rate limiting
File Storage	Use S3 with lifecycle policies

Security Considerations

 No User Authentication for Public Use: Ensures ease of access but requires moderation.
 Content Moderation & Filtering: NLP model checks and blocks inappropriate queries.
 Email Security:

- Rate-limiting implemented to prevent email spam.
- Responses are generated from known data sources only.
  Data Protection in CRM Logs:
- Only anonymized data is saved to avoid PII storage.
  API Security:
- JWT-based authentication for admin actions.
- AWS WAF (Web Application Firewall) enabled for API protection.
- Rate limiting to prevent abuses.

---

6. Conclusion

This **microservices-based chatbot system** ensures efficient public interaction by generating responses from meeting transcripts and slides. The architecture focuses on **scalability, low operational costs, and ease of maintenance** through **AWS-based deployment** and **serverless computing (Lambda & Fargate)**.

- **Frontend (React, Modern UI)**
- **Backend (FastAPI, Python-based Chatbot & Integration Services)**
- **AI Processing using Phi-2 for cost-efficient NLP**
- **AWS infrastructure for scalability & reliability**
- **Security-first approach with content filtering and privacy compliance**

This system **automates responses, integrates with emails, and provides admins insights through analytics**, improving efficiency while maintaining low costs. **# Relational Database Schema Design for AI Chatbot System**

1. List of Tables & Column Definitions

Below is the detailed relational database schema design, considering all functionalities outlined in the requirements.

1.1. Users Table

Stores administrator user data (public users do not require authentication).

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(100) UNIQUE NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    role ENUM('admin') NOT NULL DEFAULT 'admin', -- Only admin roles are stored
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

Constraints:

- Users must have unique usernames and emails.
- Passwords are stored as hashed text.

1.2. Predefined Responses Table

Stores predefined chatbot responses for common queries.

```
CREATE TABLE predefined_responses (
    response_id SERIAL PRIMARY KEY,
    question TEXT NOT NULL UNIQUE,
    answer TEXT NOT NULL,
    created_by INT REFERENCES users(user_id) ON DELETE SET NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

Constraints:

- Questions must be unique.
- Supports admin-based management.

1.3. Meeting Transcripts Table

Stores ingested Microsoft Teams transcripts used by the chatbot.

```
CREATE TABLE meeting_transcripts (
    transcript_id SERIAL PRIMARY KEY,
    file_name VARCHAR(255) UNIQUE NOT NULL,
    content TEXT NOT NULL, -- Extracted and indexed textual content
    uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    processed BOOLEAN DEFAULT FALSE
);
```

Constraints:

- Ensures uniqueness of files.
- Boolean flag to track processing completion.

1.4. PowerPoint Slides Table

Stores ingested slides for reference.

```
CREATE TABLE ppt_slides (
    slide_id SERIAL PRIMARY KEY,
    file_name VARCHAR(255) UNIQUE NOT NULL,
    content TEXT NOT NULL, -- Extracted slide text
    uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    processed BOOLEAN DEFAULT FALSE
);
```

Constraints:

- Ensures uniqueness of slide files.

1.5. Chatbot Queries Table

Logs user queries and corresponding chatbot responses.

```
CREATE TABLE chatbot_queries (
    query_id SERIAL PRIMARY KEY,
    user_ip VARCHAR(50) NOT NULL, -- Stores public user’s IP address
    query_text TEXT NOT NULL,
    response_text TEXT NOT NULL,
    source ENUM('meeting_transcripts', 'ppt_slides', 'predefined_responses', 'none') DEFAULT 'none',
    source_id INT, -- Generic reference to source data
    logged_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Constraints:

- Queries and responses are stored anonymously.
- `source_id` dynamically refers to transcripts, presentations, or predefined responses.

1.6. Suggested Questions Table

Stores suggested questions displayed to users.

```
CREATE TABLE suggested_questions (
    question_id SERIAL PRIMARY KEY,
    question_text TEXT NOT NULL UNIQUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Constraints:

- Ensures unique suggested questions.

1.7. Moderation Logs Table

Stores instances where the chatbot filters inappropriate queries.

```
CREATE TABLE moderation_logs (
    log_id SERIAL PRIMARY KEY,
    user_ip VARCHAR(50) NOT NULL,
    query_text TEXT NOT NULL,
    reason TEXT NOT NULL,
    flagged_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Constraints:

- Ensures logging of flagged queries without user details.

1.8. CRM Chatbot Logs Table

Logs chatbot interactions for integration with CRM.

```
CREATE TABLE crm_chatbot_logs (
    crm_log_id SERIAL PRIMARY KEY,
    query_id INT REFERENCES chatbot_queries(query_id) ON DELETE CASCADE,
    logged_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Constraints:

- Queries are referenced anonymously.

1.9. Uploaded Files Table

Stores metadata of user-uploaded documents.

```
CREATE TABLE uploaded_files (
    file_id SERIAL PRIMARY KEY,
    user_ip VARCHAR(50) NOT NULL,
    file_name VARCHAR(255) NOT NULL,
    file_path TEXT NOT NULL, -- Stores S3 path or disk reference
    processed BOOLEAN DEFAULT FALSE,
    uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

---

2. Relationships Between Tables

- `users` → `predefined_responses` *(one-to-many)*
- `meeting_transcripts` & `ppt_slides` → `chatbot_queries` *(one-to-many, source reference)*
- `chatbot_queries` → `crm_chatbot_logs` *(one-to-one, tracking CRM integration)*
- `chatbot_queries` → `moderation_logs` *(one-to-one, if moderated)*
- `users` → `predefined_responses` *(one-to-many, created_by reference)*

---

3. Indexing & Performance Optimization Strategies

Indexes for Faster Query Performance

1. `INDEX idx_chatbot_queries_query_text ON chatbot_queries(query_text);`
   (Speeds up lookups for user queries)
2. `INDEX idx_predefined_responses_question ON predefined_responses(question);`
   (Speeds up predefined response searches)
3. `INDEX idx_meeting_transcripts_file_name ON meeting_transcripts(file_name);`
   (Speeds up transcript lookups)
4. `INDEX idx_ppt_slides_file_name ON ppt_slides(file_name);`
   (Speeds up slide lookups)
5. `INDEX idx_moderation_logs_user_ip ON moderation_logs(user_ip);`
   (Speeds up flagged user query retrieval)
6. `INDEX idx_uploaded_files_file_path ON uploaded_files(file_path);`
   (Speeds up file lookup for processing)

Partitioning & Retention Strategy

- **Chatbot Query Logs:** Partitioning by date (`logged_at TIMESTAMP` column) to speed up retrieval and **automatic deletion after 3 months**.

```
CREATE TABLE chatbot_queries_old PARTITION OF chatbot_queries FOR VALUES FROM ('2024-04-01') TO ('2024-07-01');
```

- **CRM Logs:** Logged separately and cleaned via automated scripts.

---

4. Security Considerations

Encryption & Data Protection

- **Password Security:**
  - Use **bcrypt hashing** for `users.password_hash`.
- **File Storage:**
  - Encrypt sensitive user-uploaded files before storing in Amazon S3.
- **Query Logging Anonymization:**
  - Only IP addresses are stored without user identifiers.

Access Control

- **Role-Based Access:**
  - Admin-only access to: `users`, `predefined_responses`, `moderation_logs`, and `crm_chatbot_logs`.
- **Database User Permissions:**
  - Restrict database write access for chatbot services.

SQL Injection & Data Sanitization

- Prepared statements for all database queries:

```
cursor.execute("SELECT * FROM predefined_responses WHERE question = %s", (user_input,))
```

---

5. Summary of Key Design Decisions

Feature	Design Decision
Data Storage	PostgreSQL (AWS RDS)
Query Retention	**3 months** retention policy
Email & CRM Integration	Separate logging with **anonymous queries**
Web Users	**Public access (no authentication)**
Admin Management	Only for **predefined responses & logs**
Security Measures	**Encryption, Access Control, Indexing Optimization**

Final Notes

- **Scalability:** Since public users don't need authentication, we use **stateless queries** reducing DB load.
- **Performance:** Heavy indexing and **partitioning on logs** ensures fast lookups.
- **Security**: Chat logs are **anonymized**, passwords are **hashed**, and file uploads are **encrypted** in storage.

This schema ensures an optimized, secure, and scalable database for AI-driven chatbot functionalities. # **API Contract (OpenAPI 3.0 Specification)**

This document outlines a structured **API contract** for the **AI Chatbot System**, adhering to **RESTful best practices** and security considerations.

---

1. API Overview

- **Base URL**: `https://api.company.com/v1`
- **Authentication**: JWT-based authentication for admin actions
- **Security**: AWS Cognito for admin authentication, no authentication for public chatbot users

- **Content-Type**: `application/json`

---

2. API Endpoints & Methods

Module	Endpoint	Method	Description
Chatbot Service	`/chatbot/query`	`POST`	Process user queries and return AI-generated responses.
Knowledge Ingestion	`/knowledge/upload`	`POST`	Upload new knowledge sources (transcripts, slides).
Predefined Responses	`/responses`	`GET`	Retrieve predefined chatbot responses.
	`/responses`	`POST`	Create a predefined response (Admin only).
	`/responses/{id}`	`PUT`	Update a predefined response (Admin only).
	`/responses/{id}`	`DELETE`	Delete a predefined response (Admin only).
Email Query Handling	`/email/handle`	`POST`	Process email queries via Outlook integration.
Logging & Analytics	`/logs`	`GET`	Retrieve chatbot interaction logs (Admin only).
Moderation Service	`/moderation/check`	`POST`	Validate queries before chatbot processing.
CRM Integration	`/crm/log`	`POST`	Log chatbot interactions into CRM.

---

3. API Definitions & JSON Schemas

3.1 - Chatbot Service

Process User Query

- **Endpoint**: `/chatbot/query`
- **Method**: `POST`
- **Authentication**: None (public access)

Request Body

```
{
    "user_ip": "192.168.1.1",
    "query_text": "What is the refund policy?"
}
```

Response Body

```
{
    "response_text": "Our refund policy allows returns within 30 days. Please contact support.",
    "source": "predefined_responses",
    "source_id": 12
}
```

Error Responses

HTTP Code	Message
`400`	Invalid query format.

HTTP Code	Message
`429`	Rate limit exceeded.

3.2 - Knowledge Ingestion Service

Upload Knowledge File

- **Endpoint**: `/knowledge/upload`
- **Method**: `POST`
- **Authentication**: `JWT - Admin Only`

Request Body

```
{
    "file_name": "meeting_transcript_0424.txt",
    "file_url": "s3://knowledge-storage/transcripts/0424.txt",
    "file_type": "transcript"
}
```

Response Body

```
{
    "message": "File uploaded successfully.",
    "file_id": 1023,
    "status": "processing"
}
```

Error Responses

HTTP Code	Message
`400`	Unsupported file type.
`403`	Unauthorized access.

3.3 - Predefined Responses Management

Retrieve All Predefined Responses

- **Endpoint**: `/responses`
- **Method**: `GET`
- **Authentication**: None (public access)

Response Body

```
[
    {
        "response_id": 12,
        "question": "What is your refund policy?",
        "answer": "Our refund policy allows returns within 30 days."
    }
]
```

Create New Predefined Response

- **Endpoint**: `/responses`
- **Method**: `POST`
- **Authentication**: `JWT - Admin Only`

Request Body

```
{
    "question": "How do I reset my password?",
    "answer": "To reset your password, go to Settings > Reset Password."
}
```

Response Body

```
{
    "message": "Response created successfully.",
    "response_id": 23
}
```

Update Predefined Response

- **Endpoint**: `/responses/{id}`
- **Method**: `PUT`
- **Authentication**: `JWT - Admin Only`

Request Body

```
{
    "answer": "Please visit the account settings page to reset your password."
}
```

Response Body

```
{
    "message": "Response updated successfully."
}
```

Delete Predefined Response

- **Endpoint**: `/responses/{id}`
- **Method**: `DELETE`
- **Authentication**: `JWT - Admin Only`

Response Body

```
{
    "message": "Response deleted successfully."
}
```

3.4 - Email Query Handling

Process Email Query

- **Endpoint**: `/email/handle`
- **Method**: `POST`
- **Authentication**: `JWT - Admin Only`

Request Body

```
{
    "email_id": "msg_12345",
    "sender": "customer@example.com",
    "subject": "Refund Process",
    "body": "How do I request a refund?"
}
```

Response Body

```
{
    "reply": "Our refund policy allows returns within 30 days. Please visit our website for details.",
    "auto_handled": true
}
```

3.5 - Logging & Analytics

Retrieve Chatbot Logs

- **Endpoint**: `/logs`
- **Method**: `GET`
- **Authentication**: `JWT - Admin Only`

Response Body

```
[
    {
        "query_id": 1545,
        "user_ip": "192.168.1.1",
        "query_text": "What is the refund policy?",
        "response_text": "Refunds are allowed within 30 days.",
        "logged_at": "2024-06-01T12:45:00Z"
    }
]
```

---

4. Security Considerations

Measure	Description
JWT Authentication	Required for admin routes using AWS Cognito.
Rate Limiting	Public chatbot API limited to 60 requests per minute.
Input Validation	Requests undergo strict format validation.
Content Moderation	Chatbot queries filtered against inappropriate content.
Data Encryption	Sensitive fields (emails, logs) encrypted at rest.

---

5. Error Handling

HTTP Code	Meaning
`400`	Bad Request (Invalid format).
`401`	Unauthorized (Missing or Invalid Token).

HTTP Code	Meaning
`403`	Forbidden (Access Denied).
`404`	Resource Not Found.
`429`	Too Many Requests (Rate Limit Exceeded).
`500`	Internal Server Error.

 Consistent error format:

```
{
    "error": "Invalid request format.",
    "code": 400
}
```

Conclusion

This **API contract** ensures a clean, modular, and secure **chatbot integration** with vital features like **AI query processing, knowledge ingestion, email handling, and CRM logging**. The architecture prioritizes **scalability, security, and maintainability**, leveraging **JWT authentication**, **AWS services**, and **efficient logging** mechanisms.