Chatbot Architecture

1. Objective

To build an AI-driven support chatbot within the Central Tickets (CT) member area that can:

- Answer all customer support questions (login, registration, bookings, third-party bookings, refunds, payments, waiting list, coupons, wallet, membership, etc.)
- Perform basic account-level actions (check refund, cancel booking, join waiting list, update preferences)
- Learn and evolve as our policies and features change
- Operate securely under GDPR and company privacy rules

The chatbot will appear **after user login** inside the member facing website/app.

2. Overview of How the System Will Work

High-level data flow

1. **User (React chat window)** sends a message.
2. **Chatbot API (Node.js microservice)** receives the query.
3. The message is analysed by the **LLM (GPT-4o-mini or Hugging Face model)** to detect:
    - Intent (actionable vs informational)
    - Entities (booking ID, coupon, etc.)
4. If the query is informational, it goes through the **RAG (Retrieval-Augmented Generation)** pipeline:
    - Query converted to vector embedding
    - Vector DB (Pinecone) retrieves the most relevant CT documents
    - Retrieved context + user question passed to LLM for summarisation and answer
5. If the query is actionable, backend routes the intent to the **appropriate CT internal API** (refund, wallet, etc.).
6. The LLM reformats the API response into natural human language and sends it back to the user.

3. Core Components and Their Roles

3.1 LLM (Large Language Model)

- Acts as the **"brain"** of the chatbot.
- Understands natural language, interprets intent, summarises retrieved info, and generates readable responses.
- Models considered:
    - **OpenAI GPT-4o-mini** (fast, reliable, minimal setup)
    - **Hugging Face open models** (Mistral 7B, Llama 3, Gemma 2B) for future self-hosting
- The LLM is used for:
    - Intent detection
    - Generating replies using RAG context
    - Re-phrasing API results into user-friendly text

3.2 RAG (Retrieval-Augmented Generation)

- The process that connects the LLM to our company's internal knowledge.
- Steps:
    - Convert our CT documents into embeddings (numeric vectors).
    - Store those vectors in the Vector DB (Pinecone).
    - When a user asks a question, generate an embedding for the query and search for the most semantically similar text.
    - Pass those text chunks to the LLM so it answers accurately based on real CT data.

- Benefits:
    - Keeps answers consistent with our official documentation.
    - No need for model training or fine-tuning.
    - Easy to update, just replace or re-embed new docs.

3.3 Vector DB (Pinecone)

- Stores all document embeddings for fast semantic search.
- Acts like a search engine that finds meaning, not keywords.
- Two collections planned:

    1. **Official Knowledge Base** – refund rules, wallet usage, membership, etc.
    2. **Email Knowledge Base** – cleaned and rephrased historical support emails (Phase 2).

- Hosted via Pinecone (managed) or self-hosted Qdrant container on AWS EC2.

3.4 Node.js Chatbot Microservice

- Serves as the orchestration layer between React frontend, LLM, Vector DB, and CT internal APIs.

- Handles:

    - Message routing and session management
    - Calling LLM and Pinecone
    - Fetching or performing actions through intents
    - Logging, analytics, and feedback capture

3.5 React Frontend Integration

- A chat window embedded in the member dashboard.
- Shows typing indicator, past messages, and feedback buttons.
- Communicates with the Node.js chatbot API using authenticated requests (Scantum token).

4. Phase1: MVP Chatbot (Official Documentation Based)

Goal

Deliver a reliable, GDPR-safe, and production-ready chatbot that answers all known FAQs and interacts with CT APIs.

Knowledge Base

- Use only verified, written documentation and policies.
- Create short documents for each CT feature:
    - Booking flow
    - Event listing and ticket types
    - Refund process
    - Coupon usage and wallet recharge
    - Waiting list
    - Seat allocation and special access requests
    - Membership benefits

    - Account settings and email preferences
    - Privacy policy and GDPR guidelines

- Store each as text/Markdown or DB rows.
- Convert these docs into vector embeddings and push to Pinecone.

How RAG Works in Phase 1

- When the chatbot gets an informational query, it:

    1. Generates embedding for the query.
    2. Searches Pinecone for similar content.
    3. Retrieves top matches (e.g., refund policy).
    4. Passes them with the query to GPT-4o-mini.
    5. GPT writes a concise, UK-tone reply.

How Actionable Queries Work

- For intent like *cancel booking*, *check wallet balance*, etc.:
    - The microservice classifies the intent.
    - Calls the appropriate CT API endpoint (Laravel backend).
    - LLM rephrases JSON output into plain text.

GDPR and Privacy

- Chatbot will never send PII (emails, phone, names, booking IDs) to LLM.
- All data processing for bookings or refunds happens inside CT APIs only.
- Audit logs stored in MySQL (conversation ID, user ID, message, timestamp).

Updating Rules / Policies

- Example: *Wallet cashback 5 % this month, 4 % next month*.

    - Policy stored as text document in knowledge base.
    - When marketing updates the rule, update the document → regenerate embedding → re-upload to Pinecone.
    - Takes < 5 minutes, no model retraining required.

User Feedback Loop

- Each response will have "👍 / 👎 Was this helpful?"
- Feedback stored in DB → used later to improve docs or RAG results.

Expected Outcome

- Answers all static queries accurately.
- Handles live actions via CT APIs.

5. Phase 2: Email Knowledge Enrichment Project

Purpose

Use 3 years of historical customer-support emails to expand chatbot intelligence and cover edge cases.

Approach

- Treat as **a separate project** running after MVP.

- Pipeline:

    1. Export all support emails.
    2. Filter out auto-replies, signatures, duplicates, disclaimers.
    3. Redact all PII (names, emails, phones, booking IDs).
    4. Extract clean Q→A pairs (customer question + agent reply).
    5. Optionally rephrase answers using an LLM for concise, timeless tone.
    6. Tag each pair by topic (refund, coupon, booking, wallet, etc.) and date.
    7. Remove duplicates.
    8. Embed and store in a second Pinecone collection named `email_knowledge`.

Usage

- During query processing:

    - System first searches **official docs collection**.
    - If similarity score below threshold, fallback to **email knowledge**.

    - Merge top results and send to LLM for combined context.

- This adds realistic phrasing and examples to chatbot responses.

Benefits

- Chatbot learns natural tone and language of real CT agents.
- Covers exceptional or corner-case queries.
- Data stays compliant and controllable.

---

6. Handling Constantly Changing Information

- All promotional or time-sensitive details (e.g., *wallet 5 % cashback this month*) are stored in a **separate dynamic policy table** in MySQL with effective dates.
- The chatbot API checks this table before responding.

Example flow:

```
Query: "How much wallet cashback do I get?"
→ RAG finds general policy doc: "Wallet cashback percentage may vary monthly."
→ API call retrieves current percentage = 4%
→ LLM merges both pieces: "Wallet cashback is currently 4 % until 30 Nov."
```

- Ensures responses always reflect live data.

7. Adding New Features in the Future

Whenever CT adds a new product or feature:

1. Create a short internal doc describing the feature and user process.
2. Add that doc to the knowledge base directory.
3. Run embedding script → upload to Pinecone.
4. (Optional) Update intent list and add new API route if the feature has live actions.

5. Test with sample user queries and deploy.

No retraining or model changes required only data updates.

---

8. GDPR and Security Considerations

- No customer personal data leaves our AWS environment.
- Vector DB stores only general texts, not user-specific records.
- Any refund or booking lookup is handled internally by a secured Laravel API over HTTPS.
- Chatbot microservice logs anonymised metadata only.
- Email dataset anonymised and rephrased before embedding.
- Ensure OpenAI complies with GDPR policies

---

9. Benefits of Official Documentation and Privacy Policy

- Official docs ensure **accuracy, consistency, and compliance**.
- Privacy policy inclusion helps the chatbot answer GDPR or data-rights questions reliably.
- Document-based knowledge reduces dependency on training datasets.
- Updating docs automatically improves chatbot responses.
- Policies act as a single source of truth across departments.

10. Benefits of User Feedback Collection

- Real-time tracking of helpful/unhelpful answers.
- Metrics for top failed queries.
- Easy retraining or doc-updating based on real usage.
- Builds a continuous-learning loop without expensive model training.

11. Infrastructure Summary

Component	Platform	Description
Chatbot API	Node.js on AWS EC2	Gateway between React, LLM, and CT APIs
Knowledge Base Docs	S3 / Git / MySQL	Stores all official CT feature docs
Vector DB	Pinecone (managed)	Stores embeddings for fast semantic search
LLM	GPT-4o-mini (later Hugging Face)	Generates and summarises answers
Core APIs	Laravel/Node	Handles bookings, wallet, membership, etc.
Monitoring	CloudWatch	Tracks latency and success rate