Part 2 Submission: Role-Governed Chatbot System

1. Role-Based Chatbot Flow Structure

The chatbot architecture leverages an ****Isolated Intelligence Silo**** model. Instead of utilizing a single monolithic bot traversing varied conditionals, each authenticated role is dynamically mapped to a specialized agent instance. These instances are governed by distinct personalities and limited functional domains.

Agent Class	Target Role	Persona Profile	Functional Expertise
BuyerBot	**BUYER**	Residential Sales Expert	Floorplans, pricing, EMI options, booking, availability
PartnerBot	**CHANNEL_PARTNER**	Partner Relations Manager	Commission structures, referral policies, payouts
VisitBot	**SITE_VISIT**	Site Visit Coordinator	Visit scheduling, logistics, directions
EnquiryBot	**ENQUIRY**	General Enquiry Specialist	Developer legacy, high-level project overview

2. Routing logic explanation

The system utilizes an initial ****Verified Identity Check**** to establish a secure and dedicated conversational domain:

- **Identity Acquisition**: The user supplies a unique identifier, primarily a phone number.
- **Database Lookup**: The routing engine queries the verified classification data (`classified_leads_output.csv` from Part 1) to retrieve the official Assigned_Role mapped to that user.

- **Instantiation Locking**: Based on the result, a factory patterns initializes the corresponding specialized Agent object (e.g., BuyerBot). The session state acts as a hard boundary, effectively "locking" the user's entire interaction workflow to that strictly defined class. A user identified as a "Buyer" cannot route their query through the "Partner" module regardless of their phrasing.

3. Knowledge isolation strategy

Information security and compartmentalization are enforced via a stringent **Triple-Layer Strategy**:

- **Layer 1: Contextual Memory Constraints**: Each Agent class contains its own dedicated knowledge dictionary upon initialization. A `BuyerBot` simply does not have commission data injected into its context array.
- **Layer 2: Prompts and Persona Scoping**: The core LLM system prompt explicitly mandates the Bot to respond **only** based on its localized knowledge string, ensuring the AI model acknowledges its boundary limitations.
- **Layer 3: Pre-Response Moderation (Guardrails)**: All incoming user queries are algorithmically scrubbed against a "Banned Keyword" array unique to the bot's class **before** LLM inferencing. If a keyword breach occurs (e.g., a Site Visit lead asking about financial payouts), a declarative static refusal is returned, preventing the AI reasoning engine from even parsing the out-of-bounds text.

4. Risk mitigation strategy

Maintaining data integrity and user authorization requires the following proactive security principles:

- **Preventing Cross-Role Data Leakage**: By employing pre-response algorithmic guardrails, sensitive topics are intercepted prior to reaching the language processing layer. This makes unauthorized data hallucination technically implausible within the context window.

- **Handling Incorrect Role Data**: Discrepancies between expected roles and assigned database roles are mediated by a direct ****Reporting Loop****. If the system denies a user access due to incorrect pre-classification, they utilize the "Report Role Mismatch" function. This triggers a priority flag (`ROLE_MISMATCH_REPORTED`) in the admin audit log for rapid remediation.
- **Controlling Hallucinations**: Agent output generation is firmly contextualized within restricted local memory arrays. The system rejects extraneous conversational threads, reducing the risk of generating inaccurate business logic.
- **Compliance Auditability**: Every interaction node—authorized queries and blocked violations—is asynchronously written to `interaction_audit.json` with associated metadata (time, role, phone, policy flag). This creates a persistent and verifiable telemetry trail for compliance supervisors.

4. Architecture diagram