Chatbot For Dumka Engineering College

Submitted by

Akash Kumar

(22040440007)

Under the guidance of

Mrs. Sunidhi Priyadarshini

(Asst. Prof., Department of Computer Science & Engineering)

Polytechnic Compound Road, Dumka, Jharkhand 814101

SRS Report

submitted to

Dumka Engineering College

in

partial fulfilment

for

the award of the degree of

BACHELOR OF TECHNOLOGY

In the department of

Computer Science & Engineering

August 2025

JharkhandUniversity of Technology, Ranchi

Table of Contents

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions, Acronyms, and Abbreviations
2. Overall Description
   - 2.1 Product Perspective
   - 2.2 Product Functions
   - 2.3 User Characteristics
   - 2.4 Operating Environment
   - 2.5 Assumptions and Dependencies
   - 2.6 Tools and Technologies
3. General Requirements
   - 3.1 Non-Functional Requirements
   - 3.2 Functional Requirements
4. Constraints
   - 4.1 Operational and Environmental
   - 4.2 Legal Constraints
   - 4.3 Design Constraints
5. External Interfaces
   - 5.1 User Interfaces
   - 5.2 API and SystemInterfaces
6. System Architecture and Data Flow
7. Data Model & APIs
8. Security & Privacy
9. Deployment, Maintenance & Testing
10. Appendices
11. References

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document provides a detailed overview of the requirements, functions, and architecture for the DEC Chatbot — a conversational virtual assistant tailored for Dumka Engineering College. The aim of this SRS is to establish a clear understanding among stakeholders, including developers, maintainers, testers, and college administration, of what the chatbot system should accomplish, how it will be built, and which standards and practices will govern its evolution and operation.

1.2 Scope

The DEC Chatbot system is designed to provide real-time, automated responses to a wide range of college-related queries originating from students, faculty, and prospective applicants. The system reduces the workload on administrative staff and makes college information easily accessible, improving user satisfaction and operational efficiency. Its capabilities span intent classification, entity extraction, knowledge base retrieval, session management, human-in-the-loop escalation, and notification alerts. The chatbot is engineered for reliability, scalability, speed, and data security.

1.3 Definitions, Acronyms, and Abbreviations

- API: Application Programming Interface
- KB: Knowledge Base
- DB: Database

- NLP :Natural Language Processing
- SSO:Single Sign-On
- PII:Personally Identifiable Information
- RBAC :Role-Based Access Control

2. Overall Description

2.1 Product Perspective

The DEC Chatbot will function as a modular application accessible via a web-based chat interface. The ecosystem comprises a user-friendly frontend, a backend API powered by Flask, an NLP service for intent and entity identification (initially using Dialogflow), a structured knowledge base (PostgreSQL), and a Redis cache for performance enhancement. The design supports easy expansion, such as adding voice interaction, multilingual capability, or analytics integrations in the future.

2.2 Product Functions

- Accepts user input in natural language (typed or spoken if expanded).
- Identifies the user's intent and extracts relevant information.
- Looks up authoritative answers from the college knowledge base.
- Recognizes follow-up context for maintaining conversational flow.
- Allows real-time escalation to college staff/admins when confidence is low or on user request.
- Stores and retrieves chat sessions and transcripts.

- Sends notifications (e.g., deadlines, event reminders).
- Provides an administrative dashboard for content management and analytics.

2.3 User Characteristics

- Primary Users : Students (current and prospective) seeking college information.
- Secondary Users: Faculty, administrative staff managing queries, content, and reports.
- Users are expected to have basic digital skills and internet-enabled devices.

2.4 Operating Environment

- Production: Ubuntu 20.04+ server; Docker containerization; Gunicorn and Nginx for HTTP serving.
- Development: Compatible with Windows, macOS, or Linux with Python 3.10+.
- Web Access: Supports latest versions of Chrome, Firefox, Edge, and Safari; mobile-optimized UI.
- Network: Works reliably on campus and public internet connections; has fallbacks for low-bandwidth conditions.

2.5 Assumptions and Dependencies

- The application assumes consistent access to college-maintained databases and KB resources.
- Regular data backups and infrastructure maintenance are handled by IT staff.

- Timely updates to the knowledge base are provided by authorized administrators.

2.6 Tools and Technologies

Following are used for feature implementation and system development:

- Backend: Python 3.10+, Flask, Gunicorn, Docker
- NLP: Dialogflow (with pluggable NLP abstraction)
- DB: PostgreSQL, Redis (Cache Layer)
- Frontend: HTML5, JavaScript, Responsive CSS
- Admin Dashboard: React (preferred), WTForms, RBAC
- DevOps: Docker Compose, GitHub Actions for CI/CD
- Testing Tools: Pytest, k6, locust, Sentry for live monitoring
- Miscellaneous: Postman for API testing, DBeaver/pgAdmin for DB management

3. General Requirements

3.1 Non-Functional Requirements

- Performance: Average chatbot response time should be less than 1.5 seconds, with 95% of queries handled in under 3 seconds during peak loads (up to 200 simultaneous users).
- Reliability: Target minimum uptime of 99% during business hours, with automated alerts and daily data backups.

- Scalability: The API backend must remain stateless, utilize Redis for session management, and support horizontal scaling through Docker or Kubernetes orchestration.
- Security: All communication must be encrypted with TLS/HTTPS; passwords stored using bcrypt hashing; implement RBAC and minimal privilege for DB access.
- Privacy: Only essential PII is stored, with mechanisms for users to request export or deletion of their data; all logs/analyzed data must be pseudonymized where feasible.
- Maintainability: Code must be modular, well-documented, and accompanied by a robust suite of automated tests and static code checks (linting, formatting).
- Usability: The system must be easy to learn, allow efficient task completion, and maintain consistent feedback for user actions. The UI should use clear language and intuitive controls.
- Availability: The chatbot system must be operational at all hours, with scheduled maintenance windows communicated in advance.
- Portability: The application must be accessible across multiple operating systems (Windows, MacOS, Linux) and devices (PC, tablet, smartphone).
- Look and Feel: The interface design should use a clean, professional palette (3–4 color combinations), standardize on a readable font (e.g., Times New Roman for formality), and provide coherent navigation controls (buttons, sliders, search).

3.2 Functional Requirements

Each function is mapped to clear trigger, input, output, and responsible party for traceability and validation.

No.	Description	Input	Output/Response	Stakeholder
FR-01	Allow natural-language user queries	Text input (≤512 chars)	Chatbot acknowledges and begins processing	Student, Faculty
FR-02	Intent classification with accuracy >85% on core queries	User query	Top 3 intents and confidence values	System
FR-03	Named entity extraction from queries	Query text	Extracted entities (e.g., date, dept.)	System
FR-04	Retrieve fact-based answer from KB or escalate	Intent, Entities	Factual answer, fallback, or escalation	System/Admin
FR-05	Escalation on low-confidence or explicit user request	Query, confidence <60%/user prompt	Admin notified; real-time chat offered	System/Admin
FR-06	Persist and provide recent user sessions	Session ID	List of last 5 chats	User/Admin
FR-07	RBAC-protected dashboard for admin tasks	Admin login	Edit KB, view logs, analytics, export data	Admin
FR-08	Event and escalation notifications by email/push	System trigger/event	Email/push notification sent	Admin/User

4. Constraints

4.1 Operational and Environmental

- The system must operate efficiently on standard college-provided hardware, cloud, or VPS infrastructure as specified.
- Periodic maintenance and vulnerability patching are required for continued secure operation.

4.2 Legal Constraints

- The system must comply with college and national data protection policies.
- Visible privacy policy and user-controls must be provided in the UI.

4.3 Design Constraints

- Only licensed or college-approved libraries, APIs, and tools may be included.
- The NLP engine should be easily swappable (e.g., can replace Dialogflow if needed).

5. External Interfaces

5.1 User Interfaces

- Chat Widget: Responsive design, with typing indicators, quick-reply options, and accessible color scheme.
- Admin Dashboard: Role-based access, tables for KB management, logs, and user controls. Exports support CSV and PDF.
- Mobile View: Optional Android/iOS wrapper for push notifications.

5.2 API and System Interfaces

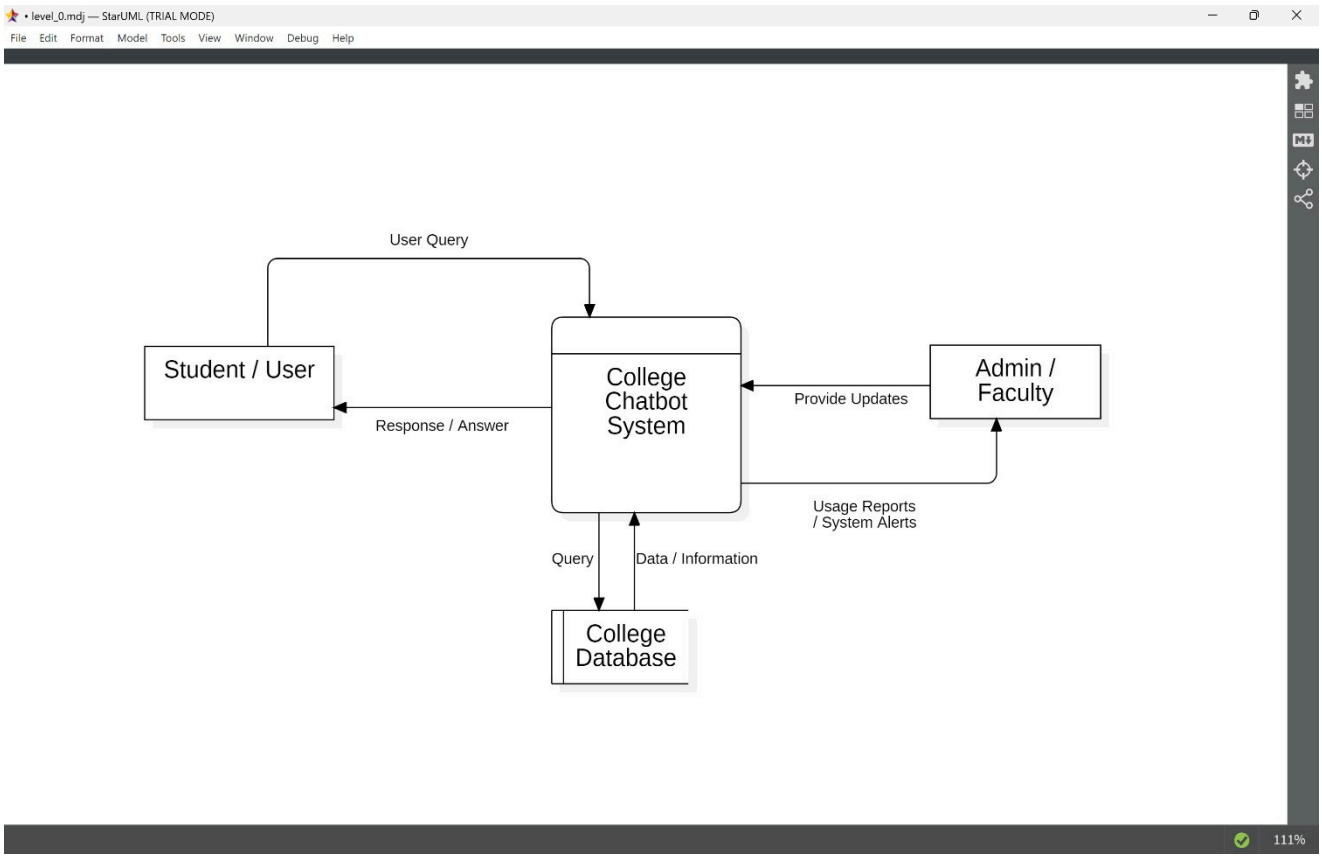
- Primary REST Endpoints:
    - o POST /api/v1/query: Accepts session ID and query, provides intents, confidence, and answer/suggestions.
    - GET /api/v1/session/{id}: Returns chat session transcript and metadata.
    - POST /api/v1/admin/kb: Permits add/update KB entries.
    - GET /api/v1/metrics: Analytics dashboard access (admin only).
- Database/Cache: PostgreSQL for persistent records, Redis for active sessions.

6. System Architecture and Data Flow

6.1 Data Flow Diagrams:

- DFD Level-0:

➢ Level 0 Data Flow Diagram for College Chatbot System



Entities:

- Student / User
- Admin / Faculty

Process:

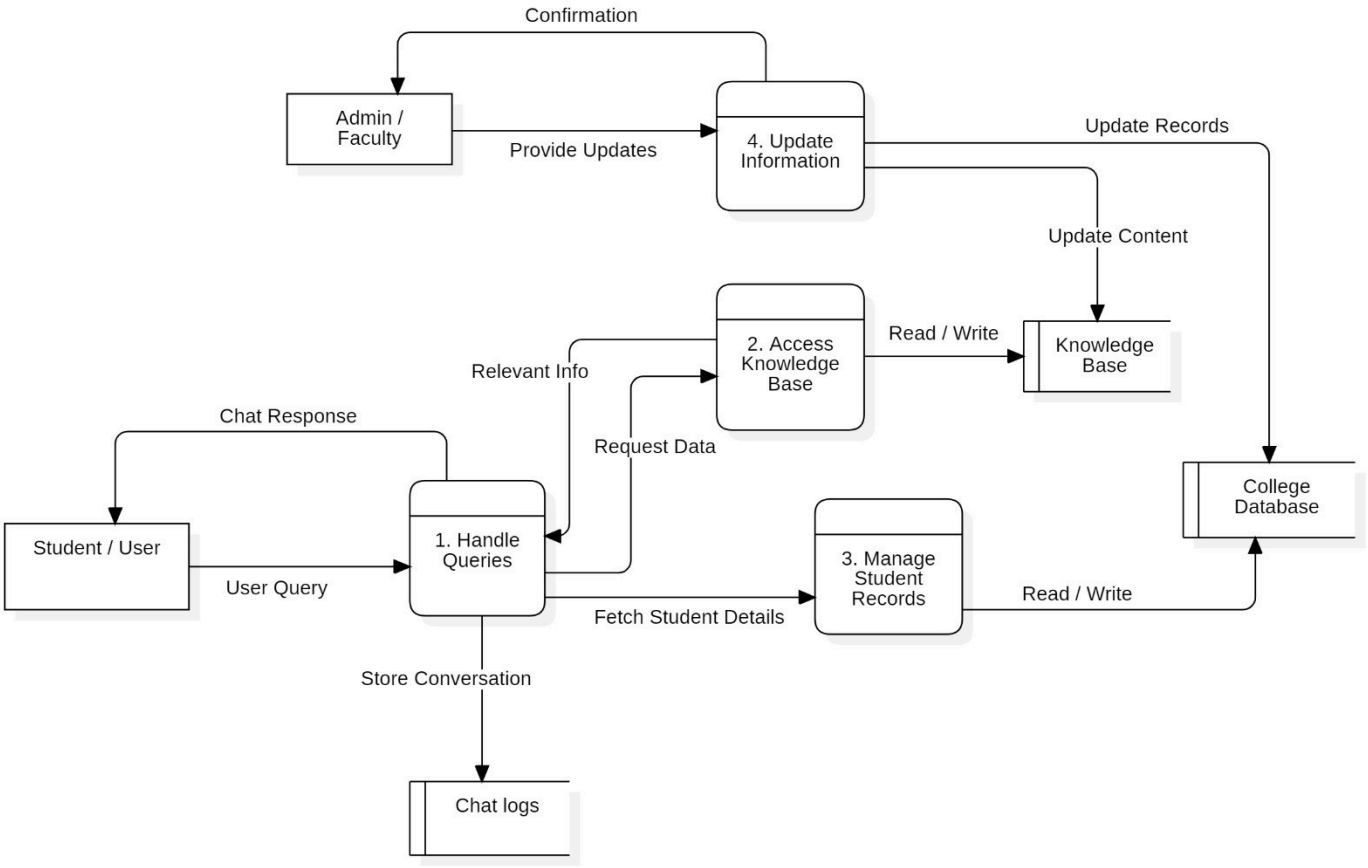
- College Chatbot System

Database:

- College Database

- DFD Level-1:

➢ Expanding ‘ College Chatbot System’



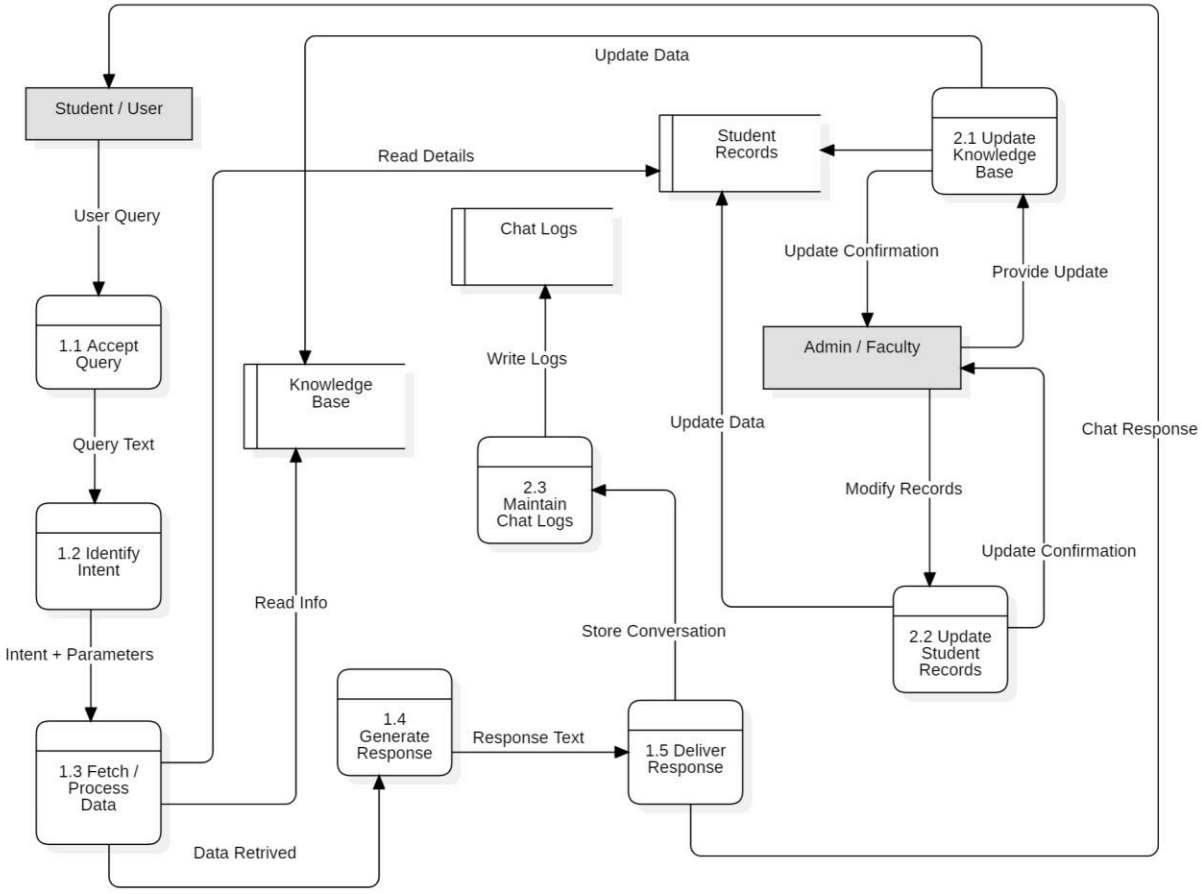
Processes:

1. Handle Queries
2. Access Knowledge Base
3. Manage Student Records
4. Update Information

- DFD Level-2 :

➢ Further expanding Processes-

  - o Handle Queries
  - o Access Knowledge Base



1. Handle Queries:

1.1 Accept Query

1.2 Identify Intent

1.3 Fetch/Process Data

1.4 Generate Response

1.5 Deliver Response

2. Access knowledge Base:

2.1 Update Knowledge Base

2.2 Update Student Records

2.3 Maintain Chat logs

7. Data Model & APIs

- users: id,name, email, role, registration date
- sessions:id, user_id, start/end time, transcript (JSON)
- kb_entries: id, title, intent linkage, content, updated_at
- logs: id,session, request/response, confidence, timestamp

8. Security & Privacy

- All communication is protected by HTTPS/TLS with HSTS.
- Passwords are hashed (bcrypt); admin logins support OAuth2/SSO where available.
- Rate limiting and IP-based throttling reduce abuse.

- All admin actions are audited; data is anonymized/pseudonymized for analytics.
- Backups are stored encrypted at rest.

9. Deployment, Maintenance & Testing

- Services are containerized using Docker. A `docker-compose` script supports local development and deployment.
- Continuous Integration pipeline (using GitHub Actions or similar) runs unit/integration tests, code linting, and builds production images.
- Staging is used for QA before rollout. Production updates follow tagged releases.
- Daily database backups; weekly retention reviewed by IT.
- Monitoring via Prometheus and Grafana. Sentry is enabled for error tracking.
- Testing strategy includes: unit tests for each module, end-to-end/bot API tests, and load testing with k6 or locust.

10. Appendices

10.1 Sample Test Cases

- *TC-01* — Query: "Which B.Tech courses are available?"
  Expected: List of courses, mapped to KB; NLP confidence above 70%.
- *TC-02* — Query: "Show me the fee structure for Computer Science."
  Expected: Fee details plus official college link; fallback if KB missing.
- *TC-03* — Escalation: "I want to speak to a human."
  Expected: Chat escalated to admin, notification sent.

11. References

- Dialogflow Documentation
- Flask Documentation
- OWASP Guidelines
- PostgreSQL Documentation
- IEEE SRS Standard