Bank Bot Low-Level Design Document

Jaideep Jaiswal

26/01/2025

Document Version Control

Date Author	Version	Description
26/01/2025 Jaideep Jaiswal	1.0	Initial LLD for Bank Bot Project

Contents

1. Introduction
    - Purpose of the Document
    - Scope
    - Definitions and Acronyms
2. System Overview
    - Functional Requirements
    - Non-Functional Requirements
3. Detailed Design
    - Module Breakdown
    - User Interaction
    - Backend Operations
    - Database Design
    - Integration Design
4. Process Workflow
    - User Query Handling
    - Suspicious Activity Alert
    - Transaction Notification
5. Error Handling and Logging

6. Performance Metrics
7. Deployment
    - Tools and Technologies
    - Cloud Architecture
8. Test Cases
    - Unit Testing
    - Integration Testing
    - Performance Testing
9. Security Measures
10. Conclusion

1 Introduction

1.1 Purpose of the Document

This Low-Level Design (LLD) document provides a detailed breakdown of the Bank Bot project components. It describes each system module, their internal logic, data flows, and interactions. This document ensures developers can directly implement the program from the specifications outlined here.

1.2 Scope

The scope of the Bank Bot project includes:

- Resolving customer queries using natural language.
- Sending notifications for transactions and suspicious activities.
- Integrating seamlessly with banking APIs.

1.3 Definitions and Acronyms

Term	Description
NLP	Natural Language Processing
API	Application Programming Interface
CI/CD	Continuous Integration/Continuous Deployment
FAQ	Frequently Asked Questions

2 System Overview

2.1 Functional Requirements

- Accept natural language queries from users.
- Provide accurate responses based on pre-trained models (Random Forest).
- Generate transaction summaries and fraud alerts.
- Update users on their recent banking activities.

2.2 Non-Functional Requirements

- **Scalability:** Handle high concurrency during peak hours.
- **Performance:** Deliver responses within 2 seconds.
- **Reliability:** Ensure 99.9% uptime.
- **Security:** Adhere to banking security protocols.

3 Detailed Design

3.1 Module Breakdown

- **User Interface Module:** Handles user input and displays responses.
- **Preprocessing Module:** Cleans and tokenizes input text.
- **Intent Classification Module:** Classifies user queries into predefined categories (e.g., balance inquiry, fraud alert).
- **Response Generation Module:** Dynamically generates responses based on query type and intent.
- **Database Management Module:** Manages user data, transaction records, and interaction logs.
- **Fraud Detection Module:** Analyzes transaction patterns using Random Forest for detection.

3.2 User Interaction

Users initiate interaction through a web interface. The system processes the input and provides a response through the same interface.

3.3 Backend Operations

The backend operates by:

- Preprocessing the user input.
- Vectorizing the input using a TF-IDF model.
- Predicting the appropriate category or response using the Random Forest model.

3.4 Database Design

- User Data
- Transactions
- Interaction Logs
- Fraud Alerts

3.5 Integration Design

APIs connect the chatbot to the banking system. Middleware layers ensure secure and efficient communication between modules.

4 Process Workflow

4.1 User Query Handling

1. User submits a query through the web interface.
2. Query is preprocessed (lowercased, digits and punctuation removed).
3. The preprocessed text is transformed into TF-IDF format.
4. The Random Forest model predicts the category or answer.
5. The appropriate response is sent back to the user.

4.2 Suspicious Activity Alert

1. Transaction data is monitored in real-time.
2. Fraud detection using Random Forest is performed on transactions.
3. Alerts are generated for flagged transactions.

4.3 Transaction Notification

1. Transaction events trigger notifications for users.
2. Notifications are sent to users based on their preferences.

5 Error Handling and Logging

- **Error Categories:**
  - System Errors
  - User Input Errors
- **Logging Mechanisms:**
  - Logs are captured using Python's logging module.
  - Logs include timestamps, error codes, and descriptions for diagnostics.

6 Performance Metrics

- Response time: ¡ 2 seconds.
- Accuracy of Random Forest predictions: ¿ 90%.
- System uptime: 99.9%.

7 Deployment

7.1 Tools and Technologies

- **Programming Languages:** Python
- **Frameworks:** Flask
- **Machine Learning Libraries:** scikit-learn
- **CI/CD Tools:** GitHub Actions
- **Cloud Services:** Render

7.2 Cloud Architecture

- The application is deployed on Render with automatic scaling capabilities.
- Logs and error reports are stored for monitoring and troubleshooting.

8 Test Cases

8.1 Unit Testing

- Test preprocessing functions to handle a variety of inputs.
- Ensure the Random Forest model makes accurate predictions on test data.

8.2 Integration Testing

- Verify the connection between the web interface and backend services.
- Ensure the prediction and response generation work together.

8.3 Performance Testing

- Measure system response time under different loads.
- Test scalability and reliability during peak usage times.

9 Security Measures

- **Authentication:** OAuth 2.0 for secure API access.
- **Encryption:** TLS/SSL encryption for all communication.
- **Monitoring:** Real-time security event monitoring.

10 Conclusion

This document provides a comprehensive LLD for the Bank Bot project. The design includes well-defined modules, processes, and error-handling strategies. It ensures the development team has a clear understanding of the system architecture and implementation steps.