# CHAPTER FOUR

# SYSTEM DESIGN AND IMPLEMENTATION

## 4.1 Introduction

This chapter serves as the technical core of the research project, detailing the transition from the theoretical analysis phase to the practical construction of the Intelligent Chatbot for Northwest University, Kano (NWU). It provides a comprehensive, granular description of the system architecture, the database schema design, and the Unified Modeling Language (UML) diagrams that visualize the system's behavioral and structural characteristics.

Furthermore, this chapter documents the actual implementation process in rigorous detail. It covers the specific algorithms utilized for Natural Language Processing (NLP), the justification for the chosen programming environment, the modular structure of the code, and the testing procedures undertaken to validate the system's performance. The objective is to demonstrate that the developed software not only meets the requirements defined in Chapter Three but also adheres to standard software engineering best practices regarding modularity, scalability, and usability.

## 4.2 System Architecture

The system utilizes a **Client-Server Architecture** (specifically, a Three-Tier Web Architecture) to ensure scalability, maintainability, and a clear separation of concerns. This architectural pattern divides the application into three logical and physical computing tiers:

1. **The Presentation Tier (Client Side):**  
   This is the user-facing interface accessible via a standard web browser (Google Chrome, Mozilla Firefox, Safari, or Opera Mini) on both mobile smartphones and desktop computers. It is responsible for the user experience (UX).
   * **Function:** It accepts user input (text queries) and renders the JSON responses received from the server into a readable chat format. It uses Asynchronous JavaScript and XML (AJAX) to send data without reloading the page, ensuring a smooth, app-like experience.
2. **The Application Tier (Server Side):**  
   Hosted on a Python-based web server using the **Flask** framework, this tier acts as the middleware containing the core business logic.
   * **Function:** It hosts the **NLP Engine**, which is the "brain" of the chatbot. When a request is received, this tier processes the text, performs tokenization and stemming, calculates similarity scores against the trained model, and determines the user's intent. It acts as the bridge between the user interface and the database.
3. **The Data Tier (Database):**  
   This tier consists of the database management system (SQLite for development / MySQL for production) which stores the **Knowledge Base**.
   * **Function:** It serves as the structured repository for all data, including the Intents (categories of questions), Patterns (training sentences), Responses (official answers), and system logs.

## 4.3 System Design

### 4.3.1 Database Design and Entity Analysis

To ensure efficient data retrieval and data integrity, a normalized relational database model was designed. The database is structured around the concept of "Intents," which groups related user queries together.

**Entity-Relationship Analysis:**

The system relies on a **One-to-Many Relationship** between Intents and Patterns (one intent, like "School Fees," can be asked in many ways) and a **One-to-One (or One-to-Many)** relationship between Intents and Responses.

**Table 4.1: The Intents Table**

*Description:* This is the parent table that stores the categories or "topics" the bot can understand. It acts as the primary key reference for the system.

| **Field Name** | **Data Type** | **Constraint** | **Description** |
| --- | --- | --- | --- |
| intent\_id | Integer | Primary Key | Unique identifier for the intent (Auto-increment). |
| tag | Varchar(50) | Unique | A semantic label for the intent (e.g., fees\_science, admission\_req, hostel\_allocation). |
| description | Text | Nullable | Administrative notes explaining the purpose of this tag. |

**Table 4.2: The Patterns Table**

*Description:* This table stores the training data—the various permutations of natural language a student might use to phrase a question.

| **Field Name** | **Data Type** | **Constraint** | **Description** |
| --- | --- | --- | --- |
| pattern\_id | Integer | Primary Key | Unique identifier. |
| intent\_id | Integer | Foreign Key | logical link to the Intents table. |
| pattern\_text | Text | Not Null | The actual phrase (e.g., "How much do Science students pay?", "Science fees breakdown"). |

**Table 4.3: The Responses Table**

*Description:* This table stores the official, verified answers approved by the university management.

| **Field Name** | **Data Type** | **Constraint** | **Description** |
| --- | --- | --- | --- |
| response\_id | Integer | Primary Key | Unique identifier. |
| intent\_id | Integer | Foreign Key | logical link to the Intents table. |
| response\_text | Text | Not Null | The text displayed to the user. It supports HTML tags for formatting (e.g., <br> for line breaks). |

### 4.3.2 Unified Modeling Language (UML) Diagrams

**A. Use Case Diagram**

The Use Case diagram describes the functional requirements of the system by illustrating the interaction between the primary actors and the system boundaries.

* **Actor 1: The Student (User)**
  + *Initiate Session:* The student navigates to the URL and opens the chat widget.
  + *Post Query:* The student types a natural language question (e.g., "When is resumption?").
  + *Receive Feedback:* The student views the instant response generated by the bot.
  + *Reset Chat:* The student clears the conversation history to start a new topic.
* **Actor 2: The Administrator**
  + *Login:* Secure authentication to the backend dashboard.
  + *Update Knowledge Base:* Adding new fee schedules or modifying admission cut-off marks.
  + *Train Model:* Triggering the retraining script after adding new patterns.
  + *View Logs:* Monitoring conversation history to identify unanswered queries.

**B. Sequence Diagram (Query Processing Flow)**

The sequence diagram illustrates the chronological sequence of messages between objects:

1. **Student** inputs a text message via the **User Interface (UI)**.
2. **UI** formats this input into a JSON packet and sends an HTTP POST request to the **Flask Server API**.
3. **Flask Server** receives the payload and forwards the raw text to the **NLP Processor**.
4. **NLP Processor** performs preprocessing (cleaning) and compares the input vector against the **Database/Model**.
5. **NLP Processor** identifies the Intent with the highest confidence score.
6. **Database** retrieves the response\_text associated with that identified Intent.
7. **Flask Server** packages the response into JSON and returns it to the **UI**.
8. **UI** parses the JSON and dynamically renders the message bubble to the **Student**.

## 4.4 System Implementation

### 4.4.1 Development Environment

The implementation was carried out using a specific set of robust, open-source tools selected for their compatibility with AI tasks:

* **Operating System:** Windows 10 Pro (64-bit).
* **Programming Language:** Python 3.9. This version was chosen for its improved dictionary merge operators and stability with ML libraries.
* **Web Framework:** Flask (Micro-web framework). Flask was selected over Django because it is lightweight and allows for greater flexibility in API design, which is ideal for a single-page chatbot application.
* **Key Libraries:**
  + nltk: Used for tokenization, stemming, and stop-word removal.
  + numpy: Used for high-performance array processing and matrix calculations during similarity matching.
  + json: Used for storing and retrieving the training data structure.
  + pickle: Used for serializing the trained model to a file for quick loading.

### 4.4.2 The NLP Algorithm (Pattern Matching & Similarity)

The core intelligence of the chatbot does not rely on "understanding" text like a human, but rather on **Pattern Matching** using the **Bag of Words (BoW)** model and **Cosine Similarity**.

**The Algorithmic Pipeline:**

1. **Preprocessing:**
   * *Tokenization:* The user's input string is split into individual words (tokens).
   * *Stemming:* Words are reduced to their root form using the Lancaster Stemmer (e.g., "paying," "pays," and "paid" all become "pay"). This reduces the complexity of the vocabulary.
   * *Noise Removal:* "Stop words" (common words like "is", "the", "a", "an") are removed as they carry little semantic meaning regarding the intent.
2. **Feature Extraction (Bag of Words):**
   * The system creates a "vocabulary list" of all known words from the training patterns.
   * The user's input is converted into a binary vector (an array of 0s and 1s). If a word from the vocabulary exists in the user's sentence, the corresponding position in the array is marked as 1.
3. **Classification/Matching:**
   * The system compares the user's input vector against the stored pattern vectors.
   * It calculates a **Probability Score**.
   * **Confidence Threshold:** If the highest probability score is below a set threshold (e.g., 0.7 or 70%), the system returns a "Fallback Response" (e.g., "I'm sorry, I didn't understand that. Can you rephrase?"). This prevents the bot from giving wild guesses.

**Pseudo-code of the Logic:**

INPUT: User\_Message  
LOAD: Trained\_Model (Weights, Vocabulary)  
  
1. Clean\_Sentence = Tokenize(User\_Message) + Stem(User\_Message)  
2. Input\_Vector = Bag\_Of\_Words(Clean\_Sentence, Vocabulary)  
  
3. For each Intent\_Class in Model:  
 Probability = Calculate\_Neural\_Activation(Input\_Vector, Intent\_Class)  
  
4. Best\_Intent = Max(Probability)  
5. IF Best\_Intent.Score > 0.75:  
 Response = Random\_Choice(Intents[Best\_Intent].Responses)  
 RETURN Response  
 ELSE:  
 RETURN "I am sorry, I do not understand. Please rephrase."

### 4.4.3 Module Description

To ensure code maintainability, the system is modularized into three main Python files:

1. **train.py (The Trainer):** This script is run only when the database is updated. It reads the raw intents.json file, performs the NLP preprocessing on all patterns, trains the classification model, and saves the output as chatbot\_model.h5 and words.pkl.
2. **chat\_processor.py (The Engine):** This module contains the helper functions. It loads the saved model and exposes a function get\_response(text) which takes raw text and returns the predicted answer. It separates the AI logic from the web logic.
3. **app.py (The Server):** This is the entry point for the Flask application. It handles the URL routing (e.g., @app.route('/') for the homepage). It manages the HTTP requests and responses, serving the HTML frontend and handling the API endpoints.

## 4.5 User Interface Design

The user interface was designed with a **"Mobile-First"** philosophy, recognizing that approximately 95% of Northwest University students access the internet via handheld smartphones.

* **Chat Widget:** The entry point is a non-intrusive floating button (FAB) located at the bottom right of the screen, consistent with modern design standards (Material Design).
* **Chat Interface:** Upon clicking the widget, a card slides up containing:
  + *Header:* Displays the Bot status (Online) and the NWU logo.
  + *Message Area:* A scrollable container where messages appear. User messages align right (Green), and Bot messages align left (Grey).
  + *Input Area:* Contains a text box for typing and a "Send" icon.
* **Accessibility & Branding:** The interface utilizes the official corporate colors of Northwest University (Green and White) to foster a sense of official authority and trust. The font size is set to a readable 16px to accommodate smaller screens.

## 4.6 System Testing

After implementation, the system underwent a rigorous testing phase to verify that it meets the functional and non-functional objectives outlined in Chapter One.

### 4.6.1 Unit Testing

Individual components of the source code were tested in isolation to ensure they function correctly.

* **Tokenizer Test:** Verified that the tokenizer correctly handles punctuation (e.g., "Hello!" becomes ['hello', '!']).
* **Database Connection Test:** Verified that the Flask app connects to the SQLite database without timeout errors.

### 4.6.2 Integration and Compatibility Testing

This phase tested how the different modules work together and how the interface behaves across different platforms.

* **Browser Compatibility:** The application was tested on Google Chrome (Desktop), Firefox (Desktop), and Chrome for Android. The layout remained responsive, and the AJAX calls functioned correctly on all platforms.
* **Latency Test:** The response time was measured. On a standard 4G network, the average response time was recorded at 0.4 seconds, which is well within the acceptable limit for real-time interaction.

### 4.6.3 System Testing (Black Box Testing)

Real-world queries, varying in complexity and spelling accuracy, were entered into the system to verify the accuracy of the Intent Classification.

**Table 4.4: Test Cases and Results**

| **Test Case ID** | **Input (User Query)** | **Expected Output (Intent)** | **Actual Output** | **Status** |
| --- | --- | --- | --- | --- |
| **TC\_01** | "Good morning" | Greeting | "Hello! How can I assist you with your NWU enquiries today?" | **Pass** |
| **TC\_02** | "How much is school fees for CS?" | Fees Information | "The school fees for Computer Science is N55,000 for Indigenes..." | **Pass** |
| **TC\_03** | "When is the deadline for registration?" | Deadline Info | "The portal closes on 30th November 2024." | **Pass** |
| **TC\_04** | "Where can I buy suya?" | Unknown Intent | "I'm sorry, I only answer academic questions." | **Pass** |
| **TC\_05** | "I lost my password" | Password Reset | "Please visit the ICT center to reset your portal password." | **Pass** |
| **TC\_06** | "hw mch is fees" | Fees Information | "The school fees for Computer Science is N55,000..." | **Pass** |

### 4.6.4 Usability Testing

A diverse group of five students from the Computer Science department was invited to test the beta version of the system.

* **Feedback:** All 5 users reported that the interface was intuitive and easy to navigate.
* **Observation:** One user noted that the bot initially struggled with "text speak" or shorthand (e.g., "hw mch 4 fees").
* **Correction:** Based on this feedback, the Patterns table was updated to include common Nigerian student abbreviations like "hw much," "reg," and "dept."

## 4.7 System Documentation

To ensure the long-term maintainability of the software, comprehensive documentation was created.

1. **Inline Documentation:** Comments were added to the Python code explaining the logic of complex blocks (e.g., the cosine similarity calculation).
2. **User Manual:** A README.txt file was created detailing the installation steps:
   * How to install Python and Flask dependencies (pip install -r requirements.txt).
   * How to activate the virtual environment.
   * How to execute the server (python app.py).
   * How to access the admin panel to update the Knowledge Base.