**Final Project Report**

**NLP Chatbot Development using Dialogflow**



**Project Supervisor**

Abdullah Qamar

**Submitted By**

**F2502F9228** (BC210414509)

Name: Muhammad Arslan Akhtar

VUID: BC210414509

**Software Projects & Research Section,**

**Department of Computer Sciences,**

**Virtual University of Pakistan**

|  |
| --- |
| **Table of Contents**  [**CHAPTER-1** 5](#_Toc215775268)  [**GATHERING & ANALYZING INFO** 5](#_Toc215775269)  [1.1 Introduction 6](#_Toc215775270)  [1.2 Purpose 6](#_Toc215775271)  [1.3 Scope 6](#_Toc215775272)  [1.4 Definitions, Acronyms, and Abbreviations 7](#_Toc215775273)  [1.5 Use Cases and Usage Scenarios 8](#_Toc215775274)  [1.5.1 Use Case Diagrams 8](#_Toc215775275)  [1.5.2 Usage Scenarios 9](#_Toc215775276)  [1.6 Supplementary Requirements 9](#_Toc215775277)  [1.6.1 Usability 9](#_Toc215775278)  [1.6.2 Reliability 9](#_Toc215775279)  [1.6.3 Supportability 10](#_Toc215775280)  [1.6.4 System Requirements 10](#_Toc215775281)  [11](#_Toc215775282)  [**CHAPTER-2** 11](#_Toc215775283)  [**PLANNING THE PROJECT** 11](#_Toc215775284)  [2.1 Introduction 12](#_Toc215775285)  [2.2 Methodology 12](#_Toc215775286)  [2.2.1 Available Methodologies 12](#_Toc215775287)  [2.2.2 Chosen Methodology 13](#_Toc215775288)  [2.2.3 Reasons for Chosen Methodology 14](#_Toc215775289)  [2.3 Work Plan 14](#_Toc215775290)  [2.3.1 Project Structure 15](#_Toc215775291)  [2.4 Team Structure 15](#_Toc215775292)  [2.5 Project Schedule (Submission Calendar) 15](#_Toc215775293)  [16](#_Toc215775294)  [**CHAPTER-3** 16](#_Toc215775295)  [**DESIGNING THE PROJECT** 16](#_Toc215775296)  [3.1 Introduction 17](#_Toc215775297)  [3.2 Purpose 17](#_Toc215775298)  [3.3 Scope 17](#_Toc215775299)  [3.3.1 Definitions, Acronyms, and Abbreviations 17](#_Toc215775300)  [3.4 Dynamic Model: Sequence Diagrams 18](#_Toc215775301)  [3.5 Object Model/Logical Model: Class Diagram 19](#_Toc215775302)  [3.6 Database Model (Database Diagram) 20](#_Toc215775303)  [3.7 Graphical User Interfaces 21](#_Toc215775304)  [**CHAPTER-4** 23](#_Toc215775305)  [**DEVELOPMENT** 23](#_Toc215775306)  [4.1 Development Plan (Architecture) 24](#_Toc215775307)  [4.2 Database Implementation 24](#_Toc215775308)  [4.2.1 Courses Table 24](#_Toc215775309)  [4.2.2 Registrations Table 25](#_Toc215775310)  [4.3 Backend Development (Python Logic) 25](#_Toc215775311)  [4.3.1 Configuration and Logging 25](#_Toc215775312)  [4.3.2 Modular Helper Functions 26](#_Toc215775313)  [4.3.3 Main Webhook Logic 28](#_Toc215775314)  [Python 29](#_Toc215775315)  [4.3.4 Student Registration Module 30](#_Toc215775316)  [4.4 Frontend Integration 32](#_Toc215775317)  [34](#_Toc215775318)  [**CHAPTER-5** 34](#_Toc215775319)  [**TESTING AND DEVELOPMENT** 34](#_Toc215775320)  [5.1 Introduction 35](#_Toc215775321)  [5.2 Test Environment 35](#_Toc215775322)  [5.3 Test Cases and Results 35](#_Toc215775323)  [5.3.1 Test Case 1: System Initialization 35](#_Toc215775324)  [5.3.2 Test Case 2: Greeting & Intent Recognition 36](#_Toc215775325)  [5.3.3 Test Case 3: Course Inquiry (Contextual Awareness) 37](#_Toc215775326)  [5.3.4 Test Case 4: Student Registration Validation 38](#_Toc215775327)  [5.4 Performance Testing 39](#_Toc215775328)  [41](#_Toc215775329)  [**CHAPTER-6** 41](#_Toc215775330)  [**CONCLUSION & FUTURE WORK** 41](#_Toc215775331)  [6.1 Conclusion 42](#_Toc215775332)  [6.2 Limitations 42](#_Toc215775333)  [6.3 Future Work 42](#_Toc215775334)  [7. **REFERENCES** 43](#_Toc215775335) |

# **CHAPTER-1**

# **GATHERING & ANALYZING INFO**

## 1.1 Introduction

In the era of digital transformation, educational institutions are increasingly adopting automated solutions to enhance student interaction and administrative efficiency. The project titled **"NLP Chatbot for Zainab Technical Training Institute"** is an Artificial Intelligence-based conversational agent designed to bridge the communication gap between the institute administration and prospective students.

Traditionally, students inquire about course details, fee structures, and schedules by visiting the campus or making phone calls, which is time-consuming and restricted to office hours. This project introduces a virtual assistant that leverages **Natural Language Processing (NLP)** using Google Dialogflow for frontend interaction and **Python (Flask)** with **MySQL** for backend data processing. The chatbot provides real-time, 24/7 assistance, allowing users to query information in natural English and perform tasks such as student registration seamlessly.

## 1.2 Purpose

The primary purpose of this project is to automate the inquiry and registration process for Zainab Technical Training Institute. The specific objectives are:

* **Automation of Information:** To provide instant responses regarding course fees, durations, and modes (Online/On-Campus) without human intervention.
* **24/7 Availability:** To ensure that students can access information at any time, irrespective of institute working hours.
* **Efficient Data Management:** To streamline the student registration process by capturing user details (Name, Phone, Email, Course) directly into a centralized MySQL database.
* **User Engagement:** To guide students towards suitable courses using a rule-based recommendation engine based on their interests.

## 1.3 Scope

The scope of the project defines the boundaries of the system's capabilities. The **Zainab Institute Assistant** covers the following functional areas:

* **Course Inquiries:** The bot can fetch dynamic data from the database regarding active courses, their specific fees, and duration.
* **Student Registration:** A complete dialogue flow that collects student data and saves it to the registrations table in the database.
* **Schedule Management:** Users can inquire about the specific days and timings for classes.
* **Certificate Status:** Students can check the status of their certification by providing their registered email address.
* **Course Recommendations:** The system analyzes keywords (e.g., "IT", "Design", "Office") to suggest relevant training programs.

**Out of Scope:** Currently, the system does not process online fee payments or handle human-agent handovers, which remain part of future enhancements.

## 1.4 Definitions, Acronyms, and Abbreviations

To ensure clarity throughout this documentation, the following technical terms and abbreviations are defined:

* **NLP (Natural Language Processing):** A field of AI that gives machines the ability to read, understand, and derive meaning from human languages.
* **Dialogflow:** A Google-owned developer of human-computer interaction technologies based on natural language conversations.
* **Flask:** A micro web framework written in Python used to build the webhook for this project.
* **MySQL:** An open-source relational database management system used to store course and student records.
* **Ngrok:** A cross-platform application that exposes local server ports to the Internet.
* **IDE:** Integrated Development Environment (VS Code).

## 1.5 Use Cases and Usage Scenarios

This section describes how the user interacts with the system to achieve specific goals.

### 1.5.1 Use Case Diagrams

### 1.5.2 Usage Scenarios

The following scenarios illustrate the system's functionality:

* **Scenario 1: Course Inquiry**
  + **User Action:** The student asks, "What is the fee for Web Development?".
  + **System Response:** The chatbot identifies the intent GetCourseInfo, queries the database for "Web Development", and returns the fee (12000 PKR), duration (6 weeks), and mode.
* **Scenario 2: Student Registration**
  + **User Action:** The student types "Register me".
  + **System Response:** The bot initiates a slot-filling conversation, asking for Name, Email, Phone, and Course.
  + **Validation:** If the user enters an invalid course name, the backend Python logic validates it against the database and prompts the user to select a valid course.
  + **Completion:** Upon success, the data is stored in MySQL, and a confirmation message is displayed.

## 1.6 Supplementary Requirements

These are the non-functional requirements that ensure the system is usable and reliable.

### 1.6.1 Usability

* **Interface:** The system uses a familiar chat interface (Dialogflow Messenger) integrated into a web page, making it intuitive for users.
* **Language:** The chatbot communicates in simple English, handling small talk and greetings to make the interaction feel natural.

### 1.6.2 Reliability

* **Data Integrity:** The system uses MySQL with connection pooling to ensure that student registrations are saved accurately without data loss.
* **Validation:** The backend code implements error handling to manage invalid inputs (e.g., non-existent courses) preventing system crashes.

### 1.6.3 Supportability

* **Modular Code:** The Python backend is structured using helper functions and modular architecture, making it easy to update course lists or fee structures in the database without changing the core code.

### 1.6.4 System Requirements

To run the project environment successfully, the following resources are required:

* **Operating System:** Windows 10/11.
* **Development Tools:** Visual Studio Code, XAMPP (for MySQL), Python 3.x.
* **Internet Connection:** Required for Dialogflow and Ngrok connectivity.
* **Browser:** Google Chrome or Edge for the client-side interface.

## 

# **CHAPTER-2**

# **PLANNING THE PROJECT**

## 2.1 Introduction

Project planning is a critical phase in software engineering that establishes the roadmap for the entire development lifecycle. This chapter outlines the methodology adopted to develop the **NLP Chatbot for Zainab Technical Training Institute**. It details the development model, the work breakdown structure, team roles, and the timeline followed to complete the project milestones successfully. Effective planning ensured that the project met its objectives regarding automating student inquiries and registrations within the allotted timeframe.

## 2.2 Methodology

A software development methodology is a framework that is used to structure, plan, and control the process of developing an information system. It defines the specific phases and tasks that must be completed to deliver high-quality software.

### 2.2.1 Available Methodologies

There are several standard methodologies used in the software industry:

* **Waterfall Model:** A linear and sequential approach where each phase must be completed before the next begins. It is highly disciplined and easy to manage.
* **Agile Methodology:** An iterative approach that focuses on continuous releases and incorporating customer feedback at every step.
* **Spiral Model:** A risk-driven process model generator for software projects that combines elements of design and prototyping.

### 2.2.2 Chosen Methodology

For this project, the **Waterfall Model** was selected as the development methodology. This model was chosen because the requirements for the chatbot (e.g., specific course lists, fee structures, and registration fields) were well-defined and stable at the beginning of the project.

### 2.2.3 Reasons for Chosen Methodology

The Waterfall model was deemed most suitable for this project due to the following reasons:

1. **Clear Objectives:** The requirements (Database of courses, Dialogflow integration) were fixed and understood clearly from the start.
2. **Sequential Submission:** The university project lifecycle requires sequential submissions (SRS, Design Document, Final Code), which aligns perfectly with the Waterfall phases.
3. **Manageability:** As a student project, the linear flow allowed for focused attention on one phase at a time—first designing the database, then coding the webhook, and finally testing the chatbot.
4. **Documentation:** This model emphasizes documentation at every stage, which helped in preparing the final project report.

## 2.3 Work Plan

The project work was divided into distinct phases to ensure organized development:

1. **Requirement Gathering:** Analyzed the needs of Zainab Technical Training Institute, specifically the need to automate manual calls and walk-in queries.
2. **System Design:** Designed the Entity Relationship Diagram (ERD) for the MySQL database and defined the Intents and Entities for Google Dialogflow.
3. **Implementation (Development):**
   * **Frontend:** Configured Dialogflow ES for Natural Language Understanding.
   * **Backend:** Developed the Flask Webhook (webhook.py) in Python to handle logic.
   * **Database:** Created the fyp\_chatbot\_db in MySQL and connected it via connection pooling.
4. **Testing:** Performed unit testing on Python functions and integration testing using the Dialogflow console and local server.

### 2.3.1 Project Structure

The project is structured into three main components:

* **Client Side:** The web interface using HTML/CSS and the Dialogflow Messenger integration.
* **Server Side:** The Python Flask application hosted locally (tunneled via Ngrok), which acts as the API processing user requests.
* **Database Side:** The MySQL relational database storing tables for courses, schedules, registrations, and certificates.

## 2.4 Team Structure

The project was executed by the following team member(s), responsible for the analysis, design, development, and documentation of the system.

* **Member Name:** Muhammad Arslan Akhtar
* **Student ID:** BC210414509
* **Role:** Full Stack Developer (Responsible for Dataset creation, Backend Python coding, Database management, and Report writing).

## 2.5 Project Schedule (Submission Calendar)

The following Gantt chart represents the timeline of the project, highlighting the major milestones achieved during the semester.

## 

# **CHAPTER-3**

# **DESIGNING THE PROJECT**

## 3.1 Introduction

The design phase bridges the gap between requirements and implementation. In this chapter, we define the architectural structure of the **Zainab Technical Training Institute Chatbot**. It details the flow of data between the user, the Dialogflow interface, the Python backend, and the MySQL database. This design ensures that the system is scalable, efficient, and easy to maintain.

## 3.2 Purpose

The purpose of the system design is to create a visual representation of the chatbot’s logic and data storage. It defines:

* How the system handles user queries (Sequence of events).
* How data is structured and stored (Database Schema).
* The logical relationship between different system components.

## 3.3 Scope

The design scope covers the **Backend Logic** (Python Flask Webhook), the **Database Layer** (MySQL), and the **Interaction Layer** (Dialogflow). It focuses on ensuring accurate data retrieval for course details and secure data insertion for student registrations.

### 3.3.1 Definitions, Acronyms, and Abbreviations

* **Webhook:** An HTTP callback that allows Dialogflow to communicate with the Python server.
* **JSON (JavaScript Object Notation):** The data format used to send messages between Dialogflow and the Python backend.
* **ERD (Entity Relationship Diagram):** A flowchart that illustrates how "entities" such as students or courses relate to each other within the system.
* **API Endpoint:** The specific URL (e.g., /webhook) where the Python server listens for requests.

## 3.4 Dynamic Model: Sequence Diagrams

A sequence diagram shows the order in which interactions take place. The following diagram illustrates the flow when a student asks for course details.

1. **User** sends message: "Fee for Python" -> **Dialogflow**.
2. **Dialogflow** identifies Intent -> Sends JSON Request -> **Python Backend**.
3. **Python Backend** executes SQL Query (SELECT fee...) -> **MySQL Database**.
4. **MySQL Database** returns Data (12000 PKR) -> **Python Backend**.
5. **Python Backend** formats Response -> Sends JSON -> **Dialogflow**.
6. **Dialogflow** displays Text -> **User**.

## 3.5 Object Model/Logical Model: Class Diagram

The class diagram represents the static structure of the system. Since our backend is modular, we define classes/modules based on the entities we manage.

## 3.6 Database Model (Database Diagram)

The Entity Relationship Diagram (ERD) defines the structure of the MySQL database fyp\_chatbot\_db. The database consists of four primary tables designed to store course information and student records efficiently.

## 3.7 Graphical User Interfaces

The user interface is based on the Google Dialogflow Messenger integration embedded into the institute's web portal.

*Figure 3.1: Chatbot Welcome Screen*

*Figure 3.2: Course Inquiry Interface*

# **CHAPTER-4**

# **DEVELOPMENT**

## 4.1 Development Plan (Architecture)

The development phase involves translating the design specifications into executable code. The system architecture follows a **Client-Server model** where the client is the web interface (Dialogflow Messenger) and the server is the Python Flask application connected to a MySQL database.

The development was divided into three layers:

1. **Data Layer:** Implementation of MySQL database tables.
2. **Application Layer:** Python script handling logic, database connections, and API routing.
3. **Presentation Layer:** Integration of the chatbot widget into the HTML frontend.

## 4.2 Database Implementation

The first step of development was establishing the data repository. We used **MySQL** to create a relational database named fyp\_chatbot\_db. Below are the SQL commands used to structure the tables.

### 4.2.1 Courses Table

This table stores the static details of all training programs.

SQL

CREATE TABLE courses (

id INT AUTO\_INCREMENT PRIMARY KEY,

course\_name VARCHAR(255) NOT NULL,

fee VARCHAR(50),

duration VARCHAR(50),

mode VARCHAR(50) DEFAULT 'On Campus',

is\_active TINYINT DEFAULT 1

);

### 4.2.2 Registrations Table

This table captures student data dynamically when they request registration via the chatbot.

SQL

CREATE TABLE registrations (

id INT AUTO\_INCREMENT PRIMARY KEY,

student\_name VARCHAR(100),

phone VARCHAR(20),

email VARCHAR(100),

course\_selected VARCHAR(100),

registration\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

## 4.3 Backend Development (Python Logic)

The core logic of the chatbot is built using **Python** and the **Flask** micro-framework. We implemented **Connection Pooling** to handle multiple database requests efficiently and **Logging** for error tracking.

### 4.3.1 Configuration and Logging

We initialized the Flask application and configured the MySQL Connection Pool. This ensures that the system maintains 5 active connections to reduce latency during user interactions.

**Python**

import logging

from flask import Flask, request, jsonify

import mysql.connector

from mysql.connector import pooling

# Logging Setup

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

logger = logging.getLogger(\_\_name\_\_)

app = Flask(\_\_name\_\_)

# Database Configuration with Pooling

db\_config = {

'host': 'localhost',

'user': 'root',

'password': '',

'database': 'fyp\_chatbot\_db',

'pool\_name': "chatbot\_pool",

'pool\_size': 5

}

# Initialize Pool

try:

db\_pool = mysql.connector.pooling.MySQLConnectionPool(\*\*db\_config)

logger.info("✅ Database Connection Pool Created Successfully!")

except Exception as e:

logger.error(f"❌ Error creating DB Pool: {e}")

### 4.3.2 Modular Helper Functions

To follow the **DRY (Don't Repeat Yourself)** principle, we created reusable functions for reading and writing data. This keeps the main code clean and manageable.

**Python**

def execute\_read\_query(query, params=None):

"""Helper to fetch a single row safely using the connection pool."""

conn = None

cursor = None

result = None

try:

conn = db\_pool.get\_connection()

cursor = conn.cursor()

cursor.execute(query, params or ())

result = cursor.fetchone()

except Exception as e:

logger.error(f"DB Read Error: {e}")

finally:

if cursor: cursor.close()

if conn: conn.close()

return result

def execute\_write\_query(query, params):

"""Helper to perform INSERT/UPDATE operations."""

conn = None

cursor = None

try:

conn = db\_pool.get\_connection()

cursor = conn.cursor()

cursor.execute(query, params)

conn.commit()

return True

except Exception as e:

logger.error(f"DB Write Error: {e}")

return False

finally:

if cursor: cursor.close()

if conn: conn.close()

### 4.3.3 Main Webhook Logic

The /webhook route is the entry point for all Dialogflow requests. It parses the JSON payload, identifies the user Intent, and routes the request to the specific function. It also includes "Context Hijacking" logic to handle course corrections during registration.

#### Python

@app.route('/webhook', methods=['POST'])

def webhook():

req = request.get\_json()

user\_message = req.get('queryResult', {}).get('queryText', '') or ''

intent = req.get('queryResult', {}).get('intent', {}).get('displayName', '')

output\_contexts = req.get('queryResult', {}).get('outputContexts', [])

# Special Logic: Intercept Course Correction

correction\_context = next((ctx for ctx in output\_contexts

if 'awaiting\_course\_correction' in ctx.get('name', '')), None)

if correction\_context:

# If user is correcting a course name, force routing to RegisterStudent

saved\_params = correction\_context.get('parameters', {})

req['queryResult']['parameters'] = saved\_params

req['queryResult']['parameters']['course'] = user\_message

return register\_student(req)

# Dynamic Intent Routing

intent\_map = {

"Default Welcome Intent": welcome\_response,

"GetCourseInfo": get\_course\_info,

"GetSchedule": get\_schedule,

"RegisterStudent": register\_student,

"CertificateStatus": get\_certificate\_status,

"ListActiveCourses": list\_active\_courses,

"ContactInfo": get\_contact\_info,

"CourseRecommendation": get\_course\_recommendation

}

if intent in intent\_map:

return intent\_map[intent](req)

else:

return jsonify({'fulfillmentText': f"No handler for intent: {intent}"})

## 4.3.4 Student Registration Module

This is the most critical module. It implements server-side validation to check if the course exists before registering the student. If the course is invalid, it triggers a retry loop using Dialogflow Events.

**Python**

def register\_student(req):

params = req['queryResult']['parameters']

name = params.get('name')

phone = params.get('phone')

email = params.get('email')

course\_input = params.get('course')

session\_id = req['session']

# 1. Validation: Check if course exists in DB

check\_sql = "SELECT course\_name FROM courses WHERE course\_name LIKE %s"

row = execute\_read\_query(check\_sql, (f"%{course\_input}%",))

if not row:

# If Invalid: Trigger RETRY\_COURSE event

return jsonify({

"followupEventInput": {

"name": "RETRY\_COURSE",

"languageCode": "en-US",

"parameters": {

"name": name,

"phone": phone,

"email": email,

"course": "RETRY"

}

}

})

# 2. Registration: Insert into Database

official\_course = row[0]

insert\_sql = """INSERT INTO registrations

(student\_name, phone, email, course\_selected)

VALUES (%s, %s, %s, %s)"""

success = execute\_write\_query(insert\_sql, (name, phone, email, official\_course))

if success:

return jsonify({

'fulfillmentText': (

f"✅ Thank you \*\*{name}\*\*!\n\n"

f"You have been successfully registered for \*\*{official\_course}\*\*.\n"

"Our team will contact you soon."

),

"outputContexts": [{

"name": f"{session\_id}/contexts/awaiting\_course\_correction",

"lifespanCount": 0

}]

})

else:

return jsonify({'fulfillmentText': "❌ System Error: Could not save registration."})

## 4.4 Frontend Integration

The client-side interface was developed using HTML5 and CSS3. We integrated the Google Dialogflow Messenger agent and customized it to be responsive. A media query was added to ensure the chat window fits perfectly on smaller laptop screens by adjusting the viewport height (vh).

**HTML**

<df-messenger

intent="WELCOME"

chat-title="Zainab Institute Assistant"

agent-id="YOUR\_AGENT\_ID"

chat-icon="https://cdn-icons-png.flaticon.com/512/4712/4712027.png"

language-code="en">

</df-messenger>

<style>

df-messenger {

--df-messenger-button-titlebar-color: #1e3d59;

--df-messenger-chat-background-color: #ffffff;

/\* Default Height \*/

--df-messenger-chat-window-height: 500px;

}

/\* Responsive Fix for Laptops \*/

@media screen and (max-height: 850px) {

df-messenger {

--df-messenger-chat-window-height: 60vh;

bottom: 20px;

}

}

</style>

## 

# **CHAPTER-5**

# **TESTING AND DEVELOPMENT**

## 5.1 Introduction

Testing is a crucial phase in the software development lifecycle to ensure the system meets the specified requirements and is free of defects. For the **Zainab Technical Training Institute Chatbot**, we performed both functional and non-functional testing to verify the accuracy of the Natural Language Processing (NLP) models and the reliability of the database transactions.

## 5.2 Test Environment

The system was tested in a local development environment with the following specifications:

* **Server:** Localhost (Flask running on port 5000) tunneled via Ngrok.
* **Database:** MySQL (XAMPP).
* **Client:** Google Chrome (Version 120.0).
* **Interface:** Dialogflow Messenger Integration.

## 5.3 Test Cases and Results

We defined specific test cases to validate different functionalities of the chatbot. The results are documented below.

### 5.3.1 Test Case 1: System Initialization

**Objective:** To verify that the chatbot widget loads correctly on the website. **Input:** User opens the website URL (http://localhost:5500/chatbot.html). **Expected Output:** The "Zainab Institute Assistant" chat widget should appear in the bottom right corner without errors. **Actual Result:** Pass.

Chatbot Interface

*Figure 5.1: Chatbot Interface Initialization*

### 5.3.2 Test Case 2: Greeting & Intent Recognition

**Objective:** To ensure the bot understands basic greetings and introduces itself. **Input:** User types "Hi" or "Hello". **Expected Output:** The bot should reply with the welcome message listing available services. **Actual Result:** Pass.

*Figure 5.2: Default Welcome Intent Response*

### 5.3.3 Test Case 3: Course Inquiry (Contextual Awareness)

**Objective:** To verify that the bot can fetch specific fees and details for different courses from the database. **Input:**

1. User asks: "web development course fee"
2. User asks: "data science course fee and timing" **Expected Output:**
3. Bot returns fee: 12000 PKR, Duration: 6 weeks.
4. Bot returns fee: 10000 PKR, Duration: 6 weeks. **Actual Result:** Pass. The bot successfully queried the MySQL database and returned accurate values for both distinct requests.

*Figure 5.3: Dynamic Database Retrieval for Course Fees*

### 5.3.4 Test Case 4: Student Registration Validation

**Objective:** To ensure the system prevents registration for non-existent courses. **Input:** User tries to register for "Cooking Class". **Expected Output:** The bot should detect that "Cooking Class" does not exist in the courses table and return an error message prompting a retry. **Actual Result:** Pass (As demonstrated during development testing).

*Figure 5.4: Student Registration*

*Figure 5.4.1: Student Registration Database record*

## 5.4 Performance Testing

The system response time was measured during interactions.

* **Average NLP Processing Time:** ~200ms (Google Dialogflow).
* **Database Query Time:** ~50ms (Python Connection Pool).
* **Total Response Latency:** ~0.5 - 1.0 seconds. This falls within the acceptable range for a real-time conversational agent.

## 

# **CHAPTER-6**

# **CONCLUSION & FUTURE WORK**

## **6.1 Conclusion**

The **NLP Chatbot for Zainab Technical Training Institute** was successfully designed, developed, and tested. The project aimed to automate the manual inquiry process, and the results demonstrate that this objective has been met. By integrating **Google Dialogflow** with a custom **Python Flask** backend and **MySQL** database, the system provides a robust solution for:

1. Handling student queries regarding fees and schedules 24/7.
2. Streamlining the registration process by saving data directly to the institute's records.
3. Reducing the workload on administrative staff.

The implementation of **Connection Pooling** and **Context Management** ensured that the system is both efficient and user-friendly, capable of handling conversation loops and error scenarios intelligently.

## 6.2 Limitations

While the system performs its core functions effectively, there are minor limitations:

* **Language Support:** The bot currently supports only the English language.
* **Internet Dependency:** Being cloud-based (Dialogflow), the system requires an active internet connection to function.
* **Static Responses:** Some small talk responses are hardcoded and do not evolve without manual updates.

## 6.3 Future Work

To further enhance the system, the following features are proposed for future iterations:

1. **Urdu Language Support:** Training the NLP model to understand Roman Urdu or native Urdu script for broader accessibility.
2. **Payment Gateway Integration:** allowing students to pay admission fees directly through the chat widget using JazzCash or EasyPaisa APIs.
3. **Admin Dashboard:** Developing a web-based admin panel where the institute staff can view registrations and update course fees without touching the database code.
4. **Voice Interaction:** Enabling voice-to-text functionality so users can talk to the bot instead of typing.

# **REFERENCES**

1. Google Cloud. (2024). *Dialogflow ES Documentation*. Retrieved from <https://cloud.google.com/dialogflow/docs>
2. Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.
3. MySQL. (2024). *MySQL Connector/Python Developer Guide*. Oracle.
4. Jurafsky, D., & Martin, J. H. (2024). *Speech and Language Processing*. Pearson.