Project Documentation

NATURAL LANGUAGE PROCESSING

COURSE CODE-314

Title

Text to UML Diagram Generator

Submitted by

Muhammad Abdullah Nazir

BSAI-078

Instructor name

Muhammad Javed

Department of Computational Sciences

The UNIVERSITY OF FAISALBAD

Table of Contents

1. Project Overview

1.1 Introduction

The Text to UML Diagram Generator is an NLP-powered web application that automatically converts natural language descriptions into professional UML (Unified Modeling Language) class diagrams. This tool bridges the gap between requirement specifications and visual software design, making diagram creation accessible to both technical and non-technical users.

1.2 Motivation

Traditional UML diagram creation requires:

- Deep understanding of UML notation
- Manual drawing using specialized tools
- Significant time investment
- Technical expertise

Our Solution: Simply describe your system in plain English, and get professional UML diagrams instantly!

1.3 Key Features

Natural Language Processing

- Automatic entity/class extraction
- Intelligent attribute detection
- Relationship identification using NLP

Multiple Relationship Types

- Inheritance (Generalization)
- Composition (Strong ownership)
- Aggregation (Weak ownership)
- Association (Usage)

Professional Visualization

- Color-coded relationships

- Clean, readable diagrams
- Interactive class boxes
- Proper UML notation

User-Friendly Interface

- Modern web interface
- Built-in examples
- Real-time generation
- Error handling with helpful messages

1.4 Technology Stack

Backend:

- Python 3.8+
- Flask (Web Framework)
- spaCy (NLP Library)
- Flask-CORS (Cross-Origin Resource Sharing)

Frontend:

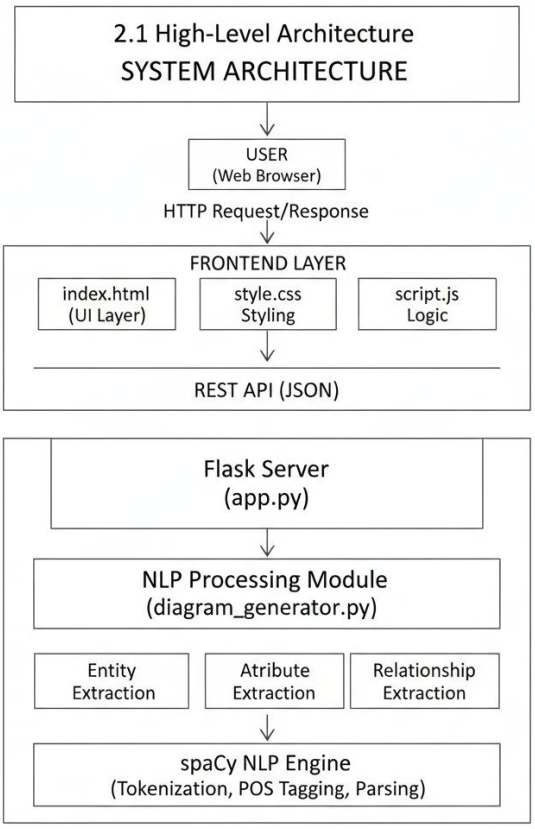
- HTML5
- CSS3
- Vanilla JavaScript
- SVG Graphics

1.5 Project Specifications

- **Project Type:** Full-stack Web Application
- **Domain:** Natural Language Processing + Software Engineering
- **Architecture:** Client-Server (REST API)
- **License:** Open Source
- **Platform:** Cross-platform (Windows, Mac, Linux)

2. System Architecture

2.1 High-Level Architecture



2.2 Component Description

Frontend Components

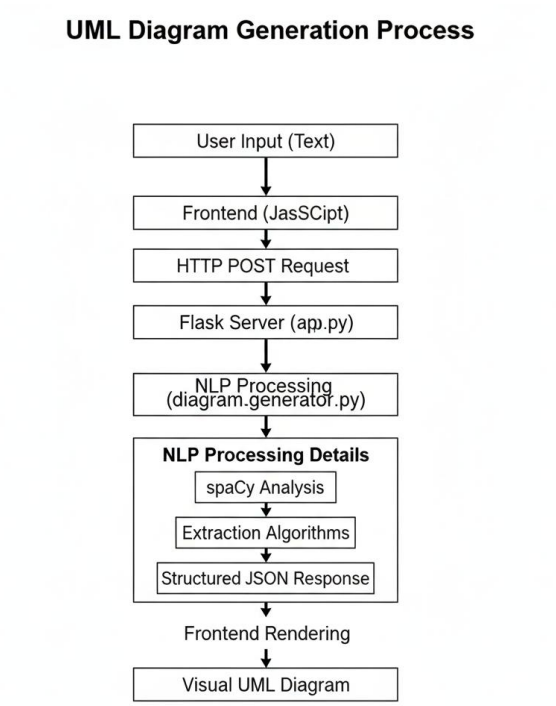
1. **index.html**
    - User interface structure
    - Input text area
    - Canvas for diagram display
    - Example buttons
    - Info panels

2. **style.css**
    - Modern gradient design
    - Responsive layout
    - Interactive elements
    - Color scheme for relationships
3. **script.js**
    - API communication
    - Diagram rendering logic
    - Event handlers
    - SVG generation for arrows

Backend Components

1. **app.py**
    - Flask web server
    - REST API endpoints
    - Request/response handling
    - CORS configuration
2. **diagram_generator.py**
    - Core NLP processing
    - Entity extraction algorithms
    - Attribute detection
    - Relationship identification

2.3 Data Flow



3. Installation Guide

3.1 Prerequisites

Required Software:

- Python 3.8 or higher
- Web browser (Chrome, Firefox, Edge, Safari)
- Internet connection (for initial setup)

Recommended:

- VS Code or any code editor
- 4GB RAM minimum
- Windows 10/11, macOS 10.14+, or Linux

4. User Manual

4.1 Getting Started

Launching the Application

1. **Start Backend Server:**
    - Open Command Prompt
    - Navigate to backend folder
    - Activate virtual environment
    - Run python app.py
    - Keep terminal open
2. **Open Frontend:**
    - Double-click index.html in frontend folder
    - Browser opens automatically
3. **Verify Connection:**
    - You should see the main interface
    - No error messages

Best Practices

✅ DO:

- Use clear, simple sentences
- Start class names with capitals
- Use commas to separate multiple attributes
- Be specific with relationship keywords

❌ DON'T:

- Use lowercase for class names
- Write long, complex sentences
- Mix different relationship types in one sentence

- Use unclear pronouns (it, they, them)

4.2 Common Use Cases

Use Case 1: Software Design

Input your system requirements in natural language, get instant class diagrams for documentation.

Use Case 2: Database Design

Describe entities and relationships, visualize ER diagrams before implementation.

Use Case 3: Learning UML

Students can practice UML by writing descriptions and seeing correct diagrams.

Use Case 4: Rapid Prototyping

Quickly iterate on system design by modifying text and regenerating diagrams.

5. NLP Techniques Used

1. Tokenization

Breaking text into individual tokens (words, punctuation).

Example:

```
Input: "Student has name and age."
Tokens: ["Student", "has", "name", "and", "age", "."]
```

2. Part-of-Speech (POS) Tagging

Identifying grammatical role of each word.

Example:

```
Student/NOUN has/VERB name/NOUN
```

Usage: Identifies nouns as potential classes/attributes.

3. Dependency Parsing

Analyzing grammatical structure and word relationships.

```
Sentence: "Student has name"
Dependencies:
    has (ROOT)
    ├── Student (nsubj - nominal subject)
    └── name (dobj - direct object)
```

4. Named Entity Recognition (NER)

Identifying and classifying named entities.

Adaptation: Treating capitalized nouns as potential class names.

5. Pattern Matching (Regex)

Finding specific text patterns.

Patterns Used:

```regex
class\s+(\w+)          # Explicit class declaration
(\w+)\s+inherits\s+from # Inheritance pattern
(\w+)\s+has\s+([^.]+)  # Attribute pattern
```

TESTING CHECKLIST

Test each example and verify:

- **Classes detected correctly**
- **Attributes shown in boxes**
- **Inheritance arrows (green) point correctly**
- **Composition arrows (red) show ownership**
- **Association arrows (gray) show usage**
- **No errors in console**