Software Design Document

for

Pineapple Chatbot

Prepared by

Kuong Thong
Enrique Castillo
Julio Aguilar
Jose Lopez
Yilin Ruan

CSULA ITS Department

Version History

Version	Date	Summary of Changes
apple	12/4/18	Added Introduction sections (1.1–1.2).
banana	12/5/18	Added early DFD diagrams and transferred initial design content.
cantaloupe	12/6/18	Added database design and sections 8–9.1.
dates	12/7/18	Completed document draft and added validation sections.
elderberry	1/3/19	Updated DFDs to match current model.
farkleberry	3/13/19	Updated feature descriptions and removed outdated content.
grapefruit	3/22/19	Cleaned document structure and updated sections 5–7.
huckleberry	9/25/19	Updated crawling approach.
jackfruit	11/15/19	Removed Amazon Lex dependency.
kiwi	12/2/19	Updated models to reflect spaCy usage.
lemon	2/12/20	Added new model with submodules.
mango	2/19/20	Switched to Simple Bayesian ranker.
nectarine	2/26/20	Fixed duplicate HTML tag entries in the database.
orange	3/11/20	Updated front-end for W3C accessibility compliance.

Contents

1 Introduction

1.1 Purpose

Pineapple Chatbot is an informational chatbot for a standalone web application that helps users find information related to the CSULA website. The chatbot processes user queries, responds with useful answers, and can receive user feedback.

1.2 Intended Audience

This document is intended for developers, testers, designers, project managers, and documentation staff involved in the development, maintenance, and operation of the Pineapple Chatbot system.

1.3 System Overview

The Pineapple Chatbot system crawls university webpages, stores extracted information in a database, and uses natural-language processing (NLP) to interpret user queries. It returns answers to users as text, links, or summaries through a web-based chat interface.

2 Design Considerations

2.1 Assumptions and Dependencies

- Backend is built using Python.
- Core modules and libraries include:
  - MySQL Connector
  - BeautifulSoup
  - `nltk`
  - ChatterBot
  - spaCy
- Other dependencies:
  - NodeJS
  - React (front-end)

2.2 Constraints

- Early versions of the system lack spell-check, web abstraction, and high-accuracy NLP capabilities.
- Overall system quality depends heavily on the data gathered by the crawler; vague or ambiguous user queries may produce incorrect or low-quality results.

2.3 Goals

- Provide helpful, conversational responses to user queries.
- Keep the design simple, following the "Keep It Simple, Stupid" (KISS) principle.

- Provide useful detail without overwhelming users with excessive information.
- Support user satisfaction through clear, understandable output.

2.4 Development Methods

Pair programming and iterative testing were used to refine modules and improve response accuracy. The team followed an incremental approach, integrating and evaluating components frequently in order to improve robustness and usability.

3 Architectural Strategy

Python is used for crawling, NLP, and chatbot logic due to its strong open-source ecosystem and availability of mature libraries for text processing and web scraping. MySQL is used to store processed data because the ChatterBot framework integrates well with SQL-based storage and provides efficient data retrieval.

The chatbot runs inside a web application built with React. This enables a responsive text-based user interface where users can enter queries and view responses within a browser.

4 System Architecture

The system consists of five major modules:

1. **Input Module** – Collects user inputs.
2. **Preprocessor Module** – Cleans and normalizes text.
3. **Logic Module** – Identifies user intent and keywords, and selects an appropriate response.
4. **Storage Module** – Stores and retrieves crawled data and responses.
5. **Output Module** – Formats and returns the final answer to the user.

Data Flow Summary

The high-level data flow of the system can be summarized as follows:

1. User submits a query via the user interface.
2. The query is preprocessed (cleaning, normalization).
3. Intent recognition and keyword identification are performed.
4. Relevant data is retrieved from the storage module.
5. The system constructs and returns a response to the user.

5 Policies and Tactics

5.1 Tools and Technologies

The primary tools and technologies used in the system include:

- ChatterBot
- React
- Python
- MySQL

5.2 Requirements Traceability

Crawled data is backed up regularly to support requirements for reliability and recovery. This ensures that information used by the chatbot can be restored in case of failures or data corruption.

5.3 Testing Approach

User testing and quality assurance (QA) are employed to verify the accuracy and relevance of responses. Test cases cover both the crawling and NLP components, as well as the integrated end-to-end chatbot experience.

5.4 Engineering Trade-offs

Data is stored from many different campus websites in order to improve coverage and answer a broad range of questions. This is balanced against response speed and storage size. Design choices aim to maintain acceptable performance while maximizing informational coverage.

5.5 Coding Guidelines

Standard Python and JavaScript coding conventions are followed. Version control is used throughout the project to manage changes, facilitate collaboration, and track the evolution of the codebase.

5.6 Source Code Organization

- **React App:** User interface components.
- **Python Backend:** Chatbot logic, server-side processing, and integration with the database.
- **Database:** MySQL tables and associated schemas.

5.7 System Build Instructions (Abbreviated)

1. Start the virtual machine (VM) or server hosting the application.
2. Run the React frontend using:

   ```
   npm start
   ```

3. Run the Python backend using:

   ```
   python manage.py runserver
   ```

4. Access the web interface at the configured CSULA server URL (when available).

6 Detailed System Design

6.1 Input Module

Responsibilities:

- Convert user input (text, speech, or terminal input) into `Statement` objects used by the chatbot.

Constraints:

- Must handle plain text and minimal structured formats.

Interactions:

- Interacts directly with user input and passes processed text or `Statement` objects to the Logic Module.

6.2 Preprocessor Module

Responsibilities:

- Clean whitespace.
- Remove HTML artifacts.
- Convert Unicode characters to ASCII where appropriate.

Limitations:

- Must preserve important keywords while removing noise and irrelevant symbols.

6.3 Storage Module

Responsibilities:

- Store responses and crawled data.
- Support efficient queries for keyword and URL lookup.

Technology:

- MySQL database.

Built-in Options:

- ChatterBot `SQLStorageAdapter`
- ChatterBot `MongoDatabaseAdapter` (alternative)

Limitations:

- Requires valid user credentials and appropriate access permissions.
- Large datasets may slow queries and require optimization or indexing.

6.4 Logic Module

Responsibilities:

- Recognize user intent.
- Identify keywords in queries.
- Select the best-matched response based on similarity or ranking.
- Support time-based and mathematical query evaluation where applicable.

Limitations:

- Accuracy depends on the available training data and quality of crawled content.
- Complex or ambiguous phrasing may reduce match accuracy and lead to suboptimal responses.

6.5 Output Module

Responsibilities:

- Format and return the final response to the user via the React frontend.

Possible Output Mechanisms:

- Terminal-based output (for development or debugging).
- API-based output for integration with other systems (if extended).

7 Lower-Level Component Design

7.1 Input Module Components

- Creates `Statement` objects from input.
- Serves as the base class for all input types.
- Only text, dictionary, or `Statement` formats are supported.

7.2 Output Module Components

- Returns `Statement` objects as responses.
- Reads data from the Storage Module before formatting it for display.
- Has minimal performance constraints since most computation occurs in the Logic Module.

7.3 Storage Module Components

- Performs all CRUD (Create, Read, Update, Delete) operations for stored responses and crawled data.
- Requires valid database connection credentials.
- Uses the SQL-based module as the primary implementation.

7.4 Logic Module Components

- Compares user statements with known statements stored in the database.
- Uses similarity thresholds, maximum similarity checks, and exclusion rules to rank candidate responses.
- Selects the closest match and retrieves an appropriate response from the database.

8 Database Design

A MySQL database stores the crawled website data and associated metadata.

Key Table: `termUrlKeywords`

This table contains approximately 300,000 records and includes the following columns:

- **term** – The search term or key phrase.
- **URL** – The target URL associated with the term.
- **keywords** – Additional keywords related to the term and URL.

Additional tables store faculty information and labels for the Bayesian classifier used in query interpretation and ranking.

The database schema is designed to enable fast keyword matching, URL lookup, and long-term storage of crawled data.

9 User Interface

9.1 Overview

The user interface is a simple, chat-style interface built with React. It provides a conversational experience where users can enter queries and view responses within a web application.

9.2 Screens and Layouts

Key UI elements include:

- An input box where users type their queries.
- A response window where the chatbot displays answers, links, and summaries.

9.3 UI Flow Model

1. User opens the web application.
2. User enters a question or statement in the input box.
3. The system processes the input and calls backend APIs.
4. The chatbot generates a response, which is displayed in the response window.
5. Optional: user provides feedback or continues the conversation.

Future enhancements may include richer UI components, additional accessibility improvements, and support for more interactive elements.

10 Requirements Validation and Verification

To ensure correctness, reliability, and usability, the following testing activities are performed:

- **Unit tests** for crawler components, NLP functions, and storage operations.
- **Integration tests** for the end-to-end query flow from user input to response.
- **User testing** to verify clarity, relevance, and accuracy of responses in realistic scenarios.

11 Glossary

Statement
Object representing a processed user input used by the chatbot for comparison and response selection.

Intent
The interpreted purpose or goal of a user query (e.g., asking for information, requesting a URL, etc.).

Similarity Score
A metric used to choose the closest matching response to a given user query based on stored statements and training data.

12 References

- Original Pineapple Chatbot design documents.
- Tutorials and documentation for Python NLP libraries, including ChatterBot, spaCy, `nltk`, and BeautifulSoup.
- MySQL documentation for database configuration, optimization, and query techniques.