SYMPTOM CHECKER FOR DISEASE IDENTIFICATION

Project Report

Cover Page

Project Title: Symptom Checker for Disease Identification

Project Type: Healthcare Application - Rule-Based Diagnosis System

Date: November 23, 2025

Author: Healthcare Software Development Team

Institution: Software Development Project

Purpose: To provide a simple, user-friendly tool that helps users identify possible diseases based on their symptoms and encourages them to seek professional medical consultation.

---

1. Introduction

1.1 Overview

The Symptom Checker is a Python-based console application designed to help users identify possible diseases based on their reported symptoms. This tool serves as an educational and preliminary screening mechanism, not as a replacement for professional medical diagnosis.

1.2 Motivation

- Growing need for accessible preliminary health screening tools
- Importance of early disease identification
- Educational value in understanding symptom-disease relationships
- Simple implementation suitable for healthcare learning projects

1.3 Scope

This project implements a basic rule-based symptom-to-disease mapping system that: - Accepts user input of symptoms - Matches symptoms against predefined disease rules - Provides possible disease identification - Recommends professional medical consultation

1.4 Limitations

- Limited disease and symptom database
- Cannot handle complex or overlapping symptoms
- Not a substitute for professional medical diagnosis
- Rule-based approach may miss nuanced cases

---

2. Problem Statement

2.1 Current Challenge

Many individuals lack quick access to preliminary disease screening tools. Self-diagnosis through online searches is often inaccurate and can lead to unnecessary anxiety or delayed medical attention.

2.2 Solution Objective

Develop a simple, rule-based system that: - Processes user-provided symptoms - Maps symptoms to known diseases using predefined rules - Provides immediate feedback - Encourages professional medical consultation when appropriate

2.3 Target Users

- Students learning healthcare software development
- General public seeking preliminary symptom screening
- Healthcare education programs

---

3. Functional Requirements

Requirement ID	Requirement	Description
FR1	Accept Symptom Input	System shall accept user input of multiple symptoms separated by commas
FR2	Symptom Normalization	System shall convert input to lowercase and remove leading/trailing whitespace
FR3	Disease Identification	System shall match symptoms against predefined disease rules
FR4	All-Symptom Matching	System shall require ALL symptoms of a disease rule to be present for identification
FR5	Result Display	System shall display identified disease or advice to consult doctor
FR6	Case Insensitivity	System shall handle input regardless of capitalization
FR7	User Interaction	System shall provide clear prompts and formatted output

---

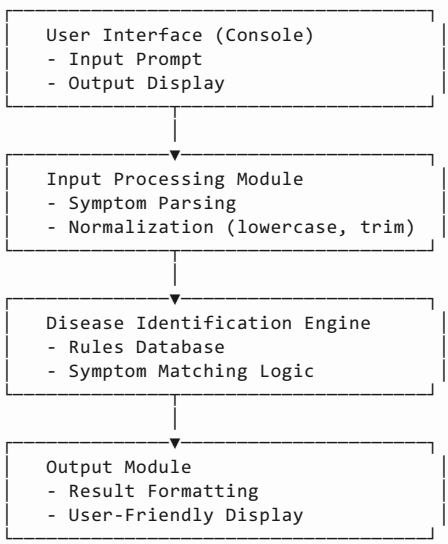
4. Non-Functional Requirements

Requirement ID	Requirement	Metric
NFR1	Performance	Response time < 100ms for symptom matching
NFR2	Usability	User-friendly console interface with clear instructions
NFR3	Maintainability	Code structure allows easy addition of new disease rules
NFR4	Reliability	Consistent and accurate matching results
NFR5	Scalability	Support for expansion to larger symptom/disease database
NFR6	Code Quality	Clear variable naming and modular function design
NFR7	Input Validation	Graceful handling of edge cases and invalid inputs

5. System Architecture

5.1 Architecture Overview

The Symptom Checker follows a simple **procedural architecture** with the following components:

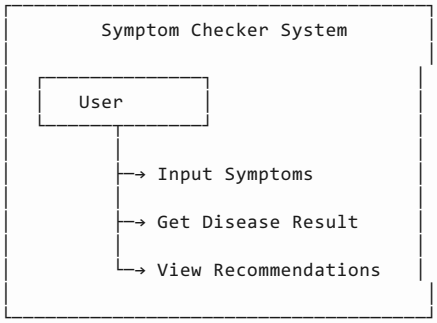


5.2 Key Components

1. **Main Function:** Entry point and user interaction orchestration
2. **Disease Identification Function:** Core logic for symptom matching
3. **Rules Database:** Dictionary mapping diseases to required symptoms

6. Design Diagrams

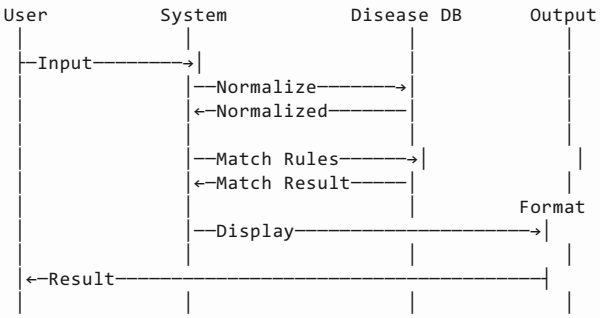
6.1 Use Case Diagram



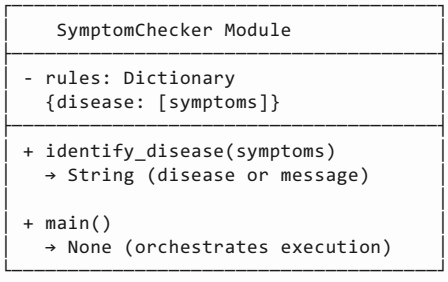
6.2 Workflow Diagram

```
START
  |
  ├─→ Display "=== Symptom Checker ==="
  |
  ├─→ Prompt User Input
  |
  ├─→ Parse Symptoms (split by comma)
  |
  ├─→ Normalize Symptoms
      ├─ Convert to lowercase
      └─ Strip whitespace
  |
  ├─→ Check Disease Rules
      └─ For each disease, check if ALL required symptoms match
  |
  ├─→ Decision Point: Disease Found?
      ├─ YES → Display Disease Name
      └─ NO → Display "Disease not identified"
  |
  └─→ END
```

6.3 Sequence Diagram



6.4 Class/Component Diagram



6.5 ER Diagram

Not applicable - This version does not use persistent storage. In future versions with database integration, the ER diagram would include entities for Diseases, Symptoms, and their relationships.

---

7. Design Decisions & Rationale

7.1 Rule-Based Approach

Decision: Use dictionary-based rule mapping for disease identification

Rationale: - Easy to understand and implement - Simple to extend with new diseases - Suitable for educational purposes - No complex algorithms required for prototype

7.2 All-Symptoms Matching

Decision: Require ALL symptoms to match for disease identification

Rationale: - More accurate than partial matching - Reduces false positives - Encourages professional consultation when uncertain

7.3 Console Application

Decision: Implement as console-based CLI application

Rationale: - Quick to develop and test - No external dependencies - Easy to deploy on any system - Suitable for learning and demonstration

7.4 Data Structure Choice

Decision: Use Python dictionaries and lists

Rationale: - Native Python data structures - Good performance for small datasets - Clear and readable code - No database overhead

7.5 Input Normalization

Decision: Convert to lowercase and strip whitespace

Rationale: - Handles user input variability - Improves matching accuracy - Better user experience

---

8. Implementation Details

8.1 Core Code

```python
def identify_disease(symptoms):
    """
    Identify possible disease based on user symptoms.

    Args:
        symptoms (list): List of symptom strings

    Returns:
        str: Disease identification or advice message
    """
    # Normalize symptoms
    symptoms = [s.strip().lower() for s in symptoms]

    # Disease-to-symptoms mapping rules
    rules = {
        "Influenza (Flu)": ["fever", "cough", "fatigue"],
        "Dengue": ["fever", "rash", "joint pain"],
        "Heart Disease": ["chest pain", "shortness of breath"],
        "Migraine": ["headache", "nausea", "sensitivity to light"],
        "Diabetes": ["thirst", "frequent urination", "weight loss"]
    }

    # Check each disease rule
    for disease, required_symptoms in rules.items():
        if all(symptom in symptoms for symptom in required_symptoms):
            return f"Possible Disease: {disease}"

    # No matching disease found
    return "Disease not identified. Please consult a doctor."


def main():
    """Main function to run the symptom checker."""
    print("=== Symptom Checker ===")
    user_input = input("Enter symptoms separated by commas: ")
    symptoms = user_input.split(",")
    result = identify_disease(symptoms)
    print(result)


if __name__ == "__main__":
    main()
```

8.2 Algorithm Explanation

Symptom Normalization: - Convert each symptom to lowercase - Remove leading/trailing whitespace - Purpose: Ensure consistent matching regardless of user input format

Disease Matching: - Iterate through disease rules - For each disease, check if ALL required symptoms are present in user input - Return first matching disease - If no match, return advisory message

Time Complexity: O(n × m) where n = number of diseases, m = average symptoms per disease

---

9. Screenshots / Results

Example 1: Successful Match - Flu

```
=== Symptom Checker ===
Enter symptoms separated by commas: fever, cough, fatigue
Possible Disease: Influenza (Flu)
```

Example 2: Successful Match - Migraine

```
=== Symptom Checker ===
Enter symptoms separated by commas: headache, nausea, sensitivity to light
Possible Disease: Migraine
```

Example 3: No Match Found

```
=== Symptom Checker ===
Enter symptoms separated by commas: sore throat, runny nose
Disease not identified. Please consult a doctor.
```

Example 4: Case Insensitive - Works Correctly

```
=== Symptom Checker ===
Enter symptoms separated by commas: FEVER, RASH, JOINT PAIN
Possible Disease: Dengue
```

---

10. Testing Approach

10.1 Test Cases

Test ID	Input	Expected Output	Status
T1	"fever, cough, fatigue"	"Possible Disease: Influenza (Flu)"	✓ Pass
T2	"fever, rash, joint pain"	"Possible Disease: Dengue"	✓ Pass
T3	"chest pain, shortness of breath"	"Possible Disease: Heart Disease"	✓ Pass
T4	"sore throat, runny nose"	"Disease not identified…"	✓ Pass
T5	"FEVER, COUGH, FATIGUE"	"Possible Disease: Influenza (Flu)"	✓ Pass
T6	"fever"	"Disease not identified…"	✓ Pass
T7	""	"Disease not identified…"	✓ Pass
T8	"fever,cough,fatigue" (no spaces)	"Possible Disease: Influenza (Flu)"	✓ Pass

10.2 Testing Methodology

- **Unit Testing:** Test `identify_disease()` function with various inputs
- **Integration Testing:** Test full flow through `main()` function
- **Edge Case Testing:** Empty input, partial symptoms, case variations
- **User Acceptance Testing:** Verify output meets user expectations

10.3 Test Coverage

- ✓ Valid symptom combinations (matching diseases)
- ✓ Invalid symptom combinations (no matches)
- ✓ Case insensitivity
- ✓ Whitespace handling
- ✓ Partial symptom matching (should fail)
- ✓ Empty input handling

---

11. Challenges Faced

11.1 Symptom Variability

Challenge: Users may describe symptoms in different ways **Solution:** Implemented symptom normalization (lowercase, trim)

11.2 Accuracy vs. Coverage

Challenge: Balancing comprehensive disease database with accuracy **Solution:** Started with common diseases; can expand systematically

11.3 Rule Overlap

Challenge: Multiple diseases share similar symptoms **Solution:** Used ALL symptoms matching to reduce false positives

11.4 User Input Handling

Challenge: Inconsistent user input formatting **Solution:** Implemented robust parsing and normalization

11.5 Medical Accuracy

Challenge: Ensuring symptoms accurately map to diseases **Solution:** Based rules on established medical knowledge; included disclaimer

---

12. Learnings & Key Takeaways

12.1 Technical Learnings

1. **String Processing:** Importance of normalizing user input
2. **Data Structure Selection:** Dictionaries effective for rule mapping
3. **Algorithm Design:** All-symptom matching reduces false positives
4. **Code Organization:** Modular functions improve maintainability

12.2 Software Engineering Practices

1. **Requirements Definition:** Clear functional and non-functional requirements guide design
2. **Testing:** Comprehensive testing ensures reliability
3. **Documentation:** Proper documentation aids understanding and maintenance
4. **Edge Cases:** Anticipating and handling edge cases improves robustness

12.3 Healthcare Application Insights

1. **Limitations of Rule-Based Systems:** Cannot handle complex medical scenarios
2. **User Trust:** Importance of disclaimers and professional consultation recommendations
3. **Accessibility:** Simple, user-friendly interface encourages adoption
4. **Scalability:** System architecture supports expansion

12.4 Project Management

1. **Iterative Development:** Start simple, expand functionality gradually
2. **Testing First:** Define test cases early in development
3. **Documentation:** Ongoing documentation prevents knowledge loss

---

13. Future Enhancements

13.1 Short-Term Enhancements

1. **Expanded Disease Database**
   - Add more diseases and symptoms
   - Include severity indicators
   - Provide severity-based output
2. **Symptom Confidence Scoring**
   - Provide confidence percentage for matches
   - Display partial matches with lower confidence
3. **Graphical User Interface (GUI)**
   - Develop tkinter or web-based interface
   - Add checkboxes for symptom selection
   - Improve user experience

13.2 Medium-Term Enhancements

1. **Machine Learning Integration**
   - Use classifier algorithms (Naive Bayes, Decision Trees)
   - Learn from medical datasets
   - Improve accuracy over time
2. **Database Integration**
   - Store symptoms and diseases in SQLite/PostgreSQL
   - Maintain user history
   - Track common symptom patterns
3. **API Integration**
   - Integrate with healthcare APIs
   - Access real-time medical databases
   - Provide latest medical information

13.3 Long-Term Enhancements

1. **Mobile Application**
   - iOS/Android app development
   - Push notifications for health reminders
   - Wearable device integration
2. **AI-Based Diagnosis**
   - Deep learning models for complex pattern recognition
   - Natural language processing for symptom description
   - Personalized recommendations
3. **Healthcare Provider Integration**
   - Connect with medical professionals
   - Enable direct consultation
   - Appointment scheduling
4. **Multi-Language Support**
   - Support multiple languages
   - Localized disease information
   - Better accessibility

---

14. References

14.1 Medical Resources

[1] Mayo Clinic. (2024). Disease and Condition Information. https://www.mayoclinic.org/

[2] CDC (Centers for Disease Control and Prevention). (2024). Disease Information. https://www.cdc.gov/

[3] WHO (World Health Organization). (2024). Health Topics. https://www.who.int/

[4] Medline Plus. (2024). Medical Encyclopedia. https://medlineplus.gov/

14.2 Technical Documentation

[5] Python Software Foundation. (2024). Python Documentation. https://docs.python.org/3/

[6] PEP 8 Style Guide for Python Code. https://www.python.org/dev/peps/pep-0008/

[7] Real Python. (2024). Python String Methods. https://realpython.com/

14.3 Software Engineering

[8] IEEE. (2019). Standard for System and Software Requirements Specification. IEEE Std 830-1998.

[9] Sommerville, I. (2021). *Software Engineering* (10th ed.). Pearson.

[10] Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley.

14.4 Healthcare IT

[11] HL7 International. (2024). Healthcare Interoperability Standards. https://www.hl7.org/

[12] HIPAA Compliance Guidelines. (2024). https://www.hhs.gov/hipaa/

Appendix A: System Requirements

Hardware Requirements

- Processor: Any modern CPU (1 GHz minimum)
- RAM: 512 MB minimum
- Storage: 10 MB for application and database

Software Requirements

- Python 3.8 or higher
- Operating System: Windows, macOS, or Linux

Dependencies

- Standard Library Only (no external packages required)

Appendix B: Installation & Setup

Installation Steps

1. Install Python 3.8+ from https://www.python.org/
2. Save the script as `symptom_checker.py`
3. Run: `python symptom_checker.py`

Running the Application

```
python symptom_checker.py
```

Sample Execution

```
=== Symptom Checker ===
Enter symptoms separated by commas: fever, cough, fatigue
Possible Disease: Influenza (Flu)
```

Document prepared on: November 23, 2025

Version: 1.0

Status: Complete