



MediCheck: A Symptom Based Disease Prediction Application

Software Requirements Specification

04.04.2025

Şükrü Can Mayda – 150120031

Turgut Köroğlu – 150121065

Nurbetül Çakır – 150121545

Prepared for
CSE3044 Software Engineering Term Project

Table of Contents

1. INTRODUCTION	4
1.1 PURPOSE	4
1.2 SCOPE	4
1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS	5
1.4 REFERENCES	6
1.5 OVERVIEW	7
2. GENERAL DESCRIPTION	7
2.1 PRODUCT PERSPECTIVE	7
2.2 PRODUCT FUNCTIONS	8
2.3 USER CHARACTERISTICS	8
2.4 GENERAL CONSTRAINTS.....	9
2.5 ASSUMPTIONS AND DEPENDENCIES	9
3. SPECIFIC REQUIREMENTS.....	10
3.1 EXTERNAL INTERFACE REQUIREMENTS	10
3.1.1 <i>User Interfaces</i>	10
3.1.2 <i>Hardware Interfaces</i>	10
3.1.3 <i>Software Interfaces</i>	11
3.1.4 <i>Communications Interfaces</i>	11
3.2 FUNCTIONAL REQUIREMENTS	11
3.2.1 <i>Symptom Input Module</i>	11
3.2.2 <i>Disease Prediction Module</i>	12
3.2.3 <i>Prediction Result Display</i>	12
3.2.4 <i>User Feedback Module</i>	13
3.2.5 <i>User Profile Management</i>	14
3.3 NON-FUNCTIONAL REQUIREMENTS	15
3.3.1 <i>Performance</i>	15
3.3.2 <i>Reliability</i>	15
3.3.3 <i>Availability</i>	15
3.3.4 <i>Security</i>	15
3.3.5 <i>Maintainability</i>	15
3.3.6 <i>Portability</i>	15
3.4 INVERSE REQUIREMENTS	16
3.5 DESIGN CONSTRAINTS	16
3.6 LOGICAL DATABASE REQUIREMENTS	16
3.7 OTHER REQUIREMENTS	17
4. UML DIAGRAMS.....	17
4.1 USE CASES.....	17
4.1.1 <i>Use Case #1: Symptom Input</i>	18
4.1.2 <i>Use Case #2: Disease Prediction</i>	18
4.1.3 <i>Use Case #3: Result Display</i>	18
4.1.4 <i>Use Case #4: Feedback Submission</i>	18
4.1.5 <i>Use Case #5: User Account and History</i>	19
4.1.6 <i>Use Case #6: View Detailed Condition Info</i>	19
4.1.7 <i>Use Case #7: Upload and Train New ML Model</i>	20
4.1.8 <i>Use Case #8: Handle Inactive Session</i>	20
4.1.9 <i>Use Case #9: Condition Search</i>	20
4.1.10 <i>Use Case #10: Clear Inputss</i>	21
4.1.11 <i>Use Case #11: Report Prediction Issue</i>	21

4.1.12 Use Case #12: Export Feedback Data	21
4.1.13 Use Case #13: Register New User	22
4.1.14 Use Case #14: User Login	22
4.1.15 Use Case Diagram	23
4.2 CLASSES / OBJECTS	24
4.2.1 User	24
4.2.2 MainPage	24
4.2.3 SymptomInputHandler	25
4.2.4 SymptomProcessor	26
4.2.5 DiseasePredictor	27
4.2.6 ResultPresenter	27
4.2.7 FeedbackManager	28
4.2.8 ModelManager	29
4.3 SEQUENCE DIAGRAMS	31
4.4 DATA FLOW DIAGRAMS (DFD)	32
4.5 STATE-TRANSITION DIAGRAMS (STD)	33
A. APPENDICES	34
A.1 APPENDIX 1	34
A.2 APPENDIX 2	34
A.3 APPENDIX 3	35
CREDITS	40

1. Introduction

1.1 Purpose

The main goal of this Software Requirements Specification (SRS) is to clearly outline what's needed for creating MediCheck, a cutting-edge disease prediction system powered by machine learning. This system will take the symptoms users provide and suggest potential medical conditions. The SRS aims to detail the features, functionality, and design expectations of the system, ensuring that everyone involved—developers, testers, and project supervisors—shares a common understanding of what's being built and how it should operate. It's crafted for developers, testers, project supervisors or instructors, team members, and future developers, giving them a clear view of the project's scope, features, and technical requirements so they can evaluate its reliability and work on maintaining and enhancing it.

1.2 Scope

The software we're developing is called MediCheck. It's a web-based app that helps predict potential diseases based on the symptoms you enter. By using machine learning algorithms trained on publicly available medical datasets, it offers smart and data-driven suggestions for conditions. The system will:

- Allow users to input multiple symptoms through an interactive and user-friendly interface.
- Analyze those symptoms with trained ML models to predict possible medical conditions.
- Show a ranked list of potential conditions along with brief descriptions.
- Keep getting better over time by allowing model updates as new data comes in.
- Offer general advice, like suggesting that users seek professional medical help when needed.

The system will not:

- Replace professional medical diagnosis or treatment plans.
- Include real-time consultations with doctors.
- Handle emergency or critical care situations.

MediCheck is aimed at individuals who wish to gain preliminary understanding into potential health issues based on observed symptoms. It is useful for:

- Offering quick and easy symptom analysis without needing immediate access to healthcare professionals.
- Supporting early awareness and self-care by offering condition predictions.
- Helping users to make more informed decisions about whether to seek professional medical assistance.

Key Benefits and Goals:

- Accurate multi-symptom analysis powered by machine learning, rather than rule-based logic.
- Fast and flexible interface, capable of returning results in real-time.

- Scalable architecture that allows for future integration of additional data sources and ML enhancements.
- A user-friendly design that clearly presents results and explanations for better transparency.

This scope aligns with the incremental development model chosen for the project, allowing core features to be delivered early while maintaining flexibility for enhancements based on testing and feedback.

1.3 Definitions, Acronyms, and Abbreviations

SRS (Software Requirements Specification): A document that outlines the functional and non-functional requirements for a software system.

ML (Machine Learning): A subset of artificial intelligence (AI) that enables systems to learn from data and improve over time without being explicitly programmed.

UI (User Interface): The space where interactions between humans and machines occur. In this context, it refers to the web interface through which users input symptoms.

Dataset: A structured collection of data used to train and evaluate machine learning models. MediCheck uses open-source medical datasets for training.

Prediction: In the context of ML, it refers to the output generated by the system based on learned patterns from the input data (e.g., predicting possible diseases from symptoms).

Open-source: Refers to software or datasets that are freely available for use, modification, and distribution.

Symptom: A physical or mental feature that is regarded as indicating a condition of disease, especially such a feature that is apparent to the patient.

Diagnosis: The identification of the nature and cause of a certain phenomenon, typically related to a medical condition.

Incremental Model: A software development approach where the system is developed and delivered in small, manageable increments rather than a single final product.

MediCheck: The name of our proposed software system that predicts possible diseases based on user-inputted symptoms using machine learning techniques.

AI (Artificial Intelligence): The simulation of human intelligence in machines that are programmed to think, learn, and make decisions.

UX (User Experience): The overall experience of a person using a product, especially in terms of how easy and pleasant it is to use.

CSV (Comma-Separated Values): A common file format used to store tabular data, often used for training datasets in ML.

Frontend: The client-side portion of a web application that users interact with directly.

Backend: The server-side portion of an application that processes logic, manages data, and communicates with databases.

Training (ML context): The process of feeding a machine learning algorithm with data so that it can learn patterns and improve accuracy.

Medical Condition: A broad term that includes all diseases, disorders, and injuries that affect a patient's health.

Scalability: The ability of a software system to handle increased load or data volume without compromising performance.

NoSQL Database: A type of database that allows flexible and schema-less data storage (e.g., MongoDB).

HTTPS: A secure version of the HTTP protocol used for encrypted communication between client and server.

1.4 References

WebMD Symptom Checker

Title: WebMD Symptom Checker

URL: <https://symptoms.webmd.com>

Publisher: WebMD Health Corp.

Mayo Clinic Symptom Checker

Title: Symptoms - Mayo Clinic

URL: <https://www.mayoclinic.org/symptoms>

Publisher: Mayo Foundation for Medical Education and Research (MFMER)

Isabel Symptom Checker

Title: Isabel Symptom Checker Tool

URL: <https://symptomchecker.isabelhealthcare.com>

Publisher: Isabel Healthcare

Scikit-learn Documentation

Title: Scikit-learn: Machine Learning in Python

URL: <https://scikit-learn.org>

Publisher: scikit-learn developers, Open Source

Public Medical Dataset

Title: Disease Symptom Data (Kaggle Dataset)

URL: <https://www.kaggle.com/datasets/dhivyeshrk/diseases-and-symptoms-dataset>

Publisher: Kaggle

Date: 2022

Cleveland Clinic Health Library

Title: *Health Symptoms & Conditions*

Publisher: Cleveland Clinic

URL: <https://my.clevelandclinic.org/health/symptoms>

MedlinePlus – Medical Encyclopedia

Title: *Medical Encyclopedia – Symptoms Index*

Publisher: U.S. National Library of Medicine

URL: <https://medlineplus.gov/encyclopedia.html>

NHS (UK) – A to Z of Symptoms

Title: *Symptoms Guide*

Publisher: National Health Service (UK)

URL: <https://www.nhs.uk/symptom-checker/>

IEEE Std 830-1998

Title: IEEE Recommended Practice for Software Requirements Specifications (SRS)

Report No.: IEEE Std 830-1998

Publisher: IEEE Computer Society

Date: October 20, 1998

1.5 Overview

This Software Requirements Specification (SRS) document offers a thorough overview of the MediCheck system, which is a web-based application designed to predict potential diseases based on symptoms that user's input, utilizing machine learning techniques. The document begins with an introduction to the system, moves on to a review of similar tools, and then lays out both the functional and non-functional requirements for the software. Additionally, it details the system architecture, development model, user interface design, and key use cases. The SRS is structured to help all stakeholders—developers, testers, and supervisors—navigate the project's scope, objectives, and technical expectations in a clear and organized way.

2. General Description

2.1 Product Perspective

MediCheck is a innovative web-based application that provides symptom-based disease predictions using machine learning algorithms. Unlike traditional symptom checkers that rely on static databases, MediCheck takes a dynamic, data-driven approach. While it shares conceptual similarities with existing tools like WebMD, Mayo Clinic Symptom Checker, and Isabel, it makes difference use of open-source datasets and continuously trainable models, enabling it to evolve and improve over time. MediCheck will not depend on any external commercial systems, but it may integrate with open-source ML libraries and publicly available medical data repositories during development and future expansion.

2.2 Product Functions

MediCheck is designed to assist users in identifying possible medical conditions based on symptoms they experience. The core functionality of the system revolves around a symptom input interface and a machine learning-powered prediction engine. The main product functions are summarized as follows:

- Symptom Input Interface: Users will be able to input one or more symptoms through a simple and intuitive user interface. The system will support both free-text entry or selectable symptom lists.
- Symptom Processing and Standardization: Once symptoms are entered, the system will process and standardize them, mapping them to recognized medical terms to ensure everything is consistent for analysis.
- Disease Prediction Engine: Once symptoms are submitted, MediCheck's machine learning model will analyze the input data and return a ranked list of possible diseases. Each prediction will be accompanied by a confidence score.
- Information Display: For each predicted condition, MediCheck will provide a short description, common symptoms, and general information to help users better understand the result. The system will also include disclaimers that it does not replace professional medical advice.
- Model Update Capability: The system will allow for regular updates and retraining of the underlying ML model as new datasets become available. This ensures improved accuracy and relevance.
- User Feedback Module: Users may optionally provide feedback on the accuracy or relevance of predictions, which could later be used to enhance model performance in future versions.
- Responsive Web Interface: MediCheck will be accessible via a modern web browser, with a responsive layout that supports desktop, tablet, and mobile devices.

2.3 User Characteristics

MediCheck is aimed for non-technical users from the public who are seeking quick, preliminary health insights based on symptoms they experience. These users are expected to have very basic understanding of digital devices, such as the ability to operate a web browser and interact with common user interface elements like buttons, forms, and search bars. Medical background is not required. In addition to general users, secondary user groups may include:

- Students and researchers in the health sector who want to explore symptom-condition relationships through machine learning based tools.
- Developers who may require access to system logs, model updates, and administrative functions. These users are expected to have technical knowledge, including familiarity with web development and machine learning concepts.

The system's interface will be designed to accommodate users of varying ages and educational levels by using clear, jargon-free language and accessible visual elements. No experience with healthcare systems or symptom checkers is necessary for the core functionality. However, the system will clearly communicate its limitations to prevent misunderstanding of the results by users who might be unfamiliar with medical practices.

2.4 General Constraints

Several general constraints will influence the design and development of the MediCheck system:

- **Data Quality:** Since the system relies on open-source medical datasets for training the machine learning model, the quality, accuracy, and completeness of these datasets will directly affect system performance.
- **Ethical Boundaries:** MediCheck must clearly state that it does not provide medical diagnoses and it is not a replacement for professional healthcare. This means the system has to be careful about how it presents results and can't include features like direct treatment recommendations or emergency advice.
- **Platform Independence:** The application must be accessible via modern web browsers across desktop and mobile platforms, which limits the use of certain technologies that may not be compatible.
- **Performance Constraints:** Disease prediction must occur in real-time or near-real-time to ensure a responsive user experience. This limits the complexity of the ML models used and may require optimization or simplification.
- **User Privacy and Data Protection:** Even though the system does not collect identifiable health records, any temporary data handling must comply with general privacy standards. This constraint influences how symptom data is stored and secured.
- **Resource Constraints:** The development team works within academic and has budgetary limitations. Therefore, reliance on free and open-source technologies (e.g., Python, Scikit-learn, Flask) is required, and cloud services or paid APIs may be avoided.

2.5 Assumptions and Dependencies

- **Availability of Open-Source Medical Datasets:** We expect that there will be reliable and publicly available datasets—like those from Kaggle or various public health organizations—that we can use to train our machine learning models. If we can't find these datasets or if they don't meet the necessary quality or quantity, it could really affect how well the system predicts outcomes.
- **Access to Machine Learning Libraries and Tools:** The development of the system is based on having access to open-source libraries such as Scikit-learn, Pandas, NumPy, TensorFlow and PyTorch. If we run into any limitations with these tools whether it's compatibility issues, licensing restrictions, or platform problems we might need to rethink our implementation strategy.

- **Web Browser Compatibility:** We're assuming that users will be accessing MediCheck through modern, standard compliant web browsers like Chrome, Firefox, or Edge. If some users are stuck using outdated or unsupported browsers, they might find that certain features of the user interface don't work quite right and we don't suggest that.
- **Stable Internet Connection:** Since MediCheck is web-based, it assumes users will have a stable internet connection to access the platform. In environments with limited connectivity, user experience and access to prediction results may be broken.
- **Basic Health Literacy of Users:** The system operates on the assumption that users can identify and accurately describe the common symptoms they're facing. If symptoms are misinterpreted or described vaguely, it could affect the relevance and accuracy of the results.
- **Academic Use:** It is assumed that the system is being developed for educational purposes and not designed for money purpose or something else, not for deployment in clinical or commercial environments. Moving to real-world medical applications would require additional validation, certifications, and compliance with legal standards.
- **Team Member Availability:** The project assumes that all listed team members will remain available throughout the development cycle. Changes in team participation could impact project milestones and the scope of deliverables.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

- **EIR-1.1:** The interface shall include a text input for free-text input or a symptom select list.
- **EIR-1.2:** The results page shall present the top 3 or 5 condition predictions with brief descriptions.

3.1.2 Hardware Interfaces

- **EIR-2.1:** No specific hardware interfaces are required; system runs on standard browsers.

3.1.3 Software Interfaces

- **EIR-3.1:** The system should use a trained ML model and called via backend services.
- **EIR-3.2:** The backend shall be developed using Flask or a similar Python framework.

3.1.4 Communications Interfaces

- **EIR-4.1:** The web application shall communicate via HTTP/HTTPS protocols.

3.2 Functional Requirements

3.2.1 Symptom Input Module

3.2.1.1 Introduction / Description

This feature allows users to enter one or more symptoms as the primary input to the system. Inputs can be provided with a search-enabled list or a free-text box. This module serves as the entry point to the disease prediction flow.

3.2.1.2 Inputs / Display

- Input: Free-text symptom entry box or select based list with predefined symptoms.
- Display: symptom input fields, submit button, error messages (if any).

3.2.1.3 Processing

- Validates input to ensure at least one symptom is entered.
- Standardize input text to match medical dataset formats (e.g., lowercase conversion, spelling correction, keyword mapping).
- Sends cleaned symptom list to the prediction module.

3.2.1.4 Outputs

- Forwarded, cleaned symptom list.
- On-screen confirmation of received input.
- Error output if input is invalid or empty.

3.2.1.5 Constraints

- Users can enter up to 10 symptoms per session.
- Only supported browsers (Chrome, Firefox, Edge, Safari...) guaranteed for compatibility.

3.2.1.6 Error/Data Handling

- If no symptoms are entered, display “Please enter at least one symptom.”.
- If the input includes unsupported characters or unknown terms, suggest closest matches.

3.2.2 Disease Prediction Module

3.2.2.1 Introduction / Description

This feature is responsible for generating probable disease predictions based on the user's symptom input, using a trained machine learning model.

3.2.2.2 Inputs / Display

- Input: Standardized symptom list received from the symptom input module.
- Display: Loading animation (only text) while processing, followed by a ranked list of predicted diseases with descriptions.

3.2.2.3 Processing

- Loads the pre-trained machine learning model from storage.
- Transforms symptoms into the required format.
- Performs prediction and calculates confidence score.
- Ranks results in order of confidence score.

3.2.2.4 Outputs

- A list of top 3–5 predicted diseases.
- Associated confidence scores (85%, 77% etc.).
- Descriptive text about each disease, including symptoms.

3.2.2.5 Constraints

- Prediction must be completed within a few seconds under average conditions.
- Limited to predefined condition labels in the training dataset.

3.2.2.6 Error/Data Handling

- If the model returns no matching results, display a generic message like “No clear match found.”
- Log failed predictions stored for further analysis.
- Handles backend errors with user-friendly error messages.

3.2.3 Prediction Result Display

3.2.3.1 Introduction / Description

This module presents the predicted diseases to the user in a clear, understandable, and ranked format. It is responsible for making the output of the ML model accessible and useful to non-technical users.

3.2.3.2 Inputs / Display

- Input: List of predicted conditions and confidence scores from the prediction module.
- Display: Condition name, confidence score, brief description of each disease, message indicating that this is not a professional diagnosis.

3.2.3.3 Processing

- Sorts and formats prediction data for UI.
- Retrieves condition descriptions from the database.
- Implements warning/disclaimer message into result page

3.2.3.4 Outputs

- A web page text showing disease suggestions.
- Optional what to do next section (“Consider seeing a doctor.” etc.).
- Button to return to the home by exiting system or re-enter symptoms.

3.2.3.5 Constraints

- Only top 3–5 conditions will be displayed.
- Results should be readable on mobile and desktop without layout issues.

3.2.3.6 Error/Data Handling

- If any prediction field is missing or corrupted, display a message: “We couldn’t process the result properly. Please try again later.”.
- Ensure the display handles special characters or missing disease descriptions.

3.2.4 User Feedback Module

3.2.4.1 Introduction / Description

This optional feature allows users to provide feedback on accuracy of the predicted diseases. Collected feedback can be used for future model improvements.

3.2.4.2 Inputs / Display

- Input: User rating (out of 10), optional comment.
- Display: Simple feedback form will be shown below the prediction results.

3.2.4.3 Processing

- Validates feedback fields (if submitted).
- Logs data to the backend for storage and possible future analysis.
- Thanks the user and confirms submission with a text.

3.2.4.4 Outputs

- Confirmation and thankful message like “Thanks for your feedback!”.
- Logged feedback data including date, rating, and optional comment.

3.2.4.5 Constraints

- Feedback is optional and does not affect real-time prediction.
- A user can submit feedback only once per session (couldn't be changed).

3.2.4.6 Error/Data Handling

- If feedback submission fails, notify the user: "Could not send feedback. Try again later."
- Store incomplete feedback entries with error tags for debugging (optional).

3.2.5 User Profile Management

3.2.5.1 Introduction / Description

This feature allows users to optionally create a basic account to track their previous symptom inputs and predictions. The system does not store medical history or personal identifiable health data, but only session-level data with user knowing.

3.2.5.2 Inputs / Display

- Input: Username, email, password (those are not essential to use this program, only to track information).
- Display: Login form, registration form, user dashboard (with history list).

3.2.5.3 Processing

- On registration, the system validates the form and creates a user profile.
- On login, user information are verified against stored records.
- Once logged in, the system links each symptom prediction to the user's profile.
- User history is fetched from the database and displayed on the profile page.

3.2.5.4 Outputs

- Confirmation messages after registration and login.
- Display of previous predictions in the user dashboard, including input symptoms, date, and predicted diseases.

3.2.5.5 Constraints

- Email must be unique across all users.
- The password must be at least 6 characters long and should include at least one number and one letter.
- The system does not ask, collect or display sensitive health data beyond what the user submits voluntarily.

3.2.5.6 Error/Data Handling

- If email is already used, show: "This email is already registered."
- If login was unsuccessful, show: "Invalid email or password."
- If database access fails, show an error and suggest retrying later.

3.3 Non-Functional Requirements

3.3.1 Performance

The system should respond to symptom submission and return prediction results within a few seconds in 99.9% of the cases (almost every case). The user interface should load fully within 3 seconds on a standard internet connection (≥ 10 Mbps). The system shall support at least 100 concurrent users without any noticeable drop in performance.

3.3.2 Reliability

The system shall have a Mean Time Between Failures (MTBF) of at least 30 days during the testing phase. In the event of backend failure, the system shall return a message and not break the user session. Prediction results must be reliable, which means same input should yield the same output unless the model has been updated.

3.3.3 Availability

The system should ensure 95% uptime during regular operation and demonstration periods. The planned maintenance windows should not exceed 2 hours per week, and users should be notified at least 1 day before in advance. In case of downtime, a static message should be shown informing users of the unavailability.

3.3.4 Security

No personally identifiable information will be collected or stored by the system. All communication between frontend and backend should occur over HTTPS to prevent any interception. The system shall sanitize all user inputs to prevent injection attacks like XSS or SQL injections. Session data should be cleared up after 15 minutes of inactivity to ensure privacy.

3.3.5 Maintainability

The codebase should follow consistent naming conventions and include inline documentation for at least 80% of the functions. Each module should be developed as an independent component to allow for modular updates (e.g., replacing the ML model without changing the UI). Bug reports or errors shall be logged with timestamps and error codes for easier debugging. The system shall be able to integrate newer datasets for retraining with minimal developer effort.

3.3.6 Portability

The application should be deployable on both Linux and Windows based servers without significant modification. The frontend shall be accessible via all modern browsers including Chrome, Firefox, Safari, and Edge. The system should support both desktop and mobile with responsive design techniques. All external dependencies should be listed and documented to enable easy migration or reinstallation on other environments.

3.4 Inverse Requirements

- This system isn't designed to give professional medical diagnoses or treatment plans. It's meant to be an informational tool that depends on the symptoms your input, and it shouldn't be replaced with a qualified healthcare professional.
- The system should not store or process any personally identifiable health information such as addresses, or medical history.
- The system is not offering real time communication with doctors, clinics, or emergency services.
- The system should not make critical health decisions about the user, such as advising medication, suggesting surgery, or recommending medical procedures.
- The system should not allow unverified data entry into the machine learning model. All model updates must be performed manually by developers.
- The system should not require user registration or login to access core features, ensuring ease of access and user privacy.
- The system should not function offline, as it depends on server processing and external datasets for disease prediction.
- The system should not present results as 100% certain, but instead provide probability suggestions with clear confidence indicators.

3.5 Design Constraints

The MediCheck system's design comes with a set of constraints that shape how we make implementation choices. Since this project is being developed in an academic environment, we need to stick to open-source tools and libraries to keep costs down. This means using Python along with frameworks like Flask or Django and leveraging libraries such as Scikit-learn for our machine learning needs. Moreover, the application is meant to function as a web app, so it must be optimized for moderate resource servers, extending clear of heavy computational models or extensive cloud services. We also have to respect basic data privacy principles, which means we won't be collecting or storing any personal or sensitive health information, the software needs to be accepted usability and accessibility standards to ensure it works well across different devices and browsers. Finally, we want to design the application in a modular and extensible way, allowing for future enhancements like adding new datasets or fine tuning the prediction model without having to override the entire system.

3.6 Logical Database Requirements

The MediCheck system doesn't require a full-scale relational or NoSQL database for its core functionality. However, those may be a part of our system. All data necessary for disease prediction, are stored in structured .xlsx or equivalent files in beginning, which are loaded into memory during training and runtime inference. These files serve as the primary data source and are processed into appropriate vector formats by the machine learning pipeline. The system ensures data integrity by validating the format and structure of the dataset before training or loading. If future updates introduce feedback collection or editable content, integration with a lightweight database like SQLite or PostgreSQL may be considered.

3.7 Other Requirements

- **Accessibility:** The system should comply with basic accessibility guidelines such as WCAG 2.1 Level AA to ensure it is usable by individuals with visual impairments or other disabilities. Features such as high-contrast display, keyboard navigation, and screen reader support should be considered in future iterations.
- **Language and Localization (Future Scope):** The initial version of MediCheck shall support English only. However, the system architecture should allow for easy integration of additional languages in the future through external language files or translation modules according to demand of people.
- **Open-Source Licensing:** Any third-party libraries, datasets, or tools used in the development must be compliant with open-source licenses that permit academic and ensure non-commercial use.
- **Deployment Environment:** The application should be designed for deployment on a Linux-based server using a minimal setup (e.g., Python environment with virtualenv, Gunicorn, Nginx). The system should be able to run with minimal configuration for demonstration purposes.

4. UML Diagrams

4.1 Use Cases

4.1.1 Use Case #1: Entering Symptoms

Use Case ID: UC-01

Use Case Name: Symptom Input

Primary Actor: User

Goal: To submit symptoms for analysis

Preconditions: The user opens the MediCheck application on a compatible browser.

Main Flow:

1. The user navigates to the symptom input page.
2. The user either types in their symptoms or selects them from a dropdown menu.
3. The user clicks the “Submit” button.
4. The system checks the input for validity.
5. The system sends the cleaned symptoms to the prediction module.

Postconditions: Valid symptoms are sent to the next module for processing.

Exceptions:

- If the input is empty, display “Please enter at least one symptom.”
- If the input is invalid, suggest corrections or notify the user.

4.1.2 Use Case #2: Predicting Diseases

Use Case ID: UC-02

Use Case Name: Disease Prediction

Primary Actor: System (Triggered by User Action)

Goal: To generate a list of potential diseases based on the input symptoms

Preconditions: A valid list of symptoms is available.

Main Flow:

1. The system receives the standardized symptom input.
2. The system loads the pre-trained machine learning model.
3. The system performs vector transformation and makes predictions.
4. The system ranks the diseases according to their confidence levels.
5. The results are sent to the display module.

Postconditions: Predictions are ready to be shown.

Exceptions:

- If no matches are found, display “No clear match found.”
- In case of a system error, show a fallback message.

4.1.3 Use Case #3: Displaying Prediction Results

Use Case ID: UC-03

Use Case Name: Result Display

Primary Actor: User

Goal: To view disease predictions and related information

Preconditions: Prediction results are ready

Main Flow:

1. The user is taken to the results page.
2. The system shows the top 3 to 5 diseases along with their confidence scores and brief descriptions.
3. A disclaimer message is included by the system.
4. The user has the option to click a button to start over or provide feedback.

Postconditions: The prediction results are displayed.

Exceptions:

- If the prediction data is incomplete, a generic error message will be shown.

4.1.4 Use Case #4: Submitting Feedback

Use Case ID: UC-04

Use Case Name: Feedback Submission

Primary Actor: User

Goal: To provide feedback on prediction accuracy

Preconditions: Prediction results are visible

Main Flow:

1. The user selects a rating (like thumbs-up or thumbs-down).

2. The user can optionally add a comment.
3. The user clicks “Submit Feedback.”
4. The system validates and logs the feedback.
5. The system confirms that the submission was successful.

Postconditions: The feedback is stored.

Exceptions:

- If the submission fails, the user will be informed and prompted to try again.

4.1.5 Use Case #5: Managing User Profiles

Use Case ID: UC-05

Use Case Name: User Account and History

Primary Actor: User

Goal: To register or log in and view previous predictions

Preconditions: The user visits the login or registration page

Main Flow:

1. The user registers using their email, username, and password.
2. The system validates the information and creates the account.
3. The user logs in with their credentials.
4. The system links new predictions to the user’s profile.
5. The user visits their dashboard to view their history.

Postconditions: The user account is active, and the prediction history is accessible.

Exceptions:

- If the email is already in use, a warning will be displayed.
- If the credentials are incorrect, a login error will be shown.
- If there’s a database failure, a general error message will be displayed.

4.1.6 Use Case: View Disease Information

Use Case ID: UC-06

Use Case Name: View Detailed Condition Info

Primary Actor: User

Goal: To access more in-depth information about a predicted condition

Preconditions: The user has received prediction results

Main Flow:

1. The user clicks on a predicted condition.
2. The system retrieves the disease description.
3. A modal or new section appears, displaying details like symptoms, causes, and possible treatments.

Postconditions: The user gains a better understanding of the selected condition

Exceptions:

- If the content isn’t available, display: “Detailed information not found.”

4.1.7 Use Case: Admin Model Update

Use Case ID: UC-07

Use Case Name: Upload and Train New ML Model

Primary Actor: Admin

Goal: To refresh the ML model with new data

Preconditions: The admin is logged in

Main Flow:

1. The admin goes to the admin panel.
2. The admin uploads a new dataset (like CSV or XLSX).
3. The system checks the dataset format for validity.
4. The admin clicks "Train Model."
5. The model is trained and saved for future predictions.

Postconditions: The new model is now active

Exceptions:

- If the format is invalid, show: "Dataset format not supported."
- If training fails, notify: "Model training failed. See logs."

4.1.8 Use Case: Session Expiry Handling

Use Case ID: UC-08

Use Case Name: Handle Inactive Session

Primary Actor: System

Goal: To safeguard user privacy by clearing data after a period of inactivity

Preconditions: The user has been inactive for a set time (like 15 minutes)

Main Flow:

1. The system monitors session activity.
2. After the timeout, the system clears the session.
3. The user is redirected to the homepage.

Postconditions: Session data is deleted

Exceptions:

- If the user interacts before the timeout, the session is extended.

4.1.9 Use Case: Search Condition Database

Use Case ID: UC-9

Use Case Name: Condition Search

Primary Actor: User

Goal: To manually browse or search through known conditions

Preconditions: User goes to the "Explore Diseases" section

Main Flow:

1. User enters a keyword (like “diabetes”)
2. The system retrieves matching condition entries
3. Results are displayed with names and a brief description

Postconditions: User can access condition pages

Exceptions:

- If there are no results, show: “No condition matched your search.”

4.1.10 Use Case: Clear Input Form

Use Case ID: UC-10

Use Case Name: Clear Inputs

Primary Actor: User

Goal: To reset the entered symptoms without refreshing the page

Preconditions: User has input one or more symptoms

Main Flow:

1. User clicks the “Clear” button
2. The system clears all input fields

Postconditions: The form is reset

Exceptions: –

4.1.11 Use Case: Report Incorrect Prediction

Use Case ID: UC-11

Use Case Name: Report Prediction Issue

Primary Actor: User

Goal: To inform the team about an inaccurate or irrelevant prediction

Preconditions: User has finished a prediction session

Main Flow:

1. User clicks “Report” next to a listed disease
2. User chooses a reason (like “Not relevant” or “Inaccurate”) and submits
3. The system logs the report and details for review

Postconditions: The report is available for backend review

Exceptions:

- If the submission fails, notify the user accordingly

4.1.12 Use Case: Export Feedback Reports (Admin)

Use Case ID: UC-12

Use Case Name: Export Feedback Data

Primary Actor: Admin

Goal: To export user feedback for analysis

Preconditions: Admin has access to the backend dashboard

Main Flow:

1. Admin logs in and goes to the feedback section
2. Admin clicks “Export” and chooses a format (CSV, JSON)
3. The system creates the file
4. Admin downloads it

Postconditions: Feedback data is successfully exported

Exceptions:

- If there’s no data, notify: “No feedback available.”

4.1.13 Use Case: User Registration

Use Case ID: UC-13

Use Case Name: Register New User

Primary Actor: User

Goal: To set up a new user account in the system

Preconditions:

- User is not logged in
- User has access to the registration page

Main Flow:

1. The user goes to the registration form.
2. The user fills in their username, email, and password.
3. The user clicks the “Register” button.
4. The system checks the inputs (like ensuring the email is unique and the password meets length requirements).
5. The system creates a new user record in the database.
6. The system shows a success message and may redirect the user to the login page.

Postconditions:

- A new user account is created and saved.
- The user can now log in using their credentials.

Exceptions:

- If the email is already registered, display: “This email is already registered.”
- If the password is too short, display: “Password must be at least 6 characters.”
- If there’s a server error, display: “Registration failed. Please try again later.”

4.1.14 Use Case: User Login

Use Case ID: UC-14

Use Case Name: User Login

Primary Actor: User

Goal: To log in and access the user dashboard

Preconditions:

- The user has an existing account
- The user is on the login page

Main Flow:

1. The user enters their email and password in the login form.
2. The user clicks the “Login” button.
3. The system checks the credentials against the database.
4. If the credentials are correct, the user is taken to their profile/dashboard.
5. The system starts a new session for the user.

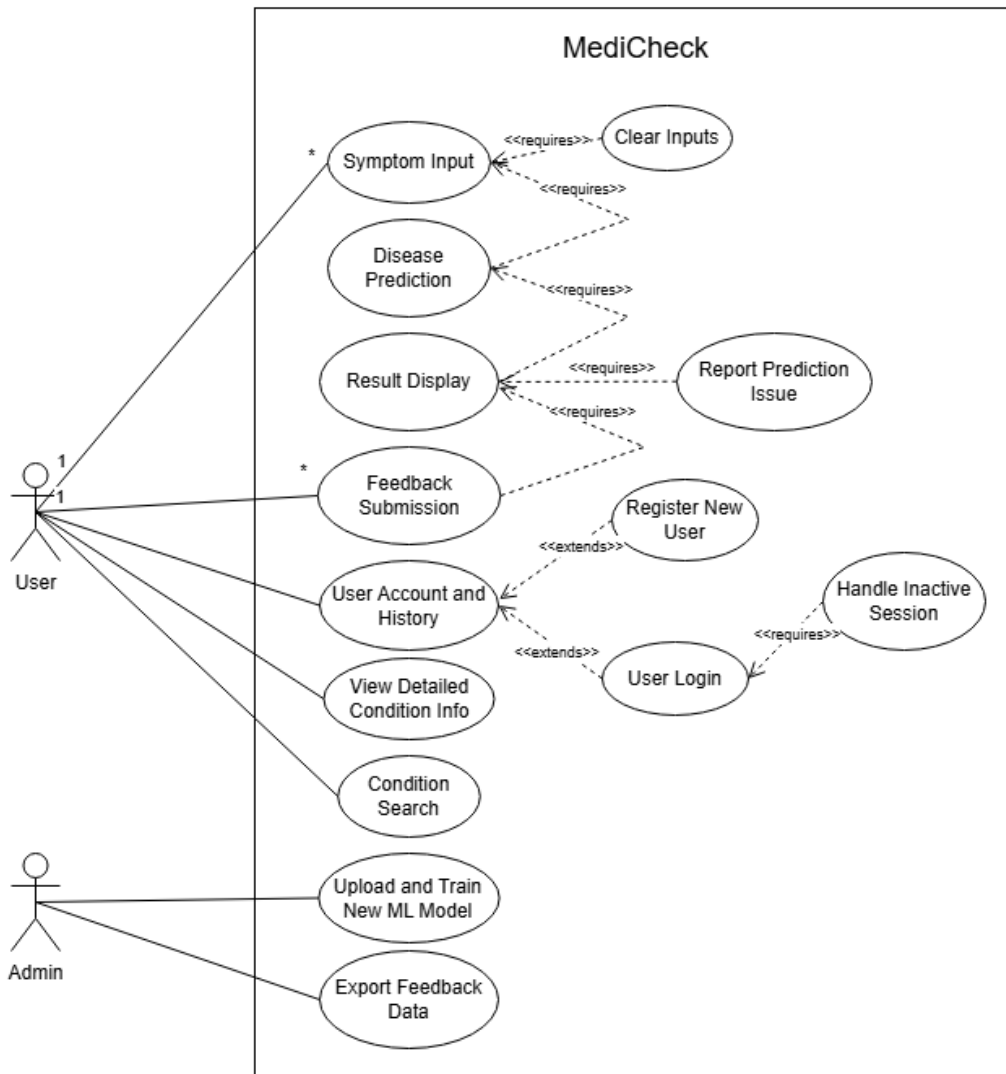
Postconditions:

- The user is authenticated and logged into the system.
- The user can now access personalized features (like prediction history).

Exceptions:

- If the email or password is incorrect, display: “Invalid email or password.”
- If the database or server is unreachable, display: “Login service unavailable. Please try again later.”

If the account isn’t found, suggest that the user registers.



4.2 Classes / Objects

4.2.1 User

This class is initiated with 3.2.1 Symptom Input Module, 3.2.2 Disease Prediction Module, 3.2.4 User Feedback Module and 3.2.5 User Profile Management.

4.2.1.1 Attributes

- **sessionID:** A temporary session identifier assigned to each user (randomly generated).
- **inputSymptoms:** The list of symptoms entered by the user during this session.
- **predictionResults:** The most recent prediction results provided by the system to the user.
- **feedbackGiven:** Boolean value indicating whether the user has submitted feedback during this session.
- **submissionDate:** Date when the symptoms were first submitted.

4.2.1.2 Functions

- **enterSymptoms(symptomList)** – Receives the list of symptoms entered by the user and stores it in **inputSymptoms**.
- **getPredictionResults()** – Returns the prediction results stored in **predictionResults**.
- **savePredictionResults(results)** – Saves the prediction results received from the system.
- **submitFeedback(rating, comment)** – Sends the user's feedback (rating and optional comment) and sets **feedbackGiven = True**.
- **resetSession()** – Resets the session by clearing all fields (symptoms, predictions, feedback) and initiate a new session.

4.2.2 MainPage

This class is the main page and organizes with all other classes. Its attributes and functions is now less than actual main page because it will be ready when all the classes are organized.

4.2.2.1 Attributes

- `sessionUser`: The current `User` object interacting with this page.

4.2.2.2 Functions

- `renderPage()` – Renders all the components of the main page to the screen.
- `initializeUserInput()` – Reads user input and stores it in the `User.inputSymptoms` attribute.
- `sendToPrediction()` – Sends the captured input to `SymptomProcessor` and `DiseasePredictor` components and retrieves prediction results.
- `displayResults(results)` – Displays the list of predicted diseases and formats them for presentation.
- `submitUserFeedback()` – Sends input to the `FeedbackManager` for processing.
- `displayError(message)` – Shows the given error message used for handling invalid inputs or system errors.

4.2.3 SymptomInputHandler

Implements 3.2.1 Symptom Input Module (handling user symptom entry, validation, and forwarding). This corresponds to the user input step in Use Case #1 (Symptom Submission) of the system.

4.2.3.1 Attributes

- `inputSymptoms`: List of symptoms entered by the user via the user interface.
- `maxSymptomsAllowed`: Maximum number of symptoms allowed per session (e.g. 10) to enforce input constraints.
- `errorMessage`: Message to display if input validation fails. (for example, “Please enter at least one symptom.”)

4.2.3.2 Functions

- **takeSymptoms()** – Gathers symptom inputs from the user interface. (free-text or selected from list)
- **checkSymptoms()** – Checks that the user has entered at least one symptom and does not exceed the allowed number. If validation fails, sets an appropriate error message with the attribute of `errorMessage`.
- **organizeSymptoms()** – Calls the `SymptomProcessor`(4.2.2) to normalize the raw input symptoms before prediction if user enters free-text.
- **confirmSymptoms()** – Forwards the cleaned symptom list to the `DiseasePredictor`(4.2.3) component for analysis and may display a confirmation that input was received.

4.2.4 SymptomProcessor

Supports 3.2.1.3 (Processing) of the Symptom Input Module by standardizing inputs and handling unknown terms. This is part of the symptom normalization step in the symptom submission use case.

4.2.4.1 Attributes

- **knownSymptoms**: A dataset(array) of known symptom terms (from the medical dataset) used to validate and map user input.
- **synonymMap**: A dictionary of common symptom synonyms/misspellings to standard terms for normalization.

4.2.4.2 Functions

- **standardizeSymptoms(symptoms)** – Takes the raw user symptom list and converts each entry to a standardized format (e.g., lowercasing, spelling correction, replacing synonyms with official terms).
- **validateSymptom(term)** – Checks if a given symptom term exists in the `knownSymptoms`. If not, it can invoke suggestions of a known term with `suggestCorrection(term)`.
- **suggestCorrection(term)** – Suggests the closest matching known symptom for an unrecognized input term.

4.2.5 DiseasePredictor

Implements 3.2.2 Disease Prediction Module, handling model loading, symptom vectorization, prediction computation, and result ranking. Error handling aligns with 3.2.2.6 (logging no-match scenarios)

4.2.5.1 Attributes

- **dpModel**: The pre-trained machine learning model loaded into memory for inference.
- **maxResults**: The maximum number of top predictions to return to the user.
- **predictionResults**: Data structure holding the latest prediction outcomes (e.g., list of disease predictions with confidence scores).

4.2.5.2 Functions

- **loadModel(filepath)** – Loads the trained ML model from storage at system startup or on demand. Ensures the model is ready to generate predictions.
- **predictDiseases(symptomList)** – Given a cleaned list of symptoms, transforms it into the model's input format, runs the prediction, and calculates confidence scores for each potential disease. The results are sorted and trimmed to the top N predictions by confidence.
- **rankResults(predictions)** – Sorts prediction outcomes in descending order of confidence and selects the top results.
- **handleNoMatch(symptomList)** – If the model returns no strong matches or cannot make a prediction, triggers a graceful handling: e.g., flags that no clear result was found and provides a fallback message. It may also log the failed prediction event for further analysis.

4.2.6 ResultPresenter

Makes connection with 3.2.3 Prediction Result Display by presenting the prediction outcomes to the user in a clear format, including disease details and disclaimers. It ensures only top results are shown and handling errors as 3.2.3.6 error/data handling.

4.2.6.1 Attributes

- **displayLimit:** Maximum number of predicted conditions to display to the user (e.g., top 3).
- **diseaseDescriptions:** A reference to a data source containing medical descriptions for known diseases.
- **disclaimerText:** Predefined disclaimer message to inform the user that the prediction is not a professional diagnosis.
- **predictions:** Container for the prediction results to be displayed (list of tuples: (disease name, confidence, description)).

4.2.6.2 Functions

- **formatResults(predictions)** – Prepares the raw prediction results for display, formats each predicted condition with its name, confidence (e.g., “Likely – 85%”), and a brief description. Ensures only the top results are included.
- **getDiseaseDescription(disease)** – Retrieves a short description for a given disease from the **diseaseDescriptions**.
- **displayResults()** – Renders the formatted predictions to the user interface as a user-friendly web page. This includes each condition’s name, confidence score, description, and the disclaimer text displayed. It may also provide a “What to do next” suggestion or a link to restart the symptom input flow.
- **showNotProcessedMessage()** – If the predictions data is missing or corrupted, displays a generic error message like “We couldn’t process the result properly. Please try again later.” instead of results, to handle any unexpected display issues.

4.2.7 FeedbackManager

Implements 3.2.4 User Feedback Module, which allows optional user feedback after seeing results. It covers validating feedback input and storing it, enforcing that feedback is optional/one-time, and providing confirmation or error messages as defined in 3.2.4.4 and 3.2.4.6. This corresponds to Use Case #2 (Submit Feedback) in the system.

4.2.7.1 Attributes

- **feedbackData:** Structure to hold the user's feedback input (rating value out of 10 and optional comment text) for a given output.
- **feedbackStorage:** Backend storage where feedback entries are saved (database table).
- **isFeedbackSubmitted:** Boolean value indicating whether the user has already submitted feedback in the current session.

4.2.7.2 Functions

- **collectFeedback(rating, comment)** – Interfaces with the user interface to collect the user's feedback input (feedbackData).
- **validateFeedback(rating, comment)** – Ensures the feedback form is filled out correctly, rating is given between 1 and 10. Does not require a comment, as that is optional.
- **saveFeedback()** – Stores the feedbackData to feedbackStorage for future analysis. This includes recording the feedback as described in the requirements (for potential model improvement use).
- **confirmSubmission()** – Returns a confirmation message to the user interface successful feedback submission ("Thanks for your feedback!").
- **feedbackError()** – In case of a failure during submission, notifies the user ("Could not send feedback. Try again later.").

4.2.8 ModelManager

Addresses the Model Update Capability of the system as described in the project overview. This class is aligned with the maintainability requirements that the ML model be updatable independently from the user interface. While not directly invoked in a user interface (model updates are performed by developers), it supports the system's ability to improve over time by integrating new datasets and retrained models.

4.2.8.1 Attributes

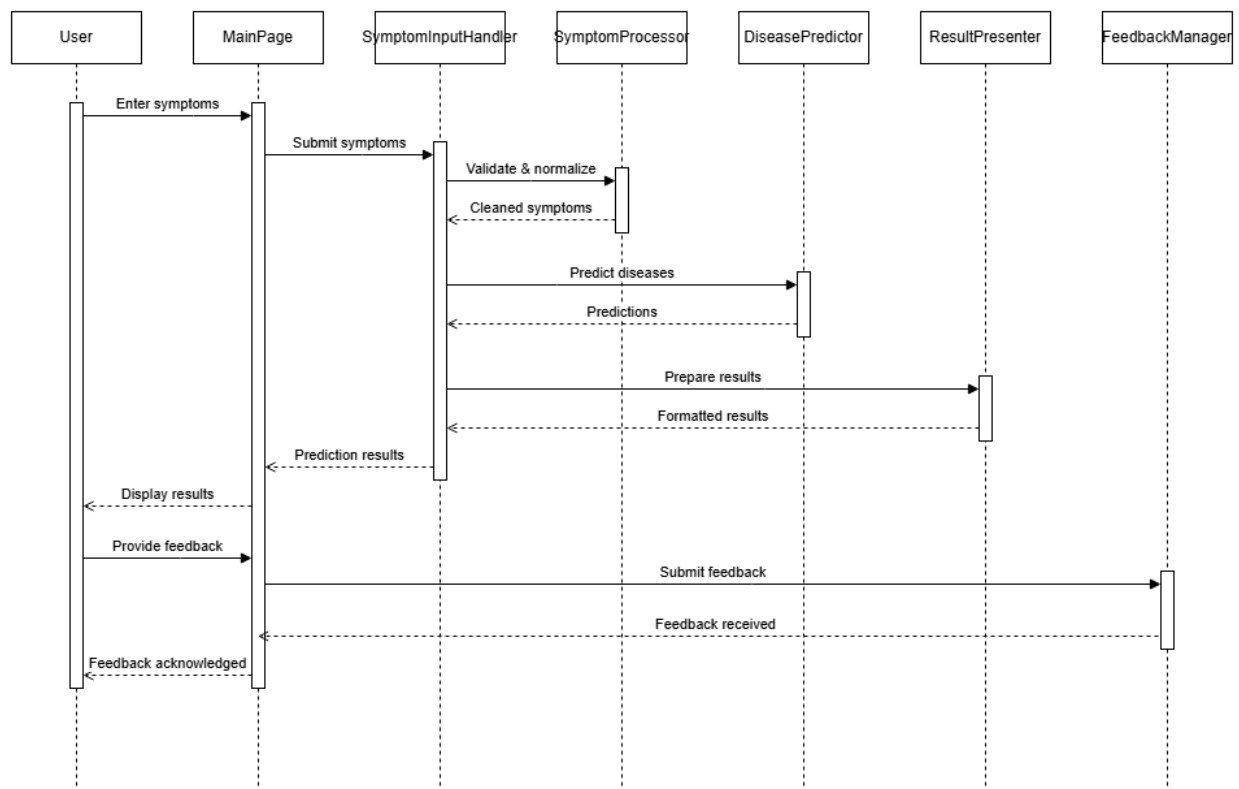
- **currentModelVersion**: Identifier for the currently deployed ML model (date of last update).
- **modelFilePath**: The file path to the stored ML model that the system is using for predictions.
- **trainingDataset**: Dataset used for training/retraining the model, if model updates are performed.
- **updateHistory**: Log of model updates performed, to keep track of changes over time.

4.2.8.2 Functions

- **loadModel()** – Loads the current ML model from the **modelFilePath** in memory by the system on startup. This ensures the latest model is used by the DiseasePredictor.
- **retrainModel(newData)** – Initiates a retraining process using new or additional dataset. Produces an updated model based on **newData** to improve accuracy. This is run by developers, not users.
- **updateModel(newModelFile)** – Replaces the current model with a new model (after retraining). It updates **currentModelVersion** and **modelFilePath** to point to the new model, allowing the system to integrate model updates without other components changing.
- **getModel()** – Provides the currently loaded model to other components like DiseasePredictor, abstracting the model retrieval so that the predictor need not know file paths.

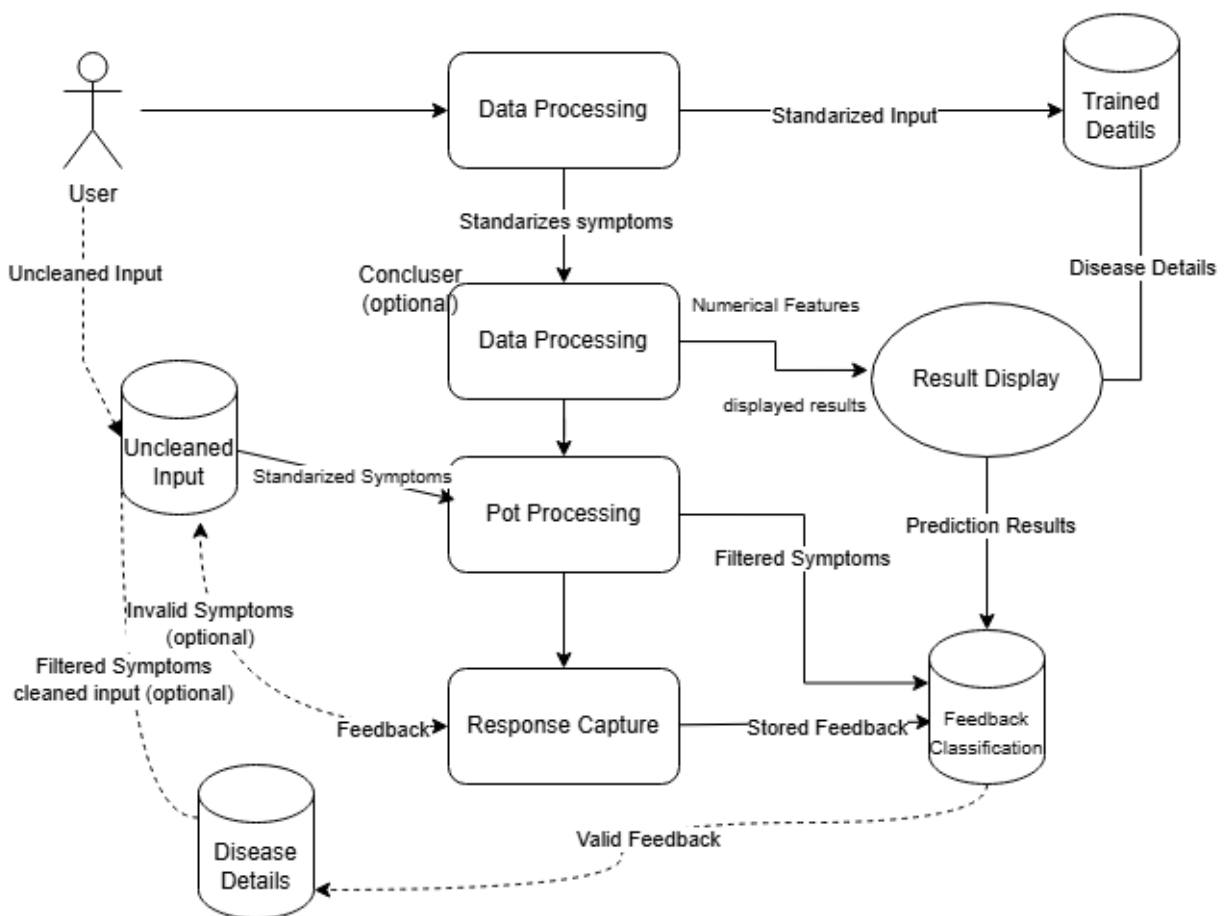
4.3 Sequence Diagrams

This sequence diagram illustrates the complete user interaction flow (without login, login is not essential to use the application) within the MediCheck application. The user begins by entering symptoms via the MainPage, which forwards the data to the SymptomInputHandler. The handler validates and normalizes the input using the SymptomProcessor, then sends the cleaned symptoms to the DiseasePredictor for analysis. The predicted diseases are ranked and passed to the ResultPresenter, which formats and displays the results back to the user. Optionally, the user can provide feedback, which is submitted to the FeedbackManager for storage and acknowledgment. The diagram represents both the main functional path and optional feedback, aligned with the system's core use cases.



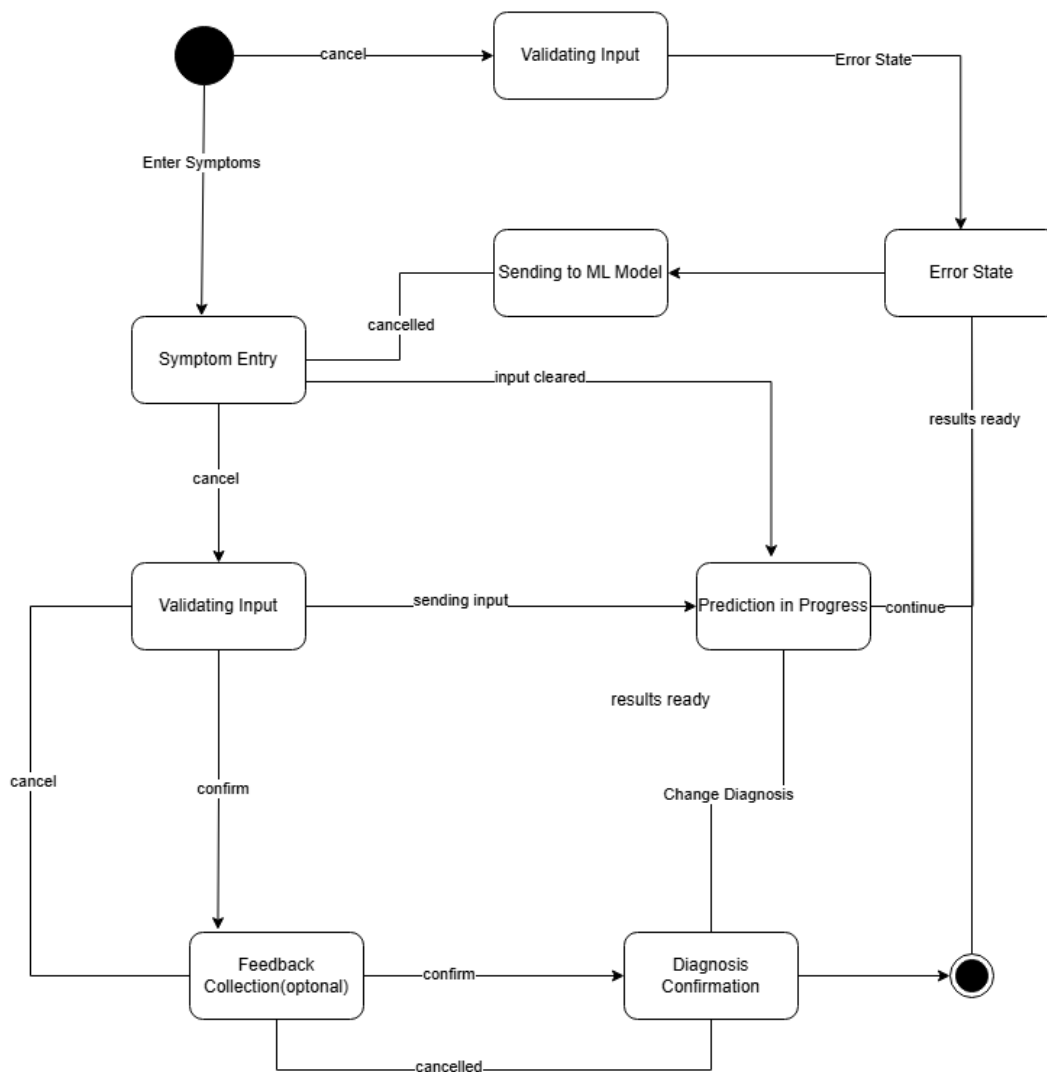
4.4 Data Flow Diagrams (DFD)

This data flow diagram represents the core data processing pipeline of the MediCheck system. It starts when the user inputs unrefined data (symptom), which is first handled by the Data Processing module to standardize the symptoms. Optionally, a conclusion step may provide further refinement. Once standardized, the symptoms move on to the Prediction Processing stage, where they're filtered and transformed into numerical features that the machine learning model uses to predict potential diseases. Finally, The Result Display module shows the user the predicted diseases, pulling in information from both the Trained Details and Disease Details. Additionally, if the user wants to give feedback, the Response Capture module collects that and stores it in the Feedback Classification system for future analysis or to help improve the model.



4.5 State-Transition Diagrams (STD)

This state diagram shows the user interaction flow of MediCheck system from symptom entry to final diagnosis confirmation. The process begins with the Symptom Entry state, followed by Validating Input. If the input is valid, it proceeds to Prediction in Progress and then goes to Diagnosis Confirmation once results are ready. At any point, the process can be canceled or may be entered an Error State if validation fails. After confirming the diagnosis, users are optionally presented with a Feedback Collection step. They can either submit feedback and finish or skip it and directly reach the final state. The diagram emphasizes flexibility, validation, and optional user feedback after diagnosis.



A. Appendices

A.1 Appendix 1 - Dataset Description

The dataset that powers MediCheck is called “sympredict_2022.csv,” and it comes from a public medical repository. It’s designed to train and evaluate the machine learning model that predicts potential diseases based on the symptoms users input.

This dataset includes 4,961 records and 133 columns, with each row representing a unique case that connects a set of symptoms to a diagnosis, which is found in the prognosis column. The first 132 columns detail various binary-encoded symptoms—like itching, joint pain, and vomiting—with values of 1 or 0 showing whether each symptom is present or absent in a given case. The last column, prognosis, reveals the diagnosed condition and serves as the target for the machine learning model’s predictions.

All the fields in this dataset are complete, meaning there are no missing values. This ensures a smooth training process and minimizes the need for any preprocessing. The data is consistently formatted, making it ideal for algorithms that work with tabular input.

This dataset is crucial for the system’s ability to learn the relationships between symptoms and diseases, and it’s strictly used on the backend for training and making predictions. Users can’t modify it during runtime. If there are any updates or enhancements to the dataset, the prediction model will need to be retrained under supervised conditions.

A.2 Appendix 2 - Project Meeting Notes

During the planning and early development phases of the MediCheck project, our team held several meetings to nail down the system’s purpose, scope, and key implementation details. Here’s a quick rundown of the decisions and discussions that took place:

Initial Concept Discussion (Week 1):

We decided to create a machine learning-based web application that predicts potential diseases based on symptoms users input. It was clear from the start that this app wouldn’t serve as a diagnostic tool but rather as a helpful informational resource. We officially named the project MediCheck.

Dataset and Model Planning (Week 2):

We opted to use open-source medical datasets to train our machine learning model, specifically choosing the sympredict_2022.csv dataset for this purpose. We agreed that symptom entries would be binary-encoded, and we’d employ supervised learning techniques like decision trees or random forest classifiers for training the model.

Development Strategy (Week 2):

We decided to go with the Incremental Development Model, which allows us to develop and test core functionalities in stages. Our early iterations will focus on basic symptom input and prediction output, with plans to incorporate user feedback and profile features later on.

Technology Stack Selection (Week 3):

After reviewing various frameworks, we settled on using Python (Flask) for the backend, along with HTML, CSS, and JavaScript for the frontend, and MongoDB as our NoSQL database.

Requirement Breakdown and UI Planning (Week 4):

We assigned each team member to specific modules: symptom input interface, prediction logic, result display, feedback form, and a basic user account system. We also planned out and discussed initial UI wireframes.

Ethical and Scope Clarifications (Week 4):

We held discussions to clarify ethical boundaries, including a disclaimer that our system doesn't replace professional medical advice. Importantly, we agreed that the system wouldn't use personal health data for any purpose, ensuring that users remain anonymous.

These meetings have shaped the current version of the SRS and are expected to continue throughout development for coordination, issue resolution, and design validation. Meeting notes are stored collaboratively and referenced during requirement updates.

A.3 Appendix 3 - Glossary (Extended with Medical Symptom Definitions)

Itching: That annoying tingling feeling on your skin that makes you want to scratch. It often pops up during allergic reactions, eczema flare-ups, or infections.

Skin rash: Noticeable changes in your skin's texture or color, usually caused by irritation, inflammation, or infections.

Nodal skin eruptions: Bumpy, raised spots on the skin that can show up with various skin issues or systemic conditions.

Continuous sneezing: Sneezing happens frequently and isn't just due to temporary irritants; it's often linked to allergies or upper respiratory infections.

Shivering: That involuntary shaking of your body, usually in response to feeling cold or having a fever.

Chills: A cold sensation that often comes with shivering, typically due to an infection or a sudden change in temperature.

Joint pain: Discomfort or soreness in one or more joints, which could be due to arthritis or viral infections.

Stomach pain: Discomfort in your abdominal area, which can stem from a variety of causes, from indigestion to more serious infections.

Acidity: When there's too much acid in your stomach, it leads to discomfort, heartburn, or acid reflux.

Ulcers on tongue: Painful sores that can appear on your tongue, often caused by viral infections, nutritional deficiencies, or irritation.

Muscle waste: The loss of muscle mass, frequently associated with chronic diseases, malnutrition, or being immobile.

Vomiting: The involuntary act of expelling stomach contents through your mouth, often a sign of infection or gastrointestinal issues.

Burning micturition: That burning feeling when you urinate, which commonly indicates a urinary tract infection.

Spotting urination: The presence of small amounts of blood or unusual discharge during urination.

Fatigue: This is that all-too-familiar feeling of tiredness or a lack of energy, something many of us experience during both acute and chronic illnesses.

Weight gain: This refers to an increase in body mass, which can happen for various reasons, including metabolic, hormonal, or dietary factors.

Anxiety: It's that nagging feeling of worry or unease, often accompanied by physical symptoms like restlessness or sweating.

Cold hands and feet: Usually linked to poor circulation, stress, or conditions such as hypothyroidism.

Mood swings: These are those rapid and intense shifts in your emotional state, often tied to hormonal or neurological issues.

Weight loss: This is an unintended drop in body weight, which can be a sign of an underlying health problem or malabsorption.

Restlessness: It's that feeling of being unable to stay still, often connected to anxiety, stress, or neurological conditions.

Lethargy: A state where you feel physically and mentally sluggish, often associated with fatigue or illness.

Patches in the throat: These are visible white or red spots in the throat, which could indicate an infection or inflammation.

Irregular sugar levels: This refers to unstable blood glucose levels, commonly seen in diabetes or endocrine disorders.

Cough: A reflex action that helps clear the airways of mucus, irritants, or infections.

High fever: An elevated body temperature, usually a response to infection or inflammation.

Sunken eyes: A sign of dehydration or fatigue, where the eyes seem to sink back into the skull.

Breathlessness: Difficulty breathing, often linked to respiratory or heart conditions.

Sweating: Excessive perspiration that might be caused by fever, anxiety, or hormonal imbalances.

Dehydration: This occurs when your body lacks enough fluids, leading to symptoms like dry mouth, fatigue, or confusion.

Indigestion: Discomfort in the upper abdomen, often related to what you've eaten or acidity.

Headache: Pain in the head or upper neck, which can range from mild to severe.

Yellowish skin: A yellow tint to the skin, typically indicating jaundice or liver issues.

Dark urine: Discoloration of urine, often a sign of dehydration or liver-related problems.

Nausea: That uneasy feeling when you think you might throw up, often linked to infections or stomach issues.

Loss of appetite: When you just don't feel like eating, which can happen because of illness, stress, or medication.

Pain behind the eyes: That annoying discomfort you feel around or behind your eyes, possibly from sinusitis, migraines, or infections.

Back pain: Discomfort in your lower or upper back, usually tied to posture, strain, or kidney problems.

Constipation: When it's tough to pass stools, often due to a poor diet, not drinking enough water, or digestive disorders.

Abdominal pain: General discomfort in your stomach area, which can be related to various digestive or inflammatory issues.

Diarrhea: Frequent, loose, or watery stools, typically caused by infections or food intolerances.

Mild fever: A slight increase in body temperature, often signaling a mild infection or inflammation.

Yellow urine: Bright yellow urine, which might indicate hydration levels or bilirubin buildup.

Yellowing of eyes: A sign of jaundice, often due to liver or blood-related conditions.

Acute liver failure: A rapid decline in liver function that needs immediate medical attention.

Fluid overload: Too much fluid in the body, potentially caused by kidney or heart failure.

Swelling of stomach: A bloated or distended abdomen, possibly from gas, bloating, or internal fluid.

Swelled lymph nodes: Enlarged lymph glands, usually a response to infection or immune activity.

Malaise: A general sense of discomfort or unease, often without a clear reason.

Blurred and distorted vision: When your eyesight is unclear or impaired, which can be temporary or linked to eye or neurological issues.

Phlegm: Mucus is produced in your respiratory tract, especially during infections.

Throat irritation: A scratchy or sore throat feeling, often due to infection or dryness.

Redness of eyes: Bloodshot or inflamed eyes, usually from irritation, allergies, or infections.

Sinus pressure: Pain or pressure around your sinuses, often experienced during colds or sinusitis.

Runny nose: A constant flow of nasal discharge, commonly caused by colds, allergies, or infections.

Congestion: This refers to a blockage in your nasal passages, usually caused by inflammation or a buildup of mucus.

Chest pain: This is discomfort in the chest area, which can stem from issues with the heart, lungs, or even the muscles and bones.

Weakness in limbs: You might feel a lack of strength or numbness in your arms or legs, which could be due to neurological or muscular problems.

Fast heart rate: An unusually rapid heartbeat can happen, often triggered by stress, fever, or heart-related conditions.

Pain during bowel movements: This is the discomfort you might feel when passing stool, often linked to hemorrhoids or infections.

Pain in anal region: You may experience localized pain around the anus, possibly caused by fissures or infections.

Bloody stool: If you notice blood in your feces, it could indicate internal bleeding or hemorrhoids.

Irritation in anus: A burning or itching sensation near the anus is often due to irritation or infection.

Neck pain: Discomfort in the neck area can arise from strain, poor posture, or infections.

Dizziness: This sensation of spinning or feeling light-headed is often associated with balance disorders or changes in blood pressure.

Cramps: Sudden, involuntary muscle contractions can cause pain, often related to physical activity or an electrolyte imbalance.

Bruising easily: If you find yourself developing bruises from minor bumps, it could be linked to blood disorders.

Obesity: This medical condition is characterized by having excessive body fat, which can lead to various health complications.

Swollen legs: When your legs become enlarged due to fluid buildup, injury, or circulatory issues, it can be concerning.

Swollen blood vessels: Enlarged veins or arteries may occur, potentially due to inflammation or strain on the circulatory system.

Puffy face and eyes: Swelling around your face or eyes is commonly caused by allergies, infections, or fluid retention.

Enlarged thyroid: A noticeable swelling in the neck area can be associated with thyroid disorders.

Brittle nails: If your nails crack or break easily, it might be linked to nutritional deficiencies or hormonal issues.

Swollen extremities: General swelling in your arms or legs can often result from fluid retention or vascular problems.

Excessive hunger: Feeling hungrier than usual, which can sometimes be linked to metabolic or hormonal issues.

Drying and tingling lips: Experiencing dry or numb lips, often due to dehydration or irritation of the nerves.

Slurred speech: Having trouble speaking clearly, frequently associated with neurological conditions or strokes.

Knee pain: Discomfort in the knee joint, usually resulting from strain, injury, or arthritis.

Hip joint pain: Pain around the hip area, often related to joint or muscle problems.

Muscle weakness: A decrease in muscle strength, which could stem from nerve or muscle disorders.

Stiff neck: Limited movement and pain in the neck, often caused by strain or infection.

Swelling joints: Enlargement of the joints due to fluid buildup or inflammation, commonly seen in arthritis.

Movement stiffness: Feeling restricted or uncomfortable when moving limbs or joints, typical in musculoskeletal issues.

Spinning movements: A sensation of spinning, characteristic of vertigo or balance disorders.

Loss of balance: Difficulty maintaining posture or orientation, often due to neurological factors.

Unsteadiness: A feeling of instability while standing or walking, linked to coordination problems.

Weakness of one body side: Weakness on one side of the body, which may indicate a stroke or nerve damage.

Loss of smell: The inability to detect odors, often resulting from infections or nerve damage.

Bladder discomfort: Experiencing pain or pressure in the bladder area, frequently due to infection.

Foul smell of urine: An unusual odor in urine, which can be caused by dehydration or a urinary tract infection.

Continuous feel of urine: A frequent urge to urinate, commonly associated with infections or bladder irritation.

Passage of gases: The release of gas from the digestive system, which is normal but can be excessive in gastrointestinal issues.

Internal itching: An itching sensation inside the body that can be hard to pinpoint; it may be psychosomatic or systemic.

Toxic look (typhus): A term used to describe severely ill patients with high fevers, often seen in cases of typhoid.

Depression: A mental health condition characterized by ongoing sadness and a lack of interest, often accompanied by physical symptoms.

Irritability: A heightened sense of annoyance or frustration, which could stem from psychological factors or hormonal changes.

Muscle pain: Discomfort or soreness in the muscles, often resulting from strain, infections, or underlying health issues.

Altered sensorium: A decrease in awareness or alertness, typically linked to confusion or neurological problems.

Red spots over body: A rash-like appearance of red dots on the skin, often associated with infections or allergic reactions.

Belly pain: General discomfort in the abdominal area, often used interchangeably with stomach pain.

Abnormal menstruation: Irregularities in menstrual cycles, which may be due to hormonal or reproductive concerns.

Dyschromic patches: Variations in skin pigmentation, potentially caused by fungal infections or autoimmune disorders.

Watering from eyes: An increase in tear production, usually due to irritation, allergies, or infections.

Increased appetite: A rise in hunger levels, which could be related to hormonal imbalances.

Polyuria: Frequent urination, commonly seen in conditions like diabetes or kidney issues.

Family history: A genetic tendency towards certain diseases based on the medical history of family members.

Mucoid sputum: Thick mucus produced by the lungs, typically seen in respiratory illnesses.

Rusty sputum: Discolored mucus that may indicate a bacterial lung infection.

Lack of concentration: Challenges in focusing or maintaining attention, often linked to psychological or neurological conditions.

Visual disturbances: Any unusual visual experiences, such as blurriness or flashes of light.

Receiving blood transfusion: A history of receiving donor blood, which may pose risks for certain health conditions.

Receiving unsterile injections: Exposure to contaminated medical equipment, increasing the risk of infections.

Coma: A profound state of unconsciousness that requires immediate medical intervention.

Stomach bleeding: The presence of blood in the digestive tract, often signaling ulcers or serious gastrointestinal issues.

Distention of abdomen: Swelling or enlargement of the abdominal area, possibly due to gas, fluid, or organ enlargement.

History of alcohol consumption: A lifestyle factor that can impact various health conditions.

CREDITS

Şükrü Can Mayda – Introduction (1.4, 1.5), General Description (2.3, 2.4, 2.5), Specific Requirements (3.2, 3.3), UML Diagrams (4.2, 4.3), Appendices (A2, A3)

Nurbetül Çakır – Introduction (1.1, 1.2, 1.3), General Description (2.1, 2.2), Specific Requirements (3.1, 3.4, 3.5), UML Diagrams (4.1, 4.5), Appendices (A1, A2, A3)

Turgut Köroğlu – Specific Requirements (3.6, 3.7), UML Diagrams (4.4, 4.5), Appendices (A1, A2)

As a note, in every step we helped each other, so the credits section includes especially hard-working members on that specific topic.

Acknowledgment: Sections of this document are based upon the IEEE Guide to Software Requirements Specification (ANSI/IEEE Std. 830-1984). The SRS templates of Dr. Orest Pilskalns (WSU, Vancouver) and Jack Hagemester (WSU, Pullman) have also be used as guides in developing this template.