“ML-Driven Early Detection for Optimal Health – Empowering You with Accurate Predictive Health Analytics”

# SUMMER INTERNSHIP SOPHOMORES PROJECT SPJ 2001­­­­­­

**AP20232410000130**

## Summer Id: 20240007

**First Review**

# UNDER THE GUIDANCE OF

Prof. Anil Vitthalrao Turukmane

**DEPARTMENT OF SCOPE (Speci. Networking And Security)**

# SUBMITTED BY

Madhu Alapaka - 21MIS7022.

Kothamasu Kaarthikeya - 21MIS7039.

Pulluru Nikhilesh - 21MIS7087.

**DEPARTMENT OF SCOPE(Integ M.TechSpec.in Software Engineering)**



**SYSTEM REQUIREMENT SPECIFICATIONS (“S.R.S”)**

“ML-Driven Early Detection for Optimal Health – Empowering You with Accurate Predictive Health Analytics”

**1. Purpose of the System:**

The purpose of the Multiple Disease Prediction System using Machine Learning is to provide early detection and risk assessment for multiple diseases, such as Diabetes, Heart Disease, and Parkinson's Disease. This system leverages advanced machine learning algorithms to analyze patient data and predict the likelihood of these conditions, enabling timely medical intervention and personalized healthcare management. The deployment interface is built on Streamlit, offering a user-friendly web application where users can input health data and receive instant predictions. This proactive approach aims to improve health outcomes and reduce the burden on healthcare systems by facilitating early diagnosis and treatment.

**2. Problems in the Existing System :**

* **Lack of Early Detection:** Current systems may not provide timely detection of multiple diseases, leading to delayed treatment, which can result in worsening conditions and reduced treatment effectiveness.
* **Limited Access to Diagnostics:** Not all patients have access to comprehensive diagnostic facilities due to geographical or financial constraints, limiting their ability to get timely and accurate diagnoses.
* **Fragmented Data:** Patient health data is often scattered across various sources, making it difficult to aggregate and analyze comprehensively for predictive analysis. This fragmentation can lead to incomplete or inaccurate predictions.
* **Complexity of Use:** Existing predictive tools can be complex and not user-friendly, limiting their accessibility to healthcare providers and patients. This complexity can result in the underutilization of these tools.
* **Limited Accuracy**: Traditional methods may not leverage advanced machine learning techniques, resulting in lower prediction accuracy. This can lead to misdiagnoses or missed diagnoses, affecting patient care.

**3. Solution to These Problems:**

* + **Early Detection:** The system provides early disease detection, allowing for timely medical interventions and better health outcomes. This is achieved through the use of advanced machine learning algorithms that analyze patient data to identify early signs of disease.
  + **Accessibility:** An online interface (Streamlit) that can be accessed from anywhere allows users to input their data and get predictions. This increases the reach of diagnostic tools, making them available to a wider audience.
  + **Integrated System:** The proposed system integrates data from various sources, providing a comprehensive dataset for analysis. This integration improves the quality and completeness of the data used for predictions.
  + **User-Friendly Interface:** Deploys a web application using Streamlit for easy data input and result visualization. The intuitive interface ensures that both healthcare providers and patients can use the system with minimal training.
  + **Machine Learning Algorithms:** Utilizes advanced machine learning algorithms to improve prediction accuracy. These algorithms are trained on large datasets to learn patterns and correlations that indicate the presence of disease.

**4. Scope of the Project:**

* + **Disease Coverage:** Focuses on predicting Diabetes, Heart Disease, and Parkinson's Disease, with potential expansion to other diseases. The system is designed to be scalable and adaptable, allowing for the addition of new disease models in the future.
  + **User Base:** Designed for individuals and healthcare providers. The system aims to serve a diverse user base, including patients seeking self-assessment and healthcare professionals looking for diagnostic support.
  + **Geographical Reach:** Global, as the web application can be accessed from any location with internet connectivity. This broad reach ensures that the system can benefit users worldwide, regardless of their location.
  + **Scalability:** Capable of integrating additional diseases and scaling the infrastructure as needed. The system's architecture is designed to handle increased user load and data volume, ensuring consistent performance even as demand grows.

**5. Functional Components of the Project**

* **Data Ingestion and Preprocessing Module**

1. Data Acquisition: Collects raw data from various sources such as medical datasets (e.g., UCI Machine Learning Repository, CDC, WHO). The data includes patient demographics, medical history, lab results, and other relevant health indicators.

2. Data Refinement: Cleans and preprocesses data to ensure quality. This includes handling missing values, normalizing data, and removing outliers to prepare the data for analysis.

3. Feature Creation: Develop new features to enhance model performance. This involves selecting relevant variables and creating new features through techniques like feature extraction and engineering.

* **Machine Learning Model Building Module**

1. Model Selection: Chooses appropriate machine learning algorithms based on the nature of the disease and the data characteristics. Examples include Logistic Regression, Support Vector Machines (SVM), Random Forest, and Deep Learning models.

2. Model Training: Trains the chosen model on a portion of the data. This involves splitting the data into training and testing sets and using the training data to build the model.

* **Model Validation Module**

1. Validation Engine: Conducts thorough validation of predictive models. This includes techniques like cross-validation to ensure the model's generalizability and robustness.

* **User Interface with Streamlit Module**

1. Interactive Dashboard: Provides a user-friendly interface for data input and result visualization. Users can easily navigate through the dashboard to enter their health data and view predictions.

2. User Input Interface: Facilitates user data entry and parameter adjustments. The interface is designed to be intuitive and accessible, minimizing the effort required from users.

3. Prediction Display: Shows prediction outcomes and risk assessments interactively. Users receive clear and concise information about their health risks, along with recommendations for further action.

* **Reporting Module**

1. Report Generation: Generates comprehensive reports on model performance metrics. These reports can be used by healthcare providers to understand the model's accuracy and reliability.

* **Disease-Specific Predictive Models Module**

1. Heart/Cardio Disease Prediction: Utilizes algorithms like Logistic Regression, Decision Trees, Random Forest, and SVM. Key factors include cholesterol levels, blood pressure, age, and smoking status.

2. Diabetes Prediction: Uses models such as Logistic Regression, K-Nearest Neighbors, and Random Forest. Important predictors include BMI, glucose levels, insulin levels, and age.

3. Parkinson's Disease Prediction: Implements techniques like SVM, Random Forest, and Neural Networks. Critical features include motor symptoms, voice patterns, and other clinical assessments.

* **Deployment Module**

1. Integration: Integrates the final models into a user-friendly web application using Streamlit. The integration ensures seamless interaction between the model and the user interface.

2. Interactive Platform: Provides an interactive platform for users to input their health data and receive instant predictions. The platform is designed to be responsive and accessible, ensuring a smooth user experience.

**6. Study of the System:**

1. **Functional Requirements:**

* **Data Sources**

1. Data Selection: Analyze and select datasets that provide diverse and comprehensive medical data. The datasets should cover a wide range of patient demographics and health conditions to ensure the model's applicability.

2. Data Quality: Ensure datasets have high-quality, labeled data for training machine learning models. Data quality is crucial for building accurate and reliable predictive models.

* **Machine Learning Algorithms**

1. Algorithm Evaluation: Evaluate different algorithms for their suitability to predict each specific disease. This involves comparing the performance of various models to identify the best approach.

2. Model Comparison: Compare model performances to select the best approach for each disease. Metrics such as accuracy, precision, and recall are used to assess the models.

* **Web Framework**

1. Streamlit Assessment: Assess Streamlit for its capabilities in developing interactive and user-friendly web applications. Streamlit's simplicity and flexibility make it a suitable choice for deploying machine learning models.

2. Framework Support: Ensure the framework supports necessary integrations and provides a smooth user experience. The framework should facilitate easy deployment and maintenance of the application.

* **User Interaction**

1. Interface Design: Design an intuitive user interface that simplifies data input and result interpretation. The interface should be user-centric, focusing on ease of use and accessibility.

2. Usability Testing: Conduct usability testing to refine the user experience. Feedback from users helps identify areas for improvement and ensures the application meets user needs.

1. **Non-Functional Requirements**:

1. Performance: The application should respond to user inputs and generate predictions within an acceptable time frame.

2. Usability: The Streamlit interface should be user-friendly and easy to navigate.

3. Scalability: Although there is no database, the application should handle different sizes of input data efficiently.

4. Reliability: The application should consistently produce accurate predictions and visualizations without errors.

5. Maintainability: The code should be well-documented and modular to allow for easy updates and maintenance.

6. Portability: The application should be easily deployable on different systems with minimal configuration.

**Input / Output and Major Functions of the Multiple Disease Prediction System:**

#### **Main Inputs:**

1. **Patient Data**
   * Demographic information (e.g., age, gender)
   * Clinical data (e.g., lab results, medical history)
   * Lifestyle factors (e.g., smoking status, physical activity)
2. **Machine Learning Models**
   * Trained models for disease prediction (e.g., Diabetes, Heart Disease, Parkinson's Disease)
   * Models integrated with algorithms and parameters for inference

#### **Main Outputs:**

1. **Prediction Results**
   * Likelihood or risk assessment of diseases (e.g., probability scores, risk categories)
   * Diagnostic outcomes based on input data

**7. Performance Requirements**

**1. Accuracy**

Goal: Achieve a high predictive accuracy (at least 85%) for each disease model.

Measure: Calculate and monitor the accuracy metric during model evaluation and after deployment to ensure consistent performance.

**2. Response Time**

Goal: Ensure the system provides predictions within 5 seconds of data input.

Measure: Implement efficient data processing and model inference techniques to minimize latency.

**3. Scalability**

Goal: Design the platform to handle multiple concurrent users without performance degradation.

Measure: Use scalable cloud infrastructure and load balancing to manage increased user load and data volume effectively.

**4. Reliability**

Goal: Maintain consistent performance with minimal downtime.

Measure: Ensure high availability through robust infrastructure, regular maintenance, and monitoring. Target uptime should be at least 99.9%.

**5. Robustness**

Goal: Ensure the system can handle a wide range of data inputs and maintain performance.

Measure: Implement extensive testing, including edge cases and stress tests, to validate system robustness.

**6. User Satisfaction**

Goal: Achieve a high level of user satisfaction with the system's performance and usability.

Measure: Collect and analyze user feedback regularly to identify and address any performance-related issues. Aim for a user satisfaction rate of at least 90%.

**8. Feasibility Report:**

* **Technical Feasibility**

1. Availability of Tools: The system is technically feasible given the availability of machine learning frameworks (e.g., TensorFlow, Keras), data manipulation libraries (e.g., NumPy, Pandas), and deployment tools (e.g., Streamlit). These tools provide the necessary infrastructure for developing and deploying the system.

2. Computational Resources: The necessary computational resources, including high-performance laptops and cloud platforms, are accessible. These resources ensure the system can handle large datasets and complex computations.

* **Economic Feasibility**

1. Cost Management: The project is economically feasible as the costs involved in development, deployment, and maintenance are manageable within the allocated budget. Open-source tools and frameworks help reduce costs.

2. Healthcare Savings: Potential savings in healthcare costs due to early disease detection and management can offset the initial investment. Early intervention can reduce the need for expensive treatments and improve patient outcomes.

* **Operational Feasibility**

1. Workflow Integration: The system is operationally feasible as it aligns with the workflows of healthcare providers and patients. The system is designed to integrate smoothly into existing healthcare processes.

2. User-Friendly Interface: The user-friendly interface ensures ease of use, making it accessible to both patients and healthcare providers. Minimal training is required to use the system effectively.

* **Schedule Feasibility**

1. Realistic Timeline: Develop a realistic timeline for data collection, model training, system testing, and deployment. A well-planned schedule ensures that the project stays on track and meets deadlines.

2. Resource Allocation: Allocate sufficient resources to each phase to ensure timely completion. Adequate staffing and budget allocation are essential for the project's success.

**UNIFIED MODELING LANGUAGE DIAGRAMS (“U.M.L”)**

### **Unified Modeling Language (UML) Diagrams Overview**

* **Purpose**: UML provides a standardized notation to express software analysis and design models, governed by syntactic, semantic, and pragmatic rules. It enables software engineers to visualize, specify, construct, and document the artifacts of a software system.

### **Key Components of UML**

#### 1. **User Model View**

* **Purpose**: Represents the system from the users' perspective.
* **Diagrams**:
  + **Use Case Diagram**: Illustrates the interactions between users (actors) and the system, highlighting the functionalities (use cases) the system must support.
  + **Actors**: External entities such as users or other systems that interact with the system.
  + **Use Cases**: Specific functionalities or services provided by the system.

#### 2. **Structural Model View**

* **Purpose**: Depicts the static structure of the system, including its data and functional components.
* **Diagrams**:
  + **Class Diagram**: Shows the system's classes, attributes, operations, and the relationships between objects. It is essential for modeling the static design view of the system.
  + **Object Diagram**: Provides a snapshot of the system at a particular point in time, showing instances of classes and their relationships.

#### 3. **Behavioral Model View**

* **Purpose**: Represents the dynamic behavior of the system, showing how it interacts over time.
* **Diagrams**:
  + **Sequence Diagram**: Illustrates how objects interact in a particular sequence, focusing on the order of messages exchanged.
  + **Activity Diagram**: Captures the workflow of activities and actions within the system, similar to a flowchart.
  + **State Diagram**: Describes the states an object goes through during its lifecycle and the transitions between these states.

#### 4. **Implementation Model View**

* **Purpose**: Details how the system's structural and behavioral elements are to be realized in code.
* **Diagrams**:
  + **Component Diagram**: Depicts the organization and dependencies among software components, such as libraries, packages, and files.
  + **Deployment Diagram**: Shows the physical deployment of artifacts (software components) on nodes (hardware), including the relationships between hardware and software.

#### **5. Environmental Model View**

* **Purpose**: Illustrates the external environment where the system will operate, including physical and operational contexts.
* **Diagrams**:
  + **Deployment Diagram**: Also used here to show the runtime configuration in a specific environment.

### **DOMAINS OF UML:**

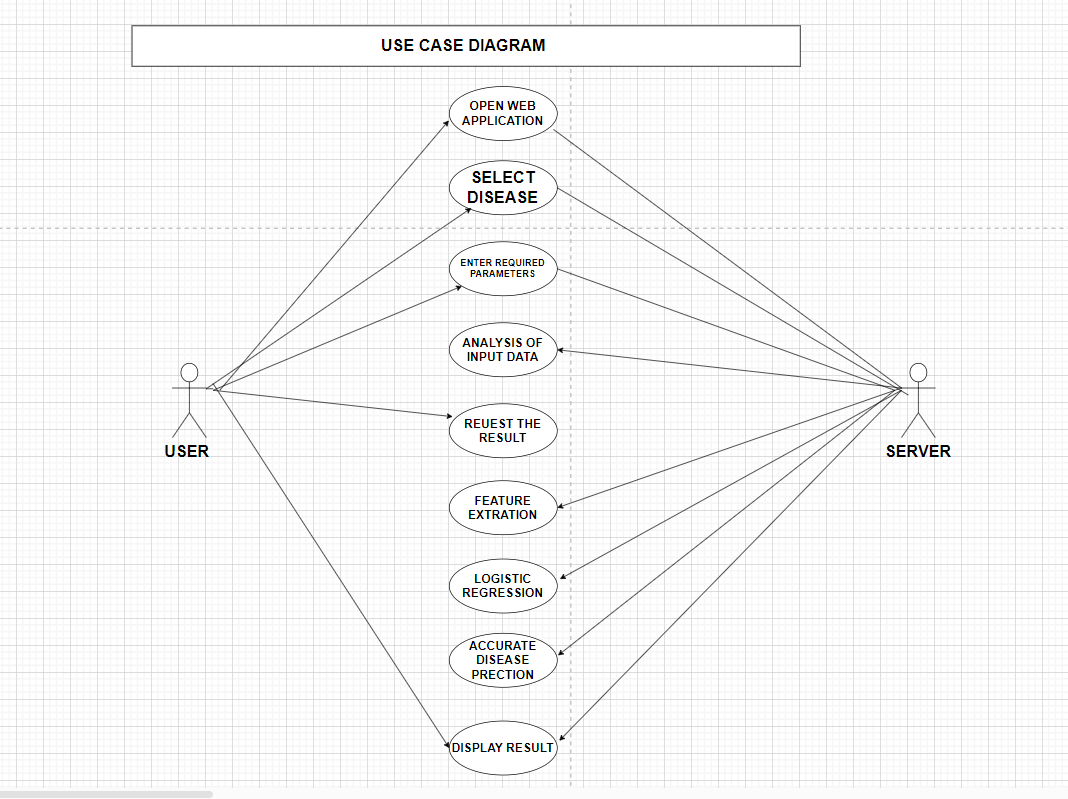
1. **UML Analysis Modeling**
   * Focuses on understanding and defining what the system must do from a user's perspective and its static structure.
   * Primarily uses the User Model and Structural Model views to capture requirements and define the static architecture of the system.
2. **UML Design Modeling**
   * Focuses on how the system will be built and how it will behave dynamically.
   * Uses Behavioral Model, Implementation Model, and Environmental Model views to define the dynamic aspects, implementation details, and environmental interactions.

### **Benefits of Using UML:**

* **Standardization**: Provides a common language for stakeholders, developers, and analysts to communicate.
* **Visualization**: Helps in visualizing the design of the system before implementation, making it easier to understand complex systems.
* **Documentation**: Offers a comprehensive documentation tool that can be referred to throughout the development lifecycle.
* **Analysis and Design**: Facilitates thorough analysis and design of the system, ensuring that all aspects are considered and properly addressed.

**“U.M.L” DIAGRAMS :**

* **USE CASE DIAGRAM :**
* Use case diagrams are visual representations of use cases. They show the actors, the system, and the use cases themselves, along with the relationships between them. These diagrams provide a high-level overview of the system's functionalities from the user's perspective.

****

* use case diagram depicting the interactions between a user and a web application for disease identification through an open web application. Here's a breakdown of the components and their relationships:

**Components:**

* **Actor:** Represented by a stick figure labeled "User," this signifies the person interacting with the system.
* **System:** A rectangle labeled "Open Web Application" represents the software system, the web application for disease identification.
* **Use Cases:** Ellipses depict the functionalities provided by the system. Here, there are three use cases:
  + "Select Disease" allows the user to choose a specific disease from the application.
  + "Enter Required Parameters" signifies the user inputting relevant information needed by the system to identify the disease. This might include symptoms, medical history, or other details.
  + "Request Result" represents the user's action to initiate the analysis and receive results.

**Relationships:**

* Solid lines connect the user to each use case, indicating that the user can perform these actions or functionalities within the web application.
* **Overall Explanation:**
* This use case diagram provides a high-level overview of the web application's functionalities from the user's perspective. It shows that users can select a disease, enter required parameters, and request results, presumably to get the application to identify the disease based on the information provided.

**Key Points:**

* The use case diagram doesn't show the internal workings of the web application or how it identifies diseases.
* It focuses on user interactions and the functionalities available through the web application.
* **CLASS DIAGRAM :**
* A class diagram, in the context of software engineering, is a visual representation of the classes in a system and the relationships between them. It's a core concept in object-oriented design (OOD) and is used for various purposes throughout the software development lifecycle.

Here's a breakdown of the key aspects of class diagrams:

* **Main Components:**
* **Classes:** These are blueprints that define the properties (attributes) and functionalities (methods) of objects. In the class diagram, classes are typically depicted as rectangles with compartments for the class name, attributes, and methods.
* **Attributes:** These represent the characteristics or data points associated with an object of a particular class. They are listed within the first compartment of the class rectangle.
* **Methods:** These represent the actions or functionalities that objects of a class can perform. They are listed within a separate compartment of the class rectangle, often below the attributes.
* **Relationships:** Relationships between classes are depicted using lines or arrows connecting them. These relationships can represent different types of interactions, such as inheritance (where one class inherits properties and methods from another), association (where objects of two classes have a connection), or aggregation (where one class is a part of another).
* **Key components:**

 **User**: Interacts with the web application to get disease predictions.

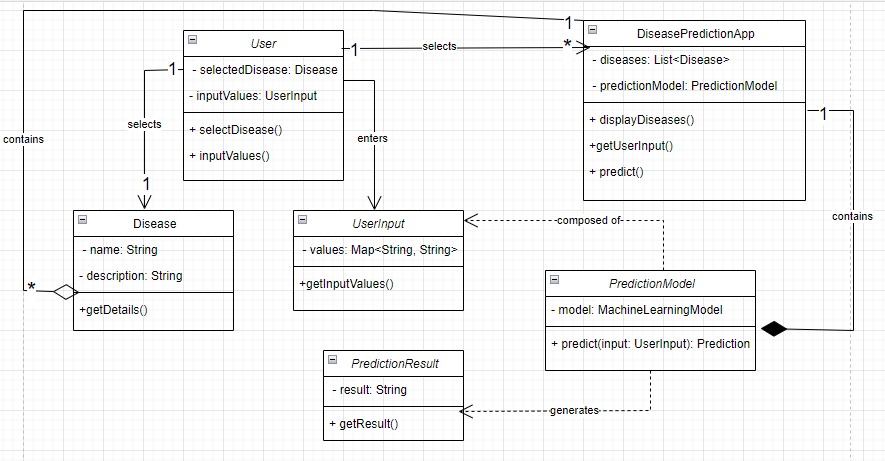
 **DiseasePredictionApp**: The main application that coordinates the interaction between the user, disease list, and prediction logic.

 **Disease**: Represents a disease that the user can select.

 **PredictionModel**: Contains the machine learning logic to make predictions based on user input.

 **UserInput**: Collects and stores the input values from the user.

 **PredictionResult**: Stores the result of the prediction.



### **Explanation:**

1. **User**:
   * Attributes:
     + selectedDisease: The disease selected by the user.
     + inputValues: The values input by the user.
   * Methods:
     + selectDisease(): Allows the user to select a disease.
     + inputValues(): Allows the user to input values.
2. **DiseasePredictionApp**:
   * Attributes:
     + diseases: A list of available diseases.
     + predictionModel: The machine learning model used for predictions.
   * Methods:
     + displayDiseases(): Displays the list of diseases for the user to select.
     + getUserInput(): Gets the input values from the user.
     + predict(): Makes a prediction based on the user's input and selected disease.
3. **Disease**:
   * Attributes:
     + name: The name of the disease.
     + description: A description of the disease.
   * Methods:
     + getDetails(): Returns the details of the disease.
4. **PredictionModel**:
   * Attributes:
     + model: The machine learning model used for prediction.
   * Methods:
     + predict(input: UserInput): PredictionResult: Makes a prediction based on the user's input and returns the result.
5. **UserInput**:
   * Attributes:
     + values: A map of input field names to input values.
   * Methods:
     + getInputValues(): Returns the user's input values.
6. **PredictionResult**:
   * Attributes:
     + result: The prediction result.
   * Methods:
     + getResult(): Returns the prediction result.

### **Relationships Explained:**

1. **User to DiseasePredictionApp**: The user interacts with the DiseasePredictionApp to perform actions like selecting a disease and inputting values (Association).
2. **DiseasePredictionApp to Disease**: The DiseasePredictionApp contains a list of Disease objects, indicating aggregation.
3. **DiseasePredictionApp to PredictionModel**: The DiseasePredictionApp includes a PredictionModel, implying a composition relationship.
4. **User to Disease**: The user selects a disease from the available options provided by the DiseasePredictionApp (Association).
5. **User to UserInput**: The user provides input values that are stored in a UserInput object (Association).
6. **PredictionModel to UserInput**: The PredictionModel depends on the UserInput to make predictions (Dependency).
7. **PredictionModel to PredictionResult**: The PredictionModel produces a PredictionResult based on the UserInput (Dependency).

* **STATE DIAGRAM :**
* A state chart diagram represents the different states of an object and how it transitions from one state to another based on events. Here is a state chart diagram for the web application described

### 

### **Explanation:**

1. **START: the process starts being**
2. **Initial State**: The starting point of the state chart.
3. **Waiting for the User to Start**: The application is waiting for the user to initiate the process.
4. **Display Diseases**: The state where the application displays the list of diseases.
5. **Disease Selected**: The state entered after the user selects a disease.
6. **Prompt for User Input**: The application prompts the user to enter the required input values for the selected disease.
7. **Collecting Input**: The state where the application is collecting input values from the user.
8. **Predicting Disease**: The application uses the input values to make a prediction using the machine learning model.
9. **Displaying Result**: The state where the application displays the prediction result to the user.
10. **Final State**: The endpoint of the state chart, indicates that the process is complete.
11. **END**: THE PROCESS ENDING STATE

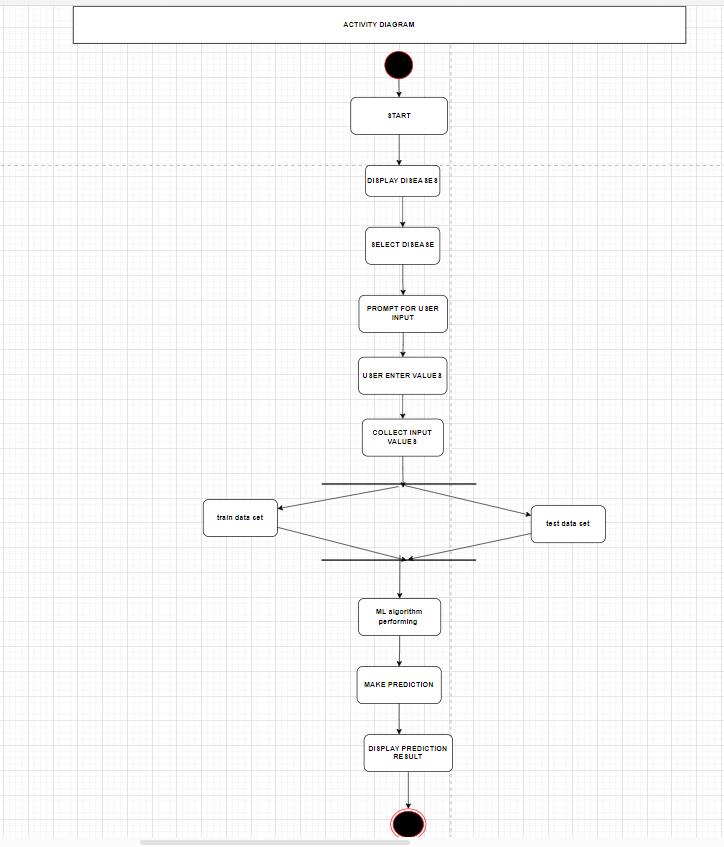
* This state chart diagram provides a clear view of the different states the application goes through, from the initial state to the final state, based on user interactions and events within the system.
* **ACTIVITY DIAGRAM :**

### **Explanation:**

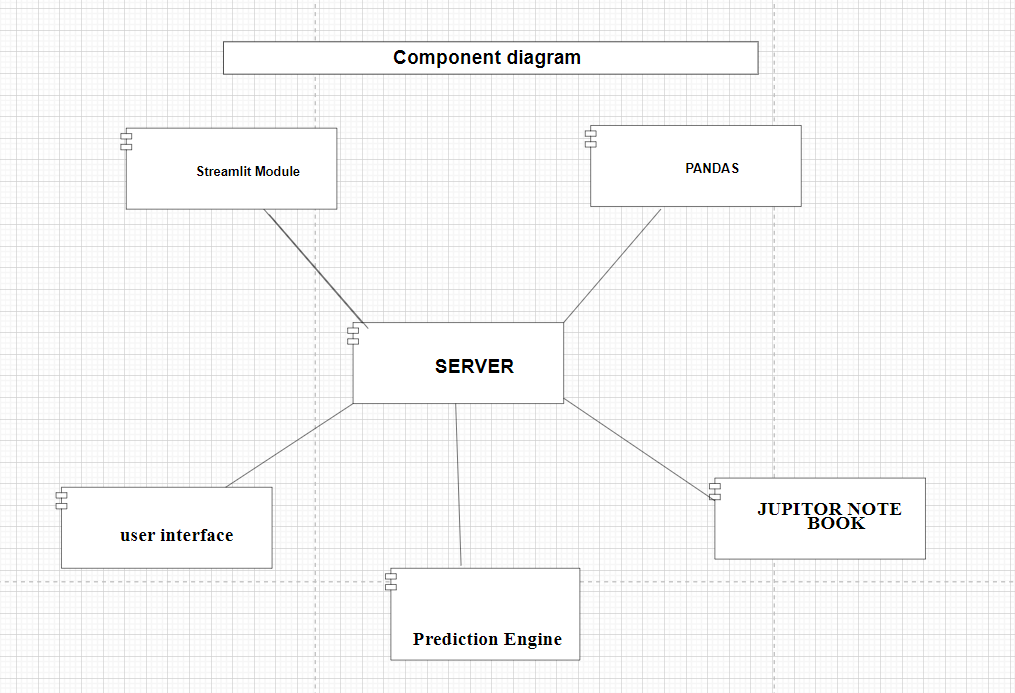
### An activity diagram provides a visual representation of the workflow or process.OR is a graphical representation that depicts the flow of activities and actions within a system. It focuses on the sequential or concurrent steps involved in a process, highlighting decision points, alternative paths, and synchronization points between different parts of the system.

1. **Start**: The activity begins.
2. **Display Diseases**: The application displays the list of diseases available for selection.
3. **Select Disease**: The user selects a disease from the displayed list.
4. **Prompt for User Input**: The application prompts the user to enter the required input values for the selected disease.
5. **User Enters Values**: The user inputs the required values.
6. **Collect Input Values**: The application collects the input values entered by the user.
7. **Make Prediction**: The application uses the collected input values and the selected disease to make a prediction using the machine learning model.
8. **Display Prediction Result**: The application displays the prediction result to the user.
9. **End**: The activity ends.

* This activity diagram provides a clear view of the steps involved in the process, from the user selecting a disease to receiving a prediction result.

****

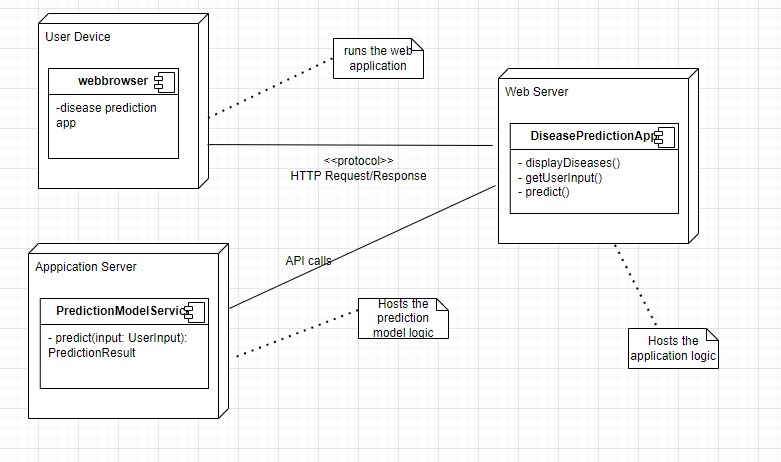
* **COMPONENT DIAGRAM:**
* Absolutely! Here's a breakdown of the UML component diagram based on the title "ML-Driven Early Detection for Optimal Health – Empowering You with Accurate Predictive Health Analytics":

****

* **Key Components:**
* **User Input (data):** This represents the data a user provides to the system. In the context of healthcare, this could be information about symptoms, health history, or other relevant details.
* **Streamlit Module:** This likely signifies a software library used to create the user interface (UI) of the system. Streamlit allows for building data apps quickly, suggesting an interactive interface for users to input data and potentially view results.
* **Server:** This is the central component of the system, responsible for processing user data and generating predictions. It acts as a bridge between the user interface and the prediction engine.
* **Pandas:** This is likely a Python library used for data analysis and manipulation on the server side. Pandas help manage and clean the user data before feeding it into the prediction engine.
* **Jupyter Notebook:** This component signifies a software environment where data scientists or developers might have created the machine learning model used for predictions. It wouldn't be a direct part of the deployed system but signifies the origin of the prediction logic.
* **Relationships:**
* **User Input (data) connects to Server:** This arrow depicts the flow of user data from the input interface to the server for processing.
* **Streamlit Module connects to Server (user interface):** This connection signifies that the Streamlit module provides the user interface that interacts with the server. It allows users to input data and potentially receive results generated by the server.
* **Jupyter Notebook connects to Server (Prediction Engine):** This doesn't represent a direct connection in the deployed system but indicates that the prediction engine logic likely originated from a Jupyter Notebook where it was developed and trained. The server would use this pre-trained model for making predictions.
* **Pandas Connect to Server:** This implies that the Pandas library is used on the server side, likely for data manipulation and preparation before feeding it into the prediction engine.
* **Overall Explanation:**

This UML component diagram depicts a system designed for Machine Learning-based health predictions. Users interact with a UI likely built using Streamlit, providing their health data. The server receives this data, potentially cleans and prepares it using Pandas, and then utilizes a pre-trained prediction engine (developed in Jupyter Notebook) to generate predictions.

* **DEPLOYMENT DIAGRAM:**
* A deployment diagram in UML shows the physical deployment of artifacts on nodes, such as how software components (artifacts) are deployed on hardware components (nodes). Here's a deployment diagram for the Disease Prediction System:



### **Explanation of the Deployment Diagram:**

1. **User Device**:
   * Represents the device used by the user to interact with the web application.
   * Contains a Web Browser running the DiseasePredictionApp.
2. **Web Server**:
   * Hosts the DiseasePredictionApp.
   * Handles HTTP requests and responses between the user device and the web server.
3. **Application Server**:
   * Hosts the PredictionModelService.
   * Handles API calls for predictions and processes user inputs using the prediction model.

### **Key Components:**

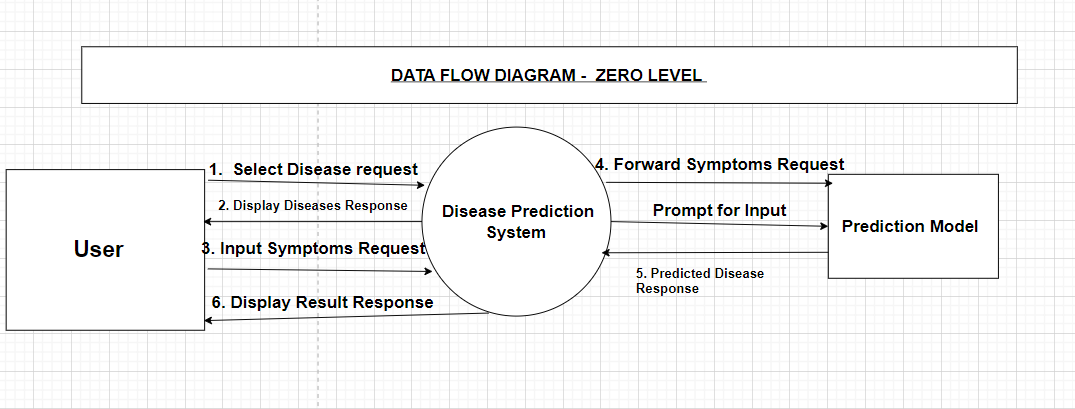
* **Web Browser**:
  + Executes the front-end part of the DiseasePredictionApp and communicates with the Web Server.
* **DiseasePredictionApp**:
  + Handles displaying diseases, collecting user input, and making predictions.
  + Deployed on the Web Server.
* **PredictionModelService**:
  + Contains the prediction logic using the machine learning model.
  + Deployed on the Application Server.

### **Communication:**

* **HTTP Request/Response**:
  + Between the User Device and the Web Server.
  + For interactions like selecting diseases and submitting input values.
* **API Calls**:
  + Between the Web Server and the Application Server.
  + For invoking prediction logic.

**DATA.FLOW.DIAGRAM -DFD :**

* This DFD provides a high-level overview of the disease prediction system. It shows that the system interacts with a user, retrieves information about a disease, leverages a disease prediction model, and displays the predicted disease back to the user. It doesn't delve into the specifics of how the prediction model works but focuses on the data flow between the user and the system.
* Data Flow Diagrams (DFDs) are graphical representations used to depict the flow of data within a system. They break down the system into processes and show how data moves between them. Here's a breakdown of the different levels of DFDs:
* **DATA FLOW DIAGRAM – DFD ( 0th level) :**
* **Level 0: Context Diagram**
* **Focus:** Provides a high-level overview of the entire system.
* **Components:**
  + **Single Process:** Represents the entire system as a single bubble.
  + **External Entities:** Actors or systems outside the system that interact with it (e.g., users, databases).
  + **Data Flows:** Arrows depicting the flow of data between external entities and the system process.
* **Purpose:** Provides a starting point for system analysis, defining the overall scope and interactions with the external world.



* **Key components:**

**1. User:**

* This represents the individual who interacts with the system. They might input information about their health condition, symptoms, or any relevant details regarding a potential disease. This information could be entered through a user interface (UI) that is not explicitly shown in the description.

**2. Disease Prediction System:**

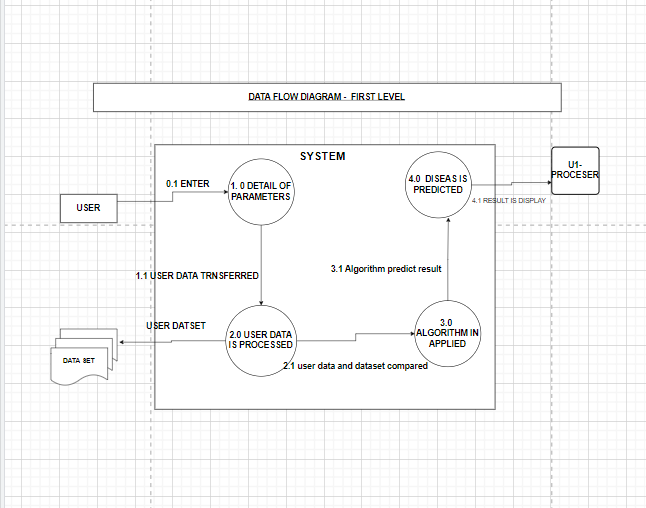
* This is the core system responsible for processing user data and generating disease predictions. It acts as an intermediary between the user and the prediction model. Here's what the system might do:
  + **Receive User Input:** The system gathers the information provided by the user.
  + **Process and Prepare Data:** The system might clean, format, or transform the user data into a format suitable for the prediction model.
  + **Interact with Prediction Model:** The system sends the processed user data to the prediction model.
  + **Receive Prediction Results:** The system retrieves the predicted disease information from the model.
  + **Display Results:** The system displays the predicted disease information for the user, likely through a user interface.

**3. Prediction Model:**

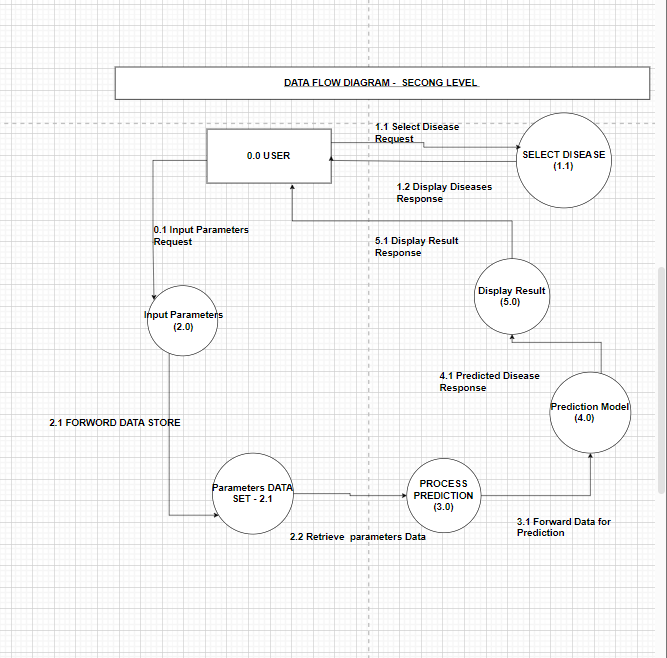
* This component represents the external resource used for generating disease predictions. It could be a pre-trained machine learning model, a database of medical knowledge, or any other external system capable of analyzing the user's data and providing disease predictions.
* The prediction model likely receives the processed user data from the disease prediction system and analyzes it based on its internal mechanisms. It then generates a prediction or response, which is sent back to the disease prediction system.
* **Overall Interaction:**

1. The user interacts with the system by providing information about their health condition.
2. The disease prediction system receives this information and processes it.
3. The system sends the processed data to the prediction model.
4. The prediction model analyzes the data and generates a predicted disease.
5. The prediction model sends the prediction result back to the disease prediction system.
6. The disease prediction system displays the predicted disease information for the user.

* **DATA FLOW DIAGRAM – DFD ( 1st level) :**
* **Level 1: Detailed Context Diagram**
* **Focus:** Expands on the context diagram by decomposing the single system process into its primary sub-processes.
* **Components:**
  + **Multiple Processes:** This breaks down the central process from level 0 into more specific sub-processes that perform distinct functionalities.
  + **Data Stores:** Represents temporary or permanent repositories where data is stored within the system.
  + **Data Flows:** Arrows depicting the flow of data between processes, external entities, and data stores.
* **Purpose:** Provides a more detailed view of the system's internal workings, highlighting the main functionalities and data flow between them.



* **Main Aspects:**
* **User:** This represents the individual who interacts with the system, providing their health information and receiving disease predictions.
* **Dataset (Disease Prediction Model):** This signifies the external resource used for generating disease predictions. It could be a pre-trained machine learning model, a database of medical knowledge, or any system capable of analyzing user data and predicting potential diseases.
* **User Interface (Implicit):** While not explicitly shown in the DFD, a user interface (UI) is likely present. This is the software element that the user interacts with to provide input and receive disease prediction results.
* **Key Components:**
* **Process (Disease Prediction System):** This central rectangle represents the core system that manages the entire disease prediction process. It interacts with the user, the dataset (disease prediction model), and the UI (implicit).
* **Data Flows (4):**
  + **Select Disease Request (implicit):** This flow likely originates from the User through the UI and points to the Disease Prediction System. It signifies the user initiating the process, possibly by selecting a disease of interest or requesting a general health evaluation.
  + **Forward Symptoms Request:** This flow originates from the Disease Prediction System and points to the Dataset (disease prediction model). It depicts the system sending the user's information or symptoms to the model for analysis.
  + **Predicted Disease Response:** This flow originates from the Dataset and points back to the Disease Prediction System. It represents the predictions or analysis results sent back by the model.
  + **Display Real-Time Response:** This flow originates from the Disease Prediction System and points to the implicit User Interface. It signifies the system displaying the disease predictions or health analysis results for the user.
* **DATA FLOW DIAGRAM – DFD ( 2nd level) :**
* **Level 2: Exploded View**
* **Focus:** Dives deeper into a specific sub-process from level 1, illustrating its internal details.
* **Components:**
  + **Sub-process Breakdown:** Expands on a particular sub-process from level 1, further decomposing it into even more granular sub-processes or steps.
  + **Data Stores (Optional):** This may include additional data stores relevant to the specific sub-process being analyzed.
  + **Data Flows:** Arrows depicting the flow of data between the sub-processes within the exploded process and any relevant data stores.
* **Purpose:** Provides a very detailed view of a specific sub-process, allowing for a detailed step-by-step process.



the key components and their relationships in the level 2 DFD appear to be as follows:

* **Key Components:**
* **User:** This represents the external entity that interacts with the system, likely a patient providing their symptoms or medical information.
* **Select Disease:** This could be interpreted in two ways:
  + **Process:** It might represent a user action within the system, where they select a specific disease of interest from a menu or list.
  + **Data Flow:** Alternatively, it could depict the user's input specifying the disease they are interested in learning more about.
* **Input Parameters:** This signifies the data provided by the user, likely their symptoms or medical history details.
* **Parameters Data:** This could be a data store that holds the user-provided input parameters (symptoms/medical history).
* **Process Prediction:** This represents the central process responsible for analyzing the user's input data to predict a potential disease. It likely leverages the Prediction Model (data store) for this analysis.
* **Prediction Model:** This signifies a data store or external resource that contains the knowledge or model used for disease prediction. It could be a pre-trained machine learning model or a database of medical knowledge.
* **Display Result:** This could be interpreted in two ways:
  + **Process:** It might represent the system's action of displaying the predicted disease(s) for the user.
  + **Data Flow:** Alternatively, it could depict the output data from the system, which is the predicted disease information.
* **Key Relations:**
* Data flow diagrams (DFDs) typically use arrows to depict data flow, not numbers to represent relations. Here's an explanation of the data flows based on the interpretation of the components:

1. **The user initiates:** The process starts with the user initiating the interaction, possibly by selecting a disease of interest (if interpreted as a process) or providing their symptoms/medical history (if interpreted as data flow). This user input is represented by the arrow labeled "Select Disease" or "Input Parameters".
2. **Data flow to process:** The user-provided data (symptoms/medical history) flows into the "Process Prediction" block.
3. **Data flow to data store (optional):** The user input might be stored in the "Parameters Data" store for further processing or reference.
4. **Process Prediction interacts with model:** The "Process Prediction" block interacts with the "Prediction Model" (data store), likely using the user data to query the model and generate predictions.
5. **Prediction results to display:** The outcome of the prediction process, likely the disease predictions, flows to the "Display Result" block.
6. **The system displays results:** The "Display Result" block (interpreted as a process) presents the predicted disease information to the user.

**Overall, the level 2 DFD depicts the user providing medical information, the system analyzing it using a prediction model, and displaying the predicted disease(s) for the user.**

**ER – Diagram / Model Design :**

* The relation upon the system is structured through a conceptual ER-Diagram, which not only specifics the existential entities but also the standard relations through which the system exists and the cardinalities that are necessary for the system state to continue.

• The entity Relationship Diagram (ERD) depicts the relationship between the data objects. The ERD is the notation that is used to conduct the data modeling activity the attributes of each data object noted in the ERD can be described resign a data object description.

• The set of primary components that are identified by the ERD are

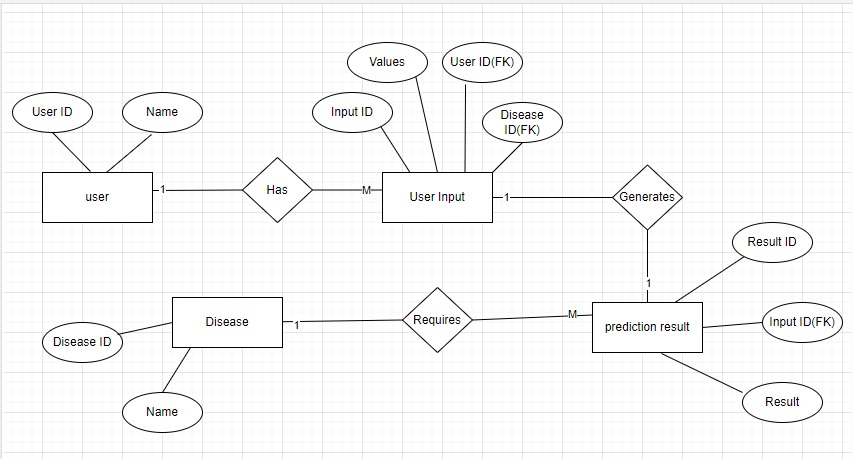
• Data object

• Relationships

• Attributes

• Various types of indicators.

The primary purpose of the ERD is to represent data objects and their relationships.



### **Explanation:**

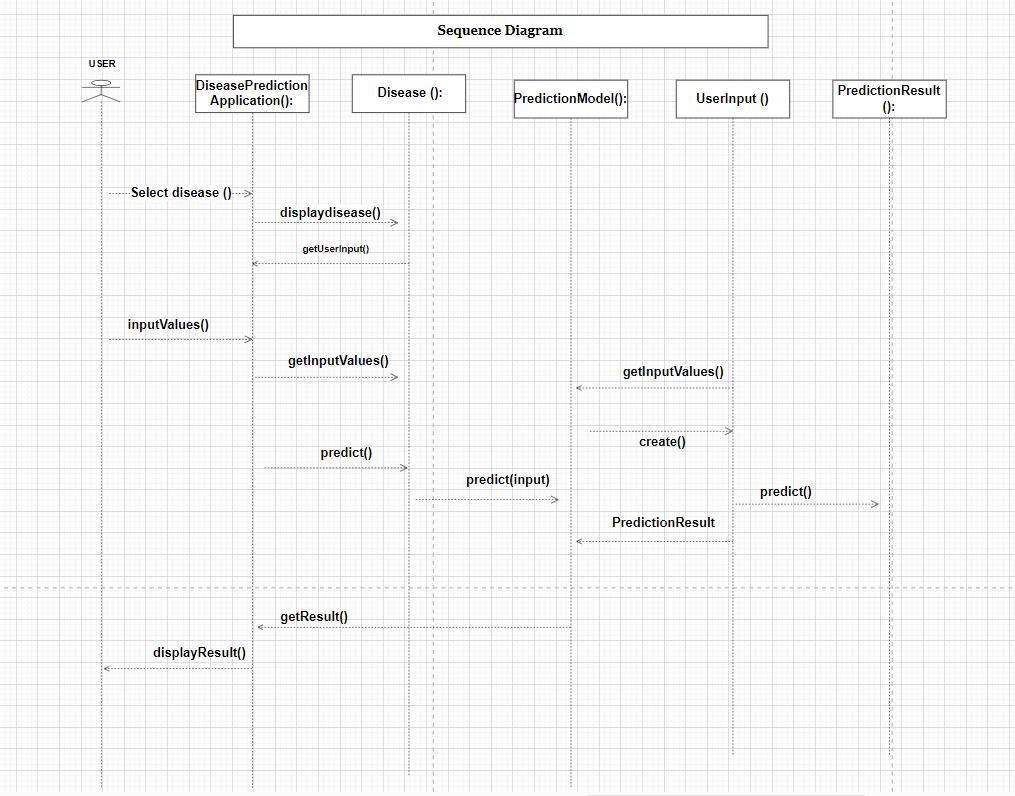
1. **User**:
   * Attributes:
     + UserID: Unique identifier for the user.
     + Name: The name of the user.
2. **Disease**:
   * Attributes:
     + DiseaseID: Unique identifier for the disease.
     + Name: The name of the disease.
     + Description: A description of the disease.
3. **UserInput**:
   * Attributes:
     + InputID: Unique identifier for the user input.
     + Values: The input values provided by the user.
     + UserID (FK): Foreign key referencing the User.
     + DiseaseID (FK): Foreign key referencing the Disease.
4. **PredictionResult**:
   * Attributes:
     + ResultID: Unique identifier for the prediction result.
     + Result: The result of the prediction.
     + InputID (FK): Foreign key referencing the UserInput.

### **Relationships:**

1. **User to UserInput**:
   * A User can have many UserInput entries, but each UserInput is associated with only one User (One-to-Many relationship).
2. **UserInput to Disease**:
   * A UserInput is associated with one Disease (Many-to-One relationship).
3. **UserInput to PredictionResult**:
   * Each UserInput generates one PredictionResult (One-to-One relationship).
4. **PredictionResult requires Disease**:
   * A PredictionResult requires a Disease to provide context for the prediction (Many-to-One relationship).

This ER diagram helps visualize the data structure and relationships in your web application.

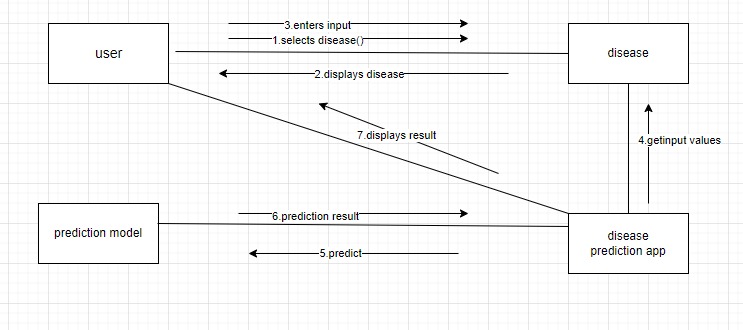
* **SEQUENCE DIAGRAM:**
* Overall, sequence diagrams are a valuable tool for visualizing and documenting the interactions between objects in a system, especially when focusing on the message flow in a specific scenario

****

* **Explanation:**
* This sequence diagram outlines the flow of actions from user interaction to prediction result generation and sequence diagram is a type of interaction diagram that focuses on the message flow between objects in a specific scenario or use case. It depicts the interactions chronologically, highlighting the sequence of messages exchanged between objects as they collaborate to achieve a particular goal.

1. **User selects a disease**:
   * The User selects a disease via the selectDisease() method.
   * DiseasePredictionApp calls displayDiseases() to show the list of diseases.
   * The User then sees the list and selects a disease.
2. **User input values**:
   * The User provides input values via the inputValues() method.
   * DiseasePredictionApp calls getInputValues() to retrieve the user inputs.
3. **PredictionModel processes the input**:
   * DiseasePredictionApp creates a UserInput object with the collected values.
   * DiseasePredictionApp calls predict() on the PredictionModel, passing the UserInput.
   * The PredictionModel processes the input and generates a PredictionResult.
4. **A user receives the prediction result**:
   * DiseasePredictionApp retrieves the result using getResult().
   * The result is displayed to the User via the displayResult() method.

* **COLLABORATION DIAGRAM :**
* A collaboration diagram, also known as a communication diagram, focuses on the interactions between objects and their relationships. It is based on the sequence diagram provided earlier.



### **Explanation:**

1. **User selects a disease**:
   * The User initiates the interaction by calling selectDisease() on the DiseasePredictionApp.
2. **DiseasePredictionApp displays diseases**:
   * The DiseasePredictionApp responds by calling displayDiseases() to show the list of available diseases.
3. **User input values**:
   * The User provides input values by calling inputValues() on the DiseasePredictionApp.
4. **DiseasePredictionApp collects input values**:
   * The DiseasePredictionApp collects the input values by calling getInputValues() on itself.
5. **DiseasePredictionApp makes a prediction**:
   * The DiseasePredictionApp calls predict() on the PredictionModel to make a prediction based on the user inputs.
6. **PredictionModel processes the input**:
   * The PredictionModel calls predict(input) using the UserInput to generate a PredictionResult.
7. **PredictionResult generated**:
   * The PredictionModel returns a PredictionResult.
8. **DiseasePredictionApp retrieves the result**:
   * The DiseasePredictionApp retrieves the prediction result by calling getResult() on the PredictionModel.
9. **Displaying the result to the user**:
   * Finally, the DiseasePredictionApp displays the prediction result to the User by calling displayResult().

* This collaboration diagram showcases how objects interact to fulfill the use case of predicting a disease based on user input.

**SOFTWARE DESIGN AND ARCHITECTURE**

**(“S.D.A”)**

* the software architecture pattern that most closely aligns with your setup is the Layered Architecture Pattern. Here’s how it fits your project:

**Layered Architecture Pattern**

1. Presentation Layer

Technology: Streamlit

Functionality: Provides a web interface for user interaction.

Components: Input forms for health data, sections for displaying predictions and visualizations.

2. Application Layer

Data Preprocessing Module:

Tasks: Cleaning, normalization, feature engineering.

Libraries: Pandas, NumPy

Model Training and Validation Module:

Tasks: Training machine learning models, validating performance.

Libraries: Scikit-learn, TensorFlow/Keras

Models: Logistic Regression, Random Forest, SVM, Neural Networks

Evaluation: Accuracy, precision, recall, F1-score, ROC-AUC.

Prediction Module:

Tasks: Real-time predictions based on user input.

Libraries: Trained models (Scikit-learn, TensorFlow/Keras)

Integration: Embedded within the Streamlit app.

Reporting Module:

Tasks: Generate performance reports and visualizations.

Libraries: Matplotlib, Seaborn

3. Data Layer

Data Sources:

Datasets: Diabetes, Heart Disease, Parkinson's Disease datasets from reliable sources.

Storage: Local files or cloud storage.

4. Infrastructure Layer

Development Environment:

Tools: Jupyter Notebook, VS Code

Dependencies: Virtual environment setup with necessary libraries.

Deployment Environment:

Platform: Local machine or cloud-based platforms (AWS, Google Cloud, Azure).

Configuration: High-performance laptops with potential GPU support.

Explanation of Fit

Layered Structure: Your project clearly separates concerns into different layers (Presentation, Application, Data, Infrastructure).

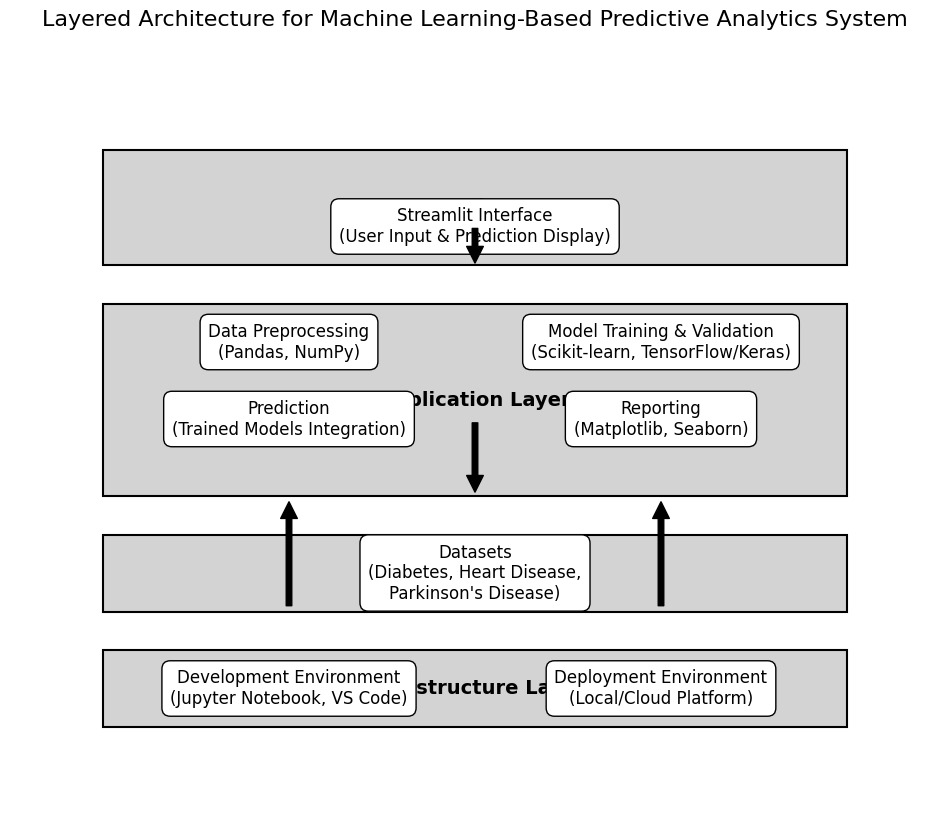
Modularity: Each module has a distinct responsibility, promoting a clean separation of concerns.

Scalability and Maintenance: The layered architecture makes it easier to maintain and scale individual components of the application.

User Interface: The Presentation Layer is distinct and uses Streamlit for the front-end, aligning well with the layered pattern.

**Conclusion**

While other architecture patterns could potentially be used depending on further specifics and scalability requirements, the Layered Architecture Pattern best describes your current approach. It emphasizes the separation of concerns, modularity, and a clear hierarchical structure, making it suitable for developing and maintaining a complex system like a multiple disease prediction application.

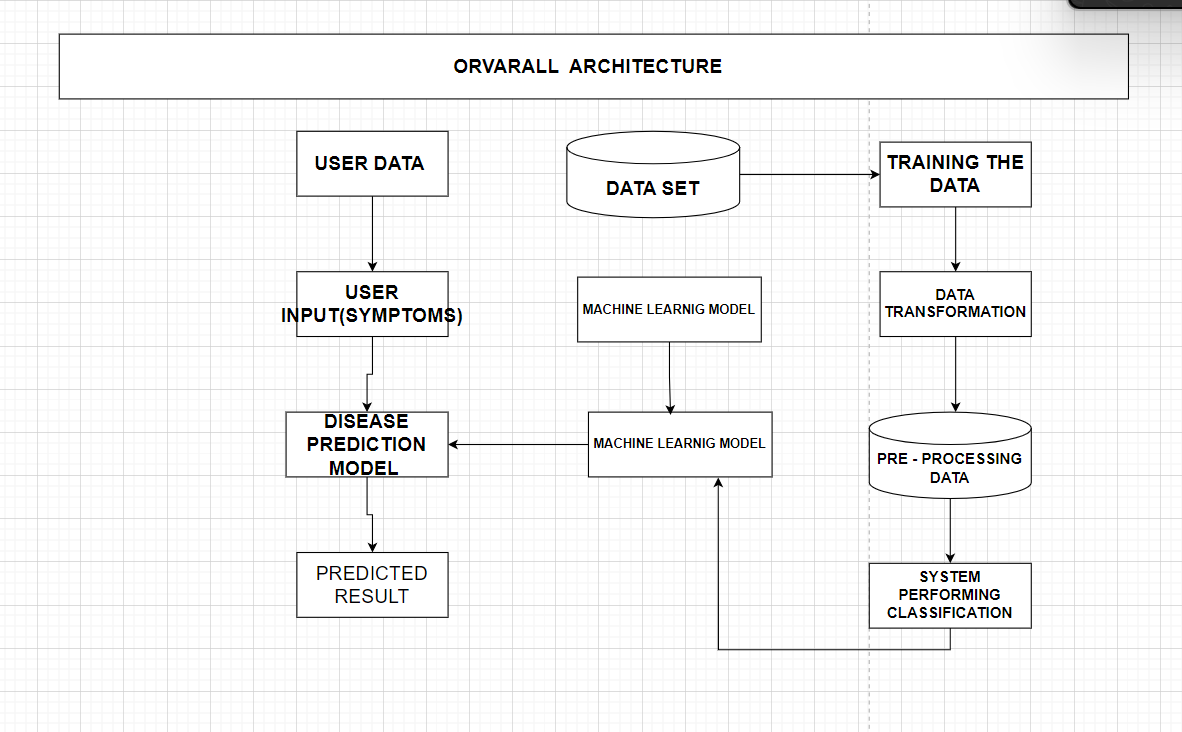


THE MAJORLY THE SOFTWARE DESIGN AND ARCHITECTURE (“S.D.A”) :

* Software Architecture & Design - The architecture of a system describes its major components, their relationships (structures), and how they interact with each other.
* Software architecture and design includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.
* Also, we are specified that S.D.A as majorly 3 designs and architecture

**I.OVERALL ARCHITECTURE DIAGRAM:**

* An architectural diagram is a visual representation that maps out the physical implementation of components of a software system. It shows the general structure of the software system and the associations, limitations, and boundaries between each element.



**Detailed Explanation of the Overall Architecture Diagram:**

* The architecture diagram is the workflow of the MLDriven Early Detection for Optimal Health system. It can be divided into several main components, each playing a critical role in the disease prediction process.

**Steps:**

1. User Data Input:

User Input (Symptoms): The process starts with the user entering their symptoms or health data into the system. This data is crucial as it serves as the primary input for the prediction model.

2. Data Set and Machine Learning Model Training:

Data Set: This block represents the historical medical data collected from various sources. This data is used to train the machine learning models.

Training the Data: This step involves preparing the data for training. It includes:

Data Transformation: Converting the raw data into a suitable format for machine learning algorithms.

Preprocessing Data: Cleaning and normalizing the data to ensure high quality and consistency.

System Performing Classification: Using various algorithms to classify the data into different disease categories.

Machine Learning Model: After preprocessing, the data is used to train machine learning models. These models learn patterns and relationships within the data, which are crucial for accurate predictions.

3. Disease Prediction Model:

Machine Learning Model: The trained machine learning models are applied to the user input data. These models analyze the symptoms provided by the user and predict the likelihood of different diseases.

Disease Prediction Model: This is the core component where the actual prediction happens. The machine learning model processes the user input data and generates a predicted result.

4. Predicted Result:

Predicted Result: The final output of the system is the predicted result. This result indicates the likelihood of the user having a particular disease based on the input symptoms.

**Conclusion**

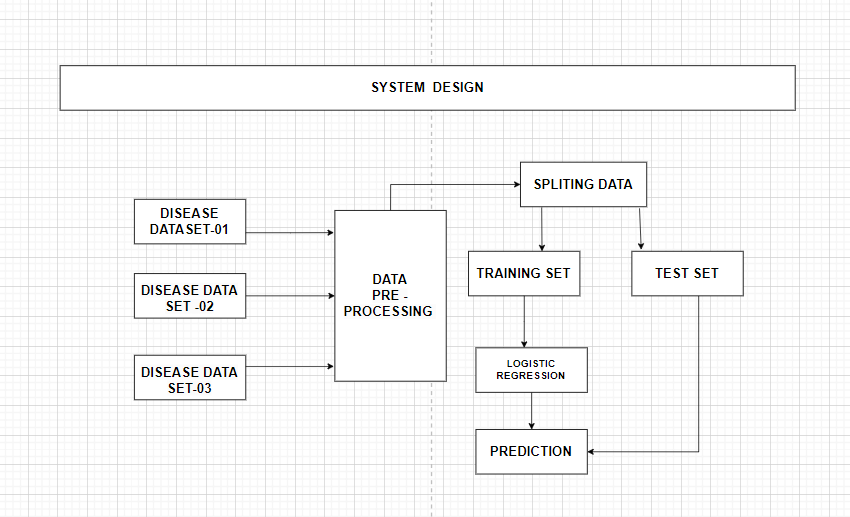
* The architecture of the MLDriven Early Detection for Optimal Health Systems demonstrates a robust framework for disease prediction using machine learning. By integrating user input with historical data, transforming and preprocessing the data, and utilizing advanced machine learning models, the system can provide accurate and timely predictions for various diseases.

1. User-Friendly: The system is designed to be user-friendly, allowing individuals to input their symptoms easily and receive quick predictions.
2. DataDriven: Leveraging a vast dataset and machine learning, the system ensures high accuracy and reliability in predictions.
3. Comprehensive Workflow: The workflow from data input to prediction result is well-structured, ensuring that each step is optimized for the best performance.
4. Scalable: The architecture is scalable, allowing for the integration of additional diseases and improvements in prediction models as more data becomes available.

In conclusion, this architecture lays a solid foundation for a powerful disease prediction system, aiming to revolutionize how early detection and preventive healthcare are approached. By continuing to refine and expand this system, it can significantly contribute to better health outcomes and proactive disease management.

**II.SYSTEM DESIGN:**

* Systems Design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. It involves translating user requirements into a detailed blueprint that guides the implementation phase.



**Detailed Explanation of the System Design Diagram:**

* The system design diagram provides a step-by-step workflow of the disease prediction model, from data collection to prediction. Below is a detailed explanation of each step in the process:

**Steps:**

1. Disease Datasets:

Disease Dataset01, Disease Dataset02, Disease Dataset03: These datasets represent various sources of disease-related data. Each dataset contains records of patient symptoms, medical history, and diagnosis information.

2. Data PreProcessing:

Data PreProcessing: This is a crucial step where the raw data from different datasets is cleaned and transformed to ensure quality and consistency. It involves handling missing values, normalizing data, and encoding categorical variables.

3. Splitting Data:

Splitting Data: After preprocessing, the data is split into two main sets:

Training Set: This set is used to train the machine learning model. It constitutes the majority of the data.

Test Set: This set is used to evaluate the performance of the trained model. It helps in assessing how well the model generalizes to new, unseen data.

4. Training Set:

Training Set: The subset of the data used to train the machine learning algorithms.

5. Logistic Regression:

Logistic Regression: This specific algorithm is used for training in the given system design. Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. It is particularly useful for binary classification problems (e.g., predicting the presence or absence of a disease).

6. Prediction:

Prediction: Once the logistic regression model is trained, it is used to make predictions on new data. The model analyzes the test set and provides predictions based on the learned patterns.

7. Test Set:

Test Set: This subset of the data is used to test the performance of the trained model. The predictions made on the test set are compared with the actual outcomes to evaluate the accuracy and reliability of the model.

**Conclusion**

* The system design for the MLDriven Early Detection for Optimal Health system illustrates a comprehensive and systematic approach to developing a disease prediction model. Here are the key points:

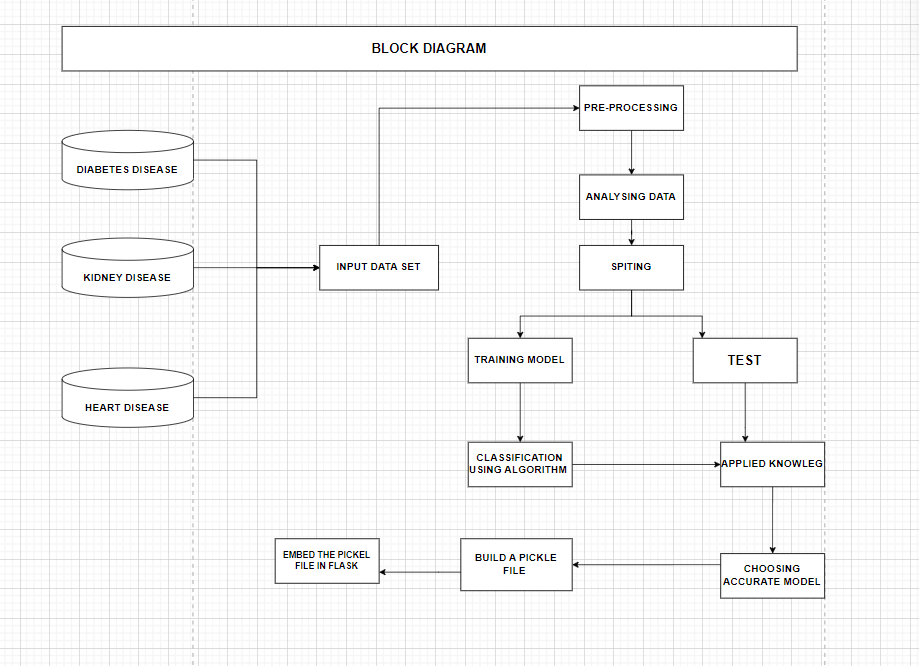
1. Data Integration and PreProcessing: The integration of multiple disease datasets and thorough preprocessing ensure that the data fed into the model is clean, consistent, and suitable for analysis.

1. Model Training and Evaluation: Splitting the data into training and test sets allows for effective training of the logistic regression model and accurate evaluation of its performance.
2. Logistic Regression for Prediction: Utilizing logistic regression, a robust and interpretable machine learning algorithm, enhances the model's ability to predict the likelihood of diseases based on user input.
3. Reliable Predictions: By rigorously testing the model on unseen data, the system ensures that the predictions are reliable and generalizable to new cases.

In summary, this system design lays out a clear and efficient workflow for building a disease prediction model. By following these steps, the system aims to provide accurate and timely predictions, contributing significantly to early disease detection and proactive healthcare management.

**III. BLOCK DIAGRAM :**

* A block diagram is a drawing illustration of a system whose major parts or components are represented by blocks.



**Detailed Explanation of the System Design Diagram**

* The system design diagram provides an in-depth view of the process flow from data acquisition to the final prediction. Below is a detailed step-by-step explanation of each component in the diagram:

**Steps:**

1. Disease Data Set01, Disease Data Set02, Disease Data Set03:

These are the different datasets containing disease-related data. Each dataset might correspond to different diseases such as diabetes, heart disease, and kidney disease.

2. Data PreProcessing:

Data PreProcessing: This is a critical step where raw data from the datasets is cleaned and prepared for analysis. This includes:

Data Cleaning: Handling missing values, removing duplicates, and correcting inconsistencies.

Data Transformation: Normalizing data, encoding categorical variables, and scaling numerical features.

Feature Engineering: Creating new features or modifying existing ones to improve model performance.

3. Splitting Data:

After preprocessing, the data is split into training and test sets. This step is essential for evaluating the model's performance on unseen data.

Training Set: This subset of the data is used to train the machine learning models.

Test Set: This subset is used to evaluate the performance of the trained model and ensure it generalizes well to new data.

4. Training Set:

Training Set: The training set is used to fit the machine learning model. The model learns patterns and relationships within the data during this phase.

5. Logistic Regression:

Logistic Regression: This is one of the machine learning algorithms used for training. It is particularly effective for binary classification problems, such as predicting the presence or absence of a disease.

6. Prediction:

Prediction: Once the model is trained, it can be used to make predictions on new, unseen data. The predictions indicate the likelihood of the disease based on the input features.

**Conclusion**

The system design diagram provides a comprehensive view of the data flow and processing involved in the disease prediction system. Key points include:

1. Diverse Data Sources: The system uses multiple datasets, each potentially representing different diseases, to ensure a broad and inclusive model training process.
2. Robust Data PreProcessing: Critical steps are taken to clean, transform, and enhance the data, ensuring that the models are trained on high quality and relevant features.
3. Effective Data Splitting: The data is split into training and test sets to train the model effectively and evaluate its performance rigorously.
4. Utilization of Logistic Regression: Logistic regression is used as one of the primary algorithms, suitable for binary classification tasks in disease prediction.
5. Accurate Predictions: The trained model can predict the likelihood of diseases based on new user data, aiding in early detection and intervention.

* In summary, the system design diagram outlines a structured and systematic approach to developing a machine learning-based disease prediction system. By following these steps, the system aims to provide accurate and timely predictions, supporting proactive healthcare management and better health outcomes.
* **IMPLEMENTATION:**
* Multiple Disease Prediction System using Machine Learning in Python | Streamlit Web App - Deployment" involves creating a web application for predicting multiple diseases using machine learning algorithms. The framework and tools used in the project include:

1. **Machine Learning Algorithms**: Logistic Regression, Decision Tree, Random Forest, etc., for disease prediction.
2. **Python Libraries**: pandas, scikit-learn, numpy, etc., for data manipulation and model training.
3. **Streamlit**: For building and deploying the web application.
4. **Deployment**: The application is deployed using Streamlit sharing or similar services

* Jupyter notebook "HeartExploration.ipynb" primarily focuses on predicting heart disease using the Support Vector Machine (SVM) algorithm. Here's a detailed breakdown of the algorithms and framework used:

### Algorithms:

* **Support Vector Machine (SVM)**: The notebook uses svm.SVC with a linear kernel to build the classification model for predicting heart disease.

### Framework and Tools:

* **Python Libraries**:
  + **NumPy**: For numerical operations.
  + **Pandas**: For data manipulation and analysis.
  + **Seaborn**: For data visualization.
  + **Matplotlib**: For plotting graphs.
  + **scikit-learn**: For machine learning tasks including model training and evaluation.
  + **Pickle**: For saving and loading the trained model.

### Workflow:

1. **Data Loading**: The dataset is loaded using pandas.
2. **Data Exploration**: Initial exploration includes checking the shape, description, and missing values in the dataset.
3. **Data Visualization**: Pair plots and group-by operations to understand data distribution and relationships.
4. **Data Preprocessing**: Separating features (X) and target (Y), and splitting the data into training and test sets.
5. **Model Training**: Using svm.SVC with a linear kernel to train the model on the training data.
6. **Model Evaluation**: Predicting on the test set and calculating the accuracy.
7. **Model Deployment**: Using Pickle to save the trained model and demonstrate how to load and use the model for predictions.

**EXPERIMENT ALGORITHM PERFORMING FIRST IMPLEMENTATION OF PROGRAMMING CODE :**

* 1. **Background:**
* In India, huge mortality occurs due to cardiovascular diseases (CVDs) as these diseases are not diagnosed in early stages. Machine learning (ML) algorithms can be used to build efficient and economical prediction system for early diagnosis of CVDs in India.
  1. **Methods**
* A total of 1670 anonymized medical records were collected from a tertiary hospital in South India. Seventy percent of the collected data were used to train the prediction system. Five state-of-the-art ML algorithms (k-Nearest Neighbours, Naïve Bayes, Logistic Regression, AdaBoost and Random Forest [RF]) were applied using Python programming language to develop the prediction system. The performance was evaluated over remaining 30% of data. The prediction system was later deployed in the cloud for easy accessibility via the Internet.
  1. **Results**
* ML effectively predicted the risk of heart disease. The best performing (RF) prediction system correctly classified 470 out of 501 medical records thus attaining a diagnostic accuracy of 93.8%. Sensitivity and specificity were observed to be 92.8% and 94.6%, respectively. The prediction system attained positive predictive value of 94% and negative predictive value of 93.6%.
  1. **Conclusions**
* ML-based prediction system developed in this study performs well in early diagnosis of CVDs and can be accessed via Internet. This study offers promising results suggesting potential use of ML-based heart disease prediction system as a screening tool to diagnose heart diseases in primary healthcare centres in India, which would otherwise get undetected.