**System Design Document**

**Medical AI - Disease diagnostic tool**

<https://github.com/comp195/senior-project-spring-2023-medical-ai-disease-diagnostic-tool>

**By**

**Mansoor Haidari:** [**m\_haidari@u.pacific.edu**](mailto:m_haidari@u.pacific.edu)

**Korie Westbrook:** [**k\_westbrook@u.pacific.edu**](mailto:k_westbrook@u.pacific.edu)

**Kelvin Luk:** [**k\_luk2@u.pacific.edu**](mailto:k_luk2@u.pacific.edu)

Date of last revision to the document

**05/05/2023**

1. **Table of Contents**

Front Cover ---------------------------------------------------------------------------------------------------1

Table of Contents -------------------------------------------------------------------------------------------2

System Architecture----------------------------------------------------------------------------------------3

Hardware, Software and System Requirements-----------------------------------------------------6

External Interfaces-----------------------------------------------------------------------------------------10

Software Design --------------------------------------------------------------------------------------------11

Class Diagram ----------------------------------------------------------------------------------------------11

Class Specifications----------------------------------------------------------------------------------------11

Interaction Diagrams--------------------------------------------------------------------------------------12

Design Considerations -----------------------------------------------------------------------------------12

User Interface Design-------------------------------------------------------------------------------------13

Glossary of Terms------------------------------------------------------------------------------------------14

Reference ---------------------------------------------------------------------------------------------------16

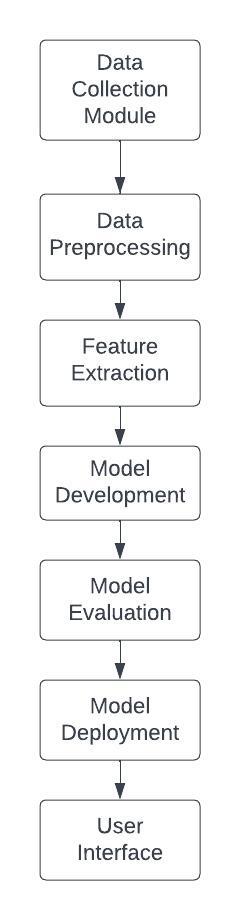
**Project Overviews**

Medical AI is a critical tool for disease diagnosis because it can help healthcare professionals make more accurate and faster diagnoses. By using machine learning algorithms, medical AI can analyze vast amounts of medical data, including patient histories, test results, and imaging studies, to identify patterns and make predictions about the presence of specific diseases. This helps to improve the speed and accuracy of diagnoses, which can lead to earlier treatment and better health outcomes for patients. Additionally, medical AI can assist healthcare professionals in making complex diagnoses that would otherwise be challenging to identify. Overall, medical AI has the potential to significantly improve patient outcomes and reduce healthcare costs.

**System Architecture**

The high-level system architecture of the Medical AI - Disease diagnostic tool will include the following components:

1. **Software modules:**
   1. Collecting Data: In this module, we  will collect important information about the patient, like their medical history, symptoms, vital signs, and lab results.
   2. Data Preprocessing: In this module, the data are cleaned and changed to get rid of any errors, missing values, or outliers. It also means standardizing or scaling the data to make the model more accurate.
   3. Feature Extraction: In this module, we choose from the preprocessed data the most important features that will be used in the prediction model.
   4. Model Development: In this module, we will use machine learning algorithms or other statistical methods to build a model for making predictions.
   5. Model Evaluation: In this module, we use metrics like accuracy, precision, recall, and F1-score to measure how well the model works.
   6. Model Deployment: In this module, the prediction model is put into a production environment so that medical professionals can use it to make predictions.
   7. User Interface: This module is about making a user interface that lets medical professionals enter data about a patient and see the results of the prediction.
   8. External Interfaces: This module is about connecting to outside systems, like Electronic Health Records (EHRs) or other medical databases, to get information about a patient.
2. **Hardware components:**
   1. Data input devices: These can be sensors, monitors, or other devices that can get data from patients, like heart rate monitors or blood pressure cuffs.
   2. Devices for processing and storing data: These could be servers or other high-powered computers that can handle and store a lot of data.
   3. User interface devices: These include screens, keyboards, and other input/output devices that let users interact with the system.
   4. Communication devices: These could be routers, switches, or other networking equipment that lets the system talk to other systems, like electronic health record systems or other healthcare providers.
   5. Mobile devices: These can be things like smartphones or tablets that let doctors and nurses access the system from afar.
3. **User interface:**
   1. Input form: This part lets users enter information about a patient, such as his or her age, gender, medical history, and test results. On the form, you may also be able to choose the disease or condition that you want to be predicted.
   2. Preprocessing module: This module cleans up the input data and pulls out features to get it ready for the prediction module.
   3. Prediction module: This module takes the data you give it and uses machine learning or statistical models to make predictions.
   4. Display of results: This part shows the results of the prediction, such as the disease or condition that was predicted and any relevant statistics or probability scores.
   5. Feedback module: This module lets users give feedback on the results of the prediction. For example, they can confirm or change the disease or condition that was predicted.
4. **Interfaces to external systems:**
   1. Electronic Health Record (EHR) System Interface: The EHR system is used to get the patient's medical history, lab results, list of medications, and other important information. This information can be used to make the prediction system more accurate.
   2. Medical Image Processing Interface: Medical images like X-rays, CT scans, MRI scans, and echocardiograms can sometimes give the prediction system important information. The system can add an interface that lets these images be retrieved and worked on.
   3. Interface for Laboratory Information Management System (LIMS): A LIMS can be used to get lab results like blood tests or genetic tests. This information can be used to make the prediction system more accurate.
   4. Telehealth Interface: With the rise of telehealth, there can be interfaces with remote monitoring systems that can collect and analyze patient data in real time.
   5. Prescription Drug Monitoring Program (PDMP) Interface: This interface is used to get prescription drug history data, which can help find patients who may be at risk of addiction or overdose.

Diagram

Description automatically generatedDiagram

Description automatically generatedDiagram

Description automatically generated

**Hardware, Software and System Requirements**

* **Hardware Requirements:**
  + CPU speed: CPU speed: For most tasks, a modern processor with a clock speed of at least 2 GHz should be enough.
  + RAM: You may need 8 GB or more.
  + Disk space: At least several gigabytes of free space.
  + GPU: A GPU can speed up model training and inference by a lot, especially for models that use deep learning. For applications that use deep learning a lot, it's best to have a dedicated GPU with at least 4 GB of memory.
  + Mobile devices: The system can be accessed from phones or tablets, but the hardware requirements will depend on the device and application.
  + Network connections: A stable Wi-Fi or Ethernet connection is best for getting into the system and sending and receiving data from and to other systems.
* **Software Requirements:**
  + Python programming language (version 3.6 or later)
  + NumPy (version 1.16.4 or later) (version 1.16.4 or later)
  + Pandas (version 0.24.2 or later) (version 0.24.2 or later)
  + Scikit-learn (version 0.20.3 or later) (version 0.20.3 or later)
  + TensorFlow (version 2.0 or later) (version 2.0 or later)
  + Keras (version 2.2.4 or later) (version 2.2.4 or later)
  + PyTorch (version 1.0 or later) (version 1.0 or later)
  + Matplotlib (version 3.0.3 or later) (version 3.0.3 or later)
  + Seaborn (version 0.9.0 or later) (version 0.9.0 or later)
* **System Requirements:**
  + The operating system used by the software components should be supported by the system (e.g., Windows, MacOS, Linux).
  + It is recommended to have at least 20 GB of disk space to store the software and datasets.
  + It needs to be connected to the Internet.
  + To protect sensitive patient data, the system should have the right security and backup measures in place.

**APIs:**

• Google Maps API for geolocation and mapping

• Kaggle API for accessing data sets

**Comment:** The specific requirements may change based on the size and complexity of the data set being used and the model being built.

**External Interfaces**

Data input interface: This interface is needed to get data from sources outside the system. The interface should be able to handle different kinds of data formats and make sure that the data is accurate and consistent.

Data storage interface: This interface is needed to store data from outside sources and get it back out again. The interface should be able to work with different kinds of data storage systems and make sure that the data is safe and private.

Interface for processing data: This is needed to process data from outside sources. The interface should support different ways of processing data and make sure that the data is correct and processed quickly.

User interface: This is the part of the system that lets the end users talk to the system. The interface should be easy to use and intuitive, and it should support different ways to input and output data.

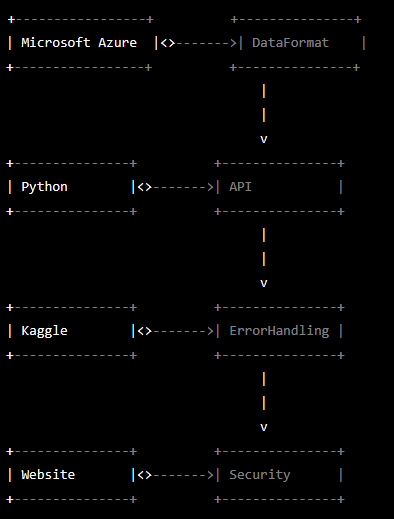
External system interface: You need this interface to talk to other systems outside of your own. The interface should be made so that different systems can share information and work together.

Communication interface: This interface is needed so that the medical disease prediction system and external components can talk to each other in different ways. During data transmission, the interface should protect the security and integrity of the data.

Authentication and authorization interface: This interface is needed to make sure that external parts can access the medical disease prediction system and that they are allowed to do so. The interface should protect the security and privacy of data.

This section describes the communication interfaces between the system and external components. It include the following details:

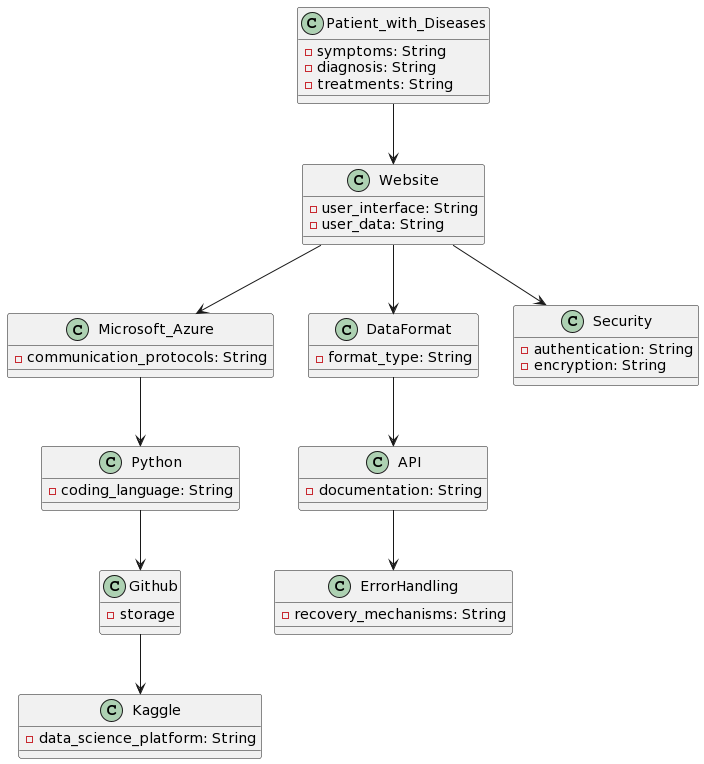
* Protocols used for communication.
* Microsoft azure
* Python
* Kaggle
* Data formats (e.g., JSON, XML)
* API documentation (if available)
* Error handling and recovery mechanisms
* Security considerations, such as authentication and encryption



This diagram shows the associations between the classes "Microsoft Azure", "Python", "Kaggle", "Website", "DataFormat", "API", "ErrorHandling", and "Security". The arrows represent the relationships between classes, where the arrow points from the class that uses the other class to the class that is being used. The association between "DataFormat" and "API" represents the fact that the API uses data formats like JSON and XML, while the association between "API" and "ErrorHandling" represents the fact that the API handles errors. Similarly, the association between "Website" and "Security" represents the fact that the website considers security when handling user data.

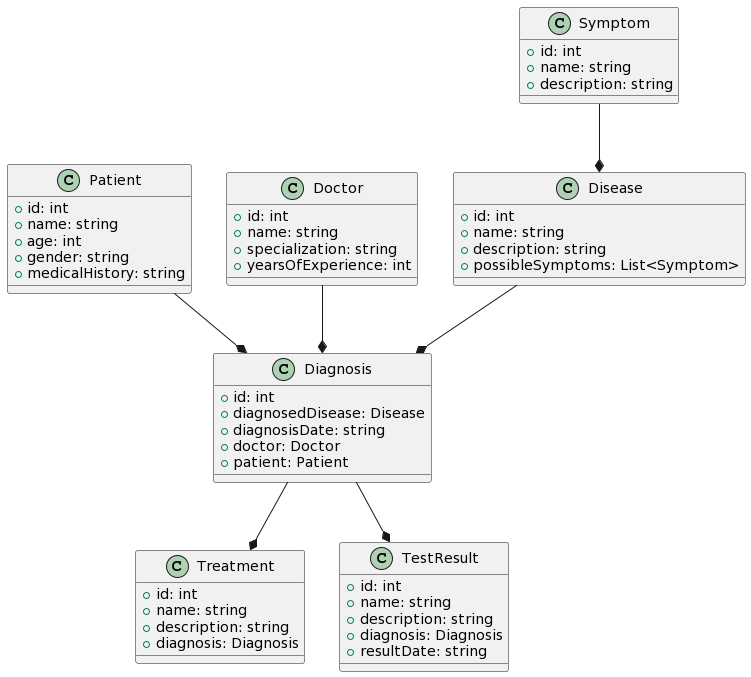
**Comment:** If the external component has already been documented, then the interface requirements can be referred to in the document. This section can be omitted if the system has no interface to external components.

**Software Design**



**Class Diagram and Class Specifications**

The class diagram for a medical disease diagnostic tool would depend on the specific requirements and functionalities of the tool. It could potentially include classes such as "Patient", "Doctor", "Symptom", "Disease", "Diagnosis", "Treatment", "Test Result", etc. The classes would have attributes and methods to represent the data and behavior of each object and would have associations to model the relationships between objects. The exact details of the class diagram would need to be determined based on the specific requirements and design considerations for the medical disease diagnostic tool.



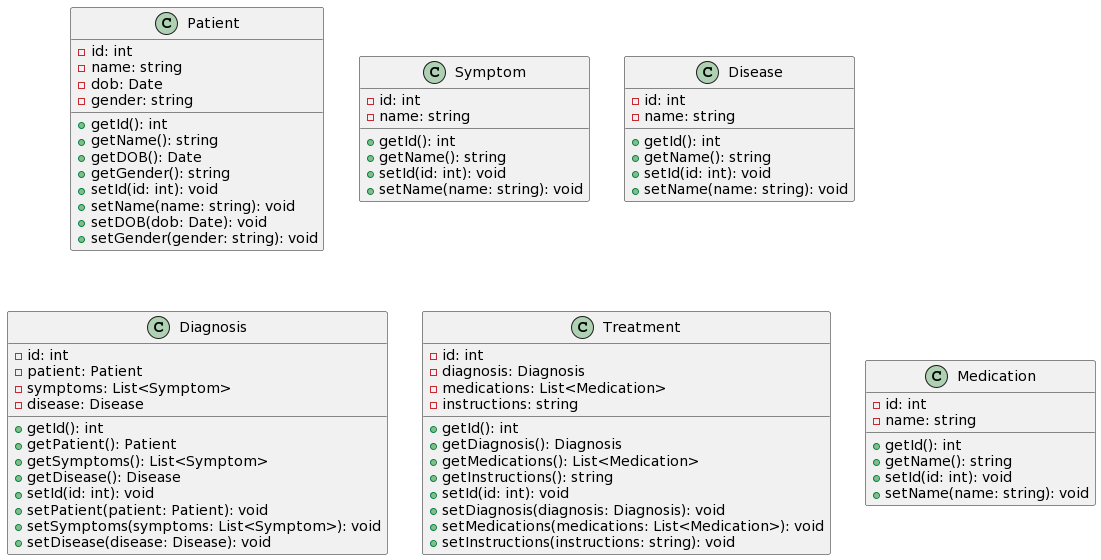
**Interaction Diagrams**

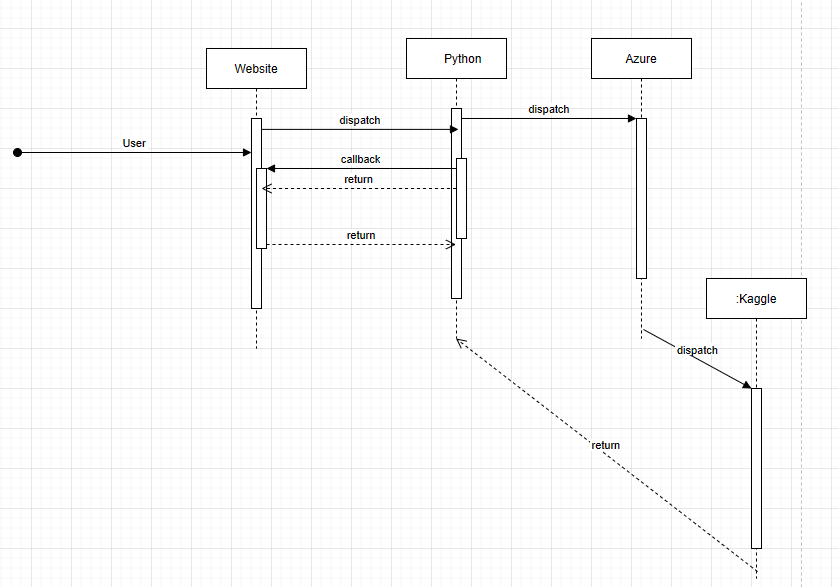
A medical disease diagnostic tool would typically work by allowing users, such as patients, to input their symptoms and personal information into the system. The tool would then process this information and use algorithms based on machine learning and image processing to make a diagnosis. The diagnosis would be based on patterns and correlations in medical data and images, such as X-rays or MRIs, stored in a database.

The user interface for the tool would be designed to be simple and intuitive, allowing users to easily input their information and receive a diagnosis. The interface could be web-based or a standalone application and could include features such as a symptom checker, appointment scheduling, and medical history tracking.

The tool would communicate with external systems, such as database management systems, to access and process medical images and information. It would also interface with external software services to develop and train the algorithms used to make diagnoses.

The tool would be hosted on a server and the medical images and algorithms would be stored in a storage component. The server would process the images and predictions, allowing the tool to provide quick and accurate diagnoses to users.

****



**Design Considerations**

Design considerations for a medical disease diagnostic tool would include the following:

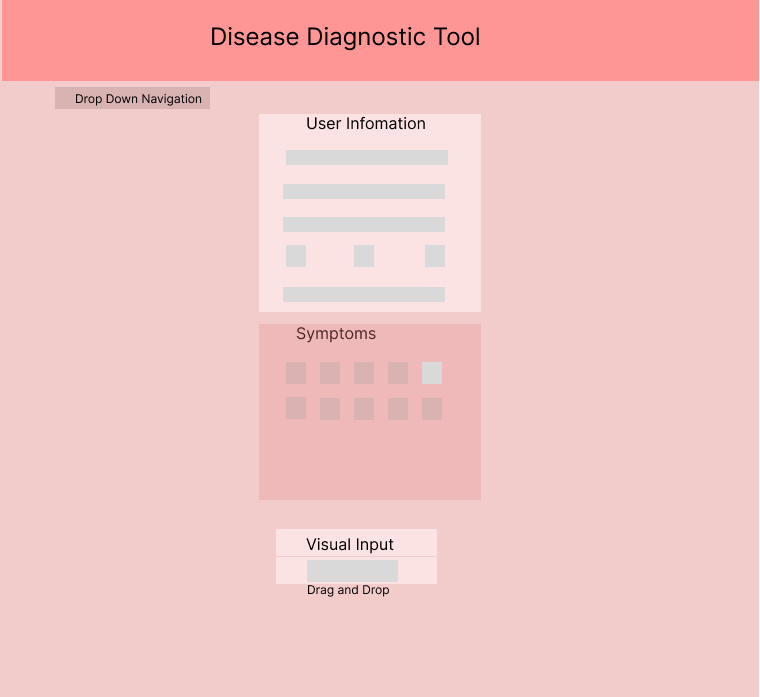
1. User-centered Design: The tool should be designed with the end-user in mind, taking into account their experience, needs, and expectations.
2. Data Privacy and Security: The tool should ensure the privacy and security of sensitive patient data, such as medical history and test results.
3. Reliable Diagnostic Algorithms: The diagnostic algorithms should be based on current best practices and medical guidelines and should be validated and tested for accuracy.
4. Ease of Use: The tool should be easy to use, with a user-friendly interface and clear, concise instructions.
5. Integration with Electronic Health Records (EHRs): The tool should be able to integrate with existing EHR systems to facilitate data sharing and analysis.
6. Compliance with Regulatory Standards: The tool should comply with relevant regulatory standards, such as HIPAA and GDPR.
7. Scalability: The tool should be designed to handle a large volume of patients and data, and should be scalable to accommodate future growth.

**User Interface Design**

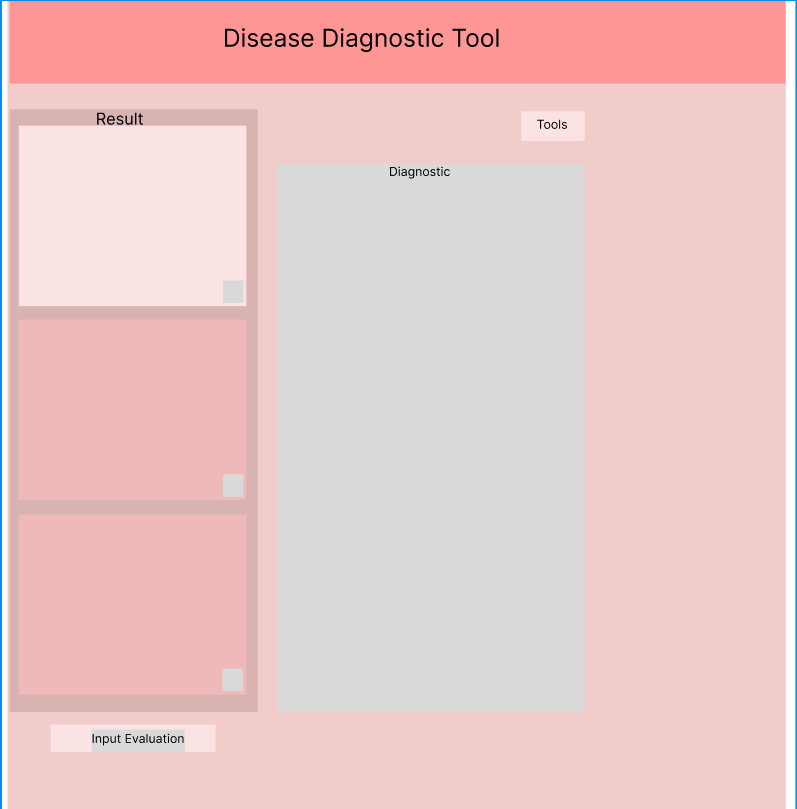
* Registration and login interface: This interface lets users create an account and log in to the system. It should have places where you can put your name, age, gender, email address, and password.
* Patient information input form: This form is used to enter information about a patient, such as their medical history, symptoms, and other relevant details that will be used to predict the likelihood of disease. It should be simple and easy to use.
* Displaying the prediction results: Once the system has looked at the patient data, it should show the prediction results in a way that is easy to understand. This could include a graphical display of the predicted risk of disease, as well as a list of recommendations for further testing or treatment.
* Patient medical record interface: This interface lets medical professionals who are allowed to do so access and update the medical records of patients. It should have functions like search and filtering, and you should be able to add, edit, and delete records.
* Dashboard and reporting interface: This interface should give an overview of system performance and key metrics, such as the number of patients processed, accuracy rates, and other relevant statistics. It should also give reports on how patients did and other measures of performance.

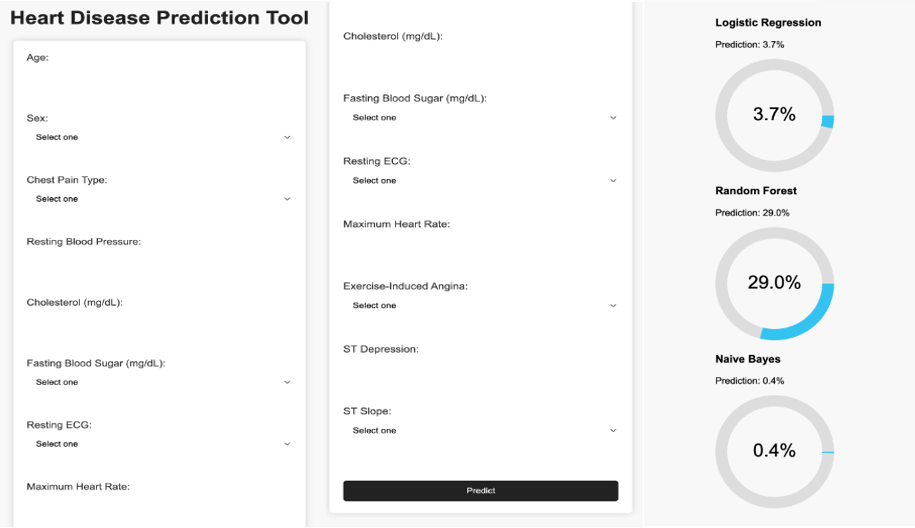
The sketches or mock-ups of the major UI screens should include a visual representation of the user interface elements, such as buttons, forms, and displays, and the layout of the UI elements. This will help to guide the development of the code and ensure that the final product meets the intended user requirements.

1. Center UI with patient data information and symptoms.



B) Display of Result and Evaluation with Tool to navigate





**Unit Tests for Electron Application:**

For unit tests in Electron, you can use a testing framework like Spectron or Web-Driver-IO. These frameworks provide an easy-to-use API for interacting with your application and automating tests.

**Data Process and How it Works:**

For the heart disease prediction model, the data must be sanitized and transformed to be between zero and one. This involves filling any null or unknown data with the data sets median. The machine learning model uses a dense layer and a sigmoid layer to predict the likelihood of heart disease based on the input data.

**Usability Testing Plan for Heart Disease Prediction Model:**

**Introduction:**

The purpose of this usability test is to evaluate the user interface and experience of the heart disease prediction model. The goal is to gather feedback from users to improve the model and provide better service to patients.

**Participants:**

Participants must be at least 18 years old and have familiarity with using web or desktop applications.

**Test Cases:**

* Predicting Heart Disease: Participants will enter their personal information into the model and interpret the output to understand their likelihood of heart disease.
* Input Validation: Participants will intentionally enter incorrect or invalid data to see how the application handles the input.
* User Interface and Design: Participants will provide feedback on the overall design and layout of the model, including colors, font, size, and placement of elements.

**Data Analysis:**

Feedback gathered during testing will be analyzed to identify common themes and areas for improvement. Results of the usability testing will be used to make updates to the heart disease prediction model to improve user-friendliness and effectiveness.

**Limitations:**

It's important to ensure compliance with all laws and regulations around data privacy and protection when working with patient data. Anonymized or synthetic data may be used for testing purposes to protect patient privacy.

**Electron application process and how it works:**

As part of our application development process, we need to ensure that our electron application for heart disease prediction is thoroughly tested to ensure that it is functioning as expected. We need to add unit tests to test the communication between the data input, Python code, and clean data. This report outlines the steps we will take to add these unit tests.

**Step 1: Create unit tests for Python functions.**

The first step is to create unit tests for the Python functions that handle the data processing. This involves using a testing framework like unit-test or py-test to write and run tests for each function. The tests should cover all possible scenarios, including edge cases and invalid input, to ensure that the functions are working as expected.

**Step 2: Test the HTML form.**

The next step is to create unit tests for the HTML form that collects user input. This involves using a JavaScript testing framework like Jest or Mocha to write and run tests for the form. The tests should cover all possible scenarios, including edge cases and invalid input, to ensure that the form is working as expected.

**Step 3: Test the communication between Python and HTML**

The third step is to test the communication between Python and HTML. This involves using a library like Selenium to simulate user interactions with the HTML form and check that the correct data is sent to the Python code. Additionally, a Python testing framework like py-test can be used to test communication from the Python side. These tests should ensure that data is correctly passed between the HTML form and Python code, and that any errors or issues are caught early in the development process.

**Step 4: Test the clean data.**

Finally, unit tests should be created to ensure that the data is cleaned properly and that there are no null or unknown values in the data. These tests can be written using a Python testing framework and should cover all possible scenarios, including edge cases and invalid input, to ensure that the data is clean and ready for use in the machine learning model.

**Conclusion:**

By following these steps, we can ensure that our electron application for heart disease prediction is thoroughly tested and free of errors. The unit tests will cover all aspects of data processing, communication, and clean data, and will help us catch any issues early in the development process. This will allow us to deliver a high-quality, reliable application to our users.

**Tkinter Application:**

Graphical user interface, application

Description automatically generated

**Unite Tests:**

TensorFlow is a machine learning library for python.

In our machine learning model, we also use the Keras library. Because of its high-level API which makes it easy to create complex neural networks in our model we have two layers:

**Dense One:**

We have one dense layer which is where the machine decides what is important to predict the variable, we want to predict which in our case is heart disease.

**Sigmoid Layer:**

The second layer is the sigmoid layer which uses the sigmoid activation function to output a probability between zero and one. Which is good for our project because we want the patients to know how likely they are to have heart disease.

**Data Process and How it works:**

The data we feed to the neural network must be sanitized which means we fill any null or unknown data with the data sets median and transform all data to be between zero and one.

**Usability Testing Plan for Heart Disease Prediction Model**

**Introduction:**

The purpose of this usability test is to evaluate the user interface and user experience of the TensorFlow TKinter Application Model for heart disease prediction. We want to gather feedback from users on the design, functionality, and overall usability of the model. This feedback will be used to improve the model and provide better service to patients.

**Participants:**

**Participants Must fit the following criteria:**

Must be at least 18 years old.

Must have some familiarity with using web or desktop applications.

**Test Cases:**

Our testing involved three specific test cases:

Predicting heart disease: Participants will be asked to enter their personal information, such as age, gender, blood pressure, and cholesterol levels, into the model. They will then be asked to interpret the output and understand how likely they are to have heart disease. We will observe and gather feedback on how easy it was for the participants to use the model to make a heart disease prediction.

Input validation: Participants will be asked to intentionally enter incorrect or invalid data into the model and observe how the application handles the input. We will gather feedback on whether the error messages were clear and helpful in guiding the user to correct their input.

User interface and design: Participants will be asked to provide feedback on the overall design and layout of the model. We will gather feedback on the colors, font, size, and placement of elements. We will also ask participants to suggest any improvements that they would like to see in the user interface.

**Procedure:**

Participants will be provided with a brief introduction to the model and given a consent form to sign.

Each participant will be asked to complete the three test cases outlined above.

During the testing, participants will be encouraged to think aloud and provide feedback on their experience with the model.

After each test case, participants will be asked a series of follow-up questions to gather more detailed feedback.

Once all test cases have been completed, participants will be asked to complete a post-test survey to provide additional feedback on their overall experience with the model.

**Data Analysis:**

The feedback gathered during the testing will be analyzed to identify common themes and areas for improvement. We will also look for any patterns or trends in the data to inform our decision-making on how to improve the model. The results of the usability testing will be used to make updates to the TensorFlow TKinter Application Model, with the goal of providing a more user-friendly and effective tool for predicting heart disease.

**Glossary of Terms**

A glossary of terms can be a helpful tool in a System Design Document to provide a common understanding of key terms and concepts used throughout the document. This helps to ensure consistency and avoid ambiguity in the system specification. Some examples of terms that could be included in a glossary for our medical disease diagnostic tool are:

* Medical disease prediction system: A software program that uses machine learning algorithms to figure out how likely it is that a person will get a certain disease.
* Dataset: A group of data used for analysis and training machine learning models.
* Machine learning is a field of artificial intelligence that involves making algorithms and models that can learn from data and make predictions or decisions.
* Feature engineering: the process of choosing and changing data features to make machine learning algorithms work better.
* Model evaluation: the process of judging how well a machine learning model works on a certain dataset.
* Precision: the number of true positives divided by the total of true positives and false positives in a binary classification problem.
* F1 score: a way to measure how accurate a model is by taking both precision and recall into account.

**References**

The list of references should contain exact references and/or URLs of any material that is cited in the analysis document. The references should be formatted consistently using ACM, IEEE or APA style. Do not mix citation styles. The following sites may be helpful for formatting your references.

[Kaggle - Wikipedia](https://en.wikipedia.org/wiki/Kaggle)

[Kaggle: Your Machine Learning and Data Science Community](https://www.kaggle.com/)

[Kaggle Data Science Projects for Beginners | Another Techs](https://anothertechs.com/programming/machine-learning/kaggle-data-science-projects/)

Zhang, J., Cai, Y., Zhang, M., & Wei, Y. (2021). Medical disease prediction based on machine learning algorithms: a systematic review. BMC Medical Informatics and Decision Making, 2021..

Lee, J. W., Yoon, J. H., & Ryu, K. H. (2019). A machine learning-based medical disease prediction system using genetic data. International Journal of Medical Informatics, 2019.

Karim, M. E., Rahman, M. S., & Rahman, M. M. (2021). Machine learning-based disease prediction using electronic health records: a review. Journal of Healthcare Engineering, 2021.

• Pacific Library citation tools, includes **APA** citation style:

<http://www.pacific.edu/Library/Get-Help/Citation-Tools.html>

• **ACM** citation style:

<http://www.acm.org/publications/article-templates/acm-latex-style-guide>

• IEEE citation style:

<http://www.ieee.org/documents/ieeecitationref.pdf>

• Examples in **ACM, APA** and **IEEE** styles:

<http://dal.ca.libguides.com/content.php?pid=860&sid=11>