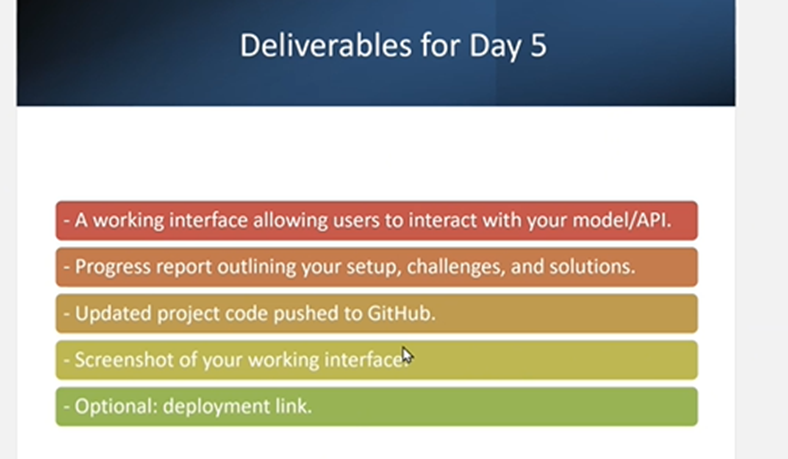
**Day5. CardioGuard.AI - Integrations and UI**

We made heart disease prediction model and APIs,

We are taking static CBC parameters frequently to better predict the cardio risk and personalization, today job is the UI/UK to trigger the API\s and provide the warning and risk , we have tested multiple models and kept one model for testing by a web app.

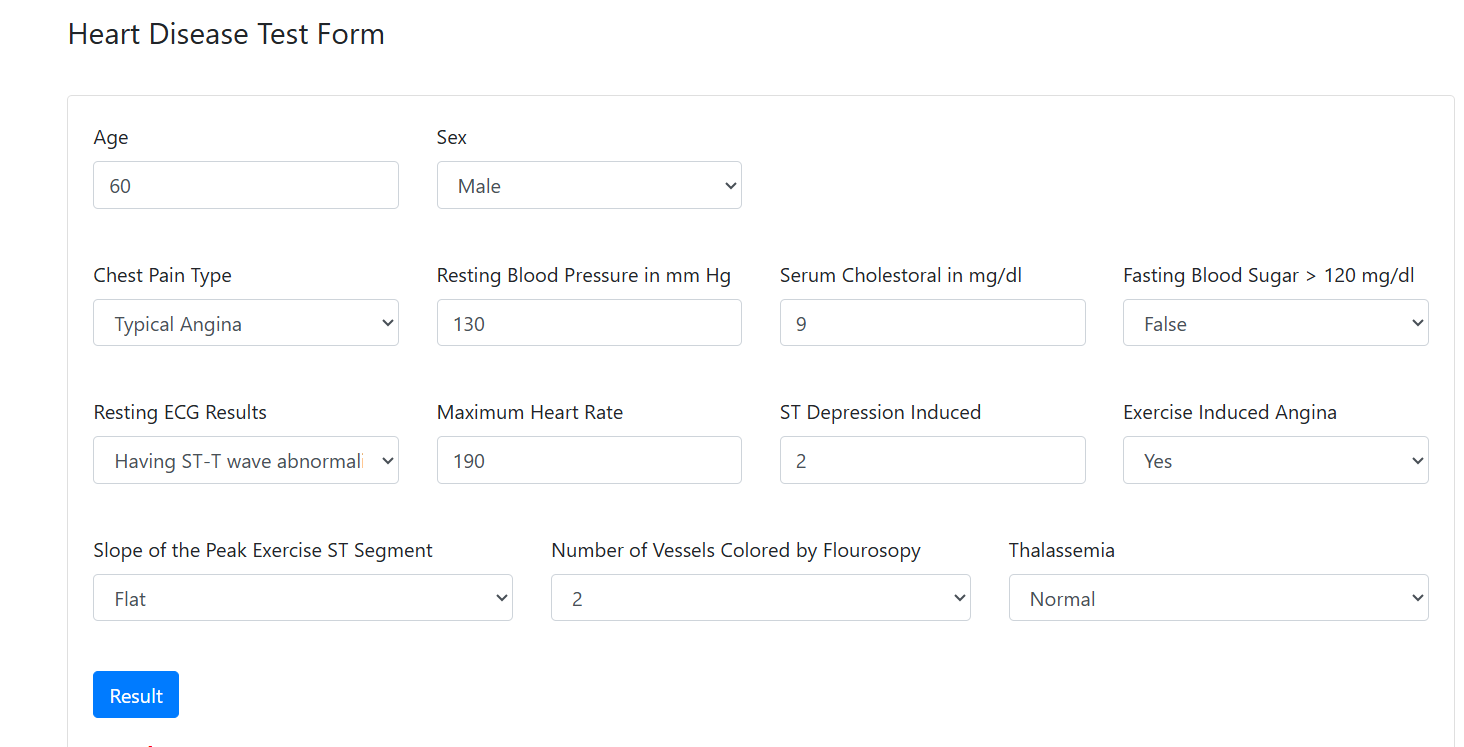
Today test UI and results from model ..this will later be integrated to device and based on the data API will trigger to get the risk which then alert the patient.



**Deliverables:**

1. A working interface that allows users to interact with your model/API, including input fields and displaying results.
2. Progress Report:
3. Describe your interface setup and how it interacts with the backend API/model.
4. Outline any challenges faced and how they were resolved.
5. Updated Project Code: Push your latest project code to GitHub, including interface and backend integration.
6. Screenshot of the Interface: Capture the final interface with an example of user input and the model's output.
7. Optional: If deployed, share the link to your web app.
8. This checkpoint will help participants focus on user-facing features of their AI application, ensuring a polished and interactive interface.

CardioGuard.AI 



**You are at RISK**

**Data Set**

The Heart Disease dataset parameters, crucial for the ML model's reliability, encompass a range of patient health indicators: age (years), sex (male/female), chest pain type (typical angina, atypical angina, non-anginal pain, asymptomatic), resting blood pressure (mm Hg), serum cholesterol (mg/dl), fasting blood sugar (>120 mg/dl), resting electrocardiographic results (normal, ST-T wave abnormality, left ventricular hypertrophy), maximum heart rate achieved, ST depression induced by exercise relative to rest, slope of the peak exercise ST segment, number of major vessels colored by fluoroscopy, and thalassemia, all of which must be consistently formatted and accurately reflect the patient's condition**.**

**2 sets of data sets tried -simulated and other from kaggle**

Steps

** Loading the Data: Reading the dataset into a pandas DataFrame.**

** Exploratory Data Analysis (EDA): Understanding the data through visualizations and statistical summaries.**

** Data Preprocessing: Handling missing values, encoding categorical features, scaling numerical features.**

** Feature Selection/Engineering: Selecting the most relevant features or creating new ones.**

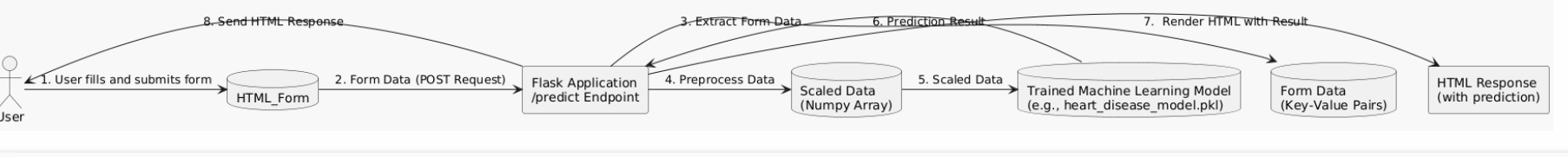
** Model Selection: Choosing appropriate classification algorithms.**

** Model Training: Training the chosen model on the training data.**

 **Model Evaluation:** Assessing the model's performance on the testing data using relevant metrics (accuracy, precision, recall, F1-score, AUC, etc.).

 **Model Tuning** Optimizing the model's hyperparameters to improve performance

Testing the webapp /Artcjitecture



**API Documentation: Heart Disease Prediction**

This document describes the API for predicting the likelihood of heart disease based on user-provided health data. The API is built using the Flask framework in Python.

**1. Overview**

The API provides a single endpoint, /predict, which accepts patient health information via an HTML form submission (POST request). It uses a pre-trained machine learning model to predict the probability of heart disease and returns the result to the user.

**2. Endpoint: /predict**

* **Method:** POST
* **Description:** This endpoint receives patient health data from an HTML form, preprocesses the data, uses a pre-trained machine learning model to predict the likelihood of heart disease, and returns the prediction result.
* **URL:** /predict

**3. Request Parameters**

The /predict endpoint expects the following parameters, submitted as form data in a POST request. These parameters correspond to the input fields in the heart\_disease\_form.html template. All parameters are **required**.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Data Type** | **Description** |
| age | integer | Patient's age in years. |
| sex | integer | Patient's sex (0: Male, 1: Female). |
| cp | integer | Chest pain type (0: Typical Angina, 1: Atypical Angina, 2: Non-anginal Pain, 3: Asymptomatic). |
| trestbps | integer | Resting blood pressure in mm Hg upon admission to the hospital. |
| chol | integer | Serum cholesterol in mg/dl. |
| fbs | integer | Fasting blood sugar > 120 mg/dl (0: False, 1: True). |
| restecg | integer | Resting electrocardiographic results (0: Normal, 1: Having ST-T wave abnormality, 2: Probable or definite left ventricular hypertrophy). |
| thalach | integer | Maximum heart rate achieved. |
| oldpeak | float | ST depression induced by exercise relative to rest. |
| slope | integer | The slope of the peak exercise ST segment (0: Upsloping, 1: Flat, 2: Downsloping) |
| ca | integer | Number of major vessels (0-3) colored by fluoroscopy. |
| thal | integer | Thalassemia (3: Normal, 6: Fixed defect, 7: Reversible defect). |

**4. Request Example**

A typical request to the /predict endpoint would be a POST request with the following form data:

age: 55

sex: 0

cp: 0

trestbps: 130

chol: 250

fbs: 0

restecg: 1

thalach: 140

oldpeak: 2.3

slope: 2

ca: 0

thal: 1

**5. Response**

The API returns an HTML page (generated by Flask's render\_template) displaying the prediction result. The result is embedded within the heart\_disease\_form.html template.

* **Success Response (HTTP 200 OK):** The HTML page will contain a text string indicating the model's prediction:
  + "The model predicts: No Heart Disease"
  + "The model predicts: Heart Disease"
* **Error Response (HTTP 200 OK, with error message in HTML):** The HTML page may also contain an error message in the following cases:
  + "Error: Invalid input data. Please check the form and try again.": This indicates that the user provided invalid data in the form (e.g., non-numeric values where numbers were expected).
  + "Error: Machine learning model not loaded.": This indicates that the heart\_disease\_model.pkl file was not found or could not be loaded.

**6. Error Handling**

The API includes basic error handling:

* **Invalid Input Data:** The preprocess\_data function attempts to convert the form data to the correct data types. If this conversion fails (e.g., due to non-numeric input), it returns an error message to the user in the HTML response.
* **Model Loading Failure:** The API checks if the machine learning model file (heart\_disease\_model.pkl) is loaded successfully. If the file is not found, it returns an error message to the user.

**7. Dependencies**

The API requires the following Python libraries:

* Flask
* scikit-learn
* numpy

These dependencies should be installed before running the application (e.g., using pip install Flask scikit-learn numpy).

Challenges

 **Data Preprocessing Consistency:** The diagram shows that the Flask\_App is responsible for preprocessing the data, including scaling. A challenge here is ensuring that the scaling applied to the user-provided data is consistent with the scaling applied to the training data used to train the ML\_Model. Any discrepancy in preprocessing can lead to inaccurate predictions.

 **Model Accuracy and Generalization:** The effectiveness of the API depends on the accuracy and generalization ability of the pre-trained ML\_Model. If the model is not well-trained or if it overfits the training data, it may not perform well on new, unseen data provided by users.

3.Simulator for band is happening in parallel , our custom band hardware is in manufacturing so using simulator