CSL2060 Project

Group 44

About the Topic

Topic: An app to detect the chances of a cardiovascular disease in humans

We have designed an app that takes the health parameters of the user as input, and then predicts the chance of a cardiovascular disease occurring.

An ML model has been trained using existing datasets on the same topic, and based on it, risk assessment has been done.

The app also includes a few additional features, which are mentioned in the subsequent sections.

The workflow, and other additional features, can best be visualized through these 5 UML Diagrams.

Software Requirements Specification Document

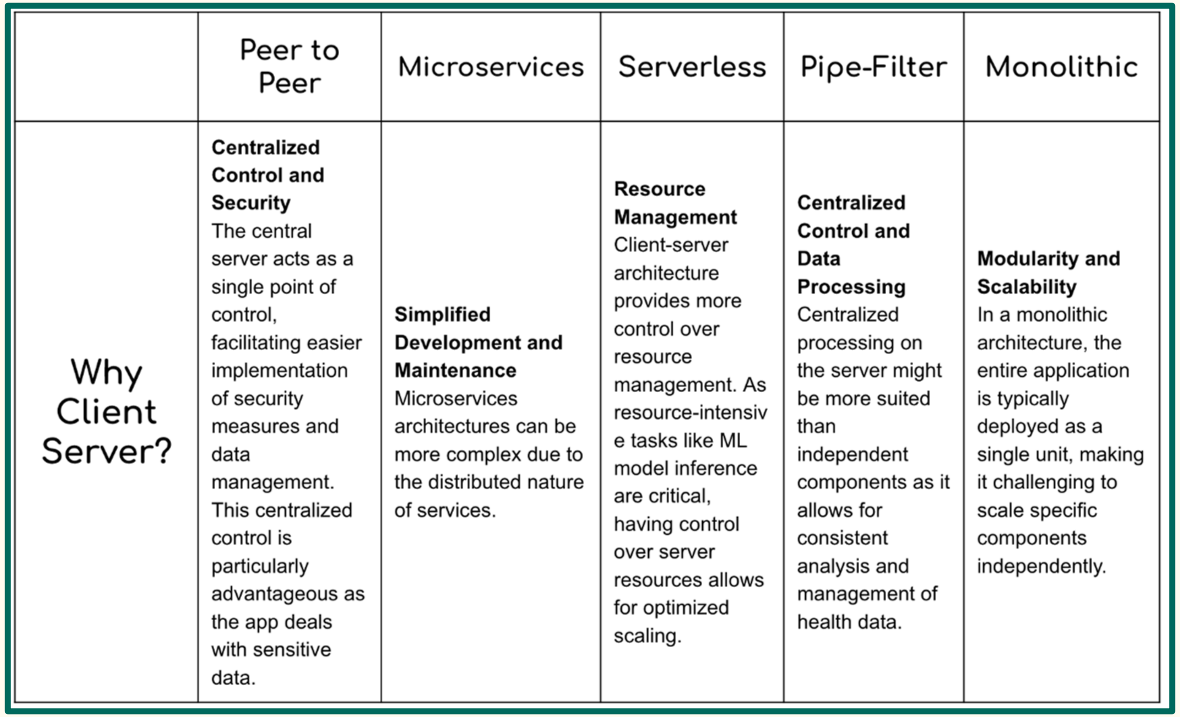
We have made 4 versions of SRS document through the course of this project.

Changes made from draft 3 to draft 4:

1. Modification in the overall architecture style: User's information shall be stored locally and not on the central server to enhance security. This implies the usage of serverless architecture for this purpose, while the rest of the application proceeds with client-server architecture, as had been discussed earlier. This change is reflected in point (2.1.3).
2. Addition of the option to update profile information, as is reflected in the application (2.2.2)
3. Utilization of Flask for ML model is now reflected (2.4.4).
4. Usage of SQLite database is now mentioned (2.7.2).
5. Addition of scalability as a non-functional requirement (3.2).
6. Addition of security as a non-functional requirement (3.4).
7. Updates in the legal considerations as the application will be directly related to the health of personnels (4.1.2).

Software Architecture

Our initial idea was to use **client-server architecture.**



	Peer to Peer	Microservices	Serverless	Pipe-Filter	Monolithic
Why Client Server?	**Centralized Control and Security** The central server acts as a single point of control, facilitating easier implementation of security measures and data management. This centralized control is particularly advantageous as the app deals with sensitive data.	**Simplified Development and Maintenance** Microservices architectures can be more complex due to the distributed nature of services.	**Resource Management** Client-server architecture provides more control over resource management. As resource-intensiv e tasks like ML model inference are critical, having control over server resources allows for optimized scaling.	**Centralized Control and Data Processing** Centralized processing on the server might be more suited than independent components as it allows for consistent analysis and management of health data.	**Modularity and Scalability** In a monolithic architecture, the entire application is typically deployed as a single unit, making it challenging to scale specific components independently.

Modified Architecture

User’s information shall be stored locally and not on the central server. This implies the usage of serverless architecture for this purpose, while the rest of the application proceeds with client-server architecture, in a **hybrid architecture model**.

Reasons for this change:

1. Data privacy
2. Reduced latency
3. Offline access

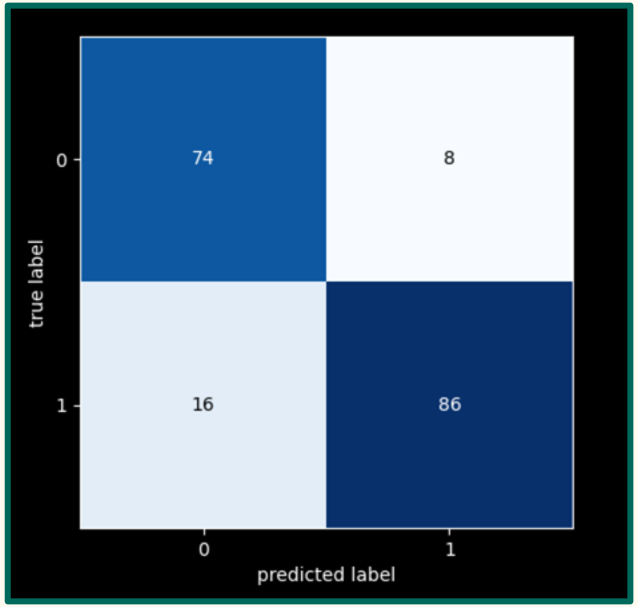
ML Model

Basic approach: The model was refined with the aim of **increasing the recall score** as far as possible because given that this app deals with health issues, we need to minimise the cases of false negatives.

The dataset can be viewed here.

Final Accuracy: 87%

Final ML Model



Confusion Matrix

Basic Functionality

- The trained classifier is saved and accessed in **Flask**, using which we build an API which returns a dictionary of key-value pair, where the key is ‘output’ and the value is probability for developing cardiovascular disease by using the saved pre-trained classifier.
- Profiles get saved on local storage to prevent the risk of theft/leakage of personal data. Storage is done on **SQLite Database**.
- The table of data in SQLite can be viewed in a compact manner by tapping ‘saved profiles’ button.

UI Diagrams

Original UI diagrams that we had designed can be viewed here.

However, we did not get sufficient time to format the front end exactly as these images.

Implementation

Project Management

This document contains the following details:

1. Risk analysis and risk assessment matrix
2. Gantt charts (3 versions, the final one being updated to date)
3. Cost analysis (from the perspective of an organisation that desires to pursue this project)
4. Task dependencies and critical path

Software Versioning

Version 1.0.0: Basic ML model integration with the interface, giving prediction based on input data.

Version 1.1.0: Addition of features such as data visualization and generic health recommendations.

Version 1.1.1: Using unit testing to fix the bug in updation of details in saved profiles.

Version 2.0.0: Improved user interface based on feedback and enhanced frontend aesthetics.

Version 2.1.0: Addition of more graphs for better visualisation; limiting the input parameters to specific ranges for better accuracy.

Version 2.1.1: Fixed the bug of screen updating incorrectly after the details are edited.

Software Metrics

COCOMO: We would define our project in the category **Semi-detached**. So

Effort level:Medium, Time given: Medium

Cyclomatic Complexity: approximately 12-13 modules are being integrated, so moderate risk.

LOC: approximately 2300 LOC for development ; 400 LOC for ML model

Coupling: Low coupling, so good cohesion

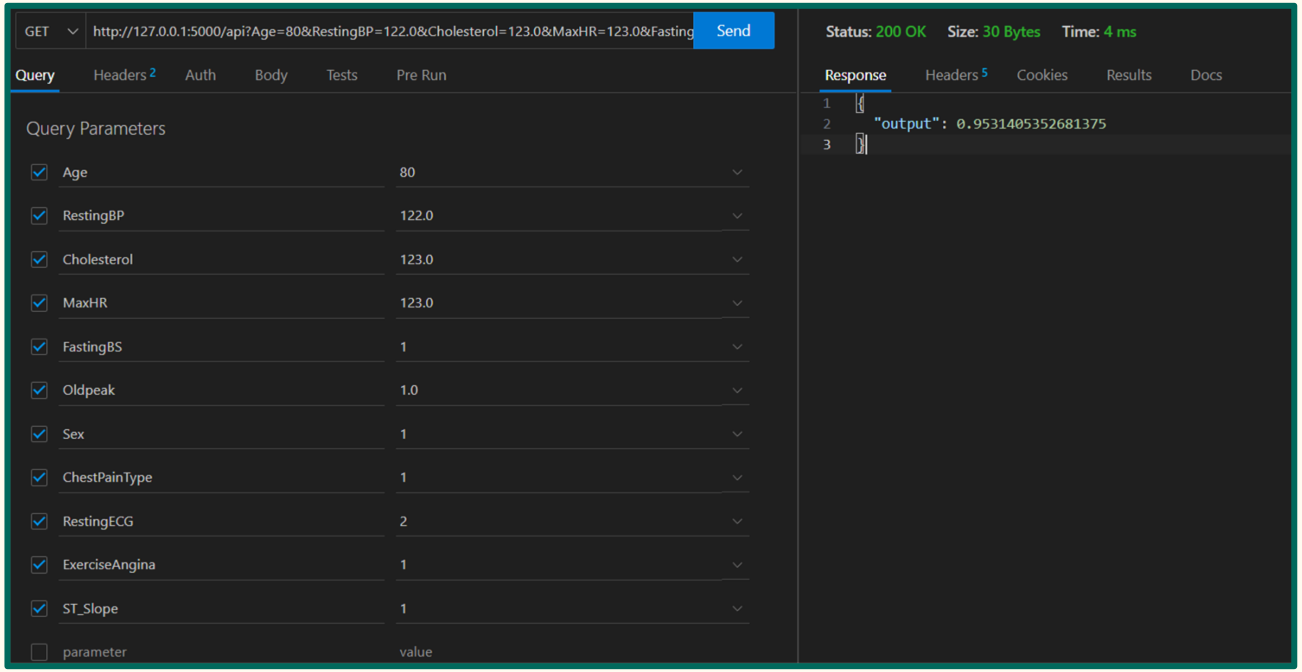
Testing

1. Unit Testing:
    a. We used this primarily for the app interface.
    b. We were able to detect bugs such as incorrect updation of details post changes in input parameters, screen not being refreshed precisely, etc.

1. ML Model Testing:
    a. Tested the ML model with absurd values to check the response of the model and determine important parameters.
    b. Eg: We found that age is an important feature of the dataset as the predicted label was reducing tremendously on modifying just the age.

API Testing

We used Thunder Client extension of VS Code to check the performance of our locally created API.



GET http://127.0.0.1:5000/api?Age=80&RestingBP=122.0&Cholesterol=123.0&MaxHR=123.0&Fasting

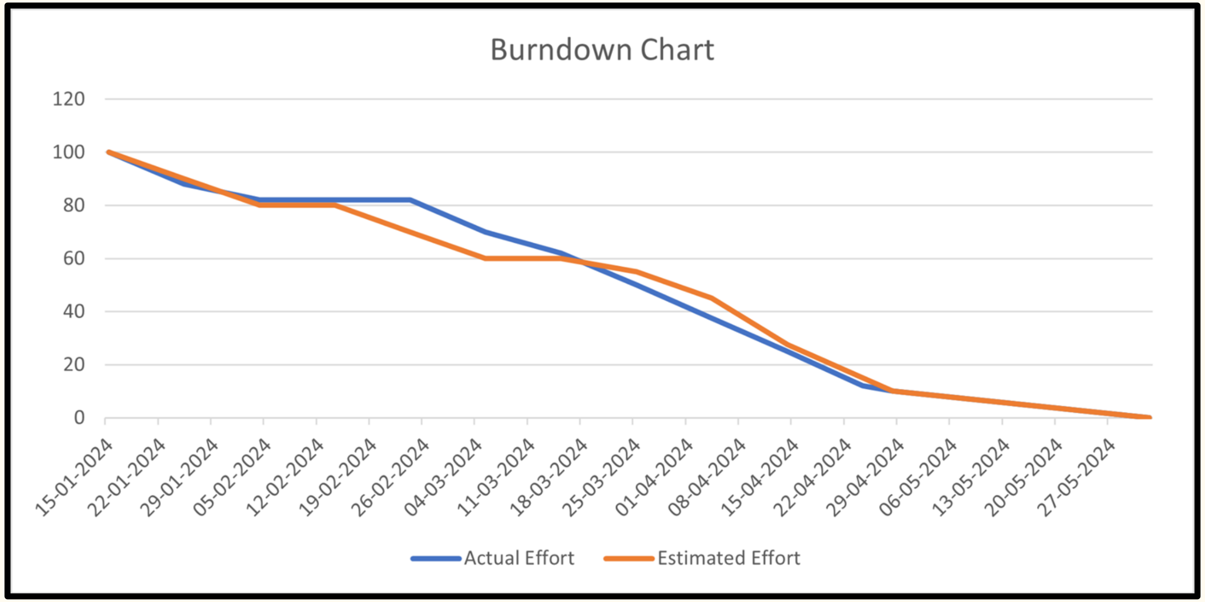
Query Parameters

✓	Age	80
✓	RestingBP	122.0
✓	Cholesterol	123.0
✓	MaxHR	123.0
✓	FastingBS	1
✓	Oldpeak	1.0
✓	Sex	1
✓	ChestPainType	1
✓	RestingECG	2
✓	ExerciseAngina	1
✓	ST_Slope	1
	parameter	value

```
{
  "output": 0.9531405352681375
}
```

Progress Trajectory

1. Gantt Chart



Real Life Applications

1. Our project was designed with the hope to empower individuals to take proactive steps towards their heart health by identifying potential risks early.
2. Given that most users may not be familiar with the health parameters required to be given as inputs, this software can be used in medical setups which the users can access after taking blood tests.
3. Furthermore, the app can also be used by working medical professionals as a tool to assist them in making informed decisions and prioritizing patients for further evaluation or intervention.
4. Lastly, we intend to use additional data available in the medical world to train our model further at the end of regular timespans.

Challenges Faced	Solutions
ML Model Selection and Refinement	Explored a range of classifiers, and finally selected the one that offered a nice combination of accuracy and recall.
Data Privacy and Security	Implemented a hybrid SW architecture to ensure that the personal details of users are not susceptible to thefts/leakages.
Model and Database Integration	Created a local server which calls the API
Interpretability of Prediction Results	Provided graphical visualizations of the outputs generated to aid in the understanding of the users
Regulatory Obligations	Did not provide user-specific health recommendations as that would require legal permissions; instead provided generic guidelines for users suffering from such ailments.

Thank You!