HeartScope

"Fighting Back Against America's #1 Health Crisis"

Created by Nicholas Farkash, Bryan Mejia, Adrian Tapia

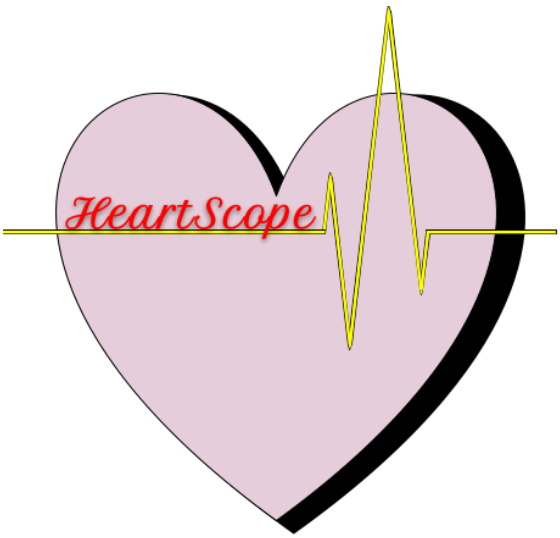


Table of Contents

Introduction - pg 2
Glossary - pg 3
User Requirements Definition - pg 6
System Requirements Specification - pg 7
Architectural Design -pg 10
Structural Modeling - pg 12
Interaction Modeling - pg 13
Programming Environment - pg 15
Data Description - pg 16
Test Cases -pg 18
Testing Results - pg 27
User Guide - pg 31
Summary of Complete and Incomplete Work - - - - - - - - - - - - - - - - - - pg 33
Planning for Phase II - pg 34
GitHub Repository -pg 35

Introduction

Cardiovascular disease is the leading cause of death in the United States and has been for over a century. In 2023, over 680,000 Americans (over 22%) died from cardiovascular disease. Many of these people are not even aware that they are at risk, nor are they aware of the individual factors that are contributing to their increased risk.

In light of this, we have created HeartScope. Our HIPAA-compliant prediction software utilizes data from previous real-world studies to create a predictor that analyzes user-provided data to determine whether the user has cardiovascular disease. Each user will be asked a series of questions and prompted to enter or select a response. Their answers will be analyzed using the predictor to produce personalized feedback tailored to their specific situation. The user will receive suggestions as to which attributes they can improve upon to more closely align with a person who is not at risk for cardiovascular disease.

HeartScope intends to detect cardiovascular disease early to prevent further progression, allowing our users to live happy and healthy lives.

Glossary

- **Access Control Testing** - a security process to ensure that all data is accessible only by those who should have access to it, such as administrators, whereas users are exposed to limited information.

- **Bootstrapping** - the process of selecting random subsets of data to create new data points to be used in training and testing a Random Forest.

- **Cardiovascular Disease (CD)** - a group of diseases affecting the cardiovascular system, including heart disease, stroke, heart failure, and atrial fibrillation.

- **Classification System/Classifier** - *see Random Forest.*

- **Decision Tree** - a structure composed of a series of nodes that are used to classify data into groups.

- **HeartScope** - the name of our software, which will process a user's data and use it to predict the user's likelihood of suffering from cardiovascular disease. It will also provide the user with information customized to their specific situation to help them prevent cardiovascular disease.

- **HIPAA (Health Insurance Portability and Accountability Act)** - a privacy law that protects patient's privacy in medical/health environments.

- **Random Forest** - a machine learning algorithm that combines the outputs of multiple decision trees to produce a single result.

The following are descriptions of the attributes present in the dataset:

- **Age** - the number of years the patient has lived.

- **Sex** - the biological sex of the patient. (Categorical)
    - M: Male
    - F: Female

- **ChestPainType** - the type of chest pain experienced by the patient. (Categorical)
    - TA: Typical Angina - Chest pains consisting of:
        1. Substernal chest pain or discomfort that is…
        2. Provoked by exertion or emotional stress and…
        3. Relieved by rest or nitroglycerine (or both).
    - ATA: Atypical Angina - Chest pains where 2 out of the 3 criteria of typical angina are present.
    - NAP: Non-Anginal Pain - Chest pains when 1 or fewer of the criteria of typical angina are present, as well as unknown causes.
    - ASY: Asymptomatic - No chest pain is present.

- **RestingBP** - The patient's resting blood pressure, measured in $mm \cdot Hg$.

- **Cholesterol** - The patient's level of serum cholesterol in mg/dl.

- **FastingBS** - the patient's fasting blood sugar. (Categorical)
    - 1: if FastingBS > 120 mg/dl
    - 0: otherwise

- **RestingECG** - the patient's resting electrocardiogram results. (Categorical)
    - Normal: Normal
    - ST: Having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV).
    - LVH: (Left Ventricular Hypertrophy) Showing probable or definite left ventricular hypertrophy by Estes' criteria.

- **MaxHR** - the maximum heart rate, or number of times the heart beats within one minute, of the patient.

- **ExerciseAngina** - whether the patient experienced exercise-induced angina, a type of chest pain caused by reduced blood flow to the heart. (Categorical)
    - Y: Yes
    - N: No

- **Oldpeak** - a measure of how much the heart's electrical activity changes during physical exertion, which can signal how well the heart is being supplied with blood. Measured in ST [Numeric value measured in depression]

- **ST_Slope** - The slope of the peak exercise ST segment of an electrocardiogram, which is the part between two beats. (Categorical)
    - Up: upsloping
    - Flat: flat
    - Down: downsloping

- **HeartDisease** - the output class - whether the patient has heart disease or not. (Categorical)
    - 1: Heart disease
    - 0: Normal

Requirements

User Requirements Definition

1. (Functional) HeartScope shall be trained by learning from a dataset to produce a classification system.
    a. Rationale: Training on a dataset will allow HeartScope to differentiate a user with cardiovascular disease and a user without it by seeing learning from previous examples with known results.

2. (Functional) HeartScope's classifier shall be used to predict whether the user does or does not have cardiovascular disease, given the user's provided input, which includes demographics and medical test results.
    a. Rationale: Predicting whether a user has cardiovascular disease or not will inform users of their risk of suffering from serious and potentially fatal medical events. The prediction allows a user to become aware of their current health condition and potentially save the user's life.

3. (Functional) HeartScope shall provide unique information catered to each user's specific situation that will assist the user in lowering their risk of developing cardiovascular disease.
    a. Rationale: HeartScope desires to go above and beyond a simple cardiovascular disease detector. Every user is different, so every user should be treated differently. Each person who uses HeartScope should be provided with information relevant to them.

4. (Non-Functional) HeartScope shall produce an accurate prediction for the user.
    a. Rationale: HeartScope strives to be a reliable product. In order to build and keep a positive reputation, it should be as accurate as possible.

5. (Non-Functional) HeartScope shall ensure that the user's data is kept private and will function as a one-time-use value for providing the user with their results.
    a. Rationale: Privacy is important - the user should not be afraid to use HeartScope out of fear that their data will be used or sold to a third-party. Furthermore, HeartScope will operate in compliance with HIPAA laws.

6. (Non-Functional) HeartScope shall be able to be utilized and have results returned quickly.
    a. Rationale: The user's time is valuable and should not be wasted. Especially when dealing with cardiovascular disease, time is of the essence. Therefore, HeartScope shall be able to be used in a timely manner.

System Requirements Specification

1.

Description	Utilizes training on a dataset to produce a Random Forest
Inputs	Dataset
Source	Fedesoriano's "Heart Failure Prediction Dataset" from Kaggle
Outputs	A Random Forest
Destination	Code
Precondition	The dataset used should contain complete entries with relevant data.
Algorithm	Prior to training, the modal value for each feature is calculated and stored. Null values will be handled by replacing the empty entries with the modal value for that feature. Feature projection will not be utilized. The data will be split in a 8:2 ratio and used to produce training and testing datasets, respectively. Bootstrapping shall be done by selecting 50 random data points from the dataset with replacement. The bootstrapped data will be used to create a fully grown decision tree. A collection of 50 decision trees will be used to create the Random Forest.
Postcondition	The resulting Random Forest

2.

Description	Utilizes a random forest to accurately categorize a user's data into one of two categories; has cardiovascular disease or does not have cardiovascular disease
Inputs	A Random Forest
Source	The training algorithm described in System Requirement #1
Outputs	0 if the algorithm predicts the user does not have cardiovascular disease, and 1 if the algorithm predicts the user does have cardiovascular disease
Destination	Code
Precondition	The Random Forest used shall be the result of training performed on the dataset.
Algorithm	The user's data shall be passed into each decision tree in the Random Forest. Each decision tree will process the user's data, resulting in an output of 0 or 1. The results from all decision trees in the Random Forest will be aggregated by taking whichever binary result is in the majority. That binary result shall be the prediction made by the Random Forest.
Postcondition	A binary prediction (0 or 1) associated with the user's data; a prediction of whether the user does (1) or does not (0) have CD.

3.

Description	Displays beneficial medical information related to the user's input
Inputs	User-entered data, the prediction from the Random Forest classifier
Source	User, Random Forest Algorithm
Outputs	Information personalized to the user's specific situation
Destination	Screen
Precondition	The user's input data should be complete, and the binary classification shall be calculated already.
Algorithm	The user shall be compared to the people whose information is stored in the dataset. The comparison will be against those people of the same sex and within a 2-year age difference of the user. If no such users fit the criteria, then the age factor will be dropped and the user will be compared to all same-sex users in the dataset. Average values for those users will be displayed, allowing the user to see how they compare to their same-sex-and-age peers.
Postcondition	Information displayed on the screen

4. (Nonfunctional) Accuracy: User data shall be accurately classified by the Random Forest.
    a. Measurement: The out-sample error produced when validating the RandomForest shall be below 10% (0.1). This will be calculated by comparing the RandomForest's prediction with the actual value for each entry in the testing dataset. The number of wrong predictions divided by the total number of entries in the testing dataset will produce the error percentage.

5. (Nonfunctional) Security: In compliance with HIPAA, the user's data shall never be saved or visible to any outside parties.
    a. Measurement: The webpage will utilize an automatic data clearing system that will clear the Chatbot's logs if the user does not interact with the Chatbot for 30 seconds. Furthermore, the Chatbot's logs will not display a user's information on screen, ensuring that no passersby can peek at the screen and see the user's info.

6. (Non-Functional) Speed: The user shall receive their results in 10 seconds.
    a. Measurement: This shall be verified by timing how long it takes to validate all of the samples in the testing dataset and dividing it by the time it takes, providing the average time to process one sample.

Architectural Design

There are five architectural patterns from which HeartScope could be built upon. These patterns are Model-View-Controller (MVC), Layered, Repository, Client-Server, and Pipe-And-Filter. In order for HeartScope to be utilized most effectively by both users and software engineers, each pattern was thoroughly considered as a viable option.

HeartScope does not utilize a large or online database, as one of the non-functional constraints of the product is that user data is never saved or used for any purposes other than providing the service promised to the user. Therefore, there is no reason for its architecture to be built following a Client-Server or Repository architectural pattern, as the main reason to use one of these patterns is for easy management of databases. As such, there would be no benefit to using either of these. However, there is an argument to be made for using each of MVC, Pipe-And-Filter, and Layered architectural patterns.

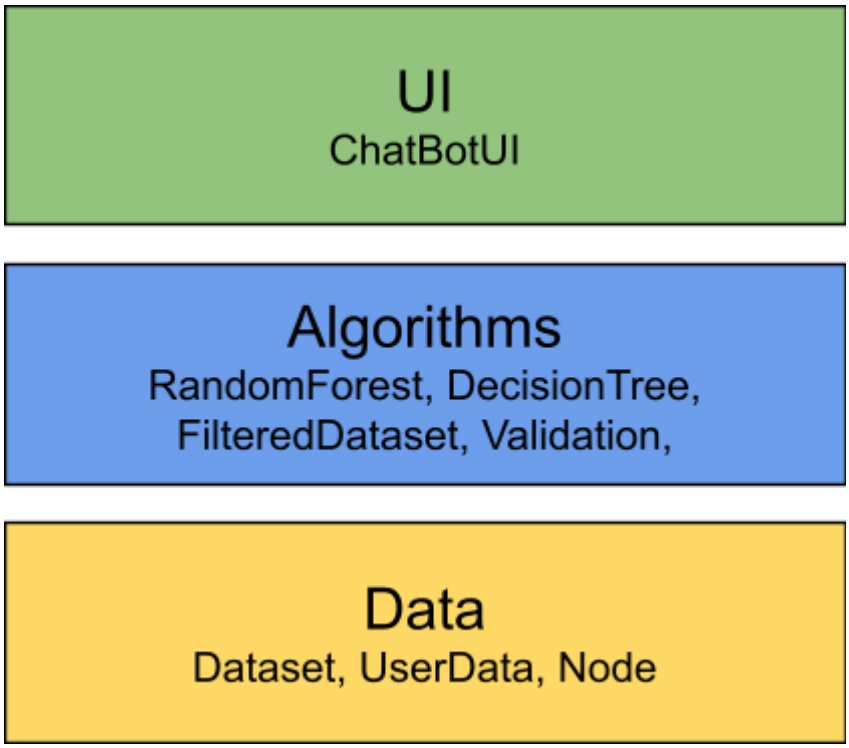
One of the main appeals of MVC architecture is that it allows data to change format without the user needing to be concerned with it. This is certainly appealing, as one of HeartScope's main features is manipulating user-entered data and presenting it back to the user. It would be inconvenient for the user to be forced to enter their data in a format that HeartScope can utilize, just as it would be confusing for the user to receive information in a format they cannot understand. Having this freedom would be a great help. However, HeartScope does not have a particularly complex user interaction aspect, and so the transformation of data can be performed by hand without much difficulty. Moreover, because of the simple model, there is a risk of additional complex code being needed to manage everything. As such, the benefit of using MVC is reduced, and so the conclusion is that MVC is not the best architectural pattern for HeartScope.

At first glance, Pipe-And-Filter seems like a good pattern to follow. The data is extracted from the user, calculations are performed on the data to convert it into a prediction, and the prediction is displayed to the user. There also isn't much interaction with the data while these calculations are being performed. Indeed, these are all desirable features to incorporate into our architecture. However, this requires the representation of the data to become separated from the object itself. This can pose a challenge to the engineers, as managing the split up data can require a lot of overhead and excessive management, resulting in slower results for the user, which HeartScope does not desire. As such, the Pipe-And-Filter architecture is also not ideal for HeartScope.

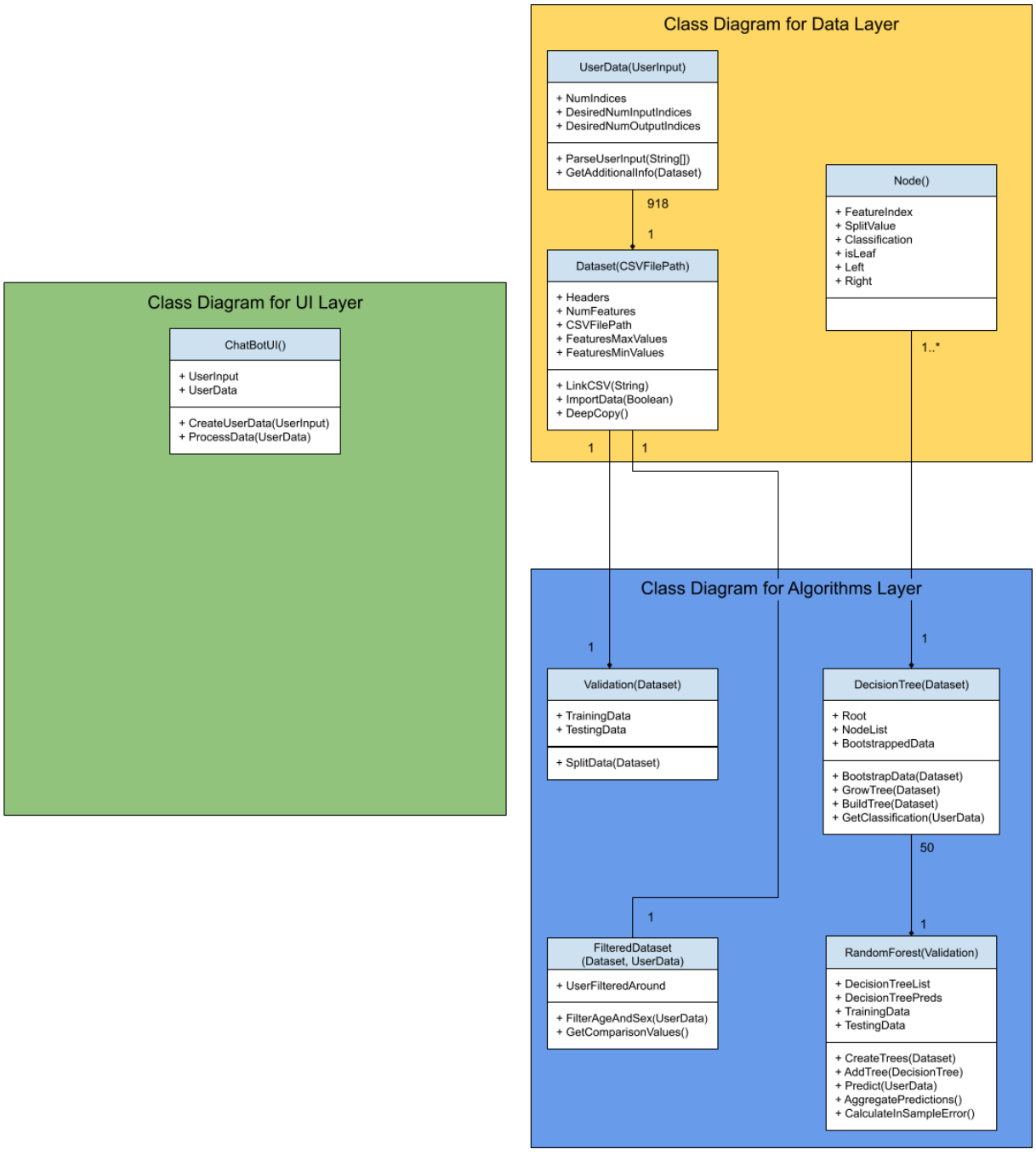
The best pattern, and the one HeartScope will utilize, is Layered architecture, as it is best utilized when building on top of already existing structures, such as the user interface we will be reusing. Furthermore, each member of the software development team will be responsible for a different part of development. Therefore, using an architecture that naturally divides the system into independent layers will allow each member to focus on their own tasks. This architecture also speeds up development time and reduces communication errors, as each person is free to make changes within their layer without notifying other members—so long as they do not

change the format of the data being transferred between layers. Utilizing the Layered Architecture pattern is the best choice for HeartScope.

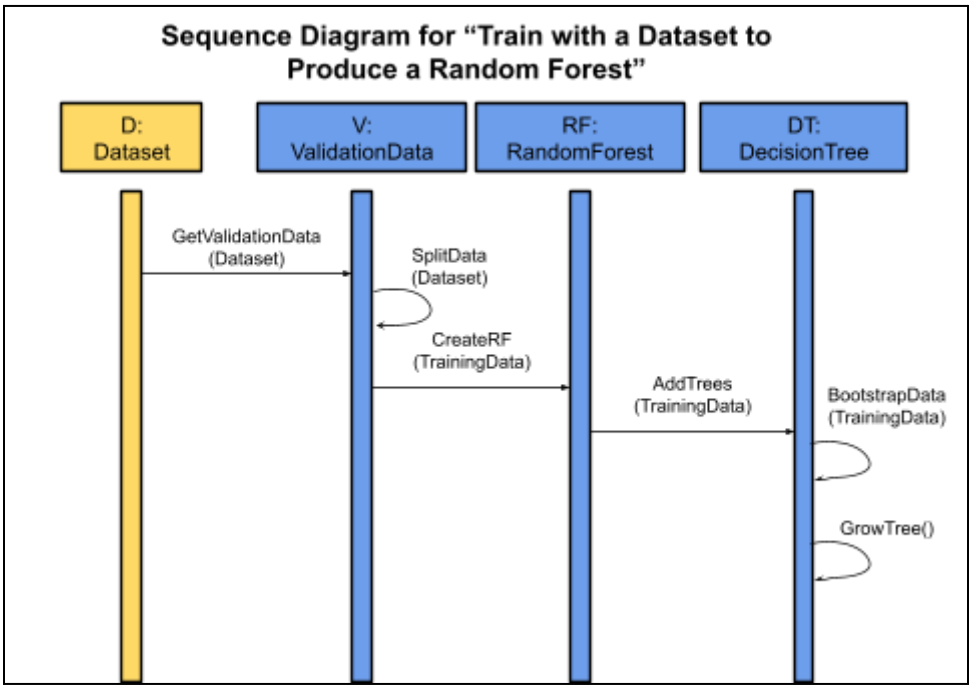
The architectural organization that we have concluded to be the best choice for our project's management is Layered Architecture. Specifically, HeartScope will use a 3-Layer Architecture, consisting of a UI, Algorithms, and Data layer. The UI layer will consist of all of the classes and elements related to extracting user input and displaying results, such as text fields, number fields, etc. Algorithms will be used to process and interact with the data, including normalizing the data, bootstrapping the data into decision trees, and creating and using the Random Forest. The Data layer contains the dataset used to create the Random Forest, as well as user data and the nodes that comprise it. We believe this to be the best architectural pattern for our system because of how it easily incorporates reused assets, naturally divides the components, freely allows for changes to be made within each layer, and reduces complexity for the software engineers.

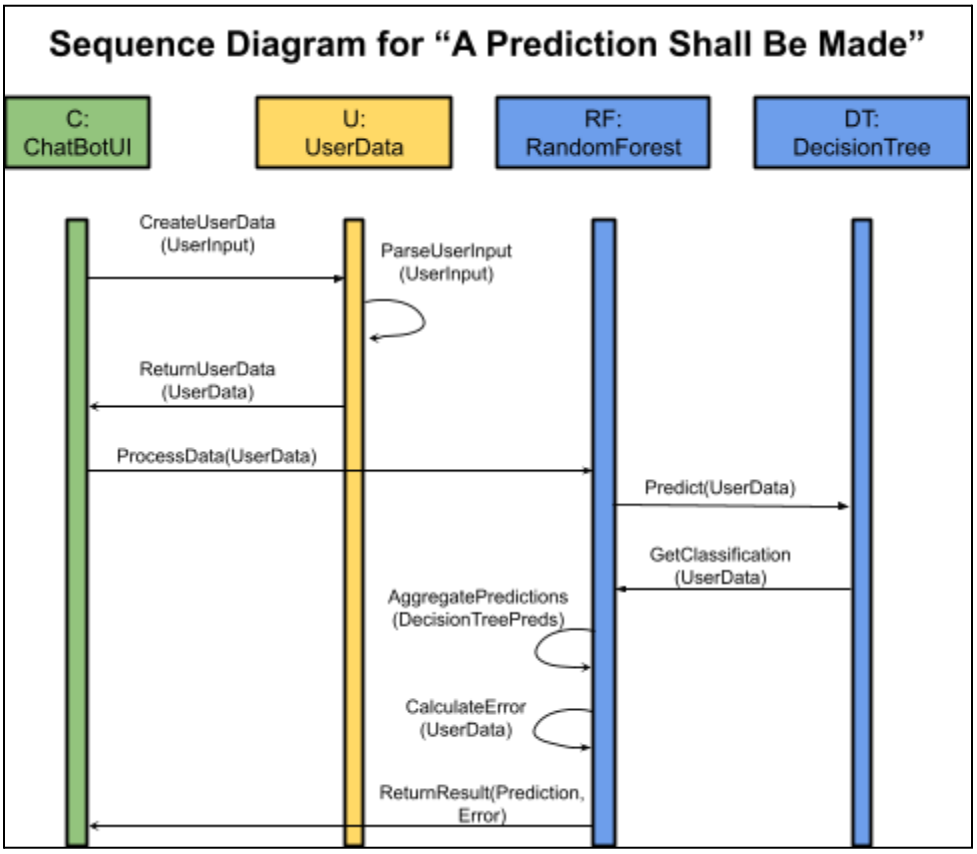


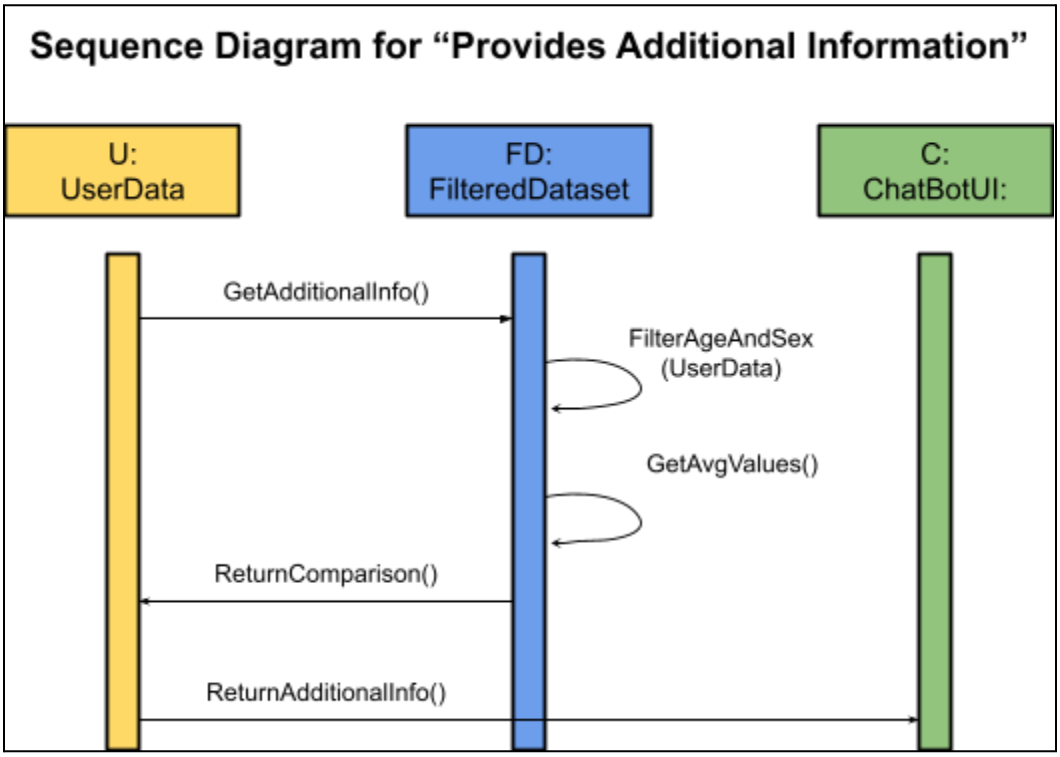
Structural Modeling



Interaction Modeling






Sequence Diagram for “Provides Additional Information”
U:
UserData
FD:
FilteredDataset
C:
ChatBotUI:
GetAdditionalInfo()
FilterAgeAndSex
(UserData)
GetAvgValues()
ReturnComparison()
ReturnAdditionalInfo()

Programming Environment

The development of this project featured multiple programming languages. The backend, including all elements of the Algorithms and Data layers, was programmed in Java using the *eclipse* IDE. The frontend, including all elements of the UI layer, were originally programmed in JavaScript, HTML, and CSS using the VS Code and IntelliJ IDEs. Our original plan was to ensure that the UI/UX interface for our application was user friendly and easily accessible for all ages. We used bright colors, dynamic animations and an easy to use GUI that can help anyone detect whether or not they have heart disease. Our frontend was composed of HTML, CSS and Javascript with a Spring Maven build Web App, which helps connectivity and launchability of our project. Asides from using documentation of specific languages : ChatGPT, UIverse.io and Spring.io were used to perfect the functions of our wonderful frontend.

Data Description

- Size of Total Dataset - The dataset contains 918 data-entry rows and 12 feature columns, resulting in a 918 x 12 dataset.

- Training Dataset - Utilizing the 8:2 Training:Testing data split, the size of the training dataset will be 734 entries.

- Testing Dataset - Utilizing the 8:2 Training:Testing data split, the size of the training dataset will be 184 entries.

- Data Attributes and Types - The raw dataset contains a combination of quantitative and categorical features. These features are listed in the table below, alongside their data type and the index in which they appear in the dataset.

Attribute	Type	Attribute	Type
0. Age - Range [28, 77]	Integer	6. RestingECG - 3 Categories	String
1. Sex - 2 Categories	Character	7. MaxHR - Range [60, 202]	Integer
2. ChestPainType - 4 Categories	String	8. ExerciseAngina - Range [0, 1]	Character
3. RestingBP - Range [0, 200]	Integer	9. Oldpeak - Range [-2.6, 6.2]	Float
4. Cholesterol - Range [0, 603]	Integer	10. ST_Slope - 3 Categories	String
5. FastingBS - Range [0, 1]	Integer	11. HeartDisease - Range [0, 1]	Integer

In order to work with the data more efficiently, data transformation is performed. Each categorical feature whose data type is a String value (ChestPainType, RestingECG, and ST_Slope) will have one-hot encoding applied. This will create more feature columns in an instance of UserData, but will make it easier for both the programmer and the RandomForest to use the data. Furthermore, each categorical feature whose data type is a Character value (Sex and ExerciseAngina) will have their values turned into a 0 or 1, as these categories only have two options (M/F and N/Y, respectively). Upon conversion, each entry is parsed into an instance of UserData, which is an ArrayList of type Number. The final data attributes and types are shown in the table, alongside their respective indices in UserData:

Attribute	Type	Attribute	Type
0. Age - Range [28, 77]	Int	10. ECG_ST - Range [0, 1]	Int
1. Sex - 2 Categories	Int	11. ECG_LVH - Range [0, 1]	Int
2. ChestPainATA - Range [0, 1]	Int	12. MaxHR - Range [60, 202]	Int
3. ChestPainNAP - Range [0, 1]	Int	13. ExerciseAngina - 2 Categories	Int
4. ChestPainASY - Range [0, 1]	Int	14. Oldpeak - Range [-2.6, 6.2]	Float
5. ChestPainTA - Range [0, 1]	Int	15. ST_Down - Range [0, 1]	Int
6. RestingBP - Range [0, 200]	Int	16. ST_Flat - Range [0, 1]	Int
7. Cholesterol - Range [0, 603]	Int	17. ST_Up - Range [0, 1]	Int
8. FastingBS - Range [0, 1]	Int	18. HeartDisease	Int
9. ECG_Normal - Range [0, 1]	Int		

Test Cases

Code for the test cases can be found in the GitHub repository at
https://github.com/bryan3342/CSCI370_SWE_Project/tree/main/HeartScope/src/test_cases

a. UI/UX Interface

i. ChatBotUI - Are outputs being shown to the screen (Is output() working properly)?
ii. ChatBotUI – Is the script implementing the data correctly? (Ensure that responses are based on RF model)
iii. ChatBotUI - Are the inputs being correctly processed and stored (Formatting and Communication between layers [data types])?
iv. ChatBotUI - Is ChatBotUI functionality functioning as intended?
v. ChatBotUI - Is Call and Response processing the data correctly between layers?
vi. ChatBotUI - does clearInput correctly clear data logs after 30 seconds of idle time?
vii. ChatBotUI - Is ChatBotUI sending correct input as a String or String array data type.
viii. ChatBotUI - Is input being correctly parsed as a string for the backend to receive?
ix. ChatBotUI - If the user's submitted input is blank, is the user told that they can't submit blank information?

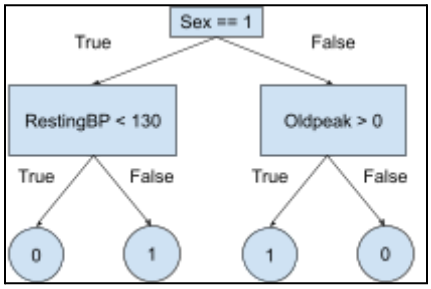
b. Algorithms Layer

i. Component Testing - DecisionTree - does BootstrapData() create a subset of data to grow the DecisionTree with?
    1. Input: Dataset = [
    [44, 0, 1, 0, 0, 0, 140, 289, 0, 1, 0, 0, 172, 0, 0.0, 0, 0, 1, 0]
    [46, 0, 1, 0, 0, 0, 130, 283, 0, 0, 1, 0, 98, 0, 0.0, 0, 0, 1, 0]
    [48, 0, 0, 1, 0, 0, 150, 195, 0, 1, 0, 0, 122, 0, 0.0, 0, 0, 1, 0]
    [50, 0, 0, 0, 1, 0, 140, 207, 0, 1, 0, 0, 130, 1, 1.5, 0, 1, 0, 1]
    [45, 1, 0, 1, 0, 0, 160, 180, 0, 1, 0, 0, 156, 0, 1.0, 0, 1, 0, 1]
    [47, 1, 0, 0, 1, 0, 138, 214, 0, 1, 0, 0, 108, 1, 1.5, 0, 1, 0, 1]
    [49, 1, 0, 1, 0, 0, 120, 339, 0, 1, 0, 0, 170, 0, 0.0, 0, 0, 1, 0]
    ]
    2. Scenario: 50 data points are selected with replacement from the original Dataset.

3. Expected Outcome: A subset of the input Dataset is created to grow the DecisionTree with. This creates a new Dataset object that stores the 50 randomly selected data points.
4. Assertion Method: Check that the new Dataset contains exactly 50 data points and that each data point can be found in the original Dataset.

ii. Component Testing - DecisionTree - does GrowTree() properly grow a tree from a bootstrapped dataset?

1. Input: Dataset = [
   [44, 0, 1, 0, 0, 0, 140, 289, 0, 1, 0, 0, 172, 0, 0.0, 0, 0, 1, 0]
   [46, 0, 1, 0, 0, 0, 130, 283, 0, 0, 1, 0, 98, 0, 0.0, 0, 0, 1, 0]
   [48, 0, 0, 1, 0, 0, 150, 195, 0, 1, 0, 0, 122, 0, 0.0, 0, 0, 1, 0]
   [50, 0, 0, 0, 1, 0, 140, 207, 0, 1, 0, 0, 130, 1, 1.5, 0, 1, 0, 1]
   [45, 1, 0, 1, 0, 0, 160, 180, 0, 1, 0, 0, 156, 0, 1.0, 0, 1, 0, 1]
   [47, 1, 0, 0, 1, 0, 138, 214, 0, 1, 0, 0, 108, 1, 1.5, 0, 1, 0, 1]
   [49, 1, 0, 1, 0, 0, 120, 339, 0, 1, 0, 0, 170, 0, 0.0, 0, 0, 1, 0]
   ]
2. Scenario: The Dataset is received from BootstrapData().
3. Expected Outcome: A fully grown DecisionTree is created, meaning that it is grown to a point where all leaves are pure - DecisionTree =



4. Assertion Method: Given the resulting node list, hand trace the data through the DecisionTree to ensure that all data ends up in a leaf with the correct binary classification. This will ensure that a fully grown decision tree is constructed, which ensures that all leaves are pure by proxy.

iii. Component - Decision Tree - Is the same DecisionTree created when the same input is provided?

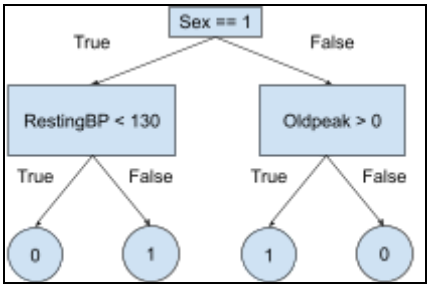
1. Input: Dataset = [
   [44, 0, 1, 0, 0, 0, 140, 289, 0, 1, 0, 0, 172, 0, 0.0, 0, 0, 1, 0]

[46, 0, 1, 0, 0, 0, 130, 283, 0, 0, 1, 0, 98, 0, 0.0, 0, 0, 1, 0]
[48, 0, 0, 1, 0, 0, 150, 195, 0, 1, 0, 0, 122, 0, 0.0, 0, 0, 1, 0]
[50, 0, 0, 0, 1, 0, 140, 207, 0, 1, 0, 0, 130, 1, 1.5, 0, 1, 0, 1]
[45, 1, 0, 1, 0, 0, 160, 180, 0, 1, 0, 0, 156, 0, 1.0, 0, 1, 0, 1]
[47, 1, 0, 0, 1, 0, 138, 214, 0, 1, 0, 0, 108, 1, 1.5, 0, 1, 0, 1]
[49, 1, 0, 1, 0, 0, 120, 339, 0, 1, 0, 0, 170, 0, 0.0, 0, 0, 1, 0]
]

2. Scenario: The Dataset is received from BootstrapData().
3. Expected Outcome: Given the same input, the same DecisionTree should be output each time.
4. Assertion Method: Create two DecisionTrees from the same input Dataset and verify that their node structures are identical to one another.

iv. Component Testing - DecisionTree - does GetClassification() return the correct value

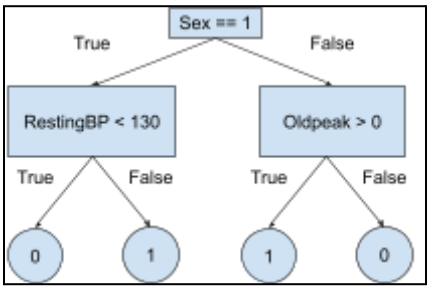
1. Input: UserData = [80, 0, 0, 0, 1, 0, 138, 214, 0, 1, 0, 0, 108, 1, 1.5, 0, 1, 0, 1]
   DecisionTree =



2. Scenario: The UserData is present in the bootstrapped data.
3. Expected Outcome: The UserData is passed through the nodes of the tree, being binarily classified as it progresses deeper down the tree, until it ends up at a leaf node (0 or 1). Whichever classification it ends up in is returned and stored in RandomForest's *DecisionTreePreds* list.
4. Assertion Method: Given a UserData, trace the path by hand to reach a classification and ensure that the function returns the same classification and stores it in *DecisionTreePreds*. At the same time, ensure that each time the same UserData is given, the same classification is returned.

v. Component Testing - RandomForest - Does AddTree() increase the number of trees in the RandomForest?

1. Input: RandomForest = []
   DecisionTree =



2. Scenario: The RandomForest is empty.
3. Expected Outcome: The RandomForest adds the DecisionTree to its *DecisionTreeList*.
4. Assertion Method: Ensure that the RandomForest's *DecisionTreeList* after the function call is completed contains the DecisionTree added and only that DecisionTree.

vi. Component Testing - RandomForest - does AggregatePrediction() properly return the most common binary classification?

1. Input: RandomForest.*DecisionTreePreds* = [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0]
2. Scenario: The *DecisionTreePreds* list contains more occurrences of one value than the other.
3. Expected Outcome: Whichever binary classification (0 or 1) appears most often in the input is returned, essentially computing the modal value of *DecisionTreePreds* - Integer = 0.
4. Assertion Method: Calculate the modal value of the array by hand and ensure that the function returns the same value.

vii. Component Testing - RandomForest - does AggregatePrediction() properly return the most common binary classification?

1. Input: RandomForest.*DecisionTreePreds* = [0, 0, 1, 0, 1, 0, 1, 0, 1, 1]
2. Scenario: The input predictions list is bimodal, containing an equal number of 0s and 1s.

3. Expected Outcome: Since there is no single modal value in the array, 1 will be returned by default.
4. Assertion Method: Ensure that the function returns the value 1.

viii. Component - FilteredDataset - does FilterAgeAndSex() properly filter entries?

1. Input: Dataset = [
   [44, 0, 1, 0, 0, 0, 140, 289, 0, 1, 0, 0, 172, 0, 0.0, 0, 0, 1, 0]
   [46, 0, 1, 0, 0, 0, 130, 283, 0, 0, 1, 0, 98, 0, 0.0, 0, 0, 1, 0]
   [48, 0, 0, 1, 0, 0, 150, 195, 0, 1, 0, 0, 122, 0, 0.0, 0, 0, 1, 0]
   [50, 0, 0, 0, 1, 0, 140, 207, 0, 1, 0, 0, 130, 1, 1.5, 0, 1, 0, 1]
   [45, 1, 0, 1, 0, 0, 160, 180, 0, 1, 0, 0, 156, 0, 1.0, 0, 1, 0, 1]
   [47, 1, 0, 0, 1, 0, 138, 214, 0, 1, 0, 0, 108, 1, 1.5, 0, 1, 0, 1]
   [49, 1, 0, 1, 0, 0, 120, 339, 0, 1, 0, 0, 170, 0, 0.0, 0, 0, 1, 0]
   ]

   UserData = [47, 0, 0, 0, 1, 0, 138, 214, 0, 1, 0, 0, 108, 1, 1.5, 0, 1, 0, 1]

2. Scenario: The Dataset contains at least one entry where the sex is the same as the user's and the age is within 2 years of the user's.
3. Expected Outcome: Only those entries where the sex matches the UserData's sex and the age is within 2 years of the UserData's age are returned - Dataset = [
   [46, 0, 1, 0, 0, 0, 130, 283, 0, 0, 1, 0, 98, 0, 0.0, 0, 0, 1, 0],
   [48, 0, 0, 1, 0, 0, 150, 195, 0, 1, 0, 0, 122, 0, 0.0, 0, 0, 1, 0]
   ]
4. Assertion Method: The entries in the Dataset produced by the function match one-to-one with the entries in the Dataset produced by filtering manually.

ix. Component - FilteredDataset - does FilterAgeAndSex() properly filter entries?

1. Input: Dataset = [
   [44, 0, 1, 0, 0, 0, 140, 289, 0, 1, 0, 0, 172, 0, 0.0, 0, 0, 1, 0]
   [46, 0, 1, 0, 0, 0, 130, 283, 0, 0, 1, 0, 98, 0, 0.0, 0, 0, 1, 0]
   [48, 0, 0, 1, 0, 0, 150, 195, 0, 1, 0, 0, 122, 0, 0.0, 0, 0, 1, 0]
   [50, 0, 0, 0, 1, 0, 140, 207, 0, 1, 0, 0, 130, 1, 1.5, 0, 1, 0, 1]

[45, 1, 0, 1, 0, 0, 160, 180, 0, 1, 0, 0, 156, 0, 1.0, 0, 1, 0, 1]
[47, 1, 0, 0, 1, 0, 138, 214, 0, 1, 0, 0, 108, 1, 1.5, 0, 1, 0, 1]
[49, 1, 0, 1, 0, 0, 120, 339, 0, 1, 0, 0, 170, 0, 0.0, 0, 0, 1, 0]
]

UserData = [80, 0, 0, 0, 1, 0, 138, 214, 0, 1, 0, 0, 108, 1, 1.5, 0, 1, 0, 1]

2. Scenario: The Dataset contains no entries where the sex is the same and the age is within 2 years of the UserData.
3. Expected Outcome: The age filtering aspect will be ignored, and all entries where the sex matches the UserData's sex are returned - Dataset = [
[44, 0, 1, 0, 0, 0, 140, 289, 0, 1, 0, 0, 172, 0, 0.0, 0, 0, 1, 0]
[46, 0, 1, 0, 0, 0, 130, 283, 0, 0, 1, 0, 98, 0, 0.0, 0, 0, 1, 0]
[48, 0, 0, 1, 0, 0, 150, 195, 0, 1, 0, 0, 122, 0, 0.0, 0, 0, 1, 0]
[50, 0, 0, 0, 1, 0, 140, 207, 0, 1, 0, 0, 130, 1, 1.5, 0, 1, 0, 1]
]
4. Assertion Method: The entries in the Dataset produced by the function match one-to-one with the entries in the Dataset produced by filtering manually.

x. Unit - FilteredDataset - does GetComparisonValues() properly return modal values for each feature?
1. Input: Dataset = [
[44, 0, 1, 0, 0, 0, 140, 289, 0, 1, 0, 0, 172, 0, 0.0, 0, 0, 1, 0]
[46, 0, 1, 0, 0, 0, 130, 283, 0, 0, 1, 0, 98, 0, 0.0, 0, 0, 1, 0]
[48, 0, 0, 1, 0, 0, 150, 195, 0, 1, 0, 0, 122, 0, 0.0, 0, 0, 1, 0]
[50, 0, 0, 0, 1, 0, 140, 207, 0, 1, 0, 0, 130, 1, 1.5, 0, 1, 0, 1]
]
2. Scenario: The Dataset contains at least one entry.
3. Expected Outcome: The modal value for each feature in the FilteredDataset is calculated and used to create a new instance of UserData, creating a hypothetical "average user" that the client can be compared to - UserData =
[47, 0, 1, 0, 0, 0, 140, 243, 0, 1, 0, 0, 130, 0, 0.375, 0, 0, 1, 0]
4. Assertion Method: The entries in the UserData produced by the function match one-to-one with the entries in the UserData produced by calculating the comparison values manually.

xi. Unit - FilteredDataset - does GetComparisonValues() properly return modal values for each feature?

1. Input: Dataset = [
   [44, 0, 1, 0, 0, 0, 140, 289, 0, 1, 0, 0, 172, 0, 0.0, 0, 0, 1, 0]
   [46, 0, 1, 0, 0, 0, 130, 283, 0, 0, 1, 0, 98, 0, 0.0, 0, 0, 1, 0]
   [48, 0, 0, 1, 0, 0, 150, 195, 0, 1, 0, 0, 122, 0, 0.0, 0, 0, 1, 0]
   [50, 0, 0, 0, 0, 1, 0, 140, 207, 0, 1, 0, 0, 130, 1, 1.5, 0, 1, 0, 1]
   ]
2. Scenario: The Dataset contains no entries.
3. Expected Outcome: Return an instance of UserData where the entries are all -1, a signal to the system that there are no entries to compare the user to. This should never happen, as there is at least one “Male” and “Female” entry each in the dataset after being filtered down - UserData = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
4. Assertion Method: Ensure that the function outputs a UserData where all entries are -1.

c. Data Layer

i. Component - UserData - does ParseUserData() accept a user’s input?

1. Input: String[] = ["55", "F", "ATA", "130", "215", "0", "LVH", "132", "Y", "0.0", "Up", "0"]
2. Scenario: The user’s input is complete.
3. Expected Outcome: A UserData whose entries store the parsed values - UserData = [55, 1, 1, 0, 0, 0, 130, 215, 0, 0, 0, 1, 132, 1, 0.0, 0, 0, 1, 0]
4. Assertion Method: The entries in the UserData produced by the function match one-to-one with the entries in the UserData produced by parsing the UserData manually.

ii. Component - UserData - does ParseUserData() accept a user’s input?

1. Input: String[] = ["55", "F", "ATA", "130", "", "0", "", "132", "", "", "", ""]
2. Scenario: The user’s input is partially complete.
3. Expected Outcome: A UserData whose entries store the user-entered data for those entries where the user provided data

and the average values (as calculated from the raw dataset) for the entries where no data was provided - UserData = [55, 1, 1, 0, 0, 0, 130, 245, 0, 1, 0, 0, 132, 0, 0.0, 0, 1, 0, 0]
4. Assertion Method: The entries in the UserData produced by the function match one-to-one with the entries in the UserData produced by parsing the UserData manually.

iii. Unit - Dataset - does LinkCSV() properly connect the dataset to a .csv file?
1. Input: String = “src/data/thisfiledoesnotexist.csv”, which is a file path that does not lead to a file.
2. Scenario: The file attempting to be linked to the Dataset does not exist.
3. Expected Outcome: The dataset will not link to the file. The console will display a warning message.
4. Assertion Method: The function returns an error code. When an invalid file path is attempted to be used to load data, the function should return an error code -1 and not link the file to the Dataset. Ensure that this error code is returned.

iv. Unit - Dataset - does LinkCSV() properly connect the dataset to a .csv file?
1. Input: String = “src/data/not_a_csv.txt”, which is a file path that leads to a .txt file.
2. Scenario: The file attempting to be linked to the Dataset exists, but is not a .csv file.
3. Expected Outcome: The dataset will not link to the file. The console will display a warning message.
4. Assertion Method: The function returns an error code. When a file that is anything but a .csv file is attempted to be used to load data, the function should return an error code -2 and not link the file to the Dataset. Ensure that this error code is returned.

v. Unit - Dataset - does LinkCSV() properly connect the dataset to a .csv file?
1. Input: String = “src/data/empty.csv”, which is a file path that leads to a .csv file.
2. Scenario: The file attempting to be linked to the Dataset exists and is a .csv file.
3. Expected Outcome: The dataset will link to the file.

4. Assertion Method: The function returns an error code. When a .csv file exists and is attempted to be used to load data, the function should return an error code 0 and link the file to the Dataset. Ensure that this error code is returned.

vi. Unit - Dataset - does ImportData() properly import the data from the linked .csv file to the Dataset?
    1. Input: String = “src/data/empty.csv”, which is a file path that leads to an empty .csv file.
    2. Scenario: The linked .csv file is empty.
    3. Expected Outcome: An empty Dataset will be created.
    4. Assertion Method: Ensure that a Dataset with no entries is created.

vii. Unit - Dataset - does ImportData() properly import the data from the linked .csv file to the Dataset?
    1. Input: String = “src/data/header_csv.csv”, which is a file path that leads to a .csv file that contains feature headers.
       Boolean = true
    2. Scenario: The linked .csv file contains data and has headers.
    3. Expected Outcome: The headers will be extracted and only the data will be stored in the Dataset.
    4. Assertion Method: Ensure that each entry in the .csv file is transferred into an entry in the Dataset. Also ensure that the headers are stored in the *headers* class variable.

viii. Unit - Dataset - does ImportData() properly import the data from the linked .csv file to the Dataset?
    1. Input: String = “src/data/no_header_csv.csv”, which is a file path that leads to a .csv file that contains only data, no headers.
       Boolean = false
    2. Scenario: The linked .csv file contains data and no headers.
    3. Expected Outcome: The data will be stored in the Dataset.
    4. Assertion Method: Ensure that each entry in the .csv file is transferred into an entry in the Dataset. Also ensure that the *headers* class variable is empty.

Testing Results

A breakdown for the Unit Test cases listed above are as follows:

Class	Results
DatasetTest	6/6
DecisionTreeTest	3/4
FilteredDatasetTest	4/4
RandomForestTest	4/4
UserDataTest	2/2
ValidationTest	1/1

System-wise Testing

1. Scenarios for testing Functional Req 1 - Train RF
    a. The RF is trained using too small of a training dataset.
        i. Expected Outcome: If there is less training data available, then it is expected that the fully grown DecisionTrees will be shallower. It will take fewer splits to fully separate the data. This increases the likelihood of underfitting the data, and so it is expected that the resulting RandomForest will be less likely to correctly classify the UserData. This will be checked by directly comparing the in-sample error after training one RandomForest with only 200 data points (small) and one with the normal amount of data points.

    b. The RF is trained using too large of a training dataset.
        i. Expected Outcome: If there is too much training data available, then it is expected that the fully grown DecisionTrees will be very deep, in order to properly classify the bootstrapped data. As such, the model will overfit according to the training data and it will be impossible to discover the underlying function that the model is attempting to learn. It is expected that the resulting RandomForest will be less likely to correctly classify the UserData.

c. The RF is trained using a properly sized training dataset
    i. Expected Outcome: With a proper amount of training data, it is expected that overfitting and underfitting will be avoided, resulting in optimal results for the RandomForest. This will result in fewer misclassifications and a higher chance that the UserData will be correctly classified.

2. Scenarios for testing Functional Req 2 - Make a prediction

a. UserInput has all entries filled properly
    i. Expected Outcome: The system will properly parse the user's input as it was given. It will create an instance of UserData that is filled with the parsed data. That instance of UserData will be provided as input to the RandomForest, which in turn will pass it to each DecisionTree. Each DecisionTree will take the UserData as input and output a binary classification, 0 or 1. Once all DecisionTrees have output their results to the RandomForest, the binary classification that is in the majority will be returned as the prediction.

b. UserInput has some entries filled properly
    i. Expected Outcome: The system will properly parse the entries that were provided by the user. Any entries left unfilled will have their values filled by interpolating in the median value across the dataset for that feature. It will create an instance of UserData that is filled with the parsed data. That instance of UserData will be provided as input to the RandomForest, which in turn will pass it to each DecisionTree. Each DecisionTree will take the UserData as input and output a binary classification, 0 or 1. Once all DecisionTrees have output their results to the RandomForest, the binary classification that is in the majority will be returned as the prediction.

c. UserInput has no entries filled properly (corrupted/blank)
    i. Expected Outcome: An error message will be displayed saying that something went wrong. The user will be asked to try again. The ChatBot's UI will be reset and the process will restart.

3. Scenarios for testing Functional Req 3 - Additional Info
    a. Upon filtering the data, there is at least one match
        i. Expected Outcome: The modal value for each feature in the filtered dataset will be calculated. These modal values will be placed into an instance of UserData, creating a hypothetical data point that the user will be compared to.

    b. Upon filtering the data, there are no matches
        i. Expected Outcome: Remove the age-wise filtering and instead filter only on sex. It is guaranteed that there will be at least one match, and so the above scenario is carried out. The user will also be notified that there is not sufficient data to compare them to their peers age-wise and sex-wise, and so they have instead been compared to their peers sex-wise only.

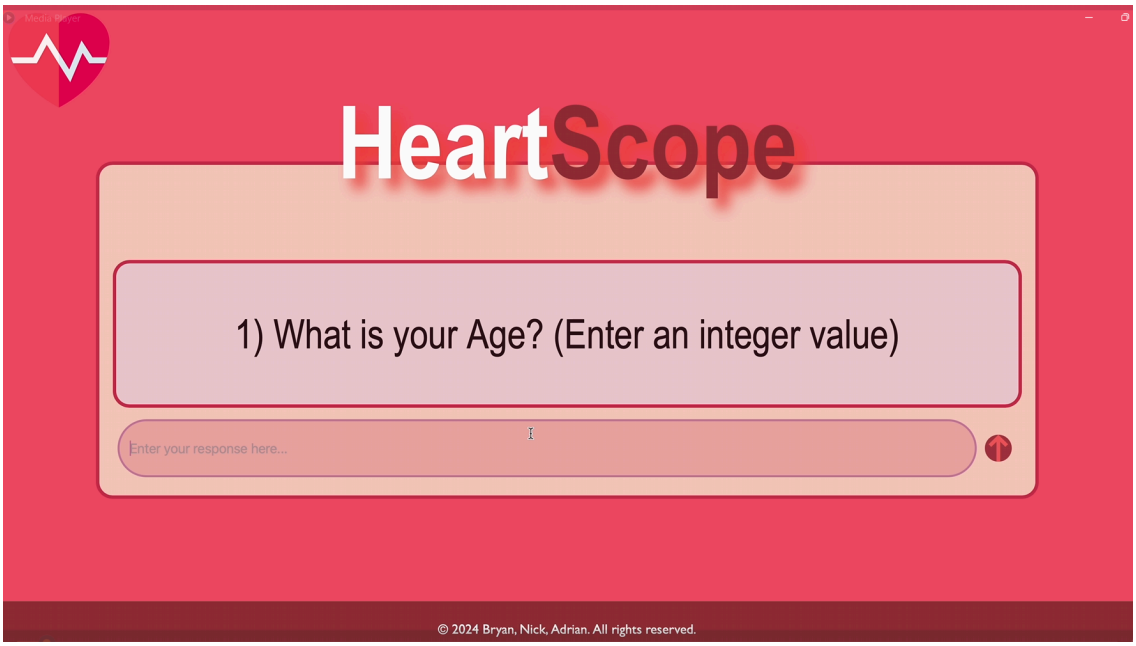
Acceptance Testing

1. Accuracy Measurement
    i. To measure accuracy, the out-sample error produced when validating the RandomForest will be calculated. For each data point in the testing dataset, the actual outcome for each data point will be compared to the outcome predicted by the RandomForest. The number of wrong matches divided by the number of data points in the testing dataset will be the error of the RandomForest model.

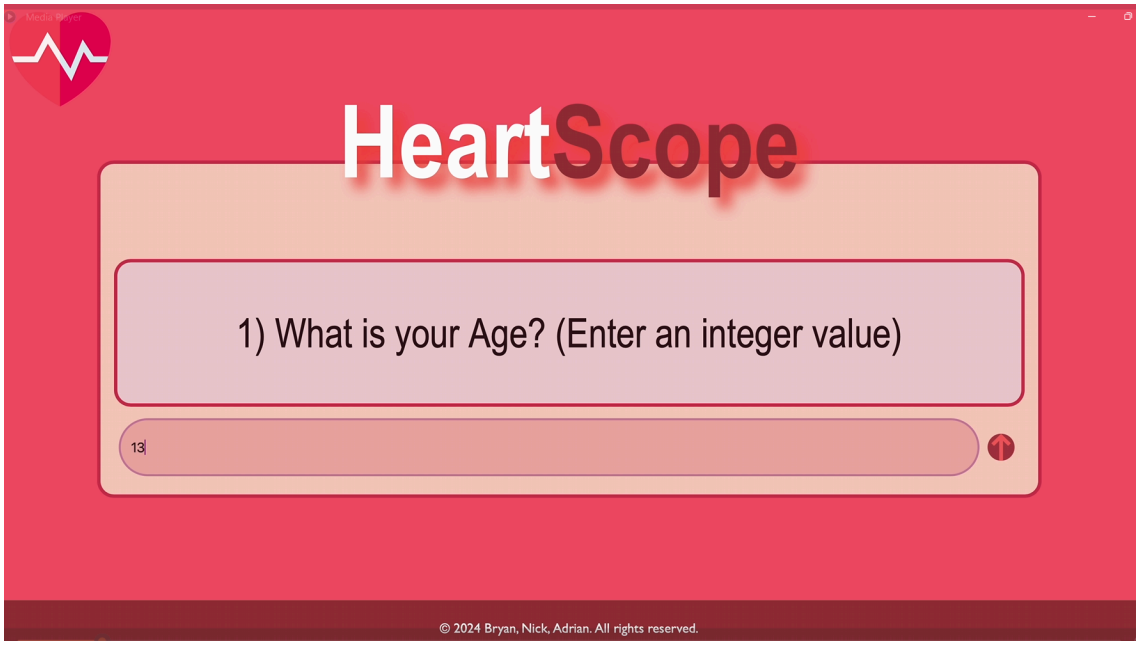
b. Security Measurement
    i. To ensure security, two measurements will be taken. First, to ensure that the user's data is not stored, the dimensions of the original dataset will be compared to the final dataset. If the dimensions match, then no data was added to the dataset. Second, to ensure privacy for the user's information, the ChatbotUI will utilize automatic data clearing . If the user does not interact with the ChatbotUI for 30 seconds, then the ChatbotUI will clear its records.

c. Speed Measurement
    i. To measure the amount of time required for a user to use the software, two timestamps will be recorded. The first timestamp will be when the ChatbotUI sends the UserInput to the backend. The second timestamp will be when the backend sends all relevant information to the ChatbotUI. The difference between these two timestamps will be the time it takes to process the UserInput.

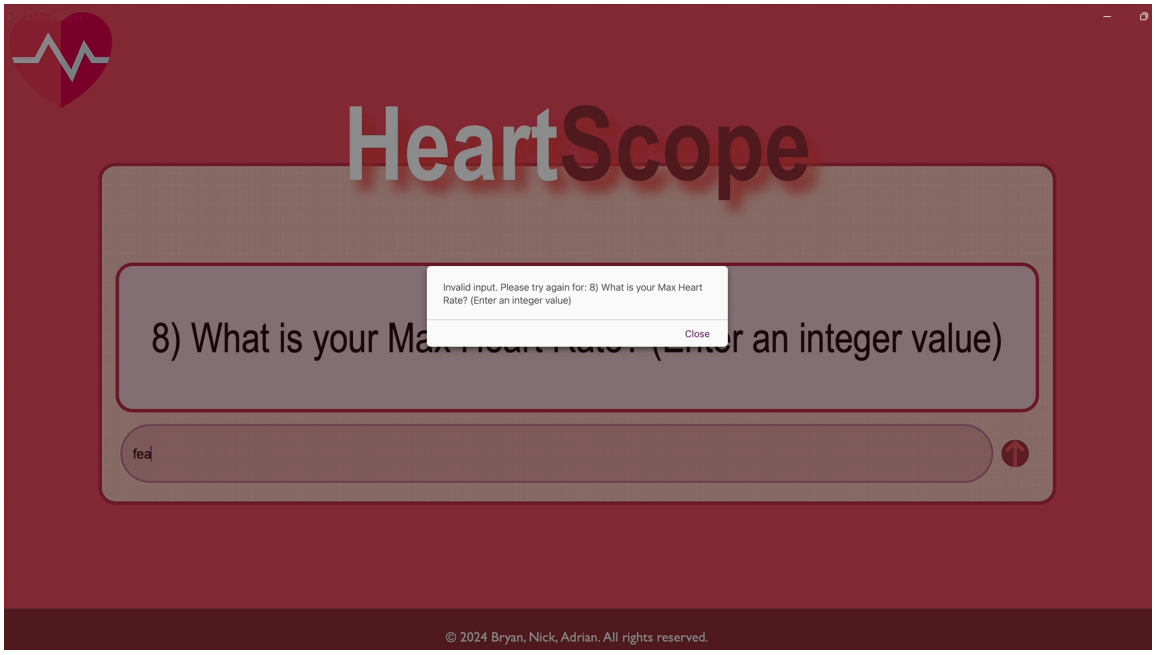
User Guide



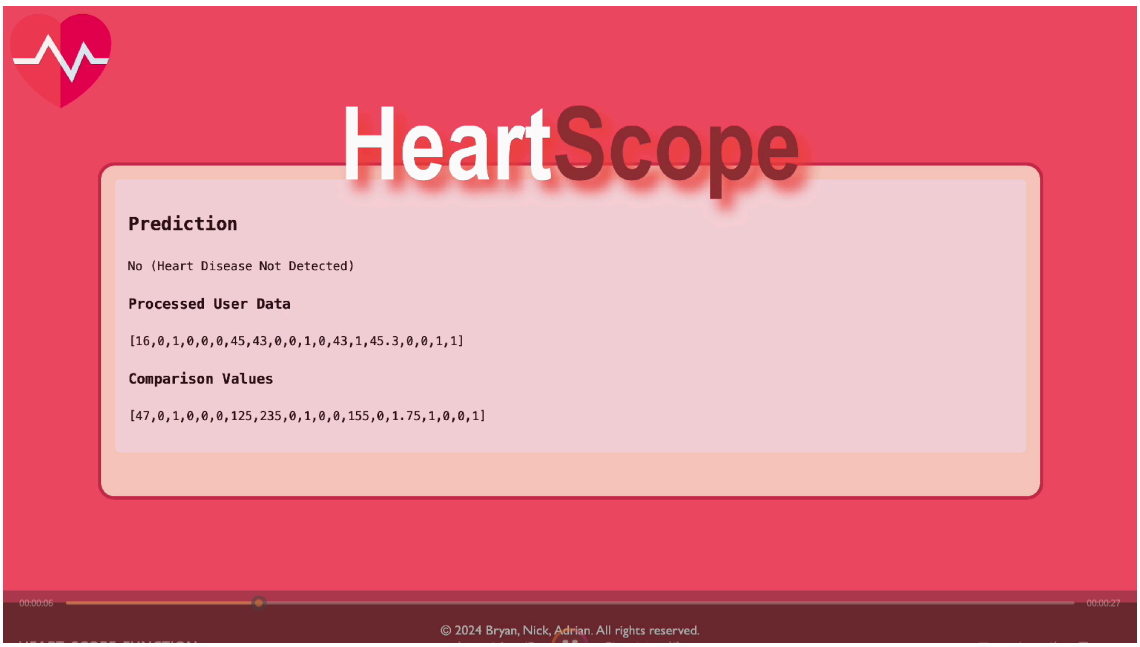
This is the home screen of the UI. When the software opens, this is the screen that will greet the user.



The user will be prompted to answer a series of questions. The user should enter their answer to each question in the box below, following the instructions prompt. For example, if the prompt says to enter an integer value, then the user should enter an integer value and only an integer value. The user is required to enter their age and sex, but all other questions may be skipped.



If the user enters a response that is not expected (for example, they enter letters when numbers are expected), they will be warned and asked to submit another value.



Upon answering all 12 questions, the user's response is automatically evaluated. The user is then shown two pieces of information. The first is our software's prediction of whether the user does or does not have heart disease. The second is the additional information our software provides the user; a peer-wise comparison. This filters our dataset for entries with similar sex and age to the user's and computes a "hypothetical average person" for that group. This "hypothetical average person" is shown to the user, to allow them to see how they compare to their peers.

Summary of Complete and Incomplete Work

There were three user requirements that HeartScope was required to fulfill, The first of these requirements was that HeartScope must train a RandomForest classifier. Without the classifier, HeartScope would not be able to function. We successfully trained a RandomForest classifier on our dataset. The second user requirement was that HeartScope must make a prediction. This prediction is the main service offered by the software - to predict whether the user has heart disease or not. Through the RandomForest and its DecisionTrees, we were able to generate a prediction for the user. The third and final user requirement was that HeartScope must provide some additional feedback to the user aside from the prediction. Our software achieves this by comparing the user to those people in the dataset who are of the same sex and similar age. This provides the user with information on how healthy they are compared to their peers. With all three of these incorporated into HeartScope, we successfully fulfilled all user requirements.

There were three system requirements that HeartScope was intended to fulfil as well. The first of these was that the RandomForest classifier had to be accurate. We defined our goal to be a classifier whose in-sample validation error was at or below 10%. The final results saw a typical error of around 12%. Although this is a very good result, it fell short of our standards, meaning that this requirement is incomplete. The second system requirement was security, which was defined to include two parts, both of which we were able to incorporate into HeartScope:

1. The UI and its logs would be cleared and reset if the user remained inactive for 30s, which is done to assure the user that their data is not being saved and protect their information if they were to leave the computer.
2. The UI would never display the user's information on the screen while collecting data, to ensure that no passersby can see the user's information.

The third system requirement was a rapid response from the software, providing the user with results in 10 seconds. This would be measured by timing how long it took, on average, to validate a single data point in the validation dataset. The end result of this found a single data point to only take a few milliseconds to be processed. Our time requirement was set early in production, before we had an idea of how long this would take. Perhaps a better measure would be to ask how long it takes the user to complete the entire process, from filling out the form to receiving a response. This timing would vary from user to user, but we expect anyone to be able to complete the questionnaire within two minutes. Regardless, this system requirement was completed. To summarize, we successfully implemented all three user requirements and two of the three system requirements, with the one failed system requirement being nearly completed.

Planning for Phase II

The next steps for HeartScope would include improving the model, UI, and services provided to the user. The model would be improved in terms of its ability to make predictions. This can be achieved by training the model on more data, which would require the procurement of said data. If a dataset were found that included the same features as the one used on the current model, that data could be extracted and trained upon. However, it is unlikely to find such data, so the model may need to be trained on a different dataset altogether. Improvements to the UI can also be made. For one, the time constraint prevented us from making the UI as visually appealing as we would have liked to, so some cosmetic improvements are in order. Also, the prompts could be improved for more clarity. Additionally, we were not able to implement the feature of the user being informed if they are only compared sex-wise, not age-and-sex-wise, for the additional information. This would be another logical next step. In a similar manner, we would have liked to display more information to the user, which ties into the services we would like to provide. For example, if HeartScope determines that the user's health is in danger, the software can provide the user with resources to reach out to, such as nearby medical clinics. We would also like to provide more information on how the user compares to their peers. Rather than just showing the user what the average person looks like, we would like to show what categories the user is "better" in and which they are "worse" in, relative to the hypothetical average person. This is all in addition to the "incomplete works" in the above section.

GitHub Repository

The GitHub repository that hosts our project can be found at:

https://github.com/bryan3342/CSCI370_SWE_Project

The source code for the object classes, including the classifier, can be found under:

"src/main/java/heartscope_final/com/heartscope"

The source code for the unit test cases can be found under:

"src/test/java/heartscope_final/com/heartscope"