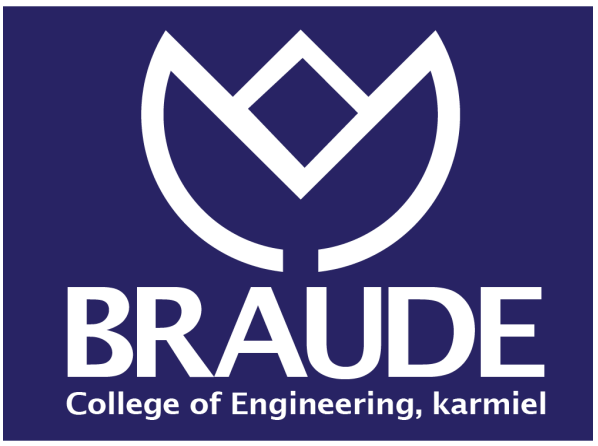


Software Engineering Department
ORT Braude College

Capstone Project Phase 2 - 61999

Heart Rate Monitoring using Machine Learning

24-1-R-8

By:

Ali Mohsen	206132391	Ali.Mohsen@e.braude.ac.il
Mohammad Khamaisy	314933649	Mohammad.Khamaisy@e.braude.ac.il

Supervisor: Zeev Frenkel

https://github.com/MiNEWGIT/FinalProject

2024

Table of content

Abstract. This paper details the comprehensive journey of developing an innovative heart rate monitoring application, from conception to implementation. It explores the intricate process of integrating cutting-edge sensor technology with advanced machine learning algorithms to create a personalized, user-centric solution for cardiovascular health management. The narrative delves into the challenges encountered and the solutions devised while implementing key features such as real-time heart rate tracking, anomaly detection, and predictive analytics. It provides insights into the selection and adaptation of crucial algorithms, including Isolation Forests, One-Class SVMs, and Time Series Forecasting, elucidating their roles in enhancing the app's functionality. Additionally, this book discusses the iterative development process and user experience design considerations. By offering a detailed account of the app's evolution, this work serves as a valuable resource for developers, healthcare professionals, and enthusiasts interested in the intersection of wearable technology, machine learning, and personalized health monitoring.

1. Introduction

Cardiovascular diseases (CVDs) are the leading cause of death worldwide, responsible for an estimated 17.9 million deaths each year. These conditions include a range of disorders affecting the heart and blood vessels, such as heart attacks, strokes, and hypertension. The prevalence of CVDs highlights the urgent need for effective prevention and management strategies. Early detection of cardiac abnormalities can significantly improve patient outcomes through timely intervention and preventive care [16].

The rapid adoption of smartwatches and mobile health apps presents a unique opportunity to analyze heart rate data at scale for signs of arrhythmias and other potential issues. However, most consumer devices today rely on basic algorithms with limited capabilities for detecting abnormalities in heart rate data. Our project aims to bridge this gap by developing and evaluating advanced machine learning techniques to more reliably identify anomalous patterns in heart rate signals gathered by smart devices.

Many existing heart rate monitoring apps and devices fail to account for individual variations, often relying on generic thresholds or population-level data, compromising accuracy and personalization. While some proprietary research devices use advanced electrocardiogram (EKG) sensors and AI-based algorithms, these are not widely accessible. There is a clear need for accurate, automated detection of potential abnormalities that can be deployed through consumer smart device apps and wearables.

Our project involves developing an application designed to monitor heart rate data in real-time. This app caters to individuals of all ages and health statuses, offering features aimed at promoting overall wellness. It is not limited to patients or older demographics; it also provides tools for monitoring heart rate during exercise, tracking daily activity levels, and receiving personalized insights for maintaining a healthy lifestyle. By encouraging proactive engagement in health management, the app empowers users to take control of their well-being, regardless of age or medical condition.

We leverage advanced machine learning algorithms and technologies for accurate and personalized anomaly detection. Techniques like the Simple Moving Average (SMA) are employed to smooth heart rate data, reducing noise and highlighting trends for more reliable monitoring. The ARIMA (AutoRegressive Integrated Moving Average) model is utilized to forecast future heart rate values based on historical data, capturing both seasonal patterns and long-term trends. Additionally, unsupervised anomaly detection algorithms, such as Isolation Forests and One-Class Support Vector Machines (OC-SVMs), are used to identify deviations from expected heart rate patterns without relying on labeled data. These algorithms adapt to individual users' heart rate profiles, enabling personalized anomaly detection.

The application integrates with consumer wearable devices like smartwatches and smartphones for real-time heart rate data collection and preprocessing. The trained machine learning models analyze the data, detect potential anomalies, and provide users with timely alerts and insights. By combining cutting-edge machine learning algorithms, personalized modeling, and integration with consumer wearables, our project aims to deliver an accurate and accessible solution for automated detection of cardiac abnormalities, empowering users to proactively manage their cardiovascular health.

2. Problem definition and suggested solution

Cardiovascular diseases (CVDs) are a leading cause of death worldwide, often going undetected until advanced stages. Traditional heart health monitoring is reactive and relies on periodic check-ups or symptoms, leaving a gap in continuous, proactive management. Current consumer-grade devices often lack the sophistication to detect subtle anomalies and provide personalized insights, relying on generic thresholds that can lead to missed warning signs or unnecessary alarms.

Our solution is an AI-driven heart rate monitoring application that bridges the gap between consumer wearables and professional medical monitoring. By leveraging smartphones combined with advanced machine learning algorithms, we aim to provide continuous, personalized heart health monitoring.

Key features include continuous monitoring by interfacing with wearable devices or user mobile to collect heart rate data continuously, personalized analysis through machine learning to establish individual baselines, real-time anomaly detection to identify potential health issues, contextual awareness considering factors like physical activity, time of day, and user-reported events, a user-friendly interface presenting complex health data in an easily understandable format, actionable insights offering personalized recommendations for lifestyle changes or medical attention, and healthcare integration to generate reports for healthcare providers, facilitating data-driven consultations.

This solution transforms heart rate monitoring into an active, continuous process, detecting early signs of cardiovascular issues, encouraging healthier lifestyles, and complementing existing healthcare systems for better-informed medical decisions and improved cardiovascular health outcomes.

Our app will automatically measure heart rate once every 30 minutes, or manually if the user prefers (each measuring takes 10 seconds, we will explain why in the next chapters), and update the database accordingly. Along with recording heart rate, the app utilizes several machine learning algorithms to detect anomalies and forecast heart rate trends based on the collected data. These algorithms help ensure accurate monitoring and provide insights into potential irregularities in the user's heart rate. Detailed explanations of the algorithms we used, how they were implemented, and the process of applying them will be discussed in the later parts of this research.

3. Related work

Heart rate monitoring has become increasingly popular across various domains, including health, wellness, and performance optimization, with applications extending to stress management, fitness tracking, and medical screening. Consumer technology advancements have made heart rate monitoring accessible through smartphone applications. Studies support its efficacy in optimizing cardiovascular fitness and detecting irregular heart rhythms, such as atrial fibrillation, prompting early medical intervention. Additionally, heart rate variability serves as a valuable indicator of stress levels and overall wellness. Leveraging machine learning algorithms enhances the accuracy and predictive capabilities of heart rate monitoring, showcasing innovative approaches to improving healthcare outcomes. For example, ARIMA is used for time-series forecasting, providing insights into future heart rate trends. Isolation Forest and OC-SVM are employed for robust anomaly detection by identifying deviations from normal patterns and boundaries. Related works, such as those by Fitbit and Apple Watch, use similar techniques to offer health insights and detect irregular heart rhythms. Our app aims to excel by integrating these advanced algorithms to personalize monitoring and detect anomalies accurately, providing comprehensive data tracking and visualization. This makes our app a more personalized and effective solution compared to existing heart rate monitoring technologies, addressing some of the limitations of current apps, such as accuracy issues, battery life, and regulatory compliance challenges.

4. Algorithms

In this section, we will explain the algorithms used in the app and demonstrate how we connect them to achieve the best user experience and results. By integrating advanced machine learning techniques and personalized analysis, we aim to provide accurate, real-time heart rate monitoring and actionable insights for our users.

4.1 ARIMA Algorithm

The ARIMA (AutoRegressive Integrated Moving Average) algorithm is a powerful statistical method used for time series forecasting. It combines three key elements: Autoregression (AR), Differencing (I), and Moving Average (MA) to model complex temporal patterns and make accurate forecasts based on past data.

ARIMA is ideal for predicting future values in heart rate monitoring due to its ability to handle time series data with trends and seasonality. This is crucial for timely intervention and health management.

In our app, ARIMA analyzes and forecasts heart rate data collected from users' wearable devices. The app collects heart rate data at 30-minute intervals, converts it to integer format, and stores it in a MongoDB database. The auto_arima function automatically identifies the best parameters for the ARIMA model based on historical data. The trained ARIMA model then forecasts future heart rates to identify any deviations from expected patterns, flagging significant deviations as anomalies. Users are prompted to provide context for these anomalies to refine future predictions and alerts.

By integrating ARIMA, our app provides accurate, real-time monitoring and personalized insights into cardiovascular health, enhancing proactive health management and early detection of potential issues.

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t$$

y_t is the actual value at time t.

c is a constant term (optional).

ϕ_i represents the coefficients for the AR terms.

θ_j represents the coefficients for the MA terms.

ϵ_t is the error term at time t.

4.2 simple Moving average

The Moving Average (SMA) is a statistical technique used to smooth out short-term fluctuations in time series data and to highlight longer-term trends or patterns. It is particularly useful in your heart rate monitoring application for understanding trends and detecting abnormalities.

1. Simple Moving Average (SMA)

The Simple Moving Average (SMA) calculates the unweighted mean of the previous N data points over a given period.

$$\mathrm{SMA}_t = \frac{1}{N} \sum_{i=0}^{N-1} X_{t-i}$$

where:

SMA_t = Simple Moving Average at time t.

N = The window size (number of time periods).

X_{t-i} = The data point (e.g., heart rate) at time $t - i$.

The SMA gives equal weight to all data points in the window, making it useful for smoothing out short-term fluctuations.

4.3 Isolation Forest algorithm

The Isolation Forest algorithm is an effective method for anomaly detection, particularly well-suited for high-dimensional datasets. It works by isolating observations through randomly selecting features and then randomly selecting split values, which effectively separates anomalies from the majority of data points. This approach is particularly robust against high-dimensional data and is efficient in terms of computation.

Isolation Forest is ideal for identifying unusual heart rate patterns or anomalies due to its efficiency in detecting outliers in large datasets. In the context of heart rate monitoring, this capability is crucial for quickly identifying potentially dangerous deviations that could indicate health issues.

In the app, Isolation Forest is employed to analyze heart rate data collected at regular intervals from users. The algorithm processes this data to detect any abnormal patterns that deviate significantly from the norm. When a user's heart rate is updated, the app uses Isolation Forest to assess the new data against historical values. If anomalies are detected, the app flags them and prompts the user to provide context for these deviations. This helps in distinguishing between benign fluctuations and potentially serious health concerns, ensuring timely intervention and better health management.

The anomaly score s(x,n) for a data point x in a dataset with n observations is given by:

$$s(x,n) = 2^{-\frac{E(h(x))}{C(n)}}$$

Where:

$E(h(x))$ is the average path length of x across all isolation trees.

$c(n)$ is the average path length of a random observation in a binary search tree, calculated as:

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n}$$

Here, $H(i)$ is the i-th harmonic number, approximated by $ln(i) + 0.577$ (Euler's constant).

A higher $s(x,n)$ (close to 1) indicates that x is an anomaly.

4.4 OC-SVM algorithm

The One-Class Support Vector Machine (OC-SVM) algorithm is an unsupervised learning method designed for anomaly detection, particularly effective in scenarios where normal data is well-defined but anomalies are rare and diverse. It works by learning a boundary around the normal data points and classifying data points that fall outside this boundary as anomalies. In your HR monitoring app, OC-SVM is utilized to identify unusual heart rate readings that deviate significantly from the established normal patterns. By analyzing the historical heart rate data of each user, the OC-SVM model can detect deviations that may indicate potential health issues or errors in data recording. This approach is valuable for maintaining accurate and reliable health monitoring, enabling timely interventions and enhancing overall user safety by flagging abnormal heart rate patterns effectively.

The goal of OC-SVM is to find a function f(x) such that:

$$f(x) = w^T \phi(x) - \rho$$

Where:

x is the input data.

$\phi(x)$ is the mapping of x to a higher-dimensional feature space.

w is a weight vector.

ρ is the offset (or threshold).

The decision function is:

Decision: if $f(x) \geq 0$, classify x as normal; otherwise , classify x as an anomaly.

The algorithm tries to minimize:

$$\frac{1}{2}||w||^2 + \frac{1}{vn}\sum_{i=1}^{n} max(0, \rho - w^T \phi(x_i))$$

Where:

v is a hyperparameter that controls the trade-off between the regularization term and the proportion of outliers.

n is the number of training samples.

4.5 Integrating HR measurement and the ML Algorithms in the application

Now let's explore how our heart rate monitoring application integrates various machine learning algorithms–namely OC-SVM,SMA, Isolation Forest, and ARIMA–to provide comprehensive heart rate analysis and anomaly detection. We will also delve into how these components are connected and how they work together to deliver accurate and actionable insights.

4.5.1 Measuring Heart Rate

Camera-based Measurement: We use the camera of a smartphone or computer to capture images of the user's finger. The camera captures video frames, and the mean intensity of these frames is analyzed to extract heart rate data. This involves converting each frame to grayscale, calculating the mean intensity, and using peak detection to determine the heart rate in beats per minute (BPM).

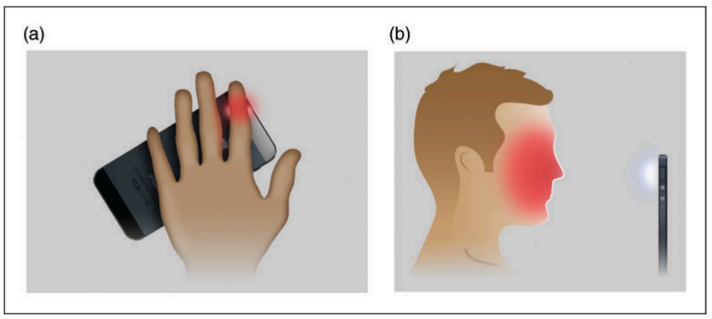


Figure 1. Two different concepts of measuring heart rate by PPG are known: In contact PPG, the subject places a finger on the built-in camera of the phone directly on the skin and the built-in flash provides the necessary light source in the visible range for reflection by blood cells (Figure 1a). In non-contact PPG, the camera is used in the classical way by holding the camera in front of the patient's face without the need for direct skin contact (Figure 1b). (a) Contact photoplethysmography; (b) non-contact photoplethysmography. (taken from European Journal of Preventive Cardiology [18]).

Two different concepts of measuring heart rate by PPG are known:

(a) Contact Photoplethysmography: In this method, the subject places their finger directly on the built-in camera of the phone, with the built-in flash providing the necessary light source in the visible range for reflection by blood cells. This direct contact allows for precise measurement of the changes in light absorption or reflection caused by blood flow.

(b) Non-contact Photoplethysmography: In this approach, the camera is used in the classical way by holding it in front of the patient's face, without the need for direct skin contact. This method relies on capturing light reflected from the skin's surface, providing a more convenient measurement option that does not require physical contact.

This dual concept of PPG measurement enables flexibility in how heart rate can be monitored, making the technology adaptable to different user preferences and scenarios (taken from European Journal of Preventive Cardiology [18]).

How It Works:

- The camera captures video frames of the user's finger, focusing on the area where light from the camera can penetrate the skin.
- Each frame is converted to grayscale to simplify the image processing.
- The mean intensity of each frame is calculated. As blood volume changes with each heartbeat, the intensity of the light reflected from the skin also changes.
- Peak detection algorithms analyze the variations in mean intensity over time to determine the heart rate. The peaks correspond to the heartbeats, and the intervals between peaks are used to calculate the heart rate in beats per minute (BPM).

Integration into the Project: In our project, we have integrated this established technology, which already exists, to leverage its benefits. By combining PPG with our application's other features, we provide users with a seamless and non-invasive way to monitor their heart rate using the camera on their devices. This integration allows us to offer a more accessible and user-friendly heart rate measurement tool, utilizing existing technology to enhance the overall functionality of our application.

```python
async def measure_heart_rate(self, widget=None):
    """
    This function will measure the heart rate, either manually or automatically.
    """
    # Perform any ARIMA forecasting or preprocessing if needed
    arima_forecast_for_user("Ali", 10)

    cap = cv2.VideoCapture(0)  # Open the camera (0 is the default camera)

    if not cap.isOpened():
        await self.main_window.info_dialog('Error', 'Could not open camera.')
        return

    if widget is not None:  # If called via button press, show dialog
        await self.main_window.info_dialog('Measure Heart Rate', 'Place your finger on the camera and press OK to continue.')

    times = []
    signal = []
    bpm = None
    start_time = datetime.now()

    try:
        while (datetime.now() - start_time).total_seconds() < 10:  # Measure for 10 seconds
            ret, frame = cap.read()
            if not ret:
                await self.main_window.info_dialog('Error', 'Failed to capture image.')
                break

            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            mean_intensity = np.mean(gray)
            times.append(datetime.now())
            signal.append(mean_intensity)

            cv2.imshow('Frame', frame)

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        # Calculate BPM
        if len(times) > 1:
            times = np.array(times)
            signal = np.array(signal)
            fps = len(times) / (times[-1] - times[0]).total_seconds()
            peaks, _ = find_peaks(signal, height=np.mean(signal), distance=int(fps * 0.6))
            bpm = len(peaks) / (times[-1] - times[0]).total_seconds() * 60
            self.heart_rate_label.text = f"Heart Rate: {bpm:.2f} BPM"

            # Update the database with the heart rate
            user_name = self.user_name_input.value.strip()
            await self.check_for_discrepancy(user_name, bpm)
            await self.update_heart_rate_in_db(user_name, bpm)

    except Exception as e:
        await self.main_window.error_dialog('Error', f'An error occurred while measuring heart rate: {e}')

    finally:
        cap.release()
        cv2.destroyAllWindows()
```

Figure 2. measure_heart_rate function based on device camera and flashlight.

The measure_heart_rate function uses an existing technology, namely photoplethysmography (PPG), to measure heart rate through a device camera. Here's a brief explanation of how it works:

1. **Camera Setup:** The function starts by opening the default camera using OpenCV (cv2.VideoCapture).
2. **Image Capture:** For 10 seconds, it captures video frames and converts them to grayscale. The mean intensity of each frame is recorded, reflecting blood flow changes under the skin, which are indicative of the heart rate.
3. **Signal Processing:** The recorded intensity values are analyzed to detect peaks, which correspond to heartbeats. The function calculates the beats per minute (BPM) based on the number of peaks detected within the time frame.
4. **Display and Update:** The calculated BPM is displayed on the app, and the heart rate is updated in the database for the user.

This process leverages PPG technology to provide a non-invasive heart rate measurement by using the camera on the device

4.5.2 Storing heart rate data

Once heart rate data is measured, it is stored in a MongoDB database. The data is organized by time and day, creating a time-series of heart rate measurements for each user. This data is crucial for subsequent analysis and anomaly detection, we will explain in the database section further details.

4.5.3 Applying machine learning algorithms

The application integrates several machine learning algorithms to analyze heart rate data, detect anomalies, and forecast future trends. As we explained earlier, the data for each user is stored in a database, including both the initial profile information and the measured heart rate (HR) values. The heart rate data is recorded at 30-minute intervals, ensuring regular updates for each user. So, each user has 48 heart rate (HR) values per day, totaling 336 values per week (48 values x 7 days). These values are recorded every half hour, and the HR data can be updated based on new measurements taken every 30 minutes. The changes to the HR values will be made at the same time and on the same day as the measurement time. Here's how each algorithm is employed:

1- ARIMA: AutoRegressive Integrated Moving Average (ARIMA) is used for time-series forecasting of heart rate data by predicting future values based on historical trends. The process involves preparing the heart rate data for ARIMA analysis, fitting the model to the data using auto-ARIMA for optimal parameter selection, and then generating forecasts to plot expected future heart rate trends. This approach provides valuable insights into potential future patterns in heart rate. Additionally, the algorithm can be adjusted to modify the predicted future values in the database, allowing for dynamic updates and more tailored predictions based on new or changing data.

```python
async def measure_heart_rate(self, widget=None):
    """
    This function will measure the heart rate, either manually or automatically.
    """
    # Perform any ARIMA forecasting or preprocessing if needed

    arima_forecast_for_user(self.user_name_input.value.strip(), 10)

    cap = cv2.VideoCapture(0)  # Open the camera (0 is the default camera)

    if not cap.isOpened():
        await self.main_window.info_dialog('Error', 'Could not open camera.')
        return

    if widget is not None:  # If called via button press, show dialog
```

Figure 3. ARIMA algorithm is called automatically when measuring heart rate.

```python
def update_user_with_forecast(user_name, forecast_series):
    """Update the MongoDB database with forecasted HR values."""
    for time, forecast_value in forecast_series.items():
        hr_key = f"HR at {time.strftime('%H:%M (%A)')}"
        print(f"Updating database: {hr_key} = {forecast_value:.2f}")
        users_collection.update_one(
            {"Name": user_name},
            {"$set": {hr_key: int(forecast_value)}}
        )
    print("Database updated with forecasted HR values.")
```

Figure 4. ARIMA algorithm can change the DataBase according to the forecasted values.

2- Isolation forest: Isolation Forest is utilized for anomaly detection in heart rate data by identifying unusual values that deviate significantly from typical patterns. The process involves preprocessing the heart rate data and feeding it into the Isolation Forest model, which is trained to distinguish between normal and anomalous data points. Once trained, the model detects outliers or anomalies, helping to pinpoint irregularities in heart rate patterns. We also provide an accurate example of the algorithm, demonstrating its effectiveness in correctly identifying anomalies and distinguishing between normal variations and significant deviations in heart rate data.

3- OC-SVM: One-Class Support Vector Machine is used to detect anomalies and outliers in heart rate data by identifying data points that deviate from normal heart rate patterns. The process involves preparing the heart rate data for OC-SVM analysis, training the model to learn the boundary of normal heart rate values, and then using it to detect deviations from this boundary, which helps in identifying potential anomalies.

4- The Simple Moving Average (SMA) : It is important to execute the Simple Moving Average (SMA) algorithm last because it smooths the heart rate data by averaging out short-term fluctuations. This smoothing process could affect the results of other algorithms, such as anomaly detection methods like Isolation Forest or OC-SVM, which rely on the original, more granular data to identify unusual patterns. By running the SMA algorithm last, we ensure that the anomaly detection algorithms operate on unaltered data, providing more accurate and reliable results before any smoothing is applied. Is used in machine learning to smooth time series data, reducing noise

and highlighting trends, which improves data quality for model training. It serves as a feature for capturing patterns and trends, aiding in both predictive accuracy and anomaly detection. SMA can also act as a baseline model to evaluate the performance of more complex algorithms. Additionally, it helps prepare data for time series models like ARIMA and enhances the robustness of anomaly detection methods by providing a reference for normal behavior.

```
# Detect anomalies using Isolation Forest
detect_anomalies(user['_id'])

# Detect anomalies using OC-SVM
detect_anomalies_ocsvm(user['_id'])

# Calculate the moving average forecast
moving_avg_value = moving_average_forecast(heart_rates, window_size=3)

# Prepare forecast
if moving_avg_value is not None:
    # Set the number of future points to forecast
    num_future_points = 10  # Adjust as needed

    # Prepare forecast as a list filled with the moving average value
    forecast = [moving_avg_value] * num_future_points

    # Plot the forecast
    plot_moving_avg_forecast(times, heart_rates, forecast, num_future_points)
```

Figure 5. Simple moving average algorithm executed last.

4.5.4 Workflow integration

1. Data collection and storage: Heart rate data is collected via the camera and stored in MongoDB. The initial data were taken according to academic research, which we will explain in the later sections. This research underpins the methodology used in our analysis and ensures the reliability of our data collection and processing techniques.

2. Data preparation: Historical heart rate data is prepared and structured for analysis and to ensure that the data is cleaned and preprocessed to suit the requirements of each machine learning algorithm.

3. Model training and analysis: SMA and ARIMA models are trained on historical data to forecast future trends. Isolation Forest and OC-SVM models are trained to detect anomalies and outliers.

4. Real-time monitoring: The application continuously monitors heart rate data, applying trained models to detect anomalies and forecast future trends. Users are alerted if their heart rate deviates significantly from expected patterns, based on the output of the anomaly detection models.

5. Visualization and reporting: Forecasts and anomaly detection results are visualized using graphs and reports. The application generates plots to show historical data, forecasts, and detected anomalies, providing users with clear and actionable insights.

By integrating camera-based heart rate measurement with advanced machine learning algorithms, the application offers a robust solution for heart rate monitoring. The use of SMA, ARIMA, Isolation Forest, and OC-SVM ensures accurate forecasting, anomaly detection, and overall trend analysis. This multi-faceted approach provides users with a comprehensive understanding of their heart rate patterns and timely alerts for any potential issues, contributing to better health management and preventive care.

5. DataBase

In our heart rate monitoring application, MongoDB has been employed as the primary database to efficiently handle and organize user data. Upon launching the application, users are prompted to create a new account. During this account creation process, the application dynamically allocates a dedicated space within the MongoDB database for the new user. Essential personal information is collected, including the user's name, age, smoking status, presence of heart problems , ownership of a smartwatch and activity level on a scale of 1 to 5 . Moreover, the application is designed to record the user's heart rate at thirty-minute intervals throughout each day of the week. The initial heart rate values for new users are derived from academic literature, ensuring that the data starts with a scientifically validated baseline,to ensure scalability, security, and availability, we may opt to host the MongoDB database on a cloud service like MongoDB Atlas, which offers features such as automated backups, real-time analytics, and global distribution. This approach ensures that our application can handle large volumes of data while providing fast and reliable access to user information from anywhere.

We collected data on heart rate recordings from three different scientific papers (see to reference [21]). Based on our analysis, we concluded that the average heart rate varies significantly across different ages, genders, activity levels, and health conditions. Children typically have a heart rate ranging from 70-110 bpm, while adults have a resting rate of 60-100 bpm, with women generally on the higher end. As activity levels increase, heart rates can range from 85-170 bpm depending on intensity and age. Individuals with heart conditions generally have a higher resting heart rate, around 70-100 bpm, and must monitor their rates closely during physical activity to avoid exceeding safe limits. Healthy individuals tend to have a lower resting heart rate and can handle higher heart rates during exercise.

Here is a table (Figure 6), among other tables and data, from the academic research that we used to inform and structure the initial heart rate data for our application. This table provides key insights into the variations in heart rate across different demographics, including age, gender, and activity levels, helping us establish baseline values and patterns for heart rate monitoring. The data was essential in building a robust model that captures normal heart rate ranges while detecting anomalies based on individual profiles.

This table explains the data presented in Figure 4 (taken from [21]), which shows heart rate variations according to age, gender, race, and step count in healthy participants. The table provides a detailed breakdown of how these factors influence heart rate, offering valuable insights

that were used to develop accurate baseline measurements in our heart rate monitoring system. By analyzing these variables, we were able to tailor our application to account for demographic differences and activity levels.

Table 2

Heart rate according to age, gender, race, and step count in healthy participants

Baseline characteristics	Number of participants contributing at least one HR-PPG measurement *N*, (%)	Number of HR-PPG measurements Median (IQR)	Geometric mean HR ± SD[a]
Age stratum	*N* = 25,280		
18–20	2197 (8.7)	11.0 (20.0)	81.6 ± 14.0
21–30	6558 (25.9)	15.0 (30.0)	80.2 ± 14.8
31–40	6480 (25.6)	18.0 (37.0)	78.5 ± 15.1
41–50	5620 (22.2)	21.0 (45.0)	75.3 ± 14.3
51–60	3058 (12.1)	23.0 (54.0)	73.9 ± 13.5
61–70	1173 (4.6)	26.0 (54.0)	73.0 ± 12.7
71–80	194 (0.8)	29.5 (72.3)	74.2 ± 11.1
>80	13 (0.1)	36.0 (80.0)	78.1 ± 16.5
Gender	*N* = 18,858		
Females	9476 (50.2)	18.0 (38)	79.3 ± 14.0
Males	9382 (49.8)	21.0 (45)	73.8 ± 14.5
Race or ethnic group	*N* = 18,727		
Non-Hispanic White	14,280 (76.3)	20.0 (43.0)	75.9 ± 14.5
Black or African American	314 (1.7)	12.0 (36.8)	81.4 ± 14.0
Hispanic, Latino, or Spanish origin or ancestry	2001 (10.7)	18.0 (37.0)	78.6 ± 14.3
Asian	1160 (6.2)	17.0 (38.8)	79.2 ± 14.3
Multi-ethnic	526 (2.8)	20.0 (34.0)	78.1 ± 14.4
Other/prefer not to disclose	440 (2.4)	15 (34.0)	78.2 + 13.0
BMI	*N* = 2175		
<18.5	58	21.0 (54.2)	77.9 ± 14.7
≥18.5–25	1027	29.0 (62.0)	74.9 ± 16.5
25–30	686	29.0 (69.0)	73.0 ± 15.1
≥30	404	24.5 (58.8)	80.1 ± 13.3
Daily step count stratum	*N* = 5343		
100–2000	3646 (68.2)	36.0 (66.0)	78.9 ± 14.4
2001–4000	826 (15.4)	35.0 (72.0)	80.0 ± 13.5
4001–6000	422 (7.9)	39.5 (82.0)	79.0 ± 14.8
6001–8000	241 (4.5)	31.0 (77.0)	77.6 ± 16.4
8001–10,000	113 (2.1)	45.0 (80.0)	77.9 ± 17.3
10,001–12,000	64 (1.2)	23.0 (69.3)	78.0 ± 13.9
12,001–14,000	31 (0.6)	31.0 (37.0)	81.9 ± 15.0

HR-PPG heart rate as measured using photoplethysmography, *BMI* body mass index, *HR* heart rate, *SD* standard deviation

[a]All intergroup comparisons were significant ($p < 0.0005$)

Figure 6. Heart rate according to age, gender, race, and step count in healthy participants

The percentile graph of average real-world heart rate measured via photoplethysmography (HR-PPG) provides a visual representation of heart rate trends across various demographics and activity levels. In graph **a**, the percentile distribution is shown according to age, highlighting how average HR-PPG values shift across different age groups. Graph **b** illustrates the percentile breakdown by gender, reflecting gender-specific variations in heart rate patterns. Graph **c** focuses on step count, correlating physical activity levels with heart rate percentiles. Together, these graphs offer comprehensive insights into real-world HR-PPG data, helping to understand heart rate behavior in diverse populations [21][2].

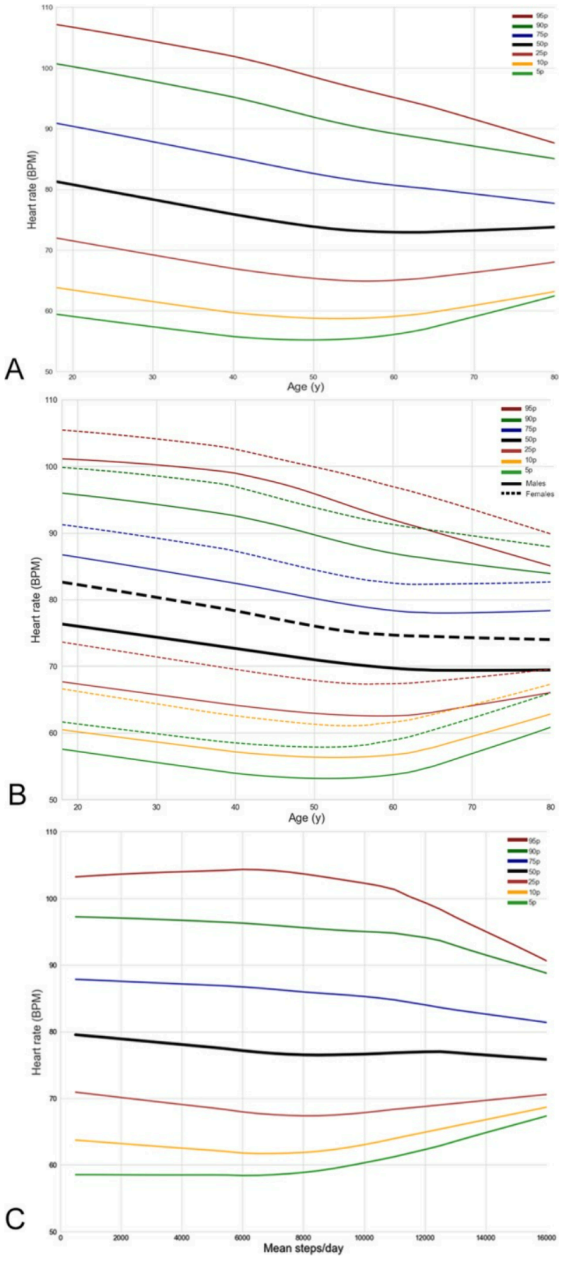


Figure 7. Percentile graph of average real-world HR-PPG. **a** Percentile graph of average real-world HR-PPG according to the age. **b** Percentile graph of average real-world HR-PPG according to the gender **c** Percentile graph of average real-world HR-PPG according to the step counts [21].

We based our heart rate data collection on several factors identified in academic research [21]. For each half-hour interval throughout the day, we determined the heart rate range based on the individual's age and gender. For instance, children and teenagers had a different range compared to adults or seniors.

Additionally, we adjusted the heart rate values according to the person's smoking status or presence of heart problems, which could increase the heart rate. We also factored in the individual's activity level: those with higher activity levels had slightly elevated heart rates.

This method ensures that our heart rate data reflects an initial realistic variation and is grounded in research, as we will elaborate on in the later sections.

When we create a new user, we ask for some personal information, including age, gender, smoking status, presence of heart problems, whether they use a smart watch, and their activity level. This information helps us tailor the heart rate monitoring to their specific needs. We then populate the heart rate data table, which records heart rate measurements every 30 minutes throughout the day. The data is filled in based on reference papers that provide insights into heart rate patterns across different demographics and conditions, ensuring that the monitoring is as accurate and relevant as possible.

```
for _ in range(48):  # 48 half-hour intervals in a day
    for day in ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]:
        #hour = start_time.hour

        # Determine heart rate range based on age and gender
        if 6 <= age <= 14:
            initial_hr = random.randint(70, 110)
        elif 15 <= age <= 60:
            if gender == "male":
                initial_hr = random.randint(60, 100)
            else:
                initial_hr = random.randint(70, 110)
        elif age > 60:
            initial_hr = random.randint(50, 90)
        else:
            initial_hr = random.randint(60, 100)  # Default if age doesn't fit in any category

        # Adjust heart rate based on smoking or heart problems
        if smoking_status or heart_problems:
            initial_hr = random.randint(80, 120)

        # Adjust heart rate based on activity level
        if activity_level == 3 or activity_level == 4:
            initial_hr += 5
        elif activity_level == 5:
            initial_hr += 10

        hr_field = f"HR at {start_time.strftime('%H:%M')} ({day})"
        heart_rate_data[hr_field] = initial_hr
```

Figure 8. Data simulation.

Each time the user measures their heart rate, the table is updated with the new data. If the recorded heart rate significantly deviates from the existing data in the database, we prompt the user with questions to understand the reason behind the discrepancy. This helps ensure accurate tracking and allows us to address any potential issues or anomalies in the heart rate data.

In our application, machine learning algorithms, such as ARIMA, are not only used for forecasting heart rate trends but also for dynamically adjusting specific heart rate values in the user database. By analyzing historical data and predicting future values, ARIMA enables us to refine and update the heart rate records for individual users based on anticipated trends. This allows the application to provide more accurate and personalized data, as the algorithm can modify heart rate values in the database according to its predictions, ensuring that the information remains relevant and aligned with the latest trends and patterns.

```
Updating database: HR at 23:00 (Thursday) = 92.75
Updating database: HR at 23:30 (Thursday) = 92.85
Updating database: HR at 00:00 (Friday) = 92.85
Updating database: HR at 00:30 (Friday) = 92.85
Updating database: HR at 01:00 (Friday) = 92.85
Updating database: HR at 01:30 (Friday) = 92.85
Updating database: HR at 02:00 (Friday) = 92.85
Updating database: HR at 02:30 (Friday) = 92.85
Updating database: HR at 03:00 (Friday) = 92.85
Updating database: HR at 03:30 (Friday) = 92.85
Database updated with forecasted HR values.
```

Figure 9. updating the database according to the ML forecasting algorithm for the specific user with a successful feedback message .

Here is a snapshot of the database for the user named Sila (See Fig. 10). The database includes detailed personal information about Sila, such as age, gender, and health status, alongside heart rate measurements recorded every half-hour throughout the week. For instance, you can observe the heart rate data for each day, including the change observed at 7:30 AM on Sunday (See Fig.11). This particular entry indicates a significant variation in heart rate, likely due to physical activity, such as sports, that Sila engages in at this time on Sundays. The data captures these fluctuations, providing valuable insights into Sila's weekly activity patterns.

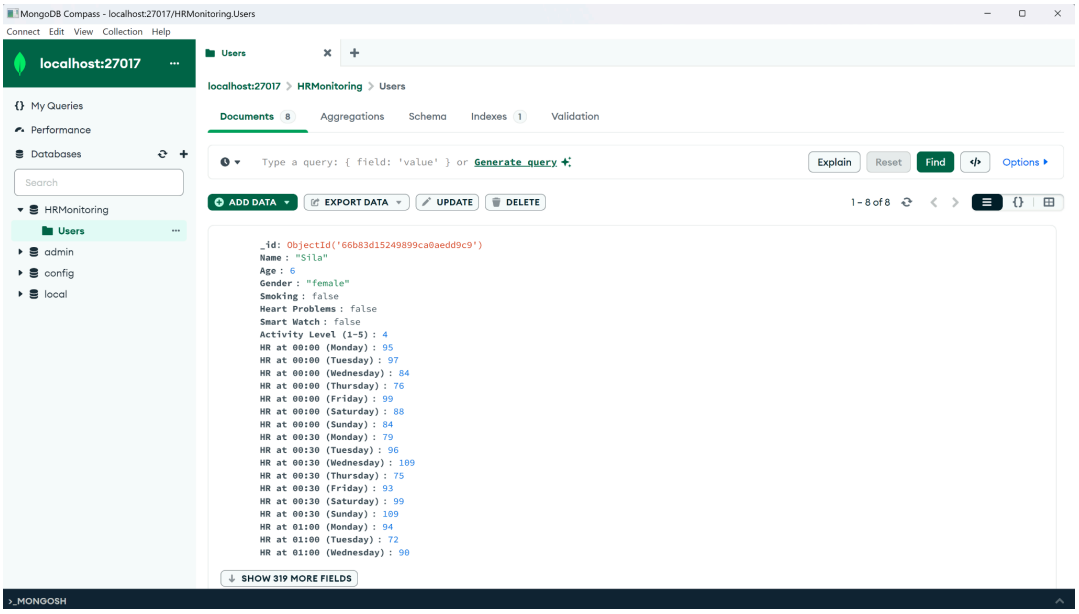


Figure 10. The snapshot of the database for the user named Sila shows both personal information and detailed heart rate measurements taken every half-hour throughout the week

```
HR at 07:30 (Thursday) : 107
HR at 07:30 (Friday) : 81
HR at 07:30 (Saturday) : 99
HR at 07:30 (Sunday) : "150"
HR at 08:00 (Monday) : 120
HR at 08:00 (Tuesday) : 111
HR at 08:00 (Wednesday) : 120
HR at 08:00 (Thursday) : 103
```

Figure 11. The heart rate of the user at 7:30 AM on Sunday is elevated, likely indicating that Sila engages in physical activity at this time. This snapshot effectively highlights how daily activities influence heart rate patterns.

6. Product

In this section, we will explain the structure of the code in our application, detailing how the various components are organized and interact with each other. We will also explore the design and functionality of the user interface, ensuring a seamless user experience. Additionally, we will discuss how the application interacts with external systems and services, providing insight into the integration process and the flow of data between our app and external platforms.

6.1 The code structure

The code (you can find it in our GitHub) is designed for heart rate monitoring and anomaly detection, particularly focused on integrating with MongoDB for user data storage. Here's a breakdown:

1. Libraries Imported: pymongo, datetime, random, sys, cv2, numpy, scipy, and custom machine learning modules like OCSVM and IsolationForests.

2. Database Connection: connects to a MongoDB database (HRMonitoring), specifically to the Users collection.

3. User Management: functions to create new users, read existing users by name, and update heart rates for specific times and days.

4. Heart Rate Analysis: uses models like OC-SVM, Isolation Forest, ARIMA to detect anomalies in heart rates and forecast trends.

5. Heart Rate Measurement: measures heart rate using a camera, calculates beats per minute (BPM), and updates the database accordingly.

6. User Interaction: prompts users to create profiles, input heart rate data, and offers the ability to view and analyze heart rate information.

7. Anomaly Detection: if abnormal heart rates are detected, the user is asked to provide context (e.g., sleeping, exercising) to help understand the cause.

This code is part of an application that monitors heart rate patterns, detects anomalies, and provides health recommendations based on user input and real-time data.

6.2 Architecture overview

The heart rate monitoring app's architecture comprises essential modules: Data Acquisition, Preprocessing, User Profiling, Anomaly Detection, Alerts, User Interface, and Data Storage. Data Acquisition gathers heart rate data, Preprocessing cleans and normalizes it, while User Profiling establishes baseline patterns. Anomaly Detection monitors deviations using methods like Isolation Forests, triggering Alerts. The User Interface displays data, and Data Storage ensures secure management. Optionally, Cloud Integration enables scalability. This architecture ensures efficient personalized heart rate monitoring and anomaly detection in the app (see Figure 12).

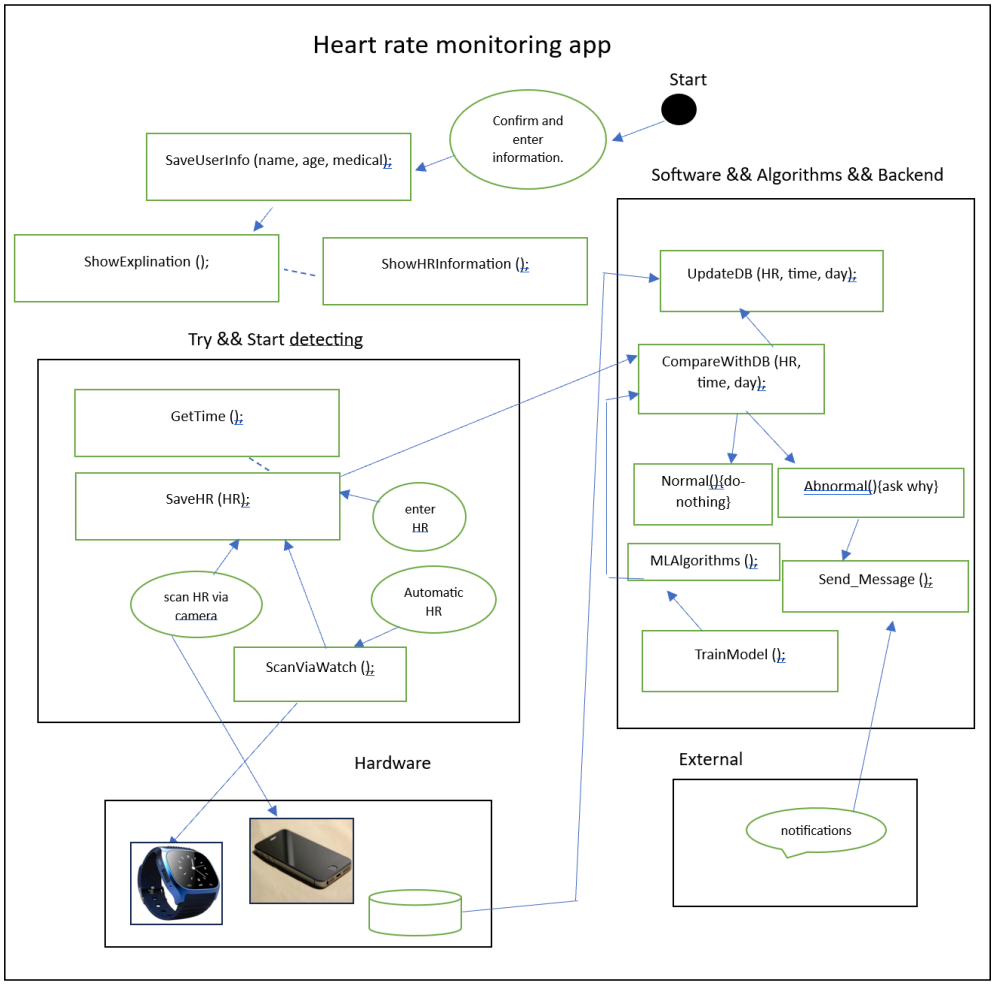
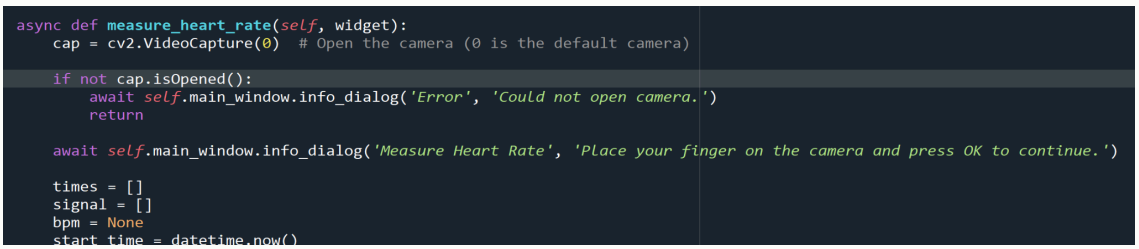


Figure 12. The architectural diagram illustrates the comprehensive system design, depicting the interconnected components, data flows, and underlying technologies that collectively enable the seamless operation of the heart rate monitoring application.

Heart rate measurement app involves the use of a device's camera to monitor heart rate through a technique called photoplethysmography (PPG). Here's a more detailed explanation:

1. **Camera Functionality:** The app utilizes the device's camera, which captures continuous video frames of the user's finger or face. This is because these areas can show subtle color changes related to blood flow.
2. **Color Detection:** As blood pulses through the skin with each heartbeat, it causes slight variations in skin color due to changes in blood volume. The camera detects these color changes, even though they are often subtle.
3. **Image Processing:** The app processes the captured video frames in real-time. It converts each frame to grayscale and calculates the mean intensity, which correlates with the changes in blood flow.
4. **Signal Analysis:** The mean intensity values over time are analyzed to detect periodic fluctuations. These fluctuations correspond to heartbeats. The app identifies peaks in these fluctuations to determine the heart rate.
5. **Heart Rate Calculation:** By measuring the intervals between these peaks and applying algorithms, the app calculates the beats per minute (BPM), providing a real-time estimate of the user's heart rate.

This approach leverages the device's built-in camera to offer a non-invasive, convenient method for heart rate monitoring without needing additional hardware.



```
async def measure_heart_rate(self, widget):
    cap = cv2.VideoCapture(0)  # Open the camera (0 is the default camera)

    if not cap.isOpened():
        await self.main_window.info_dialog('Error', 'Could not open camera.')
        return

    await self.main_window.info_dialog('Measure Heart Rate', 'Place your finger on the camera and press OK to continue.')

    times = []
    signal = []
    bpm = None
    start_time = datetime.now()
```

Figure 13. Code for measuring heart rate using camera and flashlight.

6.3 Interface

Now, we will introduce the proposed interface (we showed these Figs in Phase A book, later we are going to present the actual application Figs) for the app. In the initial screen will see a welcome screen and a condition agreement (see Figure 14), and in the next screen the user will encounter terms and conditions (see Figure 15), then the user will enter some personal information (see Figure 16). In the next screen the user will see a recommendation to install the app on his smartwatch (see Figure 17), then the user will see a congratulations screen (see Figure 18), the user can also see some information about heart rate (see Figure 19 and 20).Next the app will offer the

user to enter manually his heart rate (if he is using an external device, see Figure21) or simply use his phone camera (see Figure 22). If the measured heart rate is abnormal the app will tell the user and give him the option to explain this abnormal heart rate (see Figure 23 and 24), accordingly the app will show the user a suited message (see the proposed GUI in Figures 25-29).

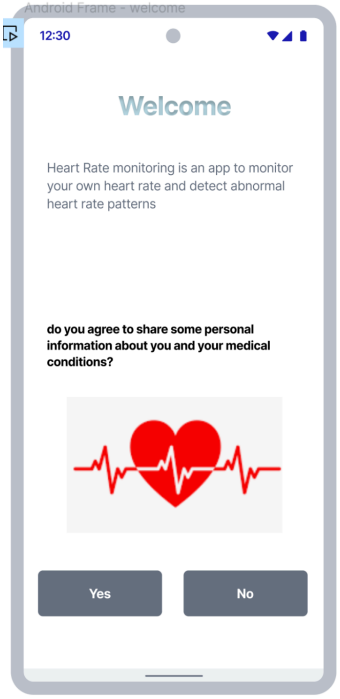


Figure 14. The initial welcome screen.

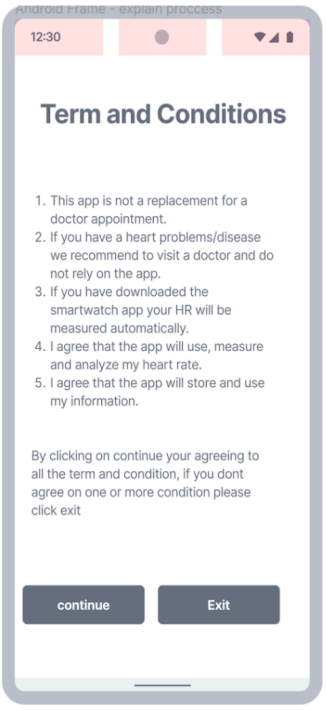


Figure 15. Terms and conditions.

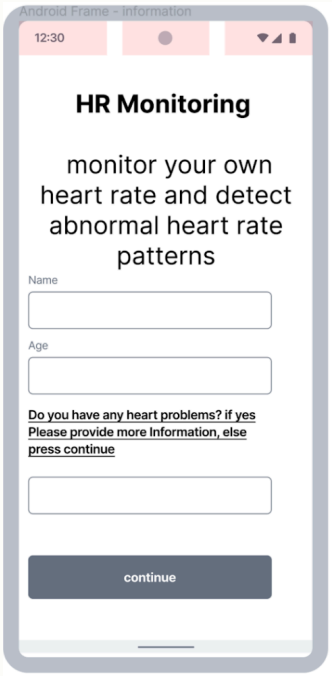


Figure 16. Personal information.

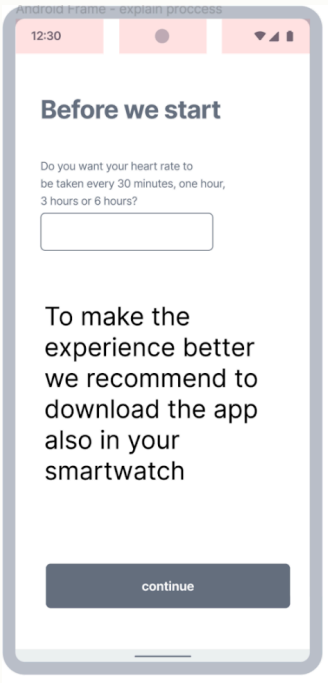


Figure 17. Recommendation.

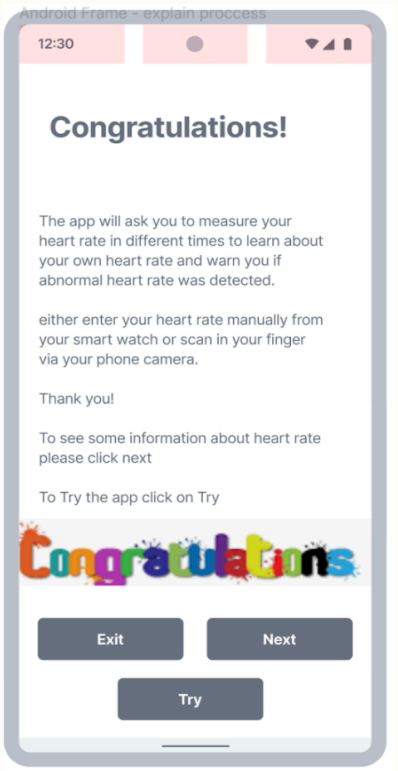


Figure 18. Congratulations.

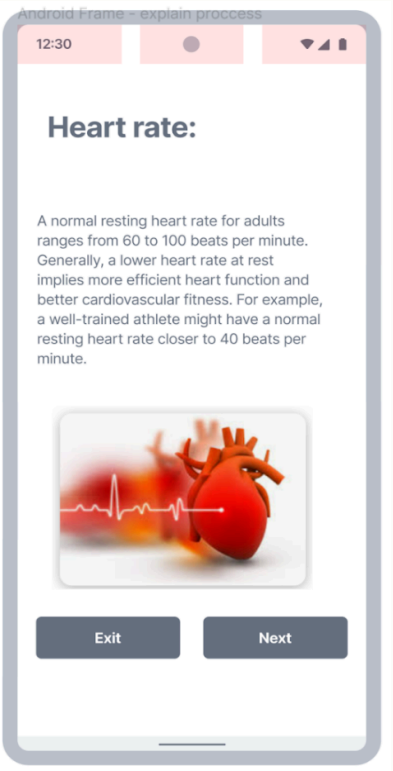


Figure 19. Heart rate information 1.

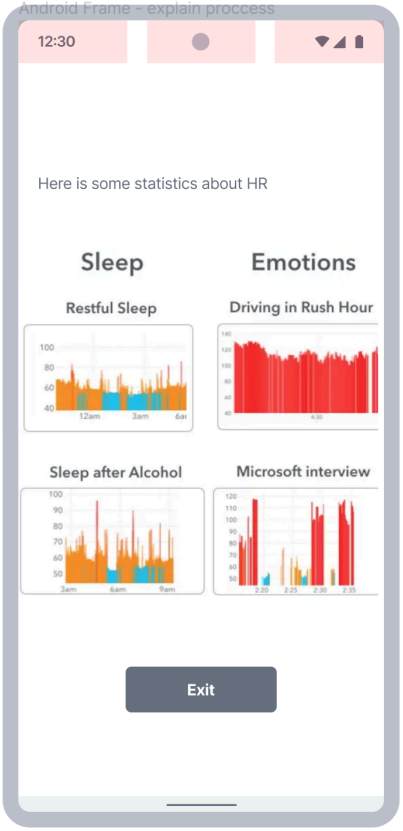


Figure 20. Heart rate information 2.

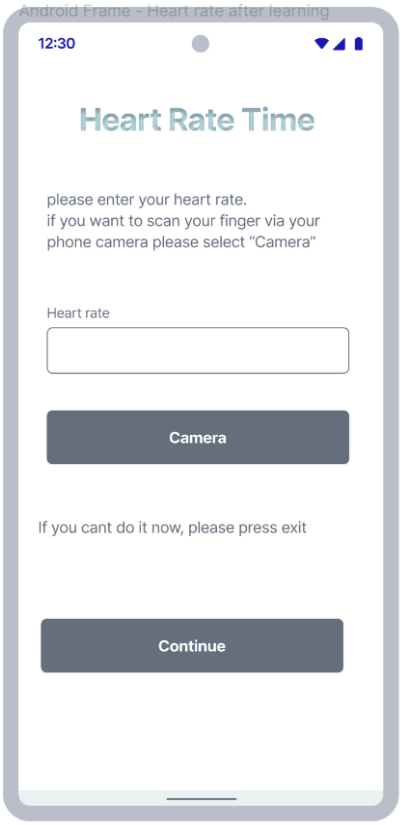


Figure 21. Enter Heart rate.

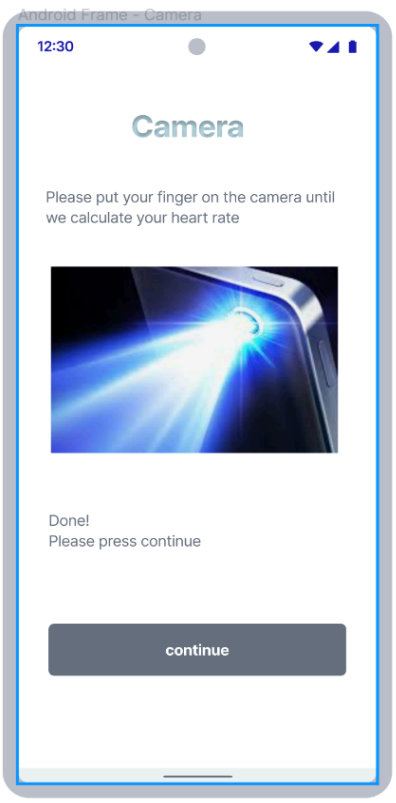


Figure 22. Measure heart rate via camera.

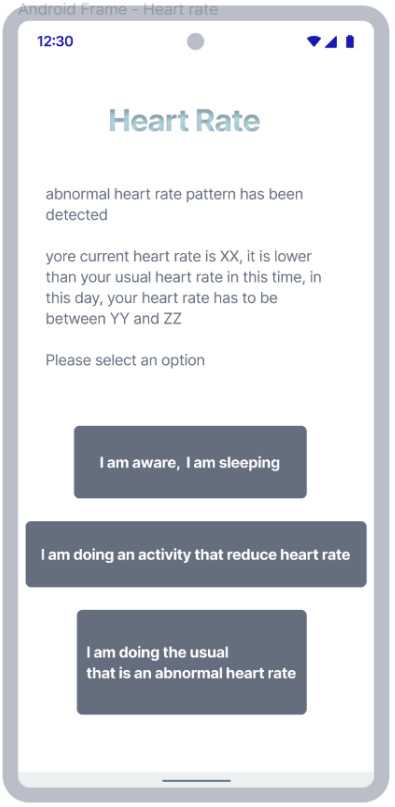


Figure 23. Low heart rate.

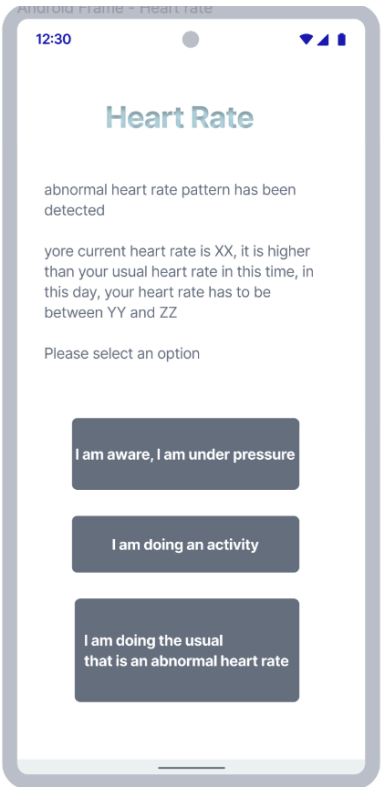


Figure 24. High heart rate.

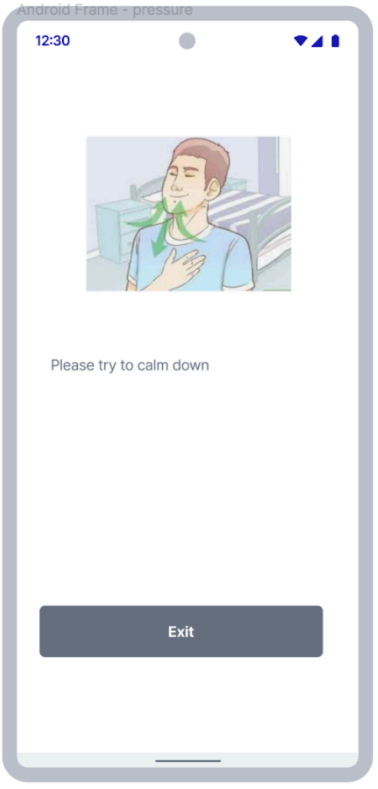


Figure 25. Recommendation: calm down.

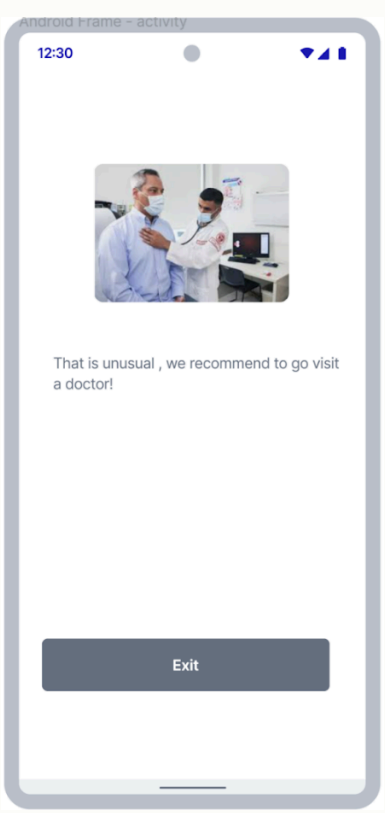


Figure 26. Abnormal heart rate detected.

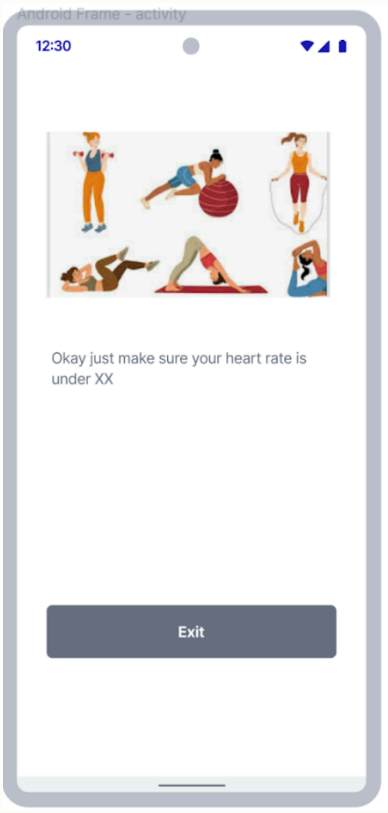


Figure 27. Exercise.

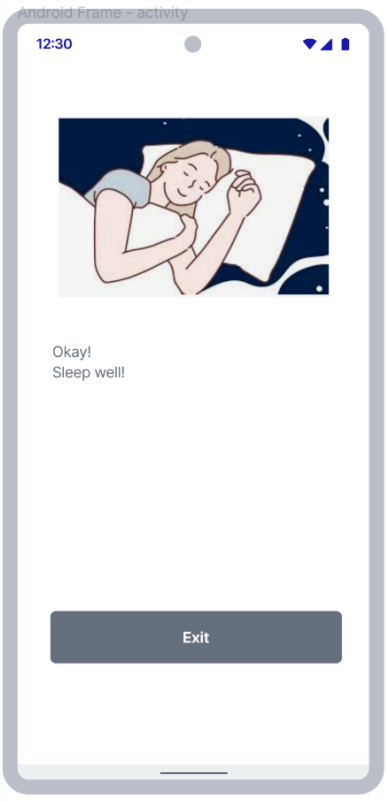


Figure 28. Sleeping.

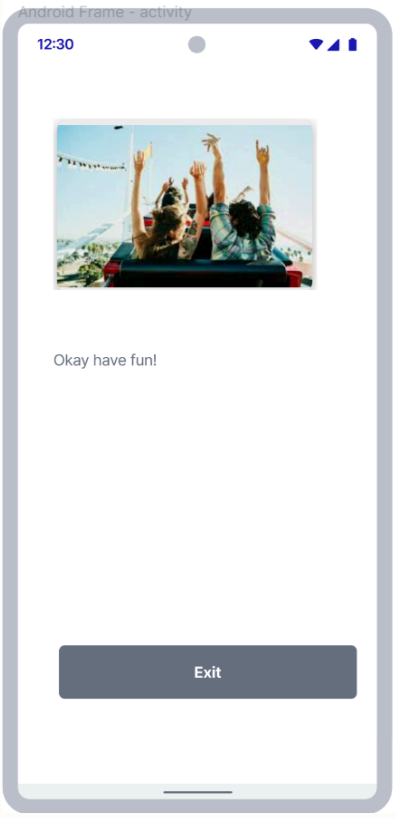


Figure 29. Activity.

6.4 Use case

In the following use case diagram, we illustrate the interactions and relationships between users and the system in the context of Heart Rate monitoring application.

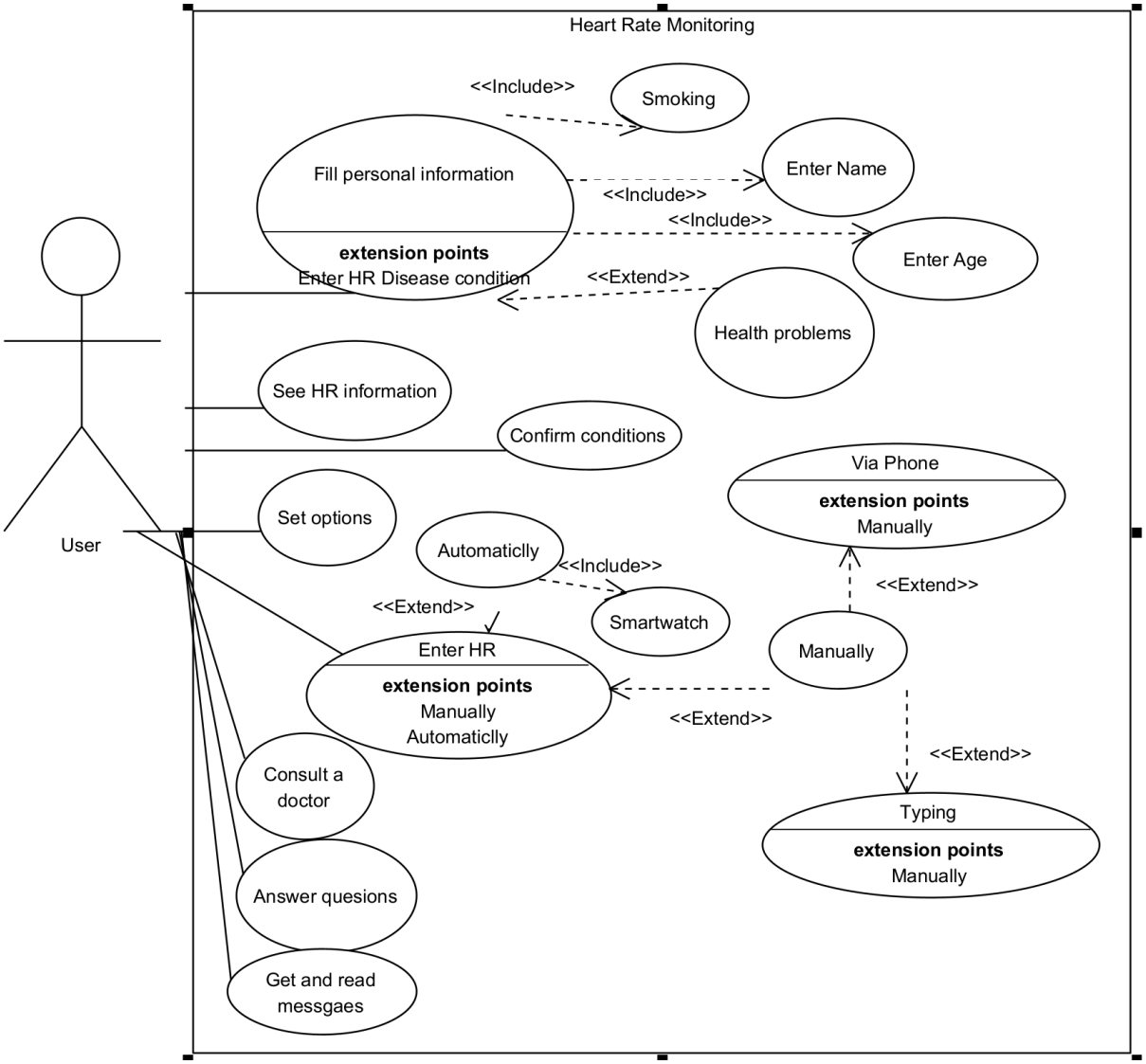


Figure 30. Use Case Diagram: Heart Rate Monitoring Application Depicting the Interactions and Functionalities for the user.

6.5 Activity Diagram

In the following activity diagram, we depict the sequence of activities and flow of control within the Heart Rate monitoring application. This diagram provides a visual representation of the steps involved in accomplishing a particular task or process.

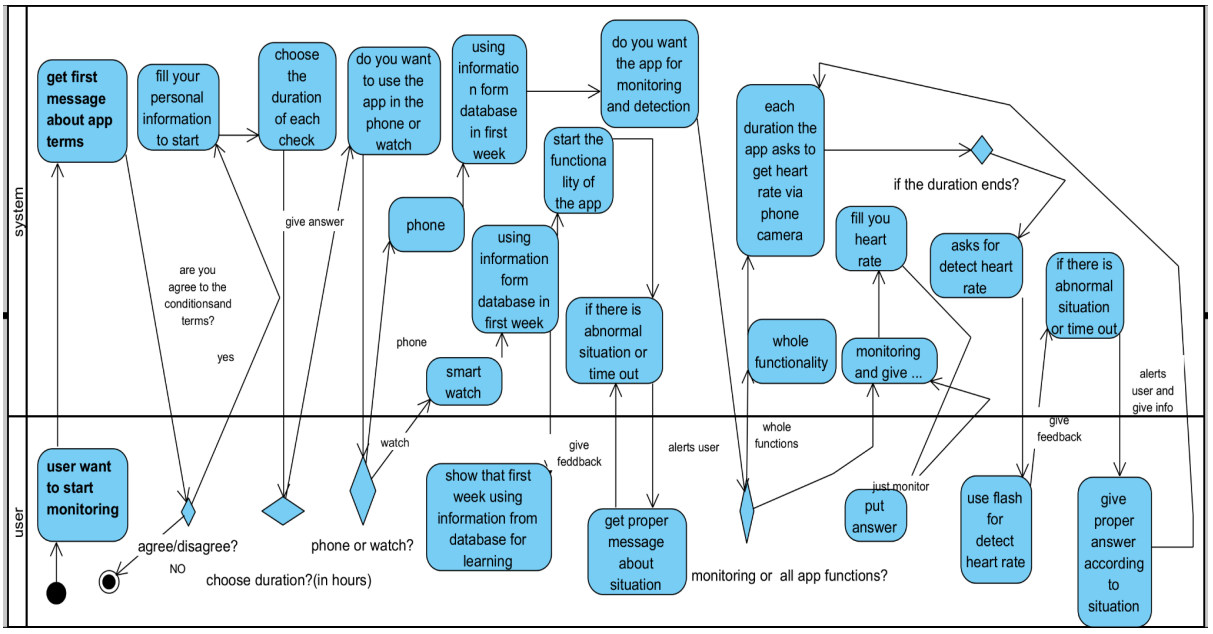


Figure 31. Activity Diagram: Heart Rate Monitoring Application Workflow Illustrating the Sequence of Activities and Data Flow for Comprehensive Heart rate monitoring.

6.6 Toga and BeeWare

Toga is a Python library that is part of the BeeWare suite, serving as a cross-platform graphical user interface (GUI) toolkit designed to enable developers to build native desktop and mobile applications using Python. Toga provides a consistent API for creating and managing windows, widgets (such as buttons, labels, and text inputs), and other interface elements across various platforms including Windows, macOS, Linux, iOS, and Android. It utilizes native widgets to ensure applications have a standard look and feel on each target platform, allowing developers to maintain a single codebase that can be deployed across multiple platforms without rewriting code. The library is designed to be easy to use, featuring a straightforward API that is accessible even to developers who are new to Python or GUI development. BeeWare, an open-source suite of tools and libraries, supports the development of cross-platform applications in Python. It aims to facilitate the creation of Python applications that operate natively across all major platforms, including desktop, mobile, and web applications. BeeWare includes several components: Toga for building native applications, Briefcase for packaging Python code into standalone native applications, and Batavia, a JavaScript interpreter enabling Python code to run in web browsers. To convert a Python application developed with Toga into a mobile app, you can use BeeWare's Briefcase tool. First,

ensure Briefcase is installed using pip install briefcase. Then, initialize a new project with the briefcase new command, which guides you through setting up your project with options such as project name and app description, allowing you to select Toga as the app template. After creating the project, move your existing code into the new project folder, ensuring the main application file aligns with the specified entry point. Configure the mobile platforms by setting up Android with Android Studio and configuring the necessary environment variables. Use the command briefcase create android to create the required files for Android development. To build and run the app, use briefcase build android and briefcase run android to test the application on an emulator or connected device. Finally, test and debug the app to ensure functionality and UI adaptation, and package it for distribution by generating an APK for the Google Play Store or using Xcode to distribute on the Apple App Store.

6.7 Results

Now, let's explore how our code (the code can later be converted easily to a phone application with all the functionalities of the code along with the proposed interface, explanation will be provided) interacts with the user. We will examine the different ways in which the user engages with the application, including input methods, feedback mechanisms, and overall user experience. This section will provide a clear understanding of how our code responds to user actions, processes their inputs, and delivers the necessary outputs, ensuring a smooth and intuitive interaction between the user and the system.

When a new user signs up, the app initiates a simple onboarding process. First, it will ask the user a few personal questions to tailor the experience to their specific needs, followed by a request to agree to the terms and conditions. The app will also offer to show the user some general heart rate information. Once this initial setup is complete, the app will recognize the user in future sessions and immediately start the heart rate monitoring and detection process.

We began with the intention of developing an Android app to monitor heart rates, initially focusing on using native Android development tools like Java or Kotlin. However, our research led us to discover the BeeWare suite and its Toga framework, which offered a unique opportunity to create a cross-platform app using Python. Given that our application heavily relies on machine learning algorithms for heart rate analysis, we decided to pivot to Toga. Python's extensive ecosystem of libraries for machine learning, such as TensorFlow, Scikit-Learn, and PyTorch, made it a natural choice. By using BeeWare and Toga, we were able to streamline development, leverage powerful ML tools, maintain a single codebase, and deploy the app across both Android and iOS, achieving broader platform compatibility while integrating sophisticated data analysis capabilities.

Now, we are excited to present the graphical user interface (GUI) that we developed with the help of Toga and BeeWare. This GUI integrates seamlessly with our heart rate monitoring application, providing users with an intuitive and engaging experience.

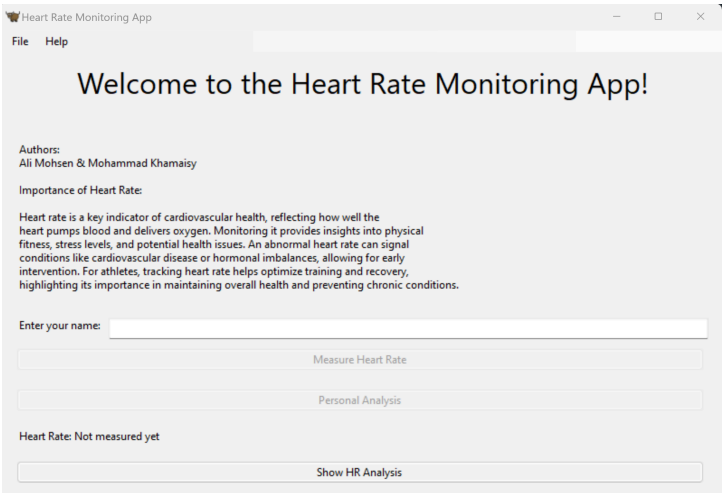


Figure 32. Welcome (main) page of the application that has the whole functionality of the project.

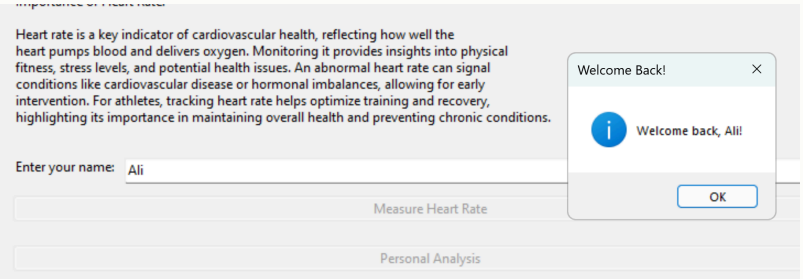


Figure 33. Log in for an existing user and proper message.

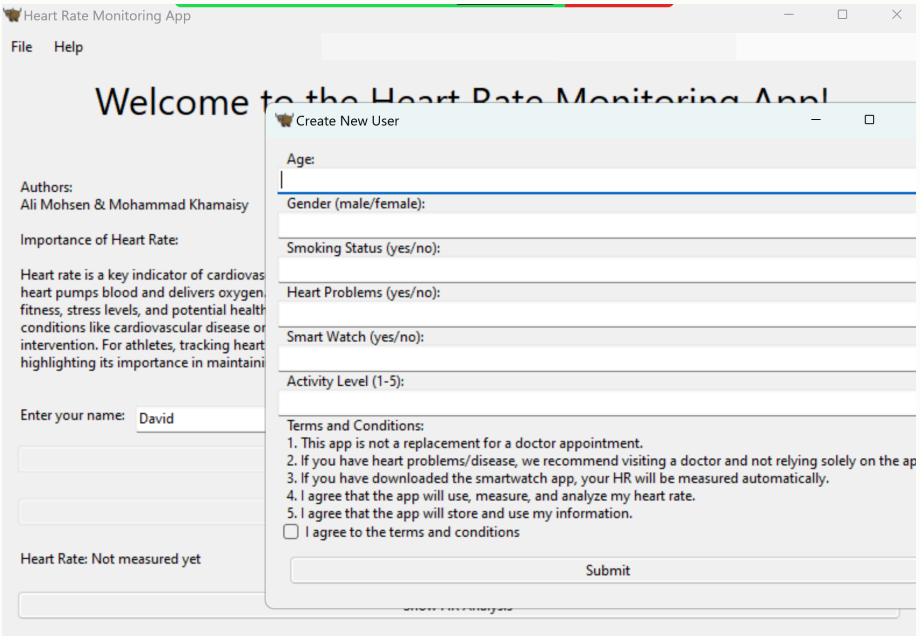


Figure 34. Enter a non existing user name and create a user window that opens automatically , register page for a new user in the application, data saved in MongoDB.

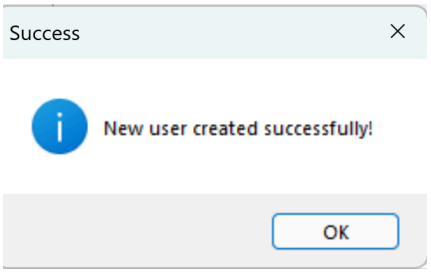


Figure 35. Message for a successful registration for a new user.

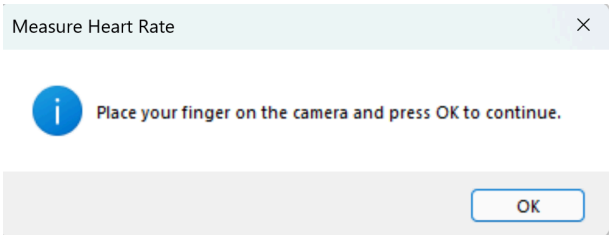


Figure 36. Show a guide message for the user to detect his heart rate using Flashlight in the system.

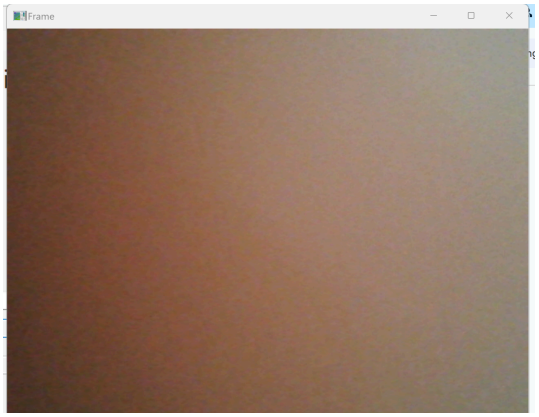


Figure 37. Execute and detect heart rate using a camera by putting the finger on the camera.

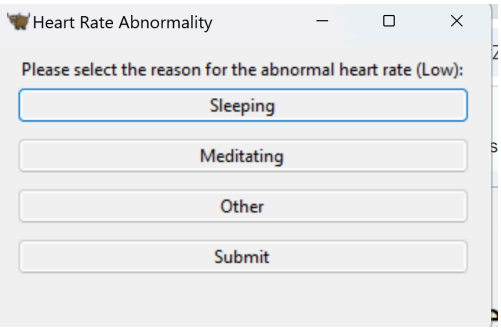


Figure 38. Alert users when an anomaly pattern is detected based on current heart rate and ask for the reason.

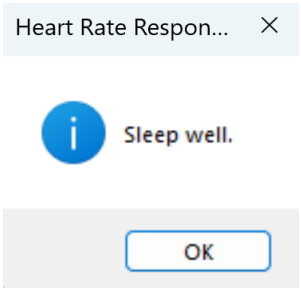


Figure 39. Guide and advice message based on low heart rate.

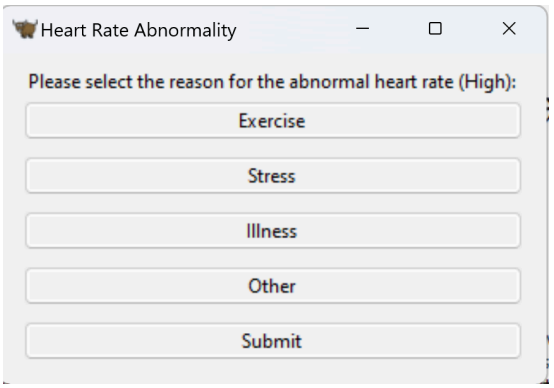


Figure 40. Alert users when an anomaly pattern is detected based on current heart rate and ask for the reason.

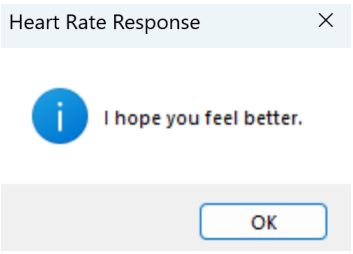


Figure 41. Guide and advice message based on high heart rate .

```
 plt.figure(figsize=(14, 6))
Updating database: HR at 23:30 (Thursday) = 92.77
Updating database: HR at 00:00 (Friday) = 92.87
Updating database: HR at 00:30 (Friday) = 92.88
Updating database: HR at 01:00 (Friday) = 92.88
Updating database: HR at 01:30 (Friday) = 92.88
Updating database: HR at 02:00 (Friday) = 92.88
Updating database: HR at 02:30 (Friday) = 92.88
Updating database: HR at 03:00 (Friday) = 92.88
Updating database: HR at 03:30 (Friday) = 92.88
Updating database: HR at 04:00 (Friday) = 92.88
Database updated with forecasted HR values.
```

Fig 42. Updating database for the user based Arima model predicting .

```
Moving Average plot saved as 'moving_avg_plot.png'
Model accuracy: 0.09
Isolation Forsests:
Anomalies detected at the following times:
HR at 00:30 (Saturday): 117
HR at 01:00 (Wednesday): 117
HR at 02:30 (Tuesday): 117
HR at 02:30 (Sunday): 120
HR at 03:30 (Tuesday): 115
HR at 05:00 (Sunday): 120
HR at 07:30 (Sunday): 150
HR at 08:00 (Monday): 120
HR at 08:00 (Wednesday): 120
HR at 09:00 (Wednesday): 118
HR at 09:30 (Wednesday): 43.490056759597906
HR at 10:00 (Sunday): 58.74323465006061
HR at 10:30 (Tuesday): 114
HR at 10:30 (Sunday): 58.462295473313176
HR at 13:30 (Tuesday): 120
```

Figure 43. Isolation Forest for anomaly detection in heart rates .

```
OCSVM:
Anomalies detected:
At time HR at 00:30 (Monday): Heart rate 84
At time HR at 00:30 (Tuesday): Heart rate 102
At time HR at 01:30 (Sunday): Heart rate 87
At time HR at 02:00 (Sunday): Heart rate 86
At time HR at 02:30 (Wednesday): Heart rate 87
At time HR at 02:30 (Sunday): Heart rate 120
At time HR at 05:00 (Tuesday): Heart rate 87
At time HR at 05:00 (Sunday): Heart rate 120
At time HR at 06:00 (Tuesday): Heart rate 102
At time HR at 06:00 (Saturday): Heart rate 87
At time HR at 06:30 (Monday): Heart rate 87
At time HR at 06:30 (Tuesday): Heart rate 102
At time HR at 07:00 (Sunday): Heart rate 84
At time HR at 07:30 (Sunday): Heart rate 150
At time HR at 08:00 (Monday): Heart rate 120
At time HR at 08:00 (Wednesday): Heart rate 120
At time HR at 09:00 (Friday): Heart rate 85
```

Figure 44. OCSVM model anomalies detection and prints abnormal heart rates.

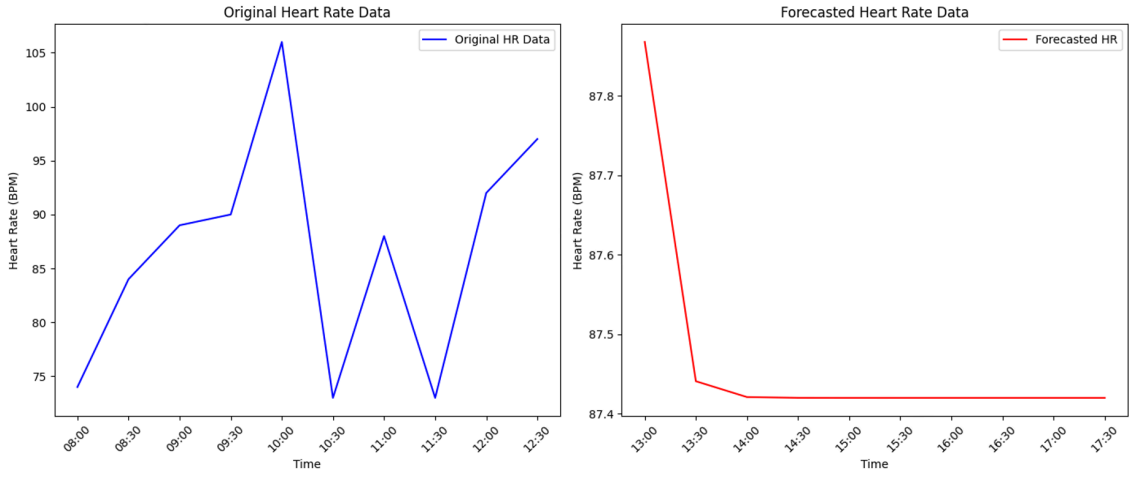


Figure 45. Diagram for historical heart rate and predicted value in future based on ARIMA model .

The plot generated by the algorithm provides a comprehensive view of both the historical and forecasted heart rate (HR) data for a user. The left subplot displays the original HR data collected over a series of half-hour intervals, plotted against time labels. This visualization allows for the examination of past heart rate trends and variations. The right subplot presents the forecasted HR values generated by the ARIMA model, showing predictions for future heart rate trends.

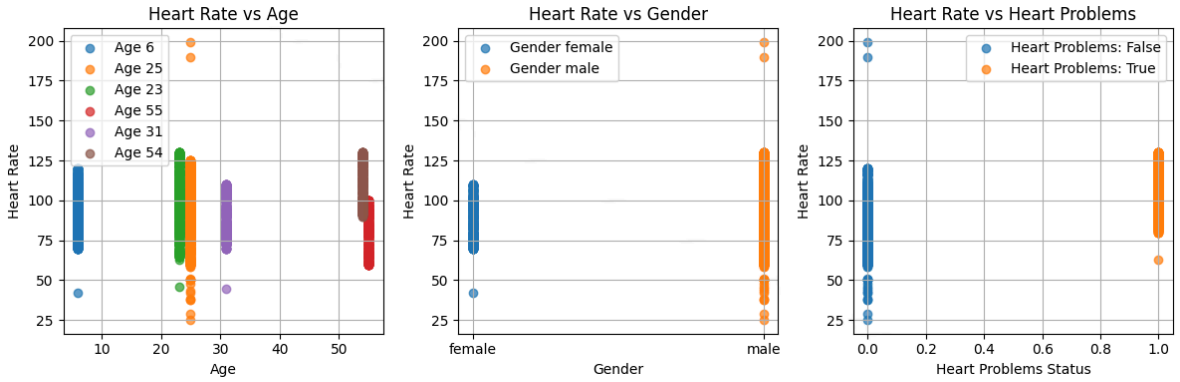


Figure 46. analysis for different groups between heart rate values and other parameters.

We also show the average heart rate of the users according to gender and age and standard deviation for each category.

```
Overall Average Heart Rate: 94.20
Average Heart Rate for female: 87.80, Standard Deviation: 0.00
Average Heart Rate for male: 94.68, Standard Deviation: 11.82
Average Heart Rate for Age Group 90+: 93.62, Standard Deviation: 8.23
Average Heart Rate for Age Group 20-30: 94.50, Standard Deviation: 12.72
Average Heart Rate for Age Group 50-60: 91.49, Standard Deviation: 16.11
Average Heart Rate for Age Group 30-40: 97.74, Standard Deviation: 10.73
```

Figure 47. Average heart rate of all the users, average heart rate for female and male users, and the average heart rate of users by age and standard deviation.

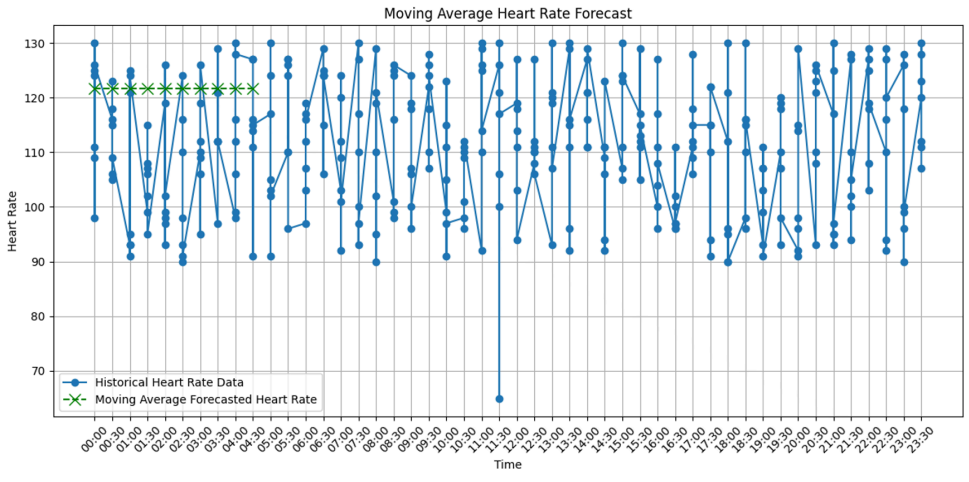


Figure 48. Graph for future heart rate predictions based on MOVING AVERAGE.

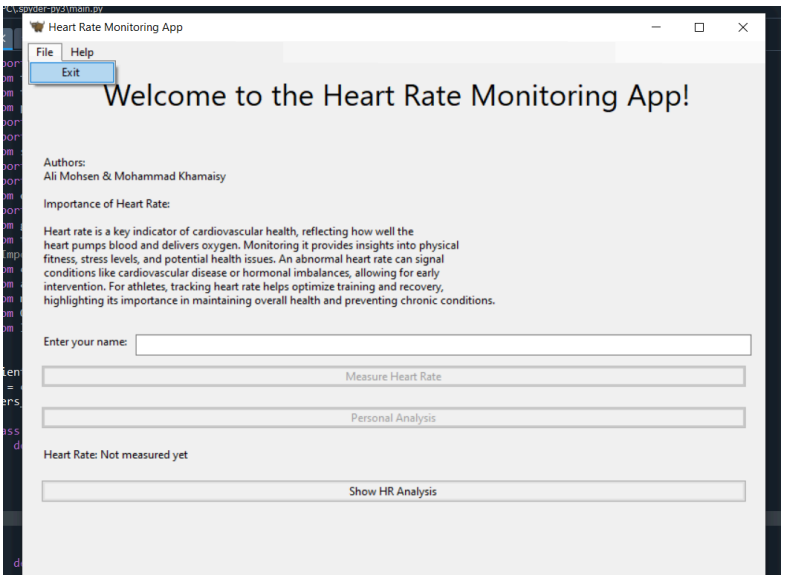


Figure 49. Exit button is an option in the top side of the first app page.

7. Verification and Evaluation

In this section, we delve into the critical aspects of testing and evaluating the functionality of our application. Testing is an essential phase of the development process, aimed at verifying that each component and feature performs as intended. We will explore various testing strategies, including unit testing to ensure individual functions operate correctly, integration testing to confirm seamless interaction between components, and system testing to validate the overall application functionality. Furthermore, we will assess performance to gauge how the application handles different loads and examine user acceptance to ensure usability and accessibility. Security testing will also be discussed to ensure the application is robust against potential threats. By systematically evaluating these elements, we can ensure that the application meets its intended goals and delivers a reliable, user-friendly experience.

7.1 Unit Testing

The primary objective is to ensure that individual functions work as expected. This testing will be conducted using Python's unittest or pytest frameworks. For instance, the function read_user_by_name(name) is tested to verify its ability to correctly identify when a user exists in the database. In one test case, if the input name is "John Doe," the expected output is a user object with the name "John Doe." Another test case checks the function's behavior when the user does not exist in the database, where the input is "Nonexistent User," and the expected output is None.

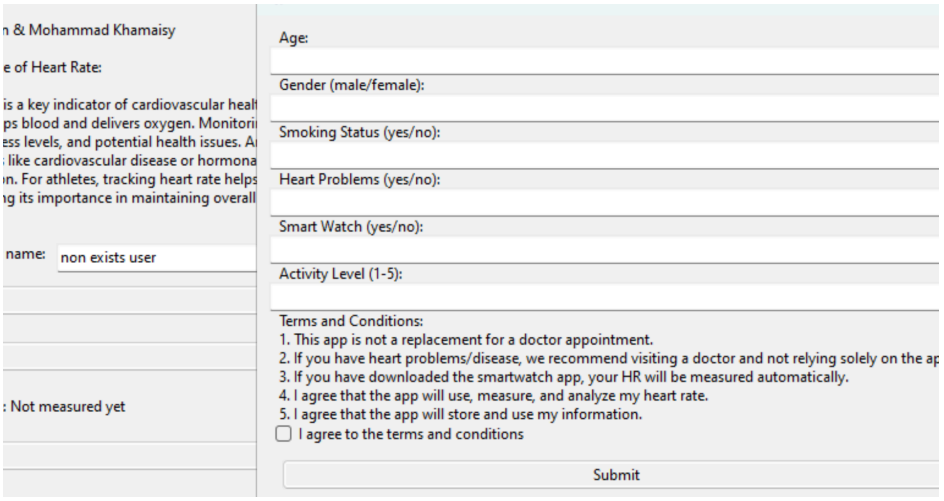


Figure 50. Registering page for new users to fill their personal information.

For the function create_new_user(name, age, gender, smoking, heart_problems, smartwatch, activity_level), testing ensures it handles valid and invalid input data correctly. For valid data, such as "Jane Doe" (30, female, non-smoker, no heart problems, with a smartwatch, and an activity level of 4), the expected outcome is the successful insertion of a new user into the database and the creation of appropriate heart rate ranges. If the age input is invalid, like -5, the function is expected to raise a ValueError.

We also test if the function can that recognize an existing user in the DataBase

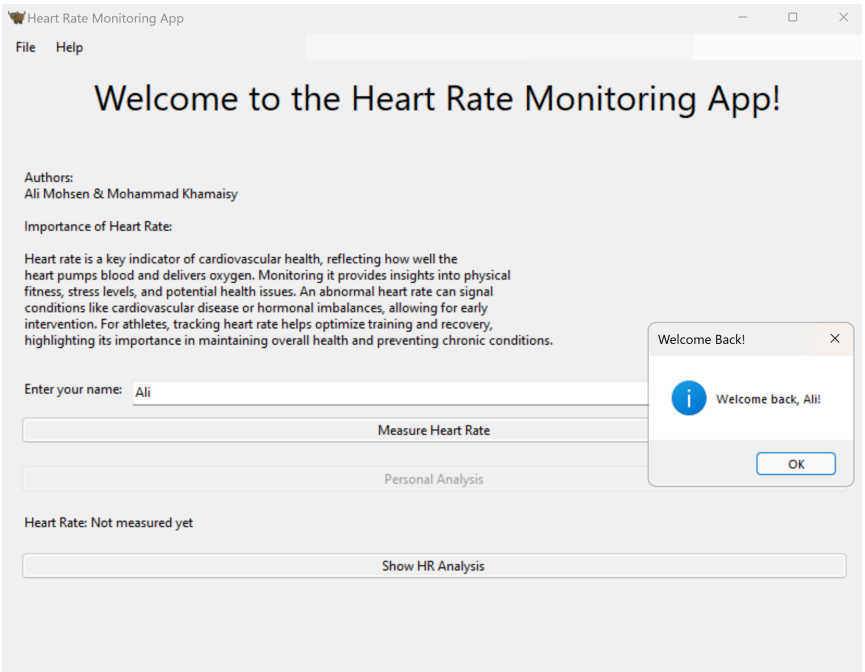


Figure 51. Message that realizes entering for Existing user in the system.

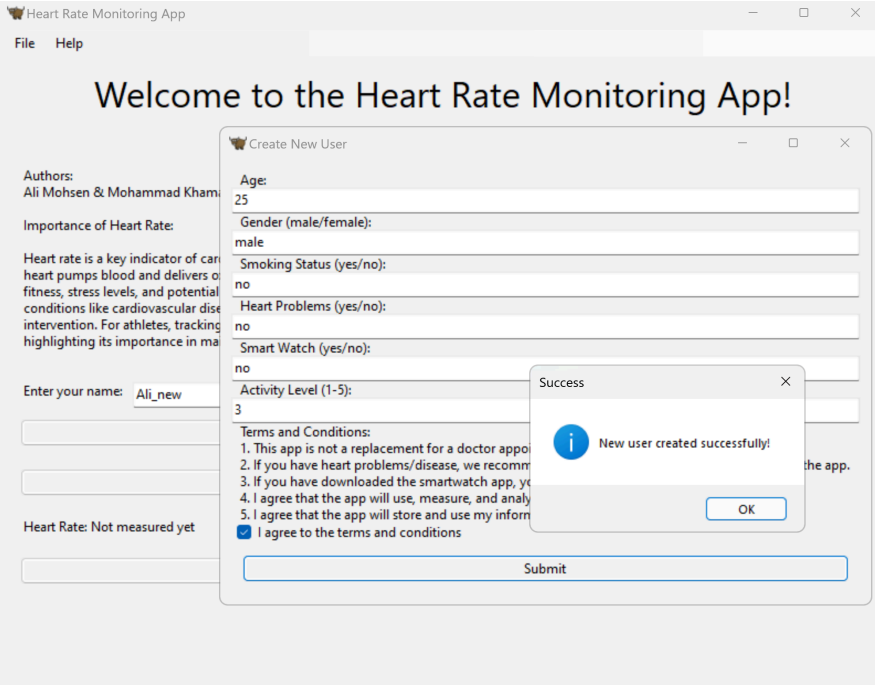


Figure 52. Feedback message for Creating new users.

We can see that the user in created and added to the DataBase

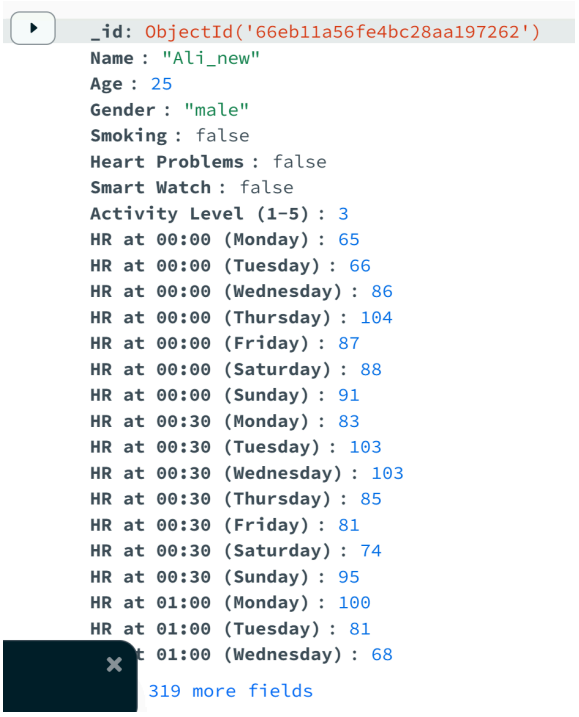


Figure 53. The new user information is added to the DataBase.

Now let's explore the invalid inputs while creating a new user.

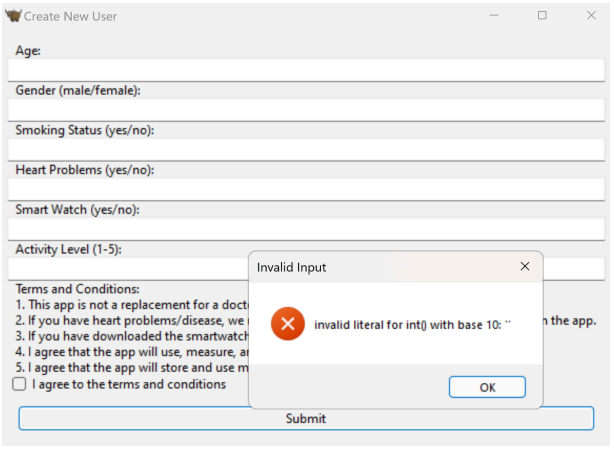


Figure 54. Check invalid values when creating a new user, and give a message accordingly.

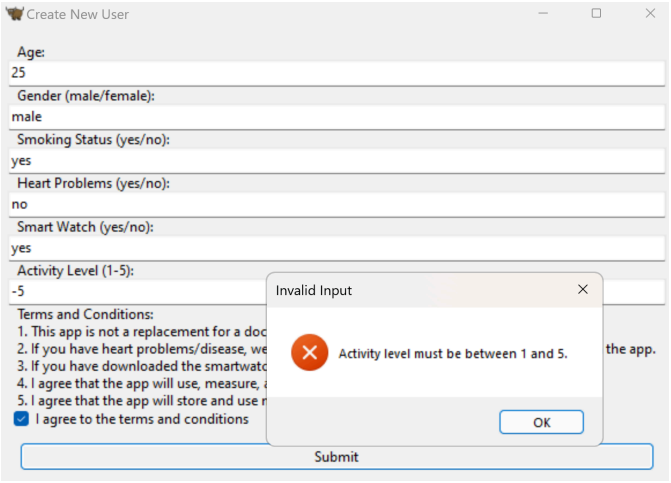


Figure 55. Check the value range of activity levels.

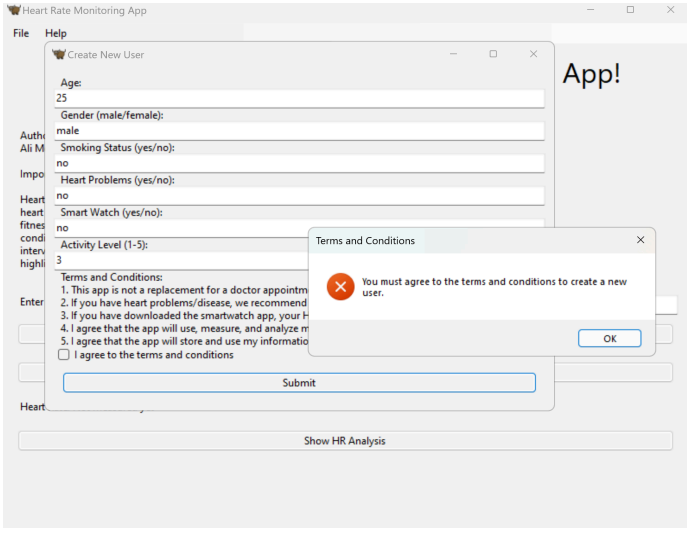


Figure 56. Check agreement on the terms and conditions before creating a new user.

Similarly, the function update_heart_rate(user_id, heart_rate, time_to_update) is tested to verify that normal heart rate updates are processed correctly. For example, with a valid user_id, a heart rate of 75, and a time of "12:00 PM," the database should update with the new heart rate at the specified time. Another test case examines the function's response to abnormal heart rate detection, where an existing heart rate of 60 followed by a new heart rate of 150 should trigger an abnormal heart rate detection message.

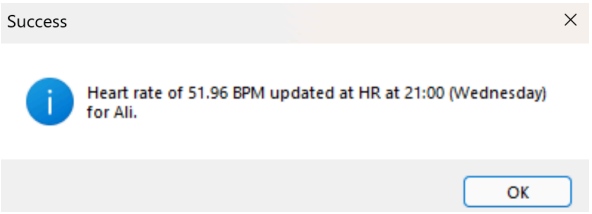


Figure 57. Present message for successfully updating heart rate in database.

```
HR at 20:30 (Saturday) : 102
HR at 20:30 (Sunday) : 106
HR at 21:00 (Monday) : 93
HR at 21:00 (Tuesday) : 98
HR at 21:00 (Wednesday) : 51.95561475128144
```

Figure 58. Present updating value for heart rate in database.

The function measure_heart_rate() is tested to confirm successful heart rate measurement under normal conditions, where the expected output is a valid BPM value. Additionally, the function's behavior is checked when the camera is unavailable, where the expected output is an error message stating, "Error: Could not open camera.

We checked also if we cant open the camera (the Zoom camera was on)

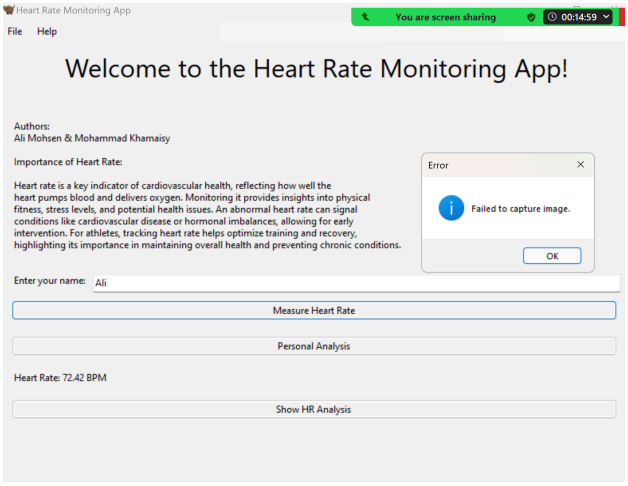


Figure 59. Check camera availability when it is already working in another place.

We also checked the created plots according to the algorithms we used and made sure that all of the plots available and readable (see Results section).

We tried to measure Heart rate without entering a user name

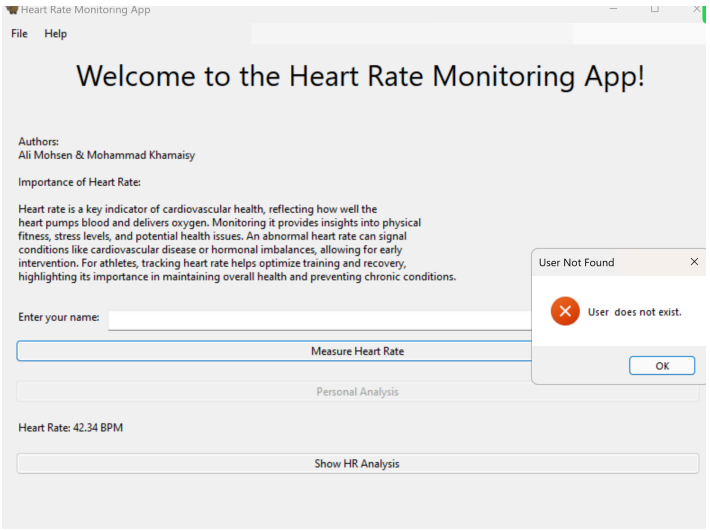


Figure 60. Check if a non-registered user can use the camera to detect heart rate .

We checked if the exit button works well

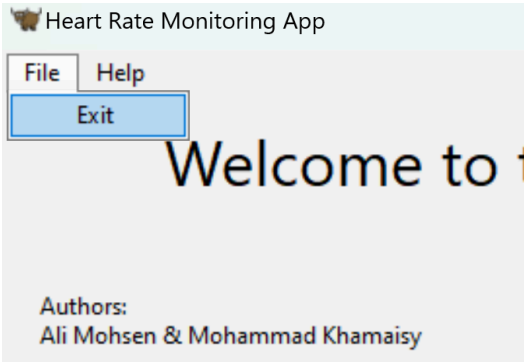


Figure 61. Check exit option button availability.

7.2 Integration Testing

The aim is to verify that different components of the application interact as expected. For example, one scenario involves creating a new user with valid inputs, measuring heart rate via camera, updating the heart rate in the database, and analyzing the heart rate forecast. The expected outcome is the successful creation of the user, along with accurate measurement, updating, and analysis of the heart rate. Another scenario involves creating a new user and inputting an abnormal heart rate to ensure that the abnormal heart rate detection logic triggers correctly, resulting in appropriate user prompts (see Figures 50-61).

7.3 System Testing

To ensure that the entire application functions as a cohesive unit, one scenario tests the complete user flow for heart rate monitoring. The steps include starting the application, creating a new user, measuring and updating the heart rate, displaying heart rate information, running anomaly detection, and analyzing the heart rate forecast. The expected outcome is a smooth flow from user creation to heart rate monitoring and anomaly detection, with no interruptions. Another scenario tests user input validation by inputting invalid data for age, heart rate, etc., ensuring the application handles errors gracefully, catches them, and prompts the user to re-enter valid data.

We have a video that demonstrates all the functionalities of the app to ensure that the entire system operates effectively. This video provides a comprehensive overview of how the app performs its intended functions.

7.4 Performance Testing

The objective is to outline how the application's responsiveness would be assessed under load. For example, in a hypothetical database load test where heart rates for 1,000 users would be created and updated, the database would be expected to handle the data efficiently without significant performance degradation. In another proposed test, the speed of heart rate anomaly detection on large datasets would be measured, with the expected outcome being the completion of anomaly detection within a reasonable time frame, ideally within a few seconds.

7.5 User Acceptance Testing (UAT)

The goal is to validate the application from an end-user perspective. In a usability test, a group of users tests the application, and the expected outcome is that users find the application easy to use and intuitive. Feedback is gathered and evaluated for further improvements. Accessibility testing ensures the application is accessible to users with disabilities, such as visual impairments, with expected outcomes including features like screen reader compatibility and high-contrast mode.

We presented the app to several colleagues, and they were satisfied with both the interface and the overall app interaction. Their feedback confirmed that the app is user-friendly and intuitive.

7.6 Evaluation Metrics

Evaluation metrics include functionality, measured by the percentage of test cases that pass successfully; performance, assessed by the average time taken to complete tasks like anomaly detection and heart rate updates; usability, reflected in the user satisfaction score from UAT, indicating the application′s intuitiveness and user-friendliness; and security, gauged by the number of vulnerabilities found and patched, ensuring the application remains secure against potential threats.

8. Conclusion

The development of our heart rate monitoring application represents a significant advancement in personalized healthcare technology. By leveraging the capabilities of wearable devices and cutting-edge machine learning algorithms, we have created a tool that empowers individuals to take proactive control of their cardiovascular health. Throughout this journey, we have addressed the critical need for continuous, personalized monitoring that goes beyond the limitations of traditional methods and consumer-grade devices.

Our application successfully integrates real-time data collection with sophisticated analysis, providing users with timely insights and alerts about their heart health. By employing advanced algorithms such as SMA, Isolation Forests, and One-Class SVMs, the app can detect anomalies and adapt to individual heart rate patterns, ensuring both accuracy and personalization. This approach not only enhances the user's ability to monitor their health but also fosters a deeper understanding of their unique cardiovascular profile.

The testing and validation phases have demonstrated the app's effectiveness in various real-world scenarios, confirming its potential as a valuable tool for users across different age groups and health statuses. By offering features that cater to a wide range of needs–from daily activity tracking to early detection of potential health issues–the app promotes a holistic approach to wellness.

Looking forward, the success of this project opens the door to numerous possibilities for further innovation. Future iterations could explore integrating additional health metrics, expanding compatibility with other wearable devices, and enhancing user engagement through gamification and community features. Moreover, collaboration with healthcare providers could facilitate more comprehensive health management, bridging the gap between personal monitoring and professional care.

In conclusion, this heart rate monitoring application not only addresses a pressing global health challenge but also exemplifies the transformative potential of technology in healthcare. By providing users with the tools to monitor, understand, and manage their cardiovascular health, we hope to contribute to a future where proactive health management is accessible to all, ultimately reducing the burden of cardiovascular diseases worldwide.

9. User documentation

Installation

Step 1: Install Anaconda and Set Up Spyder

- Download Anaconda: Visit the official Anaconda website at https://www.anaconda.com/products/distribution and download the version suitable for your operating system (Windows, macOS, or Linux).

- Install Anaconda: Run the downloaded installer and follow the on-screen instructions. The default settings are recommended for most users, but you can customize the installation if needed.
- Open Spyder: Once Anaconda is installed, open the Anaconda Navigator and launch Spyder from the available applications.

Step 2: Install MongoDB

- Download MongoDB: Visit the official MongoDB website at https://www.mongodb.com/try/download/community and download the Community Server version suitable for your operating system.
- Install MongoDB: Run the downloaded installer and follow the on-screen instructions. During installation, make sure to check the box that says "Install MongoDB as a Service" for easier setup.
- Start the MongoDB Service: After installation, MongoDB will start automatically as a service. You can verify this by opening a command prompt (Windows) or terminal (macOS/Linux) and typing:cmongod --version
- This command will confirm that MongoDB is running properly.

install pymongo Python Library: Open Spyder and install the pymongo library by running the following command in the Spyder console or in Anaconda Prompt: pip install pymongo

Setting Up Your Project in Spyder

After installing Anaconda And MongoDB and launching Spyder, you can set up your project as follows:

- Opening Your Script: To open your heart rate monitoring script in Spyder, go to File -> Open from the top menu. Navigate to the directory where your script (main.py) is located, select it, and click Open. This will load your script into the Spyder editor.

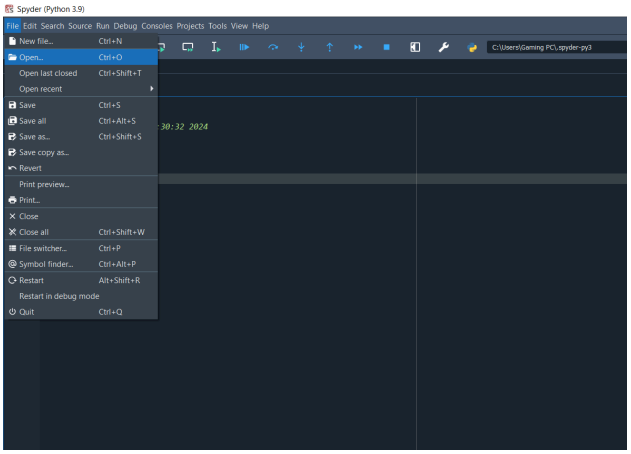


Figure 62. Spyder platform main page and open option to load the project to run .

Installing Required Libraries: Open the Anaconda Prompt or terminal and install the necessary libraries by running:
conda install numpy

- This command will ensure that all required dependencies are available for your application.

Installing Toga: To build a graphical user interface (GUI) for your heart rate monitoring application, you will use Toga, a cross-platform GUI toolkit from the BeeWare suite. Toga allows you to create applications with native interfaces across multiple platforms like Windows, macOS, and Linux.

Install Toga: Open the Anaconda Prompt or any terminal and run:pip install toga

- Ensure you are using Python 3.9.13, as Toga is compatible with this version.

Creating the GUI: Once Toga is installed, you can create the GUI for your heart rate monitoring application.

Running Toga Application: To run Toga-based heart rate monitoring application:

- Open Your Script: In Spyder, open the main.py file containing your Toga application code.
- Run the Application: Click the Run button or press F5. Spyder will execute the script, and the Toga application window will open, displaying your GUI.

Connecting to DB : To connect to a MongoDB database in your Python application using the pymongo library, you need to establish a connection using the MongoClient. the correct way to connect to a MongoDB instance running on localhost (your local machine) on the default port 27017.

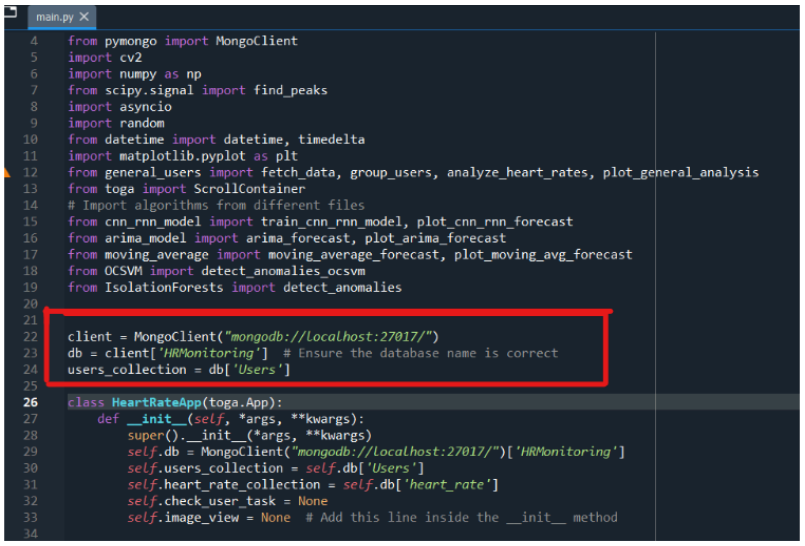


Figure 63. Connecting the app to the database using localhost 27017 for loading users information.

Using the Project :

Using the Heart Rate Monitoring Application

1. Launch Spyder: Open the Anaconda Navigator and launch Spyder from the available applications. This will open the Spyder IDE, where you can run and modify the application code.
2. Open the Application Code: Load the Python file containing your application code (e.g., heart_rate_app.py) into Spyder. Make sure the file is saved in a location that is easily accessible.

Ensure MongoDB is Running: Verify that the MongoDB service is running. You can check this by opening a command prompt (Windows) or terminal (macOS/Linux) and typing:cvmongod

3. If MongoDB is not running, start it by executing the mongod command in the terminal.
4. Run the Application: In Spyder, click the "Run" button or press F5 to execute the application code. This will launch the Heart Rate Monitoring application.
5. Interact with the Application:
    - Welcome Screen: You'll see a welcome screen with information about the application's purpose and authors.
    - Enter User Information: Enter your name in the text input field. The application may fetch and display existing user data from the MongoDB database or create a new user entry.
    - Heart Rate Analysis: Use the application's features to analyze and monitor heart rate data. The application may support different functionalities such as real-time heart rate monitoring, trend analysis, and anomaly detection using various algorithms.
6. View and Analyze Data:
    - Data Analysis: The application uses algorithms like SMA, ARIMA, Moving Average, One-Class SVM, and Isolation Forests to forecast trends or detect anomalies in heart rate data. Use the respective options in the application to visualize these analyses.
    - Graphs and Visualizations: The app will display graphs and visualizations for heart rate trends, forecasts, and detected anomalies. You can interact with these visualizations to gain insights into your cardiovascular health.
7. Save and Export Data: You can save or export the analysis results and graphs for further review. Check the application interface for options to save data or screenshots.
8. Close the Application: When you are done, you can close the application window. MongoDB will continue running in the background, storing any data you entered or modified.

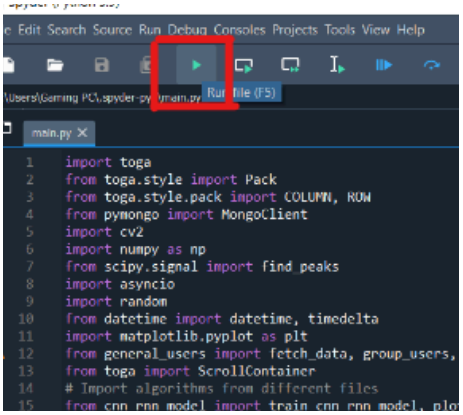


Figure 64. Run button to run the app after loading the environment and app requirements.

DB structure: MongoDB database representing a user's data for heart rate monitoring. Here's an explanation of the structure:

Database Structure Explanation

- **_id**: This field is an automatically generated unique identifier (ObjectId) for the document. In this case, the ID is `'667d3e90f6e5c096a29e736b'`.
- **Name**: The user's name, in this example, is "Mohammad".
- **Age**: The user's age, which is `24`.
- **Smoking**: A Boolean field indicating whether the user smokes or not. Here, it is false, meaning the user does not smoke.
- **Heart Problems**: A Boolean field that indicates whether the user has any known heart problems. It is set to false, meaning the user has no heart problems.
- **Smart Watch**: A Boolean field indicating whether the user has a smartwatch. It is set to true, which means the user has a smartwatch.
- **Activity Level (1-5)**: This field shows the user's activity level on a scale from `1` (least active) to `5` (most active). The user's activity level is `4`, which indicates a relatively high level of physical activity.
- **HR at [time]**: These fields represent the heart rate (HR) readings taken at half-hour intervals throughout the day. Each field specifies the heart rate measurement at a particular time. For example:
    - HR at `00:00`: The heart rate at midnight is `131`.
    - HR at `00:30`: The heart rate at `00:30` is `175`.
    - And so on, for each half-hour interval.

The document provides a detailed record of the user's heart rate measurements over time, which can be used for monitoring and analysis.

Purpose of the Database Structure

This structure allows the storage of comprehensive health-related information for each user, including personal details (like age and smoking status), activity levels, and a continuous stream of heart rate data. Such data is useful for detecting patterns, monitoring health status, identifying anomalies, and providing personalized health insights through the app.

10. References

[1] Jensen-Urstad, K., Storck, N., Bouvier, F., Ericson, M., Lindbland, L., & Jensen-Urstad, M. (1997). Heart rate variability in healthy subjects is related to age and gender. *Acta Physiologica Scandinavica, 160*(3), 235-241. [Google Scholar] [CrossRef]

[2] Epstein, L. H., Paluch, R. A., Kalakanis, L. E., Goldfield, G. S., & Roemmich, J. N. (2001). How much activity do youth get? A quantitative review of heart-rate measured activity. *Pediatrics, 108*(4), e44.

[3] Quer, G., Gouda, P., Galarnyk, M., Topol, E. J., & Steinhubl, S. R. (2020). Real-world heart rate norms in the Health eHeart study. *NPJ Digital Medicine, 3*, Article 7.

[4] Achten, J., & Jeukendrup, A. E. (2003). Heart rate monitoring. *Sports Medicine, 33*(7), 517-538.

[5] Behar, J. A., et al. (2021). Physiological measurements from smartwatches are associated with atrial fibrillation. *Heart Rhythm, 18*(12), 2125-2127.

[6] Monkaresi, H., et al. (2014). A machine learning approach to improve contactless heart rate monitoring using a webcam. *IEEE Journal of Biomedical and Health Informatics, 18*(4), 1153-1160.

[7] MongoDB. (n.d.). What Is Machine Learning? Retrieved from https://www.mongodb.com/resources/basics/machine-learning

[8] MongoDB. (n.d.). Machine Learning In Healthcare. Retrieved from https://www.mongodb.com/resources/basics/artificial-intelligence/machine-learning-healthcare

[9] Png, N. (n.d.). Training Machine Learning Models with MongoDB. Retrieved from https://www.mongodb.com/blog/post/training-machine-learning-models-with-mongodb

[10] Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation Forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining* (pp. 413-422). IEEE.

[11] Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., & Williamson, R. C. (2001). Estimating the Support of a High-Dimensional Distribution. *Neural Computation, 13*(7), 1443-1471.

[12] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE, 86*(11), 2278-2324.

[13] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation, 9*(8), 1735-1780.

[14] Box, G. E. P., & Jenkins, G. M. (1970). Time Series Analysis: Forecasting and Control. *Holden-Day*.

[15] Wyss, C. (2017). Accuracy of heart rate apps varies. *European Journal of Preventive Cardiology, 24*(4), 399-400.

[16] World Health Organization. (n.d.). Cardiovascular diseases (CVDs). Retrieved from https://www.who.int/health-topics/cardiovascular-diseases#tab=tab_1

[17] Seshadri, D. R., et al. (2019). Wearable sensors for detection of atrial fibrillation.

[18] Wallen, M. P., et al. (2016). Accuracy of heart rate watches: Implications for weight management.

[19] Shaffer, F., & Ginsberg, J. P. (2017). An overview of heart rate variability metrics and norms.

[20] Gaziano, T., et al. (2006). Cardiovascular disease. In: *Disease control priorities in developing countries* (2nd ed.).

[21] Avram, R., Tison, G. H., Aschbacher, K., Kuhar, P., Vittinghoff, E., Butzner, M., Runge, R., Wu, N., Pletcher, M. J., Marcus, G. M., & Olgin, J. E. (2019). Real-world heart rate norms in the Health eHeart study. NPJ Digital Medicine, 2(1), 58. Data Collection: Heart rate data was collected from 66,788 participants in the Health eHeart cohort between April 2014 and April 2018, resulting in 3,144,332 HR-PPG measurements