**Final Report: AI-Powered Health Monitoring System**

**Author:** Perinai Kamundu  
**Course:** AI for Software Engineering  
**Date:** 06/06/2025  
**Project Repository:** https://github.com/perinai/tracing-ai.git

**1. Executive Summary**

This report details the development and implementation of an AI-Powered Health Monitoring System, a key project for the AI for Software Engineering course. The primary objective was to build a system capable of analyzing real-time health metrics to detect anomalies. The project successfully integrates a machine learning model into a web application, which is then containerized for portability and deployment. The final deliverable is a functional, user-friendly application packaged within a Docker container, showcasing a complete, end-to-end development lifecycle using modern software engineering practices.

**2. Introduction**

**2.1. Project Overview**

The proliferation of wearable health devices has generated vast amounts of personal health data. This project harnesses that potential by creating a system that ingests user health data—specifically heart rate, blood oxygen levels, and activity status—and uses an AI model to provide immediate feedback on potential health anomalies.

**2.2. Objectives**

The core objectives of this project were to:

1. Develop a machine learning model capable of classifying health data as "Normal" or "Anomaly".
2. Build a user-friendly web interface for data input and results visualization.
3. Integrate the AI model into the web application backend.
4. Apply software engineering best practices by containerizing the entire application with Docker for consistent deployment.
5. Utilize a modern, cloud-based development workflow using GitHub and GitHub Codespaces.

**3. Methodology**

**3.1. Data Generation and Preprocessing**

Due to the unavailability of real-time patient data, a simulated dataset was generated using the Python library NumPy. The dataset included 1,000 records with the following features:

* **Heart Rate:** (integer)
* **Blood Oxygen:** (float)
* **Activity Level:** (categorical: 'resting', 'walking', 'running')

Anomalies, such as tachycardia (heart rate > 100 bpm) and hypoxemia (blood oxygen < 95%), were intentionally injected into 10% of the dataset to ensure the model had examples of abnormal conditions to learn from. For model training, the categorical 'activity\_level' feature was converted into numerical format using one-hot encoding via the Pandas library.

**3.2. Model Development and Training**

The task was framed as a binary classification problem. A **Random Forest Classifier** from the scikit-learn library was chosen for this task due to its high performance, robustness, and ability to handle both numerical and categorical data effectively.

The dataset was split into training (80%) and testing (20%) sets. The model was trained on the training data and evaluated on the unseen testing data. The final trained model, along with its feature list, was saved to disk using joblib as .pkl files for later use in the application.

**3.3. Application Architecture (Tech Stack)**

The system was designed as a classic client-server web application:

* **Backend:** A lightweight web server was built using **Flask**, a Python web framework. It handles incoming web requests, processes user input, calls the AI model for predictions, and returns the results.
* **Frontend:** The user interface was created with simple **HTML** and **CSS**, providing a form for data entry and a clear results page that changes color based on the prediction.
* **Production Web Server:** **Gunicorn** was chosen as the WSGI server to run the Flask application inside the Docker container, as it is more robust and suitable for production than Flask's built-in development server.

**4. Implementation and Deployment**

**4.1. Development Environment**

The entire project was developed using **GitHub Codespaces**, a cloud-based development environment. This approach eliminated the need for a local Python or Docker setup, demonstrating a modern workflow that ensures a consistent and reproducible development environment for all team members. All code was version-controlled using **Git** and hosted on **GitHub**.

**4.2. Containerization with Docker**

A key software engineering goal was to make the application portable and easy to deploy. This was achieved using **Docker**.

* A **Dockerfile** was written to provide a complete, step-by-step blueprint for building the application image. This blueprint starts with a lean Python base image, installs the necessary dependencies from a requirements.txt file, copies the application code, and defines the command to run the Gunicorn server.
* A clean **requirements.txt** file was created to explicitly list only the direct dependencies needed for the application to run, ensuring a small and secure final container.

This process packages the entire application into a single, self-contained unit that can be run on any machine with Docker installed, solving the "it works on my machine" problem.

**5. Results**

**5.1. Model Performance**

The Random Forest model performed exceptionally well on the test dataset, achieving high scores in accuracy, precision, and recall, indicating its effectiveness in distinguishing between normal and anomalous health readings.

**5.2. Application Functionality**

The final application successfully meets all objectives. Users can input their health data and receive an immediate, color-coded response.

**Normal Use Case:**  
When a user inputs normal values (e.g., Heart Rate: 75, Blood Oxygen: 98), the application correctly classifies the data as "Normal" and displays a green-themed results page.

**Anomaly Use Case:**  
When a user inputs values indicative of an anomaly (e.g., Heart Rate: 130, Blood Oxygen: 92), the application correctly classifies the data as "Anomaly Detected" and displays a red-themed results page with a warning.

[Insert Screenshot of the red "Anomaly Detected" results page here]

**6. Challenges and Future Work**

**6.1. Challenges Encountered**

The most significant software engineering challenge was **dependency management for Docker**. An initial attempt using pip freeze created a requirements.txt file with hundreds of system-specific packages, causing the Docker build to fail. This was resolved by manually creating a clean requirements.txt file with only the essential application dependencies, a critical lesson in creating lean production environments.

**6.2. Future Work**

This project serves as an excellent foundation for further development. Potential future enhancements include:

* **Integrating with Real Wearable APIs** (e.g., Fitbit, Apple HealthKit) to pull data automatically.
* **Implementing a more advanced model**, such as an LSTM (Long Short-Term Memory) network, to analyze trends in time-series data.
* **Adding a user database** to store historical data and track user health over time.
* **Deploying the Docker container** to a cloud service like Azure App Service or AWS Elastic Beanstalk for a permanent, public URL.

**7. Conclusion**

This project successfully demonstrates the fusion of AI and software engineering principles. A functional machine learning model was developed, integrated into a web application, and professionally packaged using Docker. The use of a modern, cloud-based workflow with GitHub and Codespaces highlights contemporary best practices. The final result is a robust, deployable prototype for an AI-powered health monitoring system, fulfilling all the requirements of the course project.