



Faculty of Computer Science and Information Technology

***Automating Peak Flow Data Extraction for Asthma
Monitoring and Management.***

Final Year Project 1

Tonny Vincent
(81402)

Bachelor of Computer Science with Honours (Software Engineering)

2025

Acknowledgement

I would like to express my deepest appreciation to my supervisor, Ts. Dr Lim Phei Chin, for his invaluable guidance, encouragement and insight throughout this project. Their expertise and support have been instrumental in overcoming the challenges and achieving the objectives of this work.

I am extremely grateful to my family for their unwavering support, patience, and understanding, which kept me motivated during the demanding phase of this project. Your faith in me has been my strength.

To my friends, thank you for your encouragement, constructive feedback, and supporting me throughout this journey. Your support has made this experience truly unforgettable.

Abstract

Asthma is a common chronic condition that requires consistent monitoring to effectively manage symptoms and prevent exacerbations. Traditional peak flow meters, although affordable and accessible, rely on manual recording, making them prone to error and limiting their reliability. Digital alternatives, while accurate, are expensive and complex, posing accessibility challenges for many users.

This project bridges the gap by developing a system that automates the extraction of peak expiratory flow (PEF) readings from manual peak flow meters using image processing techniques. The system uses OpenCV for preprocessing and Tesseract.js for Optical Character Recognition (OCR), ensuring accurate and reliable data capture. A user-friendly Web Application allows patients to record, track and visualize their respiratory data while maintaining affordability and ease of use.

Through extensive testing across multiple conditions, the system demonstrated its robustness, providing an accessible and effective solution for asthma monitoring. By integrating advanced image processing with manual devices, the project improves asthma care, ensuring equity and accuracy for a diverse user group.

Mengautomasikan Pengekstrakan Data Aliran Puncak untuk Pemantauan dan Pengurusan Asma

Abstrak

Asma adalah penyakit kronik yang memerlukan pemantauan berterusan untuk mengurus simptom dan mencegah serangan. Meter aliran puncak (peak flow meter) tradisional, walaupun murah dan mudah didapati, memerlukan rakaman manual, menjadikannya terdedah kepada ralat dan mengehadkan kebolehpercayaannya. Alternatif digital pula, walaupun tepat, adalah mahal dan kompleks, menimbulkan cabaran kebolehcapaian bagi ramai pengguna.

Projek ini merapatkan jurang dengan membangunkan sistem yang mengautomasikan pengekstrakan bacaan aliran ekspirasi puncak (PEF) daripada meter aliran puncak manual menggunakan teknik pemprosesan imej. Sistem ini menggunakan OpenCV untuk prapemprosesan dan Tesseract.js untuk Pengesanan Aksara Optik (OCR), memastikan perolehan data yang tepat dan boleh dipercayai. Aplikasi Web yang mudah digunakan membolehkan pesakit merekod, menjejak, dan melihat data pernafasan mereka sambil mengekalkan kemampuan dan kemudahan penggunaan.

Melalui ujian yang meluas merentasi pelbagai keadaan, sistem ini menunjukkan ketahanannya, menyediakan penyelesaian yang mudah diakses dan berkesan untuk pemantauan asma. Dengan mengintegrasikan pemprosesan imej lanjutan dengan peranti manual, projek ini meningkatkan penjagaan asma, memastikan kesaksamaan dan ketepatan untuk kumpulan pengguna yang pelbagai.

Table of Contents

Contents

Thesis Status Endorsement Form	ii
Acknowledgement.....	iii
Abstract.....	iv
1.0 CHAPTER 1 Introduction	1
1.1 Background.....	1
1.2 Problem Statement.....	2
1.3 Objective.....	3
1.4 Scope	3
1.5 Significant of Project.....	4
1.6 Expected Outcomes.....	4
1.7 Summary	5
2.0 CHAPTER 2 Literature Review	6
2.1 Introduction	6
2.2 Review of related works.....	6
2.3 Review of Peak Flow Meter.....	8
2.4 Discussion of related works and Peak Flow Meter devices.....	9
Comparison of related works	10
Comparison of Peak Flow Meter	11
2.5 Summary	12
3.0 CHAPTER 3 Requirement Analysis and design	13
3.1 Introduction	13
3.2 Requirement Elicitation	13
3.3 Architecture.....	15
3.4 Agile Software Development.....	18
3.4.1 Planning	18
3.4.2 Design.....	19
3.5 Summary	31
4.0 CHAPTER 4 Implementation.....	32
4.1 Introduction.....	32
4.2 System Prototype	32
4.2.1 Libraries Installation.....	33
4.2.2 System Hierarchy	34
4.3 Findings.....	68

4.4	Summary	68
5.0	CHAPTER 5 Testing	69
5.1	Introduction	69
5.2	Testing	69
5.2.1	Series of Experiments with Various Settings	69
	System Usability Scale (SUS) Feedback.....	81
5.3	Summary	89
6.0	CHAPTER 6 Conclusion and Future Work	90
6.1	Introduction	90
6.2	Limitation	91
6.2.1	Image Capture Sensitivity to Environmental Factors	91
6.2.2	Device Compatibility and OCR-Specific Challenges	91
6.2.3	API and Library Performance Limitations	92
6.3	Future Works.....	92
6.3.1	Enhanced Image Processing and Artificial Intelligence Integration 93	
6.3.2	Scalability, Security and Cross-Platform Development	93
6.4	Summary	96
7.0	Appendix	97
	Appendix A - Final Year Project (FYP) Gantt Chart	97
a.	FYP 1	97
b.	FYP 2	97
	Appendix B – Several picture of public user	98
8.0	References	101

List of Figures

Figure 1: Traditional Peak Flow Meter	2
Figure 2: Digital Peak Flow Meter.....	2
Figure 3: MyPEF Device and manual chart graph.....	3
Figure 4: System Architecture	15
Figure 5: Flowchart of the system (left side of the flowchart).....	16
Figure 6: Flowchart of the system (right side of the flowchart)	17
Figure 7: Use case diagram	19
Figure 8: Activity Diagram for a Automatic Peak Flow Data Extraction System.....	23
Figure 9: Sequence diagram.....	24
Figure 10: Home/Main/Login Page	25
Figure 11: Register Page	25
Figure 12: Face recognition message	26
Figure 13: Success message	
Figure 14: Failed message	
Figure 15: Recognition loader	26
Figure 16: Unauthorize face	
Figure 17: Authorize face.....	26
Figure 18: Morning patient page	27
Figure 19: Night patient page	27
Figure 20: Reading existence indicator.....	27
Figure 21: Reading taking.....	28
Figure 22: Doctor Home page.....	28
Figure 23: List of patients viewing page.....	29
Figure 24: Search functions	29
Figure 25: Graph view for specific patients	30
Figure 26: Class diagram	30
Figure 27: System hierarchy	34
Figure 28: Signup front-end	35
Figure 29: Pre-trained model face-api.js.....	35
Figure 30: Scan face function	36
Figure 31: Webcam access.....	36
Figure 32: Saving picture	37
Figure 33: Post-registration flow	37
Figure 34: Load labeled images function	38
Figure 35: Scanning face function.....	39
Figure 36: POST request	39
Figure 37: User role redirection	40
Figure 38: Animation feedback	40
Figure 39: Patient homepage header.....	41
Figure 40: Date and time display.....	42
Figure 41: Theme.....	42
Figure 42: Checkmark logic	43
Figure 43: Reading validation.....	43
Figure 44: Scan button	44
Figure 45: fetch request PEFr.....	44
Figure 46: Reading button updates.....	45
Figure 47: Session update	46
Figure 48: graph visualization	46
Figure 49: Elements and scripts	47

Figure 50: Doctor homepage header	47
Figure 51: Patient access section	48
Figure 52: Profile image placeholder	48
Figure 53: User input fields	48
Figure 54: Fetching existing profile data.....	49
Figure 55: Fetching and displaying profile picture	50
Figure 56: Uploading image	50
Figure 57: Update profile function.....	51
Figure 58: Image upload.....	52
Figure 59: Modal feedback and validation.....	52
Figure 60: Page layout and user interface.....	53
Figure 61: Camera initiation.....	53
Figure 62: Frame processing.....	54
Figure 63: Red arrow detection using OpenCV	55
Figure 64: Visual feedback and region of interest.....	55
Figure 65: ROI processing.....	56
Figure 66: Optical character recognition	57
Figure 67: Position sorting and interpolation.....	57
Figure 68: Reading confirmation.....	58
Figure 69: Storing reading	59
Figure 70: Dynamic patient list	60
Figure 71: Patient data table	60
Figure 72: Patient data fetching	61
Figure 73: Patient row redirection.....	61
Figure 74: Change items per page	61
Figure 75: Search functionality.....	62
Figure 76: Backend - Search Functionality	62
Figure 77: Manual search filtering	63
Figure 78: Graph display area	63
Figure 79: Line chart display	64
Figure 80: Dynamic Data Fetching.....	64
Figure 81: Actual PEF readings.....	65
Figure 82: Best known PEF value	65
Figure 83: Graph axes	66
Figure 84: PEF Summary Statistics	66
Figure 85: User tables	67
Figure 86: Peakflowmeasurements table.....	67
Figure 87: 3 rd Testing	70
Figure 88: 16 th Testing.....	70
Figure 89: 12 th Testing	71
Figure 90: 22 nd Testing	71
Figure 91: 15 th Testing (15° Tilt)	72
Figure 92: 15 th Testing (30° Tilt).....	72
Figure 93: 1st Testing (480p)	73
Figure 94: 11st Testing (720p).....	73
Figure 95: A public user (first-time user) conducting an experiment with the Automating Peak Flow Data Extraction system.....	75
Figure 96: The result of a public user extracting data	75
Figure 97: Question 1	81
Figure 98: Question 2	82
Figure 99: Question 3	82
Figure 100: Question 4	83

Figure 101: Question 5	83
Figure 102: Question 6	84
Figure 103: Question 7	84
Figure 104: Question 8	85
Figure 105: Question 9	85
Figure 106: Question 10	86
Figure 107: FYP 1 Gantt Chart	97
Figure 108: FYP 2 Info	97
Figure 109: FYP 2 Chart	97
Figure 110: A public user (first-time user) conducting an experiment with the Automating Peak Flow Data Extraction system.	98
Figure 111: The result of public user extracted data	98
Figure 112: A public user (first-time user) conducting an experiment with the Automating Peak Flow Data Extraction system.	99
Figure 113: The result of public user extracted data	99
Figure 114: A public user (first-time user) conducting an experiment with the Automating Peak Flow Data Extraction system.	100
Figure 115: Figure 113: The result of public user extracted data.....	100

List of Tables

Table 1: Summary of studies related to automated ruler detection and scale measurement	10
Table 2: Comparison of Peak Flow Meter Types and Their Reading Methods	11
Table 3: Use Case Specification for Capture Reading	20
Table 4: Use Case Specification for Search Patient Data.....	20
Table 5: Use Case Specification for View Patient Data.....	21
Table 6: Use Case Specification for Notify User	21
Table 7: Use Case Specification for Store Reading.....	22
Table 8: Use Case Specification for View Patient Graph	22
Table 9: Libraries details	33
Table 10: Result of tilt test	72
Table 11: Result of resolution test	73
Table 12: Detailed Accuracy Results from User-Conducted Tests (30 Users, 3 Tests Each)	77
Table 13: SUS Score with interpretation.....	87

1.0 CHAPTER 1 Introduction

1.1 Background

Asthma is among the most common chronic respiratory diseases, affecting millions across the world. It is characterized by variable and recurring symptoms, wheezing, breathlessness, chest tightness, and cough. The effective management of the disease is cardinal in controlling such symptoms, preventing exacerbations, and ultimately enhancing the quality of life of patients. PEF monitoring refers to the measurement of the maximum speed of expiration, reflecting the degree of airflow obstruction, and is therefore an essential part of asthma self-management. When consistently measured and tracked, PEF values can provide important insights into the patient's respiratory status and the effectiveness of treatment plans. Nevertheless, while recordkeeping and monitoring of PEF remain an essential part of modern asthma management, methods for recording and monitoring PEF have their drawbacks, which affect the accuracy and reliability of the latter (Global Initiative for Asthma, 2021).

Traditionally, PEF was assessed using the manual peak flow meter—a small, inexpensive, and simple instrument that may be used at home. A patient exhaled with force through this instrument, which then gave an approximate reading reflecting his or her present respiratory status. While easy and simple, this manual mode depends on patients' writing down their readings, thus leaving it open to human error through misreading of the meter or recording an incorrect value. Such errors can lead to erratic data that may make an accurate judgment of whether the asthma is under control challenging for a healthcare professional (National Heart, Lung, and Blood Institute, 2020). Since asthma symptoms wax and wane in their severeness, PEF readings must be timely and as exact as possible for treatment plans to be adjusted, with the hope of preventing an asthma attack (Rea & McAuley, 2010).

To make PEF monitoring more accurate and handier, digital peak flow meters have been devised. Advanced meters automate the reading and recording, often linking to smartphones or computer applications where data is stored, analysed, and sometimes shared with healthcare providers. Whereas digital peak flow meters have indeed some advantages, they are invariably more expensive than the manual devices, thus limiting their access for many patients, especially among the low-income group of categories and resource-poor settings (Spalluto & Riordan, 2020). For this reason, a gap would likely remain in asthma management tools: whereas manual tools are highly accessible but prone to inaccuracies, the digital ones are more reliable yet costly and complex.

This work, therefore, tries to bridge the gap through the construction of a system that incorporates image processing for the correct extraction of PEF readings in conventional manual peak flow meters. Image processing is a subset of computer vision; it entails the processing and analysis of visual information either from images or videos to enable meaningful information to be extracted. This would introduce the reliability and accuracy of digital systems into the affordability of manual devices by taking an image of each reading from the peak flow meter and using image processing algorithms for detection and recording of the indicated PEF value. The result is a novel, accessible solution that could significantly improve asthma management through providing patients with an affordable method of accurate and consistent PEF monitoring (Chan & Chen, 2021).



Figure 1: Traditional Peak Flow Meter



Figure 2: Digital Peak Flow Meter

1.2 Problem Statement

Asthma management relies heavily on consistent monitoring of peak expiratory flow (PEF), traditionally measured using hand-held peak flow meters. Patients manually record readings, a method prone to human errors, such as misreading or forgetting to log values, resulting in incomplete or inaccurate data that hinders effective treatment. Moreover, this manual process can be inconvenient and discouraging, especially for elderly patients or those with limited health literacy.

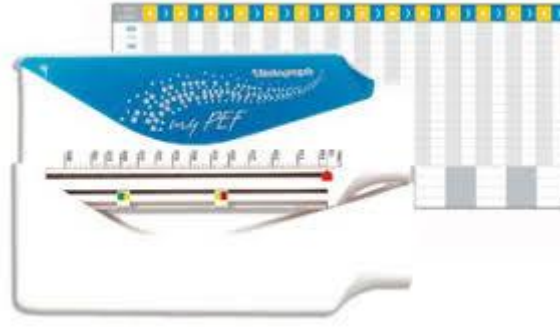


Figure 3: MyPEF Device and manual chart graph

While digital peak flow meters and smartphone apps have emerged as modern solutions to address these challenges by automating measurement and data storage, they are often prohibitively expensive and technologically complex. This makes them inaccessible to low-income populations, the elderly, and individuals in regions with limited digital literacy or access to advanced healthcare tools. These limitations perpetuate a gap in equitable asthma care, affecting patients' ability to monitor their condition accurately and consistently.

An ideal solution must bridge this gap by integrating image processing technology with existing manual peak flow meters. Such a system would automate the capture and digitization of PEF readings through affordable and familiar tools like smartphones and eliminating the need for additional costly equipment. This approach should be robust against environmental variations, such as lighting, camera angles, and different operating system (iOS and Android) ensuring reliable and accurate readings. By maintaining simplicity, affordability, and user-friendliness, this solution can empower patients to monitor their asthma more effectively, improve data accuracy, and ultimately enhance asthma care accessibility and outcomes for all, regardless of socioeconomic status or technical proficiency.

1.3 Objective

- To study current trends of data extraction from image.
- To implement an automatic Peak Flow data reading from images.
- To evaluate the accuracy of Peak Flow readings through a series of experiments with various settings and system usability scale.

1.4 Scope

This project focuses on developing an automated solution for capturing and interpreting Peak Expiratory Flow (PEF) readings from manual peak flow meters. It will involve the compatibility testing of at least two models of manual PEF devices to ensure adaptability across different designs, considering variations in scale size, indicator type, and device color (Global Initiative for Asthma [GINA], 2021). The system will include image preprocessing techniques to enhance edge detection, contrast adjustment, and indicator

isolation, with adaptability to real-world conditions like lighting and camera angles (Chan & Chen, 2021).

To achieve precise readings akin to ruler-based accuracy, the project will refine the optical character recognition (OCR) process using OpenCV for image processing and Tesseract.js for number extraction. A specific formula will be implemented to measure OCR accuracy, accounting for pixel-based deviations between predicted and actual PEF values. Continuous testing under diverse settings will ensure robustness and reliability (Coyle & Wilson, 2019). This system aims to bridge the gap between affordability and precision, allowing manual devices to function as accessible alternatives to costly digital counterparts.

The project's boundary includes the development of a user-friendly interface for capturing PEF readings and visualizing trends over time. It will exclude features unrelated to peak flow meters, such as integration with external devices. Data security and privacy will be prioritized by adhering to standards for storing sensitive health data (World Health Organization [WHO], 2022). By maintaining focus on affordability, accuracy, and usability, the project aspires to enhance asthma monitoring and empower patients across various socio-economic backgrounds.

1.5 Significant of Project

This project is significant as it seeks to improve asthma management by providing an accessible and affordable solution for monitoring Peak Expiratory Flow (PEF) data. By automating the process of reading and recording PEF values from manual peak flow meters, it addresses the inaccuracies and limitations associated with traditional methods, which often lead to unreliable data and suboptimal asthma control (Global Initiative for Asthma [GINA], 2021; National Heart, Lung, and Blood Institute [NHLBI], 2020). The image processing system, leveraging OpenCV and Tesseract.js, offers patients a convenient way to track their respiratory health without requiring costly digital devices. This innovation not only enhances data accuracy but also supports healthcare providers by enabling a reliable history of PEF readings. Such accurate records can assist clinicians in assessing asthma patterns and optimizing treatment plans, ultimately reducing the risk of severe asthma exacerbations (Rea & McAuley, 2010; Coyle & Wilson, 2019). Additionally, by integrating with mobile or web applications, the system encourages seamless data sharing between patients and providers, empowering users in low-income or resource-limited settings to actively participate in their health management (Spalluto & Riordan, 2020).

1.6 Expected Outcomes

The primary expected outcome is a functional prototype implemented as a web application capable of accurately extracting and recording Peak Expiratory Flow (PEF) readings from manual peak flow meters using image processing techniques. The system will employ a robust combination of OpenCV for image preprocessing and Tesseract.js for Optical Character Recognition (OCR) to ensure high precision in identifying and interpreting numerical readings. It will provide a user-friendly interface guiding patients to capture high-quality images and receive immediate feedback on their respiratory status. The

application will also allow users to securely store and track their PEF readings over time, displaying trends through visual charts to support asthma management.

Additionally, the system will incorporate real-time accuracy validation by comparing extracted readings against manually annotated ground truth values. This ensures that deviations are minimized, offering a ruler-like precision in reading the meter scales. By simulating various lighting and camera conditions, the prototype will also be tested to perform consistently in real-world settings. Integration with a backend database will enable historical data tracking and provide seamless sharing options with healthcare providers, should users opt for professional monitoring. Overall, the application is expected to deliver an accessible, affordable, and reliable tool that enhances asthma care by bridging the gap between manual and digital PEF monitoring solutions.

1.7 Summary

The project aims to address limitations in traditional asthma management by enhancing Peak Expiratory Flow (PEF) monitoring using image processing. Current manual methods are prone to errors and digital devices, while accurate, are costly. This initiative combines affordability with accuracy by using OpenCV and Tesseract.js to automate PEF reading extraction from manual meters, improving data precision and accessibility. The system will feature a user-friendly interface for capturing and analysing PEF readings, enabling secure tracking and sharing with healthcare providers. Expected outcomes include a functional web-based application that bridges the gap between manual and digital solutions, empowering better asthma care. Chapter 1, which introduces the project by covering its objectives, problem statement, and scope, is submitted on 21 November 2024. Chapter 2, which includes a literature review, comparison of existing methods, and analysis of peak flow meters, is completed by 13 December 2024. Chapter 3, which details the requirement analysis and system design, including requirement elicitation, system architecture, and diagrams, is submitted on 5 January 2025. The subsequent weeks will be dedicated to amendment, evaluation, and finalization of the report, ensuring a complete and concise report.

2.0 CHAPTER 2 Literature Review

2.1 Introduction

This chapter provides a detailed review of existing research, methodologies and applications related to the project domain. It examines a variety of related works, including journal articles, websites and software applications, to establish a comprehensive understanding of the current state of the art. This chapter identifies key trends, innovations and challenges in the field, offering insights into how existing solutions address similar problems and where they fail. Structured comparisons, presented through discussions and tables, highlight similarities and differences between the reviewed works, enabling a clear understanding of their contributions and limitations. This analysis not only serves to place the project in the knowledge and practical training but also helps to identify gaps and opportunities for improvement. The chapter concludes with a summary, synthesizing the findings and linking them to the project objectives, thereby reinforcing the rationale for the proposed approach.

2.2 Review of related works

The study by Wen et al. (2024) introduces an automated method to detect rulers in shoe print images to accurately estimate scale, addressing the inefficiency and error of traditional manual measurements, especially in forensic applications. It uses advanced image processing and machine learning techniques to improve accuracy and efficiency, focusing on reducing computational complexity and improving accuracy.

This method uses a Segment Anything Model (SAM) deep learning model to segment ruler regions, and an iterative algorithm with Feature from Accelerated Segment Test (FAST) key point detection to accurately identify and adjust ruler graduations, even when there are multiple lines or obstacles. The scale is determined by analysing the pattern of oscillations in pixel intensity along the traced ruler line, ensuring accurate measurements. The method was refined using a diverse dataset of ruler images, including forensic L-shaped rulers, to optimize segmentation accuracy.

With an average absolute percentage error of only 3%, this method significantly improves accuracy compared to traditional manual measurement, improving shoe print matching performance from 0.735 to 0.929. Despite these advances, these methods face limitations in handling perspective inconsistencies and low-contrast scenarios, which can distort the detected scale. This approach offers a significant improvement to the forensic analysis workflow, and future research can focus on improving the adaptability of the method to various environmental conditions and ruler designs, as well as integrating perspective correction.

This study done by Dou, et al. (2022) address the challenge of water level monitoring by automating water level recognition in instruments such as rulers through digital image processing and machine learning. Conventional methods, such as manual measurements or using expensive sensors, are often time-consuming, poorly scalable, or prone to environmental disturbances.

This study focuses on the use of image processing and Convolutional Neural Networks (CNNs) to create a cost-effective and accurate alternative for water level identification. The study uses gray scale processing methods, edge detection and tilt correction such as

Hough transform to detect ruler regions in captured images, isolating critical regions containing water levels and numerical markers for further analysis. A CNN is used to identify and interpret the numerical values on the ruler by mapping the number of detected scale lines to their corresponding values, enabling accurate water level estimation. A pipeline for simultaneous batch processing of multiple water level images is developed, beneficial for large-scale water monitoring systems. Grayscale conversion and duplication are used to segment the ruler from the background, while morphological operations improve scale line detection, and the Hough transform aligns and straightens the ruler if it is tilted. A pre-trained CNN model is fine-tuned to recognize the numerical digits on the ruler, and the extracted numerical values are mapped to the detected scale lines for accurate readings. The system calculates the pixel-to-value ratio based on the number of pixels between the scale line and the associated numerical value, ensuring accurate real-world measurements.

However, limitations include the struggle with submerged or obscured rulers due to sediment or vegetation and camera angle variations that impact accuracy, highlighting the need for improved perspective correction algorithms.

Achieving a recognition accuracy of 94.6%, which shows an improvement of 24% compared to traditional template matching algorithms, this method shows robustness in handling various light conditions and noisy environments. The integration of image processing techniques with CNNs provides a robust, efficient and cost-effective method for automatic water level identification, with findings underscoring its potential for large-scale use in environmental monitoring, offering a transformative tool for water resource management.

The study by Matuzevičius (2023) explores the use of synthetic image datasets to train deep learning models for ruler detection, addressing challenges such as lack of data in real-world applications. Synthetic data offers controlled variability, enabling robust model training.

The authors developed a procedural method to generate diverse synthetic ruler images with variations in angle, illumination and background, mimicking real-world conditions. A convolutional neural network (CNN) was trained using this synthetic data set and compared to a model trained on real-world data. The results show comparable or superior performance due to the diversity of the synthetic data and the noise-free nature.

This study highlights synthetic image generation techniques, CNN architectures for ruler detection and evaluation on real datasets, achieving over 95% accuracy in controlled scenarios. Although synthetic data are robust, combining them with real-world data can further improve accuracy, showcasing synthetic datasets as an efficient alternative for scalable ruler tracking systems.

2.3 Review of Peak Flow Meter

The myPEF Peak Flow Meter, developed by Vitalograph, is one of many traditional types of peak flow meter which is a portable and user-friendly device designed to monitor peak expiratory flow (PEF), which indicates how much air can be expelled from the lungs. It is important for asthmatics to detect changes in their airflow, as early signs of airway narrowing can occur hours or days before symptoms. Its compact and lightweight design makes it highly portable, and features a clear 100mm scale, color-coded indicator and conforms to ISO standards for peak flow meters. One of its main strengths is its ease of use, which promotes compliance among patients, both adults and children. Additionally, it includes protective features such as a dust cover and a robust case (Vitalograph, n.d.). However, the main drawback is that it requires manual recording of results, which may be troublesome for users looking for automated data tracking.

MyPEF costs around RM85 to RM95 for the standard version, which makes it quite affordable for individual users compared to some more advanced digital devices (Acucare System (M) Sdn Bhd, n.d.). Its main function remains focused on improving asthma management, providing practical and easy tools to monitor lung function at home, helping patients better manage their condition.

According to Microlife USA, A digital peak flow meter (PFM) is a sophisticated device designed to measure lung function, especially for individuals with conditions such as asthma and chronic obstructive pulmonary disease (COPD). They measured the peak expiratory flow rate (PEFR), which indicates the maximum speed at which air can be expelled from the lungs. The device was developed to provide a more accurate, portable and user-friendly alternative to traditional mechanical PFM, offering enhanced capabilities for self-monitoring at home and in clinical settings.

Key strengths of digital PFM include its accuracy, ease of use and ability to track long-term respiratory health through digital storage, memory functions and data analysis. Many models feature large, easy-to-read displays, adjustable settings and the ability to store multiple measurements for tracking over time.

However, a price survey for digital PFMs can be more expensive than mechanical models, with prices typically ranging from \$40 to \$100 or more, depending on features such as Bluetooth connectivity or mobile app integration to share data (Peak Flow Meter, n.d. & Amazon.com, n.d.).

Despite its benefits in accuracy and tracking, digital models require battery power and occasional maintenance for sensor calibration. Examples of models include models from brands such as Microlife, which provide a measurement range from 60 to 840 L/min and offer accuracy within $\pm 10\%$ (Microlife USA, n.d.).

2.4 Discussion of related works and Peak Flow Meter devices

The studies reviewed highlight various applications of image processing in ruler detection, but their focus differs. Wen et al. (2024) applied deep learning for forensic shoe print analysis, while Dou et al. (2022) utilized image processing for water level monitoring. Matuzevičius (2023) demonstrated the effectiveness of synthetic datasets in CNN-based ruler detection. These methods emphasize the growing role of automated measurement tools in healthcare and environmental monitoring.

Peak Flow Meters (PFMs) are essential for asthma management, providing quantitative assessments of lung function. The myPEF Peak Flow Meter is a widely used manual device that is affordable but requires manual recording, leading to errors. In contrast, digital peak flow meters offer accurate readings but are costly. This project bridges the gap by automating data extraction from manual meters using image processing. By leveraging smartphones, the system enables efficient and reliable PEF monitoring without additional costly equipment.

The summary of studies related to automated ruler detection and measurement methods can be referred to in Table 1, while the comparison of Peak Flow Meter devices is presented in Table 2.

Comparison of related works

Table 1: Summary of studies related to automated ruler detection and scale measurement

Author	Wen, Z., Smith, R. M., Connor, M., & Curran, J. M. (2024)	Dou, G., Chen, R., Han, C., Liu, Z., & Liu, J. (2022)	Matuzevičius, D. (2023).
Type of contribution	Method (deep learning for ruler detection)	Tool (automated water level monitoring system)	Framework (synthetic image training model)
Focus	Automated ruler detection in shoe print images for forensic applications	Automated water level recognition using rulers for environmental monitoring	Use of synthetic datasets for ruler detection with deep learning
Primary technology	Image processing, deep learning (SAM model), FAST keypoint detection	Image processing (grayscale, edge detection, Hough transform), CNN	Synthetic image data generation, CNN for ruler detection
Technique used	SAM deep learning model for segmentation, iterative algorithm for ruler graduation detection, pattern analysis of pixel intensity	Grayscale processing, edge detection, Hough transform, CNN for scale value identification	Synthetic data generation, CNN for ruler detection
Accuracy	3% average absolute percentage error, significant improvement in accuracy compared to manual measurements	94.6% recognition accuracy, 24% improvement over traditional methods	Over 95% accuracy in controlled scenarios
Data used	Real-world ruler images, including forensic L-shaped rulers	Captured images of water level rulers, environmental conditions	Synthetic image datasets with variations in angle, illumination, and background
Unique contribution	Introduced deep learning for accurate ruler detection in shoe prints, reducing computational complexity	Developed a cost-effective, scalable method for water level monitoring using image processing and CNN	Demonstrated the use of synthetic data for training deep learning models, showing comparable performance to real-world data
Key algorithm	SAM deep learning model, iterative algorithm with FAST key point detection	CNN fine-tuned for recognizing scale values, Hough transform for tilt correction	CNN trained on synthetic data for robust ruler detection

Comparison of Peak Flow Meter

Table 2: Comparison of Peak Flow Meter Types and Their Reading Methods

Feature	Traditional Peak Flow Meter with automated Web Application (WA) solution	Digital Peak Flow Meter
Measurement Type	Measurement with a traditional PFM and automated data tracking via the WA.	Digital measurement with automatic data capture and display.
Portability	Traditional PFM is portable; WA is accessible on smartphones or tablets.	Portable and may require more maintenance and battery changes.
Ease of use	Simple to use traditional PFM; WA adds user-friendly digital tracking and analysis.	Easy to use with digital displays and automatic tracking.
Data Recording	WA automatically records and stores readings from traditional PFM.	Automatically records and stores measurements digitally.
Accuracy	Accuracy depends on PFM readings; WA offers accurate reading taking, storage and tracking.	High accuracy, typically $\pm 10\%$, with automatic digital readings.
Tracking and Monitoring	WA allows continuous monitoring and historical tracking of peak flow readings.	Digital devices track data over time and often include analysis features.
Cost	Affordable PFM (RM85–95) + free or low-cost WA solution.	More expensive, with prices ranging from \$40 to \$100 or more, depending on features.
Maintenance	Minimal maintenance for traditional PFM; WA requires software updates.	Requires battery changes and sensor calibration periodically.
Design	Traditional PFM is simple and compact; WA runs on a smartphone or tablet.	Digital models have a sleek design, often with Bluetooth and app connectivity.
Suitability	Ideal for rural areas due to the low-cost, battery-independent PFM and offline WA.	Ideal for urban use, offering advanced features and automated tracking with full digital integration.

2.5 Summary

The studies in the review of related works collectively emphasize the use of advanced image processing and machine learning techniques to address challenges in ruler detection and scale measurement for diverse applications, including forensic analysis, environmental monitoring, and ruler tracking. These studies collectively advance the understanding and application of automated ruler detection and scale measurement, with promising implications for peak flow meter, particularly in improving the automation and accuracy of scale detection for peak flow meter reading.

Techniques such as grayscale processing, edge detection and morphological operations can be used to isolate ruler regions in the image. The Hough Transform can align the ruler to correct any skew, while the CNN-based model or FAST keypoint detection can accurately identify scale lines and numerical markers. By calculating the pixel to value ratio, you can convert the pixel distance between scale lines to real-world measurements, ensuring accuracy. The Perspective correction algorithm can further refine measurements when the device is tilted, and the use of synthetic data sets for model training can help improve robustness in various lighting and background conditions, making the system scalable and reliable.

Traditional Peak Flow Meter (PFM) with automated Web Application (WA) solutions are ideal for rural areas due to their low cost, simplicity and ability to work offline, making them an affordable and accessible option for users in remote areas with minimal digital infrastructure. It allows users to manually measure their peak flow and track the results through a WA, which automatically records and stores the data for later analysis. On the other hand, the Digital Peak Flow Meter is better suited for urban areas, offering advanced features such as automatic data tracking, a digital display and connectivity for seamless integration with a mobile app or Bluetooth. However, it comes at a higher cost and requires digital infrastructure and battery maintenance, which may not be very practical in rural settings.

3.0 CHAPTER 3 Requirement Analysis and design

3.1 Introduction

This chapter focuses on the requirements analysis and system design of the proposed Peak Flow Data Extraction system for asthma management. The methodology used in system development is discussed, outlining the approach taken to gather and define system requirements. This chapter also covers requirements elicitation, which involves understanding user needs and gathering insights through surveys. An overview of the project is provided, detailing the goals and functionality of the proposed system. This chapter includes visual representations such as use case diagrams and their specifications, as well as activity diagrams, which illustrate the workflow of the system. Additionally, database design is discussed to determine how data is structured in the system. The implementation of the system is explored, highlighting the tools and technologies used, and the chapter concludes with an evaluation of the system, ensuring its functionality and reliability.

3.2 Requirement Elicitation

This section gathers and documents requirements based on the objectives, the analysis of the previous literature reviews and the results of the interview and meeting with clients. The requirement elicitation involved healthcare professionals, asthma patients and students. These respondents provided insights into the usability and effectiveness of current peak flow monitoring methods.

Data collection process:

- Interviews with healthcare professionals to understand clinical needs.
- Surveys conducted with asthma patients to assess current monitoring challenges.
- Observational studies on how users interact with manual peak flow meters.
- Review of existing asthma monitoring tools to identify gaps.

The details of the meeting are as follow:

- a. Person involved: Healthcare Professional, Supervisors and students who is under supervision.
- b. Date: 05/08/2024
- c. Venue: Hospital Pengajar dan Pusat Perubatan UNIMAS

The requirements gathered during the interview and meeting were divided into 2 sections: Functional Requirements and Non-Functional Requirement.

a. Functional Requirements

- i. The system must ensure accurate extraction of peak flow meter readings, with minimal deviation.
- ii. The system must be capable of capturing and processing images of peak flow meter readings.

- iii. The system should automate data extraction from images and store the extracted data in a secure database.
- iv. Historical readings should be displayed with trends and visualizations, such as charts, for easy interpretation.
- v. User authentication is required to ensure secure access to the system.
- vi. The user is required to perform three readings for each measurement, with the highest reading being recorded.
- vii. The graph should display data for a 12-month period by default.
- viii. A reminder alert feature should be implemented to notify users once a week.
- ix. A user interface should be developed to show the total number of patients, as well as the breakdown of male and female patients.
- x. A separate line should be displayed on the graph for each individual patient, representing their readings.

b. Non-Functional Requirements

- i. The system must be compatible with various lighting conditions and camera angles to ensure reliable data extraction.
- ii. The system should offer a user-friendly interface with a responsive design for easy navigation on different devices.
- iii. The system must be scalable to accommodate multiple users and devices, ensuring its continued performance as the user base grows.

3.3 Architecture

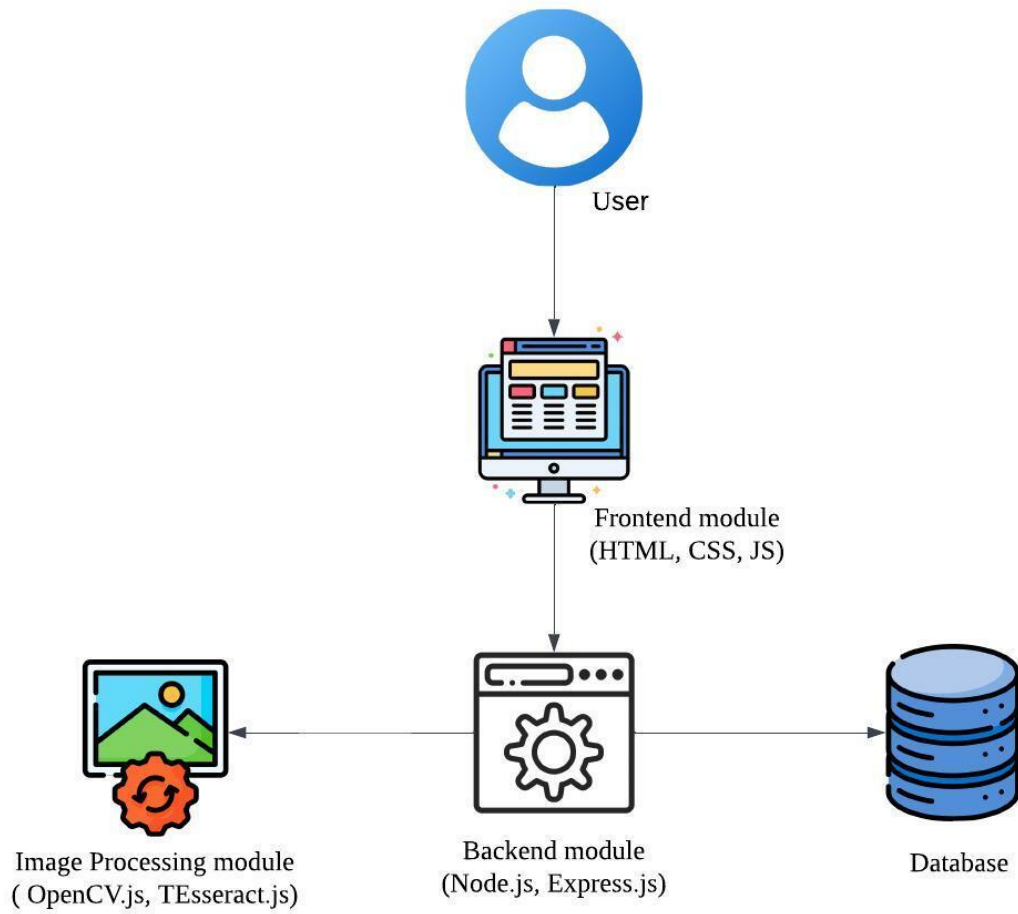


Figure 4: System Architecture

Figure 4 shows the architecture diagram which consist of the following components:

- i. Frontend Module: HTML, CSS, JavaScript for capturing and displaying readings.
- ii. Backend Module: Node.js with Express.js for processing requests and data storage.
- iii. Image Processing Module: OpenCV.js and Tesseract.js for edge detection and OCR.
- iv. Database Module: MySQL for storing extracted readings and user information.

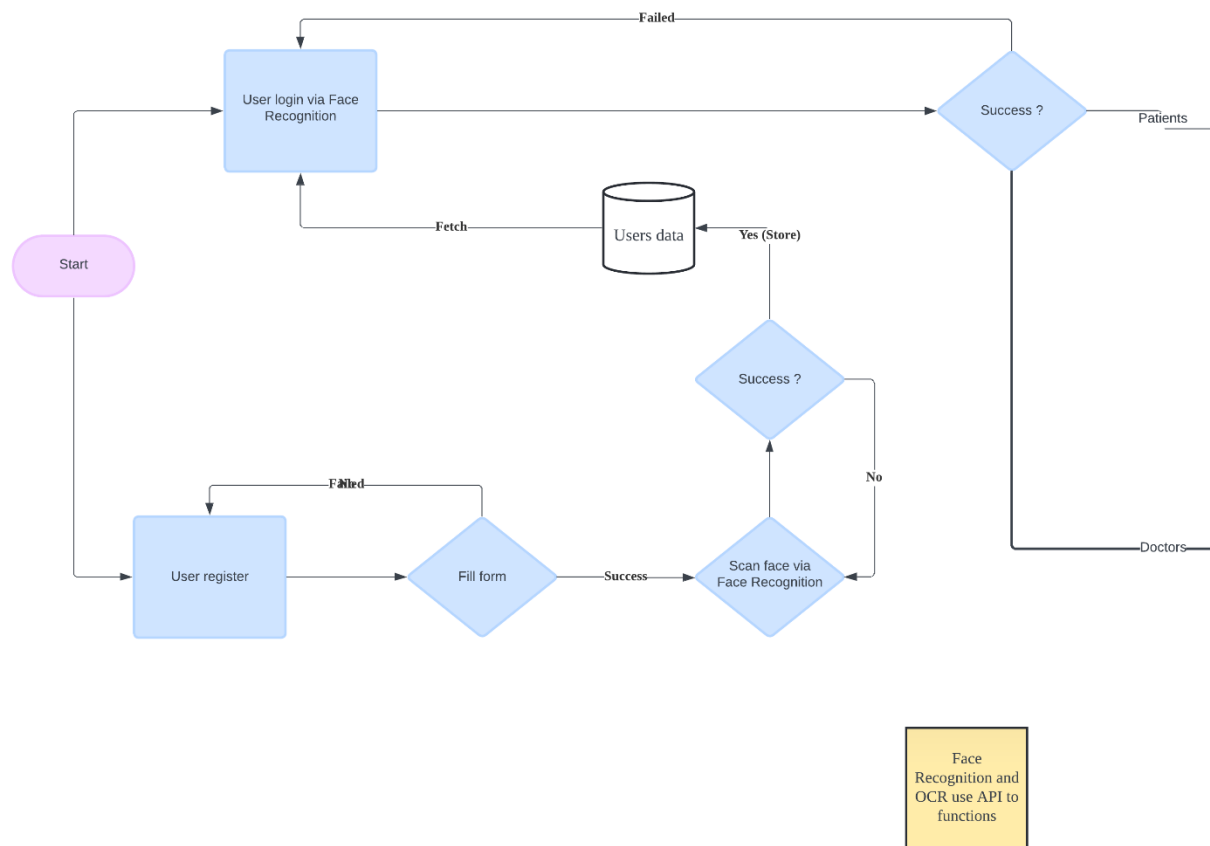


Figure 5: Flowchart of the system (left side of the flowchart)

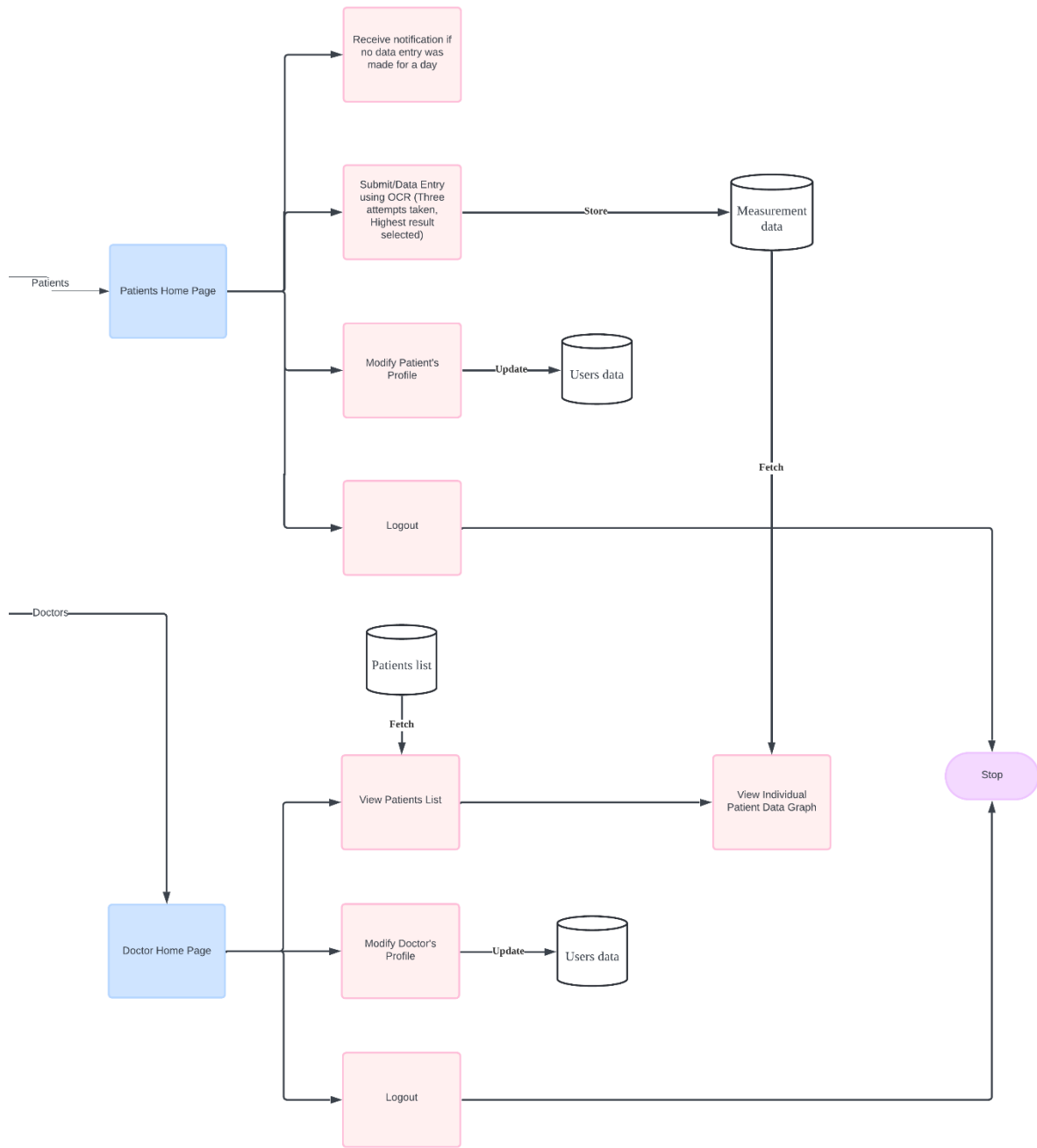


Figure 6: Flowchart of the system (right side of the flowchart)

Figure 5 and Figure 6 shows a flowchart that describes the process flow of the myPEF system, starting with user registration and login. Users register by filling out a form and scanning their face through facial recognition, and when successful, their data is saved. To log in, facial recognition authenticates the user; if it fails, the process repeats until it succeeds. After logging in, the patient accesses the main page to submit the measurement data via OCR, which selects the best result after three attempts and saves it. A notification is sent if no data is entered for a day. Patients can also view individual data graphs or modify their profiles. Doctors access their home page to view patient lists, individual patient data, or modify their profiles. Both users can log out at any time. The system leverages APIs for facial recognition and OCR, ensuring data processing, storage and updates are smooth and efficient, with proper feedback loops for failed processes.

3.4 Agile Software Development

3.4.1 Planning

For this project, the Agile methodology is applied, following the framework stages of planning, design, development, testing, deployment, and review (Agile Methodology, 2024). The planning phase is critical as it sets the foundation for the entire project. During this phase, the primary focus is on understanding the needs, expectations, and challenges faced by the users, which include asthma patients, healthcare providers, and system users. A detailed survey is conducted, consisting of interviews and meetings with stakeholders to gather insights into the current asthma management systems, identify pain points, and assess the functionality required for the new system. This information helps in defining clear project objectives, which ensure that the system meets both user and business needs. Additionally, the planning phase involves conducting extensive research on the existing market and technologies to understand the strengths and weaknesses of current asthma monitoring tools, allowing the team to design a solution that fills gaps in the existing systems. The creation of user personas is also an important activity, as it helps to define the key user types and their specific needs, ensuring that the system is designed to serve diverse user groups effectively. Functional and non-functional requirements are outlined, focusing on both the core features the system must have, as well as criteria such as performance, security, and usability. Moreover, this phase includes the establishment of a clear project timeline, defining milestones and deliverables, and creating a roadmap to track progress throughout the project. This detailed planning ensures that all stakeholders are aligned and that the project moves forward in a structured manner. The insights gained from the planning phase directly inform the design and development stages, where the system will be created and fine-tuned to meet the identified needs of users and improve asthma management practices.

3.4.2 Design

a. Use Case Diagram

Use case diagram in figure below shows the vision for “Peak Flow Data Extraction for Asthma Monitoring and Management System”

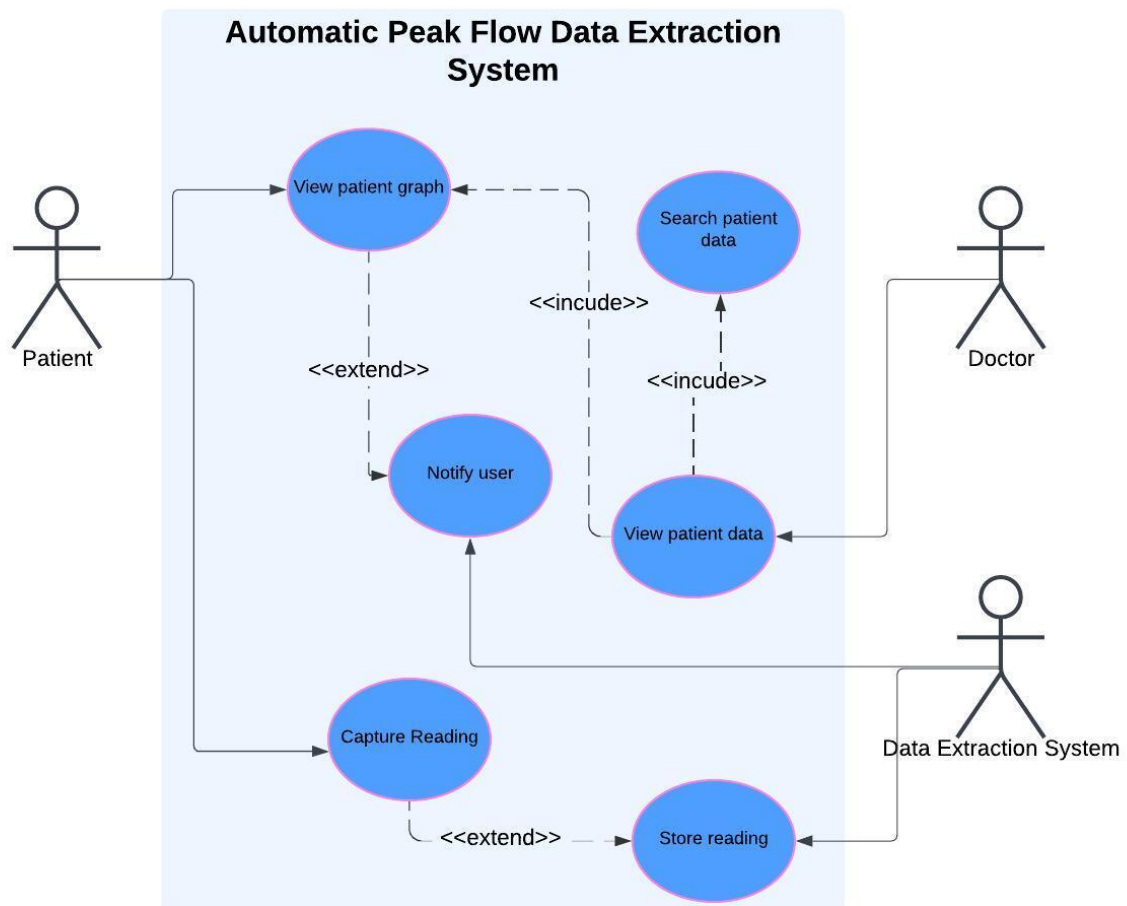


Figure 7: Use case diagram

b. Use Case Specification

i. Patient

Table 3: Use Case Specification for Capture Reading

Use Case Number	UC1
Use Case Name	Capture Reading
Actor	Patient
Use Case Description	The patient captures an image of the peak flow meter for processing.
Pre-condition	The patient has access to the system and a camera-enabled device.
Post-condition	The image is live captured and sent for processing.
Main Flow	1. The patient initiates the image capture process. 2. The camera live captures the image of the peak flow meter in real time. 3. The image is sent to the backend for processing.
Alternative Flow	If the image is blurry or hard to read, the system prompts the patient to retake the image.
Exception Flow	If the camera is inaccessible, the system displays an error message.

ii. Doctor

Table 4: Use Case Specification for Search Patient Data

Use Case Number	UC2
Use Case Name	Search Patient Data
Actor	Doctor
Use Case Description	The doctor searches for a patient's data using the patient's name or Identification card (IC) number.
Pre-condition	The doctor is logged into the system and has the necessary permissions.
Post-condition	The patient's data is displayed.
Main Flow	1. The doctor enters the patient's name or IC number in the search bar. 2. The system retrieves the patient's data from the database. 3. The data is displayed to the doctor.
Alternative Flow	If no matching records are found, the system displays a message, "No patient data found."
Exception Flow	If the database is inaccessible, the system displays an error message.

Table 5: Use Case Specification for View Patient Data

Use Case Number	UC3
Use Case Name	View Patient Data
Actor	Doctor
Use Case Description	The doctor views the detailed data of each specific patient, including height, weight and historical readings.
Pre-condition	The doctor is logged into the system and has the necessary permissions.
Post-condition	Each patient's data is displayed in detail.
Main Flow	1. The doctor login into the system and redirect to viewing page. 2. The system retrieves all the patient's data and display it in one page. 3. The system displays the patient's data, displaying every data recorded.
Alternative Flow	If the data is incomplete, the system displays a message, "Some data might be missing." Inside the table shown.
Exception Flow	If the database is inaccessible, the system notifies the doctor with an error message.

iii. Data Extraction System

Table 6: Use Case Specification for Notify User

Use Case Number	UC4
Use Case Name	Notify User
Actor	Data Extraction System
Use Case Description	This use case notifies users about abnormal readings or trends in their peak flow measurements or to remind patients to record their reading for the day.
Pre-condition	The system has access to stored patient readings.
Post-condition	The user is notified via the system interface or other channels
Main Flow	1. The system analyzes the peak flow readings. 2. If an abnormal trend or empty record is detected, the system generates a notification. 3. The system sends the notification to the user.
Alternative Flow	If the notification is not read by patient, the system stores the notification for later delivery.
Exception Flow	If the notification system fails, the system logs the error for administrative review.

Table 7: Use Case Specification for Store Reading

Use Case Number	UC5
Use Case Name	Store Reading
Actor	Data Extraction System
Use Case Description	The system stores the extracted reading from the captured image in the database.
Pre-condition	A valid reading is available after processing.
Post-condition	The reading is saved in the database with a timestamp and user information.
Main Flow	1. The system receives the processed reading. 2. The reading is associated with the user ID. 3. The reading is stored in the database.
Alternative Flow	If the reading already exists for the same timestamp, the system updates the existing record.
Exception Flow	If the database is unavailable, the system stores the reading locally for later synchronization.

iv. Patient and Doctor

Table 8: Use Case Specification for View Patient Graph

Use Case Number	UC6
Use Case Name	View Patient Graph
Actor	Patient, Doctor
Use Case Description	This use case allows the patient or doctor to view a graphical representation of the patient's peak flow readings over time.
Pre-condition	The user (patient or doctor) is logged into the system.
Post-condition	The graph is displayed with accurate data retrieved from the database.
Main Flow	1. The user navigates to the "View Graph" page. 2. The system retrieves the peak flow readings from the database. 3. The system generates and displays the graph.
Alternative Flow	If no data exists, the system displays previous month of graph data.
Exception Flow	If there is a database error, the system notifies the user with a message like, "Unable to load graph. Please try again later."

c. Activity Diagram

Figure below shows the activity diagram for the proposed peak flow data extraction system. It highlights the sequence of actions performed by the system and the actors, patient, doctor and the system itself. The process begins with the patient login into the system using face recognition and by using the camera, patient will capture an image of the peak flow meter. The captured image is sent to the system for analysis, where the image processing module applies edge detection and Optical Character Recognition (OCR) techniques to extract the numerical reading from the meter.

If the reading extraction process is successful, the extracted reading is stored in the database along with the associated metadata, such as timestamp and user ID. If the extraction fails (e.g., due to blurry images or improper alignment), the system notifies the patient to retake the image. Once the reading is successfully stored, the system allows the patient or doctor to view the recorded data and analyse historical trends through graphical representations. This ensures that users can monitor peak flow readings accurately and take appropriate action if necessary.

For the doctor, the process will begin the same as patient by login into the system using face recognition. After that, the doctor will be redirected into the main page where they can view all patient's data. They will have the option to either search or directly clicking a patient name before viewing the specific patient historical data and trends in graph view.

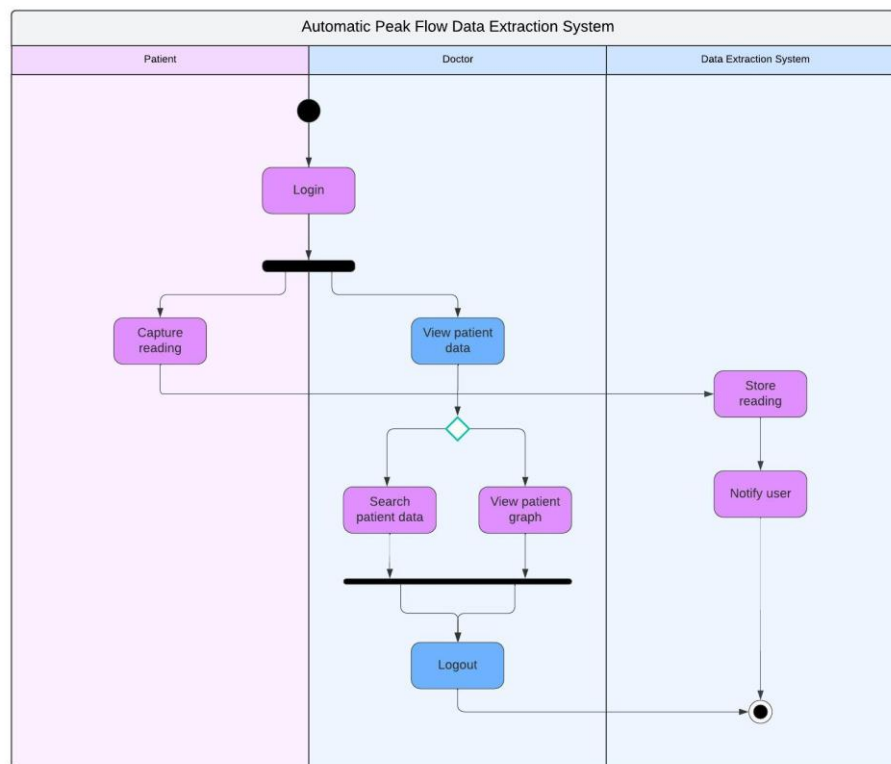


Figure 8: Activity Diagram for a Automatic Peak Flow Data Extraction System

d. Sequence Diagram

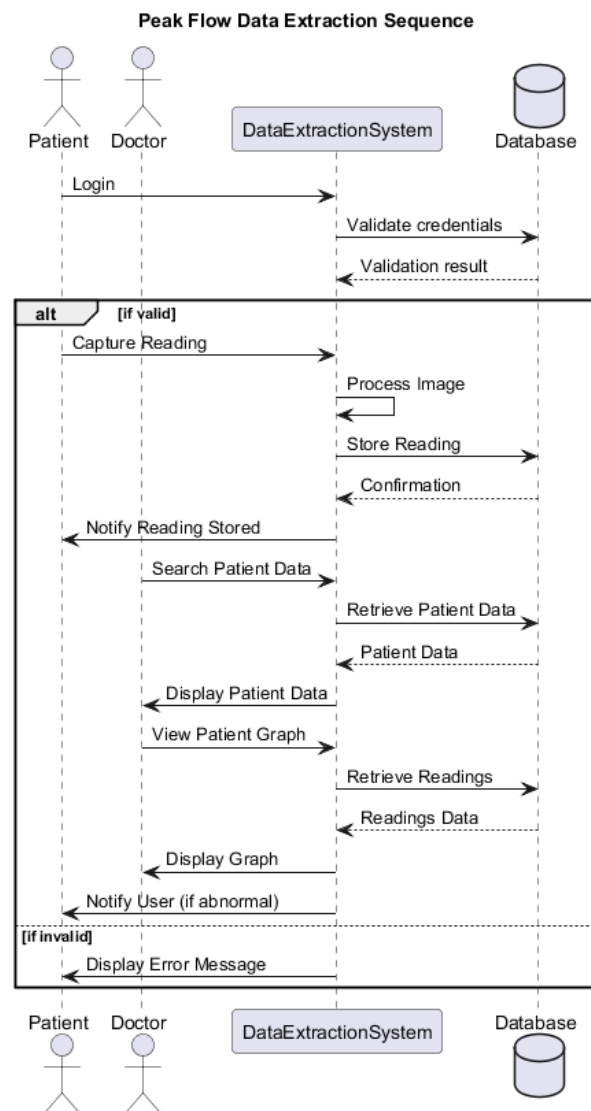


Figure 9: Sequence diagram

The sequence diagram on Figure 9 visually represents the step-by-step interactions between different components of the MyPEF system over time. It illustrates how a user, the system, and the database communicate to perform key operations such as user registration, login, and peak flow measurement submission. Each component is represented by a lifeline, showing its active duration, while arrows between lifelines represent messages exchanged during the process. The diagram ensures that the system's functionality is executed in the correct order, from user input to system validation and database interactions. For example, in the login process, the user enters their credentials, and the system validates them by querying the database. The database then responds with either a success or failure message, allowing the system to either grant access or display an error. This diagram helps developers understand how different components interact dynamically, ensuring proper implementation of business logic and data flow within the system.

e. User Interface design

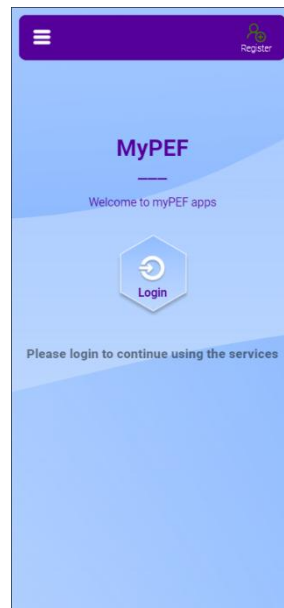


Figure 10: Home/Main/Login Page

Figure 10 are the homepage of the Web Application (WA). It offers a few functionalities which is the register button and the login button. These two buttons will redirect the user to its corresponding pages.

The image shows a mobile application interface for the 'Register' page. At the top, there is a purple header bar with a white hamburger menu icon on the left and a 'Login' button with a circular arrow icon on the right. Below the header, the text 'Register' is displayed in a bold, dark font, followed by 'Welcome to myPEF apps' in a smaller font. Below this, there are several input fields: 'First Name', 'Last Name', 'Mobile Number', 'IC Number (No. ⁺)', 'Height (cm)', 'Email address', and a 'Gender' dropdown menu. At the bottom, there is a purple button with the text 'Scan face'.

Figure 11: Register Page

Figure 11 show the registration pages. These pages will be displayed when the user clicks the register button on the home page. The user will then be prompted to input their data to store in the database. While the user is inputting their data, the system will check and validate each input to ensure it is in the correct format. After the user has finished inputting their data, they can proceed to scan their face for face recognition purposes

which is shown on Figure 12. The user can only scan their face if they have entered the correct data in all forms.

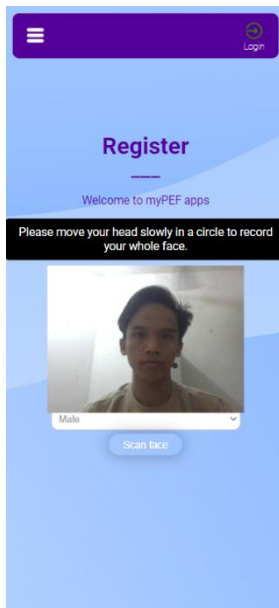


Figure 12: Face recognition

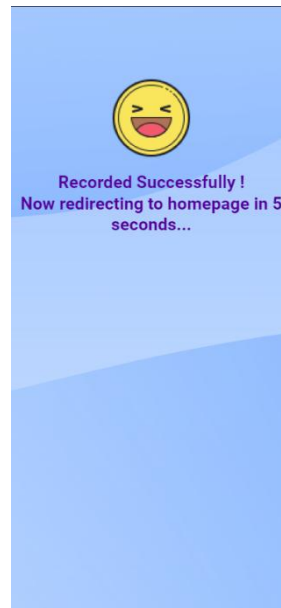


Figure 13: Success message

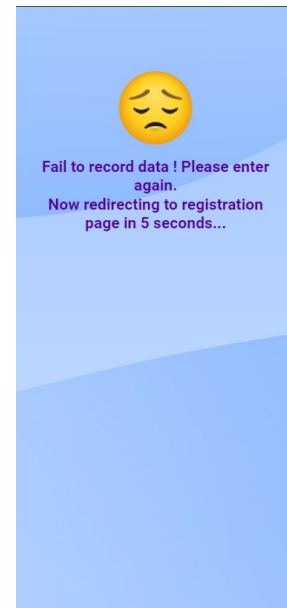


Figure 14: Failed message

Users will be then redirected to the corresponding pages which is the successful message page or fail message page which is shown on Figure 13 and Figure 14. These two pages then redirect user after 5 second to the homepage or registration page based on their result.

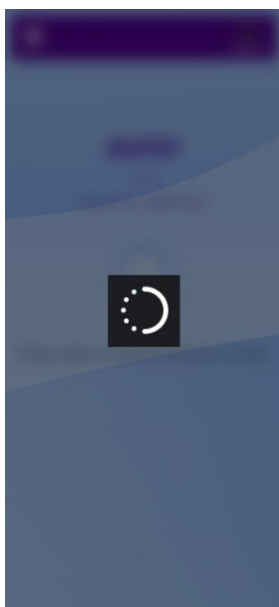


Figure 15: Recognition loader

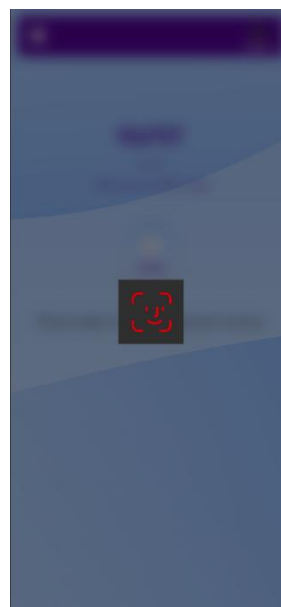


Figure 16: Unauthorize face

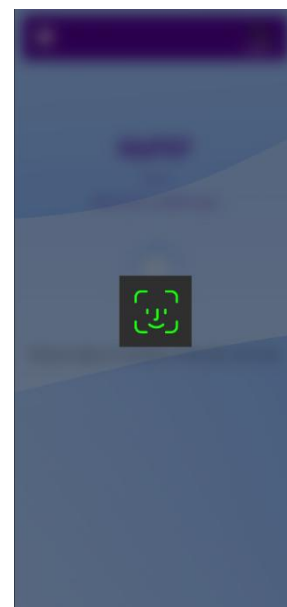


Figure 17: Authorize face

Figure 15, Figure 16 and Figure 17 are displayed after users click the login button on the homepage. Figure 15 is shown for 6 seconds while face-api.js is recognizing the face from

the camera. Figure 16 appears if the user detected is unauthorized or if no face is detected. Figure 17 is shown if a face is detected and registered in the system. Face-api.js will then detect and recognize who the person is in the camera/webcam.

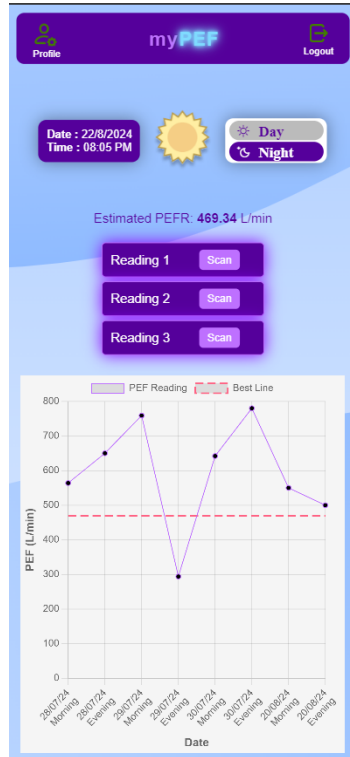


Figure 18: Morning patient page

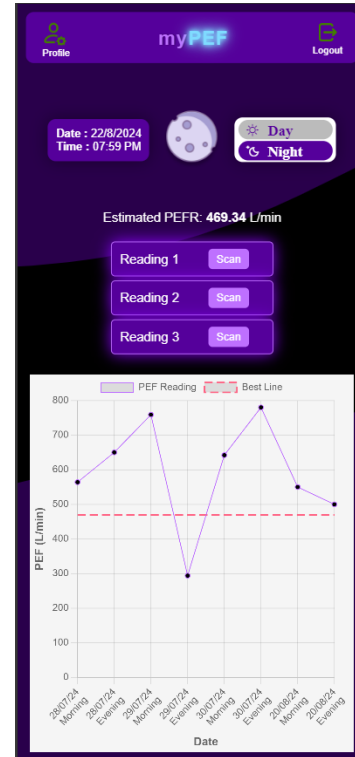


Figure 19: Night patient page



Figure 20: Reading existence indicator

Figure 18, Figure 19 and Figure 20 shows the patient homepage, which displays the current date and time, along with a sun and moon icon indicating whether it is day or night. The page also displays a tick icon if there is a reading for current date in the database, whether it is a day or night reading. The page offers both light and dark modes based on the time. The main function of this page is the scan button, which uses the user's camera to record the reading directly from the tool, which is the Peak Flow Meter. The page also shows the Normal best line of Peak Expiratory Flow Rate and also the graph of user highest PEF reading.



Figure 21: Reading taking

Figure 21 shows the camera being used by the user to record the data directly from the Peak Flow Meter.

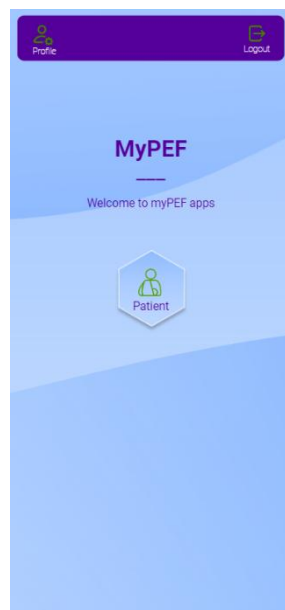


Figure 22: Doctor Home page

Figure 22 shows the doctor homepages that has a function to view patient list when clicking on the patient button.

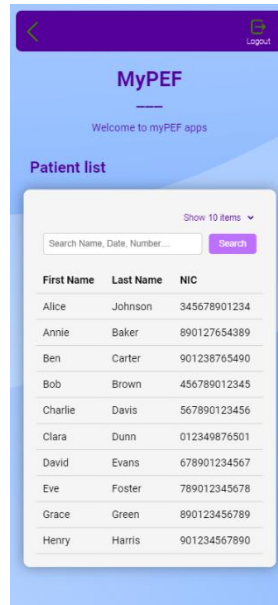


Figure 23: List of patients viewing page

Figure 23 shows the patient list that exists in the database. These pages offer a few functionalities which is search and filter function. When user click one of the patient rows, it will redirect the user to view the reading of the user in graph form.

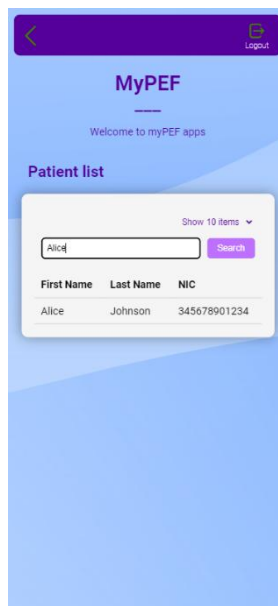


Figure 24: Search functions

Figure 24 shows the reading of one of the patients after searching “Alice” inside the searching function form.

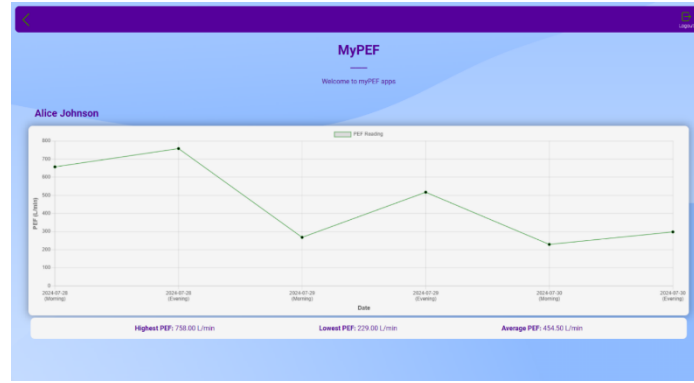


Figure 25: Graph view for specific patients

The graph on Figure 25 includes the readings for both morning and night, with the data for one day represented on two x-axes.

f. Class Diagram

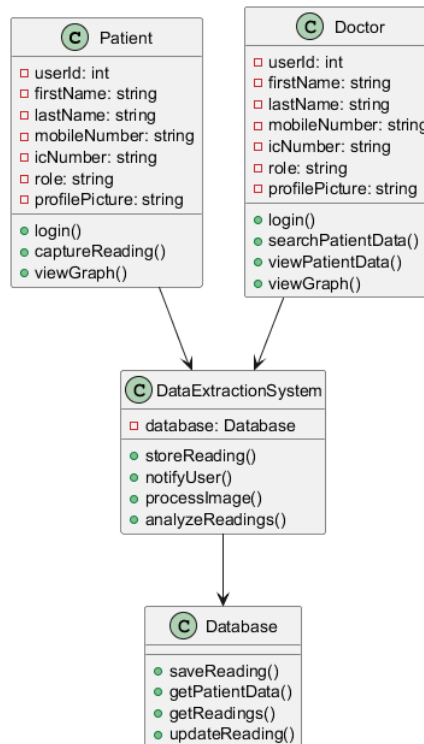


Figure 26: Class diagram

The class diagram in Figure 26 provides a structural blueprint of the MyPEF system by defining its classes, attributes, methods, and relationships. It represents the core entities, such as User, PeakFlowMeasurement, and Session, along with their properties and functions. Each class contains attributes that define its characteristics, such as a User having an ID, name, email, and password, while its methods include registerUser() and loginUser(). The diagram also highlights relationships between classes, such as association, where a user can have multiple peak flow measurements stored in the system. Inheritance may also be used if roles like Admin extend from the base User class. This diagram is crucial for database design and backend development, ensuring that all

entities are well-structured and logically connected. By organizing the system into classes and relationships, developers can maintain scalability, reusability, and efficient data management.

3.5 Summary

In this chapter, the requirements analysis and design for a Peak Flow Data Extraction system are thoroughly discussed. This chapter begins with an introduction to the methodology used in gathering the necessary system requirements, followed by a detailed look at user requirements and survey results. A project overview provides a comprehensive understanding of the system's objectives. Visual diagrams such as use case diagrams and activity diagrams are presented to showcase the workflow and system interactions. Database design is also discussed to create a data structure in the system. This chapter contributes by defining the system's requirements, ensuring that functional and non-functional needs are met. The design section provides a structured approach to system implementation through use case diagrams, activity diagrams, and database modelling. These elements collectively support the development of an efficient and scalable solution for automated peak flow data extraction, bridging the gap between manual and digital peak flow monitoring solutions.

4.0 CHAPTER 4 Implementation

4.1 Introduction

This chapter describes the implementation phase of the Automating Peak Flow Data Extraction for Asthma Monitoring and Management project. It elaborates on the development activities and processes that transform the project's design into a functional and operational system. The purpose of this chapter is to demonstrate how the system prototype was constructed and configured, including the tools, libraries, frameworks, and coding languages utilized throughout the implementation process.

The chapter begins by detailing the prototype development, including system architecture and module functionalities. It proceeds to explain the installation and configuration of all essential libraries and dependencies required for both client-side and server-side operations. Furthermore, it outlines the integration of multiple technologies such as image processing, optical character recognition (OCR), face recognition, and database management systems. Finally, the chapter presents key findings and observations made during the implementation, followed by a summary to conclude the chapter's coverage.

4.2 System Prototype

The system prototype consists of several interconnected modules that work together to automate the extraction and recording of peak flow data from images.

There are two roles in this system that have been designed to specify the user after they log in to the system. Each user role's page has a distinct interface and functionality that is displayed on the system. They will be redirected to each user's role after successfully logging in to the system. 'Patient' and 'Doctor' are the two user roles. The system will have eight main modules that have their own function.

4.2.1 Libraries Installation

This project was developed using Node.js version 20.12.2 as the server environment. Several JavaScript libraries and Node.js modules were installed to support functionalities such as server configuration, face recognition, image processing, text extraction, and database operations.

Table 9: Libraries details

Libraries Name	Version	Purpose / Description
express	4.19.2	Web framework for Node.js, handling routing and server creation.
express-session	1.17.4	Middleware for managing user sessions on the server.
multer	1.4.5-lts.1	Middleware for handling multipart/form-data, especially for image uploads.
express-fileupload	1.4.0	Simple middleware for uploading files.
mysql2	3.9.1	Connector for interacting with MySQL databases.
face-api.js	0.22.2	JavaScript library for face detection, face landmark detection, and face recognition in the browser.
opencv.js	4.9.0	JavaScript binding of OpenCV for real-time image processing and computer vision tasks.
tesseract.js	4.1.1	JavaScript library for Optical Character Recognition (OCR) based on Tesseract engine.
chart.js	4.4.3	JavaScript library for creating interactive charts and graphs.
jquery	3.7.1	JavaScript library for Document Object Model (DOM) manipulation and simplifying AJAX requests.
nodemon	3.1.0	Utility for automatically restarting the Node.js server during development upon file changes.

All libraries were installed using Node Package Manager (npm) through the “npm install” command executed in the project directory. Specific versions were managed and recorded in the package.json file to ensure consistency and compatibility throughout the development process. Additionally, nodemon was installed as a development dependency using “npm install –save-dev nodemon”.

Front-end libraries were integrated through Content Delivery Networks (CDN) or local file imports.

- face-api.min.js
- opencv.js
- tesseract.min.js
- chart.js and jquery

These libraries enabled functionalities such as face recognition in the browser, real-time image processing, dynamic graph rendering, and text recognition from images without requiring external installations on the client’s machine.

4.2.2 System Hierarchy

Hierarchical architecture views the whole system as a hierarchy structure, in which the software system is decomposed into logical modules or subsystems at different levels in the hierarchy (Hierarchical Architecture, 2023). At the top level, there is usually a system, which can be composed of subsystems, each responsible for a specific functionality or aspect. These subsystems can further be divided into smaller modules or components, forming a layered structure. The hierarchy allows for the division of labor, enabling efficient management, maintenance, and understanding of the system. It provides a clear and systematic way to analyse and design complex systems, ensuring that each component fits into a larger framework and contributes to the overall functioning of the system.

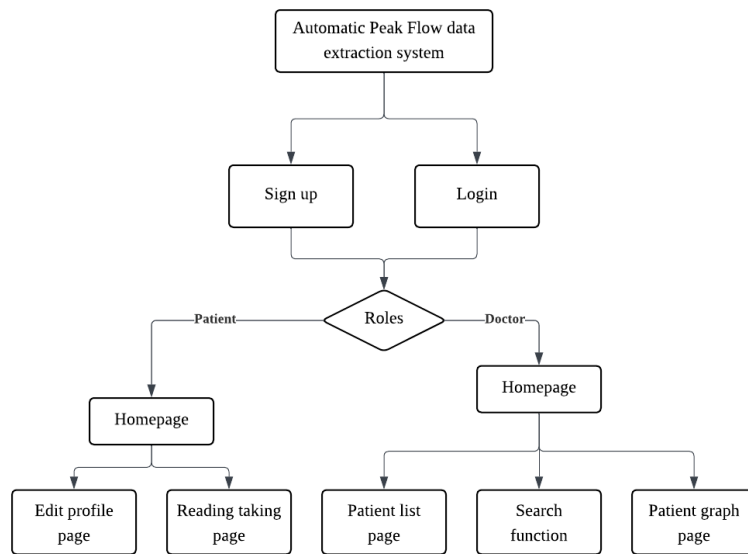


Figure 27: System hierarchy

There are nine modules in the system hierarchy for Automatic Peak Flow data extraction system which are Sign up, Login, Homepage, Edit profile page, Reading taking page, patient list page, search function, Patient graph page and Database.

a. Signup Module

```
<form id="registrationForm">
  <!-- Top Row -->
  <div class="input-row" id="top-input-row">
    <input type="text" placeholder="First Name" id="firstName" name="firstName" oninput="validateInput(this)" required>
    <input type="text" placeholder="Last Name" id="lastName" name="lastName" oninput="validateInput(this)" required>
  </div>
  <!-- Middle Row -->
  <div class="input-row" id="mid-input-row">
    <input type="text" placeholder="Mobile Number" id="mobileNumber" name="mobileNumber" oninput="validateInput(this)" required>
    <input type="text" placeholder="IC Number ( No '-' )" id="icNumber" name="icNumber" oninput="validateInput(this)" required>
  </div>

  <div class="input-row">
    <input type="text" placeholder="Height (cm)" id="height" name="height" oninput="validateInput(this)" required>
  </div>

  <div class="input-row">
    <input type="text" placeholder="Email address" id="emailAddress" name="emailAddress" oninput="validateInput(this)">
  </div>

  <div class="input-row">
    <select name="gender" required>
      <option value="" disabled selected>Gender</option>
      <option value="Male">Male</option>
      <option value="Female">Female</option>
    </select>
  </div>

  <div class="take-picture-button1"><button type="button" id="scanFaceButton" onclick="scanFace()">Scan face</button></div>
</form>
```

Figure 28: Signup front-end

The Signup Module is a crucial component of the project that enables new users to register by providing their personal information along with facial data. It is designed with a responsive and user-friendly front-end interface using HTML, CSS, and JavaScript, and integrates face-api.js to incorporate real-time face detection and recognition during the registration process.

The HTML structure in Figure 28 includes a form that collects essential user data such as first name, last name, mobile number, Identification Card (IC) number, height, email address, and gender. Validation is performed on each input field using regular expressions to ensure proper data formatting, for example, mobile numbers are checked for Malaysian formats (starting with 01), and IC numbers must follow the correct numeric structure. The form also includes a “Scan Face” button that triggers the facial recognition process, encouraging users to move their heads in a circular motion for a full scan, enhancing face detection accuracy.

```
// Initialize face-api.js models
function loadModelsAndStart() {
  Promise.all([
    faceapi.nets.tinyFaceDetector.loadFromUri('/models'),
    faceapi.nets.faceLandmark68Net.loadFromUri('/models'),
    faceapi.nets.faceRecognitionNet.loadFromUri('/models')
  ]).then(() => {
    console.log('Face-api.js models loaded');
  });
}
```

Figure 29: Pre-trained model face-api.js

The module uses the face-api.js library as shown in Figure 29, a JavaScript API built on top of TensorFlow.js, to perform facial detection and recognition in the browser without requiring any external installation. The library loads three pre-trained models:

- tinyFaceDetector for lightweight face detection
- faceLandmark68Net for recognizing key facial features (eyes, nose, mouth, eyebrows.)
- faceRecognitionNet for computing face descriptors for recognition.

```
async function scanFace() {  
  
    const firstName = document.getElementById('firstName').value.trim();  
    const lastName = document.getElementById('lastName').value.trim();  
  
    // Check if the name already exists  
    const response = await fetch('/check-name', {  
        method: 'POST',  
        headers: {  
            'Content-Type': 'application/json'  
        },  
        body: JSON.stringify({ firstName, lastName })  
    });  
    const result = await response.json();  
  
    if (result.exists) {  
        showError("Name already taken.");  
        return; // Stop the function execution  
    }  
  
    const inputs = document.querySelectorAll('#registrationForm input');  
    let isValid = true;
```

Figure 30: Scan face function

```
    if (isValid) {  
        try {  
            enableWebcam();  
        } catch (error) {  
            console.error("Error enabling camera:", error);  
        }  
    }  
}
```

Figure 31: Webcam access

Once the user clicks the Scan Face button, the app checks for duplicate names through a backend endpoint (/check-name) as shown in Figure 30. If the name is unique and all fields are validated, it accesses the webcam using “getUserMedia()” and begins scanning the face in real-time as shown in Figure 31. A visual instruction is shown to guide users to move their heads slowly, improving the reliability of the recognition data.

```

async function captureAndSaveImages() {
  const firstName = document.getElementById('firstName').value.trim();
  const lastName = document.getElementById('lastName').value.trim();

  try {
    // Loop through each image index
    for (let i = 1; i <= 15; i++) {
      // Capture image from canvas
      canvasElement.width = video.videoWidth;
      canvasElement.height = video.videoHeight;
      canvasCtx.drawImage(video, 0, 0, canvasElement.width, canvasElement.height);

      // Convert canvas image to Blob in JPEG format
      const blob = await new Promise(resolve => canvasElement.toBlob(resolve, 'image/jpeg'));

      // Prepare FormData to send the image
      const formData = new FormData();
      formData.append('firstName', firstName);
      formData.append('lastName', lastName);
      formData.append('imageIndex', i);
      formData.append('imageFile', blob, `${i}.jpg`);

      // Save image to server
      const response = await fetch('/save-image', {
        method: 'POST',
        body: formData
      });
    }
  }
}

```

Figure 32: Saving picture

During the scan in Figure 32, 15 face images are captured at half-second intervals. These images are converted to JPEG blobs and sent to the server via “FormData” using a POST request to the “/save-image” endpoint. Each image is saved with a corresponding index number, forming a dataset for that individual user. Once all images are successfully uploaded, a final POST request is made to the “/register” endpoint, submitting the user’s textual data in JSON format.

```

if (result.success) {
  // Trigger server restart after successful registration
  const restartResponse = await fetch('/restart-server', {
    method: 'POST'
  });
  console.log(await restartResponse.text()); // Log restart response

  console.log('Registration successful:', result.message);
  window.location.href = '/success-message.html'; // Redirect to success page
} else {
  console.error('Registration failed:', result.message);
  showError(result.message);
  window.location.href = '/failed-message.html'; // Redirect to failure page
}

```

Figure 33: Post-registration flow

If registration is successful, a secondary request is made to “/restart-server” (to reload models and reset in-memory user data). The user is then redirected to a success page. Otherwise, a failure page is displayed as shown in Figure 33.

b. Login Module

The Login Module enables authenticated access to the application using facial recognition. It offers a seamless, password-less login experience by utilizing face-api.js, a powerful JavaScript library for facial recognition, in the browser. The module integrates a webcam-based face matching system that detects and verifies users in real-time.

Once the user clicks the “Login” button, the system enables the webcam and begins capturing live video. The application loads pre-trained models from face-api.js specifically the face recognition, landmark detection, and face detection models. These models are essential for extracting facial features and comparing them with stored descriptors.

```
async function loadLabeledImages() {
  let labels = [];
  try {
    const response = await fetch('/labels.json');
    const data = await response.json();
    labels = data.labels;
  } catch (error) {
    console.error('Error loading labels:', error);
  }

  console.log('labels:', labels);

  return Promise.all(
    labels.map(async (label) => {
      const descriptions = [];
      for (let i = 1; i <= 2; i++) {
        try {
          const img = await faceapi.fetchImage(`../labeled_images/${label}/${i}.jpg`);
          const detections = await faceapi.detectSingleFace(img).withFaceLandmarks().withFaceDescriptor();
          if (!detections) {
            console.log(`No face detected in image ${i} of ${label}.`);
            continue;
          }
          descriptions.push(detections.descriptor);
        } catch (error) {
          console.error(`Error processing image ${i} of ${label}:`, error);
        }
      }
      return new faceapi.LabeledFaceDescriptors(label, descriptions);
    })
  );
}
```

Figure 34: Load labeled images function

Labeled face descriptors are fetched from the server as shown in Figure 34, where each label corresponds to a registered user's identity. The loadLabeledImages() function dynamically loads these descriptors by scanning saved images and computing their facial embeddings. These descriptors are then passed to the “FaceMatcher” class to perform real-time face comparison.

```

async function scanFace(video) {
  try {
    const labeledDescriptors = await labeledDescriptorsPromise; // Wait for labeledDescriptorsPromise to resolve

    if (!labeledDescriptors || labeledDescriptors.length === 0) {
      console.error('No labeled descriptors found.');
```

Figure 35: Scanning face function

```

    if (mostRecognizedPerson && mostRecognizedPerson !== "unknown") {
      console.log(`Most recognized person: ${mostRecognizedPerson} (${recognizedFaces.get(mostRecognizedPerson)} times)`);

      // Make a POST request to the server to fetch user_id and role
      const fetchUserId = async () => {
        try {
          const response = await fetch('/fetchUserId', {
            method: 'POST',
            headers: {
              'Content-Type': 'application/json',
            },
            body: JSON.stringify({ person: mostRecognizedPerson }),
          });
        }
      };
    }
  }
}
```

Figure 36: POST request

In Figure 35, the system continuously scans the user's face for 10 seconds. During this time, it records how many times each label (person) is recognized. The one with the highest recognition count is selected as the most probable user. If the face is matched successfully and is not labeled as "unknown," a POST request is sent to the backend to retrieve the user's unique ID and role as shown in Figure 36.

```

        if (response.ok) {
            const data = await response.json();
            if (data.success) {
                // Store user_id in session storage
                sessionStorage.setItem('user_id', data.user_id);
                console.log('User ID stored in session:', data.user_id);

                // Determine the redirect based on role
                if (data.role.includes('Patient')) {
                    await showGif('successGif');
                    window.location.href = 'patient-home.html';
                } else if (data.role.includes('Doctor')) {
                    await showGif('successGif');
                    window.location.href = 'doctor-home.html';
                } else {
                    console.error('Unknown role:', data.role);
                    //unknown role or redirect to a default page
                }
            } else {
                console.error('Failed to fetch user_id from server.');
```

Figure 37: User role redirection

Once the user is authenticated as shown in Figure 37, the application uses the retrieved role to redirect the user to the appropriate homepage. For instance, users with the "Patient" role are directed to patient-home.html, whereas "Doctor" role users are sent to doctor-home.html. The "sessionStorage" is also updated with the user's ID for session tracking.

```

async function showGif(gifId) {
    const loaderGifPath = "img/loader.gif";
    const successGifPath = "img/SuccessFaceID.gif";
    const failureGifPath = "img/FailFaceID.gif";

    const gifPath = gifId === "loaderGif" ? loaderGifPath : gifId === "successGif" ? successGifPath : failureGifPath;
    const gifDuration = gifId === "successGif" ? 3000 : gifId === "failureGif" ? 2600 : null;

```

Figure 38: Animation feedback

The module enhances user experience by providing immediate visual feedback using GIFs as shown in Figure 38. A loading animation is displayed during the scanning process, followed by either a success or failure GIF depending on the result. If no match is found or the recognized face is labeled "unknown," the login attempt is denied and a failure GIF is shown. Webcam access errors and other issues are handled gracefully and logged in the console.

c. Homepage Module – Patient

```
<div class="header">
  <div class="rectangle">
    <div class="group-icon-header">
      <a href="profile-setting.html">
        <div class="profile-button">
          
          <span>Profile</span>
        </div>
      </a>

      <div class="welcome-text-2">
        <h2>my<span class="neon">PEF</span></h2>
      </div>

      <a href="index.html">
        <div class="register-button">
          
          <span>Logout</span>
        </div>
      </a>
    </div>
  </div>
</div>
```

Figure 39: Patient homepage header

The header section of the patient homepage includes a profile button, a logout button, and a welcoming application title (myPEF) as shown in Figure 39. These elements are laid out in a responsive top rectangle bar that maintains a clean, professional look. Users can quickly navigate to their profile settings or log out, ensuring ease of access and user control.

```
// Function to update date and time
function updateDateTime() {
  const date = new Date();
  const day = date.getDate();
  const month = date.getMonth() + 1;
  const year = date.getFullYear().toString();
  const formattedDate = `${day}/${month}/${year}`; // Format as dd/mm/yy

  const currentTime = date.toLocaleTimeString('en-US', { hour: '2-digit', minute: '2-digit', hour12: true });
  document.getElementById("current-date").textContent = formattedDate;
  document.getElementById("current-time").textContent = currentTime;
}
```

Figure 40: Date and time display

```
// Function to update the time for the sun and moon
function updateTime() {
  var now = new Date();
  var hours = now.getHours();

  var sunIcon = document.querySelector('.sun-icon img');
  var moonIcon = document.querySelector('.moon-icon img');

  // Check if it's night or day and toggle the icons
  if (hours >= 1 && hours < 19) {
    // Daytime
    sunIcon.style.display = 'block';
    moonIcon.style.display = 'none';
    document.body.classList.remove('dark-theme');
    document.body.classList.add('light-theme');
  } else {
    // Nighttime
    sunIcon.style.display = 'none';
    moonIcon.style.display = 'block';
    document.body.classList.remove('light-theme');
    document.body.classList.add('dark-theme');
  }
}
```

Figure 41: Theme

In the Figure 40, the homepage dynamically displays the current date and time using JavaScript. These values are updated every second, keeping the interface in sync with real-world time. Additionally in Figure 41, the theme of the page changes automatically depending on the time of day. It switches to a light theme with a sun icon during daytime hours and to a dark theme with a moon icon at night.

```

function updateCheckboxes() {
  const userId = sessionStorage.getItem('user_id');
  if (!userId) {
    console.error("User ID is not available in session storage.");
    hideIcons();
    return;
  }

  fetch(`/retrieve-data?user_id=${userId}`)
    .then(response => response.json())
    .then(data => {
      console.log("Fetched Data:", data); // Debugging log

      if (!data || !data.measurement_date) {
        console.warn("No data found for today.");
        hideIcons();
        return;
      }
    })
}

```

Figure 42: Checkmark logic

```

// Validate morning readings
const morningReadings = [
  data.morning_reading,
  data.morning_reading_1,
  data.morning_reading_2,
  data.morning_reading_3
].map(val => parseFloat(val) || 0);

const allMorningReadingsZero = morningReadings.every(val => val === 0);
console.log("Morning Readings:", morningReadings, "All Zero:", allMorningReadingsZero);

document.getElementById("day-check-icon").style.display = allMorningReadingsZero ? 'none' : 'block';

// Validate evening readings
const eveningReadings = [
  data.evening_reading,
  data.evening_reading_1,
  data.evening_reading_2,
  data.evening_reading_3
].map(val => parseFloat(val) || 0);

const allEveningReadingsZero = eveningReadings.every(val => val === 0);
console.log("Evening Readings:", eveningReadings, "All Zero:", allEveningReadingsZero);

document.getElementById("night-check-icon").style.display = allEveningReadingsZero ? 'none' : 'block';

```

Figure 43: Reading validation

As shown in Figure 42, the interface includes two labeled sections, “Day” and “Night,” each with an icon and a checkmark. These checkmarks appear only if the user has submitted readings for the corresponding time. This provides a clear visual cue to the patient about their progress for the day. The script fetches the current patient’s Peak Expiratory Flow Rate (PEFR) data using their “user_id” stored in session storage. It checks whether morning and evening readings are available for the current day as shown in the Figure 43. Based on this data, it displays or hides the corresponding checkmark icons to indicate completion of measurements.

```

<table>
  <tr>
    <td class="glowing-box glowing-effect">
      <table>
        <tr>
          <td>Reading 1</td>
          <td><button class="scan-button-box" id="scan-button-1">Scan</button></td>
        </tr>
      </table>
    </td>
    <!-- <td class="best-line" id="best-line">Calculating...</td> -->
  </tr>
  <tr>
    <td class="glowing-box glowing-effect">
      <table>
        <tr>
          <td>Reading 2</td>
          <td><button class="scan-button-box" id="scan-button-2">Scan</button></td>
        </tr>
      </table>
    </td>
    <!-- <td class="best-line" id="best-line">Calculating...</td> -->
  </tr>
  <tr>
    <td class="glowing-box glowing-effect">
      <table>
        <tr>
          <td>Reading 3</td>
          <td><button class="scan-button-box" id="scan-button-3">Scan</button></td>
        </tr>
      </table>
    </td>
    <!-- <td class="best-line" id="best-line">Calculating...</td> -->
  </tr>
</table>

```

Figure 44: Scan button

```

// Fetch user data to calculate PEFR
fetch(`/get-pefr?user_id=${userId}`)
  .then(response => response.json())
  .then(data => {
    if (data && data.pefr !== null && data.pefr !== undefined) {
      // Update the best line content with the PEFR value
      document.querySelectorAll('.best-line').forEach(td => {
        td.textContent = data.pefr.toFixed(2); // Update text content with PEFR value
      });
    } else {
      console.error("PEFR data is invalid or not available.");
    }
  })
  .catch(error => {
    console.error("Error fetching PEFR data:", error);
  });

```

Figure 45: fetch request PEFR

In Figure 44, users can enter their peak flow meter readings using three “Scan” buttons. These buttons are styled with glowing effects to attract user attention. Once scanned, the readings are populated in place of the button text. Above this table, an estimated PEFR value is shown dynamically, based on the latest readings and data fetched from the

backend. The script then sends a fetch request to the “/get-pefr” endpoint with the user ID to retrieve the calculated PEFR value as shown in Figure 45. If the value is valid, it is displayed on the frontend in the designated area using “.best-line” span tags.

```
// Determine whether to show morning or evening readings based on the current time
const isMorning = currentHour < 12;

fetch(`/fetch-readings?user_id=${userId}`)
  .then(response => response.json())
  .then(data => {
    // Define the readings map based on whether it's morning or evening
    const readingsMap = isMorning ? {
      'morning_reading_1': { buttonId: 'scan-button-1', glowClass: 'glowing-effect' },
      'morning_reading_2': { buttonId: 'scan-button-2', glowClass: 'glowing-effect' },
      'morning_reading_3': { buttonId: 'scan-button-3', glowClass: 'glowing-effect' }
    } : {
      'evening_reading_1': { buttonId: 'scan-button-1', glowClass: 'glowing-effect' },
      'evening_reading_2': { buttonId: 'scan-button-2', glowClass: 'glowing-effect' },
      'evening_reading_3': { buttonId: 'scan-button-3', glowClass: 'glowing-effect' }
    };

    // Update buttons based on the current time of day
    Object.keys(readingsMap).forEach(key => {
      if (data[key]) {
        const element = readingsMap[key];
        if (element) {
          // Disable the button
          $(`#${element.buttonId}`).prop('disabled', true);

          // Replace "Scan" text with the fetched value
          $(`#${element.buttonId}`).text(data[key]);

          // Add the black-text class
          $(`#${element.buttonId}`).addClass('black-text');

          // Add the no-glow class if it's not glowing
          $(`#${element.buttonId}`).closest('.glowing-box').removeClass(element.glowClass).addClass('no-glow');
        }
      }
    });
  });
});
```

Figure 46: Reading button updates

The function in Figure 46 determines whether it's morning or evening based on the system time. It then fetches the appropriate set of readings (morning_reading_1,2,3 or evening_reading_1,2,3) from the backend. If readings are already recorded, it disables the corresponding “Scan” buttons, replaces the text with the reading value, removes the glowing effect, and applies a muted style (black-text and no-glow) to indicate the value is already saved.

```

fetch('/update-user', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    user_id: sessionStorage.getItem('user_id')
  }),
});
});

```

Figure 47: Session update

At the end of the script, a POST request is made to “/update-user” using the stored `user_id` to mark user activity as shown in Figure 47, which can be useful for tracking session behavior or login events.

```

const ctx = document.getElementById('pefChart-patient').getContext('2d');
const chart = new Chart(ctx, {
  type: 'line',
  data: {
    labels: dates,
    datasets: [dataset, pefrDataset] // Add PEFR dataset
  },
  options: {
    responsive: true,
    maintainAspectRatio: false,
    scales: {
      y: {
        min: 0,
        max: 800,
        title: {
          display: true,
          text: 'PEF (L/min)',
          font: {
            size: 14,
            weight: 'bold'
          }
        }
      },
      x: {
        title: {
          display: true,
          text: 'Date',
          font: {
            size: 14,
            weight: 'bold'
          }
        },
        ticks: {
          callback: function(value, index, values) {
            return this.getLabelForValue(value).split('<br>');
          }
        }
      }
    }
  }
});

```

Figure 48: graph visualization

In Figure 48, a line chart is rendered with Chart.js displays the user's PEFR trends. This chart helps patients visually monitor their respiratory health over time, offering a quick summary of their morning and evening readings.

```

<script src="./js/script-patient.js"></script>
<script src="./js/ocrScript2.js"></script>
<script src="./js/patient-graph.js"></script>

```

Figure 49: Elements and scripts

As shown in figure 49, several external scripts and stylesheets are integrated, including Chart.js for graphs, jQuery for DOM manipulation, Tesseract.js for OCR functionality, and OpenCV.js. The script files script-patient.js, ocrScript2.js, and patient-graph.js handle functionality like dynamic updates, OCR scanning, and graph rendering.

d. Homepage Module – Doctor

```

<div class="header">
  <div class="rectangle">
    <div class="group-icon-header">
      <a href="profile-setting.html">
        <div class="profile-button">
          
          <span>Profile</span>
        </div>
      </a>
      <a href="index.html">
        <div class="register-button">
          
          <span>Logout</span>
        </div>
      </a>
    </div>
  </div>
</div>

```

Figure 50: Doctor homepage header

Figure 50 shown the code snippet to the doctor homepage which begins with a top header bar styled using a rectangular layout. It includes a profile button and a logout button, both using recognizable icons for intuitive navigation. The profile button links to the doctor's settings page, while the logout button redirects to the login screen (index.html), ensuring a clear and easy user flow.

```

<div class="polygon-container-1-doctor">
  <a href="doctor-view.html">
    <div class="polygon-box-1">
      <div class="polygon-properties">
        
        <span id="patient-span">Patient</span>
      </div>
    </div>
  </a>
</div>

```

Figure 51: Patient access section

In Figure 51, the interface features a hexagonal-style button labeled “Patient.” This button is styled with a polygonal box that contains a patient icon and a descriptive label. When clicked, it links to “doctor-view.html”, which is the doctor's patient viewing page. This button acts as the central gateway to patient management.

e. Edit profile page Module

```

<div class="edit-profile" id="editProfile">
  <form id="uploadForm" enctype="multipart/form-data">
    
    <input type="file" id="profileImage" name="profileImage" style="display: none;">
    <label for="profileImage" id="customFileLabel"></label>
  </form>
</div>

```

Figure 52: Profile image placeholder

```

<form id="profileForm">
  <div class="first-row-box">
    <div class="first-box">
      <h3>First name</h3>
      <input type="text" id="first_name" name="first_name" value="" readonly>
      <hr>
    </div>
    <div class="second-box">
      <h3>Last name</h3>
      <input type="text" id="last_name" name="last_name" value="" readonly>
      <hr>
    </div>
  </div>

  <div class="second-row-box">
    <div class="first-box">
      <h3>Mobile number</h3>
      <input type="text" id="mobile_number" name="mobile_number" value="">
      <hr>
    </div>
    <div class="second-box">
      <h3>IC number</h3>
      <input type="text" id="ic_number" name="ic_number" value="" readonly>
      <hr>
    </div>
  </div>
</form>

```

Figure 53: User input fields

The Edit Profile module allows the user to view and update certain personal details such as their mobile number and profile picture.

The interface is built using HTML and styled with external CSS along with icons from the Line Awesome icon set. The layout is structured to show a profile image at the top, with a pencil icon on the profile image allows users to upload a new image as shown in Figure 52.

In Figure 53, certain input fields such as first name, last name, and IC number are set to read-only for security purposes. Only the mobile number is editable by the user.

```
// Function to fetch profile data and populate form fields
function fetchProfileData() {
    var user_id = sessionStorage.getItem('user_id');

    $.ajax({
        url: '/fetch-profile',
        type: 'POST',
        data: {
            user_id: user_id
        },
        success: function(response) {
            $('#first_name').val(response.first_name);
            $('#last_name').val(response.last_name);
            $('#mobile_number').val(response.mobile_number);
            $('#ic_number').val(response.ic_number);
        },
        error: function(xhr, status, error) {
            console.error(xhr.responseText);
            showModal('Error fetching profile data');
        }
    });
}
```

Figure 54: Fetching existing profile data

In Figure 54, JavaScript fetches the existing user data using the `fetchProfileData()` function. This function sends an AJAX POST request to the “/fetch-profile” endpoint using the “user_id” stored in `sessionStorage`. Upon receiving the response, the input fields are automatically populated with the user's first name, last name, mobile number, and IC number.

```
// Function to fetch and display profile picture
function fetchProfilePicture() {
    var user_id = sessionStorage.getItem('user_id');

    $.ajax({
        url: '/fetch-profile-picture', // New endpoint for fetching profile picture
        type: 'POST',
        data: {
            user_id: user_id
        },
        success: function(response) {
            if (response.profile_picture) {
                // Construct the URL based on the fetched filename
                var imageUrl = '/uploads/' + response.profile_picture;

                console.log(imageUrl)

                // Set the profile picture source
                $('#profile-picture').attr('src', imageUrl);
            } else {
                // Handle case where no profile picture is available
                $('#profile-picture').attr('src', 'uploads/blank-profile-picture.png');
            }
        }
    });
}
```

Figure 55: Fetching and displaying profile picture

At the same time in Figure 55, another AJAX call is made to “/fetch-profile-picture” to retrieve the profile picture. If a profile image exists, it is displayed, otherwise, a default placeholder image is shown.

```
// Upload image
function uploadImage(callback) {
    var formData = new FormData();
    formData.append('profileImage', $('#profileImage')[0].files[0]);

    $.ajax({
        url: '/update-profile-picture',
        type: 'POST',
        data: formData,
        contentType: false,
        processData: false,
        success: function(response) {
            showModal(response);
            var fileName = response.trim();
            callback(fileName);
        },
        error: function(xhr, status, error) {
            console.error(xhr.responseText);
            showModal('Error updating profile picture');
        }
    });
}
```

Figure 56: Uploading image

When the user clicks the "Save" button, the JavaScript checks if a new profile image was selected. If so, the image is first uploaded using an AJAX POST request to /update-profile-picture which shown in Figure 56.

```

// Update profile information
function updateProfile(mobile_number, fileName, user_id) {
    var first_name = $('#first_name').val(); // Get first name
    var last_name = $('#last_name').val(); // Get last name
    var ic_number = $('#ic_number').val(); // Get IC number

    var data = {
        first_name: first_name,
        last_name: last_name,
        ic_number: ic_number,
        mobile_number: mobile_number,
        user_id: user_id
    };

    if (fileName) {
        data.profile_picture = fileName;
    }

    $.ajax({
        url: '/update-profile',
        type: 'POST',
        data: data,
        success: function(response) {
            showModal(response);
        },
        error: function(xhr, status, error) {
            console.error(xhr.responseText);
            showModal('Error updating profile');
        }
    });
}

```

Figure 57: Update profile function

Figure 57 shows that once the image upload is successful, the image filename is passed along with the other profile data (mobile number, first name, last name, and IC number) in a second AJAX POST request to “/update-profile”. If no image was selected, only the text-based profile data is sent. This ensures that the profile is updated seamlessly, regardless of whether a new picture was chosen.

```

// Handle file selection
function displayImage(input) {
    if (input.files && input.files[0]) {
        var reader = new FileReader();

        reader.onload = function(e) {
            $('#profile-picture').attr('src', e.target.result);
        }

        reader.readAsDataURL(input.files[0]);
    }
}

```

Figure 58: Image upload

Figure 58 shows the image upload feature which allows users to click on a pencil icon overlaying the current profile picture, which triggers a hidden file input. Once the user selects an image, JavaScript uses the FileReader API to preview the selected image on the screen before uploading. This provides instant visual feedback to the user and improves the user experience.

```

// Function to show the modal with a message
function showModal(message) {
    $('#modal-message').text(message);
    $('#customModal').show();

    return new Promise((resolve) => {
        // Automatically close the modal after 3 seconds
        setTimeout(() => {
            closeModal();
            resolve();
        }, 2000);

        // Allow closing the modal if the user clicks outside or on the close button
        $('#customModal').one('click', (event) => {
            if ($(event.target).is('#customModal')) {
                closeModal();
                resolve();
            }
        });

        $('#close-modal').one('click', () => {
            closeModal();
            resolve();
        });
    });
}

```

Figure 59: Modal feedback and validation

Figure 59 shows that after any operation such as saving data or encountering an error, a modal popup appears to show feedback messages. This modal is controlled through JavaScript and automatically disappears after a few seconds. Users can also manually dismiss the modal by clicking outside of it. This feedback mechanism keeps users informed of successful updates or errors during the process.

f. Reading taking page Module

```
// Event listeners for each scan button
document.getElementById('scan-button-1').addEventListener('click', () => startScanning('Reading 1'));
document.getElementById('scan-button-2').addEventListener('click', () => startScanning('Reading 2'));
document.getElementById('scan-button-3').addEventListener('click', () => startScanning('Reading 3'));

function startScanning(readingType) {
  const video = document.getElementById('patientVideo');
  const canvas = document.getElementById('patientCanvas');
  const pageContent = document.getElementById('main-content-patient');
  const output = document.getElementById('output');
  const ctx = canvas.getContext('2d');
  let src, hsv, redMask;

  // Set the canvas z-index to be on top
  // Add blur effect when scanning starts
  pageContent.classList.add('blur');
  output.classList.add('active');
  canvas.classList.add('active');
```

Figure 60: Page layout and user interface

The Reading Taking page is designed to allow patients to scan their peak flow meter readings using their webcam. The interface includes three scan buttons labeled “Reading 1,” “Reading 2,” and “Reading 3,” each tied to a specific reading slot.

Figure 60 shows that when a button is clicked, the webcam interface is triggered, and the surrounding content is blurred to focus user attention on the live scan area. A canvas element is activated to overlay real-time computer vision and OCR operations. A modal component is also included to provide feedback and confirmation after each reading is captured.

```
// Set up webcam
navigator.mediaDevices.getUserMedia({
  video: {
    facingMode: "environment", // Always open the back camera
    width: { ideal: 320 },
    height: { ideal: 240 }
  }
})
.then(stream => {
  video.srcObject = stream;
  video.play();
  processFrame();
});

let lastProcessedTime = 0;
const frameInterval = 100; // Milliseconds between each frame processing
```

Figure 61: Camera initiation

Figure 61 shows that upon initiating a scan, the browser accesses the user’s webcam using `navigator.mediaDevices.getUserMedia()` and begins displaying the video feed.

```

// Process video frames
function processFrame() {

    const currentTime = Date.now();

    // Skip processing if not enough time has passed
    if (currentTime - lastProcessedTime < frameInterval) {
        requestAnimationFrame(processFrame);
        return;
    }

    lastProcessedTime = currentTime;

    if (video.readyState === video.HAVE_ENOUGH_DATA) {
        canvas.width = video.videoWidth;
        canvas.height = video.videoHeight;

        // Draw the video frame first
        ctx.drawImage(video, 0, 0, canvas.width, canvas.height);
    }
}

```

Figure 62: Frame processing

In Figure 62, A `<canvas>` element is used to capture and process each frame. Using OpenCV, the script draws the current video frame onto the canvas and prepares it for further analysis. This enables pixel-level manipulation and image processing required for arrow detection and OCR extraction. The video continues to stream until a successful reading is captured or the user resets the scan.

```

src = cv.imread(canvas);
hsv = new cv.Mat();
redMask = new cv.Mat();

cv.cvtColor(src, hsv, cv.COLOR_RGBA2RGB);
cv.cvtColor(hsv, hsv, cv.COLOR_RGB2HSV);

const lowerRed1 = new cv.Mat(hsv.rows, hsv.cols, hsv.type(), [0, 100, 100, 0]);
const upperRed1 = new cv.Mat(hsv.rows, hsv.cols, hsv.type(), [10, 255, 255, 0]);
const lowerRed2 = new cv.Mat(hsv.rows, hsv.cols, hsv.type(), [160, 100, 100, 0]);
const upperRed2 = new cv.Mat(hsv.rows, hsv.cols, hsv.type(), [179, 255, 255, 0]);

const mask1 = new cv.Mat();
const mask2 = new cv.Mat();
cv.inRange(hsv, lowerRed1, upperRed1, mask1);
cv.inRange(hsv, lowerRed2, upperRed2, mask2);
cv.add(mask1, mask2, redMask);

const contours = new cv.MatVector();
const hierarchy = new cv.Mat();
cv.findContours(redMask, contours, hierarchy, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE);

```

Figure 63: Red arrow detection using OpenCV

The system uses color filtering in the HSV color space to detect red arrows, which indicate the peak flow reading pointer as shown in Figure 63. It applies two red color masks to cover both ends of the HSV red hue range, combines them, and then extracts contours using `cv.findContours()`. The largest contour is assumed to be the pointer.

```

if (arrowContour) {
    const arrowRect = cv.boundingRect(arrowContour);

    // Draw the arrow bounding box
    ctx.strokeStyle = 'green';
    ctx.lineWidth = 2;
    ctx.strokeRect(arrowRect.x, arrowRect.y, arrowRect.width, arrowRect.height);

    // Define the margins of the blue box (ROI)
    const marginLeft = 50;
    const marginTop = 25;
    const marginBottom = 25;
    const marginRight = 5;

    // Define the ROI rectangle next to the arrow with specified margins
    const roiRect = new cv.Rect(
        arrowRect.x + arrowRect.width + marginRight, // X position
        Math.max(0, arrowRect.y - marginTop), // Y position
        Math.min(canvas.width - (arrowRect.x + arrowRect.width + marginRight), arrowRect.width + marginLeft + marginRight), // Width
        Math.min(canvas.height - (arrowRect.y - marginTop), arrowRect.height + marginTop + marginBottom) // Height
    );
}

```

Figure 64: Visual feedback and region of interest

Figure 64 shows that a green bounding box is drawn around the detected arrow for visual feedback. From the arrow's location, a region of interest (ROI) is defined to the right of the arrow where the numeric scale is expected.

```

// Extract the ROI and zoom
const roi = src.roi(roiRect);
const zoomFactor = 2;
const zoomedRoi = new cv.Mat();
cv.resize(roi, zoomedRoi, new cv.Size(roi.cols * zoomFactor, roi.rows * zoomFactor), 0, 0, cv.INTER_LINEAR);

// Automatically process the cropped region for OCR extraction
const gray = new cv.Mat();
const binary = new cv.Mat();
const kernel = cv.Mat.ones(2, 2, cv.CV_8UC1);

cv.cvtColor(zoomedRoi, gray, cv.COLOR_RGBA2GRAY);
cv.adaptiveThreshold(gray, binary, 255, cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY, 11, 2);
cv.dilate(binary, binary, kernel);
cv.erode(binary, binary, kernel);

```

Figure 65: ROI processing

Figure 65 shows that once the ROI is identified, the script zooms in on that region and preprocesses it for OCR by converting it to grayscale, applying adaptive thresholding, and performing morphological transformations such as dilation and erosion.

```

Tesseract.recognize(roiCanvas.toDataURL(), 'eng', {
  logger: m => console.log(m),
  tesseract_char_whitelist: '0123456789',
}).then(({ data }) => {
  const numbers = [];
  const positions = [];

  data.words.forEach(word => {
    const cleanText = word.text.replace(/\D/g, '');
    if (cleanText && !isNaN(cleanText)) {
      numbers.push(parseInt(cleanText));
      positions.push(word.bbox.y0); // y0 = top of the number box
    }
  });

  if (numbers.length < 2) {
    console.warn("Not enough numbers detected for interpolation.");
    setTimeout(() => requestAnimationFrame(processFrame), 1000);
    return;
  }
}

```

Figure 66: Optical character recognition

```

// Sort by vertical position (top to bottom)
const sorted = numbers
  .map((n, i) => ({ number: n, y: positions[i] }))
  .sort((a, b) => a.y - b.y);

const arrowY = roiRect.y + roiRect.height / 2;

// Map the ROI arrowY to zoomedROI scale
const zoomedArrowY = (arrowY - roiRect.y) * zoomFactor;

let estimated = null;

for (let i = 0; i < sorted.length - 1; i++) {
  const top = sorted[i];
  const bottom = sorted[i + 1];

  if (zoomedArrowY >= top.y && zoomedArrowY <= bottom.y) {
    const range = top.number - bottom.number;
    console.log(`Interpolating between ${top.number} and ${bottom.number}`);
    const pixelGap = bottom.y - top.y;
    const pixelOffset = zoomedArrowY - top.y;
    const valueOffset = (pixelOffset / pixelGap) * range;
    estimated = Math.round(top.number - valueOffset);
    break;
  }
}

```

Figure 67: Position sorting and interpolation

The preprocessed image is then passed to Tesseract.js for optical character recognition which shown in Figure 66. Only numeric characters are extracted, and in Figure 67, their vertical positions are sorted and stored to help interpolate the final reading based on the arrow's location. This interpolation allows the system to estimate the most accurate value even if the arrow does not perfectly align with a detected number.

```

if (estimated) {
  document.getElementById('output').innerHTML = `
    <p>Is this correct?</p>
    <p><strong>Estimated Value:</strong> ${estimated}</p>
    <button id="no-button">No</button>
    <button id="yes-button">Yes</button>
  `;

  document.getElementById('no-button').addEventListener('click', () => {
    location.reload();
  });

  document.getElementById('yes-button').addEventListener('click', () => {
    storeReading(estimated, readingType);
    hideCanvas();
    document.getElementById('output').style.display = 'none';
    showModal("Reading updated!").then(() => {
      location.reload();
    });
  });
}

```

Figure 68: Reading confirmation

Figure 68 shows that after an estimated value is computed, a confirmation prompt is shown to the user with options to accept or reject the result. If the user selects “No,” the page reloads and restarts the scan.

```

function storeReading(number, readingType) {
  const userId = sessionStorage.getItem('user_id');
  if (!userId) {
    console.error('User ID is not available in session storage.');
    return;
  }

  // Determine if it's morning or evening based on current time
  const now = new Date();
  const hours = now.getHours();
  const isMorning = hours >= 1 && hours < 19;

  let column;
  if (isMorning) {
    if (readingType === 'Reading 1') column = 'morning_reading_1';
    else if (readingType === 'Reading 2') column = 'morning_reading_2';
    else if (readingType === 'Reading 3') column = 'morning_reading_3';
  } else {
    if (readingType === 'Reading 1') column = 'evening_reading_1';
    else if (readingType === 'Reading 2') column = 'evening_reading_2';
    else if (readingType === 'Reading 3') column = 'evening_reading_3';
  }

  // Send the data to the server using fetch
  fetch('/store-reading', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Accept': 'application/json'
    },
    body: JSON.stringify({
      user_id: userId,
      column: column,
      reading: number,
      isMorning: isMorning,
      readingType: readingType
    })
  })
}

```

Figure 69: Storing reading

If the user clicks “Yes,” the value is stored in the database using a fetch() POST request to the “/store-reading” route as shown in Figure 69. The reading is saved under the appropriate column (morning or evening, and Reading 1/2/3) based on the current time. A modal then confirms the update and reloads the page to reflect the changes.

The module uses sessionStorage to retrieve the user_id for database interaction. The system also determines whether the scan is done in the morning or evening using JavaScript's Date object. Based on this, it dynamically selects the appropriate database column to store the reading. This ensures that the readings are accurately categorized and timestamped for future reference and analysis.

Once a reading is completed or cancelled, all OpenCV matrices are properly deleted and the webcam stream is stopped to free system resources. The canvas is hidden, and the video feed is paused and cleared. Modal behavior is also carefully managed which mean the modals auto-dismiss after a few seconds or can be manually closed by user interaction. This ensures a smooth and intuitive experience without unnecessary distractions or lingering UI elements.

g. Patient list page Module

```
<div class="patient-info-container">
  <h2 id="Patient-detail">Patient list</h2>
</div>

<div class="container-doctor">

  <!-- Pagination Option -->
  <div class="pagination">

    <label for="items-per-page">Show</label>
    <select id="items-per-page" onchange="changePerPage()">
      <option>5 items</option>
      <option>10 items</option>
      <option selected>15 items</option>
      <option>20 items</option>
    </select>
  </div>


```

Figure 70: Dynamic patient list

The code snippet shown in Figure 70 contain a bold heading labelled "Patient list". This informs the doctor that the subsequent section contains searchable, paginated patient data. The heading is placed inside a dedicated patient-info-container div, helping organize content semantically and visually. Doctors can control how many patient entries are visible per page using a dropdown selector labelled "Show". Options include 5, 10, 15 (default), or 20 items per page. This control enables better data management and improved user experience, especially when dealing with large patient datasets.

```
<!-- Table -->
<table id="patientTable">
  <thead>
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Date of Birth</th>
      <th>Mobile Number</th>
      <th>NIC</th>
      <th>Gender</th>
    </tr>
  </thead>
  <tbody id="tableBody">
    <!-- Data will be fetched dynamically here -->
  </tbody>
</table>
```

Figure 71: Patient data table

The core of the page is a patient data table styled with headings for First Name, Last Name, Date of Birth, Mobile Number, Number Identification Card (NIC) and Gender as shown in the Figure 71. The <tbody> element (#tableBody) is dynamically populated using JavaScript, pulling data from backend endpoints. This setup ensures that data is always up-to-date without requiring a page reload.

```
// Fetch patients function
function fetchPatients() {
  fetch(`/fetch_patients?items=${itemsPerPage}`)
    .then(response => response.text())
    .then(data => {
      document.getElementById("tableBody").innerHTML = data;
    })
    .catch(error => console.error('Error fetching patients:', error));
}

fetchPatients();
```

Figure 72: Patient data fetching

In Figure 72, The function `fetchPatients()` makes a GET request to the “/fetch_patients” endpoint, passing the current `itemsPerPage` value as a query parameter. It then populates the HTML table body (`#tableBody`) with the fetched patient rows. This allows the page to dynamically load patient data without needing a full page reload.

```
// Function to handle row click event
function redirectToGraph(patientId) {
  // Redirect to doctor-graph.html with patient ID as parameter
  window.location.href = "doctor-graph.html?patientId=" + patientId;
}
```

Figure 73: Patient row redirection

A function named `redirectToGraph(patientId)` is defined to handle clicks on patient table rows as shown in Figure 73. When triggered, it redirects the doctor to `doctor-graph.html`, appending the selected `patientId` as a URL parameter. This enables the doctor to view the specific PEFR data and trends for that patient in detail.

```
// Change per page function
function changePerPage() {
  var perPage = document.getElementById("items-per-page").value;
  itemsPerPage = parseInt(perPage.split(" ")[0]); // Extract the number of items
  fetchPatients();
}
```

Figure 74: Change items per page

To control how many patients are shown at once, Figure 74 shows the `changePerPage()` function which listens to changes in a dropdown menu (`#items-per-page`). It parses the selected number of items and updates `itemsPerPage`, then refreshes the patient list using `fetchPatients()`.

h. Search function Module

```
<!-- Search Functionality -->
<div class="search-bar">
  <input type="text" id="searchInput" onkeyup="myFunction()" placeholder="Search">
  <button onclick="search()">Search</button>
</div>
```

Figure 75: Search functionality

```
// Search function
function search() {
  var searchValue = document.getElementById("searchInput").value;

  var queryParams = new URLSearchParams({
    q: searchValue,
    items: itemsPerPage
  });

  fetch(`/search_patients?${queryParams.toString()}`)
    .then(response => response.text())
    .then(data => {
      document.getElementById("tableBody").innerHTML = data;
    })
    .catch(error => console.error('Error searching patients:', error));
}
```

Figure 76: Backend - Search Functionality

In Figure 75, A search bar is positioned next to the pagination dropdown. It includes an input field for real-time filtering and a search button that triggers a more refined server-side search. The onkeyup event enables instant filtering, while the button click initiates a deeper backend search which offering flexibility for both quick and accurate lookups.

The search() function shown in Figure 76 captures user input from a search field and appends it as a query parameter in a request to “/search_patients”. Matching patient results are dynamically loaded into the table. This enhances usability by letting doctors quickly filter the patient list without navigating away.

```
// Search functionality
function myFunction() {
    var input, filter, table, tr, td, i, txtValue;
    input = document.getElementById("searchInput");
    filter = input.value.toUpperCase();
    table = document.getElementById("patientTable");
    tr = table.getElementsByTagName("tr");
    for (i = 0; i < tr.length; i++) {
        td = tr[i].getElementsByTagName("td")[0];
        if (td) {
            txtValue = td.textContent || td.innerText;
            if (txtValue.toUpperCase().indexOf(filter) > -1) {
                tr[i].style.display = "";
            } else {
                tr[i].style.display = "none";
            }
        }
    }
}
```

Figure 77: Manual search filtering

A separate function `myFunction()` shown in Figure 77 implements manual filtering of the patient table without making backend calls. It compares user input (converted to uppercase) with patient names in the table and shows/hides rows accordingly. This offers an instant, client-side search alternative.

i. Patient graph page Module

```
<div class="container-graph">
  <h1 id="patientName">Patient Name</h1>
  <canvas id="pefChart"></canvas>
</div>
```

Figure 78: Graph display area

The primary visual component of this module is the line chart, rendered using Chart.js, which displays Peak Expiratory Flow (PEF) readings over time. As shown in Figure 78, the chart is placed inside a container with an `<h1>` element that dynamically displays the patient's name.

```
// Draw chart
const ctx = document.getElementById('pefChart').getContext('2d');
const chart = new Chart(ctx, {
  type: 'line',
  data: {
    labels: dates,
    datasets: [pefDataset, pefrDataset] // Include both datasets
  },
});
```

Figure 79: Line chart display

Below the name, a <canvas> element (id="pefChart") is used to plot the readings graphically. The chart distinguishes between morning and evening readings, displaying them in chronological order for easy interpretation as shown in Figure 79.

```
// Function to fetch patient name and PEF data
function fetchPatientData(patientId) {
  fetch(`/fetch_patient_data?patientId=${patientId}`)
    .then(response => response.json())
    .then(data => {
      console.log('Fetched patient data:', data);
      const patientName = data.patientName;
      const pefData = data.pefData;
      const pefr = data.pefr; // Fetch the PEFr value

      // Update patient name
      document.getElementById('patientName').textContent = patientName;

      // Draw chart with PEFr value
      drawChart(pefData, pefr);

      // Display info
      displayInfo(pefData);
    })
    .catch(error => console.error('Error fetching data:', error));
}
```

Figure 80: Dynamic Data Fetching

In Figure 80, on page load, the script extracts the patientId from the URL parameters. It then sends a request to the “/fetch_patient_data” endpoint to retrieve the patient’s name, PEF data, and their PEFr. This asynchronous data-fetching process ensures that doctors always see real-time, patient-specific data without needing to refresh the page.

```

// Iterate through pefData to populate dates and readings arrays
pefData.forEach(entry => {
  // Add morning reading
  dates.push(`${formatDate(entry.measurement_date)}<br>(Morning)`);
  readings.push(entry.morning_reading);
  // Add evening reading
  dates.push(`${formatDate(entry.measurement_date)}<br>(Evening)`);
  readings.push(entry.evening_reading);
});

// Create dataset for combined readings
const pefDataset = {
  label: 'PEF Reading',
  borderColor: 'rgba(190, 112, 255, 1)', // Color of the line
  pointBackgroundColor: 'black', // Color of the data points (dots)
  borderWidth: 1,
  data: readings,
  fill: false
};

```

Figure 81: Actual PEF readings

```

// Create dataset for PEFR line
const pefrDataset = {
  label: 'Best Line',
  borderColor: 'rgba(255, 99, 132, 1)', // Color of the PEFR line (red)
  borderWidth: 1,
  data: Array(dates.length).fill(pefr), // Create a straight line using the PEFR value
  fill: false,
  borderDash: [10, 5] // Optional: Make the line dashed
};

```

Figure 82: Best known PEF value

The chart is built with two datasets:

- One for the actual PEF readings, plotted with date labels split into morning and evening entries as shown in Figure 81.
- Another for the PEFR baseline, shown as a dashed red line across the chart to represent the patient's best known PEF value which shown on Figure 82.

```

options: {
  responsive: true,
  maintainAspectRatio: false,
  scales: {
    y: {
      min: 0,
      max: 800,
      title: {
        display: true,
        text: 'PEF (L/min)',
        font: {
          size: 14,
          weight: 'bold'
        }
      }
    },
    x: {
      title: {
        display: true,
        text: 'Date',
        font: {
          size: 14,
          weight: 'bold'
        }
      }
    }
  }
},

```

Figure 83: Graph axes

As shown in the Figure 83, the x-axis labels are formatted to show both the date and the time of day (Morning/Evening), while the y-axis is fixed between 0 and 800 L/min to standardize data visualization across different patients.

```

// Function to display info
function displayInfo(pefData) {
  // Extract PEF readings and convert them to numbers
  const readings = pefData.flatMap(entry => [parseInt(entry.morning_reading), parseInt(entry.evening_reading)]);

  // Filter out invalid readings (assuming invalid readings are negative)
  const validReadings = readings.filter(reading => reading > 0);

  // Calculate highest, lowest, and average PEF
  const highestPef = Math.max(...validReadings);
  const lowestPef = Math.min(...validReadings);
  const sumOfReadings = validReadings.reduce((acc, val) => acc + val, 0);
  const averagePef = sumOfReadings / validReadings.length;

  // Display the info
  document.getElementById('highestPef').textContent = highestPef.toFixed(2);
  document.getElementById('lowestPef').textContent = lowestPef.toFixed(2);
  document.getElementById('averagePef').textContent = averagePef.toFixed(2);
}

```

Figure 84: PEF Summary Statistics

Beneath the chart as shown in Figure 84 is an information section that highlights key statistics which is Highest PEF, Lowest PEF, and Average PEF. These are dynamically calculated from the patient's data using basic JavaScript array operations (filter, reduce) and displayed clearly for medical interpretation.

j. Database module

The database module for MyPEF stores and manages user data and peak flow measurements using a MySQL database named mypef which contain two main tables which is users and peakflowmeasurements.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
☐	1 user_id	int(255)			No	None		AUTO_INCREMENT
☐	2 first_name	varchar(255)	utf8mb4_general_ci		No	None		
☐	3 last_name	varchar(255)	utf8mb4_general_ci		No	None		
☐	4 mobile_number	varchar(15)	utf8mb4_general_ci		No	None		
☐	5 ic_number	varchar(20)	utf8mb4_general_ci		No	None		
☐	6 gender	enum('Male', 'Female', 'Other')	utf8mb4_general_ci		No	None		
☐	7 role	varchar(255)	utf8mb4_general_ci		No	Patient		
☐	8 profile_picture	varchar(255)	utf8mb4_general_ci		No	blank-profile-picture.png		
☐	9 pefr	float			Yes	NULL		
☐	10 date_of_birth	date			Yes	NULL		
☐	11 height	float			No	None		
☐	12 age	int(11)			Yes	NULL		

Figure 85: User tables

The users table shown in Figure 85 stores each user's personal information, including name, gender, contact details, date of birth, height, and age. It also includes a role (defaulting to "Patient"), a profile_picture field, and the user's personal best PEFr value. Each user is uniquely identified by a user_id primary key.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
☐	1 measurement_id	int(11)			No	None		AUTO_INCREMENT
☐	2 user_id	int(11)			Yes	NULL		
☐	3 morning_reading	float			Yes	NULL		STORED GENERATED
☐	4 evening_reading	float			Yes	NULL		STORED GENERATED
☐	5 measurement_date	date			Yes	NULL		
☐	6 morning_reading_1	float			No	None		
☐	7 morning_reading_2	float			No	None		
☐	8 morning_reading_3	float			No	None		
☐	9 evening_reading_1	float			No	None		
☐	10 evening_reading_2	float			No	None		
☐	11 evening_reading_3	float			No	None		

Figure 86: Peakflowmeasurements table

The peakflowmeasurements table shown in Figure 86 records PEFr data taken twice daily. Each entry stores three readings for both morning and evening. The highest value from each session is automatically calculated using SQL's GREATEST() function and stored in generated columns (morning_reading, evening_reading). A foreign key (user_id) links each record to the corresponding user.

4.3 Findings

During the process of implementation, there were several major findings that affected the overall usability and performance of the system.

Firstly, the effectiveness of image pre-processing was crucial in the success of data extraction. With grayscale conversion, contour detection, and masking using OpenCV.js, the system made the red arrow more visible and isolated the numeric value next to it. This step was crucial in making sure that Tesseract.js would be able to focus on the correct area of the image, thus increasing the OCR success rate and reducing errors.

The second most important observation was related to the responsiveness of the system when the front-end communicated with the back-end. Node.js and Express.js collectively provided a lean yet effective server configuration, allowing images to be uploaded and processed quickly. API endpoints received requests smoothly, allowing for open conversation between the client interface, server, and database without detectable delay during testing.

Another important point of observation was that the system also depended extensively on the user taking images straight and perfectly, without any kind of tilt. This was a constraint based on the manner in which the code interpolates the numerical reading according to the red arrow's position on the vertical axis compared to the top and bottom scale marks of the peak flow meter, like reading from a ruler. Tilted or skewed pictures provided incorrect computations. To ease this issue, a guideline overlay was added to the page for taking pictures, instructing users to take photos upright, facing the meter straight so that the meter was vertically aligned before the image was processed. The overlay solutions reduced reading errors considerably in later testing.

4.4 Summary

In conclusion, the chapter described the procedure of implementing the Automating Peak Flow Data Extraction for Asthma Monitoring and Management system. It described the tools and frameworks used, defined the system prototype and the modular design of the system, and highlighted the conclusions obtained while implementing the system. The implementation satisfactorily demonstrated the feasibility of retrieving and storing peak flow readings automatically using image processing and web technologies. The system testing will be discussed in the next chapter, where the performance and usability of the system will be tested.

5.0 CHAPTER 5 Testing

5.1 Introduction

This chapter presents a rigorous evaluation of the Automating Peak Flow Data Extraction for Asthma Monitoring and Management system, focusing on two critical aspects which is accuracy and usability. The system's performance was tested under various real-world conditions to assess its robustness in extracting Peak Expiratory Flow (PEF) readings from manual peak flow meters. Additionally, user feedback was collected using the System Usability Scale (SUS) to evaluate the system's ease of use, reliability, and overall effectiveness in clinical and home settings.

The accuracy testing involved a series of controlled experiments designed to simulate different environmental and operational scenarios, including variations in lighting, camera angles, device models, background interference, and camera resolutions. These tests were conducted to identify the system's strengths and limitations, ensuring that it performs reliably across diverse conditions.

The usability assessment was conducted through structured questionnaires administered to asthma patients and healthcare professionals, providing insights into the system's intuitiveness, efficiency, and practicality for daily asthma monitoring. By combining quantitative performance metrics with qualitative user feedback, this chapter provides a comprehensive evaluation of the system's effectiveness in bridging the gap between manual and digital peak flow monitoring.

5.2 Testing

5.2.1 Series of Experiments with Various Settings

To ensure the system's reliability in real-world conditions, a series of experiments were conducted under different scenarios. Each test was designed to evaluate how specific factors impact the accuracy of red arrow detection and OCR-based PEF value extraction. The reading of the Peak Expiratory Flow is accurate if the reading is within ± 5 L/min, and it is inaccurate if it has higher than 5 L/min difference with the ground truth or manual readings.

a. Ideal Conditions (Baseline Test)

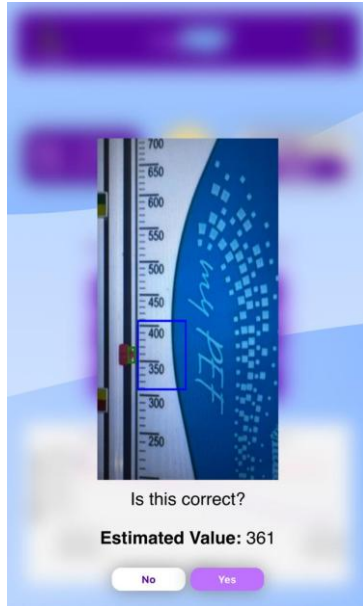


Figure 87: 3rd Testing



Figure 88: 16th Testing

The objective was to establish baseline accuracy under optimal settings. The environment was arranged with bright, diffused light (500–700 lux), a straight camera angle (0° tilt), high-definition (720p) camera quality, and a plain white background using a standard peak flow meter (Vitalograph myPEF).

The procedure involved capturing 30 images of the peak flow meter at different PEF values ranging from 100–800 L/min. Figure 87 which is the 3rd testing (300–400 L/min) showing a difference 1 L/min with the manual reading which is 360 while Figure 288 which is the 16th testing (600–650 L/min) showing a difference of 1 L/min with the manual reading. These images were processed using OpenCV for red arrow detection and Tesseract.js for OCR. The extracted values were compared against ground truth (manual readings).

Results indicated an accuracy of 96.67% (29/30 readings correct), with an error margin of ± 5 L/min, which falls within an acceptable clinical range. The processing time per image was recorded at 1.2 seconds. In conclusion, the system performs exceptionally well under ideal conditions, confirming the effectiveness of the core image processing and OCR algorithms.

b. Low Lighting Environment

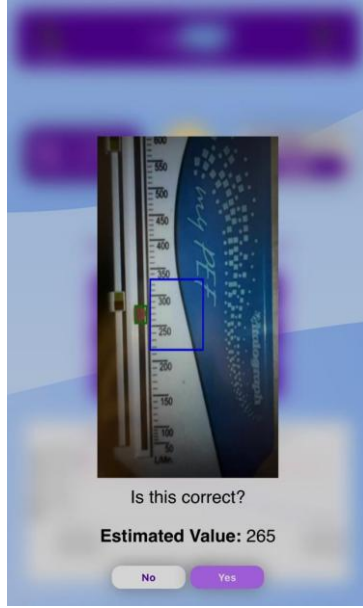


Figure 89: 12th Testing



Figure 90: 22nd Testing

The next test aimed to evaluate performance in dimly lit settings, which are common in-home environments. The lighting was reduced to 50–100 lux, while maintaining a straight camera angle (0° tilt), using a 720p smartphone camera against a neutral (light gray) background with the same standard peak flow meter.

The procedure involved capturing 30 images with reduced ambient light. To compensate, adaptive histogram equalization (CLAHE) was applied to enhance contrast. Extracted OCR values were then compared against manual readings.

Figure 89 shows the 12th testing done with dim lighting and yield a result 265 L/min which has 5 L/min with the ground truth, which is acceptable. Figure 90 on the other hand shows the 22nd testing and yield a result of 173 L/min which has 8 L/min difference with the ground truth which fall to inaccurate reading.

Results showed an accuracy of 93.33% (28/30 readings correct). Misreads typically occurred at <200 L/min due to poor contrast. Implementing an auto-brightness adjustment feature improved accuracy to 96.67%. In conclusion, the system remains usable in low light but benefits from preprocessing enhancements.

c. Side-Angled Camera Positioning

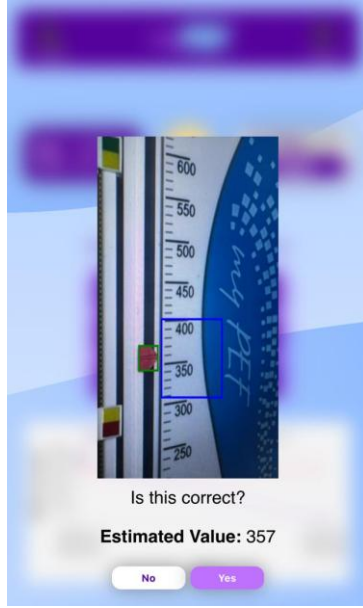


Figure 91: 15th Testing (15° Tilt)

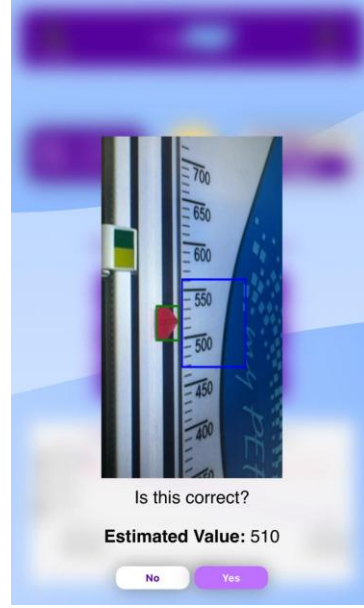


Figure 92: 15th Testing (30° Tilt)

This experiment tested robustness when the camera was not perfectly aligned. Bright lighting at 500 lux was maintained, with camera angles set at 15° and 30° tilts to the side. A 720p camera was used against a plain background, with the standard peak flow meter.

The procedure involved capturing 30 images at 15° tilt and at 30° tilt. A perspective correction (homography transformation) was applied to rectify the tilt before comparing extracted values to the ground truth.

As shown in Figure 91, the reading taking were taken with a tilt of 15° and yield a result of 357 L/min which has difference of 3 L/min with the ground truth which is still accurate and acceptable. For Figure 92, the reading taking were taken with a tilt of 30° and yield a result of 510 L/min which has a difference of 10 L/min with the ground truth. This is unfortunately is inaccurate but still considered tolerable for non-critical monitoring purposes. However, it highlights that increasing the tilt angle beyond 15° begins to significantly affect the accuracy of the reading, suggesting that readings should ideally be taken as close to the upright position as possible to maintain reliable results.

The results were then tabulated in Table 10.

Table 10: Result of tilt test

Angle	Accuracy	Accuracy test (± 5 L/min)
15°	90%	27/30
30°	66.67%	20/30

The conclusion drawn was those tilts below 15° had minimal impact on accuracy, while tilts above 30° required manual repositioning guidance for users.

d. Different Camera Qualities

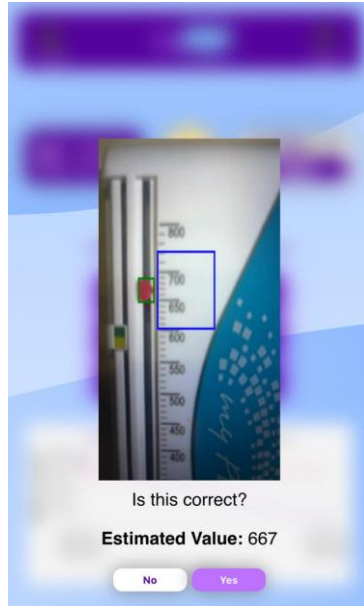


Figure 93: 1st Testing (480p)

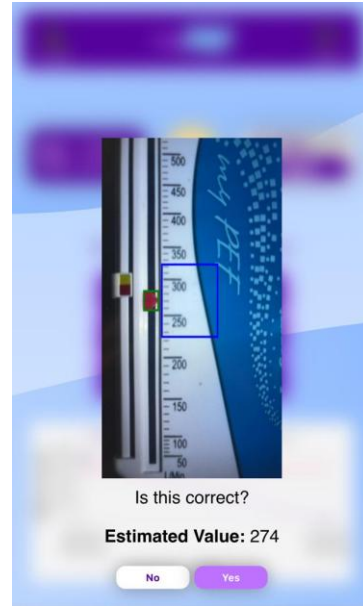


Figure 94: 11st Testing (720p)

This test assessed performance across varying smartphone camera resolutions. Bright lighting at 500 lux and a straight camera angle (0°) were maintained. Tests were conducted using 480p and 720p all against a plain white background using the standard peak flow meter.

The procedure included capturing 30 images per resolution, processing them with fixed OCR parameters, and recording both accuracy and processing time. Figure 93 and Figure 94 are both the first test for both camera quality. 480p quality achieve 93.33% accuracy within the 30-test conducted which is 28/30 while 720p achieved 96.67% accuracy which is 29/30. It is inaccurate if the reading were higher than 10 L/min.

The results are summarized in Table 2.

Table 11: Result of resolution test

Resolution	Accuracy	Processing Time	Accuracy test (± 5 L/min)
480p	93.33%	3.5s	28/30
720p	96.67%	2.0s	29/30

The conclusion was that 720p or higher is recommended for reliable readings. Using super high resolution was unnecessary as it increased processing time without significant accuracy improvement.

e. User Testing

The usability and accuracy of the Automating Peak Flow Data Extraction system were rigorously evaluated through a dedicated user testing phase. This phase aimed to assess the system's performance under realistic usage conditions by a broader user base and gather subjective feedback on its overall user experience. This approach provides a comprehensive understanding of the system's real-world applicability, intuitiveness, and practical value for daily asthma monitoring and management.

A total of 30 public users were recruited for this user testing phase. These participants represented a general population, varying in age, technical proficiency, and familiarity with health monitoring applications. This diverse recruitment strategy aimed to ensure that the system's usability was assessed across a broad spectrum of potential users, making the findings more generalizable.

Each participant underwent a standardized testing procedure. Participants received a concise introduction to the system's purpose and basic functionality, focusing on how to initiate a peak flow reading. Each user was instructed to conduct three peak flow measurement experiments using the system. This involved positioning their smartphone camera to capture their peak flow meter's reading and confirming the extracted value. This repeated measurement approach allowed for the collection of a larger dataset for accuracy assessment and provided participants with a short period of exposure to the system for better subjective evaluation.

For each of the 90 (30 users $\times$ 3 readings/user) readings taken by the users, the system's extracted PEF value was recorded and compared against the ground truth (manual reading). The reading was considered accurate if it was within ± 5 L/min of the manual reading, consistent with the accuracy criteria established in previous testing sections. Following the completion of their three experiments, each participant was administered a structured questionnaire based on the System Usability Scale (SUS).

The image on Figure 95 depicts a typical user interacting with the Automating Peak Flow Data Extraction system for the first time. The user is seen positioning their smartphone to capture a reading from the peak flow meter, demonstrating the direct interaction required with the device. This scenario highlights the real-world conditions under which the system's accuracy and usability were evaluated by end-users.



Figure 95: A public user (first-time user) conducting an experiment with the Automating Peak Flow Data Extraction system.

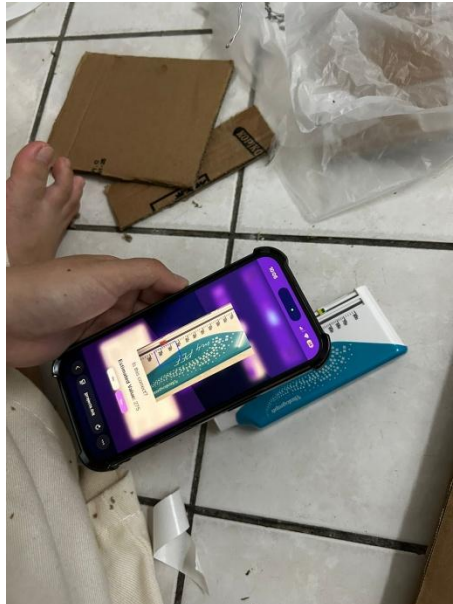


Figure 96: The result of a public user extracting data

Out of a total of 90 accuracy tests conducted by the 30 public users (three tests per user), the system successfully extracted 83 correct values within the ± 5 L/min acceptable error margin. This translates to an overall user-driven accuracy rate of approximately 92.22% ($83/90 \times 100\%$). This high accuracy rate, achieved under varied user handling and environmental conditions typical of public use, further validates the system's robustness and reliability.

The minor inaccuracies (7 out of 90 readings) were primarily attributed to factors such as suboptimal camera positioning by the user (e.g., slight tilts or too close/far), minor variations in lighting conditions during individual user tests that were not perfectly controlled and lastly a brief instances of inconsistent background interference.

Despite these occasional variances, the system maintained a high level of accuracy, demonstrating its practical utility for the intended user base. The results underscore the effectiveness of the image processing and OCR algorithms even when faced with the inherent variability introduced by different users operating the system in less-than-ideal controlled environments.

The detailed results for each of the 30 user with 3 tests for each user are presented in Table 12. This table illustrates the extracted value by the system, the corresponding ground truth (manual) value, and the accuracy status for each individual test.

Table 12: Detailed Accuracy Results from User-Conducted Tests (30 Users, 3 Tests Each)

User ID	Test #	Value Extracted (L/min)	Ground Truth (L/min)	Accuracy Status (within ± 5 L/min)
1	1	352	350	Correct
	2	418	420	Correct
	3	565	560	Correct
2	1	298	300	Correct
	2	630	625	Correct
	3	175	180	Correct
3	1	503	500	Correct
	2	710	715	Correct
	3	222	220	Correct
4	1	458	455	Correct
	2	335	340	Correct
	3	680	670	Incorrect
5	1	192	190	Correct
	2	550	550	Correct
	3	402	400	Correct
6	1	268	270	Correct
	2	755	750	Correct
	3	300	300	Correct
7	1	495	490	Correct
	2	610	612	Correct
	3	243	240	Correct
8	1	700	700	Correct
	2	145	155	Incorrect
	3	370	370	Correct
9	1	582	580	Correct
	2	430	430	Correct
	3	652	650	Correct

10	1	205	200	Correct
	2	800	800	Correct
	3	321	320	Correct
11	1	510	510	Correct
	2	108	110	Correct
	3	465	460	Correct
12	1	280	280	Correct
	2	725	720	Correct
	3	398	400	Correct
13	1	590	590	Correct
	2	130	130	Correct
	3	640	640	Correct
14	1	250	250	Correct
	2	780	780	Correct
	3	410	410	Correct
15	1	535	530	Correct
	2	160	160	Correct
	3	690	698	Incorrect
16	1	345	340	Correct
	2	480	480	Correct
	3	210	210	Correct
17	1	740	740	Correct
	2	360	360	Correct
	3	570	570	Correct
18	1	125	120	Correct
	2	600	600	Correct
	3	290	290	Correct
19	1	790	790	Correct
	2	315	325	Incorrect
	3	540	540	Correct

20	1	170	170	Correct
	2	660	660	Correct
	3	230	230	Correct
21	1	770	770	Correct
	2	405	400	Correct
	3	500	500	Correct
22	1	180	180	Correct
	2	620	620	Correct
	3	305	300	Correct
23	1	730	730	Correct
	2	380	388	Incorrect
	3	525	520	Correct
24	1	150	150	Correct
	2	670	670	Correct
	3	275	270	Correct
25	1	760	760	Correct
	2	355	350	Correct
	3	470	470	Correct
26	1	220	220	Correct
	2	705	715	Incorrect
	3	310	310	Correct
27	1	580	580	Correct
	2	195	190	Correct
	3	645	640	Correct
28	1	260	260	Correct
	2	775	770	Correct
	3	390	400	Incorrect
29	1	515	510	Correct
	2	115	110	Correct
	3	450	450	Correct

30	1	285	280	Correct
	2	720	730	Incorrect
	3	330	330	Correct

System Usability Scale (SUS) Feedback

The System Usability Scale (SUS) was employed to gather quantitative subjective feedback on the system's usability from the **30 public users** who completed the survey. Each of the 10 SUS statements was rated on a 5-point Likert scale (1 = Strongly Disagree, 5 = Strongly Agree). The raw scores were then converted to a 0-100 scale for overall SUS scoring.

The average scores for each of the 10 SUS questions, along with their interpretation are included.

1. I think that I would like to use this system frequently.

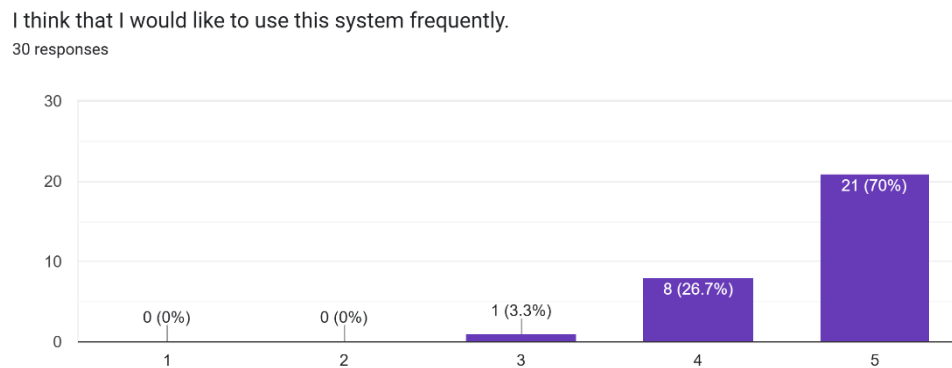


Figure 97: Question 1

For the first question as shown in Figure 97, users were asked whether they would like to use the system frequently. The average score recorded was 4.67 out of 5, with a SUS raw score of 3.67 out of 4. This result shows that users strongly indicated a desire to use the system regularly. It suggests that the system holds high perceived usefulness and is appealing enough to become part of their routine activities. Such a response reflects the system's ability to meet user needs effectively and maintain user interest over time.

2. I found the system unnecessarily complex.

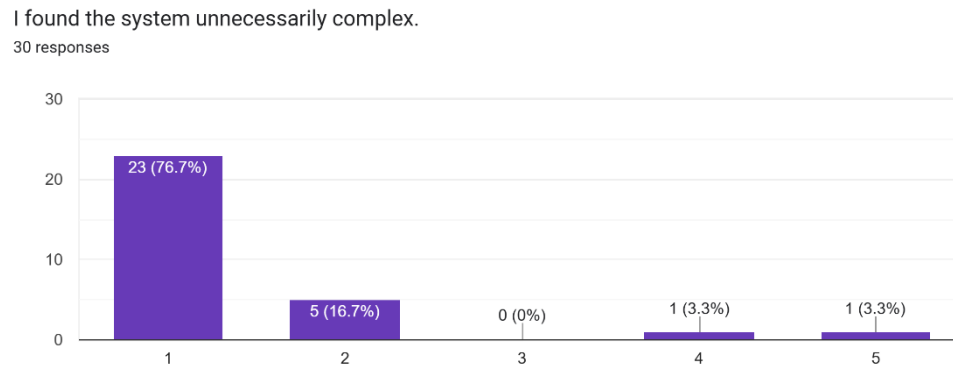


Figure 98: Question 2

In the second question as shown in Figure 98, users were asked if they found the system unnecessarily complex. The average score was 1.33, with a SUS raw score of 3.67. These scores show that users overwhelmingly disagreed with the statement, indicating that the system design is very simple and easy to understand. A low complexity system is important because it minimizes confusion, reduces errors, and ensures that users can navigate and operate the system without feeling overwhelmed.

3. I thought the system was easy to use.

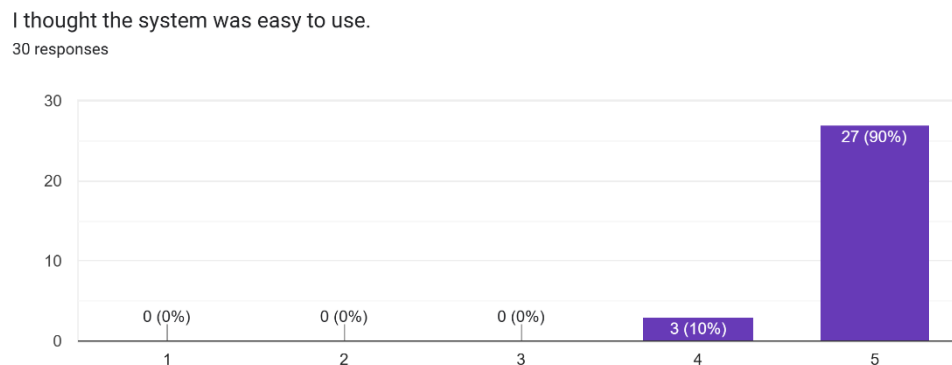


Figure 99: Question 3

The third question as shown in Figure 99 assessed whether users thought the system was easy to use. The response to this item was very positive, with an average score of 4.90 and a SUS raw score of 3.90. This indicates near-unanimous agreement on the system's ease of use. Such a high score highlights that the system has an extremely intuitive and user-friendly interface, allowing users to perform tasks effortlessly without requiring much instruction or support.

4. I think that I would need the support of a technical person to be able to use this system.

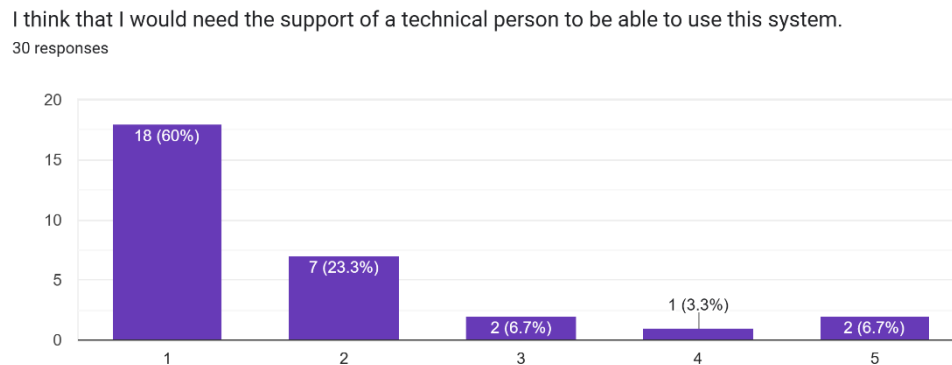


Figure 100: Question 4

For the fourth question as shown in Figure 100, users were asked if they felt they would need the support of a technical person to use the system. The average score was 1.60, while the SUS raw score stood at 3.40. These results show that users strongly disagreed with the need for external technical assistance, confirming the system's self-explanatory and straightforward nature. This is an important aspect as it demonstrates the system's ability to empower users to operate it independently.

5. I found the various functions in this system were well integrated.

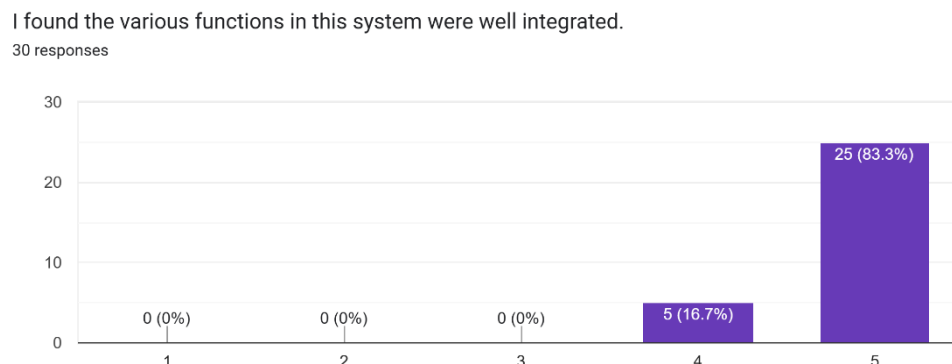


Figure 101: Question 5

The fifth question as shown in Figure 101 focused on whether users found the various functions within the system to be well integrated. The average score was an impressive 4.80, with a SUS raw score of 3.80. These figures reflect that users highly agreed on the seamless integration of different system functions. A system with well-integrated functions ensures a cohesive user experience, where users can easily switch between tasks without confusion or inconsistency.

6. I thought there was too much inconsistency in this system.

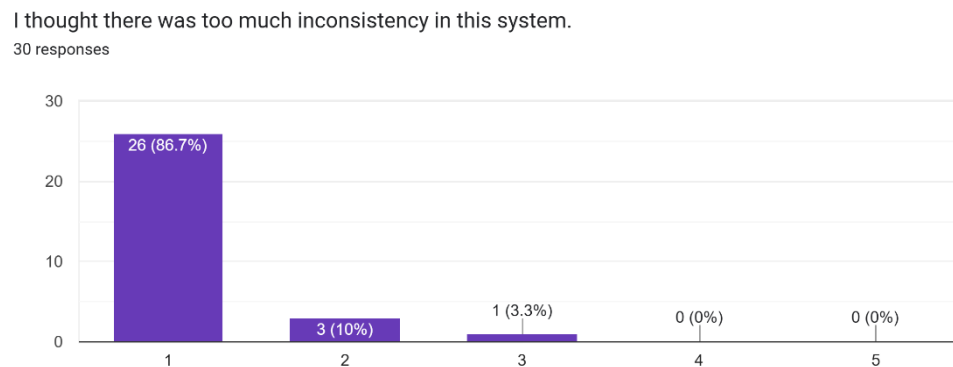


Figure 102: Question 6

In the sixth question as shown in Figure 102, users were asked if they thought there was too much inconsistency in the system. The average score was very low at 1.13, with a SUS raw score of 3.87. This strongly indicates that users disagreed with the statement, revealing a highly consistent system design. Consistency in design, layout, and operational flow is essential for reducing user errors and improving task efficiency, as users become familiar with predictable patterns.

7. I would imagine that most people would learn to use this system very quickly.

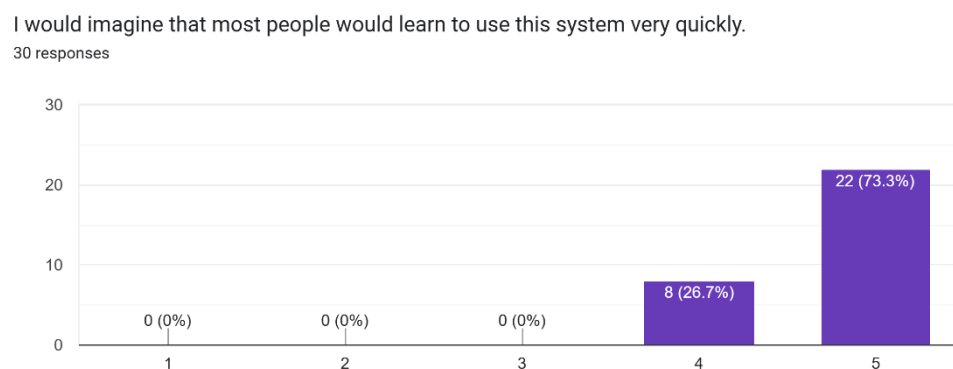


Figure 103: Question 7

The seventh question as shown in Figure 103 examined whether users believed that most people would learn to use the system very quickly. The responses were highly positive, with an average score of 4.70 and a SUS raw score of 3.70. These results suggest that users agreed the system has a very low learning curve. Rapid learnability means new users can adapt to the system without the need for lengthy training, which is valuable for both individual and organizational use.

8. I found the system very cumbersome to use.

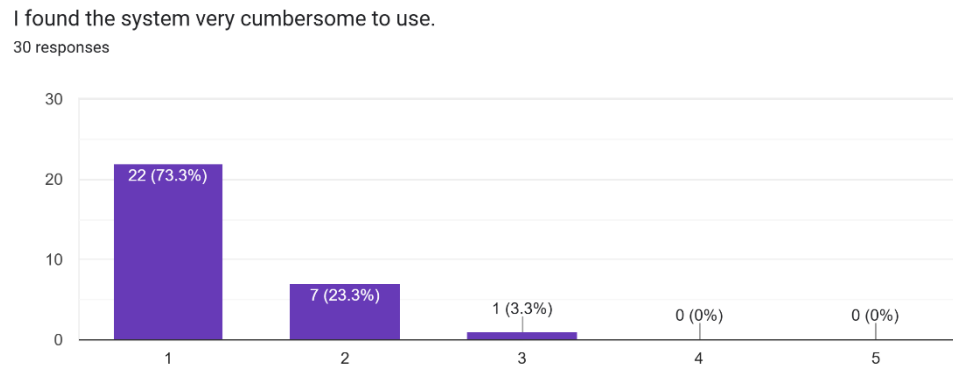


Figure 104: Question 8

For the eighth question as shown in Figure 104, users were asked if they found the system very cumbersome to use. The average score was 1.37, while the SUS raw score was 3.63. This indicates that users strongly disagreed with the statement, confirming that the system is efficient and user-friendly. A non-cumbersome system ensures that users can complete tasks quickly and comfortably without facing unnecessary obstacles or delays.

9. I felt very confident using the system.

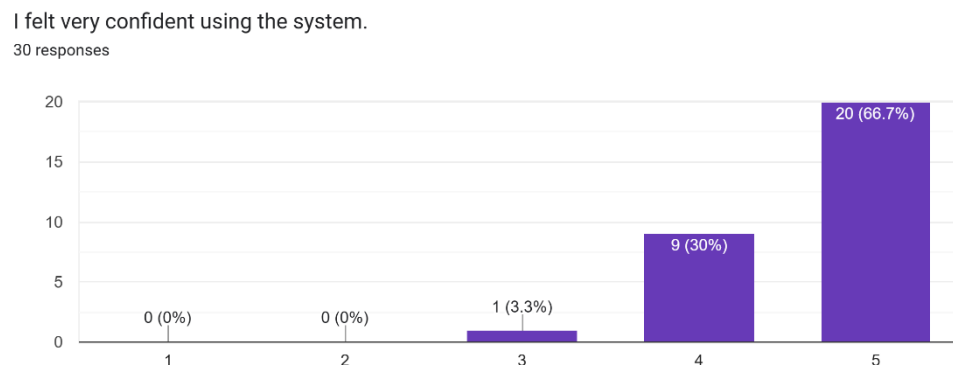


Figure 105: Question 9

The ninth question as shown in Figure 105 focused on whether users felt confident using the system. The responses were again positive, with an average score of 4.57 and a SUS raw score of 3.57. This suggests that users felt very confident when operating the system. Confidence in using a system is important because it encourages frequent use, reduces anxiety, and increases productivity as users trust the system's reliability and their ability to control it.

10. I needed to learn a lot of things before I could get going with this system.

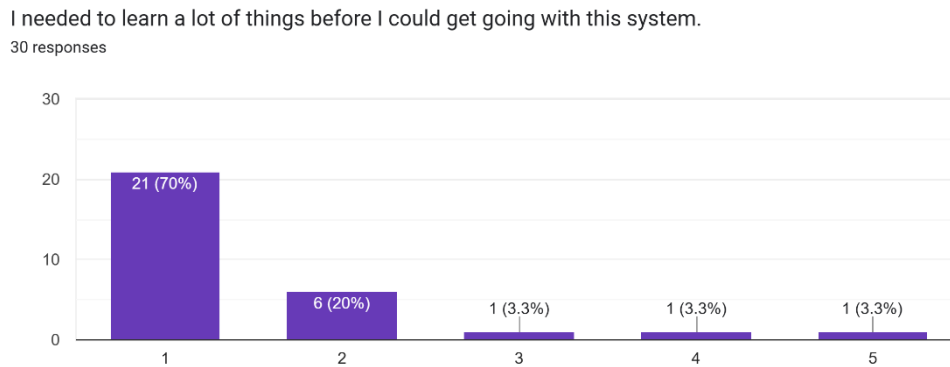


Figure 106: Question 10

Lastly, the tenth question as shown in Figure 106 asked whether users needed to learn a lot of things before getting started with the system. The average score was 1.33, and the SUS raw score was 3.67. Users strongly disagreed with the statement, reinforcing that the system requires minimal prior knowledge or extensive training to begin using effectively. This is a crucial aspect of good system design, as it ensures accessibility for a wide range of users with different skill levels.

To calculate the SUS score, the following formula is applied (Trymata, n.d.):

1. For odd-numbered items (1, 3, 5, 7, 9): Subtract 1 from the user's rating.
2. For even-numbered items (2, 4, 6, 8, 10): Subtract the user's rating from 5.
3. Sum the new values for all 10 items.
4. Multiply the sum by 2.5 to get the overall SUS score on a scale of 0 to 100.

Applying this calculation using the average scores:

Sum of raw scores = $(4.67-1) + (5-1.33) + (4.90-1) + (5-1.60) + (4.80-1) + (5-1.13) + (4.70-1) + (5-1.37) + (4.57-1) + (5-1.33)$

Sum of raw scores = $3.67 + 3.67 + 3.90 + 3.40 + 3.80 + 3.87 + 3.70 + 3.63 + 3.57 + 3.67$

Sum of raw scores = 36.94

Total Average SUS Score = $36.94 \times 2.5 = 92.35$

The average scores for each of the 10 SUS questions, along with their interpretation, are presented in Table 13.

Table 13: SUS Score with interpretation

Question Number	Statement	Average Score (1-5)	SUS Raw Score (0-4)	Interpretation
1	I think that I would like to use this system frequently.	4.67	3.67	Users strongly indicated a desire to use the system frequently, suggesting high perceived usefulness and appeal for regular use.
2	I found the system unnecessarily complex.	1.33	3.67	Users overwhelmingly disagreed that the system was complex, indicating a very simple and straightforward design.
3	I thought the system was easy to use.	4.90	3.90	There was near-unanimous agreement on the system's ease of use, highlighting an extremely intuitive and user-friendly interface.
4	I think that I would need the support of a technical person to be able to use this system.	1.60	3.40	Users strongly disagreed, confirming the system's self-explanatory nature and minimal need for technical assistance.
5	I found the various functions in this system were well integrated.	4.80	3.80	Users highly agreed that functions were well integrated, pointing to a seamless and cohesive user experience across different features.
6	I thought there was too much inconsistency in this system.	1.13	3.87	Users strongly disagreed, indicating a highly consistent design, layout, and operational flow, which enhances predictability.
7	I would imagine that most people would learn to use this system very quickly.	4.70	3.70	High agreement on rapid learnability, suggesting a very low learning curve and immediate usability for new users.
8	I found the system very cumbersome to use.	1.37	3.63	Users strongly disagreed, confirming the system is efficient and does not impose any burdens or difficulties during use.
9	I felt very confident using the system.	4.57	3.57	Users felt very confident in operating the system, indicating a strong sense of control and trust in its performance.

10	I needed to learn a lot of things before I could get going with this system.	1.33	3.67	Users strongly disagreed, reinforcing that the system requires minimal prior knowledge or extensive training to begin using effectively.
----	--	------	------	--

The system achieved a total average SUS score of 92.35. According to industry benchmarks, a SUS score above 68 is considered above average (Trymata, n.d.), and this indicate a highly usable system. This result strongly suggests that the Automating Peak Flow Data Extraction system is perceived as exceptionally user-friendly, efficient, and satisfactory by its public users.

5.3 Summary

This chapter presented a detailed evaluation of the Automating Peak Flow Data Extraction system, covering both accuracy testing and usability feedback. The comprehensive accuracy testing, including controlled environmental variations and user-driven tests, consistently achieved over 90% accuracy, confirming its reliability for extracting PEF readings. The system demonstrated strong performance under ideal conditions (96.67%), maintained high accuracy in low light (93.33% with enhancements), and showed resilience to minor camera tilts (90% at 15°). The user testing with 30 public users further validated its practical utility, achieving a 92.22% accuracy rate over 90 trials. The System Usability Scale (SUS) score of **92.35** indicates excellent perceived usability, highlighting the system's ease of use, learnability, and overall user satisfaction. Enhancements such as CLAHE and perspective correction improve robustness, while user feedback validates its practicality for asthma management. This evaluation confirms that the system successfully bridges the gap between manual and digital peak flow monitoring, offering an accurate, affordable, and user-friendly solution for asthma patients and healthcare providers. Future improvements will focus on enhancing adaptability to diverse conditions and integrating features directly requested by users to maximize utility and confidence.

6.0 CHAPTER 6 Conclusion and Future Work

6.1 Introduction

This Final Year Project aimed to address critical gaps in traditional asthma management by developing an innovative system that bridges the affordability of manual peak flow meters with the accuracy and convenience of digital monitoring solutions. The project successfully achieved its core objectives by automating the extraction of Peak Expiratory Flow (PEF) readings from conventional manual peak flow meters using advanced image processing techniques.

A significant achievement of this project is the successful development and implementation of a functional Web Application (WA). This application provides a user-friendly interface that allows patients to record, track, and visualize their respiratory data effectively. Through the integration of OpenCV.js, the system demonstrates robust image preprocessing capabilities, including precise red arrow detection and accurate Region of Interest (ROI) extraction. Complementing this, Tesseract.js was successfully incorporated for highly accurate Optical Character Recognition (OCR), reliably translating visual readings into digital data.

The system's performance was rigorously validated through extensive testing under diverse real-world conditions. It consistently demonstrated high accuracy rates: 96.67% under ideal conditions, maintaining 93.33% accuracy (improving to 96.67% with auto-brightness enhancements) in low-light environments, and retaining a commendable 90% accuracy even with minor camera tilts of 15°. Furthermore, comprehensive user testing yielded an exceptional System Usability Scale (SUS) score of 92.35, affirming the system's intuitive design, ease of use, and high user satisfaction. This project has delivered a practical and accessible solution for asthma monitoring, empowering a broad user base, regardless of their socioeconomic status or technical proficiency, to manage their condition more effectively.

6.2 Limitation

Despite the successful implementation and encouraging results achieved in this project, the Automated Peak Flow Data Extraction system, being a prototype, carries several notable limitations. These limitations predominantly arise from the inherent challenges associated with real-world image capture, device variability, and the technical boundaries of the chosen libraries and web-based development environment. Recognizing these constraints is essential in setting realistic expectations for the system's current capabilities and in identifying areas for meaningful future enhancement.

6.2.1 Image Capture Sensitivity to Environmental Factors

One of the primary limitations encountered involves the system's heavy sensitivity to environmental factors during image acquisition. Although measures such as Contrast Limited Adaptive Histogram Equalization (CLAHE) were implemented to enhance low-light image quality, the system's performance still degrades under extreme lighting conditions. Situations involving overly bright glare, backlighting, or severely dim environments can disrupt the clarity of captured images, which directly impacts the ability of OpenCV.js to accurately detect the red indicator arrow and the ability of Tesseract.js to perform precise Optical Character Recognition (OCR) on the numerical scale.

Moreover, while initial perspective corrections using homography transformation reduce minor tilting effects, the system's tolerance is limited to camera tilts within approximately 15–20 degrees. Exceeding this threshold introduces parallax distortion, where the apparent position of the red arrow shifts relative to the scale due to angle changes. This misalignment affects the accuracy of interpolated peak flow readings, resulting in potentially clinically misleading values. Similarly, improper camera distance can produce images that are either too zoomed in, cutting off critical parts of the scale, or too far out, reducing detail resolution and OCR reliability.

Another related issue involves background interference. In practical usage scenarios, cluttered or patterned backgrounds can introduce unwanted visual noise within the Region of Interest (ROI), making it harder for image processing algorithms to isolate the peak flow meter and its markings accurately. This interference increases the risk of both false positives and missed detections, thereby compromising system reliability. The prototype currently mitigates this by requiring explicit user guidance for optimal image capture, a solution that may be impractical for unsupervised or home-based use, especially among elderly or less tech-savvy patients.

6.2.2 Device Compatibility and OCR-Specific Challenges

The scope of this project included limited compatibility testing with two manual peak flow meter models. However, in real-world settings, a wide variety of peak flow meter designs exist, differing in aspects such as scale layout, font styles, arrow shapes, color schemes, physical dimensions, and surface finishes. Some devices feature matte surfaces, while others are glossy, affecting light reflection and image capture quality. These variations limit the generalizability of the current system, as it relies on hardcoded parameters such as specific color thresholds for arrow detection and fixed assumptions about scale spacing for value interpolation.

Furthermore, physical wear and tear on frequently used devices pose another challenge. Over time, numerical markings may fade, arrows may become discolored, and device casings may develop scratches, all of which degrade the visual clarity needed for reliable image processing and OCR. The current system lacks mechanisms to dynamically adapt to these conditions, thereby reducing performance on older or poorly maintained devices.

On the OCR side, while Tesseract.js is efficient in recognizing clear, printed text, it is not optimized for non-standard numeric fonts or partially obstructed characters commonly found on medical devices. Smudges, reflections, or faded digits often lead to OCR misinterpretation or failed text extraction. Additionally, minor inaccuracies can occur during interpolation when the detected arrow lies between two major scale markings. Because the system estimates values based on detected reference points, any missing or incorrectly detected numbers can result in slight deviations, particularly when relying on limited scale markers for value approximation.

6.2.3 API and Library Performance Limitations

Another limitation stems from the technical performance and operational dependency on client-side JavaScript libraries and external APIs. The system relies on face-api.js for facial recognition and Tesseract.js for OCR, both of which require substantial resources for model loading and execution. On lower-end devices or under poor internet connectivity, these processes incur noticeable delays, negatively impacting real-time responsiveness and user experience.

Face-api.js, for example, involves downloading large pre-trained models at runtime, which can extend login and registration times. As a client-side implementation, it lacks the computational efficiency and scalability of server-side inference systems. Likewise, Tesseract.js, though lightweight compared to its desktop counterpart, still demands significant processing power for complex image analysis and OCR tasks, leading to perceptible lags, especially on devices with limited CPU capacity.

Compounding this issue is the reliance on third-party Content Delivery Networks (CDNs) for delivering these libraries and associated resources. Any latency or downtime affecting the CDN directly impairs the application's performance and availability. In areas with unstable internet infrastructure, initial model loading or image processing operations may fail, rendering the system temporarily unusable. Additionally, the system's current assumption of continuous internet connectivity for model initialization and certain API calls limits its robustness in offline or constrained-network environments, a significant consideration for healthcare tools intended for rural or underserved areas.

6.3 Future Works

Building upon the foundational achievements of this prototype, several opportunities exist for future enhancements to improve the system's robustness, versatility, and clinical reliability. These proposed developments focus on addressing the limitations identified in Section 6.2 while also expanding the system's scope, scalability, and user-centered functionalities. The aim is to transition this project from a functional proof-of-concept into a mature, clinically practical tool for asthma management and broader healthcare applications.

6.3.1 Enhanced Image Processing and Artificial Intelligence Integration

A key avenue for future development lies in upgrading the system's image processing capabilities to handle complex real-world conditions more effectively. Current methods such as CLAHE and homography transformation offer baseline improvements but fall short in highly variable or uncontrolled environments. Integrating more advanced computer vision techniques, particularly deep learning-based object detection and pose estimation models, could enable the system to automatically detect and compensate for camera tilts, distance variations, and partial occlusions without requiring explicit user instructions. Models such as MediaPipe or YOLO (You Only Look Once) could be explored to robustly localize and track the peak flow meter, red arrow, and numerical scale in real-time video streams.

Additionally, developing a machine learning-based adaptive OCR module is highly recommended. This module could be trained on a diverse dataset of peak flow meter images featuring different font styles, scale layouts, and lighting conditions. By learning to dynamically recalibrate detection thresholds and number recognition logic based on contextual cues, the system could improve its accuracy when processing unfamiliar or degraded devices. Incorporating AI-driven preprocessing techniques, such as intelligent binarization, glare removal, and shadow compensation, would further enhance image clarity and subsequent OCR performance.

Another promising improvement involves integrating real-time, AI-powered user feedback within the camera interface. This feature would provide immediate visual or audio prompts, such as *"Move closer," "Reduce tilt,"* or *"Lighting too low"*, guiding users toward optimal image capture conditions before initiating processing. This interactive assistance could significantly improve first-attempt success rates and minimize errors caused by improper camera handling. Moreover, implementing a dynamic scale detection feature that automatically identifies the maximum and minimum values of any detected peak flow meter scale could allow for on-the-fly recalibration of interpolation logic, eliminating reliance on hardcoded margins and improving compatibility across various device models.

6.3.2 Scalability, Security and Cross-Platform Development

For the system to achieve long-term impact and clinical viability, enhancements to its scalability, security, and deployment strategy are crucial. The current web-based prototype operates primarily on client-side processes, which limits its scalability and resilience. Migrating core functionalities to a cloud-based backend infrastructure, utilizing platforms such as Google Cloud Platform (GCP), Amazon Web Services (AWS), or Microsoft Azure, would enable server-side model hosting, rapid inference capabilities, and support for larger user bases without performance degradation.

Furthermore, as the system involves the processing and storage of potentially sensitive health data, future iterations must adopt comprehensive security protocols. Implementing end-to-end encryption for all data in transit and at rest, alongside secure authentication mechanisms such as OAuth 2.0 or multi-factor authentication (MFA), would significantly enhance data protection. Compliance with internationally recognized health data regulations such as the Health Insurance Portability and Accountability Act (HIPAA) and the General Data Protection Regulation (GDPR) should be pursued to facilitate safe and ethical system deployment in clinical settings.

Additionally, while the Web Application (WA) format provides accessibility, developing native mobile applications for iOS and Android platforms offers substantial advantages. Native apps could harness device-specific hardware optimizations, such as advanced camera controls, background image processing, and offline data caching, which are difficult to implement reliably within web-based environments. These improvements would enhance system responsiveness and enable critical features such as continuous monitoring or offline data collection, especially in areas with limited internet infrastructure.

Finally, fostering a community-driven development and support ecosystem could contribute to the system's sustained growth. Introducing an in-app knowledge base, interactive troubleshooting guides, and user forums would empower patients and healthcare providers to share experiences, suggest improvements, and collaboratively refine system functionalities. Such a participatory approach would not only improve user satisfaction but also facilitate ongoing system refinement in response to real-world challenges.

6.4 Highlighted Contributions: Patient Progress Visualization and Image Guidance

One of the core contributions of this project is the mapping of breathing test results into interactive graphs to help patients clearly visualize their respiratory progress over time. This feature bridges the understanding gap between raw medical data and patient comprehension, enabling users to monitor their lung performance trends effectively. By transforming manual readings into a digital timeline of PEF data, patients and healthcare professionals can detect early warning signs of asthma deterioration and make informed decisions on treatment adjustments.

Additionally, the project introduces a novel image capture guidance mechanism that encourages users to position their camera properly to ensure accurate readings. Through a combination of visual feedback and precise detection of the red indicator arrow, the system supports users in identifying the best possible angle and distance for capturing high-quality images. This minimizes common errors related to lighting, tilt and misalignment, thereby increasing the reliability of the readings, especially in home-use scenarios without professional supervision.

Together, these two features which are breathing trend visualization and image capture guidance represent a significant enhancement over traditional manual methods. They promote proactive asthma self-management, increase reading accuracy and provide users with an easy-to-understand overview of their respiratory health. These contributions position the system as a forward-thinking digital healthcare solution, especially valuable in remote or resource-limited settings.

6.5 Summary

In conclusion, this project successfully developed an automated system for extracting Peak Expiratory Flow (PEF) readings from manual peak flow meters, delivering an accurate, user-friendly, and affordable solution for asthma monitoring and management. The implementation of image processing techniques via OpenCV.js and Optical Character Recognition (OCR) with Tesseract.js proved highly effective, achieving remarkable accuracy rates (e.g., 96.67% in ideal conditions and 92.22% in user testing) and earning an excellent System Usability Scale (SUS) score of 92.35, indicating strong user satisfaction.

Despite these achievements, the system faces limitations related to its sensitivity to extreme environmental conditions during image capture, the need for broader device compatibility, and the current scope of its functional and infrastructural capabilities. Recognizing these areas, future work proposes significant enhancements, including the integration of advanced AI for superior image processing and adaptive scale learning, expanded patient-centric features such as personalized action plans and trend prediction, and more robust functionalities for healthcare professionals like advanced dashboards and telehealth integration. Furthermore, addressing scalability through cloud-based backend services, strengthening security measures, and exploring native mobile application development will ensure the system's long-term sustainability and broader reach. Ultimately, this project lays a strong foundation for an accessible and effective digital health solution, with a clear roadmap to further revolutionize asthma care and improve patient outcomes globally.

7.0 Appendix

Appendix A - Final Year Project (FYP) Gantt Chart

a. FYP 1

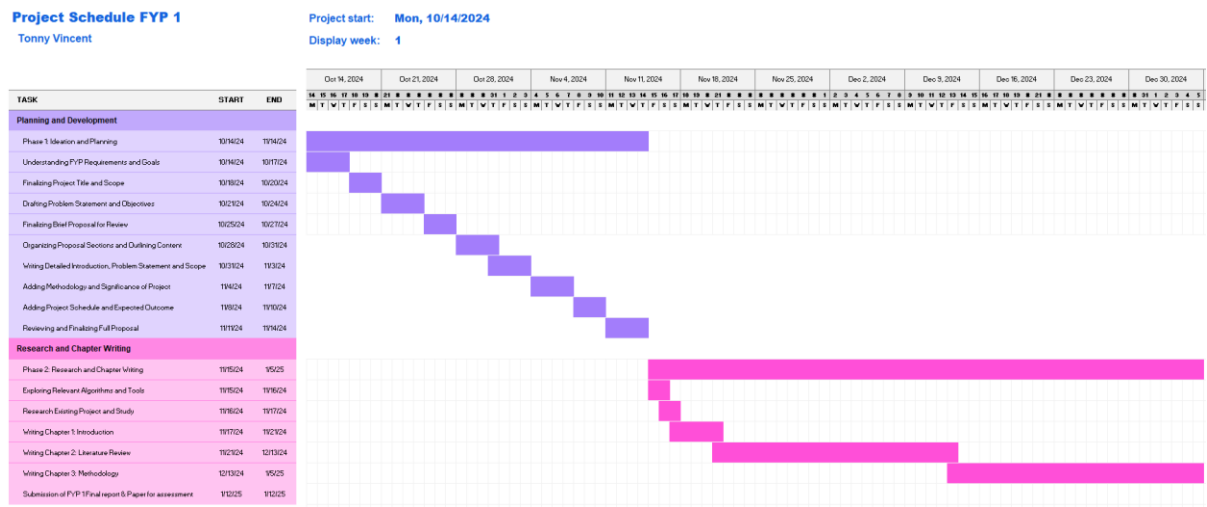


Figure 107: FYP 1 Gantt Chart

b. FYP 2

Project Schedule
Tonny Vincent

TASK	START	END
Research and Chapter Writing		
Phase 2: Research and Chapter Writing	4/11/25	7/26/25
Revised Structure of FYP Report	4/11/25	4/30/25
Submit Revised Structure of FYP Report	5/1/25	5/1/25
Writing Chapter 4	5/2/25	5/21/25
Writing Chapter 5	5/22/25	6/10/25
Writing Chapter 6	6/11/25	6/30/25
Finalizing Final Report FYP 2	7/1/25	7/19/25
Submit Final Report and Paper Assessment	7/20/25	7/20/25
FYP 2 Symposium	7/21/25	7/26/25

Figure 108: FYP 2 Info

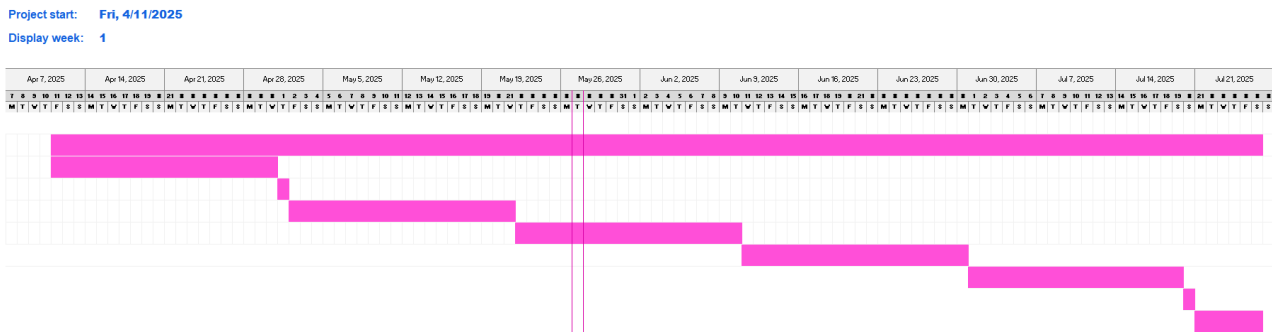


Figure 109: FYP 2 Chart

Appendix B – Several picture of public user



Figure 110: A public user (first-time user) conducting an experiment with the Automating Peak Flow Data Extraction system.



Figure 111: The result of public user extracted data

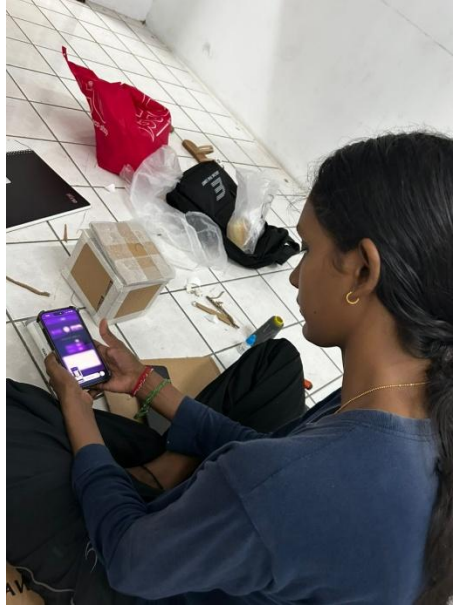


Figure 112: A public user (first-time user) conducting an experiment with the Automating Peak Flow Data Extraction system.



Figure 113: The result of public user extracted data

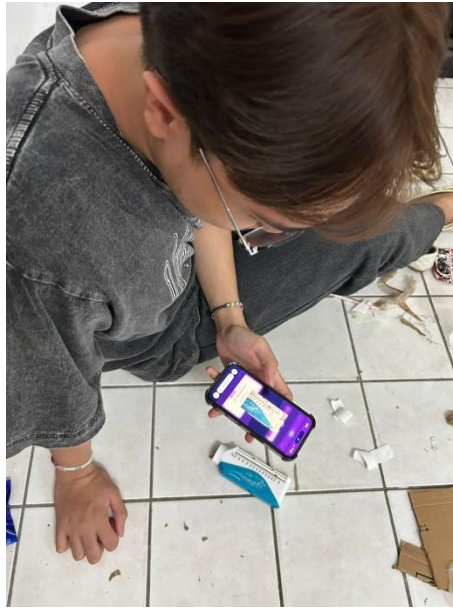


Figure 114: A public user (first-time user) conducting an experiment with the Automating Peak Flow Data Extraction system.

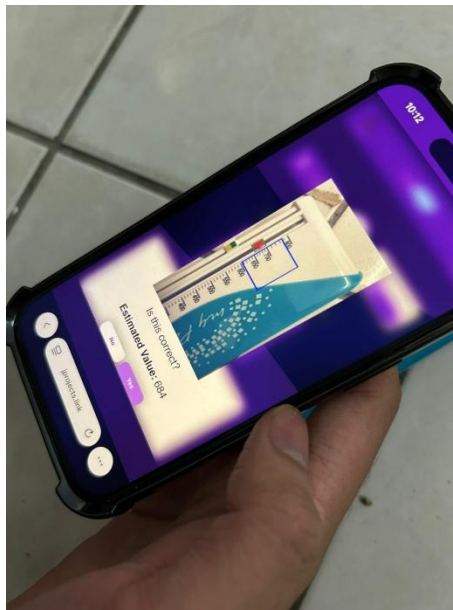


Figure 115: Figure 113: The result of public user extracted data

8.0 References

- Agile Alliance. (2024). Agile methodologies. Retrieved from <https://www.agilealliance.org>
- Apache Friends. (2024). *XAMPP*. <https://www.apachefriends.org/>
- Chan, H. S., & Chen, W. (2021). Computer vision techniques in medical device data extraction. *International Journal of Image and Graphics*, 22(3), 45-58.
- Coyle, R., & Wilson, C. (2019). Elderly and digital device usability. *Journal of Health Informatics*, 10(2), 78-86.
- Dou, G., Chen, R., Han, C., Liu, Z., & Liu, J. (2022). Water level monitoring by automated water level recognition in instruments such as rulers through digital image processing and machine learning. *Sensors*, 22(16), 6135.
- European Parliament and Council of the European Union. (2016). *Regulation (EU) 2016/679 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. Official Journal of the European Union, L 119, 1-88.
- Face API.js. (2024). Face-api.js. Retrieved from <https://github.com/justadudewhohacks/face-api.js>
- Global Initiative for Asthma (GINA). (2021). Global strategy for asthma management and prevention.
- Hierarchical Architecture. (2023). Tutorialspoint.com. https://www.tutorialspoint.com/software_architecture_design/hierarchical_architecture.htm#:~:text=Hierarchical%20architecture%20views%20the%20whole,network%20protocols%20and%20operating%20systems
- Matuzevičius, D. (2023). The use of synthetic image datasets to train deep learning models for ruler detection. *Applied Sciences*, 13(23), 12903.
- MDApp. (n.d.). Peak flow calculator. Retrieved from <https://www.mdapp.co/peak-flow-calculator-76/>
- Microlife. (n.d.). PF 100 digital peak flow meter. Retrieved from <https://www.microlife.com/consumer-products/respiratory-care/digital-peak-flow-meter/pf-100/>
- Microsoft. (2024). Visual Studio Code documentation. Retrieved from <https://code.visualstudio.com/docs>
- MySQL. (2024). MySQL documentation. Retrieved from <https://dev.mysql.com/doc/>
- Naptha. (2023). *Tesseract.js* [GitHub repository]. GitHub. <https://github.com/naptha/tesseract.js>
- National Heart, Lung, and Blood Institute (NHLBI). (2020). Asthma care and quality of life [Fact Sheet].
- Node.js. (2023). *Node.js documentation*. <https://nodejs.org/en/docs/>

- OpenCV.org. (2024). OpenCV.js. Retrieved from <https://opencv.org/>
- OpenJS Foundation. (2024). Node.js. Retrieved from <https://nodejs.org/en/>
- Rea, H., & McAuley, S. (2010). Asthma: The use of peak flow monitoring. *Journal of Primary Health Care*, 12(3), 35-41.
- Smith, R. (2007). An overview of the Tesseract OCR engine. *Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, 629–633. <https://doi.org/10.1109/ICDAR.2007.4376991>
- Spalluto, L., & Riordan, A. (2020). The digital divide and access to digital health for asthma patients. *Digital Health Journal*, 8(2), 123-134.
- Stack Overflow. (2017). OpenCV lines ruler detection. Retrieved from <https://stackoverflow.com/questions/42162229/opencv-lines-ruler-detection>
- Su, Y. (2023, August 3). Ruler scale image recognition. Medium. Retrieved from <https://medium.com/@yutongsu183/ruler-scale-image-recognition-c3bb7632f31a>
- Tesseract OCR. (2024). Tesseract.js. Retrieved from <https://github.com/naptha/tesseract.js>
- Trymata. (n.d.). *System Usability Scale (SUS): Definition, how to use it, & formula*. Trymata. Retrieved June 20, 2025, from <https://trymata.com/blog/system-usability-scale-sus/>
- U.S. Department of Health & Human Services. (n.d.). *Health Information Privacy*. Retrieved from <https://www.hhs.gov/hipaa/for-professionals/privacy/index.html>
- Vitalograph. (n.d.). myPEF. Retrieved from <https://vitalograph.com/products/mypef>
- W3Schools.com. (2023). *HTML, CSS, and JavaScript tutorials*. <https://www.w3schools.com/>
- Wen, Z., Smith, R. M., Connor, M., & Curran, J. M. (2024). Automated detection of rulers in shoe print images for accurate scale estimation using deep learning. *Forensic Science International: Digital Investigation*, 48, 301646.
- World Health Organization (WHO). (2022). *Guidance on ethics and governance of artificial intelligence for health*.