MONASH University

FIT3164 Data Science Project Part 2

Final Project Report

Automated Health Information System

Team MDS2

Foo Kai Yan | 33085625 | kfoo0012@student.monash.edu

Alicia Quek Chik Wen | 33045240 | aque0004@student.monash.edu

Eunice Lee Wen Jing | 33250979 | elee0075@student.monash.edu

Jesse Yow San Gene | 32794649 | jyow0001@student.monash.edu

Supervised by:

Dr Muhammad Fermi Pasha

Word Count: 11,307

Date: 25th October 2024

Table of Contents

1. Introduction

1.1. Project Introduction

The goal of the Automated Health Information System (AHIS) project is to completely transform the management of medical data. By replacing traditional manual data entry techniques with cutting-edge automated data entry technology, AHIS seeks to improve healthcare data entry processes. By automatically recognizing and identifying handwritten text from patient registration forms and instantly filling out all necessary patient details within the digital patient registration form on the web application, our solution will significantly speed up the data entry process and increase overall data entry accuracy.

The main objective of AHIS is to develop and implement a smart, intelligent system that incorporates autonomous handwriting recognition into a web-based platform so that medical professionals can quickly record and store patient data. By integrating this technology, AHIS ensures higher levels of precision in patient data recording while also alleviating the administrative workload on medical professionals. A large variety of handwriting styles can be accommodated by AHIS since it is made to handle and recognize both cursive and non-cursive handwriting styles.

Healthcare professionals will eventually be able to scan and upload handwritten documents using their mobile devices thanks to AHIS's anticipated growth to mobile platforms, which will boost user flexibility and accessibility. Strong access control procedures are incorporated into AHIS to protect patient confidentiality, allowing only those with authorization to view and access potentially sensitive data. In order to satisfy future demands in the healthcare business, the system's scalable database architecture is also made to accommodate the increasing volume of healthcare data.

1.2. Report Overview

The project is introduced, along with its primary objectives in the very start of this report. After the introduction, the report discusses the project's history through a comprehensive literature review, highlighting related projects and research that was previously covered in the project proposal.

The next section would be the project outcome, which also describes the project's limitations, constraints and how the final project deliverables align with the initial project requirements. This is followed by the project methodology section, which describes how the final project design was implemented and details the changes made from the original project design.

The software deliverables section then provides a detailed evaluation of the project’s software quality, with references to the code sections, which are included in the appendix. Before concluding, software and project critique will be presented, comparing the overall execution with the initial expectations and discussing how the project could have been improved to further enhance its success.

2. Project Background

2.1. Background Information

2.1.1. Health Information System (HIS)

The use of Health Information Systems (HIS) that allows medical facilities to retain patient records digitally and enhance overall patient care, has grown substantially over the years. These systems facilitate doctor-patient interactions and improve the quality of care by giving medical staff rapid access to patient data. By digitizing health records, HIS lessens the need for paper-based processes which facilitates the management, retrieval and exchange of patient data. Even while HIS has several advantages, there are drawbacks as well. Medical personnel frequently invest a lot of time in manually entering patient data. As a result, they take their focus away from providing direct patient care. In addition to decreasing efficiency, this raises the possibility of human error, which may have a detrimental impact on patient outcomes.

Furthermore, many healthcare professionals may find it difficult to switch to completely digital systems since they are unfamiliar with the necessary technology, especially older practitioners used to handwritten records. Since human data entry is time-consuming and susceptible to errors, this opposition may make it even more difficult for HIS to be widely adopted. Understanding these problems, our research aims to solve them by utilizing cutting-edge AI-based technologies like Handwritten Text Recognition (HTR) to automate data entry and raise the system's overall effectiveness.

Our goal in developing a web-based AHIS is to make data entry easier. Handwritten notes can be scanned by medical staff enabling the system to automatically extract and enter the necessary data into digital records. Healthcare workers can concentrate on patient care rather than administrative duties because this lowers the possibility of errors that come with manual data entry. In the end, using AI-powered solutions like HTR improves patient data accuracy as well as the effectiveness of healthcare data administration.

2.2. Project Rationale

2.2.1. Optical Character Recognition (OCR)

Optical Character Recognition (OCR) technology was originally developed in the 1970s to convert handwritten or printed content into machine-readable text (IBM, 2024). Many industries now employ OCR extensively, including healthcare, where it is essential for digitizing paper records. OCR improves healthcare systems' capacity to model and analyze huge amounts of data by converting scanned files into digital representations. OCR simplifies data entry and increases the accessibility of medical information in the healthcare industry by enabling the rapid and effective extraction of information from handwritten prescriptions, patient notes, and other paper-based documents.

However, OCR algorithms frequently struggle to recognize handwritten text, especially in healthcare settings where handwriting is complicated and fluctuating. This is where HTR is useful in this case.

A more specialized type of OCR called HTR was created especially for handwritten content recognition and digitization. HTR is essential to our AHIS project since it allows us to automatically convert handwritten notes into digital records. Prescriptions and patient paperwork are frequently written by hand by medical practitioners. By incorporating HTR into the system, these handwritten notes may be easily converted to digital format, greatly cutting down on the time and effort needed for manual data entry.

Since HTR has the potential to completely transform data entry in the healthcare industry, our project gives it top priority despite its implementation obstacles which include the necessity for high levels of accuracy and variations in handwriting styles, including cursive and non-cursive writing. HTR increases data integrity, lowers the possibility of transcription errors and boosts the general effectiveness of healthcare workflows by precisely capturing and digitizing handwritten text.

The integration of HTR into AHIS directly addresses the issues identified with traditional HIS systems. Healthcare workers can concentrate more on patient care and achieve better results by removing the need for manual data entry. Furthermore, AHIS is built to be scalable, ensuring that the system can manage rising demands as the amount of healthcare data increases. By restricting data access to authorized personnel only, AHIS also protects patient privacy through robust access control and security measures.

2.3. Related Works and Improvements Done

SWOT Analysis (Updated)	https://docs.google.com/spreadsheets/d/16cKLOg-zi5sZ8UK7EIWypiIHLunk8ErHrsvfGVaL68k/edit?usp=sharing

Figure 2.3-1 SWOT Analysis (Updated)

The updated SWOT analysis is shown above in Figure 2.3-1, we have incorporated insights from additional literature that further aligns with our project objectives. While some literature reviews from previous versions were referenced less extensively, we have added relevant comments reflecting shared strengths, weaknesses, opportunities and threats. Three additional literature reviews were included to provide more comprehensive coverage and closely relate to our project's scope. We realized that only two literature reviews that are highlighted yellow in the SWOT analysis, were considered out of the original Project Proposal & Literature Review. Our related works and any improvements done will be using these 5 literature reviews as good reference.

HTR has evolved substantially due to advancements in deep learning architectures like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) which address core challenges in recognizing complex handwriting features, processing sequences, and managing various styles. Research by Manchala et al. (2020) and Balci et al. (2017) has emphasized the strengths and limitations of these methods particularly when applied to specific datasets such as the IAM dataset. This review synthesizes improvements in HTR by an approach that emphasizes the contributions of recent work

to enhancing model robustness and accuracy particularly in domains with noisy or stylized handwriting such as medical records and personal health information.

2.3.1. Convolutional Neural Networks (CNNs)

CNNs have become central in extracting features from handwriting images, facilitating accurate recognition even across complex or cursive handwriting styles. CNNs' feature extraction abilities were highlighted by Manchala et al. (2020) who pointed out that these tools help recognize organized patterns in handwriting and capture important character details. On carefully prepared datasets, their CNN-RNN layer combination which was designed to take sequential dependencies into account achieved an accuracy of nearly 90.3%. Unfortunately when it came to cursive writing, they found that CNNs by themselves were inadequate, requiring preprocessing and data augmentation to handle overlapping or connected characters.

Similarly, Balci et al. (2017) highlighted CNNs' capabilities in character-level classification particularly in reducing vocabulary complexity. Their method improved accuracy that highlights the significance of CNNs for intricate character recognition. However, they noted that in order for CNNs to generalize effectively, they need big, annotated datasets like the IAM dataset. Sable et al. (2024) echoed these difficulties when they used CNNs to identify names of medicines and doctor notes in a healthcare setting. While highlighting CNNs' potential, they also pointed out problems like overfitting and picture quality dependency that could eventually cause performance to deteriorate.

Besides that, the study by Li et al. (2015) on structured personal health records from photographic medical records again emphasizes the use of CNNs in structured text extraction but it also shows how unreliable OCR may be because of image flaws like uneven shading or low light. CNNs shown efficacy in projects involving the extraction of personal health records from handwritten inputs without the need for additional algorithms to improve performance making adoption easier in healthcare settings.

2.3.2. Recurrent Neural Networks (RNNs)

RNNs particularly Long Short-Term Memory (LSTM) networks are essential for managing sequential dependencies in handwriting, capturing context over long sequences. In a significant advancement in the recognition of continuous handwriting strokes, Manchala et al. (2020) discovered that RNN layers in conjunction with CNNs improved accuracy in identifying characters often seen in cursive text. However, they pointed out that RNNs are sensitive to sequence quality which might deteriorate in low resolution data, making them difficult to use with noisy or constrained datasets.

The key to RNNs' performance is their capacity to retain memories of prior inputs which is especially important for handwriting identification as characters are less frequently distinct. Wang et al. (2021), for example, noted that adding RNN layers to HTR systems makes it easier to comprehend characters in a wider context that increases overall accuracy. RNNs can better handle handwriting variations by

processing information sequentially and catching crucial elements that most static models would miss.

RNNs struggle with training and performance stability in spite of their advantages. According to Manchala et al. (2020), standard RNNs frequently have trouble with vanishing gradients as sequences become longer which might impede learning. To enhance training results, they underlined the necessity of meticulous architecture design and the application of strategies like gradient clipping. Furthermore, Sable et al. (2024) pointed out that handwritten note quality and individual writing style variances might cause significant variability when applying RNNs to medical prescriptions requiring strong preprocessing to improve input quality prior to RNN application.

2.3.3. Connectionist Temporal Classification (CTC) for Sequence Prediction

Connectionist Temporal Classification (CTC) is an advanced technique designed to enhance sequence prediction tasks particularly in the context of unsegmented data. For HTR applications, CTC is especially useful since it dispenses with the necessity for previous segmentation into set character units by enabling models to predict sequences of different lengths immediately. Models may learn to map input sequences to output sequences using CTC without the need for alignment information which makes it perfect for handwritten text recognition where characters may be closely spaced or connected.

CTC works well with HTR systems because it can handle the complex structure of handwritten data which frequently has overlapping letters, varying spacing and a variety of writing styles. Better recognition accuracy is made possible by CTC that can offer a way to deal with the unpredictability of character predictions. Wang et al. (2021) claim that adding CTC to RNN designs greatly improves the system's capacity to comprehend continuous sequences which facilitates connected handwriting recognition without the drawbacks of conventional segmentation techniques. In applications like medical records where precise continuous text recognition is essential. This strategy fits the demand for reliable HTR systems.

Furthermore, CTC's effectiveness is directly related to data preprocessing and augmentation techniques which are essential for improving the quality of the training set. Sable et al. (2024) demonstrated how the prediction performance of the model is much increased when CTC is combined with preprocessing methods such image normalization and noise reduction. This synergy emphasizes how crucial high quality input data is especially for healthcare applications where things like blur or fading ink can make text difficult to read. All things considered, CTC is a major development in the field of HTR offering a versatile and efficient way to deal with the difficulties presented by handwritten and unstructured text input.

2.3.4. Data Augmentation and Dataset Limitations

The lack of high-quality and varied datasets has presented difficulties for HTR systems. The significance of data augmentation in overcoming dataset limits was highlighted by Manchala et al.

(2020) who proposed the creation of synthetic handwriting styles to improve generalization across a variety of styles. In the same way, Balci et al. (2017) identified dataset restrictions and pointed out that CNN and RNN models greatly benefit from augmentation which raises variability and accuracy in difficult situations like historical study or healthcare.

As Sable et al. (2024) demonstrated, data augmentation strategies are essential for managing prescription note quality variance in the healthcare industry. The accuracy of recognition is increased by methods including image enhancement, noise reduction, and the creation of synthetic data. According to Li et al. (2015), pretreatment and dataset augmentation can reduce OCR extraction mistakes related to image quality, which is important when OCR systems are used to digitize PHRs from printed medical records. The need for extensive labeled data remains a challenge reinforcing the value of weakly supervised learning which enables systems to function with fewer labeled examples in low-resource settings.

2.4. Dataset and Handwriting Text Recognition (HTR) Model

The success of our AHIS heavily relies on the quality and relevance of the datasets utilized for training our HTR model. The effectiveness of the system hinges not only on the underlying algorithms but also on the breadth, depth, and complexity of the training data which enables the model to accurately interpret and transcribe handwritten content commonly found in healthcare environments.

2.4.1. IAM Handwriting Dataset

The IAM Handwriting Dataset serves as a comprehensive and invaluable resource for training HTR models. It includes more than 1,500 handwritten texts produced by 657 separate individuals representing a wide variety of handwriting features and styles. Our model may be trained on a variety of writing forms that are commonly found in healthcare settings, including prescriptions, patient notes, and administrative forms due to this dataset's inclusion of both cursive and non-cursive scripts.

The dataset is carefully annotated and the handwritten images are accompanied by transcriptions at the word and character levels. This level of specificity makes it easier to create reliable models that can recognize text even when handwriting varies naturally. By training the model with the extensive variety of handwriting styles within the IAM dataset, our solution can effectively meet the diverse needs of medical professionals, resulting in a more robust and adaptable model.

In addition, the IAM dataset offers a notable benefit for data variability. The model's capacity for generalization is improved by including several authors with different writing styles which enables it to function effectively even when confronted with unfamiliar handwriting. This is crucial in the medical field because handwritten paperwork might differ greatly amongst practitioners.

2.4.2. Handwriting Text Recognition Model

Our HTR model is meticulously designed to automatically recognize and convert handwritten text into a machine-readable format. The architecture that is based on modern machine learning methods, combines RNNs for sequence prediction and CNNs for feature extraction. The CNN component captures key patterns and attributes of handwritten text by extracting important characteristics from input photos. Through this procedure, 2D picture data is converted into a more abstract representation that emphasizes important visual clues required for precise identification.

The RNN component takes over to model the sequential nature of handwritten text after the features have been retrieved. The RNN efficiently learns connections across time steps allowing it to read feature sequences even when handwriting can differ in alignment and spacing. The CTC layer, which is the peak of this design, enables end-to-end model training without necessitating exact alignment between input (handwritten text) and output (recognized text) sequences. When working with handwritten inputs, this flexibility is especially helpful because it enables the model to take into account handwriting inconsistencies such as variations in character forms and spacing.

The CTC layer enhances CNNs and RNNs to create a model that can handle a variety of handwriting styles and complexities making it ideal for the healthcare industry. Our method aims to improve the accuracy and efficacy of data entry processes in healthcare settings by effectively learning from the contextual information and geographical elements of the input photographs. Our AHIS is positioned to greatly improve patient care and streamline administrative procedures by utilizing the IAM dataset and cutting-edge modeling techniques.

In the end, our method reduces human error and frees up healthcare workers to concentrate more on providing direct patient care rather than laborious data entry duties by making it easier to accurately transcribe handwritten records.

3. Outcomes

3.1. What Has Been Implemented

3.1.1. Web Application

The AHIS has been developed as a progressive web application that can be accessed through any platform with a browser. The web-based system has been implemented with the necessary components required to operate as a functional health information system. The core components include patient registration, doctor appointment scheduling, medical consultations, diagnosis tracking, medication management, prescription handling, and physician profile management. These features are presented through a user-friendly interface that allows users to easily navigate through the system. Each component supports five standard operations, which are creating, listing, updating, deleting, and viewing records.

Additionally, field input validation and security features, namely role-based access controls and user authentication with one-way encryption used to ensure the protection of users' passwords, were added to the system. Input validation is applied to critical components, such as the login and registration pages, where users must provide the necessary credentials in the correct format, such as emails, before being granted access to the system. Furthermore, the smart data entry feature has been integrated into the patient registration module. This feature automates the entry of patient details by allowing users to upload a completed patient registration form, which helps streamline the registration process and significantly reduces the time spent on manual input, thereby enhancing overall efficiency.

3.1.2. HTR and ICROP Models

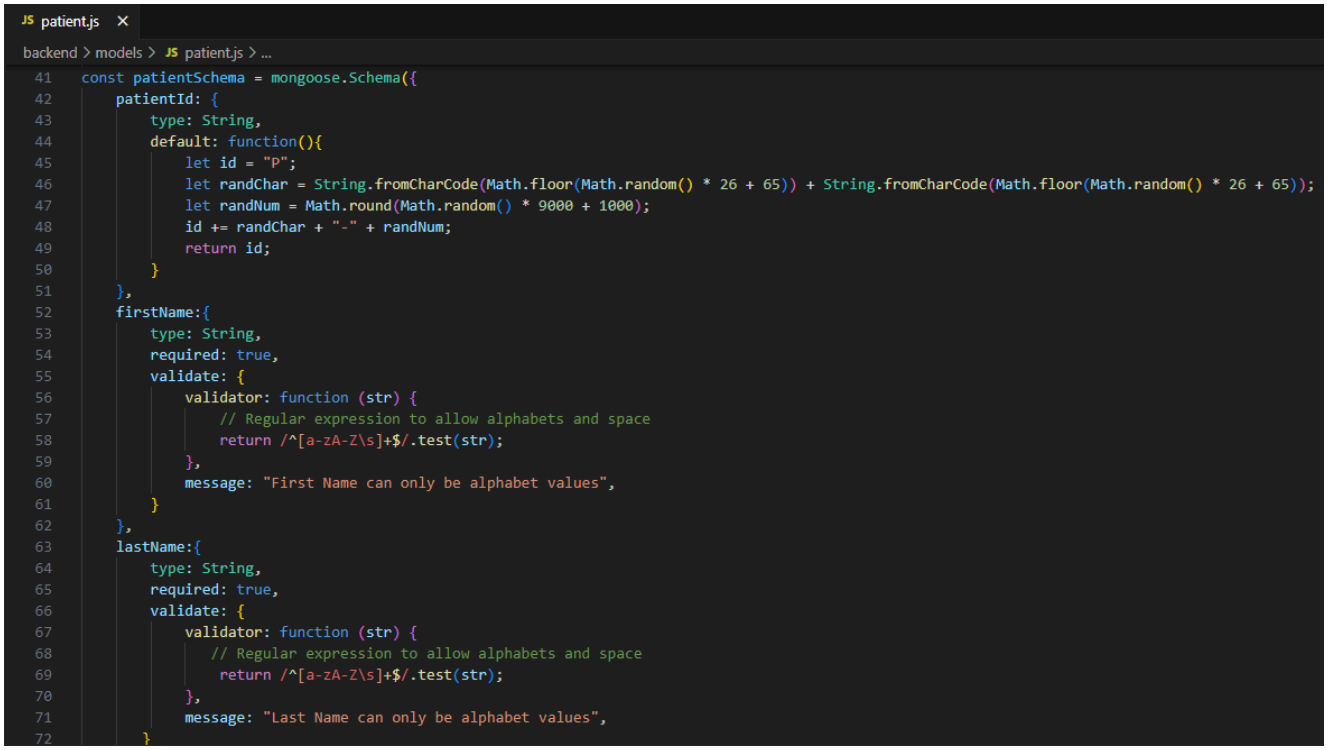
The Image Cropping (ICROP) model and a HTR model were integrated into the web-based system to facilitate smart data entry during registration. The ICROP model was developed to help enhance accuracy of the intended texts that's to be extracted by precisely isolating the intended handwritten text on the patient registration form. This is done by cropping the form into individual fields, which significantly improves the performance of the HTR model with each cropped field being processed by the HTR model to extract the handwritten text corresponding to each field. With the usage of a prebuilt HTR model, it was modified to meet the system's requirements, allowing it to process the cropped images and extract the text efficiently. The extracted text is then sent to the frontend, where it is displayed in a digital form for users to review and make any necessary edits before submission.

3.2. Results Achieved

The project has successfully delivered a comprehensive, functional and user-friendly web-based system with incorporation of a handwritten text recognition system to enhance the daily data entry operation. On further achievement of the web application, given the importance of data privacy and security in healthcare, security measures have been successfully implemented to ensure full compliance with privacy regulations and protect sensitive patient information. The user authentication feature safeguards the system against unauthorized access and operations, preventing data breaches and ensuring that confidential information is not leaked to third parties.

Additionally, the use of MongoDB as the system's database enables it to efficiently handle large volumes of data and a high insertion rate that meets the scalability requirement for the database of the health information system. The database schema is designed using a reference-based data model, which involves storing related data in separate collections and maintaining relationships between documents through references (Embedded Data versus References — MongoDB Manual, n.d.). This approach allows the system to efficiently manage distinct collections for various patient-related records, including but not limited to patient details, diagnoses, treatments, doctor consultations, and medical appointments. By separating these records into collections instead of embedding related data to the respective patient, the system optimizes storage efficiency, which is crucial for handling the growing size of patient data in healthcare systems. Furthermore, each collection incorporates schema

validation mechanisms to eliminate data errors and ensure data completeness and accuracy. The figure below shows samples of patient data model schema design with validation for name fields.



```
JS patient.js
backend > models > JS patient.js > ...
41  const patientSchema = mongoose.Schema({
42      patientId: {
43          type: String,
44          default: function(){
45              let id = "P";
46              let randChar = String.fromCharCode(Math.floor(Math.random() * 26 + 65)) + String.fromCharCode(Math.floor(Math.random() * 26 + 65));
47              let randNum = Math.round(Math.random() * 9000 + 1000);
48              id += randChar + "-" + randNum;
49              return id;
50          }
51      },
52      firstName:{
53          type: String,
54          required: true,
55          validate: {
56              validator: function (str) {
57                  // Regular expression to allow alphabets and space
58                  return /^[a-zA-Z\s]+$/.test(str);
59              },
60              message: "First Name can only be alphabet values",
61          }
62      },
63      lastName:{
64          type: String,
65          required: true,
66          validate: {
67              validator: function (str) {
68                  // Regular expression to allow alphabets and space
69                  return /^[a-zA-Z\s]+$/.test(str);
70              },
71              message: "Last Name can only be alphabet values",
72          }
```

Figure 3.2-1 Sample Of Patient Data Model Schema Design

Onto the model's results and achievements, the ICROP model is able to successfully crop the registration form into individual field images according to the coordinates of the form input fields and named based on its respective form field. Figure 3.2-2 shows an example of a filled registration form, while figure 3.2-3 shows examples of the cropped field images and its respective file name.

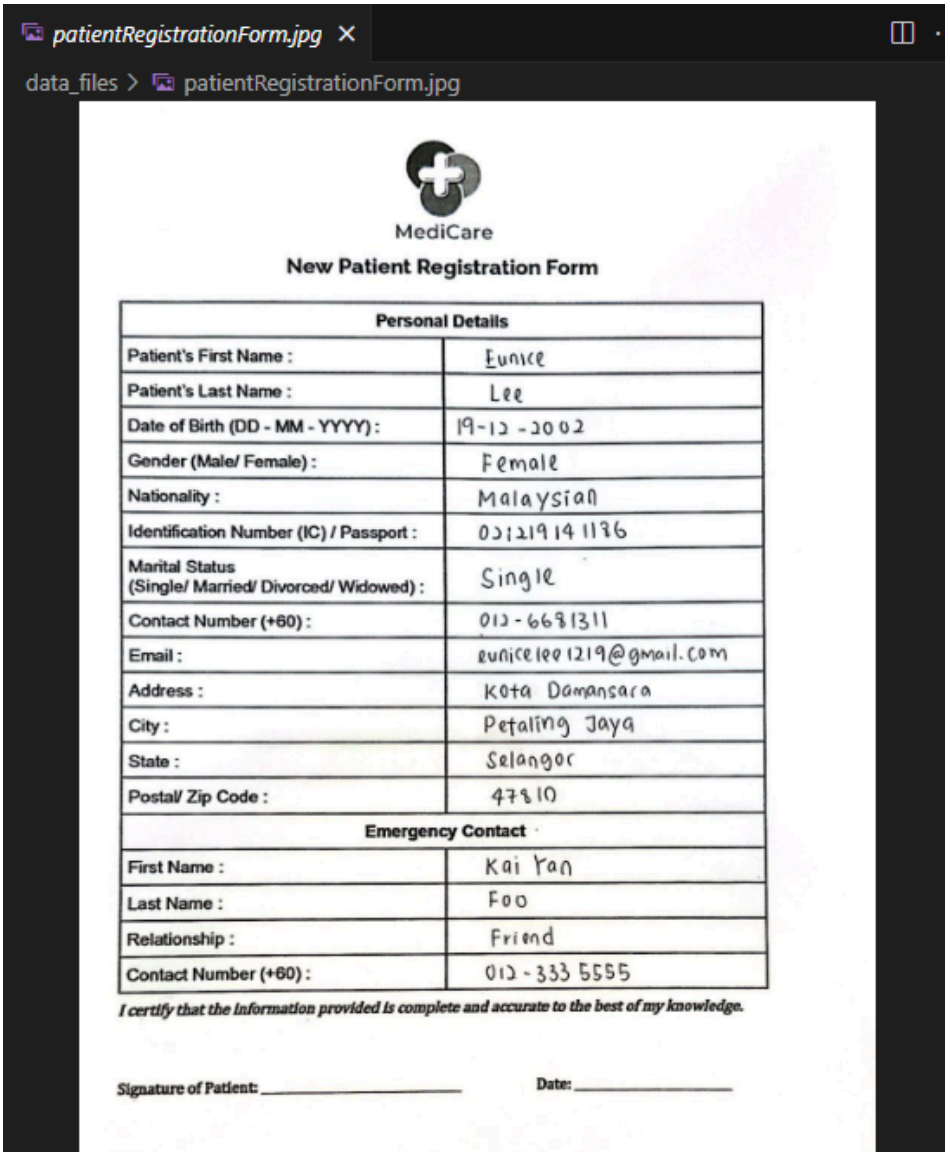


Figure 3.2-2 Completed Patient Registration Form

Figure 3.2-3 Cropped Images of Street Address, City and Birth Date Fields

The HTR model successfully extracts handwritten text with an accuracy rate of 75%. Each extracted text is stored together in an output JSON file according to its respective field, where it will be further processed in the frontend to be displayed, reviewed and modified as needed. The figures below show the sample output generated by the HTR model and the sample output displayed in the frontend after model processing is complete.

Figure 3.2-4 HTR Output JSON file

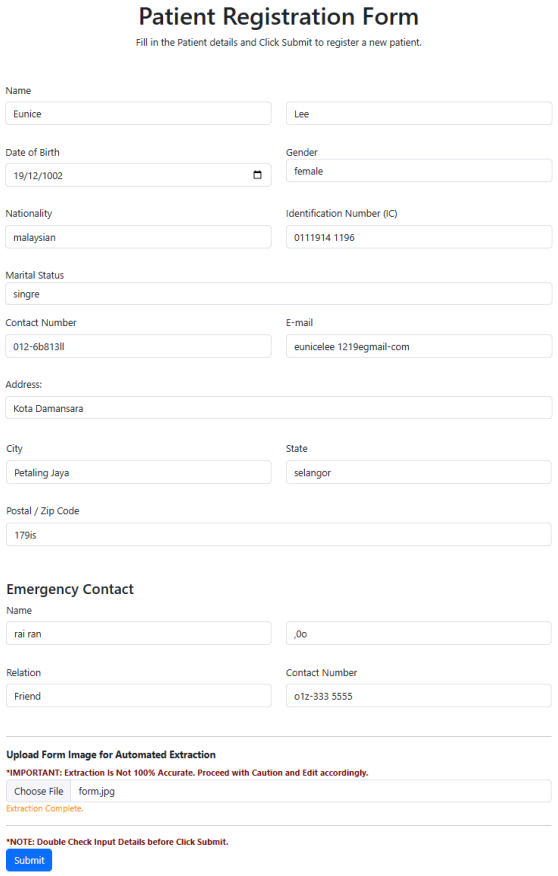


Figure 3.2-5 Frontend Autofill Display of HTR Output

3.3. Requirements Met

The project requirements are successfully fulfilled from the web system to the HTR model. The two tables below outline the functional and non-functional requirements with discussion on how each of these requirements have been met.

Requirements	**Discussion on how requirements are met**
Patient Registration Feature	Implemented add patient component that allows manual input or smart data entry option using HTR module.
Integrate ICROP and HTR model to web app	Since the web application is developed with Node.js and Express.js, both models integrated seamlessly into the backend using the child_process module. This module enables the

	execution of scripts in languages beyond JavaScript, allowing smooth communication between the models and the backend. The models interact harmoniously across both the frontend and backend with efficient data exchange through RESTful APIs.
User Authentication	This feature is carried out on the Login page, where upon accessing the system, users are required to log in their credentials to authenticate their identity before gaining access to the system functionality. An example of the Login page is shown in appendix F, figure 3.3-1. The user authentication is implemented using one-way encryption with the bcrypt library, which is a password-hashing function to securely store passwords.
Profile Management	This feature is implemented through the "Patient" and "Physician" components, where profiles of the patients and physicians are managed. Both profile details can be added, viewed, edited, or deleted as needed, ensuring comprehensive profile management. Example of the patient and physician profile management is shown in appendix F, figure 3.3-2 and figure 3.3-3.
Contains required features for a complete healthcare web app	The system contained essential features necessary for a comprehensive, fully functional health information system, including patient registration, doctor appointment scheduling, medical consultations, diagnosis tracking, medication management, prescription handling, and physician profile management. Each feature supports five key operations, which are creating, listing, updating, deleting, and viewing records.
Appointment and Encounter (Consultation) Feature	
Review patient past medical records	This feature is implemented in the "View Patient Detail" component, where all records related to the selected patient, including scheduled appointments and diagnosis, are displayed on a single page. This allows users to conveniently review all patient-related information in one go as shown in appendix F figure 3.3-4.

Table 3.3-1 How Functional Requirements are Met

Requirements	**Discussion on how requirements are met**
Scalable database design to store data	MongoDB was used as the system database as MongoDB efficiently handles large volumes of data and high insertion rate that meets the scalability requirement for the database of the

	health information system. The database schema is structured using a reference-based data model, which organizes related data in separate collections linked by references. This approach optimizes data storage, ensuring the database remains well-organized, scalable, and maintainable, while supporting strong performance as data volume grows (Codecademy, n.d.).
Data privacy and confidentiality	Data privacy and confidentiality are ensured through the implementation of robust user authentication and role-based access controls within the web system, which prevents unauthorized access, restricts unauthorized operations, and safeguards confidential information from being leaked to third parties.
Results displayed on web app is easily to comprehend	This is handled by storing the generated output of the HTR model in a JSON file, which is then passed to the frontend to be processed and displayed on the digital registration form, mapping each entry to its corresponding form field. An example of the result displayed on the digital registration form can be seen in section 3.2, figure 3.2-5 Frontend Autofill Display of HTR Output.
Satisfaction on the HTR model accuracy	The HTR model achieved a 75% accuracy rate by utilizing the IAM Handwriting Dataset for training. Despite the slight error that might be encountered due to the HTR limitation in recognising certain special symbols such as @, # and /, accuracy is satisfactory with the majority of the extracted text being accurate.
User Friendly User Interface	The user interface is designed to be easily navigable for users of all technical skill levels as verified by usability testing documented in the testing report. A navigation bar and side menu featuring frequently used components, such as the appointment list, patient list, and home page, have been added to facilitate smooth navigation throughout the system. An example of the navigation bar and menu is shown in appendix F, figure 3.3-5.

Table 3.3-1 How Non-Functional Requirements are Met

3.4. Justification of Decisions Made

3.4.1. Using Prebuilt HTR Model

One of the team's initial challenges was implementing the HTR model, as we had no prior experience with building deep learning algorithms. Combined with the project's demanding workload and the need to balance assignments from other FIT units, the team decided to use a prebuilt HTR Model, upon supervisor approval, to help speed up the project progress and to alleviate the workload needed to complete the project on time. After researching available options, the team has decided to use a HTR model built by Harald Scheidl, which offers detailed documentation and an overview of its functionality (Scheidl, 2020). This made it easier to adapt the model to the project's needs and streamline its integration into the web-based system.

This decision had a major impact on the team project success for it allowed the team to focus on other important aspects of the project, such as identifying the need for an additional model to crop the form according to form fields. This adjustment enabled the HTR model to transcribe the intended text in the scanned form image into digital text more accurately, supporting the project requirement of achieving a satisfactory HTR model accuracy rate.

3.4.2. Incorporation of ICROP Model

As previously noted, the need for an ICROP model was identified after testing the workings of the pre-built HTR model. The ICROP model was developed to enhance the accuracy of text extraction by precisely isolating the handwritten text fields on the patient registration form. By cropping the form into individual fields, the ICROP model significantly improves the HTR model's performance, as each isolated field is processed to accurately extract the handwritten text corresponding to each specific form field.

3.4.3. Web Application Framework

The MEAN stack framework, which consists of MongoDB, Express.js, Angular, and Node.js, was chosen over newer framework components such as ReactJS and NestJS, as half of the team had prior experience in developing a web application using MEAN stack framework. This decision allowed the team to split into two sub-teams, with the experienced members focusing on building the web-based system, as they didn't have to spend additional time learning new frameworks. As a result, the team could concentrate more on the application's logic and functionality, accelerating the development process. Additionally, this provided more time to focus on complex tasks, such as integrating the two models into the system without compromising their performance.

3.4.4. Using Bcrypt in User Authentication

Bcrypt is used in user authentication as a cryptographic hash function to securely store passwords by incorporating a salt, which is a string of random characters, to create a unique hash to protect against rainbow table attack and brute force attacks (Bcrypt, 2020). Additionally, bcrypt is adaptive, allowing the iteration count to be increased over time to make hashing slower, which helps maintain its

resistance to brute-force attacks as computation power grows (Bcrypt, 2020; Grigutytė, 2023). This resilience to brute-force and rainbow table attacks makes bcrypt an ideal solution for securely storing passwords in application backends, reducing susceptibility to dictionary-based cyberattacks (Grigutytė, 2023). During user login, bcrypt hashes the entered password, and the system compares this hash to the stored hash, ensuring secure password verification (Grigutytė, 2023).

3.5. Project Result Discussion

The project aims to expedite the data entry process within a health information system, particularly during patient registration process, by integrating a handwritten text recognition module. This enhancement serves to optimize daily workflows in hospitals and clinics, accelerating data entry tasks, boosting user productivity, and ultimately contributing to better patient care. Such an aim was fulfilled through our AHIS with the smart data entry feature integrated with the ICROP model and HTR model in the patient registration component.

In discussing the result of the HTR model, through testing the model with our own form samples with various handwriting styles, it is evident that the performance of the model heavily depended on image quality. Specifically, images with background shadows led to poorer accuracy compared to clear, shadow-free scans, underscoring the importance of high-quality image input. Detailed results are documented in the testing report with image examples, providing a comprehensive analysis of the model's performance. Besides that, the model demonstrated satisfactory accuracy in transcribing numbers and letters, though it showed minor inaccuracies when detecting special symbols such as @, /, and #.

3.6. Project Outcome Limitations

Lack of Search Box Feature

Finding particular records, such as patient records, is a tedious and lengthy procedure due to the lack of a search box option. Healthcare professionals are required to manually scroll through the entire patient list in the absence of this functionality, which can be particularly time-consuming in large databases with thousands of entries. In addition to impeding prompt patient care, this delay in obtaining vital medical data can hinder timely patient care, reduce productivity, and decrease overall efficiency within the healthcare system.

Huge volume of dependencies for HTR Model

The large volume of dependencies required for the HTR model results in longer execution time required, particularly during the model's initial run. This delay occurs because the HTR model must load and execute numerous dependencies to function as intended. However, this delay is typically reduced in subsequent runs, as the system can utilize cached memory from the initial execution, allowing certain functions to bypass reprocessing and improve overall performance.

Specific Structured Form

The ICROP model requires the use of a specific structured form for accurate detection of handwritten text, as the model identifies text areas based on predefined pixel coordinates. This presents a limitation in terms of flexibility, as the model is tailored to a particular form layout. However, this limitation is considered minor, as each healthcare institution typically uses its own standardized forms. Adapting the project to work with different forms would only require updating the pixel coordinates to match the new form's structure.

File Format Limitation

The ICROP model is limited to working with specific image file formats such as .JPG or .PNG. This means that when physical forms are scanned, healthcare professionals must ensure that the forms are saved in these supported formats. If the forms are saved in other formats, such as .PDF or .DOCX, the ICROP model will be unable to detect the handwritten text based on its predefined pixel coordinates, resulting in functionality issues.

3.7. Potential Future Improvements

Search Box Feature Implementation

The absence of a search box significantly hampers the user experience, making it difficult and time-consuming to locate specific patient records. Productivity is eventually impacted as a result of annoyance and discontent with the system's use. By enabling users to swiftly search through the database and retrieve data information, a search box feature would significantly increase the system's efficiency. This improvement is essential for guaranteeing seamless and user-friendly interactions between AHIS and the users.

Web Application Analytical Tools Implementation

The web application can be significantly enhanced with additional analytical features to support user analysis. These may include an overall timeline of patient registrations and discharges, enabling users to identify specific time periods throughout the year and observe patterns in disease occurrence. Additionally, the application could generate weekly or monthly progress reports, providing insights into the diseases patients were diagnosed with, thereby facilitating more informed decision-making and improving overall healthcare strategies.

Web Application Deployment on the Cloud

In the future, when external constraints are minimized, the AHIS would be deployed on a cloud platform. Through this change, authorized users would be able to easily access the web application from any device, including smartphones, tablets, and desktop computers. By utilizing cloud technology, AHIS would improve accessibility and flexibility, allowing healthcare professionals to use the system in different environments, which would ultimately improve collaboration and the quality of patient care.

3.8. Other Outcome - Related Issue of Relevance

Impact of Language Variability

The differences in language and handwriting styles found in various healthcare settings are also important factors to take into account. It may be difficult for the model to generalize accurately since doctors and other healthcare workers may have unique handwriting patterns influenced by their cultural, linguistic or educational backgrounds. This is especially crucial in areas with many languages or where the system's training data is in a language other than the predominant language of healthcare professionals. The accuracy and dependability of medical records and data input procedures may be impacted by such linguistic diversity which could result in inconsistent data interpretation.

Integration with Existing Healthcare Systems

There are advantages and disadvantages to integrating the HTR system with well established HIS. Although speeding up data entry is the aim, there may be compatibility issues that call for additional modifications or API connections. Workflow modifications may be necessary to existing HIS infrastructures in order to properly integrate the HTR system and any incompatibility could have a negative effect on the project's long-term viability. Maintaining accurate and consistent healthcare documentation across platforms would need smooth data entry, retrieval and updating of medical records.

4. Methodology

4.1. Project Design

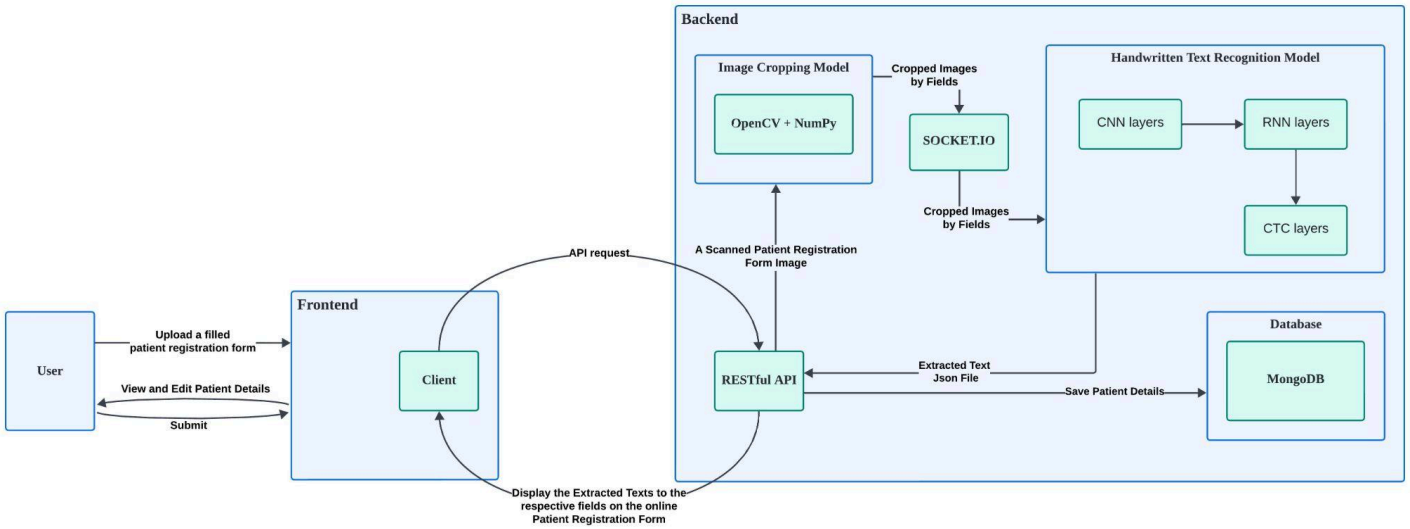


Figure 4.1-1 Project Final System Design

The final project design presented in figure 4.1.1. showcases a sophisticated and fully developed web application for AHIS. It encompasses a seamless logical flow, beginning with patient registration and culminating in the issuance of medication as per the prescriptions provided by the assigned physician for each diagnosis.

The process begins when a user uploads a completed patient registration form in an image file format (.JPG or .PNG) through the AHIS web application. Once the image is successfully uploaded, the

ICROP model is called upon in the backend. This model crops the uploaded image into specific fields based on preset pixel coordinates, and the cropped images are saved in a designated folder, each named according to its respective field. Upon completion of this step, a text would be shown in the user interface (UI) to notify the user that the images are ready for text extraction.

Next, the user would need to initiate the text extraction process, which calls upon the HTR model in the backend. The HTR model identifies and extracts the text from each of the cropped images. The extracted text is then automatically transferred into a digital form displayed on the AHIS web application, pre-filling the relevant fields. At this stage, the user is given the option to review and correct any inaccuracies in the recognised text before submitting the final patient registration form, which is stored safely in the MongoDB database.

4.1.1. Image Cropping (ICROP) Model Design

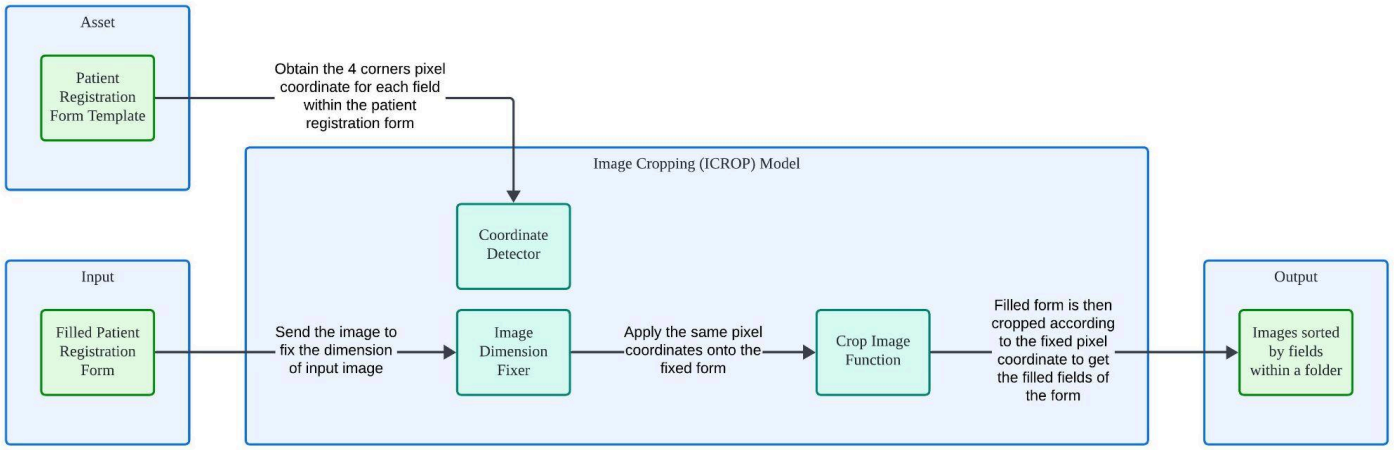


Figure 4.1.1-1 ICROP Model Design

Before the filled patient registration form is processed by the ICROP model, it is first sent to the coordinate detector function, where the pixel coordinates of the four corners of each field are saved. Once the filled-in patient registration form is successfully uploaded and sent to the backend, the ICROP model begins by preprocessing the image. This involves converting the form to grayscale and adjusting its dimensions to match the standard patient registration form template, which is set to 1242x1755 pixels. The forms uploaded must maintain these dimensions to ensure accurate field extraction that will be done by the HTR model.

Using the pixel coordinates stored from the template, the ICROP model identifies and crops the corresponding fields from the filled-in form. This ensures consistency and accuracy in detecting the handwritten text areas. After the fields are successfully cropped, each image is renamed according to its field name and saved in a designated folder. This folder is located within the HTR model directory, where the cropped images will later be processed for text recognition.

4.1.2. Handwritten Text Recognition (HTR) Model Design

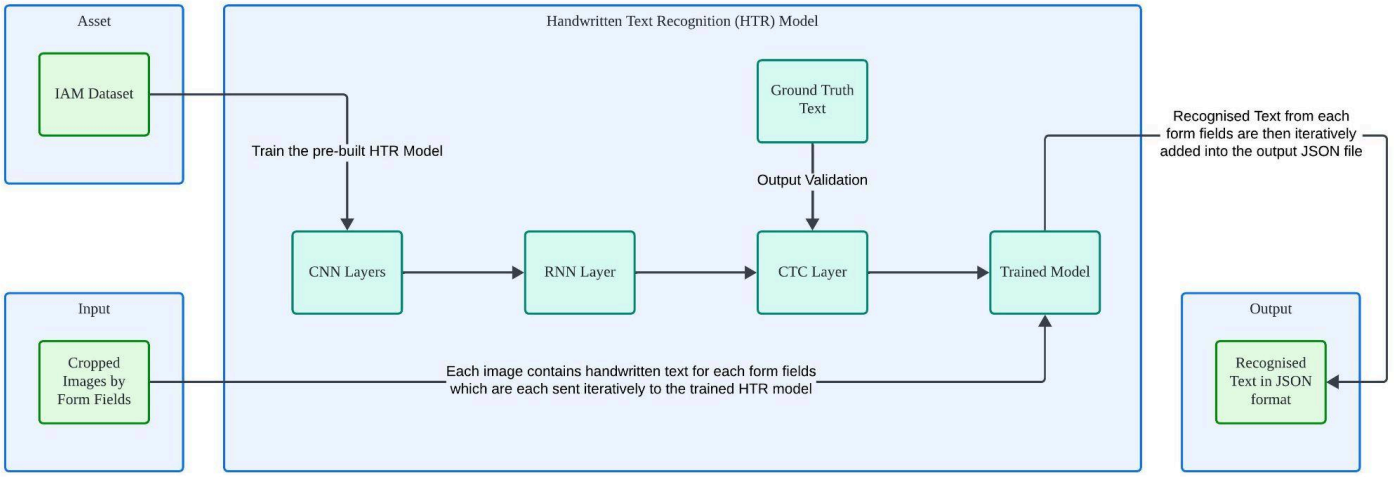


Figure 4.1.2-1 HTR Model Design

Once the user initiates the text extraction process, the HTR model begins by iteratively selecting the cropped images, which have been organized by form fields, from the designated folder. Each cropped image is processed by the HTR model to extract the handwritten text. The extracted text is then saved in a JSON file, ensuring that the data is structured for easy retrieval and use.

Upon completing the extraction, the text stored in the JSON file is automatically populated into the corresponding fields of the digital patient registration form within the AHIS web application. This allows the user to review and, if necessary, correct any errors in the recognized text before submitting the final form. The HTR model ensures that the handwritten text is digitized efficiently while providing flexibility for user validation.

4.2. Deviation From Initial Design

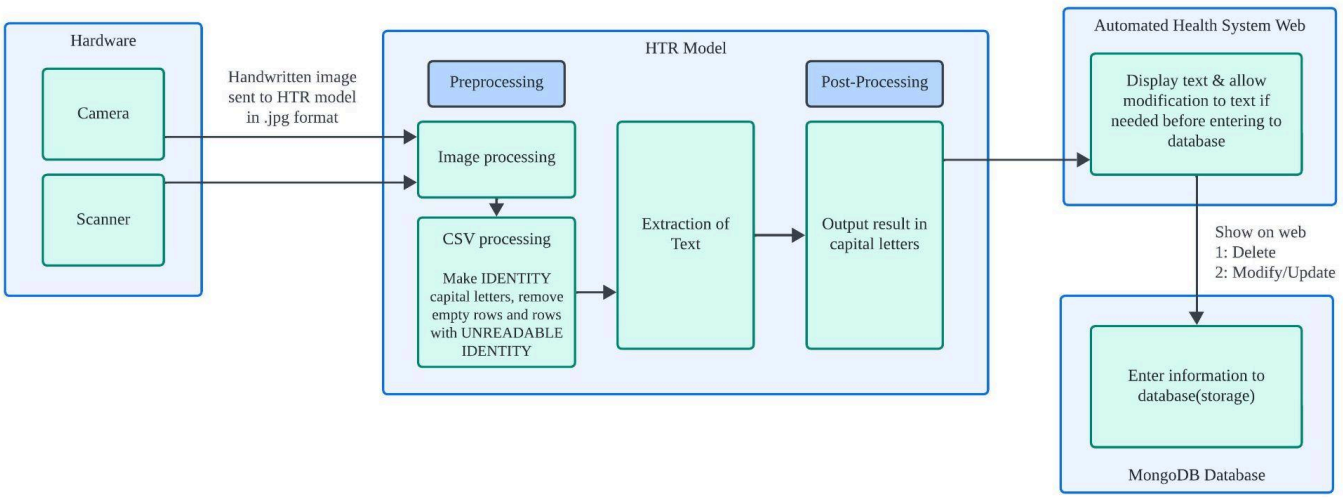


Figure 4.2-1 Project Initial System Design

During the planning phase of the project, the team initially intended to develop both the AHIS web application and the HTR model from scratch. However, due to time constraints and other unavoidable circumstances, the team decided to use a pre-built commercial HTR model discovered on the web instead. This model was trained and adapted by the team before being integrated into the AHIS web application. On the other hand, the development of the AHIS web application proceeded as originally planned, and it was built from scratch by the team.

As illustrated in Figure 4.2-1, the ICROP model was not part of the initial design plan. However, during the project execution, the team realized that due to the specific requirements of the pre-built HTR model, the images needed to be cropped and pre-processed before being inputted into the HTR model. Without cropping, the HTR model would recognize unintended text within the image rather than the targeted handwritten text fields of the image form. To address this issue, the team decided to develop the ICROP model to extract and crop the relevant sections of the patient registration form, ensuring that only the desired text fields are processed by the HTR model. This addition to the project scope marked a significant but necessary deviation from the initial design plan.

4.3. Design Implementation

4.3.1. Data Collection and Analysis

Due to the limited availability of suitable datasets for effectively training, testing, and validating the HTR model, the team selected the IAM Handwriting Database as the primary dataset for this project. As mentioned in Section 2.4.1., the IAM dataset includes handwritten contributions from 657 writers, resulting in a total of 1,539 pages, with each page containing a unique array of handwritten words. Altogether, these pages represent 115,320 distinct words written in a variety of handwriting styles, offering a rich source of variation for training the HTR model (Nader AbdalGhani, 2021).

With an average of roughly 2.34 pages contributed by each writer, the dataset's broad writer pool offers significant handwriting diversity. The HTR model can be more able to generalize and identify a variety of handwritten inputs from various individuals due to the variation in writing styles, letter forms, and spacing. Furthermore, every page in the IAM dataset includes a variety of both uppercase and lowercase characters, punctuation, and numbers, which enhances the training process by including a wide variety of character combinations.

By examining the dataset's word frequency and handwriting complexity distribution, the team was able to assess the HTR model's performance across a range of common and uncommon phrases as well as handwriting style differences. In order to make the HTR model appropriate for real-world applications in healthcare documentation, where handwriting variability is a common difficulty, the team used this large dataset to build a strong model that could accurately recognize a variety of handwriting patterns.

4.3.2. Software Tools and Hardware Requirements

This project encompasses two primary automation components—the HTR and ICROP models—and a fully functional AHIS web application. The team was divided into two sub-teams, with one group focusing on the model development, including training and implementation of the HTR and ICROP models, while the other team focused on the front-end and back-end development of the AHIS web application, MongoDB database management, and model integration.

The AHIS web application, HTR, and ICROP models were primarily developed using personal laptops and desktop devices. Each model and component was implemented and trained on different devices based on available hardware resources. As shown in Figure 4.3.2-1, the HTR model training occurred on Laptop D, while the ICROP model development, implementation, and training were conducted on Laptop E and Laptop C. The AHIS web application's development was distributed across Laptops A, B, and F, all of which were the team members' personal devices.

These devices had varied processing capabilities, which influenced the selection of hardware for specific tasks for the project. Laptops featuring higher memory and processing power allocated to model training and computationally demanding tasks. For a detailed breakdown of hardware specifications, refer to Figure 4.3.2-1.

Hardware	**Processor Specification**	**Installed RAM**
Laptop A	8th Gen Intel(R) Core(TM) i5-8265U	8.0 GB
Laptop B	11th Gen Intel(R) Core(TM) i5-1135G7	16.0 GB
Laptop C	8th Gen Intel(R) Core(TM) i5-8250U	8.0 GB
Laptop D	12th Gen Intel(R) Core(TM) i5-12500U	32.0 GB
Laptop E	11th Gen Intel(R) Core(TM) i5-11400H	16.0 GB
Laptop F	Intel(R) Core(TM) Ultra 7 155H	16.0 GB

Figure 4.3.2-1 Hardware Specification

Category	**Components**	**Tools / Libraries / Framework**
Web Application Development	Framework	MEAN Stack
Frontend Development	Web Development Platform	Angular

Backend Development	Database	MongoDB
	Runtime Environment	Node.js
	API Testing	Postman
Model Development	Deep Learning Framework	PyTorch, TensorFlow, Keras, OpenCV
Code Development	Web Application Programming Language	JavaScript, CSS, TypeScript, HTML
	Model Programming Language	Python
	Integrated Development Environment (IDE)	Visual Studio Code (VSC)
	Debugger	Built-in debugger in VSC
	Repository Host / Code Management Process	GitLab

Figure 4.3.2-2 Software Specification

In the development of the AHIS system, other than hardware, a range of software components was also utilized to ensure efficient and effective model training, implementation, and web application deployment. Figure 4.3.2-2 details the software components used throughout the project.

The project's model development relied heavily on libraries such as PyTorch, TensorFlow, Keras, and OpenCV. TensorFlow, with its extensive resources for machine learning, was essential for creating and implementing the HTR model, while Keras, an API within TensorFlow, simplified the deep learning aspects necessary for handwriting recognition (Terra, 2020). OpenCV played a key role in both the HTR and ICROP models, as it contains many comprehensive sets of computer vision algorithms that will be used to handle image processing to crop images and identify handwritten text (OpenCV, 2020). Through the integration of these libraries, both models gained enhanced functionality and reliability in their respective processing tasks.

Python was selected for model development due to its comprehensive library support and its native compatibility with TensorFlow. For backend development, JavaScript served as the primary language for the AHIS web application, while TypeScript, HTML, and CSS were the main languages used for frontend development, ensuring a responsive and user-friendly interface.

To manage project version control, GitLab was employed, facilitating efficient tracking of code modifications and seamless collaboration across the team. GitLab's compatibility with Continuous Integration/Continuous Deployment (CI/CD) systems helped streamline deployment, supporting the team's iterative development approach. Together, these software and hardware tools established a robust environment for the successful development and deployment of the AHIS project.

4.3.3. Packages and Libraries Requirements

Numerous packages and libraries were utilized during the development stage of the project. These packages and libraries support everything for the project, from frontend and backend development to data processing and model training. An exhaustive list of all the core libraries utilized throughout the project can be found in Figure 4.3.3-1 alongside on how they were used in the project. The project's intended results were made possible in large part by these packages and libraries.

Project Domain	Packages/Libraries	Version Used	Purpose
AHIS Web Application	Node.js	v18.17.0 - v20.17.0	JavaScript runtime environment was used to run backend code, enabling server-side logic and handling client-server interactions in the AHIS web application
	Socket.IO	4.8.0	Allows real-time communication between web clients, servers, and the models
	Bootstrap	5.3.3	Assist the development of responsive web pages without needing to write extensive custom CSS
	Express.js	4.19.2	Help set up the backend server, define API endpoints, and handle routing for seamless data flow within AHIS
	Bcrypt.js	2.4.3	Used to encrypt sensitive data like user account passwords
	Cookie-parser	1.4.6	Used for managing cookies in the web application
	JSONWebToken	9.0.2	Used to secure user authentication with token-based access control
	Mongoose	8.5.2	Used to make it easier to work with data within MongoDB database
	RxJS	7.8.0	Used to efficiently handle events and observables, manage asynchronous data flows in the Angular frontend
HTR and ICROP Models	Edit Distance	0.5.2	Used within the HTR model to improve text recognition accuracy by comparing

			and aligning recognized text with expected text
	Lightning Memory-Mapped Database	1.0.0	Integrated with the ICROP and HTR model to handle high-performance, in-memory data processing for efficient storage and retrieval of images and data
	Matplotlib	3.2.1	Used to help visualize data during the model training and validation phases, aiding in analysis and fine-tuning of the models
	NumPy	1.19.5	Used to handle large arrays, matrices, and mathematical functions for image data processing for the ICROP model
	OpenCV	4.4.0.46	Allows image preprocessing, manipulation, and field extraction in the ICROP model
	Path.py	15.0.0	Used in the ICROP and HTR models for effective file path management
	TensorFlow	2.4.0	Used to develop and train the ICROP and HTR model

Figure 4.3.3-1 Packages and Libraries Specification

The functionality of the HTR model required particular version requirements in addition to the core libraries and packages mentioned above. To maintain compatibility and best performance, the HTR model's implementation requires Python version 3.8. Because it complies with the dependencies and configurations of the libraries used for the HTR model, this version requirement is essential for ensuring smooth operation within the AHIS project environment.

4.3.4. Activities Performed

4.3.4.1. Model Training, Testing, and Validation

The IAM dataset was utilized by the team to train the pre-built HTR model. The training process, as illustrated in Figure 4.1.2-1, involves multiple stages, starting with CNN layers, followed by RNN layers, and concluding with a CTC layer for decoding.

Initially, the CNN layers are responsible for extracting various features from the input images, identifying patterns, and transforming these patterns into sequences of text (Craig & Awati, 2024).

The extracted features are then passed to the RNN layers, which perform sequential text analysis by considering prior inputs in the sequence to influence the current input and output data. This allows the RNN to accurately predict handwritten text sequences by learning dependencies across time steps (What Are Recurrent Neural Networks? | IBM, n.d.).

Finally, the CTC layer decodes the output from the RNN into readable text by comparing it with the ground truth text. It calculates the loss during the testing and validation phases and backpropagates the model to adjust the neural network weights, ensuring accurate text prediction (A. Ansari, B. Kaur, M. Rakhra, A. Singh, & D. Singh, 2022). This comprehensive training process enables the HTR model to effectively recognize and predict handwritten text with high accuracy.

4.3.4.2. Web Application Development

In addition to implementing the HTR and ICROP models, the team dedicated significant efforts to developing the AHIS web application. As detailed in Sections 3.4.3. and 4.3.2., the project team was divided into two sub-teams, with one sub-team focusing specifically on the AHIS web application, leveraging the MEAN stack framework due to the sub-team's members familiarity with its components.

The web application development began with a comprehensive project requirements gathering session, during which the team collaborated to identify necessary features, functionalities, and design considerations for the AHIS application. These initial requirements were then submitted to the project supervisor to ensure its alignment with the project goals.

Once the project requirements are confirmed, the sub-team then starts on the core development activities which includes building web functionalities, integrating the MongoDB database, and creating RESTful APIs to support confirmation between the frontend and backend. Functionality testing occurred concurrently with development, allowing developers to verify each component before moving to the next, facilitating a smoother, issue-free progression. This development process was mainly influenced by the hybrid project management approach where waterfall methodology was specifically used for web development. This combination allowed for iterative improvements within a structured framework, ensuring robust and efficient implementation of the AHIS web application.

4.3.4.3. Model and Web Application Integration

Both the ICROP model and the HTR model will be integrated into the backend of the AHIS web application as both these models combined serves as an additional functionality that will streamline the patient registration process in healthcare institutes.

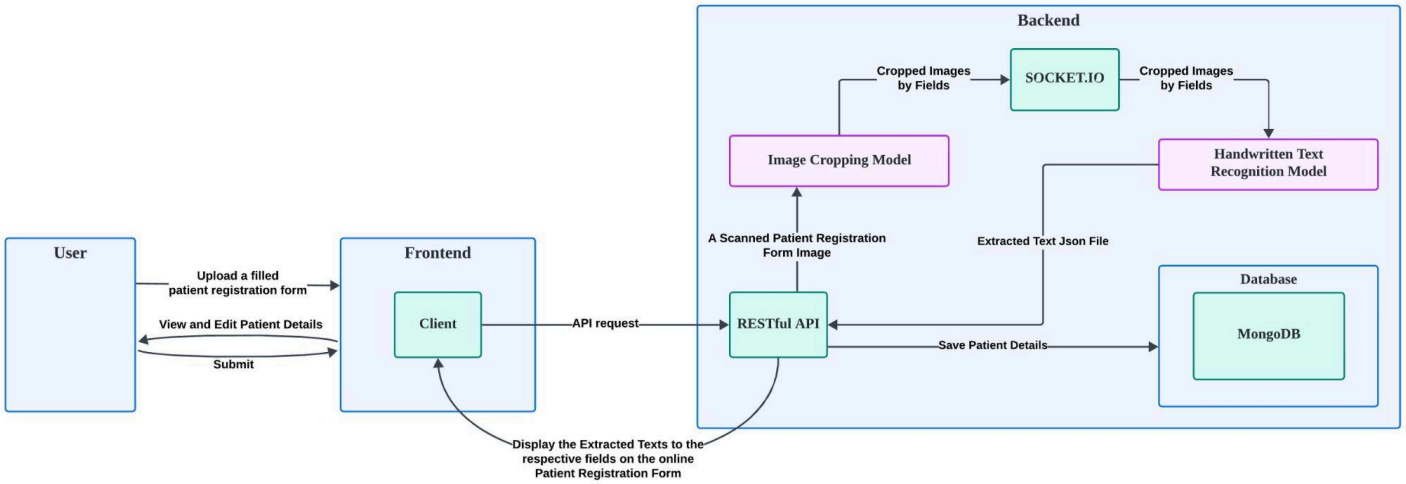


Figure 4.3.4.3-1 Model and Web Application Integration Architecture

To automate data entry during the patient registration process, the HTR and ICROP models are integrated within the current version of the AHIS web application. The workflow and system integration of these models are shown in Figure 4.3.4.3-1 above. A physical registration form is first completed by patients and presented to the administrative department of the healthcare center. The filled-out form will be then scanned by an administrative staff and uploaded to the AHIS web application.

Once the form is submitted through the frontend, it is sent to the backend, where the HTR and ICROP models are deployed. The ICROP model processes the form by cropping and preparing specific fields for the HTR model, which then performs text recognition tasks. Once the models have processed the form and generated the recognized text, a JSON output is created and sent back to the frontend to be displayed on the UI. Users can examine the recognised text, fix any errors if needed, and submit the completed data to the MongoDB database after it is shown on the user interface.

All communication between the frontend and backend of the AHIS web application is managed via a RESTful API, ensuring efficient and secure data handling throughout the process.

4.3.4.4. Project Testing

Software testing is an essential phase in software development, serving to identify bugs and errors that may impact the AHIS web application's functionality and performance. The team enhances and optimizes the AHIS web application iteratively by identifying and fixing these problems. The tests carried out for this project are described in Sections 4.3.4.4.1 to 4.3.4.4.3 below with the specific tests can be found in the testing report linked in Figure 4.3.4.4-1 below. The tests that were conducted included Blackbox testing, Integration testing, and Usability testing, each targeting different aspects of the system to ensure the reliability of the web application.

Testing Report	https://drive.google.com/file/d/1nGodHxzlkOi4TOq-AghdMXEnfZaRe7z0/view?usp=sharing

User Guide	https://drive.google.com/file/d/1wVjQ5bEnU8M8Tp0UchdEIos7zDrqYK7v/view?usp=sharing

Figure 4.3.4.4-1 Testing Report and User Guide

4.3.4.4.1. Blackbox Testing

The black box testing phase conducted by the team focused on evaluating key functional aspects of the AHIS web application and the HTR model to ensure each component operated correctly from an end-user perspective. More detailed information on how each test was carried out and their results can be found in the testing report as linked in Figure 4.3.4.4-1 above.

For the web application, the black box testing involved assessing the web application's core functionalities, such as the User Login feature, New Patient Registration feature, and the integrated Appointment Scheduling, Diagnosis, and Prescription System. Every feature was thoroughly tested to ensure that user inputs, outputs, and interactions fulfilled the necessary standards. This helped the team to identify any inconsistencies or mistakes that would impact the application's functionality.

For the HTR model, Blackbox testing was performed under a range of scenarios to evaluate its accuracy and robustness in recognizing handwritten text. This included tests with various input conditions, such as backgrounds with and without shadows, text written in different ink colors (blue, red, pink), cursive or capitalized handwriting, and entries containing symbols and numbers. Additionally, we tested the model's performance with thicker pens, exploring how well it generalizes across varying handwriting styles and ink types. For further details on the testing procedures and results, please refer to the specific sections in the testing report.

4.3.4.4.2. Integration Testing

The Integration testing phase focused on verifying that the AHIS system's core components interacted seamlessly and met functional requirements when integrated. The core components of the project includes the ICROP and HTR models, the MongoDB database, and the frontend and backend of the web application. The purpose of this testing was to identify any issues with performance, compatibility, or API interactions that would not have been shown in individual components testing. Through this process, the team identified and resolved integration defects early, thereby increasing system robustness and ensuring all components could work together effectively.

The testing included integration of the ICROP model with the web application to confirm accurate image cropping and submission functionalities, the HTR model to ensure effective handwritten text recognition and processing within the web interface, and seamless data storage and retrieval between the web application and MongoDB database. For detailed testing procedures and observed results, please refer to the Integration Testing section of the testing report as linked in Figure 4.3.4.4-1 above.

4.3.4.4.3. Usability Testing

The usability testing for the AHIS web application involved gathering individuals from both individuals in and out of Monash University to interact with the web application system freely and provide feedback through a structured Google Form. Test participants were encouraged to explore various features, such as user login, patient registration, and appointment scheduling, without guidance from the project team to ensure impartial and honest feedback. The feedback covered various aspects from ease of use and functionality to overall user experience, highlighting areas where the web application system performed well and where improvements could be made.

From the user responses, it was noted that while the web application system was generally well-received, multiple users suggested implementing search and filtering functions to enhance data navigation as the database grows. Although these improvements were not possible within the current project timeline, they will be considered for future improvements. For a comprehensive review of the usability testing procedures, responses, and further details, please refer to the testing report linked above in Figure 4.3.4.4-1.

5. Software Deliverables

5.1. Summary of Software Deliverables

5.1.1. Automated Health Information System

The software deliverables include a comprehensive web application along with two crucial model components, ICROP model and HTR model that's integrated into the web-based system, to fulfil the project requirements. The ICROP model is responsible for cropping the uploaded patient registration form image into individual field images to isolate the intended text of each form field and enhance accuracy of text extraction by the HTR model. Once cropped, the HTR model transcribes the handwritten text from each field image into digital text, which is then automatically populated into the corresponding input fields of the digital patient registration form.

The web-based system is a progressive web application that provides all necessary functionalities for a health information system, including patient registration, doctor appointment scheduling, medical consultations, diagnosis tracking, medication management, prescription processing, and physician profile management. The application features a user-friendly interface, secure user authentication, and MongoDB as the database to ensure robust data handling and accessibility.

5.1.2. Documentation

The deliverables of documentation includes a comprehensive user guide, detailed installation instructions, and clear explanations of the web component functionality. This ensures users can effectively install, operate the program. Covering everything from initial setup to troubleshooting common issues, the documentation is designed to provide a seamless user experience.

The following documentation sub-section will provide a technical setup guide and a user guide. The technical setup guide will cover steps for installing and configuring the software, while the user guide will offer clear instructions on how to use each software function component.

5.1.2.1. Technical User Guide

Here's a short and clear instruction in launching the web application:

1. Download the source code from this GitLab repository.
2. Setup the environment with installation of required libraries as stated in the team user guide documentation.
3. Run the web application by running command `ng build` followed by `node server.js`.
4. Access the web application through this link: http://localhost:8080/home#/ where you will be greeted by the system login page such as below.

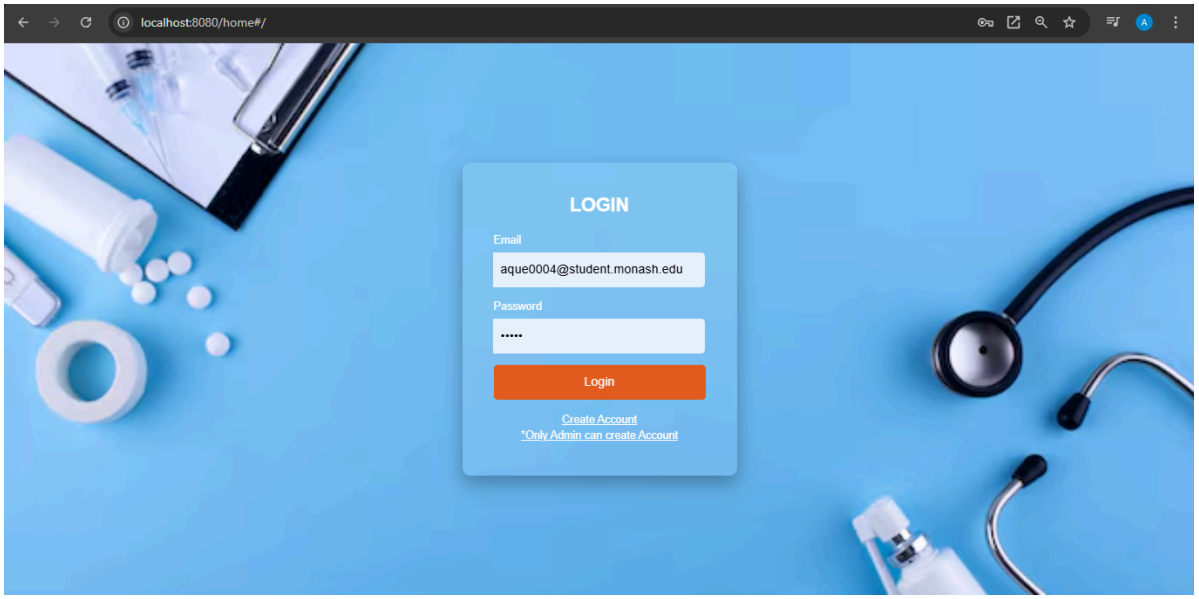


Figure 5.1.2.1-1 System Login Page

For a more in-depth technical guide, please refer to the team user guide documentation on the technical user guide section.

5.1.2.2. End User Guide

This section will provide a brief overview on the system available features and its usage. For a more detailed user guide, please refer to the end user guide section in the team user guide documentation.

Upon a successful login, the user will be directed to the system homepage as shown below:

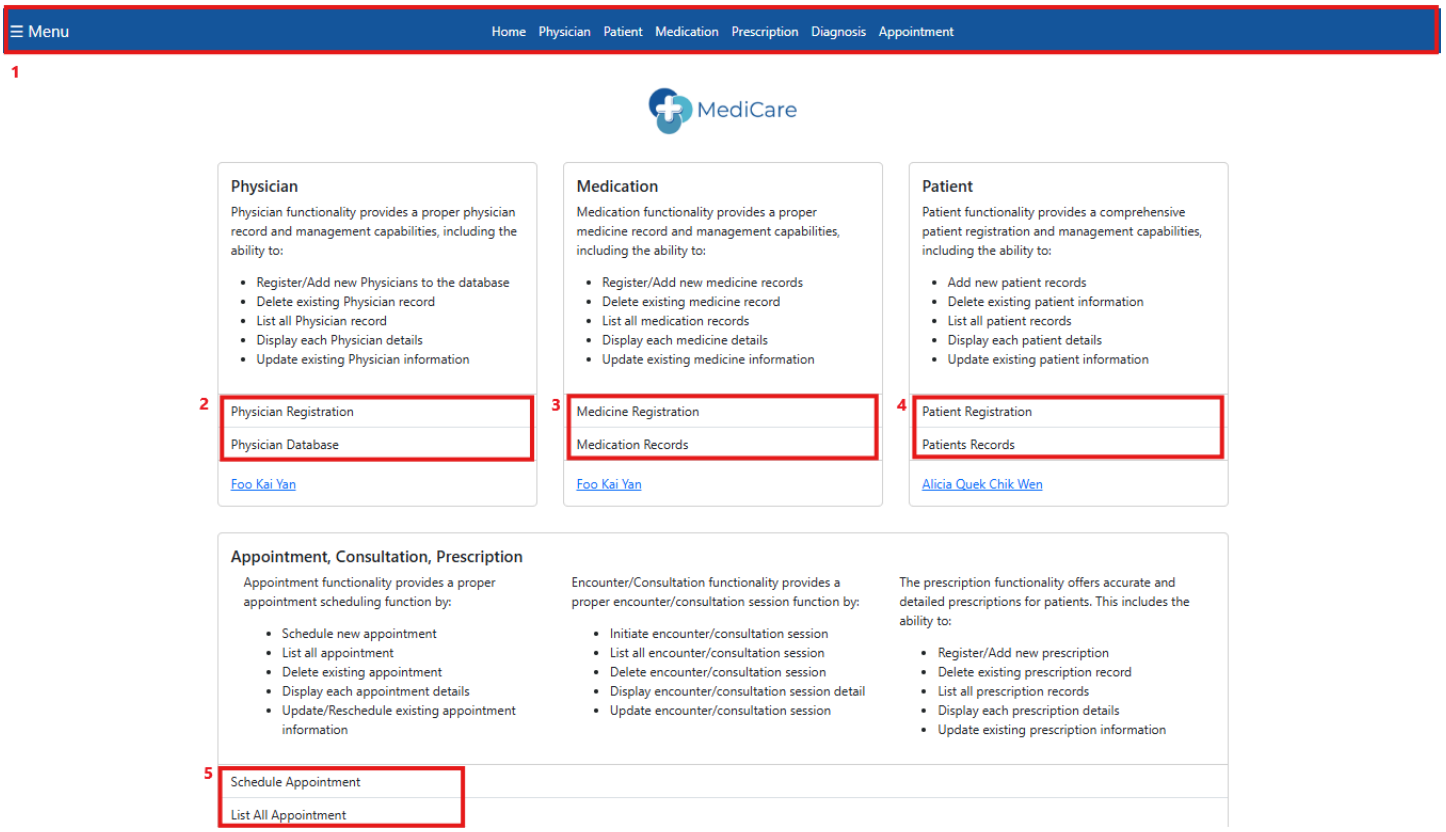


Figure 5.1.2.2-1 System Homepage

1. **Navigation Bar Menu**:
    - Provide users the ability to quickly navigate to their desired feature components.
2. **Physician Registration and its Record List**:
    - Provide users the ability to register new physicians and manage physicians profile details such as delete, update or view selected records.
3. **Medicine Registration and its Records**:
    - Provide users the ability to register new medicines and manage the medicines details such as delete, update or view selected records.
4. **Patient Registration and its Records:**
    - Provide users the ability to register new patients and manage the patients profile details such as delete, update or view selected records.
5. **Appointment Scheduling and its Record List**:
    - Provide users the ability to schedule doctor's appointments for the patient and manage the appointment details such as delete, update or view selected records, followed by conducting consultation, diagnosis and providing prescriptions.

5.2. Software Quality Summary

This section presents a summary overview of how our web application addresses key software qualities, including robustness, security, usability, scalability, portability, and maintainability. Discussion on identified shortcomings and areas for potential improvement are also included.

5.2.1. Robustness

Robustness in the AHIS web application is addressed through comprehensive error handling, exception management, and data validation mechanisms, which ensure that the system can effectively handle unexpected inputs or system errors without crashing or causing major disruptions (What Is Software Robustness? How to Ensure Quality and Reliability - Nexus Software Systems, n.d.). Several robustness features are integrated across both the frontend and backend, with a strong emphasis on data validation and error-handling practices.

The frontend, built using Angular, leverages Angular's built-in form classes for input tracking and validation at the individual form field level (Nwose Lotanna, 2024). This setup allows the system to handle a wide range of inputs, including those that are unexpected or incorrect with display of warning messages upon invalid input. For instance, on the login page, when users enter their email and password, any input that doesn't conform to the email format or left empty triggers a warning message, guiding users to correct their entry as shown in figure 5.2.1-1. This approach ensures a smooth and reliable user experience by proactively addressing potential errors before they impact the system stability.

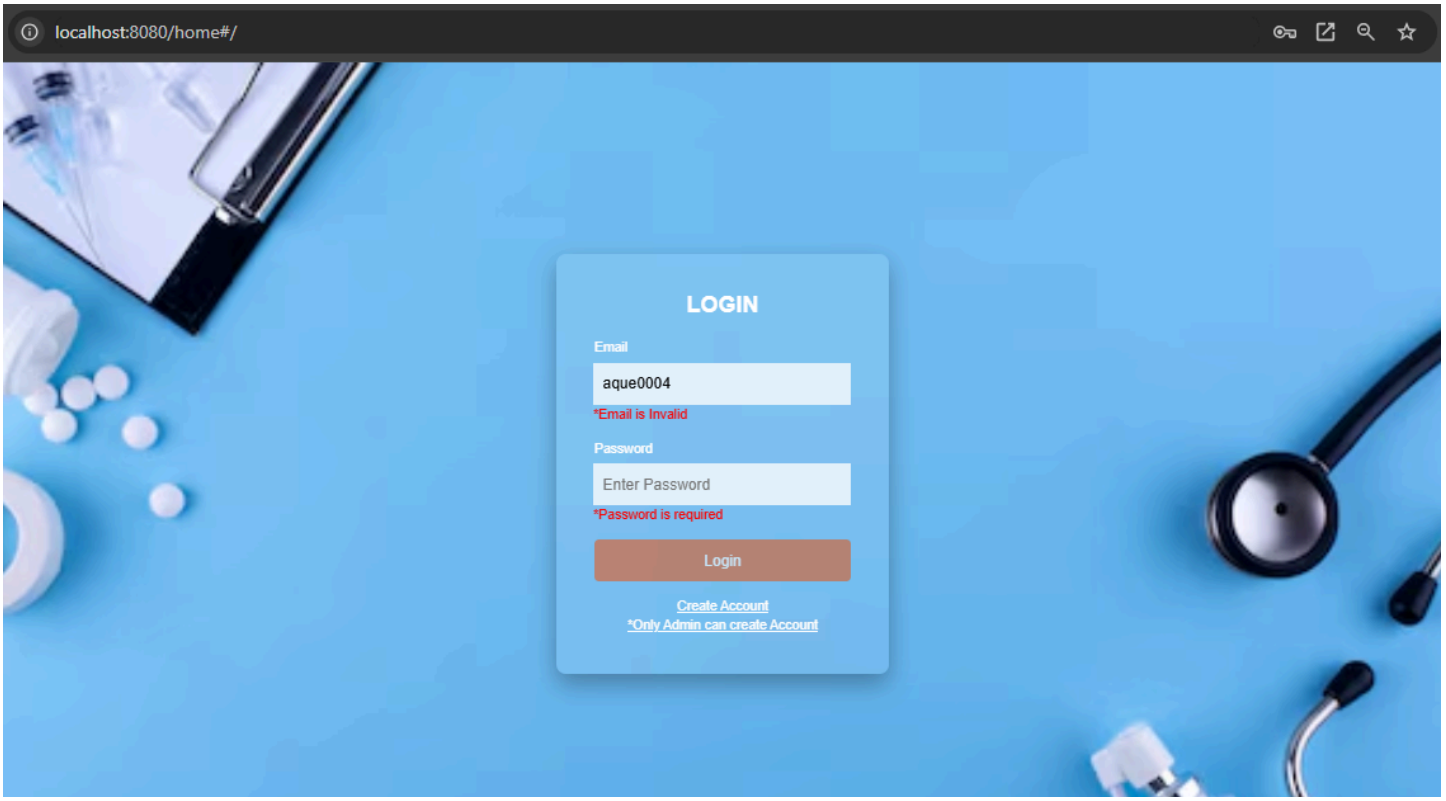


Figure 5.2.1-1 Login Page Input Validation

In the system backend, each component function is equipped with input validation, error handling, and exception management mechanisms to ensure the application can recover gracefully from potential failures. This approach enables the system to detect and manage errors without disrupting overall functionality. Additionally, the database incorporates schema validation within its data models to prevent data errors, ensuring completeness, accuracy, and consistency across all records.

This layered validation strategy strengthens the system's robustness by minimizing the risk of faulty data and enhancing its resilience against unexpected inputs or issues.

5.2.2. Security

As the AHIS handles patients' personal details and health information, ensuring data privacy and security standards are met is vital due to the sensitive nature of health data and the strict regulatory requirements surrounding its protection. The AHIS addresses security adequately by implementing sufficient layers of defense, including data encryption, user authentication and role-based access controls, to prevent unauthorized access and data breaches.

Data encryption is used to ensure that crucial information remains protected and safeguarded against unwanted security attack, with the system using Bcrypt library to implement one-way encryption, as detailed in section 3.4.4. on using bcrypt in user authentication. User authentication is used to verify user identity to reduce the risk of unauthorized access with role-based access controls limiting access to specific components based on user roles, allowing only authorized personnel to perform certain operations.

Nonetheless, there are opportunities to enhance security, such as conducting regular security audits and ensuring compliance with evolving regulatory requirements. These measures highlight the need for ongoing monitoring and updates to maintain a high standard of security and adapt to emerging threats.

5.2.3. Usability

The AHIS web application has been carefully designed in alignment with key usability principles, including Nielsen's 10 heuristics for user interface design (PB_u247-Gem, 2024). This approach provides an intuitive experience, specifically tailored for healthcare professionals.

Heuristic Rule	Web Design Choice	Justification	Evidence
Visibility of System Status	The application provides immediate feedback via error messages during data entry, as well as authentication confirmation on login.	Ensures continuous user awareness of system responses, enabling prompt action to correct errors, such as incorrect login or invalid data.	See Appendix F – Figure 5.2.3-1 Error message shown for invalid input in patient form.
Match Between System and Real World	The web system utilizes familiar healthcare terminology for fields and labels, such as	Using real-world terminology helps users quickly identify the purpose of each section	See Appendix F – Figure 5.2.3-2

	"Patient Profile," "Diagnosis," and "Medication."	and field, ensuring that the interface aligns with users' mental models based on their professional experience in healthcare settings.	Main page displaying labeled cards for "Physicians," "Medication," and "Physicians".
Consistency and Standards	A uniform colour scheme (blue), serif font style, and standardised layout across sections, including common buttons like "View," "Update," and "Delete" for each profile management page.	Consistency in visual and interactive elements allows users to predict behaviours across the web application. For instance, standardised action buttons for managing profiles support ease of navigation and reduce the learning curve..	See Appendix F – Figure 5.2.3-3 Standardized layout in Patient and Physician Profile pages with "View," "Update," and "Delete" buttons.
Error Prevention	Patient registration form fields are validated to accept only appropriate characters (e.g., numeric values for phone and ID numbers).	This feature prevents input errors by guiding users to enter the correct data format, reducing the risk of errors in critical fields and improving the accuracy of stored patient data. Immediate error alerts inform users of entry requirements, enabling efficient corrections.	See Appendix F – Figure 5.2.3-1 Error message shown for invalid input in patient form.
Flexibility and Efficiency of Use	Users can seamlessly transition between pages, such as moving from appointment scheduling to consultation, diagnosis, and prescription stages.	This design enhances workflow efficiency by minimising steps needed to complete routine processes. The application's layout groups related information under each function, allowing	See Section 4.3.4.4.1. Blackbox Testing

		healthcare professionals to manage data entry swiftly and accurately, as verified by black box testing in Section 4.3.4.4.1.	
Aesthetic and Minimalist Design	The overall web application maintains a minimalist approach, displaying only essential information and eliminating non-essential elements ("chartjunk").	A clean, uncluttered interface ensures that users can concentrate on core tasks without distraction, aligning with the minimalist heuristic.	See Appendix F – Figure 5.2.3-4 Minimalist main page layout.
Help Users Recognize, Diagnose, and Recover from Errors	Error messages are provided to guide users in diagnosing and correcting issues, such as incorrect email/password.	Descriptive messages support error resolution by specifying the nature of the issue, whether it's incorrect login credentials.	See Appendix F – Figure 5.2.3-5 Error message for incorrect login credentials.

Table 5.2.3-1. Evaluation of AHIS web system's alignment with Nielsen's heuristics.

The AHIS web application effectively applies Nielsen's heuristics to create a dynamic, user-centered interface. By prioritizing visibility, consistency, error prevention, and simplicity, it enables healthcare professionals to perform data entry and patient management tasks with greater ease. Future enhancements such as adding a search bar and filtering options are identified to further improve functionality as the system expands.

5.2.4. Scalability

The AHIS web application was created with scalability in mind that allows the system to grow over time to support more users, medical records and features. The web application created with Angular and Node.js architecture facilitates effective and modular scaling. The selected database, MongoDB, is well-suited to managing huge datasets and has the capability of horizontal scaling which enables data to be split among several servers to enhance performance as usage increases.

API caching, asynchronous data processing and improved backend efficiency and resource management all help to lower server load during periods of high traffic. Furthermore, the AHIS can manage an increasing amount of data thanks to MongoDB's native support for distributed database systems which is crucial for a health information system that will eventually handle an increasing

number of patients, doctors and appointments. Implementing load balancers and making use of cloud-based infrastructure to effectively increase server capacity are examples of potential future improvements.

The HTR and ICROP models which are essential to the AHIS's patient registration module, are also subject to scalability concerns as the system develops. These models may be enhanced with batch processing to handle bigger input datasets without losing speed or quality that also ensures performance and accuracy as usage increases. The AHIS is ready to meet the demands of a growing healthcare environment by anticipating expansion in both application capabilities and AI functionality.

5.2.5. Portability

The AHIS's portability is essential as it allows it to be used on a variety of devices and operating systems. The AHIS is a progressive web application that works on desktop and mobile platforms that gives healthcare professionals access to patient data from a variety of devices. The system was developed to work with the latest versions of the main web browsers that guarantees accessibility across a range of devices without losing functionality.

The AHIS is compatible with Linux, Windows and MacOS systems thanks to the use of Node.js and Angular which further improve portability and facilitate cross-platform deployment. The codebase also incorporates responsive design ideas and adheres to web standards which enable it to adapt smoothly to various screen sizes and resolutions. Potential interaction with cloud-based services would enable healthcare organizations to access and manage the AHIS from geographically distant places for more reliable portability in future versions.

Furthermore, the AHIS architecture is made to be flexible enough to accommodate many deployment scenarios. It may be installed on-site for organizations with specific data protection needs, and it may eventually be deployed on cloud platforms. Because of this flexibility, institutions can select a deployment strategy that best suits their operational and regulatory requirements. The AHIS's emphasis on portability guarantees that medical professionals can rely on the system regardless of their preferred technological configuration.

5.2.6. Maintainability

The long term maintainability of the AHIS system guarantees that upgrades, debugging and extension are simple. The codebase complies to industry-standard development techniques such as modularization which lowers the possibility of system wide problems during maintenance by allowing components to be updated or debugged individually. The solution makes use of Node.js's modular design and Angular's component-based architecture to increase maintainability by streamlining updates and making the codebase easier to navigate.

Technical and end user aspects are covered in detailed documentation which helps developers understand the structure of the system and assists effective troubleshooting. Git facilitates the

implementation of version control procedures for continuous maintainability that enables tracking of code modifications and simple rollback in the event of any issues. In the future, automated testing and deployment can be made possible by a CI/CD pipeline that would further improve maintainability by guaranteeing that updates are integrated smoothly without compromising system stability.

The modular architecture of the ICROP and HTR models also makes it possible to make isolated updates and changes without needing to modify the entire system. This configuration minimizes development overhead while allowing for gradual modifications to individual components that guarantees that the models maintain their accuracy and effectiveness over time. The AHIS can adjust to changing healthcare needs with little interference to system functionality thanks to its emphasis on flexibility across application levels.

5.3. Sample Source Code

5.3.1. AHIS Web Application

AHIS Web Application Frontend Repository		
Description	This repository contains code for the frontend web application. Each component contains functions that are dependent on the HttpClient to make HTTP requests via RESTful API to connect to the backend.	
Components	Angular Components	Each component has its own html and css files responsible for defining the style, content and structure of web content. Each component contains a typescript file that consists of the component logic. The components include appointment, consultation, diagnosis, homepage, login, medication, page not found, patient, physician, prescription and register.
	Angular pipes	Angular pipes are functions that are used to transform data in Angular templates to desired format. These functions take input data values, transform/process them and return a single output value. The pipes function include calculate age, date format, datetime format and uppercase format.
	Angular Database Service	An object that gets instantiated and contains methods that maintain data throughout the life of an application. The main objective of a service is to organize and share component logic, models, data and functions with different components of an

		Angular application. Every component that passes data to the backend uses the HTTP request functions defined in the database service.
Github Link	https://git.infotech.monash.edu/mds2_ahis_2024/automated-health-information-system/-/tree/main/MDS2_AHIS/src?ref_type=heads	

Figure 5.3.1-1. Overview of Frontend Web Application Repository - Source Code Components

AHIS Web Application Backend Repository		
Description	This repository contains code for the backend web application, with database data model schema design.	
Components	Controller javascript files	Each controller contains a set of functions that perform operations such as inserting, listing, deleting, and updating records. There are controllers for appointment, consultation, diagnosis, authentication, user role, medication, patient, physician and prescription.
	Models javascript files (Database Schema)	Each model defines a component schema with specified fields, data type, weather its mandatory and optional default value. There are models for appointment, consultation, diagnosis, user, role, medication, patient, physician and prescription.
	Routes javascript files	Configured RESTFul Endpoints for each respective component.
	Server.js file	The main server script for the web application that primarily sets up the server, connects the MongoDB database and defines key functionalities for handling API requests, file uploads, and socket connections for real-time interactions.
Github Link	https://git.infotech.monash.edu/mds2_ahis_2024/automated-health-information-system/-/tree/main/MDS2_AHIS?ref_type=heads	

Figure 5.3.1-2. Overview of Backend Web Application Repository - Source Code Components

5.3.2. Image Cropping (ICROP) Model

ICROP Model Repository

Description	This repository contains code for the ICROP Model, which automates the extraction of key data fields from patient registration forms by detecting and cropping specific sections, such as patient name, age, and address. The model utilises OpenCV for image processing and customised algorithms to ensure accuracy in data extraction.	
Components	ICROP Model	A self-implemented model coded from scratch in **AHIS_CropModel1.py**, designed to automatically extract specific fields from patient registration forms.
	Coordinates detector	Implemented in **selectCoordinate.py**, this tool detects and records the coordinates of each field for precise cropping.
	Customised Patient Registration Forms	Standardised patient registration forms with consistent field layouts, enhancing accuracy and compatibility with the ICROP Model.
Github Link	https://github.com/EuniceLEEWJ/MDS02	

Table 5.3.2-1. Description of source code components in ICROP Model repository

5.3.3. Handwritten Text Recognition (HTR) Model

HTR Model Repository		
Description	This repository contains the source code for the HT) Model designed to convert handwritten text from scanned images into digital text. Leveraging a deep learning architecture, this model uses a combination of CNNs and RNNs with a CTC layer for transcription that streamlines the transformation of handwritten medical records into structured electronic text.	
Components	main.py	Primary script for executing the HTR model, handling tasks such as training, validation and inference. This file serves as the main entry point, with modes for both training and deploying the model.
	model.py	Defines the architecture of the neural network, including CNN, RNN and CTC layers by providing the core functionality for text recognition by defining layers and their connections.

	preprocessor.py	Contains pre-processing functions to prepare image data for model input including resizing, normalization and any other required transformations for optimizing the data for accurate recognition.
	dataloader_iam.py	Manages the loading and batching of the IAM dataset, a handwriting dataset widely used in research ensuring efficient data management and model-ready input.
	create_lmdb.py	Facilitates the creation of an LMDB (Lightning Memory-Mapped Database) for storing image-label pairs enabling faster data retrieval during model training and evaluation.
Github Link	https://git.infotech.monash.edu/mds2_ahis_2024/automated-health-information-system/-/tree/master/HTR_Model/SimpleHTR-master/src	

Table 5.3.3-1. Description of source code components in HTR Model repository

6. Software and Project Critique

6.1. Project Overall Execution

By creating a web system that improved healthcare data administration by utilizing machine learning, the AHIS project successfully met its main goals. The project's goal was to create a web-based health information system with automated data entry features, particularly by integrating the HTR and ICROP model into the web application.

These vital components have been successfully integrated into the project to streamline patient record data entry. Although mobile integration was initially planned, the team decided to prioritise optimising the web-based health information system and its data entry functionality. By preprocessing fields in patient registration forms, the ICROP model increased the accuracy of text extraction, while the integration of the HTR model automates the entry of patient data into the database.

6.2. Project Outcome VS Initial Proposal

While the AHIS project met the majority of its objectives, some elements deviated from the original proposal due to practical constraints. The initial plan was to develop a fully custom HTR model from scratch. However, due to time and resource limitations, we decided to integrate a pre-built open source HTR model. This adjustment also led to the integration of the ICROP model, which improved text extraction accuracy, providing an efficient and practical solution that aligned with the project's constraints.

Initially, the project also proposed incorporating HTR functionality in the diagnosis and medical prescription sections to capture handwritten information during consultations. Nevertheless, due to the inconsistencies and readability problems commonly seen in physicians' handwriting, which can be hard to understand even for non-medical professionals, we concluded that this tool was not feasible within the project's parameters. As a result, we introduced HTR capabilities exclusively in the patient registration section, as the handwritten data was more uniform and easier for the system to handle.

Additionally, as previously mentioned in Section 6.1, the team initially included plans for mobile integration for the project but due to limited resources and time constraints the team has chosen to prioritise enhancing the main web system over mobile development. Moreover, healthcare professionals typically access healthcare systems through laptops and desktops rather than mobile devices, which supported our decision to focus on the web interface. This approach ensured that the core functionality of the web system was robust, though mobile integration remains a viable enhancement for future development.

Finally, some features, like real-time data sharing via cloud-based deployment, could not be implemented due to limited infrastructure. However, these functionalities are intended for future development to enhance the project's capabilities and support its long-term vision.

6.3. Critical Discussion

The AHIS project management approach utilised an Agile-Waterfall hybrid methodology, which allowed for structured phases of development while incorporating flexibility to address unforeseen challenges. This approach enabled us to clearly define objectives, establish workflows, and assign responsibilities early on, while still allowing for iterative adjustments as challenges emerged. Regular team meetings and progress reviews from our supervisor kept us aligned with our goals and enabled us to overcome technical and resource challenges. In our opinion, the project's time limitations were not caused by poor time management but resulted from the wide project scope and the sharp learning curve required. With no prior experience in building an HTR model using TensorFlow and limited working experience with web development frameworks, the team faced the dual challenge of learning and implementing these components from scratch within a short time frame. As a result, a large portion of our time was spent on acquiring the basic knowledge needed to develop the HTR and ICROP models and developing the web application.

In terms of project outcome, the final product met the primary goals of automating patient data entry, implementing an automated data entry model which is the HTR model for text extraction. While the web application is highly functional, some areas could benefit from further refinement to enhance usability as mentioned in Section 3.7. A particular success was the patient registration form's images parsing functionality, which allows images to go through ICROP and then the HTR model, effectively autofilling patient information. Despite the adjustments, the core outcome aligns

well with our goal of streamlining data entry in healthcare settings, and we believe the project has established a robust foundation that addresses practical needs.

Reflecting on the project's challenges, we feel a sense of accomplishment, as adapting our approach allowed us to deliver a valuable solution despite the setbacks. However, the deviations from our initial plan highlighted the limitations of resource allocation and emphasised the need for flexibility in project management. The experience reinforced our understanding of balancing ambition with practical limitations, and while the project was challenging, it ultimately enhanced our skills in problem-solving, collaboration, and adaptability in the face of setbacks.

7. Conclusion

The AHIS project sought to revolutionise patient data management in healthcare by creating an efficient, web-based application designed to automate patient data entry. By leveraging machine learning models, AHIS enhances traditional data entry processes, delivering a streamlined and highly functional tool for healthcare professionals. This project centred on key objectives such as developing a scalable health information system, integrating a HTR model for processing handwritten patient forms, and employing an ICROP model to ensure precision in data extraction.

Using an Agile-Waterfall methodology allowed the team to maintain structured development phases while iteratively adapting to the technical challenges inherent in this project's scope. Building foundational skills in web development and machine learning, the team successfully delivered a prototype that meets essential goals, including automated patient registration and effective data management. Although certain components, such as a fully self-implemented HTR model and mobile integration, fell outside the scope due to time constraints, the project's outcomes align strongly with our original vision and present clear opportunities for future enhancement.

This report also highlights both our project's achievements and growth areas. The current project prototype effectively meets primary functional requirements, providing a foundation that can support further improvements in system customization, scalability and cloud-based accessibility. This experience has fostered essential skills in problem-solving, adaptive project management and collaborative development, equipping the team with valuable expertise for future endeavours in health tech innovation.

Lastly, we extend our sincere appreciation to our supervisor, Dr. Muhammad Fermi Pasha, for his guidance and expertise throughout this project. His insights and mentorship were pivotal in navigating challenges and achieving our project goals, ensuring a meaningful and rewarding development journey. We also wish to express our gratitude to Mr. Soo Wooi King, our final-year project unit coordinator for his constant support, prompt answers to our questions and his friendly, encouraging presence. His warmth and approachability made a significant difference in our project experience providing reassurance and clarity during crucial stages of our work.

8. Appendix

Appendix A - HTR Model Source Code

main.py

```python
import argparse
import json
from typing import Tuple, List

import cv2
import editdistance
from path import Path

from dataloader_iam import DataLoaderIAM, Batch
from model import Model, DecoderType
from preprocessor import Preprocessor
import sys
import os
from dataclasses import dataclass , field, asdict
import AHIS_CropModel1 as crop_model

class FilePaths:
    """Filenames and paths to data."""
    fn_char_list = '../model/charList.txt'
    fn_summary = '../model/summary.json'
    fn_corpus = '../data/corpus.txt'


def get_img_height() -> int:
    """Fixed height for NN."""
    return 32


def get_img_size(line_mode: bool = False) -> Tuple[int, int]:
    """Height is fixed for NN, width is set according to training mode (single words or text lines)."""
    if line_mode:
        return 256, get_img_height()
    return 128, get_img_height()


def write_summary(average_train_loss: List[float], char_error_rates: List[float], word_accuracies: List[float]) -> None:
    """Writes training summary file for NN."""
    with open(FilePaths.fn_summary, 'w') as f:
        json.dump({'averageTrainLoss': average_train_loss, 'charErrorRates': char_error_rates, 'wordAccuracies': word_accuracies}, f)


def char_list_from_file() -> List[str]:
    with open(FilePaths.fn_char_list) as f:
        return list(f.read())
```

```
46
47  def train(model: Model,
48            loader: DataLoaderIAM,
49            line_mode: bool,
50            early_stopping: int = 25) -> None:
51      """Trains NN."""
52      epoch = 0  # number of training epochs since start
53      summary_char_error_rates = []
54      summary_word_accuracies = []
55
56      train_loss_in_epoch = []
57      average_train_loss = []
58
59      preprocessor = Preprocessor(get_img_size(line_mode), data_augmentation=True, line_mode=line_mode)
60      best_char_error_rate = float('inf')  # best validation character error rate
61      no_improvement_since = 0  # number of epochs no improvement of character error rate occurred
62      # stop training after this number of epochs without improvement
63      while True:
64          epoch += 1
65          print('Epoch:', epoch)
66
67          # train
68          print('Train NN')
69          loader.train_set()
70          while loader.has_next():
71              iter_info = loader.get_iterator_info()
72              batch = loader.get_next()
73              batch = preprocessor.process_batch(batch)
74              loss = model.train_batch(batch)
75              print(f'Epoch: {epoch} Batch: {iter_info[0]}/{iter_info[1]} Loss: {loss}')
76              train_loss_in_epoch.append(loss)
77
78          # validate
79          char_error_rate, word_accuracy = validate(model, loader, line_mode)
80
81          # write summary
82          summary_char_error_rates.append(char_error_rate)
83          summary_word_accuracies.append(word_accuracy)
84          average_train_loss.append((sum(train_loss_in_epoch)) / len(train_loss_in_epoch))
85          write_summary(average_train_loss, summary_char_error_rates, summary_word_accuracies)
86
87          # reset train loss list
88          train_loss_in_epoch = []
89
```

```
89
90          # if best validation accuracy so far, save model parameters
91          if char_error_rate < best_char_error_rate:
92              print('Character error rate improved, save model')
93              best_char_error_rate = char_error_rate
94              no_improvement_since = 0
95              model.save()
96          else:
97              print(f'Character error rate not improved, best so far: {best_char_error_rate * 100.0}%')
98              no_improvement_since += 1
99
100         # stop training if no more improvement in the last x epochs
101         if no_improvement_since >= early_stopping:
102             print(f'No more improvement for {early_stopping} epochs. Training stopped.')
103             break
104
105
106 def validate(model: Model, loader: DataLoaderIAM, line_mode: bool) -> Tuple[float, float]:
107     """Validates NN."""
108     print('Validate NN')
109     loader.validation_set()
110     preprocessor = Preprocessor(get_img_size(line_mode), line_mode=line_mode)
111     num_char_err = 0
112     num_char_total = 0
113     num_word_ok = 0
114     num_word_total = 0
115     while loader.has_next():
116         iter_info = loader.get_iterator_info()
117         print(f'Batch: {iter_info[0]} / {iter_info[1]}')
118         batch = loader.get_next()
119         batch = preprocessor.process_batch(batch)
120         recognized, _ = model.infer_batch(batch)
121
122         print('Ground truth -> Recognized')
123         for i in range(len(recognized)):
124             num_word_ok += 1 if batch.gt_texts[i] == recognized[i] else 0
125             num_word_total += 1
126             dist = editdistance.eval(recognized[i], batch.gt_texts[i])
127             num_char_err += dist
128             num_char_total += len(batch.gt_texts[i])
129             print('[OK]' if dist == 0 else '[ERR:%d]' % dist, '"' + batch.gt_texts[i] + '"', '->',
130                   '"' + recognized[i] + '"')
131
```

```
132     # print validation result
133     char_error_rate = num_char_err / num_char_total
134     word_accuracy = num_word_ok / num_word_total
135     print(f'Character error rate: {char_error_rate * 100.0}%. Word accuracy: {word_accuracy * 100.0}%.')
136     return char_error_rate, word_accuracy
137
138
139 def infer(model: Model, fn_img: Path) -> None:
140     """Recognizes text in image provided by file path."""
141     img = cv2.imread(fn_img, cv2.IMREAD_GRAYSCALE)
142     assert img is not None
143
144     preprocessor = Preprocessor(get_img_size(), dynamic_width=True, padding=16)
145     img = preprocessor.process_img(img)
146
147     batch = Batch([img], None, 1)
148     recognized, probability = model.infer_batch(batch, True)
149     stored_value = recognized[0]
150     print(f'Recognized: "{recognized[0]}"')
151     print(f'Probability: {probability[0]}')
152
153     return stored_value
154
155
156 def parse_args() -> argparse.Namespace:
157     """Parses arguments from the command line."""
158     parser = argparse.ArgumentParser()
159
160     parser.add_argument('--mode', choices=['train', 'validate', 'infer'], default='infer')
161     parser.add_argument('--decoder', choices=['bestpath', 'beamsearch', 'wordbeamsearch'], default='bestpath')
162     parser.add_argument('--batch_size', help='Batch size.', type=int, default=100)
163     parser.add_argument('--data_dir', help='Directory containing IAM dataset.', type=Path, required=False)
164     parser.add_argument('--fast', help='Load samples from LMDB.', action='store_true')
165     parser.add_argument('--line_mode', help='Train to read text lines instead of single words.', action='store_true')
166     parser.add_argument('--img_file', help='Image used for inference.', type=Path, default='../data/word.png')
167     parser.add_argument('--early_stopping', help='Early stopping epochs.', type=int, default=25)
168     parser.add_argument('--dump', help='Dump output of NN to CSV file(s).', action='store_true')
169
170     return parser.parse_args()
171
172
173 def main():
```

```
    # parse arguments and set CTC decoder
    args = parse_args()
    decoder_mapping = {'bestpath': DecoderType.BestPath,
                       'beamsearch': DecoderType.BeamSearch,
                       'wordbeamsearch': DecoderType.WordBeamSearch}
    decoder_type = decoder_mapping[args.decoder]

    # train the model
    if args.mode == 'train':
        loader = DataLoaderIAM(args.data_dir, args.batch_size, fast=args.fast)

        # when in line mode, take care to have a whitespace in the char list
        char_list = loader.char_list
        if args.line_mode and ' ' not in char_list:
            char_list = [' '] + char_list

        # save characters and words
        with open(FilePaths.fn_char_list, 'w') as f:
            f.write(''.join(char_list))

        with open(FilePaths.fn_corpus, 'w') as f:
            f.write(' '.join(loader.train_words + loader.validation_words))

        model = Model(char_list, decoder_type)
        train(model, loader, line_mode=args.line_mode, early_stopping=args.early_stopping)

    # evaluate it on the validation set
    elif args.mode == 'validate':
        loader = DataLoaderIAM(args.data_dir, args.batch_size, fast=args.fast)
        model = Model(char_list_from_file(), decoder_type, must_restore=True)
        validate(model, loader, args.line_mode)

    # infer text on test image
    elif args.mode == 'infer':
        model = Model(char_list_from_file(), decoder_type, must_restore=True, dump=args.dump)
        recognized = infer(model, args.img_file)
        print(recognized)
        return recognized

    # infer text on test image
    elif args.mode == 'infer2':

        @dataclass
```

```
        class Patient():
            firstName: str
            lastName: str
            birthDate: str
            gender: str
            nationality: str
            ICPassport: str
            maritalStatus: str
            streetAddress: str
            city: str
            state: str
            postalCode: str
            email: str
            tel: str
            emgcyFname: str
            emgcyLname: str
            emgcyRelationship: str
            emgcyTel: str

        data = [0] * 17
        i=0

        # Patient.name = "test"

        print('entered my infer')
        model = Model(char_list_from_file(), decoder_type, must_restore=True, dump=args.dump)
        # infer(model, args.img_file)


        # Directory where the cropped images are stored
        image_dir = '..\data\Finalcrop2'

        # Get all PNG files in the directory
        image_files = [f for f in os.listdir(image_dir) if f.endswith('.jpg')]

        # Loop over each image file and run main()
        for img_file in image_files:
            path = os.path.join(image_dir, img_file)
            recognized = infer(model, path)
            data[i] = recognized
            i += 1
            # print(recognized)
```

```
173  def main():
261      
262          patient = Patient(
263              streetAddress = data[0],
264              birthDate = data[1],
265              city = data[2],
266              emgcyTel = data[3],
267              emgcyFname = data[4],
268              emgcyLname = data[5],
269              email = data[6],
270              firstName = data[7],
271              gender = data[8],
272              ICPassport = data[9],
273              lastName = data[10],
274              maritalStatus = data[11],
275              nationality = data[12],
276              tel = data[13],
277              emgcyRelationship = data[14],
278              state = data[15],
279              postalCode = data[16]
280          )
281  
282          # print(patient)
283          # Convert the patient object to a dictionary
284          patient_dict = asdict(patient)
285  
286          # Print the dictionary as a JSON string
287          #print(json.dumps(patient_dict, indent=4))
288  
289          # Define the output file path
290          output_file = os.path.join("..", "data", "patient_data.json")
291  
292          # Write the dictionary to a JSON file
293          with open(output_file, 'w') as f:
294              json.dump(patient_dict, f, indent=4)
295  
296          print(f"Patient data successfully written to {output_file}")
297          return patient
298  
299  
300  
301  if __name__ == '__main__':
302      # crop_model.crop()
303      main()
```

Figure A-1. HTR Model main.py Code Snippet

model.py

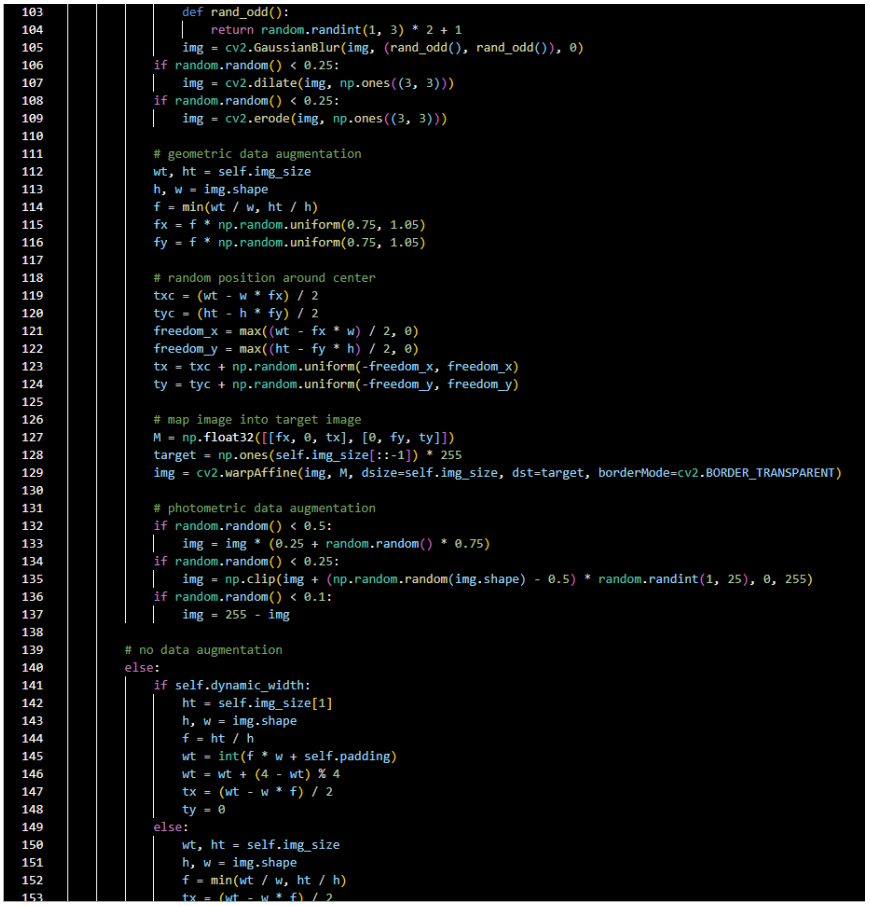
```python
import random
from typing import Tuple

import cv2
import numpy as np

from dataloader_iam import Batch


class Preprocessor:
    def __init__(self,
                 img_size: Tuple[int, int],
                 padding: int = 0,
                 dynamic_width: bool = False,
                 data_augmentation: bool = False,
                 line_mode: bool = False) -> None:
        # dynamic width only supported when no data augmentation happens
        assert not (dynamic_width and data_augmentation)
        # when padding is on, we need dynamic width enabled
        assert not (padding > 0 and not dynamic_width)

        self.img_size = img_size
        self.padding = padding
        self.dynamic_width = dynamic_width
        self.data_augmentation = data_augmentation
        self.line_mode = line_mode

    @staticmethod
    def _truncate_label(text: str, max_text_len: int) -> str:
        """
        Function ctc_loss can't compute loss if it cannot find a mapping between text label and input
        labels. Repeat letters cost double because of the blank symbol needing to be inserted.
        If a too-long label is provided, ctc_loss returns an infinite gradient.
        """
        cost = 0
        for i in range(len(text)):
            if i != 0 and text[i] == text[i - 1]:
                cost += 2
            else:
                cost += 1
            if cost > max_text_len:
                return text[:i]
        return text

    def _simulate_text_line(self, batch: Batch) -> Batch:
        """Create image of a text line by pasting multiple word images into an image."""

        default_word_sep = 30
        default_num_words = 5

        # go over all batch elements
        res_imgs = []
        res_gt_texts = []
        for i in range(batch.batch_size):
            # number of words to put into current line
            num_words = random.randint(1, 8) if self.data_augmentation else default_num_words

            # concat ground truth texts
            curr_gt = ' '.join([batch.gt_texts[(i + j) % batch.batch_size] for j in range(num_words)])
            res_gt_texts.append(curr_gt)

            # put selected word images into list, compute target image size
            sel_imgs = []
            word_seps = [0]
            h = 0
            w = 0
            for j in range(num_words):
                curr_sel_img = batch.imgs[(i + j) % batch.batch_size]
                curr_word_sep = random.randint(20, 50) if self.data_augmentation else default_word_sep
                h = max(h, curr_sel_img.shape[0])
                w += curr_sel_img.shape[1]
                sel_imgs.append(curr_sel_img)
                if j + 1 < num_words:
                    w += curr_word_sep
                    word_seps.append(curr_word_sep)

            # put all selected word images into target image
            target = np.ones([h, w], np.uint8) * 255
            x = 0
            for curr_sel_img, curr_word_sep in zip(sel_imgs, word_seps):
                x += curr_word_sep
                y = (h - curr_sel_img.shape[0]) // 2
                target[y:y + curr_sel_img.shape[0]:, x:x + curr_sel_img.shape[1]] = curr_sel_img
                x += curr_sel_img.shape[1]

            # put image of line into result
            res_imgs.append(target)

        return Batch(res_imgs, res_gt_texts, batch.batch_size)

    def process_img(self, img: np.ndarray) -> np.ndarray:
        """Resize to target size, apply data augmentation."""

        # there are damaged files in IAM dataset - just use black image instead
        if img is None:
            img = np.zeros(self.img_size[::-1])

        # data augmentation
        img = img.astype(np.float)
        if self.data_augmentation:
            # photometric data augmentation
            if random.random() < 0.25:
```

```python
                def rand_odd():
                    return random.randint(1, 3) * 2 + 1
                img = cv2.GaussianBlur(img, (rand_odd(), rand_odd()), 0)
            if random.random() < 0.25:
                img = cv2.dilate(img, np.ones((3, 3)))
            if random.random() < 0.25:
                img = cv2.erode(img, np.ones((3, 3)))

            # geometric data augmentation
            wt, ht = self.img_size
            h, w = img.shape
            f = min(wt / w, ht / h)
            fx = f * np.random.uniform(0.75, 1.05)
            fy = f * np.random.uniform(0.75, 1.05)

            # random position around center
            txc = (wt - w * fx) / 2
            tyc = (ht - h * fy) / 2
            freedom_x = max((wt - fx * w) / 2, 0)
            freedom_y = max((ht - fy * h) / 2, 0)
            tx = txc + np.random.uniform(-freedom_x, freedom_x)
            ty = tyc + np.random.uniform(-freedom_y, freedom_y)

            # map image into target image
            M = np.float32([[fx, 0, tx], [0, fy, ty]])
            target = np.ones(self.img_size[::-1]) * 255
            img = cv2.warpAffine(img, M, dsize=self.img_size, dst=target, borderMode=cv2.BORDER_TRANSPARENT)

            # photometric data augmentation
            if random.random() < 0.5:
                img = img * (0.25 + random.random() * 0.75)
            if random.random() < 0.25:
                img = np.clip(img + (np.random.random(img.shape) - 0.5) * random.randint(1, 25), 0, 255)
            if random.random() < 0.1:
                img = 255 - img

        # no data augmentation
        else:
            if self.dynamic_width:
                ht = self.img_size[1]
                h, w = img.shape
                f = ht / h
                wt = int(f * w + self.padding)
                wt = wt + (4 - wt) % 4
                tx = (wt - w * f) / 2
                ty = 0
            else:
                wt, ht = self.img_size
                h, w = img.shape
                f = min(wt / w, ht / h)
                tx = (wt - w * f) / 2
                ty = (ht - h * f) / 2

            # map image into target image
            M = np.float32([[f, 0, tx], [0, f, ty]])
            target = np.ones([ht, wt]) * 255
            img = cv2.warpAffine(img, M, dsize=(wt, ht), dst=target, borderMode=cv2.BORDER_TRANSPARENT)

        # transpose for TF
        img = cv2.transpose(img)

        # convert to range [-1, 1]
        img = img / 255 - 0.5
        return img

    def process_batch(self, batch: Batch) -> Batch:
        if self.line_mode:
            batch = self._simulate_text_line(batch)

        res_imgs = [self.process_img(img) for img in batch.imgs]
        max_text_len = res_imgs[0].shape[0] // 4
        res_gt_texts = [self._truncate_label(gt_text, max_text_len) for gt_text in batch.gt_texts]
        return Batch(res_imgs, res_gt_texts, batch.batch_size)


def main():
    import matplotlib.pyplot as plt

    img = cv2.imread('../data/test.png', cv2.IMREAD_GRAYSCALE)
    img_aug = Preprocessor((256, 32), data_augmentation=True).process_img(img)
    plt.subplot(121)
    plt.imshow(img, cmap='gray')
    plt.subplot(122)
    plt.imshow(cv2.transpose(img_aug) + 0.5, cmap='gray', vmin=0, vmax=1)
    plt.show()


if __name__ == '__main__':
    main()
```

Figure A-2. HTR Model model.py Code Snippet

preprocessor.py

```python
import random
from typing import Tuple

import cv2
import numpy as np

from dataloader_iam import Batch


class Preprocessor:
    def __init__(self,
                 img_size: Tuple[int, int],
                 padding: int = 0,
                 dynamic_width: bool = False,
                 data_augmentation: bool = False,
                 line_mode: bool = False) -> None:
        # dynamic width only supported when no data augmentation happens
        assert not (dynamic_width and data_augmentation)
        # when padding is on, we need dynamic width enabled
        assert not (padding > 0 and not dynamic_width)

        self.img_size = img_size
        self.padding = padding
        self.dynamic_width = dynamic_width
        self.data_augmentation = data_augmentation
        self.line_mode = line_mode

    @staticmethod
    def _truncate_label(text: str, max_text_len: int) -> str:
        """
        Function ctc_loss can't compute loss if it cannot find a mapping between text label and input
        labels. Repeat letters cost double because of the blank symbol needing to be inserted.
        If a too-long label is provided, ctc_loss returns an infinite gradient.
        """
        cost = 0
        for i in range(len(text)):
            if i != 0 and text[i] == text[i - 1]:
                cost += 2
            else:
                cost += 1
            if cost > max_text_len:
                return text[:i]
        return text

    def _simulate_text_line(self, batch: Batch) -> Batch:
        """Create image of a text line by pasting multiple word images into an image."""

        default_word_sep = 30
        default_num_words = 5

        # go over all batch elements
        res_imgs = []
        res_gt_texts = []
        for i in range(batch.batch_size):
            # number of words to put into current line
            num_words = random.randint(1, 8) if self.data_augmentation else default_num_words

            # concat ground truth texts
            curr_gt = ' '.join([batch.gt_texts[(i + j) % batch.batch_size] for j in range(num_words)])
            res_gt_texts.append(curr_gt)

            # put selected word images into list, compute target image size
            sel_imgs = []
            word_seps = [0]
            h = 0
            w = 0
            for j in range(num_words):
                curr_sel_img = batch.imgs[(i + j) % batch.batch_size]
                curr_word_sep = random.randint(20, 50) if self.data_augmentation else default_word_sep
                h = max(h, curr_sel_img.shape[0])
                w += curr_sel_img.shape[1]
                sel_imgs.append(curr_sel_img)
                if j + 1 < num_words:
                    w += curr_word_sep
                    word_seps.append(curr_word_sep)

            # put all selected word images into target image
            target = np.ones([h, w], np.uint8) * 255
            x = 0
            for curr_sel_img, curr_word_sep in zip(sel_imgs, word_seps):
                x += curr_word_sep
                y = (h - curr_sel_img.shape[0]) // 2
                target[y:y + curr_sel_img.shape[0]:, x:x + curr_sel_img.shape[1]] = curr_sel_img
                x += curr_sel_img.shape[1]

            # put image of line into result
            res_imgs.append(target)

        return Batch(res_imgs, res_gt_texts, batch.batch_size)

    def process_img(self, img: np.ndarray) -> np.ndarray:
        """Resize to target size, apply data augmentation."""

        # there are damaged files in IAM dataset - just use black image instead
        if img is None:
            img = np.zeros(self.img_size[::-1])

        # data augmentation
        img = img.astype(np.float)
        if self.data_augmentation:
            # photometric data augmentation
            if random.random() < 0.25:
```



```
                def rand_odd():
                    return random.randint(1, 3) * 2 + 1
                img = cv2.GaussianBlur(img, (rand_odd(), rand_odd()), 0)
            if random.random() < 0.25:
                img = cv2.dilate(img, np.ones((3, 3)))
            if random.random() < 0.25:
                img = cv2.erode(img, np.ones((3, 3)))

            # geometric data augmentation
            wt, ht = self.img_size
            h, w = img.shape
            f = min(wt / w, ht / h)
            fx = f * np.random.uniform(0.75, 1.05)
            fy = f * np.random.uniform(0.75, 1.05)

            # random position around center
            txc = (wt - w * fx) / 2
            tyc = (ht - h * fy) / 2
            freedom_x = max((wt - fx * w) / 2, 0)
            freedom_y = max((ht - fy * h) / 2, 0)
            tx = txc + np.random.uniform(-freedom_x, freedom_x)
            ty = tyc + np.random.uniform(-freedom_y, freedom_y)

            # map image into target image
            M = np.float32([[fx, 0, tx], [0, fy, ty]])
            target = np.ones(self.img_size[::-1]) * 255
            img = cv2.warpAffine(img, M, dsize=self.img_size, dst=target, borderMode=cv2.BORDER_TRANSPARENT)

            # photometric data augmentation
            if random.random() < 0.5:
                img = img * (0.25 + random.random() * 0.75)
            if random.random() < 0.25:
                img = np.clip(img + (np.random.random(img.shape) - 0.5) * random.randint(1, 25), 0, 255)
            if random.random() < 0.1:
                img = 255 - img

        # no data augmentation
        else:
            if self.dynamic_width:
                ht = self.img_size[1]
                h, w = img.shape
                f = ht / h
                wt = int(f * w + self.padding)
                wt = wt + (4 - wt) % 4
                tx = (wt - w * f) / 2
                ty = 0
            else:
                wt, ht = self.img_size
                h, w = img.shape
                f = min(wt / w, ht / h)
                tx = (wt - w * f) / 2
                ty = (ht - h * f) / 2

            # map image into target image
            M = np.float32([[f, 0, tx], [0, f, ty]])
            target = np.ones([ht, wt]) * 255
            img = cv2.warpAffine(img, M, dsize=(wt, ht), dst=target, borderMode=cv2.BORDER_TRANSPARENT)

        # transpose for TF
        img = cv2.transpose(img)

        # convert to range [-1, 1]
        img = img / 255 - 0.5
        return img

    def process_batch(self, batch: Batch) -> Batch:
        if self.line_mode:
            batch = self._simulate_text_line(batch)

        res_imgs = [self.process_img(img) for img in batch.imgs]
        max_text_len = res_imgs[0].shape[0] // 4
        res_gt_texts = [self._truncate_label(gt_text, max_text_len) for gt_text in batch.gt_texts]
        return Batch(res_imgs, res_gt_texts, batch.batch_size)


def main():
    import matplotlib.pyplot as plt

    img = cv2.imread('../data/test.png', cv2.IMREAD_GRAYSCALE)
    img_aug = Preprocessor((256, 32), data_augmentation=True).process_img(img)
    plt.subplot(121)
    plt.imshow(img, cmap='gray')
    plt.subplot(122)
    plt.imshow(cv2.transpose(img_aug) + 0.5, cmap='gray', vmin=0, vmax=1)
    plt.show()


if __name__ == '__main__':
    main()
```

Figure A-3. HTR Model preprocessor.py Code Snippet

dataloader_iam.py

```python
import pickle
import random
from collections import namedtuple
from typing import Tuple

import cv2
import lmdb
import numpy as np
from path import Path

Sample = namedtuple('Sample', 'gt_text, file_path')
Batch = namedtuple('Batch', 'imgs, gt_texts, batch_size')


class DataLoaderIAM:
    """
    Loads data which corresponds to IAM format,
    see: http://www.fki.inf.unibe.ch/databases/iam-handwriting-database
    """

    def __init__(self,
                 data_dir: Path,
                 batch_size: int,
                 data_split: float = 0.95,
                 fast: bool = True) -> None:
        """Loader for dataset."""

        assert data_dir.exists()

        self.fast = fast
        if fast:
            self.env = lmdb.open(str(data_dir / 'lmdb'), readonly=True)

        self.data_augmentation = False
        self.curr_idx = 0
        self.batch_size = batch_size
        self.samples = []

        f = open(data_dir / 'gt/words.txt')
        chars = set()
        bad_samples_reference = ['a01-117-05-02', 'r06-022-03-05']  # known broken images in IAM dataset
        for line in f:
            # ignore empty and comment lines
            line = line.strip()
            if not line or line[0] == '#':
                continue

            line_split = line.split(' ')
            assert len(line_split) >= 9

            # filename: part1-part2-part3 --> part1/part1-part2/part1-part2-part3.png
            file_name_split = line_split[0].split('-')
            file_name_subdir1 = file_name_split[0]
            file_name_subdir2 = f'{file_name_split[0]}-{file_name_split[1]}'
            file_base_name = line_split[0] + '.png'
            file_name = data_dir / 'img' / file_name_subdir1 / file_name_subdir2 / file_base_name

            if line_split[0] in bad_samples_reference:
                print('Ignoring known broken image:', file_name)
                continue

            # GT text are columns starting at 9
            gt_text = ' '.join(line_split[8:])
            chars = chars.union(set(list(gt_text)))

            # put sample into list
            self.samples.append(Sample(gt_text, file_name))

        # split into training and validation set: 95% - 5%
        split_idx = int(data_split * len(self.samples))
        self.train_samples = self.samples[:split_idx]
        self.validation_samples = self.samples[split_idx:]

        # put words into lists
        self.train_words = [x.gt_text for x in self.train_samples]
        self.validation_words = [x.gt_text for x in self.validation_samples]

        # start with train set
        self.train_set()

        # list of all chars in dataset
        self.char_list = sorted(list(chars))

    def train_set(self) -> None:
        """Switch to randomly chosen subset of training set."""
        self.data_augmentation = True
        self.curr_idx = 0
        random.shuffle(self.train_samples)
        self.samples = self.train_samples
        self.curr_set = 'train'

    def validation_set(self) -> None:
        """Switch to validation set."""
        self.data_augmentation = False
        self.curr_idx = 0
        self.samples = self.validation_samples
        self.curr_set = 'val'

    def get_iterator_info(self) -> Tuple[int, int]:
        """Current batch index and overall number of batches."""
        if self.curr_set == 'train':
            num_batches = int(np.floor(len(self.samples) / self.batch_size))  # train set: only full-sized batches
```

```
        else:
            num_batches = int(np.ceil(len(self.samples) / self.batch_size))  # val set: allow last batch to be smaller
        curr_batch = self.curr_idx // self.batch_size + 1
        return curr_batch, num_batches

    def has_next(self) -> bool:
        """Is there a next element?"""
        if self.curr_set == 'train':
            return self.curr_idx + self.batch_size <= len(self.samples)  # train set: only full-sized batches
        else:
            return self.curr_idx < len(self.samples)  # val set: allow last batch to be smaller

    def _get_img(self, i: int) -> np.ndarray:
        if self.fast:
            with self.env.begin() as txn:
                basename = Path(self.samples[i].file_path).basename()
                data = txn.get(basename.encode("ascii"))
                img = pickle.loads(data)
        else:
            img = cv2.imread(self.samples[i].file_path, cv2.IMREAD_GRAYSCALE)

        return img

    def get_next(self) -> Batch:
        """Get next element."""
        batch_range = range(self.curr_idx, min(self.curr_idx + self.batch_size, len(self.samples)))

        imgs = [self._get_img(i) for i in batch_range]
        gt_texts = [self.samples[i].gt_text for i in batch_range]

        self.curr_idx += self.batch_size
        return Batch(imgs, gt_texts, len(imgs))
```

Figure A-4. HTR Model dataloader_iam.py Code Snippet

create_lmdb.py

```
import argparse
import pickle

import cv2
import lmdb
from path import Path

parser = argparse.ArgumentParser()
parser.add_argument('--data_dir', type=Path, required=True)
args = parser.parse_args()

# 2GB is enough for IAM dataset
assert not (args.data_dir / 'lmdb').exists()
env = lmdb.open(str(args.data_dir / 'lmdb'), map_size=1024 * 1024 * 1024 * 2)

# go over all png files
fn_imgs = list((args.data_dir / 'img').walkfiles('*.png'))

# and put the imgs into lmdb as pickled grayscale imgs
with env.begin(write=True) as txn:
    for i, fn_img in enumerate(fn_imgs):
        print(i, len(fn_imgs))
        img = cv2.imread(fn_img, cv2.IMREAD_GRAYSCALE)
        basename = fn_img.basename()
        txn.put(basename.encode("ascii"), pickle.dumps(img))

env.close()
```

Figure A-5. HTR Model create_lmdb.py Code Snippet

Appendix B - ICROP Model Source Code

selectCoordinates.py

```python
## To select coordinates of the crop fields
import cv2

# Load the image
image = cv2.imread('Form_Template.jpg')

# Resize the image for display
scale_percent = 50  # Adjust this percentage as needed
width = int(image.shape[1] * scale_percent / 150)
height = int(image.shape[0] * scale_percent / 150)
dim = (width, height)

# Resize image
resized_image = cv2.resize(image, dim, interpolation=cv2.INTER_AREA)

# Select ROI on the resized image
roi = cv2.selectROI("Select ROI", resized_image)

# Print the coordinates of the selected ROI on the resized image
print("Selected ROI on resized image:", roi)

# Close the window
cv2.destroyAllWindows()

# Calculate the scaling factor
scale_x = image.shape[1] / resized_image.shape[1]
scale_y = image.shape[0] / resized_image.shape[0]

# Map the coordinates back to the original image size
x, y, w, h = roi
original_roi = (int(x * scale_x), int(y * scale_y), int(w * scale_x), int(h * scale_y))

# Print the mapped coordinates
print("Mapped ROI on original image:", original_roi)
```

Figure B-1. ICROP Model Source Code Snippet

AHIS_CropModel1.py

```python
import cv2
import numpy as np
import os


def crop():
    # Load the template image (reference form)
    template_image_path = r"C:\Users\Eunice Lee\OneDrive - Monash University\Desktop\HTR Model\SimpleHTR-master\src\Form_Template.jpg"
    template_image = cv2.imread(template_image_path, cv2.IMREAD_GRAYSCALE)
    if template_image is None:
        raise ValueError(f"Template image not found at {template_image_path}")

    # Load the filled form image (scanned form from patient)
    filled_image_path = r"C:\Users\Eunice Lee\OneDrive - Monash University\Desktop\HTR Model\SimpleHTR-master\src\Form_UpperCase2.jpg"
    filled_image = cv2.imread(filled_image_path, cv2.IMREAD_GRAYSCALE)
    if filled_image is None:
        raise ValueError(f"Filled image not found at {filled_image_path}")

    # Resize filled image to match template dimensions
    template_height, template_width = template_image.shape
    filled_image = cv2.resize(filled_image, (template_width, template_height))

    # Detect keypoints and descriptors using ORB detector
    orb = cv2.ORB_create()
    keypoints_template, descriptors_template = orb.detectAndCompute(template_image, None)
    keypoints_filled, descriptors_filled = orb.detectAndCompute(filled_image, None)
```

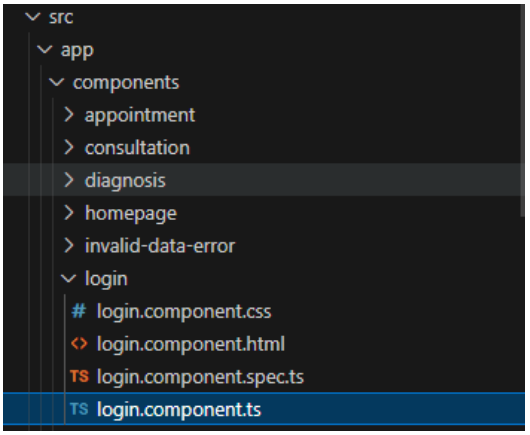
```python
    # Match descriptors using BFMatcher
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(descriptors_template, descriptors_filled)

    # Sort matches based on distance
    matches = sorted(matches, key=lambda x: x.distance)

    # Use homography to align the filled image with the template
    if len(matches) >= 4:
        src_pts = np.float32([keypoints_template[m.queryIdx].pt for m in matches]).reshape(-1, 2)
        dst_pts = np.float32([keypoints_filled[m.trainIdx].pt for m in matches]).reshape(-1, 2)

        # Compute the homography matrix
        matrix, _ = cv2.findHomography(dst_pts, src_pts, cv2.RANSAC, 5.0)
        aligned_image = cv2.warpPerspective(filled_image, matrix, (template_width, template_height))

        # Define the field coordinates for cropping
        fields_coordinates_adjusted = {
            'FirstName': (820, 546, 672, 76),
            'LastName': (820, 630, 670, 74),
            'BirthDate': (822, 710, 666, 74),
            'Gender': (820, 790, 666, 74),
            'Nationality': (822, 872, 662, 72),
            'IC_Passport': (822, 952, 660, 70),
            'MaritalStatus': (824, 1034, 656, 114),
            'PhoneNo': (820, 1158, 662, 70),
            'Email': (820, 1238, 660, 68),
            'Address': (822, 1318, 658, 68),
            'City': (820, 1396, 660, 70),
```

```
            'State': (822, 1476, 660, 70),
            'ZipCode': (822, 1560, 654, 66),
            'EC_FirstName': (822, 1722, 654, 66),
            'EC_LastName': (822, 1800, 654, 69),
            'Relationship': (822, 1887, 654, 60),
            'EC_ContactNo': (819, 1962, 654, 69)
        }

        # Extract the filled image filename (without extension) to use as directory name
        filled_image_filename = os.path.splitext(os.path.basename(filled_image_path))[0]
        save_directory = os.path.join(
            r'C:\Users\Eunice Lee\OneDrive - Monash University\Desktop\HTR Model\SimpleHTR-master\data',
            filled_image_filename)

        # Create directory if it does not exist
        if not os.path.exists(save_directory):
            os.makedirs(save_directory)

        # Log file to store cropped field paths
        log_file_path = os.path.join(save_directory, 'cropped_fields_log.txt')
        with open(log_file_path, 'w') as log_file:
            # Loop through the coordinates and crop each field
            for field_name, (x, y, w, h) in fields_coordinates_adjusted.items():
                cropped_field = aligned_image[y:y + h, x:x + w]
                cropped_image_path = os.path.join(save_directory, f'{field_name}.jpg')
                cv2.imwrite(cropped_image_path, cropped_field, [int(cv2.IMWRITE_JPEG_QUALITY), 95])

                # Log the path of the saved image
                log_file.write(f'{field_name}: {cropped_image_path}\n')

        print(f"Cropping completed successfully. Images saved in directory: {save_directory}")
    else:
        print("Not enough matches found to align images.")


# Run the crop function
if __name__ == "__main__":
    crop()
```

Figure B-2. ICROP Model Source Code Snippet

Appendix C - Web Application Source Code

An example of the Angular Component file structure: Login folder



An example of the Angular Component file: login.component.ts

login.component.ts

src > app > components > login > login.component.ts > ...

```ts
/**
 * @author Alicia Quek Chik Wen <aque0004@student.monash.edu>
 * @author Foo Kai Yan 33085625 <kfoo0012@student.monash.edu>
 *
 * Alicia responsible for backend
 * Kai Yan responsible for frontend
 */

import { Component, inject, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { Router, ActivatedRoute } from '@angular/router';
import { DatabaseService } from 'src/app/services/database.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})

export class LoginComponent {
  /**
   * @constructor The constructor of this component has three dependencies: the DatabaseService, the ActivatedRoute, and the Router
   * @param dbService Service used to fetch and manage data from the database
   * @param router Service used for navigation between components/pages
   * @param route Provides information about the current route in the application
   */
  constructor(private dbService: DatabaseService, private router: Router, private route: ActivatedRoute) { }

  fb = inject(FormBuilder);
  loginForm !: FormGroup;

  /**
   * @function ngOnInit Initializes the login form when the component is loaded
   * @description The form includes email and password which require the user to fill in
   */
  ngOnInit(): void {
    this.loginForm = this.fb.group({
      userEmail: ['', Validators.compose([Validators.required, Validators.email])],
      password: ['', Validators.required],
    },
  );
  }

  /**
   * @function login function passes login object to the service to be sent to the server via a post request.
   * @summary A login object is created based on the form inputs and is submitted to be stored in the MongoDB database.
   * @description onSubmit is triggered when the html form is submitted and the login object is created and sent to the DatabaseService
   * to add it to the MongoDB database.
   */
  login() {
    console.log(this.loginForm.value);
    this.dbService.loginService(this.loginForm.value).subscribe({
      // uses the router service to redirect the client to another component.
      // In other words, replace the current component with another one.
      next: (res) => {
        alert("Login Success");
        console.log("res.toString()", res.toString());
        localStorage.setItem("user_id", res.toString());
        this.loginForm.reset();
        this.router.navigate(["/home"])},
      error: (error) => {
        console.log('Error:', error);
        alert("Login Failed! Incorrect password or Invalid email! Try Again!");
        this.router.navigate(["/login"])}
      })
  }

}
```

Figure C-1. AHIS Frontend Web Application Source Code Snippet

A file example of the Angular Pipe: date-format.pipe.ts

TS date-format.pipe.ts M

src > app > pipes > TS date-format.pipe.ts > ...

```ts
// Import necessary modules from Angular core library
import { Pipe, PipeTransform } from '@angular/core';

// Define a new pipe with the name 'dateFormat'
@Pipe({
  name: 'dateFormat' // This is the pipe name used in templates
})

// The pipe class implements the PipeTransform interface
export class DateFormatPipe implements PipeTransform {

  /**
   * The transform method receives a date value, processes it, and returns the date formatted as a string in the
   * format "day-month-year".
   *
   * @param value The input value to be transformed (expected to be a date or timestamp)
   * @param args Additional optional arguments
   * @returns The date formatted as "day-month-year"
   */
  transform(value: any, ...args: unknown[]): string {
    // Convert the input value to a Date object
    let date = new Date(value);
    // Format the date as "day-month-year"
    let formatedDate = `${date.getDate()}-${date.getMonth() + 1}-${date.getFullYear()}`;
    // Return the formatted date string
    return formatedDate;
  }
}
```

Figure C-2. AHIS Frontend Web Application Source Code Snippet

An example of the Angular Service file: database.service.ts

TS database.service.ts M ×

src > app > services > TS database.service.ts > DatabaseService > insertConsultation

```ts
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Injectable } from '@angular/core';

// Define HTTP options with JSON content type for requests
const httpOptions = {
  headers: new HttpHeaders({ "Content-Type": "application/json" }),
};

// Injectable decorator to make the service available throughout the app
@Injectable({
  providedIn: 'root'
})

/**
 * The main benefit of having a service is to organize and share methods, models, or data with different
 * components of an Angular application.
 * This Database Service is responsible for the communication with the RESTFul server.
 */
export class DatabaseService {

  // Inject HttpClient to make HTTP requests
  constructor(private http: HttpClient) { }

  /**
   * Patient
   * @author Alicia Quek Chik Wen <aque0004@student.monash.edu>
   */
  insertPatient(aPatient: any){
    return this.http.post("/patient/add", aPatient, httpOptions);
  }

  getPatients(){
    return this.http.get("/patient/list-patients", httpOptions);
  }

  deletePatient(patientId: string){
    console.log("Database service delete patient is invoked", patientId);
    return this.http.delete("/patient/delete-patient", {body: {patientId: patientId}, ...httpOptions});
  }

  updatePatient(aPatient: any){
    console.log("Database service update patient is invoked", aPatient)
    return this.http.put("/patient/update-patient", aPatient, httpOptions);
  }

  getJsonData(){
    return this.http.get("/patient_data.json");
  }

  /**
   * Consultation
   * @author Alicia Quek Chik Wen <aque0004@student.monash.edu>
   */
  insertConsultation(aConsultation: any){
    console.log("Database service add consultation is invoked", aConsultation)
      return this.http.post("/consultation/add", aConsultation, httpOptions);
  }
```

```
59    getConsultations(){
60      return this.http.get("/consultation/list-consultation", httpOptions);
61    }
62
63    deleteConsultation(consultationId: string){
64      console.log("Database service delete consultation is invoked", consultationId)
65      return this.http.delete("/consultation/delete-consultation", {body: {consultationId: consultationId}, ...httpOptions});
66    }
67
68    updateConsultation(aConsultation: any){
69      console.log("Database service update patient is invoked", aConsultation)
70      return this.http.put("/consultation/update-consultation", aConsultation, httpOptions);
71    }
72
73    /**
74     * Diagnosis
75     * @author Alicia Quek Chik Wen <aque0004@student.monash.edu>
76     */
77    insertDiagnosis(aDiagnosis: any){
78      console.log("Database service add Diagnosis is invoked", aDiagnosis)
79        return this.http.post("/diagnosis/add", aDiagnosis, httpOptions);
80    }
```

Figure C-3. AHIS Frontend Web Application Source Code Snippet

An example of a Component Controller file: consultation_controller.js

JS consultation_controller.js

backend > controllers > JS consultation_controller.js > ...

```
1   /**
2    * @author Alicia Quek Chik Wen <aque0004@student.monash.edu>
3    */
4
5   /**
6    * Import required packages
7    * @constant
8    */
9   const Patient = require("../models/patient");
10  const Diagnosis = require("../models/diagnosis");
11  const Consultation = require("../models/consultation");
12
13  /**
14   * Export an object with functions that
15   * - Insert new Consultation
16   * - List all Consultation
17   * - Delete Consultation by Id
18   * - Update Consultation info by Id
19   * @exports consultationFunctions
20   */
21  module.exports = {
22      /**
23       * Function that insert new consultation into the database
24       * @name addConsultation
25       * @function
26       * @param {Object} req - includes physicianId, generalAppearance, height, weight, bloodPressure, breathingPattern
27       *                       pupilReflexCondition, nerveReflexCondition, other, patientSymptom, patientId, temperature
28       * @param {Object} res - returns the id of the consultation as a json object
29       */
```

```
30      addConsultation: async function (req, res) {
31          try{
32              aConsultation = new Consultation({
33                  physicianId: req.body.physicianId,
34                  generalAppearance: req.body.generalAppearance,
35                  temperature: req.body.temperature,
36                  height: req.body.height,
37                  weight: req.body.weight,
38                  bloodPressure: req.body.bloodPressure,
39                  breathingPattern: req.body.breathingPattern,
40                  pupilReflexCondition: req.body.pupilReflexCondition,
41                  nerveReflexCondition: req.body.nerveReflexCondition,
42                  other: req.body.other,
43                  patientSymptom: req.body.patientSymptom,
44                  patientId: req.body.patientId
45              });
46              console.log(aConsultation)
47              await aConsultation.save();
48
49              // Add the consultation to related patient object consultationList
50              await  Patient.findOneAndUpdate(
51                  { patientId: req.body.patientId },
52                  { $push: {consultationList: aConsultation._id} },
53                  {new: true});
54
55              res.status(200).json(aConsultation); //HTTP response status codes
56          }
57          catch{
58              res.status(400).json({ error: "Invalid Data"});
59          }
60      },
```

```
61      /**
62       * Function that list all Consultation in the database
63       * @name getAll
64       * @function
65       * @param {Object} req
66       * @param {Object} res - returns the list of diagnosis as a json object
67       */
68      getAll: async function (req, res) {
69          let consultation = await Consultation.find().populate("diagnosisList");
70          res.status(200).json(consultation);
71      },
```

```
    /**
     * Function that delete a Consultation by its id
     * @name deleteById
     * @function
     * @param {Object} req - includes the id of the diagnosis
     * @param {Object} res - returns the deleted diagnosis as a json object
     */
    deleteById: async function (req, res) {
        console.log("Backend controller DELETE Consultation", req.body)
        try{
            let consultation = await  Consultation.findOne({ consultationId: req.body.consultationId });
            let diagnosisListLength = consultation.diagnosisList.length;
            console.log("Backend Consultation", consultation)

            // Remove and delete any diagnosis related to the consultation
            for(let i=0; i < diagnosisListLength; i++){
                let diagnosisId = consultation.diagnosisList[consultation.diagnosisList.length-1];
                consultation.diagnosisList.pop(diagnosisId);
                await Diagnosis.deleteOne({ _id: diagnosisId });
            }
            await consultation.save();

            // Remove and delete the consultation related to the patient
            await Patient.updateMany({ patientId: consultation.patientId }, { $pull: { consultationList: consultation._id }

            // Delete the Consultation object
            let deleteConsultation = await Consultation.deleteOne({ _id: consultation._id });

            res.status(200).json(deleteConsultation);
        }
        catch{
            res.status(400).json({ error: "Invalid Data"});
        }
    },
    /**
     * Function that update Consultation information by its id
     * @name updatConsultation
     * @function
     * @param {Object} req - includes the id of the Consultation, and the new other, patientSymptom
     * @param {Object} res - returns the status of the update as a json object
     */
    updateConsultation: async function (req, res) {
        console.log("Backend controller update Consultation", req.body)
        try{
            let consultation = await  Consultation.findOneAndUpdate({ consultationId: req.body.consultationId },
                {   other: req.body.other,
                    patientSymptom: req.body.patientSymptom,
                },
                {new: true});
            let status = "";
            // Return the correct status message accordingly
            if(consultation  === null){
                status = "ID not found"
            }
            else{
                status = "Consultation information updated successfully"
            }
            res.status(200).json({
                "status": status
                });
            }
            catch{
                res.status(400).json({ error: "Invalid Data"});
            }
        }
    };
```

Figure C-4. AHIS Backend Web Application Source Code Snippet

A file example of the Component Model : diagnosis.js

diagnosis.js

backend > models > diagnosis.js > ...

```
/**
 * @author Alicia Quek Chik Wen <aque0004@student.monash.edu>
 * Represents a Mongoose Schema for Diagnosis class
 */

/**
 * Import required package to reference the Mongoose package to create your schema:
 * @constant
 */
const mongoose = require("mongoose");

/**
 * Create Diagnosis schema with the following fields, specifying its type, whether its mandatory and its default value.
 * @property {String} diagnosisId               - The Diagnosis Id
 * @property {String} diagnosisDescription      - Required Diagnosis description
 * @property {String} diagnosisAdditional       - Diagnosis Addiotional note
 * @property {Date} diagnosisDate               - Diagnosis default value to the current date/time
 * @property {String} physicianId                - The name of the physician-in-charge
 * @property {ObjectId} patientId               - patient id reference
 * @property {ObjectId} consultationId          - consultation id reference
 * @property {ObjectId} prescriptionList        - Array of Prescription reference
 * @constant
 */
```

```
const diagnosisSchema = mongoose.Schema({
    diagnosisId: {
        type: String,
        default: function(){
            let id = "D";
            let randChar = String.fromCharCode(Math.floor(Math.random() * 26 + 65)) + String.fromCharCode(Math.floor(Math.r
            let randNum = Math.round(Math.random() * 9000 + 1000);
            id += randChar + "-" + randNum;
            return id;
        }
    },
    diagnosisDescription: {
        type: String,
        required: true,
    },
    diagnosisAdditional: {
        type: String,
        default: "NA",
    },
    diagnosisDate: {
        type: Date,
        default: function(){
            let date = new Date();
            return date;
        },
    },
    physicianId: {
        type: String,
        required: true,
    },
```

```
    patientId: {
        type: String,
        required: true,
    },
    consultationId: {
        type: String,
        required: true,
    },
    prescriptionList: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: "Prescription",
        },
    ],


});

/**
 * Export Diagnosis Model that uses Diagnosis schema
 * Mongoose will create a collection named 'diagnosis'
 * @exports Diagnosis
 */
module.exports = mongoose.model("Diagnosis", diagnosisSchema);
```

Figure C-5. AHIS Backend Web Application Source Code Snippet

A file example of a Component Router : consultation-api.js

consultation-api.js

backend > routes > consultation-api.js > ...

```
/**
 * @author Alicia Quek Chik Wen <aque0004@student.monash.edu>
 * Configuring RESTFul Endpoints to manage consultation
 */

/**
 * Import required packages
 * @constant
 */
const express = require("express");
const consultationCont = require("../controllers/consultation_controller"); // where the implementation are in
const router = express.Router();

/**
 * Configuring RESTFul Endpoints
 * - Insert new consultation
 * - List all consultation
 * - Delete consultation by Id
 * - Update consultation by Id
 */
router.post("/add", consultationCont.addConsultation);
router.get("/list-consultation", consultationCont.getAll);
router.delete("/delete-consultation", consultationCont.deleteById);
router.put("/update-consultation", consultationCont.updateConsultation);

/**
 * Export router
 * @exports consultationRouterAPI
 */
module.exports = router;
```

Figure C-6. AHIS Backend Web Application Source Code Snippet

Server.js

JS server.js > ...

```
/**
 * @author Alicia Quek Chik Wen <aque0004@student.monash.edu>
 * @author Foo Kai Yan 33085625 <kfoo0012@student.monash.edu>
 * Represents the main server file
 */

/**
 * Import required packages
 * @constant
 */
const express = require("express");
const path = require("path");
const mongoose = require("mongoose");
const fs = require("fs"); // may be required for htr
const cookieParser = require("cookie-parser");
const multer = require('multer');
const bodyParser = require('body-parser')
const { spawn } = require('child_process');



/**
 * API Router and Middleware functions
 * @constant
 */
const patientRouter = require("./backend/routes/patient-api.js");
const consultationRouter = require("./backend/routes/consultation-api.js");
const diagnosisRouter = require("./backend/routes/diagnosis-api.js");
const roleRouter = require("./backend/routes/role-api.js");
const authRouter = require("./backend/routes/auth-api.js");
const api_medication = require("./backend/routes/medication-api.js");
const api_prescription = require("./backend/routes/prescription-api.js");
```

```
const api_prescription = require("./backend/routes/prescription-api.js");
const api_appointment = require("./backend/routes/appointment-api.js");
const api_physician = require("./backend/routes/physician-api.js");

/**
 * Port number
 * @constant
 */
const PORT_NUMBER = 8080;

/**
 * App instance
 * @constant
 */
const app = express();
const server = require('http').Server(app);
const io = require("socket.io")(server);


/**
 * Middleware functions
 * Using express.static built-in middleware function in Express to serve static files
 * @constant
 */
//app.use(express.static(path.join(__dirname, "audio_files")));
//express will server angular as static asset
app.use(express.static(path.join(__dirname, "dist/mds2-ahis")));
app.use(express.static(path.join(__dirname, "data_files")));
app.use(express.json());
app.use(cookieParser());
```

```
/**
 * Configure to listen to the specify port number
 * @name listen
 * @function
 * @param {int} PORT_NUMBER - Express port number
 * @param {Function} callback - Express callback
 */
server.listen(PORT_NUMBER, function () {
    console.log(`listening on port ${PORT_NUMBER}`);
});

/**
 * Mongoose URL string which will be used to connect to the database named HealthDB
 * @constant
 */
const url = "mongodb://127.0.0.1:27017/HealthInformationDB";

/**
 * Connect to database
 * @param {string} url - Mongoose URL string to connect to the database
 * @returns - String to indicate that database is successfully connected
 */
async function connect(url) {
    await mongoose.connect(url);
    return "Connected Successfully";
}
connect(url)
  .then(console.log)
  .catch((err) => console.log(err));
```

```
/**
 * Middleware functions
 *
 * using Express.Router to separate the routes for different components, including
 * - patient
 * - medication
 * - prescription
 * - consultation
 * - appointment
 * - diagnosis
 * - physician
 * @constant
 */
app.use("/patient", patientRouter);
app.use("/medication", api_medication);
app.use("/prescription", api_prescription);
app.use("/consultation", consultationRouter);
app.use("/appointment", api_appointment);
app.use("/diagnosis", diagnosisRouter);
app.use("/physician", api_physician);
app.use("/role", roleRouter)
app.use("/auth", authRouter)
```

```
/**
 * Configure Multer Storage
 * Define storage using multer.diskStorage.
 */
const storage = multer.diskStorage({
    destination: function (req, file, callBack) {
        // Set the destination to the desired location
        callBack(null, "data_files");
    },
    filename: function (req, file, callBack) {
        // Set the filename function to create unique filenames
        callBack(null, "patientRegistrationForm.jpg");
    }
});

var upload = multer({ storage: storage});
```

```
/**
 * Create an Upload Route
 * Create a route (e.g., /upload) to handle file upload requests with POST method.
 * Use upload.single('file') middleware to handle single file uploads with the field name file.
 * In the route handler function, you can access uploaded file details in req.file.
 */
app.post('/file', upload.single('file'), (req, res, next) =>{
    const file = req.file;
    console.log("filename", file.filename);
    if(!file){
        const error = new Error('Please upload a file');
        error.httpStatusCode = 400;
        return next(error)
    }
    console.log('File uploaded successfully.');
    res.send(file);
})
```

```
app.get('/patientData', (req, res) => {
    const pathJsonFile = "./data_files/patient_data.json";
    fs.readFile(pathJsonFile, 'utf8', (err, file) => {
        // Read/modify file data here
        // check for any errors
        if (err) {
            console.error('Error while reading the file:', err)
            return
        }
        try {
            const data = JSON.parse(file);
            // output the parsed data
            console.log(data);
            res.send(data)
        } catch (err) {
            console.error('Error while parsing JSON data:', err);
        }
    })

})
```

```
io.on("connection", (socket) => {
    console.log("new connection made from client with ID=" + socket.id);

    socket.on("cropping", async(data) => {
        var resultData = "";
        // spawn new child process to call the python script
        // Define the path to main.py
        const pythonScriptPath = path.join(__dirname, 'HTR_Model', 'SimpleHTR-master', 'src', 'AHIS_CropModel1.py');
        const python = spawn('python', [pythonScriptPath]);
        console.log('Calling cropping function');

        // collect data from script
        python.stdout.on('data', function (data) {
            console.log('Pipe data from python script ...');
            resultData += data.toString();
            console.log("resultData", resultData);
        });

        // in close event we are sure that stream from child process is closed
        python.on("close", function () {
        console.log("child process end all stdio with code");
        try {
            // send data to browser
            console.log("resultData", resultData);
            io.emit("cropped", resultData)
        } catch (err) {
            console.error('Error sending resultData:', err);
        }
        });

      });
```

```
    socket.on("htr", async(data) => {
    var resultData = "HTR completed";
    var dataToSend = "";

    // spawn new child process to call the python script
    // Define the path to main.py
    const pythonScriptPath = path.join(__dirname, 'HTR_Model', 'SimpleHTR-master', 'src', 'main.py');
    const python = spawn('python', [pythonScriptPath]);
    console.log('Calling htr function');

    // collect data from script
    python.stdout.on('data', function (data) {
        console.log('Pipe data from python script ...');
        resultData += data.toString();
        console.log("resultData", resultData);
    });

    //in close event we are sure that stream from child process is closed
    python.on("close", function () {
    console.log("child process end all stdio with code");
    try {
        console.log("resultData", resultData);
        dataToSend = resultData;
        console.log("dataToSend", dataToSend);
        io.emit("processed", dataToSend)
    } catch (error) {
        console.error('Error parsing dataToSend:', err);
    }
    });
    });
});
```

Figure C-7. AHIS Backend Web Application Source Code Snippet

Appendix D - Requirements Traceability Matrix

Req. ID	Requirement Description	Category	Requirement Type	Source	Completion Status
1	External devices to capture form images	Feature	Functional	Dr. Fermi	Completed
2	Scalable database design to store date	Feature	Non-Functional	Dr. Fermi	Completed
3	Data privacy and confidentiality	Security	Non-Functional	Dr. Fermi	Completed
4	Integrated CROP and HTR model to web app	Feature	Functional	Dr. Fermi	Completed
5	User Authentication	Security	Functional	Dr. Fermi	Completed
6	Profile Management	Feature	Functional	Dr. Fermi	Completed
7	Contains required features for a complete healthcare web app	Feature	Functional	Dr. Fermi	Completed

8	Appointment and Encounter Feature	Feature	Functional	Dr. Fermi	Completed
9	Review patient past medical records	Feature	Functional	Dr. Fermi	Completed
10	Results displayed on web app is easily to comprehend	Quality	Non-Functional	Dr. Fermi	Completed
11	Satisfaction on the HTR model accuracy	Performance	Non-Functional	Dr. Fermi	Completed

Figure D. Requirement Traceability Matrix (RTM)

Appendix E - Image Evidence for Requirement Met

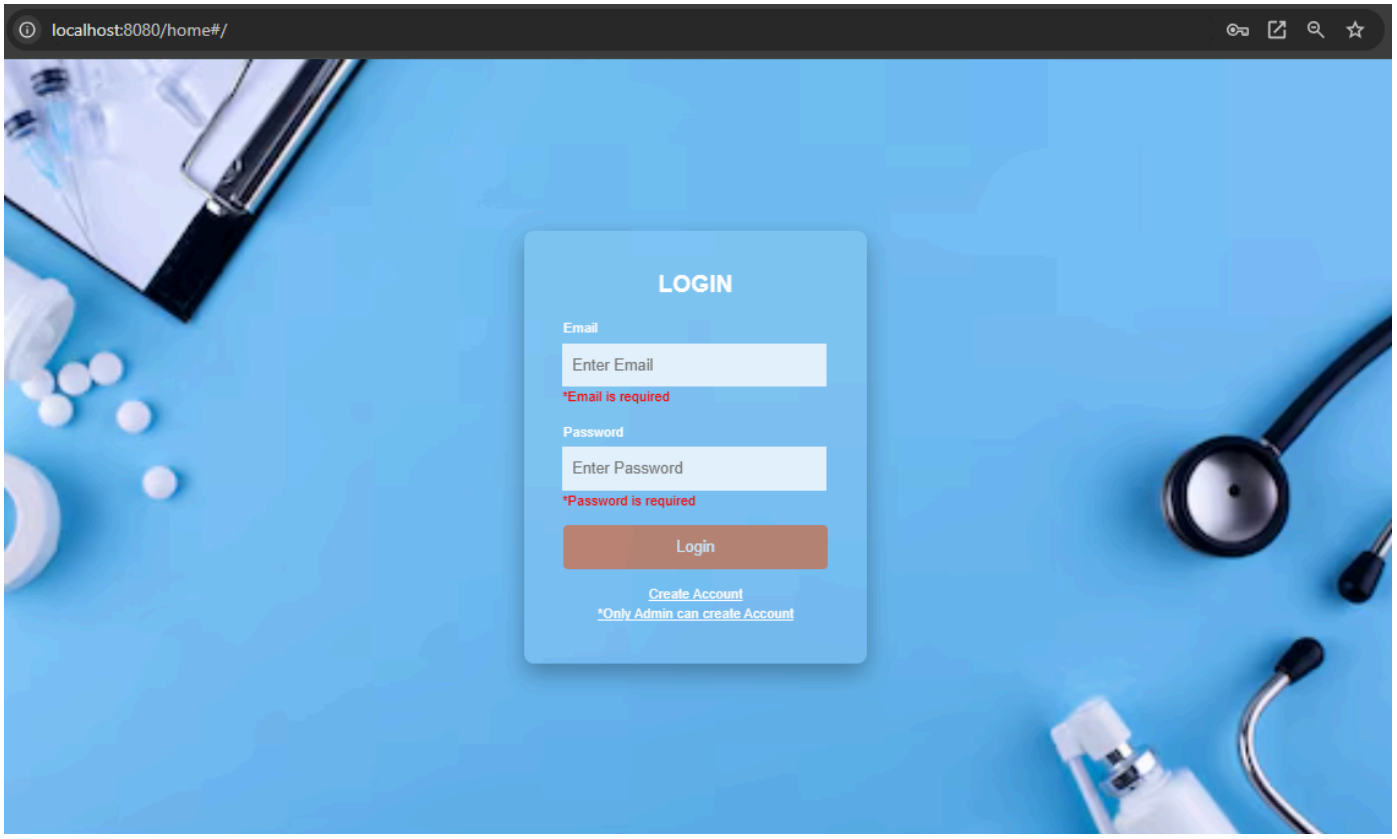


Figure 3.3-1 Login Authentication Page

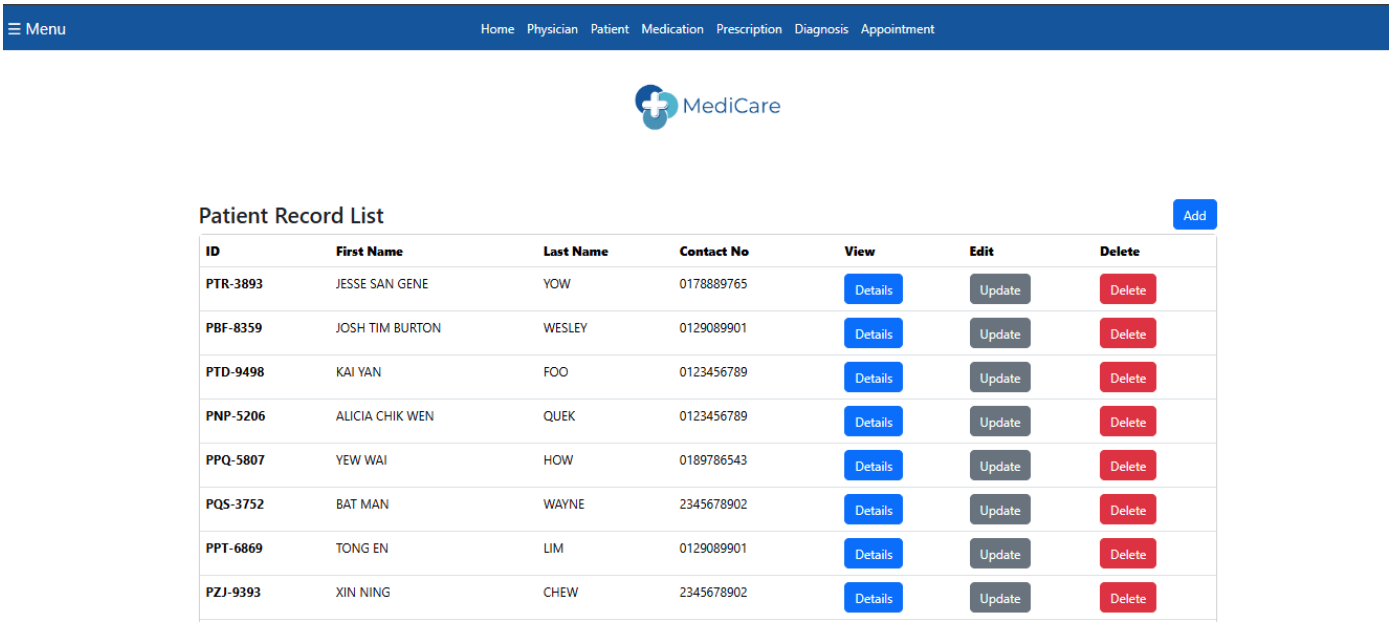


Figure 3.3-2 Patient Profile Management Page

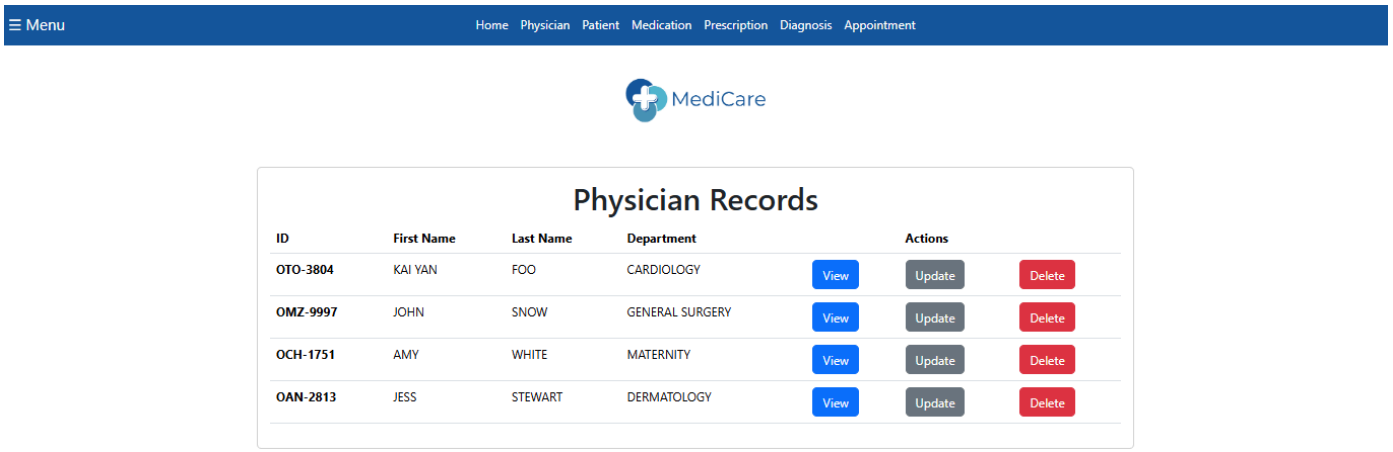


Figure 3.3-3 Physician Profile Management Page

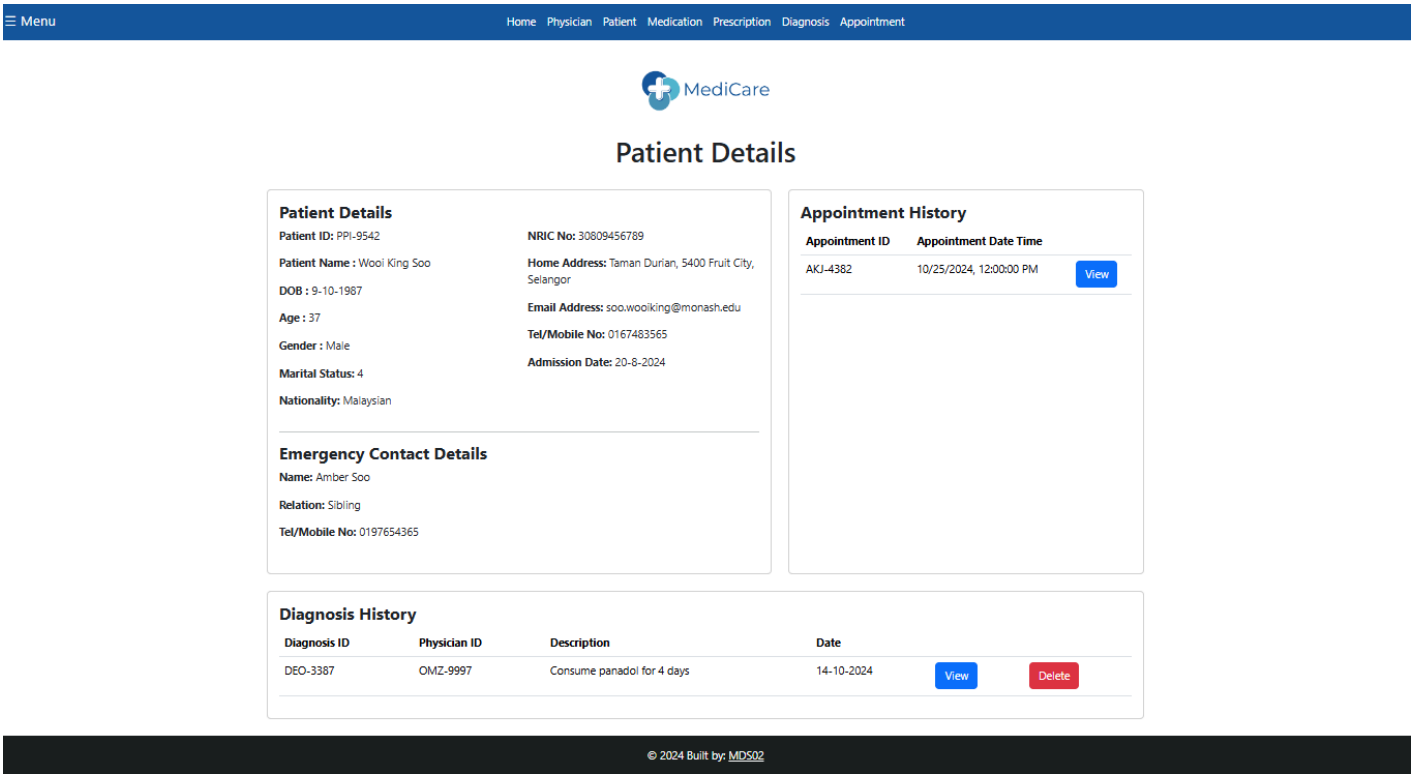


Figure 3.3-4 A Selected Patient Information Record View

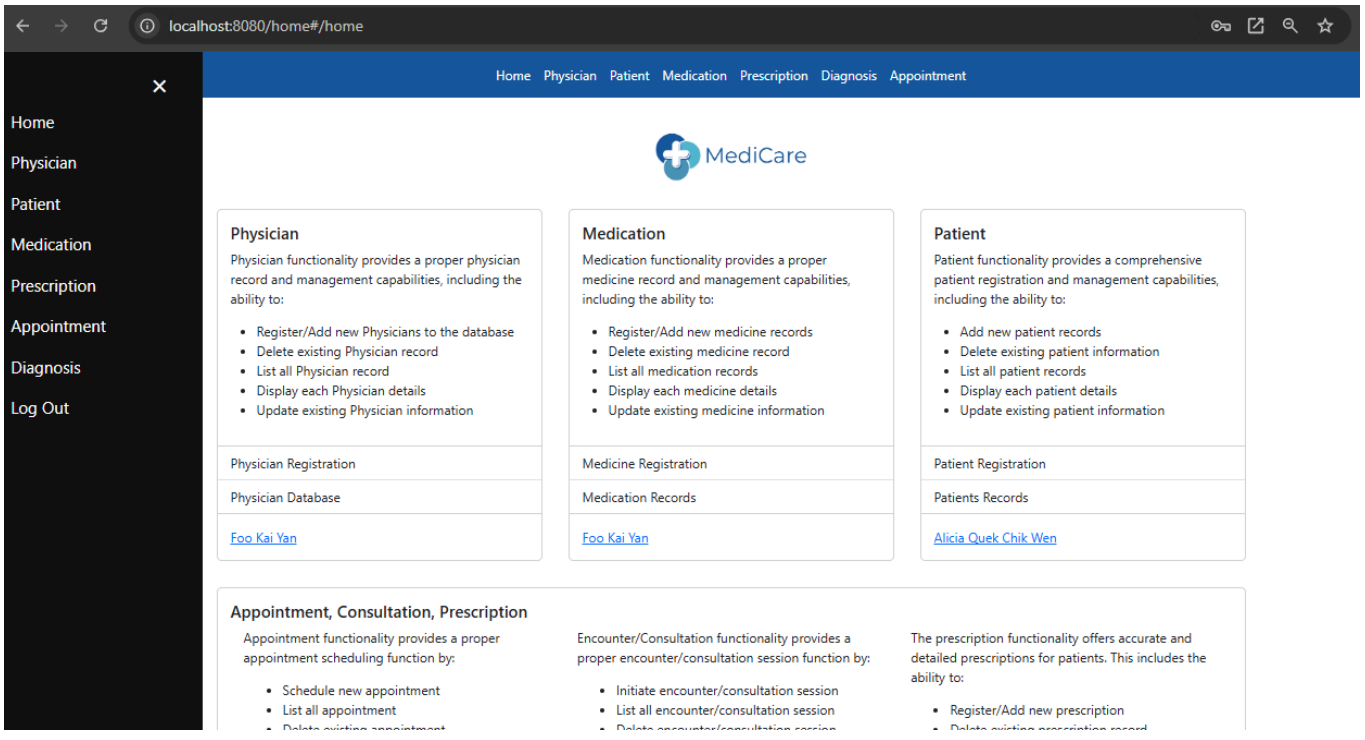


Figure 3.3-5 Navigation Bar and Menu

Appendix F - Image Evidence for Software Quality Summary

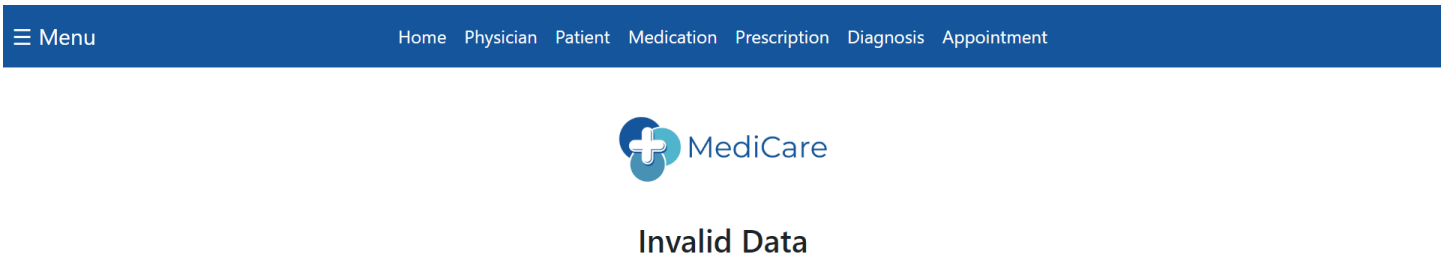


Figure 5.2.3-1 Error Message for invalid input for patient form

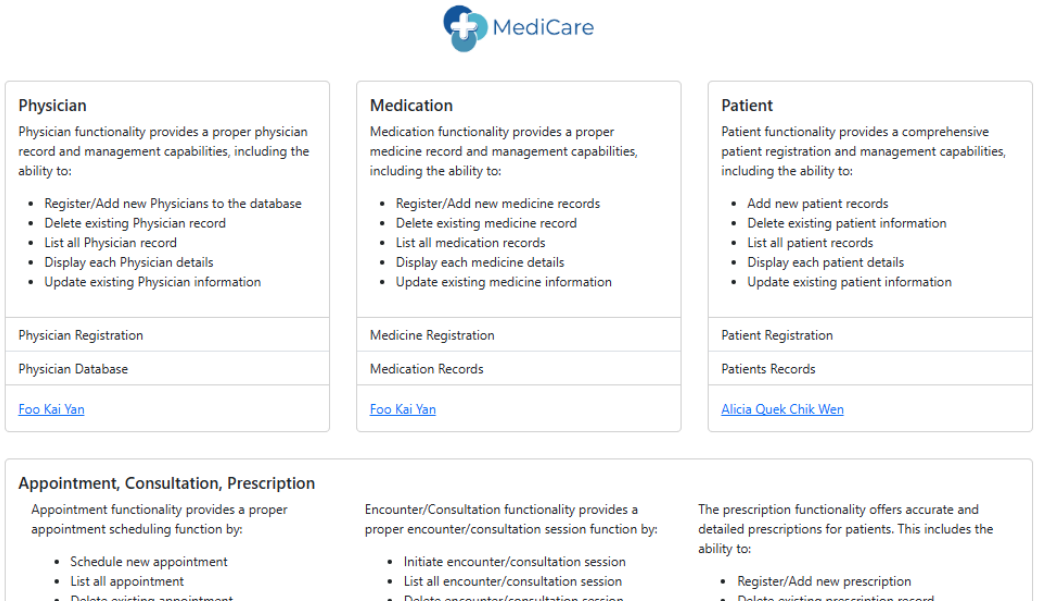


Figure 5.2.3-2 Main page displaying labeled cards for "Physicians," "Medication," and "Physicians".

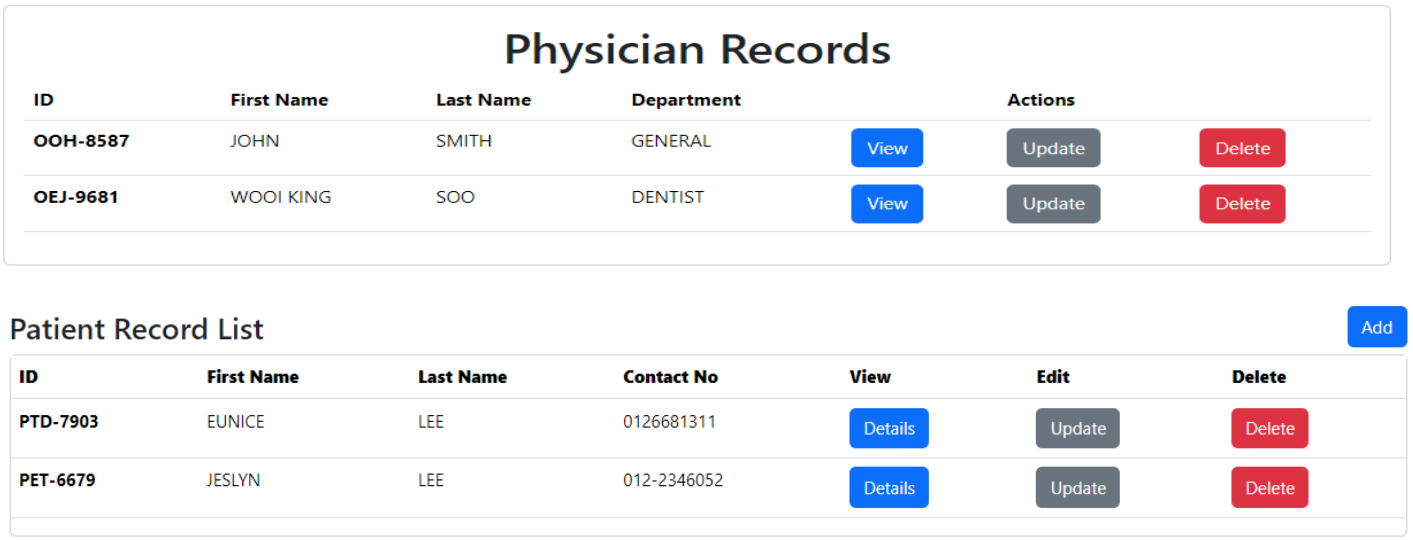


Figure 5.2.3-3 Standardized layout in Patient and Physician Profile with “View,” “Update,” and “Delete” buttons.

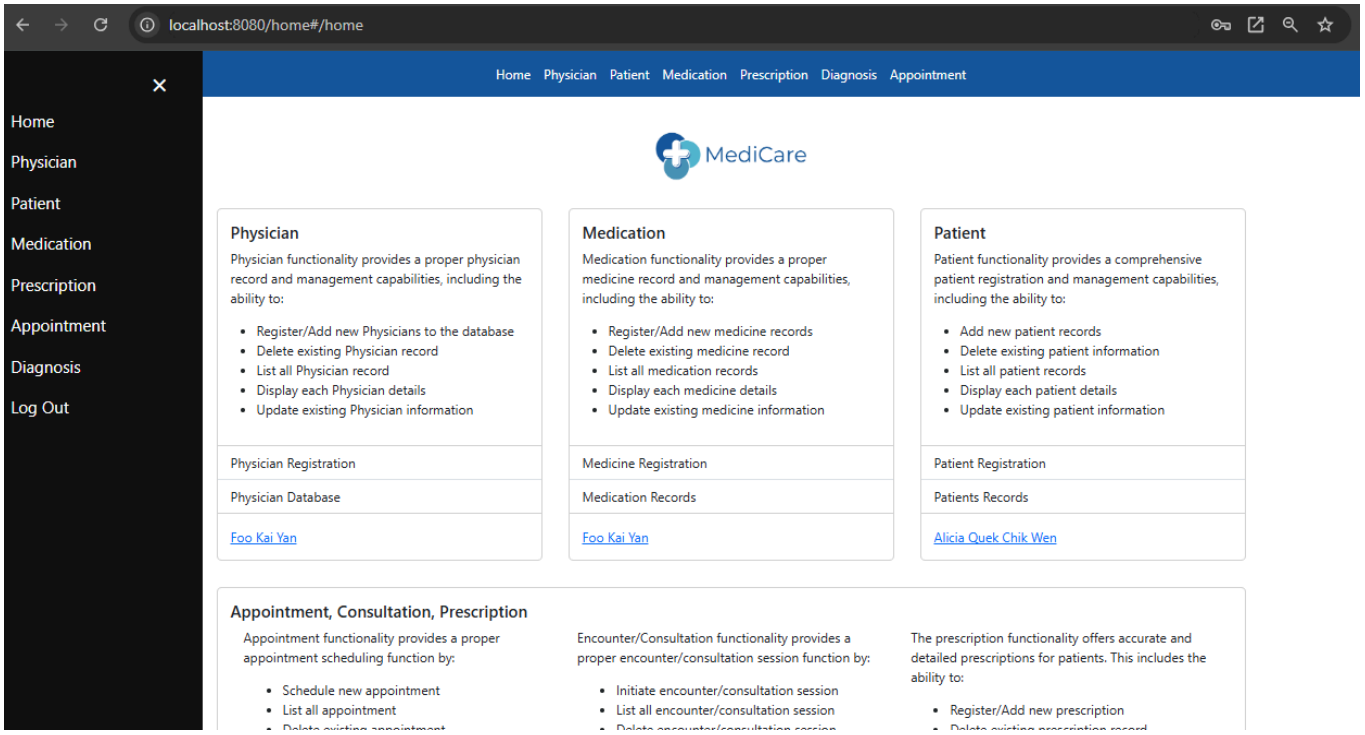


Figure 5.2.3-4 Minimalist dashboard layout.

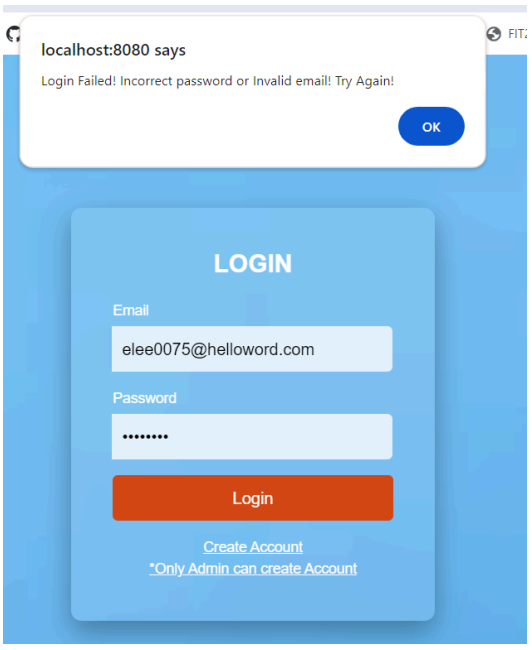


Figure 5.2.3-5 Error message for incorrect login credentials.

9. References

Craig, L., & Awati, R. (2024, January 11). convolutional neural network (CNN). Enterprise AI. https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network

What are Recurrent Neural Networks? | IBM. (n.d.). https://www.ibm.com/topics/recurrent-neural-networks

A. Ansari, B. Kaur, M. Rakhra, A. Singh and D. Singh, "Handwritten Text Recognition using Deep Learning Algorithms," 2022 4th International Conference on Artificial Intelligence and Speech Technology (AIST), Delhi, India, 2022, pp. 1-6, doi: 10.1109/AIST55798.2022.10065348.

What is OCR (Optical Character Recognition)? | IBM. (2024, April 22). Www.ibm.com. https://www.ibm.com/topics/optical-character-recognition

Embedded Data Versus References — MongoDB Manual. (n.d.). Www.mongodb.com. https://www.mongodb.com/docs/manual/data-modeling/concepts/embedding-vs-references/#std-label-data-modeling-referencing

Nader AbdalGhani. (2021). IAM Handwritten Forms Dataset. Kaggle.com. https://www.kaggle.com/datasets/naderabdalghani/iam-handwritten-forms-dataset/data

last_theorem. (2023). iam_handwriting_word_database. Kaggle.com. https://www.kaggle.com/datasets/nibinv23/iam-handwriting-word-database

Papers with Code - Machine Learning Datasets. (2022). Paperswithcode.com. https://paperswithcode.com/datasets?task=handwritten-text-recognition

Terra, J. (2020, July 26). Keras vs Tensorflow vs Pytorch: Key Differences Among the Deep Learning Framework. Simplilearn.com; Simplilearn. https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article#:~:text=TensorFlow%20is%20an%20open%2Dsourced

Compare OpenCV VS Tensorflow | Techjockey.com. (n.d.). Techjockey. Retrieved May 27, 2024, from https://www.techjockey.com/compare/opencv-vs-tensorflow#:~:text=Speed%2FPerformance%3A%20OpenCV%20is%20known

OpenCV. (2020, November 4). About - OpenCV. https://opencv.org/about/#:~:text=OpenCV%20(Open%20Source%20Computer%20Vision,perception%20in%20the%20commercial%20products.

Codecademy. (n.d.). MongoDB Data Modeling Basics. Codecademy. https://www.codecademy.com/article/mongo-db-data-modeling-basics

Scheidl, H. (2020, August 9). Build a Handwritten Text Recognition System using TensorFlow. Medium.

https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5

bcrypt. (2020, April 8). Wikipedia. https://en.wikipedia.org/wiki/Bcrypt

Grigutytė, M. (2023, June 16). What is bcrypt and how does it work? | NordVPN. Nordvpn.com. https://nordvpn.com/blog/what-is-bcrypt/

Balci, B., Saadati, D., & Shiferaw, D. (2017). Recognition using Deep Learning. Retrieved from https://www.semanticscholar.org/paper/Recognition-using-Deep-Learning-Balci-Saadati/3339237110cd5fd3fa8206623e9b740be1d72c9e

Manchala, N. S. Y., Kinthali, N. J., Kotha, N. K., & Jayalaxmi, N. K. S. K. J. (2020). Handwritten Text Recognition using Deep Learning with TensorFlow. *International Journal of Engineering Research And*, *V9*(05). https://doi.org/10.17577/ijertv9is050534

Wang, Y., Xiao, W., & Li, S. (2021). Offline Handwritten text recognition using Deep Learning: A review. *Journal of Physics Conference Series*, *1848*(1), 012015. https://doi.org/10.1088/1742-6596/1848/1/012015

PB_u247-Gem. (2024, March 20). Usability Principles – Jakob Nielsen's 10 Usability Heuristics for User Interface Design. UX24/7. https://ux247.com/usability-principles/

What is Software Robustness? How to Ensure Quality and Reliability - Nexus Software Systems. (n.d.). Nexwebsites.com. https://nexwebsites.com/blog/software-robustness/

Nwose Lotanna. (2024, August 21). FormGroup and FormControl in Angular - LogRocket Blog. LogRocket Blog. https://blog.logrocket.com/formgroup-formcontrol-angular/