TAYLOR' S UNIVERSITY
Wisdom • Integrity • Excellence

Bachelor of Computer Science (Hons)
Bachelor of Software Engineering (Hons)

Module Code: ITS66704 (March 2023)

Module Name: Advanced Programming

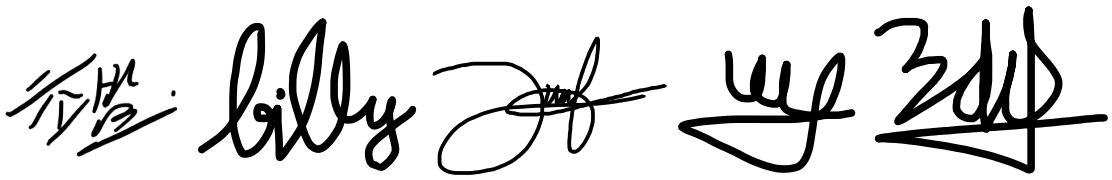
Assignment No./Title	**Assignment Task 2 & Task 3 (Group Project)** **20% (PART A - ANALYSIS AND DESIGN)** **30% (PART B - DEVELOPMENT)** **10% (PRESENTATION)**
Course Tutor/Lecturer	Dr. Steve Teoh Chee Hooi
Submission Date	02/06/2023 (PART A - ANALYSIS AND DESIGN) 23/06/2023 (PART B - DEVELOPMENT) 28/06/2023 (PRESENTATION)
Student Name, ID and Signature 1. Cheah Zixu, 0356946 2. Edward Prilvanto Djong, 0356787 3. Billy Taslim, 0356887 4. Aldrich Eugene Terimsia, 0354515 5. Wong Zi Quan, 0350934	

Declaration *(need to be signed by students. Otherwise, the assessment will not be evaluated)*

Certify that this assignment is entirely my own work, except where I have given fully documented references to the work of others, and that the material contained in this assignment has not previously been submitted for assessment in any other formal course of study.

Marks/Grade:	**Evaluated by:**
Evaluator's Comments:	

Signatures:



Part A

Introduction

Electronic Medical Records System is a computer programme that stores, controls, and allows online access to patient health data. It is made to take the place of the conventional paper medical records kept in healthcare facilities. Electronic medical records System enables patient information may be created, updated, and retrieved by healthcare practitioners quickly and securely. Typically, they contain information about the patient's demographics, medical history, diagnoses, medications, allergies, results of laboratory tests, imaging findings, and therapy recommendations. Coordinated and integrated care is made possible by the system's access and sharing of patient records across authorised healthcare providers in various locations and departments. Although EMR systems have many advantages, it's crucial to remember that they also need to take user training, data privacy, security, and interoperability standards into account when they are implemented and used.

In order to create a simple Electronic Medical Record (EMR) System, our team is going to use Java FX to create the system in order to keep track of patients' medical history. Java FX is a framework and library for Java-based graphical user interface (GUI) desktop and rich internet applications (RIAs). It offers a collection of APIs and resources for building aesthetically pleasing, interactive, and platform-neutral applications. Oracle introduced JavaFX to replace Swing, another Java GUI framework. Since JDK 7u6, it has been included with Java as a component of the Java Development Kit (JDK). But as of Java 11, JavaFX is no longer connected to the JDK and is being developed as a distinct open-source initiative named OpenJFX. JavaFX provides cross-platform functionality, which enables programmes to function on different operating systems, such as Windows, macOS, Linux, and embedded systems. It supports a range of deployment techniques, including web starts, applets, and standalone apps. With JavaFX, Modern desktop programmes with appealing visuals and engaging user interfaces can be made by developers. Developers can create versatile, responsive, and platform-independent applications using a wide range of tools and APIs that are provided to streamline the development process.

Key Features

The feature of this application includes the 6 forms which are Patient Form, Medical History Form, Treatment Course, Analysis Form, and Diagnosis Form:

Patient Form:

This allows the patient to add their information for the database so they could be identified by the medical authorities, this includes, name, national id, age, gender, contact number, and address.

Medical History Form:

This contains the medical history of the patient (which means what sickness or treatment has been done before, or what disease he encountered before), this is done by the doctor, or could be filled by the patient him/herself (but for accuracy purposes, it should be filled by doctor or referenced from a medical personnel) this will include date, time, ward, treatment results, observation, major complications of the patient, including attending doctor/nurse information.

Treatment Course:

This will contain the start date and the end date of the patients' treatment during its treatment time.

Analysis Form:

This will be filled with the information about the test that will be or already been done on the patient, this form will include, the date of which where the test executed, type of test that is being done, result of the test.

Diagnosis Form:

The diagnosis form will contain information about the diagnosed sickness the patient is currently having including the date of the diagnosis, name of the patient, and the patients' diagnosed sickness.

Procedure and Medicine Form:

This will contain information about the procedures, and the medication that is used, this will include date, time, procedure type, medication that is needed, amount of medicine needed, frequency of the usage.

Benefits of EMR System Implementation

An EMR is a digitised representation of a patient's medical records. Healthcare practises can make, save, and receive these charts using EMR systems, which are software programmes. The adoption of EMRs in clinics brings over various advantages, such as:

1. Save space

   By removing the need for keeping, processing, and retrieving paper records, clinics can conserve physical space. This makes it possible to use clinic space more effectively.

2. Enhanced workflows

   By making it easier to track and manage patients, EMRs facilitate clinic workflow improvements. They offer features and tools that simplify a variety of procedures, making it simpler for healthcare professionals to deliver care.

3. Lower operational costs

   Clinics can save money on operating expenses by combining all patient records in one digital location. Printing, storing, and upkeep costs for tangible records are included in this.

4. Contact other medical facilities

   With the use of EMRs, clinics can communicate with and exchange patient data with labs, pharmacies, hospitals, and other healthcare facilities. Coordinated care is made easier to provide and patient outcomes are often improved by this seamless communication.

5. Scalability

   EMRs are made to provide patient records that may be scaled and customised. The digital nature of EMRs makes it simple to expand the system and adapt it to clinics' changing needs as they expand or change.

6. Enables outreach

   With EMRs, clinics can gather and analyse patient data that enables healthcare professionals to pinpoint groups that may benefit from specialised interventions or preventive measures. The general health and wellbeing of the community are enhanced.

7. Documentation

   By lowering documentation errors, EMRs improve the standard of treatment. Accurate and complete documentation is ensured by the digital format of records, which reduces the possibility of missing data or illegible handwriting.

8. Supports research

   EMRs allow medical professionals to monitor patients more efficiently and collect data for research by digitising patient records. This helps to support medical research and healthcare improvements while also improving the standard of care.
9. Checks conflicting treatments

   To defend against potential conflicts between various treatments or medications, EMRs contain built-in safeguards. To protect patients from conflicting therapies, the system can highlight them and notify healthcare professionals.
10. Staff communications

    EMRs make it easier for hospitals, laboratories, clinics, and other staff members to collaborate and communicate effectively. It is simpler for healthcare teams to coordinate care and share crucial information since the system keeps track of vital communications.

Using EMRs offers numerous benefits, not only for the clinics, but also for the patients, including fewer mistakes than with paper records, better and more efficient healthcare, more organised historical data and outcomes, improved diagnosis and therapy treatment, better assessment on who needs screenings and preventive care, improved privacy and security for patient health data, encouraging data-based decision-making, providing follow-up support, and gives patients access to their own personal information.

Aside from that, EMRs offers numerous benefits not only for the clinics, but also for the patients, for example:

1. Fewer mistakes than with paper records

   With EMRs, problems like illegible handwriting or missing data are significantly less likely to happen than they may be with paper records. The risk of medical errors is reduced because to the accuracy of data entry and retrieval possible with electronic records for healthcare practitioners.
2. Better and more efficient healthcare

   Healthcare professionals can easily obtain thorough and current patient information thanks to EMRs. As a result, patients receive more effective and coordinated care. This also helps to improve coordination and communication among healthcare providers.
3. Analyse historical data and outcomes

EMRs make it possible to examine a patient's medical history, including previous diagnoses, treatments, and results. For better patient care, this data can be utilised to spot trends, enhance treatment strategies, and make informed choices.

4. Improve diagnosis and therapy
   The full medical histories of patients, including lab findings, radiology reports, and previous treatments, are accessible to healthcare professionals thanks to EMRs. Having access to such thorough data makes it easier to provide an accurate diagnosis and creates individualised treatment strategies.
5. Assess who needs screenings and preventive care
   Depending on their age and medical history, EMRs can identify individuals who need screenings, shots, or other preventative treatment. This facilitates proactive identification and management of potential health hazards by healthcare professionals, resulting in early intervention and enhanced preventive treatment.
6. Improved privacy and security
   Advanced encryption and access controls can be used to secure electronic records, lowering the possibility of unauthorised access and preserving patient privacy. As an added security feature, EMRs provide audit trails that show when and by whom the data was accessed.
7. Data-based decision-making
   EMRs offer a lot of data that may be analysed to spot trends, gauge results, and assess the efficiency of treatment strategies. Informed decisions may be made, and patient care is continuously improved thanks to this data-driven approach.
8. Follow-up support
   EMRs can be used to give patients follow-up help by reminding them, linking to pertinent resources, and offering self-care guidance. Better health results result from patients staying involved in their own healthcare and following treatment regimens.
9. Patients' accessibility
   EMRs can provide patients with access to their personal medical information, including prescription lists, test results, and treatment guidelines. As a result, patients are more equipped to take an active role in their healthcare, comprehend their problems, and choose the best course of therapy for themselves.

Overall, implementing EMRs in clinics has many benefits, such as increased effectiveness, cost savings, greater patient care, and better provider coordination. Moreover, employing

EMRs benefits patients in a variety of ways, including better accuracy, higher efficiency, increased privacy and security, individualised care, and patient empowerment.

Chosen Treatment Course: COVID-19

Introduction:

COVID-19 also known as the corona virus disease which was caused by the virus SARS-CoV-2. It is a highly contagious virus that are spread through respiratory systems through bodily fluid of infected people. The first recorded patient with this virus is from Wuhan, China in the late 2019. Ever since, the virus has spread across the world infecting millions.

Most people infected with the virus will experience mild flu like symptoms in their respiratory system. People infected can recover without any special treatments, but if there are complications, infected people are advised to seek medical help. These complications may include cardiovascular diseases, diabetes, chronic respiratory disease, or cancer. The virus may also be deadly towards people without any complications, hence why we need to prevent the spread of this virus.

To prevent the spread of this virus, we are advised to keep a healthy distance with other people and to wear mask at all time to block any unwanted bodily fluids to be transmitted to our orifices. Washing hands regularly will also help as hygiene can kill the virus' traces from our hands. You are also advised to get a vaccine shot for the virus. For people that are feeling unwell, they are advised to stay inside their own homes to recover before going out again.

Medicine:

For the treatment of COVID-19, a number of medications have been used or are currently being studied. It's crucial to keep in mind that the therapeutic landscape is always changing, and new advancements could have happened since then. Here are some of the notable medications:

The first medicine is Remdesivir. It is an antiviral drug that were first created to treat Ebola. The U.S. FDA granted Remdesivir an Emergency Use Authorization (EUA) for the treatment of COVID-19 in some circumstances. It is given intravenously, and it works by preventing the virus from replicating.

The second medicine is Dexamethasone. This is a corticosteroid that has showed promise in lowering mortality in COVID-19 individuals with severe illness. It aids in reducing the cytokine storm, a hyperactive immunological reaction. When a patient needs oxygen support while hospitalised, dexamethasone is frequently utilised.

The third medicine is Tocilizumab. It is a medication that suppresses the immune system and is used to treat rheumatoid arthritis. Tocilizumab inhibits the interleukin-6 (IL-6) pathway, which is thought to be responsible for the COVID-19-related inflammatory response. It is applied in cases of extreme inflammation.

The fourth medicine is baricitinib. It is an oral Janus kinase (JAK) inhibitor that was first created for the treatment of rheumatoid arthritis. It was granted EUA by the U.S. FDA to be used in conjunction with remdesivir for the treatment of COVID-19-positive hospitalised patients.

The fifth medicine is Convalescent plasma. Plasma from patients who have recovered from COVID-19 is used in this treatment. Antibodies in the plasma may aid in the virus's defence. Although its efficacy is still being investigated, it is employed in specific situations.

For the most recent details on COVID-19 treatment choices, it's vital to speak with medical experts or to consult updated guidelines and clinical trials. Since the last update, new drugs may have been authorised or continuing studies may have revealed new information.

Procedure:

The best course of action for treating COVID-19 is always changing. Methods that target the virus directly (such as antivirals, passive immunity, and interferons) are more likely to be effective early in the course of infection based on the pathophysiology of COVID-19, whereas techniques that modify the immune response may be more effective later in the illness course.

In the absence of dyspnoea, mild illness is characterised by fever, malaise, cough, upper respiratory symptoms, and/or less prevalent COVID-19 characteristics. The majority of these people don't need to be hospitalised. On the other hand, patients who have dyspnoea should likely be hospitalised since it indicates an illness of at least moderate severity. However, the presence of any of the following signs or symptoms implies severe illness. Patients might have infiltrates on chest imaging and yet be deemed to have moderate disease.

- Hyperoxia (oxygen saturation of less than 94% in room air)
- Need for ventilator support or oxygenation

Because of this, the treatment protocol for COVID-19 is divided into two kinds: patients without oxygen requirements and patients with oxygen requirements, with exception to the special cases like the treatment to pregnant and breastfeeding woman, or to people with HIV. For people who do not need oxygen support treatment, whether patients have clinical, or laboratory risk factors linked to the development of a more serious illness and the cause of hospitalisation determines the treatment plan.

- Doctors suggest remdesivir for patients who had COVID-19 hospitalisations and had risk factors for serious illness. Remdesivir may speed up recovery in these individuals, according to trial results, albeit the exact extent of the improvement is unknown. Dexamethasone should be avoided in these individuals, as it may lead to poorer results. Remdesivir administration, dosage, and supporting data are covered elsewhere.
- Doctors assess eligibility for medicines approved for certain high-risk outpatients for persons with risk factors for severe illness who were hospitalised for a reason other than COVID-19 and had incidental SARS-CoV-2 infection (or acquired infection during hospitalisation). These treatments' administration, dose, and effectiveness for non-severe COVID-19 are covered in depth elsewhere.

- Doctors advise just supportive care for individuals who do not require oxygen and who have no risk factors for the development of severe illness.

For hospitalised patients with severe illness and COVID-19-related oxygen supplementation needs, you will be given priority to COVID-19-specific treatment. The method is determined by whether oxygen or ventilation is needed.

- Doctors advise remdesivir as well as low-dose dexamethasone for the majority of patients receiving low-flow supplemental oxygen. However, if the patient is immunocompromised and early in the course of the disease, and is stable on minimum supplementary oxygen, it is permissible to forego dexamethasone and use remdesivir alone, especially if there are reservations about using glucocorticoids. It is unclear if any patients in this relatively heterogeneous group would benefit more than others from dexamethasone, and those stable on minimal oxygen may not have the excess inflammation that dexamethasone is intended to treat. Trial data suggest that dexamethasone reduces mortality in patients who are receiving non-invasive oxygen supplementation. Moreover, a few trials also suggest that remdesivir may improve survival and reduce mechanical ventilation in patients on non-invasive oxygen supplementation.
- Doctors advise low-dose dexamethasone for individuals receiving non-invasive ventilation or high-flow oxygen. We also recommend supplementary baricitinib or tocilizumab for patients who are between 24 and 48 hours away from receiving ICU-level treatment or being admitted there, as well as within 96 hours after being admitted to the hospital. According to trial results, dexamethasone lowers mortality in individuals receiving non-invasive oxygen supplementation, and baricitinib or tocilizumab may further lower mortality. Aside from that, based on the theoretical advantage of combining antiviral medication with anti-inflammatory treatment, we also recommend remdesivir, especially in immunocompromised individuals.
- Doctors advise low-dose dexamethasone for patients who require mechanical ventilation or extracorporeal membrane oxygenation (ECMO), and adjunctive tocilizumab or baricitinib for those who are within 24 to 48 hours of ICU admission and within 96 hours of hospitalisation. According to trial findings, when given early in the hospitalisation process, dexamethasone and the inclusion of tocilizumab or baricitinib each enhance mortality in this cohort. It is advisable to not initiating remdesivir in this population on a regular basis. Remdesivir can be added to patients

who have only been intubated for 24 to 48 hours, although it is unclear if doing so would have any therapeutic advantages. Remdesivir can be continued by patients who started it when they needed less oxygenation assistance.

Analysis:

There are two types of tests that are run in order to find out if the patient is infected with the covid-19 virus which are the Polymerase Chain Reaction (PCR) Test and the Antigen tests. The PCR test takes longer time than the antigen test but is more accurate than the antigen test.

The first one is the PCR test, this is done through nucleic acid amplification test (NAAT), which will be done in a lab, that will take approximately three days before being sent back to the patient for results. The first step is using a swab stick to extract substance from the respiratory system (nose, nasal cavity) in order to be put and brought to a lab. In the lab the genetic material is separated from the substance leaving the rest to be analyzed. A machine called the thermal cycler is used along with some special chemical to identify the covid-19 virus through heating , cooling, and amplifications in the cycle until the virus is visible, then after that, the machine will notify in the lab if the substance sample tested positive or not.

The second one is Antigen test, where the first step is similar to that of the PCR test, however, it differs, where the sample doesn't go to the lab, instead an antigen test will be put in a buffer and reagent and will be tested on a test card instead, which will show two lines for positive and one line for negative. This is done through detector molecules (antibodies) which will trap the viral antigen (which is a protein owned by the covid virus).

Data Processing Steps:

By integrating serialisation, our JavaFX programme will support turning input data into a serialised text file (txt), enabling effective data storage and retrieval in a portable format. In order to facilitate subsequent manipulation and reconstruction of the original data as needed, this method entails converting the input data into a binary representation that may be stored in a text file.

Here are the steps to carry out serialisation:

1. To begin, a Java class must be defined that implements the Serializable interface. This shows that the class's instances can be serialised.
2. The Java program's input data is then obtained. This could come from a database query, human input, or any other source.
3. The serialised data is then written to a file with the.txt extension using an output stream, such as a FileOutputStream.
4. Next, we build an instance of ObjectOutputStream that encloses the output stream. This object offers serialisation and writing techniques for objects.
5. We use the writeObject() method of the ObjectOutputStream and supply the input data object as the parameter to serialise the data.
6. To free up system resources, we close the streams, specifically the ObjectOutputStream and FileOutputStream. By doing this, the data is guaranteed to be flushed and written to the file correctly.

Since our program will support all four CRUD operations including create, read, update & delete, we incorporate these steps for each operation:

1. Create Operation:

   We would do the previously specified serialisation procedures, where we serialise the input data and send it to the text file, to create a new record.

2. Read Operation:

   We would carry out the opposite procedure to read the serialised data from the text file. To read from the text file, we would first create an input stream, such as a FileInputStream, and then an ObjectInputStream instance to surround the input stream. We may deserialize the data and acquire the Java objects that represent the input data using the readObject() function of the ObjectInputStream.

3. Update Operation:

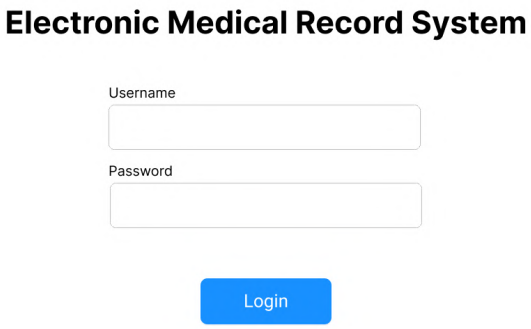
   In order to update a particular record, we would read the serialised data from the text file using the read operation. Following that, we can change the pertinent fields in the Java object that represents the record. We would serialise the updated object back to the text file after making the adjustments, replacing the old data.

4. Delete Operation:

   In order to delete a record, we would read the serialised data from the text file using the read operation. Depending on the needs, we would either erase or mark as deleted the pertinent Java object after getting it. After that, we would overwrite the text file while serialising the revised data but leaving out the deleted record.

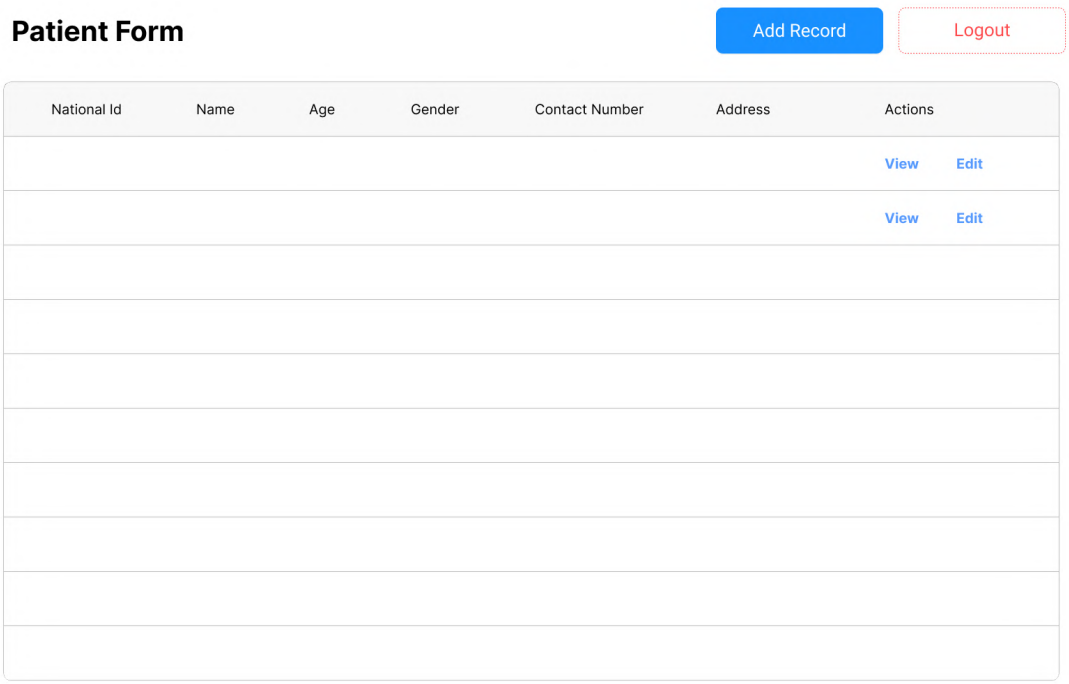
User Interface and Navigation:

Login Screen [Admin]



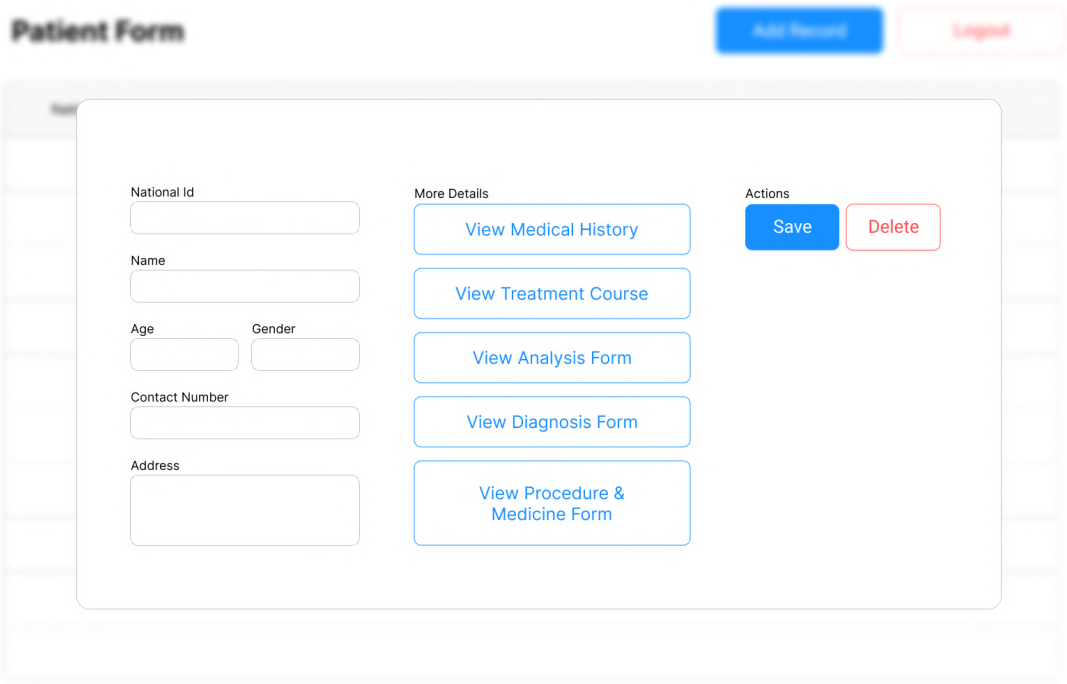
The user will be greeted with a login screen upon launch, and is required to enter the username and respective password. Then, press the Login button to access the system.

Patient Form [Admin]



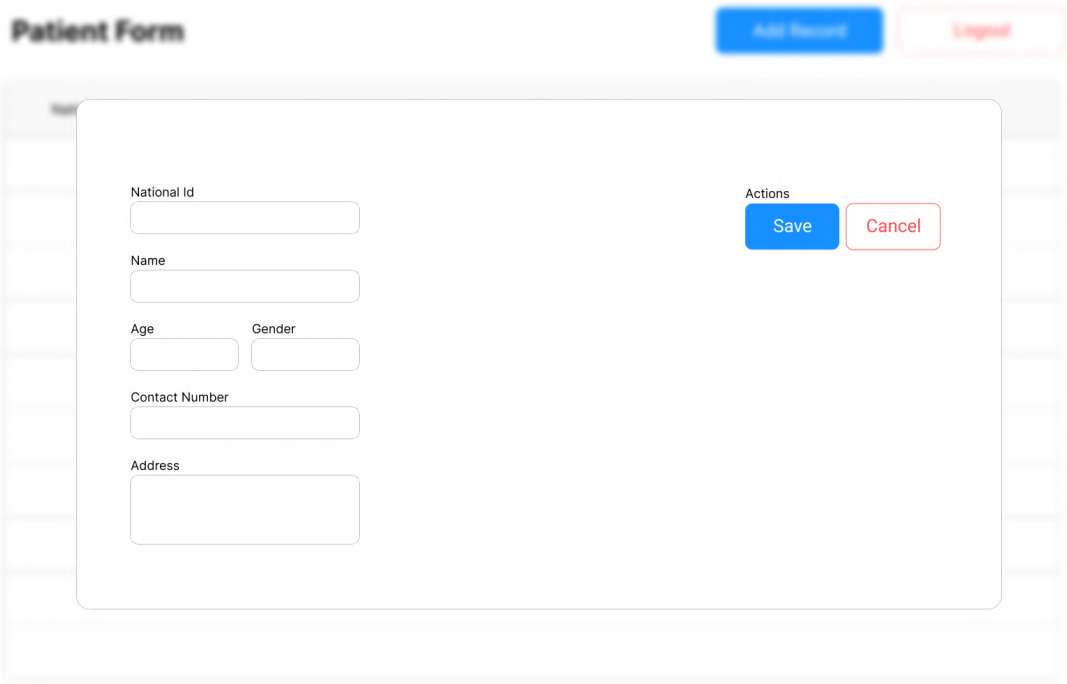
After logging into the system, the Patient Form will be shown. It includes a table of patients'. Data, showing their national ID, name, age, gender, contact number and address. On the top right corner, user can add new record to the system. "View" and "Edit" buttons allow the user to further access the data & make changes if needed. The "Logout" button is at top right corner for user to logout.

View Patient Form [Admin]



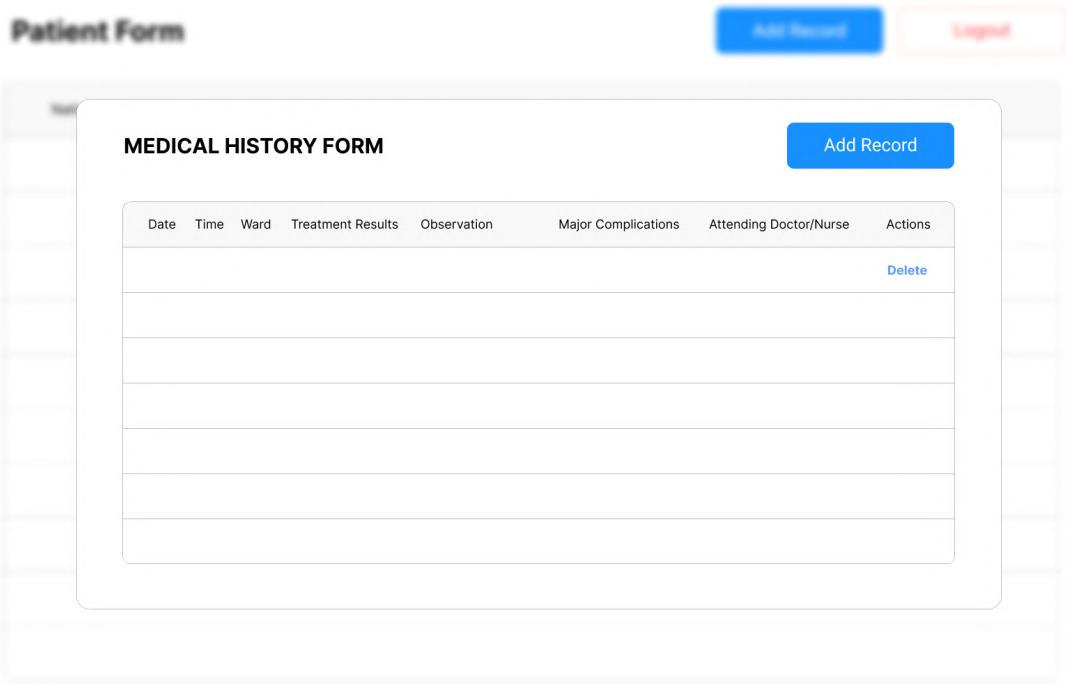
If the “View” button is clicked, the system will show all details, and also a few more buttons to access more data including the patient’s medical history, treatment course form, analysis form, diagnosis form, and procedure & medicine form. The “Save” and “Delete” button are at the right panel to save the changes (if any) and delete the patient’s record if necessary.

New Patient Form [Admin]



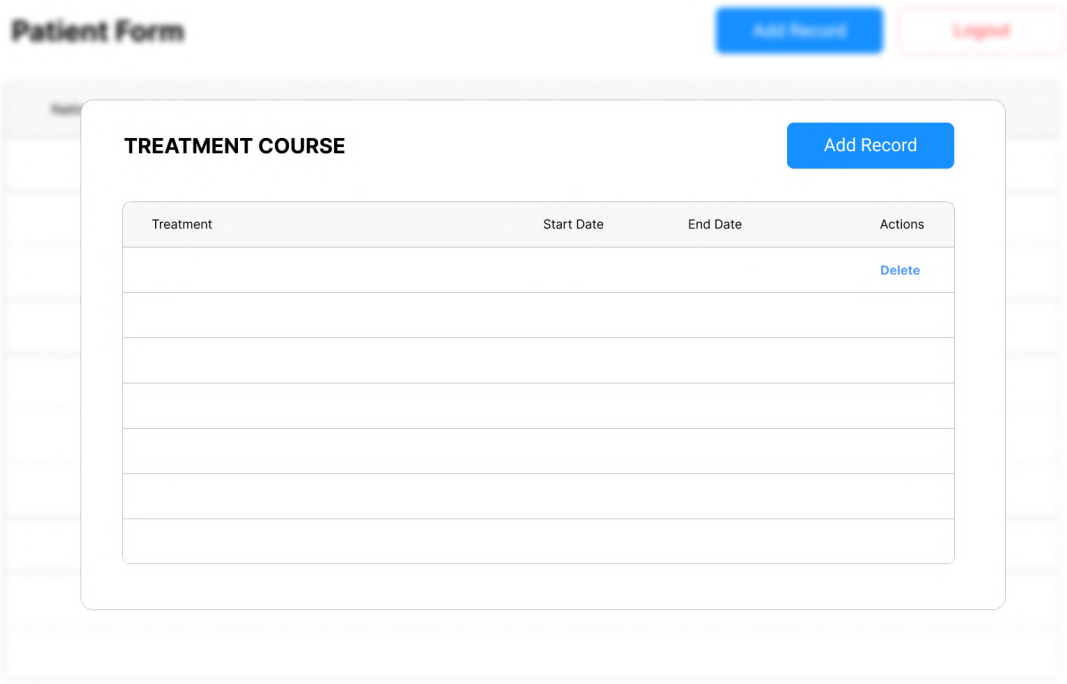
If the “Add Record” button is clicked instead, a similar screen will be shown, allowing the user to key in new data for the patient. The “Save” button at the right side should be pressed when done and “Cancel” button if user wants to abort the action.

Medical History Form [Admin]



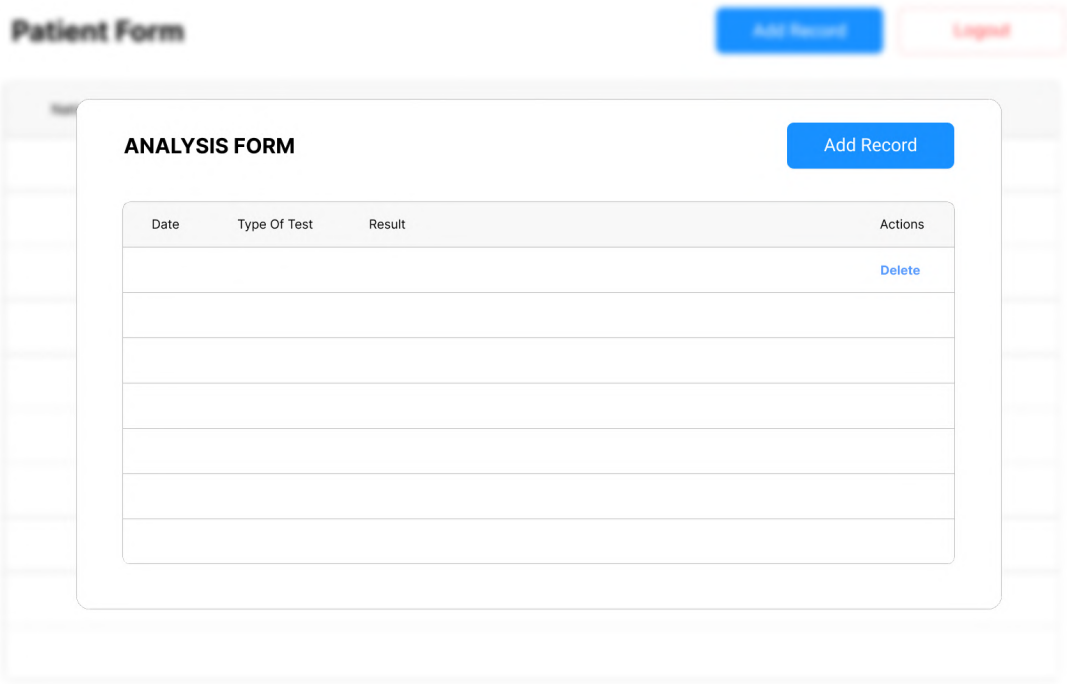
When the “View Medical History Form” button is pushed, the whole medical history of patient will be displayed in the form of table for ease of access. A “Delete” button is prepared, allowing user to delete certain record(s). If the user wants to add a new record to the medical history, “Add Button” on the top right corner can be clicked to perform the action.

Treatment Course Form [Admin]



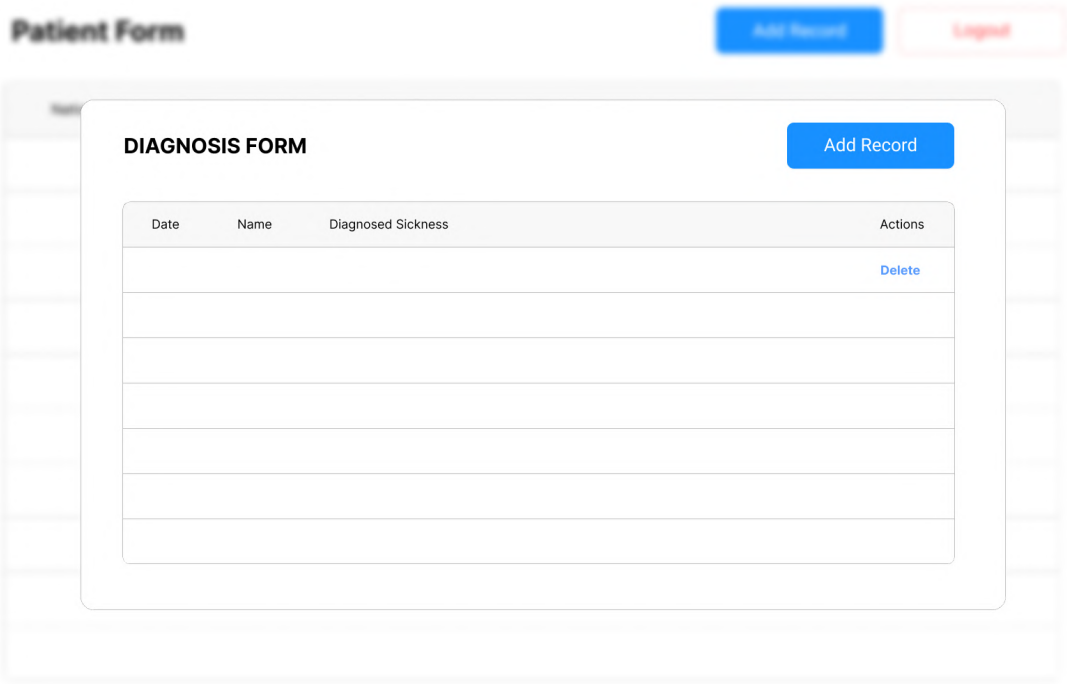
When the “View Treatment Course Form” button is pushed, all treatment courses of patient will be displayed in the form of table. A “Delete” button is also prepared, allowing user to delete certain record(s). If the user wants to add a new record to the database, “Add Button” on the top right corner can be clicked to perform the action.

Analysis Form [Admin]



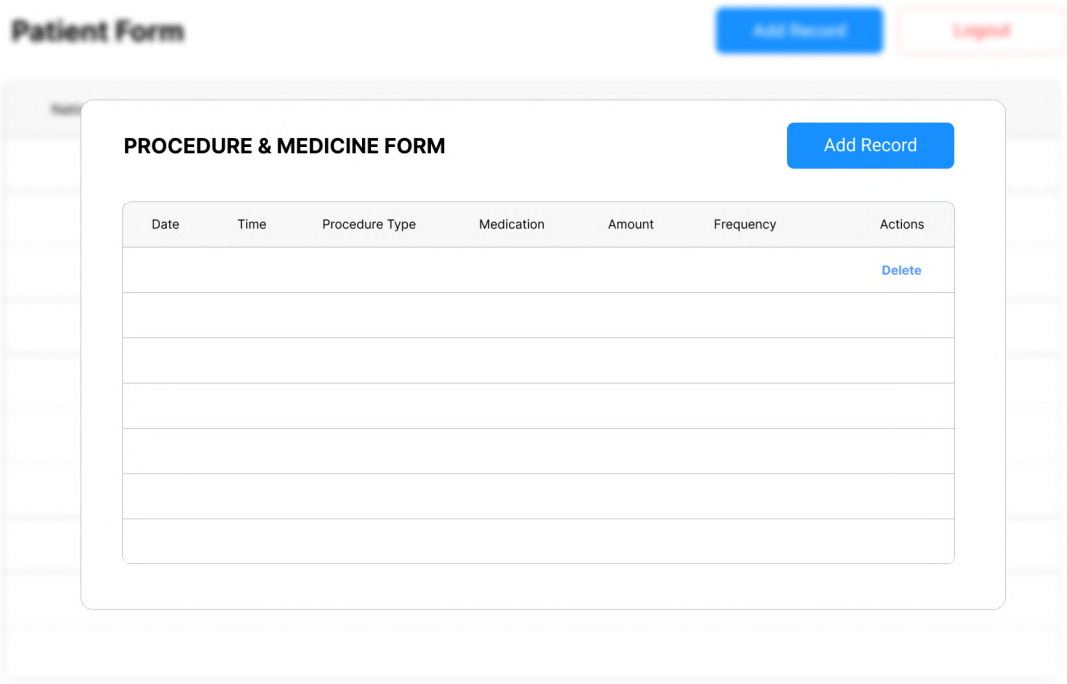
When the “View Analysis Form” button is pushed, every past analysis of patient will be displayed in the form of table. A “Delete” button is shown at the right side of every record, allowing user to delete certain record(s). If the user wants to add a new record to the analysis history, “Add Button” on the top right corner can be clicked to perform the action.

Diagnosis Form [Admin]



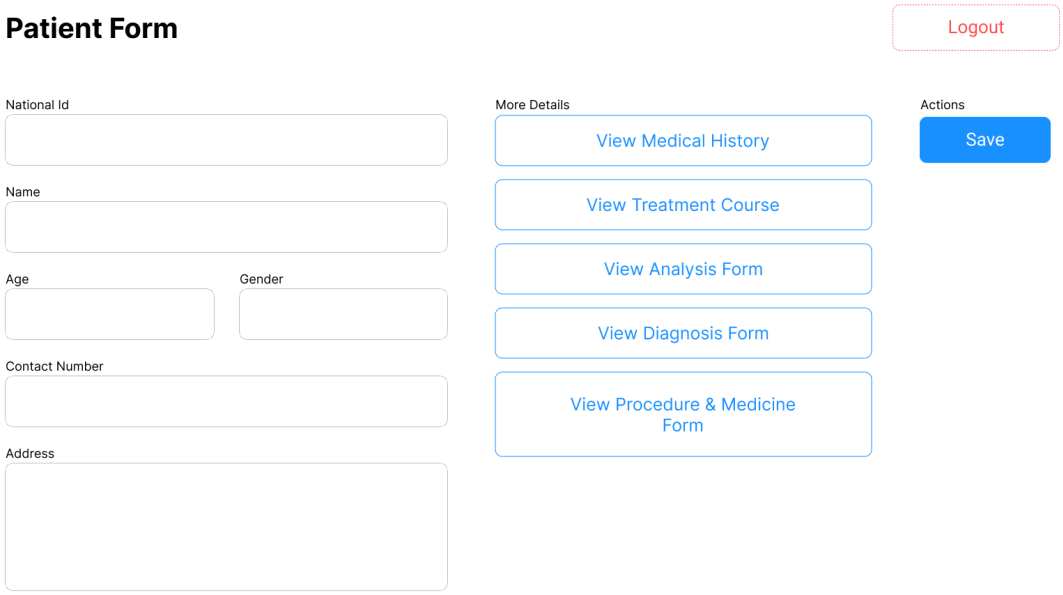
When the “View Diagnosis Form” button is pushed, the whole diagnosis history of patient will be displayed in the form of table. A “Delete” button is prepared, allowing user to delete certain record(s). If the user wants to add a new record to the diagnosis history, “Add Button” on the top right corner can be clicked to perform the action.

Procedure & Medicine Form [Admin]



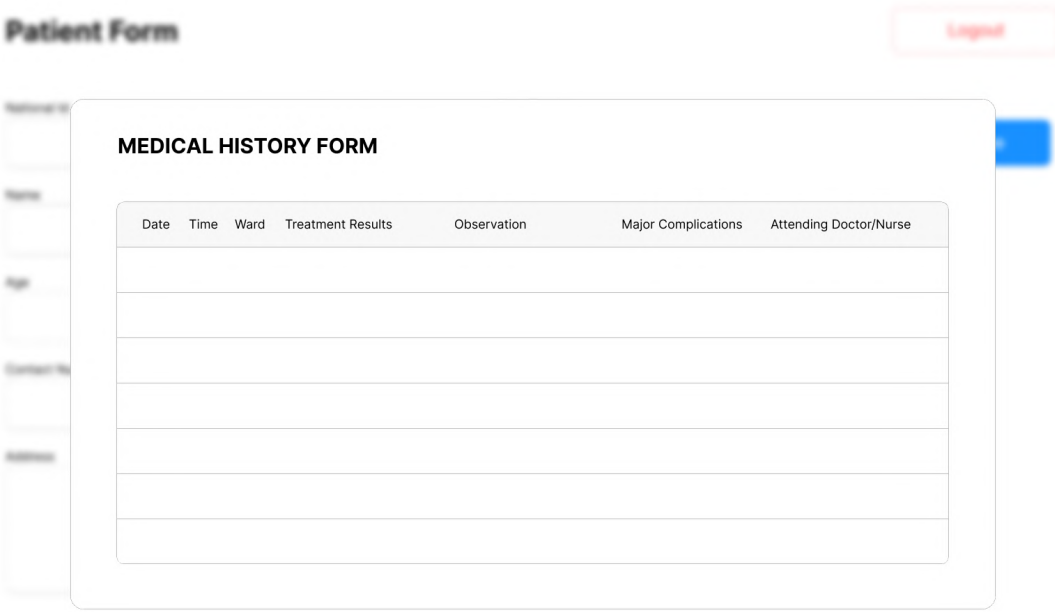
When the “View Procedure & Medicine Form” button is clicked, the whole medical history of patient will be displayed in the form of table. A “Delete” button is at the right side, allowing user to delete certain record(s). If the user wants to add a new record to the procedure & medicine history, the “Add Button” on the top right corner can be clicked to perform such action.

Patient Form [Patient]



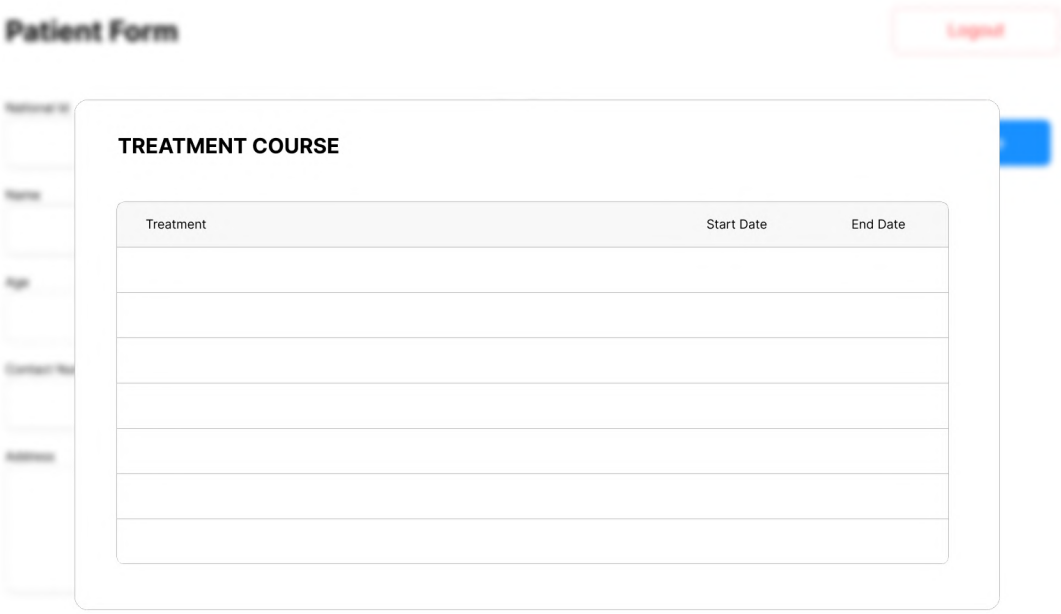
After logging into the system, it will show all details, and also a few more buttons to access more data including the patient's medical history, treatment course form, analysis form, diagnosis form, and procedure & medicine form. The "Save" button is at the right panel to save the changes if necessary. The "Logout" button is at top right corner for user to logout.

Medical History Form [Patient]



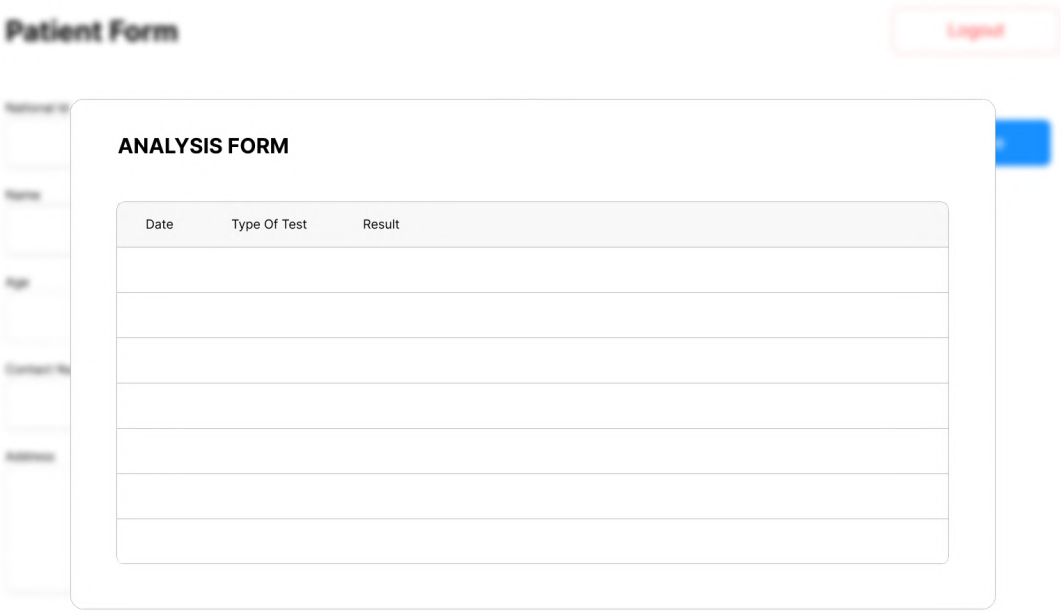
When the “View Medical History Form” button is pushed, the whole medical history of patient will be displayed in the form of table for ease of access.

Treatment Course Form [Patient]



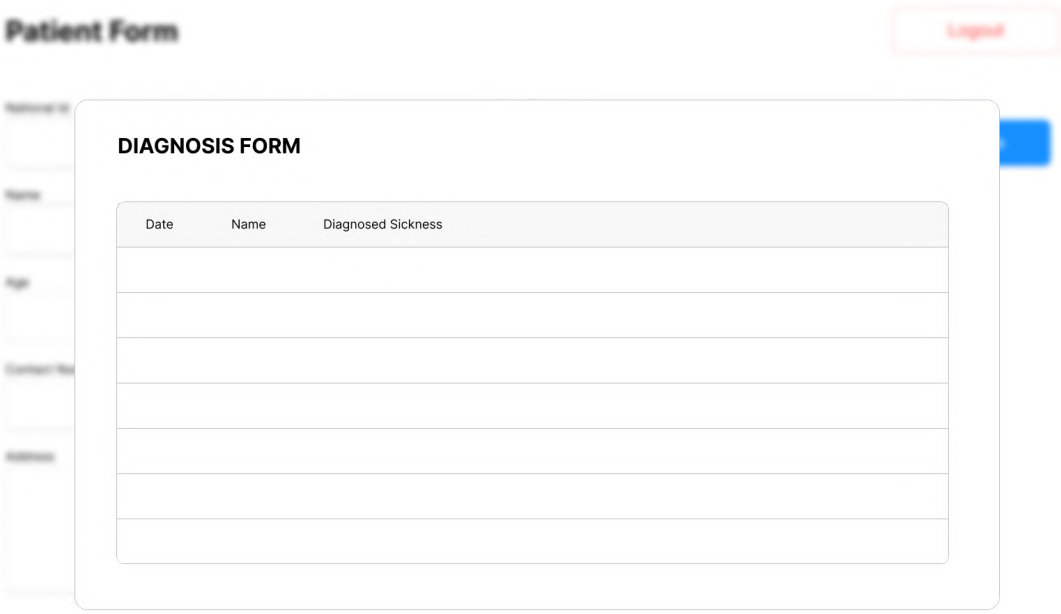
When the “View Treatment Course Form” button is pushed, all treatment courses of patient will be displayed in the form of table.

Analysis Form [Patient]



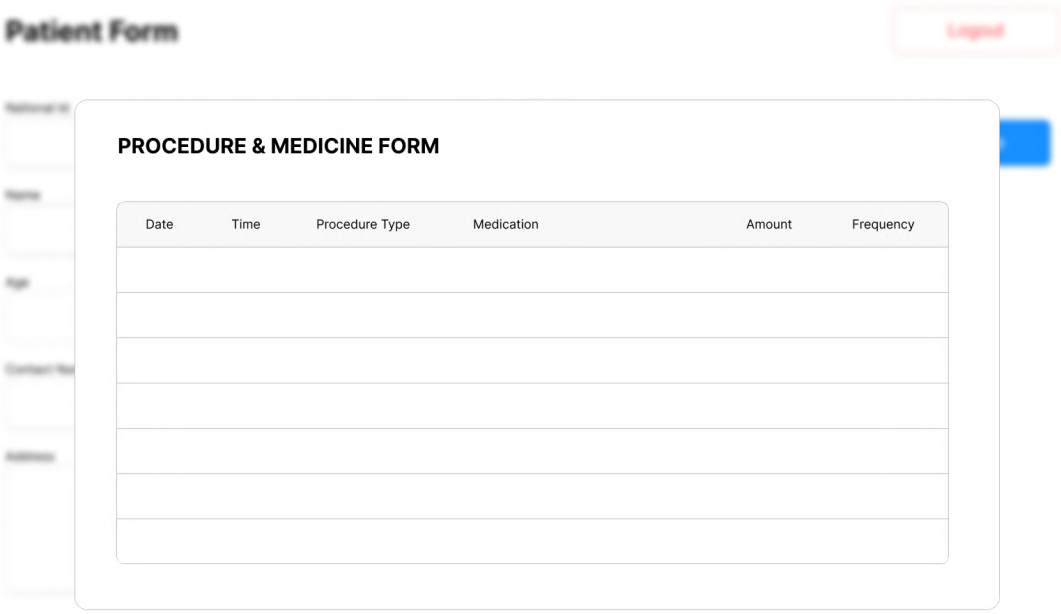
When the “View Analysis Form” button is pushed, every past analysis of patient will be displayed in the form of table.

Diagnosis Form [Patient]



When the “View Diagnosis Form” button is pushed, the whole diagnosis history of patient will be displayed in the form of table.

Procedure & Medicine Form [Patient]



When the “View Procedure & Medicine Form” button is clicked, the whole medical history of patient will be displayed in the form of table.

Future Developments and Upgrades:

Some future developments and upgrades to the application are needed for the application to run smoothly and to be reliable. Reliability of a hospital application is vital because patients' life is at stake. Some upgrades that can be done are:

1. Security and privacy

   Patient's data privacy and security are crucial and important to the application. In the future, the application needs to incorporate newest security measures like encryption and secure authentication.
2. Real time data synchronization

   The application can implement real time data synchronization that can update instantly across multiple devices to ensure that every data is up to date.
3. Advanced analytics and reporting

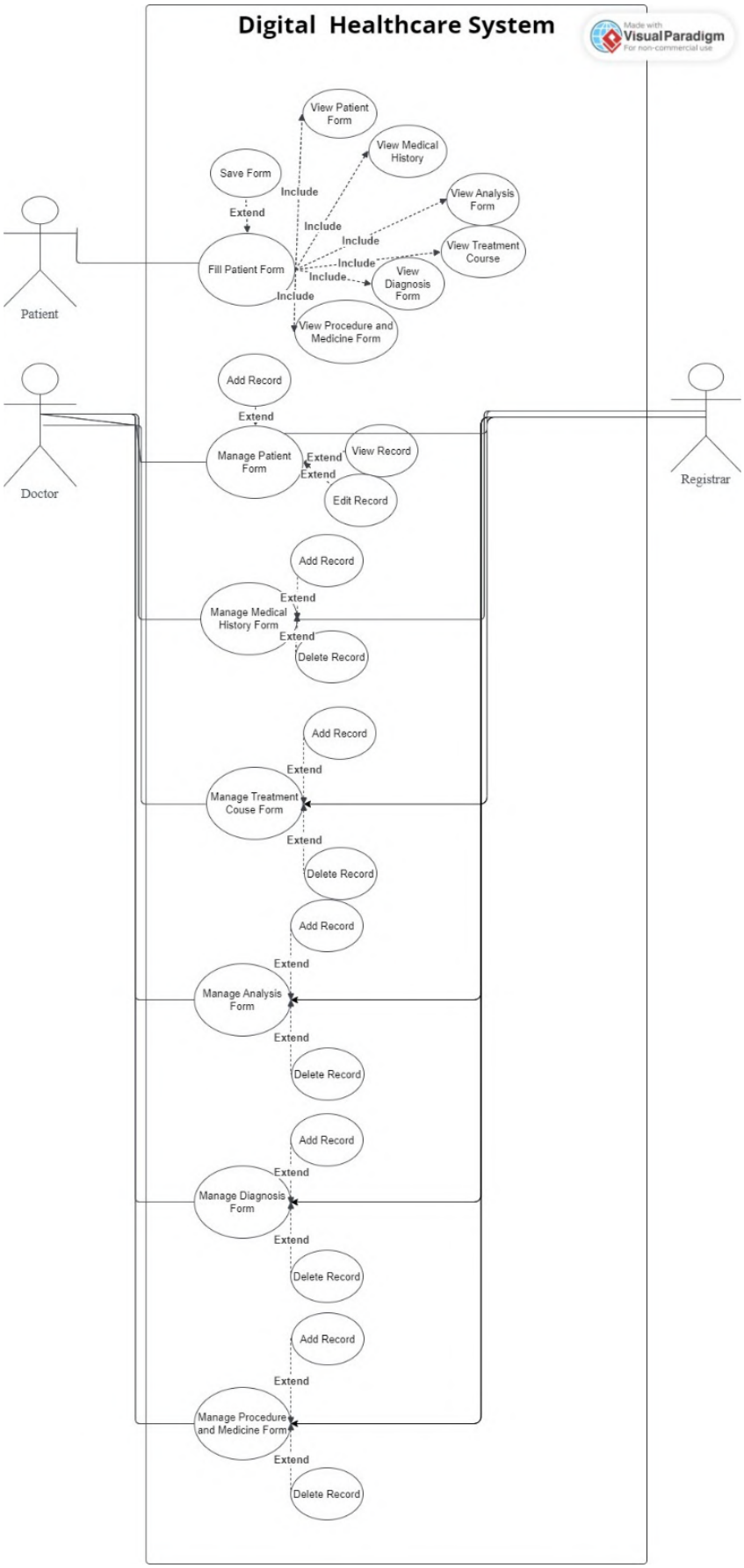
   The application can use advanced analytics to analyse incoming data to be processed and to generate a good report to the user.
4. Mobile application

   By developing a mobile application to the hospital application, patients can access their data easily through their phones and it can also be used to communicate with the doctor, while also being able to make a reminder to the patient for appointments.
5. Patient engagement features

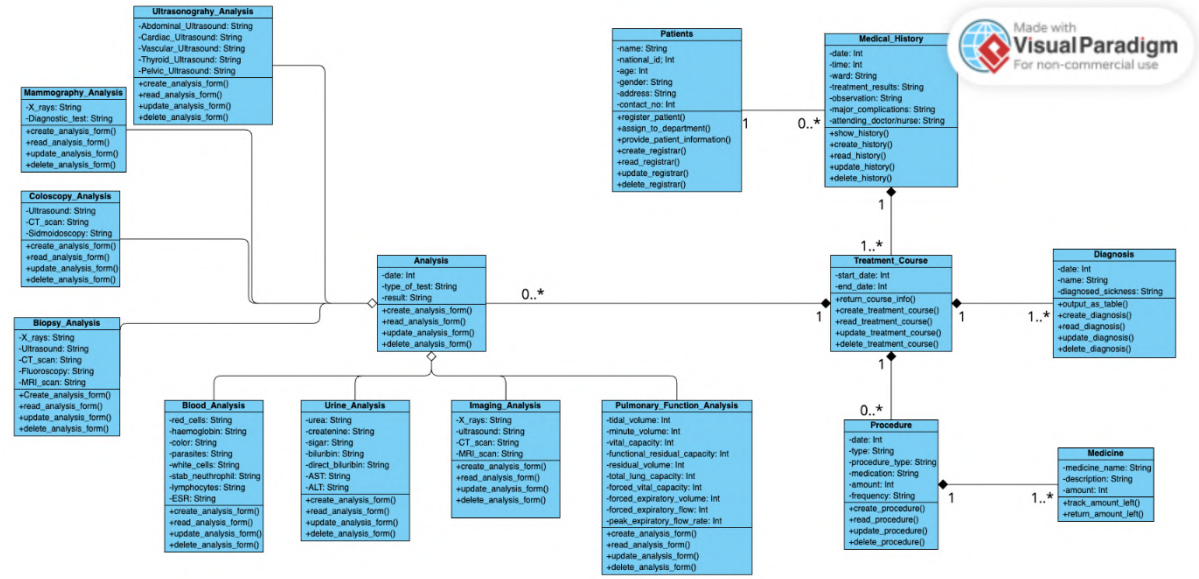
   The application can include features to be more engaged with the users, such as personalized health tips, medication reminders and access to educational features. These features can also be integrated with health tracking apps to track heartbeat, steps and many more for the application to record.

In conclusion, future development and upgrades of a hospital application would depend on technological advancements, regulatory requirements, and user needs.

Use Case Diagram(s)



Class Diagram(s)



Part B

The purpose of this document is to provide a step-by-step guide for starting and explaining the OOP concepts applied in the Electronic Medical Record System. It is made to take the place of the conventional paper medical records kept in healthcare facilities.

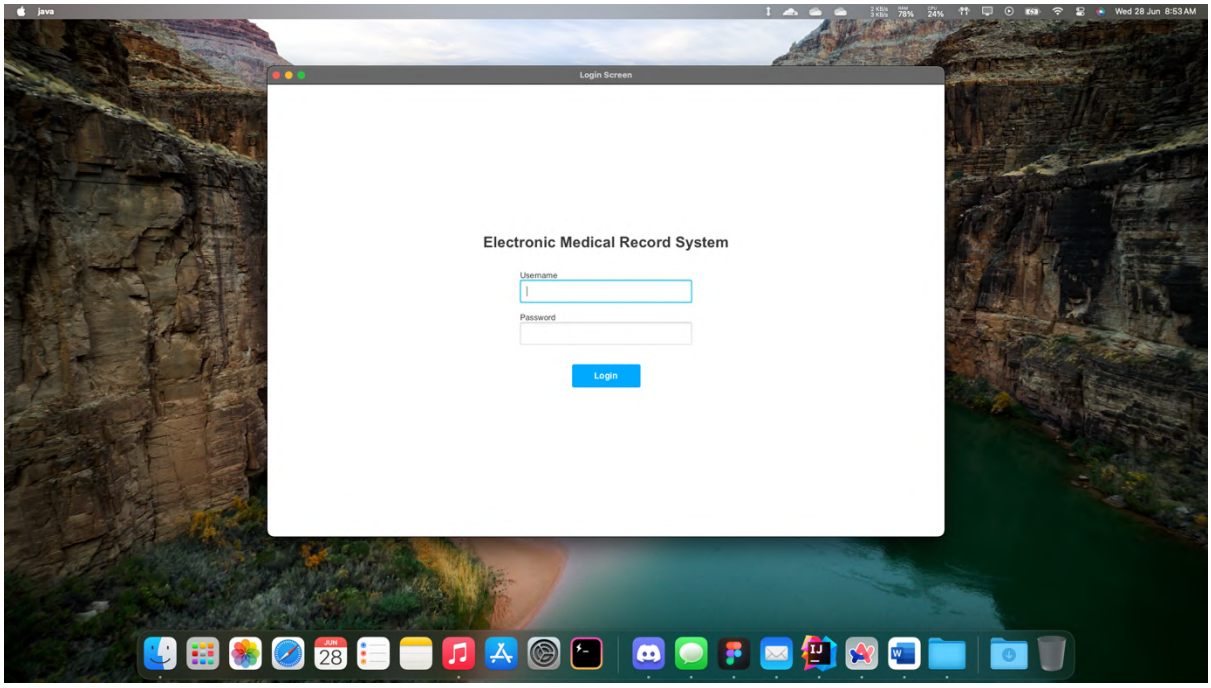
Before starting the electronic medical record system, ensure that the following prerequisites are met:

- Computer or device with a compatible operating system (e.g., Windows, macOS, Linux), the system works best with laptop/desktop screens
- Access credentials (username and password) provided by the system administrator

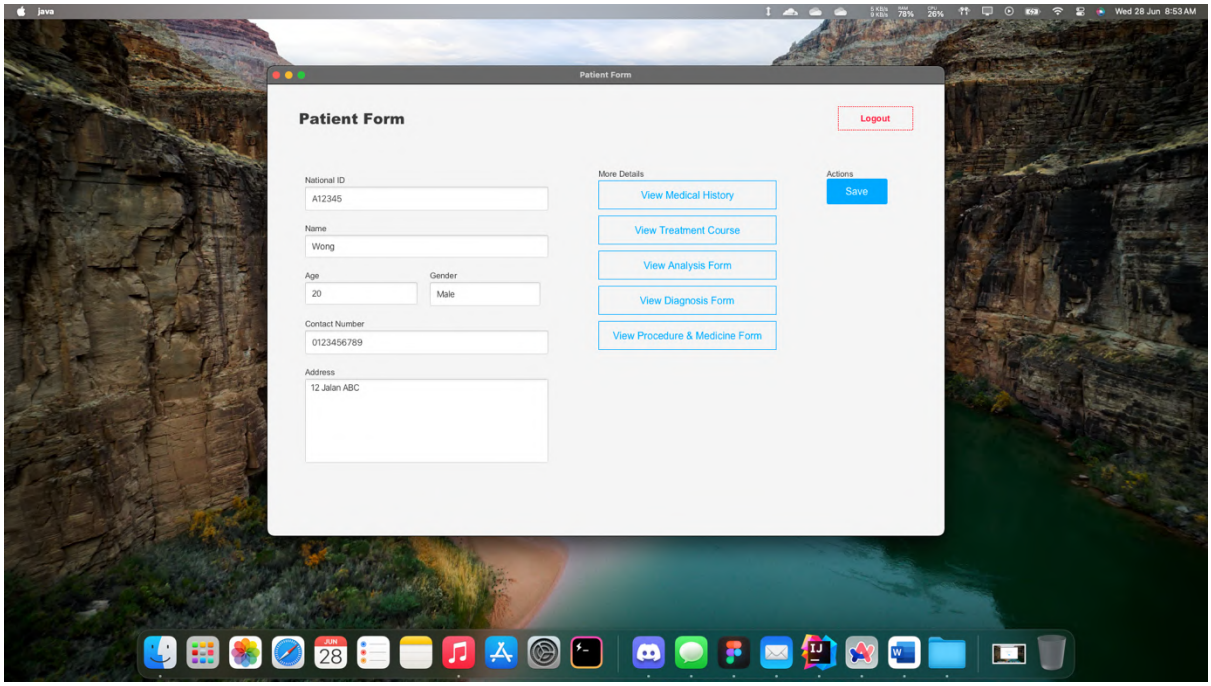
To access the electronic record system, launch the Main class then proceed with entering your login credentials.

User Interface Guideline

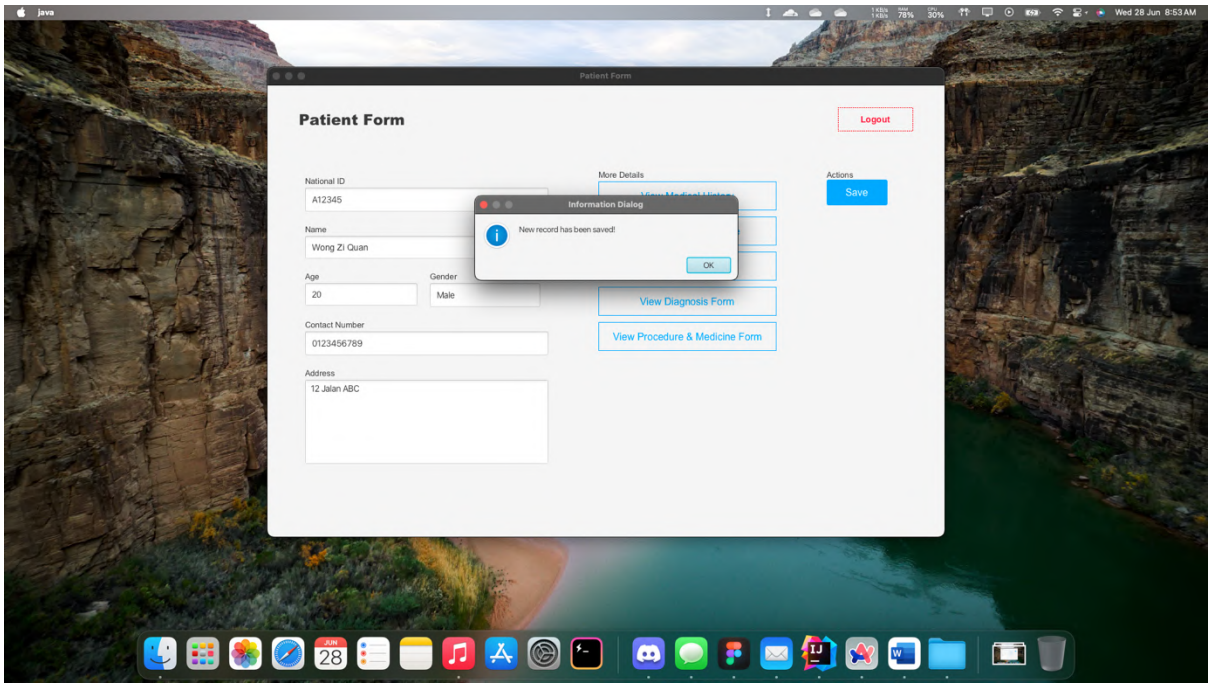
Once the system is launched, users will be greeted with a login screen. Enter the username and password provided by system administrator then press the Login button to access the system.



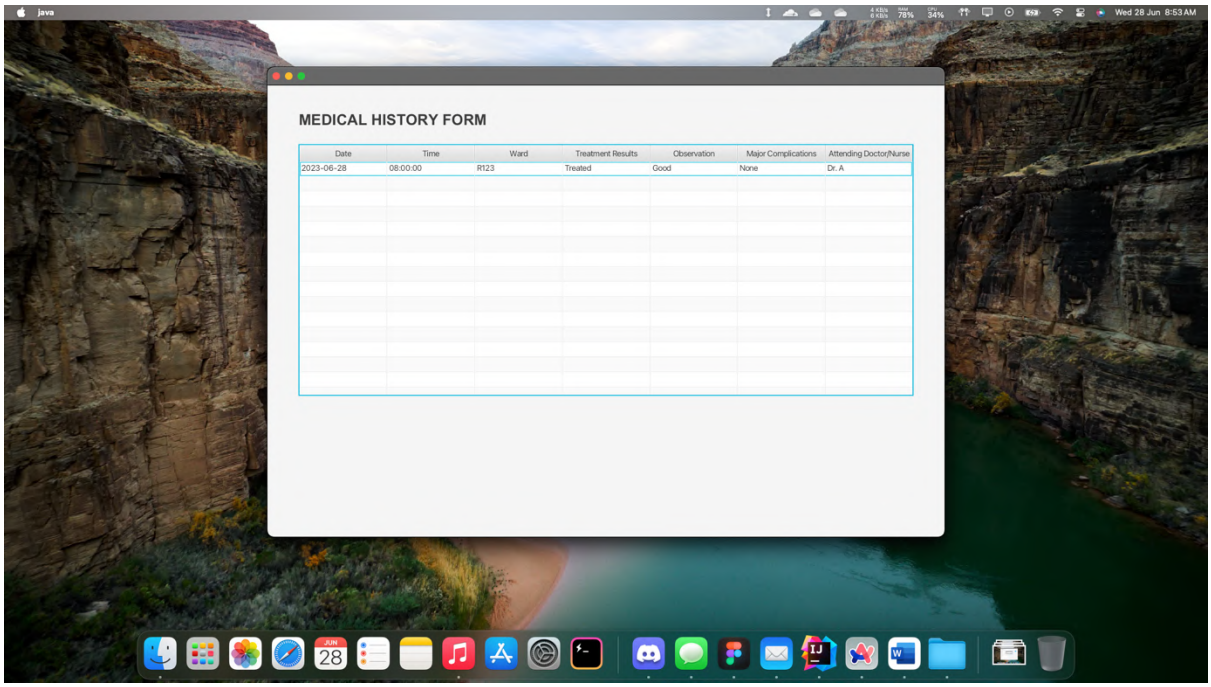
If you have logged into the system as a patient, you will see the screen below, displaying your personal information such as national ID, name, age, gender, contact number and address. Clicking the buttons (View Medical History, View Treatment Course, View Analysis Form, View Diagnosis Form, View Procedure & Medicine Form) will bring you to respective screens to access more of your data. If you are done, you may press the Logout button to quit the system. The system will logout itself by default when you shut down the application or device.



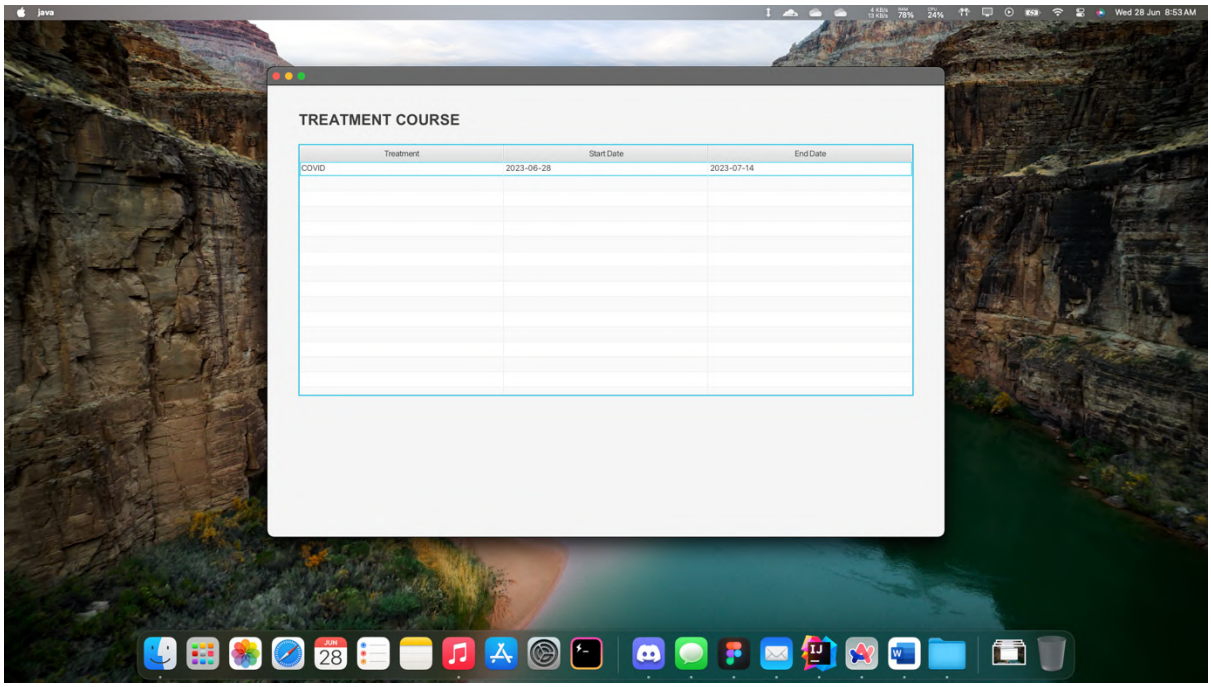
If you need to change any of the personal information, change the value in the given text fields then press the Save button to save the changes made. A popup dialog will then be shown to indicate that the changes have been saved successfully.



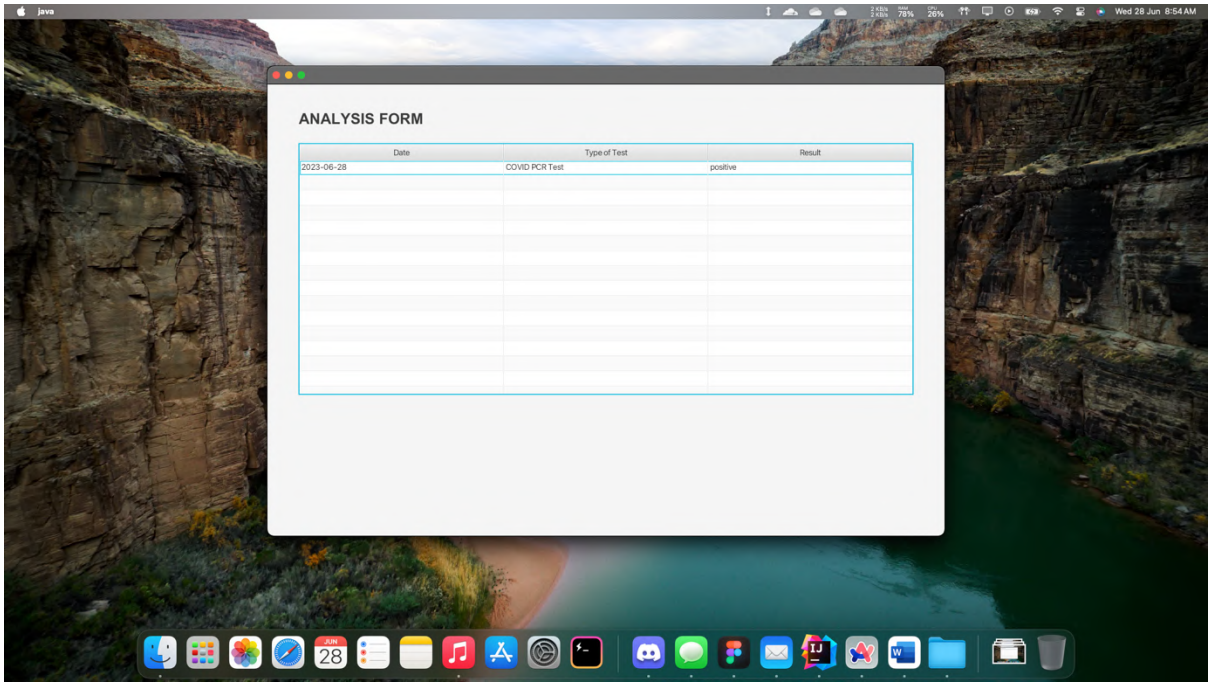
Clicking the View Medical History button will bring you to the Medical History Form to show all of your previous medical records history with details such as date, time, ward, treatment results, observation, major complications and attending doctor/nurse.



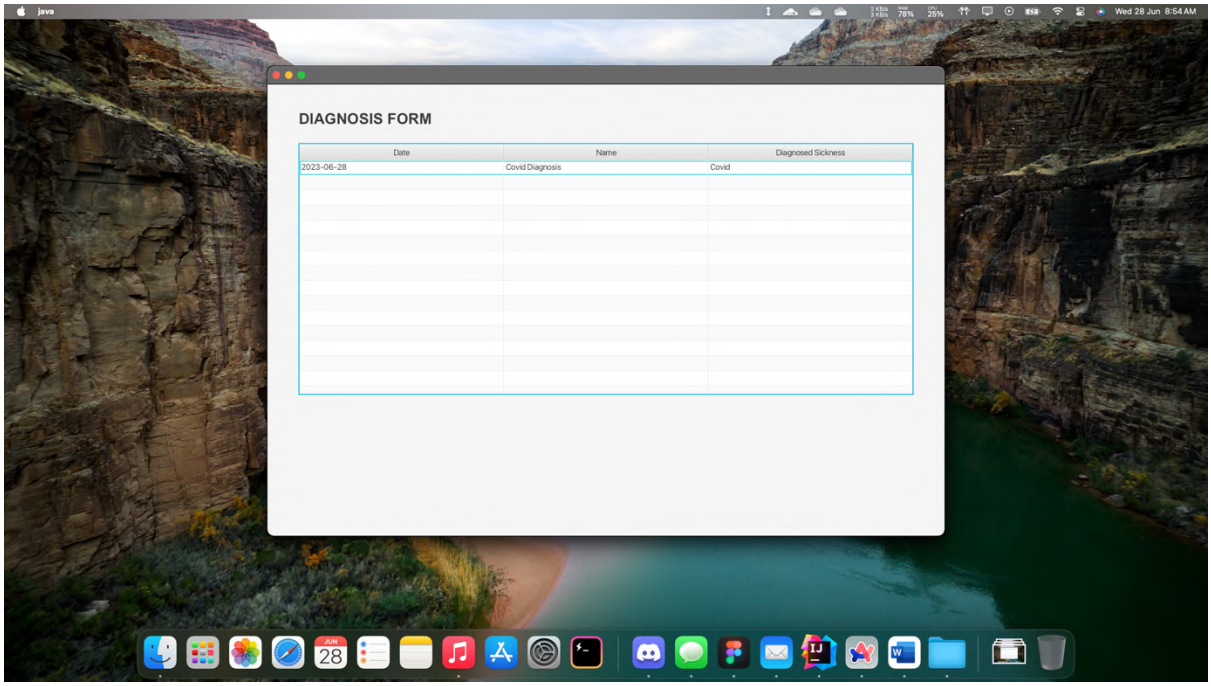
Clicking the View Treatment Course button will bring you to the Treatment Course screen to show all of your previous treatment course history with details such as treatment name, start date and end date.



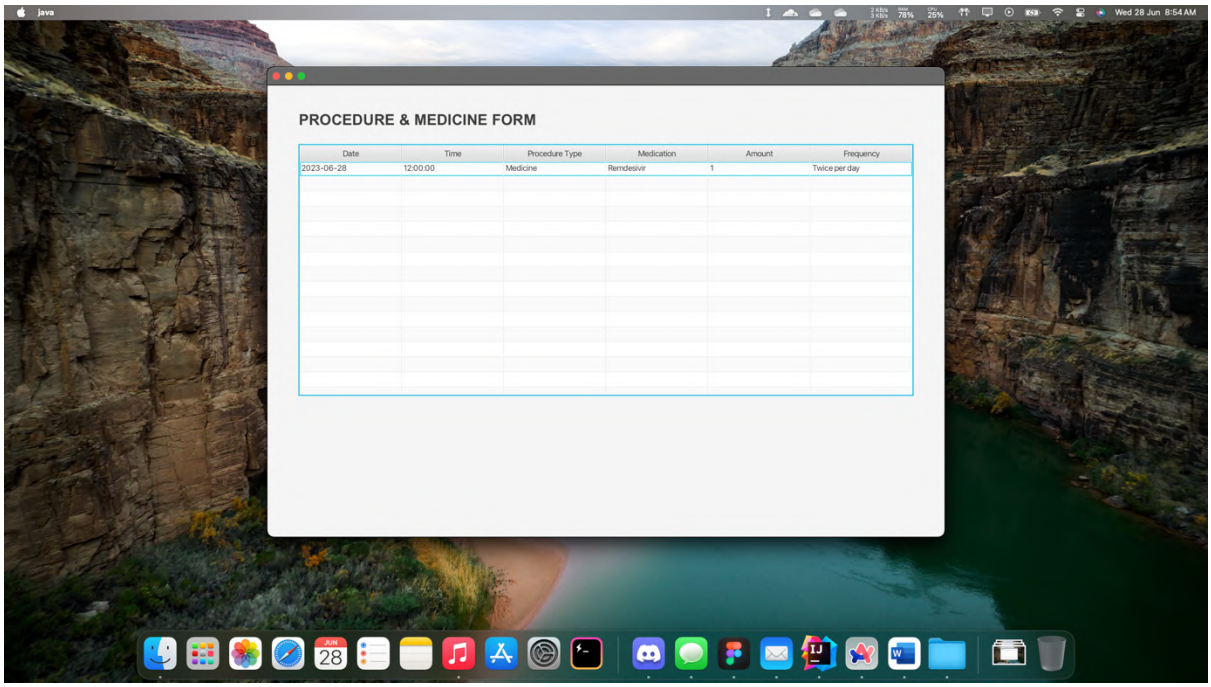
Clicking the View Analysis Form button will bring you to the Analysis Form to show all of your previous analysis history with details such as date, type of test and result.



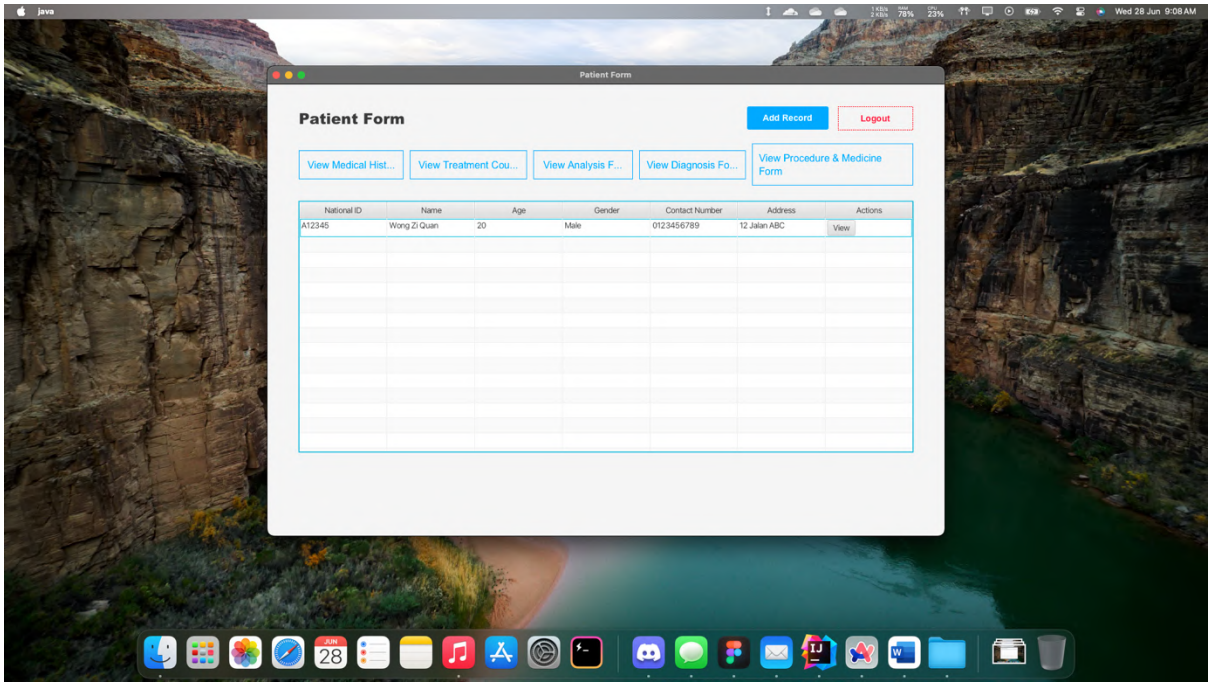
Clicking the View Diagnosis Form button will bring you to the Diagnosis Form to show all of your previous diagnosis history with details such as date, name and diagnosed sickness.



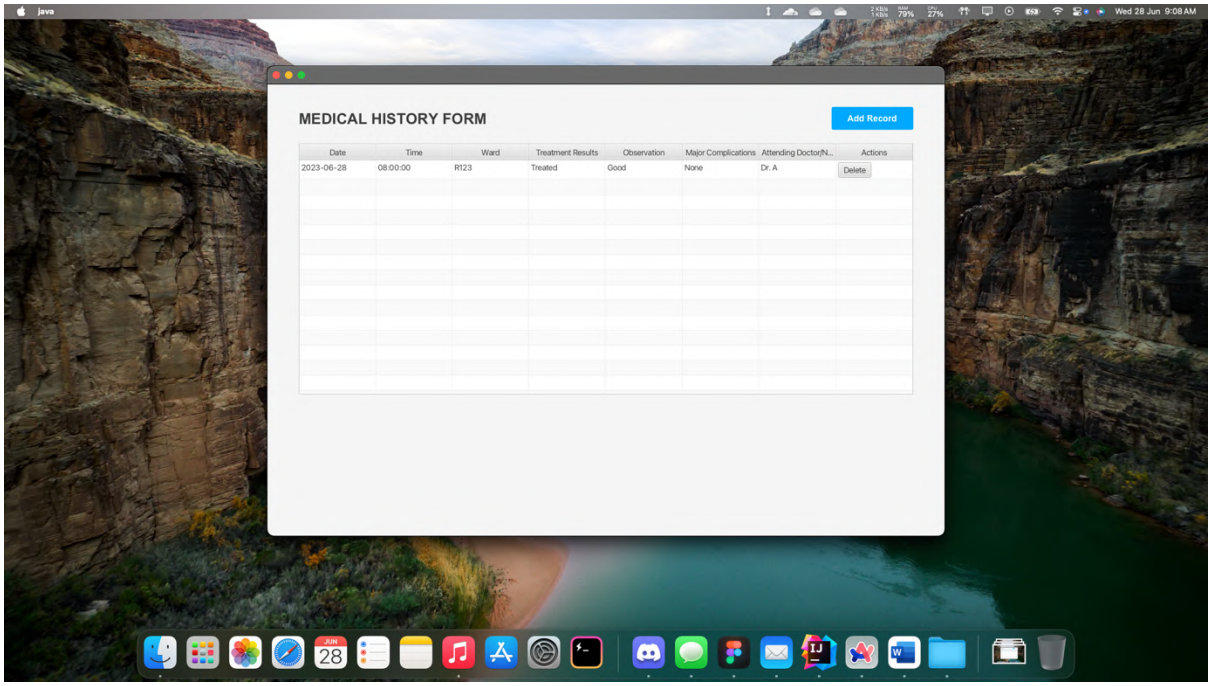
Clicking the View Procedure & Medicine Form button will bring you to the Procedure & Medicine Form to show all of your previous procedure & medicine history with details such as date, time, procedure type, medication, amount and frequency.



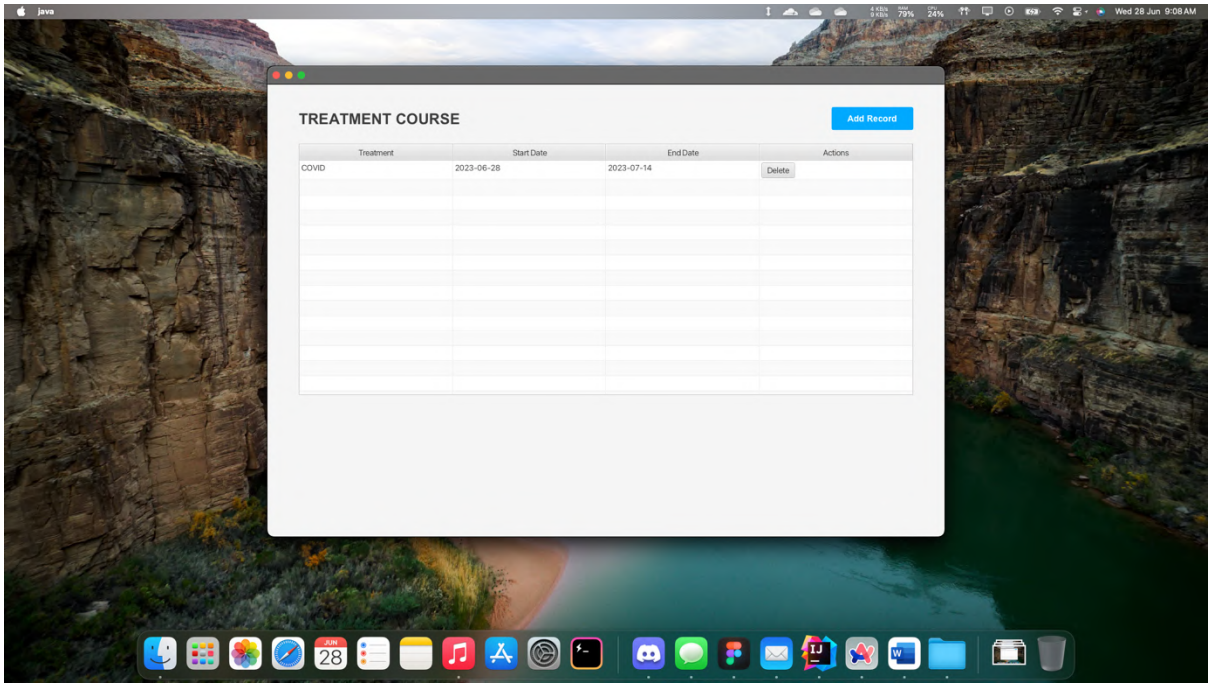
If you have logged into the system as an admin, you will see the screen below, displaying all patients' information such as national ID, name, age, gender, contact number and address. Clicking the buttons (View Medical History, View Treatment Course, View Analysis Form, View Diagnosis Form, View Procedure & Medicine Form) will bring you to respective screens to access more of their data. If you are done, you may press the Logout button to quit the system. The system will logout itself by default when you shut down the application or device.



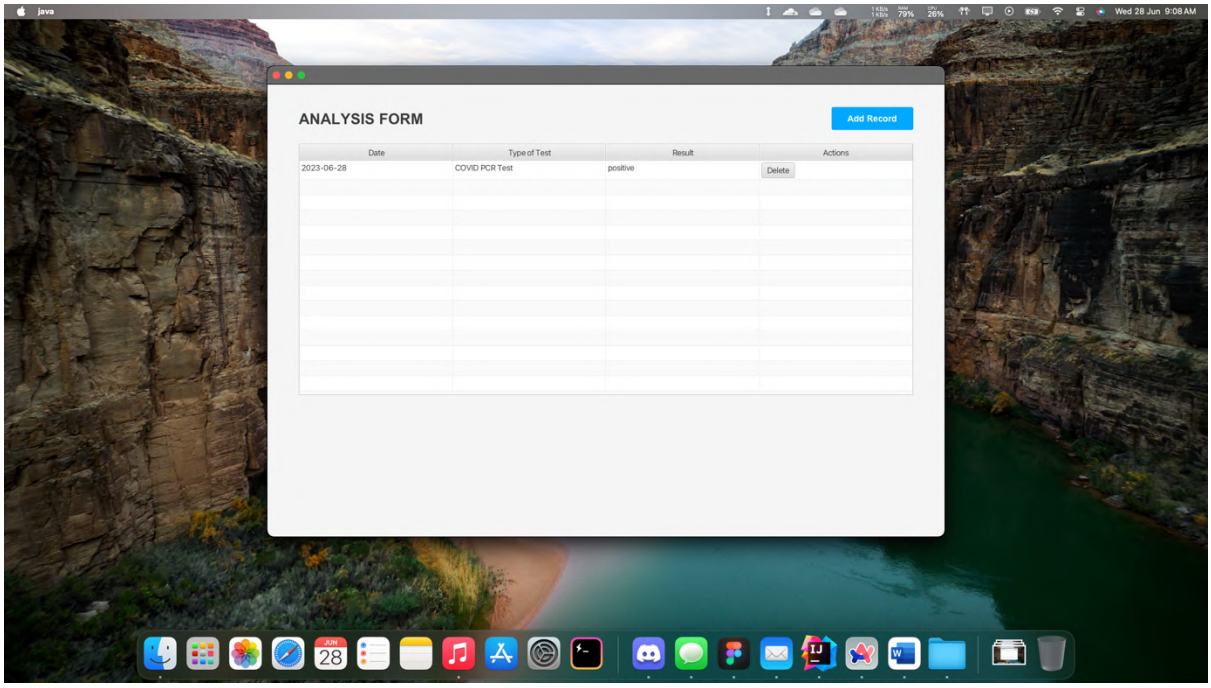
Clicking the View Medical History button will bring you to the Medical History Form to show all of the patients' medical records history with details such as date, time, ward, treatment results, observation, major complications and attending doctor/nurse. Pressing the Delete button will delete the record from the system. A popup dialog will also be shown to indicate that the record has been deleted.



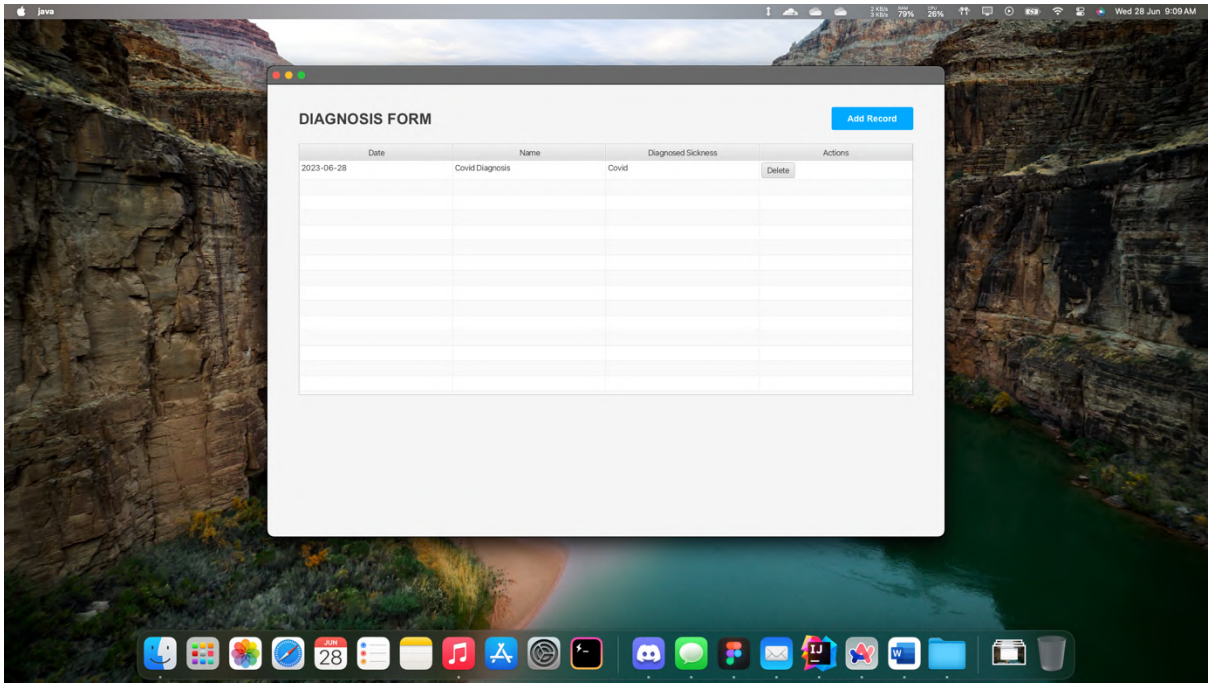
Clicking the View Treatment Course button will bring you to the Treatment Course screen to show all of the patients' previous treatment course history with details such as treatment name, start date and end date. Pressing the Delete button will delete the record from the system. A popup dialog will also be shown to indicate that the record has been deleted.



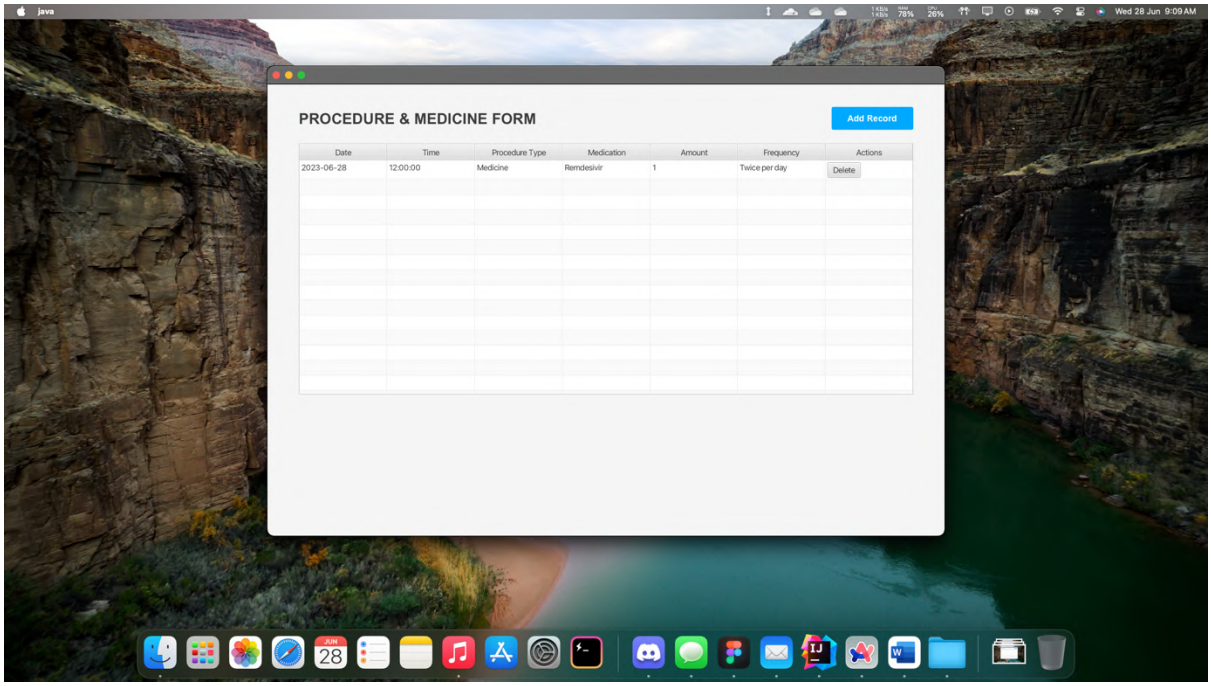
Clicking the View Analysis Form button will bring you to the Analysis Form to show all of the patients' previous analysis history with details such as date, type of test and result. Pressing the Delete button will delete the record from the system. A popup dialog will also be shown to indicate that the record has been deleted.



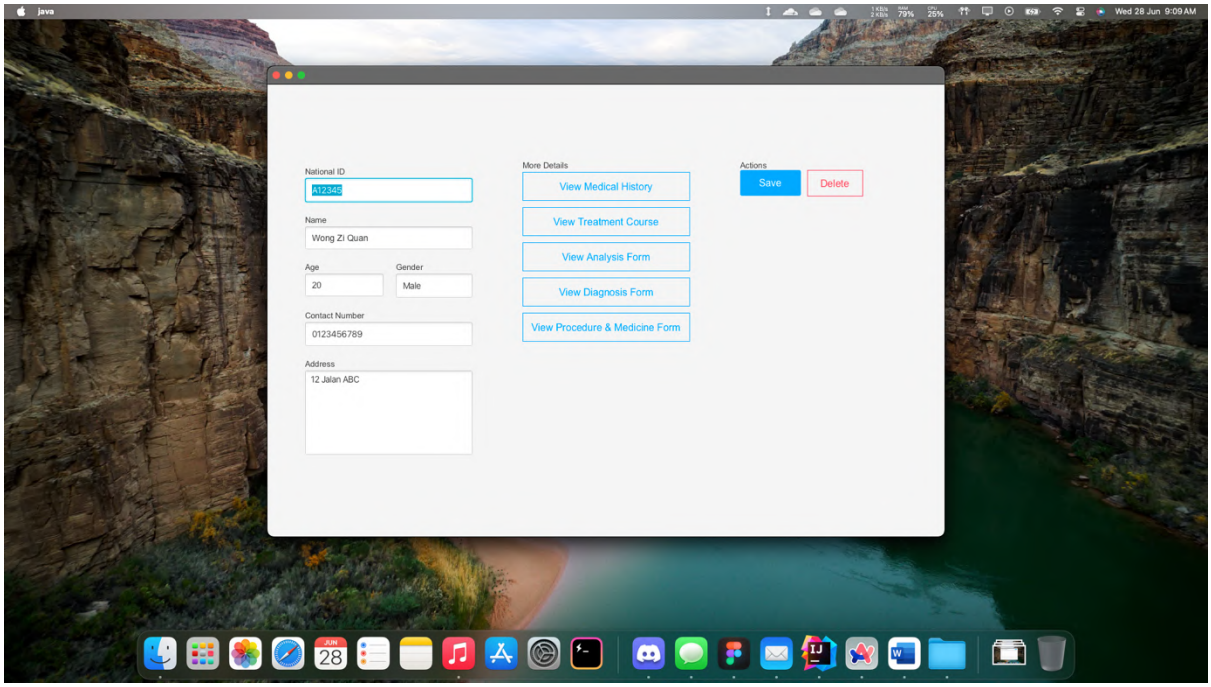
Clicking the View Diagnosis Form button will bring you to the Diagnosis Form to show all of the patients' previous diagnosis history with details such as date, name and diagnosed sickness. Pressing the Delete button will delete the record from the system. A popup dialog will also be shown to indicate that the record has been deleted.



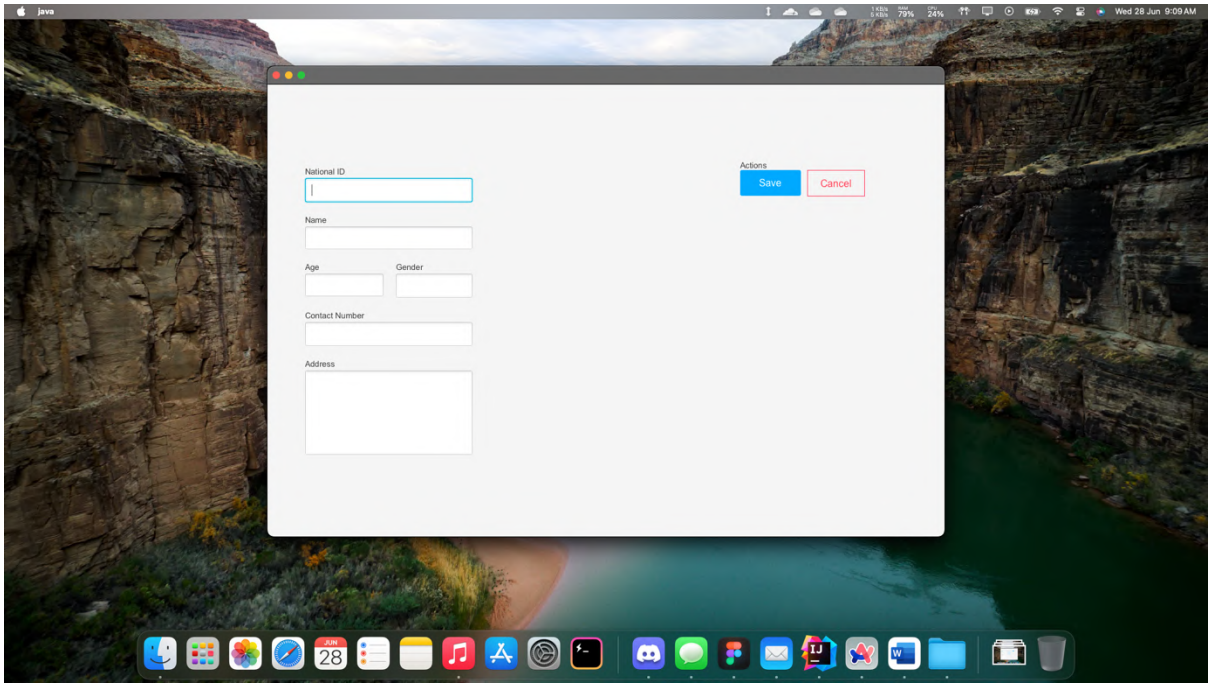
Clicking the View Procedure & Medicine Form button will bring you to the Procedure & Medicine Form to show all of the patients' previous procedure & medicine history with details such as date, time, procedure type, medication, amount and frequency. Pressing the Delete button will delete the record from the system. A popup dialog will also be shown to indicate that the record has been deleted.



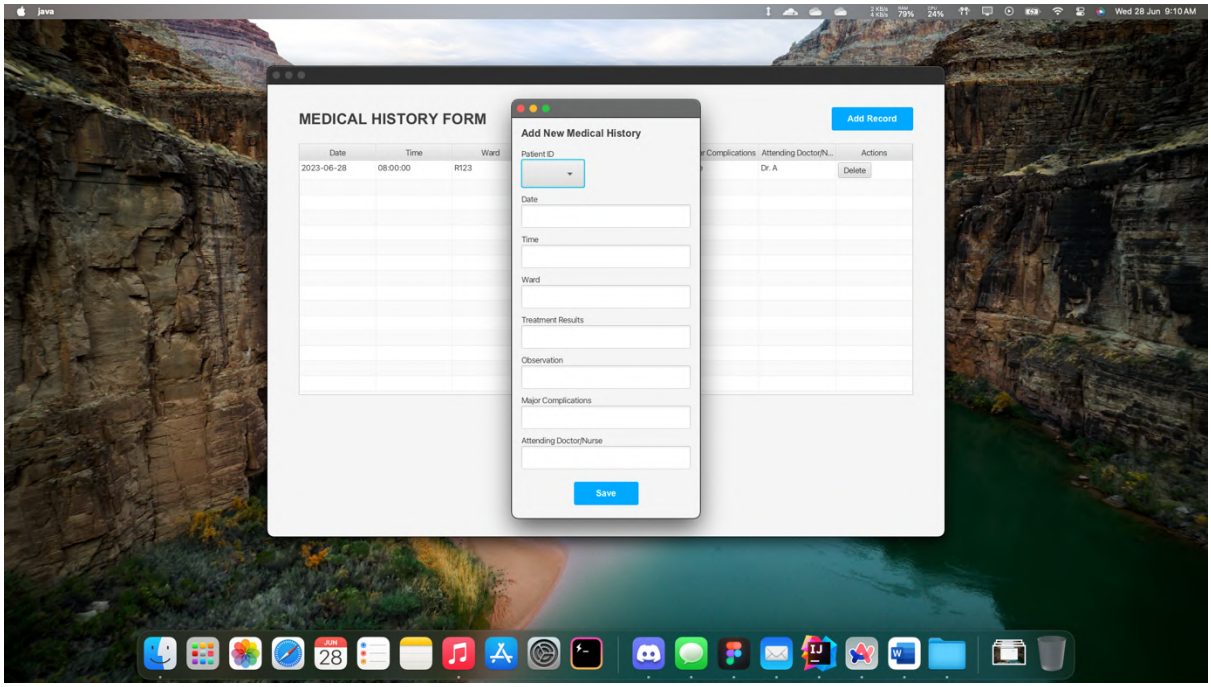
Clicking the View button of any patient's record will give you the ability to modify their personal information stored in the system. Change the value in the given text fields then press the Save button to save the changes made. A popup dialog will then be shown to indicate that the changes have been saved successfully. Similar to what the patients see, clicking the buttons (View Medical History, View Treatment Course, View Analysis Form, View Diagnosis Form, View Procedure & Medicine Form) will bring you to respective screens to access more of that particular patient's data. Pressing the Delete button will delete the patient's record from the system. A popup dialog will also be shown to indicate that the record has been deleted.



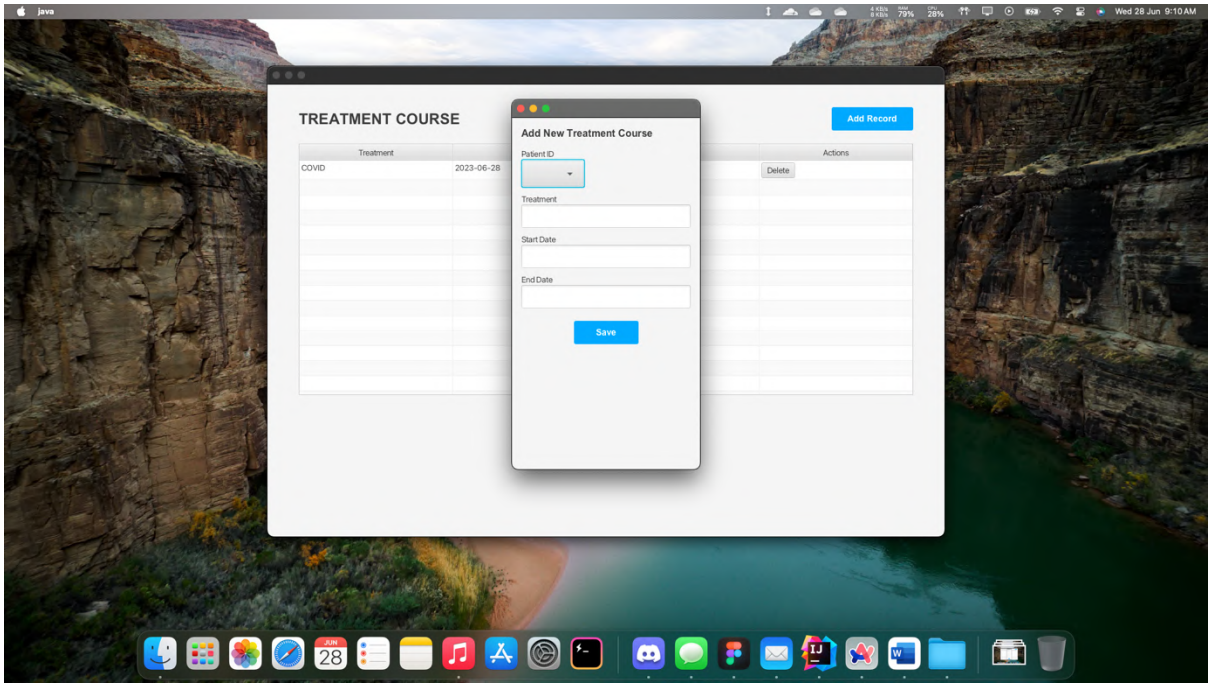
Clicking the Add Record in the Patient Form will bring you to this screen below to create new patient record. Enter the values in the given text fields then press the Save button to save the record. A popup dialog will then be shown to indicate that the changes have been saved successfully.



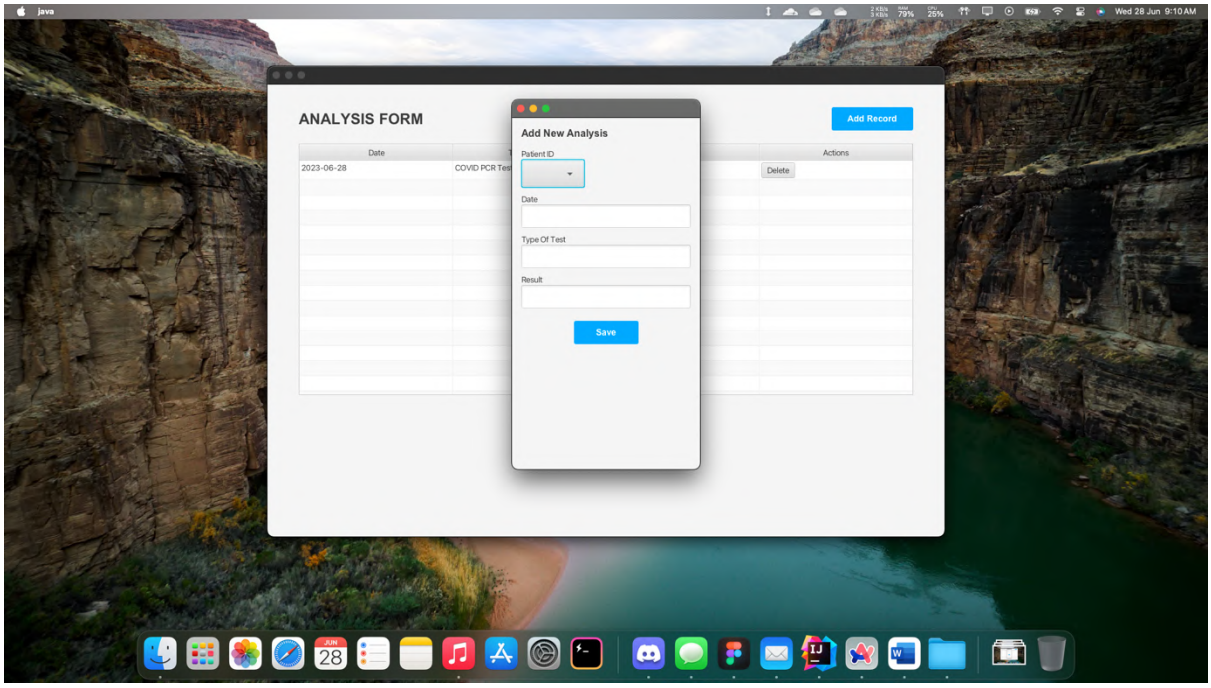
Clicking the Add Record in the Medical History Form will bring you to this screen below to create new medical history record. Select and enter the values in the given text fields then press the Save button to save the record. A popup dialog will then be shown to indicate that the changes have been saved successfully.



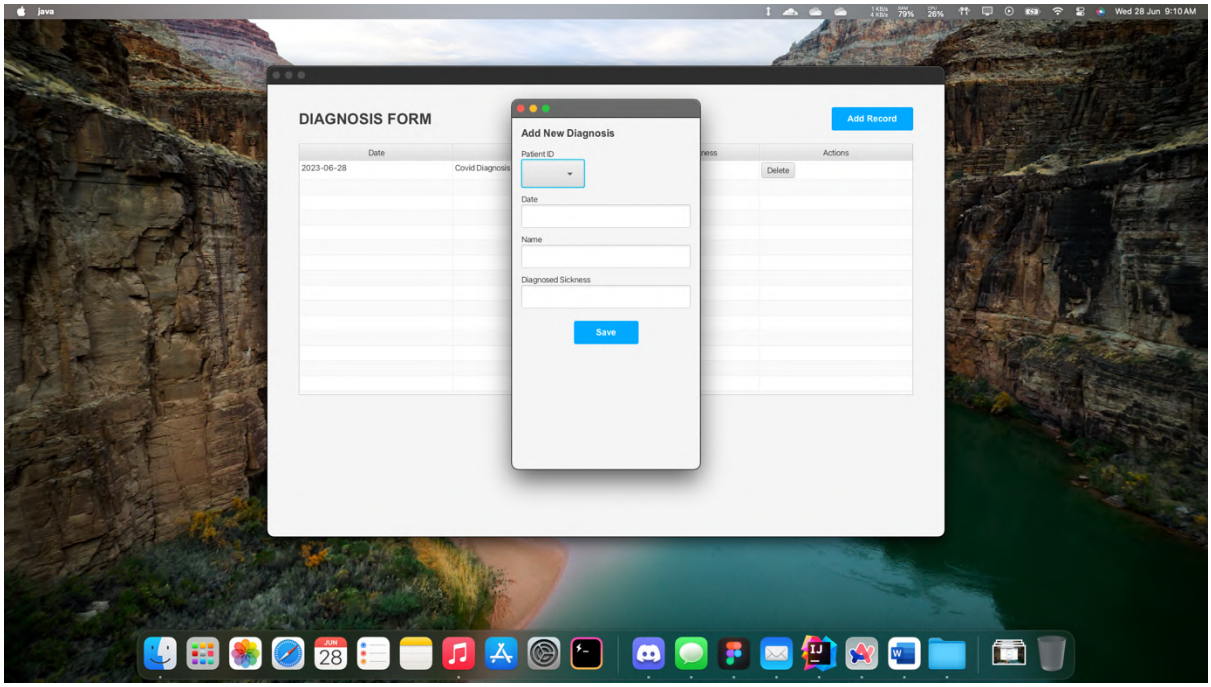
Clicking the Add Record in the Treatment Course screen will bring you to this screen below to create new treatment course record. Select and enter the values in the given text fields then press the Save button to save the record. A popup dialog will then be shown to indicate that the changes have been saved successfully.



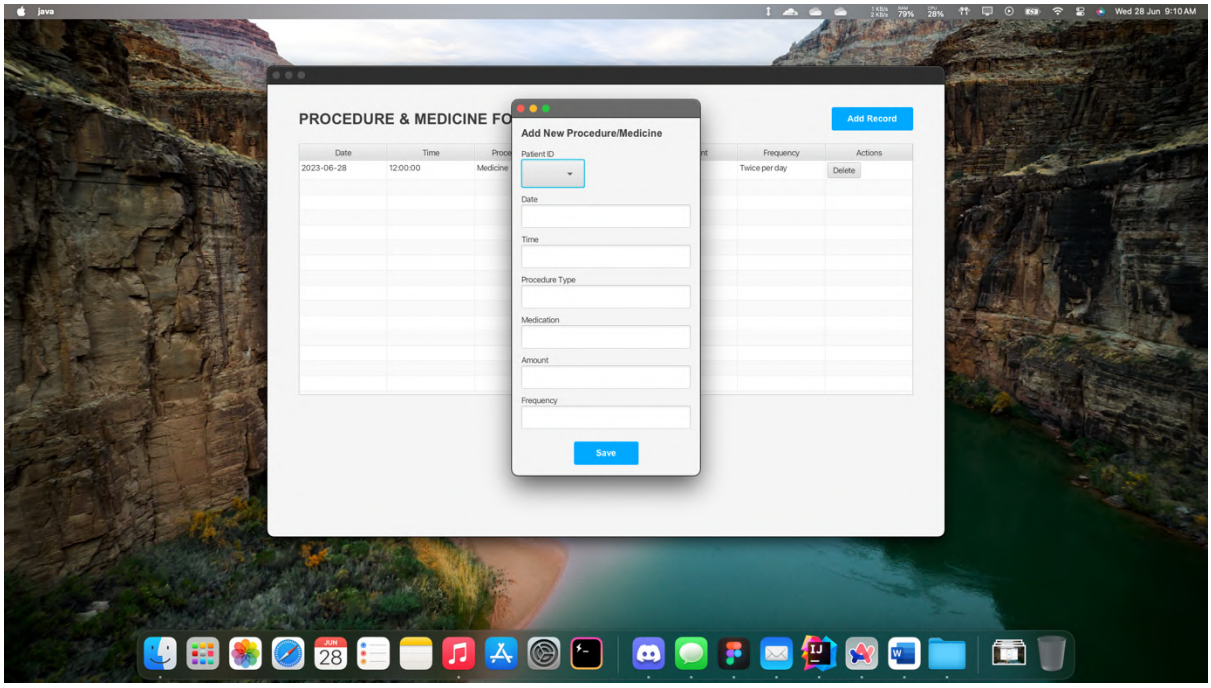
Clicking the Add Record in the Analysis Form will bring you to this screen below to create new analysis record. Select and enter the values in the given text fields then press the Save button to save the record. A popup dialog will then be shown to indicate that the changes have been saved successfully.



Clicking the Add Record in the Diagnosis Form will bring you to this screen below to create new diagnosis record. Select and enter the values in the given text fields then press the Save button to save the record. A popup dialog will then be shown to indicate that the changes have been saved successfully.



Clicking the Add Record in the Procedure & Medicine Form will bring you to this screen below to create new procedure & medicine record. Select and enter the values in the given text fields then press the Save button to save the record. A popup dialog will then be shown to indicate that the changes have been saved successfully.



Applied OOP Concepts

By leveraging OOP principles, our system achieves better organization, modularity, and code reusability. This part of the documentation will discuss the concepts are applied in our system design.

In the provided code snippet below, the class **Patient** represents a patient entity in a healthcare system.

The **Patient** class demonstrates encapsulation by encapsulating the patient's data within private fields (**id**, **name**, **national_id**, **age**, **gender**, **address**, **contact_no**). The fields are accessed through public getter methods (**getName()**, **getID()**, **getNationalID()**, **getAge()**, **getGender()**, **getAddress()**, **getContactNo()**), which provide controlled access to the data. By encapsulating the fields and providing getters, the class maintains data integrity and controls the access to the patient's information.

The private fields in the **Patient** class (**id**, **name**, **national_id**, **age**, **gender**, **address**, **contact_no**) are not directly accessible from outside the class. Instead, the class exposes public getter methods to access the data. This information hiding promotes encapsulation and ensures that the internal representation of the patient's data is not exposed to external code.

The **Patient** class has a constructor that takes in various parameters (**id**, **name**, **national_id**, **age**, **gender**, **address**, **contact_no**) to initialize the object. This constructor demonstrates the concept of object instantiation, allowing the creation of **Patient** objects with specified attributes during runtime.

The **Patient** class abstracts the patient's information by using the **SimpleStringProperty** class from JavaFX for the fields (**name**, **national_id**, **age**, **gender**, **address**, **contact_no**). By using this abstraction, the actual data representation is encapsulated within the **SimpleStringProperty** objects, providing additional functionality and allowing for potential data binding and event handling in UI frameworks.

```java
public Patient(String id, String name, String national_id, String age, String gender, String address, String contact_no) {
    this.id = id;
    this.name = new SimpleStringProperty(name);
    this.national_id = new SimpleStringProperty(national_id);
    this.age = new SimpleStringProperty(age);
    this.gender = new SimpleStringProperty(gender);
    this.address = new SimpleStringProperty(address);
    this.contact_no = new SimpleStringProperty(contact_no);
}

public String getName() {
    return name.get();
}

public String getID() {
    return id;
}

public String getNationalID() {
    return national_id.get();
}

public String getAge() {
    return age.get();
}

public String getGender() {
    return gender.get();
}

public String getAddress() {
    return address.get();
}

public String getContactNo() {
    return contact_no.get();
}
```

```java
public List<Patient> readCSV(String fileName) {
    String delimiter = ",";
    BufferedReader bReader = null;
    File file = new File(fileName);
    List<Patient> data = new ArrayList<>();
    try {
        String line = "";
        bReader = new BufferedReader(new FileReader(file));
        bReader.readLine();
        while ((line = bReader.readLine()) != null) {
            String[] tokens = line.split(delimiter);
            if (tokens.length > 0) {
                Patient patient = new Patient (String.valueOf(Integer.parseInt(tokens[0])), tokens[1], tokens[2], tokens[3], tokens[4],
                data.add(patient);
            }
        }
    } catch (FileNotFoundException e) {
        try {
            FileWriter fileWriter = new FileWriter(file);
            fileWriter.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            if (bReader != null)
                bReader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    return data;
}
```

For another example, the **Diagnosis** class represents a diagnosis entity in a healthcare system and applies several key OOP concepts.

The class encapsulates the diagnosis data within private fields (**id**, **date**, **name**, **diagnosed_sickness**). Access to these fields is provided through public getter methods (**getId()**, **getDate()**, **getName()**, **getDiagnosedSickness()**), ensuring controlled access to the data and maintaining data integrity.

By making the fields private, direct access to the internal representation of the diagnosis data is prevented. External code interacts with the class solely through the public getter methods, promoting information hiding and encapsulation.

The class features a constructor that allows for the creation of **Diagnosis** objects with specific attributes (**id**, **date**, **name**, **diagnosed_sickness**). This constructor demonstrates object instantiation, enabling the initialization of diagnosis instances during runtime.

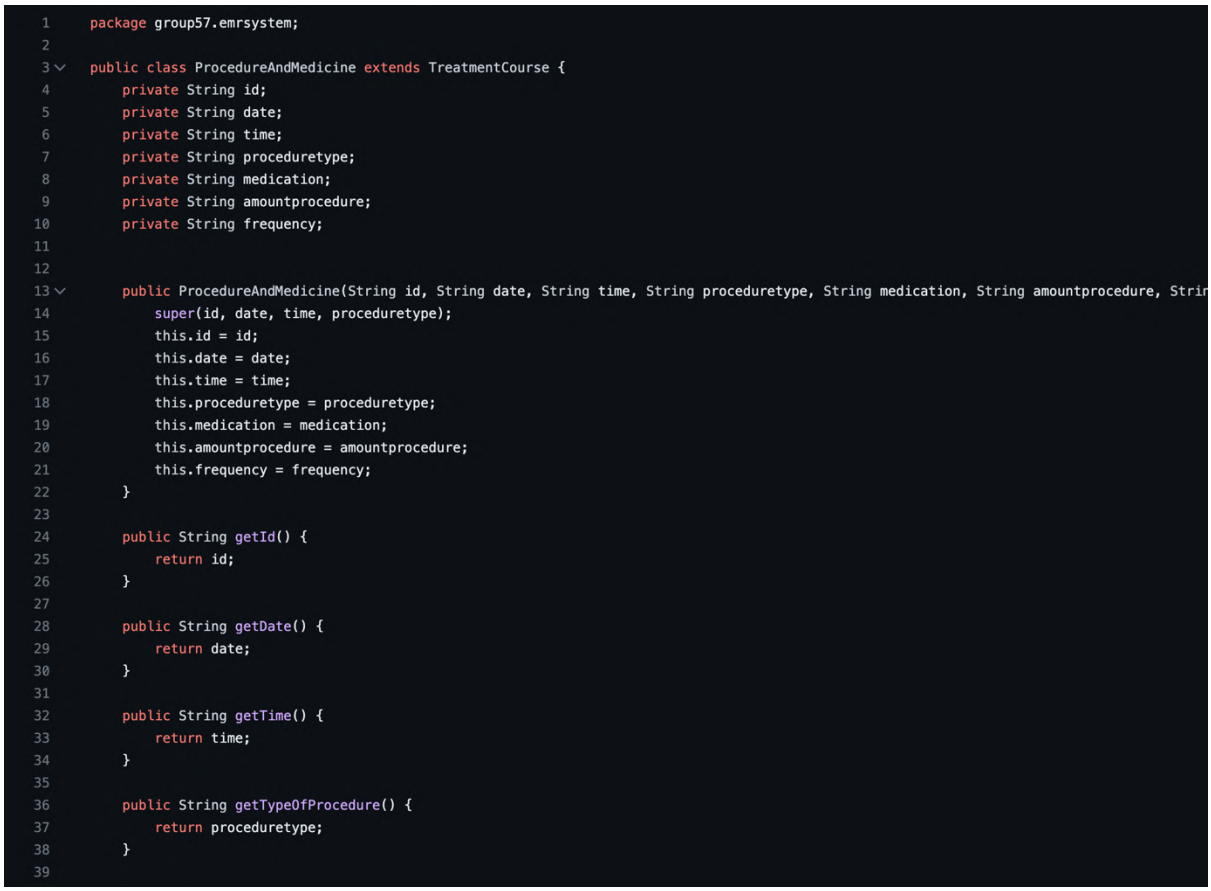
The class leverages the **SimpleStringProperty** class from JavaFX to abstract the actual data representation of the fields (**id**, **date**, **name**, **diagnosed_sickness**). This abstraction provides additional functionality and facilitates potential data binding and event handling in UI frameworks.

```java
package group57.emrsystem;

import javafx.beans.property.SimpleStringProperty;

public class Diagnosis {
    private SimpleStringProperty id;
    private SimpleStringProperty date;
    private SimpleStringProperty name;
    private SimpleStringProperty diagnosed_sickness;

    public Diagnosis(String id, String date, String name, String diagnosed_sickness) {
        this.id = new SimpleStringProperty(id);
        this.date = new SimpleStringProperty(date);
        this.name = new SimpleStringProperty(name);
        this.diagnosed_sickness = new SimpleStringProperty(diagnosed_sickness);
    }

    public String getId() {
        return id.get();
    }

    public String getDate() {
        return date.get();
    }

    public String getName() {
        return name.get();
    }


    public String getDiagnosedSickness() {
        return diagnosed_sickness.get();
    }
}
```

```java
public List<Diagnosis> AdminReadCSV(String fileName) {
    String delimiter = ",";
    BufferedReader bReader = null;
    File file = new File(fileName);
    List<Diagnosis> data = new ArrayList<>();
    try {
        String line = "";
        bReader = new BufferedReader(new FileReader(file));
        bReader.readLine();
        if (username.equals("admin")) {
            while ((line = bReader.readLine()) != null) {
                String[] tokens = line.split(delimiter);
                if (tokens.length > 0) {
                    Diagnosis diagnosis = new Diagnosis(tokens[1], tokens[2], tokens[3], tokens[4]);
                    data.add(diagnosis);
                }
            }
        } else {
            while ((line = bReader.readLine()) != null) {
                String[] tokens = line.split(delimiter);
                if (tokens.length > 0) {
                    if (tokens[1].equals(username)) {
                        Diagnosis diagnosis = new Diagnosis(tokens[1], tokens[2], tokens[3], tokens[4]);
                        data.add(diagnosis);
                    }
                }
            }
        }
    } catch (FileNotFoundException e) {
        try {
            FileWriter fileWriter = new FileWriter(file);
            fileWriter.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
```

The **extends** keyword signifies that the **Analysis** class, **Diagnosis** class, and **ProcedureAndMedicine** class inherits from the **TreatmentCourse** class. Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class to inherit properties and behaviors from another class, called the superclass or parent class (**TreatmentCourse** in this case).

By extending the **TreatmentCourse** class, the classes gains access to all the public and protected members (fields and methods) of the **TreatmentCourse** class. It effectively inherits the attributes and behaviors defined in the superclass, allowing code reuse and specialization.



```java
package group57.emrsystem;

public class ProcedureAndMedicine extends TreatmentCourse {
    private String id;
    private String date;
    private String time;
    private String proceduretype;
    private String medication;
    private String amountprocedure;
    private String frequency;


    public ProcedureAndMedicine(String id, String date, String time, String proceduretype, String medication, String amountprocedure, Strin
        super(id, date, time, proceduretype);
        this.id = id;
        this.date = date;
        this.time = time;
        this.proceduretype = proceduretype;
        this.medication = medication;
        this.amountprocedure = amountprocedure;
        this.frequency = frequency;
    }

    public String getId() {
        return id;
    }

    public String getDate() {
        return date;
    }

    public String getTime() {
        return time;
    }

    public String getTypeOfProcedure() {
        return proceduretype;
    }

```

```java
package group57.emrsystem;

import javafx.beans.property.SimpleStringProperty;

public class Diagnosis extends TreatmentCourse {
    private SimpleStringProperty id;
    private SimpleStringProperty date;
    private SimpleStringProperty name;
    private SimpleStringProperty diagnosed_sickness;

    public Diagnosis(String id, String date, String name, String diagnosed_sickness) {
        super(id, date, name, diagnosed_sickness);
        this.id = new SimpleStringProperty(id);
        this.date = new SimpleStringProperty(date);
        this.name = new SimpleStringProperty(name);
        this.diagnosed_sickness = new SimpleStringProperty(diagnosed_sickness);
    }

    public String getId() {
        return id.get();
    }

    public String getDate() {
        return date.get();
    }

    public String getName() {
        return name.get();
    }


    public String getDiagnosedSickness() {
        return diagnosed_sickness.get();
    }
}
```

```java
package group57.emrsystem;

public class Analysis extends TreatmentCourse {
    private String id;
    private String date;
    private String type_of_test;
    private String result;

    public Analysis(String id,String date, String type_of_test, String result){
        super(id, date, type_of_test, result);
        this.id = id;
        this.date = date;
        this.type_of_test = type_of_test;
        this.result = result;
    }

    public String getId(){
        return id;
    }

    public String getDate(){
        return date;
    }

    public String getTypeOfTest(){
        return type_of_test;
    }

    public String getResult(){
        return result;
    }
}
```

The class **NewPatientController** is a JavaFX controller responsible for handling interactions with a user interface for adding new patients. Let's focus on the **initialize()** method, which is overridden from the **Initializable** interface, and explain its purpose:

The **initialize()** method is invoked automatically when the controller is initialized by the JavaFX framework. It takes two parameters: the **URL** representing the location of the FXML file and a **ResourceBundle** containing localized resources.

Within the **initialize()** method, two event handlers are registered:

1. **saveButton.setOnAction(actionEvent -> ToBeSaved())**: This line assigns an event handler to the **saveButton** button. When the button is clicked, the **ToBeSaved()** method is called. This method is responsible for gathering patient information from the UI fields, processing the data, and saving it.
2. **cancelButton.setOnAction(actionEvent -> stage.close())**: This line assigns an event handler to the **cancelButton** button. When the button is clicked, the **stage.close()** method is called, which closes the current stage/window.

The **initialize()** method serves as a central place to set up event handlers and perform any necessary initialization logic for the controller. In this case, it sets up the button click event handlers for saving and canceling new patient entries.

```
18    public class NewPatientController implements Initializable {
93        private void ToBeSaved(){
100           readCSV();
101           String id = String.valueOf((int) (Math.random() * 10));
102           while (usedIds.contains(id)) {
103               id = String.valueOf((int) (Math.random() * 10));
104           }
105           List<String> stringArrays = new ArrayList<>();
106           stringArrays.add("id,name,national_id,age,gender,address,contact_no\n");
107           stringArrays.addAll(data);
108           String newPatientString = id + "," + name + "," + nationalID + "," + age + "," + gender + "," + address + "," + contactNumber + "\n
109           stringArrays.add(newPatientString);
110           try (BufferedWriter writer = new BufferedWriter(new FileWriter(Objects.requireNonNull(NewDiagnosisController.class.getResource("pat
111               for (String stringArray : stringArrays) {
112                   writer.write(stringArray);
113               }
114               System.out.println("Data has been written to the file.");
115               Alert alert = new Alert(Alert.AlertType.INFORMATION);
116               alert.setTitle("Information Dialog");
117               alert.setHeaderText(null);
118               alert.setContentText("New record has been saved!");
119               alert.showAndWait();
120               stage.close();
121           } catch (IOException e) {
122               System.err.println("An error occurred while writing to the file: " + e.getMessage());
123           }
124           parentController.renderData();
125       }
126
127       @Override
128       public void initialize(URL location, ResourceBundle resources) {
129           saveButton.setOnAction(actionEvent -> ToBeSaved());
130           cancelButton.setOnAction(actionEvent -> stage.close());
131       }
132   }
```

```java
package group57.emrsystem;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

public class TreatmentCourseScreen extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(TreatmentCourseController.class.getResource("treatmentcourse-admin.fxml"));
        TreatmentCourseController controller = new TreatmentCourseController(stage, true, "admin");
        fxmlLoader.setController(controller);
        Scene scene = new Scene(fxmlLoader.load(), 1080, 720);
        stage.setTitle("Treatment Course");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch();
    }
}
```

The class **LoginScreen** extends the **Application** class from JavaFX. The **LoginScreen** class extends the **Application** class, which is a fundamental class provided by the JavaFX framework. By extending **Application**, the **LoginScreen** class becomes a JavaFX application itself, capable of launching and managing the graphical user interface (GUI).

```java
package group57.emrsystem;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

public class LoginScreen extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(LoginController.class.getResource("login.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 1080, 720);
        stage.setTitle("Login Screen");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch();
    }
}
```

```java
public class LoginController implements Initializable {
    @FXML
    public TextField usernameField;
    @FXML
    public TextField passwordField;
    @FXML
    public Button loginButton;

    public LoginController(Stage stage)
    {
        this.stage = stage;
    }

    private void login() throws IOException {
        String username = usernameField.getText();
        String password = passwordField.getText();
        if (username.equals("admin") && password.equals("admin")) {
            stage.close();
            PatientScreen patientScreen = new PatientScreen();
            patientScreen.hackedStart(new Stage(), true, username);
        } else {
            stage.close();
            PatientScreen patientScreen = new PatientScreen();
            patientScreen.hackedStart(new Stage(), false, username);
        }
    }

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        loginButton.setOnAction(actionEvent -> {
            try {
                login();
            } catch (IOException e) {
                e.printStackTrace();
            }
        });
    }
}
```

Work Responsibilities & Delegation:

No.	Student Name	Task
1	Cheah Zixu	UI Static Prototype, Brief Documentation: Data Processing & User Interface and Navigation, JavaFX User Interface, CRUD operations including Serialization, Documentation
2	Edward Prilvanto Djong	Use Case Diagram, Brief Documentation: Future Developments & Upgrades, Treatment Course, Procedure class, Medicine class
3	Billy Taslim	Use Case Diagram, Brief Documentation: Key Features, Treatment Course, Diagnosis class, Medical History class
4	Aldrich Eugene Terimsia	Class Diagram, Brief Documentation: EMR Benefits, Treatment Course, Login class, Patient class
5	Wong Zi Quan	Class Diagram, Brief Documentation: Introduction, Treatment Course, Analysis class, Treatment Course class