Group 4: E-Healthcare Management System

Cheng-Han Hsieh, 謝承翰
B103040012

Shih Yu Sun, 孫世諭
B103040001

Casper Liu, 劉世文
B093040051

Tina Tsou, 鄒宜庭
B096060032

Chia-Yen Huang, 黃嘉彥
B103040051

Ting-Hao Hsu, 許廷豪
B103040008

December 25, 2023

1 Outline

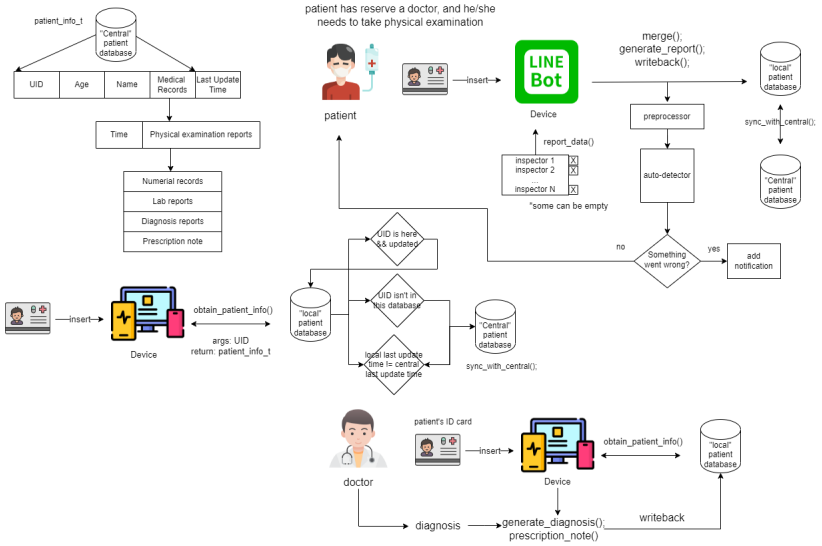

Figure 1: A simple example of outline.

Figure 1 shows the skelton of the whole system. The left-top part is the scrawled database, which defines "what" is in the database. As shown, the database contains necessary medical information of patients. The right-top part is the situation that a patient has maken an appointment with doctor and he need to take a physical examination first. By insert the ID card, the physical data will be recorded into the local database, syncing with the central database, and in the mean time, the result of examination will be sended to "auto-detectors", which detect the abnormal data in the physical reports, and notify the patient. The auto-detectors can detect potential diseases like cancer by determining the gene expression profiling, or diabetes by the physical data. The middle part is the situation that a patient or a doctor wants to obtain the information of the patient. After inserting the ID card, the local database will check whether the status of corresponding data. If it is out of date or not exists, the local database will try to sync with the central database, otherwise create a new one if this patient is totally new. Eventually, the information of this patient is returned to the client (device). The bottom part shows a doctor make the diagnosis and prescription for the patient. The new diagnosis and prescription are write-backed to the local database, and also sync with the central one.

2 Features

With this E-healthcare management system, the hospital/clinic can easily sync the information of patients with other health systems, manage the information of patients. For patients, the patient can do most of the things online, for example, make a doctor appointment, look up the medical records and prescriptions, and obtain the physical reports at home. Even more, the system use machine learning to detect the abnormal data in the reports, notify the patient to prevent the disease becoming worse. To conclusion, the major features of the system can be summarized as followings:

- Automatically sync the information of patients between different health systems by using local and central database.
- Facilitate the accessing of medical records and prescriptions, for both doctors and patients.
- Introduce the automatic disease detectors by leveraging machine learning and big data.

3 Methodology

Database

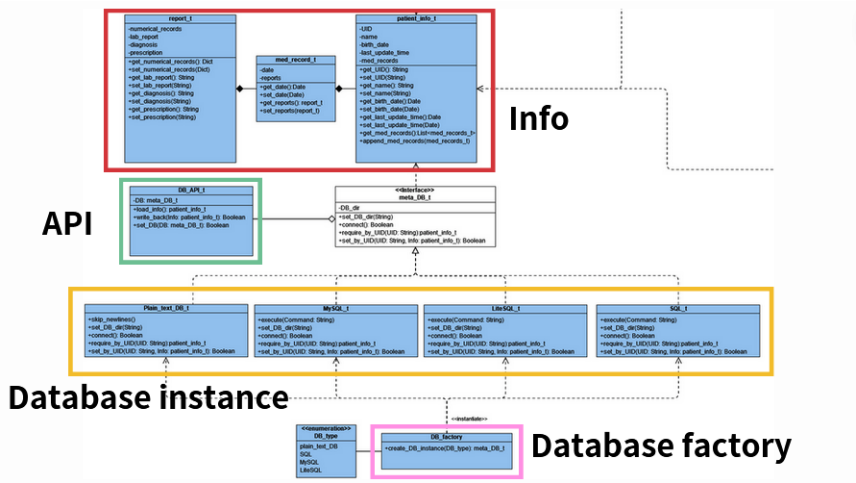

Figure 2: The UML of the whole database.

Figure 2 shows the UML of the database. It can be divided into four parts, the class definition of patients' information, the database API, the class definition of database, and the database factory.

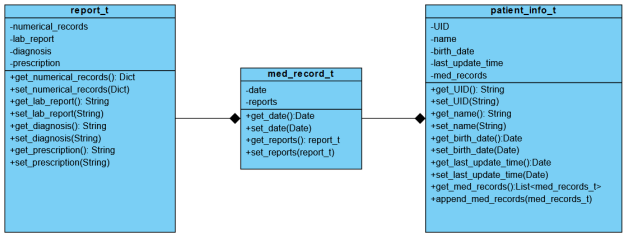

Figure 3: The UML of the class definition of patient information.

Figure 3 shows the class definition of the patients' information. The class, `patient_info_t`, contains the necessary information of a patient, for example, UID, name, birth date, and the list of medical records. And

in the class, `med_record_t`, is date and reports, like diagnosis, prescriptions, and physical data and reports. In this part, it provides a definition of patients' information for the whole database. Later, the database will depend on the class defined in this part.

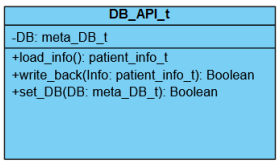

Figure 4: The UML of the class definition of database API.

Figure 4 is the definition of the API of the database. It provide some simple and secure methods to access the database, for example, load the information of a patient, update the information, and set the database. Later, those parts of the system that need to access the database rely on this API.

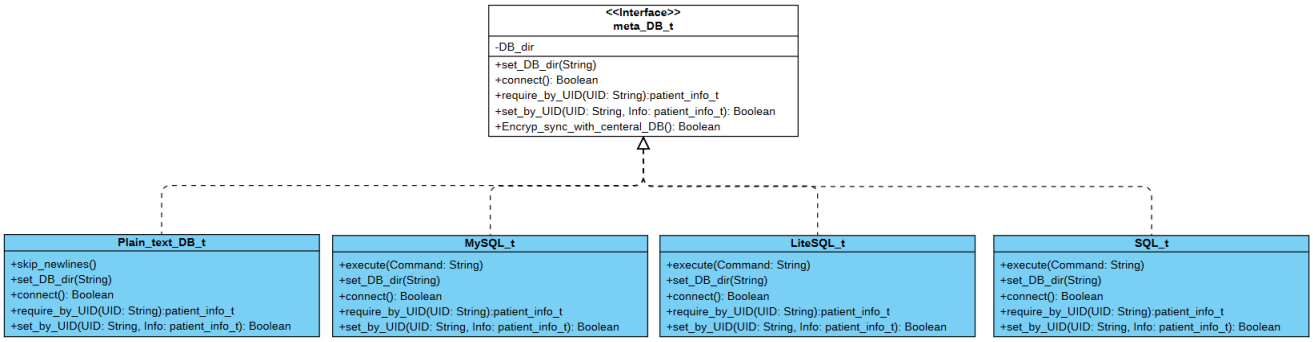

Figure 5: The UML of the class definition of database instance.

Here is the definition of the databases. First is an interface of the database, it defines some common methods of database, like connect, require, update, and sync with other database. Then are the specialized databases. Here, plain text database, SQLs are defined.

The rationales behind the need of an interface is, in the early development, we did not determine which type of database should be use. And second, when scaling the system up, the database may need to be replaced with other databases that have higher throughput and lower response time, like ScyllaDB. An interface of the database solves the problems, because it abstracts the database and unify the methods. The additional databases can be easily provided by following the interface.

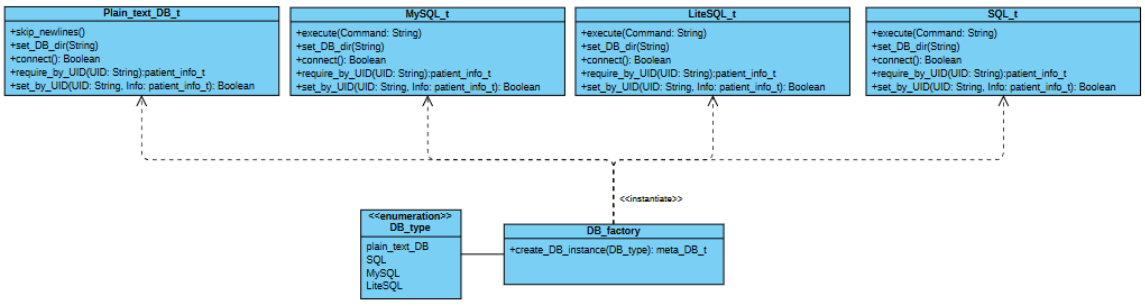

Figure 6: The UML of the factory of databases.

Then is the factory of database, which shown in Figure 6. In short, factory is responsible for instantiating the database. And it is actually a pattern from *Design Patterns*. The general UML of factory pattern is

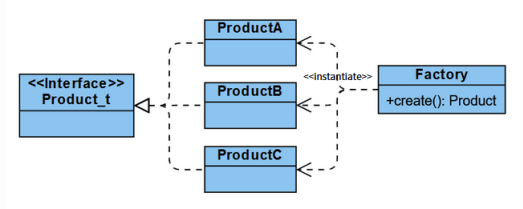

Figure 7: The UML of a simple factory.

shown in Figure 7. The products inherit a common interface, and they are all instantiated by the factory. The advantage of factory patterns is it hides the details of "creation", which allows the factory changing the implementation without modifying the usage of the creation. It is a very import characteristics in developing a big system. The modification of the usage of methods can cost lots of time and effort, because all the programs that use the methods should be modified.

To conclusion, this design of database has the following features:

- Hide the detail of the creation.
- An uniform application interface.
- High Scalability.
- Easy to maintain.

Frontend and Medium

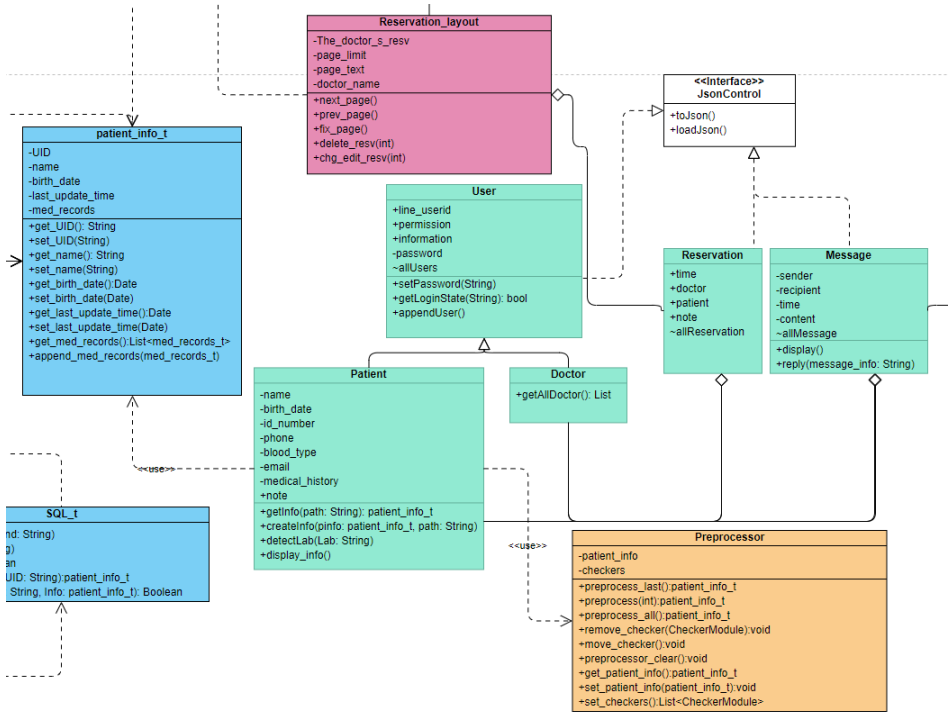

Figure 8: The UML of frontend and medium.

In the code for the patient frontend and middleware, our functionalities are as follows (refer to Figure 8):

1. Utilize Ngrok to forward external requests to the locally specified port.

2. Integrate LineBot, HTML, JS, and CSS, serving as the patient frontend to send requests such as registration, message sending, and reservations.
3. Connect to the database to retrieve detailed information based on the user's ID.
4. Interface with the Processor (Lab), sending detailed information retrieved from the database to the C++ Processor (Lab) for processing and receiving the information back.
5. Enable the Doctor GUI to utilize the Patient Class to
   (i) obtain detailed information from the database,
   (ii) access return information from the Processor (Lab),
   (iii) retrieve the Message Class, and
   (iv) use the `Reply()` method to quickly respond to messages via LineBot (refer to Figure 8),
   (v) obtain detailed data from the Reservation Class,
   (vi) access the Doctor Class, and
   (vii) verify its name and password for login.

Next, we will explain each class in the frontend and medium, highlighting some special member functions and attributes:

User

- An abstract class primarily responsible for handling account information such as:
  (i) `permission:` Manages permissions, used to confirm the current user mode (admin or guest).
  (ii) `line_id:` Mainly used to record the user's Line ID, this information is automatically obtained from Line during registration and transmitted to our server.
  (iii) `getLoginState(String):` Takes a password as input and checks if it is the correct password.

Patient

- A child class of User, mainly deals with patient account information, where it interfaces with the `patient_info_t` in the database to obtain detailed data such as heart rate, blood glucose:
  (i) `detectLab(String):` Will send the information of `patient_info_t` to the Preprocessor, obtaining a detailed diagnosis such as "high heart rate," "diabetes risk," etc (refer to Figure 9).

JsonControl

- An interface implemented by the User, Reservation, and Message classes. Its main purpose is to serialize objects into JSON files for easy storage and access:
  (i) `toJson():` Stores all objects of the entire class in JSON format.
  (ii) `loadJson():` Reads a JSON file and restores all objects from it into a list.

Reservation

- When a patient uses the frontend LineBot to make a reservation, a Reservation object is created. It is used and deleted by the Doctor GUI and includes Patient and Doctor objects to determine the patient and the reserved doctor (refer to Figure 10).

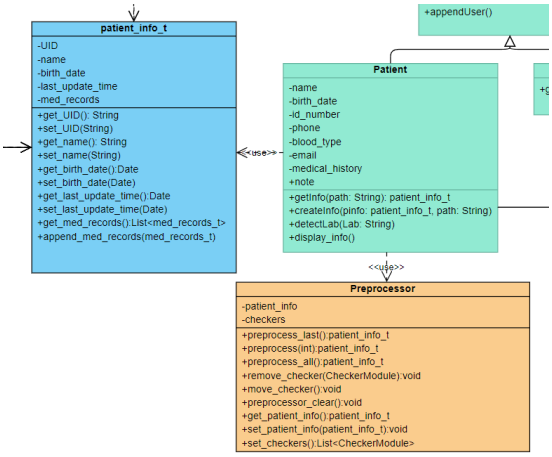

Figure 9: The UML of interaction between medium, preprocessor and database.

Message

- When a patient uses the frontend LineBot to send a message, a Message object is created. It is used, replied to, and deleted by the Doctor GUI and includes Patient and Doctor objects to determine the patient and the doctor being messaged (refer to Figure 10):

  (i) `reply():` Based on its own Patient object, it uses LineBot to send a message back to the patient (because the Patient object contains the line_id attribute).

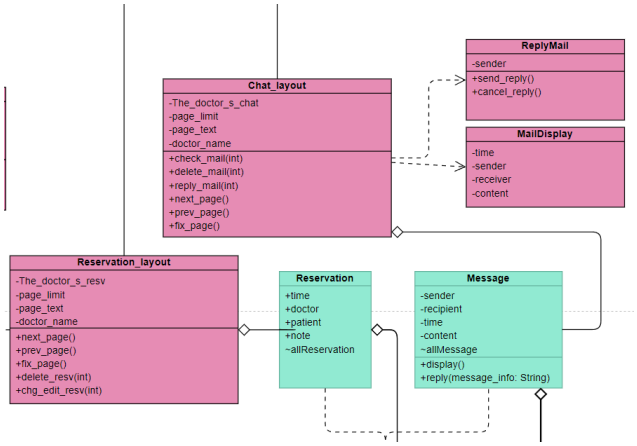

Figure 10: The UML of interaction between medium and Doctor GUI.

Preprocessor

- The preprocessor is created with its checkers made with CheckerModule. Before any kind of preprocessor is created, one or more checkers should be created first by the programmer. The programmer should create checker based on the demand from the dotcor and under the rule of ChckerModule. The preprocessor is useful to preprocess the data, like fixing the wrong data based on the right data and existing formula, or deleting the NAN data and so on. Input the data into the preprocessor's preprocessing-specific function, data then be preprocessed. Modify or add the checker(s) into the preprocessor with checker-modifying function. When the data is preprocessed, the programmer can assume those data ready for detecting in the next part: Detector.(refer to Figure 11):

(i) `preprocess_last()`: Preprocess the data in the last and newest profile stored in patient_info.

(ii) `preprocess(x)`: Preprocess the data the no.`x` profile stored in patient_info.

(iii) `preprocess_all()`: Preprocess the data of all profiles stored in patient_info.

(iv) `move_checker(x)`: Move the new checker "`x`" into the preprocessor.

(v) `remove_checker()`: Remove the last moved checker from the preprocessor.

(vi) `clear_checker()`: Clear all the checkers from the preprocessor.

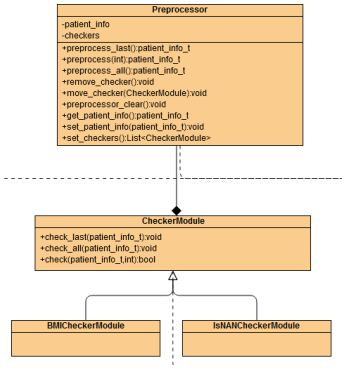

Figure 11: The UML of preprocessor and its composition: CheckerModule.

Detector

- The metadetector is created with its detectors made with DetectorModule. Before any kind of meta-detector is created, one or more detectors should be created first by the programmer. The programmer should create detector based on the demand from the dotcor and under the rule of DetectorModule. The meta-detector is used to detect the illness or abnormal value from the patient. The programmer can get the result from Detector_Status in a general way, or list of strings that contains detailed information to be used in more specific situation. (refer to Figure 12):

  (i) `detect(Description,Status)`: Detect the illness or abnormal value in the last and newest profile stored in patient_info. `Description` contains list of string that stores more detailed information, and `Status` give an easy for programmer to verify the result. The detector do not support detecting for non-last profile, because it's without apparent cause.

  (ii) `move_detector(x)`: Move the new detector "`x`" into the meta-detector.

  (iii) `remove_detector()`: Remove the last moved detector from the meta-detector.

  (iv) `clear_detector()`: Clear all the detector from the meta-detector.

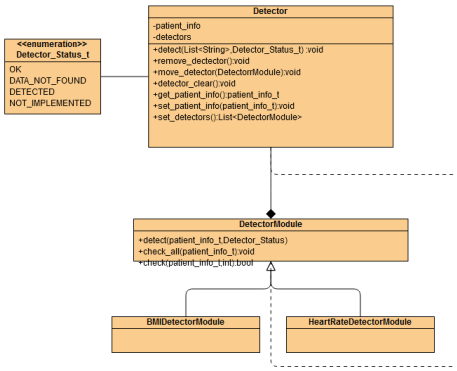

Figure 12: The UML of detector and its composition: DetectorModule.
The Detector_status_t is contained in the picture as well, which is used to present the status of the result.

Frontend(DoctorGUI)

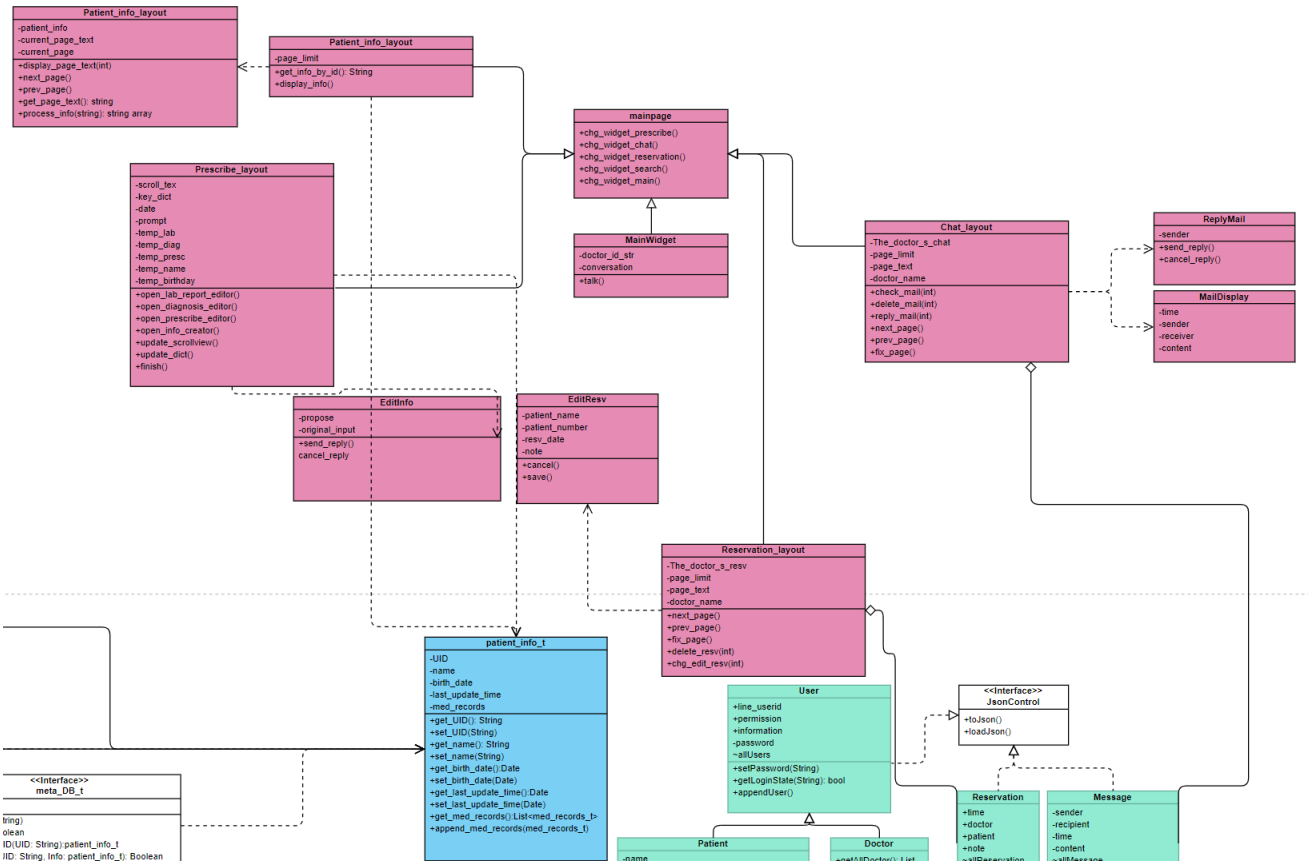

Figure 13: The UML of frontend.

Abstract

The frontend, also can be called Doctor GUI, provide doctors with complete medical consultation functions such as :

- Viewing patient information
- Editing prescriptions
- Replying to patient messages
- Viewing and modifying patient reservation information

By Using this software, doctors can overcome space and time constraints. In addition to communicating with patients and making appointments in a timely manner, they can also synchronize all doctors' information in a timely manner, so that patients do not have to re-establish files every time they go to a hospital.

The GUI mainly communicates with the database and medium (linebot), displays the data on the interface, and provides editing and deletion functions.

Next, I will explain in detail the method of each class and how they communicate with other classes.

MainWidget and Design

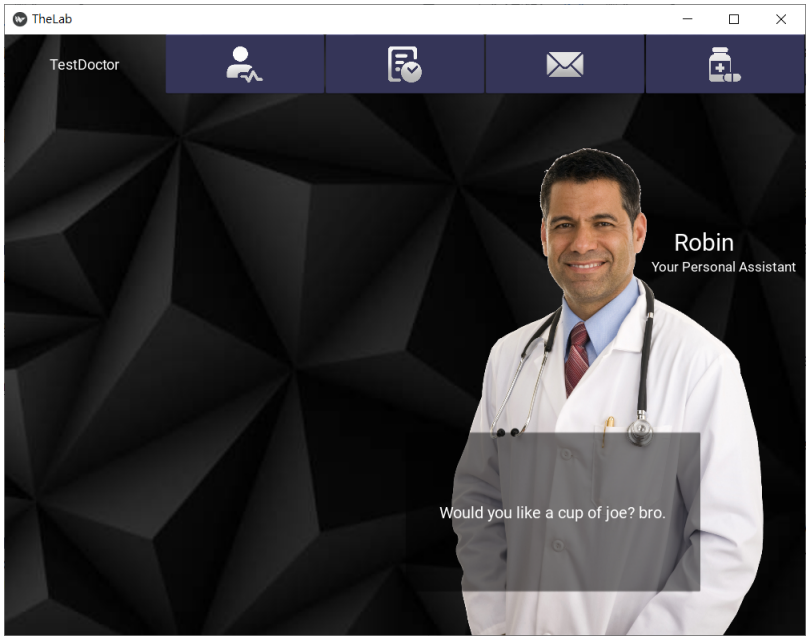

Figure 14: This is the preview of Mainwidget. The doctor will switch to this page after successfully logging in, which vaguely represents the general architecture design of the entire front-end.

- There is a row of buttons at the top. Press to switch pages. These buttons exists on every main page, allowing doctors to jump more conveniently.
- There is a personal assistant in front of you. Press the dialog box and the assistant will talk to you. Currently, the conversations are random. If necessary in the future, you can communicate with other classes to implement functions similar to prompt notifications.

Mainpage

mainpage is a class mainly provides the function of switching the page, allowing other pages to inherit and increase the reuse rate of the program.

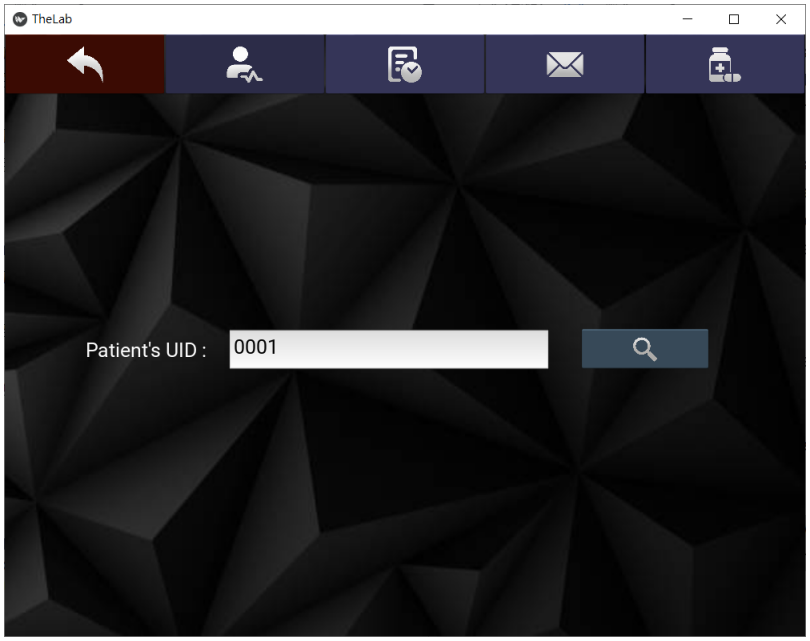

Figure 15: The prview of patient_info_layout

Patient_info_layout

The page firstly get input from textbox, and use `get_info_by_id()` to call serials of methods in `patient_info_t` to load info of the patient the doctor wants to search. Then, it will call `display_info()` to open a subprocess and pass the patient's info it gets. The subprocess can use a describe object to present these info. The describe object will firstly process the info by using `process_info(String)`, put it into a suitable

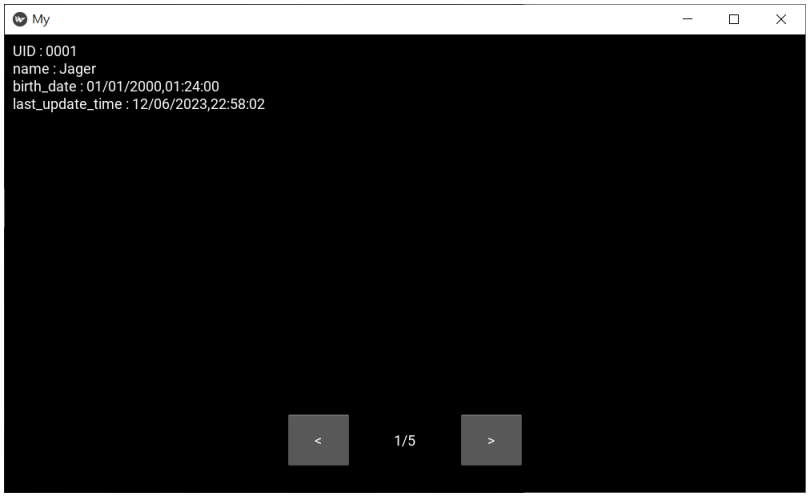

Figure 16: The prview of describe object

format and divide it into parts. Then, it will present these information by using a scoll window, there are buttons to go prev and next pages, each pages present individual part of informations it just processed by `process_info(String)`.

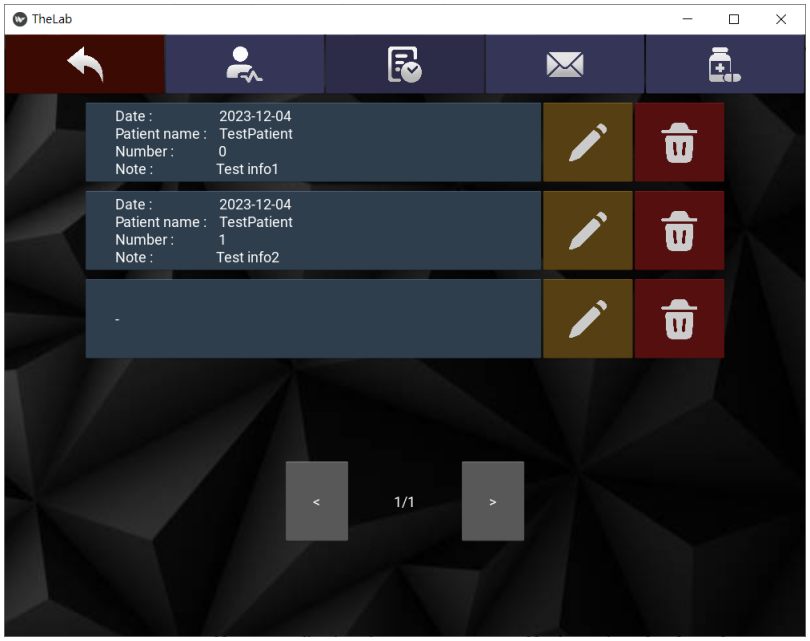

Figure 17: The prview of reservation_layout

Reservation_layout

Reservation will load the reservation objects of medium (linebot), divide it into multiple pieces, store them as attributes, and display it in the main window as an embedded page. Each piece of data will display the name of the reservation person, reservation time, reservation time and remarks. The button below can call the prev_page and next_page methods to browse other reservation information.

- `delete_resv():` Activated by the red trash can button. Will delete reservation the button belongs to from its own attributes, then will call `fix_page()` to refresh the embedded page.
- `chg_edit_resv():` Activated by the yellow pen button. The method will change current page to `EditResv` class, doctor can edit the reservation the button belongs to. After press the save button the data will be stored to that reseravtion and called `change_to_Reservatoin()` to return to original page.

Chat_layout

The approach of Chat is very similar to that of Reservation. It will load the Message objects of medium (linebot), divide them into blocks, and display their summary on the embedded page. What is different from Reseravtion is the button part, which will be explained in detail below.

- `check_mail():` Activated by clicking the summary of that message. It will call a subprocess and passing needed arguments such as sender, date, message. Then the subprocess will display these information by `MailDisplay`.
- `delete_mail():` Activated by the red fire button. Will delete message the button belongs to, then will call `fix_page()` to refresh the embedded page.
- `reply_mail():` Activated by the yellow paper airplane button. It will also call a subprocess to let doctor write message to the sender. After pressing save button, the message will be send to the sender by using `reply` method that the mail have defined in medium(linebot)'s `Message`.

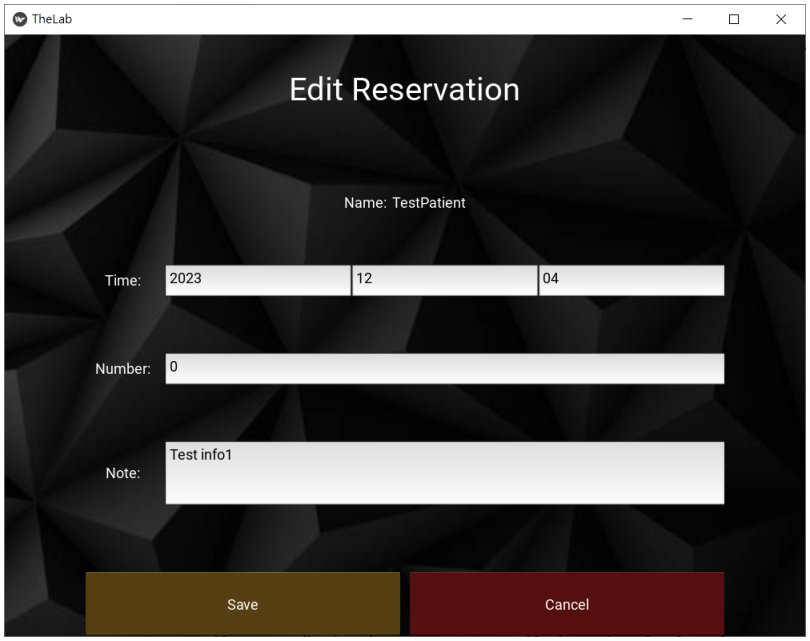

Figure 18: The prview of EditResv

Prscribe_layout

Doctors can create/attach prescribe information to speific patient such as associative data, lab report or diagonosis. The following will introduce the functions of each method and button in detail.

- `update_scollview()`: Activated by the preview button. Will load the textinput right of "Key Value:" text, process them into proper format and update the scroll window right of "Preview:" text. The reason to create such a function like this is cause associative values are strictly format, doctors can preview the result to check if there is something wrong by this method.
- `open_...()`: Several methods behave very similarly. They all start a subprocess to allow doctors to edit data and store it in their corresponding attrubute.
- `finish()`: After all the data has been edited, press the finish button to call. First, the function will search for the corresponding patient based on the entered UID. If it is found, it will directly load the patient's information (`patient_info_t`). If the patient cannot be found, the program will first ask the doctor to create a file for the patient. Next, save all the entered data into a new `reports_t` object, then save `reports_t` into a new `med_record_t` object, append the data into the `patient_info_t` object just loaded or created, and finally save the updated `patient_info_t` to database

4 Code

All the code can be found here.

5 Conclusion

In this project, we develope a E-healthcare management system, which solves the syncing problem between different health systems by using local and central database, and facilitate most of the services by transfer them into internet. Even more, we introduce the automatic disease detectors, which is able to detect multiple diseases by leveraging the machine learning techniques.

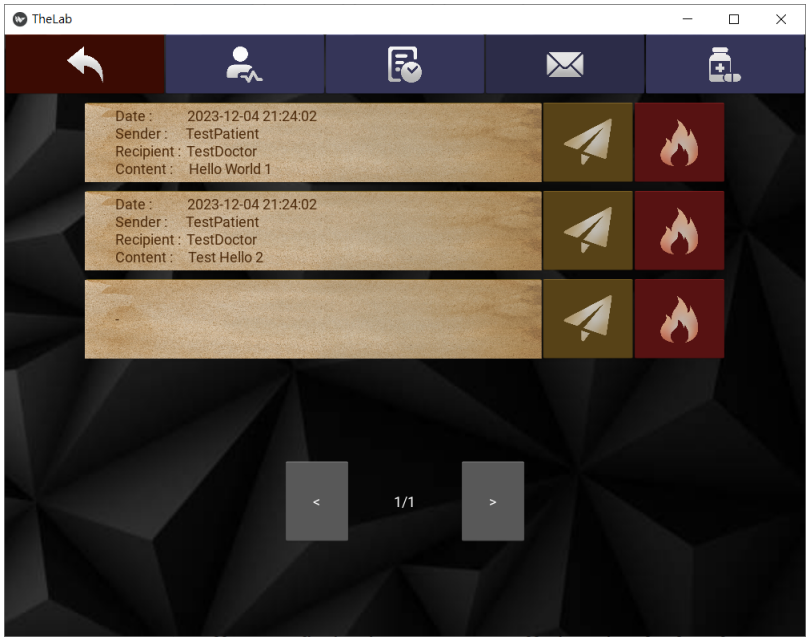

Figure 19: The prview of Chat_layout

In technological aspect, we use object-oriented programming to shorten the development time, and modularize the whole system into several parts, like database, frontend, medium, and detector (processor). With that, the system is highly scalable and easy to maintain.

6 Contribution

`B103040012` (Cheng-Han Hsieh, 謝承翰): the design of the whole architecture, database, paper report `B103040001` (Shih-Yu Sun, 孫世諭): medium, frontend, linebot, coordination and communication, paper report `B103040051` (Chia-Huang, 黃嘉彥): Doctor GUI(include design, data process, images, code) paper report `B103040008` (Ting-Hao Hsu, 許廷豪): Preprocessor and Detector, paper report

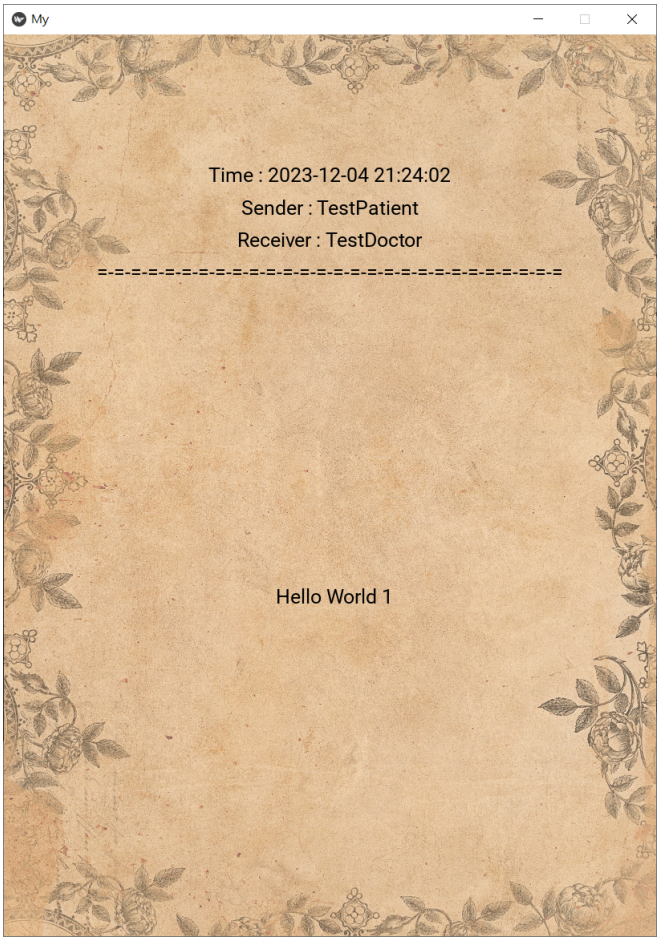

Figure 20: The prview of MailDisplay

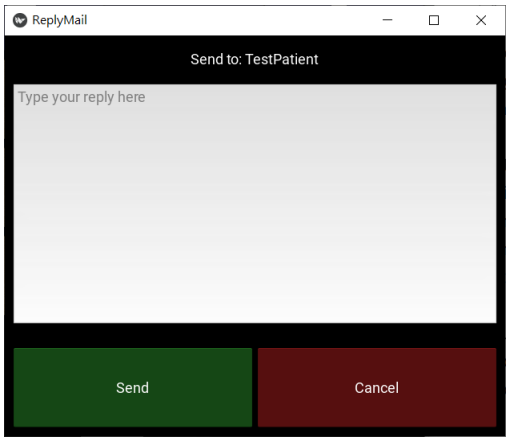

Figure 21: The prview of ReplyMail

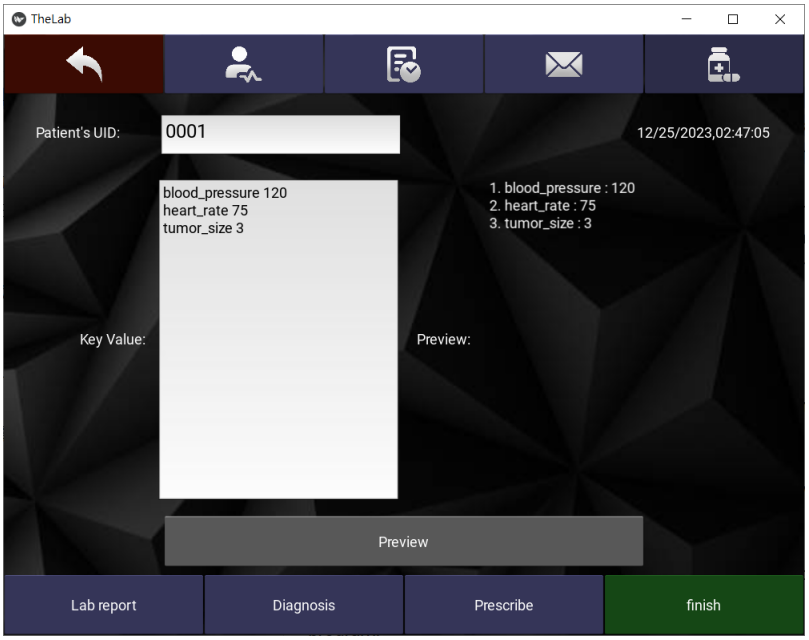

Figure 22: The prview of Prescibe_layout