Telemetry ECG Holter Monitoring System Windows Client

by

Linfeng Xu

MEng, University of Victoria, 2021

A Project Report Submitted in Partial fulfillment of the

of the Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

Supervisory Committee

Telemetry ECG Holter Monitoring System Windows Client

by

Linfeng Xu

MEng, University of Victoria, 2021

Supervisory Committee

Dr. XiaoDai Dong, Department of Electrical and Computer Engineering,

University of Victoria (Departmental Member)

Dr. Issa Traore, Department of Electrical and Computer Engineering,

University of Victoria (Departmental Member)

Abstract

In this report, we describe a Windows client of a smartphone based telemetry ECG holter monitoring system. Traditional long period ECG holter monitoring requires patients to visit a clinic personally, and does not allow clinics to check electrocardiographs in real time. The ECG holter system we built solved the disadvantages of traditional ECG holter monitoring. The new system is built by one back-end and two front-ends. One front-end is a cell phone application for patients, the other one is a Windows client for clinics. Particularly, we focus on the software architecture and user interface of the Windows client, the mechanism of WinForms design and interaction, implementation techniques, and illustration of user interface. Additionally, we introduce imported third party software packages which help build the client, and recommend future developments. Basing on Microsoft .NET framework makes the client adaptive well with Windows. By using WinForms, the client is wieldy and versatile. Importing Json.NET provides us a simple and convenient tool to solve JSON serialization and deserialization issues.

Acknowledgments

I would like to thank my supervisor Dr. Xiaodai Dong for her continuous support and her invaluable suggestions without which I could not have completed this project. Her patience and encouragement have backed me up the whole time, and she has always been open and honest to me, I would have never completed my degree without her supervision.

Additionally, I would like to thank my parents and my friends for their motivation and support through my studies.

Table of Contents

List of Figures

List of Code Snippets

Abbreviations

ECG	Electrocardiogram
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
SSL	Secure Sockets Layer
WinForms	Windows Forms
JSON	JavaScript Object Notation
URL	Uniform Resource Locator
UI	User Interface
EO	Entity Object
VO	Value Object
CLR	Common Language Runtime
CER-S	Continuous ECG Recording Suite
AMPS	Analyzing Medical Parameters for Solutions
UTC	Coordinated Universal Time

Chapter 1 Introduction

1.1 Background

Heart disease, also known as ischemic heart disease or coronary heart disease, refers to the buildup of plaque in the heart's arteries that could lead to a heart attack, heart failure, or death. It is the 2nd leading cause of death in Canada. Data from the Public Health Agency of Canada's Canadian Chronic Disease Surveillance System (CCDSS) indicate that [15]:

- About 1 in 12 (or 2.4 million) Canadian adults age 20 and over live with diagnosed heart disease;
- Every hour, about 12 Canadian adults age 20 and over with diagnosed heart disease die.

To detect heart problems and monitor heart’s health, an electrocardiogram test is usually recommended. An electrocardiogram, commonly known as an ECG or EKG, records how patients’ heart is functioning by measuring its electrical activity through the placement of small sensors on the chest, arms, and lower legs. By measuring the rate and regularity of heartbeats, it can show an abnormal heartbeat, the presence of any damage, or enlargement of the heart muscle. An ECG can also show if the heart is not getting enough oxygen, as well as show the effects of drugs or regulating devices [16]. But the traditional ECG test requires patients to visit clinics frequently, and wait for appointments. This is inefficient and ineffective, especially to patients with risk. Therefore, we design and build a telemetry ECG monitoring system to deliver a convenient and efficient service. In this chapter, we briefly introduce the Telemetry ECG monitoring System, and outline the whole report.

1.2 Introduction to Telemetry ECG monitoring System

The traditional long period ECG monitoring, also known as holter monitoring, is inconvenient for both clinics and patients. It requires patients to visit the clinic to pick up and wear the device, and return back the device in twenty four hours. For clinics, clinics are not able to check patient status at all during the course. Clinics cannot access the electrocardiograph data until the device is retrieved. For patients, patients would waste a lot of time commuting if patients do not live near the clinic.

In this Telemetry ECG monitoring System, patients do not need to go back and forth to the clinic. Clinics set up device storage sites in the areas where they provide service. Patients only need to go to the closest site to pick up and return the phone and device. After patients acquire the phone and device, the clinic communicates with the patient via video conference, and guides patients to set up the phone application and wear the device. During the course of monitoring, the clinic is capable of checking the electrocardiograph at any time via our designed Arbutus Holter Windows Client.

The Telemetry ECG monitoring System is built by one back-end and two front-ends. One front-end is the Arbutus Holter Windows Client, which is a user interface application that is provided to nurses. The other front-end is the cell phone application, which is a user interface application that is provided to patients. Both front-ends generate and require numerous types of data. Data is sent to the back-end server and stored in the database.

Fig. 1 illustrates the structure of the system.

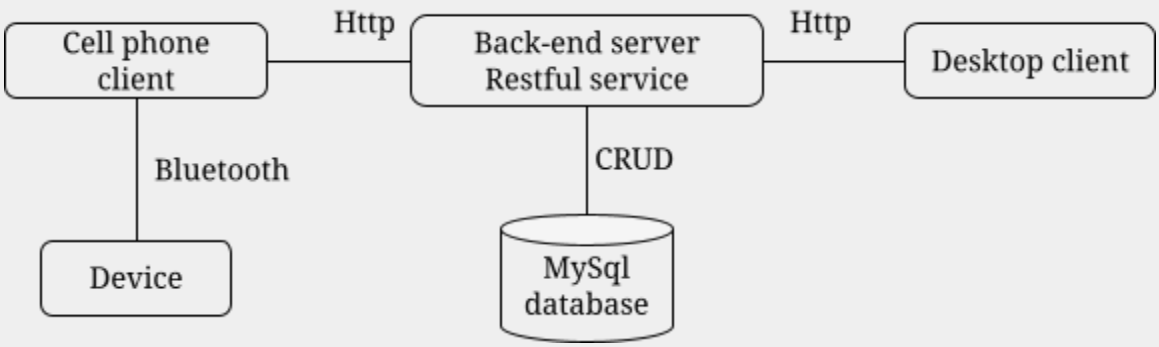


Fig. 1 Telemetry ECG Monitoring System structure

The general workflow of three terminals is the following.

1. The Arbutus Holter Windows Client generates numerous types of datas that are related to the monitoring such as patient information, appointments, ECG tests, etc.. Data types are based on the inputs from the nurse.
2. The Arbutus Holter Windows Client sends data to the server through the provided API via HTTP protocol. The server stores them into the database after some processing.
3. The device sends the collected data to the cell phone application via Bluetooth protocol. The cell phone application sends data to the server through the provided API via HTTP protocol.
4. The server classifies and stores received data based on the time and device properties of data.
5. When the Arbutus Holter Windows Client requires downloading data, the server finds all datas belonging to the ECG test that the client required, and sends them in a package.

1.3 Outline of report

The report is written in accordance with the following outline.

Chapter 2 explains the internal structure of the Arbutus Holter Windows Client, and mechanism of the project.

Chapter 3 describes the composition and functionalities of the Arbutus Holter Windows Client, and our implementation of complex functions.

Chapter 4 introduces the third party software packages which are essential to the project.

Chapter 5 summarizes the finished work and looks forward to possible future improvements.

Chapter 2 Client User Interface

In this chapter, we focus on the Arbutus Holter Windows client. First, we analyze the needs of target users. Then we explore the interactive interface and functionalities of the client, and how we implemented the functionalities.

2.1 Functional requirement

The client's target user is the clinic. The client is required to provide fundamental services to clinic staff such as:

- User admission (registration and login).
- Manage patient information.
- Create and update appointments
- Start ECG holter monitoring tests.

In addition to these, there are some services requiring higher authorization to access such as management of ECG recording devices and cellphones. This divides the client into two types of sections in terms of authentication and authorization, which are public section and nurse section. For the public section, the user is able to access it without authentication. For the nurse section, the user has to access it with authentication. They and their forms have relationships as shown in Fig. 2.

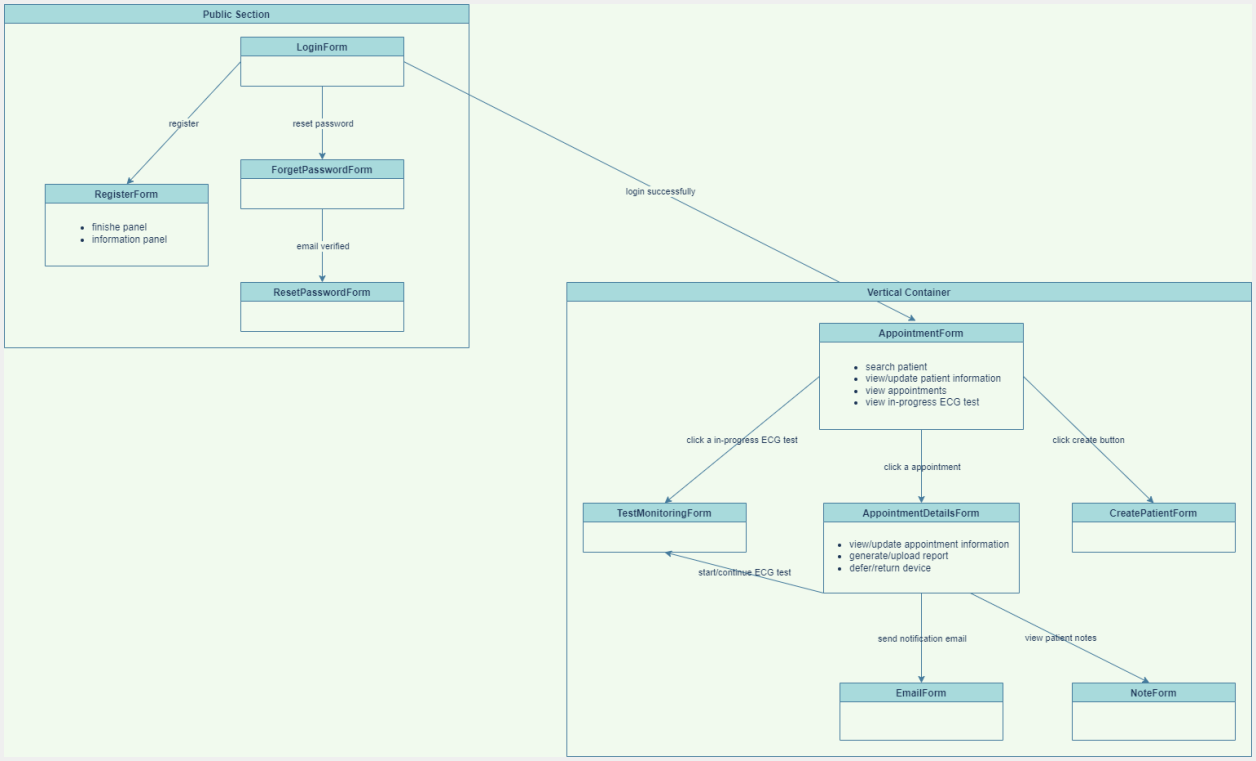


Fig. 2 Public section and nurse section

2.2 Public section

The public section includes the following four forms:

- LoginForm
- RegisterForm
- ForgetPasswordForm

- ResetPasswordForm.

The LoginForm is the entrance of the client.

2.2.1 Login

Fig. 3 LoginForm

The LoginForm is the first form displayed after the program starts, which has admission functions. As shown in Fig. 3, there are two textboxes placed in the center to let the user enter their credential information. The placeholder inside the textbox indicates its type. Either clicking the “login button” or pressing enter key triggers the login event. The event sends the text of two textboxes as credential to the server via an HTTPS request. The server verifies it and gives feedback, and the program responds differently based on feedback. In our case, there are two responses. First is that there is no matched user in the system, or matched user with wrong password. The program displays a message box

to inform the user login failed and remain in the login form as shown in Fig. 4. Second is that the user with nurse authorization. The program hides the LoginForm and displays the AppointmentForm.

Fig. 4 Login unsuccessfully

Before login, users obviously need to register their account, and they possibly forget the password so that they have to reset it. Clicking the “register button” guides users to the RegisterForm, and clicking the “forget password button” guides users to the ResetPassword form.

2.2.2 Registration

In the RegisterForm, there are two panels. A panel is a control that contains other controls. We can use a panel to group collections of controls such as a group of buttons. All controls added to the panel are children of the panel. Therefore, we are able to

manipulate all children's controls at once by manipulating their parent control. Usually, a registration includes two steps, which are entering user information and email address verification. These steps are sequential and should not interfere with each other. The two steps are not complicated, and hence do not need to be separated into two forms. Therefore, we use panels. First, creating two panels to group needed controls for each step. The info filling panel and verification panel as shown in Fig. 5 and 6, and setting the verification panel to invisible. Since the verification panel is the parent of all labels, textboxes, and buttons, all of them are invisible to users. Once the user finishes the first step, the verification panel is set to visible. Then users can proceed to the next step.

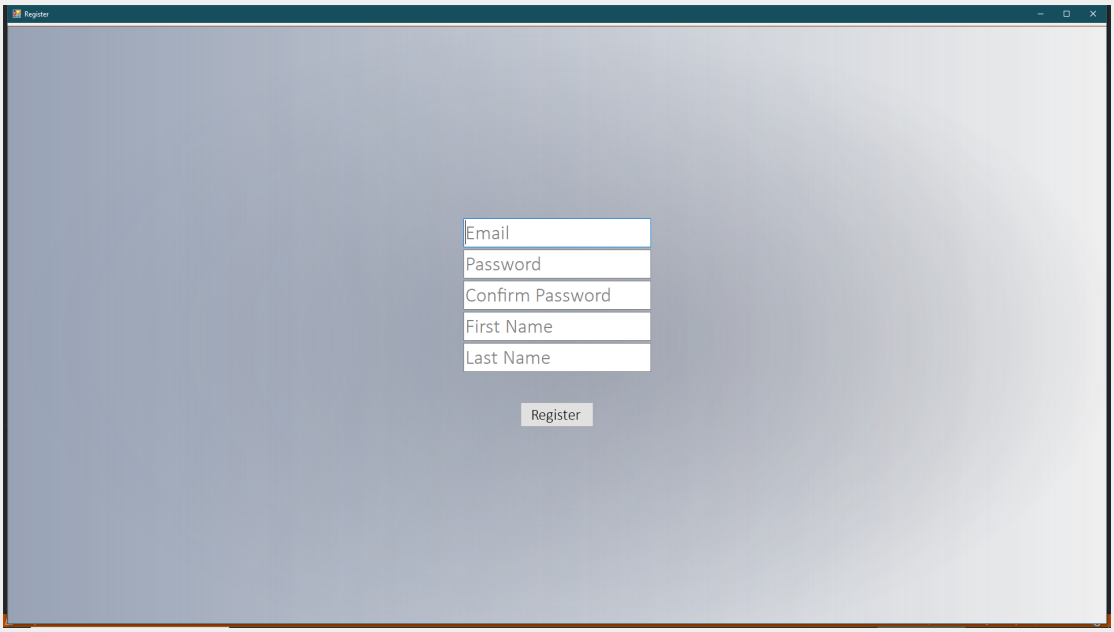


Fig. 5 Info filling panel in the RegisterForm

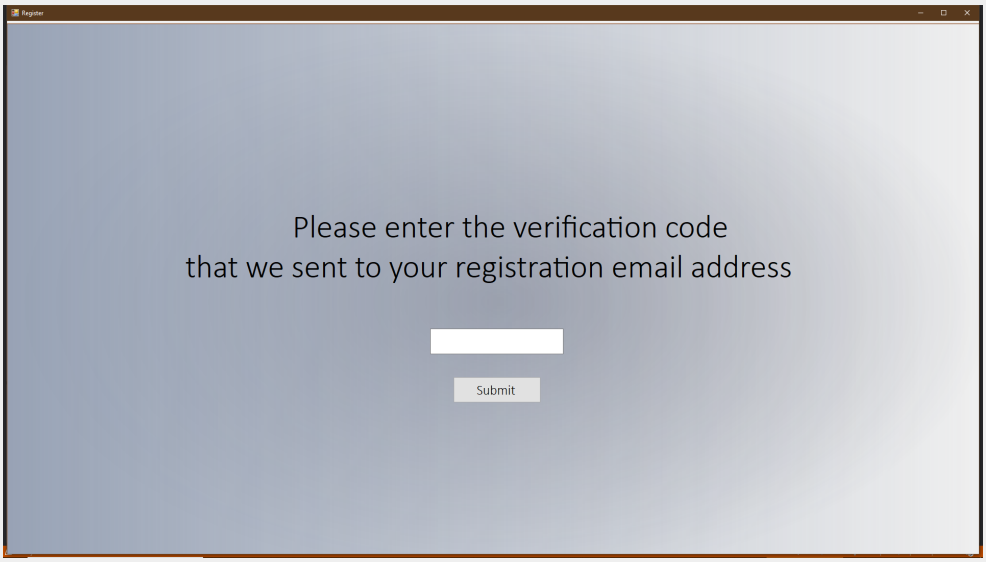
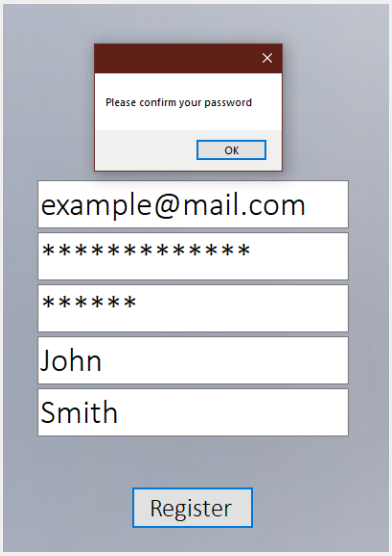


Fig. 6 Verification panel in RegisterForm

In the info filling panel, users must enter their email address, password, and names, because these four pieces of information are mandatory in the system. The client checks the information entered according to the following rules.

- Every textboxes must be filled.
- The length of password must be greater or equal to 10.
- The content of password and confirm password must be the same.

The user cannot proceed to the next step unless the entered information meets the requirement above. Otherwise, the client gives a corresponding warning as shown in Fig. 7 to Fig. 9.



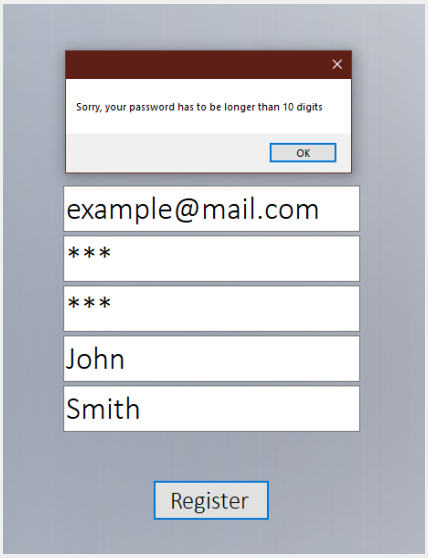


Fig. 7 Missing field Fig. 8 Inconsistent password Fig. 9 Password too short

If all requirements are not violated, the server sends an email containing a verification code to the email address entered. The user needs to enter the code in the verification panel. If the code is correct, the system automatically closes the form and goes back to the LoginForm.

2.2.3 Reset password

Usually, users should follow the following steps to reset the password.

- Enter the email address that needs a new password.
- Enter the verification code that the system sent.
- Set the new password.

Similar to registration, we use two panels in the ForgetPasswordForm to achieve the first two steps, and the ResetPasswordForm to achieve the last step. Therefore, there are two forms for resetting forgotten passwords.

2.3 Nurse section

After successfully login, users with nurse authorization are guided to the nurse section. The nurse section includes the following forms:

- AppointmentDetailsForm
- AppointmentForm
- CreatePatientForm
- EmailForm
- NoteForm
- TestMonitorForm.

The first form that users use after login is the AppointmentForm, and it integrates major functions. Users operate on the AppointmentForm most of the time.

2.3.1 Appointment form

The AppointmentForm integrates several functions, and it can be divided into two sections. Patient information management section, appointment and ECG test management section.

2.3.1.1 Patient information management

As shown in Fig. 10, the patient information management section is constructed by three groups:

- Search patient group
- Patient list group
- Patient detail information group

Each group is assembled by a groupbox control. Similar to the panel we introduced before, the groupbox is also a collection type control. Unlike a panel, a groupbox has a visible borderline and a title as shown in Fig. 10.

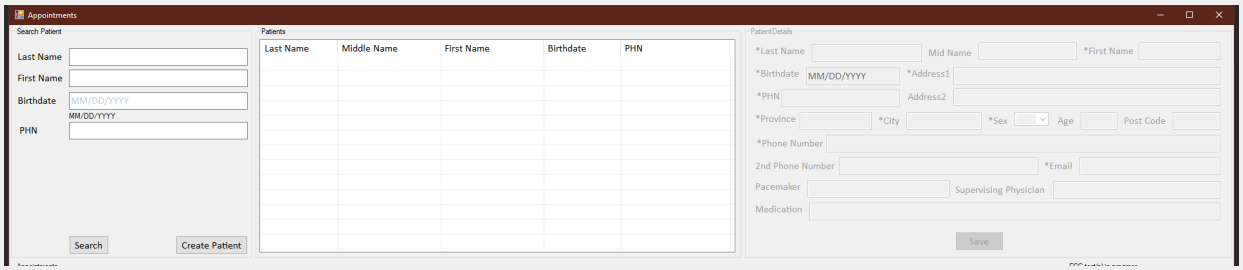


Fig. 10 Patient information management section

In the search patient group, users can search patients through any one of four entries:

- Last name
- First name
- Birthdate
- Personal health number

The birthdate has to be the certain format to be acceptable, which is “MM/DD/YYYY”. The search result is listed in the patient list group if there is any result. Otherwise, the system informs the user that there is no result. The user can select one patient to check more detailed information. The detailed information is displayed in the patient detail information group. If the user needs to update some attributes, the user can edit it directly in the patient detail information group. Clicking the save button saves the changes.

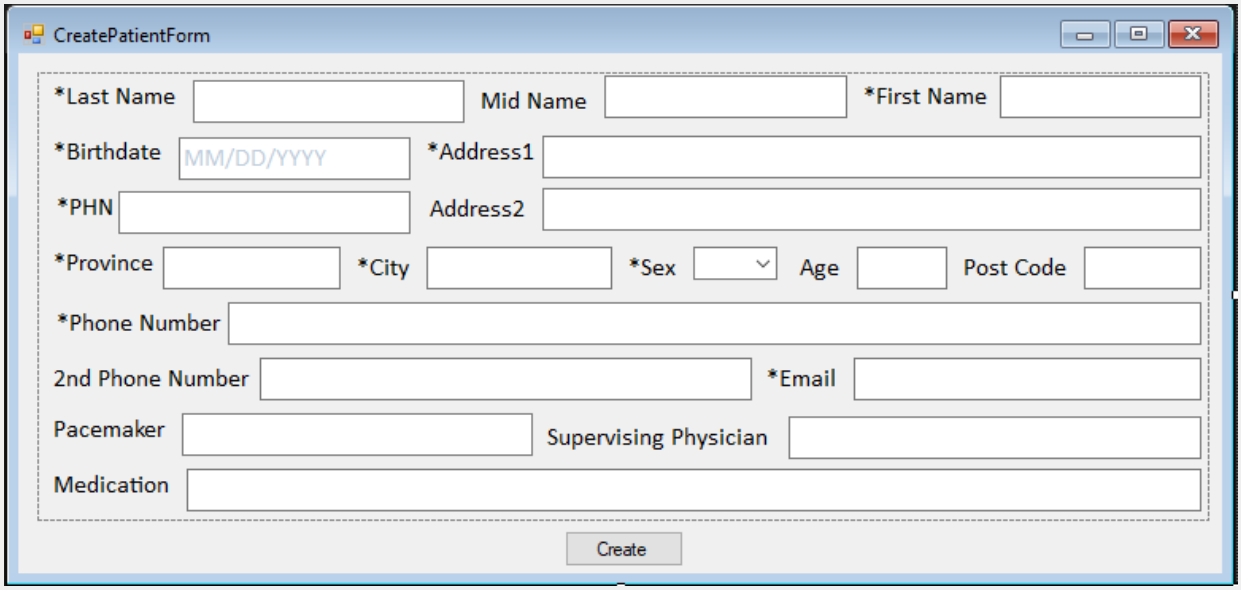


Fig. 11 CreatePatientForm

To create a new patient, users can click the create button on the bottom right corner of the search patient group to open the CreatePatientForm. For users' convenience, the CreatePatientForm is exactly the same as the patient details group as shown in Fig. 11. According to the design, the following attributes are mandatory to a patient object:

- Last name
- First name
- Birthdate
- Address
- Personal health number
- Province and city of current living location
- Sex
- Phone number
- Email address

In the form, the above ten attributes are marked with an asterisk. Before creating or updating the patient, the system checks whether ten attributes are filled.

2.3.1.2 Appointment and ECG test management

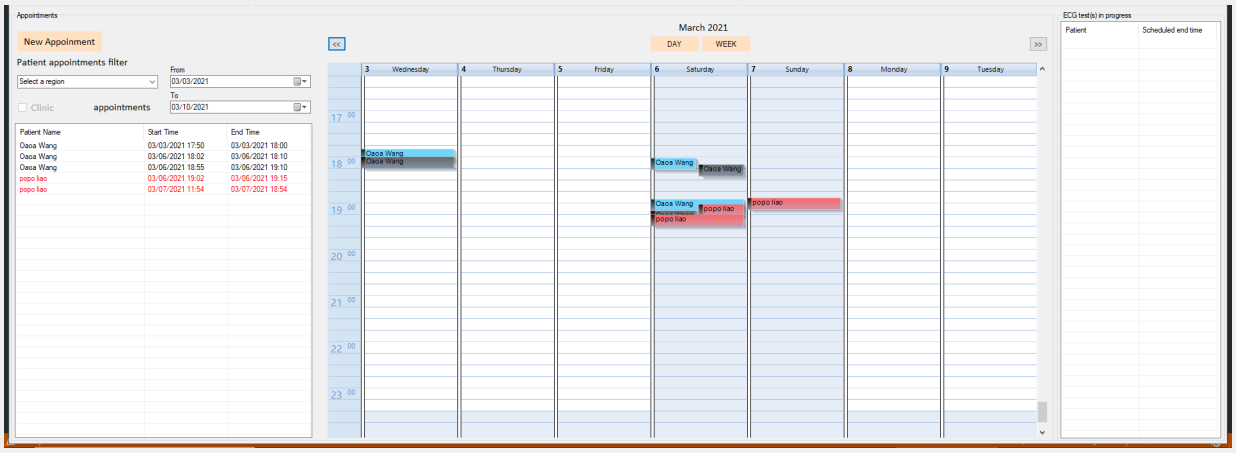


Fig. 12 Appointment and ECG test management section

The Appointments and ECG test management sections are on the bottom half of the AppointmentForm as shown in Fig. 12. It can be divided into two groups, which are the appointments group and the in-progress ECG test group. In the appointments group, the part on the left side displays appointments as text in a list, and the part on the right side displays appointments as tiles in a calendar. In the appointment list, a regular appointment is displayed in black text. In the calendar, the blue tile marks the start time of a regular appointment, and the black tile marks the end time of it. If an appointment becomes abnormal, text and both tiles are in red.

For one appointment, the return of the assigned device is essential information. If the assigned device is not returned on time, it may cause that the patient who is needing the device cannot acquire it. This appointment is considered abnormal. Hence, it is important that the user deals with the problem. In order to guarantee that the user can acknowledge the device status in time, the client automatically refreshes both the list and the calendar every ten minutes. Every appointment that the assigned device is not returned on time is marked as abnormal.

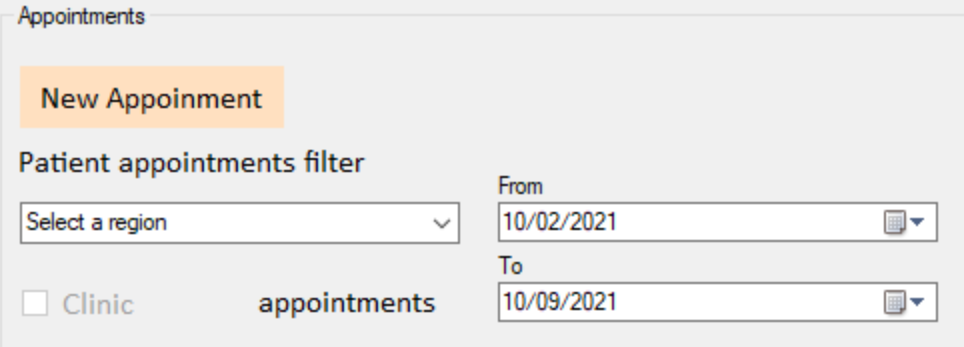


Fig. 13 Button, checkbox, and time pickers

As shown in Fig. 13, there is a button to add new appointments, and several other controls above the appointment list. Clicking the button opens an empty appointment details form, and users can create a new appointment there. The other controls are used as filters of appointments displaying. They are one combobox, one checkbox, and two date pickers. The combobox is used to choose the region where devices are placed. The checkbox is used to differentiate one patient's appointments and all clinic's appointments. By default, the checkbox is unchecked which means all clinic's appointments are displayed. When the user selects a patient from the patient list, the checkbox becomes checked and the patient name is displayed nearby. Only appointments of the patient are displayed. Uncheck the checkbox brings the list and calendar from one patient mode to all clinic mode. Two date pickers are the range of time when appointments are, and the default range is one week. Both list and calendar display appointments that conform to the filter's result. Figs. 14 and 15 illustrate results with or without selecting a patient.

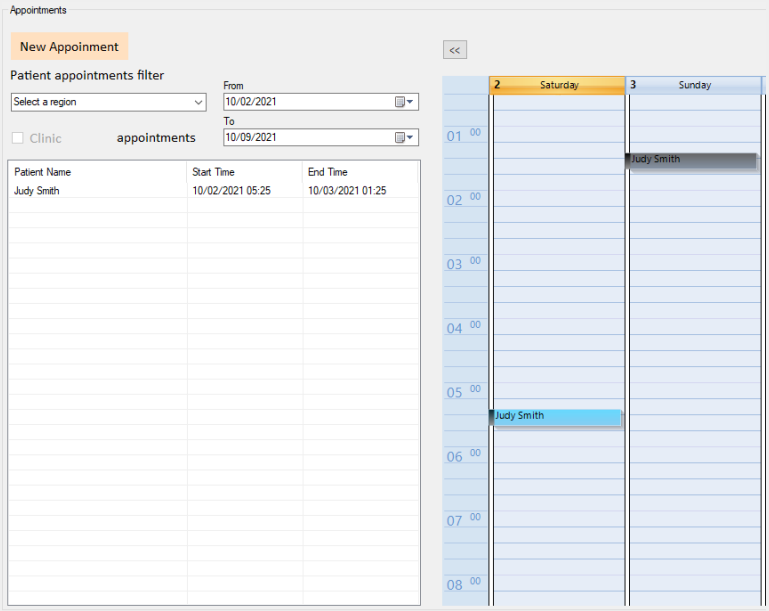


Fig. 14 Without selecting a patient

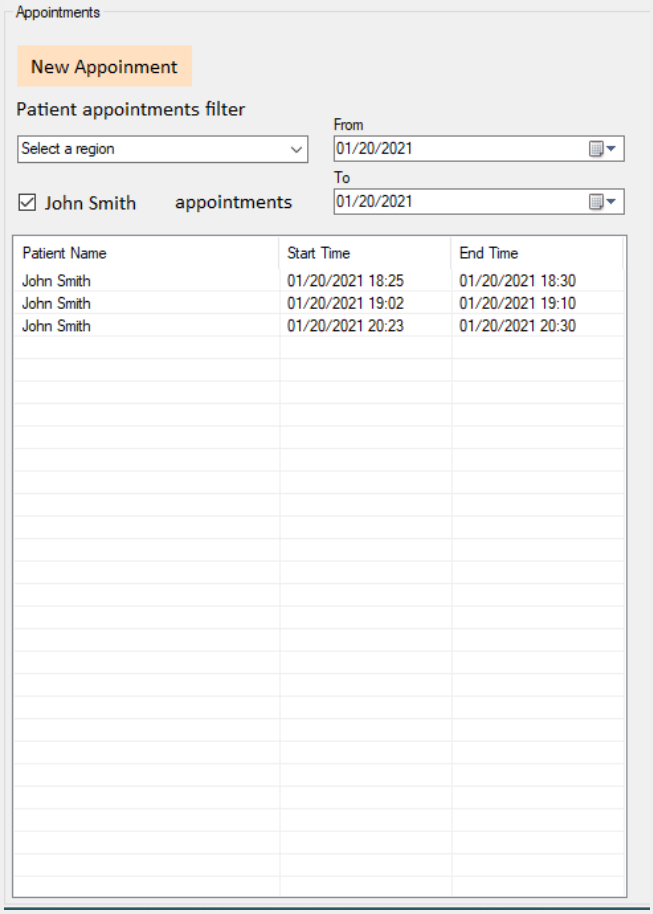


Fig. 15 Select one patient

There are four buttons above the calendar, and the buttons with arrows are the page turning buttons. The calendar is turned to the last page or next page according to the button that is clicked. The day and week buttons are used to switch calendar displaying mode. By default, the calendar is in week mode. The user can select day mode if they need to check more detailed appointments arrangement for one specific day.

Because one ECG test monitoring course could last for dozens of hours, and multiple tests may proceed simultaneously. Users may frequently quit and enter ECG test monitoring. It is necessary to provide users an entry to acknowledge and enter an in-progress ECG test. The in-progress ECG test group displays all on-going ECG tests. Similar to the appointments list, the client refreshes the in-progress ECG test every ten minutes. Double clicking the ECG test item opens the test monitoring form.

2.3.2 Appointment details form

In the AppointmentForm, double clicking on the text in the list or the tile in the calendar opens the AppointmentDetailsForm as shown in Fig. 16. The AppointmentDetailsForm is used to provide detailed information and derived functions of an appointment. It can be divided into two sections, which are the appointment details section and the derived functions section.

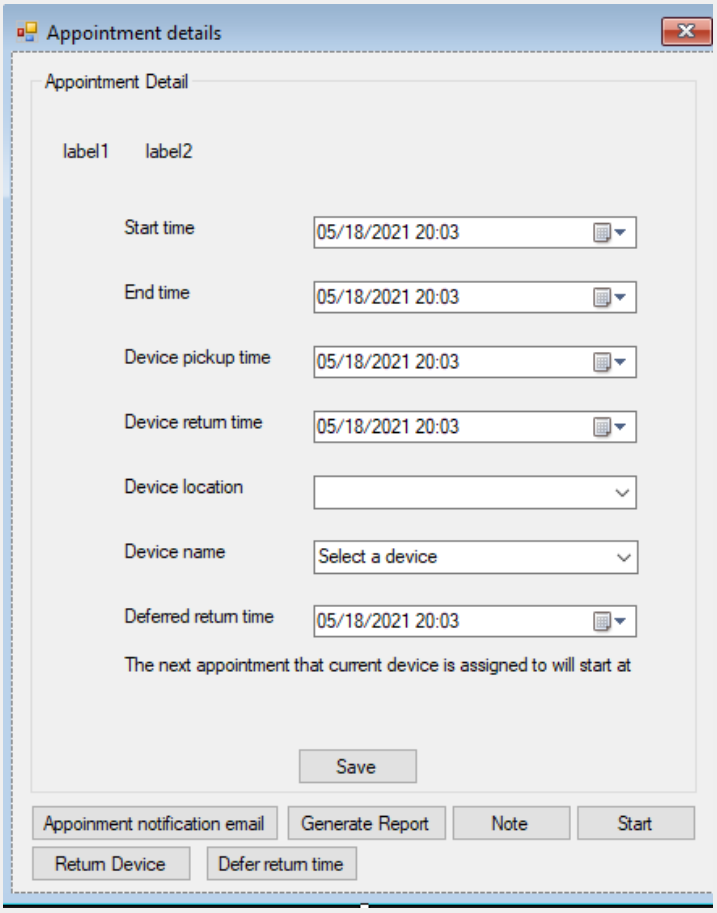


Fig. 16 AppointemntDetailsForm

The appointment details section allows users to view and modify the details of one appointment. Depending on the user operation and appointment status, the section could be used to create, modify, or view details of an appointment. The section is constructed by five datetime pickers, two comboboxes, and several labels. The deferred return time picker is invisible unless the device return time has to be deferred.

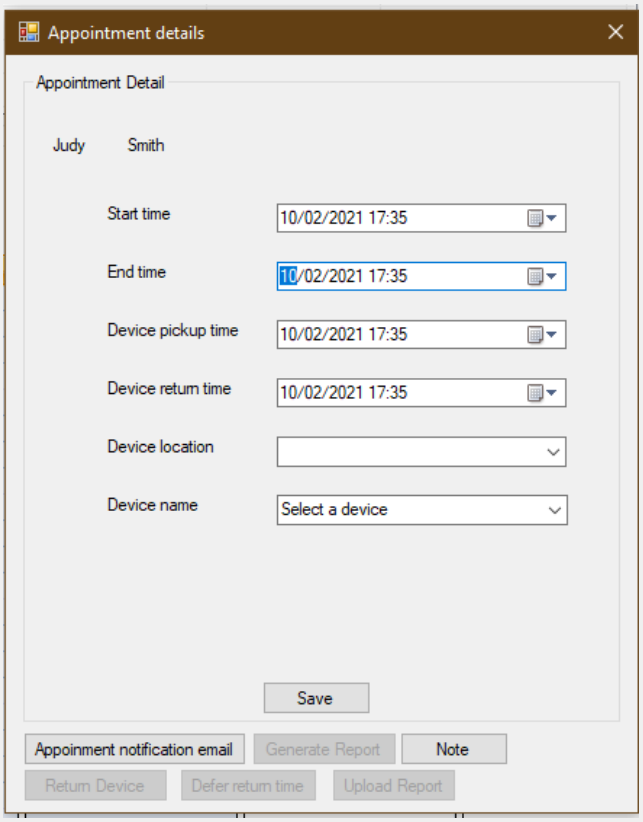


Fig. 17 Create a new appointment

If the user is creating an appointment, then all time pickers are default values as shown in Fig. 17. If the user is checking an appointment, then all time pickers and comboboxes are set values. When the save button is clicked, the client examines whether time values are in the correct order.

- The device pickup time must be earlier than the appointment start time.
- The device return time must be later than appointment end time.
- The appointment start time must be earlier than the appointment end time.

To choose the proper device, the user must select a designated location first. Then the client provides a candidate device list based on the location and device pickup andreturn time. In addition to the appointment details section, the form has the derived functions section. Because the derived functions require the information related to the

corresponding appointment, they are integrated inside the AppointmentDetailsForm. The derived functions section includes the following functions:

- Sending the notification email
- Generating the report
- Viewing the patient's note
- Starting the ECG test course
- Returning the device
- Setting up deferred device return time

2.3.3 Sending notification email

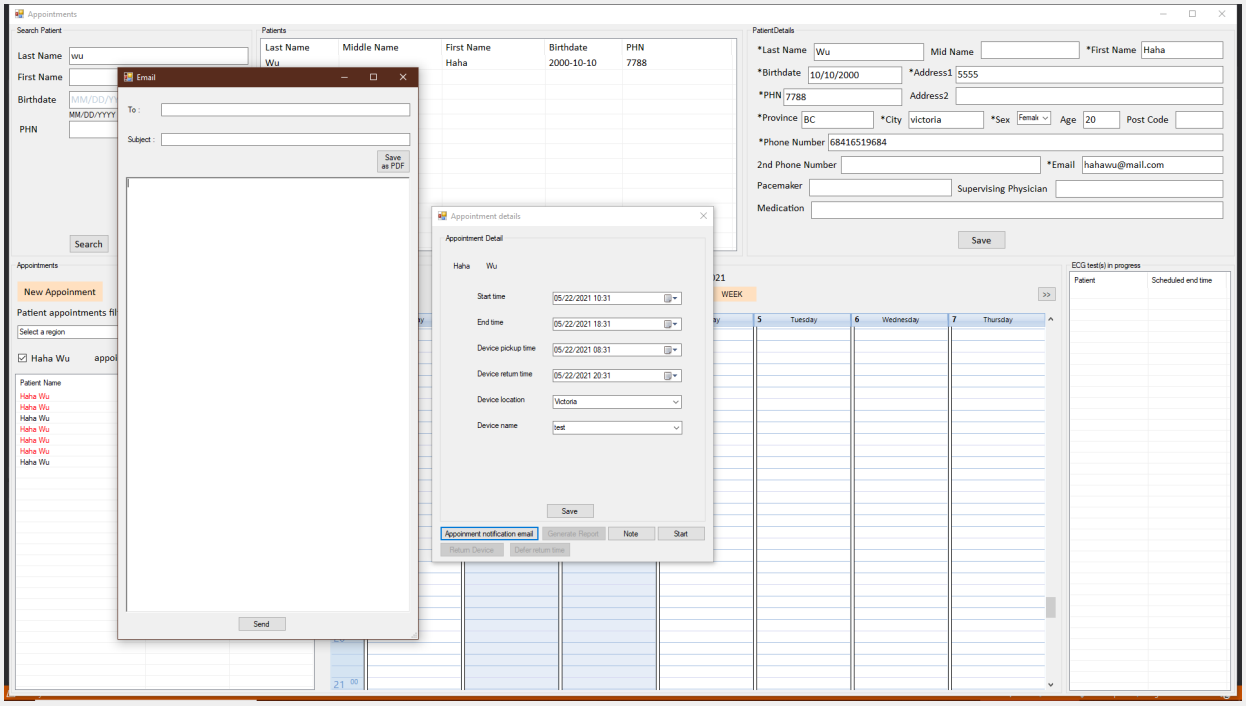


Fig. 18 Notification EmailForm

As shown in Fig. 18, the appointment notification email button is the link to the email form. After a nurse has made an appointment for a patient, the appointment should be sent to the patient via email according to the design. For every clinic, there is a

predefined email template, and the main content of the notification email is generated based on the template. The email content includes the appointment start and end time, and guidance of picking and returning the device. After confirmation, the notification email is sent. If the nurse needs to save the email to the local, they can click the save as pdf button to save the pdf version.

2.3.4 Generating report

Clicking the generate button triggers the generating process. The generate report button is only available when the appointment is finished. The software that is used to generate the report is a third party software, CER-S, which is developed by AMPS (Analyzing Medical Parameters for Solutions). The CER-S takes the ishne file as input, and outputs the diagnose report in pdf. After an appointment is finished, the client downloads the ECG raw data from the server and converts the date to ishne file. This process usually is executed automatically in the background. For more details, please refer to the section “Backstage process”. When the generate button is clicked, the client checks the existence and integrity of the corresponding files. Because the folder is named in the “EcgTest (id)” pattern, which is easy to find. If the files have been downloaded and converted to ishne file, the system generates a command line that opens the CER-S and imports the ishne file. Then the user is able to operate on the CER-S to generate the report. If the files have not been downloaded yet, the client downloads the requested data immediately, and proceeds to the next step after downloading finished.

2.3.5 Viewing note

Clicking the note button opens the NoteForm as shown in Fig. 19. The NoteForm is provided to view notes left by the patient during the ECG test course. During the ECG test

course, patients could write notes about their activities and feelings on the cellphone. To help the medical side analyze, patients should specify the time when they were describing.

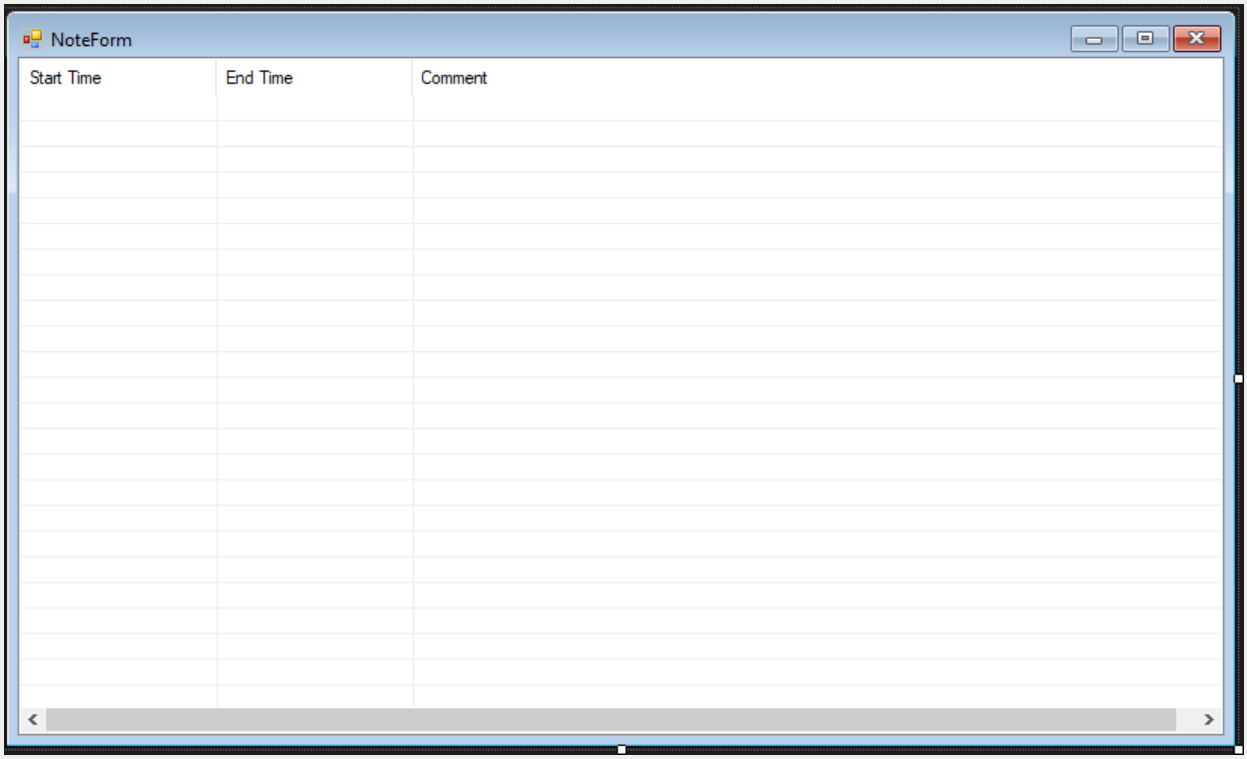


Fig. 19 NoteForm

2.3.6 Starting and continue the test

Clicking the start button opens the TestMonitoringForm. The user can manage the ECG test course in the TestMonitoringForm. Depending on the ECG test status, the start button may be switched by the continue button. The start button is only available for the ECG test that has not started yet. If the ECG test is in progress, the start button is replaced by the continue button. Clicking the continue button also opens the TestMonitoringForm, but the form remains the status that the user left last time. It is equivalent to clicking the running ECG test in the in-progress ECG test group. For instance, if the user started an ECG test course in the hookup status and quit. After continuing the ECG test, the ECG test

remains in hookup status. For the finished appointment, both the start and continue button is disabled. Users cannot access the TestMonitoringForm.

2.3.7 Returning the device

After the device is returned to the storage site, the patient is supposed to update the device status on cellphone application. Therefore, the server acknowledges that the device has been returned, and it is available to future appointments. However, patients may physically return the device, but forget to update it on the cellphone application. This may cause chaos on management of the device. To avoid this situation, the client allows nurses to update the device status from the client side.

2.3.8 Monitoring ECG test

Users can start an ECG test course via clicking the start button in the AppointmentDetailsForm, or continue an in-progress ECG test course via clicking the continue button or in-progress test group. Either action opens the TestMonitoringForm as shown in Fig. 20.

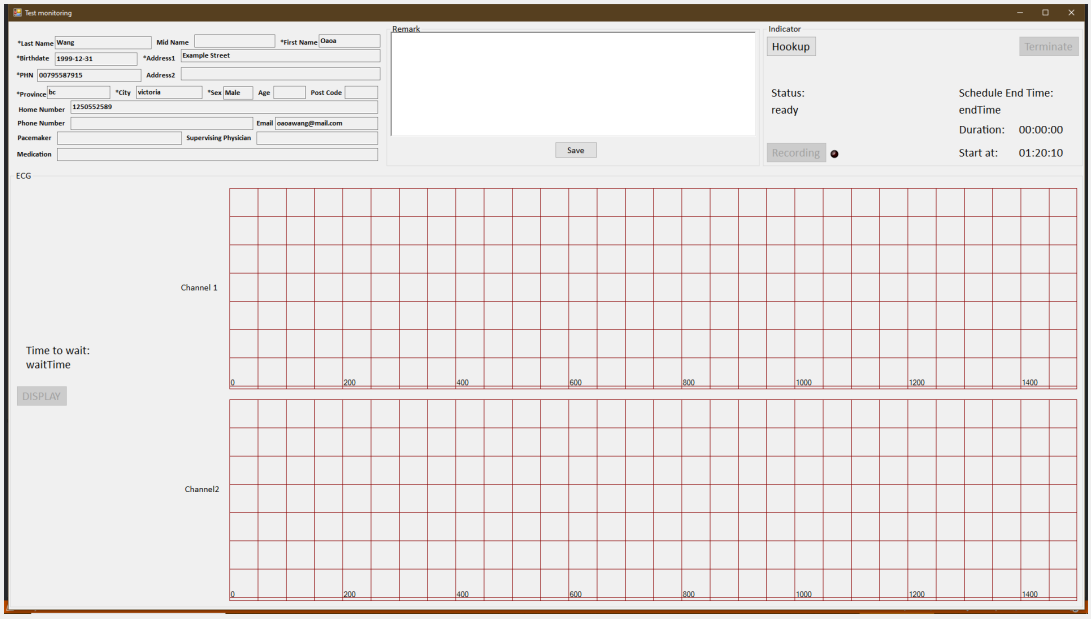


Fig. 20 TestMonitoringForm

The TestMonitoringForm can be divided into two sections, which are the patient information section and the ECG test monitoring section. The patient information section is similar to the patient details section from the AppointmentForm. More than that, there is a remark group in the TestMonitoringForm. This remark is designed to display some extra patient information, such as allergy and medical records. In the AppointmentForm, there is not enough space to place the remark group. Users could view and modify the remark of a patient in the TestMonitoringForm.

The ECG test monitoring section contains two groups, which are the indicator group on the right side of the remark group, and the ECG animation group on the bottom half of the form. The indication group is the part that displays the majority of ECG test course information, such as ECG test status, timing information. It is also the part that the user interacts with for most of the time. The indication group is constructed by three buttons, which are four labels, and a lightbulb. Three buttons are hookup button, record button, and terminate button respectively. As its name says, clicking the button changes the ECG

test course to the corresponding status. Four labels are status label, start time label, duration label, estimated end time label. They display key information of the ECG test course. The status label displays the current status of the ECG test course. Other three labels display the start time, duration, and estimated end time of the ECG test course.

When the ECG test is not started, the start time label keeps ticking like a clock, and other labels do not display any information. In the implementation, the client asks the server to create an ECG test in the database when the user clicks the hookup button for the first time. Hence, the time the hookup button is clicked is regarded as the start time. The start time label stops ticking, and the duration label starts ticking. The estimated end time label displays the estimated end time, and its value usually is the appointment end time. According to the design, one ECG test course is supposed to finish within twenty four hours. Therefore, the client compares the appointment end time and twenty four hours after the ECG test start time. The estimated end time label displays the earlier one.

2.3.8.1 ECG test state

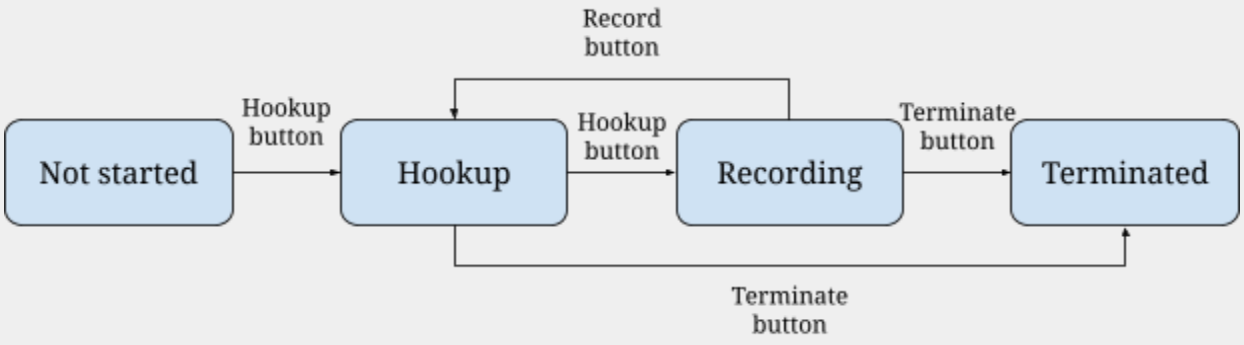


Fig. 21 ECG test state diagram

An ECG test course has four states: notstarted, hookup, recording, and terminated. These four states have the following transitions as shown in Fig. 21. An ECG test course always

starts with the default state, notstarted. Notstarted state can only switch to hookup state, and no state is able to switch back to notstarted state. Hookup state and recording state can switch to each other freely. Both hookup state and recording state can switch to terminated state. The terminated state is the ending state, which cannot switch to any state. The ECG data is uploaded to the server while the ECG test course is in hookup and recording status. The difference between hookup and recording is uploading frequency. In the hookup status, uploading frequency is once every five seconds. In the recording state, it uploads data every one minute.

2.3.8.2 ECG animation

Fig. 22 ECG animation

As shown Fig. 22, the ECG animation group is where the user can view the actual ECG image. Usually, the ECG animation group does not play ECG animation throughout the whole ECG test course. The ECG animation group only plays ECG animation when the user needs to view. Therefore, there is a display button and a waiting time label alongside the channels. Clicking the display button requires the server to transmit ECG data. Because the ECG test course could be either hookup state or recording state. The

time from clicking the display button to actual animation is playing could differ. The shortest waiting period is five seconds, and one minute for the longest. To notify the user that the client is waiting instead of crashing, and how long the waiting period is. Above the display button, the waiting time label displays the waiting time in the form of a countdown. During the transmission from server to the client, there is a possibility of data packet loss. To handle such situations, the program retries three more times before informing users the connection failed. During the retry period, the waiting time label displays as a loading icon.

2.4 Backstage process

According to the design, all ECG data should be downloaded from the server to the local hard drive. So that the client can convert the data to an ishne file which is used to generate reports. But the downloading process is not executed during the ECG test course. The downloading process is executed in the background automatically. To achieve the goal, the program must keep running in the background for a long time. We altered the closing of the AppointmentForm, while users clicking the X icon of the AppointmentForm, the form closing event is cancelled. The form is minimized rather than closed, so it keeps running in the background. When the appointment form is minimized, a balloontip notification is displayed on the screen as shown in Fig. 23. The balloontip notification vanishes in one second, and the hidden icon is added to the taskbar notification area. The icon is a red heart symbol on a white background as shown in Fig. 24. Double clicking the icon can bring the appointment form back, and it is exactly the same as the form when the user closed.

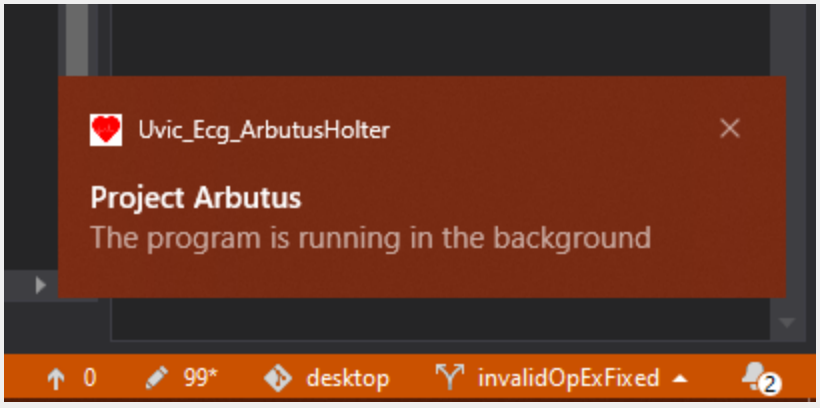


Fig. 23 Notify balloontip

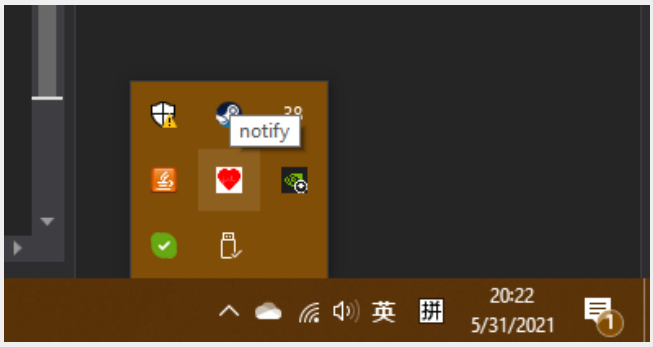


Fig. 24 Taskbar notification

2.4.1 Local folder and file structure

Data that the client needs downloading could be enormous in size, complicated in type. Therefore, for the convenience of client and user. All downloaded data and files are stored according to strict regulations. They are stored in the folder named “Data” under the application root folder. Inside the “Data” folder, all data is classified based on the ECG test they belong to. There is a folder for each ECG test and is named “ECGTestXX”, which “XX” is the id of the ECG test. Other than the test folder, there are three important files that are essential during the downloading process. They are “lastRequestTime.txt”, “failedECGTests.json”, and “failedECGRawDatas.json”. The “lastRequestTime.txt” is used to record the last successful downloading request time. The last successful downloading

request time decides the start of the time period of the next request. The next request only asks for ECG tests after the time to avoid duplication. The “failedECGTests.json” and “failedECGTests.json” are used to record ECG tests and raw datas which download unsuccessfully. All three files are read at the start of each downloading process, and be written at the end of each downloading process.

Inside each ECG test folder, there are test json files and corresponding patient files. If the test has been converted to ishne, the ishne file is inside the test folder as well. For the ECG raw datas, the system needs to store two files, which are ECG raw data json file and actual binary data file. In order to avoid chaos, there is a folder for each ECG raw data and is named “ECGRawDataXX”, which “XX” is the id of the ECG raw data. In general, the local folders and files structure is organized as shown in Fig. 25.

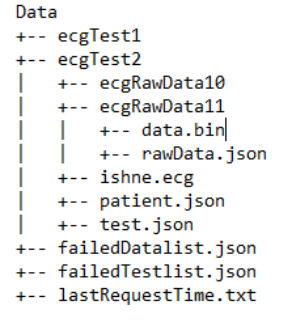


Fig. 25 Folder and file structure

2.4.2 Downloading procedure

The overall downloading procedure is stated below,

1. The downloading program is called by the AppointmentForm timer every one hour.
2. The system reads “lastRequestTime.txt”, “failedECGTests.json”, and “failedECGRawDatas.json” from the local drive.

2.1. If there is no last request time, the system recognizes that the downloading program is the first time to be executed. Then ask users whether they need to download all previous data.

2.1.1. If users answer yes, ask the server to send the id list of all ECG tests that are completed. Recording the current time as last request time and writing it to the local drive. Then proceed to step 3.

2.1.2. If users answer no, record the current time as last request time and write it to the local drive.

2.2. If there is last request time, ask the server to send the id list of all ECG tests that are completed from last request time to now. If the request succeeds, update the current time as the last request time. If the request failed, keeping the old last request time.

3. For every ECG test id in the list returned by the server, ask the server to send the ECG test json file, corresponding patient json file, and the id list of all ECG raw datas that belong to the ECG test. If the request succeeds, writing the ECG test and patient json files to the local drive. Otherwise, add the ECG test to the failed ECG test list.

3.1. For every ECG raw data id, ask the server to send the ECG raw data json file and actual data. If the request succeeds, writing them to the local drive. Otherwise, add the ECG raw data to the failed ECG raw data list.

4. For every ECG test in the failed ECG test list, ask the server to send the ECG test json file, corresponding patient json file, and the id list of all ECG raw datas that belong to the ECG test. If the request succeeds, writing the ECG test and patient json files to the local drive. Then adding the ECG raw datas to the failed ECG raw

data list, and removing this ECG test from the failed ECG test list. If the request failed, keep this ECG test in the failed ECG test list.

5. For every ECG raw data in the failed ECG raw data list, ask the server to send the ECG raw data json file and actual data. If the request succeeds, writing them to the local drive. Then removing this ECG raw data from the failed ECG raw data list. If the request failed, keep this ECG raw data in the failed ECG raw data list.
6. Before converting data to an ishne file, the system needs to check the integrity of files. For every local ECG test folder, check if there is the ishne file. If the ishne file exists, proceed to step 7.
    6.1. If the ishne file does not exist, then check if there is the patient json file. If the patient json file does not exist, get the patient id from the ECG test json file, and ask the server to send the patient json file.
    6.2. Get the ECG test id from the ECG test json file, and ask the server to send the list of all ECG raw datas that belong to the ECG test. Use the list as a reference list in the later comparison. Construct a local data dictionary that takes the ECG raw data id as the index, and the actual data fileinfo as the value. The fileinfo contains the file size of actual data on the drive. It is used to compare the file size between the data on the drive and the data on the server.
        6.2.1. For every local ECG raw data folder, get the id from the ECG raw data json file and read the actual data fileinfo. Add them to the local data dictionary.
    6.3. For every ECG raw data from the reference list, using its id to retrieve the fileinfo from the local data dictionary.

6.3.1. If there is no corresponding fileinfo in the local data dictionary. Then the ECG test folder lacks at least one data, it is incomplete.

6.3.2. If there is a corresponding fileinfo. Then compare the size on the drive and the size on the server. If the size is different, then part of data is lost during the transmission, the ECG test is incomplete.

6.4. After step 6.3, if all ECG raw data of the ECG test folder is complete. Converting the datas to the ishne file and save it in the same ECG test folder.

7. Write updated failed ECG test list and failed ECG raw data list to the files. Preparing for the next downloading round.

8. Finally, one downloading process is finished, waiting for the appointment form timer calling in one hour.

Chapter 3 Client Software Architecture

In this chapter, we explain and introduce the software architecture of the Arbutus Holter Windows Client. First, we introduce the WinForms project which is the project type of the client. Then we focus on the design of the project structure and how we implemented the project.

3.1 Introduction

Windows Forms (WinForms) is a free and open-source graphical (GUI) class library included as a part of Microsoft .NET Framework or Mono Framework[1]. Providing a platform to write rich client applications for desktop, laptop, and tablet PCs[2]. The characteristic of the WinForms project is the Form. In our project, it is constructed by

four namespaces, which are ArbutusHolter, DayView, ECG_ISHNE, and Model. A namespace is a declarative region that provides a scope to the identifiers (the names of types, methods, variables, and etc,) inside it. Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when the program base includes multiple libraries.

The ArbutusHolter namespace is the primary part of the project. It contains all forms and their services. The DayView namespace provides the control calendar which is an essential part of the AppointmentForm. The ECG_ISHNE namespace is responsible for converting ECG data to the ishne file that is used for generating the diagnose report. The Model namespace is a class collection namespace that we defined, and most of them correspond with tables in the database.

3.2 ArbutusHolter

A WinForms project is an event-driven application. When a form is activated, its event handler methods are waiting for the user’s actions such as clicking a button on its interface to trigger. Therefore, the general workflow of a WinForms application is stated below.

1. The form is rendered according to the designer file.
2. User's operation on the interface raises an event.
3. An event raised, and event handler method received.
4. The event handler method processes and responds. If methods are complex, asking an underlayer to do subdivision work.
5. The underlayer does the subdivision work and returns the result back after completion.

Based on the above steps, the structure can be divided into two parts. The form that provides the interface for the user, and the underlayer that executes subdivision work. The underlayer has various functions, and most of which are related to the internet. They have relationships as shown in Fig. 26.

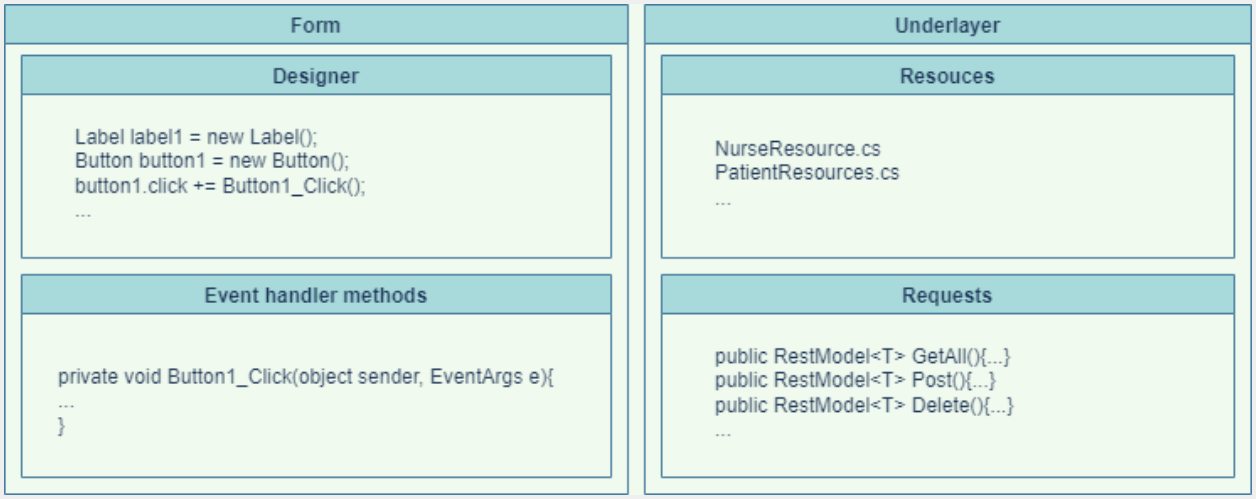


Fig. 26 Form and underlayer

3.2.1 Form

Forms is the core component of a WinForms project, and the Form class is provided by .NET framework. The .NET framework is a software framework developed by Microsoft that runs primarily on Microsoft Windows. It is a platform made up of tools, programming languages, and libraries for building many different types of applications. The Form represents a window or dialog box that makes up an application's user interface. We can customize the form based on demands. Every customized form is a derived class of parent class Form. When we need, a form object can be initialized, and a class can create multiple objects. The file that records customization and renders the window is called form designer. A form designer has two parts, which are the designer file and rendered window which are shown in Figs. 27 and 28.

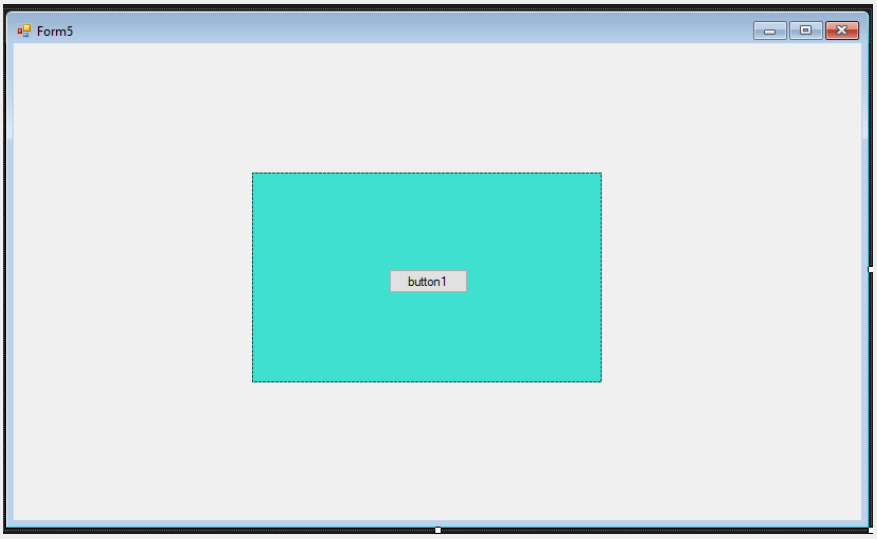


Fig. 27 Example window

```
1 reference
private void InitializeComponent()
{
    this.button1 = new System.Windows.Forms.Button();
    this.panel1 = new System.Windows.Forms.Panel();
    this.panel1.SuspendLayout();
    this.SuspendLayout();
    //
    // button1
    //
    this.button1.Location = new System.Drawing.Point(129, 91);
    this.button1.Name = "button1";
    this.button1.Size = new System.Drawing.Size(75, 23);
    this.button1.TabIndex = 0;
    this.button1.Text = "button1";
    this.button1.UseVisualStyleBackColor = true;
    this.button1.Click += new System.EventHandler(this.button1_Click);
    //
    // panel1
    //
    this.panel1.BackColor = System.Drawing.Color.Turquoise;
    this.panel1.Controls.Add(this.button1);
    this.panel1.Location = new System.Drawing.Point(225, 122);
    this.panel1.Name = "panel1";
    this.panel1.Size = new System.Drawing.Size(330, 198);
    this.panel1.TabIndex = 1;
    //
    // Form5
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(800, 450);
    this.Controls.Add(this.panel1);
    this.Name = "Form5";
    this.Text = "Form5";
    this.panel1.ResumeLayout(false);
    this.ResumeLayout(false);

}

#endregion

private System.Windows.Forms.Button button1;
private System.Windows.Forms.Panel panel1;
```

Fig. 28 Example designer

3.2.1.1 Define and add controls

Components on the form are called controls, and the form itself is also considered as a control. .NET framework provides a lot of basic controls such as button, textbox, list, panel, and etc. A control is constructed by two parts, properties and events. Properties define a control's static attributes, and events decide a control's dynamic action. Text, color, position, size are properties, and moving, clicking, entering, leaving, text changing are events. Different kinds of controls may have diverse properties and events. For some collection kinds of controls such as form and panel, they have subcollections to contain other controls. When a control is added to another control's control collection, they become child and parent. For example, as shown in the figure, "button1" is added to "panel1"'s control collection, then "button1" becomes a child of "panel1". This defines strict hierarchical relations. Actions or properties changed on the parent apply to all its children. Two controls with the same parent may cause blocking if they are overlapped and neither of them is transparent. The basic .NET controls are enough for most scenarios. When demands are unfulfilled, we are able to design our own control.

Fig. 29 LoginForm

Using the LoginForm which is shown in Fig. 29 as an example. The indication of which textbox is used to enter the email address or password is supposed to be achieved by placeholder. However, the default textbox provided by the .NET framework does not have a placeholder function. Hence, we have to create a new class called “placeHolderTextbox” to achieve the goal. The placeHolderTextbox is derived from textbox, and so it keeps all functions of the default textbox. We only need to add the placeholder function on it. Then, add the class to the namespace, and it becomes a new control that meets our need for a placeholder textbox. Similar to this, we create the ECG animation view control used in the TestMonitoringForm and the calendar control used in the AppointmentForm. Because the calendar control is much more complicated, it has a lot of functions to implement. It is inappropriate to be written in one file, therefore it is an independent namespace named DayView.

In addition to controls added to the form or its child, there are some other controls “outside” of the form. They are not visible to the user, and not even interactable to the user sometimes. These are the controls that are mostly utilized by the program and running in the background. The best example is the timer. The timer is a countdown tool, and it is a simple but useful control in the project. Its only function is triggering the event handler method after a certain time and repeating the circulation. The time period is defined and it can be altered anytime. Based on its feature, the timer usually is applied to the process that needs running automatically in the background. Using the AppointmentForm as an example. The program needs to update the appointments and in-progress ECG tests every ten minutes. So that the user does not obtain outdated information. To achieve the goal, we set a ten minute timer, and ask the server for new

information in its event handler method. Then appointments and ECG tests are refreshed.

Apart from this, the timer sometimes is used as a clock. Using the Indicator group in the TestMonitoringForm which is shown in Fig. 30 as an example. We want to use a clock which can display seconds. But the .NET controls library does not provide such a clock, and it is redundant to design a new control. The actual implementation is to set a one second timer. Writing now to a label in the event handler method. Therefore, a label is updated every second. From users' perspective, it is a clock that ticks every second. Similar to this, the blinking function of the indicator light in the TestMonitoringForm is also implemented via a short period timer.

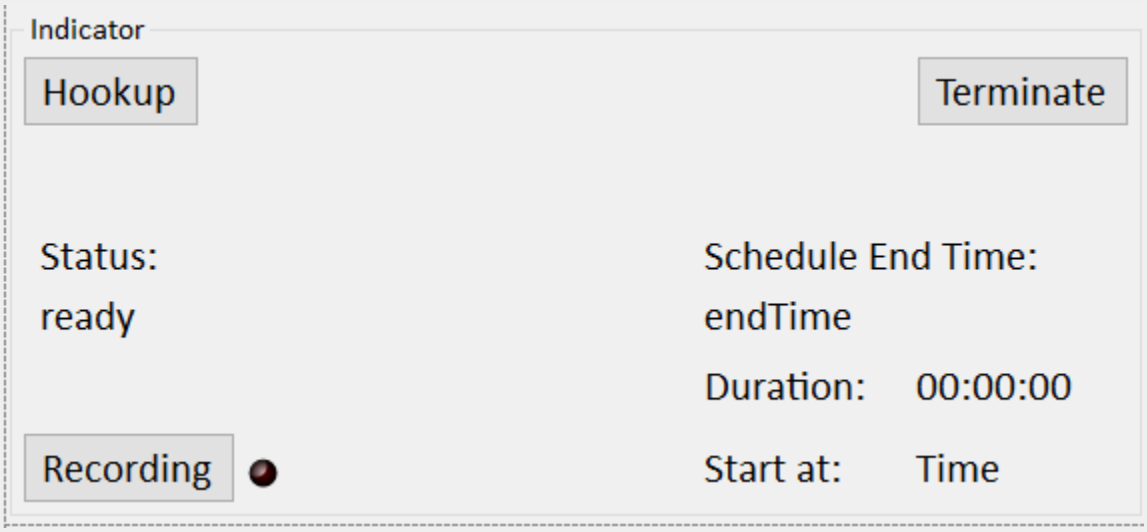


Fig. 30 Indicator group

The designer file decides the initial status of a form, how the form looks when rendered. But to enable a form interactive, it depends on events and event handler methods.

3.2.1.2 Raise and handle events

Events in .NET are based on the delegate model. The delegate model enables a subscriber to register with and receive notifications from a provider. An event sender pushes a notification that an event has happened, and an event receiver receives that notification and defines a response to it. An event is a message sent by an object to signal the occurrence of an action. The action can be caused by user interaction, such as a button click, or it can result from some other program logic, such as changing a property's value. The object that raises the event is called the event sender. The event sender doesn't know which object or method will receive (handle) the events it raises [8].

A delegate is a type that holds a reference to a method. A delegate is declared with a signature that shows the return type and parameters for the methods it references, and it can hold references only to methods that match its signature. A delegate is thus equivalent to a type-safe function pointer or a callback. A delegate declaration is sufficient to define a delegate class. Delegates have many uses in .NET. In the context of events, a delegate is an intermediary (or pointer-like mechanism) between the event source and the method that handles the event. Including the delegate type in the event declaration associates a delegate with an event. .NET provides the EventHandler and EventHandler<TEventArgs> delegates to support most event scenarios. For all events that do not include event data, using the EventHandler delegate. For events that include data about the event, using the EventHandler<TEventArgs> delegate [8]. The data that is associated with the event is the event data. The EventArgs class is the base type for all event data classes. The object of an EventArgs or its derived class can be passed as a parameter in the event handler method. For example, in the AppointmentForm_FormClosing() as shown in Code Snippet 1, we need to cancel users’

operation on closing the form, but allow the program to close the form. In order to differentiate two scenarios, the event handler method relies on the formclosing event data.

```
private void AppointmentForm_FormClosing(object sender,
FormClosingEventArgs e)
        {
            if (e.CloseReason == CloseReason.UserClosing &&
!programClosing)
            {
                e.Cancel = true;
                WindowState = FormWindowState.Minimized;
                ShowInTaskbar = false;
                notify.Visible = true;
                notify.ShowBalloonTip(1000);
            }
        }
```

Code Snippet 1 FormClosing() event handler method

To respond to an event, we define an event handler method in the event receiver. This method must match the signature of the delegate for the event being handled. Code Snippet 2 illustrates how to raise a button clicking event. In the event handler, performing the actions that are required when the event is raised.

```
button1.Click += Button1_Click;
private void Button1_Click(object sender, EventArgs e){...}
```

Code Snippet 2 Raise event

3.2.1.3 Initialize and operate forms

```
namespace WinFormsProject{
      public partial class Form5 : Form{
            InitializeComponent();
            // rendering the form according to the designer
      }
      ...
}
```

Code Snippet 3 Form constructor

In a WinForms project, a form is a derived class of .NET class Form. In addition to the inherited properties and methods from the class Form, we are able to add other features and functions based on the demands to make a custom specialization. Like every object else, a form must be created before being used. Code Snippet 3 is an example form constructor. The very first form in a WinForms project is created inside the Main method. The Main method which is shown in Fig. 31 is the entry point of a C# application. (Libraries and services do not require a Main method as an entry point.) When the application is started, the Main method is the first method that is invoked, and there can only be one entry point in a C# program. In our project, the first form created is the LoginForm.

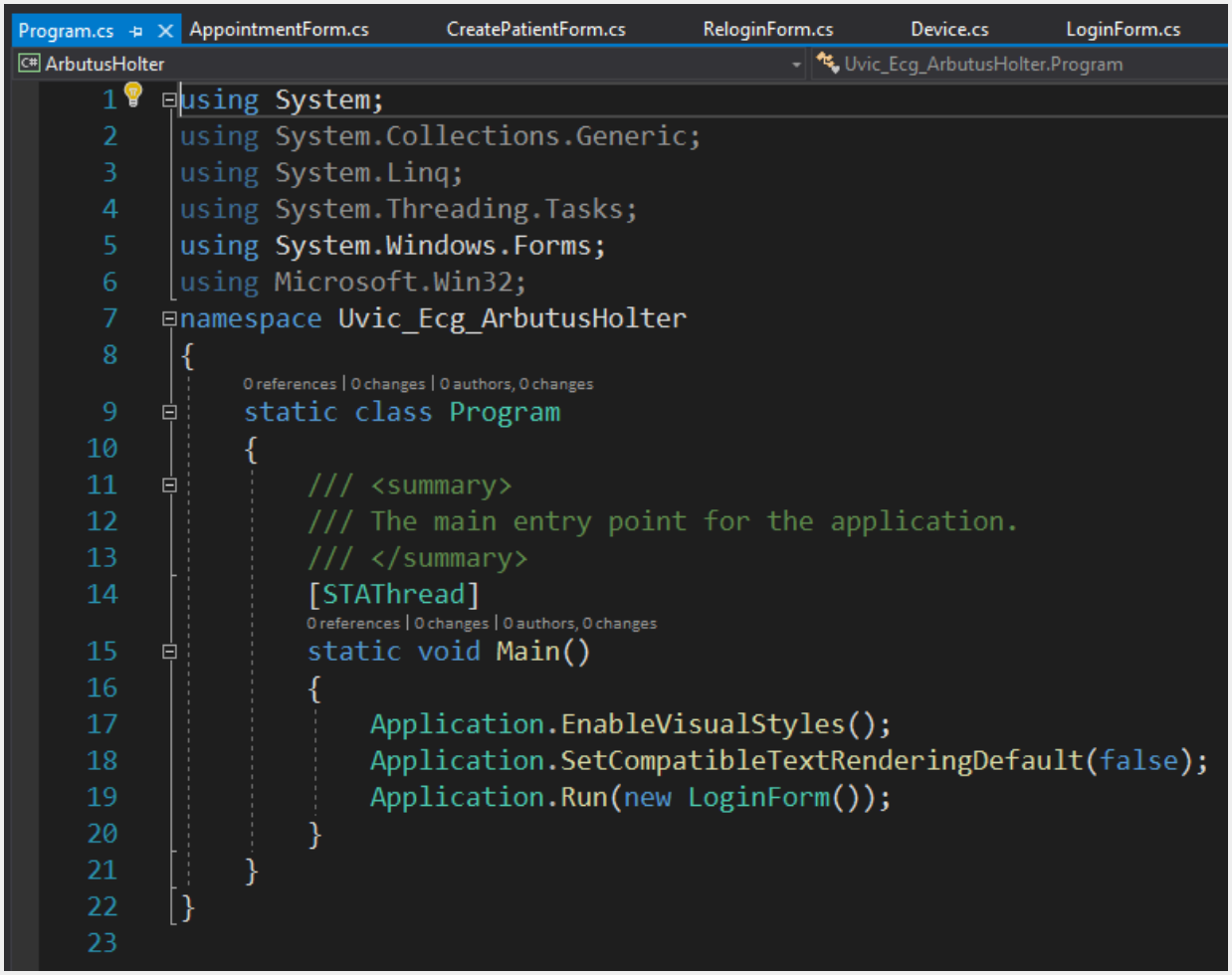


Fig. 31 Main() in program.cs

When we have multiple forms, their order and management are important. Therefore, a more common way is to create and name an instance as shown in Code Snippet 4.

```
Form1 form = new Form1(parameter);
form.Show();
```

Code Snippet 4 Create a form object

Because a form is an object, we are able to use its constructor to pass parameters, and read values from its properties. Taking the TestMonitoringForm which is shown in Code Snippet 5 as an example. There are three necessary objects that need to be inputted from another form, and two outcomes that need to be returned. To achieve the objectives, we

simply make three input objects as parameters in the form constructor. Make two outcomes as properties of the class “TestMonitoringForm”, and write a getter and a setter for them, so that the calling method can obtain them directly when needed.

```
public ECGTest theECGTest { get; set; }
public Appointment theAppoint { get; set; }
public TestMonitorForm(Client client, Appointment app, ECGTest test){ ... }
```

Code Snippet 5 TestMonitoringForm properties

In the calling method, a form can be shown in two ways, which are Show() or ShowDialog(). The Show() shows the form in a non modal form. This means that users can click on the parent form. The ShowDialog() shows the form modally, and users cannot go to the parent form. Additionally, the ShowDialog() can retrieve dialog results, and this helps us to estimate what actions users did. In the Code Snippet 6, a TestMonioringForm is created and shown in ShowDialog().

```
using (TestMonitorForm mainForm = new TestMonitorForm(inClient, theAppoint,
null))
{
    DialogResult res = mainForm.ShowDialog();
    if (res == DialogResult.Yes)
    {
        theTest = mainForm.theECGTest;
        theAppoint = mainForm.theAppoint;
        DialogResult = DialogResult.Yes;

    }
    else if (res == DialogResult.Cancel)
    {
        DialogResult = DialogResult.Cancel;
    }
}
```

Code Snippet 6 Form ShowDialog()

3.2.2 Underlayer

The underlayer means codes that are not involved directly in the operation of forms. They are invoked when forms need them. Most of them are related to internet interaction. We classify them into two categories, resources and requests.

3.2.2.1 Resources

Resources category includes the followings,

- DeviceResource
- ECGDataResource
- MailResource
- NurseResource
- PatientResource
- PublicResource
- ResportResource

They are classified according to the class they are serving. Which means one resource is dedicated to one class. The DeviceResource is only responsible for sending and receiving of device objects. Because there is only one type of object in the file, some variables can be reused to improve performance.

The request needs to be prepared in resources before sending. Since C# is an object-oriented programming language, all data and information exists in the form of an object. However, the server is designed to receive data and give feedback in the form of JSON (JavaScript Object Notation), which is a lightweight data-interchange format. We need to convert an object to a JSON string, and make it http content before sending the request. Moreover, the program needs to set up the url of a request, and one request may

be called multiple times. To avoid doing redundant work, the specific works that send requests are encapsulated into different methods in resources.

```
class PatientResource
{
        RestModel<PatientInfo> restModel;
        private Requests<PatientInfo> requests = new
Requests<PatientInfo>();
        HttpContent content;
        public async Task<RestModel<PatientInfo>> GetPatient(string
lastname, string firstname, string birth, string phn, Client client){...}
        public async Task<string> CreatePatient(PatientInfo newPatient,
Client client)
        {
            string json = JsonConvert.SerializeObject(newPatient, new
JsonSerializerSettings
            {
                DateTimeZoneHandling = DateTimeZoneHandling.Local,
                NullValueHandling = NullValueHandling.Ignore,
                DefaultValueHandling = DefaultValueHandling.Ignore
            });
            content = new StringContent(json, Encoding.UTF8,
"application/json");
            restModel = await requests.Post("patient/information", content,
client);
            return restModel.ErrorMessage;
        }
        public async Task<string> UpdatePatient(PatientInfo updatedPatient,
Client client){...}
        public async Task<RestModel<PatientInfo>> GetPatientById(int pid,
Client client){...}
}
```

Code Snippet 7 PatientResource.cs

Taking the PatientResource in Code Snippet 7 as an example, it contains three properties and four methods. Three properties are restmodel, request, and httpcontent. For details

about restmodel and request, please refer to 2.4 Model and 2.2.2.2 Requests. The httpcontent is the form of transmitted data in a http request. The four methods are GetPatient(), GetPatientById(), CreatePatient(), and UpdatePatient(). Because four methods serve class PatientInformation. To save memory and time, they are encapsulated into the PatientResource. The restmodel and request are not universal types. A restmodel of class PatientInformation is different with a restmodel of other classes. After Json content and url are ready, invoking the request to send the request.

3.2.2.2 Request

The Request is responsible for sending the request to and receiving feedback from the server, and converting feedback to restmodel. The internet interaction is based on the HTTP (Hypertext Transfer Protocol). The Hypertext Transfer Protocol is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems [9]. HTTP defines methods to indicate the desired action to be performed on the identified resource. There are nine HTTP methods: GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE, and PATCH. In our project, the methods GET, POST, PUT, and DELETE are used.

The GET method means to retrieve whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process [10]. The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line [11]. The PUT method requests that the enclosed entity be stored under the

supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server. If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI [12]. The DELETE method requests that the origin server delete the resource identified by the Request-URI [13].

As shown in Fig. 32, the Request is a class that has two properties and seven methods. The two properties are restmodel and integer named “maxHttpExCount”. The meaning and usage of “maxHttpExCount” will be introduced later. The Getall() and Delete() are responsible for GET and DELETE methods. For PUT and POST methods, they are more complicated. For the PUT, there are two methods, which are Put() and PublicPut(). The reason to design two methods for one HTTP method is that they have different root urls.

```
using ...
namespace Uvic_Ecg_ArbutusHolter.HttpRequests
{
    26 references | James, 5 days ago | 3 authors, 11 changes
    public class Requests<T>
    {
        private RestModel<T> restModel;
        private ErrorInfo errorInfo;
        private int maxHttpExCount = 3;
        7 references | James, 180 days ago | 1 author, 3 changes
        public async Task<RestModel<T>> Put(string url, HttpContent content, Client client)...
        1 reference | James, 26 days ago | 1 author, 1 change
        public async Task<RestModel<T>> PublicPut(string url, HttpContent content, Client client)...
        // Post() for httpcontent
        4 references | James, 5 days ago | 1 author, 5 changes
        public async Task<RestModel<T>> Post(string url, HttpContent content, Client client)
        {
            return await Post(url, content, client, null);
        }
        // Post() for multipartformdatacontent
        2 references | James, 5 days ago | 1 author, 1 change
        public async Task<RestModel<T>> Post(string url,
                                             HttpContent content,
                                             Client client,
                                             MultipartFormDataContent multipartFormDataContent)...
        3 references | James, 180 days ago | 1 author, 3 changes
        public async Task<RestModel<T>> PublicPost(string url, HttpContent content, Client client)...
        15 references | James, 138 days ago | 2 authors, 6 changes
        public async Task<RestModel<T>> GetAll(string url, Client client)...
        1 reference | James, 180 days ago | 1 author, 3 changes
        public async Task<RestModel<T>> Delete(string url, Client client)...
    }
}
```

Fig. 32 Request.cs

A URL (Uniform Resource Locator) is a specific type of URI (Universal Resource Identifier). A URL normally locates an existing resource on the Internet. A URL is used when a web client makes a request to a server for a resource [14]. A URL for HTTP (or HTTPS) is normally made up of three or four components:

1. A scheme. The scheme identifies the protocol to be used to access the resource on the Internet. It can be HTTP (without SSL) or HTTPS (with SSL) [14].
2. A host. The host name identifies the host that holds the resource. Host names can also be followed by a port number [14].
3. A path. The path identifies the specific resource in the host that the web client wants to access [14].
4. A query string. If a query string is used, it follows the path component, and provides a string of information that the resource can use for some purpose. The query string is usually a string of name and value pairs. Name and value pairs are separated from each other by an ampersand [14].

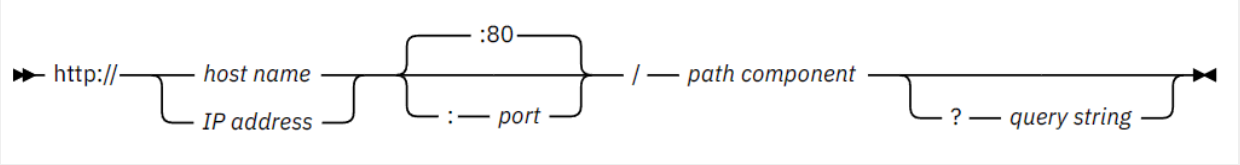


Fig. 33 URL structure[14]

The scheme and host components of a URL are not defined as case-sensitive, but the path and query string are case-sensitive. Typically, the whole URL is specified in lowercase [14].

The components of the URL are combined and delimited as shown in Fig. 33,

- scheme://host:port/path?query

- http://ECG.uvic.ca:8080/v1/test/patient/information?lastname=Chappell&firstname=&phn=

When a resource is setting the url, it does not set a full url that contains the url's scheme and host parts. A resource only sets the path and query parts of an url, and leaves it to the request to combine with the schema and host parts. The schema and host parts that every request shares are defined as the root url. According to the design of the server, requests that require token authentication need to add the name of the clinic in the root url. Requests that do not require token authentication should replace the clinic name with string “public”. Therefore, different requests may have different root urls. The method that handles the PUT has to be separated into two methods. For the same reason, the POST has the Post() and PublicPost() as well. Strictly, the GET and DELETE should also have public version methods . However, the server does not allow any GET and DELETE without token authentication. So that the GET and DELETE only have one method respectively.

3.2.3 Http request exception

During program executing, the errors and exceptions are inevitable. In general, when an exception occurs, the client records the exception in the log, and reports it to the user. Then the client goes back to the idle state, and waits for another operation. However, the internet interaction is different from the offline program. The internet interaction is unstable and easy to be interrupted, so errors and exceptions occur significantly more frequently. For internet exceptions such as packet loss and disconnection, the client is not supposed to notify the user immediately. The client should not report until multiple

consecutive attempts failed. The retry is accomplished in the request. Taking the GetAll() in Fig. 34 as an example, the internet request is placed inside a do-while loop.

The do-while loop is a control flow statement that executes a block of code at least once, and then either repeatedly executes the block or exits. Depending on the code block execution status. The do-while loop is appropriate to our needs. The internet request retries if and only if the application catches a “HttpRequestException” and consecutive occurrences is less than “maxHttpExCount”.

```
15 references | James, 139 days ago | 2 authors, 6 changes
public async Task<RestModel<T>> GetAll(string url, Client client)
{
    bool httpFailure;
    int exCount = 0;
    do
    {
        try
        {
            client.HttpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", client.Token);
            client.Result = await client.HttpClient.GetAsync(client.Url + url);
            httpFailure = false;
        }
        catch (InvalidOperationException invoex)
        {
            throw invoex;
        }
        catch (HttpRequestException hrex)
        {
            exCount++;
            if (exCount > maxHttpExCount)
            {
                throw hrex;
            }
            httpFailure = true;
        }
        catch (Exception ex)
        {
            throw ex;
        }
    } while (httpFailure);
    try...
}
```

Fig. 34 GetAll()

3.2.4 Asynchronization

A control in a form raises an event, and its event handler method asks underlayer resource and request to process. The event handler method responds appropriately based on the feedback. Users can see the reaction on the interface, and everything seems fine. But sometimes the above procedure may take a long time, or even no response. It may be due to internet latency, or it is reading a large file. Everything is waiting under these circumstances, and the client looks like it crashed from users' perspective. To avoid this, processes that may take a long time are supposed to run asynchronized.

In a synchronous application, the whole application halts if any process halts. Because codes are running linearly in one thread. The code behind must wait for the code in front to finish before it can be executed, even though executions are independent. By using Asynchronous programming, the Application can continue with the other work that does not depend on the completion of the entire task [3]. When using an asynchronous method, the application UI is still responsive. Users can still operate the interface, or utilize other functions, even close the application if the user does not have patience for the asynchronous process to finish. The internet interaction may have problems such as latency, packet loss, failure, and etc. In order to avoid unstable internet affects the main process, all internet requests are set as asynchronous in the project.

The core of async programming is the Task and Task<T> objects, which model asynchronous operations. They are supported by the "async" and "await" keywords [4]. If a method is specified as an async method by using the async modifier, two capabilities are enabled. First, the marked async method can use await to designate suspension points. The await operator tells the compiler that the async method cannot continue past

that point until the awaited asynchronous process is complete. In the meantime, control returns to the caller of the async method. The suspension of an async method at an await expression does not constitute an exit from the method, and finally blocks do not run. Second, the marked async method itself can be awaited by methods that call it. An async method typically contains one or more occurrences of an await operator, but the absence of await expressions does not cause a compiler error. If an async method does not use an await operator to mark a suspension point, the method executes as a synchronous method does, despite the async modifier.

```
private async void button2_Click(object sender, EventArgs e){
      int i = await MethodAsync();
      Console.WriteLine(i);
}
private async Task<int> MethodAsync(){
      Task<int> task = Latency();
      Console.WriteLine("hello world");
      return await task;
}
private async Task<int> Latency(){
      await Task.Delay(3000);
      return 1;
}
```

Code Snippet 8 Example async methods

The most important thing to understand in asynchronous programming is how the control flow moves from method to method. Taking Code Snippet 8 as an example, the above functions have similar structure with our project. An event handler method button2_Click() invokes and awaits another method, the MethodAsync(). The MethodAsync() invokes and awaits the Latency() to get an integer and return it. The Latency() delays three seconds before returning. The Latency() mimics the method that takes a long time to run and the process is suspended. In order to avoid blocking the

whole application, the Latency() yields the flow control to its caller, the MethodAsync(). The Latency() returns a “Task<TResult>”, where “TResult” is an integer in this case, and the MethodAsync() assigns the “Task<int>” to the “task” variable. The “task” represents the ongoing process for the call to the Latency(), with a commitment to produce an actual integer value when the work is complete. Because the “task” has not been awaited yet, the MethodAsync() can continue with other work that does not depend on the result from the Latency(). The MethodAsync() outputs the string “hello world” and now the MethodAsync() has run out of work that it can do without the result from the Latency(). Therefore, the MethodAsync() uses an await operator to suspend its progress and to yield control to the method that called the MethodAsync(). The MethodAsync() returns a “Task<int>” to the caller. The task represents a promise to produce an integer result.

This is the general processing pattern of an async method. The caller may do other work that does not depend on the result from the method it awaits, or the caller might await immediately. The method that the caller is waiting for is also waiting for its await method. The Latency() completes and returns an integer result. The integer result is not returned to the MethodAsync() since it already returned a task before. Instead, the int result is stored in the task that represents the completion of the method, the variable “task”. The await operator retrieves the result from “task”. The assignment statement assigns the retrieved result.

3.3 DayView

The DayView namespace is a collection of the files of a control named “Calendar” which is self-defined. Due to its complexity, it is not appropriate to be placed inside the ArbutusHolter namespace. As a complicated self-designed control, the DayView

namespace contains thirteen files. They can be classified into two groups, which are rendering files and functioning files. The rendering files are used to render and draw the control's graphic interface. The functioning files are used to provide functions and event methods. Their roles and relations are similar to the form designer and event handler method. The rendering files are:

- The AbstractRenderer
- The DrawTool
- The ITool
- The Office11Renderer
- The Office12Renderer
- The SelectionTool
- The Appointment
- The DataBoundAppointment

The AbstractRenderer" is used to render the static background of DayView such as table and datetime. "DrawTool", "ITool", and "SelectionTool" are used to render dynamic animation of users' actions such as drag and click. "Office11Renderer" and "Office12Renderer" are two styles of DayView rendering, Microsoft Office 2011 and 2012 style respectively. The functioning files are:

- The AppointmentEventArgs
- The NewAppointmentEvent
- The ResolveAppointmentEvent
- The Selection

The DayView is only utilized in the AppointmentForm. It is used to display appointments as tiles on the calendar. In the AppointmentForm, the DayView uses some basic functions such as displaying, turning pages, and clicking. The DayView uses a class named

"Appointment" to visualize the tiles. Note that the class “Appointment” here is different from the class “Appointment” in the Model namespace. The DayView implements the following events that can be invoked by its caller. In our case, the AppointmentForm utilizes the former two events.

- DayView.NewAppointment()
- DayView.RsolveAppointments()
- DayView.SelectionChanged()

3.3.1 DayView.Appointment

The DayView.Appointment is the class that represents the tiles that are displayed on the interface. The Model.Appointment is the class that represents appointments stored in the database. They are different classes but with the same name. In the program, they can be differentiated via different namespaces as shown in Code Snippet 9.

```
DayView.Appointment appointment = new DayView.Appointment();
Model.Appointment appoint = new Model.Appointment();
```

Code Snippet 9 Same Appointment with different namespace

To display an appointment on the calendar, an instance of the DayView.Appointment is created. The DayView.Appointment only has one constructor as shown in Code Snippet 10, and it sets three attributes to default value. Three fields are “color”, “borderColor”, and “title”, and default values are white, black, and “New Appointment” respectively.

```
public Appointment()
{
color = Color.White;
borderColor = Color.Black;
title = "New Appointment";
}
```

Code Snippet 10 DayView.Appointment constructor

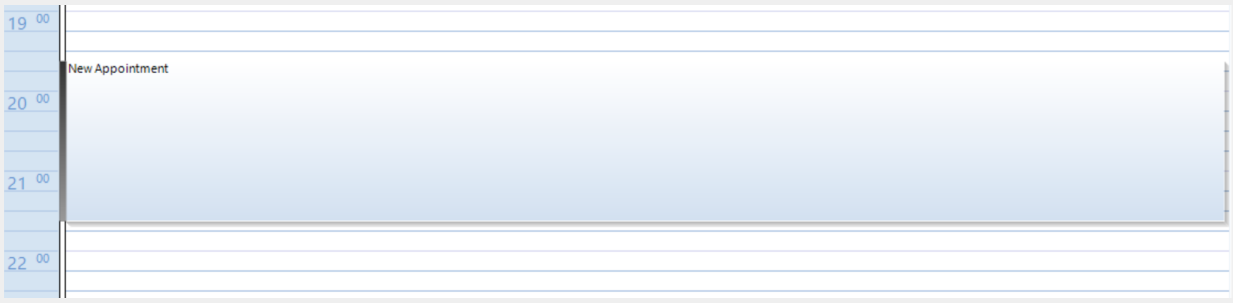


Fig. 35 An appointment in default style

In addition to these attributes in the constructor, a DayView.Appointment contains ten others attributes, which are “allDayEvent”, “appoint”, “conflictCount”, “drawBorder”, “endDate”, “group”, “layer”, “locked”, “startDate”, and “textColor”. All of them are literal except “appoint”. The “appoint” is actually the Model.Appointment (since they are the same name class, name has to be an abbreviation). Therefore, the DayView.Appointment links with the Model.Appointment. After a DayView.Appointment was created, we assigned the corresponding Model.Appointment. A default style appointment is shown in Fig. 35 .

3.3.2 NewAppointment event

```
private void DayView_NewAppointment(object sender, NewAppointmentEventArgs
args){
      DayView.Appointment appointment = new DayView.Appointment();
      appointment.StartDate = args.StartDate;
      appointment.EndDate = args.EndDate;
      appointment.Title = args.Title;
      appointLs.Add(Appointment);
}
```

Code Snippet 11 DayView_NewAppointment()

The DayView.NewAppointment() which is shown in Code Snippet 11 is raised when we want to create a DayView.Appointment. The NewAppointmentEventArgs contains the start date, end date, and title of the new appointment. The code sample application just creates a new appointment, and adds it to the list collection. Besides, the appointment can be created and added outside of the DayView.NewAppointment() as well. As long as the appointment is added into the designated list. When the DayView.ResolveAppointments() is raised, the appointment is displayed. In our case, the designated list is the “appointLs”.

3.3.3 ResolveAppointments event

```
private void DayView_ResolveAppointment(object sender,
ResolveAppointmentsEventArgs args){
      List<Calendar.Appointment> apps = new List<Calendar.Appointment>();
      foreach (Calendar.Appointment app in appointLs){
            if ((app.StartDate >= args.StartDate) &&
                (app.StartDate <= args.EndDate)){
                  apps.Add(app);
            }
      }
      args.Appointments= apps;
}
```

Code Snippet 12 DayView_ResloveAppointment()

This event in Code Snippet 12 is raised when the DayView control needs to show an appointment on a date. The ResolveAppointmentsEventArgs contain the start date and end date of the required range of dates. Namely, the args.StartDate and args.EndDate are the time range that the DayView control is displaying now. The range varies from twenty-four hours to seven days. The if statement here decides whether an appointment falls within the display range. If yes, the appointment is added into the list collection

“apps”. Otherwise, proceed to the next appointment. Then, the OnResolveApointments() invokes the DrawAppointment() to draw the tiles on the control.

3.4 Model

The Model namespace is a collection of classes that represent tables in the database and have no methods. These classes provide the objects that are used in other namespaces such as the Appointment, the PatientInformation, the ECGTest, and etc. Even though classes in the Model namespace roughly correspond to tables in the database, they are not identically the same. The reason that causes such a situation is that the server utilizes the VO (Value Object).

3.4.1 Entity object and value object

On the server, there are classes that are identically the same as database tables. These classes are called “EO”. EO is an acronym that stands for Entity Object. In addition to EOs, the server also has classes that are related to the database tables but different. These classes are called “VO”. VO is an acronym that stands for Value Object. There are two main reasons to create VOs. First, its convenience. EOs are rigid because it has to be identically the same as the database table. But in actual programming we sometimes only need a few properties from one class, or need multiple properties from different classes. If we only use EOs, the program is bloated. Instead, creating VOs is simple and convenient. A VO can use getter to retrieve properties from a EO, and setter to assign properties to a EO. Moreover, VOs support nested structure. We can put another VO inside a VO and retrieve it directly. Otherwise, we can only retrieve object id from a EO, and search the object via the id. Second, for safety. We are not supposed to allow others to infer the database table structure via the class structure. Therefore, the server is the

only one which is allowed to access the database. When the client is visiting, the server should adjust and return VOs based on the request interface. Therefore, classes in the Model namespace are actually identically the same with the VOs.

3.4.2 Classes

In the Model namespace, there are seventeen classes in total, which are ten VO classes and eight other classes. The ten VO classes are the Appointment, the AppointmentMail, the Device, the ECGRawData, the ECGTest, theNurse, the PatientInfo, the Phone, the StatusType, and the Reports. The seven other classes are the RestModel, the Feed, the Entity, the ErrorInfo, the ResultJson, the FileRelatedException, and the TokenExpiredException.

- The Appointment represents the appointment that is booked for a patient by a nurse. It contains four other VOs that are related to the appointment, which are the Nurse, the PatientInfo, the Device, and the ECGTest, and seven datetime variables. Four datetime variables are not nullable, which are the appointmentStartTime, the appointmentEndTime, the devicePickupTime, and the deviceReturnTime. An appointment is not qualified without above four times, and missing one of them may cause problems. Three datetime variables are nullable, which are the reservationTime, the delayedDeviceReturnTime, and the actualDeviceReturnTime. Because an appointment does not necessarily have them.
- The AppointmentMail represents the appointment notification email that is sent from the clinic to the patient. It contains two string variables, which are patient email address and email content.

- The Device represents the ECG recording equipment. It contains information about the sensor that the clinic distributes to every patient. Including the deviceName, the deviceLocation, the phoneId, the clinicId, the occupied. The occupied attribute is a boolean value, it indicates the device is available or not.
- The ECGRawData represents the actual ECG data and its information of every ECG test. Including the receivedTime, the startTime, the endTime, the ECGTestId which is to indicate which ECG test it belongs to. Moreover, it stores the phone status of every piece of ECG raw data, which is used to represent the network connections of a cellphone. Including bluetooth connection between phone application and sensor device, and the internet connection between phone and server. Last, it contains a long variable “size”, which shows the file size of the ECG raw data. It is used in background data downloading to verify that the downloaded data is intact.
- The ECGTest represents an ECG test course. It contains information needed in the course. Including three times, the startTime, scheduledEndTime, actualEndTime. Five VOs, the PatientInfo, the Nurse, the Device, the Appointment, and the StatusType which represents the ECG test course status.
- The class Nurse represents the major user of the application. It contains its names, the clinicId, the phoneNumber, and login username, the nurseEmail, and the password.
- The class PatientInfo represents the patient who enjoys the service of the clinic. It contains detailed patient’s information such as names, address, birthdate, contact information, and medical datas.
- The class Phone represents the cellphone that transmits ECG data from the device to the server. It contains the information of every phone that the clinic distributes

to every patient. Including the clinic id, the phoneMac, which is used to identify the phone.

- The class StatusType represents the ECG test course status. It contains an enumeration object composed of four states, which are NOTSTARTED, HOOKUP, RECORDING, TERMINATED.
- The class Report represents the diagnosis report that is generated for each ECG test. It contains two VOs, the PatientInfo and the ECGTest, and one string variable, the reportContent.
- The class RestModel represents the content returned by the server for any request. It contains two VOs, Feed and Entity, and one string variable, the errorMessage.
- The class Feed represents the part of the RestModel. Since the server may return a large amount of datas, datas would be loaded into several pages in the RestModel. One page is represented by one Feed. The class Feed contains three long variables, which are the total number of pages, the page number, the pageSize. A list of the Entity, the entities.
- The class Entity represents one item in a page. It contains one integer variable the id, and one universal variable the model which could be any class.
- The class ErrorInfo is a collection class. It collects all pre-defined error messages.
- The class ResultJson is a simple class with only one property, which is a string variable message. To replace the entity in RestModel.
- The class FileRelatedException and the class TokenExpiredException are self designed exception classes. The FileRelatedException is used in the background downloading process, and the TokenExpiredException is used when the

authentication token is invalid. Because the program needs to do special handling for these two types of exceptions, they are differentiated from regular exceptions.

3.4.3 Class structure

Classes and structs have members that represent their data and behavior. A class's members include all the members declared in the class, along with all members (except constructors and finalizers) declared in all classes in its inheritance hierarchy. Private members in base classes are inherited but are not accessible from derived classes. The following list enumerates the kinds of members a class or struct may contain.

- Fields: Variables declared at class scope. A field may be a built-in numeric type or an instance of another class. For example, a device class may have a field that contains its name as shown in Code Snippet 13.

```
// The most common way to define a field. The first keyword defines the
access level, the second keyword defines the type.
private string deviceName;
```

Code Snippet 13 Example filed

- Constants: Fields whose value is set at compile time and cannot be changed as shown in Code Snippet 14.

```
private readonly string deviceName = "Device A";
```

Code Snippet 14 Example constant

- Properties: Methods on a class that are accessed as if they were fields on that class as shown in Code Snippet 15. A property can provide protection for a class field to keep it from being changed without the knowledge of the object.

```
// A property usually contains getter and setter, setter can filter some
unwanted values.
public string DeviceName
{
    get { return deviceName; }
    set
    {
        if (value.Contains("device"))
        {
            deviceName = value;
        }
    }
}
```

Code Snippet 15 Example property with getter and setter

- Methods: Methods define the actions that a class can perform as shown in Code Snippet 16. Methods can take parameters that provide input data, and can return output data through parameters. Methods can also return a value directly, without using a parameter or not return any value.

```
public void PrintNameMethod(Device device){
    Console.WriteLine(device.DeviceName);
}
```

Code Snippet 16 Example method

- Constructors: Constructors are methods that are called when the object is first created as shown in Code Snippet 17. They are often used to initialize the data of an object. The default constructor takes no parameter.

```
public Device(int id,
              string name,
              bool status,
              DateTime time)
{
    deviceId = id;
```

```
    deviceName = name;
    occupied = status;
    pickupTime = time;
}
```

Code Snippet 17 Example constructor

- Nested types: Nested types are types declared within another type as shown in Code Snippet 28. Nested types are often used to describe objects that are used only by the types that contain them.

```
public class Container
{
    class Nested
    {
        Nested() { }
    }
}
```

Code Snippet 28 Example nested type

Above are common members of a class, below are rarely used members:

- Events: Events provide notifications about occurrences, such as button clicks or the successful completion of a method, to other objects. Events are defined and triggered by using delegates.
- Indexers: Indexers enable an object to be indexed in a manner similar to arrays.
- Finalizers: Finalizers are used very rarely in C#. They are methods that are called by the runtime execution engine when the object is about to be removed from memory. They are generally used to make sure that any resources which must be released are handled appropriately.

A type that is defined as a class is a reference type. At run time, when you declare a variable of a reference type, the variable contains the value null until you explicitly create an instance of the class by using the new operator, or assign it an object of a compatible type that may have been created elsewhere. When the object is created, enough memory is allocated on the managed heap for that specific object, and the variable holds only a reference to the location of said object. Types on the managed heap require overhead both when they are allocated and when they are reclaimed by the automatic memory management functionality of the CLR, which is known as garbage collection. The CLR stands for Common Language Runtime, which is the virtual machine component of Microsoft .NET Framework. It manages the execution of .NET programs.

Chapter 4 Imported Library Introduction

The default .NET framework does not always fulfill the development needs, therefore we import third party software packages. In our project, there is one essential requirement using the third party software packages. It is converting between data and JSON (JavaScript Object Notation) files. In this chapter, we introduce the JSON and its serialization and deserialization, and the reasons for using Newtonsoft.Json packages.

4.1 JSON (JavaScript Object Notation)

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write, for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java,

JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language [6]. JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures. In JSON, they take on these forms. An object is an unordered set of pairs of name and value. It begins with a left brace and ends with a right brace. Each name is wrapped by quotation marks, followed by a colon, and each pair is separated by comma. A value can be a string, number, array, boolean or null. Since json supports nested structure, it can also be an object. An array is an ordered collection of same type values, and it begins with a left bracket and ends with a right bracket. An example JSON file is shown in Code Snippet 19.

```
{
    "string": "ABC",
    "number": 123,
    "stringArray": ["DEF", "GHI", "JKL"],
    "numberArray": [456, 789, 1011],
    "boolean": true,
    "object": {
                    "number": 1213,
                    "string": "MNO"
                }
}
```

Code Snippet 19 Example JSON

4.2 C# JSON serializers

Microsoft released their new namespace System.Text.Json with .NET Core 3.0 in 2019. Before 2019, there are no official JSON serializers for .NET framework. The Arbutus Holter Windows Client started developing before 2019. At that time, we have to rely on third party JSON serializers. We eventually chose Newtonsoft.Json, also known as Json.NET. Json.NET is a widely used JSON serializer, and it is the most popular library on NuGet. NuGet is the package manager for .NET. The NuGet client tools provide the ability to produce and consume packages [17]. There are two reasons why we chose Json.NET. First, its performance is relatively higher than its competitor. As shown in Fig. 36 , Json.Net is 50% faster than DataContractJsonSerializer, and 250% faster than JavaScriptSerializer. Second, Json.NET is open source under the MIT license and is free for commercial use.

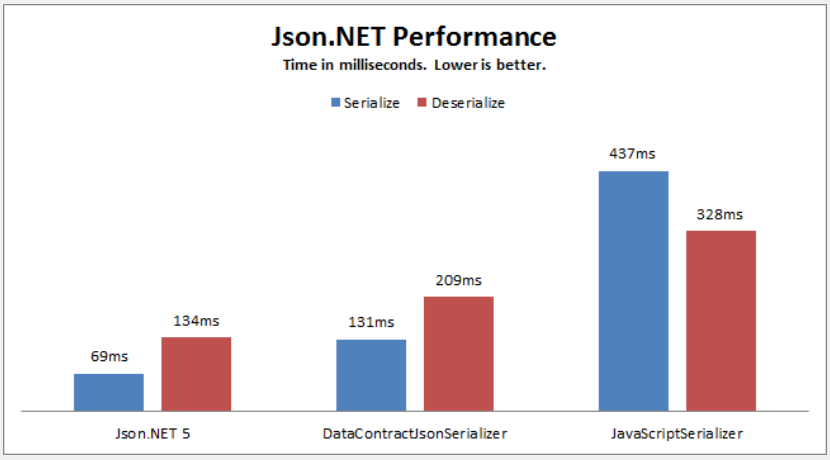


Fig. 36 Performance comparison[7]

However, while Microsoft released their new namespace System.Text.Json with .NET Core 3.0 in 2019, this brings new problems to us. Is our project supposed to migrate to System.Text.Json? Are there any advantages around performance or ease of use from

using the newly added library from Microsoft? Or should we stick with the NewtonSoft library?

4.2.1 Comparison between Json.NET and System.Text.Json

Before considering migrating, we need to inspect features of Json.NET which are not supported by System.Text.Json. If there is any feature that the project relies on, migration is impossible without significant changes. The following Json.NET features are not supported by System.Text.Json.

- Support for a broad range of types
- Polymorphic serialization
- Polymorphic deserialization
- Deserialize inferred type to object properties
- Deserialize JSON null literal to non-nullable value types
- Required setting on [JsonProperty] attribute
- DefaultContractResolver to ignore properties
- DateTimeZoneHandling, DateFormatString settings
- Callbacks
- JsonConvert.PopulateObject method
- ObjectCreationHandling global setting
- Add to collections without setters
- Snake-case property names
- ReferenceLoopHandling global setting
- Support for System.Runtime.Serialization attributes
- MissingMemberHandling global setting
- Allow property names without quotes

- Allow single quotes around string values
- Allow non-string JSON values for string properties
- TypeNameHandling.All global setting
- Support for JsonPath queries

Unfortunately, the feature DateTimeZoneHandling, DateFormatString settings are indispensable to the project. The server always uses the UTC (Coordinated Universal Time), but the client could be in any timezone. Therefore, we chose not to migrate to System.Text.Json. In the next chapter, we introduce and explain how Json.NET achieves the JSON conversion.

4.3 Serialization and deserialization

In the project, data is designed to be transmitted in JSON format. The conversion between data and JSON file is called serialization and deserialization. The serialization converts objects into JSON. The deserialization converts JSON into objects.

4.3.1 JsonConvert and JsonSerializer

Whether it is serialization or deserialization. In the Json.NET, there are two ways to accomplish the serialization or deserialization. They are the JsonConvert and the JsonSerializer as shown in Code Snippet 20 and 21. Both of them accomplish serialization and deserialization by mapping the object property names to the json property name and copies the value. For simple scenarios where we want to convert to or from a JSON string, the SerializeObject() and DeserializedObject() methods on the JsonConvert provide an easy wrapper over JsonSerializer.

```
string json = JsonConvert.SerializeObject(object, new
JsonSerializerSettings{...});
```

```
object = JsonConvert.DeserializeObject<class>(json);
```

Code Snippet 20 JsonConvert serialization and deserialization

For more control over how an object is serialized, the JsonSerializer can be used directly. It is able to read and write json text directly to a stream object such as the StreamWriter and the StreamReader. Therefore, it can be used to convert to or from a file other than just string.

```
using (StreamWriter stream = new StreamWriter(path))
{
    JsonSerializer serializer = new JsonSerializer();
    serializer.Serialize(stream, object);
}
using (StreamReader stream = new StreamReader(path))
{
    object = (class)serializer.Deserialize(stream, typeof(class));
}
```

Code Snippet 21 JsonSerializer serialization and deserialization

4.3.2 JsonSerializerSettings

During the serialization and deserialization, we occasionally customize how they are performed. The JsonSerializer has a number of properties on it to customize how it serializes json. These can also be used with the methods on the JsonConvert via the JsonSerializerSettings overloads. There are twenty-nine properties in total, and the common customization includes but is not limited to the dateTimezoneHandling, the missingMemberHandling, the nullValueHandling, and the defaultValueHandling.

The dateTimezoneHandling property gets or sets how a DateTime object's timezone is handled during serialization and deserialization. It has four members, which are the Local, the Utc, the Unspecified, and the RoundTripKind. The default value is the

RoundTripKind. The Local treats it as local time, if a DateTime object represents a Coordinated Universal Time (UTC), it is converted to the local time. The Utc is exactly opposite to the Local, it always converts the time to a UTC. The Unspecified always get rid of the timezone information when converting. The RoundTripKind always preserves timezone information when converting.

The defaultValueHandling property gets or sets how missing members (e.g. JSON contains a property that isn't a member on the object) are handled during deserialization. It has two members, which are Ignore and Error. The default value is Ignore. The Ignore ignores a missing member and does not attempt to deserialize it. The Error throws a JsonSerializationException when a missing member is encountered during deserialization.

The nullValueHandling property gets or sets how null values are handled during serialization and deserialization. It has two members, which areInclude and Ignore. The default value is Include. When serializing and deserializing objects, the Include includes null values; the Ignore ignores null values.

The defaultValueHandling property gets or sets how default values are handled during serialization and deserialization. It has four members, which are Include, Ignore, Populate, and IgnoreAndPopulate The default value is Include. The Include includes members where the member value is the same as the member's default value when serializing objects. The Ignore ignores members where the member value is the same as the member's default value when serializing objects so that it is not written to JSON. This option ignores all default values (e.g. null for objects and nullable types; 0 for integers,

decimals and floating point numbers; and false for booleans). The default value ignored can be changed by placing the DefaultValueAttribute on the property. The Populate option sets members which are not written in the Json to its default value when deserializing. The IgnoreAndPopulate option ignores members where the member value is the same as the member's default value when serializing objects and sets members which are not written in the Json to their default value when deserializing.

Chapter 5 Client Testing

Software testing is important to any software development ranging from small mobile applications to complex desktop software. Testing makes sure our software is working as we intend. In this chapter, we illustrate test cases which are designed for the client. Test cases are classified into seven categories in terms of testing functions.

5.1 Login category

Test case description	**Test case procedure**	**Expected output**
Login when user input correct email and password	1. Open the client and enter the LoginForm 2. Input correct email and password 3. Click the login button	It jumps to the AppointmentForm automatically
Checking when user inputs incorrect password	1. Open the client and enter the LoginForm 2. Input incorrect or unreal email or password 3. Click the login button	Login failed notification is displayed

Checking when user does not input all required information	1. Open the client and enter the LoginForm 2. Leave the textboxes empty or partially empty 3. Click the login button	Login failed notification is displayed

5.2 Register category

Test case description	**Test case procedure**	**Expected output**
Register a new nurse account when user inputs sufficient information	1. Open the client and enter the RegisterForm 2. Input correct information in all textboxes 3. Click the register button	It jumps to the verification panel automatically
Checking the email address that has registered	1. Open the client and enter the RegisterForm 2. Input email address that has registered and all other information 3. Click the register button	There will be a notification that notifies the email address has existed. The email textbox is cleared and the rest remain the same.
User does not input all required information	1. Open the client and enter the RegisterForm 2. Leave textboxes empty or partially empty 3. Click the register button	There will be a notification that notifies user please enter all required information
User inputs correct verification code	1. Complete registration and enter the	Register successfully notification shows up and jumps to the LoginForm

	verification panel 2. Input correct verification code 3. Click the verify button	
User inputs incorrect verification code or does not enter code	1. Complete registration and enter the verification panel 2. Input incorrect verification code or leave it empty 3. Click the verify button	Wrong verification code notification is displayed

5.3 Reset password category

Test case description	**Test case procedure**	**Expected output**
Reset user password when user type a registered email address	1. Open the client and enter the ForgetPasswordForm 2. Type a registered email 3. Click the submit button	The page automatically jumps to the verification panel
User types an non-registered email address	1. Open the client and enter the ForgetPasswordForm 2. Type an non registered email address 3. Click the submit button	The non registered email address notification will show up
User types a correct verification code in the reset password verification code interface	1. Open the client and enter the ResetPasswordForm 2. Type the correct	The page will automatically jumps to the ResetPasswordForm

	verification code 3. Click the confirm button	
User types a wrong verification code in the reset password verification code interface	1. Open the client and enter the ResetPasswordForm 2. Type the wrong verification code 3. Click the confirm button"	The wrong verification code notification will show up
The new password and confirm password does match	1. Open the client and enter the ResetPasswordForm 2. Type the password in the new password textbox and type the same password in the confirm password textbox 3. Click the confirm button"	The password reset successfully notification will show up, and automatically jumps to the LoginForm
The new password and confirm password does not match	1. Open the client and enter the ResetPasswordForm 2. Type the password in the new password textbox and type a different password in the confirm password textbox 3. Click the confirm button"	The password does not match notification will show up

5.4 Search patient category

Test case description	**Test case procedure**	**Expected output**
Search patient with Last Name, First Name, Birthdate, and PHN	1. In the search patient section of the AppointmentForm	The patient that satisfied the search conditions will shows up in the patient list

	2. Input the patient Last Name, First Name, Birthdate, PHN 3. Click the search button	section
Search patient with any combination of Last Name, First Name, Birthdate, and PHN	1. In the search patient section of the AppointmentForm 2. Input any combination of Last Name, First Name, Birthdate, and PHN 3. Click the search button	Same as above
Search patient with one search term from Last Name, First Name, Birthdate, and PHN	1. In the search patient section of the AppointmentForm 2. Input only one search term from Last Name, First Name, Birthdate, and PHN 3. Click the search button	Same as above
User does not input any search terms	1. In the search patient section of the AppointmentForm 2. Leave all input boxes empty 3. Click the search button"	Please type a search term notification will show up
Search patient with wrong birthdate format	1. In the search patient section of the AppointmentForm 2. Input birthdate in wrong format 3. Click the search button	Please type date in correct format notification will shop up

5.5 Create and edit patient information category

Test case description	Test case procedure	Expected output
User views a patient detail information	1. In the patients list section of the AppointmentForm 2. Click the target patient in the patient list	The patient information will show up in the patient details section, the create button is diabled, and save button is enabled
User edits a patient detail information	1. In the patients list section of the AppointmentForm 2. Click the target patient in the patient list 3. Edit patient information 4. Click the save button	The changes to the patient information has saved notification will show up
User edits a patient detail information and leaves some or all required textboxes empty	1. In the patients list section of the AppointmentForm 2. Click the target patient in the patient list 3. Edit patient information but leave some or all required textboxes empty. 4. Click the save button	Please input all the patient information notification will show up
User creates a patient information	1. Click the create button in the search patient section 2. Enter patient information 3. Click the save button	The CreatePatientForm will show up. After user clicked the save button, the changes to the patient information has saved notification will show up
User creates a patient information and leaves some or all required textboxes empty	1. Click the create button in the search patient section 2. Enter patient information but	The CreatePatientForm will show up. After user clicked the save button, please input all the patient information notification

	leave some or all required textboxes empty. 3. Click the save button	will show up

5.6 Manage appointment category

Test case description	**Test case procedure**	**Expected output**
User creates a new appointment without a patient	1. Click the add appointment button without selecting a patient	Please select a patient notification shows up
User creates a new appointment for a patient	1. Select a patient and click the add appointment button 2. Select correct appointment details information 3. Click the save button	The new appointment has created information will show up, and the new appointment record will shows in the AppointmentForm
User creates a new appointment for a patient, and leave some field empty	1. Select a patient and click the add appointment button 2. Leave some filed unselected 3. Click the save button	Please select device or device location notification shows up
User creates a new appointment for a patient, and input incorrect dates	1. Select a patient and click the add appointment button 2. Select overlapped date and time 3. Click the save button	Date and time is overlapped notification shows up
User edits an appointment	1. Select an appointment from appointment list or calendar 2. Edit appointment details information	The changes has saved notification shows up

	3. Click the save button	
User starts a ECG test	1. Select an appointment from appointment list or calendar 2. Click the start button	The page automatically jumps to the TestMonitoringForm
User continue a in-progress ECG test	1. Select an appointment from appointment list or calendar 2. Click the continue button Or 1. Select an in-progress ECG test from in-progress ECG test list	The page automatically jumps to the TestMonitoringForm

5.7 Manage ECG test category

Test case description	**Test case procedure**	**Expected output**
Hookup the ECG test	1. Start a ECG test 2. Click the hookup button	The hookup button is disabled. The record button is enabled. The duration starts ticking
Switch the ECG test status	1. Continue a in-progress ECG test 2. Click the hookup button or record button to switch	If the user clicks the record button, the record button is disabled and the hookup button is enabled. The indicator starts flashing. If the user clicks the hookup button, vice versa
Terminate the ECG test	1. Continue a in-progress ECG test 2. Click the terminate button	The notification that asks the user to confirm terminating the ECG test shows up. If the user confirms, it jumps to the

		AppointmentForm. Otherwise, it remains
Display ECG animation	1. Start a ECG test 2. Hookup or record the ECG test 3. Click the display button	If the ECG test is in hookup, the animation will play in 5 seconds. If the ECG test is in recording, the animation will play in 1 minute. The display button changes to the pause button
Stop displaying ECG animation	1. Display ECG animation first 2. Click the pause button	The ECG animation stops playing, and the pause button changes to the display button
ECG test has finished	1. Start a ECG test 2. Wait for it to finish	The ECG test status is updated to terminated automatically
Client lost internet connection during displaying ECG animation	1. Start a ECG test 2. Disconnect the internet 3. Click the display button	After three retires, the internet disconnected notification shows up

Chapter 6 Conclusion and Future Work

6.1 Conclusion

In this report, we have introduced the heart disease and ECG background, analyzed the pros and cons of traditional ECG tests, and presented the Telemetry ECG Monitoring System. We have described and explained the internal architecture and user interface of the Arbutus Holter Windows Client. The client is a WinForms project which is part of Microsoft .NET framework. The WinForms provide a platform to write rich client

applications for the Windows operating system. We described how a form is created and used in the client. We illustrated how to define and add controls to forms, and how to raise and handle events, and how to initialize and operate forms. When work is complicated, the underlayers are going to do the subdivision work. The underlayers are classified into two parts, which are resources and requests. The resources part is responsible for wrapping and parsing data before and after the requests. The requests part is responsible for sending the HTTP request to the server and receiving feedback from the server. We have briefly analyzed the functional requirement of the client, and meticulously demonstrated the user interface of the client. The user interface can be divided into two sections based on authentication requirements. The public section is open for every user, and the nurse section is exclusive to registered users. We have introduced and compared different JSON software packages. Including the official package System.Text.Json and third party software packages. We have explained why we chose the Json.NET instead of others. We have illustrated how Json.NET converts data and JSON files.

6.2 Future work

In addition to the work we have done so far, there are some other features that can be improved in the future.

6.2.1 Administrator interface

As mentioned in Chapter 2, the client can be divided into two sections in terms of authentication requirements. For the section which requires authentication can be further subdivided into two sections. We should provide an exclusive interface for administrators since there are some functions that require higher authorization. For

example, the device management that lets the clinic add or delist devices. It is inappropriate to allow nurses to manage devices. For the administrator interface, the client may differentiate them in the login process. If the server distinguishes between nurse account and administrator account, the client could guide users to different forms after login successfully.

6.2.2 HTTPS protocol

Unitil now, the protocol that the server uses is still HTTP, which is considered not safe enough today. In September 8, 2016, Google announced that “Beginning in January 2017 (Chrome 56), we’ll mark HTTP pages that collect passwords or credit cards as non-secure, as part of a long-term plan to mark all HTTP sites as non-secure. [18]”. Therefore, even though the server is not a webpage, it is recommended to change HTTP to HTTPS protocol. For our project, the general process for switching is the following three steps.

1. Purchase an SSL (Secure Sockets Layer) certificate.
2. Configure hosting with SSL Certificate.
3. Change all internal url schema to HTTPS.

Reference

[1] Sells, Chris (September 6, 2003). Windows Forms Programming in C# (1st ed.). Addison-Wesley Professional. p. Xxxviiii.

[2] Design and Implementation Guidelines for Web Clients by Microsoft Pattern and Practices. Microsoft. November 2003.

[3] Async and Await In C#, Retrieved Nov 10, 2021, from https://www.c-sharpcorner.com/article/async-and-await-in-c-sharp/

[4] Asynchronous programming - C#, Retrieved Nov 10, 2021, from https://docs.microsoft.com/en-us/dotnet/csharp/async

[5] The Task Asynchronous Programming (TAP) model with async and await (C#), Retrieved Nov 10, 2021, from https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/task-asynchronous-programming-model

[6] Introducing JSON, Retrieved Nov 10, 2021, from https://www.json.org/json-en.html

[7] Performance Comparison, Retrieved Nov 10, 2021, from Json.NET - Newtonsoft

[8] Handling and Raising Events, Retrieved Nov 10, 2021, from https://docs.microsoft.com/en-us/dotnet/standard/events/

[9] Hypertext Transfer Protocol – HTTP/1.1, Retrieved Nov 10, 2021, from https://www.rfc-editor.org/info/rfc2616

[10] "GET". RFC 2616. p. 53. sec. 9.3. Retrieved Nov 10, 2021, from https://datatracker.ietf.org/doc/html/rfc2616

[11] "POST". RFC 2616. p. 54. sec. 9.5. Retrieved Nov 10, 2021, from https://datatracker.ietf.org/doc/html/rfc2616

[12] "PUT". RFC 2616. p. 55. sec. 9.6. Retrieved Nov 10, 2021, from https://datatracker.ietf.org/doc/html/rfc2616

[13] "DELETE". RFC 2616. p. 56. sec. 9.7. Retrieved Nov 10, 2021, from https://datatracker.ietf.org/doc/html/rfc2616

[14] The components of a URL, Retrieved Nov 10, 2021, from https://www.ibm.com/docs/en/cics-ts/5.2?topic=concepts-components-url

[15] Heart Disease in Canada, Retrieved Nov 10, 2021, from https://www.canada.ca/en/public-health/services/publications/diseases-conditions/heart-disease-canada.html

[16] Electrocardiograms (ECG), Retrieved Nov 10, 2021, from https://www.lifelabs.com/tests-services/heart-monitoring/electrocardiograms-ecg/

[17] NuGet Gallery, Retrieved Nov 10, 2021, from https://www.nuget.org/

[18] Moving towards a more secure web, Retrieved Nov 10, 2021, from https://security.googleblog.com/2016/09/moving-towards-more-secure-web.html