Basic macros Compatibility with language.def Hooks Defining babelensure Macros for setting language files up Shorhands Language attributes Macros for saving definitions Short tags Hyphens Multiencoding strings Macros related to glyphs Bidi layout Input engine specific macros

Redefinitions for bidi layout Bidi footnotes Creating languages and reading ini files Hyphen rules for 'canadian' set to "l@english ("language0). ReportedHyphen rules for 'australian' set to "l@ukenglish ("language73). ReportedHyphen rules for 'newzealand' set to "l@ukenglish ("language73). Reported

english

VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

– – – – – – – – – –

GRADUATION THESIS

E-HEALTHCARE SYSTEM BASED ON FACE RECOGNITION AND INTERNET OF THINGS SERVICES

MAJOR: COMPUTER ENGINEERING

COUNCIL: **COMPUTER ENGINEERING 3**
INSTRUCTOR: **Dr. LE TRONG NHAN**
REVIEWER: **Assoc. Prof. Dr. TRAN NGOC THINH**

Student 1: Truong Le Vinh Khoa 1752298
Student 2: Le Nguyen An Khuong 1752305

Ho Chi Minh City, September 2021

COMMITMENT

We pledge that this project is based on our supervisors' ideas and knowledge. All studies and data have not been published. The references, numbers and statistics are reliable and honest. The group completed the thesis requirements set by faculty of computer science and engineering - department of Computer Engineering.

Sincerely,

Truong Le Vinh Khoa
Le Nguyen An Khuong

ACKNOWLEDGEMENT

First and foremost, We would express our most profound appreciation to our thesis supervisors, Ph.D. Le Trong Nhan. He has been there, providing his heartfelt support and guidance at all times. He has given us invaluable guidance, inspiration, and suggestions in our quests for knowledge during our university time. Without his assistance and dedicated involvement in every step throughout the process, this thesis would have never been accomplished.

We sincerely thank the teachers who occupy the Faculty of Computer Science and Engineering in particular and the Ho Chi Minh City University of Technology in general, who have constantly been imparting knowledge in the past four years. Their support, encouragement, and credible ideas have been great contributors to the completion of the thesis.

Last but not least, It would be inappropriate if we omit to thank our friends and family. Our late parents' unconditional love and blessings, the care of friends and acquaintances who never let things get dull have all made a tremendous contribution in helping us reach this stage in our life. We thank them for putting up with us in difficult moments where we felt stumped and for goading us on to reach for our passions.

Finally, we would like to wish you good health and success in your noble life.

ABSTRACT

To support and optimize a check-in phase of the examination process, and provides a safer environment for patients and doctors/nurses, in this graduation thesis project, we propose an **E-Healthcare System based on Face Recognition and Internet of Things Services**.

This system requires minimal human interaction between the patients and doctors (or nurses) and removes the patient's medical examination steps to accelerate it. In addition, we also build a phone app so that patients can access and track their previous visits. The project has been modified on friendly use, safety, and considerable contexts, although we came across some issues and later proposed temporary solutions for each case.

This thesis contains five chapters to explain our work obviously, and The last chapter is dedicated to setting up the system. Our final result allows the patients to check in the examination at hospitals through face recognition. The information of this examination includes patients' biological parameters and selected departments. If patients do not know which department, our assistant will help them choose an appropriate department according to their symptoms. Additionally, we create two applications; one is for the doctor to examine patients, and another is to help the patients to track up their health and previous visits.

Keyword for our work: Jetson Nano, Healthcare Sensors, Face Recognition, IoT Services.

Table of contents

List of figures

Tables

CHAPTER 1

INTRODUCTION

1.1 Thesis Introduction

According to the best of our knowledge, one bottle neck process at a hospital is the the checking process, where a patient requires a medical check-up. The check-in process in this scenarios, is to designate or identify the identity of the patient and/or information declarations. A traditional method for checking-in in Vietnam hospital follow a common sequence. When the patient come to the hospital, the nurse will call the roll one by one to register their examinations for the doctors. Then the patient will take some measurements additionally and wait for their rendez-vous. Thus, this methodology hurts the overall efficiency and data processing .The following section will describe the details of the steps in the current hospital examination and treatment process. From there, the disadvantages are figured out and improved by our proposed system.

1. **Checking-In**: is a well-know step that patients are supposed to do firstly in the traditional process at the hospital. In this step, when the patients arrive at the hospital, they will meet the nurse at the reception and tell them about their condition or request a declaration form. The nurse will then give them a form to fill in their personal information, current condition, how they feel, etc. When the form is submitted, based on their knowledge and experience, the nurses will assign a department that can help with the problem the patient is currently caught. The patient is now put into a queue, and when it is their turn to come

up, there will be an announcement to notify them (can be whether by computer or by the nurse at the reception)

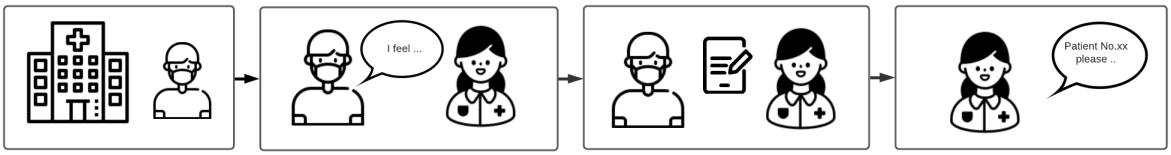

Figure 1.1: Checking-in

2. **Examination**: Once the patient gets into the examination room with the doctor, they will be asked again about the symptoms. In some circumstances, the doctor will ask the patient to perform some biometrics measurements to support their conclusion. Finally, the examination information will be recorded on the hospital database for future review if necessary for the doctor.

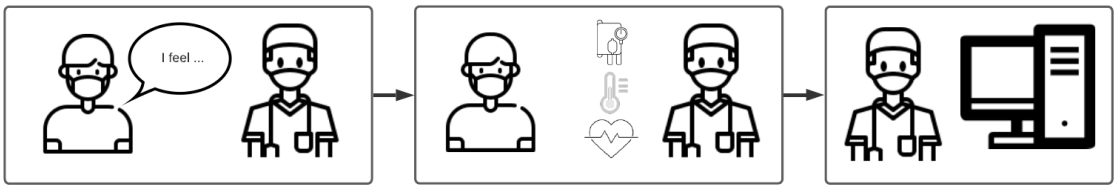

Figure 1.2: Examination

3. **Post Examination** Most of the hospitals have the patients' records on their database for tracking and future review, for the patients' part, some of the hospitals will provide the patient the examination notebook that stores the information of their past examination so they can look back and track their health condition

The traditional flow had been there for decades. However, as the regular workflows have too many steps and require many human-interaction actions (when talking to the nurse, doctor perform biometrics measurement, etc.), it may cause obstruction and make this workflow inefficient and highly unfeasible. Furthermore, in the context of an ever-evolving society and more and more new technologies being born, the demands for convenience in people's

lives are also increasingly common. Therefore, an examination process that lessens human interaction is in great need at the moment. In this thesis, we propose a system that aims to improve the shortcomings of the current process and supports the examination process at medical facilities. This system focuses on optimizing the medical examination process by combining the steps of check-in, choosing the department, and measuring the patient's biologic parameters into one with the expectation that the system will bring benefits as described below:

- *Support the medical examination* When examining at a medical institution, the patient performs the check-in process, selects the department, and takes several biologic measurements in one step. It provides convenience for doctors and helps patients have a better experience during the medical examination at the hospital. In addition, the system also helps patients recommend an appropriate department based on their described symptoms if they do not know their intent department.

- *Lessen human-to-human contact* Since the patient has complete all the check-in processes at the device, it will help reduce contact between medical staff and patients during hospital visits.

  Monitor examination history for patients For patients using smartphones, the system will provide examination information during the medical examination, such as their STT and Room code, and notify when it is their turn. At the same time, patients can track the history and details of medical examinations through the system's mobile application.

- *Mitigate the work burden for the medical team* In the current medical examination process, medical staff must receive each patient, arrange them to the appropriate department and collect their biological parameters. With this system, we can reduce the burden on the medical staff with automated processes. Instead of wasting much staff, we can arrange a specific staff to support the patient in manipulating and operating the system.

- Finally, with hospitals in the same system, patient information is shared internally between them. It allows patients to be examined and treated at different hospitals while the doctor still knows the treatment history.

Moreover, with a single registration, patients can be identified at all hospitals in the system. It provides convenience for patients to examine in different hospitals and makes it easier for them to track their visit history.

1.1.1 Proposal System

To build this system, we need to combine many different technologies, ranging from hardware to software. However, the three most important technologies needed to build this system including Face Recognition, Internet of Things, Speech Recognition, and Biologic Sensors.

Internet of Things In this system, IoT is both the core and plays an essential role in implementing the system. It supports device management, transmits patient examination information to the physician, and communicates between servers and devices. It also allows the system to be scaled more efficiently, adding and setting up devices quickly and straightforwardly.

Face Recognition: For the face recognition, in the hospital environment as well as the current covid pandemic. The most urgent and necessary thing is to mitigate contact between people in medical examination and treatment. Therefore, we apply facial recognition for patient check-in. It is a promising technology for the future. Its non-compulsive and non-contact characteristics have attracted the attention of researchers.

Although biometric technologies are worth mentioning, such as iris recognition and fingerprint identification, collecting the relevant data are complex and time-consuming processes [**?**]. Therefore, it is rational to use FRT for our check-in modules as it has much better efficiency and accuracy. Additionally, in recent years, with the rapid development of new face recognition algorithms such as Hog-SVM [**?**], Max-Margin Object Detection [**?**], CenterFace [**?**], and other algorithms to train a face recognition model with extremely high accuracy. Face recognition has much better efficiency and accuracy than the previously proposed ways of identification. Therefore, Face Recognition has great potential for applying to the healthcare system. [**?**]

Speech Recognition: In this system, Speech Recognition is used to build up an assistant. This assistant assists patients with the unknown department by recommending the appropriate department based on the symptoms they describe.

Biologic Sensors: Another part of the system is the biometric measurements. It includes collecting the patient's heart pulse, oxygen saturation, temperature, height, weight, and BMI value in the check-in phase. These parameters are then included in the examination form and transferred to the doctor, which helping them to have a general condition of patients. For each parameter, we propose all the possible sensors, evaluate the trade-off between cost and performance, and then select the one that adapts to the system's conditions.

1.2 Scope and Objectives

In the final phase of this Graduation Thesis, we implement the full system that includes three phases:

- Check-In Phase
- Take Examination Phase
- Post Examination Phase

To help us understand more about the workflow of our system, we will detail each phase with illustrations so we can better understand our process.

1.2.1 Check-In Phase

When the patient arrives at the hospital, the system will retrieve their facial features through facial recognition to identify patients to determine who they are, whether new or old.

Next, the biologic parameters of the patients, such as height, weight, temperature, blood oxygen concentration, and heart rate, will also be captured

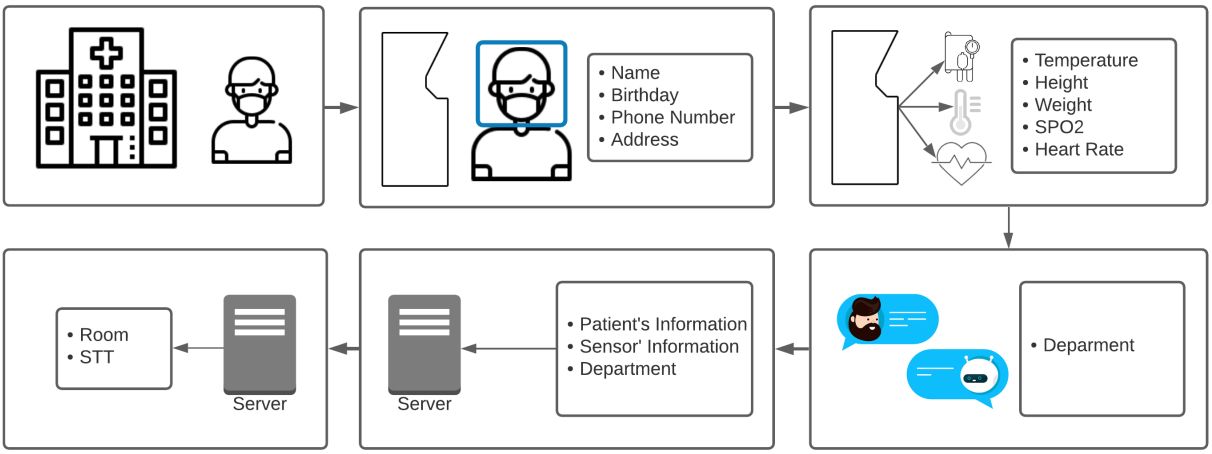

Figure 1.3: Phase One of the Workflow

by the system through the sensors of the device. At the same time, the system allows patients to choose which department they will examine or propose an appropriate department based on the symptoms provided by them if the patient does not know which department to visit.

1.2.2 Take Examination Phase

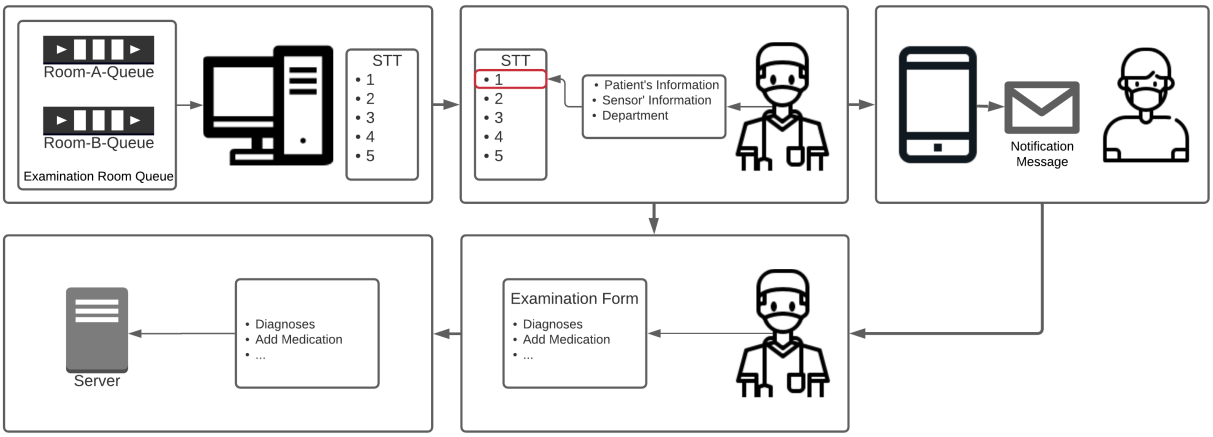

Figure 1.4: Phase Two of the Workflow

After the examination's information is sent to the system, doctors will receive information about the patient, symptoms, and biologic parameters at each respective examination room. Simultaneously, at the mobile application of each old patient, their STT and the examination room will be updated on the application screen.

When it is the patient's turn, we will notify them via mobile application

notification and mobile message (if they do not have a smartphone). Then the doctors will directly examine them, record symptoms, drugs and send all the examination information to the system's database.

1.2.3 Post Examination Phase

Finally, all information is stored on a database, and the patients can keep track of their condition and their visits through the phone application.

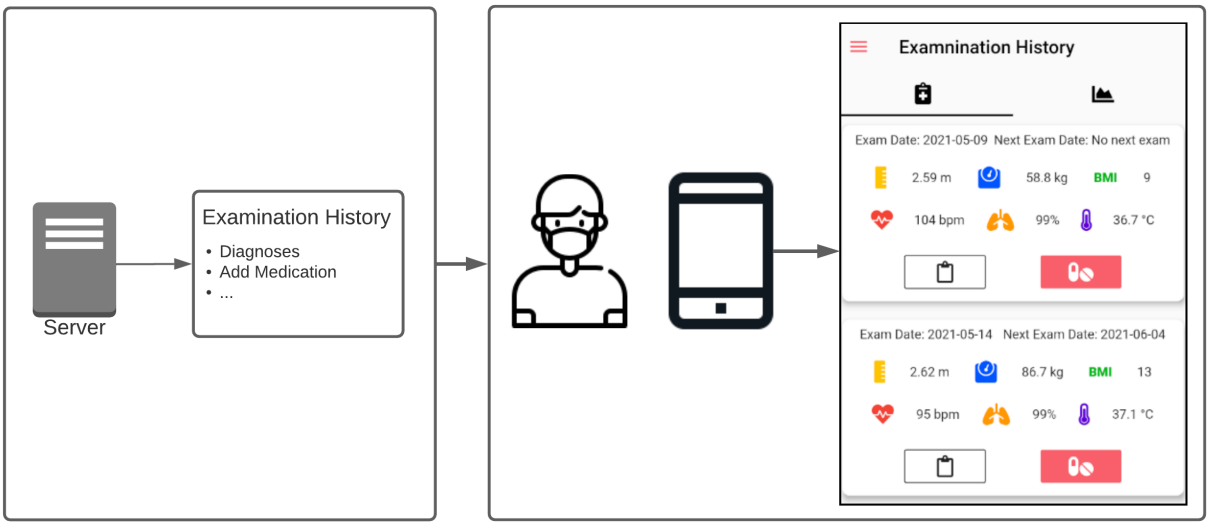

Figure 1.5: Phase Three of the Workflow

1.3 Advantages and Challenges

1.3.1 Advantages

- As it is an IoT system, we can scale up the system in many aspects:
  - Quickly build multiple servers to reduce the number of requests through load balancing
  - Connect multiple hospitals in the same system together. For example, patients can be checked in at hospitals in the system without having to register multiple times.
- Speed up patient check-in and sampling times.

- Assist patients in tracking their visits and notify them during their hospital visits through a mobile application.
- Reduce the transmission of infectious diseases in the hospital environment.
- Improve the shortage of human resources for medical check-in. For example, the hospital initially needed five nurses to record patient information, inquire about their condition, and assign them to the appropriate department. Next, in each room, there will be nurses on duty to measure biological parameters. However, we only need five machines with this system and from 1 to 2 nurses to support the patient during the procedure.

1.3.2 Challenges

Besides, in addition to the above advantages, we also face some difficulties as follows:

- As the system always scales up more and more new patients, we need to propose an appropriate solution for accurate identification.
- Since it is an IoT system, network connectivity is always essential for each check-in device. Therefore, we have to guarantee that the system is always connected to the Internet.
- For elderly and first-time patients, it will be difficult to access such automated systems.
- Many natural factors in the hospital environment can interfere with sensor sampling.

To overcome the above difficulties, in each section, we will analyze and propose appropriate solutions. At the conclusion, we will summarize the whole thing and suggest future work for unresolved difficulties.

1.4 Thesis Structure

The remainder of this report is organized following. Part 2 is background knowledge and methodologies. In part 3, we propose an overall system design

from hardware, software, and IoT architecture. Next, part 4 is our detailed implementation. Finally, the evaluation and summary of the whole system will be illustrated in part 5 and part 6, respectively.

CHAPTER 2

METHODOLOGIES

This chapter will present knowledge that contributes to the implementation of our system, including some popular models for detecting and identifying humans. For each domain, we do experiments on these models to compare their performance based on criteria before choosing the most appropriate one on each domain of our system. Finally, we will introduce additional concepts about Speech Recognition, Natural Language, and some tool kits that support our system in recommending department tasks.

2.1 Face Recognition

Face-Recognition is a method that identifies or verifies a person by their face through the image or video. This method plays an essential role in security systems and has several advantages over other biometric modalities such as fingerprint or eye iris recognition systems.

In our system, the face recognition module is implemented and worked as a recognition module. This module is responsible for detecting and identifying examining patients and is implemented with three steps *Face Detection* and *Face Identification*. Where *Face Identification* includes *Facial Representation Extraction* and *Face Classification* to identify patient. However, the above procedures are only suitable for identifying user processes. At the same time, we also have another module that needs to verify the patient with their given unique parameter(e.g., phone number, SSN). In this system, we choose the SSN.

For this one, instead of the final step, classifying the patient's face, we will replace it with another module that takes the patient's face along with their SSN (Social Security Number), retrieves the database, and compare whether this is the patient or not. The above two tasks are also described in the Figure. 2.2.

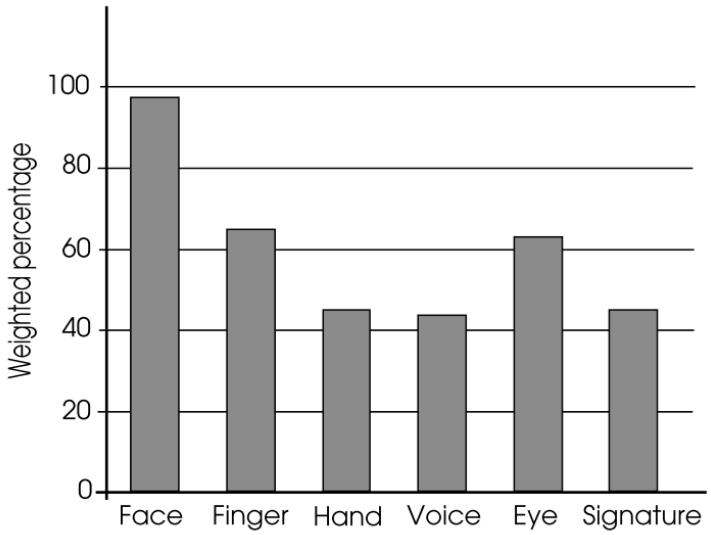

Figure 2.1: Biometric Attributes Ranking

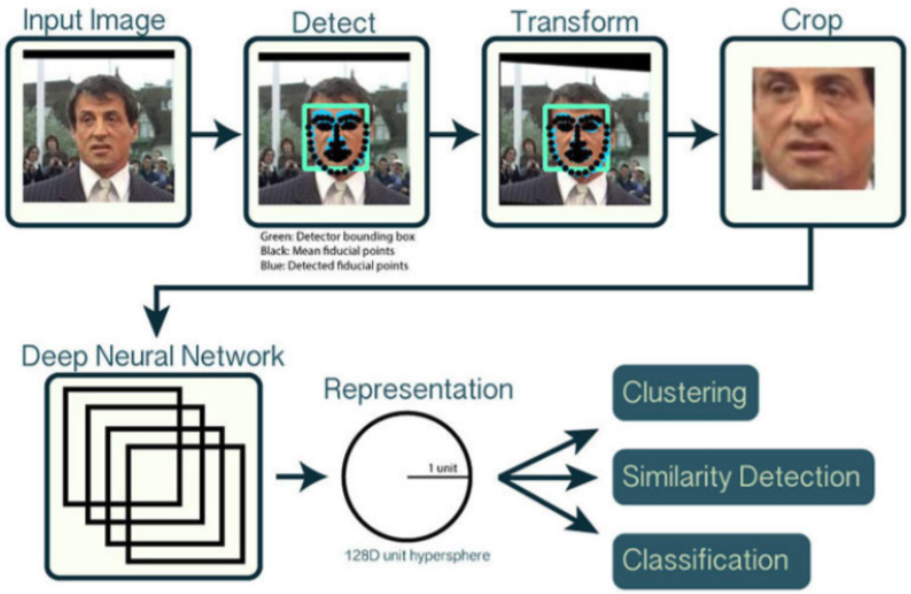

Figure 2.2: Face Recognition Workflow

2.1.1 Face Detection Methods

Currently, there are many algorithms for Face Detection, from the Machine Learning approach to the CNN approach. In the next section, we will introduce some of the algorithms that stand out and may suit our project. Besides, to be able to choose the most appropriate method for the project, we performed three tests on processing time, CPU and RAM Utilization, and the ability to detect face variety of angles.

2.1.1.1 Histogram of Oriented Gradients - Support Vector Machine (HOG - SVM)

Support Vector Machine are supervised machine learning models that divide and classify data. It is widely used for applications such as face detection, classification of images, handwriting recognition. At the same time, Histogram of Oriented Gradients or HOG is a feature descriptor that has been successfully used for object and pedestrian detection. This method simplifies the image by extracting useful information and discard non-essential information (e.g., remove constant colored background and highlighted outlines object). A HOG relies on the property of objects within an image to possess the distribution of intensity gradients or edge directions.

Figure 2.3: Left : Absolute value of x-gradient. Center : Absolute value of y-gradient. Right : Magnitude of gradient.

For example, the gradient image below is removed non-essential information (e.g., constant colored background) but highlighted outlines. In other

words, we can look at the gradient image and still easily say there is a person in the Figure. 2.3.

The descriptors are calculated over blocks of pixels with 8x8 dimensions. These descriptor values for each pixel over 8 x 8 block are quantized into nine bins. Each bin represents a directional angle of gradient and value in that bin, which is the summation of the magnitudes of all pixels with the same angle. The SVM model is trained using several HOG vectors for multiple faces.

- **Advantages**
  - Fastest method CPU
  - Works very well for frontal and slightly non-frontal faces
- **Disadvantages**
  - The major drawback is that it does not detect small faces as it is trained for a minimum face size of 80x80.
  - The bounding box often excludes part of the forehead and even part of the chin sometimes.
  - Does not work for side face and extreme non-frontal faces, like looking down or up.

2.1.1.2 Max-Margin Object Detection

In the basic training procedure, a set of positive and negative image windows are selected from training images. Then a binary classifier is trained on these windows. Lastly, the classifier is tested on images containing no targets of interest, and false alarm windows are identified and added to the training set. The classifier is then retrained and, optionally, this process is iterated.

This approach does not efficiently use the available training data since it trains on only a subset of image windows. Additionally, windows partially overlapping an object are a common source of false alarms. This training procedure makes it difficult to directly incorporate these examples into the

training set since these windows are neither fully a false alarm or a true detection. Most importantly, the accuracy of the object detection system as a whole is not optimized. Instead, the accuracy of a binary classifier on the subsampled training set is used as a proxy.

However, all these issues are addressed with this method. In particular, this method is an optimizer designed to run overall windows. It optimizes the performance of an object detection system in terms of the number of missed detections and false alarms in the final system output. Moreover, the formulation of this method leads to convex optimization, and the owners also provide an algorithm that finds the globally optimal set of parameters.

MMOD in Face Detection

This single HOG filter is scanned over the image at each level of an image pyramid to perform detection. Any windows which pass a threshold test are output after non - max suppression is performed. A ROC curve that compares this learned HOG filter against other methods is created by sweeping this threshold and can be seen in Figure. 2.4. To create the ROC curve, we followed the FDDB evaluation protocol of performing ten-fold cross-validation and combining the results in a single ROC curve using the provided. [**?**]

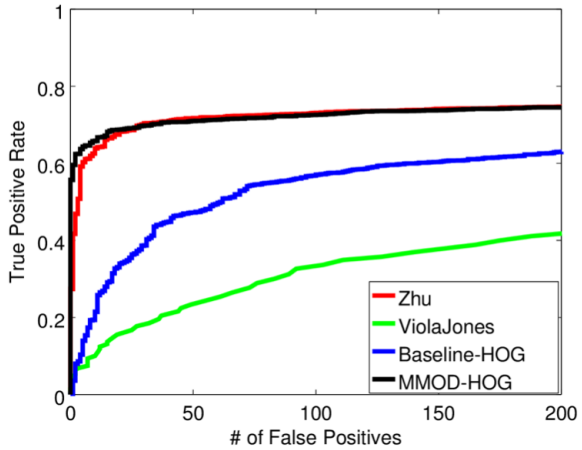

Figure 2.4: MMOD and three other techniques, including another HOG filter method learned using traditional means. The MMOD procedure results in a much more accurate HOG filter.

FDDB evaluation software. Example images with detection outputs are

also shown in Figure. 2.5. In Figure. 2.5, we see that the HOG filter learned via MMOD substantially outperforms a HOG filter learned with the typical linear SVM "hard negative mining" approach [**?**] as well as the classic Viola-Jones method [**?**]. Moreover, our single HOG filter learned via MMOD gives slightly better accuracy than the complex deformable part model of Zhu [**?**].

Figure 2.5: Example images from the FDDB dataset. The red boxes show the detections from HOG filters learned using MMOD. The HOG filters were not trained on the images shown.

- **Advantages**
  - Fast and accurate on GPU
  - Work well on side faces and every scale faces
- **Disadvantages**
  - Very slow on CPU and embedded board
  - Hard to choose best model for the project
  - FPS is 14 16 FPS in Jetson Nano

2.1.1.3 CenterFace

CenterNet

CenterNet has a simple architecture, compared with the other detectors such as Yolo, RCNN..., but can achieve the equilibrium between the processing time and accuracy. The main idea of CenterNet is to predict the heatmap of the objects' key points (center, bounding points.) (Figure. 2.6) and regress the other properties (width, height, orientation). Every pixel in the heatmap represents the probability that the center of the object is in this

position. They filter the heatmaps to find the local maxima to determine the objects' center in the postprocessing step. In this way, we can consider CenterNet as the anchor-free object detector; we do not need to pre-define anchors' sizes and ratios to predict the object. Moreover, in the postprocessing step, CenterNet does not need to perform Non-max-suppression (NMS), a time-consumption task in object detection, to eliminate overlapped predicted bounding boxes.

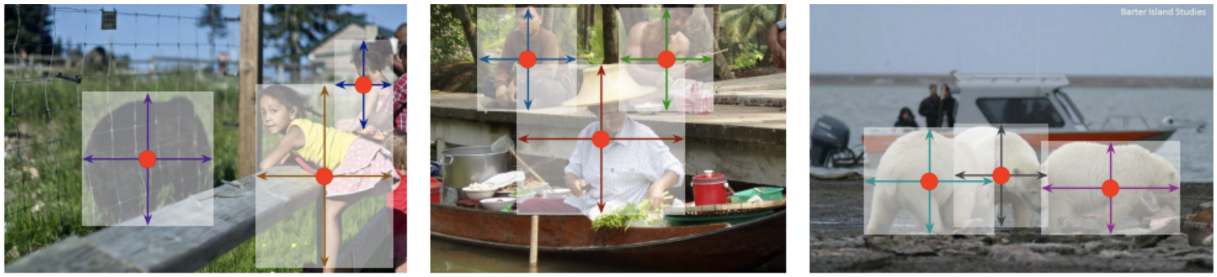

Figure 2.6: CenterNet

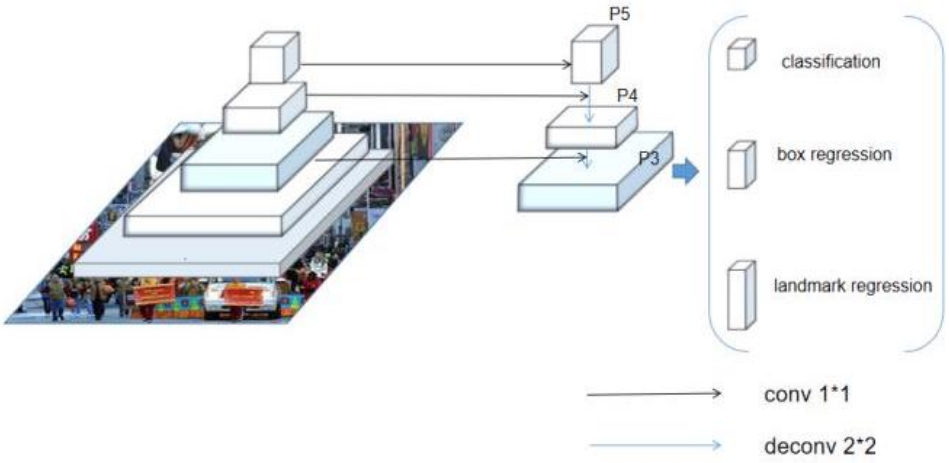

Figure 2.7: CenterNet

The focal loss is used to calculate the loss of the heatmap prediction. Focal loss is an enhancement of cross-entropy loss, used to deal with the imbalance of the points in the heatmaps (because the number of pixels that are the centers of objects is tiny compared to.the total number of pixels in the image). The formula of Focal loss is presented below.

$$L_k = \frac{-1}{N} \sum_{xyz} \begin{cases} (1 - \hat{Y}_{xyc})^{\alpha} log(\hat{Y}_{xyc}) & if Y_{xyc} = 1 \\ (1 - Y_{xyc})^{\beta} log(\hat{Y}_{xyc})^{\alpha} log(1 - \hat{Y}_{xyc}) & otherwise \end{cases}$$

Figure 2.8: Focal Loss Formula

The main drawback of CenterNet is that it does not perform well if the centers of objects are very close to each other or overlapped objects. However, in the face check-in scenario, the probability of two faces sharing the same center or overlapped faces are nearly 0; therefore, we can choose this approach for our system to achieve high accuracy as well as lightweight and fast enough for deployment on low cost embedded boards.

CenterFace

For more detail about CenterFace, it uses MobilenetV2 [**?**] as the backbone and Feature Pyramid Network (FPN) [**?**] as the neck for the subsequent detection. The detector head contains one heatmap prediction (because we only detect 1 class: face), which is the probability that the center of the face is located in this position, and two regressed branches: box regression (width, height) and landmark regression (5 landmarks). The model was trained on the training set of WIDER FACE benchmark, including 12,880 images with more than 150,000 valid faces in scale, pose, expression, occlusion, and illumination. [**?**]

The proposed method CenterFace achieves 0.935 (Easy), 0.924 (Medium) and 0.875 (Hard) for validation set, and 0.932 (Easy), 0.921 (Medium) and 0.873 (Hard) for testing set on WIDER FACE dataset. Despite the gap with state-of-the-art methods, the CenterFace method results are acceptable because it has a trade-off between accuracy and time-consumption/memory consumption.

Method	Easy	Medium	Hard
RentinaFace	0.963	0.956	0.914
DSFD	0.960	0.953	0.900
CenterFace	0.932	0.921	0.873
LFFD	0.896	0.865	0.770
MTCNN	0.851	0.820	0.607
Faceboxes	0.839	0.763	0.396

Table 2.1: Accuracy Between Center Face Other Popular Face Detection Methods

2.1.1.4 Comparison

In this section, we will experiment on the *Jetson Nano* board to compare the performance of the three Face Detection methods above. (For more detail of this board, we will introduce it in section 3.2.1.). This experience includes three parts which each part will be conducted to measure one criteria, *Processing Time*, *CPU and RAM Ultilization*. Finally, it is the ability to detect multiple face angles in various range distances.

Processing Time

Before performing the above experiment, we will re-process the image first to ensure the maximum performance of each method for each algorithm. Table 2.2 is the parameter of the image after re-processing corresponding to each method.

	Center Face	Hog - SVM	MMOD
Resolution	240x180x3	100x74x3	100x74x3
Image Format	RGB	GRAY	GRAY

Table 2.2: Re-processing Image Format of each Method

The figure below is the Processing Time of three Face Detection methods. We conduct this experience by simulating the camera program on Jetson Nano. We do it three times with three methods. Each time we let the program

detect 1500 frames, record the time before inferencing, and after inferencing the image to each model, get the time difference, and append to a list. After running it all, we have the following statistics Table 2.3:

	Center Face	**Hog - SVM**	**MMOD**
Max (ms)	15.8	14.6	33.2
Min (ms)	13.9	11.9	27.1
Mean (ms)	14.8	12.4	30.7
Std (ms)	1.1	0.3	2.2

Table 2.3: Processing Time between Face Detection Methods

As we can see in the above table, the running time of Hog-SVM is the fastest, then comes the Center Face and finally the MMOD. The above result is also understandable. Because, Hog - SVM is a method running on a Machine Learning algorithm. Whereas Center Face and MMOD run on the CNN algorithm, the difference between processing times is expected.

Nevertheless, there is such a long distance between Center Face and MMOD because MMOD is an algorithm built by Davis King in the Dlib Library. Although optimized to run on CUDA, it is not optimized and runs on TensorRT. As for CenterFace, as mentioned in section 2.2., this is an optimized model on TensorRT so that processing time will be much better than MMOD.

CPU and RAM Utilization

As we do with processing time, we document CPU and RAM Utilization when running three methods and describe in the figures below.

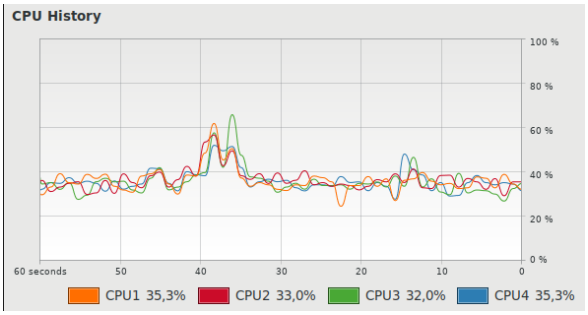

Figure 2.9: CPU Utilization Center Face

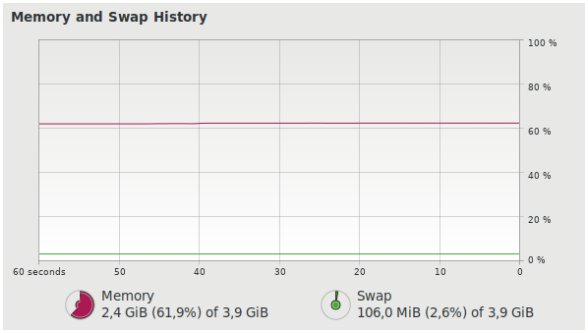

Figure 2.10: RAM Utilization Center Face

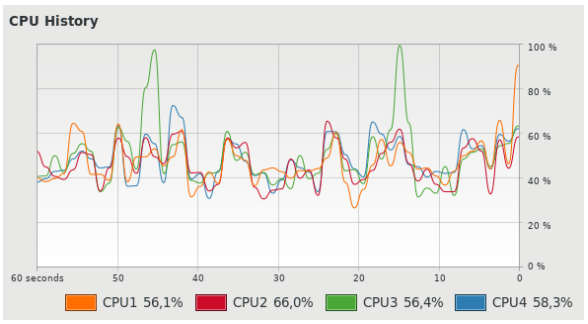

Figure 2.11: CPU Utilization HOG - SVM

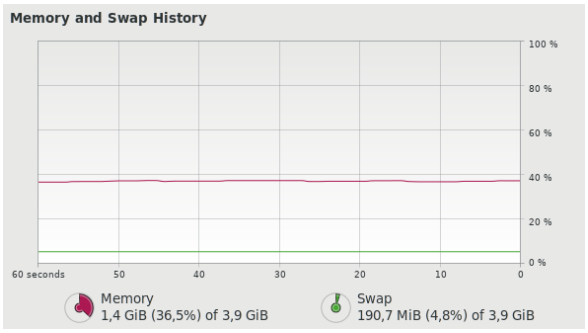

Figure 2.12: RAM Utilization HOG - SVM

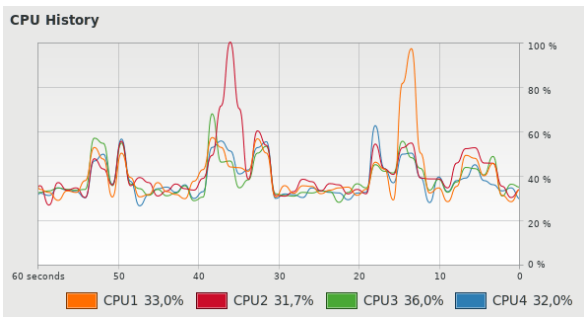

Figure 2.13: CPU Utilization MMOD

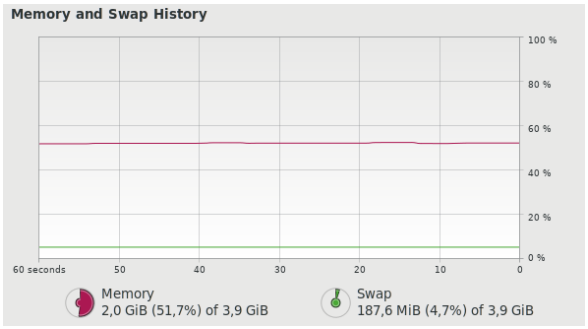

Figure 2.14: RAM Utilization MMOD

Experiments With Different Face Angles

The most important part of a Face Detection algorithm is detecting faces in many situations, such as brightness, angles, reflections, distance to the camera. Moreover, in this part, we perform face recognition experiments with five different angles: front, top, bottom, left, right, and capture the performance of each method.

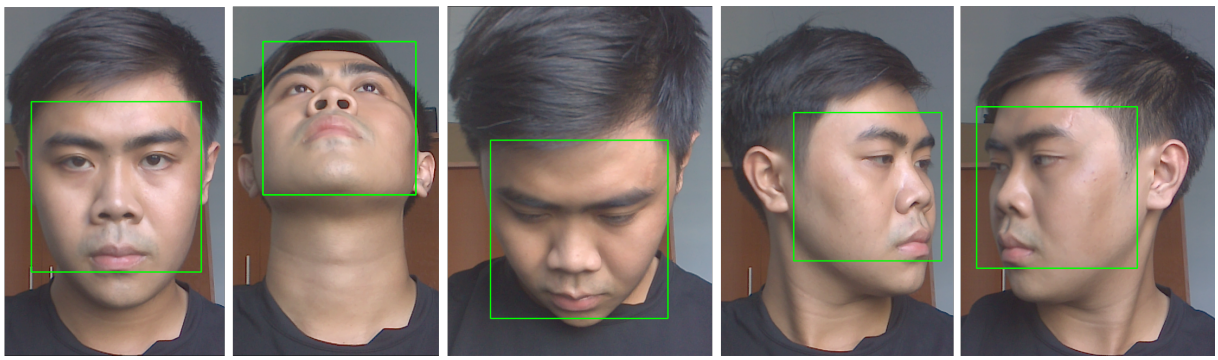

Figure 2.15: Five Poses Detected with Center Face

Figure 2.16: Five Poses Detected with Hog - SVM

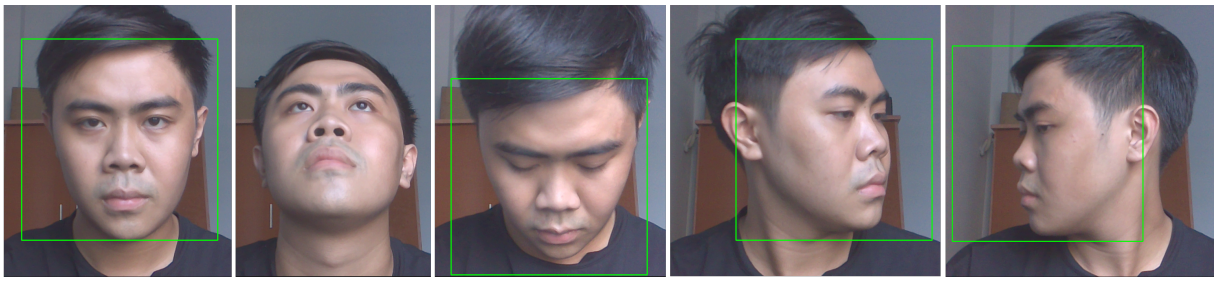

Figure 2.17: Five Poses Detected with Max Margin Object Detection

As seen in the images above, with the same distance of *0.5m* from the camera to the person and the same environmental conditions, Center Face results in all five face angles being detectable, four for MMOD (except for the upward direction). Meanwhile, only Hog can get the only front face.

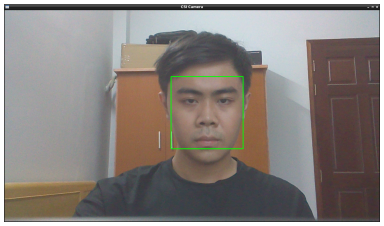

Figure 2.18: Center Face With Greater Than 1m

Figure 2.19: Hog - SVM With Greater Than 1m

Figure 2.20: MMOD With Greater Than 1m

However, when we increased the distance from the camera to the person to *1m*, only Center Face could fully detect the five face' angles stably.

Conclusion

To summarize, after the experiments on processing time, CPU and RAM Utilization, and the ability to detect five face' angles. We conclude that although Center Face consumes a lot of RAM to be able to work. However, in return, the Jetson Nano board still has the powerful ability to afford it with no difficulties. Furthermore, this method offers stability in CPU utilization, less processing time, and variety in many detection angles. From the above, we decided to choose Center Face as the Face Detection method for our project.

2.1.2 Facial Representation Extraction

Face Representation Extraction aims to extract the features from an image, which is a crucial step of face recognition. The bounding box containing humans' faces is embedded into a vector representation of features, which is very helpful in the face recognition step and further processing. For example, another close vector (by some measure from features vector) may be the same person, whereas another vector far (by some measure from features vector) may be different. The classifier model that we want to develop will take a face embedding as input and predict the identity of the face.

2.1.2.1 Dlib's open-source library

Dlib machine learning is a cross-platform open-source library providing plenty of typical machine learning algorithms and computer vision, such as face detection, facial landmarks detection, face feature extraction [**?**]. It provides a more advanced CNN-based face detector. As the Dlib can be built and done in inference with CUDA support, it can better compute performance. However, that does not work in real-time on a CPU

2.1.2.2 ResNet-29 modified by Davis King

Davis E. King modified the regular ResNet-34 structure, dropped some layers, and rebuilt neural networks consisting of 29 convolution layers. It expects 150x150x3 sized inputs and represents face images as 128-dimensional vectors.

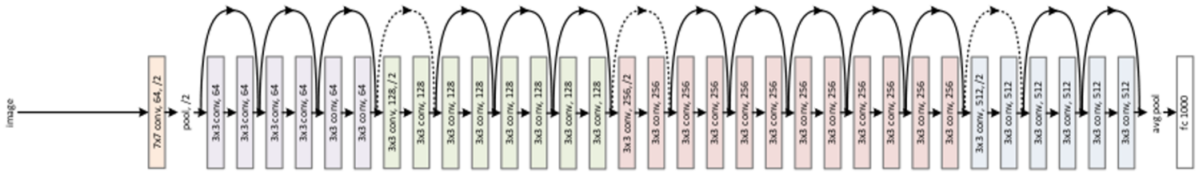

Figure 2.21: Resnet 34 Architecture

We choose Resnet-34 as our Face Representation model. This model is re-trained by various data sets, including FaceScrub and VGGFace2, to learn how to find face representations with 3-million samples. Davis E. King then tested the built for labeled faces in the wild (LFW) data set and got 99.38% accuracy, which means that the Dlib face recognition model can compete with the other state-of-the-art face recognition models and human beings.

After localizing all face coordinates in the frame and resizing them into appropriate size, the face representation model is applied to extract face features into a 128-Dimensional vector. At this time, we do not classify this embedded vector simultaneously. Instead, we push it into a vector list and send it to the server for classification as a batch after each defined period.

2.1.2.3 Experience Result

The primary purpose of this method is to decode the image with bounding containing the user into a 128-D vector. Then this vector is classified or verified by the Classification Model. For this reason, to measure the performance of this method, we have to integrate it with the Classification Model to measure the result of the Identification Task and conduct it in the next section.

Furthermore, this model is relatively small and does not affect the all performance of the system when we compare it to the Detection Model.

Therefore, in the following section, we measure its accuracy when integrating with the Classification model.

2.1.3 Face Classification Methods

This model aims to find the correct users among different faces in our system's database after we have their features extracted in the *Facial Representation Extraction* step.

When a new image containing a bounding box of users' faces comes, it is encoded into an n-dimensional vector and used as an input for classification methods. Here we propose K-Nearest Neighbors or Support Vector Machine. These classifications have the information of all users' embedded vector faces, and it aims to find the vector with the smallest Euclidean-Distance with the input. If the distance is smaller than a defined threshold, we can consider that the two images are the most similar.

2.1.3.1 K-Nearest Neighbor

The K-NN (K-nearest neighbors) algorithm is one of the simplest supervised learning algorithms proposed by Thomas Cover for classification problems and regression problems. This algorithm is classified as instance-based learning, as this method only remembers without calculating anything during training and only computes during use. Therefore, the KNN model uses input values to predict output values by classifying the data point on how its neighbor is ranked. For example, if we have a dataset of eggs and pepper, KNN will store their features, such as shape and color. It will check its similarity with the color (white or black) and shape when a new object comes.

Optimize Accuracy K-NN

At the beginning of the training state, if we do not choose a specific algorithm for the K-NN model, the Brute Force will be applied. Then when the dataset is growing massively, the naive or brute force K-NN cannot classify quickly because it has to calculate the distance between input points with all the points inside the dataset. Therefore, the consumption time for distance

calculation will increase rapidly, which cannot suit actual production.

For this reason, to address the computational inefficiencies of the brute-force approach, we will introduce two additional algorithms used to implement the K-NN for giving a better performance.

- **KD tree algorithm**: The KD tree is a binary tree structure that recursively partitions the parameter space along the data axes, dividing it into nested orthotropic regions into which data points are filed. The construction of a KD tree is high-speed. Because partitioning is performed only along the data axes, no D-dimensional distances need to be computed. Once constructed, the nearest neighbor of a query point can be determined with only O[log(N)] distance computations. Though the KD tree approach is speedy for low-dimensional (D<20) neighbors searches, it becomes inefficient as D grows very large.

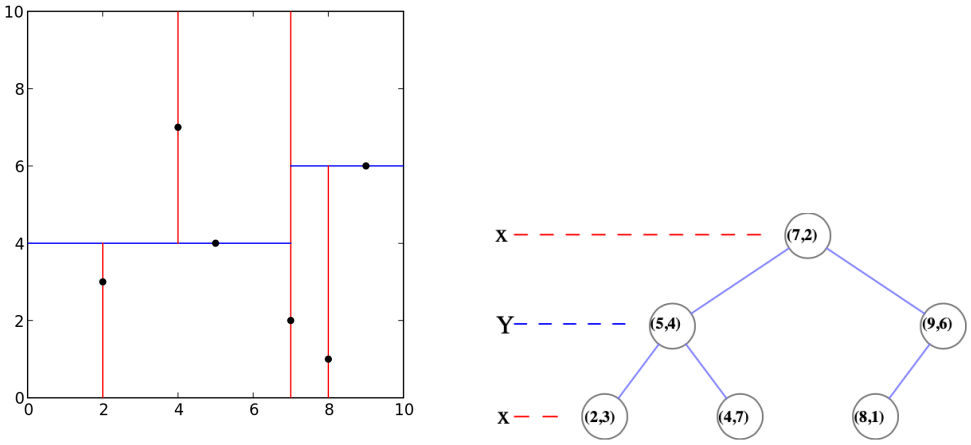

Figure 2.22: K Dimensional Tree

- **Ball tree algorithm**: The ball tree data structure was developed to address the inefficiencies of KD Trees in higher dimensions. This algorithm will construct the dataset into a tree structure where each node of the tree is a hypersphere - a set of points with a constant distance to its center. Randomly pick a point in the dataset and set that point to be the first centroid, continue to find the farthest point from that point to mark as the first farthest point, and from the first farthest point, find

the farthest point and marks it as a second farthest point. Construct a sphere where the centroid is the median of the distance between the first and second farthest points. From the median to the first and second farthest points, we continue to construct two more spheres. The process is continued until reach the limit height of the tree where each sphere is a node of the tree. The result is that each node of the tree is a cluster that can help the nearest neighbor to search faster rather than linearly search.

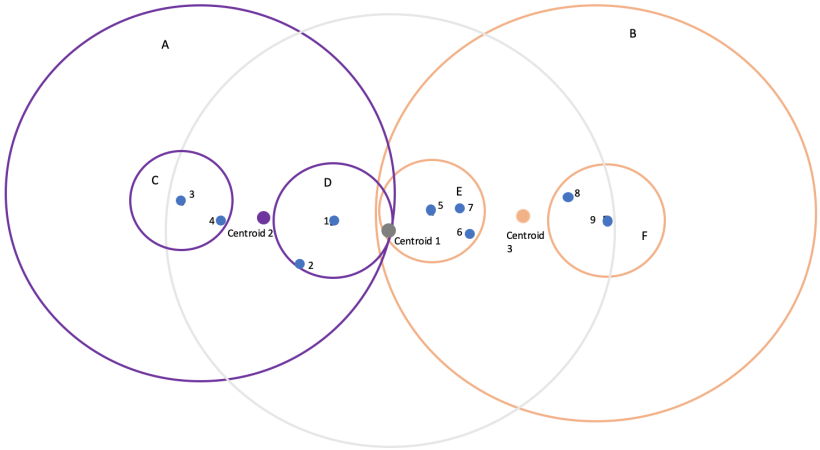

Figure 2.23: Ball Tree

Comparison: When choosing the algorithm selection for each dataset, if accuracy is the critical metric and speed is optional, brute force is the most suitable approach because there will be no wrong clustering determination for input data. If we are looking toward the speed, let us denote N is the number of samples and D is the number of dimensions of each sample. The time complexity of the brute force approach will grow at O[DN]. Ball tree and K-d tree approach will grow at O[D log N]. However, the K-d tree only works well under small dimensional data if D is too large (for example, face embedding vector 128-D); the K-d so far can be slower than the brute force due to the construction data structure and search procedure. Therefore, we decide to use the Ball Tree as the algorithm for the implementation of the K-NN model.

Pros and cons of K-Nearest Neighbor algorithm

- **Pros**:

  - The complexity of the training process is 0. The K-NN algorithm only memorizes the input data and does not calculate anything.
  - The simple implementation of K-NN is due to the simplicity of the algorithm.
  - There is no need to make any assumptions about the distribution of the classes in the data set.

- **Cons**
  - The downside of K-NN is that most of the calculations are in the usage stage. Calculating the distance to each data point and choosing the K shortest distance can be highly complicated and time-consuming.
  - Besides, the entire set of data is stored in memory, which also affects the calculation performance.

2.1.3.2 Support Vector Machine

The Support Vector Machine (SVM) is a classification model that works by building a hyperplane with $(n-1)$ dimensions in the n-dimensional space of the data, such that the hyperplane classifies layers one by one the most optimal way. In other words, for a labeled data set (supervised learning), the algorithm will rely on the learning data to build an optimal hyperplane that is used. To categorize new data. In a 2-dimensional space, this hyperplane is a line separating the space plane into two parts corresponding to 2 layers, with each layer lying on one side of the line.

SVM is a binary classification algorithm; SVM takes input and classifies them into two different classes. With a set of practice examples belonging to two categories, the SVM training algorithm builds an SVM model to classify other examples into those two categories.

Pros and cons of SVM algorithm

- **Pros**
  - SVM works relatively well when there is a clear margin of separation between classes.

  - SVM is effective in high dimensional spaces.
  - SVM is useful in cases where the number of dimensions is greater than the number of samples.
  - SVM is relatively memory efficient

- **Cons**
  - SVM algorithm is not suitable for large data sets.
  - SVM does not perform very well when the data set has more noise, i.e., target classes are overlapping.
  - In cases where the number of features for each data point exceeds the number of training data samples, the SVM will underperform.
  - As the support vector classifier works by putting data points above and below the classifying hyperplane, there is no probabilistic explanation for the classification.

2.1.3.3 Confusion Matrix

This section will introduce some statistical concepts and measurable variables, *precision score, recall score, and accuracy score*, that are used to compare the performance of the above two Classification Models in the comparison section.

If we only compare how many data points are correctly classified, it will not be clear exactly how each class is classified, and which classes are most correctly classified, and which classes are often miss-classified.

Therefore, the confusion matrix is used to find these features. At the same time, the True/False Positive/Negative evaluation is applied to binary classification problems - there are only two data classes. The advantage of this is that it helps us to see the importance of one class over another. For example, in identifying the user, the wrong recognition is more severe than the omission. Similar to predicting the presence of mines, the omission is more severe than miss-forecasting. Then the more critical data layer that needs to be correctly predicted is a Positive class, otherwise the Negative class.

The confusion matrix is shown in the following table:

	Predicted Class Positive	**Predicted Class Negative**
Actual Class Positive	True Positive (TP)	False Negative (FN)
Actual Class Negative	False Positive (FP)	True Negative (TN)

Table 2.4: Confusion Matrix

With the above matrix, we can calculate the *precision score, recall score, and accuracy score* with the following formulas:

$$Accuracy\ Score = \frac{TP+TN}{AllPrediction}$$

$$Precision\ Score = \frac{TP}{TP+FP}$$

$$Recall\ Score = \frac{TP}{TP+FN}$$

In some normal scenarios, it is sufficient to use the *Accuracy Score* instead using *Precision Score*, and *Recall Score* additionally. However, when we analyze the imbalanced classes dataset that is the total number of positive samples is significantly less than the total number of negative samples, the information of *Accuracy Score* does not help us to make any conclusion.

For example, suppose that we have 10000 valid transactions and ten fraudulent transactions. In addition, we have a classification model that classifies 11000 transactions. Then regardless of the transaction's property, fraudulent or valid, we only predict it as a valid one, and our model will have the *Accuracy score* 0.9.

In that case, the meaning of the *Accuracy score* is not valuable, as this model cannot determine which one is fraudulent or not. Therefore, we have to use *Precision Score* and *Recall Score* as the measurable variables to evaluate our Classification Model.

- *Precision* is defined as the percentage of samples that are predicted to be positive and the class that is actually positive out of the total number of samples that are predicted to be positive.

- *Recall* is the percentage of samples that are predicted to be positive and the class that is actually positive out of the total number of samples that are actually positive.
- If the precision is high, the accuracy of the find is high. And high recall means that the rate of missing really positive points is low.
- When precision = 1 then FP = 0, meaning that no truly negative scores are predicted to be positive. At that time, every score found is actually positive. However, we are not sure that the model has found all the positive points because it is likely that the value of FN (the number of points that is actually positive is predicted to be negative) is much higher than TP. While, if recall = 1 means FN = 0, we can find all positive points, but we don't know how many negative points are misclassified.
- From that, for the good Classification Model, we need to balance the *precision score* and *recall score* to achieve a higher performance of our system depending on our demands.

2.1.3.4 Comparision

To compare the performance between the KNN Model and the SVN Model, we first test two models with various set parameters to find the most appropriate one and record in Tables 2.5 and 2.6. Then we perform two experiences and compare the result in Table 2.7.

This experience aims to measure the *Processing Time* and the *Training Time* of both methods. We can ignore the CPU and RAM Utilization of them as they are minor comparing to the above factors.

As shown in the table below, with the same number of images, the precision score of the KNN model is slightly more significant than the SVM model. Specifically, the processing time and training time of the KNN outperform the SVM. For a recall score, although the value of the KNN model is smaller than the SVM, it is not essential as the other two variables, the precision score and the processing time. Therefore, we decide to use the KNN Model as the Classification Model for our system.

Training time (s)	0.016
Number of neighbors	10
KNN algorithm	ball tree
KNN weight	distance
Distance Threshold	0.45
Input	vector 128-dimension
Output	patient ID

Table 2.5: KNN-Classify Model Parameters

Training time (s)	0.668
Kernel	rbf
Class weight	balanced
C	5000.0
Gamma	0.0005
Input	vector 128-dimension
Output	patient ID

Table 2.6: SVM-Classify Model Parameters

	KNN Model	**SVM Model**
Total images	1000	1000
Precision score	0.995	0.992
Recall score	0.863	0.970
Total processing time	1.732	785.80
Max processing time (s)	0.0035	1.323
Min processing time (s)	0.0015	0.663
Mean processing time (s)	0.0017	0.785
Std processing time (s)	0.00019	0.096

Table 2.7: Statistic Result KNN and SVM Model

2.2 Acceleration with NVIDIA TensorRT

TensorRT is an SDK for high-performance deep learning inference. It includes a deep learning inference optimizer and runtime that delivers low latency and high-throughput for deep learning inference applications, mostly NVidia's embedded platform such as the Jetson family platform.

In our system, face detector models are accelerated and run with TensorRT engine to meet the real-time requirements. In detail, the models were first trained and tested on the native Pytorch framework. Next, we use a third-party library ONNX, to convert the weights into ONNX format. After that, we put the ONNX files into the Jetson Nano board and use TensorRT to optimize and build the engine. This step is hardware-dependent, meaning that it can only perform in the platform we use in the inference step. Finally, the built TensorRT engine is independent of any previous framework (Pytorch, ONNX, ...), and we can call it for inference without importing any libraries such as Pytorch or ONNX. The workflow, as summarized in the Figure 2.24.

We also measure the performance of the TensorRT's inference and Pytorch's inference with GPU support for our Face Detection models.

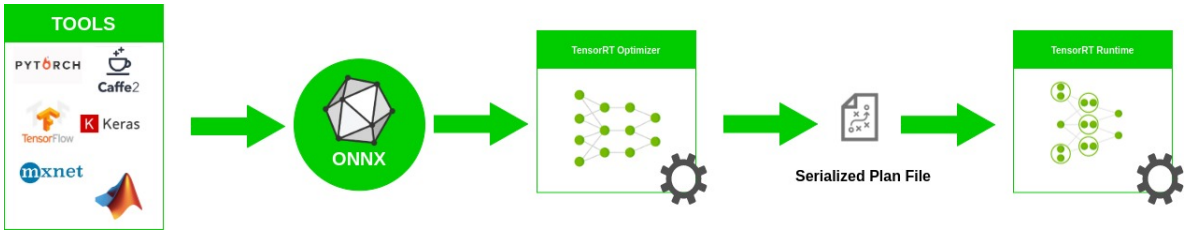

Figure 2.24: Workflow for ONNX-TensorRT

With TensorRT optimization, the model can run up to 2x-3x faster when compared with the original Pytorch inference with GPU support.

Resolution	1280x960	640x480	320x240	240x180 (used)
Pytorch GPU	200ms	70ms	35ms	30ms
TensorRT	80ms	36ms	15ms	12ms

Table 2.8: Comparison between Common Face Detection Method

2.3 Speech Recognition

In this system, the patients can select the examining department using the system GUI according to their demand. However, for some patients, if they do not know which department they should take. The system can help them recommend an appropriate department based on their displaying symptoms.

Therefore, we use the Speech Recognition model to take the description of the symptoms from the patients, convert it into text and use the Natural Language Model to extract this information for our Department Classification model.

The following section will introduce some popular models and tool kits in this field, compare them, and apply the most suitable one for our Speech Recognition module.

2.3.1 List of open source Speech Recognition libraries

2.3.1.1 DeepSpeech

DeepSpeech is an open-source Mozilla's Speech-To-Text engine, using a model trained by machine learning techniques based on Baidu's DeepSpeech architecture, which is at the cutting edge of automated speech recognition technology has gone largely under the radar. It is a 100% free and open source speech-to-text library that also implies the machine learning technology using Google's TensorFlow framework to fulfill its mission.

The engine's architecture was initially based on the one developed by Baidu and presented in a 2014 paper, *"Deep Speech: Scaling up end-to-end speech recognition"*. It has since diverged in many respects from the engine it was motivated by, and the core of the engine is a recurrent neural network (RNN) trained to ingest speech spectrograms and generate text transcriptions. Sadly it sounds like the project is currently only supporting the English pre-training model by default.

However, after the recent Mozilla restructure, the future of the project is unknown, as it may be shut down (or not) depending on what they will decide.

2.3.1.2 Kaldi

Kaldi is an open-source speech recognition software written in C++ and is released under the Apache public license. Kaldi was originally designed for speech researchers, but it is now used in transcription applications. The ambition for Kaldi is to be open-ended enough that different algorithms can be supported. A recent addition to Kaldi is a neural-net library that is believed to be a state-of-the-art algorithm. However, the model files for Kaldi are large (just under 1 GB), and its installation requires the numerical library, which means Kaldi costs a considerable amount of memory and much time for training.

Kaldi is written mainly in C/C++, but the toolkit is wrapped with Bash and Python scripts. Kaldi is best tested on Debian and Red Hat Linux but will run on any Linux distribution or Mac OSX. Windows used to be supported by Kaldi, but at present, Kaldi is not very actively maintaining the Windows compatibility of the code or the Windows build scripts, which have some problems.

2.3.1.3 Google Speech-to-text API

This method is an API powered by Google's AI technologies, which provides the service of transcription speech into text.

Unlike the above method that needs to provide a model for the toolkits to work correctly, Google Speech-to-text API is an automation tool for transcribing speech into text with its implemented model on Google Server with many Notable features:

- Support global user base with Speech-to-Text's extensive language in over 125 languages and variants.
- Apply Google's most advanced deep learning neural network algorithms for automatic speech recognition (ASR).
- Deploy speech recognition wherever we need it, whether in the cloud with the API or on-premises with Speech-to-Text On-Prem.

Since Google provides almost everything, and we only need to call the API to transfer the voice file into the transcription text, we have to pay for the subscription to this service.

However, the price is affordable for this system, just 0.006 dollars every 15 seconds, since the conversation between the system and the patient usually does not last for a long time.

Additionally, we can use an alternative method for speech-to-text is the Google Speech Recognition method. It is and free API with no charge by using the default API in the *Speech Recognition* library in Python. Although the performance is not compared with the Google Speech-to-text API, it is enough for us to build a Recommending Assistant for our system.

2.3.1.4 Comparison

In terms of community and popularity, the developers spend most interest in the *Kaldi* toolkit. As based on the *"Comparing Open-Source Speech Recognition Toolkit"* experiment [**?**], *Kaldi* outperforms all the other recognition. Furthermore, according to research by Cindi Thompson, the Head of Data Science at Silicon Valley Data Science. Both have forums and mailing lists and an active GitHub repo, which means that they have a vast community to exchange experiences and support newer.

In return, the documentation of *Kaldi* is quite comprehensive and a little bit harder for the user to implement. While in the case of Tutorials and Examples of Google Speech Recognition and Deep Speech model, they have very readable, thorough, and effortlessly follow the documentation.

However, the essential factor is that *Kaldi* still does not support embedded devices yet. Otherwise, the Jetson Nano series can install and run both Google Speech Recognition and *DeepSpeech* quickly. In the Table below, we do the experience to compare the inference time between Deep Speech and Google Speech Recognition.

	Sound File Length	**Inference Time**
Deep Speech	1.98 (s)	2.294 (s)
Google Speech Recognition	1.98 (s)	0.507 (s)

Table 2.9: Comparison between Deep Speech and Google Speech Recognition

Here we implement *DeepSpeech* pre-trained model and the Google Speech Recognition to decode a 1.98s testing audio file. As we can see in Table 2.9, in the same sound file length, the *DeepSpeech* model's inference time needs approximately 3 seconds. In comparison, the time for processing this time of *Google Speech Recognition* is only 0.5 seconds. Therefore, it could be undesirable to implement a real-time speech recognition program on embedded devices with *DeepSpeech*

2.3.2 Conclusion

In conclusion, we decide to use *Google Speech Recognition* to implement a speech recognition application. Based on our research, although *Kaldi* has the best performance among speech recognition, it is impossible to import it into the Jetson Nano when we already have the Face Detection and Face Identification model running on it. Furthermore, the task which uses the speech recognition model is an additional feature. Therefore, we should give priority to the main feature of our system.

2.4 Rasa Open Source

Rasa is an open-source machine learning framework for automated text and voice-based conversations. Understand messages, hold conversations, and connect to messaging channels and APIs.

With *Rasa*, we can create personalized, automated interactions with customers at scale. *Rasa* provides infrastructure & tools necessary for building the very best assistants - ones that meaningfully transform how customers communicate with businesses.

```
nlu:
- intent: greet
  examples: |
    - Hi
    - Hey!
    - Hallo
    - Good day
    - Good morning

- intent: subscribe
examples: |
    - I want to get the newsletter
    - Can you send me the newsletter?
    - Can you sign me up for the newsletter?

- intent: inform
  examples: |
    - My email is example@example.com
    - random@example.com
    - Please send it to anything@example.com
    - Email is something@example.com
```

Figure 2.25: NLU Training Examples

For an assistant to recognize what a user is saying, no matter how the user phrases their message, we need to provide example messages the assistant can learn. We group these examples according to the idea or the goal the message is expressing, which is also called the intent. In the code block on the right, we have added an intent called greet, which contains example messages like “Hi”, “Hey”, and “good morning”. Intents and their examples are used as training data for the assistant’s Natural Language Understanding (NLU) model.

2.4.1 NLU Training Data

In this thesis, we use *Rasa* for the classification of patient intents. Based on those intents, we will extract the required entries. For example, to select the medical department, the entities needed here are subject (i, we, ...), department name (Khoa Tai Mũi Họng, Khoa Mắt). To do that, we need to provide *Rasa* with the NLU Training Data.

NLU training data consists of example user utterances categorized by intent. Training examples can also include entities. Entities are structured pieces of information that can be extracted from a user's message. We can also add extra information such as **regular expressions** and **lookup tables** to our training data to help the model identify intents and entities correctly.

NLU training data is defined under the nlu key. Items that can be added under this key are:

Intent: When we want to extract user intent, we usually group this by intent key and listed under the examples key. Usually, we will list one example per line as the figure 2.25.

Entities

- Entities are structured pieces of information extracted from a user's message, such as the city's name, phone, and address.
- Entities are annotated in training examples with the entity's name. In addition to the entity name, we can annotate an entity with synonyms, roles, or groups.
- We have various techniques in *Rasa*, ranging from the Lookup table and Regular Expression to extract the entities. These techniques will be described in the following section.
- In training examples, entity annotation would look like the Figure 2.26:

```
nlu:
- intent: check_balance
  examples: |
    - how much do I have on my [savings]("account") account
    - how much money is in my [checking]{"entity": "account"} account
    - What's the balance on my [credit card account]{"entity": "account", "value": "credit"}
```

Figure 2.26: Entities Training Examples

- In the example above, our purpose is to extract the intent check balance. Here, we have an entity that is needed to extract, account type with three values "checking", "savings", and "credit".

Lookup Table

- Lookup tables are lists of words used to generate case-insensitive regular expression patterns. The format is as the Figure 2.27:
- For example, in figure 2.27, we have the intent **display_symptoms** and three entities needed to extract in user's messages for this intent, subject, **part_of_body**, and **problem**. These elements in each lookup table are the values that we have to find for each entity.
- When we supply a lookup table in our training data, the contents of that table are combined into one large regular expression. This regex is used to check each training example to see if it contains matches for entries in the lookup table.
- Lookup table regexes are processed identically to the regular expressions directly specified in the training data and can be used either with the RegexFeaturizer or with the RegexEntityExtractor. The name of the lookup table is subject to the same constraints as the name of a regex feature. In order words, we can imagine that the following training data in figure 2.28 can equal to all the training data in figure 2.29.

```
nlu:
- intent: display_symptom
  examples: |
    - [tôi](subject) [sốt](problem)
    - [tôi](subject) bị [cay](problem) ở [mắt](part_of_body)
    - [mắt](part_of_body) của [tôi](subject) bị [cay](problem)

- lookup: subject
    - tôi
    - mình

- lookup: part_of_body
    - mặt
    - miệng
    - mắt

- lookup: problem
    - cay
    - đau
```

Figure 2.27: Lookup Table NLU Training Examples

```
nlu:
- intent: display_symptom
  examples: |
    - [tôi](subject) bị [cay](problem) ở [mắt](part_of_body)

- lookup: subject
    - tôi
    - mình

- lookup: part_of_body
    - mặt
    - miệng

- lookup: problem
    - cay
    - đau
```

Figure 2.28: Lookup Table NLU Training Examples 1

```
nlu:
- intent: display_symptom
  examples: |
    - [tôi](subject) bị [cay](problem) ở [mắt](part_of_body)
    - [mình](subject) bị [cay](problem) ở [mắt](part_of_body)
    - [tôi](subject) bị [cay](problem) ở [miệng](part_of_body)
    - [mình](subject) bị [cay](problem) ở [miệng](part_of_body)
    - [tôi](subject) bị [đau](problem) ở [mắt](part_of_body)
    - [mình](subject) bị [đau](problem) ở [mắt](part_of_body)
    - [tôi](subject) bị [đau](problem) ở [miệng](part_of_body)
    - [mình](subject) bị [đau](problem) ở [miệng](part_of_body)
```

Figure 2.29: Lookup Table NLU Training Examples 2

2.4.2 Pipeline Component

Tokenizers

- In natural language processing, tokenization is the process of breaking human-readable text into machine-readable components. The most obvious way to tokenize a text is to split the text into meaningful words.
- Such as, if we have a sentence equals *'Chàng trai 9X Quảng Trị khởi nghiệp từ nấm sò'*, after tokenized by VietNamesese Tokenization tools, the result of the above sentence is *['Chàng trai', '9X', 'Quảng Trị', 'khởi nghiệp', 'từ', 'nấm', 'sò']*

Entity Extractor: Entity extractors extract entities, such as person names or locations, from the user message.

- **CRFEntityExtractor**: This component implements a conditional random fields (CRF) to do named entity recognition. CRFs can be thought of as an undirected Markov chain where the time steps are words and the states are entity classes. Features of the words (capitalization, POS tagging, etc.) give probabilities to certain entity classes, as are transitions between neighbouring entity tags: the most likely set of tags is then calculated and returned.
- **RegexEntityExtractor**: This component extract entities using the lookup tables and regexes defined in the training data. The component checks if the user message contains an entry of one of the lookup

tables or matches one of the regexes. If a match is found, the value is extracted as entity. This component only uses those regex features that have a name equal to one of the entities defined in the training data. Make sure to annotate at least one example per entity.

CHAPTER 3

SYSTEM DESIGN

In this chapter, we propose the architecture of our system, introducing the hardware and software components used to implement it. Besides, we also list all the services of Azure that contribute to our system, analyze its function and choose the most suitable one in each service. Next, we give the structure and build the database system for our system. Finally, we design the Check-in device, Mobile, and Web application of the system. It includes the functional and non-functional requirements and the final product.

3.1 Proposal Architecture

As described in the section introduction, in this project, we build an E-Healthcare System used at hospitals to supply the convenience for patient's experience, prevent lessens human interaction, and provide a safer environment for both patients and doctors/nurses. Furthermore, this system also speeds up the check-in phase for patients when they take examinations.

To implement this system, we divide it into three parts and implement them separately, Check-In Devices, Users' Applications, and Database, which acts as a bridge for communicating between Check-In Devices and Users' Applications. Figure 3.1 is the proposed architecture of our system. The detailed implementation of all parts will be described in Chapter 4. Before diving into these sections, we introduce our system components, including hardware and software. Then the general design of the database, Check-In device, and user's applications are also included.

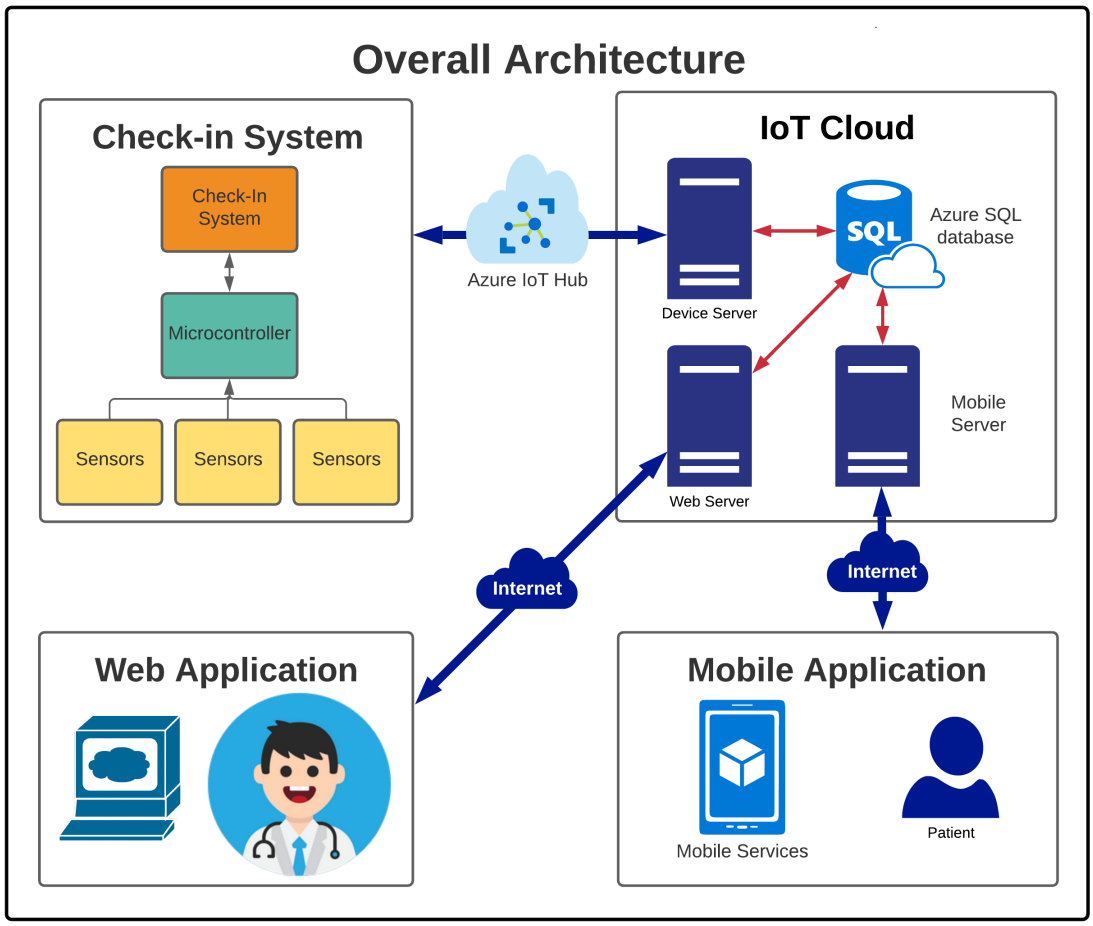

Figure 3.1: Proposal Architecture

3.2 Devices And Components

3.2.1 Hardware Components

3.2.1.1 Jetson Nano Developer Kit

Jetson Nano is a small, powerful computer for embedded AI systems and IoT. NVIDIA Jetson Nano Developer Kit has the compute performance to run modern AI workloads at unprecedented size, power, and cost. This board lets us run multiple neural networks in parallel for applications like image classification, object detection, segmentation, and speech processing.

The developer kit has incredibly power-efficient, consuming as little as 5 watts, and comes with extensive I/Os, ranging from CSI to GPIO. It makes it simple for developers to connect a diverse set of new sensors to enable

various AI applications. Jetson Nano is also supported by NVIDIA Jet Pack, available using an easy-to-flash SD card image, making it fast, easy to get started, and reducing complexity, the developers' overall effort. Figure 3.2 and Table 3.1 below shows an overview of the Jetson Nano board layout and its specification.

Based on these experiments in the section 2.1.1.4 and 2.1.3.4, we decide to choose the Jetson Nano Device as our embedded system as it has enough power to handle all the service that the system needs.

AI Performance	472 GFLOPs
GPU	128-core NVIDIA Maxwell™ GPU
CPU	Quad-Core $ARM^R Cortex^R - A57$ MPCore processor
Memory	4 GB 64-bit LPDDR4 25.6GB/s
Storage	16 GB eMMC 5.1
Power	5W / 10W
Price	$99.00

Table 3.1: Specification of Jetson Nano

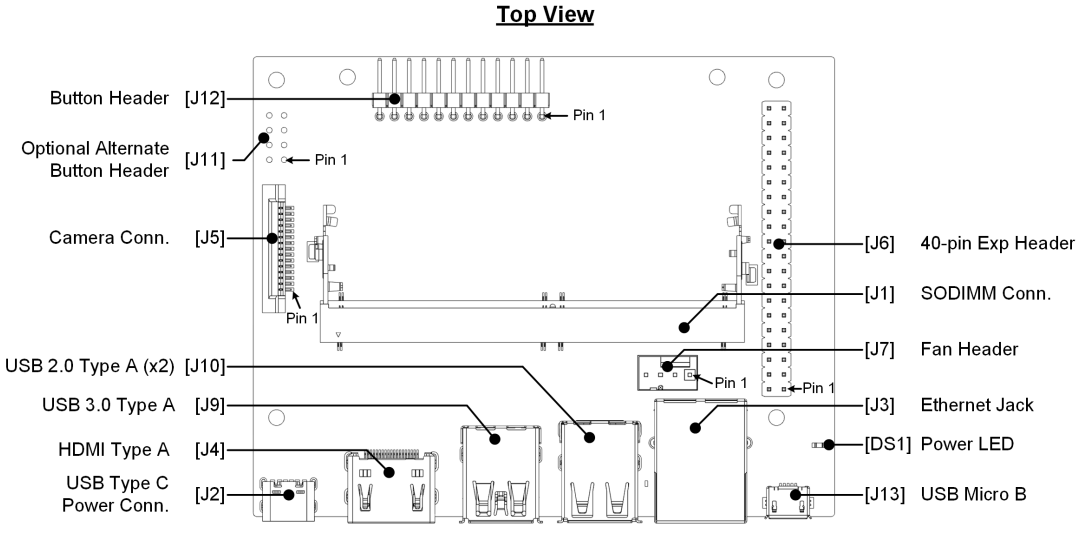

Figure 3.2: View of Jetson Nano Developer Kit Carrier Board

3.2.1.2 Seeed ReSpeaker Mic Array v2.0

The ReSpeaker Mic Array v2.0 is an upgrade to the original ReSpeaker Mic Array v1.0 - a product of Seeed Studio, the IoT hardware enabler pro-

viding services. ReSpeaker Mic Array v2.0 works as creating or adding voice-interface to projects. It has a far-field microphone array device that can capture the voices up to 5 meters away even though background noise exists. Following the manufacturer, this device's features and functions[**?**]:

- Far-field voice capture
- Support USB Audio Class 1.0 (UAC 1.0)
- Four microphones array around device support device capture surround voice easily.
- The WM8960 is a low power speaker drivers which can provide 1 W per channel into 8 W loads.
- 12 programmable RGB LED indicators used as DOA display which can point out direction of the voice come.
- Speech algorithms and features:
  - Voice Activity Detection
  - Direction of Arrival
  - Beamforming
  - Noise Suppression
  - De-reverberation
  - Acoustic Echo Cancellation

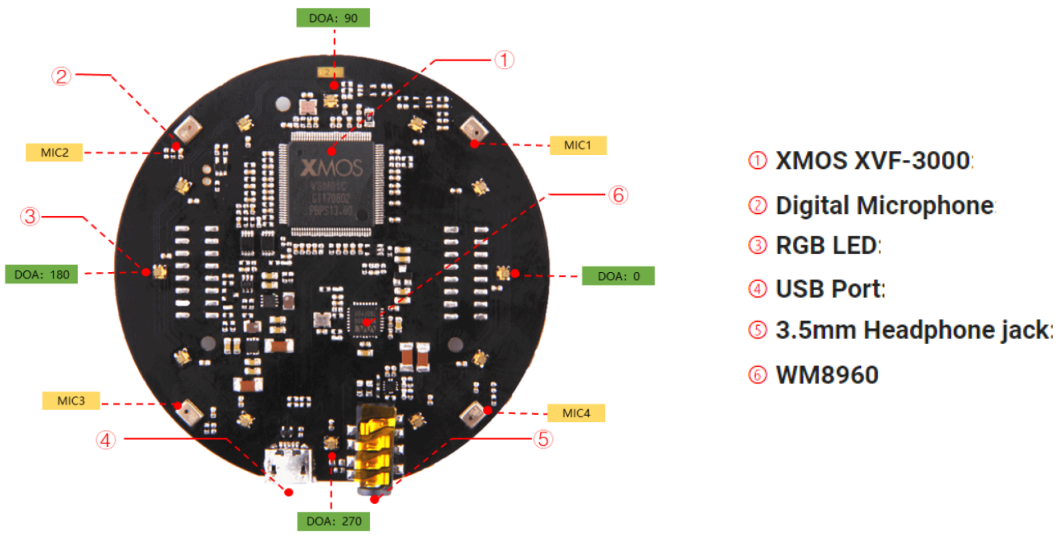

Figure 3.3: ReSpeaker Mic Array v2.0 Components

3.2.1.3 Height Sensor

For measuring height, many available sensors can provide help for measuring the patient height. In planning the systems, we came across two sensors that are suitable for the systems requirement: Ultrasonic Sensor and Mini LiDAR Module. Both modules use the similar principle of sending and receiving transmits modulation waves. Therefore, to choose the suitable devices for our system, we will decide when comparing between their specifications.

Operating range	0.3m-12m
Average power consumption	0.6W
Applicable voltage range	4.5V-6V
Acceptance angle	2.3°
Minimum resolution ratio	1cm
Accuracy	1% (less than 6m), 2% (6m-12m)
Size	42mm × 15mm × 16mm
Communication interface	UART
Wavelength	850nm
Price	$50.00

Table 3.2: Specification of Mini LiDAR

Power Supply	+5V DC
Ultrasonic Frequency	40kHz
Working Current	40kHz
Effectual Angle	$< 15°$
Ranging Distance	2cm – 400 cm/1" 13ft
Resolution	0.3 cm
Measuring Angle	30 degree
Trigger Input Pulse width	10μS
Dimension	45mm x 20mm x 15mm
Communication interface	I2C
Price	$10.41

Table 3.3: Specification of Ultrasonic Sensor

Table 3.2 and 3.3 are the specification of the Mini LiDAR and Ultrasonic Sensor, respectively. Based on that, we decided to choose the TFmini because of some higher figures compared to HC.SR04: equivalent in voltage and size, TFmini is better in detecting range, accuracy, and resolution.

3.2.1.4 Weight Sensor

To measure the weight, we had planned to use a fully pre-made smart scale that can be bought on the Internet. Unfortunately, there are none available that we can retrieve the data. We then buy the low-cost digital scale available on the Internet then remade it to satisfy our purposes.

The remake scale using the familiar principle for all digital scales. Using load cells, retrieve signal by converting module to transform the analog signal from the load cells to readable by human and machine digital data. The load cells are specially shaped metal parts that have strain gauges glued to them. The strain gauges are resistors that change their resistance when they are bent. When the metal part bends, the resistance of the load cell changes. We use the four available 50kg half-bridge load cells that come with the scale. Below is the specification:

Capacity	kg	40-50
Comprehensive	mv/v	0.05
Zero Output	mv/v	±0.1
Input Resistance	Ω	1000±20
Output Resistance	Ω	1000±20
Operation Temp. Range	°C	0±50
Price	$	0.3 x 4

Table 3.4: Load Cells Specification

To convert the analog signal sent by the load cells, we choose the most common converter module, the HX711. This module is a precision 24-bit analog-to-digital converter (ADC) designed for weighing scales and industrial control applications to interface directly with a bridge sensor. There is no programming needed for the internal registers. All controls to the HX711 are

through the pins. Below is a specification:

Working Voltage	2.7 5VDC
Working Current	15mA
Output data rate	10SPS or 80SPS (selectable)
Resolution	24 bits ADC
Voltage resolution	40mV
Size	38 x 21 x 10 mm
Price	$1.00

Table 3.5: HX711 Specification

The input multiplexer selects either Channel A or B differential input to the low-noise programmable gain amplifier (PGA). Channel A can be programmed with a gain of 128 or 64, corresponding to a full-scale differential input voltage of ±20mV or ±40mV, respectively, when a 5V supply is connected to an AVDD analog power supply pin. Channel B has a fixed gain of 32. An on-chip power supply regulator eliminates the need for an external supply regulator to provide analog power for the ADC and the sensor. The clock input is flexible. It can be from an external clock source, a crystal, or the on-chip oscillator that does not require any external component. There is no programming needed for the internal registers. All controls to the HX711 are through the pins.

3.2.1.5 Temperature Sensor

Measuring the body temperature comes in many solutions. However, for the convenience of the users/patients, we came up with using Infra-Red (IR) thermometer to measure the temperature as the IR thermometer requires non-contact measurement. We had looked into the two following IR thermometers.

- **MLX90640**: The MLX90640 is a fully calibrated 32x24 pixels thermal IR array in an industry-standard 4-lead TO39 package with a digital interface. The MLX90640 contains 768 FIR pixels. An ambient sensor is integrated to measure the ambient temperature of the chip and a supply sensor to measure the VDD. The outputs of all sensors IR, Ta, and VDD

are stored in internal RAM and are accessible through I2C. Below is the specification.

Working Voltage	3V and 5V (selectable model)
Accuracy	±0.5°C
Temperature range	-40°C ÷ 300°C
Distance range	max 7m (110°x75°)
Size	28mm × 16 mm (L x W)
Price	$96.99

Table 3.6: Specification of MLX90640

- **MLX90614**: It is an InfraRed thermometer for non-contact temperature measurements. Both the IR-sensitive thermopile detector chip and the signal conditioning ASSP are integrated into the same TO-39 can. The MLX90614 is a ready-to-use low-cost non-contact thermometer provided from Melexis with output data linearly dependent on the object temperature with high accuracy and extended resolution. Technically the principle of this module is similar to MLX90640, but instead of an infrared array, it fires only one lazer, which provides lower resolution than MLX90640. The specification can be seen below.

Working Voltage	3V and 5V (selectable model)
Accuracy	±0.5°C
Temperature range	-70 to 380°C
Distance range	2cm-5cm
Size	17 x 11mm (L x W)
Price	$15.20

Table 3.7: Specification of MLX90614

Upon testing in the implementing state, both the IR thermometer performance is great as their specification described. However, the MLX90640 can measure with a further distance, and with the infrared array, the module can provide more samples with higher accuracy, while MLX90614 can provide one sample per second. In conclusion, we choose the MLX90640.

3.2.1.6 Oxygen saturation and Heart Rate

To measure the oxygen saturation (SPO2) and Heart Rate, for better precision and save some time, we used an entirely made device, OSO2 SBPM14 - v2.5, for convenience since the machine has already been tested and industrially manufactured to ensure the accuracy of the measurement. Below is some specification:

Measurement range	Pressure: 0 - 299 mmHg Pulse: 40 - 180 beats/min
Accuracy/Calibration	Pressure: ±3mmHg (2%) Pulse: ±5% bpm
Price	$600.00

Table 3.8: Specification of Device

3.2.1.7 ESP32

ESP32 is designed for mobile, wearable electronics, and Internet-of-Things (IoT) applications. It features all the state-of-the-art characteristics of low-power chips, including fine-grained clock gating, multiple power modes, and dynamic power scaling. The low-duty cycle is used to minimize the amount of energy that the chip expends. ESP32 is a highly integrated solution for Wi-Fi and Bluetooth IoT applications, with around 20 external components. ESP32 integrates an antenna switch, RF balun, power amplifier, low-noise receive amplifier, filters, and power management modules. As such, the entire solution occupies a minimal Printed Circuit Board (PCB)area.

Micro-controller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	DOIT
Price	$5.47

Table 3.9: Specification

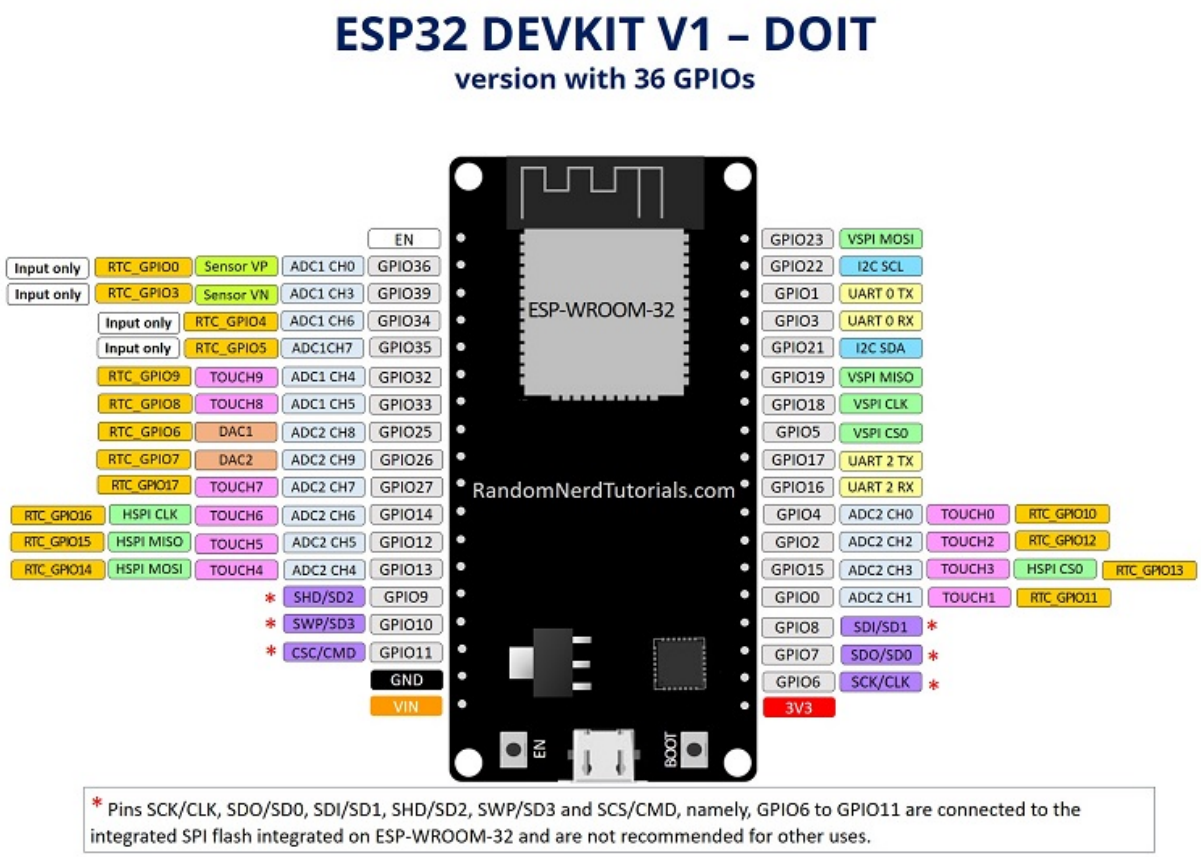

Figure 3.4: ESP32 DEVKIT V1 - DOIT Pinout

Overall Hardware Cost

Below are the summarize of the hardware components that we will use in our system and the total cost of them

Jetson Nano Developer Kit	$99.00
Seeed ReSpeaker Mic Array v2.0	$78.00
TFmini Mini LiDAR	$50.00
4 x 50kg Half-bridge Load cells	$1.20
HX711	$1.00
MLX90640	$96.99
OSO2SBPM14 - v2.5	$600.00
ESP32 DEVKIT V1 - DOIT	$5.47
Total	$931.66

Table 3.10: Total Cost

3.2.2 Software Components

- **NVIDIA JetPack SDK**: NVIDIA JetPack SDK is the most comprehensive solution for building AI applications; a reference file system derived from Ubuntu, Linux Kernel, bootloader, NVIDIA drivers, and more. It includes the latest OS images for Jetson products, along with samples, developer tools, documents, and essential libraries and APIs for AI projects. [**?**]
  - **TensorRT** is a high-performance deep learning inference runtime for AI applications such as image classification, segmentation, and object detection neural networks. It speeds up deep learning inference and reduces the runtime memory footprint for convolutional and deconv neural networks.
  - **CUDA Deep Neural Network (cuDNN)** is a library that provides high-performance primitives for deep learning frameworks. It includes support for tensor transformations, convolutions, and activation functions.
  - **NVIDIA Container Runtime** The container runtime enables the creation, distribution, and use of containerized GPU accelerated applications.
  - **OpenCV** is the leading open-source library for computer vision, image processing, and machine learning and now features GPU acceleration for real-time operation.
- **QT Creator**: Qt is a cross-platform UI framework. It was originally conceived in 1990, with its first public release in 1995 for Linux. Over time, it is developed support for embedded devices, such as cell phones, through desktop platforms, such as Windows and Mac OS X. We use the main framework to build our device user interface for a better user experience and ensure the lightweight and fast to deploy.
- **LXDE**: LXDE, which stands for Lightweight X11 Desktop Environment, is a desktop environment that is lightweight and fast. It is designed to be user-friendly and slim while keeping resource usage low. LXDE uses less RAM and less CPU while being a feature-rich desktop

environment. Unlike other tightly integrated desktops, LXDE strives to be modular so that each component can be used independently with few dependencies. It makes porting LXDE to different distributions and platforms more accessible.

- **Free Real-Time Operating System (Free-RTOS)**: A real-time operating system (RTOS) is an operating system (OS) intended to serve real-time applications that process data as it comes in, typically without buffer delays. Processing time requirements (including any OS delay) are measured in tenths of seconds or shorter increments of time. A real-time system is a time-bound system that has well-defined, fixed time constraints. Most RTOSs use a pre-emptive scheduling algorithm. FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 35 microcontroller platforms. It is distributed under the MIT License.

- **Node.js**: Nodejs is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command-line tools and server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web application development around a single programming language, rather than different languages for server-side and client-side scripts.

- **Express.js**: Express.js, or simply Express, is a back-end web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js. Reason for choosing this framework because the system web application simply a CRUD (create, read, update and delete) type of application, so we need the most compact and most straightforward approach path to reduce the unnecessary complexity of the system.

- **Flutter**: Flutter is Google's portable UI toolkit for crafting beautiful, natively compiled mobile, web, and desktop applications from a single

codebase. Flutter works with existing code, is used by developers and organizations worldwide, and is free and open source. Flutter is designed to support mobile apps that run on both Android and iOS and interactive apps that we want to run on our web pages or the desktop. (Note that desktop support is in beta, but a snapshot of the beta is available on the stable channel.) Apps that need to deliver highly branded designs are particularly well suited for Flutter. However, we can also create pixel-perfect experiences that match the Android and iOS design languages with Flutter. Flutter's package ecosystem supports various hardware (such as camera, GPS, network, and storage) and services (such as payments, cloud storage, authentication, and ads).

- **JQuery**: jQuery is a JavaScript library designed to simplify HTML DOM tree traversal and manipulation and event handling, CSS animation, and Ajax. It is free, open-source software using the permissive MIT License. As of May 2019, jQuery is used by 73% of the 10 million most popular websites. Web analysis indicates that it is the most widely deployed JavaScript library by a large margin, having at least 3 to 4 times more usage than any other JavaScript library.

- **Handlebars**: Handlebar is a simple templating language. It uses a template and an input object to generate HTML or other text formats. Handlebars templates look like a regular text with embedded Handlebars expressions. Handlebars provide the power necessary to let us build semantic templates effectively with no frustration. Handlebars compile templates into JavaScript functions. It makes the template execution faster than most other template engines. Since the server-side are run in Node.js and our client-side application are solely design with raw Html and client-side Javascript, we choose Handlebars as a substance for there are supported from npm package of Node.js and make displaying data to Html more convenient.

- **Adafruit IO**: Adafruit.io is a cloud service. It is meant primarily for storing and then retrieving data. Adafuit IO can: Display our data in real-time, online, make our project internet-connected, control motors, read sensor data, and more, and finally connect our project to other

internet-enabled devices. Adafruit.io can handle and visualize multiple feeds of data.

3.3 Database

In this project, we want to build a database that acts as the means of communication between our Local Device, Web Application of Doctors, and Mobile Application for patients. Therefore, the database needs to store the following tables.

In terms of the hospital, we need to store their information, including buildings, rooms, devices, and employees(doctor, admin). For each doctor, we have their information, such as employee id, name, or address. Furthermore, we have to track which department and hospital that this doctor currently works in.

For the examination, we have a table that contains the information of all current examination rooms. This table can change daily depending on the hospital. For each row, this table shows the department name, examing room, and the assigned doctor. Whenever a new patient submits the examination for a specific department, we have a queue that stores the target department and submission form, including information on the patient and their sensor measurements.

In order to track the check-in devices, we store their production code, registered id, location, and status for the admin to check whether it is active or not. With patients, we have their basic information and encoded images for the recognition task. Besides, we have another table that stores the history of the patients, including the examination date, diagnosis of the doctor, and their sensor measurements.

3.3.1 Entity Relational Diagram

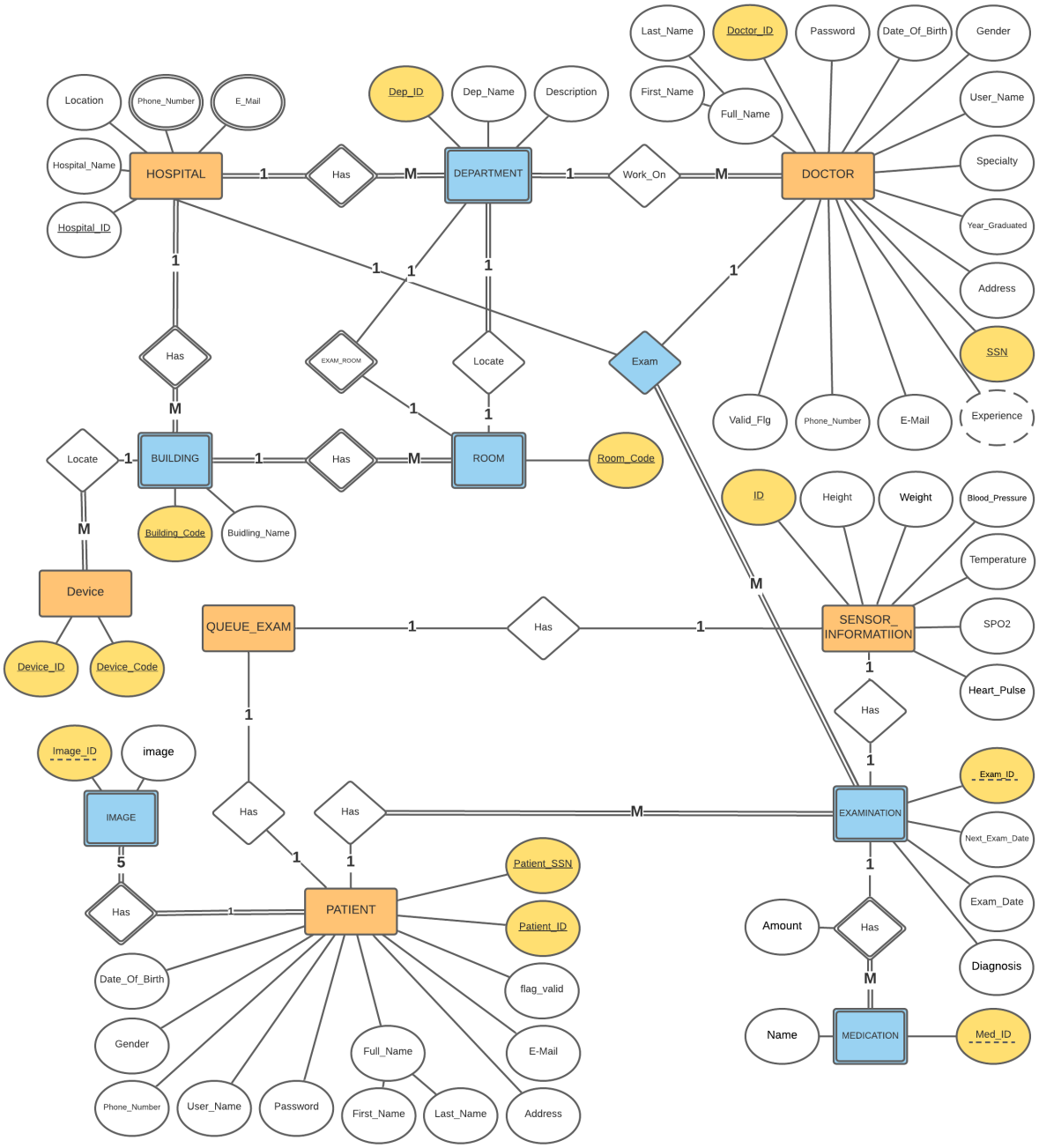

Figure 3.5: Entity Relational Diagram

3.3.2 Tables

1. **Building**

COLUMN_NAME	TYPE_NAME	LENGTH	NULLABLE
Hospital_ID	int	4	0
Building_Code	char	2	0

2. **Department**

COLUMN_NAME	TYPE_NAME	LENGTH	NULLABLE
Hospital_ID	int	4	0
Dep_ID	int	4	0
Dep_Name	nvarchar	200	0

3. **Device**

COLUMN_NAME	TYPE_NAME	LENGTH	NULLABLE
Device_ID	int	4	0
Device_Code	char	10	1
Hospital_ID	int	4	0
Building_Code	char	2	0
Valid_flg	char	1	0

4. **Exam_Room**: This table contains the information of which doctor of a department will be in charge and the number of patients of a room of a building in a hospital. The table has 5 foreign keys: *Room_Code* as *Exam_Room_Code* from table **ROOM** to determine the room code; *Building_Code* from table **BUILDING** to determine the building that the room belongs to; *Hospital_ID* from table **HOSPITAL** to determine the hospital the room and the building belongs to; *Dep_ID* from table **DEPARTMENT** to indicate the department that has the ownership on the room; *Doctor_ID* as *Doctor_Assigned* from table **DOCTOR** to indicate the doctor that will in charge of the room of the date.

COLUMN_NAME	TYPE_NAME	LENGTH	NULLABLE
Hospital_ID	int	4	0
Dep_ID	int	4	0
H_ID	int	4	0
Building_Code	char	2	0
Exam_Room_Code	char	3	0
Number_Patients	int	4	1
Doctor_Assigned	int	4	1

5. **Doctor**

COLUMN_NAME	TYPE_NAME	LENGTH	NULLABLE
Doctor_ID	int identity	4	0
First_Name	nvarchar	14	0
Last_Name	nvarchar	80	0
Date_Of_Birth	date	20	0
Gender	char	1	0
Address	nvarchar	200	0
Phone_Number	varchar	11	0
SSN	varchar	12	0
Specialty	nvarchar	200	0
Year_Graduated	date	20	0
Valid_Flg	char	1	0
User_Name	varchar	50	0
Password	varchar	32	0
E_Mail	varchar	50	0
Hospital_ID	int	4	0
Dep_ID	int	4	0

6. **Examination**: This table contains the information about the examination of patients. The examination consists of the exam date, the next exam date, the patient who took the exam, diagnosis, sensors information, and the examination location. The table has 4 foreign keys: *Patient_ID* from table **PATIENT** to determine the patient that took the

examination; *Sensor_ID* from table **SENSORS_INFORMATION** to determine the sensors value of that examination; *Doctor_ID* from table **DOCTOR** to indicate the doctor in charge of the examination; *Hospital_ID* from table **HOSPITAL** to determine the hospital that the examination took place.

COLUMN_NAME	TYPE_NAME	LENGTH	NULLABLE
Patient_ID	int	4	0
Exam_ID	int	4	0
Exam_Date	date	20	0
Next_Exam_Date	date	20	1
Diagnosis	nvarchar	200	1
Sensor_ID	int	4	0
Doctor_ID	int	4	0
Hospital_ID	int	4	0

7. **Hospital**

COLUMN_NAME	TYPE_NAME	LENGTH	NULLABLE
Hospital_ID	int identity	4	0
Hospital_Name	nvarchar	200	0
Location	nvarchar	200	0

8. **Hospital_Email**

COLUMN_NAME	TYPE_NAME	LENGTH	NULLABLE
Hospital_ID	int	4	0
E_Mail	varchar	50	0

9. **Hospital_Phone_Number**

COLUMN_NAME	TYPE_NAME	LENGTH	NULLABLE
Hospital_ID	int	4	0
Phone_Number	varchar	11	0

10. **Medication**

COLUMN_NAME	TYPE_NAME	LENGTH	NULLABLE
Patient_ID	int	4	0
Exam_ID	int	4	0
Med_Name	varchar	50	0
Description	nvarchar	400	0
Amount	int	4	0

11. **Patient**

COLUMN_NAME	TYPE_NAME	LENGTH	NULLABLE
Patient_ID	int	4	0
First_Name	nvarchar	14	0
Last_Name	nvarchar	80	0
Date_Of_Birth	date	20	0
Gender	char	1	0
Address	nvarchar	200	0
Phone_Number	varchar	11	0
SSN	varchar	12	0
User_Name	varchar	50	0
Password	varchar	32	0
E_Mail	varchar	50	0
flag_valid	char	1	0

12. **Patient_Image**: This table contains the decoded image matrix of the patients for the other server to train.

COLUMN_NAME	TYPE_NAME	LENGTH	NULLABLE
Patient_ID	int	4	0
Img_ID	int	4	0
Img	varchar	3072	1
Add_Date	date	20	0

13. **Queue_Examination**: This table contains the information of the current patient queue. The table has 5 foreign keys: *Hospital_ID* as *H_ID*

from table **HOSPITAL**, *Building_ Code* from table **BUILDING**, *Room_ Code* as *Exam_ Room_ Code* from table **ROOM** to determine the location of the queue; *Patient_ ID* from table **PATIENT** to determine the patients that are in the queue; *Sensor_ ID* from table **SENSOR** to indicate the sensors information of that patient.

COLUMN_NAME	TYPE_NAME	LENGTH	NULLABLE
H_ID	int	4	0
Building_Code	char	2	0
Exam_Room_Code	char	3	0
STT	int	4	0
Patient_ID	int	4	0
Sensor_ID	int	4	0

14. **Room**

COLUMN_NAME	TYPE_NAME	LENGTH	NULLABLE
Hospital_ID	int	4	0
Building_Code	char	2	0
Room_Code	char	3	0

15. **Sensor_Information**

COLUMN_NAME	TYPE_NAME	LENGTH	NULLABLE
ID	int	4	0
Weight	numeric	6	1
Height	numeric	5	1
Temperature	numeric	5	1
Heart_Pulse	numeric	5	1
BMI	numeric	5	1
SPO2	numeric	5	1

3.4 Microsoft Azure

3.4.1 IoT Hub

IoT hub is a service of Microsoft Azure hosted in the cloud. It acts as a central message hub for bi-directional communication between the IoT application backend server and the devices that it manages. IoT Hub enables us to build scalable, full-featured IoT solutions such as managing industrial equipment used in manufacturing, tracking valuable assets in healthcare, and monitoring office building usage with the capabilities, full-featured IoT solutions, reliable and secure communications between millions of IoT devices, and an application backend server. [**?**]

IoT Hub supports communications from the device to the cloud and from the cloud to the device with various messaging patterns, including device-to-cloud telemetry, uploading files from devices, and request-reply methods to control devices from the cloud. IoT Hub also supports monitoring to help us track creating devices, connecting devices, and device failures...

Besides, it has plenty of SDK libraries and supported languages include C, Embedded C, C#, Java, Python, Nodejs, and the following protocols for connecting devices: HTTPS, AMQP, AMQP over WebSockets, MQTT, MQTT over WebSockets

Cloud To Devices Communication

Cloud To Devices Communication is the heart of IoT Hub. It provides bi-directional communication between IoT devices and application back-end servers with three options depending on the demands of each service.

- The Direct Method.
- The Cloud To Device Message Method.

The following section will describe more details about the concept, including Scenario, Data flow, Durability, Targets, Size, and Protocol of these

Cloud To Devices Communication. Then we will come up with our decision among these choices.

3.4.1.1 Direct Method

IoT Hub gives us the ability to invoke direct methods on devices from the cloud. Direct Method represents a request-reply interaction with a device similar to an HTTP call. They require immediate confirmation of the result when sending a message to the devices by the server, such as succeed or fail immediately (after a user-specified timeout). This approach is helpful for scenarios where the course of immediate action is different depending on whether the device was able to respond.

Each device method targets a single device. Schedule jobs on multiple devices show how to provide a way to invoke direct methods on multiple devices and schedule method invocation for disconnected devices. Anyone with service connects permissions on IoT Hub may invoke a method on a device. Direct methods follow a request-response pattern and are meant for communications that require immediate confirmation of their result. For example, interactive control of the device, such as turning on a fan. [**?**]

Method Life-Cycle

Direct Method is implemented on the device and may require zero or more inputs in the method payload to instantiate correctly. This method is synchronous. Either succeed or fail, the server will wait for a response until having a notification after the timeout period (default: 30 seconds, settable between 5 and 300 seconds). Direct methods are HTTPS-only from the cloud side and MQTT, AMQP, MQTT over WebSockets, or AMQP over WebSockets from the device side.

The payload for method requests and responses is a JSON document up to 128 KB.

Scenario	Commands that require immediate confirmation, such as turning on a fan.
Data flow	Two-way. The device app can respond to the method right away. The solution back end receives the outcome contextually to the request.
Durability	Disconnected devices are not contacted. The solution back end is notified that the device is not connected.
Targets	Single device using deviceId, or multiple devices using jobs.
Size	Maximum direct method payload size is 128 KB
Protocol	Available using MQTT or AMQP.

Table 3.11: Details of Direct Method

3.4.1.2 Cloud-to-device messages

The Cloud To Device message method is one-way communication. It enables the server to notifies or send messages to devices without waiting for confirmation. Each device queue holds, at most, 50 cloud-to-device messages. To try to send more messages to the same device results in an error.

The Cloud To Device message life cycle

To guarantee at-least-once message delivery, our IoT hub persists cloud-to-device messages in per-device queues. For the IoT hub to remove the messages from the queue, the devices must explicitly acknowledge completion. This approach guarantees resiliency against connectivity and device failures.

When the IoT hub service sends a message to a device, the service sets the message state to Enqueued. When a device wants to receive a message, the IoT hub locks the message by setting the state to Invisible. This state allows other threads on the device to start receiving other messages. When a device thread completes the processing of a message, it notifies the IoT hub by completing the message. The IoT hub then sets the state to Completed.

The life-cycle state graph is displayed in the following diagram:

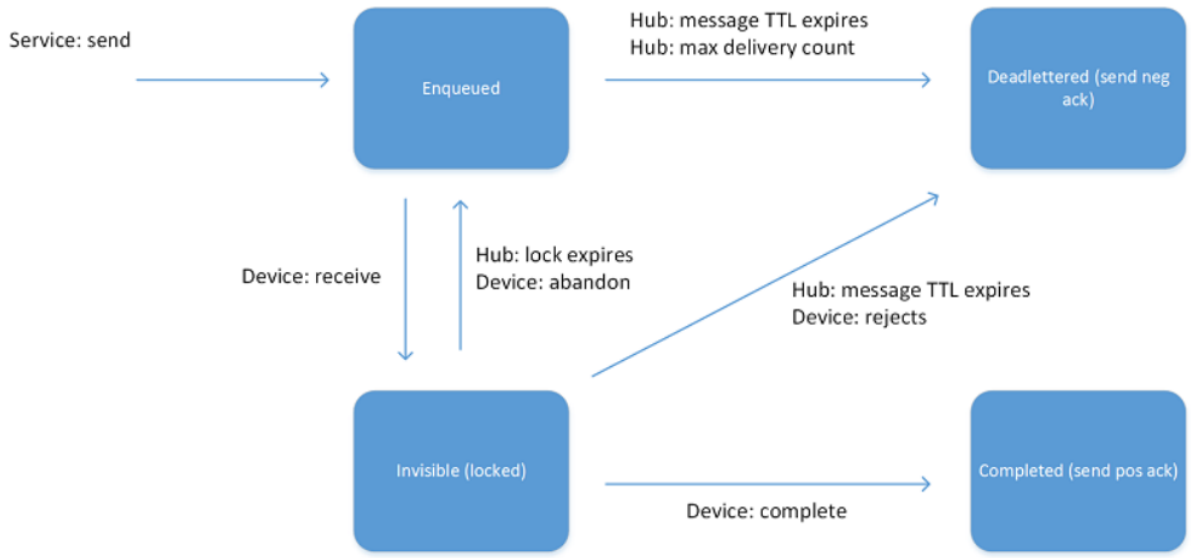

Figure 3.6: Cloud to Device message life cycle

A device can also:

- *Reject* the message, which causes the IoT hub to set it to the Dead lettered state. Devices that connect over the Message Queuing Telemetry Transport (MQTT) Protocol can't reject cloud-to-device messages.
- *Abandon* the message, which causes the IoT hub to put the message back in the queue, with the state set to Enqueued. Devices that connect over the MQTT Protocol can't abandon cloud-to-device messages.

A thread could fail to process a message without notifying the IoT hub. In this case, messages automatically transition from the Invisible state back to the Enqueued state after a visibility time-out (or lock time-out). The value of this time-out is one minute and cannot be changed.

The max delivery count property on the IoT hub determines the maximum number of times a message can transition between the Enqueued and Invisible states. After that number of transitions, the IoT hub sets the state of the message to Dead lettered. Similarly, the IoT hub sets the state of a message to Dead lettered after its expiration time.

Scenario	One-way notifications to the device app.
Data flow	One-way. The device app receives the message
Durability	Messages can be retained by IoT Hub for up to 48 hours.
Targets	Single device by deviceId.
Size	Up to 64 KB messages.
Protocol	Available on all protocols. Device must poll when using HTTPS.

Table 3.12: Details of Cloud-to-device messages

3.4.1.3 Comparison Direct and Cloud-to-Device Message method

Based on both descriptions of the two methods, we decide to use the Cloud to Device Message as the responding request for the Check-In devices rather than the Direct Method. Since our system is not afraid to lose the responses, the device will automatically resend the request to the server after a timeout when we do not receive a message. Furthermore, by implementing this method, the responding thread from the Device Server does not need to wait for the acknowledgment of the Check-In device. All they need is to send the response as fast as possible.

3.4.2 Event Hub

Azure Event Hubs is a scalable event processing service that ingests and processes large volumes of events and data by providing a distributed stream processing platform with low latency and seamless integration, with data and analytics services inside and outside Azure to build our complete significant data pipeline. This service can be considered the "front door" for an event pipeline and often called an event ingestor in solution architectures. [**?**]

3.4.2.1 Key Features of Event-Hub

Fully managed PaaS: Event Hubs is a fully managed Platform as a Service (PaaS) with little configuration or management overhead, so we focus on your business solutions. Event Hubs for Apache Kafka ecosystems give us the PaaS Kafka experience without having to manage, configure, or run our

clusters.

Support for real-time and batch processing: Event Hub enables us to ingest, buffer, store, and process our stream in real-time to get actionable insights. This service uses a partitioned consumer model, allowing multiple applications to process the stream concurrently and control processing speed.

Scalable: With Event Hubs, we can start with data streams in megabytes and grow to gigabytes or terabytes. The Auto-inflate feature is one of the many options available to scale the number of throughput units to meet our usage needs.

Rich ecosystem: With a broad ecosystem based on the industry-standard AMQP 1.0 protocol and available in various languages .NET, Java, Python, JavaScript, we can quickly start processing our streams from Event Hubs. All supported client languages provide low-level integration.

3.4.2.2 Key Architecture Components

Event producers: Any entity that sends data to an event hub. Event publishers can publish events using HTTPS or AMQP 1.0 or Apache Kafka (1.0 and above).

Partitions: Each consumer only reads a specific subset, or partition, of the message stream.

Consumer groups: A view (state, position, or offset) of an entire event hub. Consumer groups enable consuming applications to each have a separate view of the event stream. They read the stream independently at their own pace and with their own offsets.

Event receivers: Any entity that reads event data from an event hub. All Event Hubs consumers connect via the AMQP 1.0 session. The Event Hubs service delivers events through a session as they become available. All Kafka consumers connect via the Kafka protocol 1.0 and later.

Throughput units: Pre-purchased units of capacity that control the throughput capacity of Event Hubs.

The following figure shows the Event Hubs stream processing architecture:

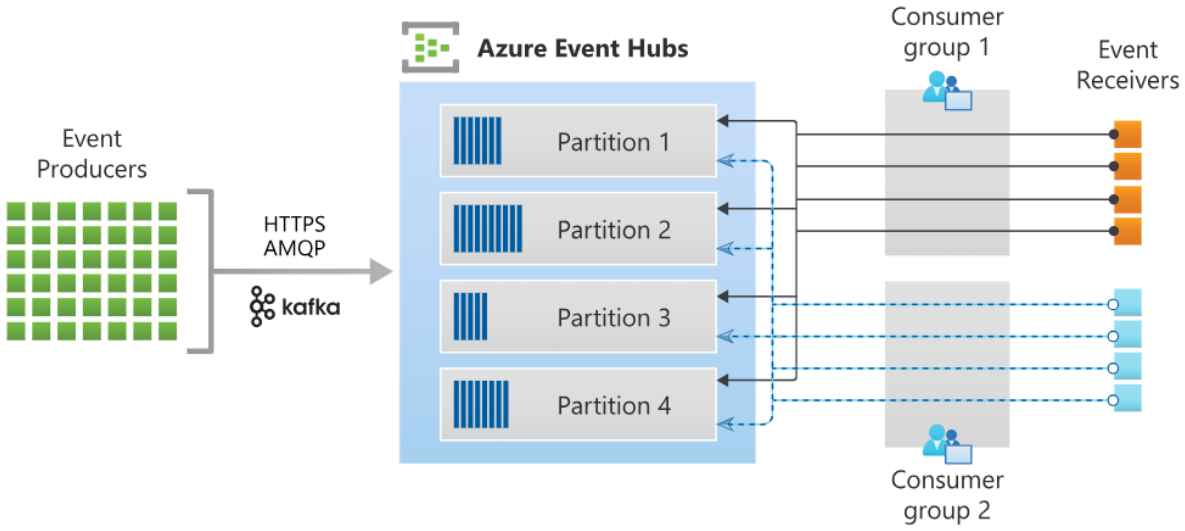

Figure 3.7: Event Hubs stream processing architecture

3.4.2.3 Event Procedures

Any entity that sends data to an Event Hub is an event producer (synonymously used with event publisher). The Event Hubs service provides REST API and .NET, Java, Python, JavaScript, and Go client libraries for publishing events to an event hub via AMQP 1.0, the Kafka protocol, or HTTPS.

Furthermore, we can publish events individually or batched messages. However, a single publication has a limit of 1 MB, regardless of whether it is a single event or a batch. Public events will be rejected when they are larger than this threshold.

As described in the Figure 3.8, the Event Hubs throughput is scaled by using partitions and throughput-unit allocations. It is a best practice for publishers to remain unaware of the specific partitioning model chosen for an Event Hub and only specify a partition key used to assign related events to the same partition consistently.

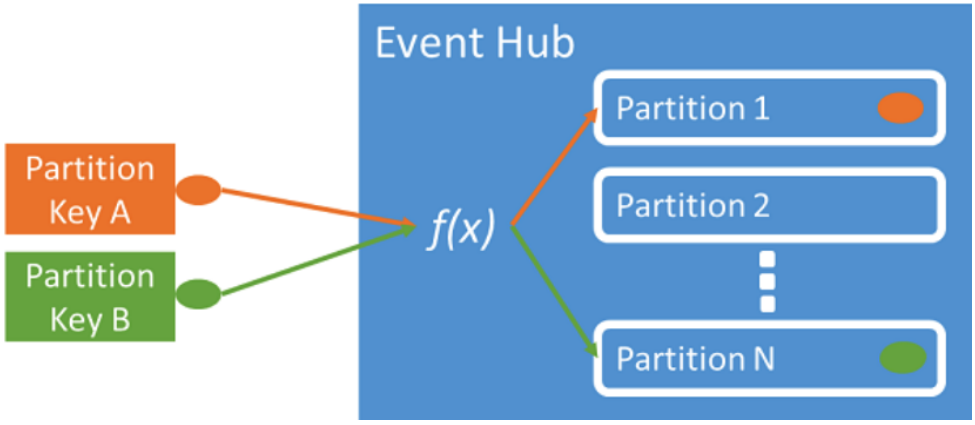

Figure 3.8: Event Hubs publishing specific partition key

3.4.2.4 Partitions

When events are sent to Event Hubs, they are organized as sequences of events in one or more partitions following the queue rule; new events are added to the end of this sequence.

Event Hub enables us to select a specific partition to publish our events depending on our purposes. If we do not choose this option when publishing an event, a round-robin assignment is used to distribute events to all partitions. When publishing an event, we only aware of its partition key, not the partition to which the events are published. This decoupling of key and partition insulates the sender from knowing too much about the downstream processing. Specifying a partition key enables us to keep related events together in the same partition and in the exact order in which they were sent.

The number of partitions is specified at creation and must be between 1 and 32 in Event Hubs Standard. The partition count can be up to 2000 partitions per Capacity Unit in Event Hubs Dedicated. However, it is recommended that the fewer partitions, the more sustained throughput units (TU) during the peak load of our application for that particular Event Hub.

Advantages of using partitions

The primary reason when design Event Hubs is to help with the processing of large volumes of events, and partitioning helps with that in two ways:

- Partitioning allows for multiple parallel logs to be used for the same event hub, therefore multiplying the available raw IO throughput capacity.
- When our applications have to handle processing the volume of events sent into an event hub, and the capacity of a single process to handle events is limited, we require scaled-out, parallel processing capabilities, and several substantial processes to overcome this issue. Here, Partitions are how our solution. It feeds those processes and ensures that each event has a clear processing owner.

3.4.3 SQL Database

Azure SQL Database is a fully managed platform as a service (PaaS) database engine that handles most database management functions such as upgrading, patching, backups, and monitoring without user involvement. Azure SQL Database is always running on the latest stable version of the SQL Server database engine and patched OS with 99.99% availability. PaaS capabilities that are built into Azure SQL Database enable us to focus on the domain-specific database administration and optimization activities that are critical for our business.

SQL Database delivers predictable performance with multiple resource types, service tiers, and compute sizes. It provides dynamic scalability with no downtime, built-in intelligent optimization, global scalability and availability, and advanced security options.

With Azure SQL Database, we can create a highly available, scalable, and high-performance data storage layer for the applications and solutions in Azure. SQL Database can be the right choice for a variety of modern cloud applications because it enables us to process both relational data and non-relational structures, such as graphs, JSON, spatial, and XML.

3.4.4 App Service

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. We can develop our favorite language, be it .NET, .NET Core, Java, Ruby, Node.js, PHP, or Python. Applications run and scale with ease on both Windows and Linux-based environments.

App Service not only adds the power of Microsoft Azure to our application, such as security, load balancing, auto-scaling, and automated management. We can also take advantage of its DevOps capabilities, such as continuous deployment from Azure DevOps, GitHub, Docker Hub, other sources, package management, staging environments, custom domain, and TLS/SSL certificates.

3.5 Check-in Device

3.5.1 Functional Requirements

- Connect into the Wifi.
- Register new user.
- Detect faces including recognizing user's identity and measuring the sensors.
- Choose the suitable department as patient's need.
- Assist the patient to recommend the appropriate department.
- Balance the number of patients in each examination room within the same department.

3.5.2 Non-Functional Requirements

- The system is able to run 24/7 without failure.
- The whole time of one cycle is lower than 2 minutes.
- The recognition time for each request server is lower than 1.5 second.

- The system's user interface is friendly and easy to use.

3.5.3 Use-Case Diagram

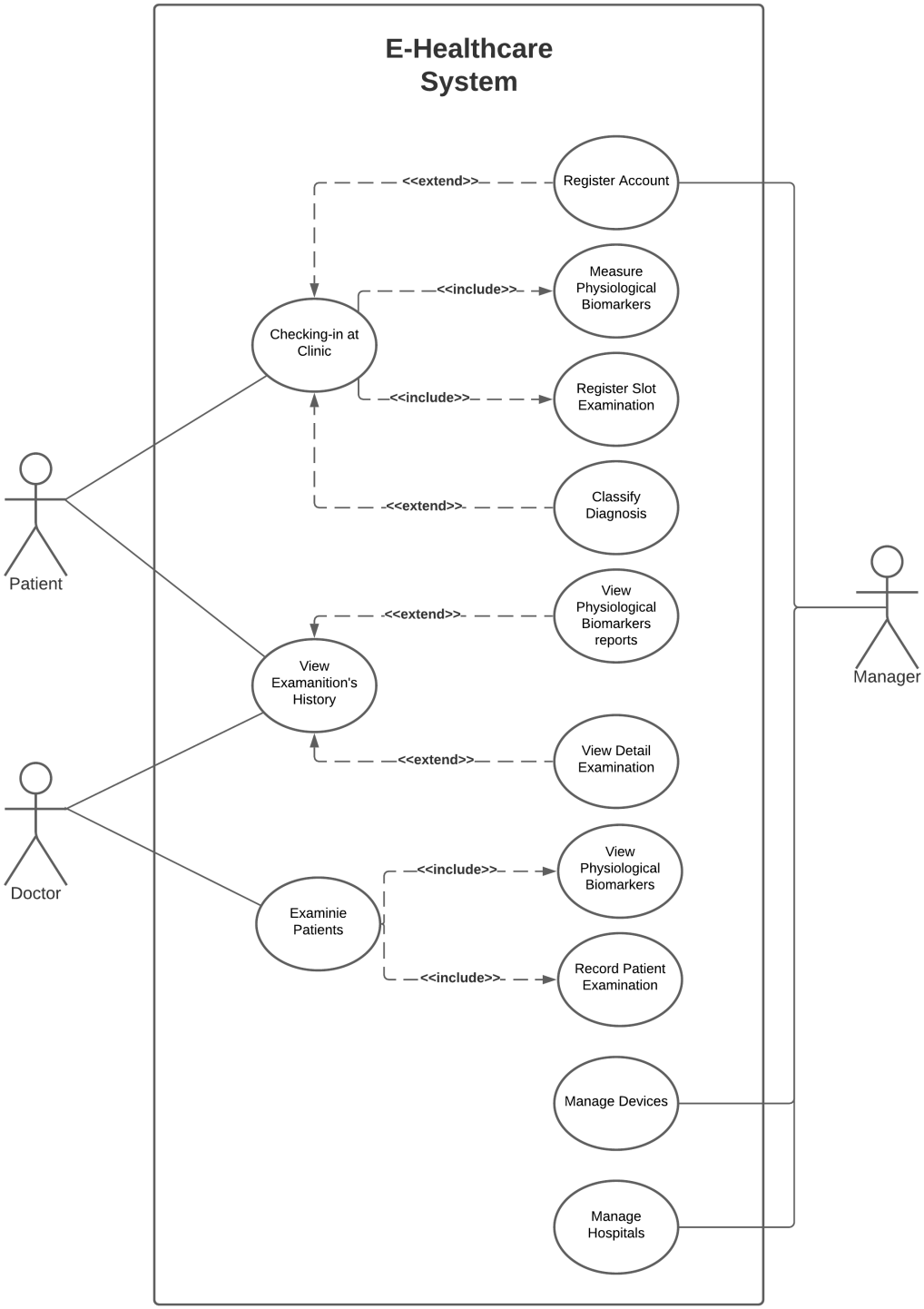

Figure 3.9: Use Case Diagram of Whole System

3.6 Mobile Application

The main objective of this application is to provide the patients that took the exam in the hospital using the e-Health system. The patients can log in using the provided account after examining and seeing the historical records of their examinations and biometric measurements. Moreover, there is a function for when the next time the patient check-in, the application will tell the user when it is their turn to take an examination from the doctors.

3.6.1 Functional Requirements

- The patients can log in to the application by their mobile phone number as id and ssid as password. They can change it later in the application.
- Patients can track their examination history, and view the detail, including biometric measurements, diagnosis, medication and arrangement for next examination.
- When taking the examination, patients can see the information of the location (building code, room code) and the STT of their upcoming examination.
- When it is their turn, the application pop-up a message to notify the patients.

3.6.2 Non-Functional Requirements

- The color of the application should be appropriate .
- The application should be able to work well on multiple platform like Android and iOS
- The response time of the application should be fast.
- The interface of the application should be clear and user-friendly.
- The amount of time that the application loads the information from the database should be insignificant.

3.6.3 Use-Case Diagram

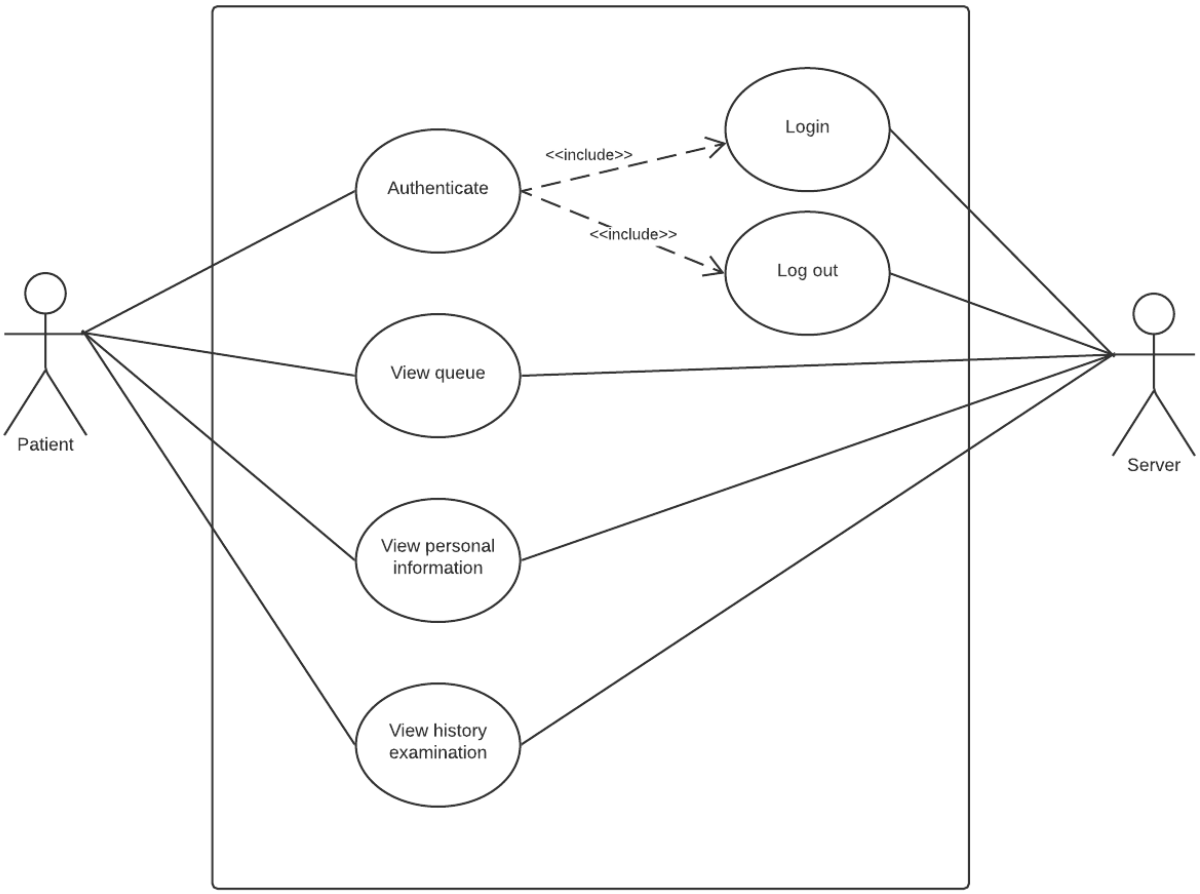

Figure 3.10: Mobile application use-case diagram

3.7 Web Application

The main objective of this application is to provide the doctors in the hospital having access to the information of previous examinations and measurements along with biometrics measurements of the patients performed earlier in the system booth. The doctors can also diagnose, inform when there will be the subsequent examination, and prescribe medication. Moreover, an administrative page for the administrator of the system can manage employees (doctors, nurses), hospitals and devices.

3.7.1 Functional Requirements

Doctor

- Doctors can login into their accounts to examine
- Change the information, including basic information or password.
- Doctor can see how many patients are assigned to their rooms
- Doctor can also see the information of all the previous examinations of the patient to track their health status
- Doctor can examine the patients by writing diagnosis of the patient, prescribing medication for patient if required, and determining future examination if necessary

Admin

- Manage the hospitals, including add, modify, or delete the buildings, rooms, departments, or employees information.
- View and add more doctor into the to a specific hospital.
- Specify the examination rooms for each department, and assign doctor to that room.
- View devices status and add new registered device to corresponding hospital.

3.7.2 Non-Functional Requirements

- The website should be able to work well on many different types of browser like Chrome, Opera, Microsoft Edge, Firefox,...
- The response time of the website should be fast.
- The interface of the website should be clear and user-friendly.
- The amount of time that the website loads the information from the database should be insignificant.

3.7.3 Use-Case Diagram

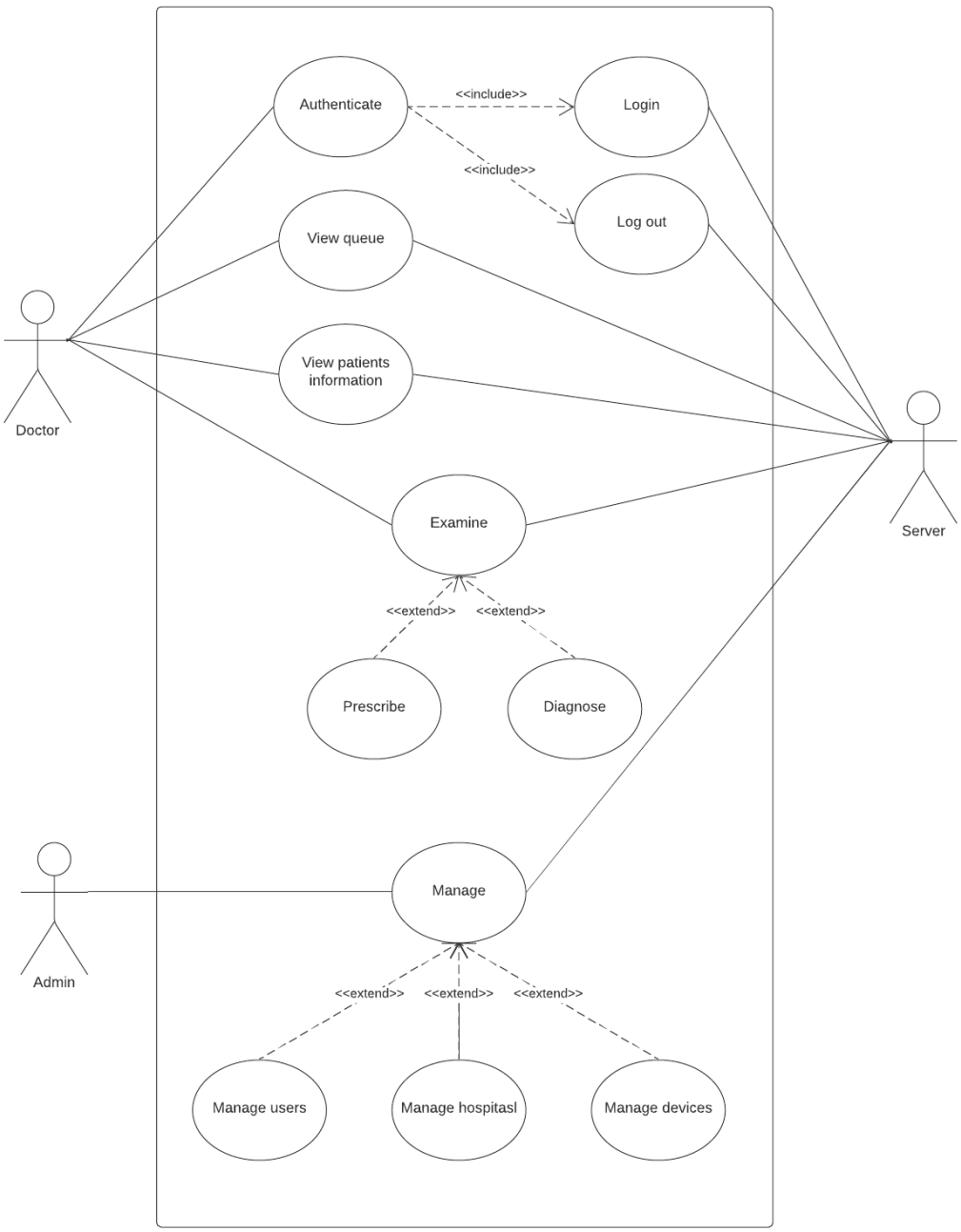

Figure 3.11: Web application use-case diagram

CHAPTER 4

IMPLEMENTATION

This section will describe the detailed implementation of three modules in our system, Web and Mobile applications.

In the first part, we implement the micro-controllers for reading sensors values. Next, the Check-In device is implemented on the Jetson Nano with three main functions: identifying patients, extracting their biologic parameters, and recommending the most appropriate department according to their symptoms. Finally, we build a Device Server for handling the Check-In Device request.

The above functions are completed with a combination of server and device based on Azure service. Moreover, the system's original purpose is to apply Internet of Things services to develop and easily manage devices; ensuring internet connection stability is essential.

4.1 Services Introduction

Before going to the beginning section, implementing the Server serving the services of the Check-In system, we will introduce concepts of all our system services.

4.1.1 Identify patient

This service performs the process of identifying patients or verifying patients. The identification or verification will depend on the patient. When the patient comes to the hospital to take the examination, the system will identify them initially. However, if our system identifies them incorrectly, the patient might use second authorization by verifying their SSN (Social Security Number).

To perform the verification task, the patients choose the Verify option at GUI, and their SSN will be added to the request message. At this time, the server will perform Task identification; otherwise, task verification will be performed.

4.1.2 Get Initial Parameters

When starting operation or restart, each device in the system will request the server to load the necessary parameters.

Next, the Device Server will determine the device's device ID to know which hospital that this device is operating. After that, it will send a query to the database to retrieve a list of departments currently working at this hospital and send it back to the device.

Also, based on that device's device ID, the server can know if the device is being activated or not. If the server detects that the device is no longer active, the server will not send any data to the system.

4.1.3 Extract Symptoms

Our system allows us to collect and analyze the patient's symptoms by converting the patient's voice into text and extracting Intent, Entities such as department name, problems, and part of the body. Based on that, the system can recommend the most appropriate department.

However, the analysis and extracting intent and entities cannot be done on each device due to hardware limits. Therefore, we migrate this task to the server-side.

For each symptom described by patients converted into text, the device will send requests to the server to retrieve intent and entities. From there, we will apply Classify algorithms to be able to predict the most appropriate department.

4.1.4 Submit Examination

This service allows the patients to submit the examination information to register a slot, including patient information, biometric parameters, and medical examination. When the server receives this request, it will update the information of that submission into the system and return the STT number with the room code of the department for the patient.

However, when first-time patients use this system, the system allows patients to register by taking five poses of the face. These images are then encoded into an array of the 128-dimensional vector float64 and attached with examination information in the Submit Examination request. After receiving this request, the server will temporarily create a dummy patient (see in Create Temporary Patient section) and update examination information in the database. Here, the Face Recognition System still does not have the face' data of this patient. After a day of patients visiting hospitals, the system will remove all the temporary information about the examination, dummy patient, and the patient's encoded image in the database if they do not participate in this examination.

4.1.5 Create Temporary Patient

This module creates a temporary patient for the new patient when taking the examination for the first time. When receiving a Submit Examination request from a new patient, the server temporarily creates a dummy patient record with flag = 0 (dummy patient) and stores a list of the patient's encoded

image in the database.

4.1.6 Active Patient

After finishing the examination, then dummy information of new patients will be updated by the doctor. Then, the web application sends a request for the Device Server to activate the face recognition identification method for these patients. When receiving this request, the Device Server loads encoded images from the database, trains, and updates the Face Recognition system.

4.1.7 Update List Examination Room

This service enables the web application to change, add, delete or modify the examination room at a specific hospital. For example, when Go Vap hospital assigns the optometry to room A1-201 or assigns doctor A as a doctor to examine that department, the web application first updates the database according to the changes at this hospital, then sending a request to the server to update the list examination room to all the devices.

4.2 Capture Biologic Parameters

This section will introduce the reading sensor mechanism of each sensor device, the implementation of microcontrollers, and the communication between Jeton and microcontrollers.

4.2.1 Capturing Height and Temperature Value

4.2.1.1 Reading TFMini sensor

The TFmini has the field of view with 2.3‘, further the object, increasing detection range. This module communicates with the microcontroller using Serial Port Data Communication Protocol and Line Sequence. The module data is hexadecimal output data; each frame data is encoded with 9 bytes, including one distance data (Dist); each distance data has corresponding signal strength information (Strength); the frame end is the data parity bit

4.2.1.2 Reading MLX90640 sensor

The sensor return value is an array of temperatures. The array consists of 768 IR sensors (also called pixels). Each pixel is identified with its row and column position as Pix(i,j), where i is its row number (from 1 to 24), and j is its column number (from 1 to 32).

The system uses the normal mode of the camera to measure. Depending on the selected frame rate Fps in the control register, the data for IR pixels and Ta will be updated in the RAM each 1/Fps seconds. In this mode, the external microcontroller has full access to the internal registers and memories of the device.

4.2.1.3 State Diagram

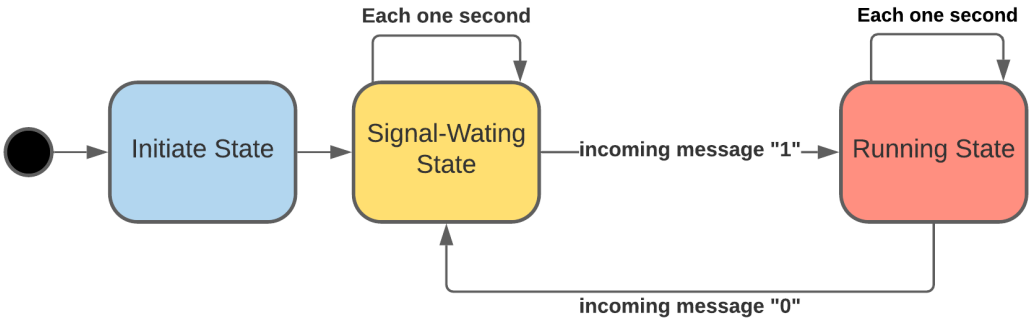

Figure 4.1: Diagram State of the Main Thread

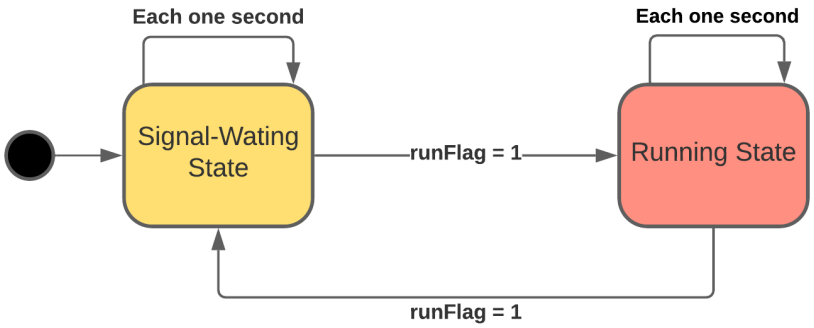

Figure 4.2: Diagram State of the Reading Sensor Thread

4.2.1.4 Input and Output

- Input: Input set includes "1" and "0" where "1" is to initiate measuring and "0" is to stop
- Output: Encoded string "0x83/height/temp/0x81"

4.2.1.5 State Diagram Explanation

This module runs with two states, Signal waiting and Sending Sensor Value in the main thread. While another two thread that runs with RTOS to read as much as many values of height and temperature value from the sensor.

This module uses RTOS to create threads that run each ten milliseconds to read temperature and height sensor value concurrency instead of reading them sequentially. Since in one second, we can only get two samples from the temperature sensor, while the height sensor receives 100 samples. Therefore, sequentially running causes the system to get only two samples of each sensor in one second. While using RTOS makes two modules concurrency running and helps us receive more than one hundred height samples and two samples for temperature.

Main Thread

1. **Initial State**: The system initiates the connection with sensors, establish a connection with Jetson device, and create two thread used to reading the temperature and height value concurrency using RTOS. These threads run with a period of ten-milliseconds and have two states *Running State* and *Waiting State*.
2. **Signal-waiting State**: When there is an incoming signal "1" from the Jetson device for getting sensor value, the *runFlag* is activated (for the two modules reading temperature and height enable to measure the sensor. As they are set with a timer to run continuously by RTOS, if the *runFlag* is not active, then the reading state is passed), and the main thread enters the running state.

3. **Running State**: At this state, the thread is set to run for one second. In each cycle, it:

   - Calculates the final value of each sensor by dividing the sum of all previous data by the number of records. In that way, we get the average value of temperature and height.
   - Then two sensor values are decoded in one message as the output format and send to the Jetson device.
   - Finally, the main thread clears the cumulative sum and the number of temperature and height sensor records.

   When the Jetson device has enough data, it sends a message "0" and makes the main thread go to the Signal-waiting state. Simultaneously, the *runFlag* is disabled. The temperature and heigh values are not read in the two extra threads.

Reading Temperature Thread

1. **Waiting State**: When the *runFlag* is disabled, this thread goes to the *Waiting State* and does nothing. At this state, the thread do nothing and wait for the signal to enter the *Running Thread*

2. **Running State**: When the *runFlag* is enabled, this thread goes to the *Running State*. At this state, the thread repeats the process, including reading the valid temperature, calculating the cumulative sum of sensor values, and increasing the number of records. This process repeats until the *runFlag* is disabled, and this thread goes to the *Waiting State*.

Reading Height Thread: Similar with the reading temperature thread, the reading height thread have two state *Reading State* and *Waiting State*. Instead of reading temperature, the thread read the height value.

4.2.2 Capturing Weight Value

4.2.2.1 Reading HX711 sensor

The HX711 has two channels: Channel A and B

- Channel A differential input is designed to interface directly with a bridge sensor's differential output. It can be programmed with a gain of 128 or 64. The large gains are needed to accommodate the small output signal from the sensor. When a 5V supply is used at the AVDD pin, these gains correspond to a full-scale differential input voltage of ±20mV or ±40mV, respectively.
- Channel B differential input has a fixed gain of 32. The full-scale input voltage range is ±80mV when a 5V supply is used at the AVDD pin.

Serial output: Pin PD_SCK and DOUT are used for data retrieval, input selection, gain selection, and power down controls. When output data is not ready for retrieval, digital output pin DOUT is high. Serial clock input PD_SCK should be low. When DOUT goes too low, it indicates data is ready for retrieval. By applying 25 27 positive clock pulses at the PD_SCK pin, data is shifted out from the DOUT output pin. Each PD_SCK pulse shifts out one bit, starting with the MSB bit first until all 24 bits are shifted out.

4.2.2.2 State Diagram

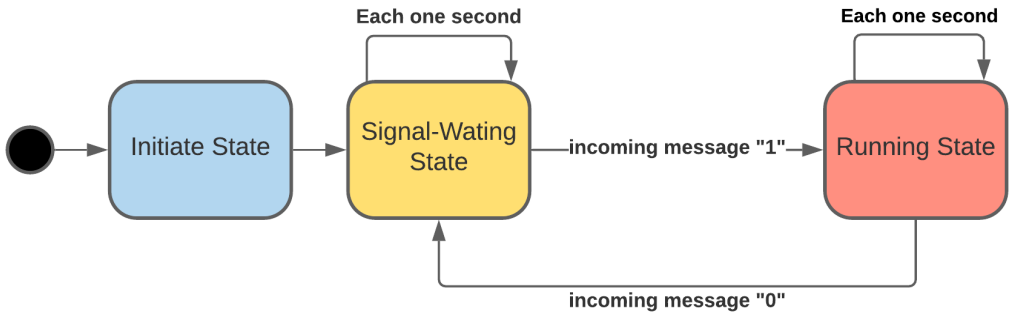

Figure 4.3: Diagram State of the Main Thread

4.2.2.3 Input and Output

- Input: Input set includes "1" and "0" where "1" is to initiate measuring and "0" is to stop
- Output: Encoded string "0x83/weight/0x81"

4.2.2.4 State Diagram Explanation

In this module, we have only one sensor to be read. Therefore, it is straightforwardly not used RTOS, and we need one main thread to read the weight sensor, receive, and respond to message to the Jetson Device

1. **Initial State**: The system initiates the connection with HX711 and establish connection with Jetson device.

2. **Signal-waiting State**: When there is an incoming signal "1" from the Jetson device for getting sensor value, the system go to the *Running State* to get the weight value.

3. **Running State**: In each loop, the system reads the weight value from the HX711 sensor, calculates the cumulative sum of sensor values, and finally increasing the number of records. We also calculate the cumulative time to check whether it equals one second or not. If the cumulative time equals one second:

   - We calculate the final value of the weight by dividing the sum of all previous data by the number of records.
   - Clear the cumulative sum of sensor values, the number of records, and the cumulative time to begin a new cycle.
   - Finally, we decode the weight value in the output format and send a response to the Jetson Device

   Otherwise, we continue to read new value, calculate cumulative sum and increase the number of records in the next loop. When the Jetson device has enough data, it sends a message "0" and makes the main thread go to the Signal-waiting state.

4.2.3 Capturing Heart Rate and SPO2 Value

For these two measurements, we will use a peripheral device (OSO2 SBPM14 - v2.5. It is more precise and carries some calculation algorithms for the microcontroller as calculating the oxygen saturation is quite delicate.

4.2.3.1 Capturing principle

The sensor will fire out a red beam; also, there will be a receiver to catch the infrared of the red ray on the sensors. Based on the strength of the receiving beam, the sensor will determine whether the beam is infrared from a heartbeat or not (or meet the density of hemoglobin in blood). For each second, the amount of beat receive can be used to calculate the heart rate:

$$beats\_per\_minute = \frac{60}{\frac{interval\_between\_heartbeats}{1000.0}} \tag{4.1}$$

And the oxygen saturation can be calculated through two equations:

1.
$$R = \frac{\frac{ACrms\_of\_Red}{DC\_of\_Red}}{\frac{ACrms\_of\_IR}{DC\_of\_IR}} \tag{4.2}$$

2.
$$SPO2 = 110\breve{}25R \tag{4.3}$$

4.2.3.2 Inter-module communication

The communication between the module and main processor is through an interface called USB HID (USB Human interface device) which very common for communication for USB peripheral (mouses, keyboards, etc.). When in measuring state, the device will send a continuous stream of data, the data contains information about patients' SPO2 and heart rate are in the following format: [79, TH_offset, Heart Rate, 0, x, x, TD0, TD1, TS_offset, 0, SPO2, 0 ,PI0, PI1].

In the sending sequence, we only care of the *Heart Rate* and *SPO2* value; the others are producer parameters for debugging and calibrating.

4.3 Device Server

Figure 4.4 is a package diagram of the entire server, and its structure is divided into four main packages. In the first package, the central package of the server (Main), which includes two main modules; one is for receiving, and another is responsible for responding to requests for each device. The second

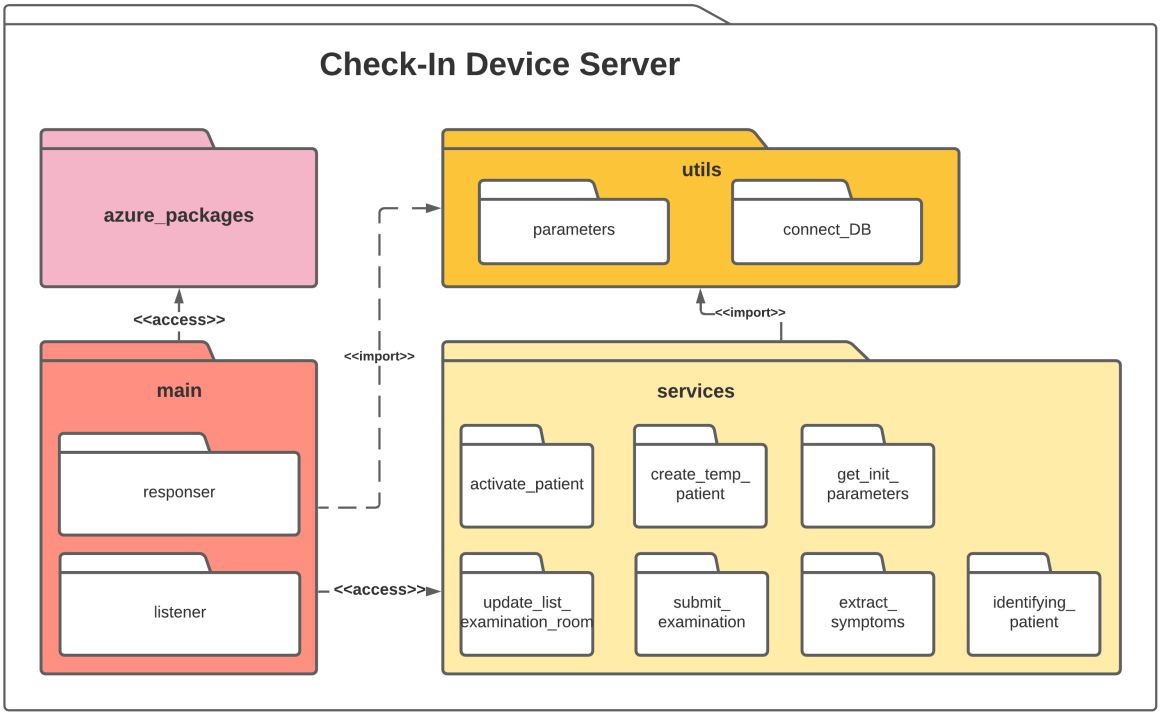

Figure 4.4: Package Diagram in Server Side

package contains all the modules to handle the services requested from the local device. The third package is the utils package, which contains the standard functions necessary for handling services, managing access, updating the database, and server operation. The last package is the necessary Azure module for device management and connection between devices.

4.3.1 Listening Requests

This module acts as the requests' listener of the server and runs in a separate thread from the main branch. When a service request arrives, this module will create a new thread to process the message.

The Check-in Device sends requests to the Device Server through the Event Hub, so each message sent follows a particular format, including *data_ body* and *properties*. Depending on each service, we will use or not use the two types above. If we do not use *data_ body*, we can leave it blank or not value it. By default, this value will be "". Same goes for *properties*, if not using the default JSON value will be .

Request Format

- **data_body** : is a string message
- **properties** : {
  "request_id" : "",
  "type_request" : "",
  "device_id" : "",
  "custom properties" : ""
  }

Figure 4.5: Request Format

- **request_id**: Each request from local devices is always assigned a request id so that it can be easily managed and handled locally (this will be discussed in more detail in the device section)
- **type_request**: Here, we have six types of requests for six services of the system.
  - Request Identification
  - Request Submit Examination
  - Request Active Patient
  - Request Get Symptoms
  - Request Get Initial Parameters
  - Request Update Examination Room
- **device_id**: each device when registered on IoT Hub will be assigned a *device_id*. That *device_id* is used to locate the device in the system through data in the database and send a response message back to the device.
- **custom_properties**: depending on each service, there will be individual properties.

4.3.1.1 Services Request Form

Get Initial Parameters

- **properties** : {
  "request_id" : "",
  "type_request" : REQUEST_GET_INIT_PARA,
  "device_id" : ""
  }

Figure 4.6: Request Format Get Initial Parameters

Extract Symptoms

- **data_body** : patient_description
- **properties** : {
  "request_id" : "",
  "type_request" : REQUEST_GET_SYMTOMS,
  "device_id" : ""
  }

Figure 4.7: Request Format Getting Symptoms of Patients

Identify Patients

- **data_body** : list_encoded_imgs
- **properties** : {
  "request_id" : "",
  "type_request" : REQUEST_IDENTIFICATION,
  "device_id" : "",
  "ssn" : (if this field is not set: "-1"; otherwise: ssn number)
  }

Figure 4.8: Request Format Identifying Patients

Submit Examination

- **data_body** : list_encoded_imgs_new_patient
- **properties** : {
  "request_id" : "",
  "type_request" : REQUEST_SUBMIT_EXAMINATION,
  "device_id" : "",
  "hospital_ID" : "",
  "building_code": "",
  "room_code" : "",
  "bmi" : "",
  "pulse" : "",
  "thermal" : "",
  "spo2" : "",
  "height" : "",
  "weight" : "",
  "patient_ID" : ""
  }

Figure 4.9: Request Format Submitting Examination

Activate Patient

```
• properties : {
    "request_id"    : "",
    "type_request" : REQUEST_ACTIVE_PATIENT,
    "device_id"     : "",
    "patient_ID"    : ""
  }
```

Figure 4.10: Request Format Activating Patient

Update List Examination

```
• properties : {
    "request_id"    : "",
    "type_request" : REQUEST_UPDATE_EXAM_ROOM,
    "device_id"     : "",
    "hospital_id"   : ""
  }
```

Figure 4.11: Request Format Updating Examination Room

4.3.2 Response Requests

After receiving and processing the requests of the Check-in Devices, the Device Server use this module to respond request. However, as mentioned in section 3.4.1, with the support from Azure Service, we have two ways to send messages back from the server to the devices, namely Direct Method and Cloud to Device message. Therefore, in the following section, we will compare two of them and choose the most suitable method for our system. Then below is the response format to the Check-in Device.

Response Format

```
• properties : {
    "request_id"     : "",
    "type_request"  : "",
    "device_id"      : "",
    "return_status" : "",
    “msg”            :
  }
```

Figure 4.12: Response Format

4.3.2.1 Services Response Form

Get Initial Parameters

```
• properties : {
    "request_id"                : "",
    "type_request"              : REQUEST_GET_INIT_PARA,
    "device_id"                 : "",
    "return_status"             : "",
    "list_examination_room"  : [
      {
        “department_id”       : ””,
        “department_name”  : ””,
        “building_code ”       : ””,
        “room_code”            : ””,
        “number_of_patient” : ””,
      },
      {
        ...
      },
      ...
    ]
  }
```

Figure 4.13: Response Format Get Initial Parameters

Submit Examination

```
• properties : {
    "request_id"     : "",
    "type_request"  : REQUEST_SUBMIT_EXAMINATION,
    "device_id"      : "",
    "return_status"  : "",
    "stt"                : ""
  }
```

Figure 4.14: Response Format Submitting Examination

Extract Symptoms

```
• properties : {
    "request_id"                : "",
    "type_request"              : REQUEST_GET_SYMTOMS,
    "device_id"                 : "",
    "return_status"             : "",
    "symptoms"                  : {
      “list_problem”        : [],
      “list_part_of_body”  : [],
    }
  }
```

Figure 4.15: Response Format Getting Symptoms of Patients

Identify Patients

- **properties** : {
  "request_id" : "",
  "type_request" : REQUEST_IDENTIFICATION,
  "device_id" : "",
  "return_status" : "",
  "patient_ID" : "",
  "name" : "",
  "birthday" : "",
  "phone" : "",
  "address" : ""
  }

Figure 4.16: Response Format Identifying Patients

Activate Patient

- **properties** : {
  "request_id" : "",
  "type_request" : REQUEST_ACTIVE_PATIENT,
  "device_id" : "",
  "return_status" : ""
  }

Figure 4.17: Response Format Activating Patient

Update List Examination

- **properties** : {
  "request_id" : "",
  "type_request" : REQUEST_UPDATE_EXAM_ROOM,
  "device_id" : "",
  "return_status" : ""
  }

Figure 4.18: Response Format Updating Examination Room

4.3.2.2 Sequence Diagram

The figure 4.19 presents the sequence diagram of the listening and responding request module in server-side

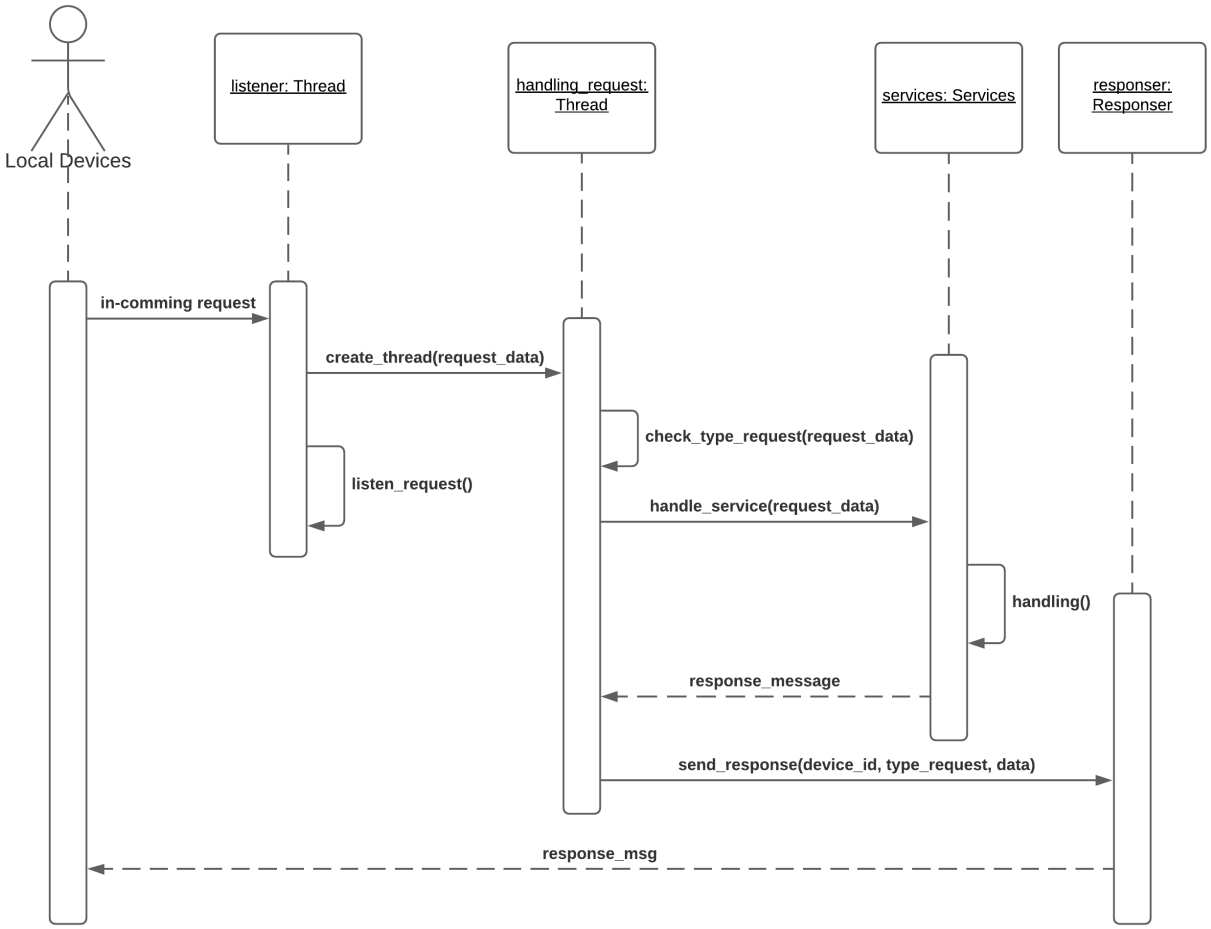

Figure 4.19: Sequence Diagram of Listening and Responding Request Service at Server Side

4.3.3 Get Initial Parameters

The server checks the device's status with the device id. If this device is not active, the server will reject this request and notify back to the device. Otherwise, the server retrieves a list of examination room at the hospital where this device is located, and send back to local device.

4.3.3.1 Sequence Diagram

The figure 4.20 presents the sequence diagram of Identifying Patient Service at Server Side

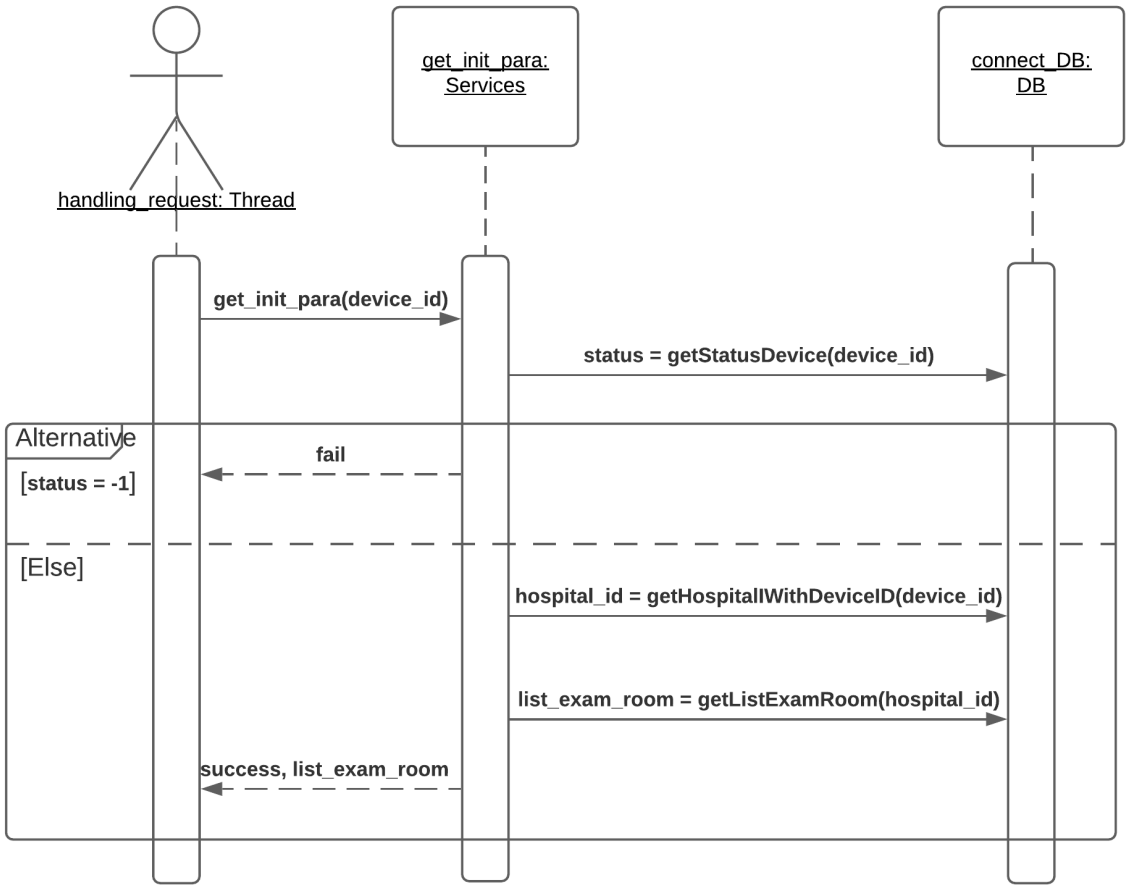

Figure 4.20: Sequence Diagram of Getting Init Parameters at Server Side

4.3.4 Extract symptoms

The goal of this service is that when the patient describes the symptom, chooses the department he wants to examine, or has a normal conversation with the assistant, the system must determine what this intent is. Once the intent is known, the system must filter out the entities(list problems and the list part of the body for the displaying symptom, and the department's name that the patient wants to visit is for choosing department). With displayed symptoms of the patient, our system has been trained to be able to identify a total of six intents, respectively:

Greet, Bye, Affirm, Deny, Display Symptoms, Choose Department

The first four intent is primarily used to support constructing the conversation with the patient. Such as, when the patient says Hello instead of displaying symptoms or choosing a medical examination, the system can

communicate with a patient based on intent extracted from that statement along with State Machine in Local Device. While the remaining two Intent, *display symptoms* and *choose department*, are used to predict the appropriate department for patients.

Suppose that we have the description symptoms of the patient as follow:

- "Xin chào Momo"
- "Tôi bị đau đầu, tiêu chảy và khó chịu trong bụng"
- "Tôi muốn khám Khoa Tai Mũi Họng"

The system will return *greet*, *display symptoms*, and *choose department* intent with the above three sentences, respectively. When the intent *display symptoms* and *choose department* are identified, the system has to extract their entities as described below:

Display symptoms intent:

- **list problems**: {"đau", "tiêu chảy", "khó chịu" }
- **list part of body**: {"đầu", "none", "bụng" }

Choose department intent: {"Tai Mũi Họng" }

The entity in choose department intent can easily be found by extracting the department name in it. However, we must find two lists (problems and part of body list) for the display symptoms intent, where two words at the same index in list problems and list part of the body combined to describe the list patients' symptoms.

For example, the sentence ""Tôi bị đau bụng, tiêu chảy và khó chịu trong người" above is extracted into 2 list

- **list problems**: {"đau", "tiêu chảy", "khó chịu" }
- **list part of body**: {"bụng", "none", "người" }

Then we combine 2 words at the same index of the above 2 lists and get a list of symptoms as our demand: {"đau đầu", "tiêu chảy", "khó chịu bụng" }

In some cases, the patient does not use an active voice to describe the symptoms as in the above example. Instead, they use passive voice as follow:

- "Bụng của tôi bị đau bụng và khó chịu"
- "Da của tôi bị rát, nổi mẩn và đầu của tôi rất ngứa"

As two sentences described above, the entities filtered by system are:

First sentence:

- **list part of body**: {"bụng", "bụng"}
- **list problems**: {"đau", "khó chịu" }

Second sentence:

- **list part of body**: {"da", "da", "đầu"}
- **list problems**: {"rát", "nổi mẩn", "ngứa" }

However, note that our system can only detect the entities of display symptoms intent when patients use either way, not both. Such as, when a patient describes symptoms as the following: *"Tôi bị đau tay, và chân tôi thấy hơi nhức"* the system will not be able to detect the whole entities in two list problems and list part of the body

4.3.4.1 Train Natural Language Understanding

Initiate The Project

First, we run the *rasa init* command to create the default project. Here, we consider two essential files that needed to train the assistant, **config.yml** and **data/nlu.yml**

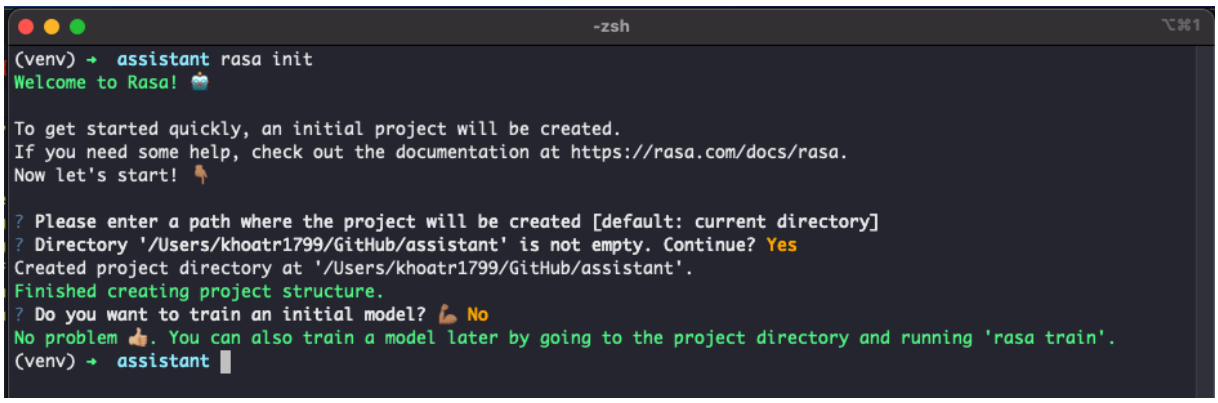

Figure 4.21: Initiate the Default Rasa Project

- **config.yml**: Store the configuration for training the assistant, including *pipeline components*, and *tokenizers*
- **nlu.yml**: Contains the training data for *intents classification* and *entities extraction*

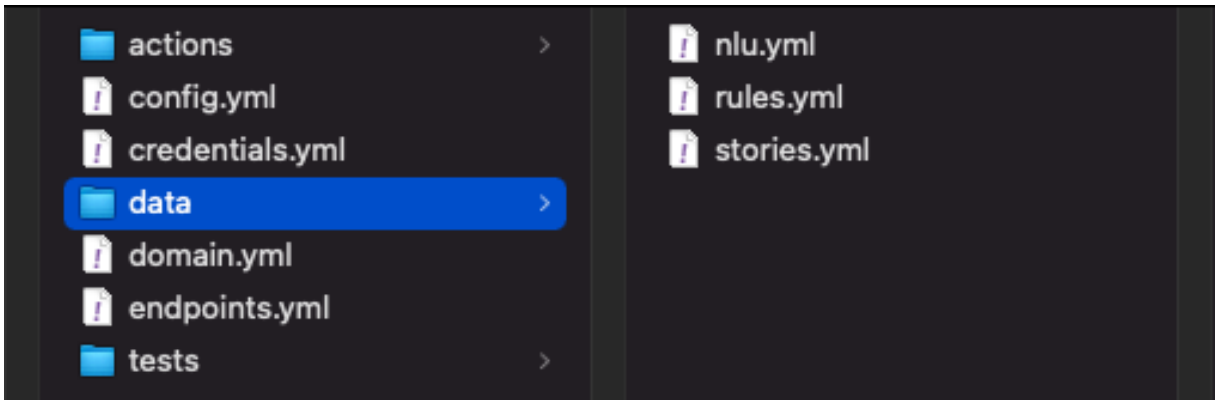

Figure 4.22: Project Directory

Prepare Training Data

Now, we will prepare the training data for the system, which is stored in the *data/nlu.yml* file. The following Table 4.1 represents intents and entities we want the model to classify and extract.

For the *intents ("greet", "goodbye", "affirm", "deny")*, we specify the common examples for training in the Figure 4.23.

Intent	Entities
- greet - bye - affirm - deny - display symptoms - choose department	- part of body - problem - subject

Table 4.1: Intents and Entities needed to be extracted by model

```
nlu:
- intent: greet
  examples: |
        - hey
        - hello
        - hi
        - xin chào
        - chào em
        - chào cảm
        - hé lô

- intent: goodbye
  examples: |
        - bái bai
        - bye bye
        - tạm biệt
        - hẹn gặp lại
        - bài bái
        - tôi muốn kết thúc

- intent: deny
  examples: |
        - no
        - không
        - không muốn
        - không cần
        - chưa
        - chưa biết
        - tôi không biết
        - tôi chưa biết
        - không biết
```

```
- intent: affirm
  examples: |
        - yes
        - đúng vậy
        - có
        - vâng
        - dĩ nhiên
         - dạ
        - dạ có
        - muốn
        - được
        - okay
        - được thôi
        - ố kề
        - rồi
        - biết
        - chắc chắn rồi
        - tôi biết
        - tôi đã biết rồi
```

Figure 4.23: Data Training For Intent Greet, Deny, Affirm, and Goodbye

As describe in the section 4.3.1, we list some of examples for training *display_ symptoms* and *choose_ department* intents. Furthermore, we provide lists of *look-up table* which contains vocabularies of three *entities ("subject", "part_ of_ body", "problem")*. All of the training data of each part above are describe in the Figure 4.24, Figure 4.25, and Figure 4.26, respectively.

intent: display_symptom
examples: |
-[tôi](**subject**) [sốt](**problem**)
-[tôi](**subject**) bị [sốt](**problem**)
-[tôi](**subject**) cảm thấy [mệt](**problem**)
-[nhức](**problem**) ở [đầu](**part_of_body**)
-[răng](**part_of_body**) [tôi](**subject**) bị [mẻ](**problem**)
-[mũi](**part_of_body**) [tôi](**subject**) bị [ngạt](**problem**)
-[tay](**part_of_body**) của [tôi](**subject**) [mỏi](**problem**)
-[tôi](**subject**) [đau](**problem**) ở [bụng](**part_of_body**)
-[tôi](**subject**) bị [cay](**problem**) ở [mắt](**part_of_body**)
-[da](**part_of_body**) của [tôi](**subject**) bị [rát](**problem**)
-[da](**part_of_body**) [tôi](**subject**) bị [nổi vẩy](**problem**)
-[tôi](**subject**) bị [ngạt](**problem**) ở [mũi](**part_of_body**)
-[tôi](**subject**) bị [đau](**problem**) ở [bụng](**part_of_body**)
-[mắt](**part_of_body**) của [tôi](**subject**) bị [cay](**problem**)
-[da](**part_of_body**) [tôi](**subject**) hay bị [nổi mụn](**problem**)
-[tôi](**subject**) cảm thấy [nhói](**problem**) ở [tai](**part_of_body**)
-[tôi](**subject**) hay bị [nổi vẩy](**problem**) ở [da](**part_of_body**)
-[tôi](**subject**) hay bị [ợ hơi](**problem**) và [chóng mặt](**problem**)
-[tôi](**subject**) cảm thấy bị [mỏi](**problem**) ở [người](**part_of_body**)
-[tôi](**subject**) cảm thấy hay bị [ợ hơi](**problem**) và [chóng mặt](**problem**)
-[bụng](**part_of_body**) của [tôi](**subject**) bị [đau](**problem**) và [khó chịu](**problem**)
-[tôi](**subject**) cảm thấy [người](**part_of_body**) của [tôi](**subject**) bị [mỏi](**problem**)
-[tôi](**subject**) cảm thấy [họng](**part_of_body**) của [tôi](**subject**) [không ổn](**problem**)
-[tôi](**subject**) cảm thấy [mắt](**part_of_body**) của [tôi](**subject**) [không được ổn](**problem**)
-[tôi](**subject**) cảm thấy [vai](**part_of_body**) của [tôi](**subject**) [không được okay](**problem**)
-[tôi](**subject**) cảm thấy hay bị [ợ hơi](**problem**) [đầy bụng](**problem**) và [khó chịu](**problem**)
-[tôi](**subject**) cảm thấy [răng](**part_of_body**) của [tôi](**subject**) [không được khoẻ](**problem**)
-[da](**part_of_body**) của [tôi](**subject**) bị [rát](**problem**) [ngứa](**problem**) và [khó chịu](**problem**)
-[tôi](**subject**) hay bị [mắc ói](**problem**), [mệt](**problem**) ở [người](**part_of_body**) và [sốt](**problem**)
-[tôi](**subject**) hay bị [đau](**problem**) ở [đầu](**part_of_body**) [chảy nước](**problem**) ở [mũi](**part_of_body**) và [sốt](**problem**) cao

Figure 4.24: Data Training For Intent Display Symptoms

intent: choose_department
examples: |
- [tôi](**subject**) muốn khám [khoa mắt](**department_name**)
- [tôi](**subject**) muốn khám [khoa da liễu](**department_name**)
- [tôi](**subject**) muốn khám [khoa thai sản](**department_name**)
- [tôi](**subject**) muốn khám [khoa sản](**department_name**)
- [tôi](**subject**) muốn khám [khoa thần kinh](**department_name**)
- [tôi](**subject**) muốn khám [khoa tai mũi họng](**department_name**)
- [tôi](**subject**) muốn khám [khoa răng hàm mặt](**department_name**)
- [tôi](**subject**) muốn khám [khoa nội thận](**department_name**)
- [tôi](**subject**) muốn khám [khoa xương khớp](**department_name**)
- [tôi](**subject**) muốn khám [khoa tổng quát](**department_name**)
- [tôi](**subject**) muốn khám [khoa nội tổng quát](**department_name**)
- [tôi](**subject**) muốn khám ở [khoa mắt](**department_name**)
- [tôi](**subject**) muốn khám ở [khoa da liễu](**department_name**)
- [tôi](**subject**) muốn khám ở [khoa thai sản](**department_name**)
- [tôi](**subject**) muốn khám ở [khoa sản](**department_name**)
- [tôi](**subject**) muốn khám ở [khoa thần kinh](**department_name**)
- [tôi](**subject**) muốn khám ở [khoa tai mũi họng](**department_name**)
- [tôi](**subject**) muốn khám ở [khoa răng hàm mặt](**department_name**)
- [tôi](**subject**) muốn khám ở [khoa nội thận](**department_name**)
- [tôi](**subject**) muốn khám ở [khoa xương khớp](**department_name**)
- [tôi](**subject**) muốn khám ở [khoa tổng quát](**department_name**)
- [tôi](**subject**) muốn khám ở [khoa nội tổng quát](**department_name**)
- [tôi](**subject**) muốn khám ở [tai mũi họng](**department_name**)
- [tôi](**subject**) muốn khám ở [răng hàm mặt](**department_name**)
- [tôi](**subject**) muốn khám ở [nội thận](**department_name**)
- [tôi](**subject**) muốn khám ở [xương khớp](**department_name**)
- [tôi](**subject**) muốn khám ở [tổng quát](**department_name**)
- [tôi](**subject**) muốn khám ở [nội tổng quát](**department_name**)
- [tôi](**subject**) muốn khám [tai mũi họng](**department_name**)
- [tôi](**subject**) muốn khám [răng hàm mặt](**department_name**)
- [tôi](**subject**) muốn khám [nội thận](**department_name**)
- [tôi](**subject**) muốn khám [xương khớp](**department_name**)
- [tôi](**subject**) muốn khám [tổng quát](**department_name**)
- [tôi](**subject**) muốn khám [nội tổng quát](**department_name**)
- [tôi](**subject**) muốn khám ở [mắt](**department_name**)
- [tôi](**subject**) muốn khám ở [da liễu](**department_name**)
- [tôi](**subject**) muốn khám ở [thai sản](**department_name**)
- [tôi](**subject**) muốn khám ở [thần kinh](**department_name**)
- [tôi](**subject**) muốn khám [mắt](**department_name**)
- [tôi](**subject**) muốn khám [da liễu](**department_name**)
- [tôi](**subject**) muốn khám [thai sản](**department_name**)
- [tôi](**subject**) muốn khám [phụ khoa](**department_name**)
- [tôi](**subject**) muốn khám [thai](**department_name**)
- [tôi](**subject**) muốn khám [thần kinh](**department_name**)
- [khoa mắt](**department_name**)
- [khoa da liễu](**department_name**)
- [khoa thai sản](**department_name**)
- [khoa thần kinh](**department_name**)
- [khoa tai mũi họng](**department_name**)
- [khoa răng hàm mặt](**department_name**)
- [khoa nội thận](**department_name**)
- [khoa xương khớp](**department_name**)
- [khoa tổng quát](**department_name**)
- [khoa nội tổng quát](**department_name**)
- [mắt](**department_name**)
- [da liễu](**department_name**)
- [thai sản](**department_name**)
- [thần kinh](**department_name**)
- [tai mũi họng](**department_name**)
- [răng hàm mặt](**department_name**)
- [nội thận](**department_name**)
- [xương khớp](**department_name**)
- [tổng quát](**department_name**)
- [nội tổng quát](**department_name**)

Figure 4.25: Data Training For Intent Choose Department

```
nlu:
- lookup: problem
  examples: |
    - bong
    - buồn nôn
    - bón
    - chuột rút
    - chua
    - chóng
    - chảy nước
    - chướng
    - co giật
    - co rút
    - căng thẳng
    - dị ứng
    - gãy
    - quên
    - hạch
    - ho
    - hắt xì
    - kiệt sức
    - khó chịu
    - khó tiêu
    - không bình thường
    - không khoẻ
    - không okay
    - không được okay
    - không được bình thường
    - không được khoẻ
    - không được ổn
    - không ổn
    - loét
    - lở
    - lở loét
    - mắc ói
    - mệt
    - mệt mỏi
    - mỏi
    - nôn
    - nổi ban
    - nổi chấm đỏ
    - nổi mẩn
    - nổi mề đay
    - nổi mụn
    - ói
    - phát ban
    - phù
    - rách
    - rát
    - stress
    - sưng
    - sốt
    - sổ
    - táo bón
    - thở
    - tiêu chảy
    - tê
    - viêm
    - vướng
    - vảy nến
    - ói
    - ù
    - uể oải
    - xót
    - đau
    - đầy hơi
    - động
    - đắng
    - đi cầu
    - đi ngoài
    - đừ
    - ợ
    - nấc cụt
    - ngủ
    - ngứa
    - nhói
    - nhức

- lookup: subject
  examples: |
    - tôi
    - mình
    - anh
    - chị
    - cô
    - bác
    - chú
- lookup: part_of_body
  examples: |
    - mặt
    - miệng
    - mồm
    - cằm
    - cổ
    - vai
    - da
    - tay
    - khớp
    - khuỷu tay
    - cẳng tay
    - nách
    - lưng
    - ngực
    - thắt lưng
    - eo
    - bụng
    - bao tử
    - dạ dày
    - đít
    - mông
    - hông
    - gân
    - chân
    - bàn chân
    - cẳng chân
    - đùi
    - bắp
    - đầu gối
    - ngón tay
    - móng tay
    - ngón trỏ
    - ngón giữa
    - ngón đeo nhẫn
    - ngón út
    - tóc
    - đầu
    - trán
    - tai
    - má
    - mũi
    - hàm
    - râu
    - ria mép
    - lưỡi
    - răng
    - nướu
    - lợi
    - môi
    - mắt
    - lông mày
    - mí mắt
    - mí
    - mi mắt
    - mi
    - lông mi
    - mống mắt
    - con ngươi
    - gót chân
    - ngón cái
    - ngón chân
    - móng
    - móng chân
    - não
    - tủy sống
    - họng
    - cuống họng
    - khí quản
    - thực quản
    - cơ
    - phổi
    - tim
    - gan
    - ruột
    - tĩnh mạch
    - động mạch
    - cật
    - tuyến tụy
    - bọng đái
    - người
    - thần kinh
    - dương vật
    - âm đạo
    - bầu
    - thai
    - vú
    - dú
```

Figure 4.26: Data Training For Intent Display Symptoms

Create Configure File for Training

At the *config.yml* file, we modify the content as the Figure 4.27 below. We use the *VietnameseTokenizer* pipeline as our Tokenizer. For extracting the entities, we use both pipeline *RegexEntityExtractor* and *CRFEntityExtractor*.

- The *CRFEntityExtractor* is used to extract *exdepartment_name* in the intent *choose_department*. While *RegexEntityExtractor* is used to extract the *part_of_body* and problem in the intent *display_symptoms*
- However, as the *RegexEntityExtractor* is one of the features of the pipeline

RegexFeaturizer, we have to include it in the configure file.

Finally, we use the *DIETClassifier* pipeline to classifying the patients' intent.

```
language: "vi"

pipeline:
- name: "VietnameseTokenizer"
- name: "RegexFeaturizer"
- name: "RegexEntityExtractor"
- name: "CRFEntityExtractor"
- name: "CountVectorsFeaturizer"
- name: "CountVectorsFeaturizer"
  analyzer: "char_wb"
  min_ngram: 1
  max_ngram: 4
- name: "DIETClassifier"
  epochs: 50
  ranking_length: 5
  batch_strategy: sequence
  constrain_similarities: True
  entity_recognition: False
```

Figure 4.27: Configure Assistant

Train and Run The Assistant

After preparing the appropriate data and configuration file for the system, we start training the assistant. Then, with the output of the training step, we run the service by deploying the model at the Device Server with the URL equal "http://localhost:5002/api". Whenever there is a *extract_symptoms* request, the Device Server runs an HTTP request to achieve the response as a JSON at the above URL.

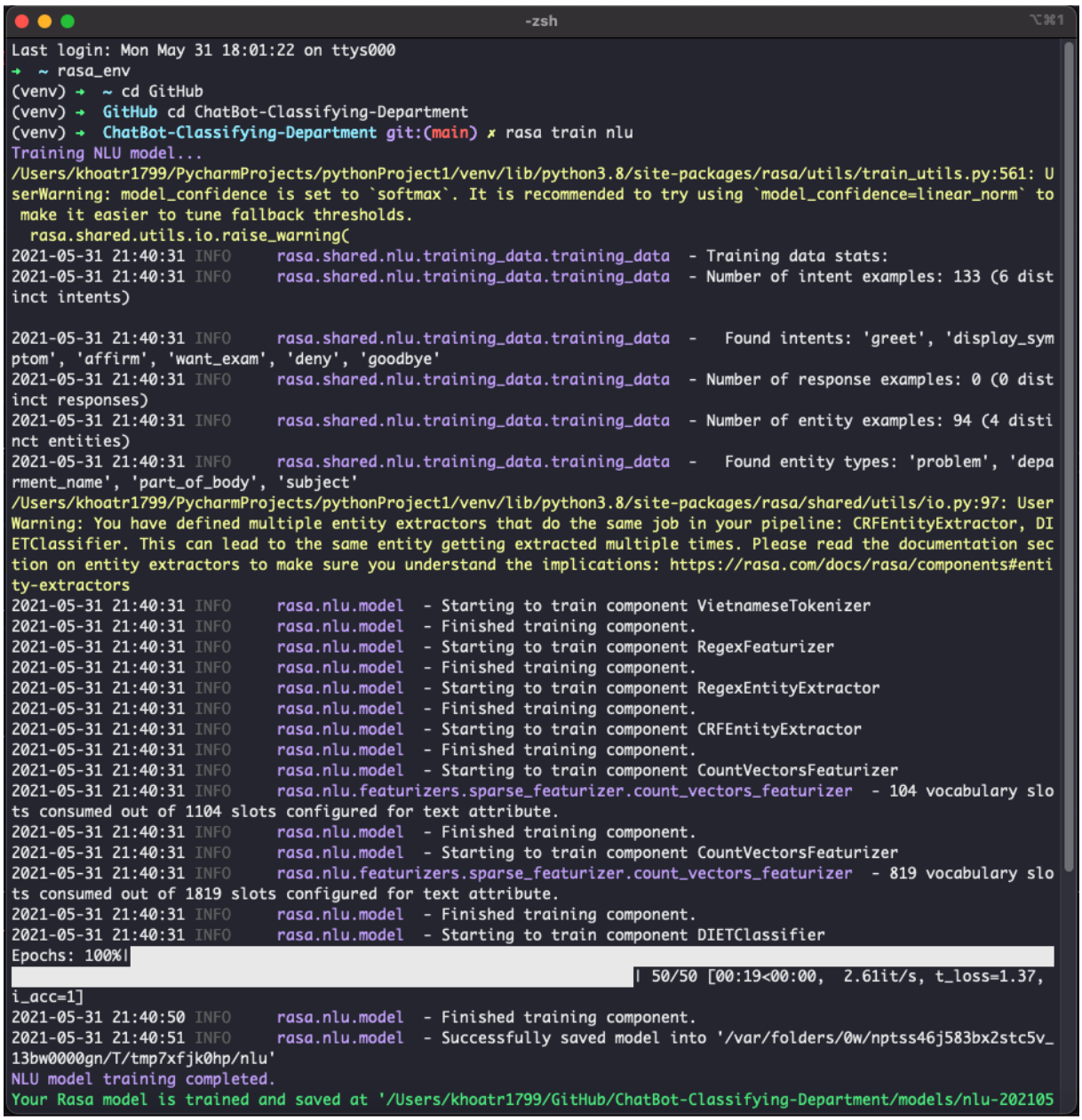

Figure 4.28: Training Assistant

Figure 4.29: Deploying Assistant

The below example displays the output of on patient's displayed sentence

```
{
    'text':
        'tôi bị đau đầu và mắc ói',

    'intent': {
        'id': 6196731624939929846,
        'name': 'display_symptom',
        'confidence': 0.9999973177909851
    },

    'entities': [
        {'entity': 'subject',         'start': 0,   'end': 3,   'value': 'tôi',       'extractor': 'RegexEntityExtractor'},
        {'entity': 'part_of_body', 'start': 11, 'end': 14, 'value': 'đầu',      'extractor': 'RegexEntityExtractor'},
        {'entity': 'problem',         'start': 7,   'end': 10, 'value': 'đau',      'extractor': 'RegexEntityExtractor'},
        {'entity': 'problem',         'start': 18, 'end': 24, 'value': 'mắc ói', 'extractor': 'RegexEntityExtractor'},
        {'entity': 'subject',          'start': 0,   'end': 3,   'confidence_entity': 0.9302337075324199, 'value': 'tôi',
        'extractor': 'CRFEntityExtractor'},
         {'entity': 'problem',        'start': 7,    'end': 14, 'confidence_entity': 0.9419532420402574, 'value': 'đau đầu',
        'extractor': 'CRFEntityExtractor'},
        {'entity': 'problem',         'start': 18,  'end': 24, 'confidence_entity': 0.862440663303628,   'value': 'mắc ói',
        'extractor': 'CRFEntityExtractor'}
    ],

    'intent_ranking': [
        {'id': 6196731624939929846,  'name': 'display_symptom',  'confidence': 0.9999973177909851},
        {'id': -5156847405924261479, 'name': 'goodbye',              'confidence': 1.3878926665711333e-06},
        {'id': -1675227833265857852, 'name': 'greet',                'confidence': 6.910915431035392e-07},
        {'id': 8209249682598640028,   'name': 'affirm',               'confidence': 4.97943119626143e-07},
        {'id': 7166095557948641121,   'name': 'deny',                 'confidence': 1.0307986997304397e-07}
    ]
}
```


Figure 4.30: Example Of Extracting Sympoms in Patiens' Displayed Sentence

- **text**: The input as well as patient's displayed sentence.
- **intent**: The final intent that classified by the system when inferring to the model.
- **entities**: A list of all detected entities in the sentence. Each element is the JSON file containing
  - The type of entity
  - Start and end position of the entity in the sentence
  - The value of this entity
  - And the pipeline used to detect it
- **intent_ranking**: Similar to the **entities**, this is a list that ranks all the intent might be classified as the intent of this sentence ordering in the decrease of each confidence value.

4.3.4.2 Sequence Diagram

The figure 4.31 presents the sequence diagram of Extracting Patients' Symptoms Service at Server Side

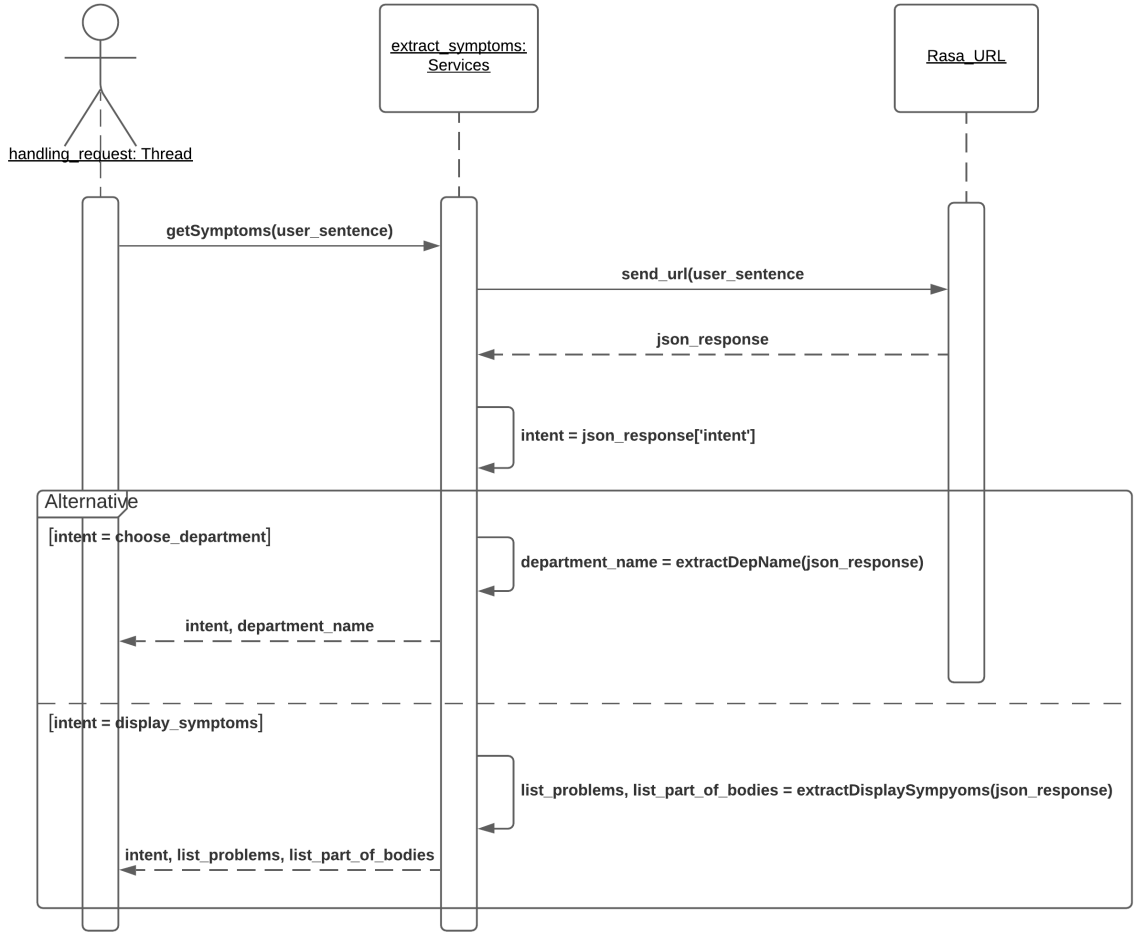

Figure 4.31: Sequence Diagram of Extracting Patients' Symptoms at Server Side

4.3.5 Identify patient

When we receive an array containing 128-float64 elements in string format at *data_ body* in the request message, we will predict the patient id of that person. Then, with the patient id just obtained, we will query the database to get the name, age, phone number, and address.

4.3.5.1 Workflow

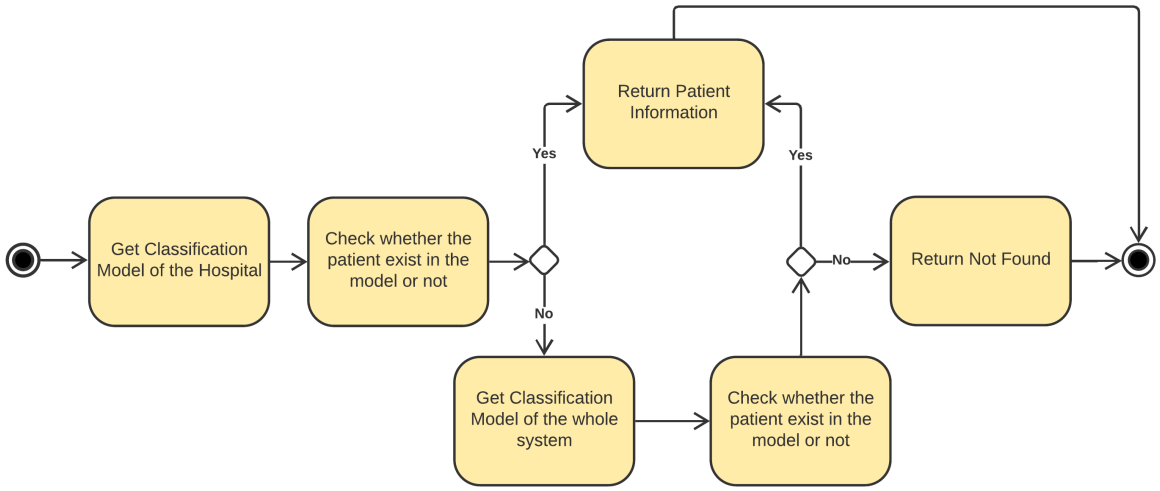

Figure 4.32: Identification Patient Activity Diagram

The image above is the overview flow of this module. With the IoT system, the main characteristic is its scalability. It means that the system can adapt to the growing number of patients but still perform the Recognition task without degrading the accuracy. Therefore, we divide the Identification task into two layers, as displayed in Figure 4.32, to overcome these drawbacks.

At the first layer, we identify the patients in a group of patients in the same hospital, where the hospital is the one that the patients currently check-in. If we find the patients in this cluster, we return the patient's information for the Check-in Device. Otherwise, we can think that the current hospital is not the one that the patient registered or the patient register in another hospital and now that take the examination at the new one. Therefore, we have to identify the patient in a larger group (the whole hospital) to find them. However, to handle this case, we enable the user to register in more than one hospital. In that case, their data will be in all the hospitals that they have registered, and the system does not need to move the next filter layer.

Now, we identify the patient in the whole system and return whether we find they or not. If yes, return their information. Otherwise, return not found.

Detail Workflow

The input of this module is the id of the current hospital and an array containing patient faces which each element has been decoded to 128-D vector float64. To be more specific, the shape of our input is nx128, where n is the number of decoded images. However, to send requests from local devices to the server straightforwardly, we have to decode that array into a string at the Check-in Devices before requesting the Device Server. That string has the format below:

$$"a_0/b_0/c_0/.../d_0\ a_1/b_1/c_1/.../d_1\ a_2/b_2/c_2/.../d_2...\ a_n/b_n/c_n/.../d_n"$$

Now, we convert it back to the original array of 128-D vector float64 by:

- Split that string into n strings with the cut character " ".
- For each element after being split, represent a 128-dimensional float64 vector. We loop each of those elements and continue to slip with the split character "/" to get list of string elements with 128 elements.
- Finally, we loop each list above, convert each value into float64, and append them to a vector with 128 float64 values.
- This process repeats until we have n array of 128-D vector float64.
- Below is the Pseudo Code for decoding list_encoded_imgs

```
def decoding_list_encoded_imgs(input):
    list_encoded_imgs = []
    splited_list_encoded_imgs = input.split(' ')
    for i in splited_list_encoded_imgs:
        if i != '':
            encoded_img = []
            splited_encoded_img = i.split('/')
            for j in splited_encoded_img:
                if j != '':
                    float_value = np.float64(j)
                    encoded_img.append(float_value)

            encoded_img = np.array(encoded_img).reshape(1,-1)
```

```
            list_encoded_imgs.append(encoded_img)
    return list_encoded_imgs

```

Listing 4.1: Pseudo Code for decoding list_encoded_imgs

After pre-processing the input as we need, we start to handle the workflow of this module. First, we check whether the SSN equals "-1" or not. If Yes:

- Based on the hospital id in the input, we load the K-NN Model, which contains the encoded data of all patients registered at this hospital.
- For each 128-D vector float64 in an array above, we infer it into the K-NN Model to get the result, including the predicted patient id and its Euclidean Distance to the vector. Only the patient id with its distance less than a set DISTANCE_IDENTIFY_THRESHOLD is appended to the selected list.
- After scanning the array, we choose the patient id with the most appearances, and its appearance is larger than NUM_PREDICTED_THRESHOLD as our predicted patient.
- If we have the suitable patient id, we retrieve the information based on the previous id and return it to local devices. Otherwise, we move to check the patient in the K-NN model of the whole system as displayed in the section workflow.

If the SSN is set:

- First, we use the SNN to get the patient id, then using this id to check whether the patient exists in the Device Server or not. If not, we load their encoded images in the Database Server and store them in the Device Server. Otherwise, we move to the next step.
- **Note**: The reason that we do not have the data of the patient in the Server Device is that the Device Server has some problems in the activating patient state. Therefore, it will wait for the next day to load all the fail training data in the Database and train again. The details of this note are described more clearly in section 4.3.7.

- We combine n 128-D vector float64 in the above array with 5 encoded images of the patient with SSN stored in our server. It means that we have n x 5 pair values. Then, for each pair, we calculate the Euclidean distance between them.
- Next, we count the number of distance which is larger than specific DISTANCE_VERIFY_THRESHOLD. If the count number is larger than our defined value, we verify this patient successfully. Otherwise, we reject this request.
- Finally, we retrieve the patient's information based on that predicted patient id and return it to local devices.

Problems With The Old Patient

This problem comes from when the patient registers their face data above ten years ago. They continue to use the system until the system recognizes their face under a specific threshold. In this case, we enable the patient to verify their identification by the SNN as displayed above. If the patient passes the defined threshold, we will update their encoded image in the Device Server.

For example, ten images sent to the system according to the SSN are used to verify the patient. After a verification step, we find the pair with the smallest distance among n x 5 pair values above. Then we take the input encoded image as the replacement for the oldest encoded image in the Device Server. The encoded images in the database are stored with their ID and updated date. Next, we update the database and train again the K-NN Model.

4.3.6 Submit Examination

First, we extract important information from request data, including patien_ID, examination room data(hospital_ID, building_code, and room_code), and sensor information(BMI, pulse, spo2, thermal, height, and weight). Based on patient_ID, we check whether it is a new patient or an old patient. If patient_ID is -1, this is a new patient; otherwise, this is an old patient with

their id.

If the patient is new, we create their temporary information and save their encoded images (*list_ encoded_ imgs_ new_ patient* in the data_body of the requested data) in the database(we will describe this step in section 4.3.7). Then we insert the sensor information into the database to get sensor_ID as the return. Finally, we insert the sensor_ID, patient_ID, and examination data into the examination queue and get the STT number for the patient.

4.3.6.1 Sequence Diagram

The figure 4.33 presents the sequence diagram of Submitting Examination Service at Server Side

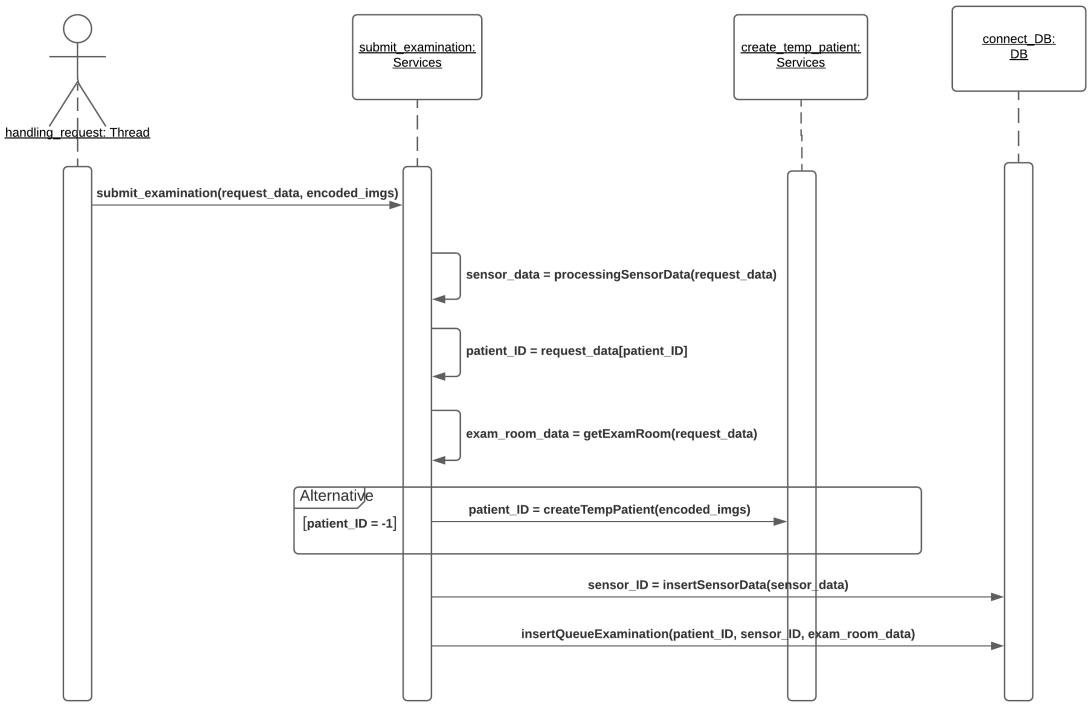

Figure 4.33: Sequence Diagram of Submitting Examination Service at Server Side

4.3.7 Create Temporary Patient

This module is the sub-module of submitting an examination module for a new patient. In this module, we also introduce additional information when we register new patients, and it is described below:

Each entity patient on the database has a field named valid flag. This field is used to check the status of the patient with 3 values.

- **'0'**: Temporary patient
- **'1'**: Ready to train
- **'2'**: Valid patient

As described in section 4.3.6, when a new patient had taken the examination and submitted it by the Check-In Device, the Device Server will create a temporary patient with a valid flag is '0'. Then, after the doctor submits the examination with the new patient's information, these values are updated on the database along with the valid flag '1'. This flag means that the patient is ready to train by the Recognition system. Next, the Web Server will send a request to Activate the Patient for the Device Server. After doing the training phase, if it is successful, the valid flag of the patient will be changed to '2'. Otherwise, it is still '1', and the Device Server will check the patients with a valid flag equals to' 1', load their encoded images on the database, and train again at the beginning of a new day.

Now, we are going to the detail of this module. After examining a new patient and submitting it to the Device Server, the Device Server creates dummy patient information with the valid flag equal to 0 and store it in the database. Then, we split the string encoded images of the new patient into a list of five elements while each element is the image of the patient decoded as a string data type. Finally, insert these encoded images into the database.

4.3.7.1 Sequence Diagram

The figure 4.34 presents the sequence diagram of Creating Temporary Patient Service at Server Side

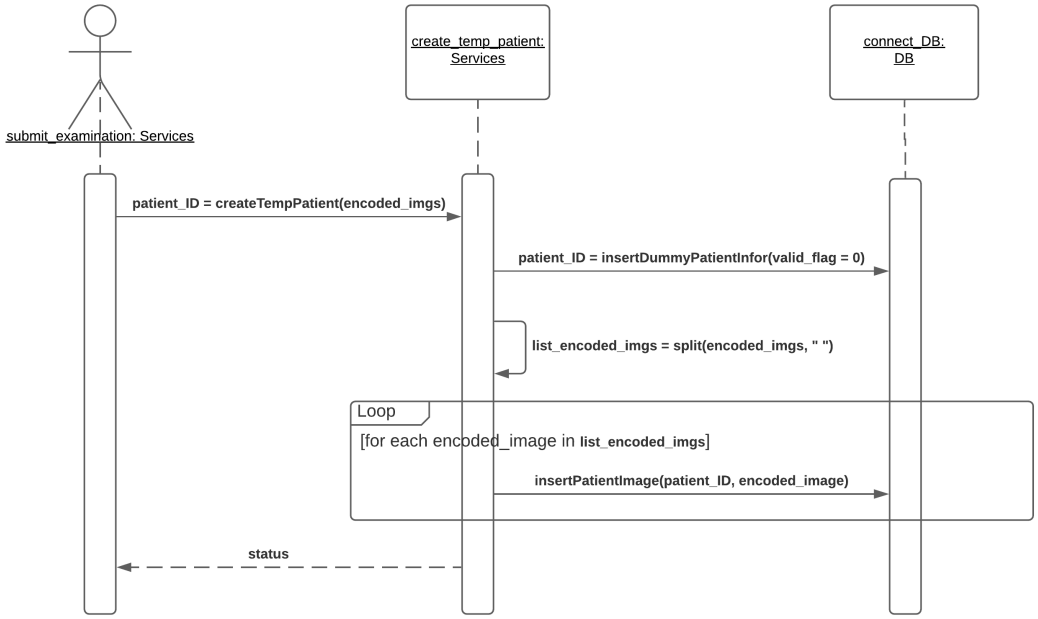

Figure 4.34: Sequence Diagram of Creating Temporary Patient Service at Server Side

4.3.8 Activate patient

First, we check whether the patient is active or not. If the patient is already in the classification model, we pass. Otherwise, we load the list encoded images of the patient in the Database.

Currently, the data type of each element in the above list is a string; we need to convert it into a list of encoded images (an array of 128-d vectors). We then pass this list and patient_ID into classification and train our model again.

If the training state is successful, we update the valid flag of the patient in the database into '2' (valid). Otherwise, we will pass.

4.3.8.1 Sequence Diagram

The figure 4.35 presents the sequence diagram of Activating Patient Service at Server Side

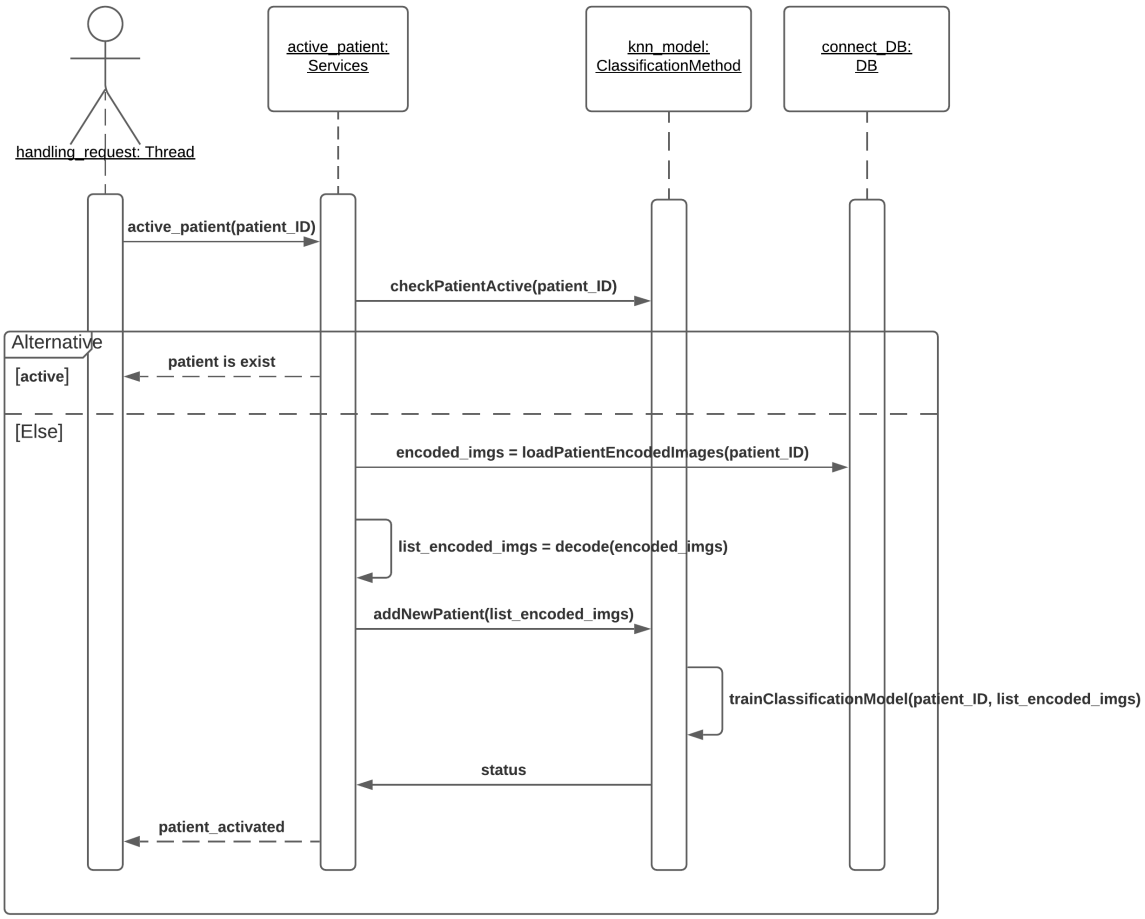

Figure 4.35: Sequence Diagram of Activating Patient Service at Server Side

4.3.9 Update list examination room

Firstly, we load the list id of devices that are located in the hospital with hospital_id.

Second, load new examination rooms in the database according to hospital_id. For each device in the above hospital, we use iothub_manager in azure to send the new examination rooms to them.

4.3.9.1 Sequence Diagram

The figure 4.36 presents the sequence diagram of Updating New Examination Rooms Service at Server Side

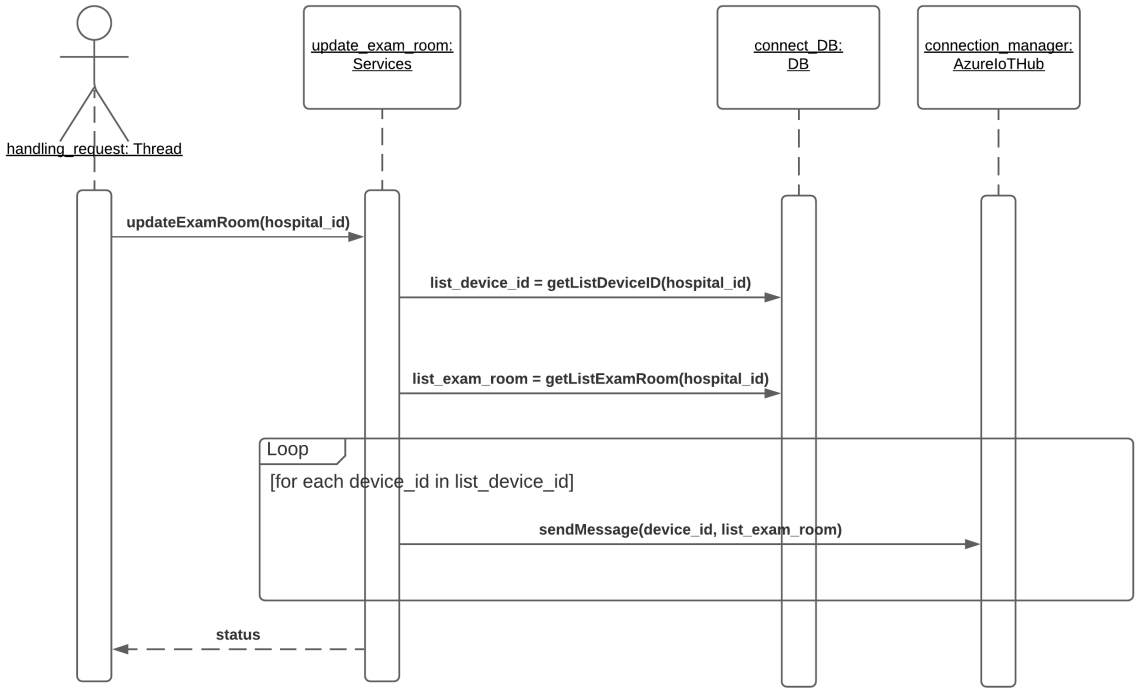

Figure 4.36: Sequence Diagram of Updating New Examination Rooms at Server Side

4.4 Check-In Device

This system is divided into three parts that run separately on its thread to serve all the functionalities and communicate with each other through global variables. These processes include *User Interface*, *Communication with Device Server*, and *Main processes* where:

User Interface: We use QT Creator to create a User Interface that displays the content, instruction, and patient information according to each state of the Local Device. Furthermore, we also provide the patient a way to interact with the Local Device through a touch screen LCD. The interface of the system for each state will be displayed in each module below.

Communication with Device Server: This process includes *Listener* module, which runs independently to receive the message from the Device Server.

Main processes: The final process is the *Main Process* of this Local

Device. It runs on the main thread as a state machine to handle four phases, *Initiating System*, *Identifying Patient*, *Collecting Sensors Data and Choosing Examination Department*, and *Submitting the Examination*. In each phase, we have additional modules that are used to support it and are displayed below:

- **Initiating System System**
- **Identifying Patient**
  - Identifying Patient
  - Count Number of Sample
  - Capturing Face Pose of New Patient
- **Collecting Sensors Data and Choosing Examination Department**
  - Collecting Biologic Parameters module
  - Choosing Department
  - Recommending Department With Assistant
- **Submitting the Examination**

In the following section, we will present the introduction and implementation of each module by giving the *Sequence Diagram*, *Workflow* used as an additional explanation for the diagram. And finally is the *Application Screen*.

Besides, in some modules, we also provide the extra section that supports this module. All the information will be displayed below. However, before going into the detailed implementation of the local device, we firstly introduce the package diagram and the whole state of our system in figure 4.37 and figure 4.38, respectively.

Package Diagram

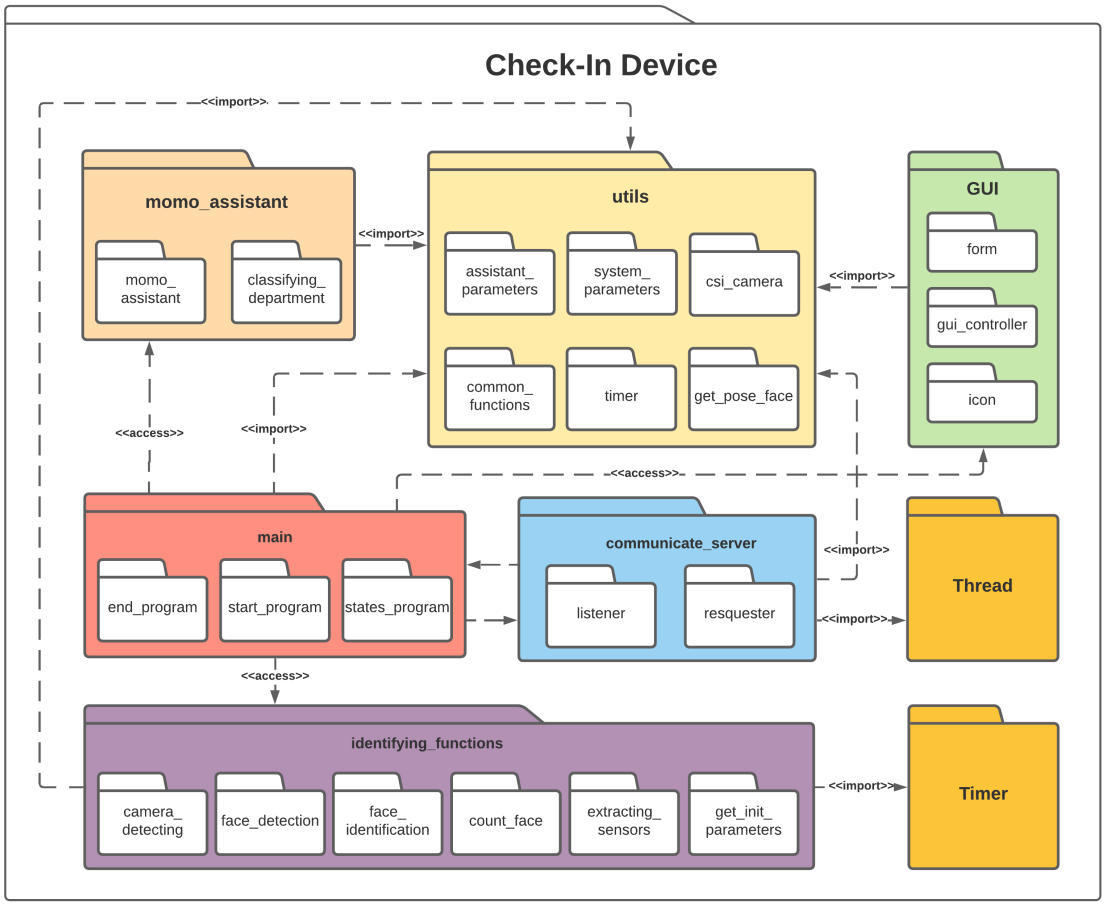

Figure 4.37: Package Diagram in Check-In System

State Diagram

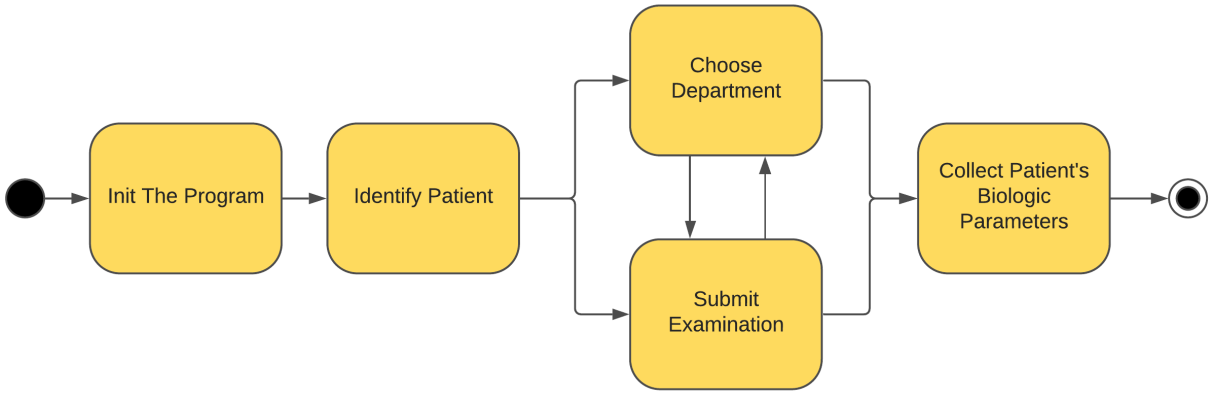

Figure 4.38: State Diagram in Check-In System

4.4.1 Listener

To utilize the whole system's performance, we create a new thread to handle listening responses or messages from the Device Server. This thread runs separately from the main thread. It processes messages asynchronously, which means that it can handle two incoming messages concurrently with the appropriate mutex lock to prevent conflict data.

When there are messages or responses from the Device Server:

- We first check type_request and request_id. type_request helps us know what request is this response, and request_id is used to match whether this is a currently waiting response from the Device Server at the Local Device.
- For example, when we send a request for validating a patient, this request is specified with request_id and type_request.
- At the same time, we store the request_id at the Local Device. However, when the previous request still does not respond or reach the timeout, the Local Device re-sent this request with a different id and updated the new request_id at the Local Device simultaneously.
- Therefore, when receiving responses from the Device Server, we only check whether this request_id is matched with the current request_id stored at the Local Device; if it matches, we process this response messages; otherwise, we discard it.

If this message is matched with the current request_id, we check type_request and pass this message to the appropriate module. At the Local Device, we have four modules to handle messages as well as responses from the Device Server.

- **Request Validation**:
  - We check the return status of the response. For validating patient, we have three type of status, **valid**, **invalid**, and **wear_mask**.
  - If the return status is **valid**, we update the global variable patient information.

- If the return status is **invalid**, we increase the global variable missing face used to check whether the patient is the first incoming patient or not. It will be described in more detail in section 4.4.4.
- For **wear_mask** status, it is used to ask the patient to wear off the mask to validate correctly

- **Request Submit Examination**: We check whether the return status of this response. If this request has no error on the Device Server, we extract the STT of the patient.
- **Request Get Symptoms**: Similarly to response of Sumit Examination request, we extract the two lists, *list_problem* and *list_part-_of_body* for recommending the appropriate department.
- **Request Get Initial Parameters**: Get initial parameters for the operation of Local Devices.
- **Request Update Examination Rooms**: Update new examination rooms of the current hospital that this device is located.

After handling the messages, turn on flag *has_server_message* to notify the main thread.

4.4.2 Requester

In this section, we provide four modules for sending a request to the Device Server.

Request Get Initial Parameters:

```
• properties : {
    "request_id"    : "",
    "type_request" : REQUEST_GET_INIT_PARA,
    "device_id"     : ""
  }
```

Figure 4.39: Request Format Identifying Patient

Request Get Identification:

- **data_body** : list_encoded_imgs
- **properties** : {
  "request_id" : "",
  "type_request" : REQUEST_IDENTIFICATION,
  "device_id" : "",
  "ssn" : (if this field is not set: "-1"; otherwise: ssn number)
  }

Figure 4.40: Request Format Identifying Patients

Request Get Submit Examination:

- **data_body** : list_encoded_imgs_new_patient
- **properties** : {
  "request_id" : "",
  "type_request" : REQUEST_SUBMIT_EXAMINATION,
  "device_id" : "",
  "hospital_ID" : "",
  "building_code": "",
  "room_code" : "",
  "bmi" : "",
  "pulse" : "",
  "thermal" : "",
  "spo2" : "",
  "height" : "",
  "weight" : "",
  "patient_ID" : ""
  }

Figure 4.41: Request Format Submitting Examination

Request Get Symptoms:

- **data_body** : patient_description
- **properties** : {
  "request_id" : "",
  "type_request" : REQUEST_GET_SYMTOMS,
  "device_id" : ""
  }

Figure 4.42: Request Format Getting Symptoms of Patients

4.4.3 Initiating System

When plugged in with the supplier power, the system begins with the Initiating System Screen. At this screen, we connect the wifi by supplying *SSID* and *password* to start the entire system. After connected with the Internet, the system sends the first request to the server to load the initial parameters for its operation. For a while, if the system does not respond from the Device Server, the timer is triggered and re-sent the request until it has the response. When receiving the initial parameters, the system moves to the Identifying patient screen.

4.4.3.1 Sequence Diagram

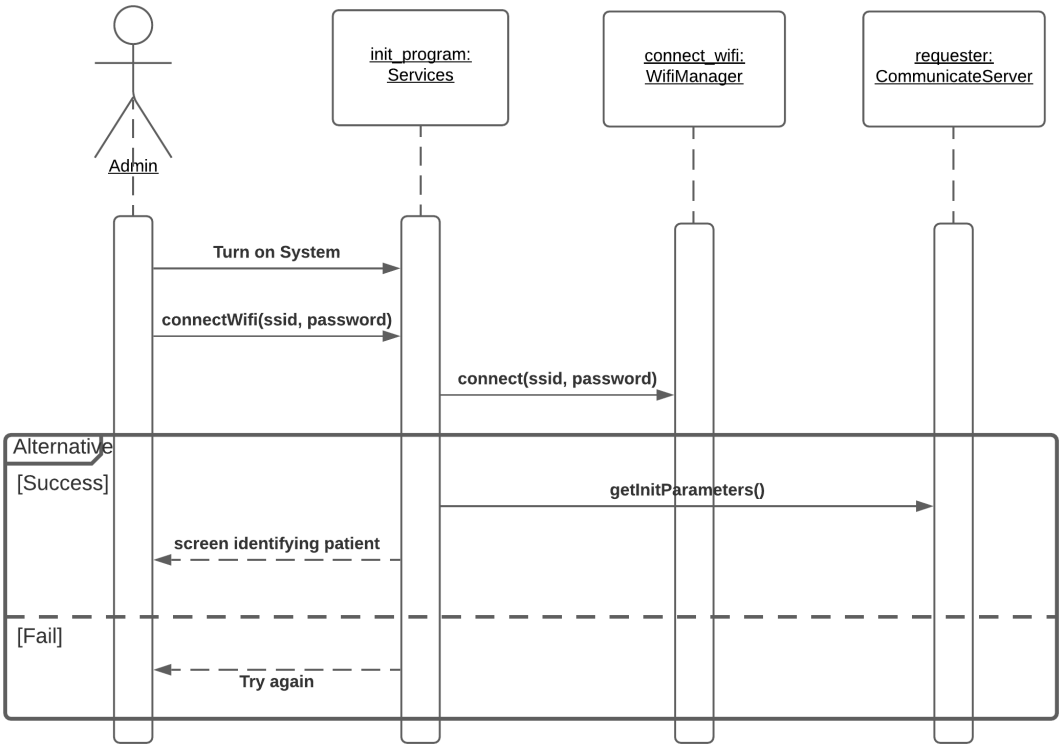

Figure 4.43: Sequence Diagram of Initiating System

4.4.3.2 Application Screen

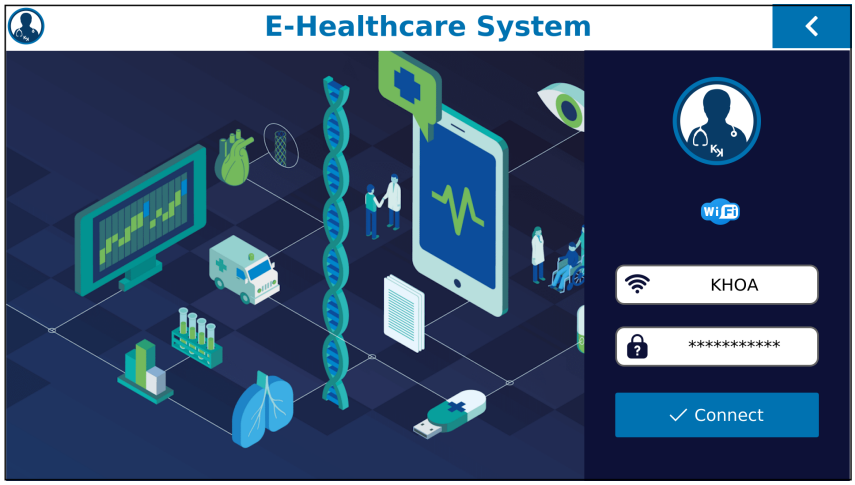

Figure 4.44: The Initial Screen Of The System

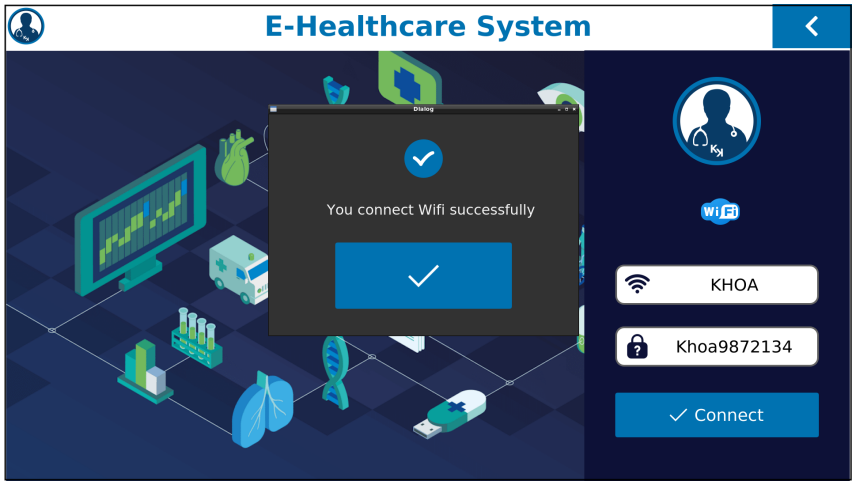

Figure 4.45: The Initial Screen Of The System When Successfully Connect The Wifi

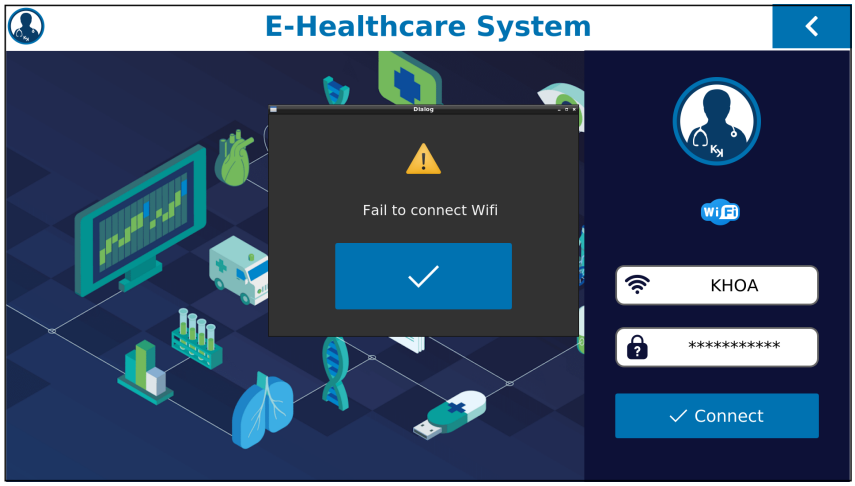

Figure 4.46: The Initial Screen of The System When Fail Connect The Wifi

4.4.4 Identifying Patients

This section responsible for identifying patients or verifying them with their SSN. We will introduce the sequence diagram and the workflow of three activities in this section. While the application screen of these activities will be displayed at the Application Screen section

- The normal flow of identifying patient
- Ask Is New Patient and Verify Patient with SSN
- Confirm patient information

Count Thread: This thread is set up to run each second to check whether we have enough encoded images in the list encoded images. If the number of the list of encoded images equal to defined THRESHOLD_NUMBER, this thread will call the Requester to send an Identification Request to the Device Server. We implement this thread to keep the Main Thread run smoothly and assure the great UI experience of the patients.

Main Thread

At the Identification State, the main thread continuously loops to read the frame from the camera and infer it into the Face Detection Model to extract the patient's face. Suppose there is a face in the current frame with the quantity equal to one. In that case, we infer that patient's face into the Face Identification model and retrieve the 128-dimensional vector float64. We then convert it into a string as the following format and update it into a list of encoded images.

"a0/a1/a2/.../a127" where *a0*, *a1*, ..., *a127* are the elements in vector, then we save it into a list of encoded images.

Then we check whether has a response from the Device Server through global variable flag *has_server_response*; if not, continue the loop. Otherwise, check the return status.

- If the return status specifies that this is a valid patient, we update the patient information in the GUI and ask the patient to confirm. If

the patient confirms this is their information, we move to measure the sensor and choose department state. Otherwise, we clear the GUI and turn back to identify the patient's state.

- If the return status is invalid, we increase the number of invalids and check whether it equals the THRESHOLD or not.
  - When the number of invalid patients equals to defined THRESHOLD, we ask the patient to check whether they are a new patient or not. If yes, we move to the New Patient state.
  - Furthermore, suppose they are old patients and want to use different authorization methods. In that case, they include their SSN at the pop-up asking for a new patient. The Device Server will use the SSN to accompany their face to perform the Face Verification task.
- Suppose the return status is wear_mask, which means that the patient is wearing a mask. The system will ask the patient to wear of mask for identification.

4.4.4.1 Sequence Diagram

Confirm patient information

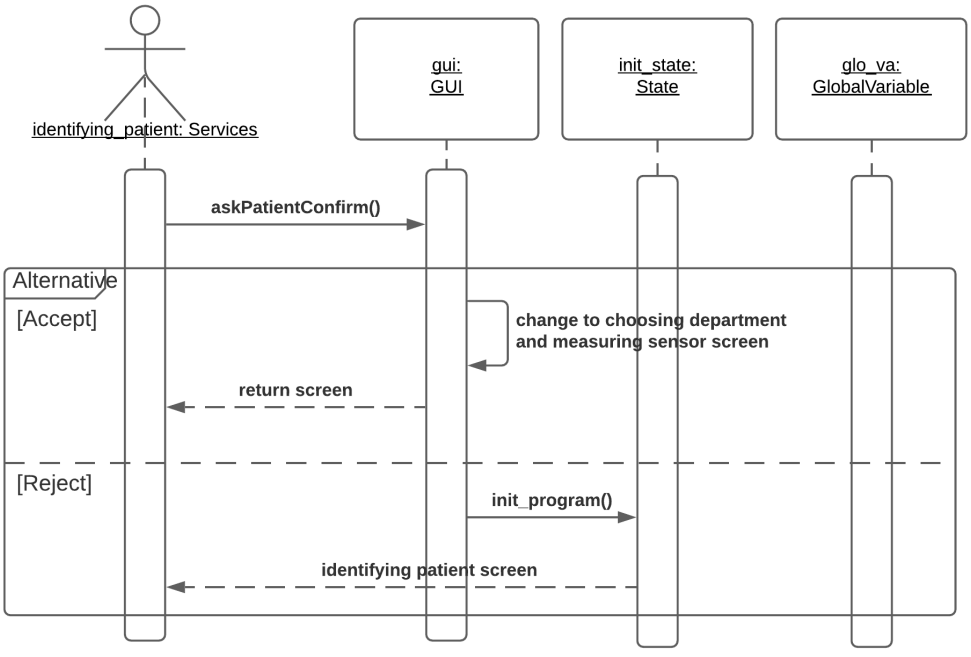

Figure 4.47: Sequence Diagram of Confirming Patient Information

The normal flow of identifying patient

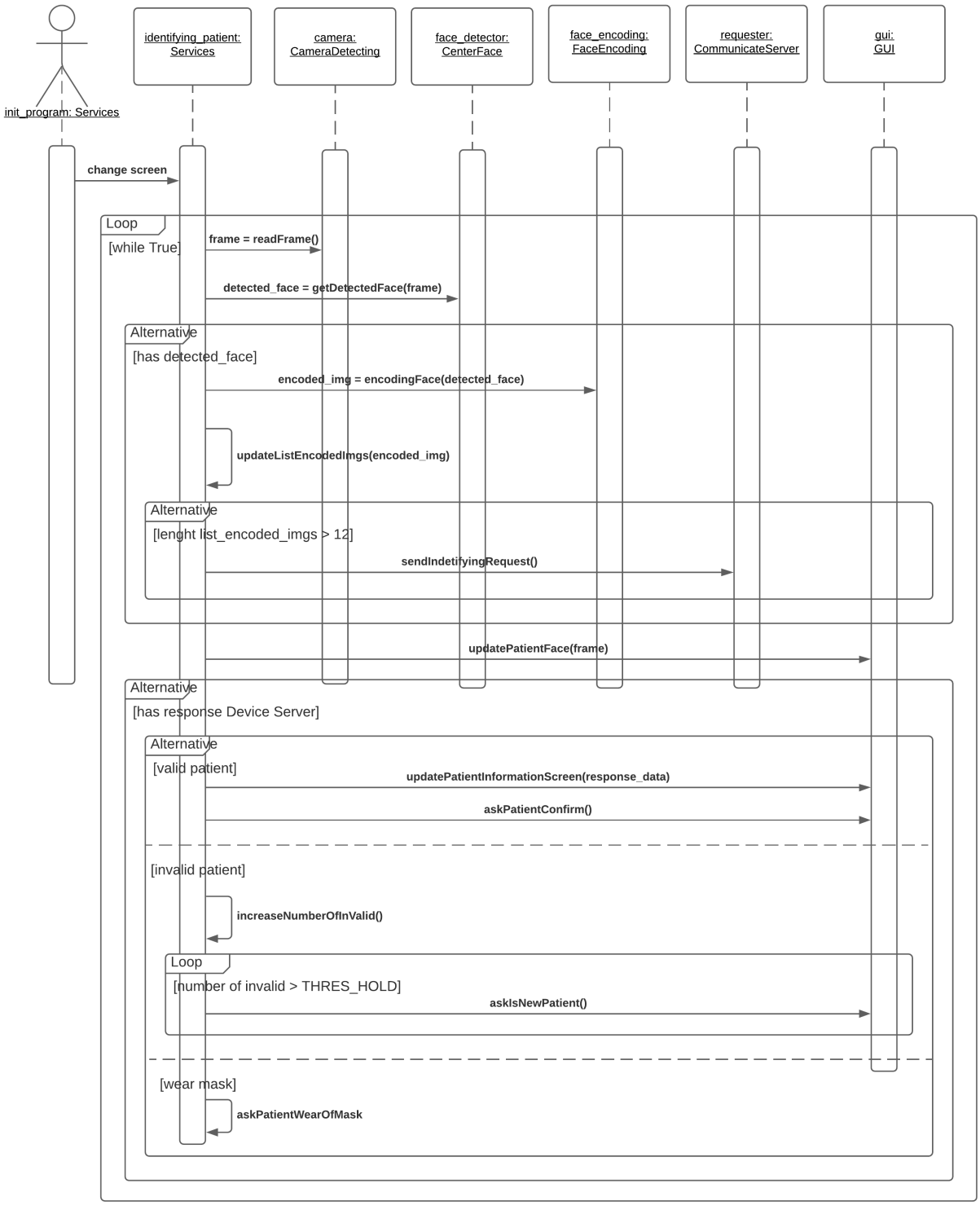

Figure 4.48: Sequence Diagram of Normal Flow of Identifying Patient

Ask Is New Patient and Verify Patient with SSN

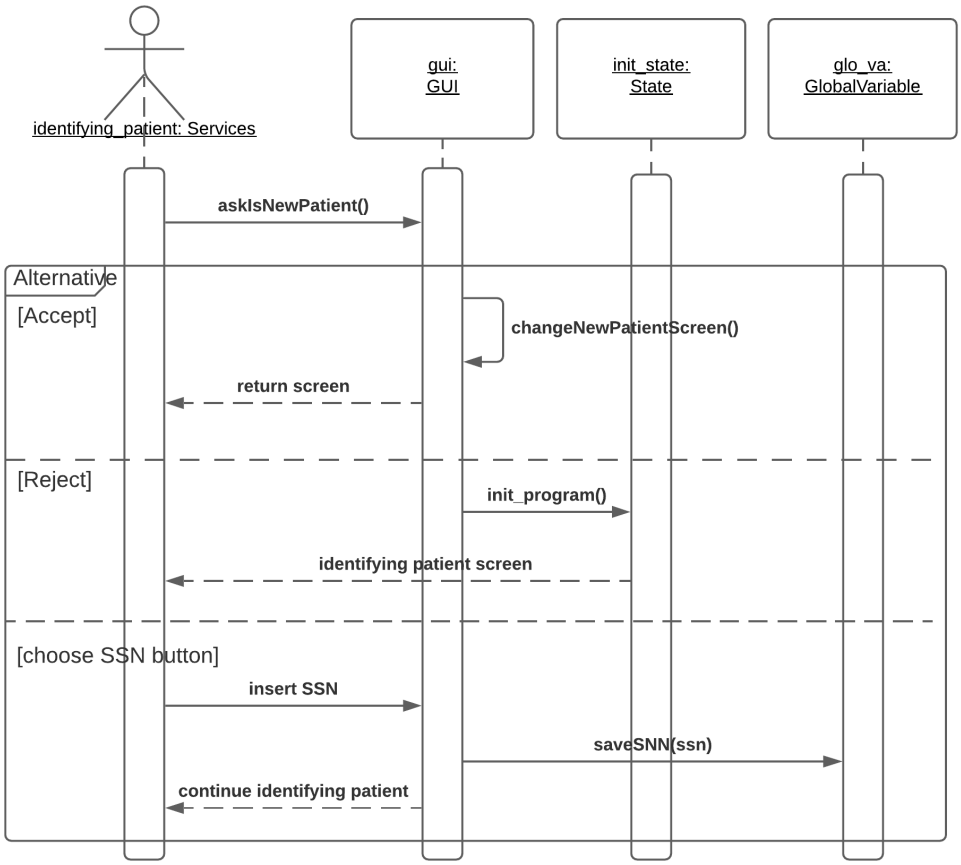

Figure 4.49: Sequence Diagram of Asking Is New Patient and Verifying Patient with SSN

4.4.4.2 Application Screen

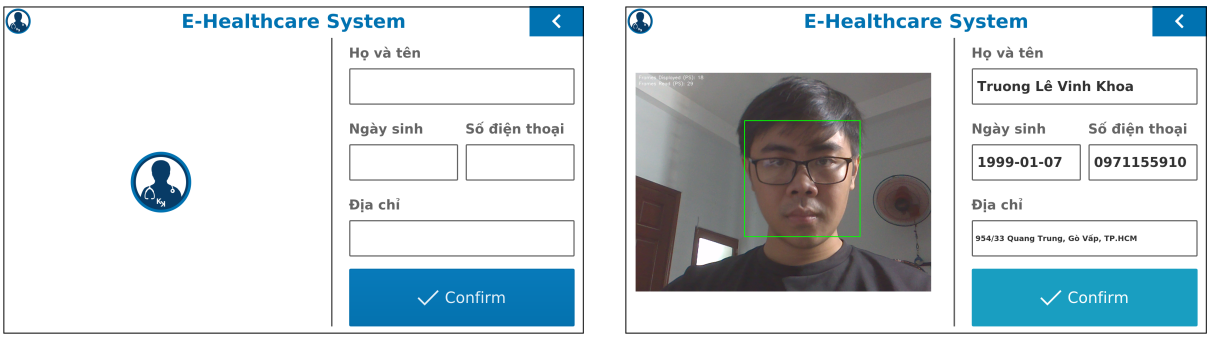

Figure 4.50: Login Screen Mobile Application For Old Patient

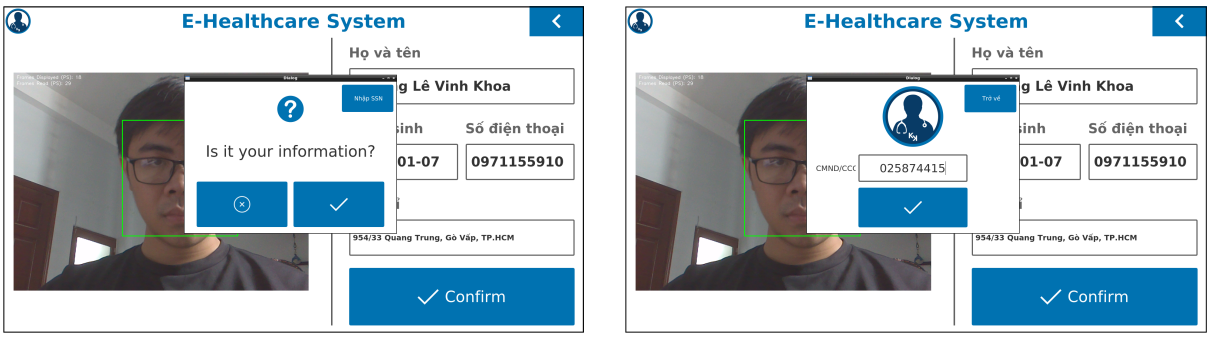

Figure 4.51: Confirm Patient Information and Verify by SSN

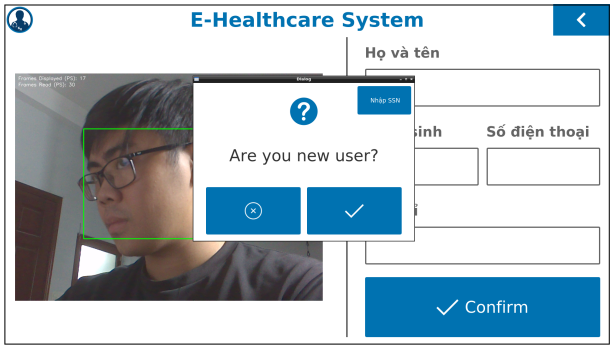

Figure 4.52: Confirming New Patient Screen

4.4.5 Capturing Face Pose of New Patient

This section describes how to capture the face data for registering new patients. When the new patient exceeds the number of trying identifying with Face Recognition Authorization, the system enables the patients to register and take the examination. In the following section, we describe the workflow of this section for more detail and give the sequence diagram.

The step of getting detected face and encoded image of a new patient is similar to the steps at Identification Patient state. However, there are some changes for a new patient at encoding detected face.

In capturing the face data of a new patient, we estimate the patient's face pose and the encoded face simultaneously in the same module called encodingFaceNewPatient, which returns the pose and encoded value of the detected face. Then we store each encoded image in the temporary list until we have enough of the defined quantity.

After that, we find the pose with maximum appearances in the above list and compare it with the current pose that we need to take. For example, if we want to take the front face, the *current_pose_face* is the front face. If the full pose face is the current pose face.

- We calculate the mean encoded image of the temporary list and change the current pose face to the next pose face.
- Store the mean encoded image into a list of encoded images for a new patient. This list will be sent with examination data when the patient finishes and submits the examination.

When the system takes enough face data of a new patient, they move to capture and measure sensor state similarly with the old patient.

4.4.5.1 Sequence Diagram

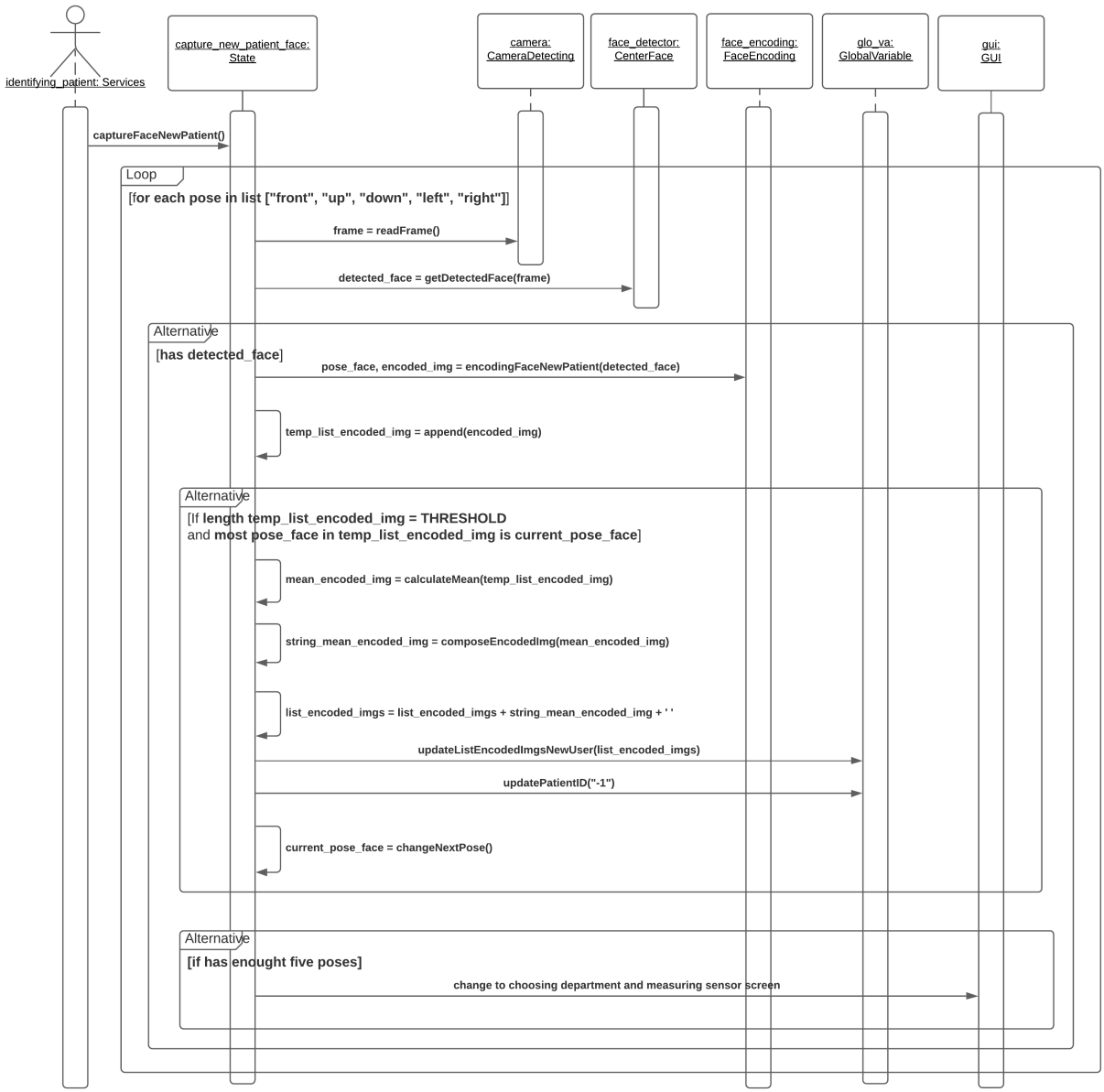

Figure 4.53: Sequence Diagram of Capturing Face Data of New Patient

4.4.5.2 Application Screen

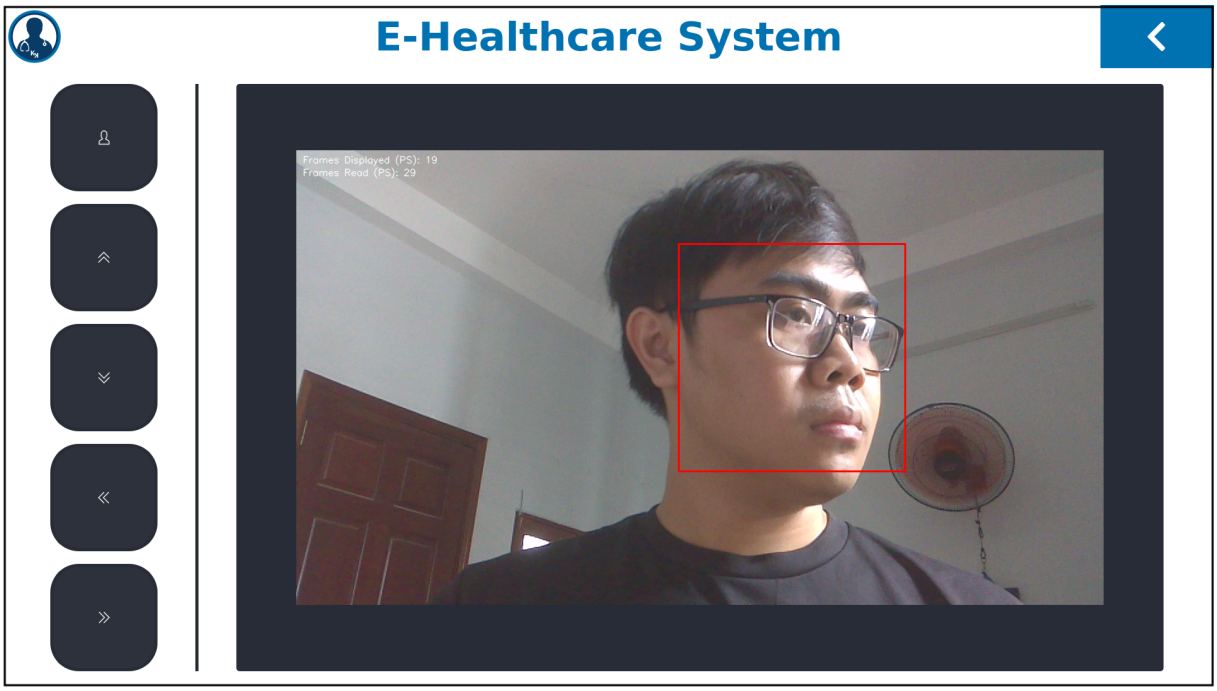

Figure 4.54: Collecting Wrongly Next Face Pose For New Patient

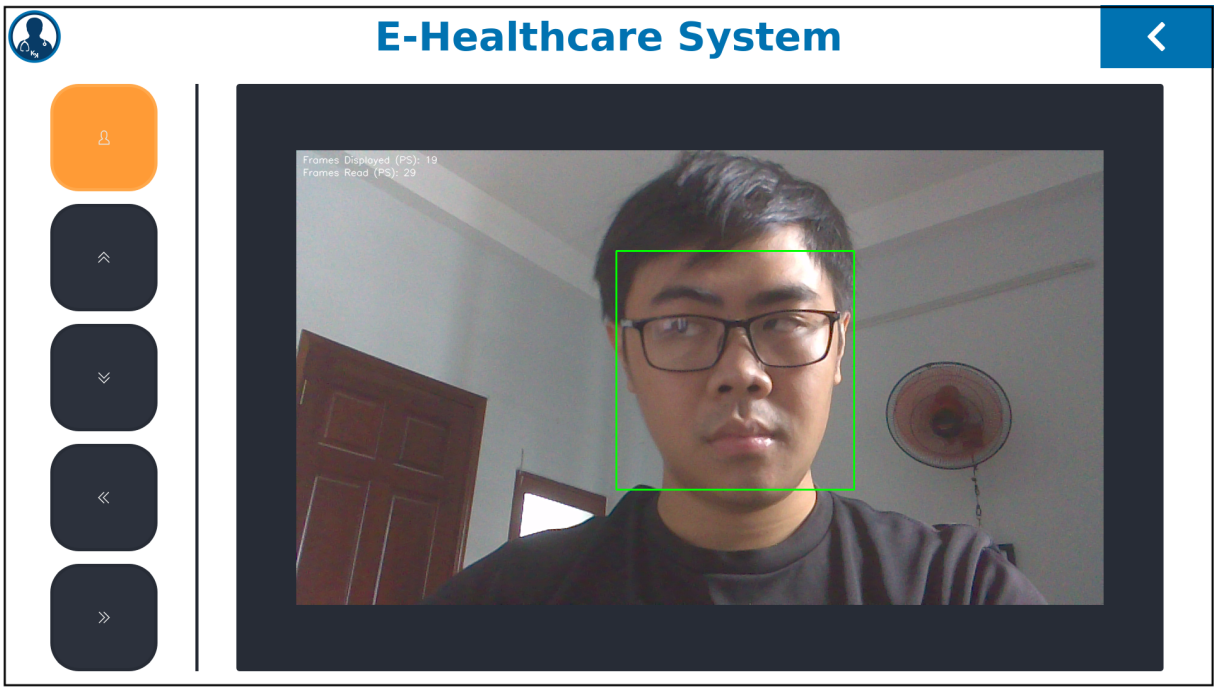

Figure 4.55: Collecting Successfully Next Face Pose For New Patient

4.4.6 Collecting Biologic Parameters

In this section, we will introduce the getting sensor data phase from the Local Device. After being identified by the system, the patient can choose a department first or measure their biologic parameters depending on their option.

If the patient decides to get the biologic parameters, they have experienced two steps in this phase. That is collecting the sensor data in ESP32, and OSO2 Devices, where the ESP32 is used to capture their height, weight, temperature, and the OSO2 is used to collect their spo2 and heart rate.

In the following section, we will introduce the sequence diagram of this phase. However, it is a pretty inconvenience for us to describe the whole workflow of this step in one diagram. For this reason, we break this diagram into three sub-diagram, including the main workflow and two sub-modules that are used to collecting sensor data from ESP and OSO2 Devices.

4.4.6.1 Workflow

When moving to the Measuring Sensor State, the system continues looping and reading the data from ESP32 and OSO2 devices until we have enough sensor data. If one of two devices finish its duty, the system will ignore them and read the rest device. To define which devices finish reading the sensor data, we check in the Sensor Entity.

Collecting Data from ESP32 Device

For each cycle, the system reads a message from the ESP32 asynchronously from the buffer. If there is no incoming message, the system continues to the next cycle. Otherwise, For each cycle, the system reads a message from the ESP32 asynchronously from the buffer. If there is no incoming message, the system continues to the next cycle. Else, we retrieve that message, preprocess, and extract the weight, height, and temperature value.

Based on comparing the current weight with the previous weight value, we detect whether there is a patient measuring sensor or not. If not, we remove all the previous data and start to measure again with zero records. Otherwise, we add three sensor values to each list, increase the number of records and display these values on the application screen.

When we have enough records, we remove the noise data of each sensor list by using its mean and standard deviation values, then calculating the mean value of each list after removing noising data. Finally, we update the final value of weight, height, and temperature value into the Sensor Entity and display it on the application screen.

Collecting Data from OSO2 Device

Similar to the first two steps in reading a message from the ESP32 device, instead of extracting weight, height, and temperature value, we extract spo2 and heart rate value.

After that, based on the state of this process, we do different things as described in the following:

- **State 0**: At this state (waiting for the first message), we check whether having the message from the device or not. If yes, we append two sensor values into its list and move to State 1 (collecting values state). Otherwise, we continue to the next loop and repeat this process.
- **State 1**: We start to receive the message from the OSO2 device.

  For each cycle, we check whether missing the data or not. If yes, we increase the missing time of reading data and check its quantity. When the number of missings is equal to the defined THRESHOLD, it means that the device finishes transferring data, and we move to the final state. Unless we have the data, we add spo2 and heart rate value into its list. Finally, we repeat the next loop.
- **State 2**: When we have enough records, we remove the noise data of each sensor list by using its mean and standard deviation values, then

calculating the mean value of each list after removing noising data. Finally, we update the final value of spo2, and heart rate value into the Sensor Entity and display it on the application screen.

4.4.6.2 Sequence Diagram

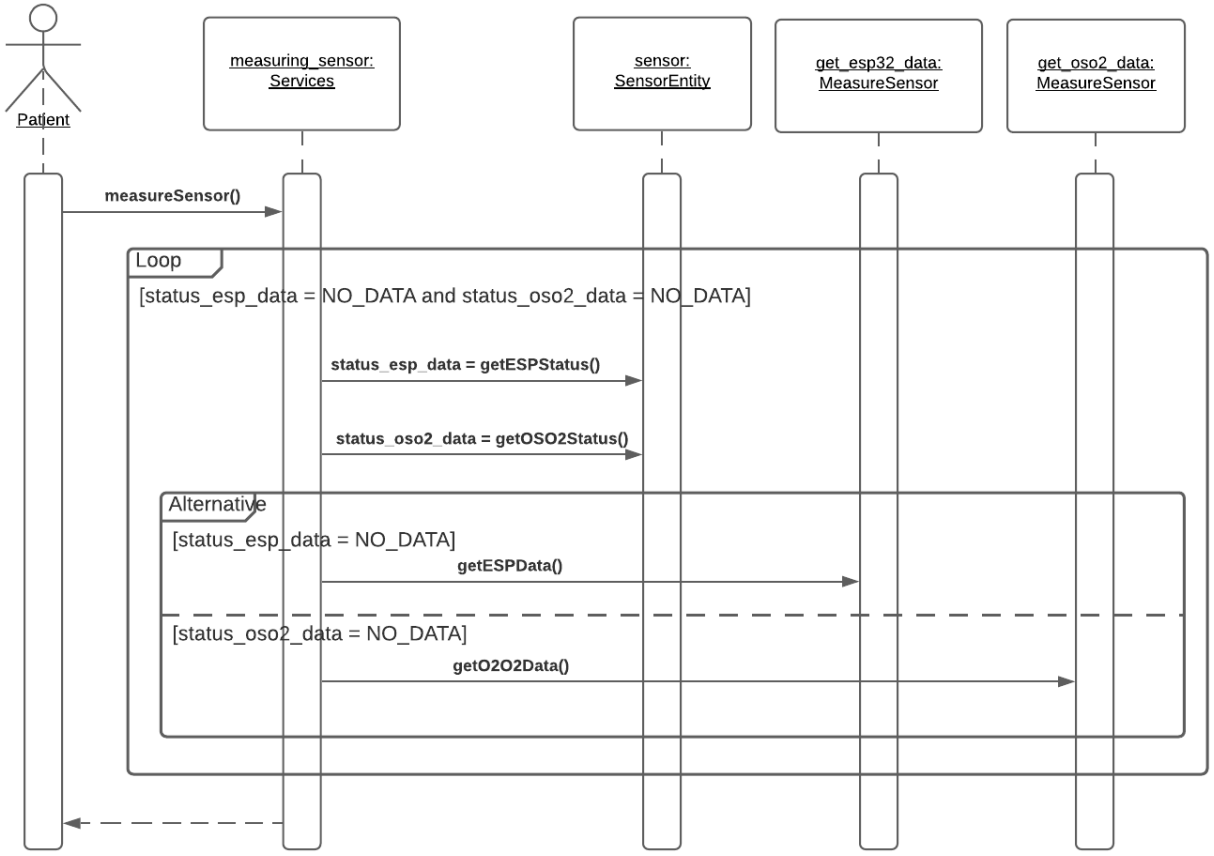

Figure 4.56: Sequence Diagram of Getting Sensor Data at Local Device

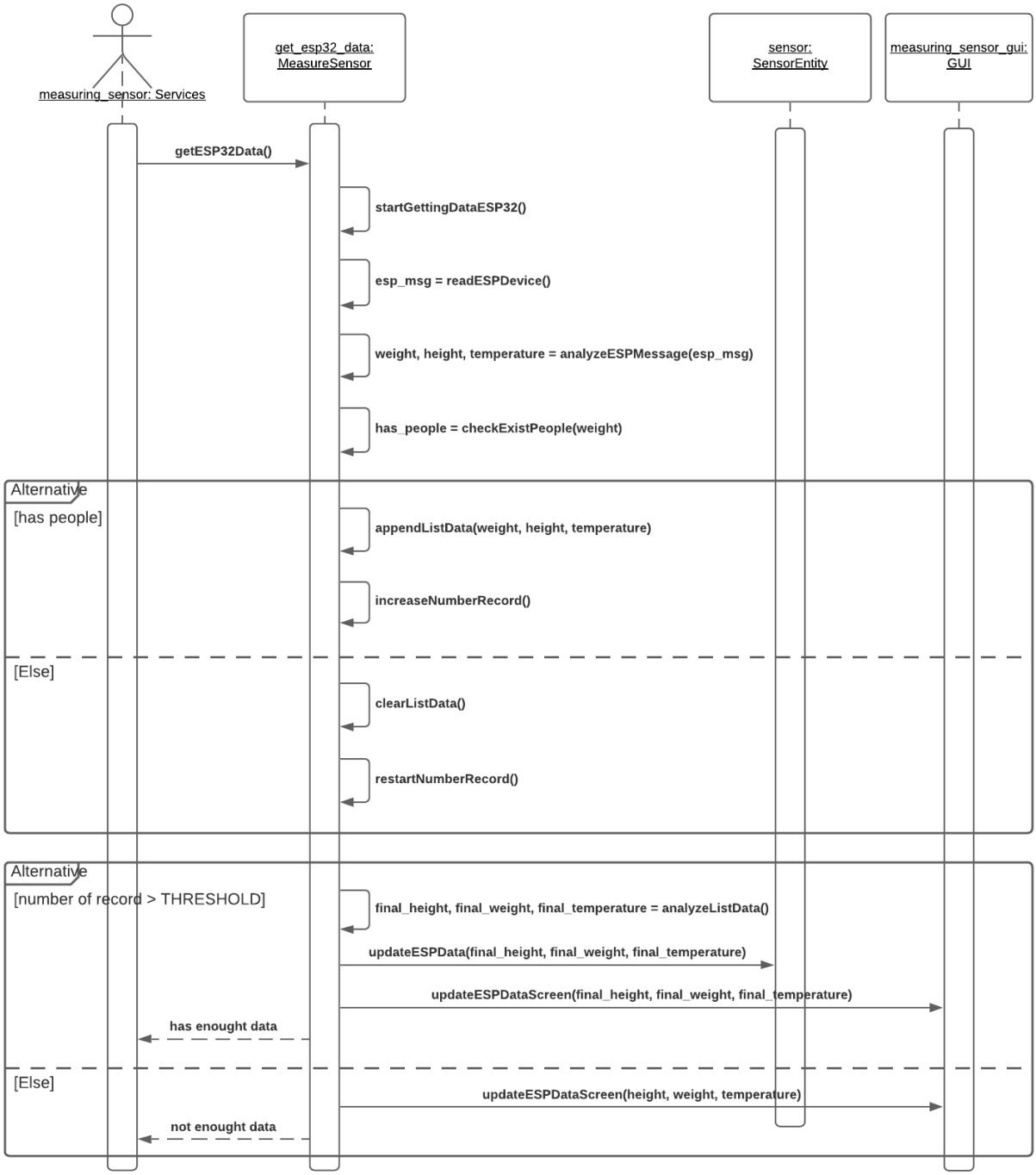

Figure 4.57: Sequence Diagram of Sub-module Getting Sensor Data from ESP Device

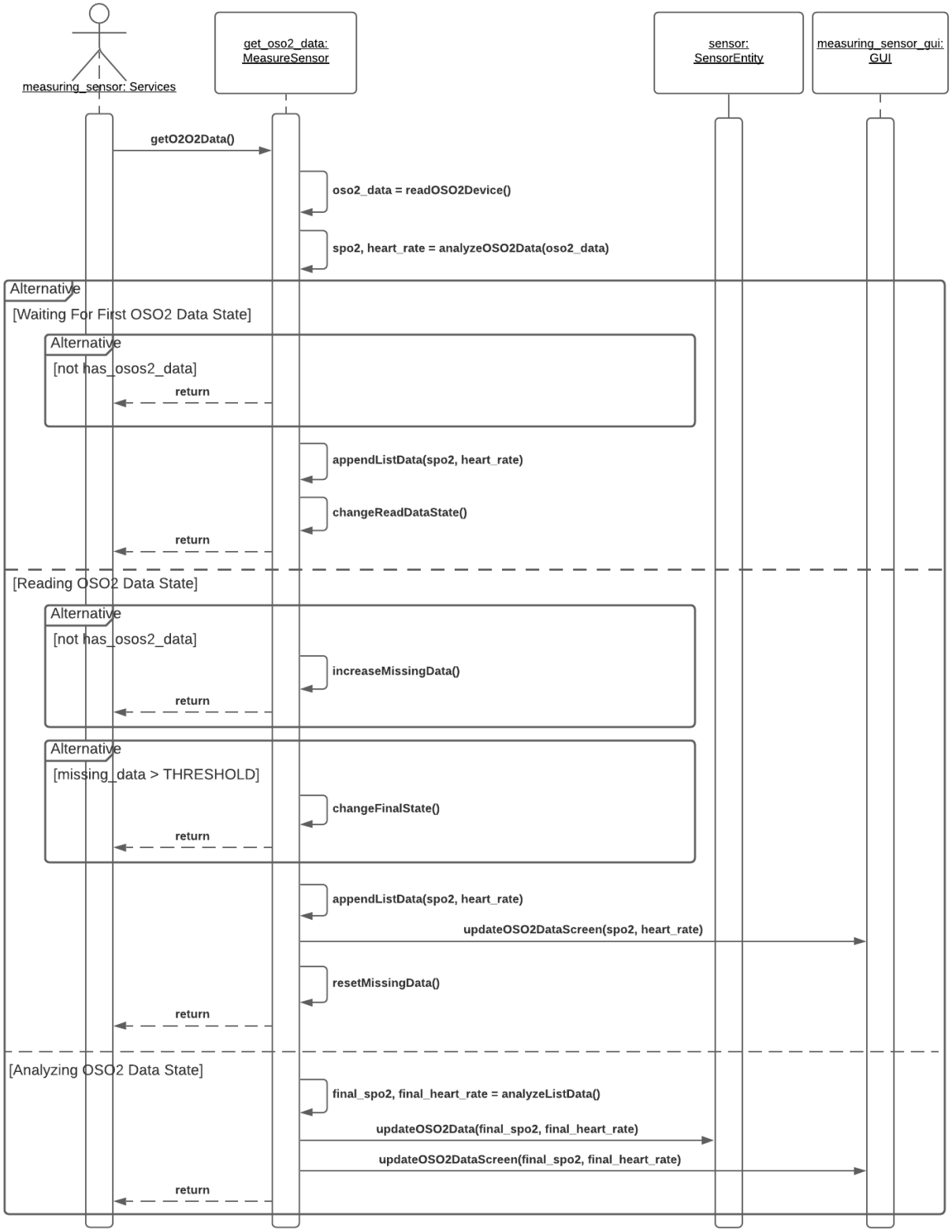

Figure 4.58: Sequence Diagram of Sub-module Getting Sensor Data from OSO2 Device

4.4.6.3 Application Screen

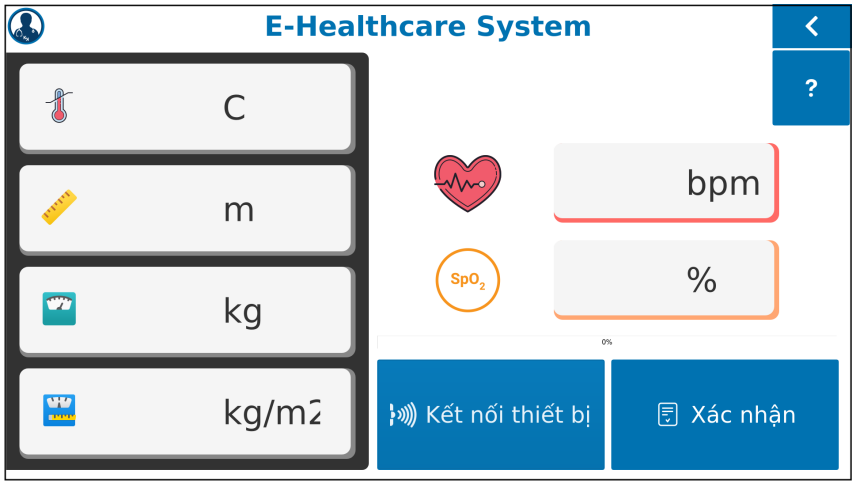

Figure 4.59: Beginning Screen of Measuring Sensors State

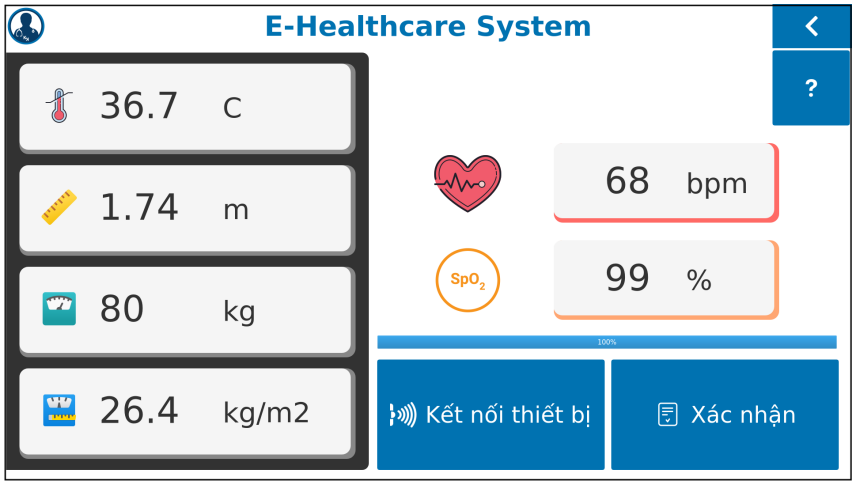

Figure 4.60: Collecting Successfully All Sensors Data

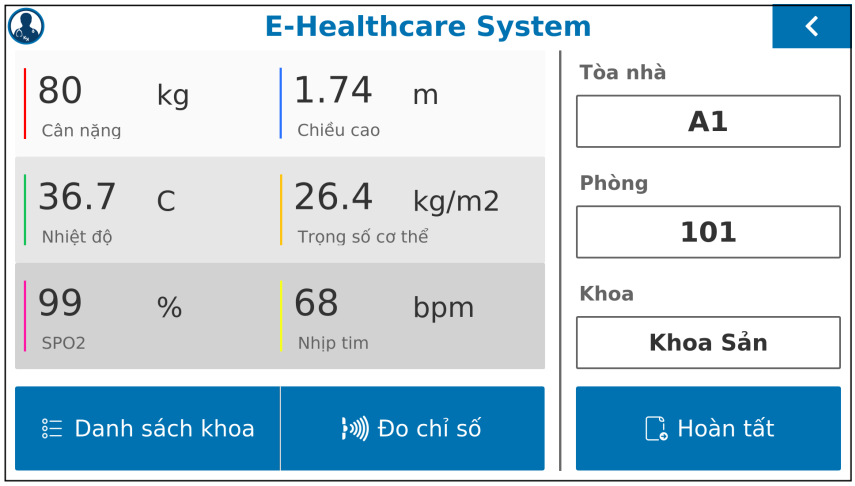

Figure 4.61: After Confirming the Sensors Data At Collecting Sensors Data Screen

4.4.7 Choosing Department

This section displays the User Interface for choosing a department. In this screen, the system enables the patient to choose which department they want to examine.

However, if they do not know which department they should choose, they may display their symptoms for the assistant. The assistant will give the patients an appropriate result. This function will be displayed in more detail in section 4.4.8. The application screen of this section is displayed in the figure below.

4.4.7.1 Sequence Diagram

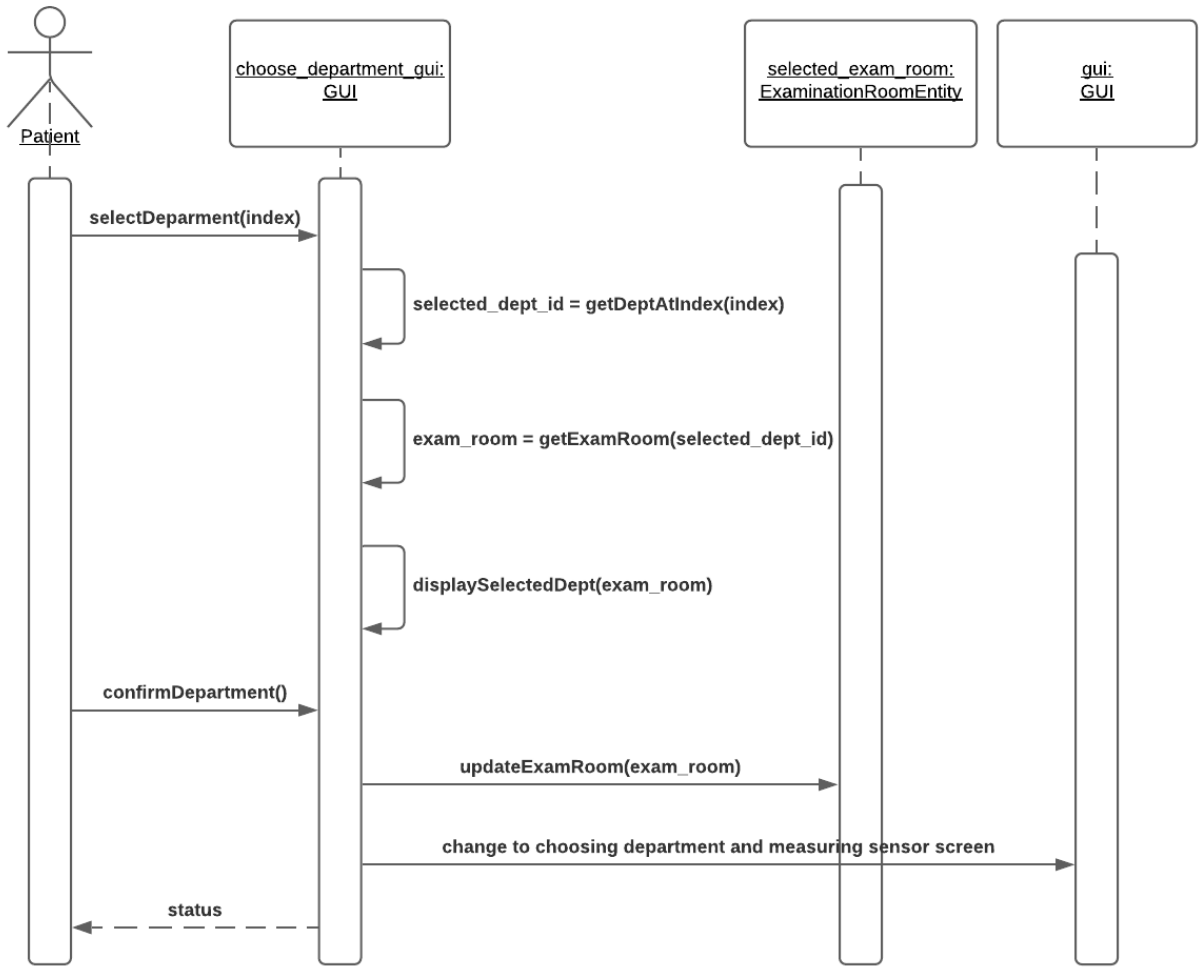

Figure 4.62: Sequence Diagram of Choosing Department of Patient

4.4.7.2 Application Screen

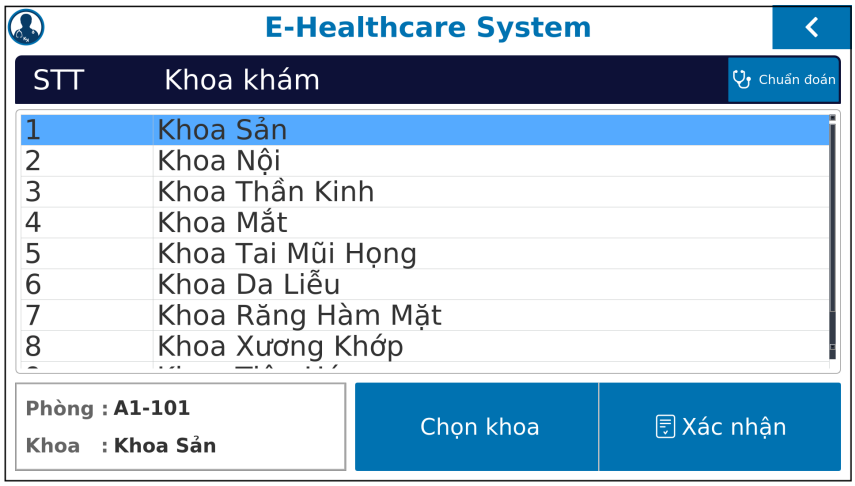

Figure 4.63: Choosing Department Screen

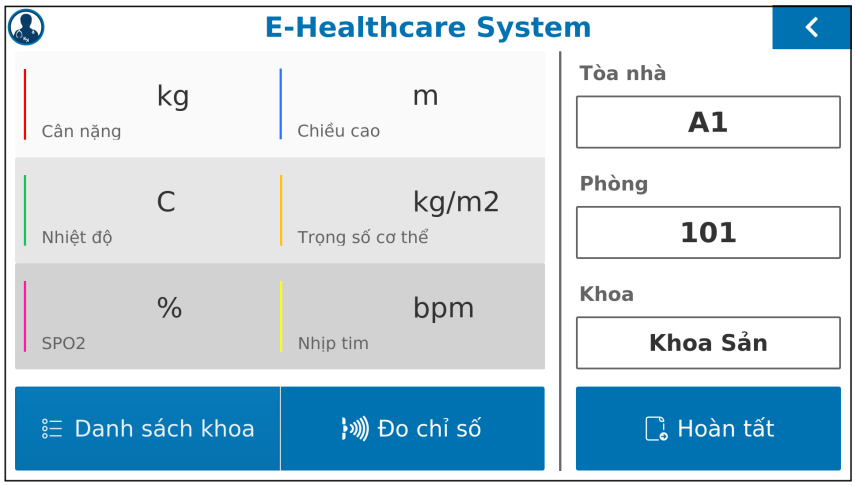

Figure 4.64: After Confirming the Department At Choosing Department Screen

4.4.8 Recommending Department With Assistant

When the patients do not know which department they should examine, our assistant can help them find the most appropriate department based on their displayed symptoms. The assistant runs on a different thread with our main program. When extracting the symptoms successfully and classifying the department, the assistant thread will terminate and update the department on the screen of Choosing Department. The remainder of this section briefly introduces the State Diagram of assistant with a full explanation, training department classification with *list_part_of_body* and *list_problem*, and the application screen.

4.4.8.1 Training Department Classification Model

This section will describe how to train a model that is used to choose the most appropriate department when we have lists of problems and parts of the body extracted from the displayed sentence of the patient. The overview of this section is described in Figure. 4.65 below describes the overview picture of this step.

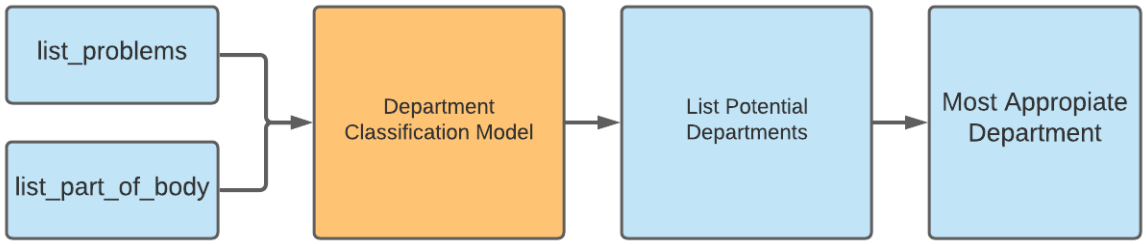

Figure 4.65: Department Classification Step

As introduced in section 4.3.4, with every sentence from the patient that follows one of two structures passive or active voice, our system will analyze it into two lists list_problems and *list_part_of_body*. Each word at the same index in both lists will present one symptom of the patient when combined. For example, with two sentences in the same meaning below, we display it in two types of structure and use our NLU model to extract list_problems and *list_part_of_body*.

Inputs:

- "Tôi bị rát và nổi mẩn ở da, ngoài ra tôi còn bị đau ngứa ở đầu"
- "Da của tôi bị rát, nổi mẩn và đầu của tôi rất ngứa"

Outputs:

- First sentence:
  - **List problems**: "rát", "nổi mẩn", "ngứa"
  - **List part of body**: "da", "da", "đầu"
- Second sentence:
  - **List part of body**: "da", "da", "đầu"
  - **List problems**: "rát", "nổi mẩn", "ngứa"

Then, we have three symptoms by combining each element according to their index in two lists list_problems and *list_part_of_body* in the below, and these data are used for the training classification model. However, it would follow the order **"problem", "part of body"**

- "ngứa", "da"
- "nổi mẩn", "da"
- "ngứa", "đầu"

Note: In some cases, when we use the active voice, such as the following sentence "Tôi bị khó chịu, mắc ói và buồn nôn". Although there is no part of the body in this sentence, we still both two lists as the below, where each element in the *list_part_of_body* is none value. It means that when we use this information for training, the model only considers the values in list_problems and it will be described in the next section.

Defining Training Data and Train Model

In this step, we discover and collect as much as the number data for classifying the department.

With each symptom extracted above, we find many departments that might associate with these symptoms and defined them in such as Figure. 4.66 in the following format: **"problem", "part of body", "list associate departments sorted in decreasing priority"** For example, with the symptoms *"đau"*, *"bụng"*, it might belong to some disease in the department "Tiêu hóa" and "Nội". Then we use the Decision Tree to train the model where the input in one symptom following the format *"problem", "part of the body"* and the out is the list of all departments associating with the above symptom in descending order.

- "problem", "part of body", {list associate departments sorted in decreasing priority}
- "problem", "part of body", {list associate departments sorted in decreasing priority}
- "problem", "part of body", {list associate departments sorted in decreasing priority}
- "problem", "part of body", {list associate departments sorted in decreasing priority}
- ...

Figure 4.66: Training Data for Department Classification

4.4.8.2 State Diagram

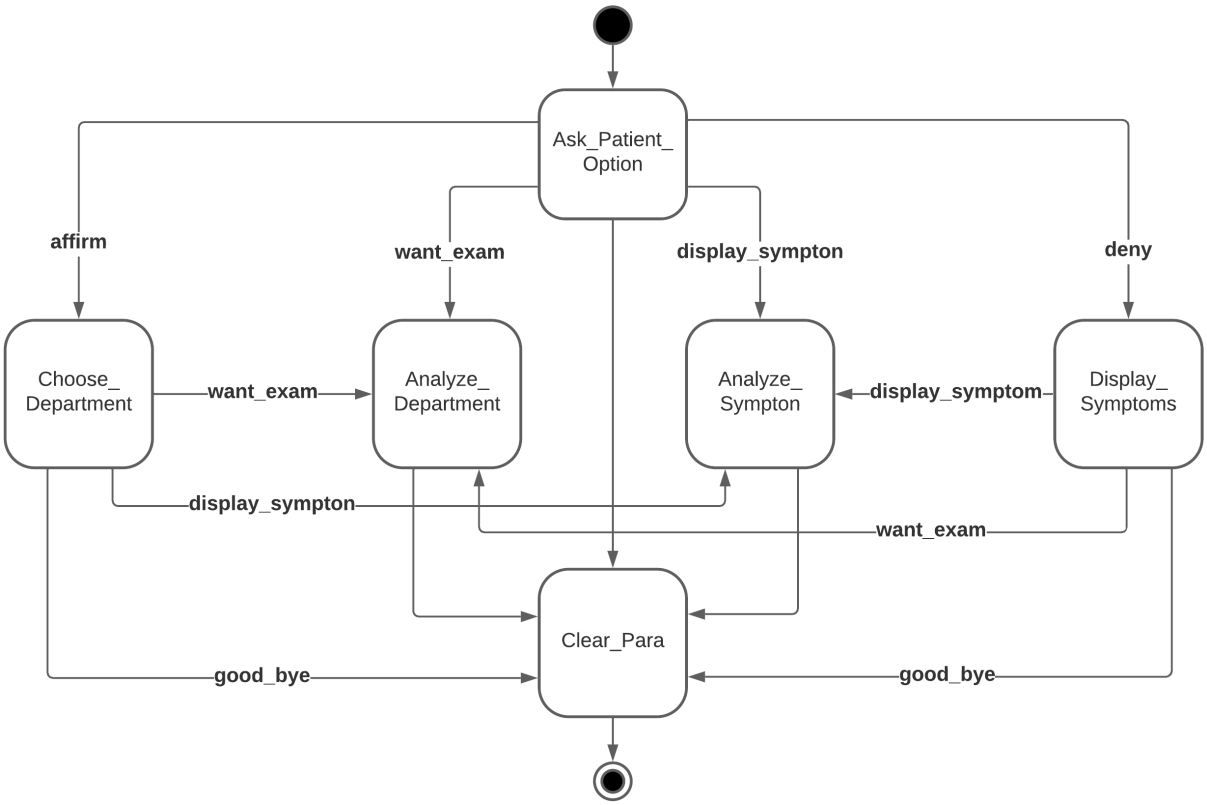

Figure 4.67: State Diagram of Recommending Symptoms Assistant

State Diagram Explanation

- **State Ask_Patient_Option**
  - At the first state, the assistant asks the patient *"which department that they want to examine"*.
  - If the patient answer the question with intent *affirm* such as *("có", "okay", "biết rồi", etc.)*, the system change to listening department name state(**Choose_Department**).
  - Or if the patient answer the question with intent *deny* such as *("chưa", "chưa biết", "không biết")* the assistant move to listening display symptoms from the patients(**Display_Symptoms**).
- **State Choose_Department**
  - In this state, when the assistant has the choosing department sentence from the patient, the system sends a request to the Device Server for extracting the *intent* and *entities* in that message.
  - If this is the *display_symptoms* intent, the system takes the department name in the response message and pass to the **Analyze_Department**
- **State Display_Symptoms**
  - After receiving the analyzed patients' sentence, we check the *intent* values in the response.
  - If this *intent* is display_symptoms, we extract the *list_part_of_body* and *list_problem* and pass to the *Analyze_Symptoms* module to get the appropriate department.
- **State Analyze_Department**
  - When having the department name returned from the server; the Local Device cannot determine what this department is.
  - For example, with the two displayed sentences below
  - "Tôi muốn khám khoa tai mũi họng" and "Tôi muốn khám tai mũi họng",.

- Although they have the same meaning in describing their interest to examine in the same department, the NLU model return two different values for *department_name* {"khoa tai mũi họng", "tai mũi họng"}. Therefore, it is unpleasant for the Local Device to choose precisely the department.
- For this reason, after receiving the response from the Device Server, we get the id of the department that the response is equivalent to by mapping the response to the list of equivalent Department Name in the following table and extract the department id.

Normalized Department Name	List of equivalent Department Name
0	“khoa thần kinh” , “thần kinh”
1	“khoa sản", "khoa phụ khoa", "khoa thai sản", "phụ khoa", "thai sản", "sản"
2	"khoa tổng quát", "khoa nội tổng quát", "tổng quát", "nội tổng quát"
3	"mắt", "khoa mắt"
4	"khoa tai mũi họng" ,"tai mũi họng", "mũi họng", "tai mũi", "tai", "mũi", "họng"
5	"răng hàm mặt", "khoa răng hàm mặt", "răng", "hàm", "mặt", "răng hàm", "hàm mặt"
6	"da liễu", "khoa da liễu", "da"
7	"khoa xương khớp", "xương khớp", "khớp", "xương"
8	"khoa tiêu hoá", "tiêu hoá"

Figure 4.68: Mapping Response to Department ID

- We then use this id to get the currently active department in the hospital and return for the patients.
- However, the above mapping table will be updated according to the changes of the department id or have a new department.

- **State Analyze_Symptoms**
  - After having *list_part_of_body* and *list_problem*, we convert it into list of the symptom, described in the section 7.3.8.1, following the format: *["problem 0", "part of the body 0"], ["problem 1", "part of the body 1"], ...]*, we start to get the most appropriate department by:

– Iterating the list of symptoms. With each element ["problem", "part of the body"], we infer them into the Decision Tree model to get the list of departments associating with that symptom.

– Then, after finishing the iteration, we analyze all the associate departments and find the one with the highest occurrence.

- **State Clear_Para**: We clear all the parameters and back to the first state of the assistant.

4.4.8.3 Sequence Diagram

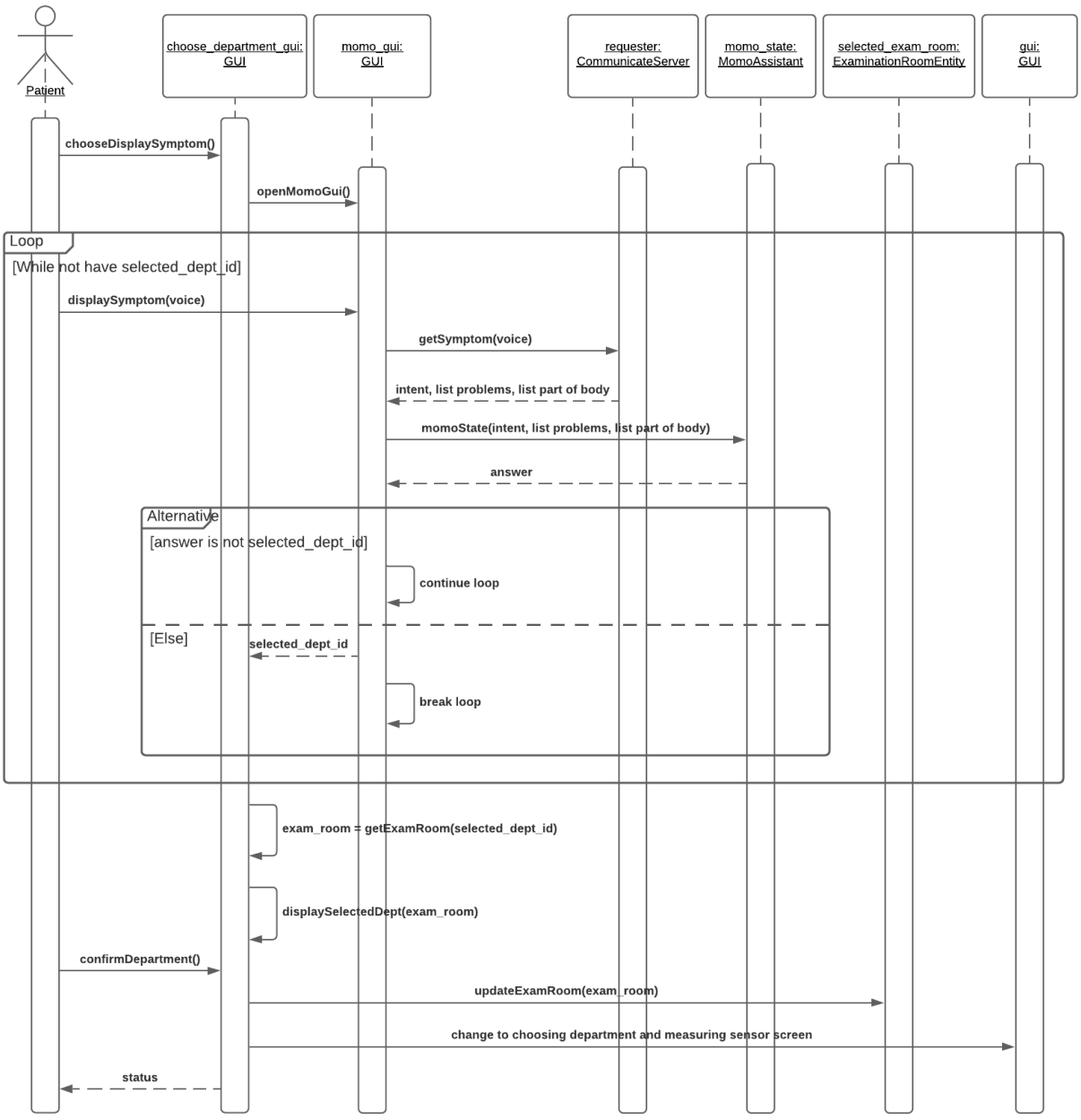

Figure 4.69: Sequence Diagram of Predicting Department

4.4.8.4 Application Screen

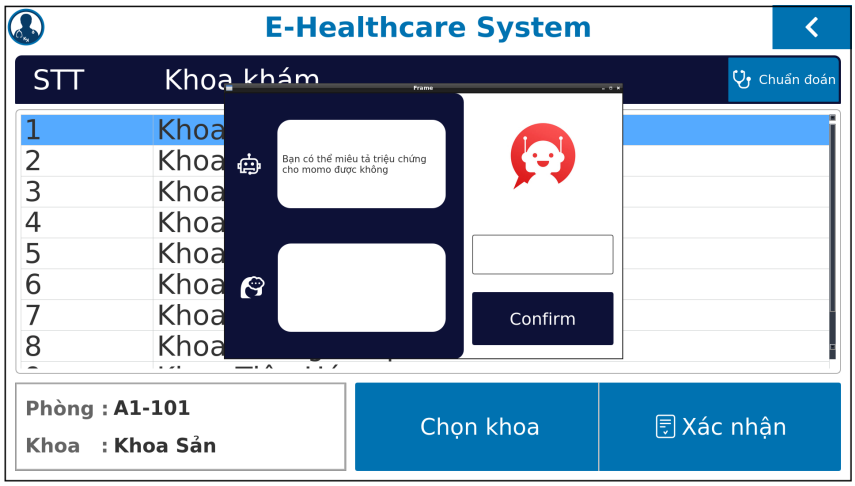

Figure 4.70: Recommending Department With Assistant Screen

4.4.9 Submitting Examination

After identifying, choosing the appropriate department, and collecting the biologic parameters, the patients can submit the examination and receive their numerical order in the examining room. However, the system has to pre-check whether the patient completes all the previous steps before submitting the examination.

If they already have sufficient data, the system sends their request and wait for the response:

- If the request fails for some cases or the system is trigged by a timer after a timeout period, the patients are notified and asked to re-submit the examination
- However, if the request is successful, the system return for patient their numerical order and the code of their examining room. At this time, we also have an application helping the patient know their examining room and notifying them when it turns to their number.

4.4.9.1 Sequence Diagram

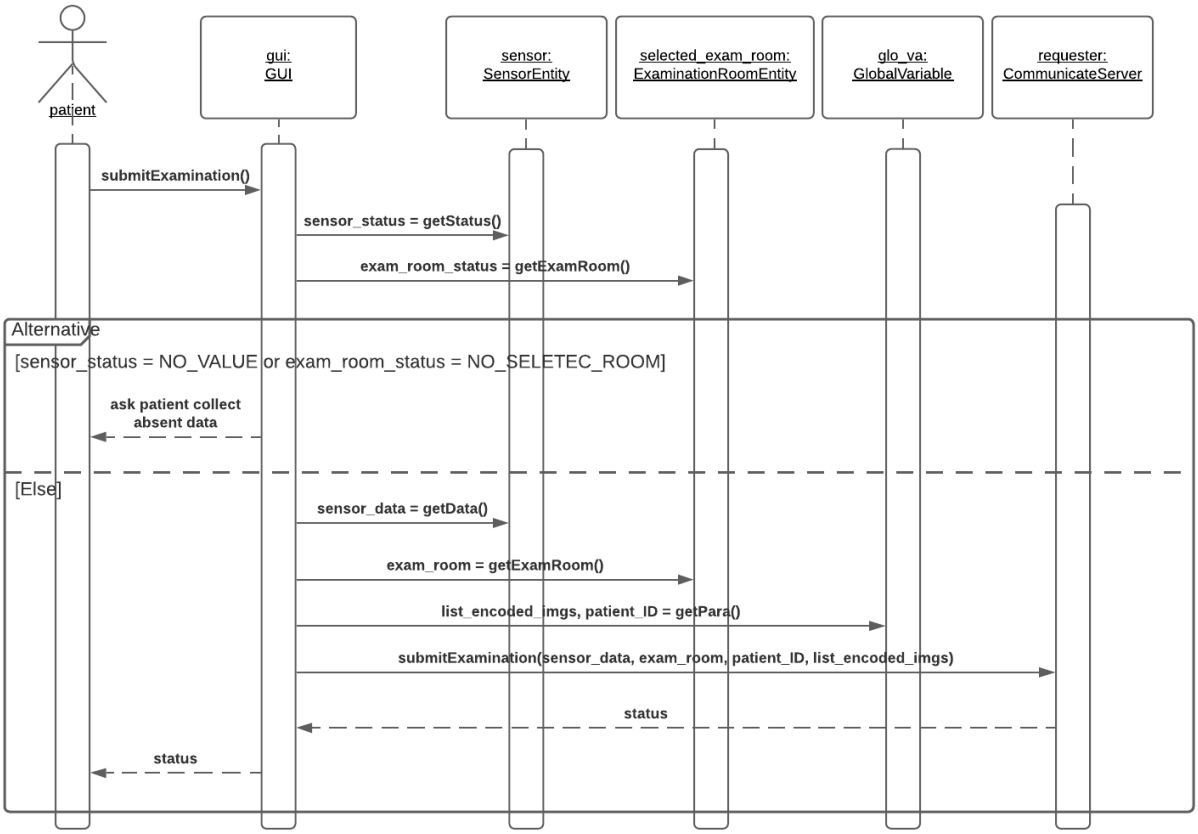

Figure 4.71: Sequence Diagram of Submitting the Examination

4.4.9.2 Application Screen

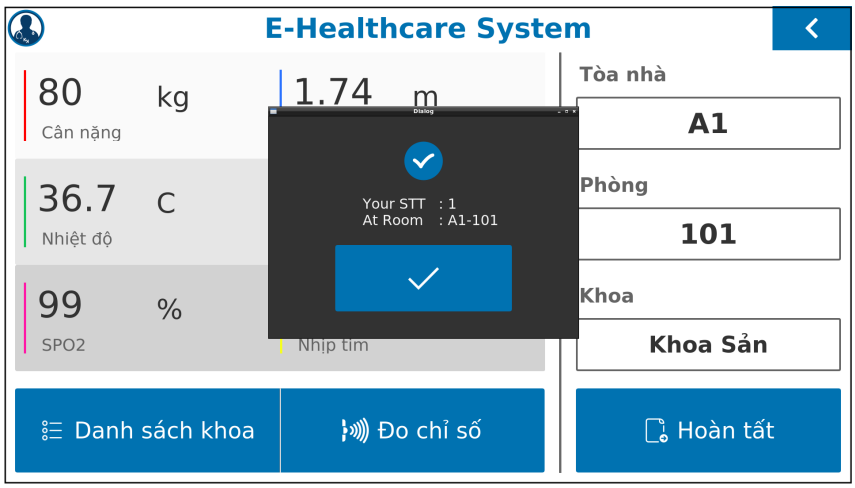

Figure 4.72: Result Of Submmitting The Examination Screen

4.5 Web Application

4.5.1 Server Side

The server side of the application are implemented under runtime environment Node.js, wrap by a back end web application framework Express.js

4.5.1.1 Design Pattern

For this application, our approach on implementing its server is using the **module pattern**. One of the reasons being we are using Node.js runtime environment, which is programmed in Javascript, prototype-based programming languages.

In software engineering, the module pattern is a design pattern used to implement the concept of software modules, defined by modular programming, in a programming language with incomplete direct support for the concept. This pattern can be implemented in several ways depending on the host programming language, such as the singleton design pattern, object-oriented static members in a class, and global procedural functions.

4.5.1.2 Module Flows

The operational flows of the server are to retrieve data from the database using the data access module; these modules will be used in the data transfer module to provide the data to other modules or the client-side of the application.

As this application has a CRUD archetype of application, there are three main flows in this application server, and each flows, the data are processed through to the following step:

Authentication:

1. Retrieve login data (username, password) from client-side

2. Checking if the login data is legitimate by performing retrieve authenticating data from database and then compare them.

3. If the authenticate success, redirect to the corresponding page (home page for doctor and management page for administrator) along with the user's information.

Examination:

1. When a doctor login in, the information of his room patients queue also loaded along

2. After finished examine a patient; the doctor will submit their examination information. The examination data will be transfer from the client-side to the data transfer module in the server.

3. In the above examination data, there will be *newPatientFlag* to determine whether it is a new patient. If it is a new patient, the doctors input the information in the client-side and transfer it after submitting the examination to the server-side and perform the add-patient module. This module includes update the new information to the dumpy data in the database and also trigger the Python server.

4. After adding a new patient or when it is an existed patient, the server will perform add-examination to the database in **Examination** module

5. Adding examination will return the *examinationId*, with the given information, if the doctor prescribes some medication, it can also be added in the **Medication** module.

6. Finishing up the adding the examination will trigger the update patient queue to retrieve the new queue. In addition, the next Patient will have the information of their order number, room code, and building code send to the Adafruit IO feeds for sending notification.

Management:

1. Retrieve the filter information from client side

2. Get the data the satisfied the filter condition in the corresponding module

3. Send back the data to the client through **Data Transfer Module**

4.5.2 Client Side

The client-side of the application will process the following steps:

1. When there is a patient in the doctors' room queue, the client will load the queue by order of the coming patient.
2. If the patient comes, the doctor will confirm the alert dialog to proceed; else, the patient will be removed from the queue and database by the server
3. If it is a new patient, the patient will be asked to fill in the form to collect the patient information (name, phone number, SSN, etc.)
4. Submitting the examination form will transfer all the data filled in the form for the server to process
5. Successfully submits it will reload the queue and load the information of the next patient.

4.5.2.1 Log In Screen

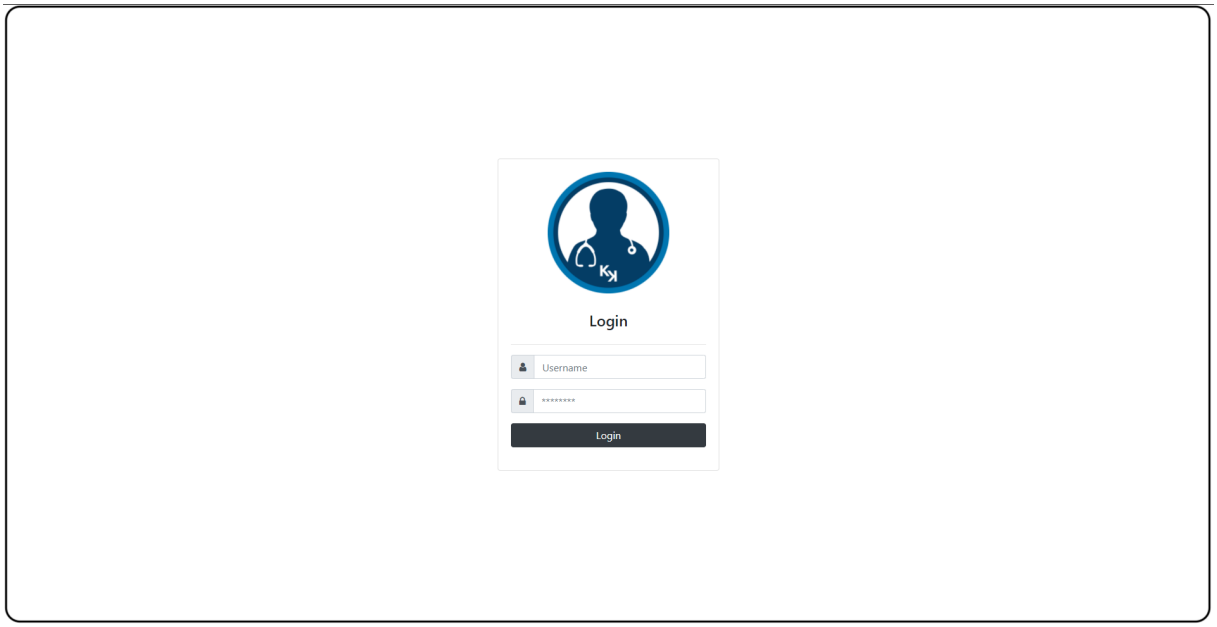

Figure 4.73: Login Page

4.5.2.2 New Patient In Queue Screen

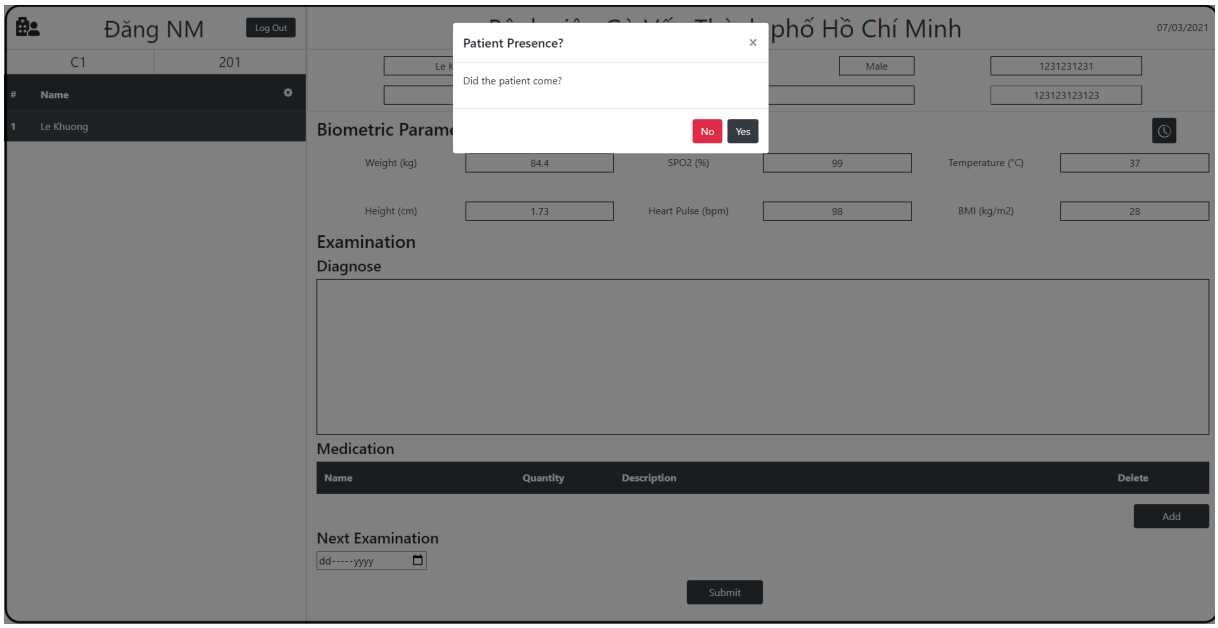

Figure 4.74: Initial home page

4.5.2.3 Confirm Patient Coming Screen

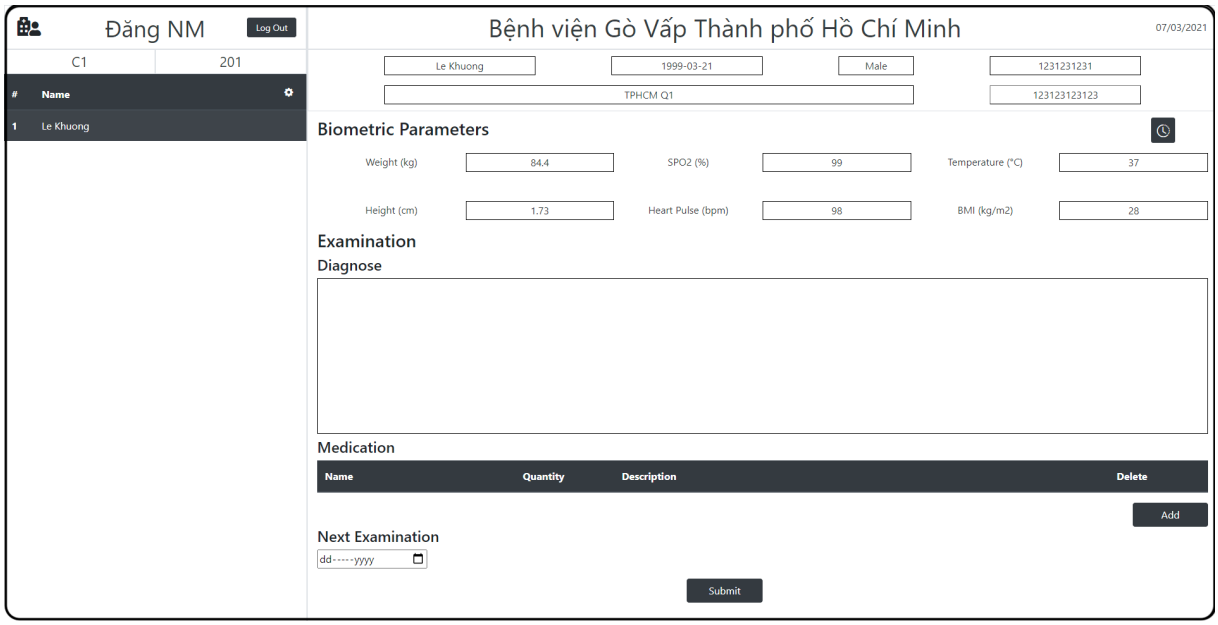

Figure 4.75: Confirm patient coming

4.5.2.4 Take The Examination Screen

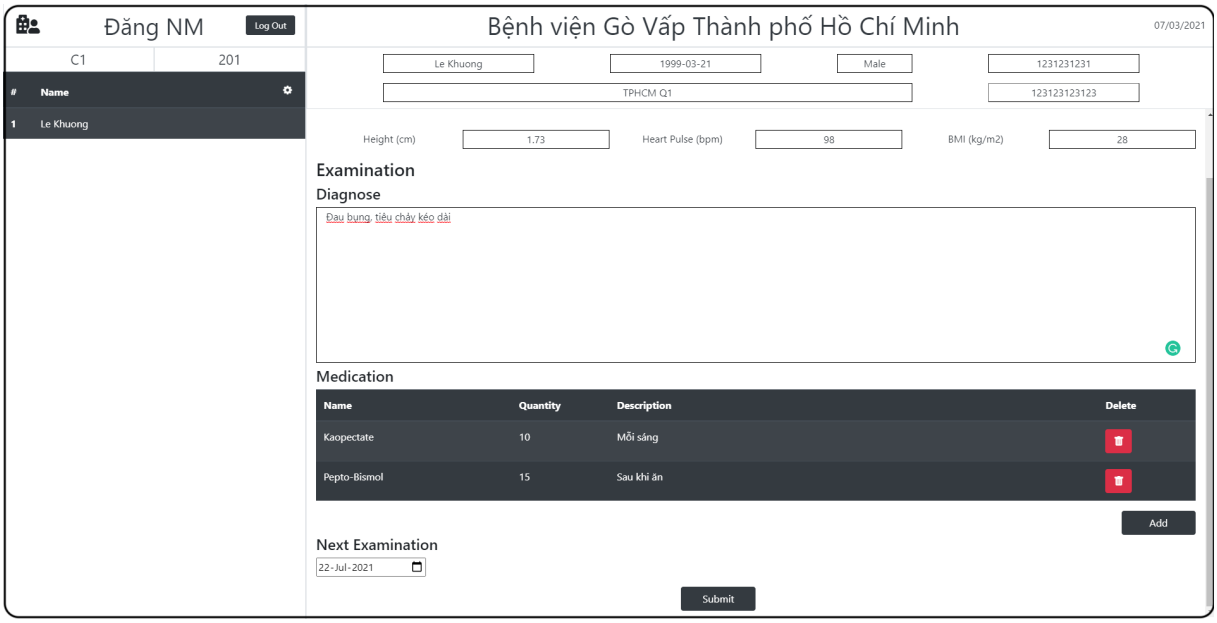

Figure 4.76: Filling examination information

4.5.2.5 Submit The Examination Screen

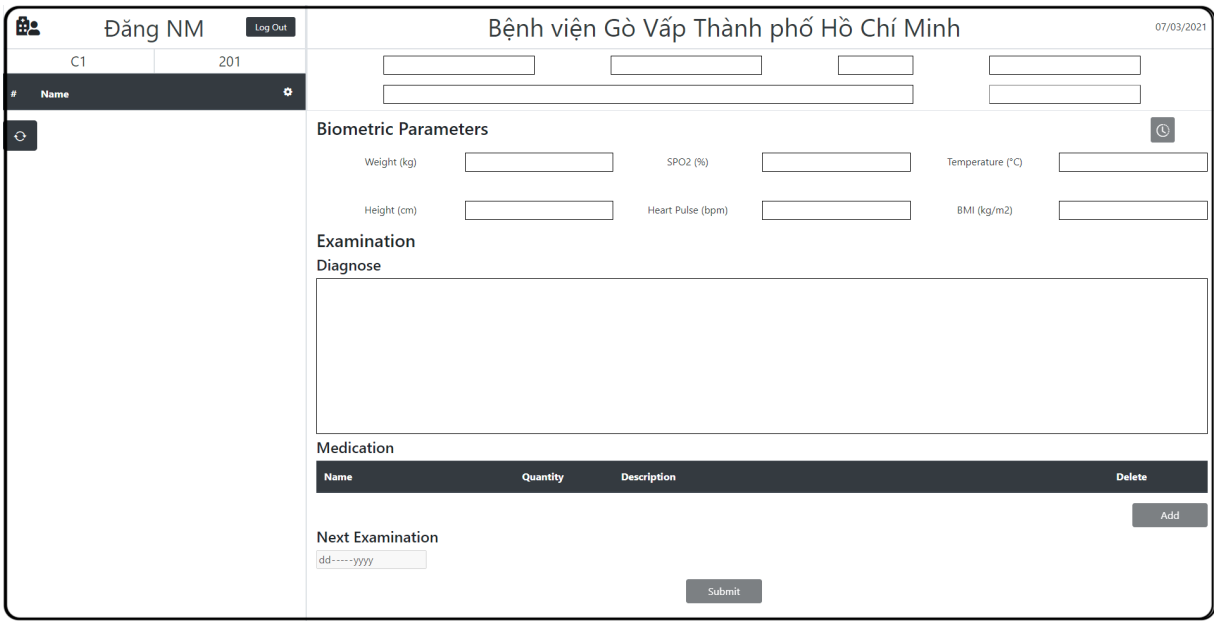

Figure 4.77: After submit examination form

4.5.2.6 Take The Examination With New Patient Screen

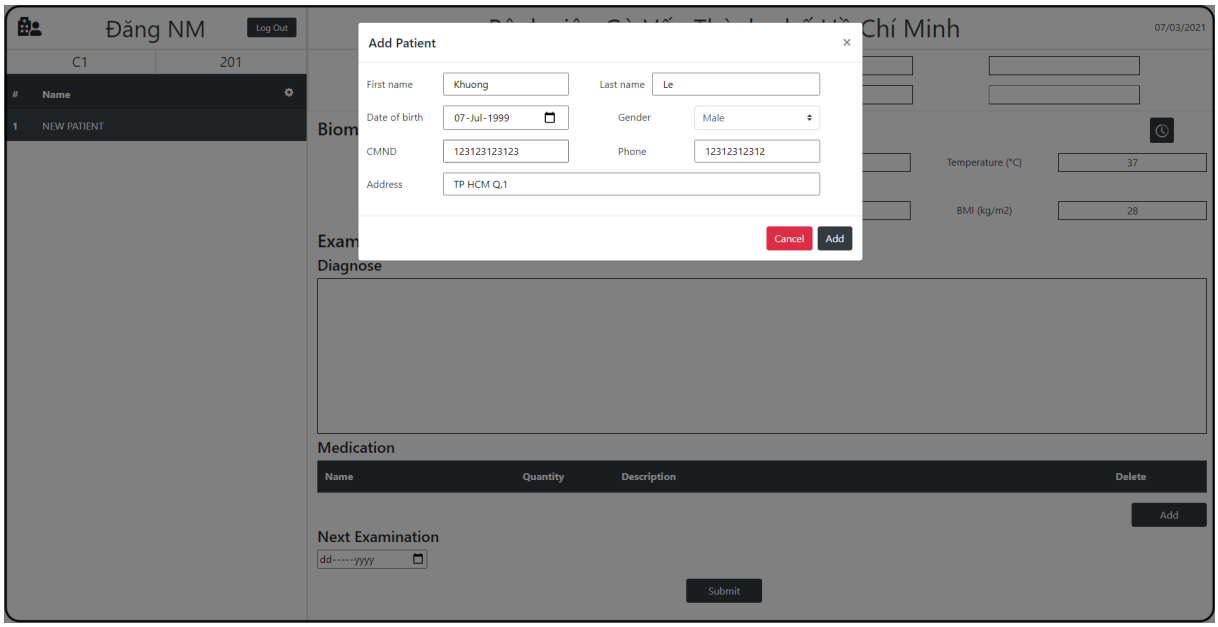

Figure 4.78: Input new patient information

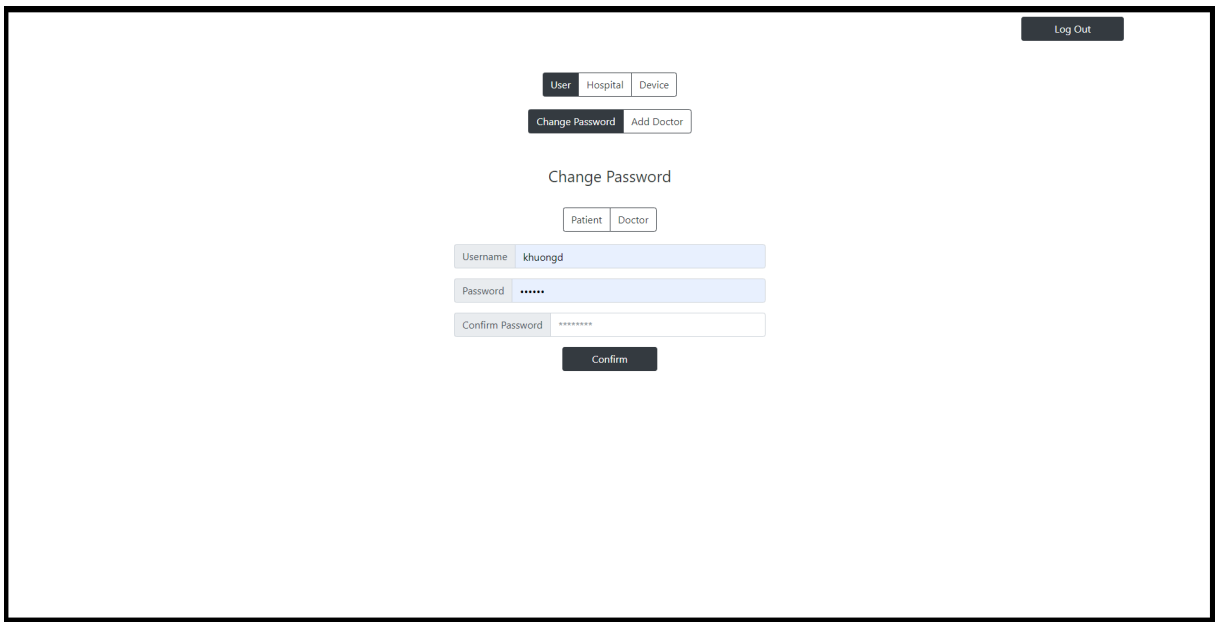

Figure 4.79: Change doctors' account password

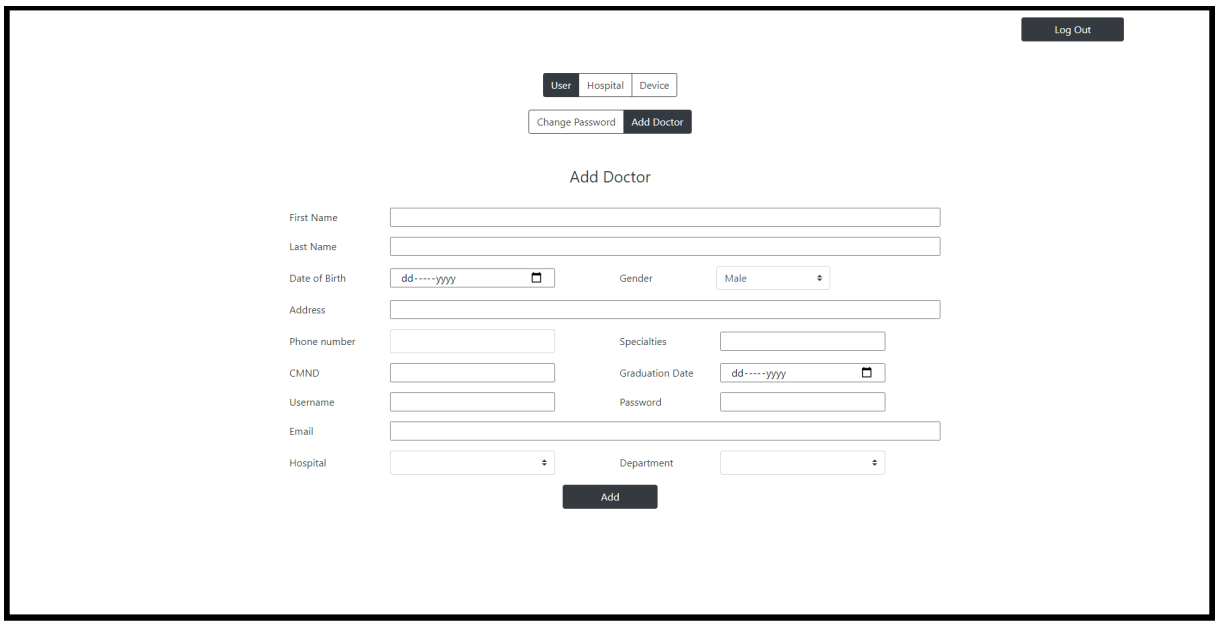

Figure 4.80: Add new doctor

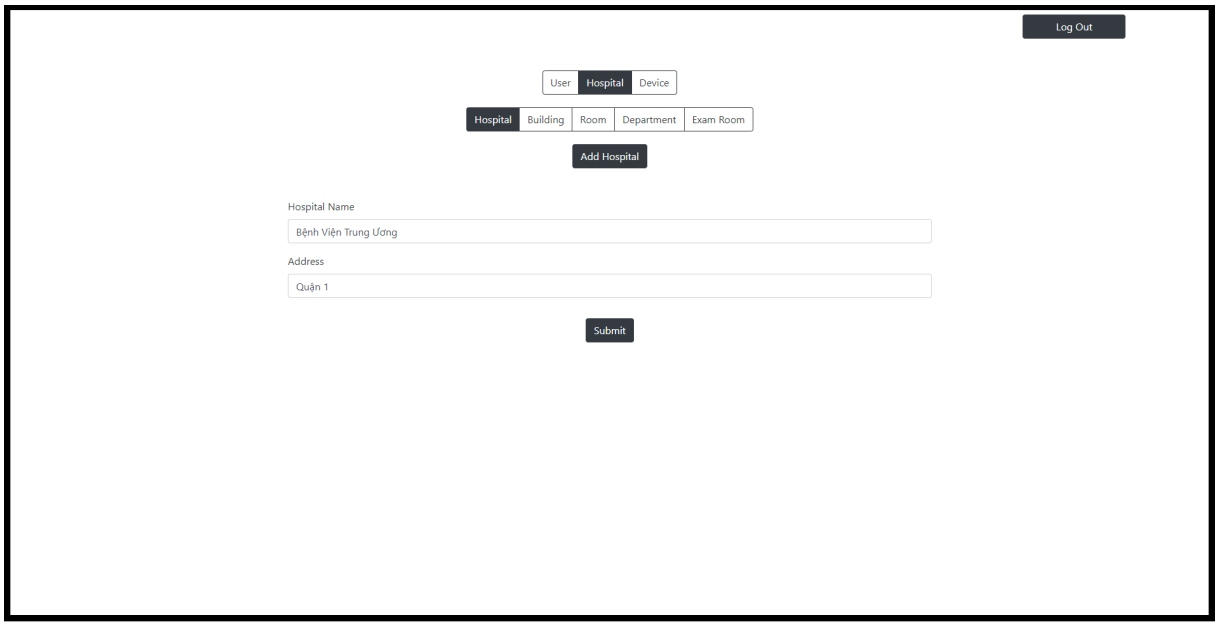

Figure 4.81: Add new hospital

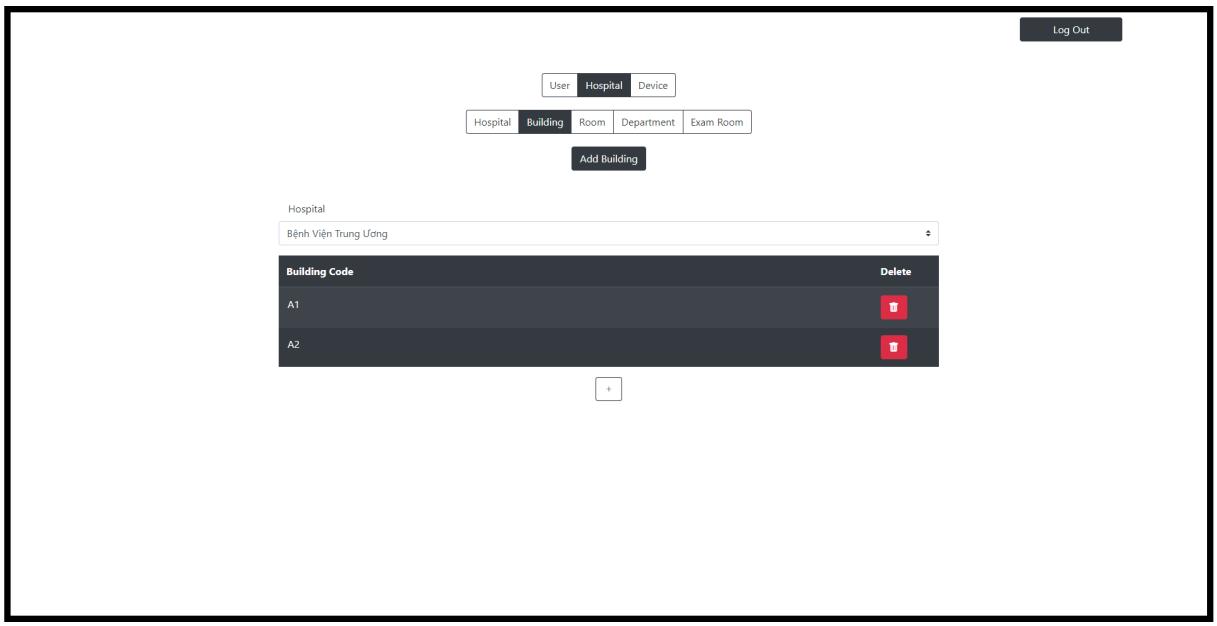

Figure 4.82: View building of hospital

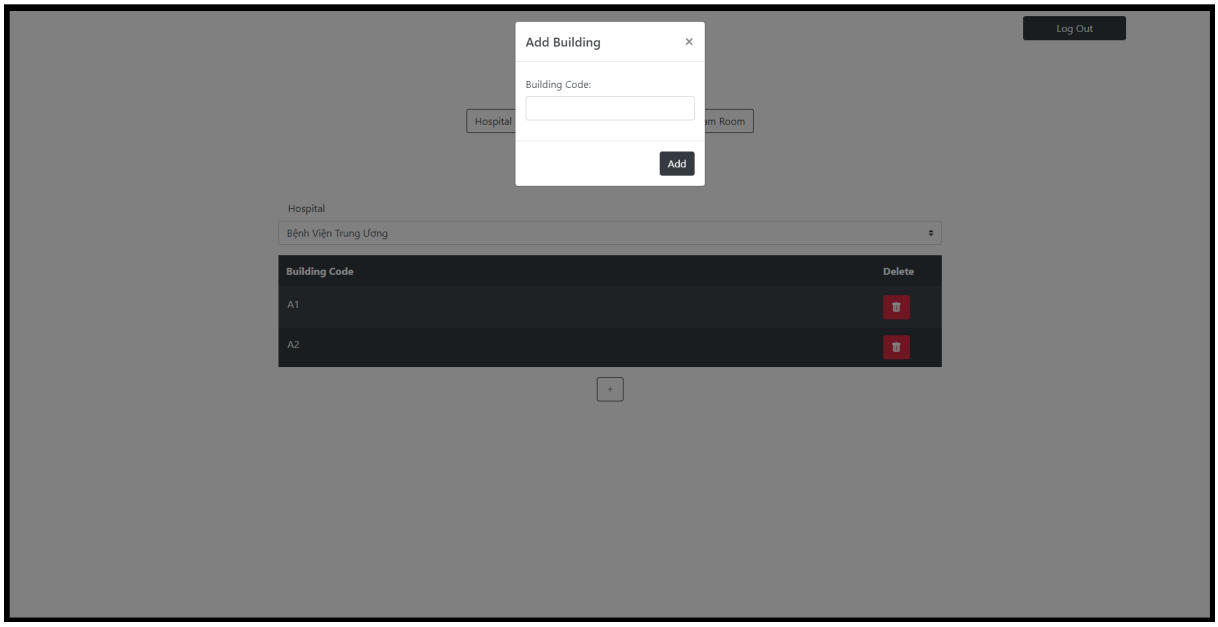

Figure 4.83: Add new building/Input new building information

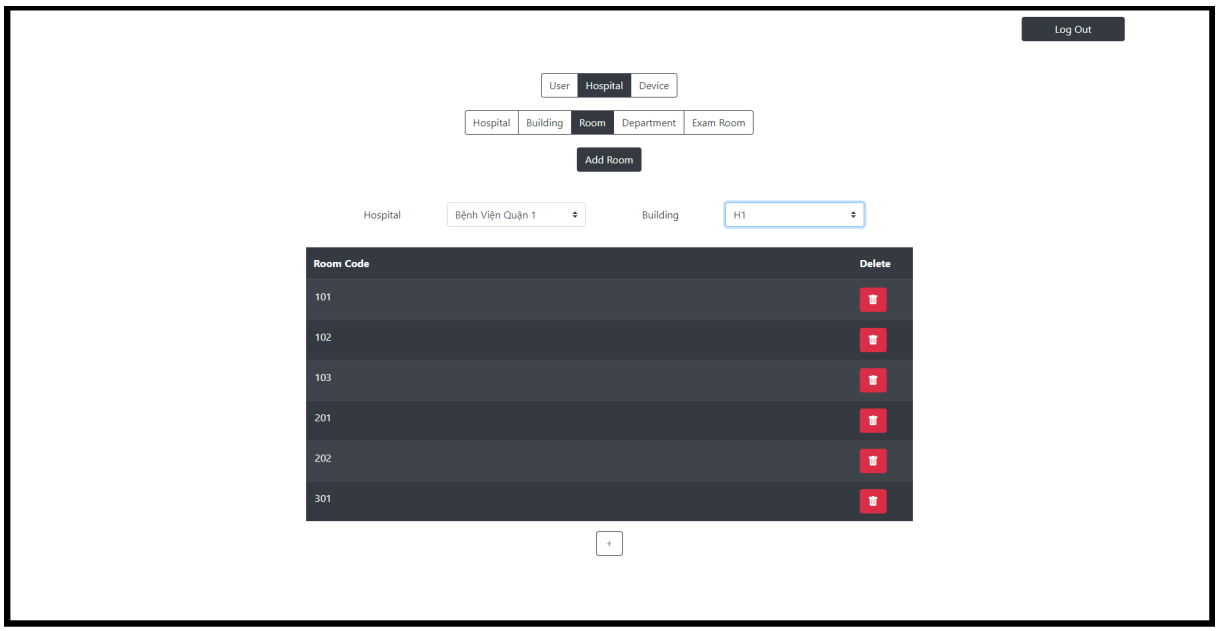

Figure 4.84: View room of a building in hospital

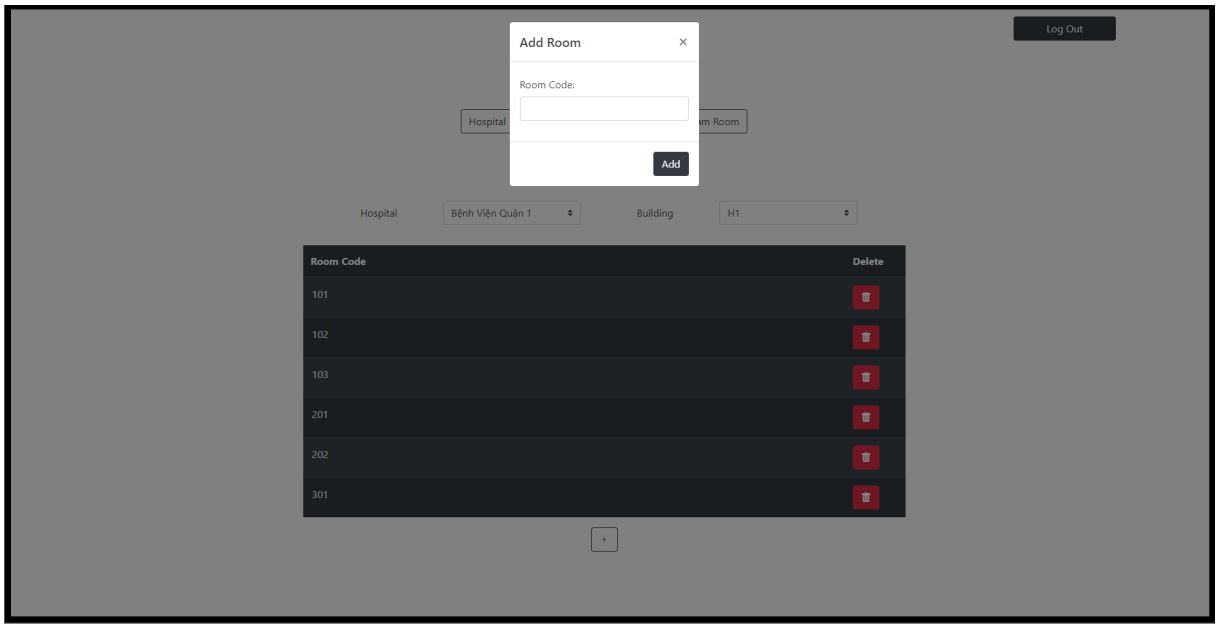

Figure 4.85: Add new room/Input new room information

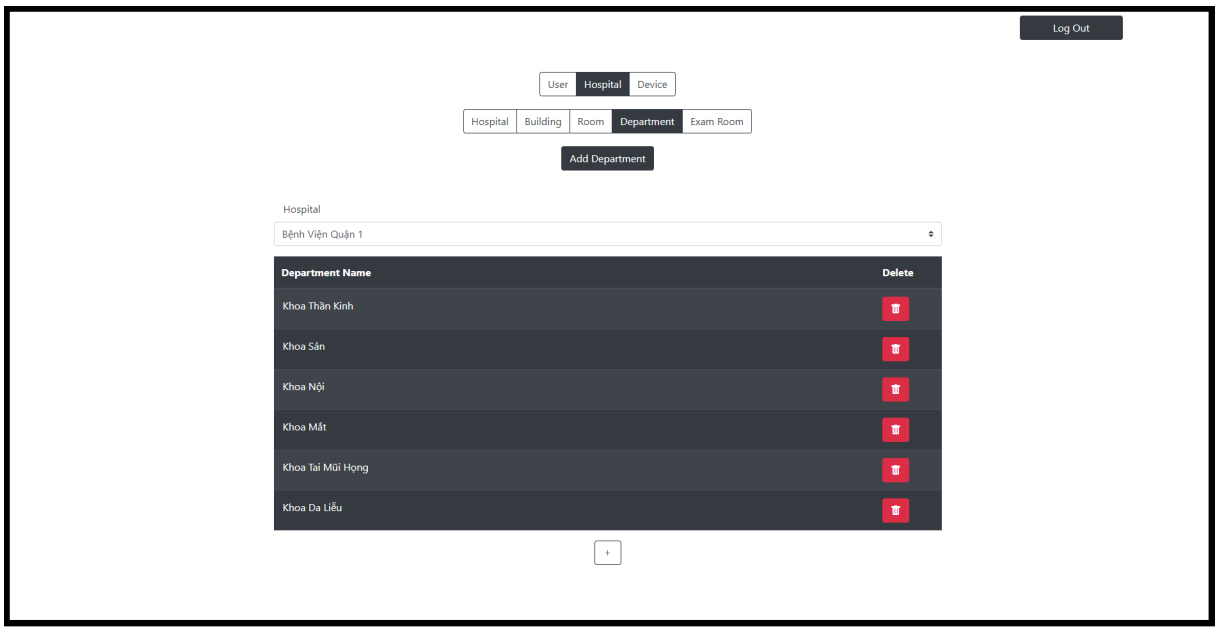

Figure 4.86: View departments of a hospital

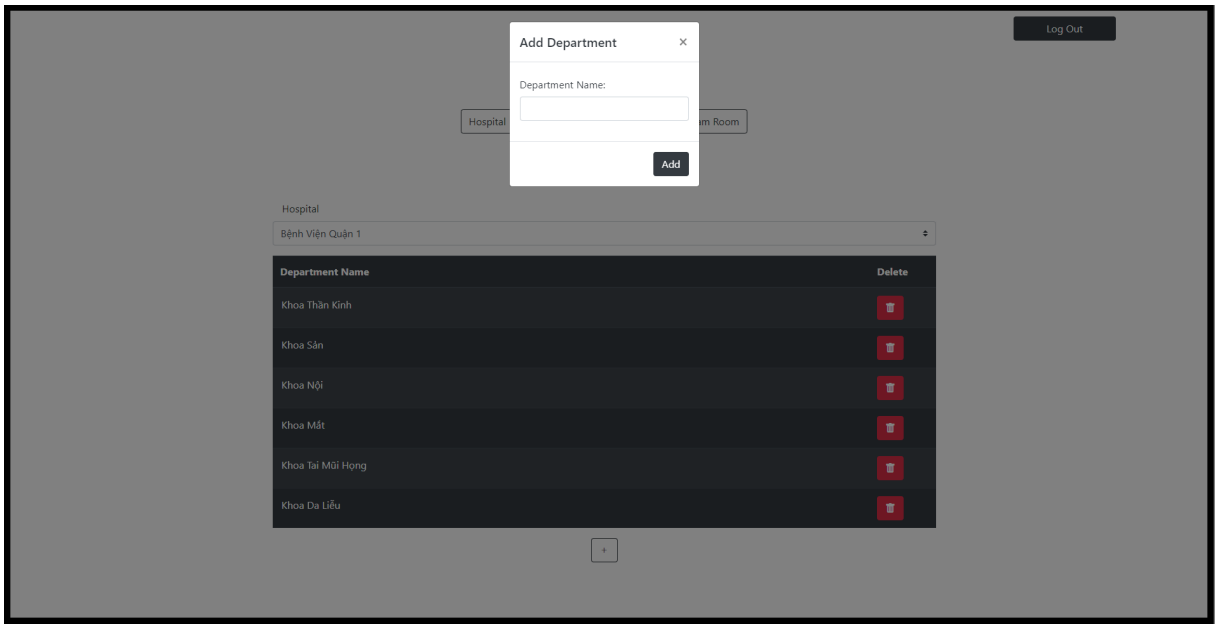

Figure 4.87: Add new department/Input new department information

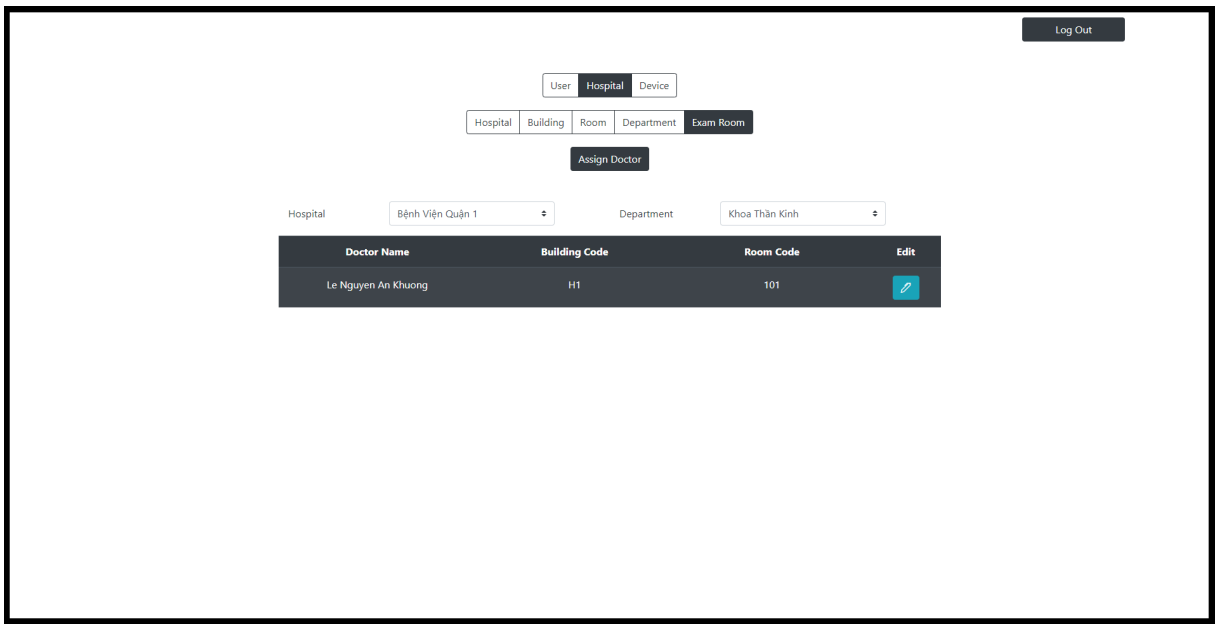

Figure 4.88: View doctor-room assignment

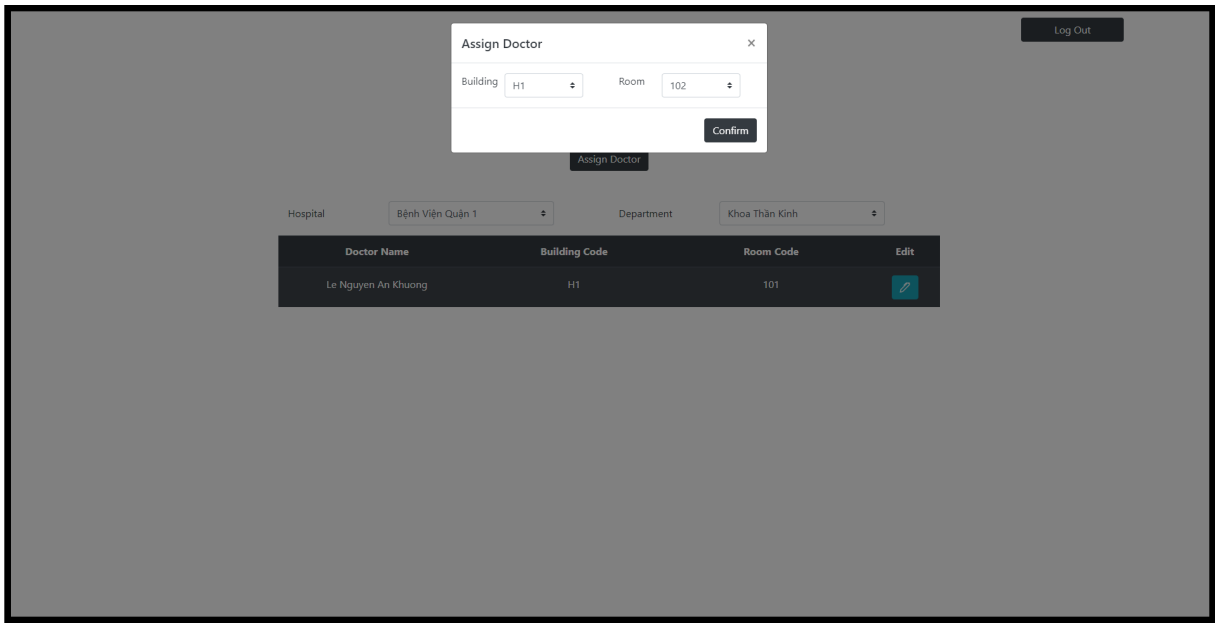

Figure 4.89: Assign/reassign examination room for doctor

Figure 4.90: Add new device to a hospital

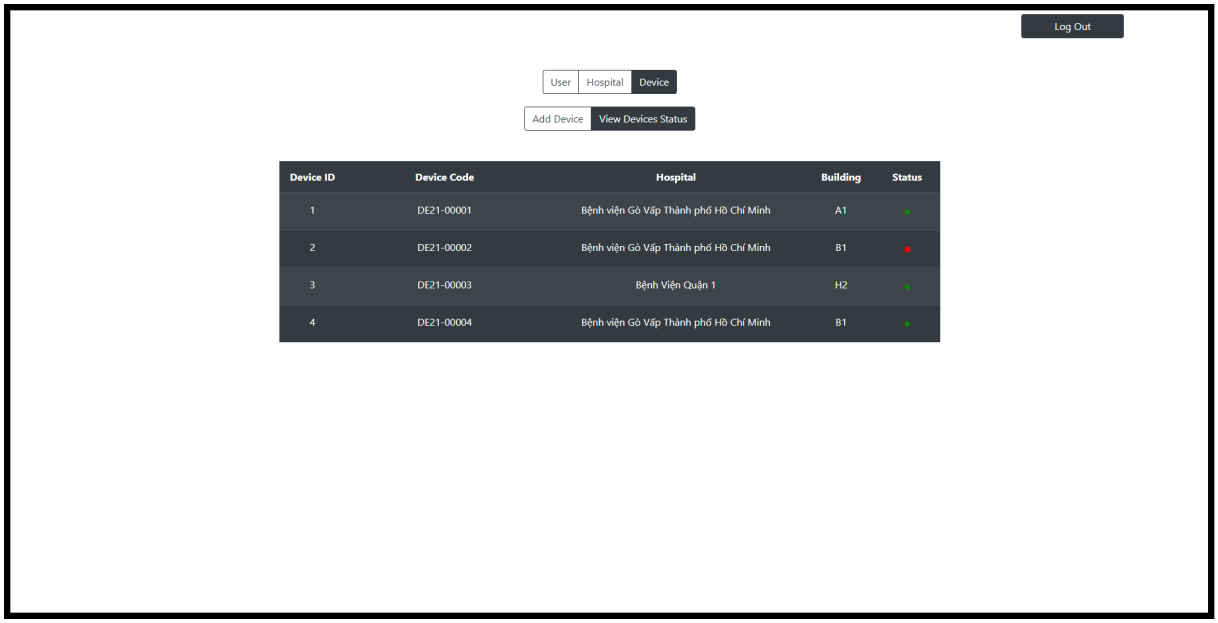

Figure 4.91: View all devices and status

4.6 Mobile Application

4.6.1 Server

For the server-side, we use the same method as the web application for simplicity and reusable reasons. In this server, we implement it as an API archetype. Meaning the mobile application will fetch the data through an

URL which is the API of our server. The specification will be explained below:

- **exam-hisotry (parameters: patientId):** This keyword will command the server to retrieve the information of all the previous examination that the patients took and transfer it to the application
- **auth (paramenters: email/username, password):** This keyword will command the server to authenticate for the user with the input information for login.
- **get-queue (paramenters: patientId):** This keyword will command the server to retrieve the queue of the current patient and callback the data of the room, and building the patients' examination will take place.

4.6.2 Mobile Application

The application contains of two parts: Data processor and view display

- Data processor: The function of this class main purpose is to fetch the information from the server which will be used to display
- View display: Take the processed data and display on the screen.

Moreover, when it is the patients turn to meet the doctor, The sms contains the information of their order number, hospital building code and room code will be sent to the patient phone.

4.6.3 Result

User Interface

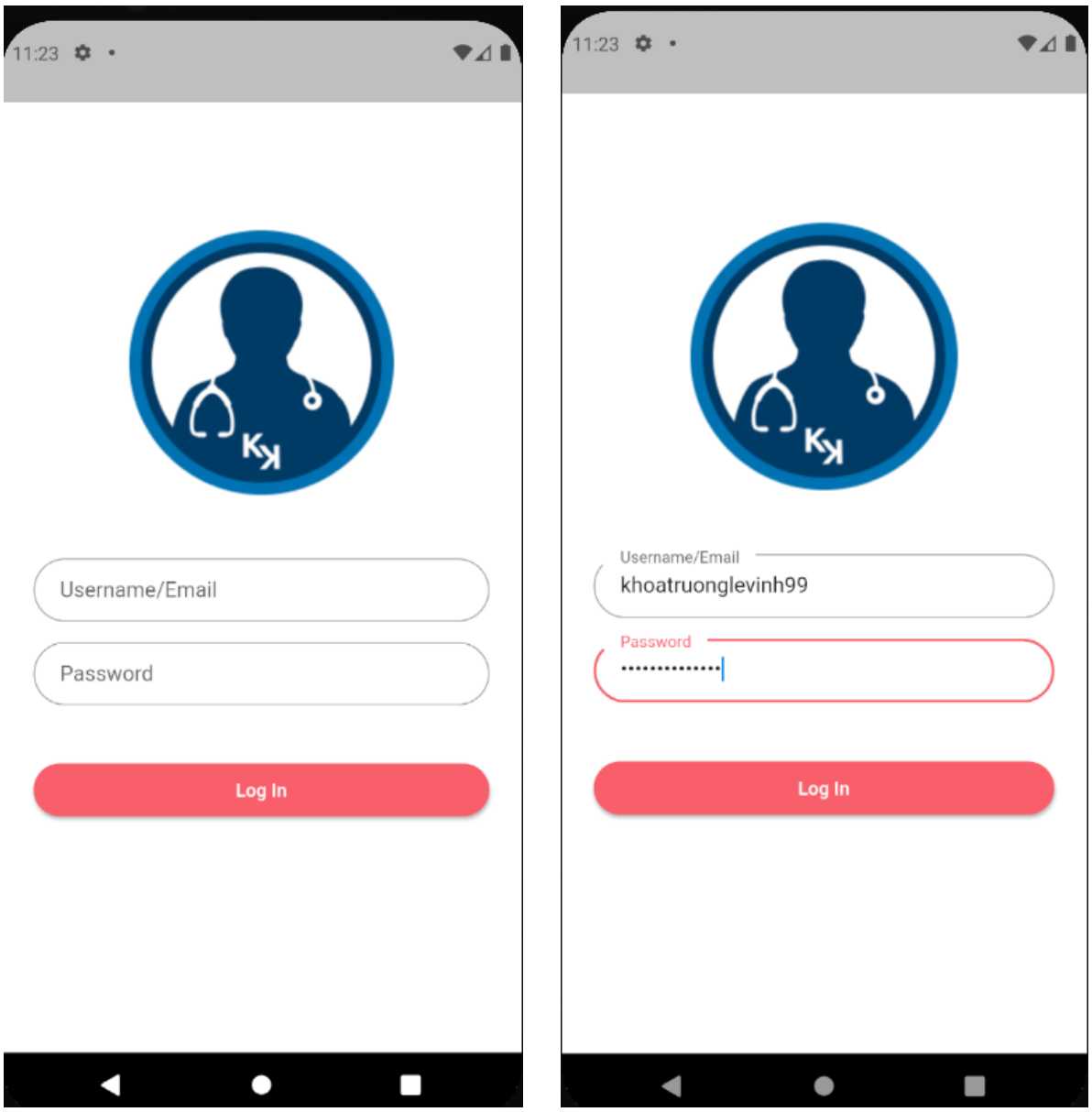

Figure 4.92: Login Screen Mobile Application

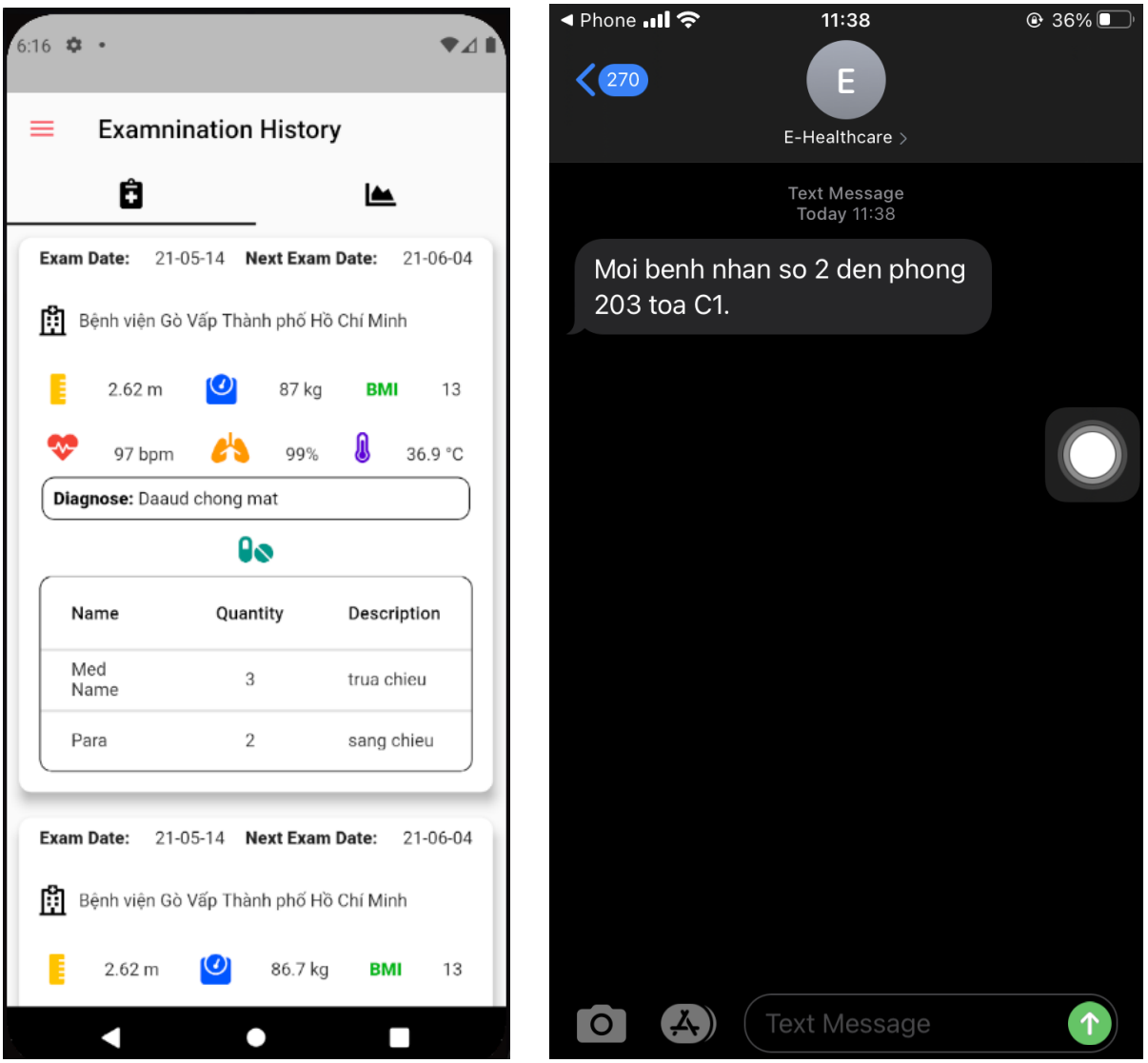

Figure 4.93: History Examination and Notification Mobile Application

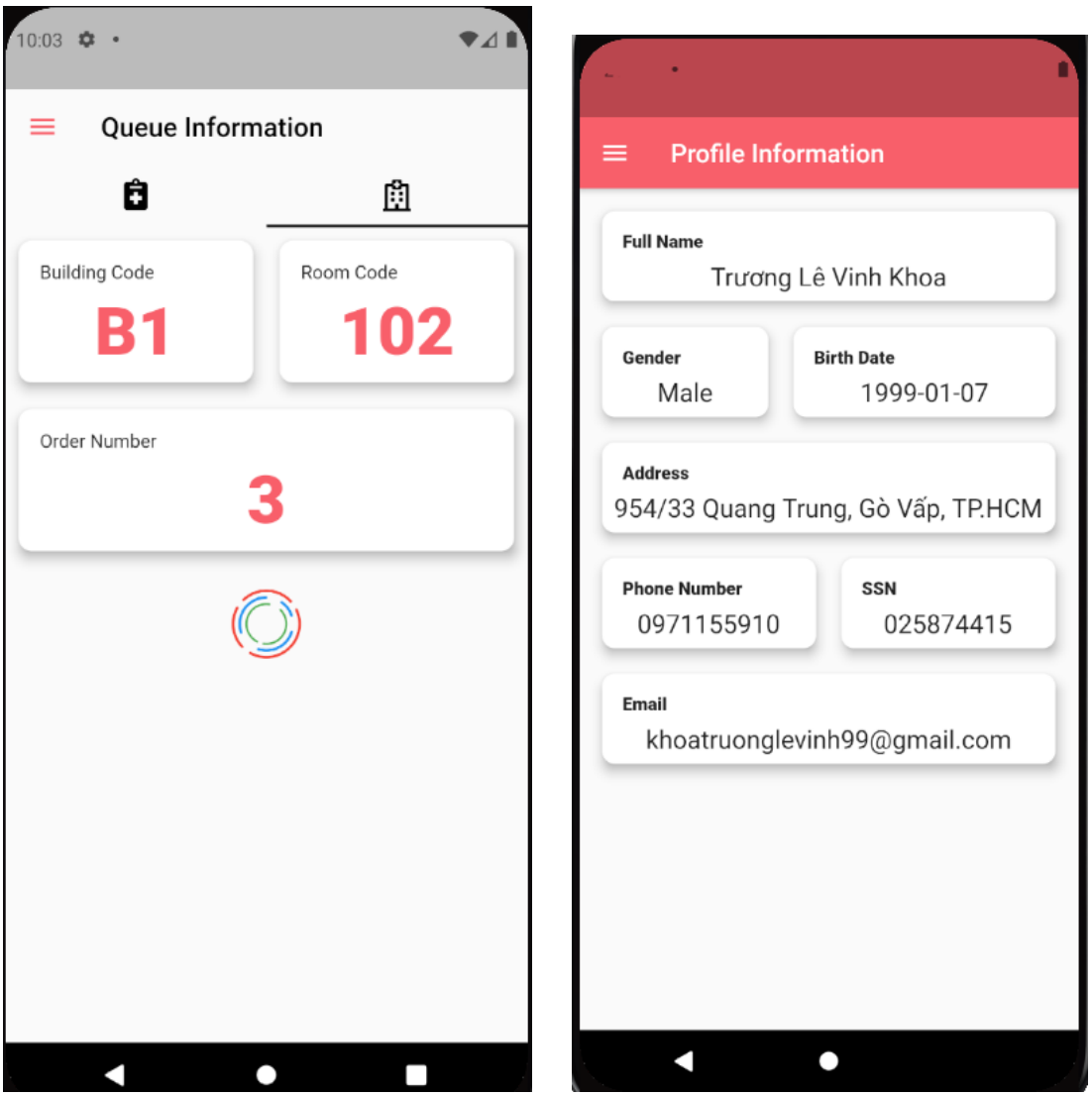

Figure 4.94: History Examination and Notification Mobile Application

CHAPTER 5

RESULT AND CONCLUSION

5.1 Result

5.1.1 Accuracy

To measure the accuracy of our face recognition model, we use the LWF dataset with reliable data labeled. The data set contains more than 11,678 images of faces collected from the web. Each face has been labeled with the name of the person pictured. 1680 of the people pictured have two or more distinct photos in the dataset.

To represent the evaluation function, we used statistical terms, including mean, standard deviation (Std.), minimum (Min), and maximum (Max) values on Euclidean-Distance (ED) result of the 128-Dimensional vector after encoding by our model. However, before going to the experiment step, we have to select some suitable parameters for the Classify Model for achieving the highest result.

There are two kinds of parameters that we must estimate is the appropriate DISTANCE_THRESHOLD and the parameters of the Classify Model. The DISTANCE_THRESHOLD is the essential parameter. It is used as the threshold to determine whether that image belongs to the current checking patient in the database when we perform the classification step.

- If the DISTANCE_THRESHOLD is too high, we will have a higher **false positive error** on our model. It means that if the invalid pa-

tient has a distance of 0.5 and DISTANCE_THRESHOLD is 0.55. The system will wrongly predict that patient.

- If the DISTANCE_THRESHOLD is too low, we have another problem that the **false negative error** is high, and the system will miss many valid patients. If the distance of the valid patient ranges from [0.4 - 0.5] and the DISTANCE_THRESHOLD is 0.45. We have to identify approximately two times to recognize this patient.
- Therefore, we have done the experiment that finds the range distance of true prediction and false prediction. From that, we can choose the most suitable DISTANCE_THRESHOLD for the Classify Model. The Table 5.1. below is the result of the experiment.

	Distance True Predict	**Distance False Predict**
Mean	0.347	0.522
Max	0.496	0.55
Min	0.082	0.326
Std	0.063	0.048

Table 5.1: The result of range distance of true prediction and false

Based on the result of the above experiment, we decide to choose DISTANCE_THRESHOLD is 0.45 and parameters of the Classify Model as described in Table 5.2 to calculate the accuracy of the Identification Model.

Number of neighbors	10
KNN algorithm	ball tree
KNN weight	distance
Distance Threshold	0.45
Input	vector 128-dimension
Output	patient ID

Table 5.2: KNN-Classify Model Parameters

Table 5.3 and Table 5.4 are the results of the KNN Model when we test in the known users and unknown users dataset. It includes accuracy, precision, recall score, and some information of the dataset. However, we only consider the accuracy score for the unknown user dataset to reduce the probability of predicting the user when they do not exist in the database.

	KNN Model
Image Shape	(150, 150, 3)
Number of known user images	3870
Precision score known user	0.990
Recall score known user	0.856
Total processing time known user images (s)	1.732
Max processing time known user images (s)	0.0060
Min processing time known user images (s)	0.0015
Mean processing time known user images (s)	0.0018
Std processing time known user images (s)	0.0003

Table 5.3: Statistic Result of KNN Model with known user

	KNN Model
Image Shape	(150, 150, 3)
Number of unknown user images	7808
Accuracy score unknown user	0.050
Total processing time unknown user images (s)	17.817
Max processing time unknown user images (s)	0.0060
Min processing time unknown user images (s)	0.0016
Mean processing time unknown user images (s)	0.0022
Std processing time unknown user images (s)	0.0004

Table 5.4: Statistic Result of KNN Model with unknown user

For the tests with the known users, the system has 99 percent corrects over all their predictions. Instead, we have only 85.6 percent *Recall score known*

user. It means that the system has 14.4 percent to reject the correct users.

With the *Accuracy score unknown user* of 0.05 in testing with the unknown user, we can understand that the system has only 5 percent to predict a user when they do not exist in the system.

Measurement range	Pressure: 0 - 299 mmHg; Pulse: 40 - 180 beats/min
Accuracy/Calibration	Pressure: ±3mmHg (2%); Pulse: ±5%
MLX90640	$\pm 0.3°C$
TF mini	±2.5 cm
HX711	±0.5kg

Table 5.5: Accuracy of Biometric Extraction Module

5.1.2 Time Processing

5.1.2.1 User Recognition

We adopted the exact mechanism for evaluating CPU and RAM utilization (i.e., run our model in real-time). We measured the processing time for each iteration. However, we separate processing time into detecting time, identifying time, and evaluating the result at the end of the examination. Table 5.5 summarizes the experimental results between two tasks, Detecting and Representing Face. As can be seen from the result, this model can perform around 20-22 FPS and be considered real-time.

	Face Detection	**Face Representation**
Mean (ms)	14.83	31.60
Max (ms)	15.82	36.78
Min (ms)	13.90	28.14
Std (ms)	1.06	2.90

Table 5.6: The experimental results between two tasks, Detecting and Representing Face

5.1.2.2 Classifying User In Server

Unlike measuring the processing time of Face Detection and Face Representation, we calculate the Round-Trip Time (RTT) of the patient identification request sent to the server by taking the differences between sending and receiving one request. Table 5.6 summarizes the experimental results when sending and receiving the response of the patient identification request.

	Round Trip Time
Mean (s)	0.957
Max (s)	1.155
Min (s)	0.860
Std. (s)	0.096

Table 5.7: The experimental results when sending and receiving the response of the patient's identification request

5.1.2.3 Biometric Measurements Extraction

The manufacturer tests the accuracy in measurement of the heart rate and oxygen saturation as below:

Min Measure Time(s)	13.4 seconds
Max Measure Time(s)	16.7 seconds
Mean (ms)	15.05 seconds

5.1.3 CPU and RAM Utilization

We run our model in real-time and record elapsed time after one iteration, including detecting and identifying face to evaluate resource utilization. We monitored the utilization of 4 CPUs and RAM on Jetson Nano while performing our model. Figures 5.1, 5.2 visualizes the experimental results in terms of CPU and RAM.

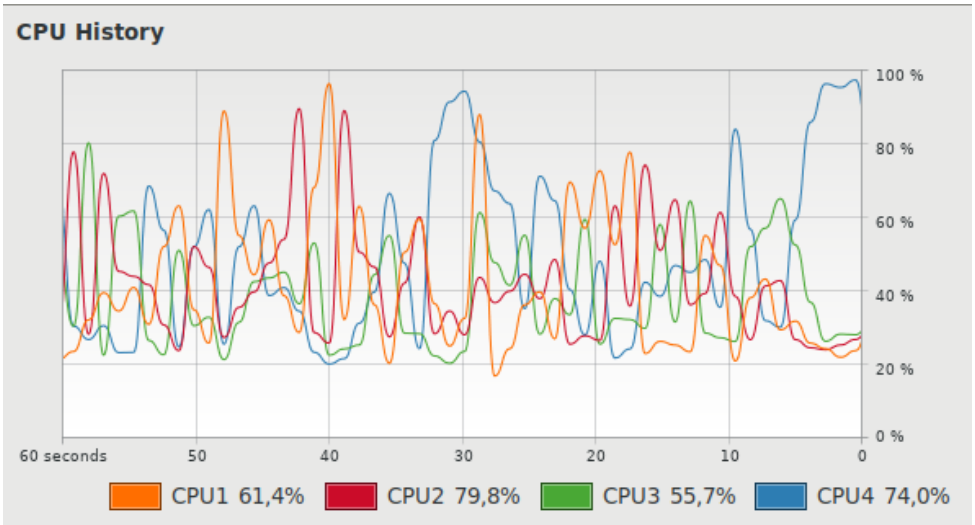

Figure 5.1: The CPU Utilization of the System

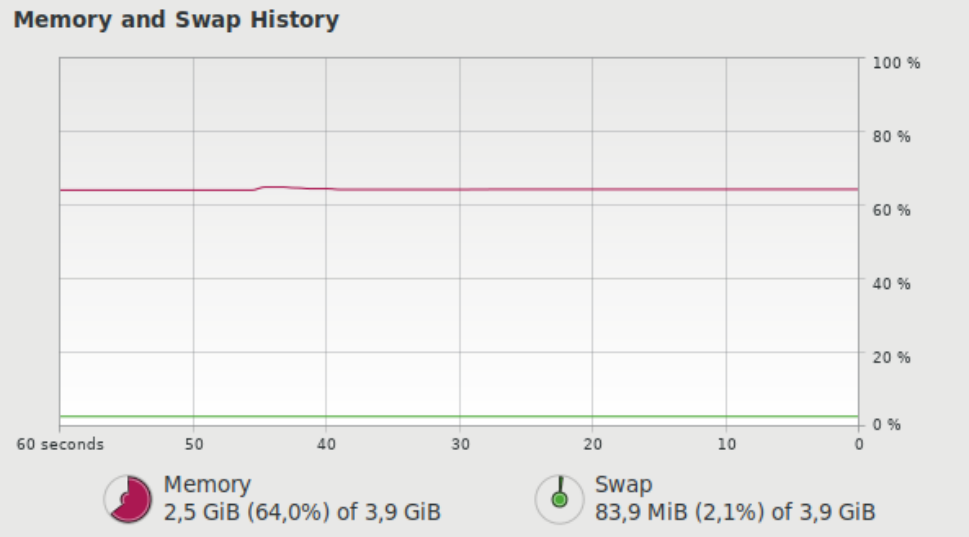

Figure 5.2: The RAM Utilization of the System

5.2 Conclusion

This chapter summarizes the thesis's progress by specifying complete tasks, our challenges, and the plan to develop our system in the future.

5.2.1 Thesis Summarize

Our thesis has completed designing and implementing the system to help accelerate the hospital's check-in and screening process. Furthermore, the system also helps patients monitor their health status by reviewing the pre-

vious visits through a mobile application. It also helps minimize face-to-face interaction between patients and healthcare staff, which is necessary for the hospital environment.

The result we get executing the system is pretty great, with an entire process of checking in nearly 4 minutes. However, there are several difficulties and unwanted drawbacks, which we will discuss in the next section.

5.2.2 Challenges

During the progress of researching, designing, and implementing the system, our group has encountered some difficulties:

- Due to the Covid-19 pandemic, we do not have the opportunity to visit the hospital and collect the time data of each examination step.
- Secondly, for the department recommendation model, we do not have enough data to examine and train our model. Therefore, we have collected the data by visiting the official medical website to get as much data as possible.
- The next challenge is to ensure data transmission between the Device Server and the Check-In system to protect the privacy of users' information.
- About the sensor, as the separation between OSO2 and ESP32 device, the Capturing Biologic Parameters step of the system is quite uncomfortable and confused for a non-tech patient. Furthermore, the accuracy of temperature measurement is affected by many factors: ambient temperature, humidity, distances, etc. Therefore, the calibration procedure requires adding more metrics, testing under different surrounding conditions to produce the correct offset value.
- Finally, we do not successfully build a complete Check-in system as the original plan since the absence of resources and materials. Therefore, it makes use hardly to retrieve precise values of sensors, such as in getting the height value of a patient, since we use LiDar to measure. It works well on a flat surface, and it is difficult for us to collect a correct value

when this LiDar measures the height by taking the reflexed beam on the patient's head(unflat surface).

5.2.3 Future Plan

- Our first and foremost plan after the Covid-19 pandemic is to check in the hospitals and collect the statistical information for comparing our system in the check-in step.
- Secondly, as our department recommendation system works in some specific scenarios as described in section 4.3.4, we have to enhance the system to adapt with more type conservations. To overcome these difficulties, we need to collect more useful medical data and enhance our system to give more reliable recommendations about the department.
- Next, to help the patient feel easier when interacting with the system, we will increase the UX/UI of the system.
- Finally, we have to build a complete product about functions and appearance. We integrate all the sensors as the union controller to communicate with the mainboard. Then, we build a suitable appearance for the system to help collect sensor values as precisely as possible.

APPENDIX A

SET UP GUIDE AND HOW TO USE

A.1 Microsoft Azure Setup

A.1.1 Event Hub

First, at the resource group, we choose to create an Event Hub with our demand tier.

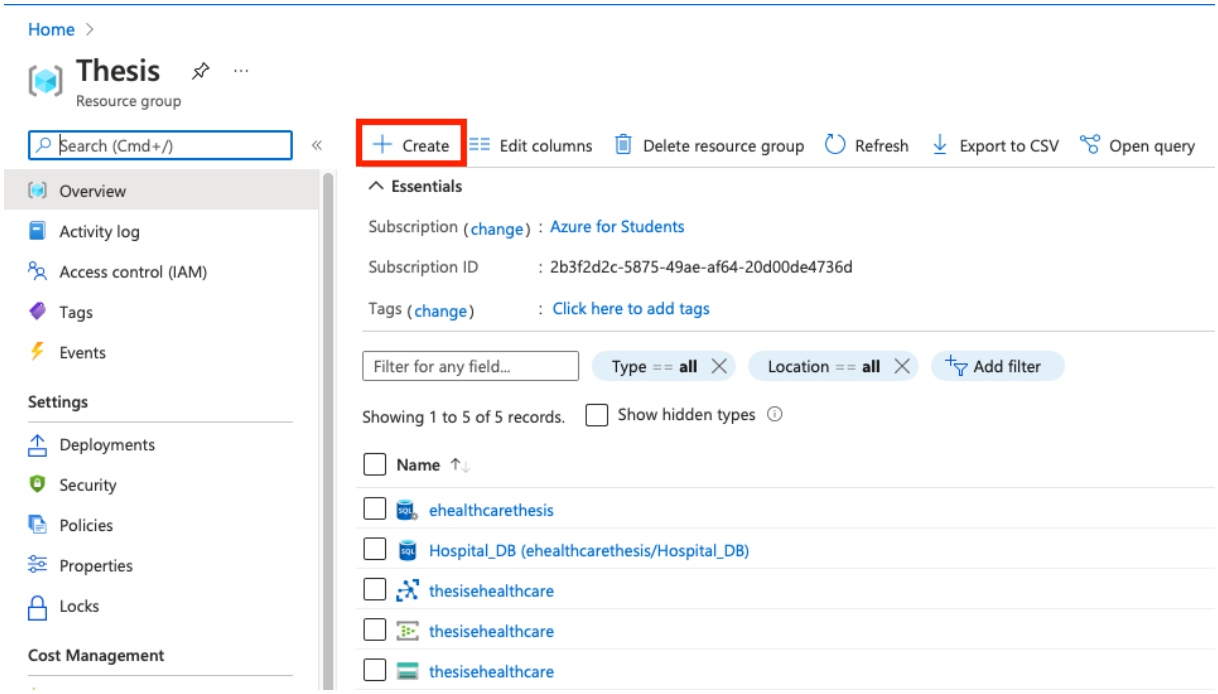

Figure A.1: Event Hub Creation

After creating and deploying the Event Hub successfully, we have a namespace. To send and receive a message from the Event Hub, we have to create a specific one below. Here we create a hub name "receivemsg". This hub is

used as a means of communication between a cluster Device Server and other Check-in devices.

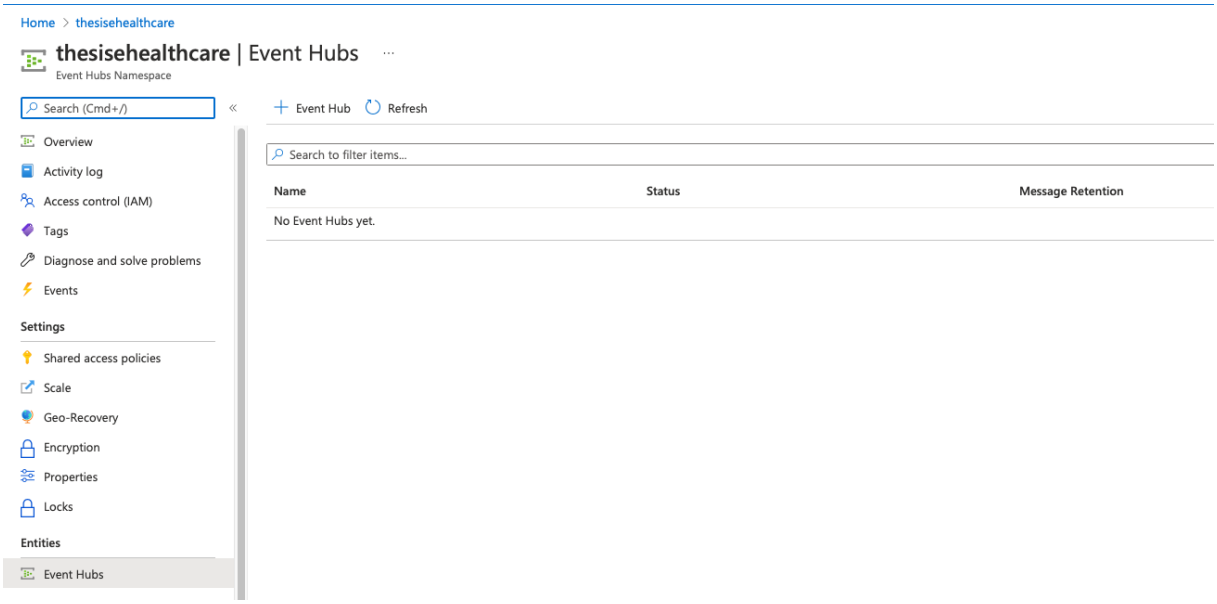

Figure A.2: Event Hub Creation

Here, we can choose the specific partitions for our demands. Then after done all the above step, we access the *Shared access policies* tab, get the *Connection string–primary key* as *eventhub_connection*, and use *receivemsg* as the *eventhub_name* for the parameters connection of event hub.

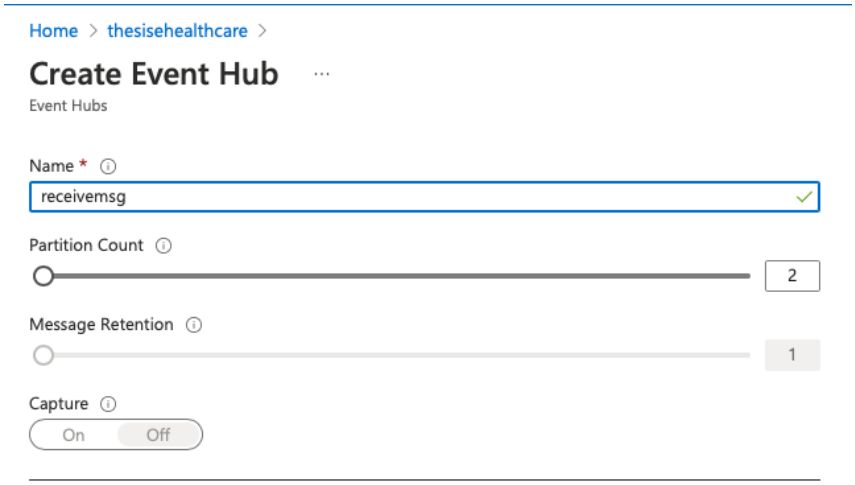

Figure A.3: Event Hub Creation

Next, we create a container in the Storage of the resource group to store all the events coming to the Event Hub. Get the Storage connection string in the *Access Keys* and the *container* name in the Containers as displayed in

Figure 6.4. Then we replace them in the file *config.yaml* similar to the above instruction.

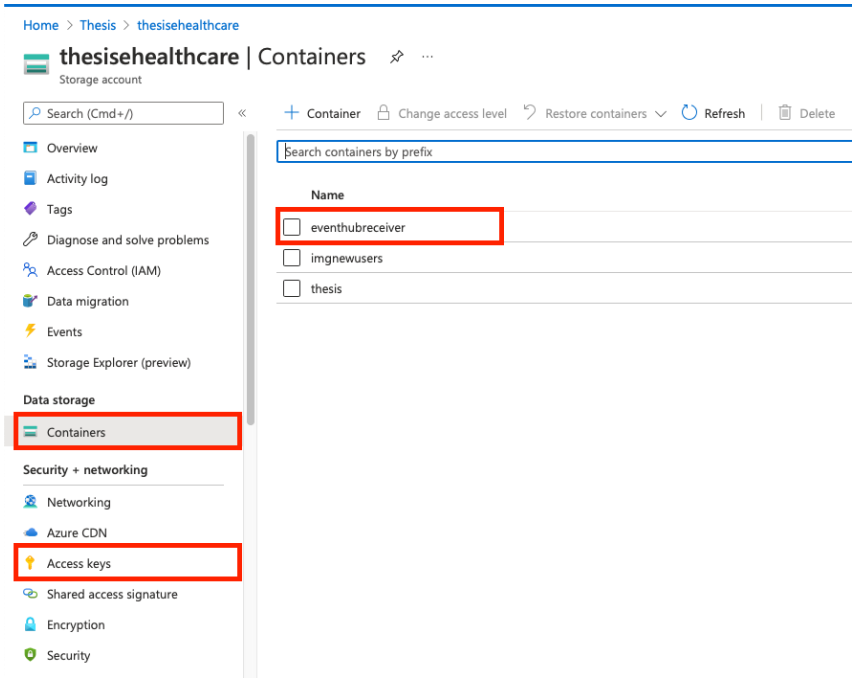

Figure A.4: Blob Storage Creation

A.1.2 IoT Hub

Similar to the Event Hub, we first create an IoT Hub in our resource group. Then at the *Shared access policies* in the IoT Hub, we get a *iothub_ connection* by creating the the *shared access policy* and get the *Primary connection string*. Depending on how we grant rights for this connection at the beginning of creation, the server or device which uses this connection might read, write or connect to the IoT Hub.

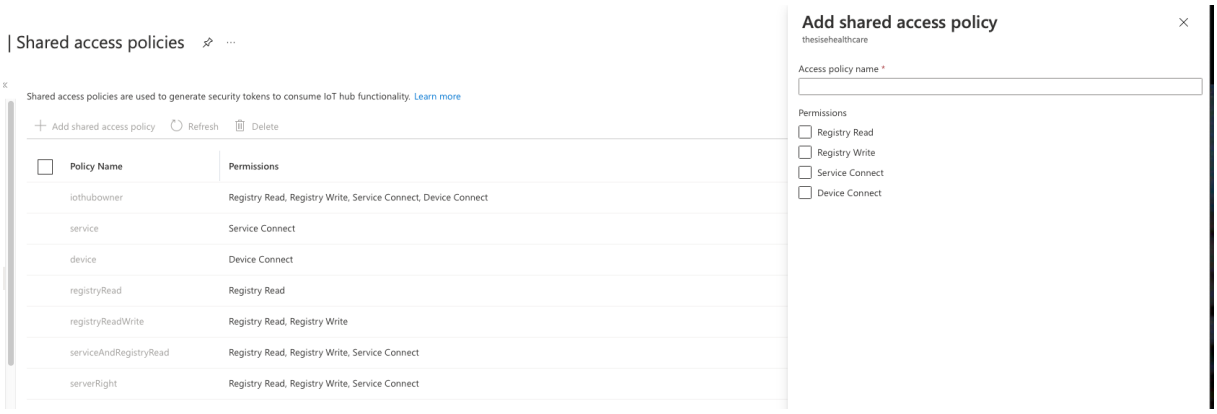

Figure A.5: Event Hub Creation

Such as, in this project, we create a policy named *serverRight* for the Device Server with these rights, read, write, and connect to the IoT Hub. This policy contains a *iothub_connection* that Device Server uses to access and manage the IoT Hub with all the rights set above. The detail is displayed in Figure 6.5.

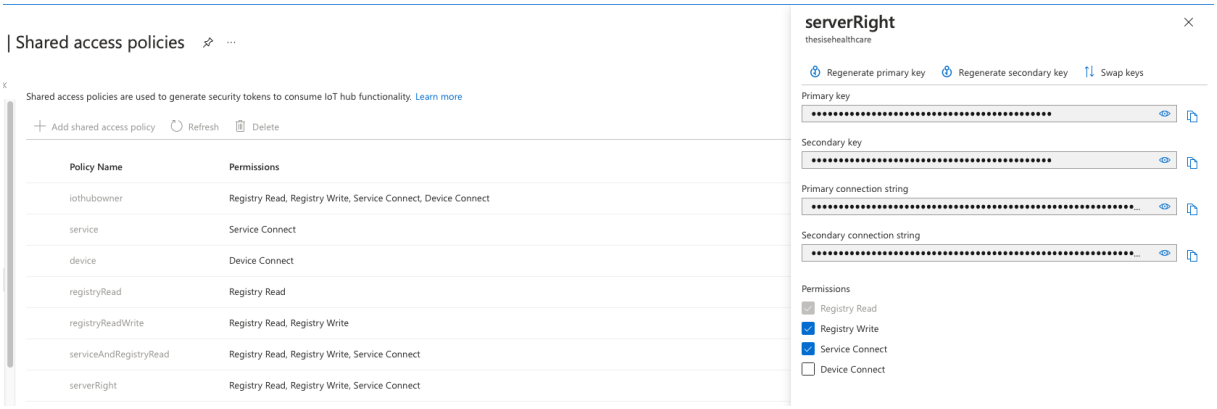

Figure A.6: Event Hub Creation

After being created at the Web Application, the information of the Check-In device is added to the database. Currently, its connection to the IoT Hub is also created.

When we initiate the device for the first time by running the script in the source, the connection to the Event and IoT Hub is loaded and stored locally. This information will be used to communicate with the Device Server at the running time. However, if we want to manage these devices, we can access this page or use the Web Application.

Additional, we have to create a connection named *serviceAndRegistryRead*. The reason is that, at the first time, we do not have a connection for this device to access the IoT Hub. Therefore, we create a common connection for all the Check-In devices use to connect the IoT Hub at the beginning and load their connection.

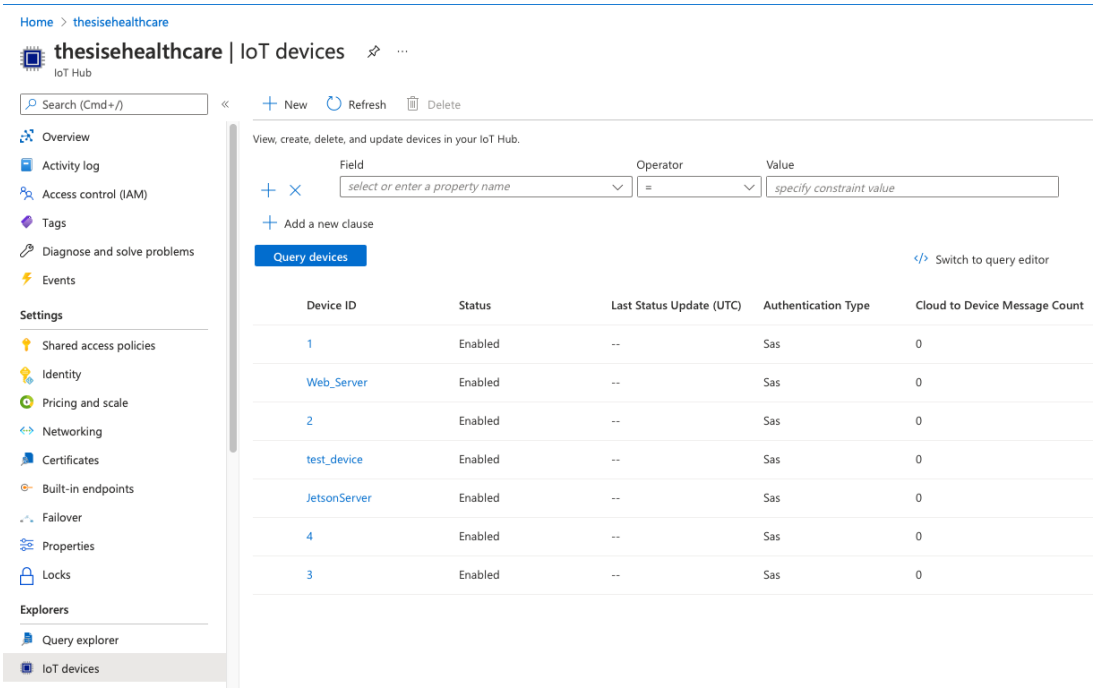

Figure A.7: Event Hub Creation

A.1.3 Database Setup

Similar to the above 2 services, we create a SQL Server named *ehealthcarethesis* at the resource group to store our Database. Then we move to the SQL server page and create a *Hospital_DB* Database for our system, as displayed in the Figure 6.8.

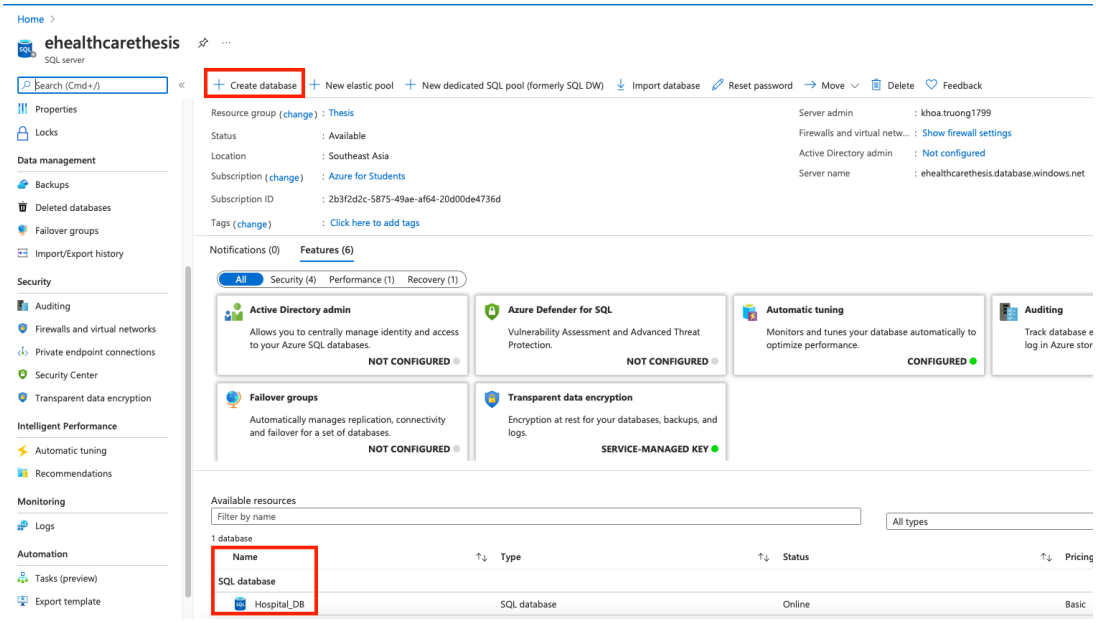

Figure A.8: SQL Server and Database Creation

After that, we connect to the Database with these values of database_server, database_user, database_password, and database_name at the current *Hospital_DB* page. With these fields, the Device Server can set up a connection with our Database.

A.1.4 App Service Setup

First we need to create a host on Azure to deploy our server. We will get our own domain name in the future (that part is unnecessary since the web application is using internally, its own domain name is not required).

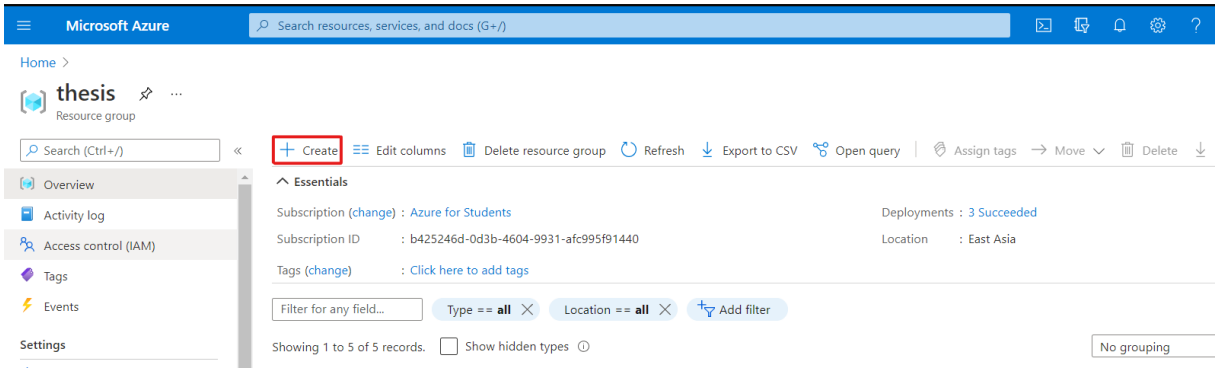

Figure A.9: App Service Create

After creating a slot for deployment, we will now connect the Github repository of the source code to the server, and the App Service will do all the hard work for us.

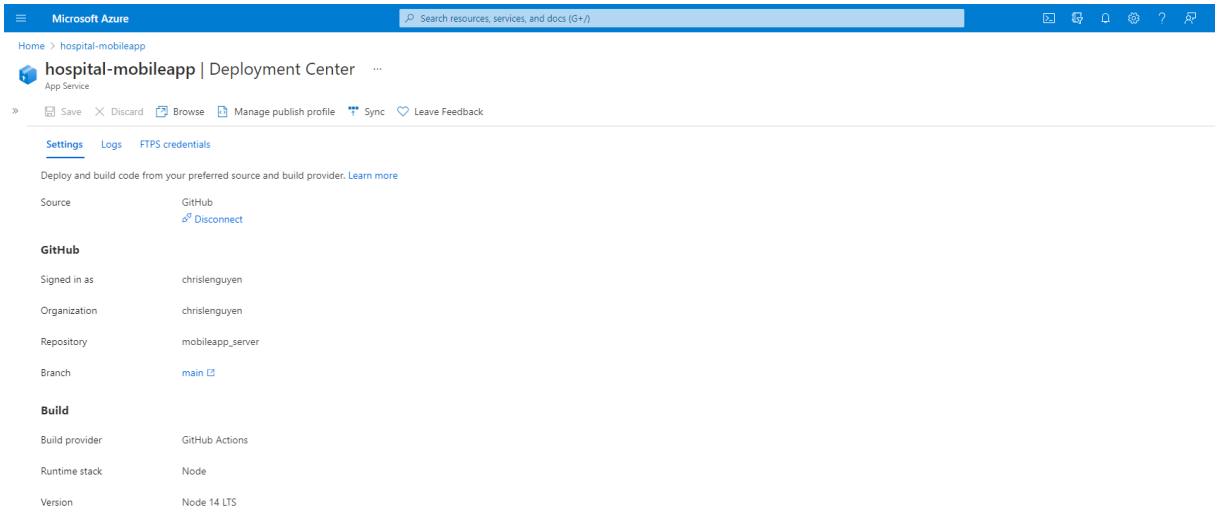

Figure A.10: App Service Deployment

A.2 Check-In Device Set Up

This part will explain how to set up the Jetson Nano hardware and software in as detail as we can for a better approach and inheritance. Before going to the detail of this section, we have to set up the Jetson Nano connection with other components. The below Figures 6.1, 6.2, and 6.3 are the setup connection of ESP32 with other sensors and the setup of Jetson Nano, respectively.

A.2.1 ESP32 Set up

The below is the connection between micro-controller and all the sensors

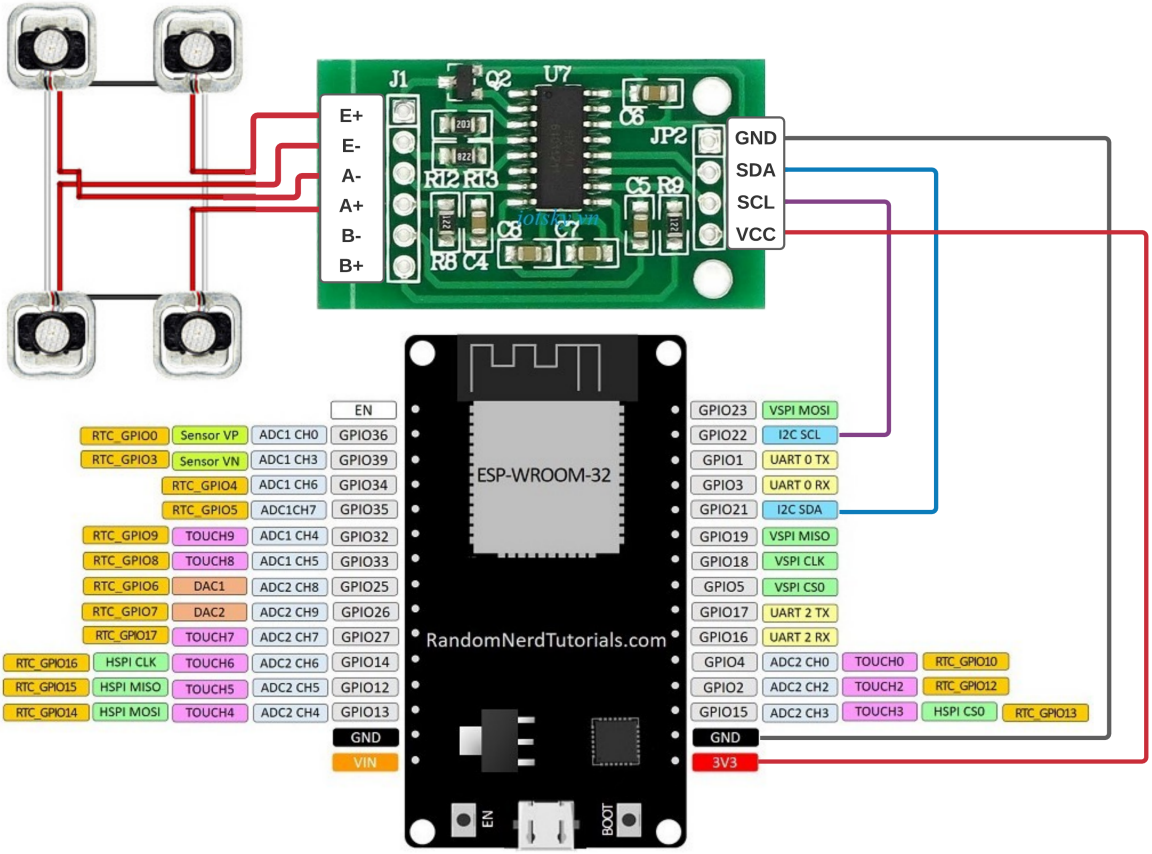

Figure A.11: Connection Schema Of Module Reading weight Sensor

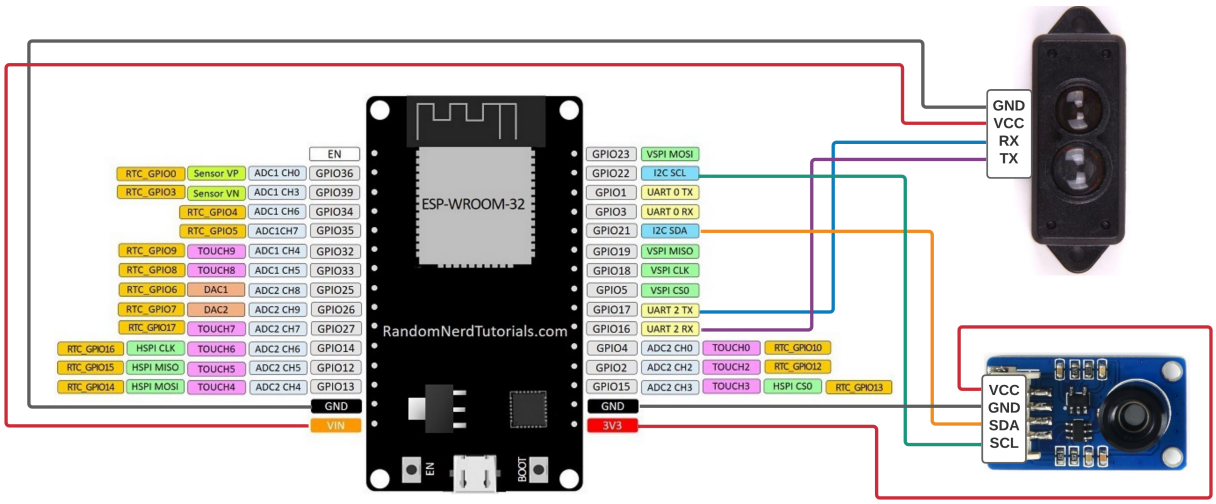

Figure A.12: Connection Schema of Module Reading Temperature and Height Sensor

A.2.2 Jetson Nano Set Up

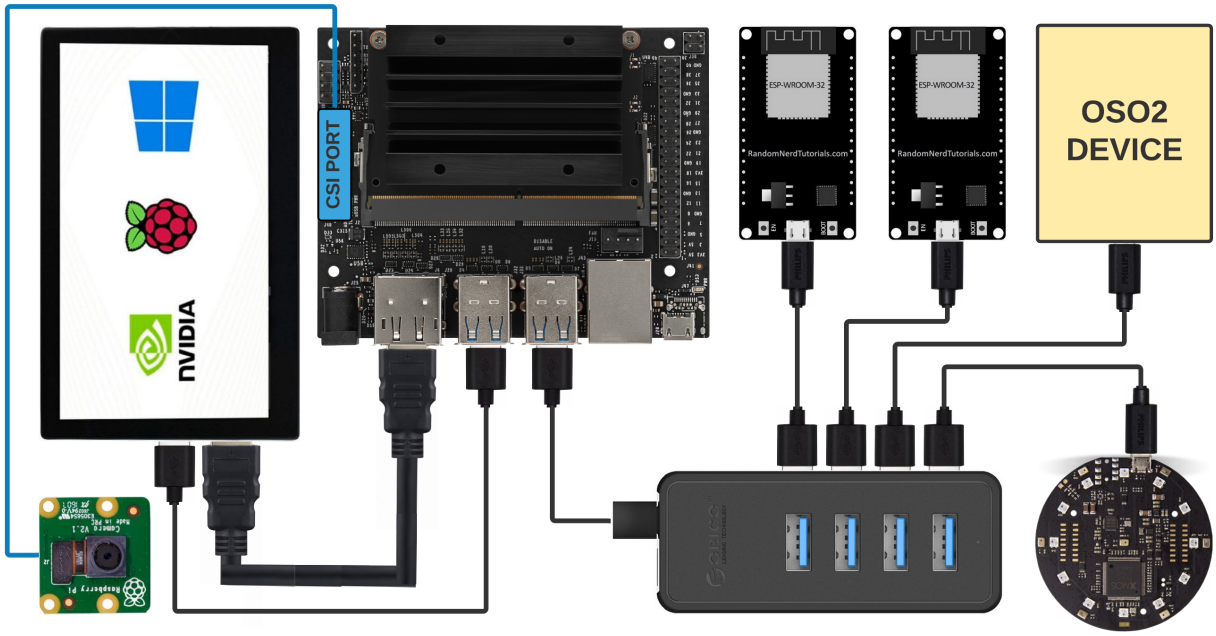

Figure A.13: Local Device Setup

The below section display the connection type between the Jetson Nano with its components:

- LCD: HDMI for displaying and USB for touching.
- CSI Camera: CSI connection
- ESP32: USB connection

- OSO2 Device: : HID connection
- Mic Array Speaker: USB connection

A.2.3 Jetson Nano Configuration

A.2.4 Jetson Nano Software Installation

A.2.4.1 Install Jetpack

First and foremost, we have to install a specific Jetpack that is compatible with the system. The below is the link for downloading the Jetpack 4.5.1 of the Jetson Nano. **https://developer.nvidia.com/embedded/jetpack** After downloading the Jetpack and flash the .img file to the micro-SD card, we insert the card into the micro-SD card slot and follow the installation on the screen.

A.2.4.2 Install system-level dependencies

After install and boost the OS on the Jetson Nano, we have to install an essential software package for the device.

```
sudo apt-get install git cmake
sudo apt-get install libatlas-base-dev gfortran
sudo apt-get install libhdf5-serial-dev hdf5-tools
sudo apt-get install python3-dev
sudo apt-get install nano locate
sudo apt-get install libfreetype6-dev python3-setuptools
sudo apt-get install protobuf-compiler libprotobuf-dev openssl
sudo apt-get install libssl-dev libcurl4-openssl-dev
sudo apt-get install cython3
sudo apt-get install libxml2-dev libxslt1-dev
sudo apt-get install libpq-dev
```

A.2.4.3 Install Virtual Environment

Before going to install these packages, we have to install a virtual environment for python. We need to delete an old environment and create a new one for our system in some circumstances.

```
wget https://bootstrap.pypa.io/get-pip.py
sudo python3 get-pip.py
rm get-pip.py
sudo pip install virtualenv virtualenvwrapper
nano ~/.bashrc
```

Once we have installed virtualenv and virtualenvwrapper, we need to update our /.bashrc file. We are choosing to use nano, but we can use whatever editor we are most comfortable with. Scroll down to the bottom of the /.bashrc file and add the following lines:

```
export WORKON_HOME=\$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh
```

After adding the above lines, save and exit the editor. Next, we need to reload the contents of the /.bashrc file using the source command:

```
source ~/.bashrc
```

Now we create our virtual environement and work on this.

```
mkvirtualenv thesis -p python3
workon thesis
```

A.2.4.4 Install OpenCV and Other support Libraries

Install the protobuf compiler to boost up performance of Tensorflow:

```
mkdir \${HOME}/project
cd \${HOME}/project
git clone https://github.com/jkjung-avt/jetson_nano.git
cd jetson_nano
./install_basics.sh
source \${HOME}/.bashrc
sudo apt update
sudo apt install -y build-essential make cmake cmake-curses-gui \
                     git g++ pkg-config curl libfreetype6-dev \
                     libcanberra-gtk-module libcanberra-gtk3-module \
                     python3-dev python3-pip
sudo pip3 install -U pip==20.2.1 Cython testresources setuptools
cd \${HOME}/project/jetson_nano
./install_protobuf-3.8.0.sh
sudo pip3 install numpy==1.19.4 matplotlib==3.2.2
```

Install numpy and scipy:

```
#install numpy
pip install numpy cython
#install scipy
wget https://github.com/scipy/scipy/releases/download/
v1.3.3/scipy -1.3.3.
tar.gz
tar -xzvf scipy -1.3.3.tar.gz scipy -1.3.3
cd scipy -1.3.3/
python setup.py install
#install opencv
sudo apt-get install python3-opencv
```

A.2.4.5 Install TensorFlow

We install this package to run the Face Detection step.

```
# get a fresh start
sudo apt-get update
```

```
sudo apt-get upgrade
# install pip and pip3
sudo apt-get install python-pip python3-pip
# remove old versions, if not placed in a virtual environment
#(let pip search for them)
sudo pip uninstall tensorflow
sudo pip3 uninstall tensorflow
# install the dependencies (if not already onboard)
sudo apt-get install gfortran
sudo apt-get install libhdf5-dev libc-ares-dev libeigen3-dev
sudo apt-get install libatlas-base-dev libopenblas-dev libblas-dev
sudo apt-get install liblapack-dev
sudo -H pip3 install -U testresources numpy
# upgrade setuptools 39.0.1 -> 50.3.2
sudo -H pip3 install --upgrade setuptools
sudo -H pip3 install pybind11 protobuf google-pasta
sudo -H pip3 install -U six mock wheel requests gast
sudo -H pip3 install Cython==0.29.21
# install h5py with Cython version 0.29.21 (± 6 min @1950 MHz)
sudo -H pip3 install h5py==2.10.0
sudo -H pip3 install keras_applications --no-deps
sudo -H pip3 install keras_preprocessing --no-deps
# install gdown to download from Google drive
pip3 install gdown
# copy binairy (./local/bin may not be in your \$PATH)
sudo cp ~/.local/bin/gdown /usr/local/bin/gdown
# download the wheel
gdown https://drive.google.com/uc?id=1oeSnkgJpwyudtTx-f5CE84B7e-Vkv3yK
# install TensorFlow (± 12 min @1500 MHz)
sudo -H pip3 install tensorflow-2.3.1-cp36-cp36m-linux_aarch64.whl
```

After successfully installing the TensorFlow package, we next install the TensorRT to optimize the performance of the Face Detection model. It is the instruction for the installation.

https://docs.nvidia.com/deeplearning/tensorrt/install-guide/index.html

A.2.4.6 Install dlib module

We install this package to run the Face Identification step.

```
wget http://dlib.net/files/dlib-19.22.tar.bz2
tar xvf dlib-19.22.tar.bz2
cd dlib-19.22/
mkdir build
cd build
cmake ..
cmake --build . --config Release
sudo make install
sudo ldconfig
cd dlib-19.17
python setup.py install
```

A.2.4.7 Install Azure packages

In this section, we install some Azure packages that are used to communicating with the Device Server.

```
pip3 install azure-core==1.13.0
pip3 install azure-eventhub==5.2.1
pip3 install azure-iot-device==2.5.0
pip3 install azure-iot-hub==2.2.3
```

A.2.4.8 Install Another Packages

The above are primary packages that are essential for the system. Besides, we have to install support packages in order to run the system entirely. Here is the link for our system requirements.txt. After download it, we run "pip3 install -r requirements.txt" to install all the packages.

A.2.4.9 Install PyQt5

This package is used to run the GUI of the system. First we run the below command to install this package.: "sudo apt-get install python-qt5"

As this package belongs to the system library and the virtual environment does not recognize this package at the run-time, we have to modify the *include-system-site-packages* field in the file *pyvenv.cfg* of the virtual environment into *true*.

A.2.4.10 Setup LXDE GNOME

At the default, the Jetson Nano uses the GNOME display manager (gdm3). However, this GNOME takes the number of RAM resources up to 2GB, while our memory is 4GB. Therefore, we change to the LXDE GNOME for saving about 1GB of memory. This selection is available when we log in as we use Jetpack 4.5.1, larger than 4.4.1.

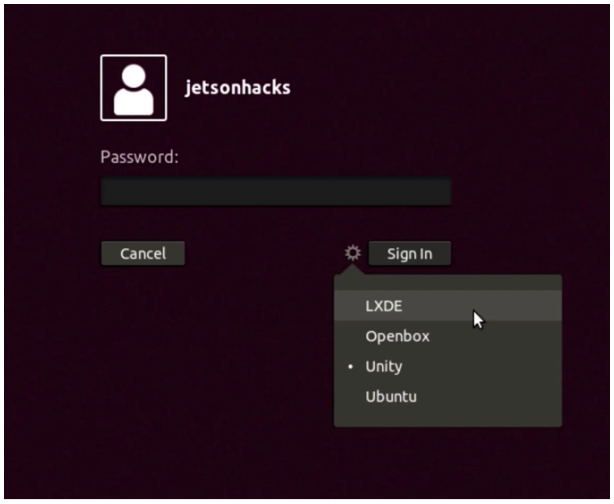

Figure A.14: LXDE Set Up

A.2.5 Initiate Check-In Connection with IoT Hub

After setting up all the needed Software Packages, we clone the source of the system as the link:
https://github.com/truongkhoa1799/E-Healthcare-System.git

Then, at the directory, we open file *config_para.yaml* and edit the *eventhub_name, eventhub_connection, and iothub_connection* as the *eventhub_connection*, *eventhub_name*, and *Primary connection string* of *serviceAndRegistryRead* in section 6.1.2, 6.1.2, and 6.1.3, respectively.

Finally, we run the command below in the terminal to load all the connections used to communicate with the server, where device_id is the id of this device created by the Web Application. "./create_device.sh device_id"

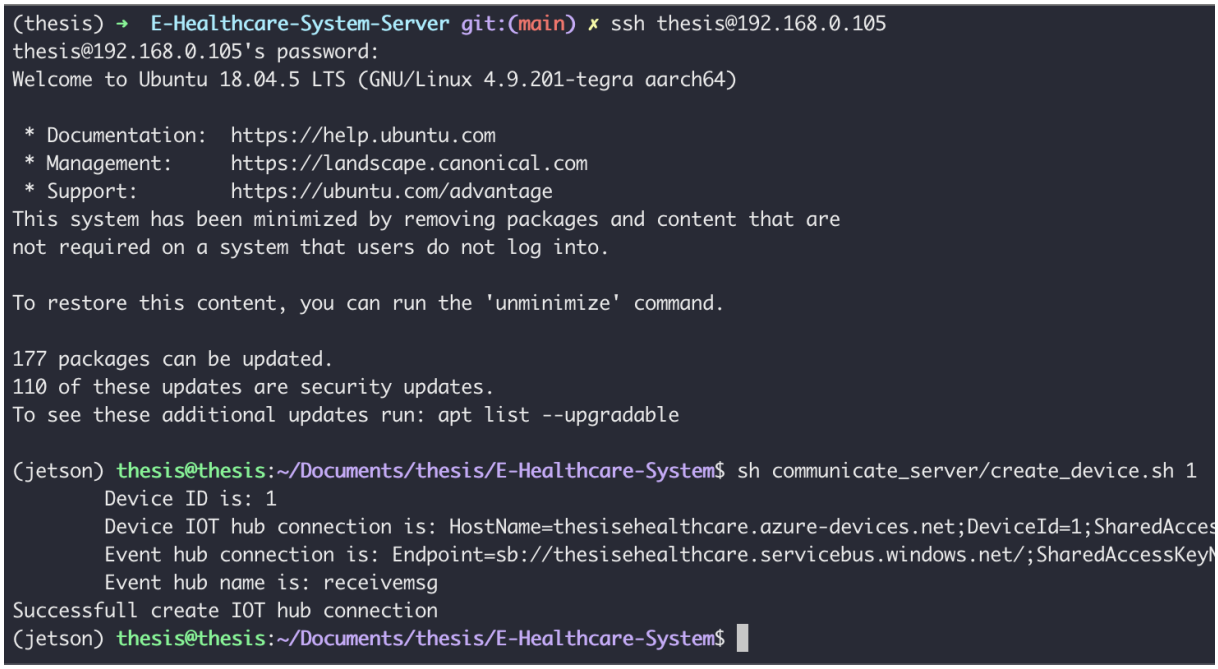

Figure A.15: Setup Jetson Nano At first

A.3 Device Server Setup

A.3.1 Install Software Packages

Before running the Device Sever, we have to install some needed packages for the running-time.

```
#install rasa open source
pip3 install rasa
#install dependencies packages
sudo apt-get install -y build-essential libssl-dev uuid-dev cmake \
    libcurl4-openssl-dev pkg-config python3-dev python3-pip
#install Azure packages
pip3 install azure-common==1.1.27
pip3 install azure-core==1.13.0
pip3 install azure-eventhub==5.2.1
pip3 install azure-eventhub-checkpointstoreblob-aio==1.1.2
```

```
pip3 install azure-iot-device 2.5.0
pip3 install azure-iot-hub 2.2.3
pip3 install azure-nspkg==3.0.2
pip3 install azure-storage==0.36.0
pip3 install azure-storage-blob==12.7.1
#install library for connect with the database
pip3 install pyodbc
```

A.3.2 Run the server

Now, we clone the source code of Device Server at the link **https://github.com/truongkhoa1799/E-Healthcare-System-Server.git**, and modify in the file *config.yaml* the *eventhub_ name, eventhub_ connection, and iothub_ connection* similar to the Check-In device. However, for the *iothub_ connection*, we get the *Primary connection string* of *serverRight*. Finally, we start the server by run the file **main.py** in the root directory. Finally is the Database Connection information. We setup the connection with 4 fieds *database_ server, database_ user, database_ password, and database_ name* in the *config.yaml* file

Next, we run the server to extract the patient's symptoms. We clone the source, move to the project's root folder, and run the below code. The server will run the address **http://localhost:5005/model/parse** similar to section 4.3.4.1.

```
git clone https://github.com/truongkhoa1799/ChatBot-Classifying-Department.git
cd ChatBot-Classifying-Department
rasa run --enable-api -m models/nlu-20210531-214052.tar.gz
```

REFERENCES

[1] Qi Liu, Hong Li, Yan Hu and Limin Sun, *"A Check-in System Leveraging Face Recognition"*, 2018 IEEE Confs on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, Congress on Cybermatics.

[2] Qiang Zhu, Shai Avidan, Mei-Chen Yeh, and Kwang-Ting Cheng, *"Fast Human Detection Using a Cascade of Histograms of Oriented Gradients"*, In Conference on Computer Vision And Pattern Recognition, 2006

[3] Davis E. King, *"Max-Margin Object Detection"*, In Conference on Computer Vision And Pattern Recognition, 2015

[4] Xu, Yuanyuan and Yan, Wan and Sun, Haixin and Yang, Genke and Luo, Jiliang, *"CenterFace: Joint Face Detection and Alignment Using Face as Point"*, In Conference on Computer Vision And Pattern Recognition, 2019

[5] Pingping Chen, *"Development and Optimization of Check-in System Based on Face Recognition Technology"*, et al 2020 IOP Conf. Ser.: Mater. Sci. Eng. 782 052022.

[6] R. Fletcher. *"Practical Methods of Optimization, Second Edition"*. John Wiley Sons, 1987.

[7] C. Huang and R. Nevatia. *"High performance object detection by collaborative learning of joint ranking of granules features"*. In Computer Vision and Pattern Recognition, 2010.

[8] X. Zhu and D. Ramanan. *"Face detection, pose estimation, and landmark localization in the wild"*. In Computer Vision and Pattern Recognition, 2012.

[9] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, *"Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation"*, CoRR, vol. abs/1801.04381, 2018. [Online]. Available:

[10] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie,, *"Feature pyramid networks for object detection"*, CoRR, vol. abs/1612.03144, 2016. [Online]. Available:

[11] D. E. King, Dlib-ml: A machine learning toolkit, The Journal of Machine Learning Research, vol. 10, pp. 1755–1758, 2009

[12] Thompson, Cindi. "Open Source Toolkits for Speech Recognition." Silicon Valley Data Science, 10 May 2018, https://www.svds.com/open-source-toolkits-speech-recognition/

[13] ReSpeaker Mic Array v2.0 Specifications

[14] "Introduction to JetPack :: NVIDIA JetPack Documentation."

[15] "What is Azure IoT Hub | Microsoft Docs"

[16] "Understand Azure IoT Hub direct methods | Microsoft Docs"

[17] "What is Azure Event Hubs? - a Big Data ingestion service"

[18] Shmyrev, N., 2020. Basic Concepts Of Speech Recognition. [online] CMUSphinx Open Source Speech Recognition. Available at: https://cmusphinx.github.io/wiki/tutorialconcepts/

[19] Jinyu Li and Yifan Gong. "Robust Automatic Speech Recognition." 2016.

[20] V. Këpuska, J. Breitfeller, Wake-up-word application for first responder communication enhancement, Proc. SPIE 6201, pp. 62011E, 2006. doi:10.1117/12.666025

[21] James, Simon. "10 Good Open Source Speech Recognition Systems [2020]." FOSS Post, 23 Aug. 2020,https://fosspost.org/open-source-speech-recognition-2020/.

[22] Firmansysah, Muhammad Hafidth, et al. A.I. Based Embedded Speech to Text Using Deepspeech. 25 Feb. 2020, https://arxiv.org/pdf/2002.12830.pdf.

STUDENT INFORMATION

List of thesis authors:

1. **Truong Le Vinh Khoa** - ID: 1752298
    - Phone number: (84)971.155.910
    - Personal Email: khoa.truong1799@hcmut.edu.vn
2. **Le Nguyen An Khuong** - ID: 1752305
    - Phone number: (84)938.537.412
    - Personal Email: khuong.le71099@hcmut.edu.vn