# SEEHR Documentation



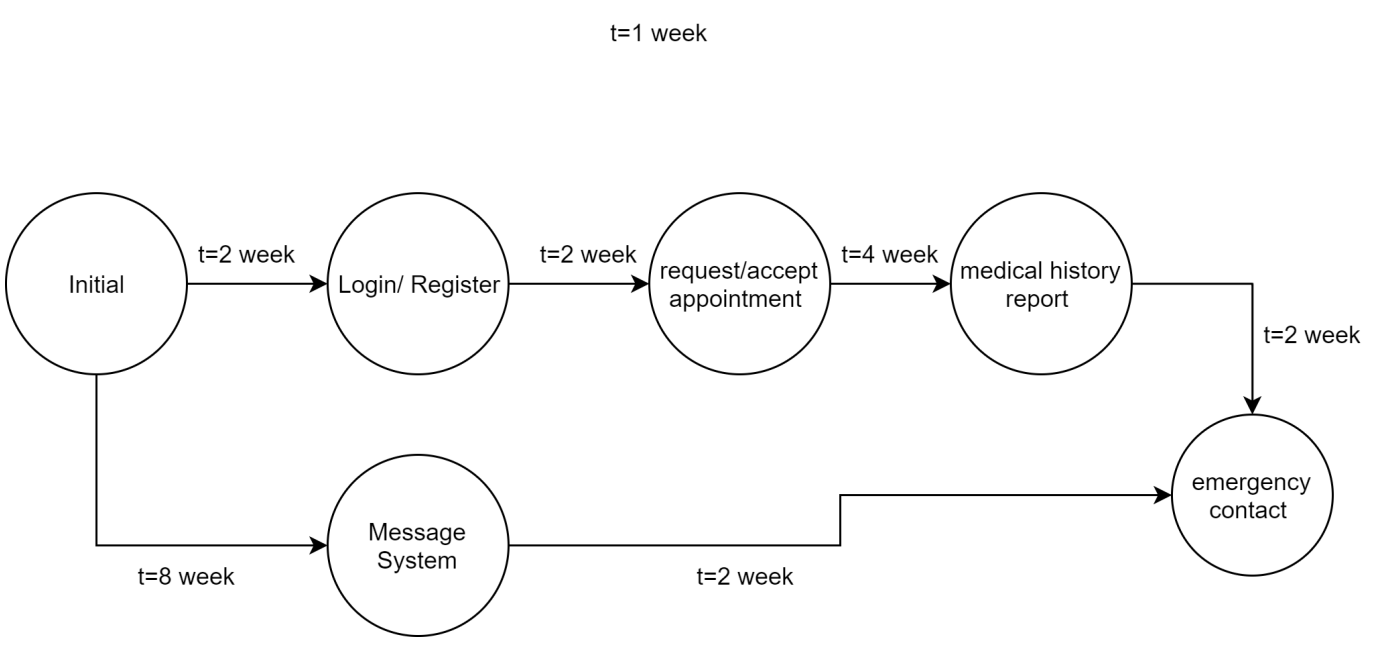
## Introduction

- With more and more people enjoying electronic medical devices and devote more trust in it, it is feasible to build an electronic health record (EHR) system to cut down hospitals' spending on hiring people working for registration and reduce the time spent on searching and scheduling. EHR stores patients' medical history records without the risk of losing the medical history book. Using data, it stores all settings that could provide plenty of health care services for patients. Also, it gives doctors decision opportunities based on the recorded information; meanwhile, it offers automated and streamlined workflows.
  The last name of our system is SEEHR, which is a combination of SE (Software Engineering) and EHR (Electronic Health Record). However, it can also be understood as See HR since our project focuses more on the administrative perspective.

## Design Process

- We release the bi-weekly artifacts throughout the semester. The order is:
  1. login, registration: in the first iteration, we mostly learned to get familiar with the coding language and environment. The login and registration situations are different among different stakeholders (see General Cases)
  2. request appointment, accept appointment: this stage connected the functions of patients' and doctors' by sending and receiving appointment requests, respectively.
  3. medical history report: it is a refactored stage. In the end, after each appointment, a doctor is required to finish the medical history report regarding that appointment. Doctors can view their patients' medical history but cannot edit it.
  4. message system
  5. emergency contact button, send the notification:
     - The emergency contact button is designed at first. However, we did not think it is reasonable compared to dial 120 directly. However, if a doctor is logging his e-mail on his phone, once a patient clicks this button, he will receive an e-mail. Then a doctor can choose to present at the hospital even if he is not on duty to support the treatment.
     - As for sending a notification, it is a function that is not initially planned. We had a problem connecting it with the message system since they are on different servers. Then we decided to present the notification as a flashed message from HTML instead of a received e-mail.
- In the first two stages, the development was not too complicated. However, we have refactored the medical history report for many times.
  - It is due to the problematic use of case design at first.
  - Previously, we plan to associate the medical history report with each patient as a database feature in the type of "text".
  - As more and more report was added, the text became too long to handle by a doctor. The doctor may have the risk of modifying or deleting the previous medical history of a patient, which was not user-friendly.
  - After refactoring, each medical report is associated with an appointment as a database feature. Therefore, doctors will only have the read access to other reports and can only edit the report.
- We also encounter some problems when merging the message system into our primary system.
  - Since we design the message system and the primary system separately, they

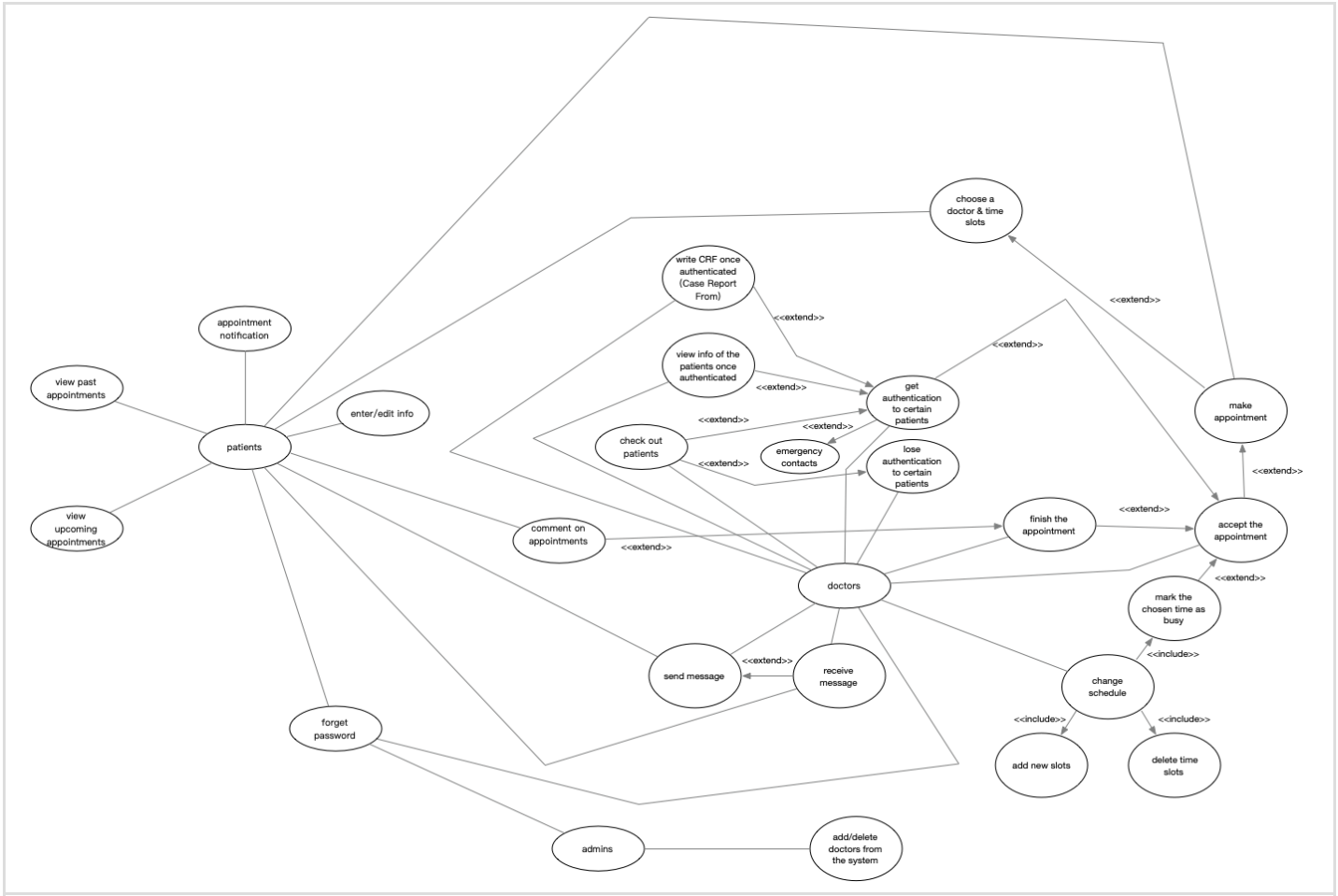
have different backend server code. The message system uses node js while the main system uses flask.
    - At first, when we tried to merge them, we always see the error of the CORS policy violation so that the message system is not working correctly.
    - Later we found out that CORS is used to prevent cross-origin requests for safety issues, and this is because we set different origins for the message system and the primary system: one is `http://localhost` and one is `http://127.0.0.1`. After we fix this, the message system can then working correctly with our main system.
- In the last iteration, we didn't encounter any significant problems.
- PERT diagram for SEEHR.
    - The message system is isolated from the whole process and took an unexpectedly long time to merge.
    - The main branch efficiently and strictly follows the speed of 2 weeks per iteration. Our developments are based on their dependencies and priorities.
    - After the merge of the message system, finally, we can develop dependent functionalities such as emergency contact.




*PERT Diagram*

# Use Cases

- The following is the Use Case Diagram.




*Use Case Diagram*

Login and registration are first released. Then we can access the interface of different stakeholders. Three interfaces of three types of stakeholders are set as black boxes, and we can add functionalities to them in detail afterward. In the design of each interface, we finished the structure and implemented the code for each button. Then we realize each function as our process develops.

This should include the user stories and use cases that you implemented in your project. You do not need to mention the requirements which were dropped.

## General Cases

- **Register/Login**
  There are three types of users in our system: administrators, patients, and doctors. Administrators are the managers of the system. There is a secret key in the system, and only people who are authenticated by the hospital can get the key and create administrator accounts. Patients are the most common types of users. Anyone will

be able to register an account for the patient. Users cannot register doctor accounts by themselves since not everyone is eligible to be a doctor. So only the administrators can create doctor accounts for users.

- **Forget Password**
  Users will be able to change passwords in case they forget the passwords. To change the password, the users have to provide some information about the account when registering it, such as the e-mail and phone number used to record the account.

- **Chat System**
  We provide the functionality of chatting in our system. Anyone would be able to send messages to others, providing the e-mail address that they want to send messages to. And anyone who receives the message will get a notification.

## Administrators

- **Browse doctors**
  Administrators can view a list of the doctors registered in the system and select a doctor to see his or her detail information.

- **Add/Delete Doctors**
  Administrators can help doctors create doctor accounts. Also, as manager of the system, they are able to delete doctor accounts from the system when the user of the account retires.

## Patients

- **Edit Personal Information**
  Patients will be able to edit their personal information, such as weight, height, and emergency contacts. However, they are not able to change information such as their name or date of birth because this information should only be provided when they register the account.

- **Make Appointments**
  Patients will be able to make appointments with doctors. This use case has two different situations.

  - **Patients do not have a provider yet**
    If a patient does not have a doctor as their provider, they can choose any

doctor in the system to make appointments with. They will also be able to search for doctors providing information such as time slots wanted, doctor name, and appointment duration.

  - **Patients already have a provider**
    If a patient already has a provider, he or she can only make appointments with that specific doctor in the provided time slots.

- **Cancel Appointments**
  Patients would be able to cancel an appointment, which has not been accepted by the doctor yet.

- **Comment on Previous Appointments**
  After a doctor finishes an appointment, the patient would be able to comment on that appointment.

- **View Past/Upcoming Appointment**
  Patients would be able to view a list of past appointments as well as a list of upcoming appointments.

- **View Medical History**
  The system will provide a timeline of all the appointments along with the corresponding medical history.

- **Visiting Warning**
  The system will automatically notify the patient if the patient hasn't made an appointment with a doctor in 30 days to remind the patient to check his or her status with a doctor in time.

- **Emergency Alert Button**
  There is a button on the right up corner of the patient's home page. When the patient is suddenly in danger, he or she can press the button to inform all the doctors in the system that he or she is danger and need help.

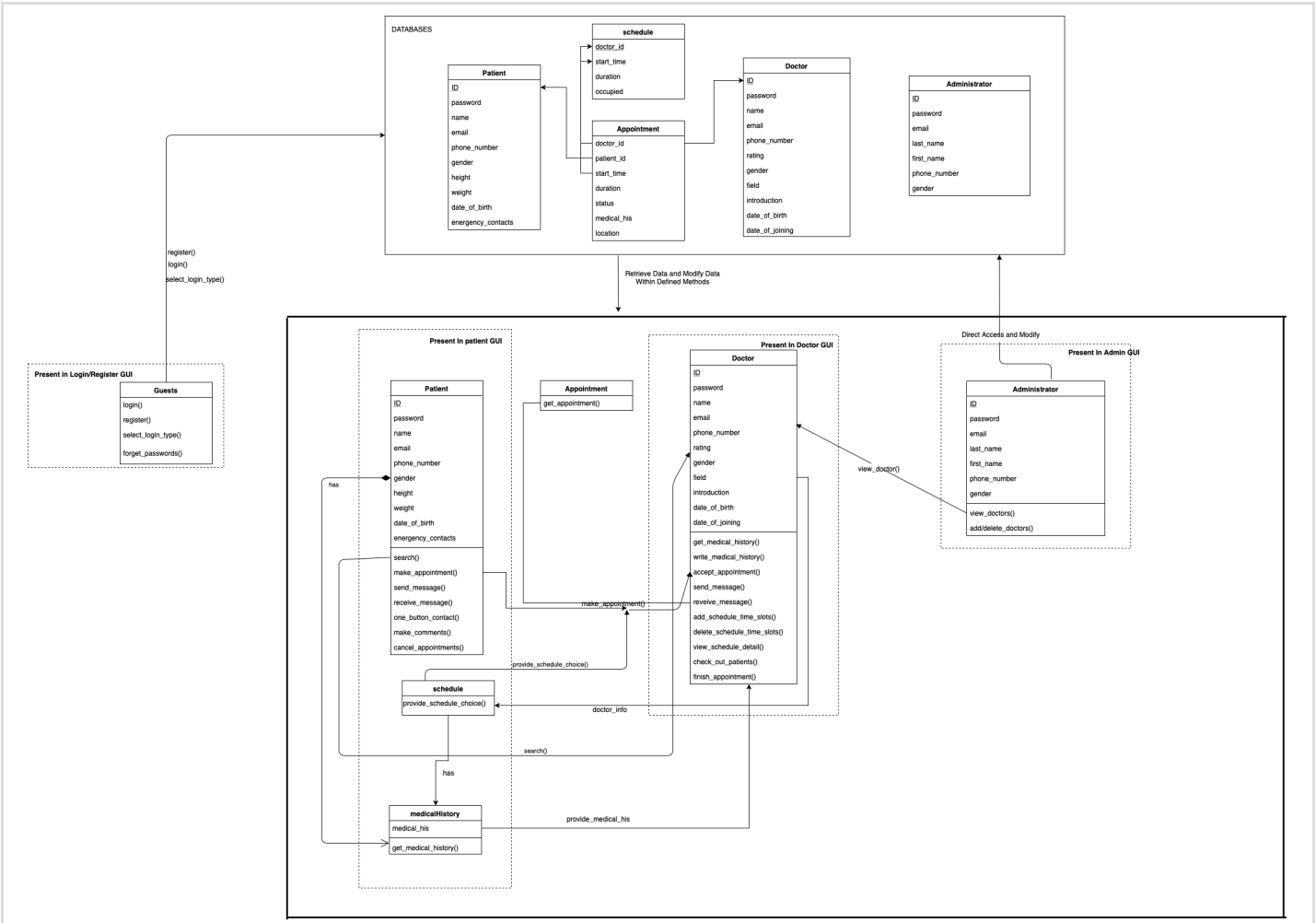
## Doctors

- **Accept Appointments**
  Doctors can accept appointments made by the patients. However, they will not be able to reject the appointments since the time slots are provided by themselves, so it's their responsibility to accept the appointments and take care of the patients.

- **Manage Schedule**
  Doctors will be able to manage their schedules.
  - **Add/Delete Time Slots**
    Doctors will be able to add new time slots for patients to make an appointment with. In the same way, doctors can delete free schedules before they are booked.
  - **View Schedule Detail**
    Doctors can select one-time slot and view the detail schedule information such as if the schedule has been occupied and if yes, it will also show the information of the patient who made the appointment.
- **Medical History Report**
  Doctors will be able to manage medical history reports.
  - **View History Report**
    Doctors will be able to view the history report for each appointment separately as well as view all the reports for a patient as a whole.
  - **Edit History Report**
    Doctors will also be able to edit the report for each of the appointments made with him. The doctor can only write the report for one time slots at a time, and this can prevent him from changing the report for other slots by mistake.
- **Finish an Appointment**
  Doctors would be able to finish an appointment and enable the patient to comment on that appointment.
- **Check Out a Patient**
  Doctors will be able to check a patient that he or she is responsible for and enable that patient to make appointments with other doctors in the system.

# Architecture/Design

- The following is the Class Diagram.




*Class Diagram*

The first thing we design is how we log in to the system. Then, since there are different roles in our system, we create separate classes for these roles, and they should have different accounts in the database. After we finish these things, we then start to consider the interaction between different functions, such as making an appointment between doctors and patients and the chat system.

- We use the following frameworks for front end:
  - Javascript
  - HTML
  - Jinja2
- We use the following frames for the back end:
  - Python
  - Flask
  - Sqlite

# Reflection

- **Reflection By Yixiang Xiao**
    - During the whole process of doing the project, I am in charge of the backend part design as well as some Javascript code for the front end. Overall I think I've done a good job since I've finished most of the functionalities we planned at the beginning of the process except for some deprecated functions. However, there are still several things that I have to reflect on.
    - Testing is one of the things that I didn't do well. I didn't write unit test along with the backend code. Instead, I test the backend code directly using the front end code after it is finished. This is not that efficient since I have to wait until the front end is done can I test my backend code. There are several ways of improvements to this. The first is writing the unit test code along with the backend code. The second way to test the code without the frontend is to use some external software such as Postman, which allows me to send queries to the server directly.
    - The second thing I want to reflect on is how I deal with the interaction between the frontend and backend. At first, I used ajax to send queries from frontend to backend. However, ajax is used to send queries to backend without refreshing the page. This means that after submitting the queries, the user has to refresh the page to view the changes, which is not very user-friendly. So later on I changed these ajax queries to a simple statement `location.href= {{url_for('target function')}} + required parameters.` . This can directly send parameters to backend and call the target function as well as refresh the page automatically which can make our system more convenience for our users.
    - I've also realized the importance of Github during the process and the fact that I have to learn more about how to make use of it to do version control. One thing we didn't do well is that we always push code to the master branch. This may lead to a problem that almost every time we push, we may encounter conflict and have to resolve it manually, which is not efficient. To do better, each person should create his branches and push code to that specific branch by himself. And at last, we can merge these branches to get our final project.
- **Reflection By Yunxiao Song**
    - Firstly big thanks to Yixiang Xiao, he is a very responsible and skillful teammate

who communicated with me well and coded backend that suits the frontend perfectly, which makes my job way more manageable.

- At the start of this semester, I was not familiar with any frontend framework as my primary language is Python and JAVA. Still, the team needs a person to code tedious frontend parts, so I pushed myself out of my comfort zone and take the responsibility of coding frontend. This is my first time coding a frontend page in a web application and reflecting all the process. I successfully beat that challenge.
- I am responsible for developing all frontend templates and all templates' encapsulated javascript code. I also factored some of Yixiang Xiao's javascript code. I did the original login& register backend part of the project. I designed and implemented frontend pages using vue.js and bootstrap. In each iteration, we start from design and decide on what backend functionalities to realize. Base on that, I design and implement a functional frontend interface that can interact with Flask backend. Moreover, there are adjustments in terms of frontend graphic interfaces because firstly interfaces may not be easy to connect with the backend, or secondly, there are changes of required functionalities. Therefore, I learned and practiced that for applications that require long-term maintenance, extensibility, and readability is essential for the program.
- There are interfaces where are write is of lousy UI/UX design, which harms user experiences, if there is more time, having a discussion, an iteration and an update on those user-unfriendly interfaces will hugely help our application. Throughout a whole semester of cooperation, I attended every group meeting and involved in the discussion on design topics and how to realize functionalities. In all iterations, I provided working and interface frontend templates as early as possible, and through communication with the backend group, we successfully finished most of our planed functionalities.

- **Reflection By Xiyan Cai**

  - I have mostly done the backend part of the project except for the separated message system. I was in charge of the test. I also planned and organized each weekly meeting to facilitate the project progress. As a reflection, I at first chose to use Python Flask as our base language since all of the team members are familiar with Python. However, it was the right moment to push me to learn a new programming language and get command of it by practicing it on a project.
  - Besides, our team did not plan the whole agile progress and the specific tasks

of each team member, which resulted in problems like less communication within the team and missing a teammate during several iterations. On the other hand, the risk of losing one team member was assessed as minor since all of our team members can do each other’s work

- When we were designing the user cases, we did not consider the aspect of being user-friendly. Instead, the purpose of our design lied in how to connect the systems as a whole and make it function smoothly. Therefore, it resulted in user-unfriendly experiences like adding a medical history report. Consequently, we should have put the user-friendly design into consideration at every moment of conducting a design.
- Last but not least, we did not have the document to keep the process of our work and to be referred at any time when we had confusion on the design.

- **Reflection By Chongling Zhu**