Patient Tracker System Report

Yujin Qin Yuqi Liu Yi Ding

1. Requirements

1.1. Overview

The primary goal of this project is to build a Patient Tracker System, which is designed for doctors and healthcare professionals. In today's era of modern big data, the importance of data recording cannot be overstated. Software like this could be an efficient and useful tool. This digital system replaces traditional paper records and allows doctors to access patient records remotely and update the database in real time. Furthermore, the system ensures data accuracy and security by using encryption to protect data both in transit and at rest. Employ protocols such as HTTPS will be used for secure communication over the network.

1.2. Features

- **User Authentication** (Doctor username and password match the existing accounts.)
- **Profile Management** (Doctor profile: create, read, and update)
- **Medication/Treatment History Tracking** (Use MongoDB to record patient health history)
- **Data Visualization** (Create charts based on vital signs, such as blood pressure, and heart rate.)
- **Alert Notification** (Alert the user when they have complete or fail on a current task)

1.3. Functional Requirements (Use cases)

1. As a doctor, I can log in to the system with my unique username and password, which will be the authentication step that verifies my identity.
2. If I don’t have an account, I am able to sign up for a new account.
3. After logging in, I can view my doctor profile and update any information if needed.
4. As a doctor, I just received a new patient and I want to log this new patient's information and general body and health data into the website. I can create and add this new patient to my portal. Once I submit successfully, I will be notified that I added the patient successfully.
5. After adding the new patient, I can click view all the patients and see a list of patients.
6. I want to review my previous patient’s medical history to prepare for my appointment, so I want to view this patient’s data which I registered a while ago.
7. While checking one specific patient profile, I am not only able to check their basic information but also able to go through their medication prescription/treatment history.
8. I want to see a trend of the patient’s blood pressure over the weeks so I generate a virtualized graph based on the record.
9. During my appointment, I diagnosed the patient and prescribed some medicine to him, so I added a treatment history for him in the portal.
10. In the next appointment, I can view all the treatment history from this patient.

11. I accidently clicked submit a treatment record/personal information that is empty, there will be a pop up window to remind me to fill out all the information first.

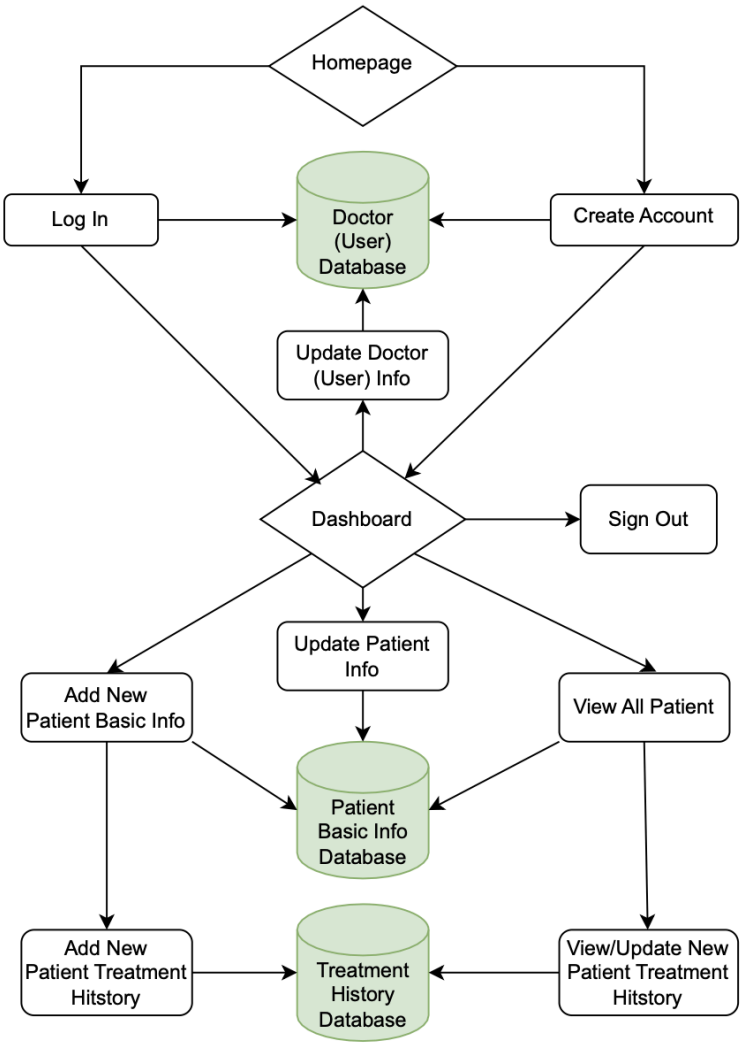
1.4. Non-Functional Requirements

Firstly, our system prioritizes security through a username and password matching mechanism, ensuring that patient data can only be accessed by authorized doctors. Secondly, our system emphasizes reliability, allowing doctors to access patient information at any time, and maintaining a comprehensive history of all patients to prevent data loss due to any modifications. Furthermore, our system features a user-friendly and intuitive interface, reducing the learning curve for users, so our system has a good usability. Our website has good UIUX feedback, when the user submit things, there are alerts to tell the users that they have successfully submitted things. Lastly, our website respond to user requests within an acceptable time frame to ensure a smooth user experience. The performance of our system is stable.

2. Design

2.1. Architecture Diagram

- At first, users will be directed to the homepage where they will be directed to either login or sign up for a new account. When a doctor signs up for an account, the system stores all the account information of each doctor, including doctor’s name, contact information, etc into the database.
- After logging in, users will be able to view the dashboard where they have the option for doctors to add new patient information or update their patient's information. This step gives access to view and modify the patient info database, including patients’ basic information, health information, and medical history.
- When users are viewing a health information page, they can click the “health data visualization” button to let the website automatically generate plots for patients’ health data, such as line graph of heart rate and oximetry. This step gives access to the patient info database.



2.2. Technology Stack with Justification

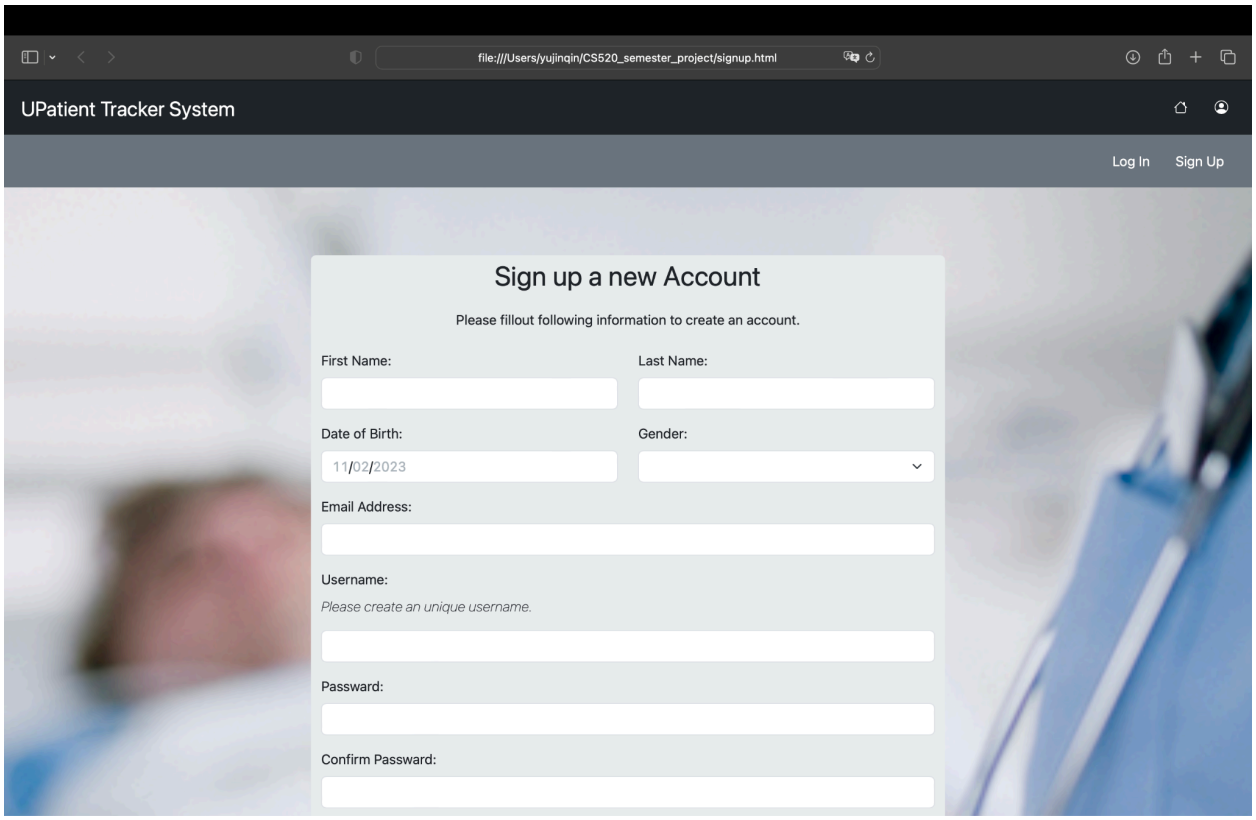
Frontend: We chose Bootstrap for web design and user interface implementation since all our members have the experience programming the Bootstrap, we wish to choose a language that all the members are familiar with. Bootstrap provides a set of pre-designed and pre-styled components, making it easy to create a consistent and visually appealing user interface. Bootstrap also comes with JavaScript plugins that enhance the functionality of components, such as sliders, carousels, and tooltips, which makes it easy for us. Lastly, Bootstrap is compatible with multiple browsers. Therefore, we chose Bootstrap as our frontend framework.

Backend: We chose JavaScript for backend development and use Node.js to run JavaScript server-side. JavaScript allows for the dynamic loading of content without requiring a full page refresh, leading to a smoother and more engaging user experience. Moreover, JavaScript allows us to do asynchronous programming, which guarantees that the application won't get stuck during long-time running and improves user experience.

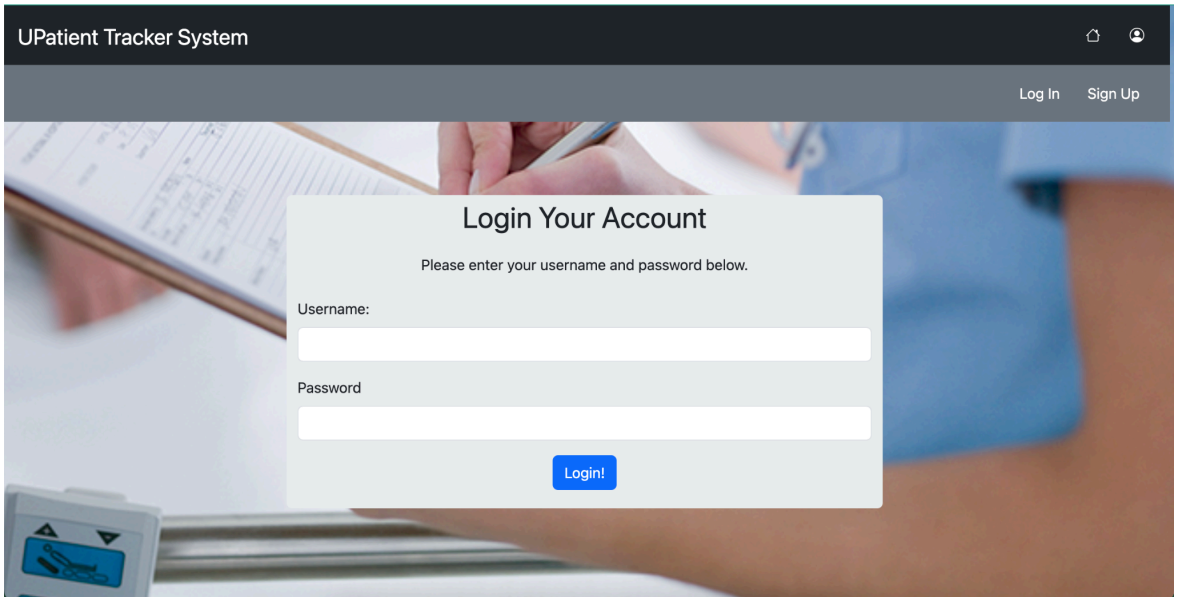
Database: We chose to use MongoDB because it's a document-oriented database, where each record is a document containing key-value pairs. This approach is beneficial for representing complex data structures, and it allows for the nesting of arrays and other structures. And it also supports a wide range of queries, including filtering, sorting, and aggregation.

2.3. UI Mockup

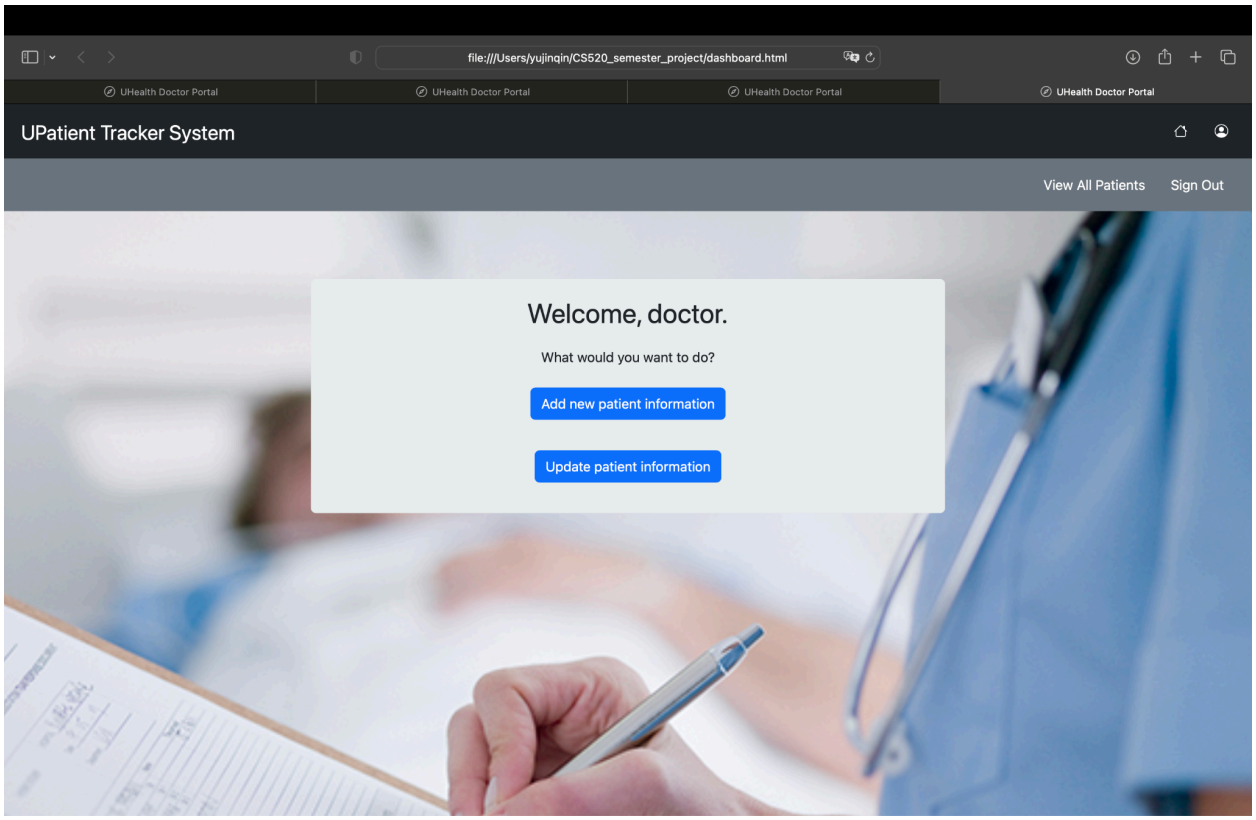
1. Sign up page



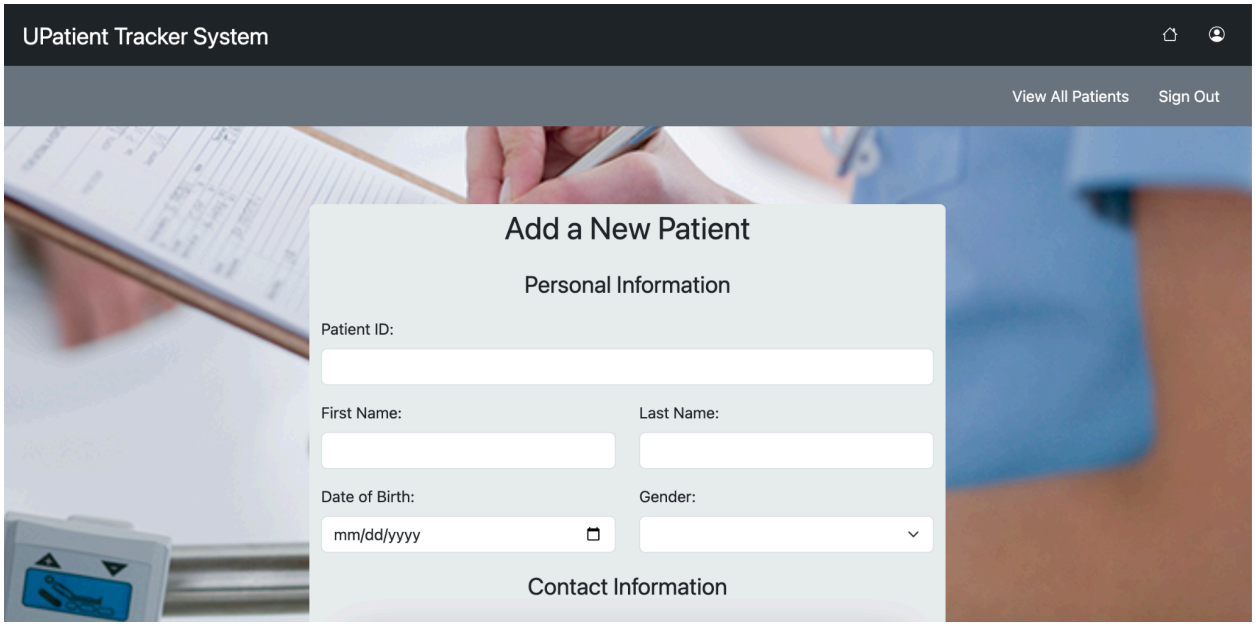
2.Sign in page



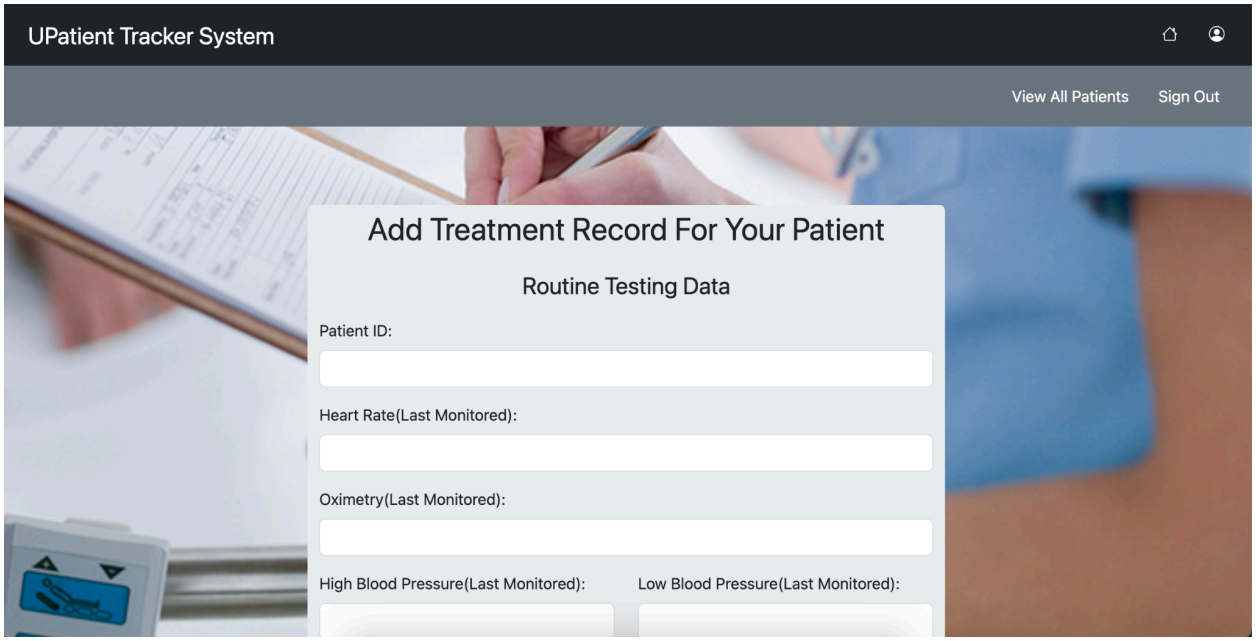
3. Dashboard



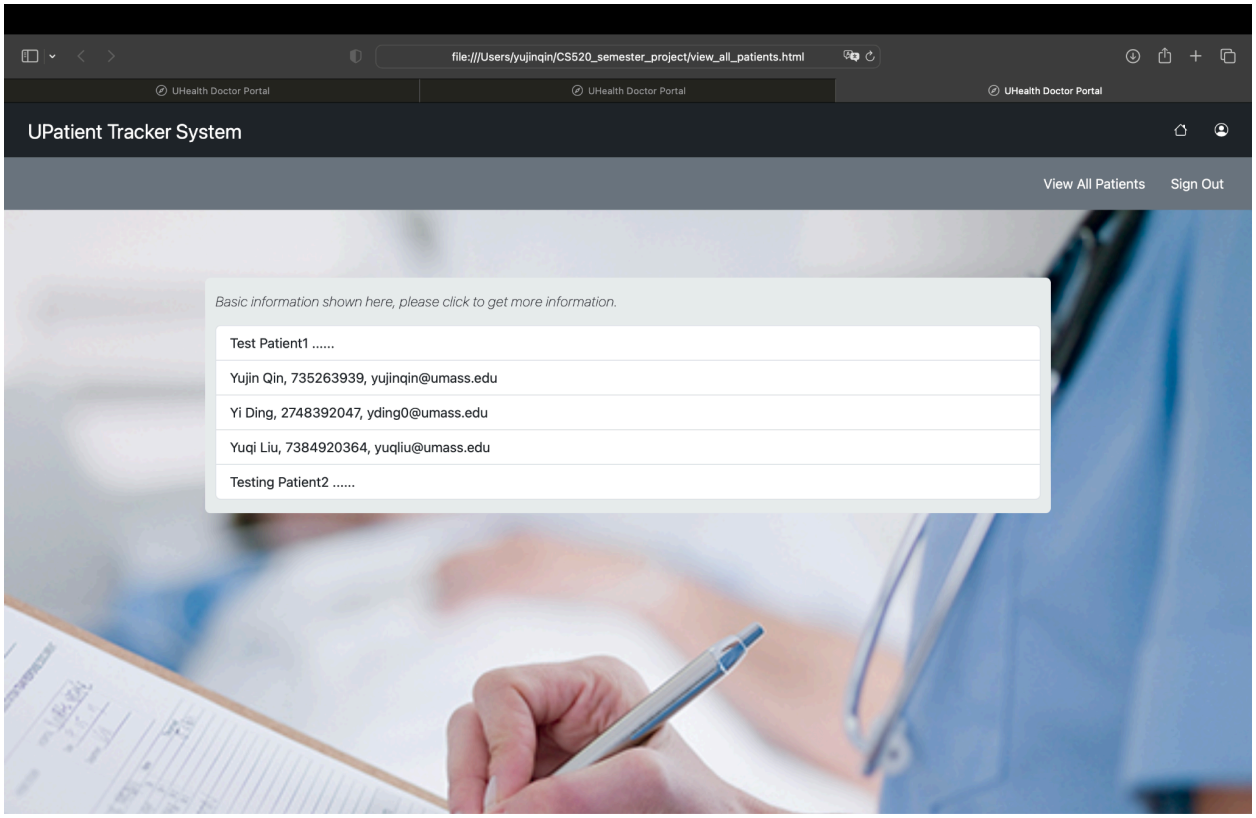
4. Add new patient information



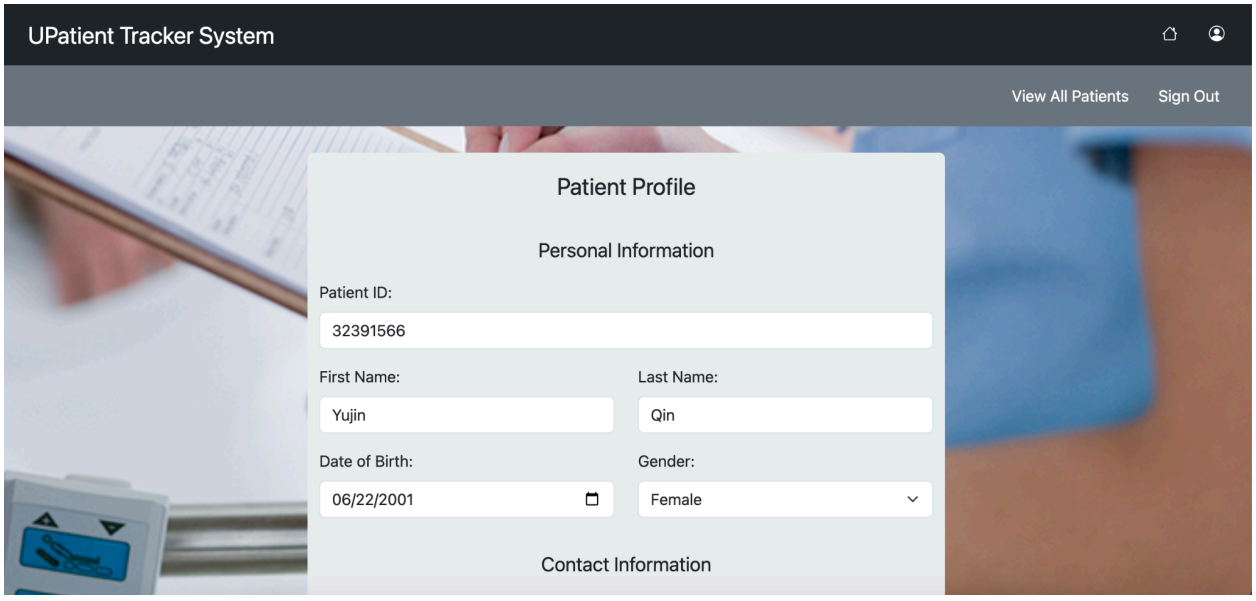
5. Add Treatment Record For Your Patient

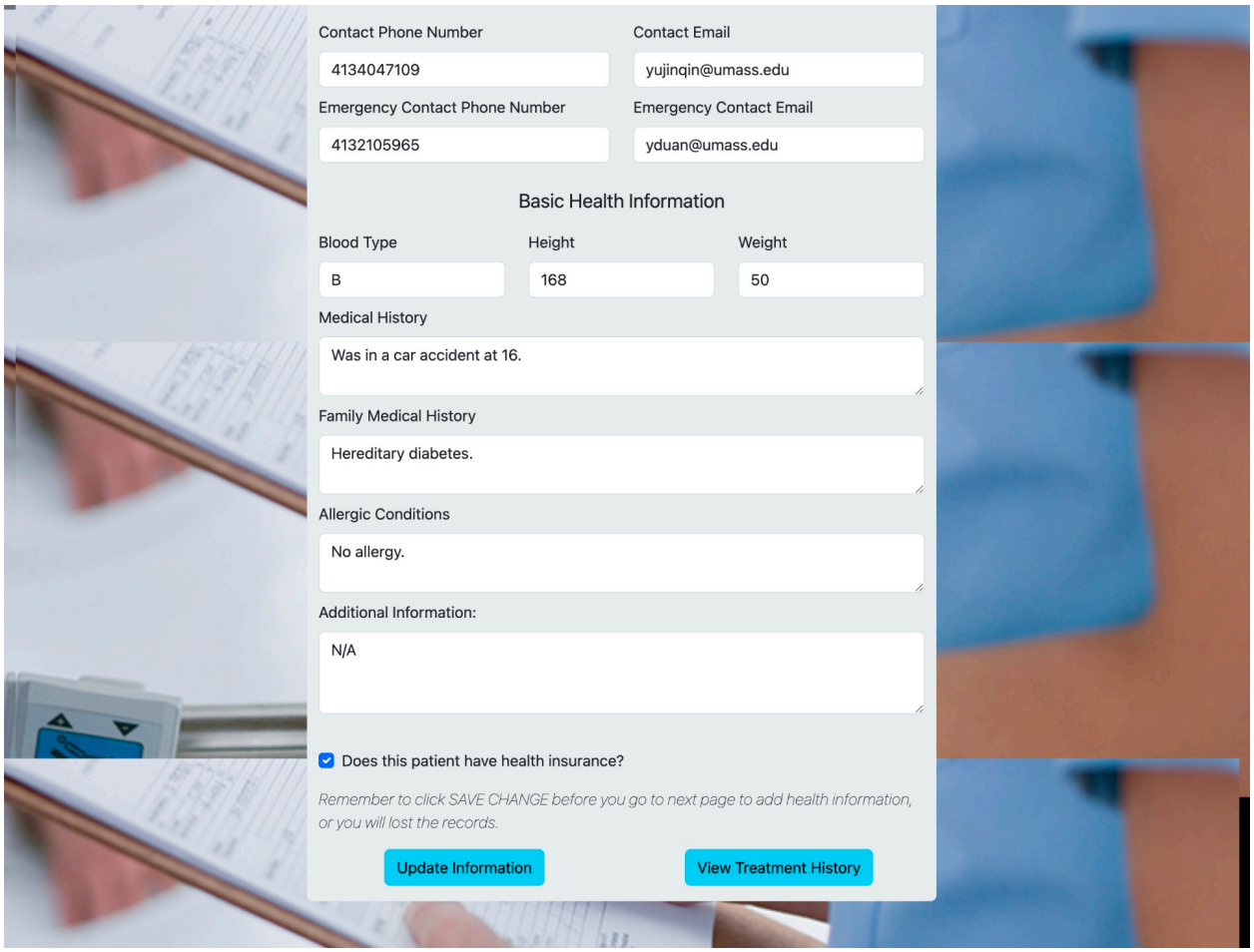


6. Update Patient Information/View All Patient

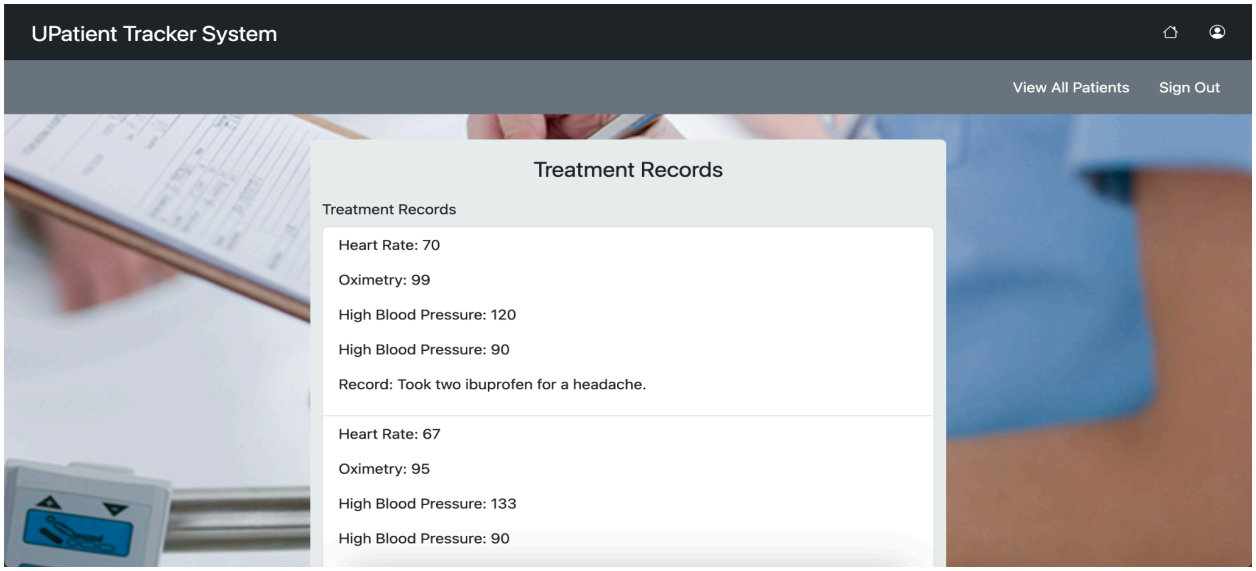


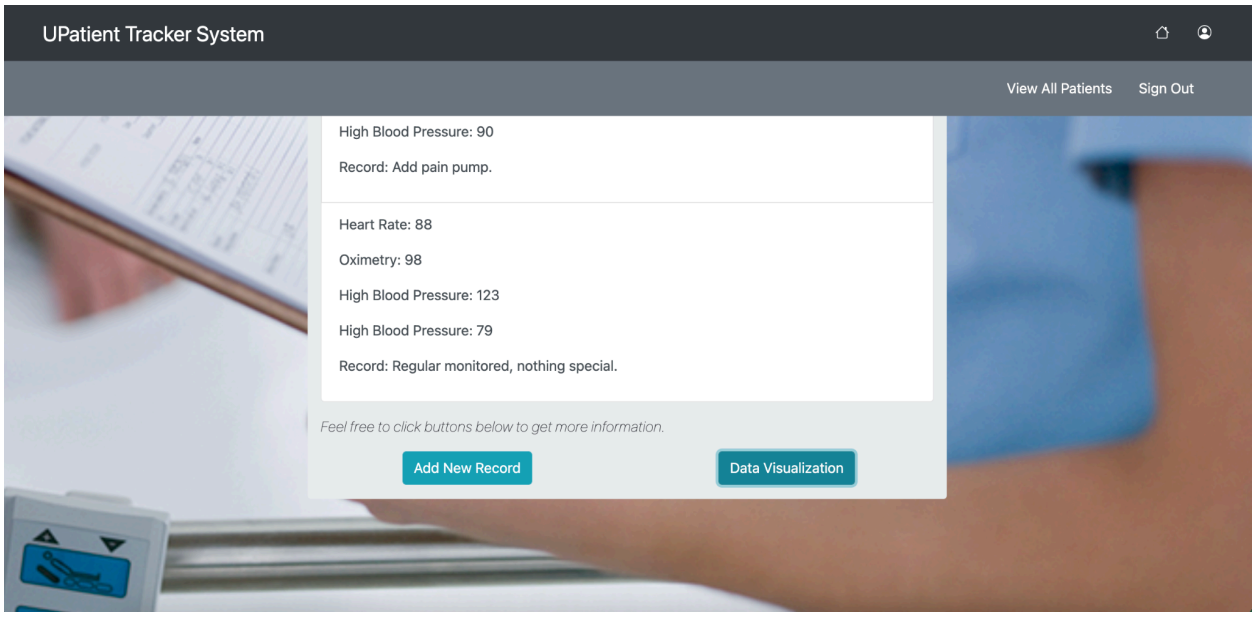
7. View Patient Profile



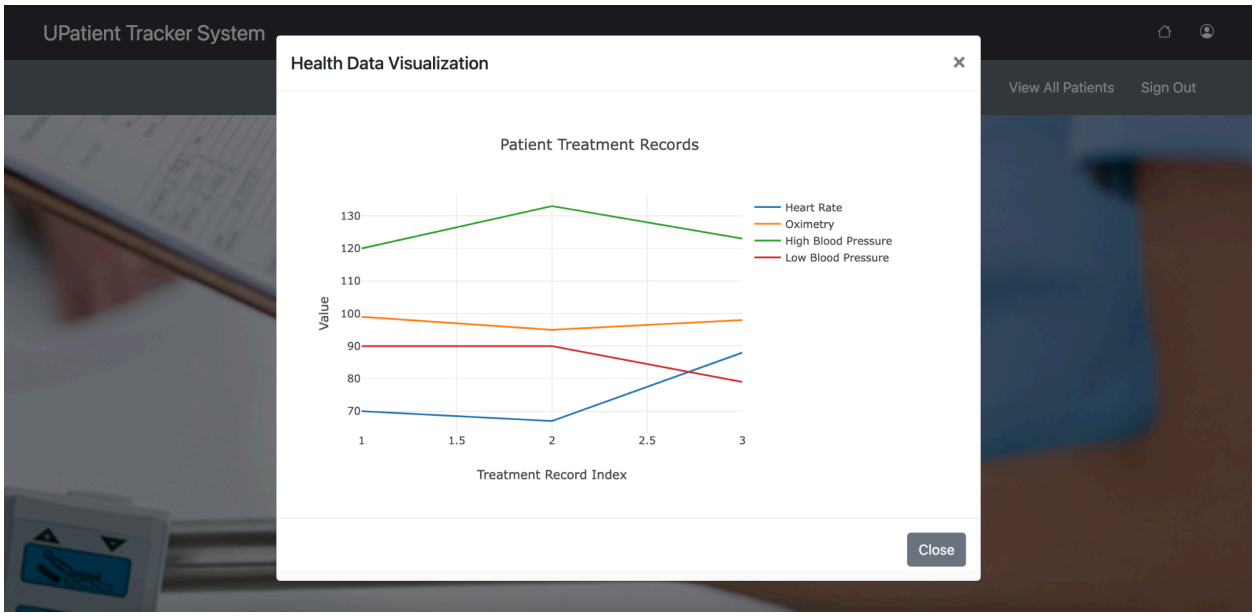


8. View Treatment History



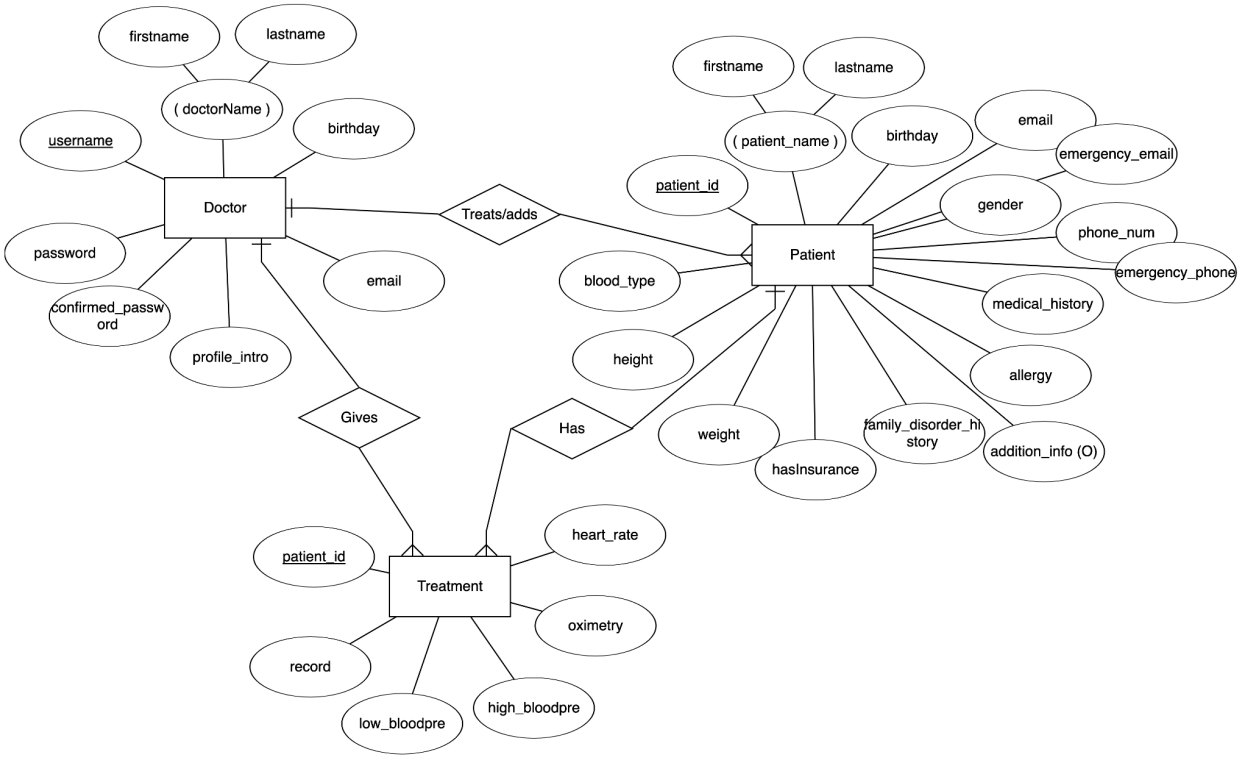


9. Data Visualization

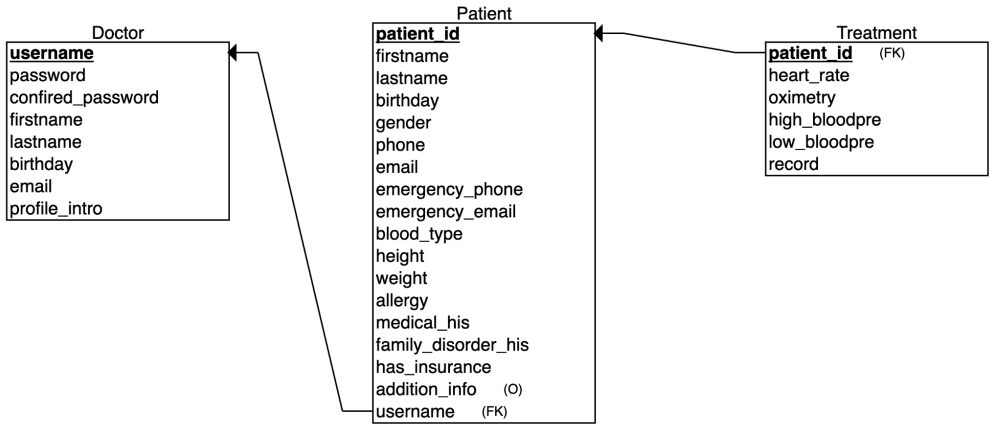


2.4. Data Model

Entity-Relationship Diagram (ERD)



Relational Schema



3. Implementation

3.1. Version Control and Collaboration

During the project building progress, we use Git to manage code changes and we ensured that everyone is up-to-date before and after every push and pull actions. When we faced the conflicts, we tried undo git commits and also force merge. Note that we only did force merge when there's small issues like a button change, but not for huge code modifications.

We didn't use branches because this is a relatively small project, but we intended to use branch functionality in the future project to avoid force merging problem.

3.2. Coding Standards and Practices

Our team maintained clear and detailed naming conventions, such as add_personal_info.html and view_all_patients.html. This helped to indicate the functionalities of the webpages and its corresponding JavaScript and CSS files. We used meaningful and descriptive variable names such as treatmentRecord and oximetry, avoiding single-letter variable name which makes users and programmers confused.

We provided easy-to-follow instructions in the README.md file as well as the gitLog.txt file, which recorded clear git commits that we made for every project modification. We also wrote comments in the code.

We provided unit testing to test the correctness of our project, such as the successful click and successful user registration.

3.3. Security and Risks

For the data security, this project ensures the potential users must be registered doctors, so that they can have access to register an account and login, in order to gain access to the patient and doctor database. We also guarantee the data security by checking matching username and password when logging in.

However, since this is a web-based application, when internet connection is influenced by any reason, the data could be influenced as well, such as failing to update/save change, or failing to delete. In the future, backup and restore functionality should be included to mitigate the risk of such data loss.

4. Evaluation

4.1 Evaluation of Functional Requirements

Unit tests were written for testing the success of the project. One of the tests passing screenshot is attached below:

```
__tests__ > JS login.test.js > ...
const { JSDOM } = require('jsdom');
const fs = require('fs');
const path = require('path');

// Load the HTML file for testing
const html = fs.readFileSync(path.resolve(__dirname, '../login.html'), 'utf8');
const { window } = new JSDOM(html);
global.document = window.document;

// Load the JavaScript file for testing
const loginScript = require('../login.js');

describe('login.js', () => {

  it('should handle sign-in click and save doctor info on success', async () => {
    // Mock the fetch function to return a successful response
    global.fetch = jest.fn().mockResolvedValue({
      json: jest.fn().mockResolvedValue({doctorInfo}),
    });


    // Mock the localStorage functions
    const localStorageMock = {
      getItem: jest.fn(),
      setItem: jest.fn(),
      removeItem: jest.fn(),
    };
    global.localStorage = localStorageMock;

    // Mock the window.location.href and alert functions
    const locationMock = { href: '' };
    global.window = { location: locationMock, alert: jest.fn() };

    // Simulate the click event on the signin button
    const signinButton = document.getElementById('signin');
    await signinButton.click();

    // Verify that the fetch function was called with the correct arguments
    expect(fetch).toHaveBeenCalledWith('/login', {
      method: 'post',
      headers: {
        'Content-Type': 'application/json;charset=utf-8',
      },
      body: JSON.stringify({
        username: '',
        password: '',
      }),
    });

    // Verify the localStorage functions were called with the correct arguments
    expect(localStorageMock.setItem).toHaveBeenCalledWith('currentDoctor', JSON.stringify({ /* Sample doctorInfo */ }));

    // Verify the console.log was called
    expect(console.log).toHaveBeenCalledWith('Save successfully');

    // Verify the window.location.href was set to "dashboard.html"
    expect(locationMock.href).toBe('dashboard.html');
  });
});
```

```
PASS  client/__tests__/signup.test.js
PASS  client/__tests__/add_personal.test.js
PASS  client/__tests__/add_medical_his.test.js
PASS  client/__tests__/login.test.js

Test Suites: 5 passed, 5 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        1.364 s
Ran all test suites.
```

4.2 Evaluation of Non-functional Requirements

Usability: Users are provided with detailed instructions of how to use the application. The buttons are also reflectable with clear text descriptions on them. Pop-up windows and error messages will be displayed accordingly.

Understandability: Users and programmers are provided with README.md file, which indicates how to use the application, basic UI information, database information, and the division of labors of this project.

Modularity: This project contains well-organized source code with client-side folder and server-side files. Every HTML file written in Bootstrap has one corresponding JavaScript file to implement the functionality. Also, the mongoDB database is separated from the client-side files and server-side files to maintain the clearness.

Security: This application guarantees the security of user data and input data to never use for any other purpose. Only registered users, i.e. doctors, can create an account in this application. Moreover, patient information will never be displayed before users choose to log in the application.

Maintainability: Modifying and updating the system is easy and straightforward. User interfaces could be adjusted in the HTML files and any functionality adjustment can be finished in corresponding JavaScript files.

5. Discussion [Limitations & Future Plans]

5.1. Challenges and Limitations

The biggest challenge is the time limit. All of us were overwhelmed by many projects and finals, so we started late and did not have enough time to finish all the desired functionalities. Integration is also another challenge because we figured out that when we built a website and implemented the UI, we have different wording conventions, so that the pages with the same functionalities may have different wording for the same thing, such as “medical records” and “treatment history”. We need to find out all the distinct wordings and make them coherent.

5.2. Future Development Plans

The username and password functionality is relatively simple in this project. In the future, we wish to employ SALT and HASH to encrypt and decrypt the password for safer data security. Also, our project is focused on the doctor version and lack of patient version. In this case, we wish to add an entrance of patients so that patients could have their ability to make appointments with their target doctors rather than only allowing doctors to make appointments.

We also would like to add interfaces for the patient to view their portal and messages that were sent from their doctor. Moreover, we would like to add an appointment system so the doctor and the patients can view their appointment time and get notification on that.

5.3. Ethical and Societal Implications

Since we only give the permission for doctors, we ensured that our patient data handling adhered to ethical standards, maintaining privacy and confidentiality. We also guarantee that user will be provided clear instructions while using this application and the data saved won’t be used for any other purpose.