CS 5200 - Database Management Systems Project Report
Clinic Management System

Yanying Xiang, Yutong Geng

December 2023

1 Introduction

In the ever-evolving landscape of healthcare, efficient clinic management plays a pivotal role in delivering exceptional patient care. Recognizing a prevalent challenge within numerous private clinics in China, where many still rely on manual paper-based information management systems, our team seeks to bridge this gap by introducing a comprehensive solution – the development and implementation of a Clinic Management System (CMS) tailored specifically for private clinics.

This preliminary version of the system, leveraging technologies like Python Flask for backend development and React for a dynamic user interface, aims to establish a fundamental digital framework. While the current functionalities are primarily focused on basic patient record management and appointment scheduling, it serves as the initial step towards digitization.

This report outlines a user guide for the application platform, designs the database schema, elucidates the procedural flow of the application, and concludes with a reflective overview and future prospects for the project.

Our goal is to empower private clinics with an advanced digital infrastructure, enhancing their operational efficiency, and ultimately elevating the overall experience for both healthcare providers and patients. While our current application is in its nascent stage, we look forward to continually expanding its features, including more sophisticated data management and communication capabilities, to comprehensively support the delivery of high-quality healthcare services.

2 Project Discription

Our project focuses on building a comprehensive database to store and manage critical data while providing a user-friendly interface for various user operations. The database will encompass a wide range of information, including employee and patient records, appointment schedules, medical histories, prescriptions, medication details, and more.

2.1 User Focus

The primary users of this system encompass both clinic employees and patients, each benefiting from tailored views and access levels based on their respective roles and needs. Upon logging in, users will be directed to different interfaces based on their roles. Patients can create accounts in the 'Register' interface. Employees' accounts are created and managed by administrators, and accounts are generated for them based on their names and birthdates, along with default passwords. The administrator account cannot be generated from the frontend, as we intend

to have only one administrator. Clinic management personnel can access an overview of the clinic's operations through this account.

2.2 Patient Perspective

Patients gain control over their healthcare journey with the ability to schedule or cancle appointments, access detailed information of appointments booked and review prescriptions through the user-friendly interface.

2.3 Doctor and Nurse Perspective

Medical professionals, including doctors and their paired nurses, enjoy a collaborative platform. They both have access to their appointment and their patient medical records. But only doctors could update patient records, and issue prescriptions collectively, promoting streamlined and efficient patient care.

2.4 Manager Perspective

Clinic managers wield powerful oversight capabilities within the system. This includes monitoring day-to-day operations, accessing visualized data on the clinic's financial status, patient statistics, appointment informations and also can add or delete employees.

3 README:

Technologically, the system is powered by React for the frontend to provide an intuitive user experience, while the backend is developed using the Python Flask framework. Additionally, MySQL is employed for secure and efficient data management.

For setting up the environment, the following software installations are necessary:

1. Python (version 3.8 or higher)
2. Node.js and npm for handling frontend dependencies
3. MySQL for the database platform

The installation process includes:

1. Cloning the repository or get the source code zip file.
2. Setting up the backend: Navigating to the backend directory and installing Python dependencies like Flask(**pip3 install flask**), Flask-CORS(**pip3 install flask-cors**), and PyMySQL(**pip3 install pymysql**).
3. Configuring the frontend: Moving to the frontend directory and running **yarn install** to download necessary Node modules.

Proper configuration of the .env files should be created in both frontend and backend directory.

```
For .env in frontend directory:
REACT_APP_BASE_URL=http://localhost:3000
REACT_APP_API_BASE_URL=http://127.0.0.1:5000

For .env backend in backend directory:
DATABASE_NAME=[your database name]
DATABASE_USER=[your database username]
DATABASE_PASSWORD=[your database password]
```

Initial database setup involves populating the MySQL server with data using the provided SQL dump file. For running the application, the backend server is started with python app.py in the backend directory, and the frontend is launched with yarn start in its respective directory. The system is accessible at http://localhost:3000. It's crucial to ensure that both frontend and backend servers are active for the system to function fully.

The application features a registration page where users can create new patient accounts for a personalized experience. Additionally, to facilitate convenience and rapid testing, the system provides several predefined accounts:

1. Patient Account - Username: johndoe, Password: password123
2. Doctor Account - Username: drsmith, Password: password123
3. Nurse Account - Username: nursejones, Password: password123
4. Admin Account - Username: adminuser, Password: adminpass

This readme can also be reached at: https://github.com/xiangddang/clinic_ms

4 Technical Specifications

4.1 Frontend Development: React.js

- **UI Design**: Utilized Bootstrap for a visually appealing and responsive user interface. Ensured accessibility and user-friendliness across devices.
- **Routing**: Implemented React Router for effective navigation, creating a single-page application with smooth transitions.
- **State Management**: Efficient state management for responsive and interactive UI components.

4.2 Backend Development: Python Flask

- **Initial Plan**: Initially considered Django, but opted for a more streamlined approach.
- **Rationale**: Flask was selected for its simplicity and flexibility, focusing on database connectivity and API provision.
- **Cross-Origin Resource Sharing (CORS)**: Implemented Flask-CORS to address cross-domain issues, enabling seamless interaction between different domains.
- **Database Connectivity**: Chose PyMySQL for its compatibility with Flask, facilitating database connectivity.

4.3 Database Design and Management

- **Focus**: Emphasis on MySQL database.
- **Approach**: Created modularized SQL procedures, functions, and triggers within MySQL, rather than embedding complex SQL in code.
- **Integration**: Invoked database elements via Python code, ensuring clean separation of concerns.
- **Benefits**: Simplified Python code, streamlined database interactions, and enhanced system maintainability.

5 Conceptual Design

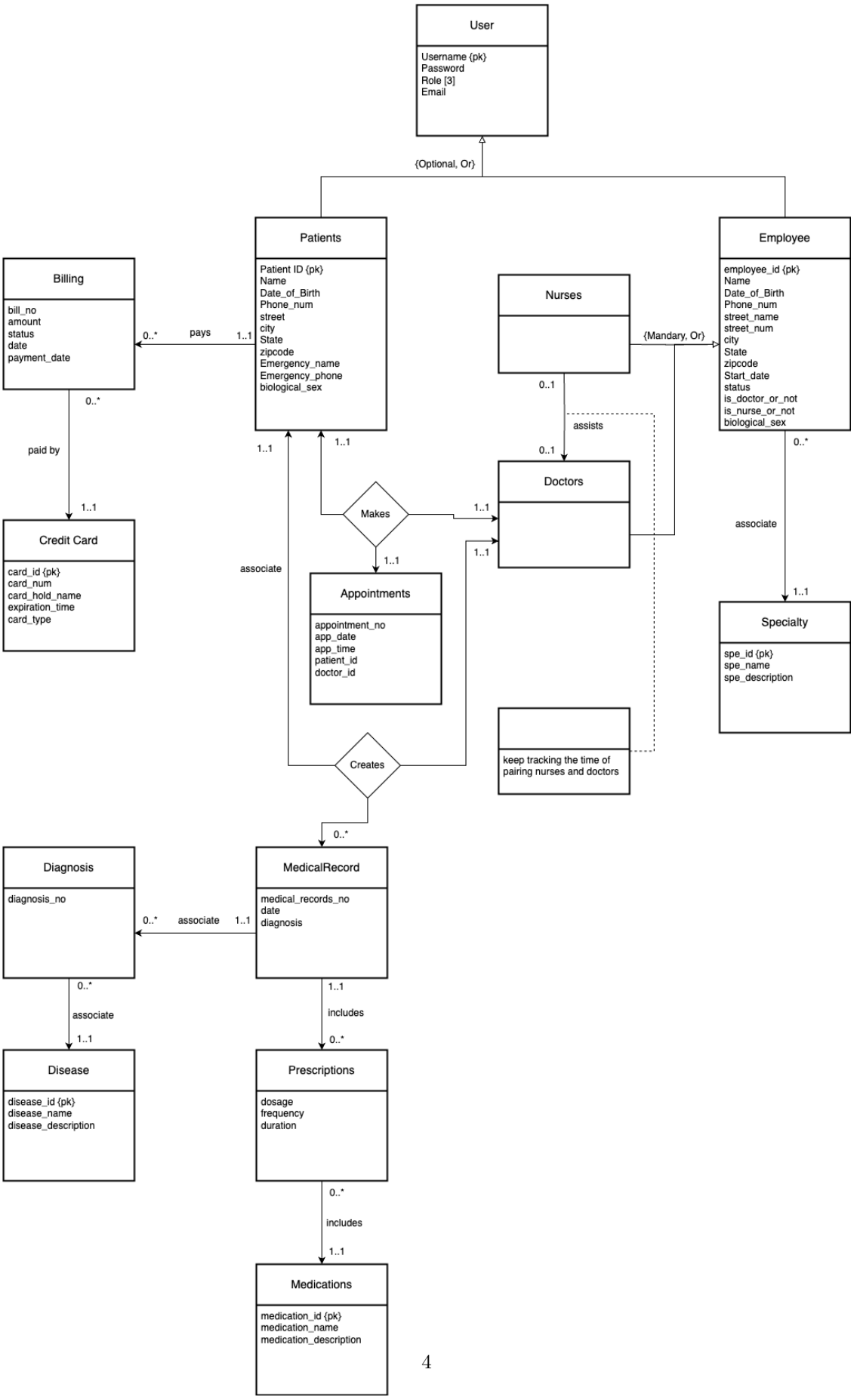

Figure 1: Logical Design

6 Logical Design

For our database design, there are several points should be emphasized.

- We assume there is only a single manager account for the application. Although the 'manager' role exists within the role attribute of User, we do not treat it as a seperate entity because it can only used for the application and there can only be one manager.
- We do not directly link credit cards and patients together. Since one credit card can be used to pay bills for multiple patients, we only associate card information with billing section for financial tracking purposes.
- Doctors can create medical records for patients, which can be understood as records for each medical visit. However, not every appointment necessarily generates a medical record, so it is not directly tied to appointments. Each medical record entry can have multiple diagnoses and medication prescriptions.
- Each doctor is associated with a nurse, but there may be situations where a nurse is not assigned.

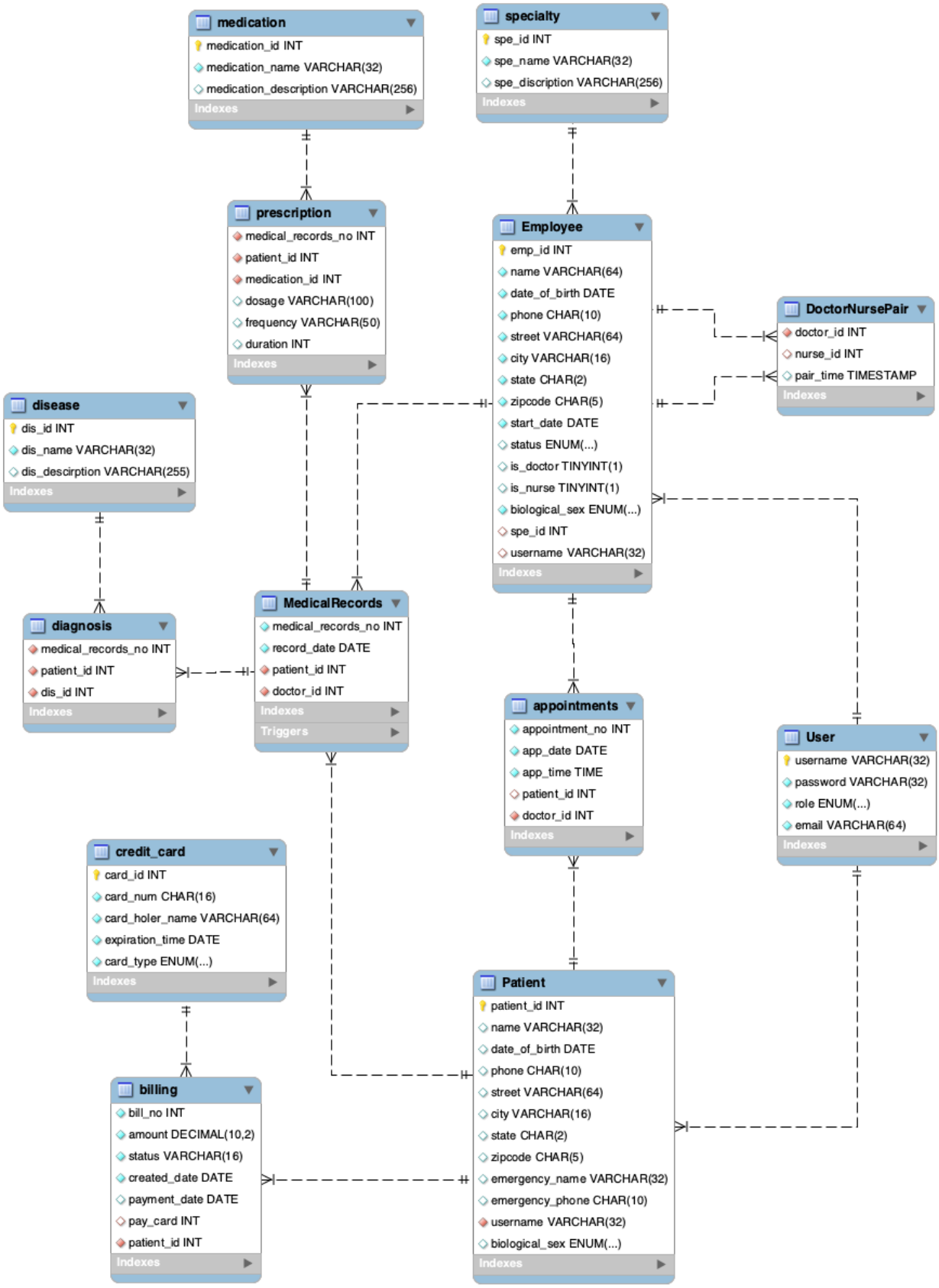

Figure 2: Logical Design

7 User Flow

- Overview
  - The system is designed with a user-friendly frontend that mimics a web browsing experience. Users can interact with the system by performing various actions through the provided user interface.
- User Interactions
  - Login/Registration
    - Users access the system by navigating to the login or registration page.
    - New users register by providing necessary information such as username, email, and password.
    - Returning users log in by entering their credentials.
  - Logout
    - Users can log out of the system by clicking on the "Logout" button located in the navigation bar on the home page, and return to the login page to restart.
  - Dashboard
    - Upon successful login, the system identifies the user's role (patient, employee).
    - Users are then directed to role-specific dashboards:
      - Patient Dashboard: Provides patients with an overview of their recent appointments, personal medical records and access to book a new appointment.
      - Employee Dashboard: Employees see a list of their upcoming appointments, and have the access to check the medical records of patients.
      - Manager Dashboard: The manager sees the employees list, appointments list and patients list, and checks the billing between the selected times.
  - Check/Edit Profile:
    - Users, based on their roles as patients and employees, can view their profiles by clicking the 'Profile' tab on the navbar. Then they will skip to the profile page.
    - To update their profile, users click on the "Edit" button in the profile page, make necessary changes, and then save or cancle the changes.
    - Users can go back to the homepage by clicking on the 'Back' button.
  - Patient Page Functionalities:
    - Book a Visit:
      - Users can initiate the appointment booking process by clicking the "Book a Visit" button.
      - Clicking on the empty input field in the modal reveals a dropdown list of available slots. Users select a preferred slot from the list.
      - The user can either confirm the selected slot by clicking the "Confirm" button or cancel the operation by clicking the "Cancel" button. And the modal dialog box will disappear.
    - View Medical Records:
      - Users can view their latest medical records on the main page.
      - Clicking on the "Check Details" tab redirects the user to a detailed view of the medical record list.
      - A "Back" button is provided on the detailed view page, allowing users to return to the main page.
    - View Appointments:
      - Users can view their latest appointment on the main page.
      - Clicking on the "Check Details" tab redirects the user to a detailed view of the appointment list.
      - In the detailed view, users have the option to cancel an appointment by clicking the "Cancel Appointment" button.

        - A "Back" button is available on the detailed view page for users to return to the main page.

- Employee Page Functionalities:
    * View Appointments:
        - Employees can see a list of appointments involving patients and relevant details on the main page.
        - Clicking on a patient's name redirects the employee to the medical record list of that patient.
    * Access Patient's Medical Records:
        - By clicking on a patient's name in the appointment list, employees can navigate to the medical record list page specific to that patient.
        - A "Back" button is available on the medical record list page to return to the main page.
        - If the user is the doctor, clicking the "Add New Medical Record" button opens a modal box where doctors can input relevant details for the new medical record. Clicking 'Close' or 'Confirm' button to cancle or save the updated information Then the modal box would disappear and the user can see the new record on this page.
- Manager Page Functionalities:
    * Employee Management:
        - Clicking on the "Check Details" tab on the first card redirects the manager to the "Employee List" page.
        - The "Delete Employee" button allows the manager to remove an employee's information card.
        - Clicking the "Add New Employee" button directs the manager to a form to input details and add a new employee information card.
        - The "Back" button returns the manager to the main page.
    * Appointment Overview:
        - Clicking on the "Check Details" tab on the second card redirects the manager to a page displaying recent appointment details.
        - The "Back" button returns the manager to the main page.
    * Check Patient List:
        - Clicking on the "Check Details" tab on the third card redirects the manager to the "Patient List" page.
        - The "Back" button returns the manager to the main page.
    * Billing Information:
        - Clicking on the "Check Details" tab on the fourth card redirects the manager to the "Billing List" page.
        - The manager can select start and end dates from the dropdowns. Clicking the "Search" button to see the billing list within the selected date range.
        - The "Back" button returns the manager to the main page.

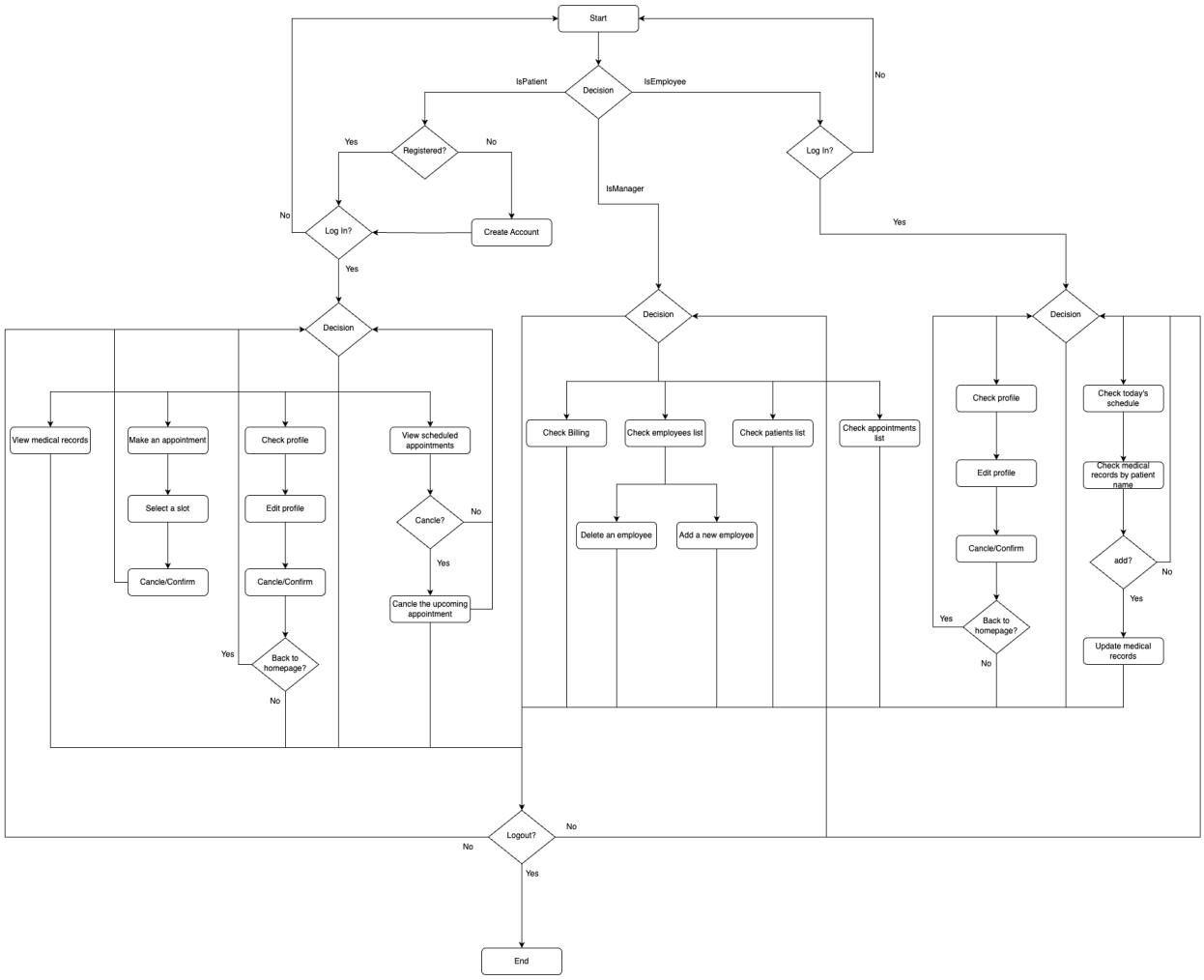

Figure 3: Flow Chart

8 Bonus Meeting

We appreciate this opportnity to deeper our understanding about Database Management System. In the process, we meet some bonus work mentioned in the Project deliverables.

Additional Frontend Functionality: Our project goes beyond meeting the basic user requirements by implementing additional frontend functionality using the React framework. This extra frontend feature presents the entire project in the form of a web application. This not only enhances the user experience but also adds a visual dimension to the project, allowing users to interact with the system intuitively through a web interface.

Complicated Schema: To fulfill the project's requirements, we have designed a complex database schema. We have created 13 distinct database tables and crafted 31 stored procedures, along with one trigger. The relationships between these tables and procedures are intricate, requiring multiple joins and data operations to meet the project's demands.

Complex Translation from User Operations to Database Operations: Our project accomplishes complex translations from user operations to database operations. For example, when an administrator (admin) creates a new employee, they only need to input essential information in the frontend. Subsequently, the system not only creates corresponding records in the employee table but also automatically generates user accounts and adds entries in the DoctorNursePair table. Similarly, the process of deleting an employee involves multiple tables and employs transactions to ensure data consistency and prevent conflicts.

Application Supports Multiple User Roles: Our application caters to multiple user roles, including patients, doctors, nurses, and one administrator (labeled as managers). Each user role enjoys access to distinct pages and functionalities, contributing to the complexity and diversity of the system. Users with different roles can access and perform role-specific actions, providing a personalized user experience.

9 Lessons Learned

9.1 Technical Expertise Gained

Throughout this project, we rapidly adapted to selecting and learning a technology stack based on our requirements. A prime example is our application of Python Flask. Our coding skills were enhanced during the extensive coding process, deepening our understanding of MySQL, Python, and React. Complex logical operations furthered our grasp of database concepts such as transactions.

9.2 Insights and Time Management

Managing time effectively was a crucial lesson learned, especially under the substantial workload of this project. Balancing learning and life, alongside multiple courses, assignments, exams, and this project, has significantly improved our time management abilities.

9.3 Realized or Contemplated Alternative Designs/Approaches

Due to time constraints, many procedures in our database have not yet been applied to the frontend. For future applications of this project, especially to enhance its functionality and management, we consider using Django instead of the current Flask framework.

9.4 Document any code not working in this section

We originally intended to use an event to automatically generate all doctors' appointments for the next week in our system. However, despite many attempts, there were no syntax errors flagged by the bench, but when running, it prompted us to check MySQL syntax. After consulting MySQL documentation, we still couldn't resolve the issue. Currently, we have split the single event into two procedures, which can generate appointment slots for all doctors in the clinic for a specific day.

10 Future work

10.1 Planned Uses of the Database

If we were to apply this project in a practical setting, as initially proposed, we would like to further refine its features for actual clinical operations. One of our team members has parents working in a clinic, providing us with a direct application scenario.

10.2 Potential Areas for Added Functionality

In the future, we aim to add more complex features. For instance, enabling doctors and nurses to communicate simply with patients through the system and implementing more sophisticated appointment scheduling functionalities. Additionally, we plan to introduce a telehealth module for remote consultations, an integrated prescription management system, and a personalized patient portal for enhanced access to health records and direct messaging capabilities.