RMIT UNIVERSITY

Subject Code:	ISYS2099
Subject Name:	Database Applications
Location & Campus:	SGS Campus
Title of Assignment:	**Assessment 2 - Group Project Report**
Lecturer:	Dr. Dang Tran Tri
Group Name:	Group 7 – HD Destroyers
Group Members:	Chau Chan Bang s3975015 Trinh Nguyen Ha s3981134 Hoang Nghia Tri Hung s3930336 Lai Dong Khoa s3926689 Chau Chan Thien s3975010
Assignment due date:	9th of Sept, 2024
Date of Submission:	9th of Sept, 2024
Declaration of Authorship	We declare that in submitting all work for this assessment, we have read, understood and agreed to the content and expectations of Assessment Declaration as specified in https://www.rmit.edu.vn/students/my-studies/assessment-and-exams/assessment

TABLE OF CONTENT

SUMMARY

The Hospital Management System (HMS) is a digital system designed specifically for a small hospital est. in 2017 and based in Ho Chi Minh City, Vietnam; with a major focus on optimizing human side such as managing patient data, staff schedules, and medical histories; it simplifies inventory and resource management like room assignments or pharmaceutical stock levels. Our solution aims to improve traditional healthcare operations by digitizing patient data, staff administration, and appointment scheduling, decreasing the need for manual paperwork. The HMS uses technologies such as MySQL relational database management system (RDBMS) for structured data management, MongoDB/Mongoose for unstructured data, and TypeScript/Nestjs with the support of ORM for web application development to create a strong platform that assures data integrity, security, and easy access to critical information. In the scope of this project, the team focused on creating an enterprise (business) application rather than a consumer-end application.

Assumptions made:

- Since allergies are scientifically discovered and recognized, we have a database of known allergens and will associate the patient with the ones that they acquire.
- When a patient arrives at the hospital for the first time, we will store their personal information (national id, name, age, address, insurance information, blood type, and allergy history if any).
- We store a treatment history (health record) of a patient based on the date they came to the hospital. If a person made multiple visits, they will have multiple treatments for each time.
    - A treatment history may include one or many procedures and upmost one admission.
    - We categorize a procedure into 6 types: Checkups, Prescription, Operation, Ultrasound, Image scans, and Lab. Each type will have a fixed price.
    - Room type for an admission can be Standard or Premium. The discharged date will be predefined.
    - We populate some sample data for medicines for each existing hospital department. Doctors and nurses can search for them while prescribing or using for any procedure that involves a type of drug.
    - A bill will be automatically generated once the treatment status is marked as Completed.
- One staff will have multiple shifts assigned to them. New shifts will be added every week.
    - Staffs role that labeled as ‘Administrative Personnel’ will have one of the highest rights.
    - The person who doesn’t have any direct manager is the President of the hospital.
    - Those who retired or left the job will have a NULL department foreign key.

DATABASE DESIGN

SQL

Conceptual Design

Entity Relationship Diagram and Descriptions *(please refer to the submitted zip file for higher image resolution)*

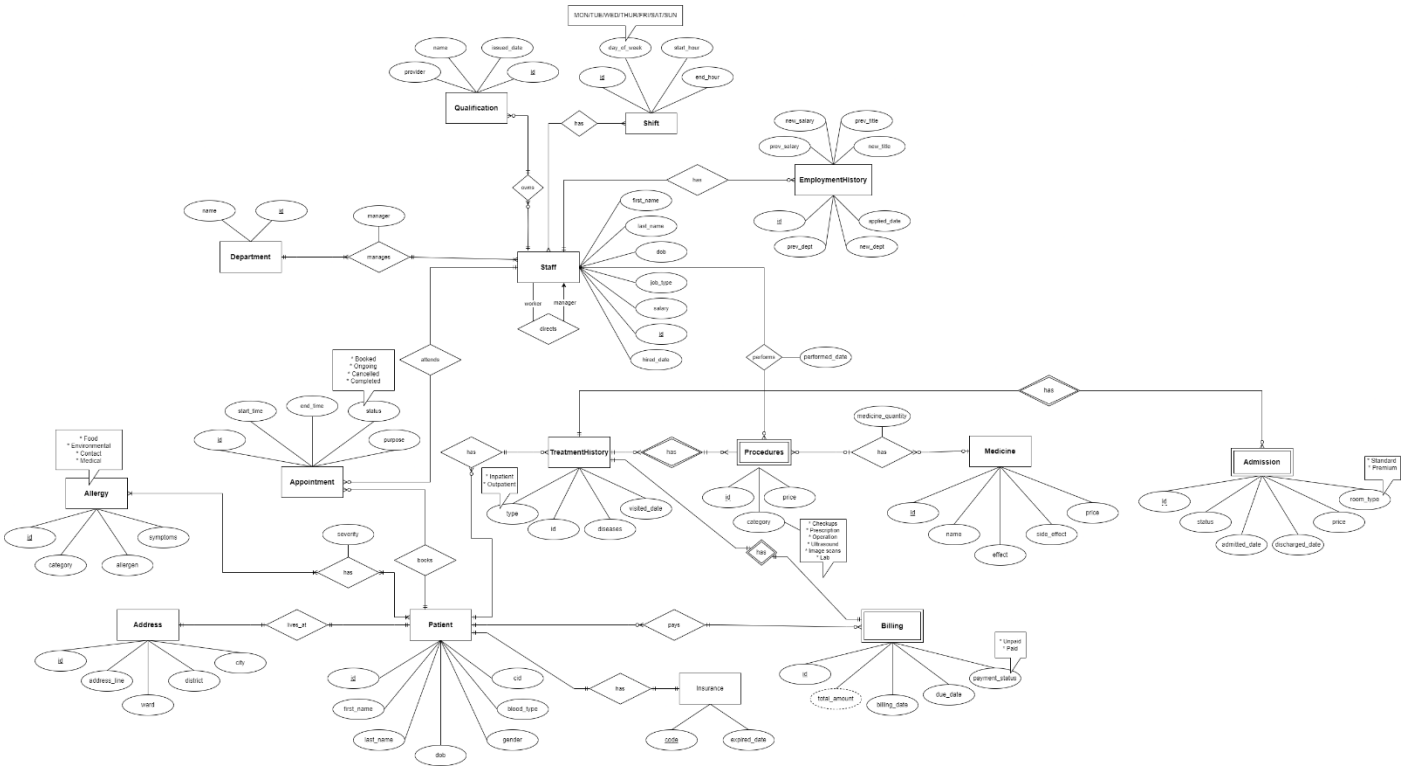

***Figure 1**. Entity Relationship Diagram.*

Entity 1	Relationship	Entity 2	Cardinality
Patient	Has	**Insurance**	One-to-One
	Lives_at	**Address**	One-to-One
	Has	**Allergy**	Many-to-Many
	Has	TreatmentHistory	One-to-Many
	Pays	Billing	One-to-Many
	Books	**Appointment**	One-to-Many
Staff	Directs	Staff	-
	Manages	**Department**	One-to-One
	Has	Appointment	One-to-Many
	Works_in	Department	Many-to-One
	Owns	**Qualification**	One-to-Many
	Has	**Shift**	Many-to-Many
	Perform	Procedures	One-to-Many
	Has	**EmploymentHistory**	One-to-Many
TreatmentHistory	Has	Procedures	One-to-Many
	Has	Admission	One-to-One
Billing	Belongs_to	TreatmentHistory	One-to-One
Procedures	Has	**Medicine**	One-to-One (optional) ** Procedures may involve a specific medicine*

Logical Design

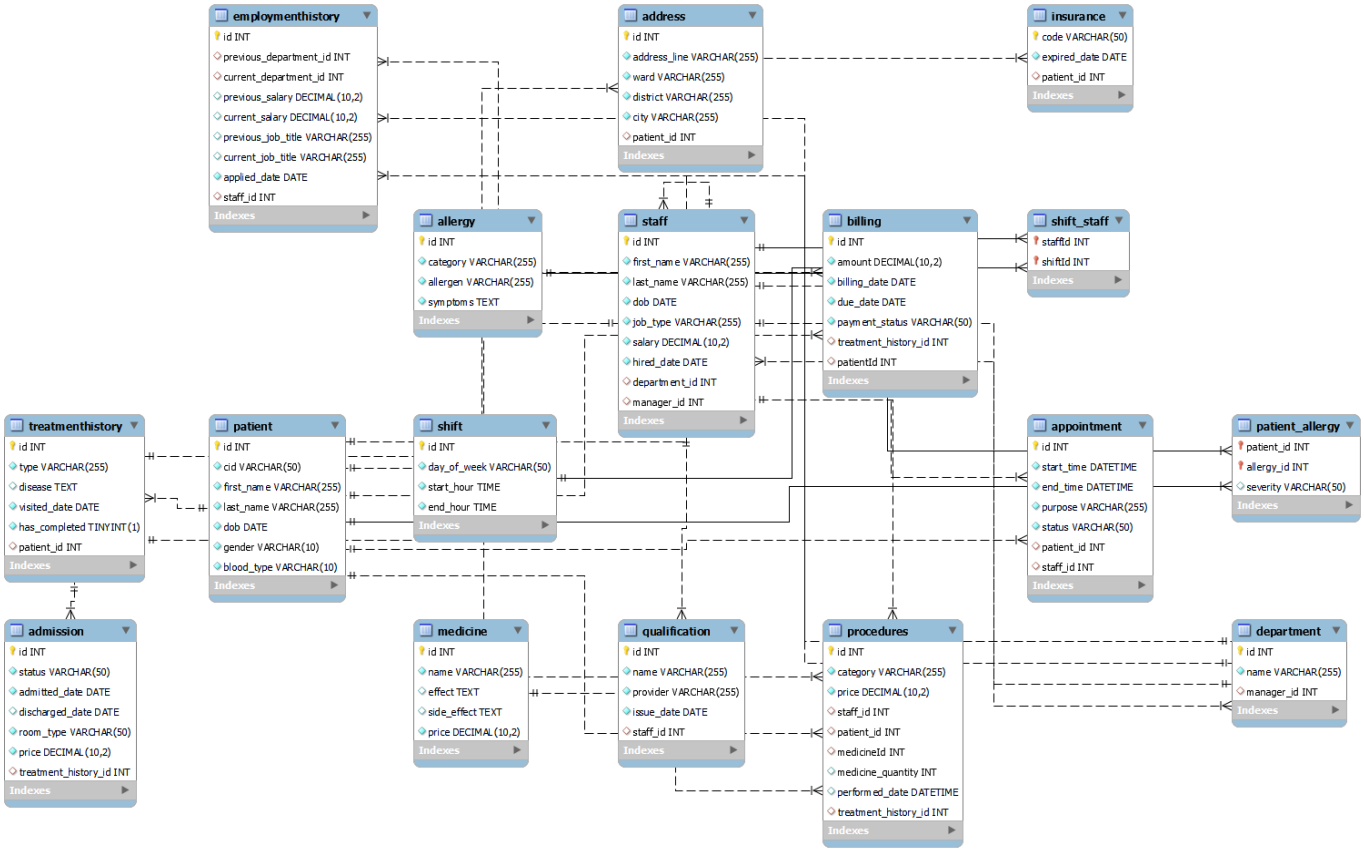

Figure 2*. Using the Reverse Engineer functionality on MySQL Workbench CE, we can generate our Schema Diagram from the database.*

Description: The schema consists of 17 tables deprived from 15 entities and 2 many-to-many relationships.

- **Medicine** (**id**, name, effect, side_effect, quantity, price);
- **Billing** (**id**, amount, billing_date, due_date, payment_status, Patient.id patient, TreatmentHistory.id history);
- **Allergy** (**id**, category, allergen, symptoms);
- **Insurance** (**code**, end_date);
- **Address** (**id**, address_line, ward, district, city, Patient.id patient);
- **Shift** (**id**, day_of_week, start_hour, end_hour);
- **Department** (**id**, name, Staff.id manager);
- **Patient** (**id**, cid, first_name, last_name, dob, gender, blood_type);
- **Staff** (**id,** first_name, last_name, dob, job_type, salary, Staff.id director);
- **Qualification** (**id**, name, provider, issued_date, Staff.id holder);
- **EmploymentHistory** (**id**, prev_dept, new_dept, prev_salary, new_salary, prev_title, new_title, applied_date, Staff.id staff);
- **TreatmentHistory** (**id**, type, diseases, visited_date, Patient.id patient);
- **Procedures** (**id**, category, price, Staff.id performer, Medicine.id medicine, med_quantity);
- **Admission** (**id**, status, admitted_date, discharged_date, room_type, price, TreatmentHistory.id history);
- **Appointment** (**id**, start_time, end_time, purpose, status, Patient.id patient, Staff.id doctor);
- **Patient_Allergy** (**Patient.id patient**, **Allergy.id allergy**, severity);
- **Staff_Shift** (**Staff.id staff, Shift.id shift**);

Additionally, the schema includes several CHECK constraints to validate data entries for fields like day names, treatment types, payment statuses, room types, severity levels, and appointment statuses. DEFAULT values are set for date fields in EmploymentHistory, TreatmentHistory, Billing, and Admission to the current date or calculated values. We also have UNIQUE/NOT NULL constraints for the necessary details are always provided.

NoSQL

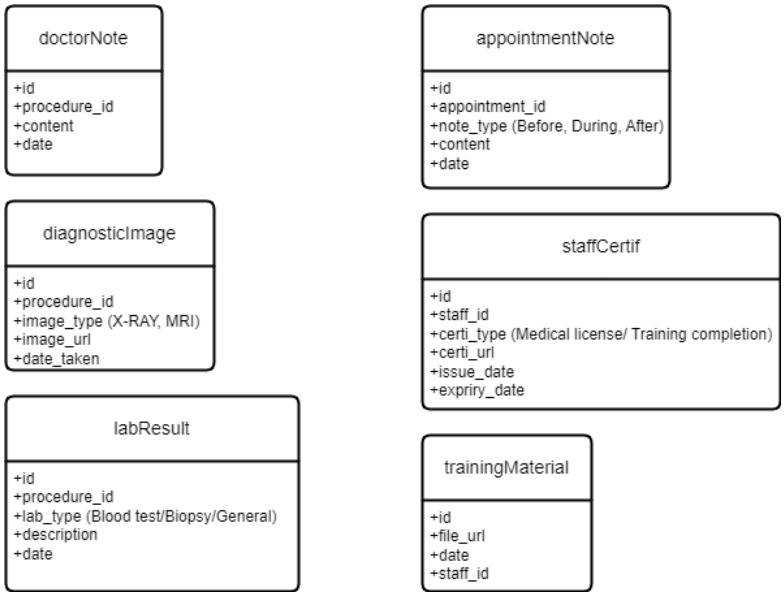

Figure 3. *MongoDB collections for unstructred data.*

- In a procedure:
  - A doctor performing it can add extra notes (doctorNote).
  - An output of some specific procedure category (e.g: Image scans) is an image (diagnosticImage).
  - An output of some specific procedure category (e.g: Lab) is a report (labResult).
- A note can be added before/during/after to a appointment.
- A staff can have their own certificates (staffCertif) and extra documents to aid their careers and personal development (trainingMaterial).

➔ Each document has a foreign key-like field (procedure_id, staff_id, appointment_id) that connects to structured data in the relational database. This guarantees the linking of structured elements and unstructured data.

PERFORMANCE ANALYSIS

Indexes

The indexing strategy serves to the most critical use cases, balancing query performance improvements (neglecting the needs to use full table scans) with the overhead of maintaining resources. Indexes are particularly helpful for read-intensive tables and columns that frequently appear in WHERE, JOIN, and ORDER BY clauses, as well as for columns with less nullable values. In our sample database, the number of records had not yet reached to millions, so the performance shown may not be fully evidential.

- **Primary Indexes** are automatically created on all PRIMARY KEY columns.
- **Foreign Key Constraint Indexes** are also implicitly created.
- **Secondary Indexes** have been implemented on non-prime attributes that are frequently involved in search queries that contains an equality predicate. Particularly on the last_name and first_name columns in the Patient and Staff tables.
- **Composite Indexes** are used to optimize queries that filter or sort data based on multiple columns. For instance, a composite index Appointment (start_time, end_time) checking for appointment availability or conflicts within a specific time frame. Procedures(category, performed_date) helps in efficiently retrieving procedural data which is essential for generating reports and analyzing procedural trends over time.
- **Inverted Indexes** (Full-Text Search) are used in the Allergy table on the symptoms column and the Medicine table on the effects/side effects columns. This full-text search index allows for efficient searching through large text fields, enabling quick retrieval of records based on descriptive conditions.

***Figure 4**. An example showing how the execution time dropped significantly after adding an index on the sorting attribute (0.175s → 0.0609s).*

Partitions

- Initially we planned to create some time-based partitions as documented in the source code, only to have later found out *MySQL 8.x version currently does not support partitioning on tables that have foreign key (fk) constraints*. This limitation is due to the internal architecture of the DBMS's partitioning mechanism, which conflicts with the way FKs are implemented. They ensure referential integrity between tables, but partitioning introduces complexities in maintaining these relationships across partitions, particularly when performing operations like cascading updates or deletes. This inability presents a challenge for maintaining both referential integrity and performance optimization. However, alternative strategies can be considered when it comes to higher scalability:
    - Using triggers or application logic to enforce referential integrity allows partitioning while maintaining data consistency. For example, in a TreatmentHistory table, by removing fk constraints and using triggers to enforce referential integrity, the system can still maintain the relationship between TreatmentHistory and Patient. A trigger can be set up to ensure that before any treatment record is inserted, the patient ID is validated against the Patient table.
    - Simulating partitioning through multiple tables and a view peserves foreign keys but increases complexity because new tables need to be created yearly or based on some other logical criteria. In this approach, instead of partitioning the TreatmentHistory table directly, multiple tables can be created based on time, such as TreatmentHistory_2019, TreatmentHistory_2020, and so on. These tables would maintain their foreign key relationships with the Patient table. A view can then be created to combine these tables into one logical table for querying.
- MongoDB's native sharding provides a scalable solution to manage large volumes of hospital data across multiple servers that tremendously helps with high availability and performance.

Query optimization

Several techniques were used to optimize query performance:

- Only project the necessary columns instead of select all.
- Subqueries were avoided due to their inefficiency in large datasets. Instead, joins were used to improve query performance.
- However, as join operations are rather expensive, we further improved it through Denormalization. By adding patient_id directly to the Billing table, unnecessary join (through TreatmentHistory) was eliminated, reducing query complexity and execution time.
- To handle large datasets efficiently such as when displaying a list of patient, server-side pagination was implemented using LIMIT and OFFSET, reducing memory usage by only retrieving the necessary data.
- Batch retrieval and joins are used to avoid the inefficiency of multiple queries for related data (N+1 problem). Instead of querying each related item separately, all necessary data is retrieved in a single query.

Concurrent access

Considering that multiple users—including doctors, clerks, and administrative personnel—are accessing and modifying data simultaneously. The system handles concurrent access efficiently using several mechanisms.

- Many of the stored procedures, such as BookAppointment and AddProcedure, utilize transactions to ensure atomicity. This means that any operation affecting critical data, like booking an appointment or adding a medical procedure, is executed entirely or not at all. If a conflict or error is detected, the transaction is rolled back, ensuring no partial updates are made.
- The BookAppointment procedure prevents double booking by checking for conflicting appointments before inserting a new one. This ensures that no doctor is overbooked during overlapping time slots, addressing concurrent access where multiple users might attempt to book the same doctor at the same time. By explicitly setting the isolation level to SERIALIZABLE, MySQL will lock entire tables or ranges of rows as needed, preventing phantom reads, dirty reads, and non-repeatable reads.
- In critical tables like Appointment, Procedures, and TreatmentHistory, row-level locking is implicitly handled by MySQL's InnoDB engine, which locks only the rows affected by a transaction. This ensures that concurrent access to the same data does not result in data corruption or conflicts.
- Procedures like InsertNewPatient and AddAdmission work independently of each other, ensuring that data related to patients, admissions, and treatments is handled without overlapping processes. This minimizes the chance of race conditions where two users attempt to modify the same data simultaneously.
- In some cases, the HMS uses optimistic concurrency control, which assumes that most operations will not conflict. For example, CheckStaffAvailability first checks if a staff member is available during a specified period before making changes. This minimizes unnecessary locking of records while ensuring that staff availability is consistently managed.

DATA INTEGRITY

Triggers for specific scenarios

a) Updating medical records process
Insert a new admission for a treatment will change its type to Inpatient (Outpatient if removed)

b) Paying bills process
After the status of a treatment is changed to Completed, insert a new billing record with the corresponding treatment_history_id in case there is none [Check if we have a bill id generated already for the treatment, and someone decided to change the status of the treatment back to Incomplete]. The bill amount will be calculated with a user-defined function (all procedures + an admission (if applicable)). Next, automatically calculate the due_date by 1-month time from the current system date.

c) Human resource management process
When a staff has some modifications in their information (job title, salary, or department change), insert a new record into the EmploymentHistory table.

d) Appointment management process
Toggle between Booked-Ongoing and Ongoing-Completed when the start_time or end_time passed the current date.

Transaction management

Stored Procedures and Transactions

- For optimal concurrency, it is important to keep transactions short and avoid prolonged operations that hold locks for extended durations.

Below is a table that summarizes what we have implemented in our project:

Interface	Features	Relevant tables/collections	Supporting database objects
Patient	Register a new patient [INSERT]	Patient Insurance Address Allergy	- Stored procedure that supports Full-text search on the symptoms of the allergy and display the table of associated allergens. - Stored procedure to add a new patient with or without address and insurance. Transaction to rollback partial inserts when some errors emerge. - Stored procedure to attach multiple allergy records to a patient.
	Search by name or by ID [QUERY]	Patient	- Secondary index on the first_name and last_name fields of the Patient table to enhance name-based search performance. - Stored procedure to search a patient by ID or name.
	Add a treatment [INSERT]	TreatmentHistory Procedure Medicine Staff Admission Billing	- Function to calculate the procedure costs based on the category and/or the medicine used. - Stored procedure to check if an associated doctor is available at the time the procedure is happening. - Stored procedure with transaction handling to add a procedure (esp the associated staff availibility). - Trigger to automatically generate a bill when the treatment history is marked as finished.
	Add a custom object [INSERT]	Procedures labResult doctorNote diagnosticImage	
Staff	Add a new staff [INSERT]	Staff Department Qualification	- Stored procedure to add a new staff with their department and qualifications.
	List the staff by department [SELECT]	Staff Department	- Secondary index on the first_name and last_name fields of the Staff table. - Stored procedure with IN param as department name/id and output the list of staff in it.
	List the staff by name (ASC and DESC order) [SELECT]	Staff	- Implement LIMIT and OFFSET for paginated results. - Create a view to order staff names in ascending or descending order.
	Update a staff information [UPDATE]	Staff	- Trigger (type: AFTER UPDATE on Staff) will insert a new record into the EmploymentHistory table.
	View a staff schedule [SELECT]	Staff Shift Shift_Staff	- Secondary index on the start_time and end_time fields of the Shift table. - Create a stored procedure to display staff schedule.
	Update a staff schedule [UPDATE]	Shift Shift_Staff Appointment	- Stored procedure to add a new shift for a staff. - Stored procedure to check if a staff is occupied within a shift that is about to be deleted.

			- Add a unique index on (day_of_week, start_time, and end_time).
	Add a custom object (notes, images, etc.)	Procedures Appointment	
Appointment	View working schedule of all doctors for a given duration (with busy or available status) [SELECT]	Shift_Staff	- Stored procedure that display all doctors availibity in a given timeframe. If they had appointments in that time, also include these details in the results table.
	Book an appointment with a doctor (for a given time, no more than one appointment for a doctor) [INSERT]	Shift_Staff Appointment	- Uses transaction handling together with a procedure to prevent double booking for doctors. Also consider both doctor availability and their working shifts
	Cancel an appointment with a doctor [UPDATE]	Appointment	- Stored procedure cancel incomplete appoinments.
	Add a note	Appointment appointmentNote	
Report	View a patient treatment history for a given duration	TreatmentHistory Admission Procedures Billing	- Stored procedure to list all treatment history of a patient_id. - Stored procedure to list all procedures and admission related to a particular treament_history_id
	View job change history of a staff	EmploymentHistory Staff	- Create a procedure to get job change of a particular staffID
	View the work of a doctor in a given duration	Staff Shift_Staff	- Utils function to get the day of week name of the current date (to later compare it with the day_of_week column in Shift_Staff table)
	View the work of all doctors in a given duration	Staff Shift_Staff	- Stored procedure to list the procedures of all staff in a given time range.

DATA SECURITY

Input Validation

- We apply constraints directly on entity properties by using decorators from the “class-validator” library so that validation rules are defined within our entity classes, making it easier to understand and reduce repetitive validation logic in our controllers. Furthermore, this practice ensures the maintainability and reusability across endpoints.

Data Encapsulation

- We apply the practice of Data Transfer Objects (DTO) to efficiently transfer data between the database layer and the presentation layer. Thus, we can transform data to appropriate formats that can exclude sensitive data and is suitable for display in the user interface.

Environment Variable Management

- Sensitive information such as database URLs and other configurations are stored in a separate secret .env file and these settings will be loaded into our application's runtime environment. As a result, we can prevent our codebase from being exposed and vulnerable to cyber threats. Moreover, this practice allows us to easily change the application's configuration without much modification, which is useful for deploying to different environments.

SQL Injection Prevention

- We use TypeORM to define entities and establish their relationships to one another. Moreover, TypeORM prevents SQL injection by automatically using parameterized queries to interact with the database. Therefore, the user input is treated as data, not directly as part of the SQL query, reducing the risk of malicious scripts being executed.

Data encryption

- For the authentication of the application, "bcrypt" library is used for hashing sensitive credentials to ensure that in case there is a data leak, the hackers cannot easily access into the system under a user account.

Database permissions

- The permission structure is designed such that each user role has the appropriate level of access based on their responsibilities, while maintaining data security and system integrity. We defined four key roles: admin, clerk, doctor, and patient, each with carefully assigned privileges (RBAC):

  + Admins have full control over the system. They can access all tables and perform any operation, including creating, updating, deleting, and managing user roles. This is essential for overall system management, including configuring users and ensuring smooth operation across all functions.
  + Clerks handle patient registration, appointments, and billing. They have the ability to read, insert, and update records related to patients and staff but cannot delete data. This prevents accidental data loss while still allowing clerks to carry out their day-to-day tasks. Clerks also have access to a custom view, ClerkAppointmentView, which simplifies appointment management by hiding sensitive patient information.
  + Doctors are granted access to the medical data they need to treat patients. They can view and update patient records, treatment histories, procedures, and prescriptions, but they can't delete records to prevent any loss of important medical data. Doctors also use a special view to quickly access patient treatment histories without navigating through unnecessary data.
  + Patients are only allowed to view and update their own records, such as personal details, appointments, and billing information. This ensures they can manage their own data while maintaining privacy, as they cannot access or modify any other patient's records.

REFERENCES

[1] MySQL, "12.9.1 Natural language full-text searches," MySQL Documentation. [Online]. Available: https://dev.mysql.com/doc/refman/5.7/en/fulltext-natural-language.html. [Accessed: Sep. 12, 2024].

[2] T. Anh, "FullText search - Đơn giản mà hữu ích!!," Viblo, Sep. 22, 2016. [Online]. Available: https://viblo.asia/p/fulltext-search-don-gian-ma-huu-ich-DXOGRjbPGdZ. [Accessed: Sep. 12, 2024].

[3] MySQL, "Foreign key constraints," MySQL Documentation. [Online]. Available: https://dev.mysql.com/doc/refman/8.4/en/constraint-foreign-key.html. [Accessed: Sep. 12, 2024].

[4] "Business applications," ScienceDirect. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/business-application. [Accessed: Sep. 12, 2024].

[5] H. Kieu, "Improve SQL pagination query with large offset," Viblo, Jun. 29, 2020. [Online]. Available: https://viblo.asia/p/improve-sql-pagination-query-with-large-offset-Rk74aD5rVeO. [Accessed: Sep. 12, 2024].