Project2 Design & Implement

- Patient Management System

Highlight nouns and verbs

1.Introduction

The Patient Management System is a critical healthcare application designed to assist patients in managing patient appointments, medical prescriptions, and the delivery of medications. This system is particularly valuable for streamlining the process of test scheduling, medicine tracking, and ensuring patients receive timely updates and treatments. By integrating patient data, medical test reports, and doctor prescriptions, the system enables efficient tracking of patient health progress and ensures proper medication adherence.

The system is designed to handle various use cases, such as appointment for tests for patients, sending prescribed medications to the patient's home address, and recording the medication cycle. The goal is to improve the overall healthcare management experience for both the patients and medical professionals involved.

2. Rules of the business

1.The Patient Management APP stores each user's most fundamental personal information. This data forms the base layer for managing patient records.

2. Tests are conducted by Labs, and each test is associated with a labId. The Lab Reports are sent back to the system where doctors can review them to prescribe or adjust treatment. Lab Reports are generated based on the test results and contain important information such as illnessName and memo for the doctor's review.

3. Doctors refer Prescriptions to patients after reviewing the Lab reports. Each Prescription contains information such as the date and dosage of Instructions. The system records multiple medications per prescription, each with a unique medicine, medicine name, and dosage frequency.

4. Patients can schedule appointments with doctors through the Patient Management APP. Each appointment records the date and time of the visit, ensuring that the doctor is assigned accordingly. Multiple appointments can be scheduled for the same doctor, and the system prevents double

booking by maintaining unique time slots per doctor.

5. Each medicine issued to a patient is tracked through a Medicine Record. This record logs the date and frequency of medication issuance, ensuring that doctors and the system have an accurate history of the patient's medication timeline. A single medicine can be recorded in multiple medicine issuance records, representing different times it was administered.

6. Doctors have access to all the lab reports for patients under their care. These reports include detailed test results and assist the doctor in making data-driven decisions about adjusting or prescribing treatments. The system ensures that only the assigned doctor has access to each patient's lab reports for privacy and security purposes.

7. After reviewing lab reports and assessing the patient's condition, doctors can adjust prescriptions by changing the dosage instructions or adding/removing medications. This ensures that prescriptions are always up-to-date based on the patient's current health status and the latest lab results.

8. Diseases History only stores records for one month to ensure that patients can have timely follow-up visits.

3. nouns and verbs

Nouns:

Patient Management System

healthcare application

patients

appointments

medical prescriptions

delivery of medications

system

test scheduling

medicine tracking

patient data

medical test reports

doctor prescriptions

progress

medication adherence

appointment

prescribed medications

medication cycle

healthcare management

medical professionals

Patient Management APP

personal information

records

primary functions

Health Surveillance

Medicine Monitor

monitoring

medications

monitor Id (Patient Id)

dosages

Health Surveillance record

symptoms

disease history

Tests

Labs

test

labId

Lab Reports

doctors

Reports

illnessName

memo

doctor’s review

Doctors

Prescriptions

Lab reports

Prescription

information

date

dosage of Instructions

medications per prescription

medicine

medicine name

dosage frequency

Delivery

prescribed medications

address

patient

delivery Id

delivery date

Delivery

Medicine

Verbs

Appointment

Assist

Recording

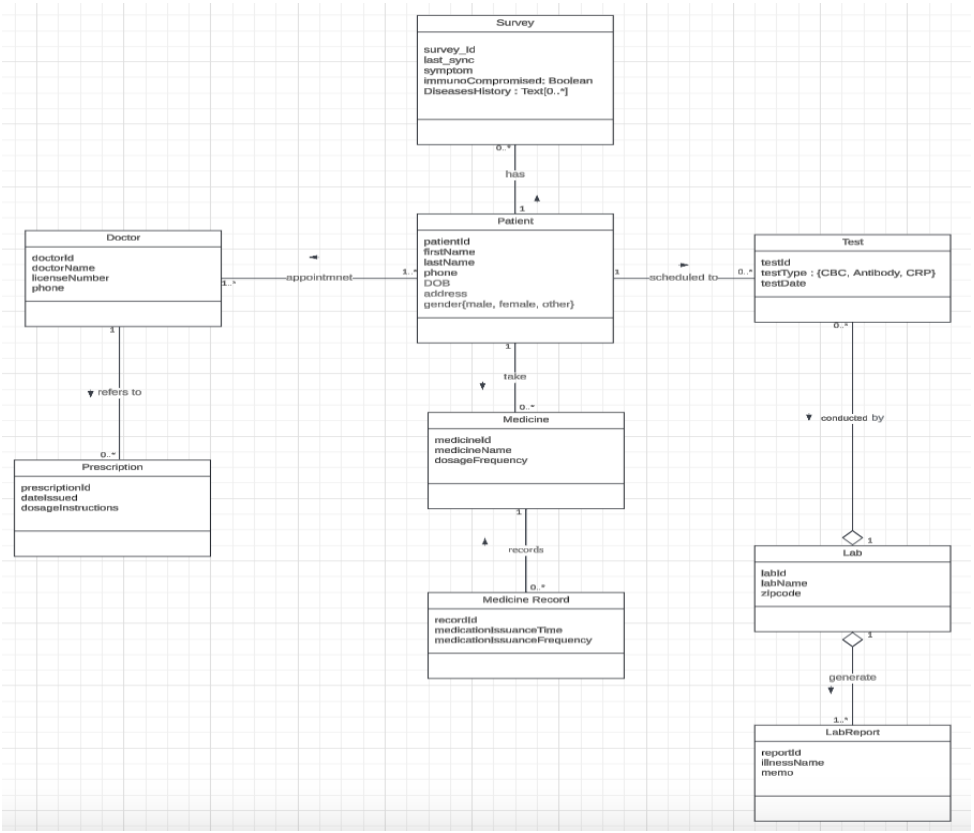
Prescribed

Conducted

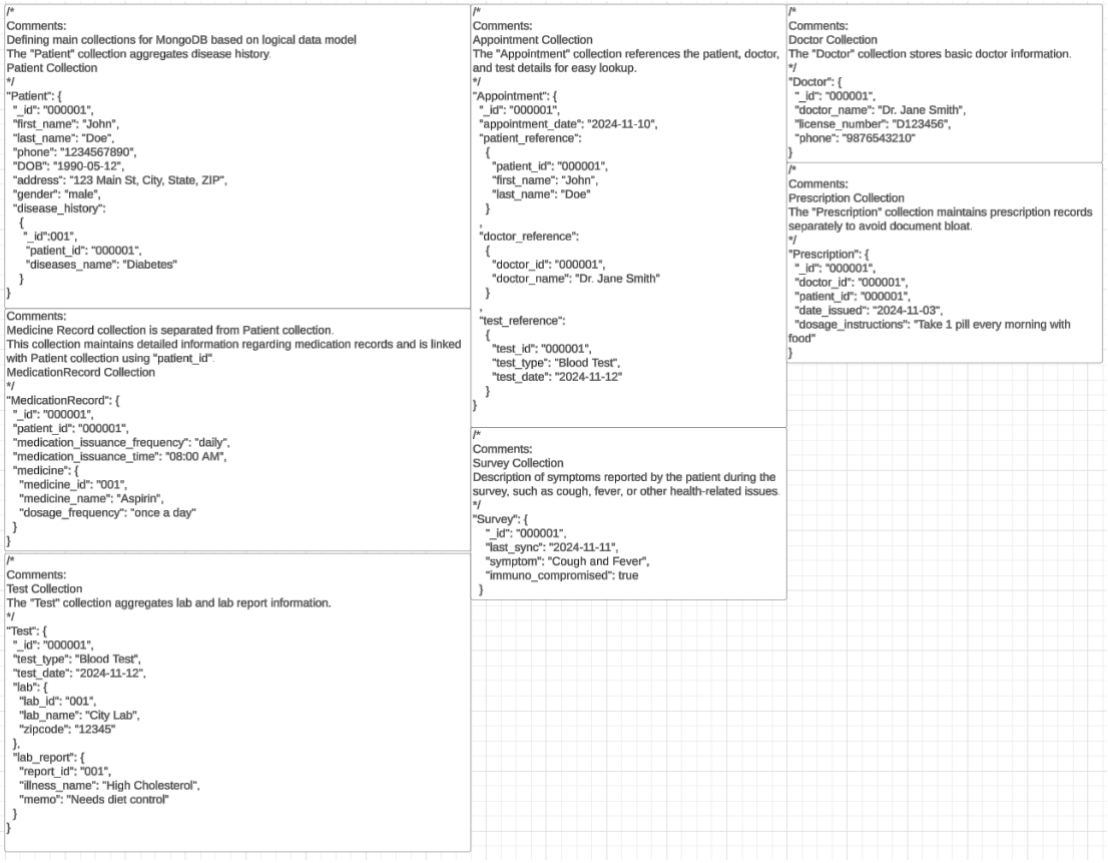
Review

Delivery

4. Conceptual model



5. JSON examples



```
/*
Comments:
Defining main collections for MongoDB based on logical data model
The "Patient" collection aggregates disease history
Patient Collection
*/
"Patient": {
  "_id": "000001",
  "first_name": "John",
  "last_name": "Doe",
  "phone": "1234567890",
  "DOB": "1990-05-12",
  "address": "123 Main St, City, State, ZIP",
  "gender": "male",
  "disease_history":
    {
     "_id":001",
      "patient_id": "000001",
      "diseases_name": "Diabetes"
    }
}
```

```
Comments:
Medicine Record collection is separated from Patient collection.
This collection maintains detailed information regarding medication records and is linked with Patient collection using "patient_id".
MedicationRecord Collection
*/
"MedicationRecord": {
  "_id": "000001",
  "patient_id": "000001",
  "medication_issuance_frequency": "daily",
  "medication_issuance_time": "08:00 AM",
  "medicine": {
    "medicine_id": "001",
    "medicine_name": "Aspirin",
    "dosage_frequency": "once a day"
  }
}
```

```
/*
Comments:
Test Collection
The "Test" collection aggregates lab and lab report information.
*/
"Test": {
  "_id": "000001",
  "test_type": "Blood Test",
  "test_date": "2024-11-12",
  "lab": {
    "lab_id": "001",
    "lab_name": "City Lab",
    "zipcode": "12345"
  },
  "lab_report": {
    "report_id": "001",
    "illness_name": "High Cholesterol",
    "memo": "Needs diet control"
  }
}
```

```
/*
Comments:
Appointment Collection
The "Appointment" collection references the patient, doctor, and test details for easy lookup.
*/
"Appointment": {
  "_id": "000001",
  "appointment_date": "2024-11-10",
  "patient_reference":
    {
      "patient_id": "000001",
      "first_name": "John",
      "last_name": "Doe"
    }
  ,
  "doctor_reference":
    {
      "doctor_id": "000001",
      "doctor_name": "Dr. Jane Smith"
    }
  ,
  "test_reference":
    {
      "test_id": "000001",
      "test_type": "Blood Test",
      "test_date": "2024-11-12"
    }
}
```

```
/*
Comments:
Survey Collection
Description of symptoms reported by the patient during the survey, such as cough, fever, or other health-related issues.
*/
"Survey": {
    "_id": "000001",
    "last_sync": "2024-11-11",
    "symptom": "Cough and Fever",
    "immuno_compromised": true
  }
```

```
/*
Comments:
Doctor Collection
The "Doctor" collection stores basic doctor information.
*/
"Doctor": {
  "_id": "000001",
  "doctor_name": "Dr. Jane Smith",
  "license_number": "D123456",
  "phone": "9876543210"
}
```

```
/*
Comments:
Prescription Collection
The "Prescription" collection maintains prescription records separately to avoid document bloat.
*/
"Prescription": {
  "_id": "000001",
  "doctor_id": "000001",
  "patient_id": "000001",
  "date_issued": "2024-11-03",
  "dosage_instructions": "Take 1 pill every morning with food"
}
```

6. Redis Data Structures and Implementation Details for Simulating Logged-in User Functionality

In simulating the functionality of a currently logged-in user, I effectively utilized Redis to manage and optimize several modules, including patient information caching, online status management, disease history caching, and pending test queue management. These modules are designed to enhance system response speed and overall user experience. Below is a detailed description of these features:

1.Patient Information Cache

Using Redis Hash to store detailed information about each patient, such as name, phone number, address, date of birth, etc.

When patient information is requested, the system first attempts to retrieve it from Redis. If the information is not found in the cache (possibly due to it being a first-time access or the cache expiring), it is retrieved from MongoDB and then cached in Redis for subsequent access.

This design significantly reduces the number of direct queries to MongoDB. For frequently accessed patient data, it minimizes query latency and improves system performance and response speed.

Cache Example: For a patient with ID 000002, the Redis key is patient: 000002, storing fields such as first_name, last_name, phone, etc. This allows for easy reading and updating of fields using the hash.

2. Disease History Cache

Using Redis Hash to cache the patient's disease history.

Each disease history entry is stored as an individual hash, allowing for convenient retrieval and updating. For example, if a patient has multiple disease records, each disease history will have a unique Redis key, such as diseaseHistory: 000002 :diseaseID.

This design allows each disease record to be managed individually and accessed quickly via Redis, improving query efficiency. If a patient has multiple disease histories, Redis caching avoids having to retrieve all history records from MongoDB every time, thereby improving data access performance.

Cache Example: For a patient with ID 000002 and a specific disease history ID 002, the Redis key is diseaseHistory: 000002:002, storing fields such as diseases_name, etc.

3. Online Status Management

Using Redis String data structure to store the online status of each patient.

When a patient logs into the system, their online status is set to "true"; when they log out or leave, it is set to "false". This status information is also given a one-hour TTL (Time to Live) to ensure real-time and accurate status management.

If a patient is inactive for one hour, the status information automatically expires and is removed, helping the system to automatically manage online statuses and avoid showing outdated information.

Status Management Example: For a patient with ID 000002, the Redis key is onlineStatus:12345, with a value of "true" or "false", and an expiration time of 3600 seconds (1 hour).

4. Pending Tests Queue

Using Redis List to manage pending tests for each patient.

Each patient has a queue of pending tests, where tests can be added via lPush and all pending tests can be retrieved via lRange. Redis lists are a LIFO (Last In, First Out) data structure, making them well-suited for managing the sequence of pending tasks.

When a patient has a new test to be scheduled, the system can add it to the head of the Redis list queue. All pending tests for the patient can also be retrieved efficiently using the lRange method.

Pending Tests Queue Example: For a patient with ID 000002, the Redis key is pendingTests:000002, with values such as ["Blood Test", "X-Ray"], representing the tests the patient needs to undergo.

This design with Redis ensures effective data management for the simulated logged-in users by providing efficient caching, status management, and queue management functionalities. Leveraging Redis's caching capabilities significantly improves data access speed and system responsiveness while reducing database dependency. This design also lays a solid foundation for future expansions, such as adding permission controls, access rate limiting, and notification features, comprehensively simulating a real user management system.

7. Redis Commands

Patient Information Cache

1. Create

   HSET patient:000002 first_name "Alice" last_name "Johnson" phone "34567890121" DOB "1978-02-15" address "789 Pine St" gender "female"

2. Read

   HGETALL patient:000002

   HGET patient:000002 phone

3. Update

   HSET patient:000002 phone "98765432109"

4. Delete

   HDEL patient:000002 address

   DEL patient:000002

Disease History Cache

1. Create

   HSET diseaseHistory:000002:003 disease_name "Asthma" patient_id "000002" date_diagnosed "2022-03-10"

2. Read

   HGETALL diseaseHistory:000002:003

   HGET diseaseHistory:000002:003 disease_name

3. Update

   HSET diseaseHistory:000002:003 disease_name "Chronic Asthma"

4. Delete

   DEL diseaseHistory:000002:003

Online Status Management

1. Create

   SETEX onlineStatus:000002 3600 true

2. Read

   GET onlineStatus:000002

3. Update

   SETEX onlineStatus:000002 7200 true

4. Delete

   DEL onlineStatus:000002

Pending Tests Queue

1. Create

   LPUSH pendingTests:000002 "Blood Test"

   LPUSH pendingTests:000002 "X-Ray"

2. Read

   LRANGE pendingTests:000002 0 -1

   LINDEX pendingTests:000002 0

3. Update

   LSET pendingTests:000002 1 "MRI Scan"

4. Delete

   LREM pendingTests:000002 1 "X-Ray"

   DEL pendingTests:000002