소프트웨어 설계 기술서

(Software Design Document)

: 분산 DB 기반의 회의실 예약 시스템

분산처리 기초

2025.06.13

데이터사이언스학과

12211262

우지민

[목 차]

1. 개요

1.1 설계 목적

본 프로젝트의 목적은 MongoDB의 Replica Set 구조를 기반으로 하는 분산 데이터베이스 환경을 구축하고, 이를 통해 리더(Primary)와 팔로워(Secondary) 노드 간의 데이터 복제, 장애 복구(failover), 정족수 기반 리더 승격 메커니즘을 실습 및 검증하는 데 있다. MongoDB는 기본적으로 단일 Primary 노드를 통해 쓰기 작업을 수행하며, Secondary 노드는 Primary로부터 데이터를 복제하여 동기화를 유지한다. 본 시스템에서는 추가적으로 Arbiter 노드를 포함하여, 데이터 저장 없이 리더 선출 투표에만 참여하는 구조를 적용함으로써, 2개의 데이터 노드와 1개의 경량 노드만으로도 자동 리더 승격이 가능한 구성을 실현하였다. 이를 통해 시스템은 리더 노드가 비정상 종료되었을 경우에도 Secondary 노드와 Arbiter의 다수결 투표를 통해 새로운 리더를 자동으로 선출할 수 있으며, 리더 노드 복귀 시 자동으로 기존 노드가 팔로워로 복원되어 복제 관계를 재구성하는 과정을 실험할 수 있다. 이러한 분산 환경을 바탕으로, 리더 노드에 Flask 기반의 경량 웹 애플리케이션을 구축하였고, 웹 페이지에서는 사용자가 사용자 목록, 회의실 목록, 예약 목록을 조회하고 새로운 예약을 등록할 수 있는 기능을 제공한다. 웹 인터페이스는 Primary 노드와 직접 연결되어 사용자 입력을 DB에 반영하며, 입력된 데이터는 자동으로 Secondary 노드로 복제된다.

1.2 설계 환경

본 프로젝트에서는 Python 기반의 웹 프레임워크인 Flask를 사용해 설계를 진행하였으며, Python은 버전 3.12.3을 사용하였다. Linux 기반의 가상머신(VMware Workstation Pro)을 이용하여 두 대의 Ubuntu 환경을 구성하고, MongoDB Replica Set 환경을 구축하였다. 리더(Primary) 노드에는 Flask 웹 애플리케이션을 설치하여 사용자 및 회의실 데이터를 웹 페이지를

통해 조회할 수 있도록 구현하였으며, 해당 웹 서버는 Python 가상환경 내에서 개발 및 테스트가 이루어졌다. 데이터베이스는 MongoDB 6.0.24 버전을 사용하였으며, Primary–Secondary 구조의 복제 환경에서 실시간 데이터 복제와 리더 장애 시 자동 승격 테스트를 수행하였다. 웹 애플리케이션은 MongoDB와의 연결을 위해 PyMongo 라이브러리를 활용하였다.

< 설계 환경 요약표 >

운영체제	Windows 10 + Ubuntu 24.04.2 (VMware)
데이터베이스	MongoDB 6.0.24 (Primary노드 + Secondary노드 + Arbiter 노드)
웹 애플리케이션	Python 3.12.3 + Flask
라이브러리	PyMongo

2. 설계 내용

2.1 프로그램 구조 및 구성요소 설계

본 회의실 예약 시스템은 두 대의 Ubuntu 기반 가상머신에서 MongoDB Replica Set을 구성하고, 리더 노드(Primary)에는 Python 기반의 Flask 웹서버를 설치하여 웹 UI를 통해 사용자가 데이터베이스와 상호작용할 수 있도록 설계되었다.

2.1.1 MongoDB Replica Set 구성

Replica Set은 장애 허용성과 고가용성을 위한 MongoDB의 기본 복제 구성으로, 1개의 리더 노드와 1개의 팔로워 노드 그리고 리더 선출 투표를 위한 Arbiter 노드 등 3개의 노드로 구성하였다.

< MongoDB 노드 기능 요약표 >

노드 유형	기능
Leader (Primary)	클라이언트의 읽기/쓰기 요청을 직접 처리함
Follower (Secondary)	Primary로부터 데이터를 복제하며 자동 승격이 가능함
Arbiter	데이터를 저장하지 않고, 리더 선출 투표에만 참여함

Arbiter 노드의 기능은 다음과 같다 : 1) Arbiter 노드는 데이터를 저장하지 않으며, 2) Primary 노드가 장애로 내려갔을 때, Secondary와 함께 투표하여 정족수 조건을 만족시킨다. 따라서 2개의 데이터 노드 + 1개의 Arbiter으로 노드를 구성함으로써 자동 리더 승격(요구사항 5번)이 가능해진다.

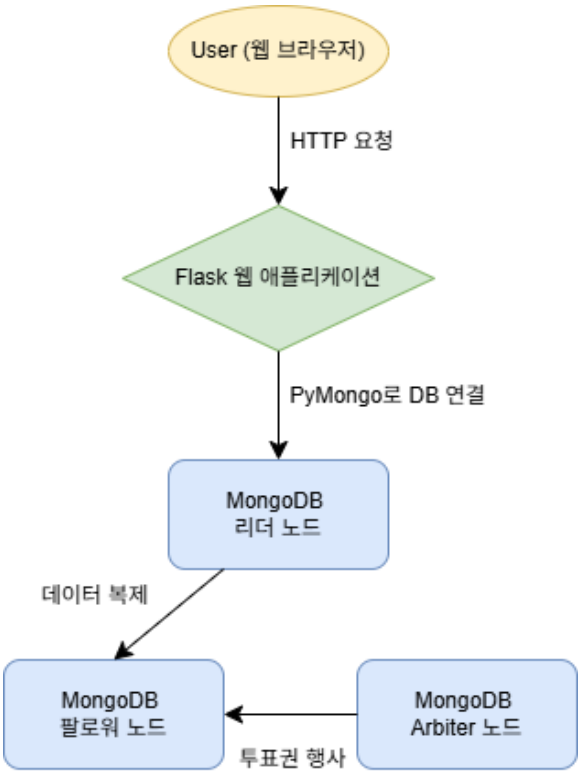
2.1.2 웹서버와의 상호작용

리더 노드에 설치된 Flask 웹서버는 MongoDB Primary에 직접 연결되어 아래 기능을 제공한다:

< MongoDB 테이블 경로에 따른 웹서버 기능 요약표 >

기능 경로	설명
/users	사용자 목록 출력 (users 컬렉션)
/rooms	회의실 목록 출력 (rooms 컬렉션)
/reservations	예약 목록 출력 (reservations 컬렉션)
/reserve	사용자 이름과 회의실 이름을 선택하여 예약 추가 기능. 예약 시에는 사용자가 선택한 이름(name) 을 기준으로, 서버는 MongoDB에서 대응되는 id를 조회하여 예약 정보를 저장함

2.1.3 프로그램 동작 흐름 요약

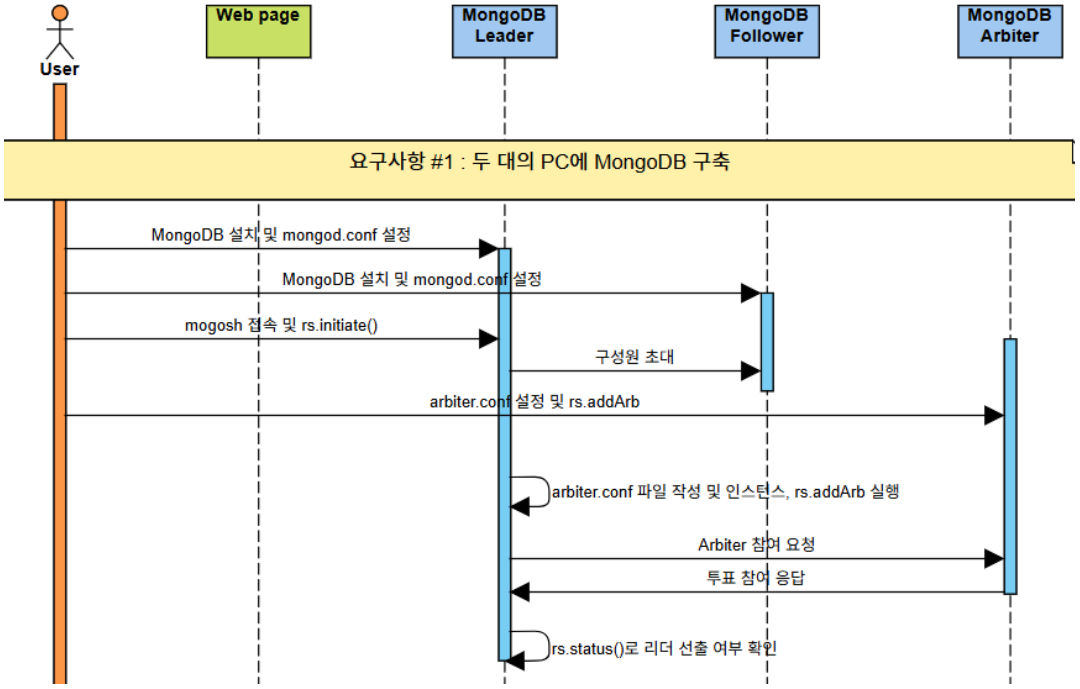


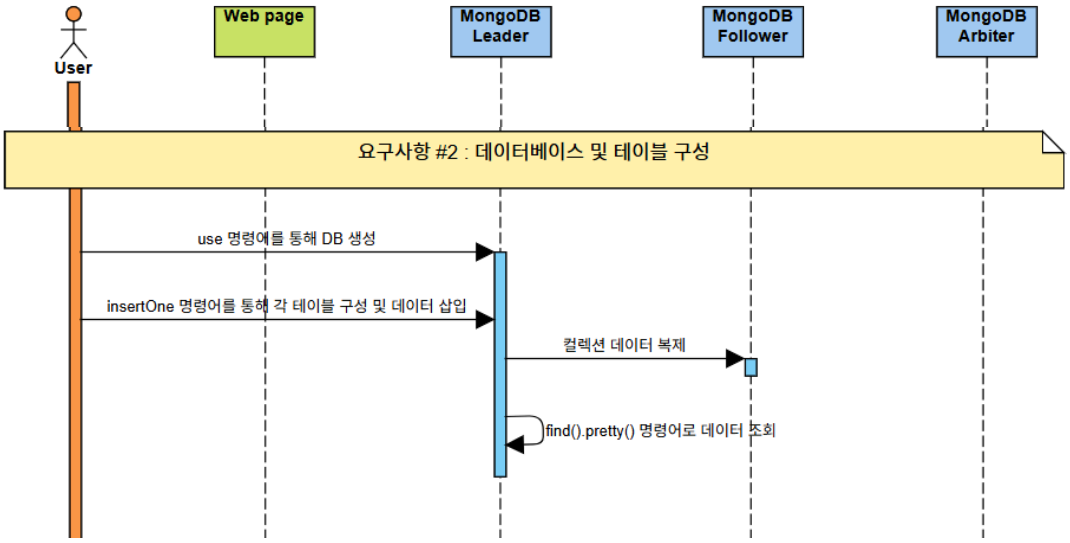
사용자와 웹 애플리케이션, MongoDB 노드들 사이의 상호작용 시나리오는 다음과 같다.

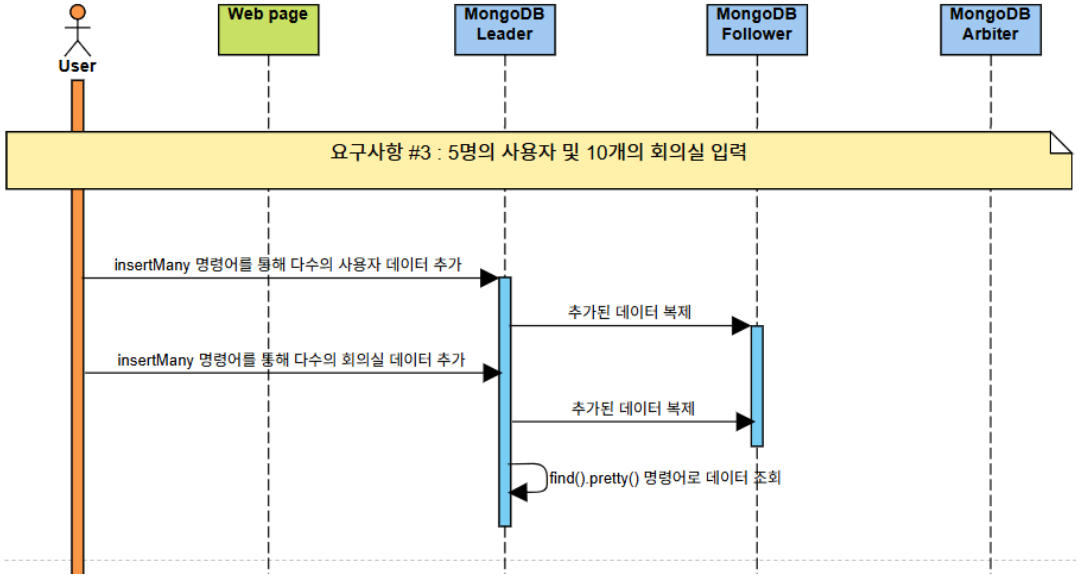
: Flask 웹서버는 항상 리더 노드에 연결되어 모든 DB 작업을 수행한다. 팔로워 노드는 실시간으로 리더 노드에서 데이터를 복제한다. Arbiter 노드는 리더 선출에만 참여하여 정족수를 유지한다. 따라서 리더 노드에 장애 발생 시 팔로워 + Arbiter가 투표하여 자동 승격이 발생하게 된다.

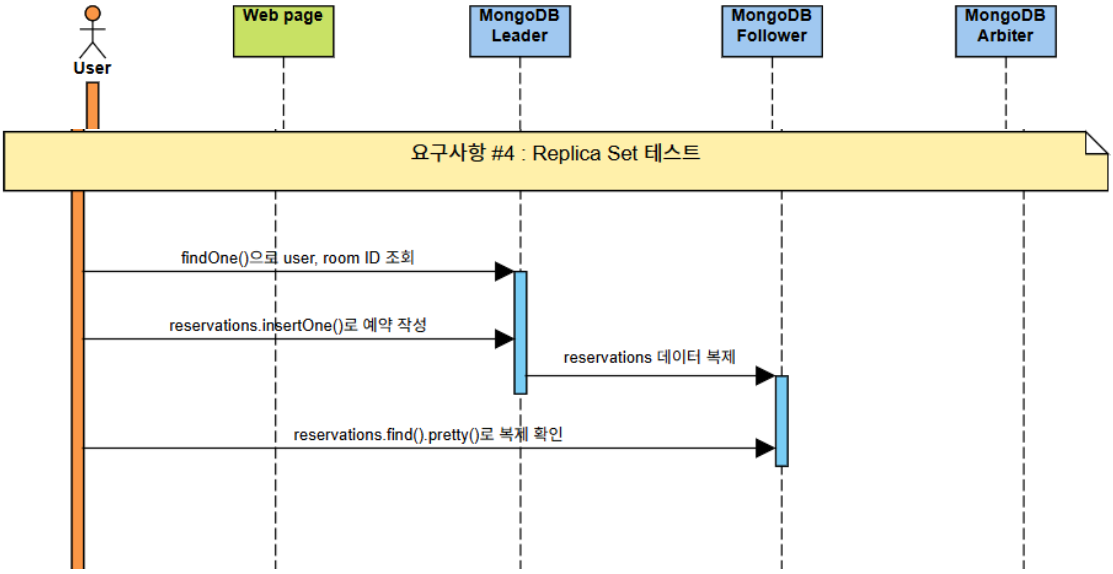
2.2 시스템 간 상호작용 및 기능 설계 (시퀀스 다이어그램)

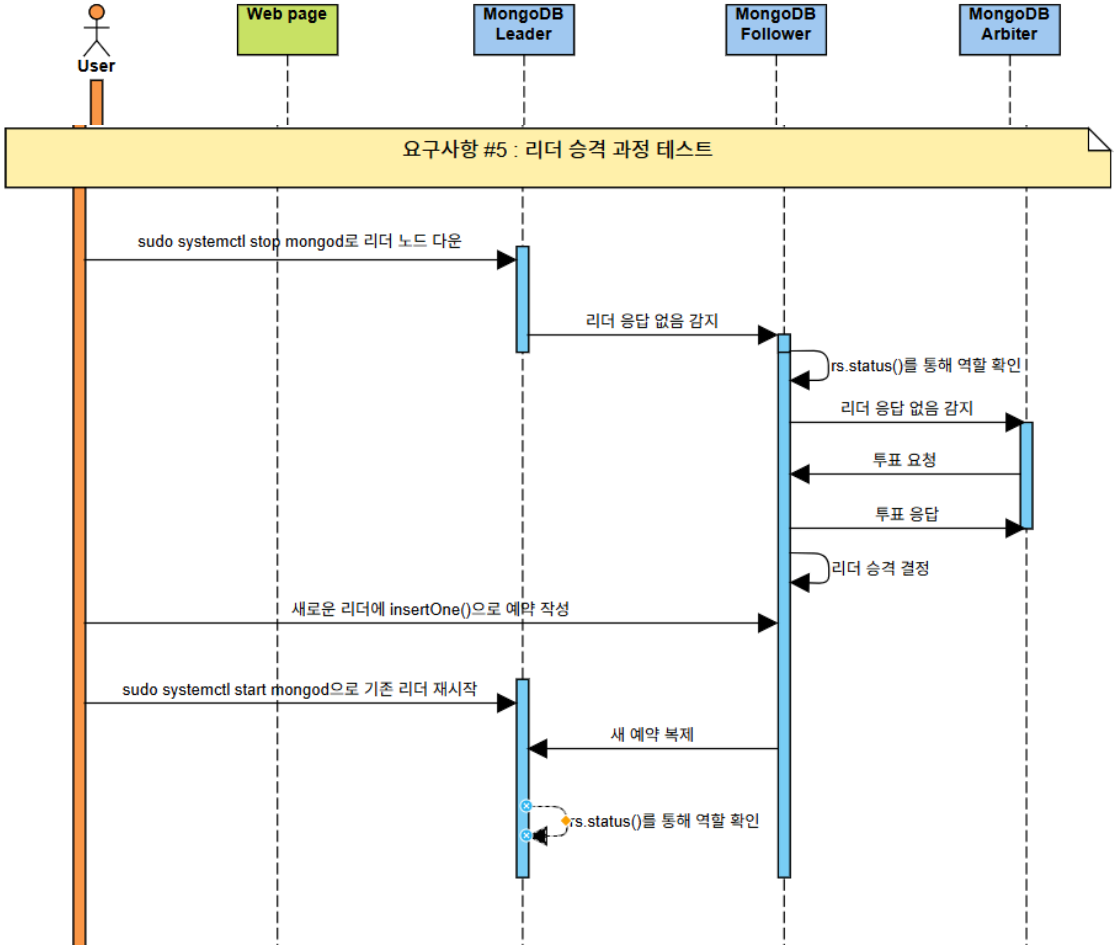
해당 회의실 예약 시스템의 객체는 세 대의 MongoDB 노드(Primary, Secondary, Arbiter)와 하나의 웹서버, 그리고 사용자(User)로 구성된다. 시스템은 MongoDB Replica Set 기반으로 운영되며, 웹 서버는 Flask 프레임워크를 기반으로 사용자 인터페이스를 제공한다. 각 6가지의 요구사항별로 객체 간의 시퀀스를 나타내는 시퀀스 다이어그램을 아래와 같이 구성하였다.

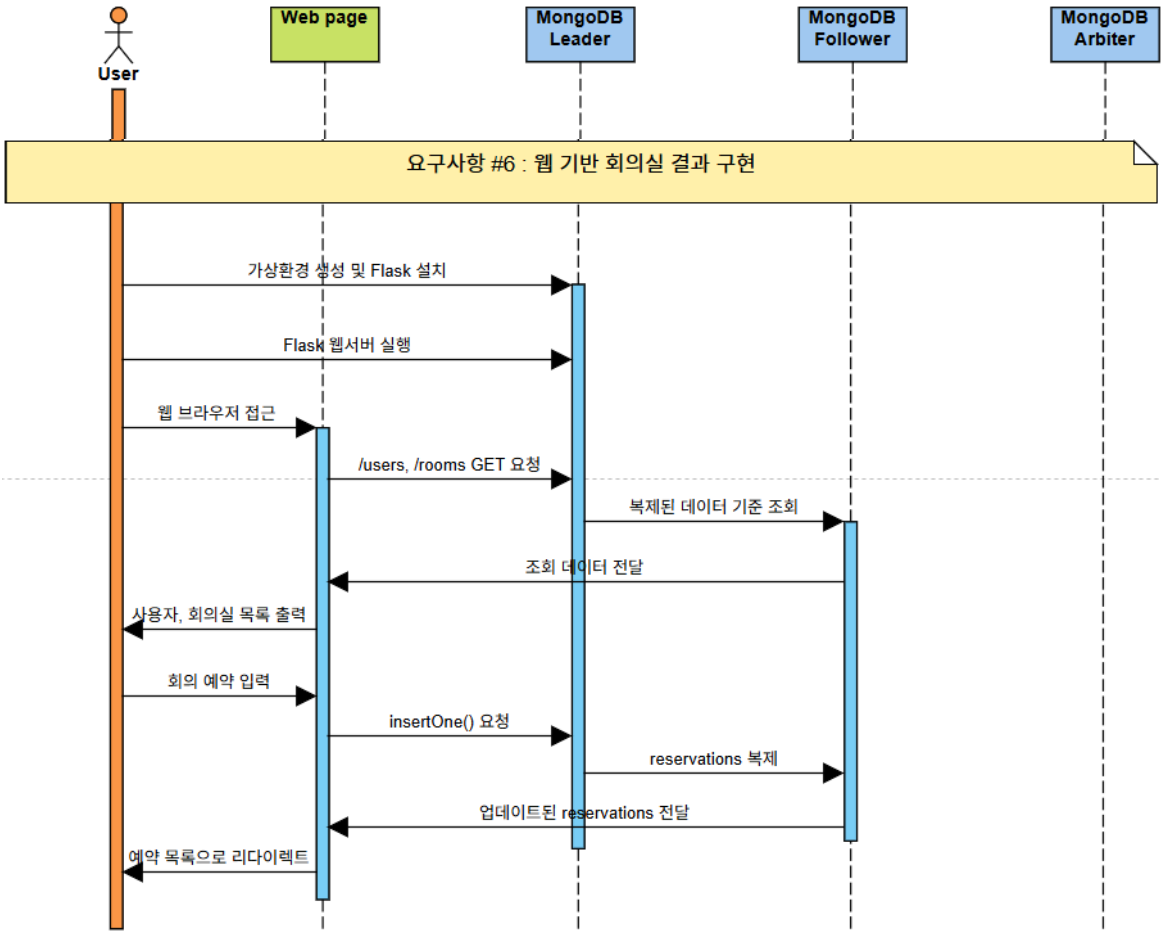

User
Web page
MongoDB Leader
MongoDB Follower
MongoDB Arbiter
요구사항 #1 : 두 대의 PC에 MongoDB 구축
MongoDB 설치 및 mongod.conf 설정
MongoDB 설치 및 mongod.conf 설정
mogosh 접속 및 rs.initiate()
구성원 초대
arbiter.conf 설정 및 rs.addArb
arbiter.conf 파일 작성 및 인스턴스, rs.addArb 실행
Arbiter 참여 요청
투표 참여 응답
rs.status()로 리더 선출 여부 확인


User
Web page
MongoDB Leader
MongoDB Follower
MongoDB Arbiter
요구사항 #2 : 데이터베이스 및 테이블 구성
use 명령어를 통해 DB 생성
insertOne 명령어를 통해 각 테이블 구성 및 데이터 삽입
컬렉션 데이터 복제
find().pretty() 명령어로 데이터 조회

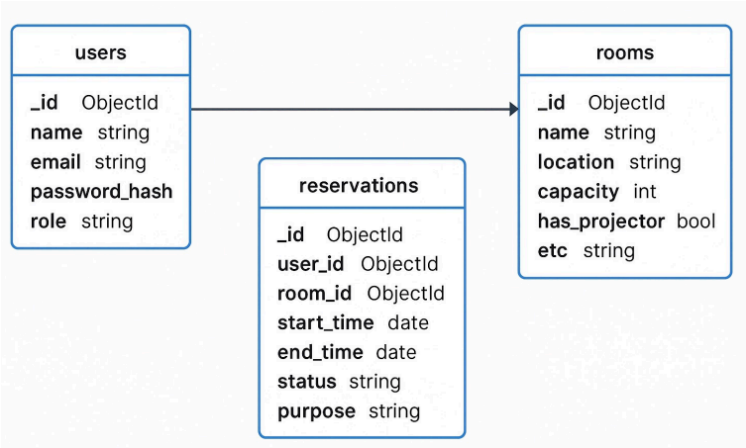

User
Web page
MongoDB Leader
MongoDB Follower
MongoDB Arbiter
요구사항 #3 : 5명의 사용자 및 10개의 회의실 입력
insertMany 명령어를 통해 다수의 사용자 데이터 추가
추가된 데이터 복제
insertMany 명령어를 통해 다수의 회의실 데이터 추가
추가된 데이터 복제
find().pretty() 명령어로 데이터 조회


User
Web page
MongoDB Leader
MongoDB Follower
MongoDB Arbiter
요구사항 #4 : Replica Set 테스트
findOne()으로 user, room ID 조회
reservations.insertOne()로 예약 작성
reservations 데이터 복제
reservations.find().pretty()로 복제 확인


User
Web page
MongoDB Leader
MongoDB Follower
MongoDB Arbiter
요구사항 #5 : 리더 승격 과정 테스트
sudo systemctl stop mongod로 리더 노드 다운
리더 응답 없음 감지
rs.status()를 통해 역할 확인
리더 응답 없음 감지
투표 요청
투표 응답
리더 승격 결정
새로운 리더에 insertOne()으로 예약 작성
sudo systemctl start mongod으로 기존 리더 재시작
새 예약 복제
rs.status()를 통해 역할 확인



2.3 데이터베이스 설계



데이터베이스는 총 3개의 테이블로 구성되며, 사용자, 회의실, 예약 데이터를 저장하고 관리하도록 설계되었다. Users 테이블은 사용자의 이름, 이메일, 비밀번호 해시값, 역할에 대한 정보를 저장한다. Rooms 테이블은 회의실의 이름, 위치, 수용인원, 프로젝터 유무, 비고에 대한 정보를 저장한다. Reservations

테이블은 Users 테이블과 Rooms 테이블과 ForeignKey로 연결되어 각각 사용자 id, 회의실 id를 참조한다. Reservations 테이블은 시작시간, 종료시간, 예약 상태, 목적에 대한 정보를 저장한다.

3. 요구사항에 따른 설계 세부사항

3.1 RQ 1 : 두 대의 PC에 MongoDB 구축

요구사항	설계 세부사항

RQ-1	**1) 모든 노드에 MongoDB 설치 및 실행**
	모든 노드에 대해 아래 명령어로 MongoDB 공식 키 및 소스 리스트를 등록한다.
	`wget-qO-https://www.mongodb.org/static/pgp/server-6.0.asc | sudo apt-key add -`
	`echo"deb[arch=amd64,arm64]https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/6.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list`
	모든 노드에 대해 아래 명령어로 패키지 업데이트 및 MongoDB를 설치한다.
	`sudo apt update`
	`sudo apt install -y mongodb-org`
	모든 노드에 대해 아래 명령어로 MongoDB를 실행한다.
	`sudo systemctl start mongod`
	`sudo systemctl enable mongod`
	2) 각 노드가 동일한 Replica Set으로 인식되도록 설정

모든 노드에 대해 Replica Set을 설정하기 위해 mongod.conf 파일을 아래와 같이 수정한다.

```
sudo nano /etc/mongod.conf
net:
  bindIp: 0.0.0.0
replication:
  replSetName: "rs0"
```

모든 노드에 대해 아래 명령어로 방화벽을 연다.

```
sudo ufw allow 27017
```

모든 노드에 대해 아래 명령어로 서비스를 재시작한다.

```
sudo systemctl restart mongod
```

3) Primary 노드에서 Replica Set 초기화 및 Secondary 노드 등록

Leader node에서 아래 명령어로 Mongosh를 실행한다.

```
mongosh
```

Leader node에서 아래 명령어로 멤버를 추가하여 Replica Set 초기화를 진행한다.

```
rs.initiate({
  _id: "rs0",
  members: [
    { _id: 0, host: "192.168.136.128:27017" },
    { _id: 1, host: "192.168.136.129:27017" }
  ]
})
```

4) Arbiter 노드를 기존 리더 노드에서 실행하고 설정

리더 장애 발생 시의 리더 승격을 정상적으로 가동하기 위한 Arbiter 노드를 설정하기 위해 아래의 단계를 수행한다.

리더 노드와 팔로워 노드에서 아래 명령어를 통해 arbiter용 디렉토리를 생성한다.

```
mkdir -p ~/arbiter_db
```

디렉토리에 아래 명령어를 통해 arbiter 설정 파일을 작성한다.

```
nano ~/arbiter.conf
```

그리고 해당 파일에 다음 내용을 붙여넣는다.

```
storage:
  dbPath: /home/jimin/arbiter_db

net:
  bindIp: 0.0.0.0
  port: 27018

replication:
  replSetName: rs0

systemLog:
  destination: file
  path: /home/jimin/arbiter_db/mongod.log
  logAppend: true

processManagement:
  fork: true
```

이후 arbiter mongod 인스턴스 실행을 위해 아래 명령어를 실행한다.

```
mongod -f ~/arbiter.conf
```

이후 리더 노드의 mongosh로 이동하여 arbiter를 추가한다.

```
mongosh
rs.addArb("192.168.163.128:27018")
```

5) Replica Set 구성 결과 및 리더 선출 여부 확인

	Leader node에서 아래 명령어로 Replica Set 구성 결과 및 리더 선출 여부를 확인한다. 또한 Arbiter가 정상적으로 추가되었는지도 확인한다. `rs.status()`

3.2 RQ 2 : DB 테이블 구성

요구사항	설계 세부사항

RQ-2	**1) 데이터 베이스 생성**
	리더 노드에서 아래 명령어로 meeting_reservation 데이터베이스를 생성한다.
	`use meeting_reservation`
	2) 컬렉션 구성 및 데이터 삽입
	리더 노드에서 아래 명령어로 users, rooms 컬렉션을 구성하고 데이터를
	삽입한다. 아래 구성된 데이터는 예시로 구성한 것이다.
	`db.users.insertOne({`
	`  name: "홍길동",`
	`  email: "hong@example.com",`
	`  password_hash: "hashed_pw",`
	`  role: "user"`
	`})`
	`db.rooms.insertOne({`
	`  name: "A101 회의실",`
	`  location: "2층",`
	`  capacity: 10,`
	`  has_projector: true,`
	`  etc: "화이트보드 있음"`
	`})`

users 컬렉션은 name, email, password_hash, role 등의 속성으로 구성하고,

rooms 컬렉션은 name, location, capacity, has_projector, etc 등의

속성으로 구성한다.

리더 노드에서 아래 명령어로 각 users와 rooms에서 첫 번째 문서를 가져온다.

```
const user = db.users.findOne({ email:
"hong@example.com" })
const room = db.rooms.findOne({ name: "A101 회의실" })
```

리더 노드에서 아래 명령어로 첫 번째 문서의 id를 적용한 reservations 컬렉션을

구성하고 예약 데이터를 등록한다. 아래 구성된 데이터는 예시로 구성한 것이다.

```
db.reservations.insertOne({
  user_id: user._id,
  room_id: room._id,
  start_time: ISODate("2025-05-30T09:00:00Z"),
  end_time: ISODate("2025-05-30T10:00:00Z"),
  status: "approved",  // or "pending", "cancelled"
  purpose: "주간 회의"
})
```

reservations 컬렉션은 가져온 id를 정의하여 user_id, room_id, start_time,

end_time, status, purpose 등의 속성으로 구성한다.

3) 구성된 컬렉션 조회

리더 노드에서 아래 명령어로 생성된 컬렉션들의 문서를 조회한다.

```
db.users.find().pretty()
db.rooms.find().pretty()
db.reservations.find().pretty()
```

3.3 RQ 3 : 5명의 사용자 및 10개의 회의실 입력

요구사항	설계 세부사항

RQ-3	**1) 컬렉션에 데이터 추가**
	리더 노드에서 아래 명령어를 통해 사용자와 회의실 데이터를 각각 추가 등록한다.
	
	아래 구성된 데이터는 예시로 구성한 것이다.
	
	`db.users.insertMany([`
	`  { name: "홍길동", email: "hong1@example.com", password_hash: "pw1", role: "user" },`
	`  { name: "김철수", email: "kim2@example.com", password_hash: "pw2", role: "user" },`
	`  { name: "이영희", email: "lee3@example.com", password_hash: "pw3", role: "admin" },`
	`  { name: "박민수", email: "park4@example.com", password_hash: "pw4", role: "user" },`
	`  { name: "최지우", email: "choi5@example.com", password_hash: "pw5", role: "user" }`
	`])`
	`db.rooms.insertMany([`
	`  { name: "A101", location: "1층", capacity: 10, has_projector: true, etc: "화이트보드" },`
	`  { name: "A102", location: "1층", capacity: 8, has_projector: false, etc: "" },`
	`  { name: "B201", location: "2층", capacity: 12, has_projector: true, etc: "빔 있음" },`
	`  { name: "B202", location: "2층", capacity: 6, has_projector: false, etc: "" },`
	`  { name: "C301", location: "3층", capacity: 20, has_projector: true, etc: "마이크 있음" },`

	{ name: "C302", location: "3층", capacity: 15, has_projector: true, etc: "" }, { name: "D401", location: "4층", capacity: 5, has_projector: false, etc: "" }, { name: "D402", location: "4층", capacity: 10, has_projector: true, etc: "커피머신 있음" }, { name: "E501", location: "5층", capacity: 25, has_projector: true, etc: "최대 수용" }, { name: "E502", location: "5층", capacity: 7, has_projector: false, etc: "" }]) **2) 추가 후 컬렉션의 데이터 개수 확인** 리더노드에서 아래 명령어를 통해 각 컬렉션에 추가된 데이터의 개수를 확인한다. db.users.countDocuments() db.rooms.countDocuments() **3) 컬렉션의 데이터 조회** 또한 find().pretty() 명령어를 통해 아래와 같이 각 컬렉션의 데이터를 조회한다. db.users.find().pretty() db.rooms.find().pretty() db.reservations.find().pretty()

3.4 RQ 4 : Replica Set 테스트

요구사항	설계 세부사항
RQ-4	**1) Leader 노드에서 새로운 예약 생성**

	두 대의 PC(가상 머신)에 구성된 MongoDB Replica Set에서 리더 노드에 데이터를 입력하고, 팔로워 노드에서 그 데이터를 읽어올 수 있는지 테스트하는 과제이다. 따라서 Primary 노드에서 예약을 1개 등록하고, Secondary 노드에서 그 데이터를 mongosh로 조회해 복제가 진행되었는지 확인한다.
	Primary 노드에서 아래 명령어를 통해 첫 문서를 가져온다.
	`db.users.findOne()`
	`db.rooms.findOne()`
	Primary 노드에서 아래 명령어를 통해 테스트 예약을 하나 생성한다.
	`db.reservations.insertOne({`
	`  user_id: user._id,`
	`  room_id: room._id,`
	`  start_time: ISODate("2025-06-09T09:00:00Z"),`
	`  end_time: ISODate("2025-06-09T10:00:00Z"),`
	`  status: "approved",`
	`  purpose: "복제셋 테스트 회의"`
	`})`
	2) Follower 노드에서 reservations 컬렉션 확인
	이후 Secondary 노드로 이동해서 아래 명령어를 통해 방금 Primary 노드에서 입력한 예약 기록이 보이는지 확인한다.
	`db.reservations.find().pretty()`

3.5 RQ 5 : 리더 승격 과정 테스트

요구사항	설계 세부사항

RQ-5	**1) Primary 노드 서비스 종료**
	
	기존 Primary 상태의 서버에서 아래 명령어로 mongod 서비스를 멈춰 리더 다운(Failover) 상태를 만든다.
	
	`sudo systemctl stop mongod`
	
	2) Secondary가 자동으로 새로운 Primary로 승되는지 확인
	
	이후 다른 Secondary 노드에서 아래 명령어로 새 리더 선출 과정을 모니터링한다.
	
	`rs.status()`
	
	3) 새로운 Primary에서 쓰기(예약 추가) 수행
	
	새로 선출된 리더 노드에서 아래 명령어를 통해 쓰기를 성공하면 리더 역할 정상 수행 중인 것이다.
	
	`use meeting_reservation`
	`const user = db.users.findOne()`
	`const room = db.rooms.findOne()\`
	`db.reservations.insertOne({`
	`  user_id: user._id,`
	`  room_id: room._id,`
	`  start_time: ISODate("2025-06-10T10:00:00Z"),`
	`  end_time: ISODate("2025-06-10T11:00:00Z"),`
	`  status: "approved",`
	`  purpose: "Failover 테스트 회의"`
	`})`
	
	4) 원래 Primary가 재시작 후 정상적으로 Replica에 합류하는지 확인
	
	이제 이전에 종료했던 원래 Primary 노드에서 아래 명령어를 통해 mongodb를 재시작한다.
	
	`sudo systemctl start mongod`

	아래 명령어를 통해 다시 Secondary로 합류했는지 확인하고 이전 리더 노드가 "SECONDARY"로 나타나면 정상 재합류 완료한 것이다. `rs.status()`

3.6 RQ 6 : 웹 기반 회의실 결과 구현

요구사항	설계 세부사항

RQ-6	**1) 리더 노드에 웹서버 환경 준비(Flask)**
	리더 노드에서 아래 명령어를 통해 웹 서버 환경을 준비한다.
	`sudo apt update`
	`sudo apt install python3-pip -y`
	`pip3 install flask pymongo`
	2) Flask 웹 서버 코드 작성
	nano 명령어를 활용하여 app_reserve.py라는 이름으로 아래 내용을 저장한다.
	`from flask import Flask, render_template`
	`from pymongo import MongoClient`
	`from flask import request, redirect`
	
	`app = Flask(__name__)`
	`client = MongoClient("mongodb://localhost:27017")`
	`db = client["meeting_reservation"]`
	
	`@app.route('/users')`
	`def show_users():`
	`users = list(db.users.find({}, {"_id": 0}))`
	`return render_template("users.html", users=users)`

```
@app.route('/rooms')
def show_rooms():
    rooms = list(db.rooms.find({}, {"_id": 0}))
    return render_template("rooms.html", rooms=rooms)

@app.route('/reserve', methods=["GET", "POST"])
def reserve():
    if request.method == "POST":
        user_name = request.form["user_name"]
        room_name = request.form["room_name"]

        # name을 기반으로 ObjectId 조회
        user = db.users.find_one({"name": user_name})
        room = db.rooms.find_one({"name": room_name})

        if not user or not room:
            return "사용자 또는 회의실을 찾을 수 없습니다.",
400

        reservation = {
            "user_id": user["_id"],
            "room_id": room["_id"],
            "start_time": request.form["start_time"],
            "end_time": request.form["end_time"],
            "status": "pending",
            "purpose": request.form["purpose"]
        }
        db.reservations.insert_one(reservation)
        return redirect("/reservations")
```

```
    # GET 요청 시 사용자와 회의실 정보 전달
    users = list(db.users.find({}, {"_id": 0, "name": 1}))
    rooms = list(db.rooms.find({}, {"_id": 0, "name": 1}))
    return render_template("reserve.html", users=users, rooms=rooms)
```

리더 노드의 아래 명령을 실행해 templates 디렉토리를 만든다.

```
mkdir -p ~/myweb/templates
```

이 디렉토리에 웹 구성을 위한 HTML 템플릿도 아래와 같이 4종 생성한다.

```
nano templates/users.html
nano templates/rooms.html
nano templates/reserve.html
nano templates/reservations.html
```

nano 명령어를 통해 users.html를 아래와 같이 작성한다.

```
<h2>사용자 목록</h2>
<table border="1">
  <tr>
    <th>이름</th><th>이메일</th><th>역할</th>
  </tr>
  {% for user in users %}
  <tr>
    <td>{{ user.name }}</td>
    <td>{{ user.email }}</td>
    <td>{{ user.role }}</td>
  </tr>
  {% endfor %}
</table>
```

nano 명령어를 통해 rooms.html를 아래와 같이 작성한다.

```
<h2>회의실 목록</h2>
<table border="1">
  <tr>
    <th>이름</th><th>위치</th><th>수용인원</th>
  </tr>
  {% for room in rooms %}
  <tr>
    <td>{{ room.name }}</td>
    <td>{{ room.location }}</td>
    <td>{{ room.capacity }}</td>
  </tr>
  {% endfor %}
</table>
```

nano 명령어를 통해 reserve.html를 아래와 같이 작성한다.

```
<h2>회의실 예약하기</h2>
<form method="POST">
  <label>사용자:
    <select name="user_name">
      {% for user in users %}
        <option value="{{ user.name }}">{{ user.name
}}</option>
      {% endfor %}
    </select>
  </label><br><br>

  <label>회의실:
    <select name="room_name">
      {% for room in rooms %}
        <option value="{{ room.name }}">{{ room.name
}}</option>
      {% endfor %}
```

```
    </select>
  </label><br><br>

  <label>시작 시간:
  <input type="datetime-local" name="start_time"
step="60" required>
</label><br><br>

  <label>종료 시간:
  <input type="datetime-local" name="end_time"
step="60" required>
</label><br><br>

  <label>목적: <input type="text"
name="purpose"></label><br><br>

  <button type="submit">예약하기</button>
</form>
```

nano 명령어를 통해 reservations.html를 아래와 같이 작성한다.

```
<h2>예약 목록</h2>
<table border="1">
  <tr>
    <th>사용자 ID</th><th>회의실
ID</th><th>시작</th><th>종료</th><th>상태</th><th>목적</th>
  </tr>
  {% for r in reservations %}
  <tr>
    <td>{{ r.user_id }}</td>
    <td>{{ r.room_id }}</td>
    <td>{{ r.start_time }}</td>
    <td>{{ r.end_time }}</td>
```

	<td>{{ r.status }}</td> <td>{{ r.purpose }}</td> </tr> {% endfor %} </table> **3) 서버 실행** 리더 노드에서 아래 명령어로 서버를 실행한다. python3 app_reserve.py **4) 브라우저로 접속하여 사용자 + 회의실 정보 확인** 이제 아래에 출력되는 주소로 웹 브라우저에 접속하여 사용자와 회의실 목록이 출력되는 것을 확인한다.

4. 요구사항 추적표

요구사항	요구사항 설명	설계 및 구현 내용
RQ-1	두 대의 PC에 MongoDB 구축	• 모든 노드에 MongoDB 설치 및 실행한다. • 각 노드가 동일한 Replica Set으로 인식되도록 설정한다. • Primary 노드에서 Replica Set 초기화 및 Secondary 노드를 등록한다. • Arbiter 노드를 기존 리더 노드에서 실행하고 설정한다. • Replica Set 구성 결과 및 리더 선출 여부를 확인한다.

RQ-2	DB 테이블 구성	• use 명령어로 데이터 베이스를 생성한다. • insertOne 명령어로 컬렉션 구성 및 데이터를 삽입한다. • find().pretty()명령어로 구성된 컬렉션을 조회한다.
RQ-3	5명의 사용자 및 10개의 회의실 입력	• insertMany 명령어를 통해 컬렉션에 데이터를 추가한다. • countDocuments 명령어를 통해 컬렉션의 데이터 개수를 확인한다. • find().pretty 명령어를 통해 컬렉션의 데이터를 조회한다.
RQ-4	Replica Set 테스트	• Leader 노드에서 insertOne() 명령어를 통해 Leader 노드에서 새로운 예약을 생성한다. • Follower 노드에서 find().pretty() 명령어를 통해 reservations 컬렉션을 확인한다.
RQ-5	리더 승격 과정 테스트	• Primary 노드 서비스를 종료한다. • Secondary가 자동으로 새로운 Primary로 승격되는지 확인한다. • 새로운 Primary에서 쓰기(예약 추가)를 수행한다. • 원래 Primary가 재시작 후 정상적으로 Replica에 합류하는지 확인한다.
RQ-6	웹 기반 회의실 결과 구현	• 리더 노드에 웹서버 환경을 준비(Flask)한다. • app_reserve.py, HTML 템플릿 등 Flask 웹 서버 코드를 작성한다. • py 파일을 통해 서버를 실행한다. • 브라우저로 접속하여 사용자 + 회의실 정보를 확인한다.