

소프트웨어 테스트 결과서

(Software Test Result)

: 분산 DB 기반의 회의실 예약 시스템



분산처리 기초

2025.06.13

데이터사이언스학과

12211262

우지민

[목 차]

1. 개요

1.1. 시험 목적

1.2. 시험 환경

2. 시험 내용

2.1. 기능 요구사항에 따른 시험 방법

3. 요구사항에 따른 설계 세부사항

3.1. RQ 1 : 두 대의 PC에 MongoDB 구축

3.2. RQ 2 : DB 테이블 구성

3.3. RQ 3 : 사용자 및 회의실 입력

3.4. RQ 4 : Replica Set 테스트

3.5. RQ 5 : 리더 승격 과정 테스트

3.6. RQ 6 : 웹 기반 회의실 결과 구현

1. 개요

1.1 시험 목적

본 시험의 목적은 MongoDB 환경을 구성하고 데이터베이스를 구축하여 회의실 예약 시스템의 기능을 갖춘 웹페이지의 구현 결과가 요청된 요구사항을 충족하는지를 확인하고 설계 시 예상된 기능이 올바르게 작동되는지 검증하기 위함이다. 시험은 2.1절 ‘기능 요구사항에 따른 시험 방법’을 바탕으로 진행하며, 각 기능이 요청된 요구사항을 잘 만족하고 문제가 발생하지 않는지 확인하는데 중점을 둔다.

1.2 시험 환경

회의실 예약 시스템은 Windows 10 운영체제에서 VMware Workstation Pro를 활용하여 구성한 2개의 Ubuntu 24.04.2 가상 머신 환경에서 구현되었다. 웹 프레임워크는 Python 기반의 Flask를 사용하였으며, 데이터베이스는 MongoDB를 Ubuntu 환경에 직접 설치하여 서버-클라이언트 구조로 구성하였다. 시험은 로컬 가상머신 환경에서 실행된 Flask 개발 서버(app_reservation.py)를 통해 수행되었으며, 서버 실행 후 `http://192.168.x.x:5000/`의 형태로 브라우저에서 접근하여 기능별 동작을 확인하였다. 해당 URL은 가상머신의 내부 IP 주소를 사용하여 동일 네트워크 상에서 접근 가능하도록 설정하였다. 또한, 리더-팔로워 구조의 MongoDB Replica Set 환경을 구축하여, 리더 노드에서 회의실 및 사용자 등록을 수행하고, 팔로워 노드에서 해당 데이터가 정상 복제되어 조회 가능한지를 검증하였다. 웹 테스트 항목으로는 사용자 목록 출력, 회의실 목록 출력, 예약 등록 기능, 등록된 예약 목록 확인, 리더 노드 장애 발생 후 팔로워 승격 테스트 등이 포함되었다.

2. 시험 내용

2.1 기능 요구사항에 따른 시험 방법

요구사항	요구사항 설명	시험 방법
RQ-1	두 대의 PC에 MongoDB 구축	<ul style="list-style-type: none">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	<ul style="list-style-type: none">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	<ul style="list-style-type: none">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	<ul style="list-style-type: none">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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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.

3. 세부 시험 결과

3.1 RQ 1 : 두 대의 PC에 MongoDB 구축

요구사항	세부 시험 절차
RQ-1	<ol style="list-style-type: none"> 1. 모든 노드에 MongoDB 설치 및 실행한다. 2. 각 노드가 동일한 Replica Set으로 인식되도록 설정한다. 3. Primary 노드에서 Replica Set 초기화 및 Secondary 노드를 등록한다. 4. Arbiter 노드를 기존 리더 노드에서 실행하고 설정한다. 5. Replica Set 구성 결과 및 리더 선출 여부를 확인한다.

1) 모든 노드에 MongoDB 설치 및 실행한다.

모든 노드에 대해 MongoDB 공식 키 및 소스 리스트를 등록하고 나서 패키지 업데이트 및

MongoDB를 설치하고 실행한다. 아래 명령어를 활용한다.

```
sudo apt update
sudo apt install -y mongodb-org
sudo systemctl start mongod
sudo systemctl enable mongod
```

2) 각 노드가 동일한 Replica Set으로 인식되도록 설정한다.

모든 노드에 대해 Replica Set을 설정하기 위해 `sudo nano /etc/mongod.conf`를 통해 `mongod.conf` 파일을 아래 <그림1>과 같이 수정하여 `net` 항목과 `replication` 항목의 내용을 업데이트한다.

<그림1> mongod.conf 파일 설정 화면

```
# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0

# how the process runs
processManagement:
  timeZoneInfo: /usr/share/zoneinfo

#security:

#operationProfiling:

#replication:
replication:
  replSetName: "rs0"
#sharding:
```

이후 모든 노드에 대해 방화벽을 열고 서비스를 재시작한다(`sudo ufw allow 27017`).

3) Primary 노드에서 Replica Set 초기화 및 Secondary 노드를 등록한다.

이후 리더 노드에서 `mongosh`를 실행하고 아래 <그림2>와 같이 멤버를 추가하여 Replica Set 초기화를 진행한다.

<그림2> 멤버를 추가 후 Replica Set 초기화 화면

```
rs0 [direct: primary] test> 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 설정한다.

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 생성하고(`mkdir -p ~/arbiter_db`) arbiter 설정 파일을 작성한다(`nano ~/arbiter.conf`). `arbiter.conf` 파일은 <그림3>과 같이 구성한다.

<그림3> arbiter.conf 파일 구성 화면

```
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 실행하고(`mongod -f ~/arbiter.conf`) 리더 노드의 mongosh로 이동하여 arbiter를 추가한다(`rs.addArb("192.168.163.128:27018")`).

5) Replica Set 구성 결과 및 리더 선출 여부를 확인한다.

Leader node에서 아래 명령어로 Replica Set 구성 결과 및 리더 선출 여부를 확인한다(`rs.status()`). 아래 <그림4>와 같이 화면이 출력되면 잘 설정된 것이다.

<그림4> `rs.status()` 후 출력 화면

```
members: [
  {
    _id: 0,
    name: '192.168.163.128:27017',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 6429,
    optime: { ts: Timestamp({ t: 1750057139, i: 1 }), t: Long('5') },
    optimeDate: ISODate('2025-06-16T06:58:59.000Z'),
    lastAppliedWallTime: ISODate('2025-06-16T06:58:59.819Z'),
    lastDurableWallTime: ISODate('2025-06-16T06:58:59.819Z'),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1750050824, i: 1 }),
    electionDate: ISODate('2025-06-16T05:13:44.000Z'),
    configVersion: 2,
    configTerm: 5,
```

```
{
  _id: 1,
  name: '192.168.163.129:27017',
  health: 1,
  state: 2,
  stateStr: 'SECONDARY',
  uptime: 3593,
  optime: { ts: Timestamp({ t: 1750057139, i: 1 }), t: Long('5') },
  optimeDurable: { ts: Timestamp({ t: 1750057139, i: 1 }), t: Long('5') },
  optimeDate: ISODate('2025-06-16T06:58:59.000Z'),
  optimeDurableDate: ISODate('2025-06-16T06:58:59.000Z'),
  lastAppliedWallTime: ISODate('2025-06-16T06:58:59.819Z'),
  lastDurableWallTime: ISODate('2025-06-16T06:58:59.819Z'),
  lastHeartbeat: ISODate('2025-06-16T06:59:06.812Z'),
  lastHeartbeatRecv: ISODate('2025-06-16T06:59:05.765Z'),
  pingMs: Long('0'),
  lastHeartbeatMessage: '',
  syncSourceHost: '192.168.163.128:27017',
  syncSourceId: 0,
  infoMessage: '',
  configVersion: 2,
  configTerm: 5
},
```

또한 아래 <그림5>와 같이 출력된다면 Arbiter 노드도 정상적으로 추가된 것이다.

<그림 5> rs.status() 후 Arbiter 노드 출력 화면

```
{
  _id: 2,
  name: '192.168.163.128:27018',
  health: 1,
  state: 7,
  stateStr: 'ARBITER',
  uptime: 6428
```

3.2 RQ 2 : DB 테이블 구성

요구사항	세부 시험 절차
RQ-2	<ol style="list-style-type: none"> 1. use 명령어로 데이터 베이스를 생성한다. 2. insertOne 명령어로 컬렉션 구성 및 데이터를 삽입한다. 3. find().pretty()명령어로 구성된 컬렉션을 조회한다.

1) use 명령어로 데이터 베이스를 생성한다.

리더 노드에서 meeting_reservation 데이터베이스를 생성한다(use meeting_reservation).

2) insertOne 명령어로 컬렉션 구성 및 데이터를 삽입한다.

리더 노드에서 users, rooms 컬렉션을 구성하고 데이터를 삽입한다. 아래의 <그림 6>과 같이 구성하여 컬렉션을 생성한다.

<그림 6> users, rooms 컬렉션 구성 및 데이터 삽입 화면

```
rs0 [direct: primary] test> use meeting_reservation
switched to db meeting_reservation
rs0 [direct: primary] meeting_reservation> 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"
... })
...
{
  acknowledged: true,
  insertedId: ObjectId('684fc5fa2fed228aea69e329')
}
```

이후 reservations 컬렉션 구성을 위해 리더 노드에서 아래 명령어로 각 users와 rooms에서 첫 번째 문서를 가져온다. 리더 노드에서 아래 <그림 7>과 같이 첫 번째 문서의 id를 적용한 reservations 컬렉션을 구성하고 예약 데이터를 등록한다.

<그림 7> reservations 컬렉션 구성 화면

```
rs0 [direct: primary] meeting_reservation> 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"
... })
...
{
  acknowledged: true,
  insertedId: ObjectId('684fc6fc2fed228aea69e32a')
}
```

3) find().pretty()명령어로 구성된 컬렉션을 조회한다.

각 테이블별로 find().pretty() 명령어를 통해 아래 <그림 8>와 같이 각 컬렉션 별로 어떤

데이터가 있는지 조회할 수 있다. 아래와 같이 조회한다.

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

<그림 8> 컬렉션 조회 화면

```
{  
  _id: ObjectId('684fc5fa2fed228aea69e328'),  
  name: '홍길동',  
  email: 'hong@example.com',  
  password_hash: 'hashed_pw',  
  role: 'user'  
}  
]  
rs0 [direct: primary] meeting_reservation> db.rooms.find().pretty()  
[  
  {  
    _id: ObjectId('684fc5fa2fed228aea69e329'),  
    name: 'A101 회의실',  
    location: '2층',  
    capacity: 10,  
    has_projector: true,  
    etc: '화이트보드 있음'  
  }  
]  
rs0 [direct: primary] meeting_reservation> db.reservations.find().pretty()  
[  
  {  
    _id: ObjectId('684fc6fc2fed228aea69e32a'),  
    user_id: ObjectId('684fc5fa2fed228aea69e328'),  
    room_id: ObjectId('684fc5fa2fed228aea69e329'),  
    start_time: ISODate('2025-05-30T09:00:00.000Z'),  
    end_time: ISODate('2025-05-30T10:00:00.000Z'),  
    status: 'approved',  
    purpose: '주간 회의'
```

3.3 RQ 3 : 사용자 및 회의실 입력

요구사항	세부 시험 절차
RQ-3	<ol style="list-style-type: none"> 1. insertMany 명령어를 통해 컬렉션에 데이터를 추가한다. 2. countDocuments 명령어를 통해 컬렉션의 데이터 개수를 확인한다. 3. find().pretty 명령어를 통해 컬렉션의 데이터를 조회한다.

1) insertMany 명령어를 통해 컬렉션에 데이터를 추가한다.

리더 노드에서 아래 <그림 9>와 같이 사용자와 회의실 데이터를 각각 추가 등록한다.

<그림 9> 사용자 및 회의실 데이터 등록 화면

```
rs0 [direct: primary] meeting_reservation> db.users.insertMany([
rs0 [direct: primary] meeting_reservation> db.rooms.insertMany([
...   { name: "A101", location: "1층 ", capacity: 10, has_projector: true, etc: "
화이트보드 있음" },
...   { name: "A102", location: "1층 ", capacity: 8, has_projector: false, etc: "
"},
...   { name: "B201", location: "2층 ", capacity: 12, has_projector: true, etc: "
빔프로젝터" },
...   { 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 회의실" },
...   { name: "E502", location: "5층 ", capacity: 7, has_projector: false, etc: "
"}
])
```

2) countDocuments 명령어를 통해 컬렉션의 데이터 개수를 확인한다.

리더노드에서 아래 <그림 10>과 같이 각 컬렉션에 추가된 데이터의 개수를 확인한다.

```
db.users.countDocuments()
```

```
db.rooms.countDocuments()
```

<그림 10> 추가된 데이터의 개수 확인 화면


```
rs0 [direct: primary] meeting_reservation> db.users.countDocuments()
5
rs0 [direct: primary] meeting_reservation> db.rooms.countDocuments()
10
```

3) find().pretty 명령어를 통해 컬렉션의 데이터를 조회한다.

아래 <그림 11>과 같이 각 컬렉션의 데이터를 조회결과를 확인할 수 있다.

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

<그림 11> 데이터 추가 조회결과 화면



```
rs0 [direct: primary] meeting_reservation> db.users.find().pretty()
[
  {
    _id: ObjectId('684fc5fa2fed228aea69e328'),
    name: '홍길동',
    email: 'hong@example.com',
    password_hash: 'hashed_pw',
    role: 'user'
  },
  {
    _id: ObjectId('684fc9342fed228aea69e32b'),
    name: '김철수',
    email: 'kim2@example.com',
    password_hash: 'pw2',
    role: 'user'
  },
  {
    _id: ObjectId('684fc9342fed228aea69e32c'),
    name: '이영희',
    email: 'lee3@example.com',
    password_hash: 'pw3',
    role: 'admin'
  },
]
```

3.4 RQ 4 : Replica Set 테스트

요구사항	세부 시험 절차
RQ-4	<ol style="list-style-type: none">1. Leader 노드에서 insertOne() 명령어를 통해 Leader 노드에서 새로운 예약을 생성한다.2. Follower 노드에서 find().pretty() 명령어를 통해 reservations 컬렉션을 확인한다.

1) Leader 노드에서 insertOne() 명령어를 통해 Leader 노드에서 새로운 예약을 생성한다.

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. 먼저 Primary 노드에서 아래 명령어를 통해 첫 문서를 가져온다(`db.users.findOne()`, `db.rooms.findOne()`). 아래 <그림 12>와 같이 구성한다.

<그림 12> 리더 노드 새로운 예약 생성 화면

```
rs0 [direct: primary] meeting_reservation> const user = db.users.findOne()
rs0 [direct: primary] meeting_reservation> const room = db.rooms.findOne()
rs0 [direct: primary] meeting_reservation> 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: "복 제 셋 테스트 회의"
... })
{
  acknowledged: true,
  insertedId: ObjectId('68457794251c55aa1169e339')
}
```

2) Follower 노드에서 `find().pretty()` 명령어를 통해 reservations 컬렉션을 확인한다.

이후 Secondary 노드로 이동해서 아래 <그림 13>과 같이 방금 Primary 노드에서 입력한 예약 기록이 보이는지 확인할 수 있다(`db.reservations.find().pretty()`).

<그림 13> 팔로워 노드에서 예약 복제 확인 화면

```
rs0 [direct: secondary] meeting_reservation> db.reservations.find().pretty()
[
  {
    _id: ObjectId('6845740b251c55aa1169e32a'),
    user_id: ObjectId('6845731b251c55aa1169e328'),
    room_id: ObjectId('6845735a251c55aa1169e329'),
    start_time: ISODate('2025-05-30T09:00:00.000Z'),
    end_time: ISODate('2025-05-30T10:00:00.000Z'),
    status: 'approved',
    purpose: '주 간 회의'
  },
  {
    _id: ObjectId('68457794251c55aa1169e339'),
    user_id: ObjectId('6845731b251c55aa1169e328'),
    room_id: ObjectId('6845767b251c55aa1169e32f'),
    start_time: ISODate('2025-06-09T09:00:00.000Z'),
    end_time: ISODate('2025-06-09T10:00:00.000Z'),
    status: 'approved',
    purpose: '복 제 셋 테스트 회의'
  }
]
```

3.5 RQ 5 : 리더 승격 과정 테스트

요구사항	세부 시험 절차
RQ-5	<ol style="list-style-type: none">1. Primary 노드 서비스를 종료한다.2. Secondary가 자동으로 새로운 Primary로 승격되는지 확인한다.3. 새로운 Primary에서 쓰기(예약 추가)를 수행한다.4. 원래 Primary가 재시작 후 정상적으로 Replica에 합류하는지 확인한다.

1) Primary 노드 서비스를 종료한다.

기존 Primary 상태의 서버에서 아래 명령어로 mongod 서비스를 멈춰 리더 다운(Failover) 상태를 만든다. 아래 <그림 14>와 같이 서비스를 멈춘다(`sudo systemctl stop mongod`).

<그림 14> 기존 리더 mongod 서비스 중단 화면

```
jimin@jimin-VMware-Virtual-Platform:~$ sudo systemctl stop mongod
[sudo] jimin 암호 :
```

2) Secondary가 자동으로 새로운 Primary로 승격되는지 확인한다.

이후 다른 Secondary 노드에서 아래 <그림 15>와 <그림 16>과 같이 기존의 팔로워 노드로 리더(PRIMARY)가 바뀌는 상황을 모니터링한다(`rs.status()`).

<그림 15> mongod 서비스 중단 전 rs.status() 화면 : 원래 status

```
members: [
  {
    _id: 0,
    name: '192.168.163.128:27017',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 10186,
    optime: { ts: Timestamp({ t: 1750060901, i: 1 }), t: Long('5') },
    optimeDate: ISODate('2025-06-16T08:01:41.000Z'),
    lastAppliedWallTime: ISODate('2025-06-16T08:01:41.750Z'),
    lastDurableWallTime: ISODate('2025-06-16T08:01:41.750Z'),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1750050824, i: 1 }),
    electionDate: ISODate('2025-06-16T05:13:44.000Z'),
    configVersion: 2,
    configTerm: 5,
    self: true,
    lastHeartbeatMessage: ''
  },
  {
    _id: 1,
    name: '192.168.163.129:27017',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 7350,
    optime: { ts: Timestamp({ t: 1750060901, i: 1 }), t: Long('5') },
```

<그림 16> mongod 서비스 중단 후 rs.status() 화면 : 리더 승격 완료

```
members: [
  {
    _id: 0,
    name: '192.168.163.128:27017',
    health: 0,
    state: 8,
    stateStr: '(not reachable/healthy)',
    uptime: 0,
    optime: { ts: Timestamp({ t: 0, i: 0 }), t: Long('-1') },
    optimeDurable: { ts: Timestamp({ t: 0, i: 0 }), t: Long('-1') },
    optimeDate: ISODate('1970-01-01T00:00:00.000Z'),
    optimeDurableDate: ISODate('1970-01-01T00:00:00.000Z'),
    lastAppliedWallTime: ISODate('2025-06-16T08:04:30.126Z'),
    lastDurableWallTime: ISODate('2025-06-16T08:04:30.126Z'),
    lastHeartbeat: ISODate('2025-06-16T08:04:48.166Z'),
    lastHeartbeatRecv: ISODate('2025-06-16T08:04:33.657Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: 'Error connecting to 192.168.163.128:27017',
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    configVersion: 2,
    configTerm: 7
  },
  {
    _id: 1,
    name: '192.168.163.129:27017',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 10233,
  }
]
```

<그림 16>을 보면, 기존 리더 노드의 상태는 '(not reachable/health)'로 표시되며, 기존 팔로워 노드의 상태는 'PRIMARY' 리더 승격이 정상적으로 일어났음을 확인할 수 있다.

3) 새로운 Primary에서 쓰기(예약 추가)를 수행한다.

새로 선출된 리더 노드에서 아래 <그림 17>과 같이 쓰기를 수행하고 find().pretty()로 조회가 가능하면 쓰기가 성공한 것이므로 새 리더 노드가 리더 역할을 정상적으로 수행 중인 것이다.

<그림 17> 새로운 리더에서 새 예약 확인 화면

```
rs0 [direct: primary] meeting_reservation> db.reservations.find().pretty()
[
  {
    _id: ObjectId('684fc6fc2fed228aea69e32a'),
    user_id: ObjectId('684fc5fa2fed228aea69e328'),
    room_id: ObjectId('684fc5fa2fed228aea69e329'),
    start_time: ISODate('2025-05-30T09:00:00.000Z'),
    end_time: ISODate('2025-05-30T10:00:00.000Z'),
    status: 'approved',
    purpose: '주간 회의'
  },
  {
    _id: ObjectId('684fd10b57e7ae478e69e328'),
    user_id: ObjectId('684fc5fa2fed228aea69e328'),
    room_id: ObjectId('684fc5fa2fed228aea69e329'),
    start_time: ISODate('2025-06-10T10:00:00.000Z'),
    end_time: ISODate('2025-06-10T11:00:00.000Z'),
    status: 'approved',
    purpose: 'Failover 테스트 회의'
  }
]
```

4) 원래 Primary가 재시작 후 정상적으로 Replica에 합류하는지 확인한다.

이제 이전에 종료했던 원래 Primary 노드에서 아래 명령어를 통해 mongodb를 재시작한다(`sudo systemctl start mongod`). 아래 <그림 18>과 같이 기존 리더 노드가 다시 Secondary로 합류했는지 확인한다(`rs.status()`). 이전 리더 노드가 "SECONDARY"로 나타나면 정상 재합류 완료한 것이다.

<그림 18> 이전 리더 노드의 "SECONDARY" 상태 확인 : 노드 복귀 완료

```
members: [
  {
    _id: 0,
    name: '192.168.163.128:27017',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 12,
    optime: { ts: Timestamp({ t: 1750061690, i: 1 }), t: Long('7') },
    optimeDate: ISODate('2025-06-16T08:14:50.000Z'),
    lastAppliedWallTime: ISODate('2025-06-16T08:14:50.177Z'),
    lastDurableWallTime: ISODate('2025-06-16T08:14:50.177Z'),
    syncSourceHost: '192.168.163.129:27017',
    syncSourceId: 1,
    infoMessage: '',
    configVersion: 2,
    configTerm: 7,
    self: true,
    lastHeartbeatMessage: ''
  },
  {
    _id: 1,
    name: '192.168.163.129:27017',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 11,
```

3.6 RQ 6 : 웹 기반 회의실 결과 구현

요구사항	세부 시험 절차
RQ-6	<ol style="list-style-type: none">리더 노드에 웹서버 환경을 준비(Flask)한다.app_reserve.py, HTML 템플릿 등 Flask 웹 서버 코드를 작성한다.

- | | |
|--|--|
| | <ol style="list-style-type: none">3. py 파일을 통해 서버를 실행한다.4. 브라우저로 접속하여 사용자 + 회의실 정보를 확인한다. |
|--|--|

1) 리더 노드에 웹서버 환경을 준비(Flask)한다.

리더 노드에서 python3-pip를 설치 후 flask pymongo를 설치하여 웹 서버 환경을 준비한다.

```
sudo apt update
```

```
sudo apt install python3-pip -y
```

```
pip3 install flask pymongo
```

2) app_reserve.py, HTML 템플릿 등 Flask 웹 서버 코드를 작성한다.

nano 명령어를 활용하여 app_reserve.py라는 이름으로 아래 <그림 19>와 같이 작성하여 저장한다.

<그림 19> app_reserve.py 구성 화면



```
GNU nano 7.2 app_reserve.py *
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"]

        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
```

templates 디렉토리를 만들고(`mkdir -p ~/myweb/templates`) nano 명령어를 활용하여

users.html라는 이름으로 아래 <그림 20>와 같이 작성하여 저장한다.

```
nano templates/users.html
```

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

<그림 20> users.html 구성 화면

```
GNU nano 7.2 templates/users.html
<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 활용하여 rooms.html라는 이름으로 아래 <그림 21>와 같이 작성하여 저장한다.

<그림 21> rooms.html 구성 화면

```
GNU nano 7.2 templates/rooms.html *
<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 활용하여 reserve.html라는 이름으로 아래 <그림 22>와 같이 작성하여 저장한다.

<그림 22> reserve.html 구성 화면

```
GNU nano 7.2 templates/reserve.html
<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 활용하여 reservations.html라는 이름으로 아래 <그림 23>와 같이 작성하여 저장한다.

<그림 23> reservations.html 구성 화면

```
GNU nano 7.2 templates/reservations.html
<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) py 파일을 통해 서버를 실행한다.

구성한 app_reservation.py 파일을 실행하면(`python3 app_reserve.py`) 아래 <그림 24>와 같이 출력된다. 해당 웹 주소로 접속한다.

<그림 24> 출력 화면

```
(venv) jimin@jimin-VMware-Virtual-Platform:~/myweb$ python app_reserve.py
* Serving Flask app 'app_reserve'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.163.129:5000
```

4) 브라우저로 접속하여 사용자 + 회의실 정보를 확인한다.

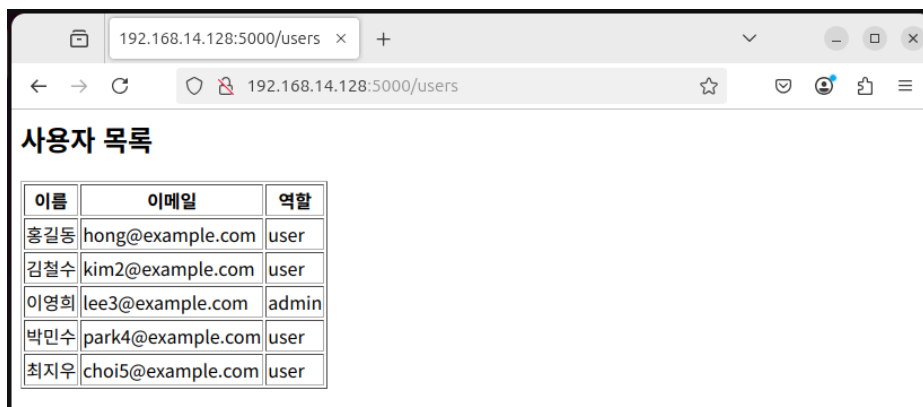
출력되는 웹 주소로 웹 브라우저를 통해 접속하면 아래 <그림 25>와 같이 메인 페이지가 출력된다.

<그림 25> 메인 페이지 화면

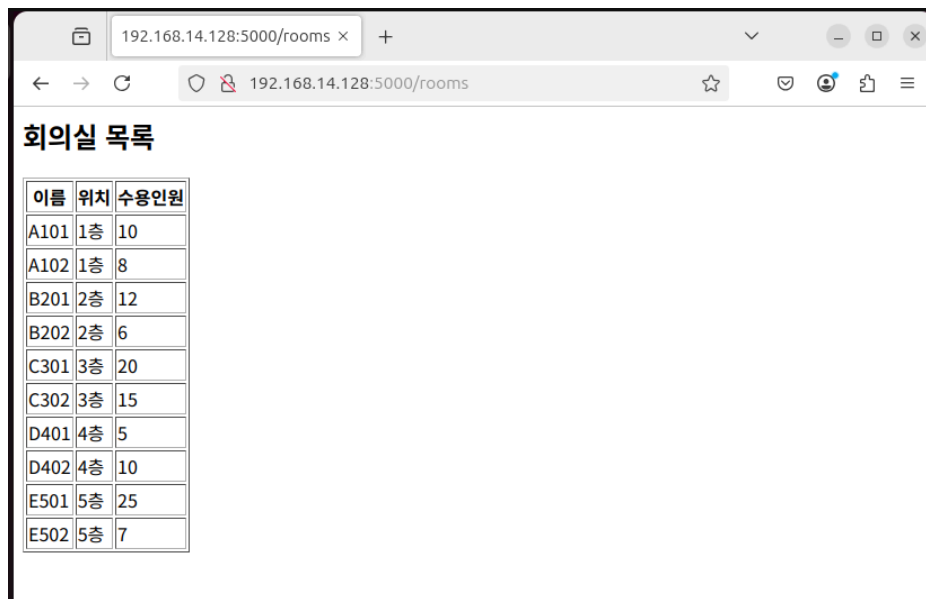


먼저 예약 시스템의 메인 페이지가 화면에 출력되었다. 메인 페이지에서 각 4가지 항목을 클릭하여 해당 페이지로 이동함을 확인할 수 있다. 아래 <그림 26>, <그림 27>, <그림 28>은 각각 순서대로 사용자 목록, 회의실 목록, 예약 목록 출력 화면이다.

<그림 26> 사용자 목록 화면

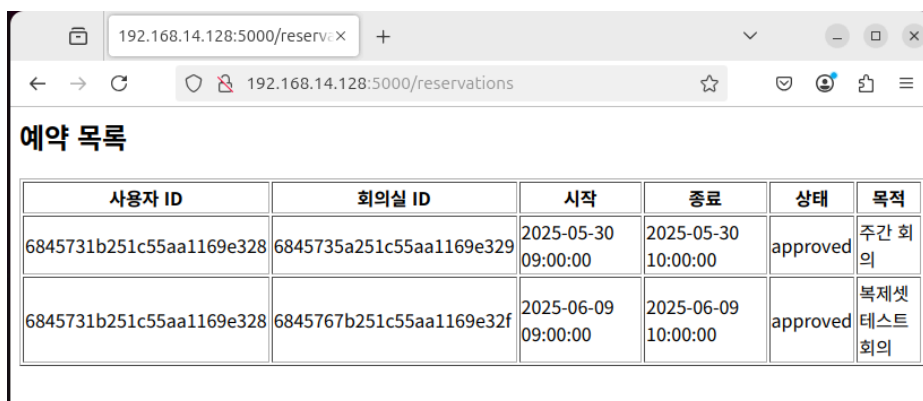


<그림 27> 회의실 목록 화면



이름	위치	수용인원
A101	1층	10
A102	1층	8
B201	2층	12
B202	2층	6
C301	3층	20
C302	3층	15
D401	4층	5
D402	4층	10
E501	5층	25
E502	5층	7

<그림 28> 예약 목록 화면



사용자 ID	회의실 ID	시작	종료	상태	목적
6845731b251c55aa1169e328	6845735a251c55aa1169e329	2025-05-30 09:00:00	2025-05-30 10:00:00	approved	주간 회의
6845731b251c55aa1169e328	6845767b251c55aa1169e32f	2025-06-09 09:00:00	2025-06-09 10:00:00	approved	복제셋 테스트 회의

이제 회의 예약하기 페이지로 이동하면 예약정보를 입력할 수 있는 항목들이 출력됨을 확인할 수 있다.

특히 사용자 이름과 회의실 이름 항목에서는 기존의 사용자 목록과 회의실 목록의 이름을 기반으로

선택할 수 있다. 또한 정보를 입력한 후 예약하기 버튼을 누르면 자동으로 예약 목록 페이지로

이동되어 업데이트된 예약 목록을 확인할 수 있다. 아래 <그림 29>에서 그 과정을 확인할 수 있다.

<그림 29> 회의 예약 화면과 예약 후, 예약 목록으로 리다이렉트되는 화면

회의실 예약

사용자: 이영희 ▼

회의실: B202 ▼

시작 시간: 2025 . 06 . 10 . 오후 02 : 00 📅

종료 시간: 2025 . 06 . 10 . 오후 03 : 00 📅

목적: 예약 테스트

예약하기

예약 목록

사용자 ID	회의실 ID	시작	종료	상태	목적
6845731b251c55aa1169e328	6845735a251c55aa1169e329	2025-05-30 09:00:00	2025-05-30 10:00:00	approved	주간 회의
6845731b251c55aa1169e328	6845767b251c55aa1169e32f	2025-06-09 09:00:00	2025-06-09 10:00:00	approved	복제셋 테스트 회의
68457588251c55aa1169e32c	6845767b251c55aa1169e332	2025-06-10T14:00	2025-06-10T15:00	pending	예약 테스트