**Full-Stack Service Programming**

**최종 보고 평가표**

|  |  |
| --- | --- |
| 년 도 | 2024년도 1학기 |
| 이 름 | 우상혁 |
| 학 번 | 2020105637 |
| 학 과 | 컴퓨터공학과 |

**[1] 정량 평가 항목**

|  |  |
| --- | --- |
| 항목 | 숫자 |
| 클라이언트 소스코드 파일 개수 | 18 |
| 클라이언트 소스코드 파일들의 라인 개수 총합계 | 2570 |
| 클라이언트 Class 개수 | 37 |
| 클라이언트 Class들의 메소드 개수 총합계 | 69 |
| 클라이언트 Class에 속하지 않은 함수 개수 총합계 | 1(main) |
| 클라이언트 Thread 개수 | 1 |
| 서버 Microservice 개수 | 15 |
| 서버 소스코드 파일 개수 | 10 |
| 서버 소스코드 파일들의 라인 개수 총합계 | 482 |
| 서버 Class 개수 | 5 (외부 클래스) |
| 서버 Class들의 메소드 개수 총합계 | 12 (외부 클래스의 메소드) |
| 서버 Class에 속하지 않은 함수 개수 총합계 | 20 (코드서 정의한 전역 함수) |
| 서버 Thread 개수 | 4 |

**[2] 기술 평가 항목**

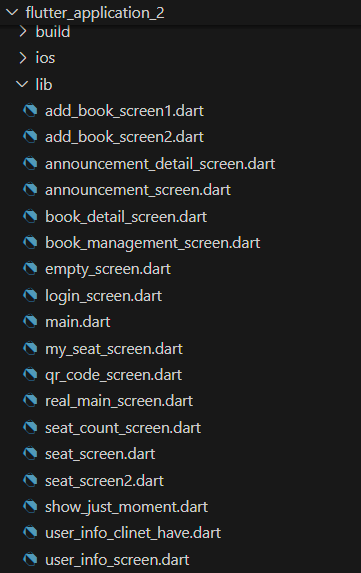
(2-1) 클라이언트가 관리하는 정보 (구조도, 코드 위치)

로그인시, 유저의 이름과 id를 저장한다. 멤버 변수로써 username과 userId로 저장된다.

실제로는 mongo DB의 username과 \_id를 다시 받아와서 저장한다.

클라이언트가 서버의 기능을 사용하기 위해 유저의 검증이 필요할때 사용된다.

예시: 좌석 예약시 mongo DB의 좌석데이터에 클라이언트가 가진 userId가 추가되는데, 이후 좌석을 반납할시 클라이언트의 userId와 일치하는지 검증하는 부분이 있다. 다른 기능들도 마찬가지다. 사용자 정보 업데이트, 도서 대여, 반납부터 QR 코드생성까지 반드시 사용되는 중요한 정보다.



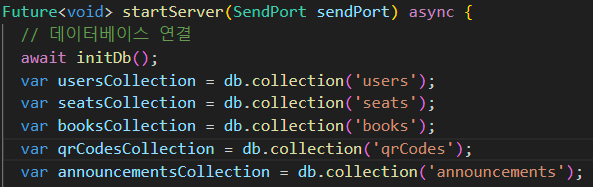
User\_info\_client\_have.dart 에 저장된다.

따로 파일을 분리시키진 않고 다른 클라이언트 dart파일과 동일한 위치에 두었다.(맨 아래서 위)

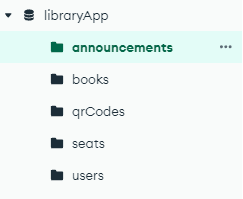
(2-2) 서버가 관리하는 정보 (구조도, 코드 위치)

Dart/flutter 서버에서 관리하는 정보는 아무것도 없다.

모든 정보는 Mongo DB 서버에 저장된다. 즉 모든 정보는 JSON형식으로 저장된다.



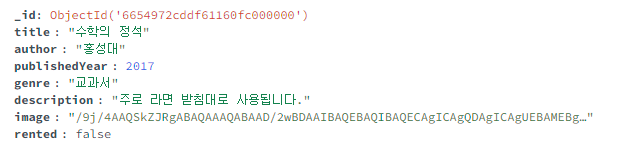
각 정보간 구별을 위해 5개의 데이터 콜렉션을 생성하여 각각 저장한다.

 (Mongo DB에서의 모습)

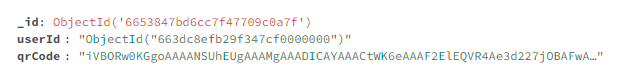
1. 공지사항에 넣을 글 (초기 데이터 삽입)



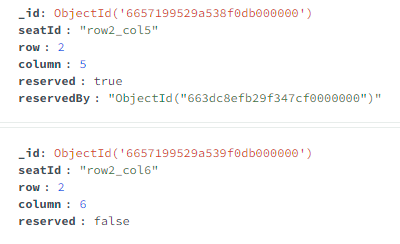
1. 책 정보



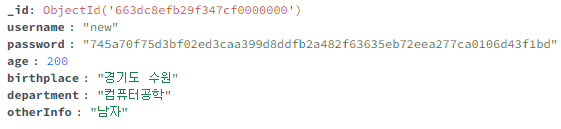
1. Qr code로 사용할 데이터 (이미지가 변환된 코드 데이터. )



1. 좌석 정보 (초기 데이터 삽입 및 이후 변경)

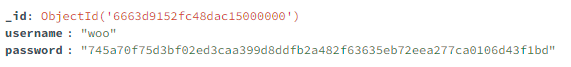


1. 유저 정보



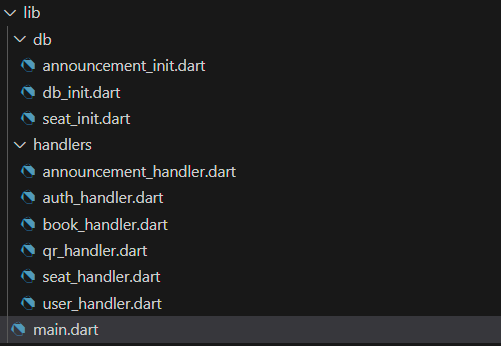
MongoDB는 NOSQL이므로 실제로 저장되는 정보는 위와 같이 모든 필드를 가지지 않아도된다.

예를들어, 맨 처음 로그인시 생성되는 유저 정보는 다음과 같지만 같은 콜렉션에 저장된다.



이후 서버코드의 기능에 의해서 데이터가 추가되거나 수정된다.

다른 데이터들도 이와 마찬가지인 과정을 겪는다.



( 전체 서버의 구조)

(2-3) 클라이언트와 서버가 송수신 하는 정보 (RESTful API, JSON 정보 등

1. 회원 가입

서버 엔드포인트: POST /signup

router.post('/signup', (Request request) => signupHandler(request, usersCollection));

• 클라이언트 요청: json

{ "username": "User", "password": "Password"}

• 서버 응답:

• 성공 시: json

{ "message": "Signup successful"}

• 실패 시: json

{ "message": "Username already exists"}

2. 로그인

서버 엔드포인트: POST /login

router.post('/login', (Request request) => loginHandler(request, usersCollection));

• 클라이언트 요청: json

{ "username": "User", "password": "Password"}

• 서버 응답:

• 성공 시: json

{ "status": "success", "userId": "someUserId"}

• 실패 시: json

{ "message": "Invalid credentials"}

3. 사용자 정보 조회

서버 엔드포인트: GET /user/info

 router.get('/user/info', (Request request) => userInfoHandler(request, usersCollection));

• 클라이언트 요청: URL에 userId 포함 전달 (예: <http://10.0.2.2:8080/user/info?userId=$userId>)

사용자 ID를 URL의 쿼리 매개변수로 포함 즉 String으로 전달. 검증에 사용된다.

• 서버 응답: json

{ "userId": "someUserId", "username": "exampleUser", "department": "Computer Science", "age": 25, "birthplace": "Seoul", "otherInfo": "Some other info"}

4. 사용자 정보 업데이트

서버 엔드포인트: POST /user/update

router.post('/user/update', (Request request) => updateUserHandler(request, usersCollection));

• 클라이언트 요청: json

{ "userId": "someUserId", "updateData": { "department": "New Department", "age": 26, "birthplace": "Busan", "otherInfo": "Updated info" }}

• 서버 응답: json

{ "message": "Update successful"}

5. 좌석 정보 조회

서버 엔드포인트: GET /seats

  router.get('/seats', (Request request) => getSeatsHandler(request, seatsCollection));

• 클라이언트 요청: get 요청만. 본문 x

• 서버 응답: json

[ { "seatId": "row1\_col1", "row": 1, "column": 1, "reserved": false }, ... ]

6. 특정 좌석 상태 확인

서버 엔드포인트: GET /seat/status

router.get('/seat/status', (Request request) => seatStatusHandler(request, seatsCollection));

• 클라이언트 요청: URL에 선택한 seatId를 포함된다. 3번의 경우처럼 보내짐

• 서버 응답: json

{ "seatId": "row1\_col1", "row": 1, "column": 1, "reserved": false}

7. 좌석 예약

서버 엔드포인트: POST /reserve

router.post('/reserve', (Request request) => reserveSeatHandler(request, seatsCollection));

• 클라이언트 요청: json

{ "seatId": "row1\_col1", "userId": "someUserId"}

• 서버 응답: json

{ "message": "Seat reserved"}

8. 좌석 반납

서버 엔드포인트: POST /cancel\_reservation

router.post('/cancel\_reservation', (Request request) => cancelReservationHandler(request, seatsCollection));

• 클라이언트 요청: json

{ "seatId": "row1\_col1", "userId": "someUserId"}

• 서버 응답: json

{ "message": "Reservation cancelled"}

9. 제 1열람실, 제 2열람실의 남은 좌석 계산

서버 엔드포인트: GET /seats/reserved\_count

router.get('/seats/reserved\_count', (Request request) => getReservedCountHandler(request, seatsCollection));

• 클라이언트 요청: get 요청만. 본문x

• 서버 응답: json

{ "reservedCountRoom1": 10, "reservedCountRoom2": 5}

10. 도서관 모바일 이용증 클릭시 QR 코드 생성

서버 엔드포인트: GET /generate\_qr

router.get('/generate\_qr', (Request request) => generateQrHandler(request, usersCollection, qrCodesCollection));

• 클라이언트 요청: • URL에 userId를 포함 3번의 경우와 비슷하다.

• 서버 응답: json

{ "qrCode": "base64EncodedQRCode"}

11. 도서 검색

서버 엔드포인트: GET /books/search

router.get('/books/search', (Request request) => searchBooksHandler(request, booksCollection));

• 클라이언트 요청: • URL에 title을 포함 3번의 경우와 비슷하다.

• 서버 응답: json

[ { "title": "Book Title", "author": "Author Name", "publishedYear": 2023, "genre": "Fiction", "description": "Book description", "image": "base64EncodedImage" }, ... more books]

12. 도서 대여

서버 엔드포인트: POST /books/rent

router.post('/books/rent', (Request request) => rentBookHandler(request, booksCollection));

• 클라이언트 요청: json

{ "bookId": "someBookId", "userId": "someUserId"}

• 서버 응답: json

{ "message": "Book rented"}

13. 도서 반납

서버 엔드포인트: POST /books/return

router.post('/books/return', (Request request) => returnBookHandler(request, booksCollection));

• 클라이언트 요청: json

{ "bookId": "someBookId", "userId": "someUserId"}

• 서버 응답: json

{ "message": "Book returned"}

14. 책 추가

서버 엔드포인트: POST /add\_book

router.post('/add\_book', (Request request) => addBookHandler(request, booksCollection));

• 클라이언트 요청: json

{ "title": "New Book Title", "author": "New Author", "publishedYear": 2024, "genre": "Non-Fiction", "description": "New book description", "image": "base64EncodedImage"}

• 서버 응답: json

{ "message": "Book added successfully"}

15. 공지사항 목록 조회

서버 엔드포인트: GET /announcements

router.get('/announcements', (Request request) => getAnnouncementsHandler(request, announcementsCollection));

• 클라이언트 요청: 본문 x get 요청만

• 서버 응답: json

[ { "title": "First Announcement", "date": "2024-01-01", "content": "First announcement content" }, { "title": "Second Announcement", "date": "2024-02-01", "content": "Second announcement content" }]

(2-4) 서버-서버가 송수신 하는 정보 (RESTful API, JSON 정보 등)

아까 2-3의 핸들러 함수를 중점으로 mongo DB서버와 dart/flutter 서버 사이의 동작을 본다.

둘간의 통신은 RESTful API가 아니고 mongo\_dart 패키지를 통해 MongoDB의 쿼리와 명령을 실행하는 형태로 이루어진다. 쿼리는 JSON과 유사한 BSON이다.

1. 회원 가입 (signupHandler)

서버 -> MongoDB:

컬렉션: users 메서드: insert

데이터: json

{ "username": "exampleUser",

"password": "hashedPassword"

}

MongoDB -> 서버:

데이터: json

{

"acknowledged": true,

"insertedId": "ObjectId('...')"

}

2. 로그인 (loginHandler)

서버 -> MongoDB: 컬렉션: users 메서드: findOne

쿼리: json

{ "username": "exampleUser",

"password": "hashedPassword"

}

MongoDB -> 서버:

데이터: json

{ "\_id": "ObjectId('...')",

"username": "exampleUser",

"password": "hashedPassword"

}

3. 사용자 정보 조회 (userInfoHandler)

서버 -> MongoDB: 컬렉션: users 메서드: findOne

쿼리: json

{ "\_id": "ObjectId('userId')"}

MongoDB -> 서버:

json

{ "\_id": "ObjectId('...')",

"username": "exampleUser",

"department": "exampleDepartment",

"age": 25,

"birthplace": "exampleBirthplace",

"otherInfo": "exampleOtherInfo"

}

4. 사용자 정보 업데이트 (updateUserHandler)

서버 -> MongoDB: 컬렉션: users 메서드: update

쿼리: json

{ "\_id": "ObjectId('userId')"

}

데이터: json

{ "$set": {

"department": "newDepartment",

"age": newAge,

"birthplace": "newBirthplace",

"otherInfo": "newOtherInfo"

}

}

MongoDB -> 서버:

데이터: json

{ "acknowledged": true,

"modifiedCount": 1,

"upsertedId": null,

"upsertedCount": 0,

"matchedCount": 1

}

5. 좌석 정보 조회 (getSeatsHandler)

서버 -> MongoDB: 컬렉션: seats 메서드: find 쿼리: {} (전체 조회)

MongoDB -> 서버:

json

[ {

"seatId": "row1\_col1",

"row": 1,

"column": 1,

"reserved": false

},

{

"seatId": "row1\_col2",

"row": 1,

"column": 2,

"reserved": true,

"reservedBy": "ObjectId('userId')"

},

...

]

6. 특정 좌석 상태 확인 (seatStatusHandler)

서버 -> MongoDB: 컬렉션: seats 메서드: findOne

쿼리: json

{ "seatId": "row1\_col1"

}

MongoDB -> 서버:

데이터: json

{ "seatId": "row1\_col1",

"row": 1,

"column": 1,

"reserved": false

}

7. 좌석 예약 (reserveSeatHandler)

서버 -> MongoDB: 컬렉션: seats 메서드: update

쿼리: json

{ "seatId": "row1\_col1"}

데이터: json

{ "$set": {

"reserved": true,

"reservedBy": "ObjectId('userId')"

}

}

MongoDB -> 서버:

데이터: json

{ "acknowledged": true,

"modifiedCount": 1,

"upsertedId": null,

"upsertedCount": 0,

"matchedCount": 1

}

8. 좌석 반납 (cancelReservationHandler)

서버 -> MongoDB: 컬렉션: seats 메서드: update 쿼리: json

{ "seatId": "row1\_col1",

"reservedBy": "ObjectId('userId')"

}

데이터: json

{ "$set": {

"reserved": false

},

"$unset": {

"reservedBy": ""

}

}

MongoDB -> 서버:

데이터: json

{ "acknowledged": true,

"modifiedCount": 1,

"upsertedId": null,

"upsertedCount": 0,

"matchedCount": 1

}

9. 제 1열람실, 제 2열람실의 남은 좌석 계산 (getReservedCountHandler)

서버 -> MongoDB: 컬렉션: seats 메서드: find

쿼리: {} (전체 조회)

MongoDB -> 서버:

데이터: json

[ {

"seatId": "row1\_col1",

"row": 1,

"column": 1,

"reserved": false

},

{

"seatId": "row1\_col2",

"row": 1,

"column": 2,

"reserved": true,

"reservedBy": "ObjectId('userId')"

},

...

]

10. 도서관 모바일 이용증 클릭 시 QR 코드 생성 (generateQrHandler)

서버 -> MongoDB: 컬렉션: users 메서드: findOne

쿼리: json

{ "\_id": "ObjectId('userId')"}

MongoDB -> 서버:

데이터: json

{ "\_id": "ObjectId('userId')",

"username": "exampleUser"

}

서버 -> MongoDB: 컬렉션: qrCodes 메서드: update

쿼리: json

{ "userId": "ObjectId('userId')"}

데이터: json

{ "$set": {

"qrCode": "base64EncodedQRCode"

}

}

MongoDB -> 서버:

데이터: json

{ "acknowledged": true,

"modifiedCount": 1,

"upsertedId": null,

"upsertedCount": 0,

"matchedCount": 1

}

11. 도서 검색 (searchBooksHandler)

서버 -> MongoDB: 컬렉션: books 메서드: find

쿼리: json

{ "title": {"$regex": "searchQuery", "$options": "i"}}

MongoDB -> 서버:

데이터: json

[ {

"title": "bookTitle",

"author": "bookAuthor",

"publishedYear": 2024,

"genre": "bookGenre",

"description": "bookDescription",

"image": "base64EncodedImage"

},

...

]

12. 도서 대여 (rentBookHandler)

서버 -> MongoDB: 컬렉션: books 메서드: findOne

쿼리: json

{ "\_id": "ObjectId('bookId')"}

MongoDB -> 서버:

데이터: json

{ "\_id": "ObjectId('bookId')",

"title": "bookTitle",

"author": "bookAuthor",

"publishedYear": 2024,

"genre": "bookGenre",

"description": "bookDescription",

"image": "base64EncodedImage",

"rented": false

}

서버 -> MongoDB: 컬렉션: books 메서드: update

쿼리: json

{ "\_id": "ObjectId('bookId')"}

데이터: json

{ "$set": {

"rented": true,

"rentedBy": "ObjectId('userId')"

}

}

MongoDB -> 서버:

데이터: json

{ "acknowledged": true,

"modifiedCount": 1,

"upsertedId": null,

"upsertedCount": 0,

"matchedCount": 1

}

13. 도서 반납 (returnBookHandler)

서버 -> MongoDB: 컬렉션: books 메서드: findOne

쿼리: json

{ "\_id": "ObjectId('bookId')"}

MongoDB -> 서버:

데이터: json

{ "\_id": "ObjectId('bookId')",

"title": "bookTitle",

"author": "bookAuthor",

"publishedYear": 2024,

"genre": "bookGenre",

"description": "bookDescription",

"image": "base64EncodedImage",

"rented": true,

"rentedBy": "ObjectId('userId')"

}

서버 -> MongoDB: 컬렉션: books 메서드: update

쿼리: json

{ "\_id": "ObjectId('bookId')",

"rentedBy": "ObjectId('userId')"

}

데이터: json

{ "$set": {

"rented": false

},

"$unset": {

"rentedBy": ""

}

}

MongoDB -> 서버:

데이터: json

{ "acknowledged": true,

"modifiedCount": 1,

"upsertedId": null,

"upsertedCount": 0,

"matchedCount": 1

}

14. 책 추가 (addBookHandler)

서버 -> MongoDB: 컬렉션: books 메서드: insert

데이터: json

{ "title": "bookTitle",

"author": "bookAuthor",

"publishedYear": 2024,

"genre": "bookGenre",

"description": "bookDescription",

"image": "base64EncodedImage"

}

MongoDB -> 서버:

데이터: json

{ "acknowledged": true,

"insertedId": "ObjectId('...')"

}

15. 공지사항 목록 조회 (getAnnouncementsHandler)

서버 -> MongoDB: 컬렉션: announcements 메서드: find

쿼리: {} (전체 조회)

MongoDB -> 서버:

데이터: json

[ {

"title": "announcementTitle",

"date": "2024-06-08",

"content": "announcementContent"

},

...

]

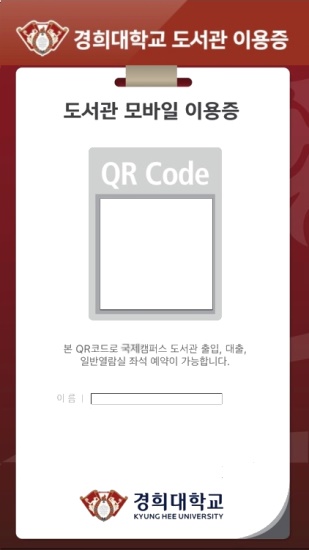
(2-5) [optional] 창의적 알고리즘

X

(2-6) [optional] 독창적 사용자 인터페이스 (UI/UX 등)

경희대 학생으로서 구 경희대 도서관 어플리케이션의 화면을 사용했습니다.(편집 됨)









화면만 가져오고 실제 버튼은 보이지 않은 버튼으로 구현했습니다.

**[3] 정성 평가 항목**

(3-1) 최종 데모 영상의 완성도 [기본 평가 항목]

다음이 데모 영상에서 보여준 기능들이다.

1. 로그인

2. 사용자 정보 조회 및 업데이트:

3. 좌석 예약 및 반납

4 . 예매한 좌석정보 확인

5. 제 1,2열람실의 남은 좌석 계산 (총 좌석+ 여석): . ‘

6. QR코드 생성:

7. 도서 검색

8 .도서 대여

9. 도서 반납

10 .도서 추가 :

11. 공지사항 목록 조회:

한정된 시간안에 제법 많은 기능들을, 사실상 대부분의 기능들을 보여 주었다.

이 데모 영상에서 사실 제일 성공적인 부분을 고르라면은 UI부분에서 구 도서관 애플리케이션의 화면을 거의 동일하게 구현했다는 것이다. 분명 이 특징 하나만으로 관심을 끌기에는 무조건 성공할 것이다. 과연 내가 생각하던 그 기능이 나올까? 하고 말이다.

그리고 UI부분에 힘을 입어서, 도서관 좌석 예약기능을 좀더 효과적으로 보여줄 수 있을것이다.

사실상, 그 부분이 사람들이 가장 많이 사용한 기능이므로 더욱 주의를 기울일것이 분명하다.

일단 이 어플리케이션의 핵심 기능이라고 할 수 있는 좌석예약과 취소기능, 도서 추가와 검색기능을 보여 주었으므로, 제작한 프로젝트의 목적성과 기능은 영상에서 잘 전달되었다.

때문에 애플리케이션의 데모 영상이라는 측면에서는, 훌륭한 완성도라고 생각한다.

(3-2) 최종 프로그램의 완성도 [기본 평가 항목]

이 프로젝트의 목적성은 이전의 구 도서관 애플리케이션을 구현해 보자는데에 중점을 둔다.

현 도서관 애플리케이션의 좌석 예약 기능은 다른 인터넷 사이트에 직접 들어가서, 그 좌석화면을 받아와서 예약을 하는 불편한 방식이기 때문이다.(구 애플리케이션은 좌석화면이 내장되있음)

어쨌든 나는 이 프로그램의 완성도를 구 도서관 애플리케이션의 기능을 어느 정도 구현했고, 또 여기서 추가한 새로운 기능이 있는가로 판단하고자 한다.



저 도서 추가가 실제로는 도서관 SNS인것을 제외하면, 이게 구 애플리케이션의 주요 화면이다.

그리고 다음이 내가 구현한 기능이다.

1. 회원가입 : 이전에 존재하지 않았던 추가한 기능. 실제 구 애플리케이션은 학번과 비밀번호만으로 접속이 가능했다. 나는 이름과 비밀번호를 등록하는 것으로 이 기능을 구현했다.

2. 로그인 및 로그아웃 : 기존 구 도서관의 기능과 같다. 서버에 등록된 이름, 비밀번호가 일치하는지 확인한다. 그리고 화면에 보이는 logout을 누르면 로그인 화면으로 이동한다.

3. 사용자 정보 조회: 이전에 존재하지 않았던 추가한 기능. 구 애플리케이션은 로그인시 이름이 화면 맨 위에 뜨는 것 말고는 그런 기능이 없었다.

4. 사용자 정보 업데이트: 3번의 연장판. 다른 속성(생일, 출생지)를 추가할 수 있다.

5. 특정 좌석 상태 확인:

6. 좌석 예약:

7. 좌석 반납: 5,6,7은 구 애플리케이션과 동일한 기능을 구현했다고 생각한다. ‘열람실 좌석 예약’을 누르면 클라이언트에 도서관 좌석 그림이 나타나고(내장됨), 서버로부터 실시간으로 데이터를 받아와 자리가 찬 좌석이 검게 표시된다. 그리고 좌석을 예약하면, 실시간으로 클라이언트의 화면에도 그게 반영되게 만들었다. 단, 좌석 반납 기능은 내 기억상에 좌석을 예약하는 화면에서 한번에 처리하지 않고 다른곳에서 처리 되게 만든 것 같은데, 나는 동일한 화면에서 그 기능을 수행하도록 처리했다. 예약된 좌석을 누르면 반납이 되게 만들었다. 기능은 같다.

8 .예매한 좌석정보: ‘내자리’ 버튼을 누르면 좌석 정보가나온다. 동일한 기능을 구현.

9. 제 1,2열람실의 남은 좌석 계산 (총 좌석+ 여석): . ‘열람실 좌석 예약’을 누르면 열람실을 선택하는 화면이 나타나는데, 이때 좌석과 여석이 나온다. 동일한 기능을 구현했다.

10. QR코드 생성: 도서관 모바일 이용증을 누르면 QR코드가 나온다. 실제 구 애플리케이션은 새로 고침을 누를 때 마다 새로 생성되는걸 보면 아마 시간 같은 변수를 QR코드 생성시에 사용한 것으로 추측된다. 나는 사용자의 이름과 ID로만 만들어서 계속 새롭게 QR코드를 생성하지는 못하지만 그래도 유일한 QR코드를 생성하게 구현했다.

11. 공지사항 목록 조회: 같은 기능을 구현했다. 공지사항(글)을 서버로부터 받아와서, 그걸 사용자에게 보여주는 기능이다.

12. 도서 검색

13 .도서 대여

14. 도서 반납

15 .책추가 : 사실 이 12, 13, 14 ,15의 기능은 구 애플리케이션에 없는 기능이다. 이 기능을 추가한 이유는, 남아있는 기능들중 메시지, 그룹 스터디실, 도서관홈페이지, 도서관 SNS는 구 애플리케이션이 아직 동작하고 있었던 시절에도 난 단 한번도 사용한적이 없었기 때문이다. 즉 별로 의미없는 기능이다. 때문에 나는 새로운 기능을 추가하기로 했다.

내가 원하는 책을 서버에 추가하고, 이를 검색해 볼 수 있다. 책을 추가할때는 단순히 문자열 만이 아니라 책을 나타내는 그림도 선택할수 있게 만들었다. 그리고 이를 서버에 추가하여 검색해 볼수 있다. 책의 제목을 끝까지 치지 않아도 그 부분까지 제목과 일치하면, 해당하는 제목들의 책이 검색에 나타난다. 그리고 책의 대여와 반납으로, 책의 상태가 바뀌도록 만들었다.

이처럼 구 도서관 애플리케이션의 핵심적인 기능을 모두 구현했고, 여기에 더해 새로운 기능을 추가했다. 나름 목적에 충실한 최종 프로그램이라고 생각한다.

(3-3) [optional] 개발의 지속성 확인 여부 (GitHub 등 활용 통한 꾸준한 개발)

X

(3-4) [optional] AppStore, PlayStore 등록 여부

X

(3-5) [optional] 서비스의 공개 운영 여부

X

(3-6) [optional] 클라이언트과 서버 프로그램의 오픈소스 공개 여부

X

(3-7) [optional] 교수에 대한 1:1 라이브 데모 및 개별 Q&A 진행 여부

X

(3-8) [optional] 테스트 코드의 정성/정량 수준

X

(3-9) [optional] 문서화의 정성/정량 수준

서버에 해당하는 flutter\_application\_1의 거의 대부분의 코드에 주석을 달아 무슨 역할을 하는지 설명. Main.dart에 모든 엔드 포인트를 모아놔서, 서버의 전체 기능을 한눈에 확인 가능.

서버의 handler 역할을 하는 함수와 초기 데이터 삽입 함수를 다른 파일에 분리시켜 기능을 명확히 했다.

(3-10) [optional] CI/CD & Microservice (container/orchestrator) 적용

X