

실무중심산학 최종 결과보고서

홈즈 프로젝트 Homes Project

2020년 12월 09일

32164959 허전진

32164420 조정민

32163006 이건욱

32150781 김승준

제출 자료 :

1. 최종 결과물 (결과물 및 S/W 소스코드, 설치 및 실행파일, 실행 동영상 파일)
2. 설치 매뉴얼 (설치방법과 사용방법 메뉴얼)
3. 최종 발표 자료

목차

1. 개발 과제 개요	1
(1) 개발 과제 목표	1
(2) 개발 과제 필요성	1
(3) 개발 제한조건	3
(4) 개발 과제의 기대효과	4
2. 과제 수행 내용	6
(1) 개발 환경 및 요구 조건	6
(2) 시스템 블록 다이어그램	6
(3) 알고리즘	8
(4) 구현 방법	10
(5) 실험 결과 및 성능 분석	22
3. 과제 수행 결과 내역	27
(1) 추진 계획 및 실적	27
(2) 참여 인원별 역할 및 수행 소감	29
(3) 과제 결과물	32
(4) 활용 방안	40
4. 후기	41
[별첨] homes 사용방법 메뉴얼	*

1. 과제 개요

(1) 개발 과제 목표

체계적인 관리가 이루어지는 아파트 단지와 달리 대학가, 주택가 등 일반 임대형 주택에서는 건물에 대한 관리가 이루어지기 힘들고, 건물주/세입자/관리자끼리의 상호 소통이 어려운 점, 세입자에 대한 건물주의 관리가 체계적이지 못하다는 점을 해결하는 것이 우리가 개발한 어플리케이션의 목표라고 할 수 있다.

(2) 개발 과제 필요성

- Requirement Analysis (연구 배경, 필요성, customer needs 조사, 문헌조사)

우리나라는 수도권, 광역시처럼 특정 토지 영역에 높은 인구 밀도를 가지는 특성을 가지고 있는 나라다. 부동산 역시 밀집이 되어 많은 건물들이 빽빽하게 형성되어 있는데, 개인주의가 팽배해진 지금의 각박한 세상에서 세입자, 건물주들 간에 문제가 많이 야기되고 있다.



<그림 1 - 소음으로 인한 조기 사망>



<그림 2 - 세입자 간 층간소음 문제>

대표적으로 층간 소음과 분리수거 문제에 대한 공지, 건물 유지/보수 의뢰 및 해결에서 우리는 평소에 불편함을 많이 겪었으며 이러한 문제들에 대한 여러 이슈들 또한 많이 접한바 이러한 문제들을 해결하는 것에 우리는 초점을 맞췄다.

우리가 가장 중요하게 생각한 우리 개발의 목표는 건물주/관리인/세입자 간의 원활한 소통이었기 때문에 우리는 이 기능에 초점을 맞추며 개발의 범위를 넓혀나갔다.



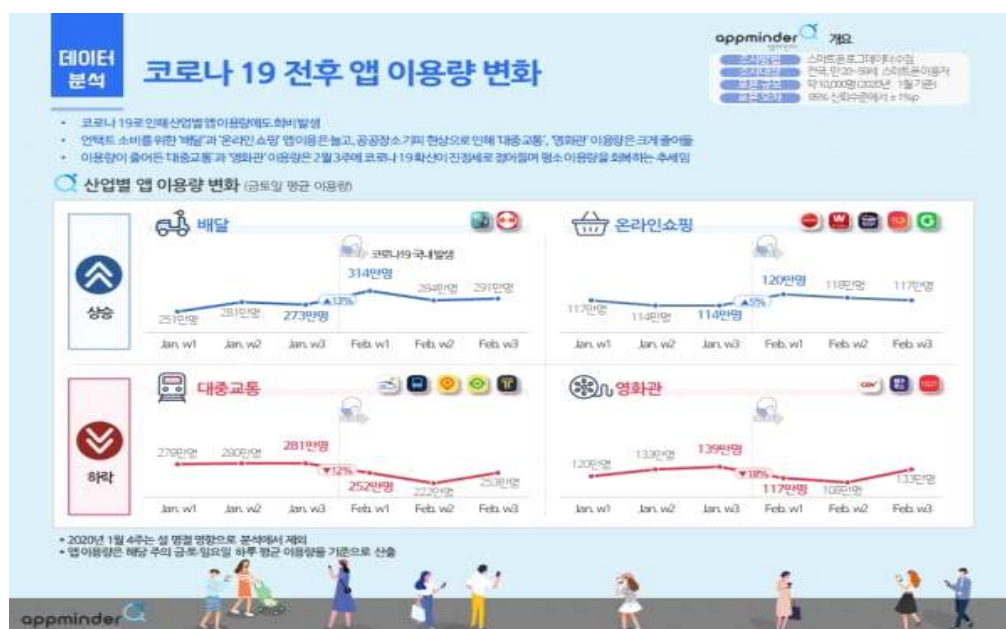
<그림 3 - 공동주택의 분리수거 현실>

<그림 4 - 유지/보수 의뢰>

홈즈 프로젝트에 참여한 조원들 모두가 자취를 하면서 건물주, 관리인과의 연락이 잘 되지 않거나 번거로운 경우가 많았고 세입자 간에 생기는 갈등, 불만을 표출하기에 제한이 많다는 것을 경험하였다.

그리고 본 프로젝트를 하기에 앞서 시장조사를 진행한 결과, 출시된 어플 중 일반 사용자가 건물 통합 관리 시스템을 목표로 사용하기엔 미흡한 어플들이 많았고, 대부분 기업을 타겟으로 만들어져 일반 사용자가 쓰기에는 제한이 많다는 것을 알 수 있었다. 그래서 일반 사용자도 쉽게 접근하고 관리할 수 있는 관리 시스템이 있으면 좋겠다고 생각을 하면서 접근을 하기 시작했다.

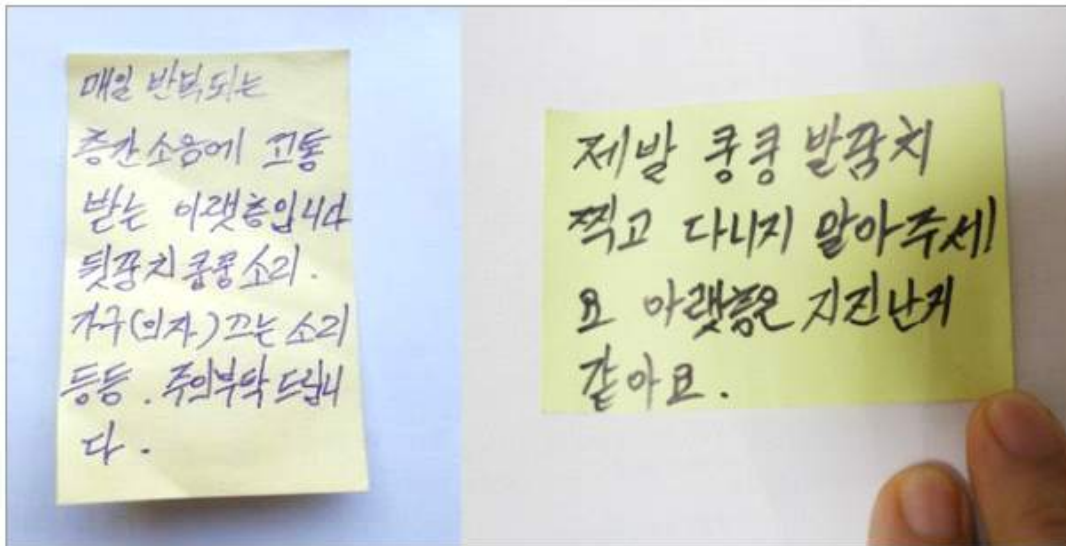
COVID-19가 창궐한 현 시국에서 모바일 시장은 더욱 활성화가 될 수 밖에 없었다. 하지만 모든 분야에서 긍정적인 것은 아니다. 다음 자료를 보면 외부 활동 시 활용하는 '대중교통' 앱은 11.7% 감소하고 '영화 예매'앱은 18.5% 줄었다.



<그림5 - 코로나 전후 산업별 앱 이용량 통계>

이 자료를 통해 언택트 시대에 걸맞게 ‘배달’과 ‘온라인쇼핑’앱과 같이 비대면으로 업무를 해결할 수 있는 산업이 시장에서 살아남는 것을 확인할 수 있었다.

따라서 홈즈 프로젝트는 비대면으로 건물주/관리인/세입자 간 용무를 해결할 수 있도록 전반적인 틀을 구성하였다.



▲ 아랫집이 우리집에 남긴 메모들 아랫집이 우리집에 남긴 4월 19일의 메모(왼쪽)과 4월 25일의 메모(오른쪽). 25일에는 새벽부터 밤까지 16시간 넘게 집이 비어있었다.

결과적으로, 홈즈 프로젝트는 모든 사용자 간 비대면 업무 처리와 더불어 서로 민감한 문제에 있어서 원만한 해결을 도울 수 있도록 공지사항, 푸시 알림 등을 건물 통합 관리 시스템에 도입하여 문제를 원활하게 해결할 수 있도록 설계하였다.

(3) 개발 제한조건

- 공학적, 경제적 제한조건

공학적인 제약조건으로서는 모든 어플리케이션들이 직면하는 문제겠지만 방대한 양의 데이터를 관리하는 것에 어려움이 많다. 여러 건물들에 대해 각각의 세대로 접근함으로써 많은 세대의 수 만큼 많은 양의 데이터를 필요로 할 것이고 각각의 세대에 대한 월세, 유지/보수 사항, 세입자에 대한 정보 등 많은 데이터에서도 더 많은 데이터를 필요로 할 것이다. 이에 따라 우리는 이러한 막대한 양에 대한 데이터를 반 정규화, 테이블 최적화를 통해 데이터 CRUD 작업(조회, 삽입, 삭제 등)에 소요되는 시간을 최소화할 계획이다.

또한, 데이터를 저장하는 데에는 항상 비용이 발생한다. 모든 데이터는 법적인 근거가 될 수도 있으므로 일정 기간 동안 안정적으로 저장되어야 한다. 이 데이터는 유저의 정보, 건물의 정보, 유지 보수의 정보를 모두 포함하고 각 세대별로 나누어 저장하기 때문에 필요한 데이터베이스의 규모가 큰 편이다. 학교에서 지원하는 서버 인스턴스의 사양은 제한적이므로 조건에 맞는 데이터베이스 서버를 구축할 필요가 있다.

- 환경적 제한조건

우리는 앱을 개발하는데 있어서 건물주/관리인/세입자 각각이 느끼고 있는 불편함을 수용하여 개발하고자 하였지만 우리 팀원 각각은 세입자의 입장으로써 아무래도 세입자의 기능에 더욱 초점이 맞춰질 수밖에 없었다. homes의 앞으로의 개발에 있어서는 보다 자세한 조사를 통해 건물주와 관리인에 더욱 초점을 맞추어 계획이다.

- 사회적 제한조건

서비스의 핵심 기능 중 하나인 알림 기능은 제대로 된 관리와 지침 없이는 또 다른 사회적 문제를 야기할 수도 있다. 예를 들어 알림을 주고받는 중 언어폭력 등의 문제가 발생하거나, 부적절한 시간에 원치 않는 알림이 오는 등의 문제로 불상사가 생길 수 있다. 따라서 사회적 순기능의 역할을 수행하는 서비스가 되기 위해서는 데이터의 실명 보관과 특정 회원의 알림 차단 등의 관리가 필요하다.

(4) 개발 과제의 기대효과

- 기존 기술의 현황, 문제점 및 개선 방안

기존의 어플리케이션은 스마트 건물관리 시스템, 누리주택 등이 알려져 있다. 이 둘의 기능은 건물주, 세입자, 관리인이 소통하기에는 한계가 있다.

스마트 건물관리 시스템은 안전점검과 관리가 주된 기능이지만 사용대상이 일반 임대형 건물의 구성원이 아닌 기업을 대상으로 하고 있다. 따라서 우리가 개발하고자 하는 과제의 목적과 부합한 부분이 존재한다.

누리주택은 일반 임대형 건물을 대상으로 하고 있지만 해당 어플리케이션은 자산관리시스템(건물수익률, 건물자산가치비교분석, 매도컨설팅)과 건물의 총 가구 관리 등으로 관리인과 세입자가 아닌 건물주에게만 적합한 기능을 제공하고 있다.

현존하는 어플리케이션의 문제점을 종합해보자면 다음과 같다.

- 건물주/관리인/세입자 간의 원활한 소통이 불가
- 월/전세 내역이 통합관리가 아닌 개인적인 관리를 통해 이루어짐
- 건물주/관리인의 비체계적인 유지/보수 관리로 인해 문제점 다수 발생
- 세대 간 문제를 세입자가 대면으로 해결할 시 문제 발생 가능성

위와 같은 문제들을 일반 주거용 임대형 건물에 대한 통합관리시스템을 통한 관리와 실질적으로 관련이 있는 모든 사람들을 연결하여 소통하며 건물 내에서 일어나는 문제들을 앱 하나로 해결할 수 있도록 개선하는 것이 주된 목표이다.

각각의 사용자에 따른 기대되는 효과는 다양하다.

건물주는 기존의 아날로그 적인 방식과는 다르게 보다 편리하고 효율적으로 월/전세 현황과 각각의 세대에 관한 정보를 관리할 수 있고 통합 알림 서비스로 관리인/세입자에게 간편하게 알림 사항을 공지할 수 있다.

관리인은 보다 체계적인 관리 건물의 유지/보수 통합 관리를 통해 보다 빠르고 정확하게 문제 상황에 대해 파악할 수 있고 이에 따라 미흡한 유지/보수의 문제점을 보완할 수 있다.

세입자는 월세, 전세 내역을 한눈에 확인할 수 있으며, 직접 연락하는 대신 클릭 몇번으로 건물주/관리인에게 건물 하자 관련 수리를 의뢰할 수 있다. 추가적으로 세대 간의 문제가 발생하였을 때 세입자 간 쪽지알림을 통해 문제를 비대면으로 해결할 수 있다.

위와 같은 서비스가 실질적으로 건물 내 구성원들에게 제공되면 앱 하나로 구성원들 간의 통합적인 문제들을 해결할 수 있을 것으로 예측된다.

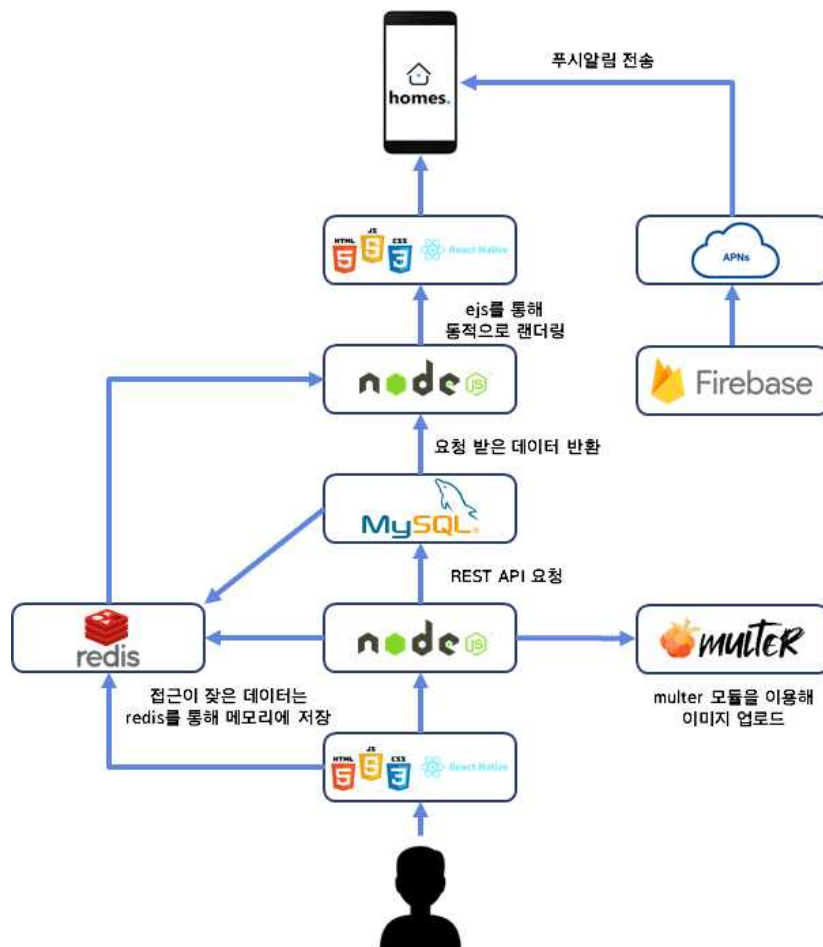
2. 과제 수행 내용

(1) 개발 환경 및 요구조건

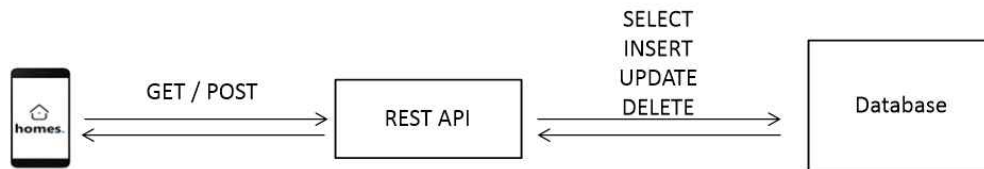
개발 환경

개발 언어	프레임워크/API	GUI tool	Cloud
HTML CSS JavaScript ES6 Node.js Express.js React Native MySQL	Firebase Expo SDK NginX	Visual Studio Code HeidiSQL MySQL workbench	Docker Ubuntu 20.04 LTS NHN Toast Cloud

(2) 시스템 블록 다이어그램



전반적인 시스템 흐름도

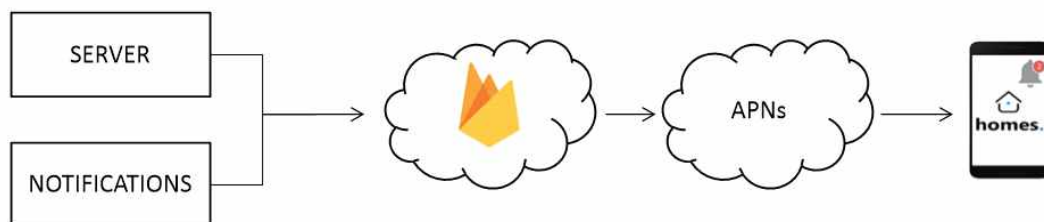


홈즈의 REST API 방식을 활용해 데이터를 처리한다. 이에 대한 기본적인 데이터 처리 기능은 CRUD 방식을 이용하였고 API에서는 SQL의 네 가지 DML 언어를 사용하여 데이터를 처리한다. 클라이언트가 API에 GET/POST 요청을 하면 REST API에서는 SELECT, INSERT, UPDATE, DELETE 등의 DML언어를 사용하여 Database에 데이터를 요청하고 Database는 요청받은 데이터를 반환한다.

- a) 클라이언트의 데이터 처리 요청(GET/POST)
- b) CRUD를 통한 REST API의 데이터 처리
- c) Database에서 요청 데이터 반환
- d) 클라이언트가 요구하는 리소스에 알맞은 데이터 제공

즉, REST API를 통해 요청이 수행될 때 REST API는 리소스 상태에 대한 표현을 요청자에게 전송하여 사용자가 요구하는 리소스에 알맞은 데이터를 제공할 수 있다.

푸시알림 흐름도



홈즈의 푸시알림을 구현하기 위해서 Firebase Cloud Messaging(FCM)을 사용한다. FCM은 애플리케이션의 키 값을 식별해 특정 스마트폰에 전달할 수 있다. 사용자가 애플리케이션을 설치한 후 알람설정에 동의면, FCM Token을 얻을 수 있고, 푸시알림을 고유한 키 값과 각 사용자들의 Token을 통해 이루어진다.

애플리케이션의 서버에서 Firebase 서버에 HTTP 통신을 통해 푸시 요청을 보내면 Firebase 서버에서는 전달받은 키 값을 처리하여 기기와 애플리케이션의 종류를 식별한다. 애플리케이션이 백그라운드 상태이면 푸시알림이 알림 목록으로 전송되고, 포어그라운드 상태라면 애플리케이션의 콜백함수가 메시지를 처리한다.

- a) 메시지/유지보수 등록
- b) 홈즈 서버에서 Firebase 서버에 HTTP 통신을 통해 푸시 요청
- c) Firebase 서버에서 고유 키 값 처리
- d) 기기와 애플리케이션 종류 식별
- e) 백그라운드, 포어그라운드에 대한 푸시알림 전송

(3) 알고리즘

비밀번호 암호화

```
const createHashedPassword = (plainPassword) => {
  new Promise(async (resolve, reject) => {
    const salt = await createSalt();
    crypto.pbkdf2(plainPassword, salt, 9999, 64, 'sha512', (err,
    key) => {
      if (err) reject(err);
      resolve({ password: key.toString('base64'), salt });
    });
  });
};
```

<createHashedPassword Function>

SHA(Secure Hash Algorithm)는 해시 함수들의 모음이다. 회원가입 시에 등록한 사용자의 비밀번호는 단방향 암호화 기법을 통해 안전하게 데이터베이스에 저장된다. crypto 모듈을 불러와 createHash 메소드를 사용한다. 이는 인자로 사용할 알고리즘을 넣어준다. 우리 프로젝트에서는 sha256보다는 더 길고 안전한 sha512를 사용하였다. update 메소드에는 암호화할 비밀번호를 넣어준다. digest에는 어떤 인코딩 방식으로 암호화된 문자열을 표시해준다. 이에 base64 방식을 사용하였다.

같은 알고리즘과 같은 인코딩 방식을 사용하면 같은 비밀번호에 대해 같은 결과를 반환한다. 이런 경우에 외부에서 공격자가 홈즈 서버의 암호화 방식에 대해 어떤 결과가 나올지 데이터베이스화 해두었다면, 암호화된 결과만 보고 원래 암호를 유추할 수 있다. 이러한 데이터베이스를 레인보우 테이블이라고 한다.

```
const createSalt = () => {
  new Promise((resolve, reject) => {
    crypto.randomBytes(64, (err, buf) => {
      if (err) reject(err);
      resolve(buf.toString('base64'));
    });
  });
};
```

<createSalt Function>

공격자가 레인보우 테이블을 사용하지 못하게 하기 위해 salt라는 특정 값을 통해 위에서 나온 결과를 변형할 수 있다. 홈즈는 사용자가 등록한 비밀번호에 공격자가 접근하지 못하도록 salt 문자열을 더한 결과를 수 만번 반복 해시화 한다. 즉, 비밀번호를 3차 암호화하여 아래와 같이 데이터베이스에 저장한다.

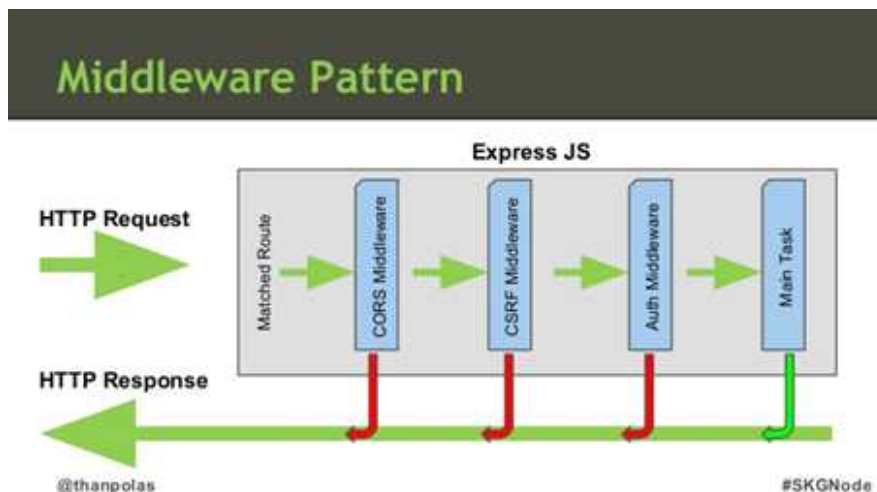
id	userId	password	salt
1	admin	IzQrGs1pKgDIzKRptVXzpwLTEP9ULb...	4k9Nm/SLD/gQFBEvKTyvA7z/IlvhiJ4D...
2	test	bYaJocwFgOrbk/L63READJqAkeKHTB...	+ISCC/ZnqPNscFVy+RqLL9Z9cKl0X/A...

<데이터베이스에 저장된 비밀번호>

미들웨어 패턴

서버는 요청에서부터 응답까지 하나의 흐름을 가지고 있다. 이 부분에서 이루어지는 요청과 응답 사이에 실행되는 함수 목록을 '미들웨어 함수'라고 한다. 미들웨어 함수는 크게 다음과 같은 두 가지 역할을 수행한다.

- 1) 요청한 클라이언트에 응답
- 2) 다음 미들웨어 함수를 호출



우리 프로젝트에서는 사용자 인증에 관한 부분을 미들웨어 패턴으로 처리하였다. 이렇게 하면 인증을 통과하지 못했을 때, 실제 서비스 로직에 접근하지 못하도록 막을 수 있다.

위와 같은 사용자 인증은 로그인 체크, 건물주/관리인/세입자 타입에 관한 체크에 적용되고, 인증이 완료되었을 때 다음 서비스 로직에 접근할 수 있다.

```
const checkLoginAndPush = (req, res, next) => {
  if (req.session.user) {
    const pushCountSql =
      'select count(msgID) as count from messages where receiver=? and isRead=0;';
    mysqlClient.query(pushCountSql, req.session.user.userId, function (err, row) {
      if (err) {
        next();
      } else {
        res.cookie('pushCount', row[0].count);
        next();
      }
    });
  }
}
```

<사용자 인증 예시 코드>

위 예시 코드는 홈즈 구현 소스코드 중 일부이다. 사용자 계정이 로그인되어 있을 시에 위 함수의 쿼리문을 실행할 수 있다. 쿼리문 수행 후에는 next를 통해 미들웨어를 순차적으로 처리한다.

(4) 구현 방법

Homes의 주요 기능에 대한 Backend 코드와 설명을 나열하였다.

▶ 건물주의 건물 통합 관리

```
const host = function (req, res) {
  const buildingInfoSql =
    'select b.buildingNum, (sum(payment_month_ok=0)) as nonPayment, b.building_name, v.Unsolved as Unsolved, name as manager_name from buildings b join user u on b.managerID=u.user_id join room r on r.buildNum=b.buildingNum join (select buildingNum, COUNT(repairNum) as Unsolved from buildings b join room ro on buildingNum=buildNum join repair re on re.roomID=ro.roomID where b.hostID=? and re.isSolved=0 group by buildingNum UNION select buildingNum, 0 as Unsolved from buildings where buildingNum not in ( select buildNum from room ro join repair re on ro.roomID=re.roomID where re.isSolved=0)) v on b.buildingNum=v.buildingNum where b.hostID=? and r.payment_type=0 group by buildingNum, building_name, Unsolved, manager_name;';

  mySqlClient.query(buildingInfoSql, [req.session.user.userId, req.session.user.userId], function (err, row, ) {
    // 건물주의 건물이 한개 이상 있는 경우
    if (row) {
      // 모든 소유 건물을 데이터에 저장
      const building_data = [];

      row.forEach((element) => {
        building_data.push(element);
      });

      res.render('host/host.html', {
        name: req.session.user.userName,
        building_data, // 건물 이름, 관리인 이름 데이터 배열
      });
    }
    // 건물주의 건물이 없는 경우
    else {
      res.render('host/host.html', {
        name: req.session.user.userName,
      });
    }
  });
};
```

buildings, room, tenant 테이블을 조인하여 UNION GROUP으로 본인이 소유한 건물의 모든 정보를 가져와 EJS로 렌더링하였다.

▶ 건물주의 월세/전세 관리

```
const paymentOkMsg = '완납 처리되었습니다.';
const paymentNotOkMsg = '미납 처리되었습니다.';

const changePaymentOk = async function (req, res) {
  const buildNum = req.query.bid;
  const roomNum = req.query.rid;

  const paymentOkSql = 'select payment_month_ok from room where buildNum=? and roomNum=?';
  const updatePaymentOkSql = 'update room set payment_month_ok=? where buildNum=? and roomNum=?';

  mysqlClient.query(paymentOkSql, [buildNum, roomNum], function (err, row) {
    if (row[0]) {
      const selectPaymentOk = row[0].payment_month_ok;
      if (selectPaymentOk === 1) {
        mysqlClient.query(updatePaymentOkSql, [0, buildNum, roomNum], function (err) {
          res.send(
            '<script type="text/javascript">alert("${paymentNotOkMsg}"); window.location="/host/aden/${buildNum}";</script>',
          );
        });
      } else {
        mysqlClient.query(updatePaymentOkSql, [1, buildNum, roomNum], function (err) {
          sendPushOfPaymentOk(buildNum, roomNum);
          res.send(
            '<script type="text/javascript">alert("${paymentOkMsg}"); window.location="/host/aden/${buildNum}";</script>',
          );
        });
      }
    }
  });
};
```

건물주가 특정 세입자에 대해 월세 납입정보의 변경을 요청하면 room 테이블의 payment_month_ok 컬럼의 데이터를 변경한다.

```
const setPaymentOkAndSmsCount = function () {
  const changePaymentOkAndResetSmsCountSql =
    'update room set payment_month_ok=0 where payment_month_day=? and payment_type=0; update user set smsCount=0;';
  mysqlClient.query(changePaymentOkAndResetSmsCountSql, date, function (err) {
    if (err) {
      console.log('Update Error>>' + err);
    } else {
      console.log(`${date} 일 payment_month_ok, smsCount 업데이트 완료`);
    }
  });
};

const timer = schedule.scheduleJob(rule, setPaymentOkAndSmsCount);
```

rooms 테이블의 월세 납입 정보는 월세 납입일을 기준으로 매일 각각 초기화된다. 이는 00시에 scheduleJob 메소드를 통하여 구현하였다. 따라서 매월 건물주가 세입자마다 다른 납입 정보를 초기화하지 않아도 서버 측에서 자동으로 초기화한다.

▶ 세입자의 본인 건물 정보, 월세/전세 납입 정보 확인

```
const tenant = function (req, res) {
  const tenantInfoSql = `select payment_month_day, payment_cash, payment_type, payment_month_ok, building_name,
  building_addr, roomNum, u.name as hostName, u.user_id AS hostID, u.tel as hostTel, bank_account, v.name as
  managerName, v.user_id as managerID, v.tel as managerTel, date_format(begin_date, '%Y-%m-%d') as begin_date,
  date_format(end_date, '%Y-%m-%d') as end_date from user u join buildings b on u.user_id=b.hostID join room r on b.
  buildingNum = r.buildNum join (select * from user) v on v.user_id=b.managerID where r.tenantID = ?`;
  mysqlClient.query(tenantInfoSql, req.session.user.userId, function (err, row) {
    if (row) {
      const tenant_data = {
        ...row[0],
      };

      res.render('tenant/tenant.html', {
        name: req.session.user.userName,
        ...tenant_data,
      });
    } else {
      res.render('tenant/tenant.html', {
        name: req.session.user.userName,
      });
    }
  });
};
```

세입자는 본인이 입주한 건물의 건물주, 관리인 정보와 주소, 입금 계좌, 월/전세 납입 정보 등의 데이터를 요청한다. DBMS에서는 tenant, room, buildings 테이블의 조인에 의해 데이터를 처리한 후 EJS 렌더링을 통해 View를 반환한다.

▶ 세입자의 유지보수 신청 (파일 업로드 포함)

```
const addRepair = function (req, res) {
  const roomID = req.body.roomID,
    title = req.body.title_content,
    content = req.body.repair_detail;

  const updateRepairSql = `insert into repair set ?`;

  const params = {
    roomID,
    title,
    content,
  };

  mysqlClient.query(updateRepairSql, params, function (err) {
    if (err) {
      console.log('Insert Err>>' + err);
      res.send(
        '<script type="text/javascript">alert("하자 등록 중 오류가 발생했습니다."); window.history.back();</script>',
      );
    } else {
      sendPushOfRepair(roomID);
      res.send(
        '<script type="text/javascript">alert("하자 등록이 완료되었습니다."); window.location="/tenant/repair_list";</script>',
      );
    }
  });
};
```

세입자가 유지보수에 대한 내용과 사진 파일을 POST 요청하면 사진 파일과 텍스트 내용을 분리하여 내용은 DBMS를 통해 repair 테이블에 삽입한다.

```

const upload = multer({
  storage: multer.diskStorage({
    destination(req, file, cb) {
      const currentDirPath = baseImgDir;
      //폴더 없으면 생성
      console.log(currentDirPath);
      mkdir(currentDirPath);
      cb(null, baseImgDir);
    },
    filename(req, file, cb) {
      var currentFileName;
      var selectLastIdSql = 'select * from repair ORDER BY repairNum DESC limit 1';
      //파일 이름: 최근(last) defect id + 1
      mysqlClient.query(selectLastIdSql, function (err, row) {
        if (err) {
          console.log('selectLastIdSql ERROR>>' + err);
        } else if (row[0]) {
          lastId = parseInt(row[0].repairNum);
          currentFileName = lastId + 1 + '.png';
          cb(null, currentFileName);
        } else {
          //하자 리스트에 아무것도 없는 경우
          console.log('last id 조회 결과 없음, id 1');
          currentFileName = '1.png';
          cb(null, currentFileName);
        }
      });
    },
  }),
  limits: {
    fileSize: 10 * 1024 * 1024, //img Limit: 10MB
  },
});

```

한편 유지보수에 대한 사진은 Node.js의 Multer 모듈을 통해 repair 테이블의 id와 동일한 파일 명으로 지정된 디렉토리에 저장한다. 파일을 업로드하는 과정과 텍스트 내용이 DB에 삽입되는 과정은 병렬적으로 처리된다.

▶ 세입자의 119 긴급 화재 신고

```
const to = '01049414921'; // 119 번호 입력
const content = `${address} ${room} 화재발생/신고자 ${tel}(앱신고)`;

mysqlClient.query(getSmsCountSql, req.session.user.userId, async (err, row) => {
  if (err) {
    res.send(failGetCountRedirect);
  } else {
    const smsCount = row[0].smsCount || 0;
    if (smsCount < 3) {
      const ncp = new NCPCClient({
        ...sensKey,});
      const { success, msg, status } = await ncp.sendSMS({
        to,
        content,
      });
      if (!success) {
        console.log(`(ERROR) node-sens error: ${msg}, Status ${status} Date ${Date.now()}`);
        res.send(failSmsRedirect);
      } else {
        mysqlClient.query(
          updateSmsCountSql,
          [smsCount + 1, req.session.user.userId],
          (err, row) => {
            // 세입자 Push 전송
            sendPushOfEmergency(bid, room, req.session.user.userId);
            res.send(successRedirect);
          });
      }
    }
  }
});
```

세입자가 119 긴급 화재 신고 버튼을 누르면 본인 건물의 정보와 함께 백엔드에서 POST 요청을 받는다. SMS 텍스트 데이터를 생성한 후 node-sens 모듈을 Naver Cloud Platform으로 문자 전송을 요청한다. 동시에 같은 건물의 모든 세입자에게 긴급 푸시 알림을 전송한다.

▶ 세입자의 건물 내 직거래 (판다)

```
const postAddProduct = function (req, res) {
  const seller = req.session.user.userId,
    title = req.body.title_content,
    content = req.body.detail,
    productState = 0;
  const insertSql = 'INSERT INTO panda SET ?';
  const selectBuildingNum = 'select buildNum from room where tenantId = ?';
  mysqlClient.query(selectBuildingNum, seller, function (err, row) {
    if (row) {
      const buildingNum = row[0].buildNum;
      const params = {
        buildingNum,
        seller,
        title,
        content,
        productState,
      };
      mysqlClient.query(insertSql, params, function (err) {
        if (err) {
          console.log('Insert err >>' + err);
          res.send(
            '<script type="text/javascript">alert("등록 중 오류가 발생했습니다."); window.history.back();</script>',
          );
        } else {
          res.send(
            '<script type="text/javascript">alert("상품 등록이 완료되었습니다!"); location.href="/panda";</script>',
          );
        }
      });
    } else {
      console.log(err);
    }
  });
};
```


세입자가 <판다>에서 상품 등록 시 백엔드 측에서 POST 요청을 받는다. 세입자의 유지보수 등록과 마찬가지로 텍스트 내용과 상품의 사진 파일을 분리한 후 텍스트 내용은 panda 테이블에, 사진 파일은 Multer 모듈을 통해 업로드하는 과정을 병렬 처리한다.

```
const getPanda = function (req, res) {
  const user_id = req.session.user.userId;
  const selectPandaList =
    'select title, productState, pandaId from room r, panda p where r.buildNum=p.buildingNum and r.tenantID=?
    order by pandaId DESC;';
  const pandaList = [];
  mysqlClient.query(selectPandaList, user_id, function (err, row) {
    if (row) {
      row.forEach((element) => {
        pandaList.push(element);
      });
      if (pandaList.length > 0) {
        res.render('panda/index.html', {
          pandaList,
        });
      } else {

```

세입자가 <판다>에 접속하면 백엔드 측에서 GET 요청을 받는다. 세입자의 ID로 DB에 접근하여 panda, room 테이블을 조인 후 입주한 건물 내의 상품 내역 데이터를 처리 후 EJS를 통해 렌더링한다.

▶ 관리인의 유지보수 관리

```
const host_getRepairsSql =
  'SELECT re.repairNum, re.title, re.isSolved, b.building_name, ro.roomNum FROM buildings b, repair re, room ro,
  user u WHERE re.roomID = ro.roomID AND ro.tenantID = u.user_id AND ro.buildNum = b.buildingNum AND hostID = ?
  order by repairNum desc';
const mgr_getRepairsSql =
  'SELECT re.repairNum, re.title, re.isSolved, b.building_name, ro.roomNum FROM buildings b, repair re, room ro,
  user u WHERE re.roomID = ro.roomID AND ro.tenantID = u.user_id AND ro.buildNum = b.buildingNum AND managerID = ?
  order by repairNum desc';

const loadRepairList = function (req, res) {
  let executeSql;
  const unsolved_repairs = [];
  const solved_repairs = [];

  if (req.session.user.userType === '건물주') {
    executeSql = host_getRepairsSql;
  } else if (req.session.user.userType === '관리인') {
    executeSql = mgr_getRepairsSql;
  }

  mysqlClient.query(executeSql, req.session.user.userId, function (err, row) {
    if (row) {
      row.forEach((element) => {
        if (element.isSolved === 0) {
          unsolved_repairs.push(element);
        } else {
          solved_repairs.push(element);
        }
      });

      res.render('common/repair_list.html', {
        userType: req.session.user.userType,
        unsolved_repairs,
        solved_repairs,

```

관리인(관리인이 없는 건물은 건물주가 관리인)이 유지보수 관리에 접속하면 백엔드 측에서 GET

요청을 받는다. 관리인(건물주)의 ID를 통해 room, tenant, repair, buildings 테이블을 조인하여 유지보수에 대한 데이터를 처리 후 EJS를 통해 렌더링한다.

▶ 이후 세입자의 유지보수 완료 처리

```
const solveRepairSql = 'update repair set isSolved = 1 where repairNum = ?';
mySqlClient.query(solveRepairSql, repairNum, function (err, row) {
  if (row) {
    res.send(
      '<script type="text/javascript">alert("완료 처리되었습니다!"); location.href="/view_repair/${req.params.id}';
    );
  } else {
    res.send(
      '<script type="text/javascript">alert("잘못된 DB 접근입니다."); location.href="/view_repair/${req.params.id}';
    );
  }
});
};
```

관리인이 유지보수를 처리하면 세입자는 유지보수 완료 버튼을 통해 POST 요청을 전송하여 repair 테이블의 isSolved(수리 완료) 데이터를 변경한다.

▶ 건물 내 구성원 간 소통 - 건물주(관리인)의 건물 전체 메시지 전송

```
const loadSendList_host = function (req, res) {
  const loadReceiverSql =
    'SELECT buildNum, building_name, roomID, tenantID, roomNum, name AS tenant_name FROM buildings b, room r, user u WHERE b.buildingNum=r.buildNum AND r.tenantID=u.user_id AND hostID = ?';

  const buildings_set = new Set();
  const buildings = [];
  const receivers = [];

  mySqlClient.query(loadReceiverSql, req.session.user.userId, function (err, row) {
    if (row) {
      row.forEach((element) => {
        if (!buildings_set.has(element.buildNum)) {
          buildings_set.add(element.buildNum);
          buildings.push({
            buildNum: element.buildNum,
            building_name: element.building_name,
          });
        }
        receivers.push(element);
      });

      res.render('common/host_mgr_message.html', {
        userType: req.session.user.userType,
        buildings,
        receivers,
      });
    } else {
      res.send(
        '<script type="text/javascript">alert("알림을 보낼 대상이 없습니다."); window.location="/function";</script>',
      );
    }
  });
};
```

건물주(관리인)가 알림 전송 기능에 접속하면 백엔드 측에서는 building, room, user 테이블을 조인한 결과를 본인이 소유(관리)하는 건물 리스트, 세입자 리스트의 데이터로 가공 처리 후 EJS를 통해 렌더링한다.

▶ 건물 내 구성원 간 소통 - 세입자 간의 메시지 전송

```
const loadSendList_tenant = function (req, res) {
  const loadReceiverSql =
    'SELECT tenantID, roomID, roomNum FROM buildings b, room r, user u WHERE b.buildingNum=r.buildNum AND r.tenantID=u.user_id AND tenantID!=? AND buildNum = any(SELECT buildNum FROM buildings b, room r, user u WHERE b.buildingNum=r.buildNum AND r.tenantID=u.user_id AND u.user_id=?);';
  const receivers = [];

  mysqlClient.query(loadReceiverSql, [req.session.user.userId, req.session.user.userId], function (
    err,
    row,
  ) {
    if (row) {
      row.forEach((element) => {
        receivers.push(element);
      });
      res.render('tenant/tenant_message.html', {
        receivers,
      });
    }
  });
}
```

세입자가 알림 전송 기능에 접속하면 백엔드 측에서는 building, room, user 테이블을 조인한 결과를 가공한 데이터를 EJS를 통해 렌더링한다. 이때, 세입자는 세입자의 이름(개인정보)을 제외한 호 정보만을 조회할 수 있다.

▶ 건물 내 구성원 간 소통 - 메시지 전송

```
// POST: 알림 전송 라우터 (+Firebase)
const sendPush = function (req, res) {
  const receivers = req.body.user_id; // 1명~여러명
  const sender = req.session.user.userId;
  const content = req.body.content;
  // msgType: 0 (가본 메시지, default로 설정되어 있음)

  const sendMessageSql = 'insert into messages (receiver, sender, content) VALUES ?';
  const params_msg = [];

  if (receivers && sender && content) {
    if (Array.isArray(receivers)) {
      // 다수인 경우
      for (var receiver of receivers) {
        params_msg.push([receiver, sender, content]);
      }
    } else {
      // 한명인 경우
      params_msg.push([receivers, sender, content]);
    }
  }

  const push_receivers = params_msg.map((p) => p[0]);
  let push_message;

  if (content.length > 15) {
    push_message = content.substr(0, 15).concat('..');
  } else {
    push_message = content;
  }

  mysqlClient.query(sendMessageSql, [params_msg], function (err, result) {
    if (err) {
      console.log('insert Error>>' + err);
      res.send(
        '<script type="text/javascript">alert("전송 중 오류가 발생했습니다."); window.location="/";</script>',
      );
    } else {
      console.log(push_receivers, push_message);
      sendPushOfMessage(push_receivers, push_message);
      res.send(
        '<script type="text/javascript">alert("알림을 보냈어요!"); location.href="/";</script>',
      );
    }
  });
};
```

건물주, 관리인, 세입자가 메시지 전송을 요청하면 백엔드 측에서는 POST 요청을 받는다. 전송하는 모든 메시지는 모듈화된 sendPush 라우터를 통해 메시지의 정보로 가공한 후 messages 테이블에 메시지 데이터를 삽입한다. 동시에 sendPushOfMessage 메소드를 통해 수신자에게 푸시 알림을 전송한다.

▶ 건물 내 구성원 간 소통 - 알림 메뉴에서 메시지 확인

```
const push = function (req, res) {
  const loadMsgSql =
    'SELECT msgID, sender, receiver, isRead, sendDate AS date, msgType, content, u.name AS sender_name, u.type AS sender_type, roomNum AS sender_roomNum, building_name AS sender_building_name, roomID as sender_roomID FROM (SELECT * FROM messages WHERE receiver=?) m JOIN user u ON m.sender=u.user_id left JOIN room r ON r.tenantID = m.sender left JOIN buildings b ON b.buildingNum=r.buildNum order by isRead asc, msgID desc;';
  const messages = [];

  mySqlClient.query(loadMsgSql, req.session.user.userId, function (err, row) {
    if (row) {
      row.forEach((element) => {
        messages.push(element);
      });

      res.render('common/push.html', {
        userType: req.session.user.userType,
        messages,
      });
    } else {
      console.log(err);
      res.send(
        '<script type="text/javascript">alert("올바른 DB 접근에 아닙니다."); window.location="/function";</script>',
      );
    }
  });
};
```

모든 종류의 사용자가 상단 알림 메뉴에 접속하면 백엔드 측은 GET 요청을 받는다. 본인의 ID를 통해 messages, building, user, room 테이블을 조인한 데이터의 결과를 메시지의 형태로 가공하여 EJS를 통해 렌더링한다.

▶ 푸시 알림

푸시 알림은 건물 구성원 간 메시지 전송, 유지보수 등록, 월세 완납 처리, 화재 알림의 이벤트가 발생할 때 동시에 요청된다.

```
// 메시지 to 수신자(공통)
const sendPushOfMessage = (receivers, message) => {
  const getTokenSql = 'SELECT token FROM user u WHERE u.user_id IN (?)';
  const tokens = [];

  mySqlClient.query(getTokenSql, [receivers], (err, rows) => {
    if (err) {
      console.error(err);
    } else {
      if (rows.length > 0) {
        rows.forEach((r) => tokens.push(r.token));
        handlePushTokens('새로운 메시지', message, tokens);
      }
    }
  });
};
```

건물 구성원 간 메시지 전송 이벤트가 발생하면, 수신자의 ID와 메시지 정보가 담긴 객체를 파라미터로 handlePushTokens 메소드를 호출한다.


```
// 유지보수 등록 to 건물주/관리인
const sendPushOfRepair = (roomId) => {
  const getTokenSql =
    'SELECT u.token AS host_token, v.token AS mgr_token, building_name, roomNum from user u join buildings b on u.user_id=b.hostID join room r on b.buildingNum = r.buildNum join (select * from user) v on v.user_id=b.managerID WHERE r.roomID = ?';
  const tokens = [];

  mySqlClient.query(getTokenSql, roomId, (err, rows) => {
    if (err) {
      console.error(err);
    } else {
      if (rows.length > 0) {
        const message = `${rows[0].building_name} ${rows[0].roomNum}호에 유지보수가 등록되었어요.`;
        if (rows[0].host_token === rows[0].mgr_token) {
          tokens.push(rows[0].host_token);
        } else {
          tokens.push(rows[0].host_token);
          tokens.push(rows[0].mgr_token);
        }
        handlePushTokens('유지보수 알림', message, tokens);
      }
    }
  });
};
```

유지보수 등록 이벤트가 발생하면 건물주와 관리인에게 유지보수 정보가 담긴 객체를 파라미터로 handlePushTokens 메소드를 호출한다.

```
// 완납처리 to 세입자 (1명) in host/host_aden_paymentok
const sendPushOfPaymentOk = (bid, rid) => {
  const getTokenSql =
    'SELECT token FROM user u, buildings b, room r WHERE u.user_id=r.tenantID AND b.buildingNum=r.buildNum AND buildingNum = ? AND roomNum = ?';
  const message = '월세가 완납처리 되었어요!';

  mySqlClient.query(getTokenSql, [bid, rid], (err, row) => {
    if (err) {
      console.error(err);
      return;
    } else {
      if (row.length > 0) {
        handlePushTokens('납부 알림', message, [row[0].token]);
      }
    }
  });
};
```

건물주의 월세 완납 처리 이벤트가 발생하면 완납 처리된 세입자를 수신자로 지정하여 월세 납부 정보에 대한 객체를 파라미터로 handlePushTokens 메소드를 호출한다.

```
// 화재 긴급 알림 (전체 세입자)
const sendPushOfEmergency = (bid, roomNum, reporter) => {
  const getTokenSql =
    'SELECT token, user_id FROM user u, buildings b, room r WHERE b.buildingNum=r.buildNum AND r.tenantID=u.user_id AND buildingNum=?';
  const tokens = [];

  mySqlClient.query(getTokenSql, bid, (err, rows) => {
    if (err) {
      console.error(err);
    } else {
      if (rows.length > 0) {
        rows.forEach((r) => {
          if (r.user_id !== reporter) tokens.push(r.token);
        });
        handlePushTokens('화재 경보!', `${roomNum}에서 화재 발생! 신속히 대비바랍니다`, tokens);
      }
    }
  });
};
```

세입자의 긴급 화재 신고 이벤트가 발생하면 같은 건물 내의 모든 세입자를 수신자로 지정하여 화재 발생 정보에 대한 객체를 파라미터로 handlePushTokens 메소드를 호출한다.

```
const handlePushTokens = (title, message, savedPushTokens, sound = 'default') => {
  let notifications = [];
  for (let pushToken of savedPushTokens) {
    if (pushToken === null || !Expo.isExpoPushToken(pushToken)) {
      continue;
    }
    notifications.push({
      to: pushToken,
      sound,
      title,
      body: message,
      data: {
        message,
      },
      channelId: 'default',
    });
  }
  let chunks = expo.chunkPushNotifications(notifications);
  (async () => {
    for (let chunk of chunks) {
      try {
        let receipts = await expo.sendPushNotificationsAsync(chunk);
        console.log(receipts);
      } catch (error) {
        console.error(error);
        return;
      }
    }
  })();
};
```

Firebase API에 푸시 알림 전송을 요청하는 handlePushTokens 메소드이다. 메시지 객체와 데이터베이스 user 테이블의 push token을 통해 수신자의 휴대전화로 푸시 알림을 전송한다. Push token은 아래와 같이 React Native 측에서 생성 후 Node.js 서버로 전송한다.

```
if (Platform.OS === "android") {
  Notifications.createChannelAndroidAsync("default", {
    name: "default",
    sound: true,
    priority: "max",
    vibrate: [0, 250, 250, 250],
  });
}

//node 서버로 토큰 전송
return fetch(PUSH_REGISTRATION_ENDPOINT, {
  method: "POST",
  headers: {
    Accept: "application/json",
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    token: this.tokenValue,
  }),
});
```

사용자의 휴대전화로 어플리케이션에 최초 접속하면 Firebase에 요청 후 Push token을 발행하여 백엔드 서버로 전송한다.

```
function tokenUpdate(token, id) {
  const setTokenSql = 'update `user` set token = ? where user_id = ?;';

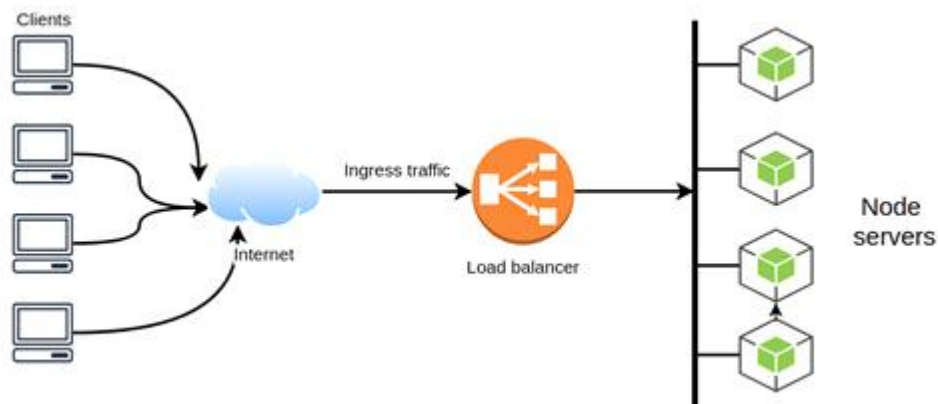
  //Token Update
  mysqlClient.query(setTokenSql, [token, id], function (err, row) {
    if (err) {
      console.log('update token error(token:${token}, user_id:${id}) -> ${err}');
      return true;
    } else {
      console.log('토큰 정상 업데이트: ' + token);
      return true;
    }
  });
}
```

백엔드 서버에서 전송받은 토큰 데이터는 사용자의 ID를 토대로 user 테이블에 업데이트된다.

(5) 실험 결과 및 성능 분석

홈즈 서비스에 대한 실험은 REST API 요청에 대한 임의 부하 실험, 데이터베이스 최적화 수행 후 MSSQL Server를 통한 cost 측정을 통해 실시하였다.

Microservices 아키텍처 구축 후 REST API 요청에 대한 임의 부하 실험



< Microservices의 구조 >

마이크로서비스 아키텍처는 어플리케이션을 Loose Services의 그룹으로 구조화하는 서비스 지향 아키텍처이다. 아키텍처의 서비스는 섬세하고 프로토콜은 가볍게 연결되어 있다. 어플리케이션을 마이크로서비스로 분해할 때 장점은 서비스를 모듈화하여 서버에서 전송받는 트래픽을 로드 밸런서를 통해 분산한다. 분할된 서비스는 데이터를 병렬적으로 처리 후 클라이언트에 반환함으로써 트래픽 부하를 현저히 줄일 수 있다.


```

Last login: Tue Dec  8 20:31:08 2020 from 220.149.255.14
root@homes-app:~# docker ps
CONTAINER ID   STATUS    IMAGE              COMMAND                  CREATED
NAMES
0258e4a02eda   Up 6 weeks   redis              "docker-entrypoint.s..." 6 weeks a
go             cocky_ritchie
55639a60932    Up 7 weeks   heojj97/homes-app:0.1 "docker-entrypoint.s..." 7 weeks a
go             homes-server
16f02bc53dd6   Up 8 weeks   heojj97/homes-mysql:0.1 "docker-entrypoint.s..." 8 weeks a
go             homes-database

```

1st server에서 docker ps 명령어 실행 결과

```

Last login: Tue Dec  8 20:31:57 2020 from 220.149.255.14
ubuntu@dku-sms-test:~$ sudo su
root@dku-sms-test:/home/ubuntu# docker ps
CONTAINER ID   STATUS    IMAGE              COMMAND                  CREATED
NAMES
19661cccab82   Up 11 days   redis              "docker-entrypoint.s..." 11 days ago
sms-redis
ce60d3469331   Up 13 days   heojj97/bluecheck:0.4 "docker-entrypoint.s..." 13 days ago
bc-app
78b300559662   Up 13 days   heojj97/mysql:latest "docker-entrypoint.s..." 13 days ago
bc-mysql

```

2nd server에서 docker ps 명령어 실행 결과

각 서버의 인스턴스 사양에 알맞게 서비스를 분배하였다. 백엔드(Node.js) 서버, 데이터베이스(MySQL) 서버, 캐싱(Redis) 서버 각 2개씩으로 나누어진다. 도커에서 각 서버는 컨테이너라고 명칭 한다. 동일한 유형의 요청이 클라이언트에서 전송되면 로드 밸런서를 통해 작업을 수행할 컨테이너를 분배한다.

```
[nodemon] app crashed - waiting for file changes before starting...
```

```

[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,html
[nodemon] starting `node ./bin/www.js`
internal/modules/cjs/loader.js:1032
  throw err;
  ^

```

기존 컨테이너 1개로 실행했을 때 10만 개의 부하 결과 App Crash 발생

```

TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE
HTTPS: Express listening on port 443
POSTMAN Test at [homes.nemobros.com]
... wait a moment
TEST SUCCEED - 300129 REST CALLS
Accuracy: 99.9892%
HTTP: Express listening on port 80

```

```

TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE
HTTPS: Express listening on port 443
POSTMAN Test at [homes.nemobros.com]
... wait a moment
TEST SUCCEED - 843042 REST CALLS
Accuracy: 99.9896%
HTTP: Express listening on port 80

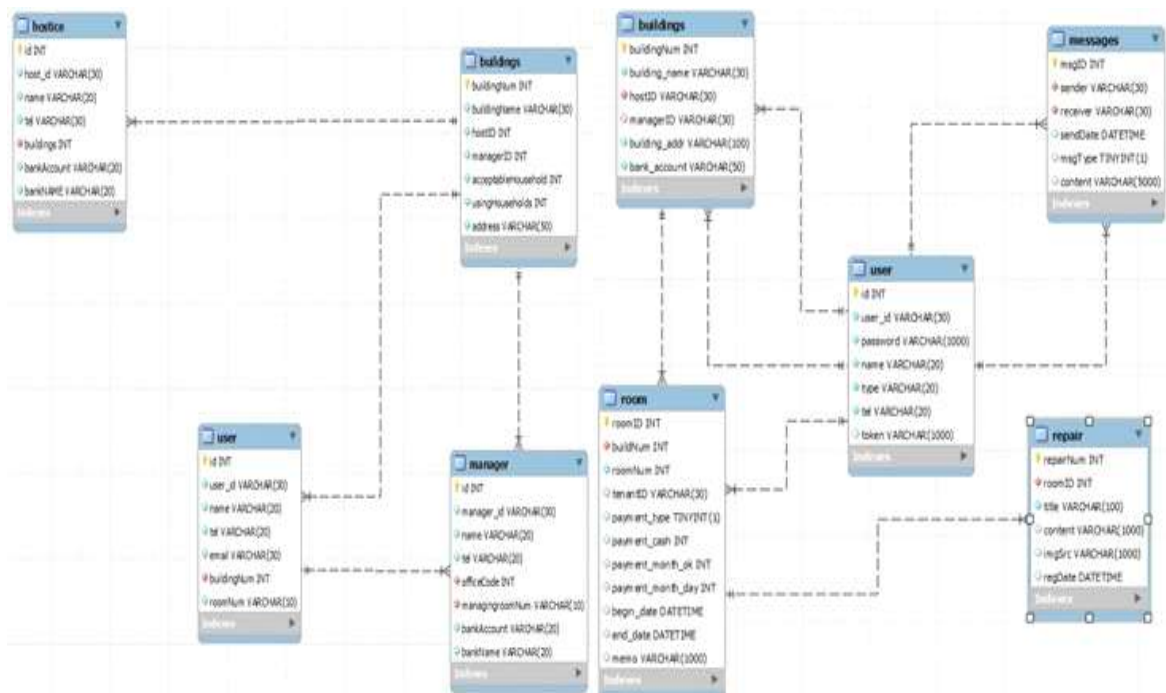
```

REST API 임의 부하 실험 결과(좌:Non-Microservices, 우:Microservices)

아무런 분산을 처리하지 않고 3개(백엔드, 데이터베이스, 캐싱)의 요청을 1개의 컨테이너로 받았을 때 10만 개의 동시 부하에서 App Crash가 발생했다. 이후 각각 1개의 컨테이너로 총 3개의 컨테이너 환경(Non-Microservices)에서는 약 30만 개의 REST Call을 정상적으로 처리했다. 최종적으로 각각 2개의 컨테이너로 총 6개의 컨테이너로 구성된 Microservices 아키텍처 환경에서는 80만 개가 넘는 REST Call 요청도 정상적으로 처리함을 볼 수 있다.

데이터베이스 최적화 수행 후 MSSQL Server를 통한 cost 측정

우선 홈즈 데이터베이스 최적화 과정은 반정규화를 통해 초기 프로토타입 데이터베이스를 현재의 데이터베이스로 최적화를 수행하였다.



왼쪽이 초기 프로토타입 데이터베이스 ERD이고 오른쪽이 현재 홈즈 데이터베이스 ERD다. 초기의 데이터베이스의 구조를 잠시 보면 유저(건물주, 건물주, 세입자)별로 테이블을 나누어 각각의 유저별로 필요한 속성들을 생성하였다. 그리고 건물 테이블을 두어 각 건물의 데이터를 저장함과 동시에 각각의 유저가 참조할 수 있도록 외래키를 지정하여 프로토타입을 사용하였다.

하지만 홈즈의 기능을 구조화하고 추가하면서 기존의 데이터베이스로는 한계를 느꼈다.

특히 각 테이블에 외래키를 사용하여 참조를 할 때 문제가 발생했는데, 추가된 기능에 맞게 테이블들을 추가로 구성하면서 각 테이블에 시퀀스 넘버를 지정하지 않아서 외래키 제약조건을 위반하였다. 그리고 유저가 건물주, 관리인, 세입자로 분류되어 있어서 과도한 조인이 발생하였다.

이를 해결하기 위해, 유저를 한 테이블로 뭉쳐서 재구성하였고 건물 정보를 저장할 빌딩 테이블, 각각의 건물의 호수별로 홈즈에서 제공하는 기능을 수행할 수 있도록 Room 테이블을 구현하고 각 유저들이 메시지를 주고 받을 때 필요한 Message 테이블과 건물 유지 / 보수를 관리할 Repair 테이블을 구현하면서 중복은 어느 정도 허용하면서 조인을 줄여주어 최적화를 꾀할 수 있는 반정규화 과정을 수행하였다. 그리고 각 테이블에 MySQL에서 지원하는 auto_increment 옵션을 사용하여 튜플의 시퀀스 넘버를 지정하여 외래키 제약조건에서 자유로워질 수 있었다.

최적화 후에 성능 평가 및 향상 작업을 진행하면서 MSSQL을 사용하여 인덱스 부분을 조정하였다. 현재 데이터베이스를 구현하면서 지정한 Primary Key가 Clustered Index로 지정된 것을 밑의 그림으로 확인할 수 있다.



Clustered Index와 NonClustered Index가 존재하여 단순 조회 성능 부분에선 이상이 없음을 발견했다. 하지만 여러 테이블을 조인하여 조회하는 상황은 자주 쓰는 속성에 인덱스를 생성하면 더 성능이 좋을 것으로 판단하여 진행하였다.

(1개 행 적용됨)
테이블 'room'. 스캔 수 1, 논리적 읽기 9, 실제 읽기 0, 페이지 서버 읽기 0, 미리 읽기 0
테이블 'users'. 스캔 수 1, 논리적 읽기 2, 실제 읽기 0, 페이지 서버 읽기 0, 미리 읽기 0

완료 시간: 2020-11-16T21:40:16.2390583+09:00

```

set statistics io on
SELECT u.users_id, count(*)
FROM users u join room r on (u.users_id=r.tenantid)
GROUP BY u.users_id, u.name;
set statistics io off
    
```

(1개 행 적용됨)
테이블 'room'. 스캔 수 1, 논리적 읽기 2, 실제 읽기 0, 페이지 서버 읽기 0, 미리 읽기 0
테이블 'users'. 스캔 수 1, 논리적 읽기 2, 실제 읽기 0, 페이지 서버 읽기 0, 미리 읽기 0

완료 시간: 2020-11-16T21:40:41.3007966+09:00

실제로 많이 조회가 되는 Room과 User테이블에 인덱스를 생성한 후, 조인을 시켜보니 성능 향상 효과가 있는 것을 알 수 있었다. 이 경우에 사용된 데이터베이스는 소량의 데이터만을 이용하여 큰 효과가 없는 것으로 보이지만 대량의 데이터가 저장된 경우라면 큰 성능 향상을 보일 것이다.

전 페이지의 증명으로 자주 사용하는 속성들에 NonClustered Index를 지정하였고 NonClustered Index를 지정함으로써 조회의 성능을 향상 시켰다.

밑은 MSSQL로 NonClustered Index를 지정해준 후의 모습이다.



3. 과제 수행 결과 내역

(1) 추진계획 및 실적

업무내용	담당자	9				10					11				12			
		1	2	3	4	1	2	3	4	5	1	2	3	4	1	2	3	4
자료조사	공통																	
System Design	공통																	
기능별 Coding	공통																	
Test	공통																	
중간보고서	공통																	
Testing 및 보완	공통																	
최종제작	공통																	
최종보고서	공통																	

월	주차	내용
9월	1주차	아이디어 회의 및 개발계획서 작성
	2주차	
	3주차	프로젝트 기초 설계, 구현 방법 구상
	4주차	데이터베이스 개념적 모델링, 앱 프로토타입 디자인
10월	5주차	협업 환경 구축 및 개발 환경 준비
	6주차	데이터베이스 구축, 프론트엔드/백엔드 설계, 테스트 서버 구축
	7주차	프론트엔드 구현, 백엔드-데이터베이스 연동
	8주차	프론트엔드/백엔드 구현, 배포 자동화, 데이터베이스 서버 구축
	9주차	클라우드 서버 배포, 어플리케이션 매핑
11월	10주차	1차 완성 - 어플리케이션 테스트
	11주차	클라우드 성능 분석, 데이터베이스 반정규화, 어플리케이션 최종 보완
	12주차	
	13주차	
12월	14주차	어플리케이션 완성 및 최종 보고서 작성

세부 목표	세부 계획	실 적	달성도(%)	사 유	참여자 성명
사용자의 타입에 따라 명확하게 어플리케이션 접근 가능	미들웨어 측에서 타입을 분류하여 View 제공	건물주, 세입자, 관리인 분류에 따라 기능에 접근하기에 무리가 없음	100%	모든 사용자가 분류에 맞게 각 기능에 접근 가능	팀원 전원
푸시 알림의 안정성 확립	각기 다른 대상에 푸시알림이 정확하게 전송	Firebase, React Native를 통해 token 발행 후 정확한 푸시 알림 전송	100%	알림이 대상에 정확하게 도달	팀원 전원
안정적인 서버 내 파일 저장(업로드)	저장된 사진 데이터 등이 안전하게 저장	Node.js의 Multer 모듈을 통해 사진 업로드를 병렬 처리	100%	대용량의 사진도 서버에 안정적으로 저장	팀원 전원
정확한 데이터의 발신/수신	REST API를 통해 명확한 URI 분류	Node.js의 Controller를 구축하여 REST 요청을 정확하게 발신, 수신	100%	정확하게 필요한 데이터만 수신	팀원 전원
서비스 보안 체계 확립	SSL 도입, 비밀번호 암호화, 인증 미들웨어 구현	비밀번호 3중 암호화, 사용자 권한 미들웨어 인증 처리, SSL 인증서 기반 HTTPS 프로토콜 구축	100%	서로 다른 권한에 접근 불가	팀원 전원

(2) 참여 인원별 역할 및 수행 소감

성명	소속	역할	참여도(%)
허전진	소프트웨어학과	FE/BE/DB 프로젝트 총괄 Server Infra 구축	25%
조정민	소프트웨어학과	Backend Server 구축	25%
이건욱	소프트웨어학과	Frontend Client 구현	25%
김승준	소프트웨어학과	Database 설계 및 구축	25%

▶ 허전진

첫 캡스톤디자인 팀의 리더를 맡았고 우여곡절 끝에 프로젝트를 끝냈다. 개발자에게 협업이라는 덕목의 필수 역량이다. 팀원의 의견을 수용하고 취합하는 과정에서 충돌이 일어나기도 하고, 프로젝트를 보다 완벽하게 완성하고 싶은 마음에 개발 일정을 뻑뻑하게 잡기도 했다.

팀원이 코드 작성에만 집중할 수 있는 환경을 제공하기 위해 큰 노력을 했다. Notion 협업 툴을 사용한 애자일 프로젝트 관리부터 시작해서 Shell script를 작성하여 서버 자동 배포 환경을 구축함으로써 코드 한 줄로 모든 서비스를 테스트할 수 있게 만들었다. 서버 인프라 구축에도 많은 신경을 썼다. 유저 기반의 서비스를 만든다는 것은 정밀한 서버 구축이 필요하고, Microservices, DevOps, Load balancing 등의 기술을 인프라에 구현하였다. 처음에 10만 개의 동시 요청도 서버에서 커버하지 못하다가 80만 개가 넘는 동시 요청도 커버하는 것을 실험하고 효율적인 서버 구축의 중요성을 뼈저리게 느꼈다. 또한, MySQL의 데이터 조회 속도가 마음에 들지 않아 프로젝트 완성 후에 Redis 캐싱을 구현하고 자주 발생하는 데이터 조회가 약 1000배 빨라졌다. 실무에서 사용하는 거의 모든 기술을 직접 적용해보니 뿌듯함이 컸다.

결론적으로 잘 따라와 준 팀원에게 감사하다. 홈즈의 전체 코드는 3만 줄이 넘는다. 혼자서는 절대 2달 안에 끝낼 수 없는 분량이다. 팀원 모두가 함께했기 때문에 궁극적인 목표에 도달할 수 있었다. 백엔드, 프론트엔드, 데이터베이스 역할 모두 서로 소통하며 의견을 공유하고, Github 협업 기능을 통해 효율적인 버전 관리를 했기에 성공적인 서비스가 만들어질 수 있었다. 다음 캡스톤디자인도 기대된다. 시급히 COVID-19 사태가 안정화되고 대면 수업에서 마지막 졸업작품을 장식하고 싶다.

▶ 조정민

처음 프로젝트 주제를 정할 때부터 어려움이 있었다. 실생활에서 많은 사람들이 불편함을 겪는 부분에서 우리가 해결해낼 수 있는 주제가 무엇일까 고민을 많이 했다. 팀원 네 명 모두 자취를 하고 있었고, 자취를 하면서 겪는 고초들이 공통점으로 도출되었다.

팀 이름을 정하고, 프로젝트에서 어떤 역할을 맡을지 정하는 과정도 어렵지 않았다. 각자 맡은 부

분에 대한 경험이 조금씩 있었기에 정확한 역할을 분담하여 맡는 것이 수월했다.

백엔드 구축을 맡은 나는 이전에 nodeJS를 사용한 간단한 실습예제를 진행했던 적이 있어 프로젝트를 진행하는 데에 큰 어려움을 겪을 것이라고 생각하지 않았다. 하지만 실제로 프로젝트를 진행해보니, 생각보다 어려운 부분이 많았다. 이런 상황이 생길 때마다 팀리더에게 도움을 받았고, 또, 백엔드 구축에 있어 필요한 코드를 검색하여 참조해 사용하기도했다. nodeJS에서 사용할 수 있는 여러 가지 모듈을 이번 팀프로젝트를 진행하면서 알게되었고, 내가 구현한 코드가 동적으로 웹페이지 상에 보여진다는 부분에 대해 흥미를 느꼈다. 기존에 프론트엔드 언어를 공부하면서 큰 흥미를 느끼지 못했는데, 백엔드 구축을 진행하면서 새로운 분야에 흥미가 생겼고, 진로를 정하는데 도움이 된 것 같아 뜻깊었다고 생각한다.

팀 프로젝트에 있어서 팀원들간의 소통이 중요하다고 생각했다. 백엔드 구축을 맡은 나는 프론트엔드를 맡은 팀원과도 소통을 자주 해야했고, 데이터베이스를 맡은 팀원과도 소통을 자주 해야했다. 처음에는 데이터베이스 역할을 맡은 팀원과의 소통이 잘 안되어 테이블명, 칼럼명 등이 통일될 수 없었다. 그래서 코드를 수정하거나 스키마를 다시 짜는 경우가 생기곤 했다. 이런 과정을 거치면서 팀프로젝트를 진행할 땐 소통이 중요하다는 것을 다시 한 번 느꼈다.

백엔드 구축이 어느 정도 마무리된 후, 내 역할은 끝났다고 생각했지만, 코드 리뷰를 통해 비효율적으로 작성된 코드의 수정이 필요하다고 느꼈다. 변수명의 특징을 정해 통일하거나 여러 파일에서 중복되는 함수는 클래스화 시켜 더욱 가독성이 뛰어나고 효율적인 코드로 수정할 수 있었다. 팀리더에게 이런 부분에 대한 도움을 많이 받았다. 미들웨어라는 용어의 개념조차 몰랐지만, 팀리더의 설명과 도움으로 미들웨어란 무엇인지 알게되었고, 해당 개념을 코드에 적용하여 더 깔끔하고 효율적인 코드를 작성할 수 있었다.

이번 캡스톤 디자인 팀 프로젝트를 한 학기동안 진행하면서, 기술적인 측면의 성장도 있었지만 무엇보다 팀 프로젝트를 진행할 때 한 팀의 구성원으로서 어떻게 행동해야 팀프로젝트를 더 수월하게 진행할 수 있는지 배울 수 있었다. 한 학기동안 백엔드 구축을 맡으며 기술적인 부분의 성장, 팀원으로서의 가져야하는 마음가짐에 대해 크게 배울 수 있는 뜻깊은 경험이었다.

▶ 이견육

이번 홈즈 프로젝트에서 내가 맡은 부분은 프론트엔드였다. 매번 프론트를 공부했지만 책의 예시나 이론, 간단한 한 페이지짜리 클론코딩 정도가 다였기에 비록 당연히 들어야 되는 캡스톤 디자인이지만 가장 좋은 공부가 될 수 있는 기회라고 생각하고 시작했다. 하지만 시작해보니 이론과 프로젝트는 역시 차이가 많았다. 유튜브와 각종 강의 영상을 찾아보며 실질적으로 많이 쓰이는 css 문법과 유용한 팁들을 많이 얻을 수 있었다. 물론 모든 분야에서 조원들에 비해 월등한 리더가 3명을 도와주었기 때문에 더욱 더 든든하게 역할에 임할 수 있었다.

큰 어려움은 없었지만 파일의 양이 많아지는 것을 최대한 줄여보려 template을 활용하려고 노력하였다. 그리고 class명을 처음에 너무 쉽게 생각하여 지은 것이 나중에 내가 css를 꾸미는데 어려움을 주어서 처음부터 class명을 주의깊게 지어야 한다는 것을 깨달았다. 또 많은 양의 코드를 작성함에 있어서 프론트엔드와 백엔드간의 호흡과 소통이 정말 중요하다는 것 또한 깨달았다.

이번 프로젝트를 통해서 왜 다들 취업을 위해선 프로젝트가 답이라고 하는지 알게되었다. 무엇보다 정말 많이 배우는 기회가 되어서 대학에 와서 수강한 과목중에 제일 나를 몰입시키는 과목이 되었다. 아직 남은 캡스톤 디자인에서는 다른 분야를 맡아 나를 또 한번 발전시키는 기회가 될 수 있도록 만들고 싶다.

▶ 김승준

건물 통합 관리 시스템에서 Database 설계 및 구축을 맡아 프로젝트를 진행하면서 내가 배운 지식들이 어플에서 어떤 식으로 동작하고 사용되는지 경험할 수 있었다.

우선 홈즈 프로젝트에 맞는 데이터베이스 테이블을 구축하는 단계에서 시행착오가 조금 있었다. 초기의 데이터베이스 구조는 단순히 유저의 권한(건물주, 관리인 세입자)에 따라 분류를 하고 건물 테이블을 따로 두어 진행을 하였다. 하지만 홈즈의 기능을 구조화하고 추가하면서 기존의 테이블의 한계점이 드러났는데, 각 테이블에 외래키를 사용하여 참조할 때, 데이터 조회, 삽입, 삭제 시 제약조건에 어려움을 겪었다.

이를 해결하기 위해 기능에 맞는 별도의 메시지 테이블과 리페어 테이블, 룸 테이블을 구축하였다. 그리고 각 테이블마다 auto_increment 옵션을 사용하여 시퀀스 번호를 자동 기입하게 하여 테이블 간 제약조건에서 좀 더 자유로워질 수 있도록 설계했다. 프로젝트에 참여하기 전까지는 막연하게 수업에서 들었던 auto_increment의 중요성을 몰랐지만 시퀀스 번호를 지정하는 것이 얼마나 효율적인지 직접 해보니 깨닫게 됐다.

이후의 전체적인 테이블 구조는 세부적으로 나누어 놔던 유저의 테이블들을 유저 테이블 하나로 통합하게 되면서 중복은 어느 정도 허용하지만 불필요한 Join을 줄여 실행 속도를 빠르게 하는 비정규화(Denormalization)를 하였다.

마지막으로 각 테이블이 참조할 때 발생하는 조인의 성능을 개선하기 위해 인덱스(INDEX)를 설정하였다. 보통 어플의 사용 패턴을 생각해봤을 때, 데이터 삽입, 삭제보단 조회가 더 활발하게 일어난다. 그래서 자주 쓰는 속성에 인덱스를 지정하여 많은 데이터가 구축이 되었을 때 성능 향상을 꾀할 수 있도록 설계를 하였다.

홈즈 프로젝트가 성공적으로 마무리되면서 앱에 구동에 관한 많은 것을 보면서 배웠고 데이터베이스 분야에 있어서도 기반을 다질 수 있는 좋은 기회가 되었다고 생각한다.

(3) 과제 결과물

▶ 메인페이지



▶ 회원가입



- 회원가입 시, 기입한 회원 정보가 데이터베이스에 저장. 비밀번호는 3차 암호화를 통해 저장

▶ 내 계정 설정

홈

알림

기능

관리

내 계정 설정

알림 설정

로그아웃

홈

알림

기능

관리

←

본인확인

소중한 개인정보 보호를 위한 본인확인을 진행합니다.

.....

확인

홈

알림

기능

관리

←

내 계정 관리

아이디

Capstone

회원유형

건물주

이름

캡스톤

전화번호

01012341234

수정

비밀번호

비밀번호

수정

- 환경설정-내 계정 설정에서 본인인증을 통해 전화번호 및 비밀번호 수정 가능

▶ 건물주-건물등록

홈

알림

기능

관리

←

건물 등록

건물의 주소를 알려주세요.

주소 검색

죽현로 59-10

16900

도로명

경기 용인시 기흥구 죽현로 59-10

지번

경기 용인시 기흥구 보정동 1248-4

12

센트레빌

Capstone

Capstone

국민 123-3838383-12383

홈

알림

기능

관리

캡스톤

건물주님,

안녕하세요!

내 건물 한눈에 보기

센트레빌

월세미남 4

관리인 캡스톤

- Daum 우편번호 API를 활용하여 건물의 정확한 주소기입 가능
- 건물에 대한 정보는 데이터베이스에 저장되고, 해당 정보가 ejs로 렌더링되어 출력

- 33 -

▶ 건물주-건물설정

홈 알림 기능 **관리**

←

건물명 **저장**

주소 **검색** **저장**

관리인 **저장**

세대 관리

일괄 설정 ☒ 월세 ☐ 전세 **저장**

금액(숫자) 입력 원

일괄설정시 모든 세대가 변경되므로 주의하세요!

101호 **저장**

세입자 ☐ 삭제

☒ 월세 ☐ 전세 0 원

월세 정산일 / 입주기간 0 일

연도-월-일 ~ 연도-월-일

- 관리-내 건물 설정으로 들어가 선택 건물에 대한 정보 수정 및
- 세입자 정보 추가/수정/삭제 가능

▶ 건물주-세입자 정보 및 월세 완납/미납처리

센트레빌

101호 월세 500000원 **미납**

캡스톤세입자 010-4321-4321 정산일 20일

메모 추가

완납 처리되었습니다.

확인

센트레빌

101호 월세 500000원 **완납**

캡스톤세입자 010-4321-4321 정산일 20일

메모 추가

센트레빌

101호 월세 500000원 **미납**

캡스톤세입자 010-4321-4321 정산일 20일

월세 2달 미납

- 건물주의 건물목록 클릭 시, 건물의 세입자 정보 확인 가능
- 한 번의 클릭으로 미납→완납, 완납→미납 처리 가능
- 각 세입자마다 메모 작성 가능

▶ 건물주, 관리인-유지보수 현황 확인

The left screenshot shows a user profile for '캡스톤 건물주님' (Capsstone Building Owner) with a '내 건물 한눈에 보기' (View my building at a glance) button. The right screenshot shows a '화장실 문 고장' (Toilet door broken) report for '센트레빌 101호' (Sentraville 101) with a photo of the door handle and a '세입자에게 메시지 보내기' (Send message to tenant) button.

- 세입자가 등록한 유지보수 현황 홈, 확인, 알림 란에서 확인 가능

▶ 세입자-건물 정보 및 월세 미납/완납 확인

The screenshot shows a tenant's profile for '캡스톤세입자' (Capsstone Tenant) with a '화재신고' (Fire Report) button. Below the profile is a card for '센트레빌 101호' (Sentraville 101) showing contact information, contract dates, and rental status.

- 홈 화면에서 건물, 건물주, 관리인 정보 확인 가능
- 월세 미납/완납 현황 확인 가능

▶ 세입자-유지보수 등록

홈

알림

기능

판다

←

유지/보수 등록

제목을 입력해주세요.

화장실 문 고장

하자의 사진을 업로드하세요.



사진 업로드

화장실 문이 잠기지 않아요

유지/보수 등록하기

홈

알림

기능

판다

←

내 유지보수 목록

모두 보기

진행중인 건

완료된 건

유지보수 신청

화장실 문 고장

⌚

홈

알림

기능


판다

←

화장실 문 고장

센트레빌 101호

캠스톤세입자 ☎ 010-4321-4321



화장실 문이 잠기지 않아요

유지보수가 완료되었어요

홈

알림

기능

판다

←

홈

알림

기능

판다

←

화장실 문 고장

센트레빌 101호

캠스톤세입자 ☎ 010-4321-4321

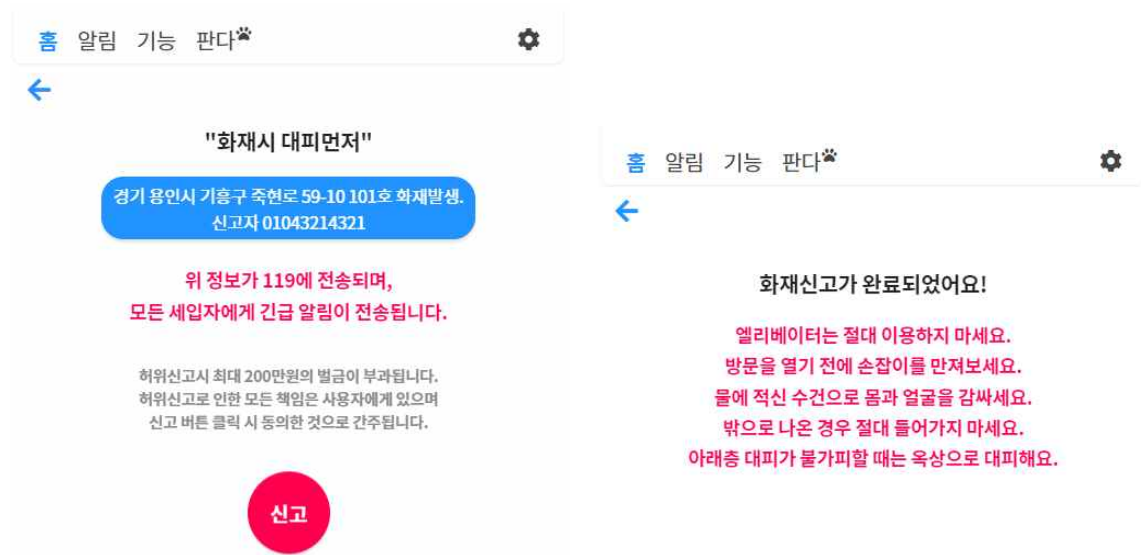


화장실 문이 잠기지 않아요

해결 완료된 건이에요

- 기능-유지/보수관리에서 유지보수 신청 및 현황 가능
- 하자 사진 및 내용 등록
- 유지보수가 완료되면 완료버튼으로 유지보수 현황 변경

▶ 세입자-119 긴급 화재신고



- 홈에 있는 화재신고 버튼으로 신속하고 빠르게 건물 내 화재신고 가능
- 신고자 번호, 건물 주소가 신고 즉시 119에 전송

▶ 세입자-건물 내 직거래 중고장터 Panda

홈 알림 기능 **판다**

우리 건물 정터,
판다

상품 등록하기



우리 건물 첫 판다를 등록하세요!

홈 알림 기능 **판다**

←

상품 등록

제목을 입력해주세요.
에어팟 팔아요!

상품의 사진을 업로드하세요.



사진 업로드

사용기간 : 2개월
가격 : 80000원

상품 등록하기

홈 알림 기능 **판다**

우리 건물 정터,
판다

상품 등록하기


에어팟 팔아요!

판매중

←

에어팟 팔아요!

101호 2020-12-09 00:20:13



사용기간 : 2개월 가격 : 80000원

판매 완료로 변경하기

- 같은 건물의 세입자들 간의 직거래 중고장터 기능
- 판매하고 싶은 물건, 공유하고 싶은 물건을 같은 건물 세입자끼리 직거래로 교환
- 판매 완료 항목은 판매 완료 버튼으로 상태 변경 가능

▶ 공용-메시지 기능

홈 알림¹ 기능 관리 

←

알림 보내기


수신자 목록
홈즈에 등록된 사용자만 표시됩니다.


✓ 전체선택

- ☒ 101호 캠프톤세입자님
- ☒ 102호 캠프톤세입자2님

음성인식으로 입력하기


월세 입금 부탁드립니다~!




홈 알림¹ 기능 판다² 

모두 읽음 처리 읽은 알림 삭제

건물주 캠프톤님이 메시지를 보냈어요. [읽음](#)

 월세 입금 부탁드립니다~! [답장](#)

2020-12-09 00:27:20

홈 알림¹ 기능 판다² 

←

알림 보내기


✓ 전체선택

내 건물의 수신자 목록

- ☒ 102호

음성인식으로 입력하기

너무 시끄럽네요! 조용히 해주시면 감사하겠습니다!



홈 알림² 기능 판다² 

모두 읽음 처리 읽은 알림 삭제

센트레빌 101호 세입자 캠프톤세입자님이 메시지를 보냈어요. [읽음](#)

 너무 시끄럽네요! 조용히 해주시면 감사하겠습니다! [답장](#)

2020-12-09 00:29:15

건물주 캠프톤님이 메시지를 보냈어요. [읽음](#)

 월세 입금 부탁드립니다~! [답장](#)

2020-12-09 00:27:20

- 건물 구성원 모두 메시지 기능 사용가능
- 개인 정보 미 노출, 각 호수 명으로 메시지 전송

(4) 활용 방안

우리 프로젝트 ‘홈즈’ 어플리케이션은 건물에 대한 관리의 간편화, 건물주/세입자/관리인 간 소통 활성화, 세입자에 대한 건물주의 관리 간편화, 세대 유지/보수에 대한 빠른 대처 등이 주요한 목표라고 할 수 있다.

이에 따라 우리는

- 건물에 대한 관리의 간편화 -> 한눈에 볼 수 있는 나의 건물 리스트
- 건물주/세입자/관리인 간 소통 활성화 -> 알림 보내기 기능
- 세입자에 대한 건물주의 관리 간편화 -> 내 건물의 세대 리스트
- 세대 유지/보수에 대한 빠른 대처 -> 유지/보수 등록 및 목록화

로 우리들의 목표를 해결하였다.

따라서 ‘홈즈’의 사용자들은 휴대폰 하나로

- 건물주: 각각의 세대에 대한 월세 여부를 하나하나 확인 후 기록
모든 세대에게 공지 전송 시 연락처 확인 후 하나하나 전송
- 세입자: 유지/보수에 대한 요청 시 건물주 연락처를 찾은 후 연락
-> 건물주가 부재 시 재 연락 혹은 망각하기 일수
소음문제로 인한 대면 해결 시도 시 마찰 발생 가능성
화재발생 시 대피 후 119 연락 시 시간지연 가능성
- 관리인: 유지/보수를 위한 방문 시 필요물품을 구비하지 않아 재방문 가능성

이러한 모든 문제를 해결할 수 있다.

4. 후기

이번 홈즈 프로젝트에서 처음으로 어플 개발을 기획하고 설계해봤다. 학과 수업에서 배우고 다듬은 실력이 개발에서 어떻게 통할지 궁금하기도 했고 한편으로는 두렵기도 했다. 하지만 리더가 하나의 큰 틀을 구성하고 팀원들은 각자의 톱니바퀴가 되어 서로 맞물려 원활하게 진행됐다.

홈즈의 궁극적인 목표는 집에서 발생하는 사회적 문제의 근본적인 해결이다. 이 목표를 달성하기 위해 많은 자문을 구하고 관련 이슈들을 찾아보며 소통의 부재가 근본적인 원인이라는 것을 깨닫고 건물 내 구성원 간의 원활한 소통 환경을 제공하기 위해 노력했다.

데이터베이스 설계/구축/보완 작업은 처음 프로토타입을 구성할 때까지 수많은 릴레이션을 생성했다 지웠다 하는 과정을 반복했었다. 이를 통해 무작정 눈에 보이는 기능들을 하나하나 끼워 넣는 것보다 건물 통합 관리 시스템이라는 큰 틀을 이해하고 테이블들을 설계하는 과정이 필요하다는 것을 인지하고 필요한 속성들을 정리하며 불필요한 속성들을 정리할 수 있었다. 이런 과정을 겪고 난 후, 홈즈의 추가 기능에 맞춘 데이터베이스를 설계하는 과정에서도 별 무리 없이 진행할 수 있었다.

백엔드 서버 구축에서는 다양한 라이브러리들이 활용되었다. Node.js의 장점을 살려 유용한 npm 패키지들을 모색하고 빠르게 홈즈 프로젝트에 녹이기 위해 노력했다. 자바스크립트의 비동기적 실행의 특징을 살려 병렬적으로 처리할 수 있는 작업, 동기적으로 처리해야 하는 작업을 분리하여 스레드 처리의 효율성을 높였다. REST API 컨트롤러를 직접 구축하여 URI 개념에 대해 깊이 학습했고 완벽한 CRUD 처리 환경을 만들 수 있었다.

프론트엔드 구현은 사용자가 접근하기에 어려움이 없는 UI를 만들기 위해 노력했다. 또한 프론트엔드에서 백엔드 서버 측에 필요한 데이터만을 요청하고 전송받기 위해 일반 HTML 페이지를 구축하는 것이 아닌 동적인 EJS 렌더링 페이지를 구축했다. 이로 코드의 반복을 피하고 알맞은 디자인 패턴을 준수할 수 있었다.

이 모든 DevOps를 Docker를 통해 서버 인프라로 구축했다. 분산된 컨테이너를 적절하게 로드 밸런싱하여 트래픽 부하를 최소화했다. 서비스의 모든 컨테이너를 자동으로 구축하고 업데이트하기 위해 Shell script를 작성하여 배포의 효율성을 최대화하기 위해 노력했다.

홈즈 프로젝트를 통해 팀원 모두가 실무에 적용되는 개념을 직접 구현해볼 수 있었다. 이번 캡스톤디자인을 통해 무형의 서비스로 사회 문제를 해결할 수 있다는 것에 팀원 모두가 뿌듯함을 느꼈고 협업이 개발자에 있어 얼마나 중요한가를 깨달았다. 각 톱니바퀴가 완벽하게 맞물릴 때 완성도 있는 서비스가 만들어질 수 있다. 팀 프로젝트는 이런 부분에서 참 매력적이다.