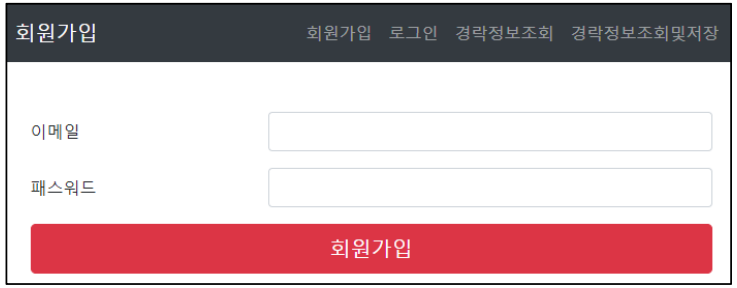
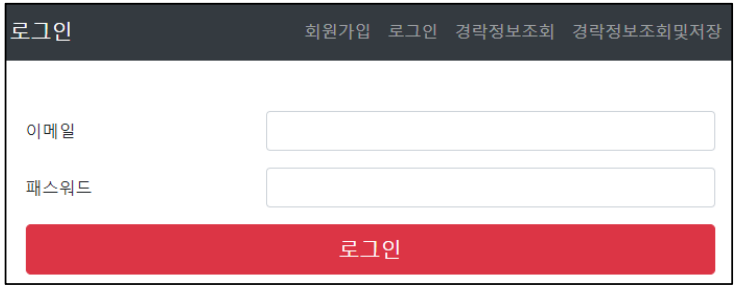
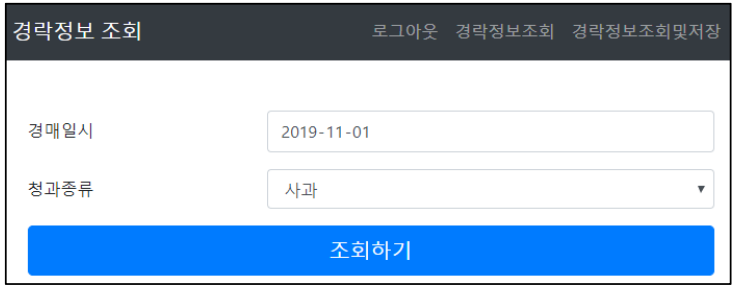
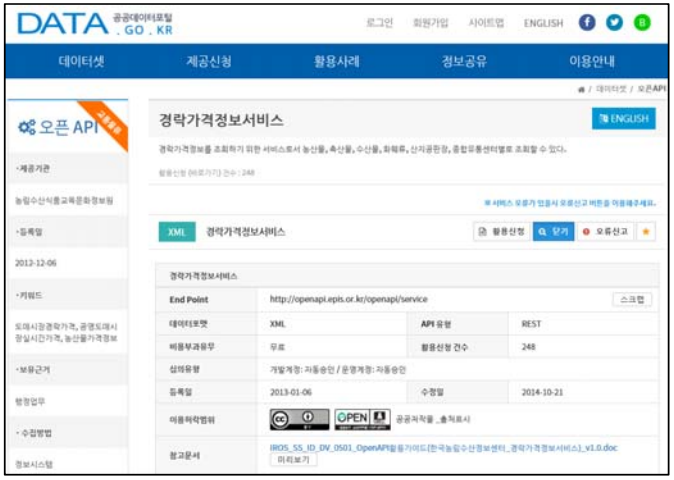
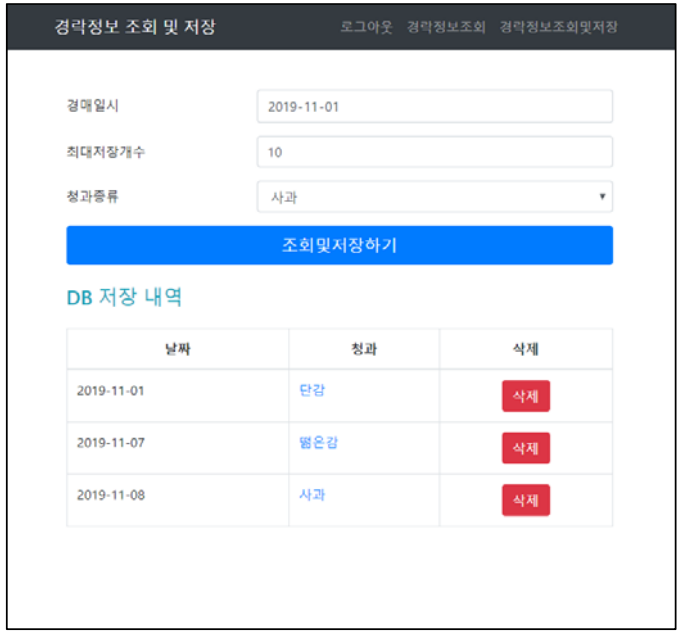
REST API를 이용한 공공데이터포털
경락가격정보서비스

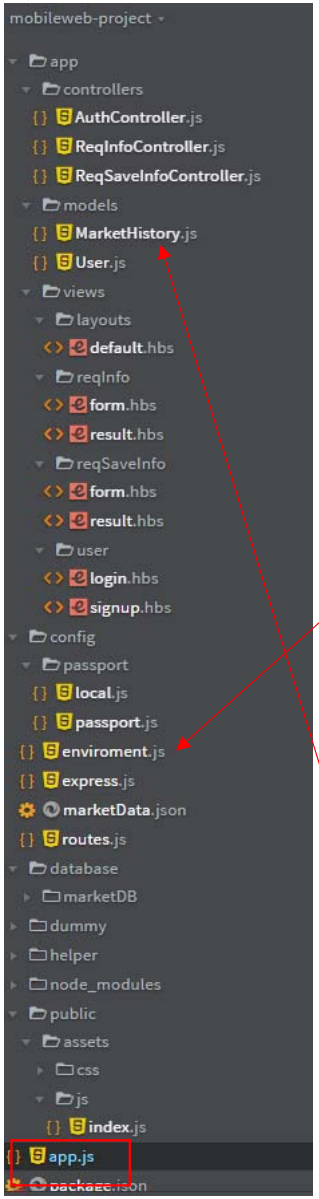






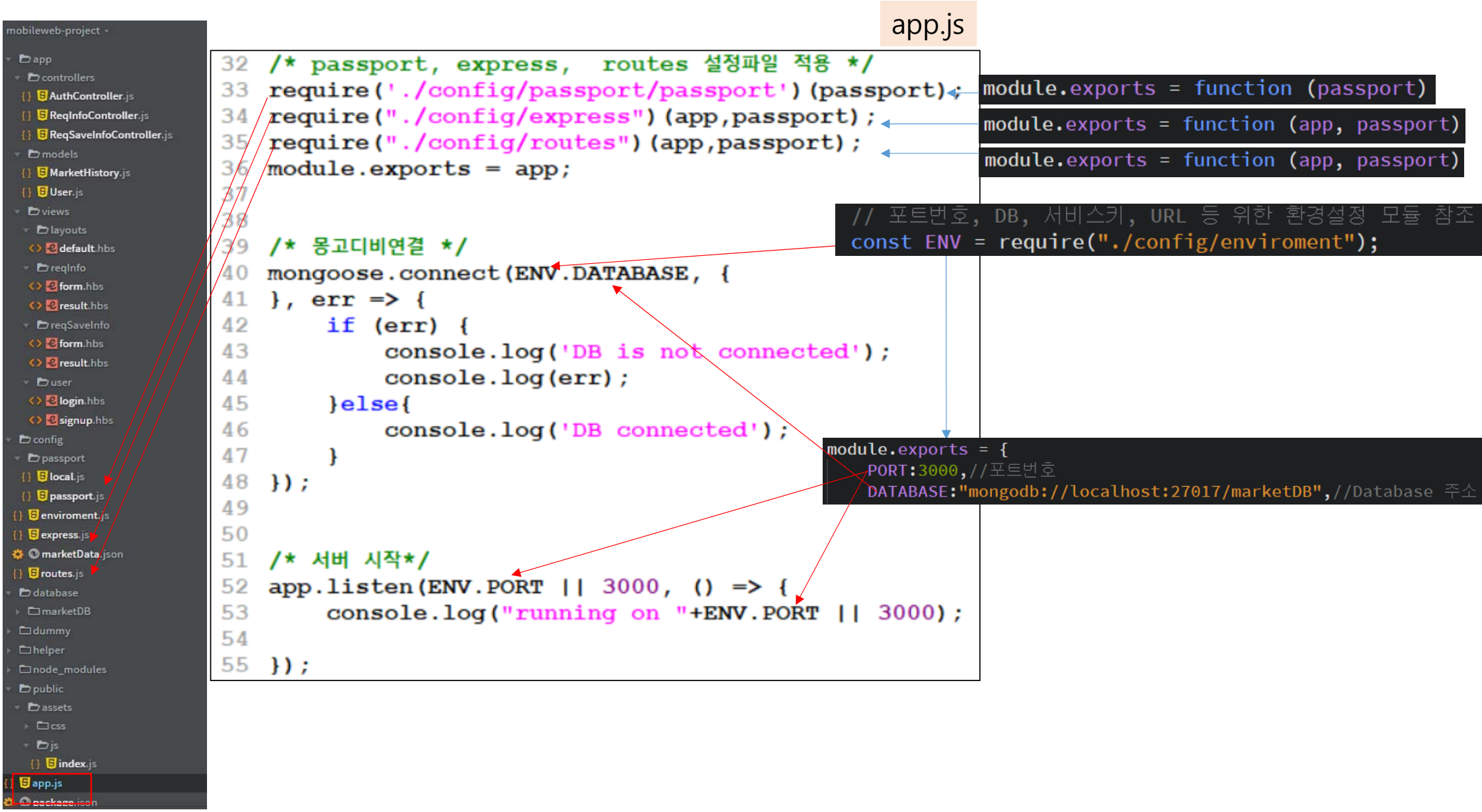


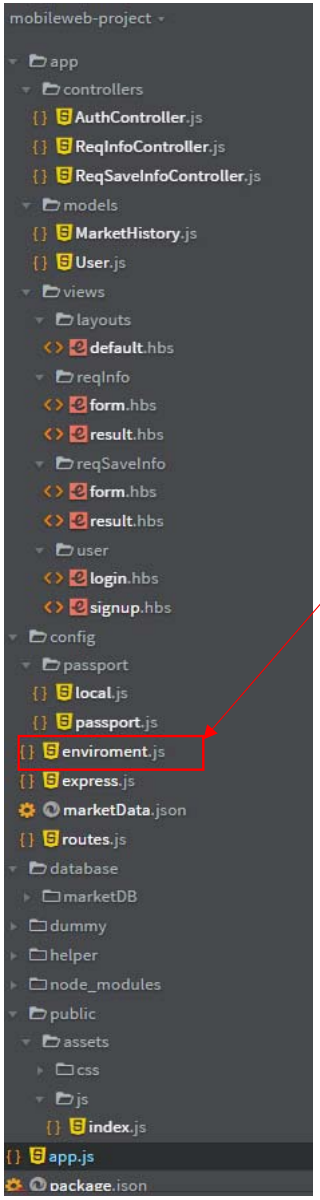




app.js

```
1  /* ===== Main program(app.js) ====*/
2
3  console.log('call : app.js');
4
5  // 필요한 모듈 참조
6  const fs = require('fs');
7  const express = require("express");
8  const mongoose = require("mongoose");
9
10 //==== 패스포트 모듈 참조(1) ======//
11 //사용자 인증 처리에 필요한 기본 기능을 제공하는 모듈
12 const passport = require('passport');
13
14 // 포트번호, DB, 서비스키, URL 등 위한 환경설정 모듈 참조
15 const ENV = require("./config/enviroment");
16
17 //express 객체 생성
18 const app = express();
19
20 /* 'app/models' 폴더 path 설정 */
21 const {join} = require('path');
22 //== const join = require('path').join;
23 const models = join(__dirname, 'app/models');
24
25 /* models 폴더에 있는 모델 설정파일(.js)을 전부 읽어서
26    해당 모듈 참조
27 */
28 fs.readdirSync(models)
29   .filter(file => ~file.search(/^[^\.].*\.js$/))
30   .forEach(file => require(join(models, file)));
```


app.js
mobileweb-project
app
controllers
AuthController.js
ReqInfoController.js
ReqSaveInfoController.js
models
MarketHistory.js
User.js
views
layouts
default.hbs
reqInfo
form.hbs
result.hbs
reqSaveInfo
form.hbs
result.hbs
user
login.hbs
signup.hbs
config
passport
local.js
passport.js
enviroment.js
express.js
marketData.json
routes.js
database
marketDB
dummy
helper
node_modules
public
assets
css
js
index.js
app.js
package.json
/* passport, express, routes 설정파일 적용 */
require('./config/passport/passport')(passport);
require("./config/express")(app,passport);
require("./config/routes")(app,passport);
module.exports = app;
/* 몽고디비연결 */
mongoose.connect(ENV.DATABASE, {
}, err => {
if (err) {
console.log('DB is not connected');
console.log(err);
}else{
console.log('DB connected');
}
});
/* 서버 시작*/
app.listen(ENV.PORT || 3000, () => {
console.log("running on "+ENV.PORT || 3000);
});
module.exports = function (passport)
module.exports = function (app, passport)
module.exports = function (app, passport)
// 포트번호, DB, 서비스키, URL 등 위한 환경설정 모듈 참조
const ENV = require("./config/enviroment");
module.exports = {
PORT:3000,//포트번호
DATABASE:"mongodb://localhost:27017/marketDB",//Database 주소



environment.js

```
console.log('call : /config/environment.js');

module.exports = {
    PORT:3000,//포트번호
    DATABASE:"mongodb://localhost:27017/marketDB",//Database url
    SERVICEKEY:'zMtFz5%2F0QfkhM90JcceDwlBZeSqCcyW1ugtJqIblSwOMr0FP                    3gxmk
    MONGO_SESSION_COLLECTION_NAME:"sessions",
    SESSION_SECRET:"your_secret", //세션 암호화에 사용할 값
    API_URL:"http://openapi.epis.or.kr/openapi/service/PcInfoService/getFrmprdPrdlstPcList",
    PAGINATION:{
        PAGE_SIZE:10
    }
};
```

DATA 공공데이터포털 .GO.KR

서비스정보

일반 인증키 (UTF-8)	zMtFz5 …wlBZeSqCcy 3gxmk …%3D 복사

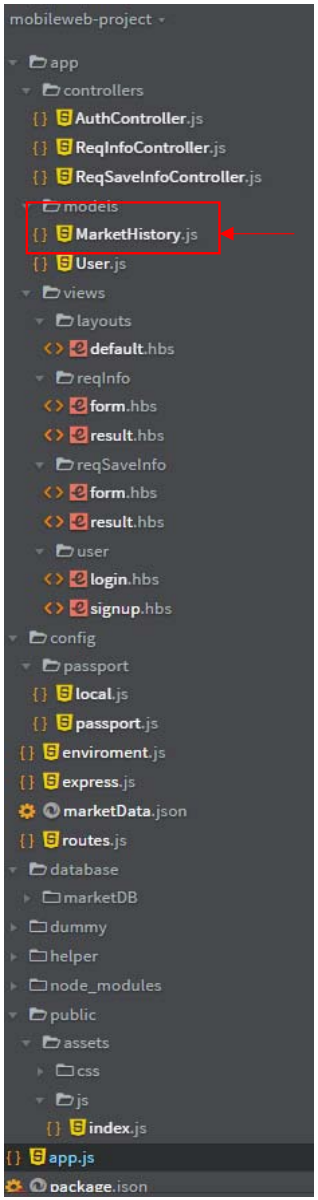
▪ 3.1. 경락가격정보서비스

▪ 가. 서비스 개요

서비스 정보	서비스 ID	SC-SD-AF-OP-001
	서비스명(국문)	경락가격정보서비스
	서비스명(영문)	PcInfoService
	서비스 설명	경락가격정보를 조회하기 위한 서비스로서 농산물, 축산물, 수산물, 화훼류 서비스
서비스 제공자정보	기관명	농림수산정보
서비스 보안	서비스 인증/권한	[ ] 서비스 [ ] Basic
	메시지 레벨 암호화	[ ] 전자서명
	전송 레벨 암호화	[ ] SSL
적용 기술 수준	인터페이스 표준	[ ] SOAP 1.1 (RPC-Encoded, Document Literal, Document Literal Wrapped) [ ] REST (GET, POST, PUT, DELETE) [ ] RSS 1.0 [ ] RSS 2.0 [ ] Atom 1.0 [ ] 기타
	교환 데이터 표준	[ ] XML [ ] JSON [ ] MIME [ ] MTOM
서비스 URL	개발환경	http://openapi.epis.or.kr/openapi/service/ PcInfoService
	운영환경	http://openapi.epis.or.kr/openapi/service/ PcInfoService
서비스 WADL	개발환경	http://openapi.epis.or.kr/openapi/service/ PcInfoService?_wadl
	운영환경	http://openapi.epis.or.kr/openapi/service/ PcInfoService?_wadl

▪ 나. 오퍼레이션 목록

일련번호	서비스명(국문)	오퍼레이션명(영문)	오퍼레이션명(국문)	메시지명(영문)
1.		getFrmprdPrdlstPcList	농산물품목별경락가격조회	FrmprdPrdlstPcRequest

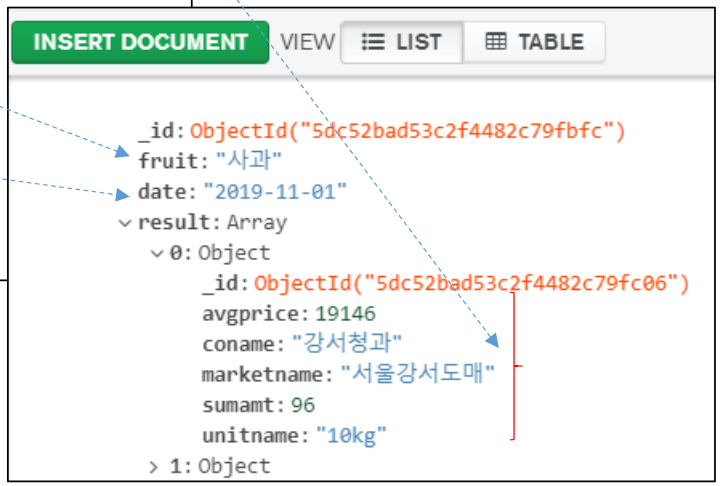


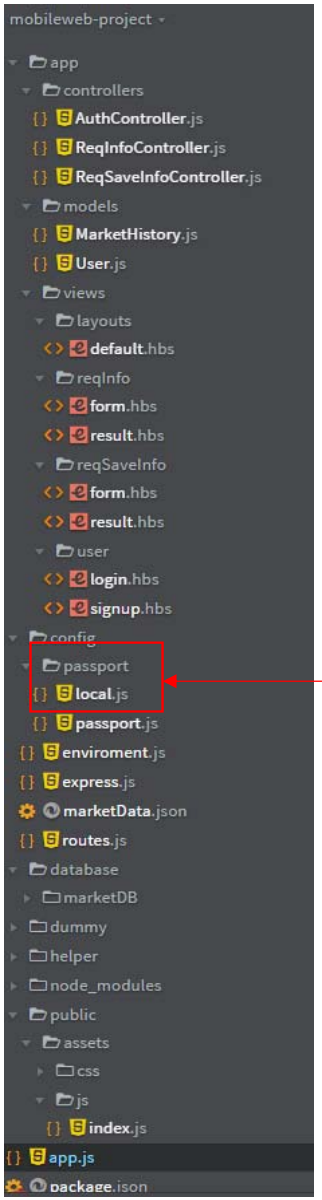
marketHistory.js

```
/* 경락가격정보 스키마 선언 및 모델 정의 */

const mongoose = require('mongoose');
const Schema = mongoose.Schema;

console.log('call : /models/MarketHistory.js');

/* MarketHistory 스키마 정의 */
const MarketHistory = new Schema({
    fruit: {type: String, required: true}, //경락정보 품종(과일종류)
    result:[{//응답결과
        marketname: {type: String, required: true},//도매시장
        coname: {type: String, required: true},//도매법인
        avgprice: {type: Number, required: true},//평균가
        sumamt: {type: Number, required: true},//거래량
        unitname: {type: String, required: true},//규격
    }],
    date: {type: String, required: true} //경락정보 날짜
}, {
    timestamp: true
});

//모델 생성
mongoose.model('MarketHistory', MarketHistory);
```

2019-11-01일 / 사과 도매경매 현황

장소	업체명	평균가	수량	기준
서울강서도매	강서청과	19146	96	10kg



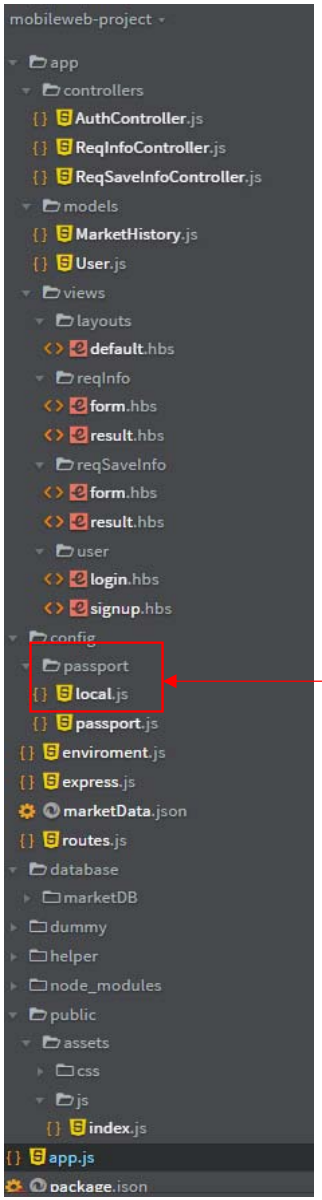


local.js

```
/* 로컬 인증 방식으로 스트래티지 설정 모듈 */
console.log('call : /config/passport/local.js');

const mongoose = require('mongoose');

// ========== 패스포트 Strategy(인증방식) 설정(8) ==========//
// ('passport-local').Strategy 참조
// 사용자의 이메일과 비밀번호를 전달받아 db에 저장된 정보와 비교하는 로컬 인증 방식 기능 제공
const LocalStrategy = require('passport-local').Strategy;

// User 모델 참조
const User = mongoose.model('User');

/**
 * login.hbs에서 입력받은 요청 파라미터 값으로 User의 아이디와 비밀번호를 인증(LocalStrategy 쪽으로 인증 처리를 위임)
 */

//첫 번째 파라미터{객체} 설명
/* login.hbs의 Form Data에서 name으로 설정한 값과 패스포트 내에서 사용할 값을 맵핑시켜준다.
   - <input name="user_email" class="form-control" type="email"/>
   - <input name="user_password" class="form-control" type="password"/>
   - 패스포트의 요청파라미터는 기본적으로 username과 password로 받도록 되어 전제되어 있기때문에
     요청파라미터 "user_email"을 usernameField로, "user_password"를 passwordField로 변경 */

// 로컬 인증 방식으로 스트래티지 설정
module.exports = new LocalStrategy({ //첫번째 파라미터
        usernameField: 'user_email',
        passwordField: 'user_password',
    },
```

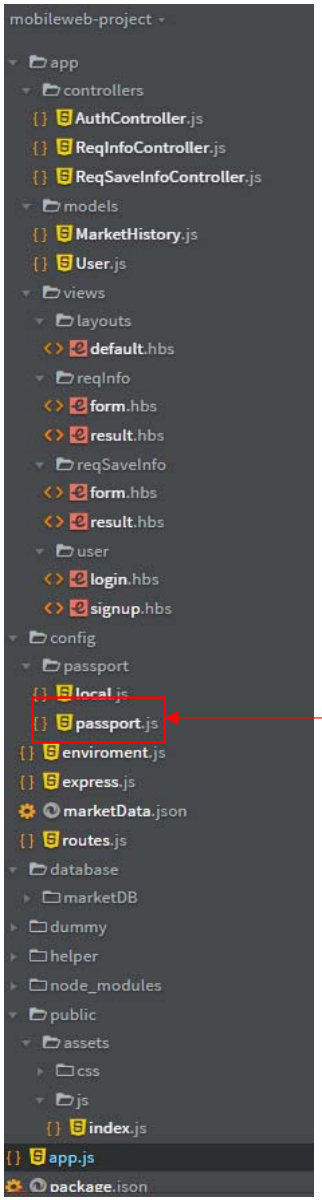
login.hbs

```
<div class="form-group row">
    <label for="example-date-input" class="col-4 col-form-label">이메일</label>
    <div class="col-8">
        <input name="user_email" class="form-control" type="email"/>
    </div>
</div>
<div class="form-group row">
    <label for="example-date-input" class="col-4 col-form-label">패스워드</label>
    <div class="col-8">
        <input name="user_password" class="form-control" type="password"/>
    </div>
</div>
```



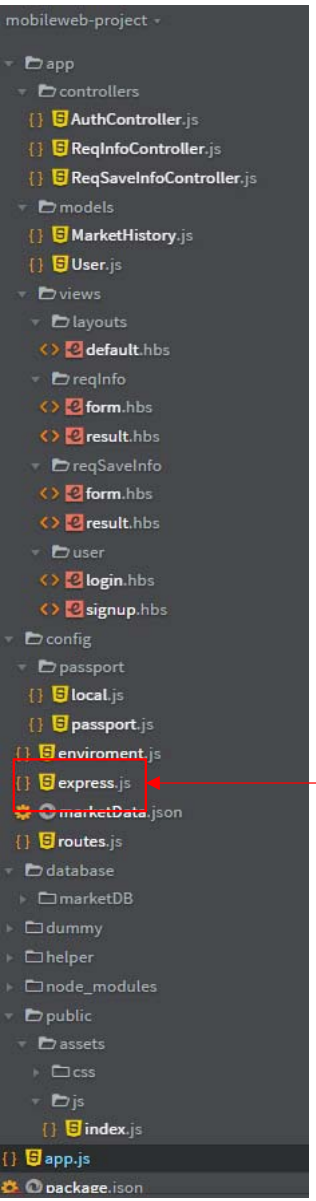
local.js

```
30      //클라이언트의 요청파라미터를 받아서 처리
31      async function (email, password, done) {//두번째 파라미터:검증 콜백 함수(local 인증처리)
32          console.log('passport local 인증 처리');
33
34          /*  User모델(users 컬렉션)에서 이메일을 기준으로 검색합니다. */
35          const user = await User.findOne({
36              "email":email,//email로 사용자 정보가 있는지 확인()"email":속성에 email 파라미터를 전달
37          }).exec();
38
39          /* 인증결과를 done()메소드를 이용하여 authenticate 쪽으로 알려 주어야
40             라우팅 함수안에서 authenticate를 호출했을 때 각각의 상황에 따라 분기가 됨 */
41
42          if (user) {
43              /* 이메일로 검색된 유저가 있다면 패스워드가 맞는지 확인합니다.*/
44
45              /** authenticate 메소드는 UserSchema에서 선언한 메소드로서,
46                  유저로부터 비밀번호를 받아 대조하는 기능을 수행하는 메소드 **/
47              if (user.authenticate(password)) {//인증된 경우
48
49                  console.log('done 콜백을 통해서 user객체 정보를 serializeUser() 메서드로 전달');
50                  /* 유저가 있다면 */
51                  return done(null, user)
52              } else {
53                  /* 패스워드가 틀리다면 */
54                  return done(null, false,"패스워드가 틀립니다."); //설정된값은 req.flash()에서 받은 값에서 error object로 받습니다.
55              }
56          } else {
57              /*이메일로도 없는경우 */
58              return done(null, false,"매칭되는 아이디가 없습니다."); //설정된값은 req.flash()에서 받은 값에서 error object로 받습니다.
59          }
60      },
61  );
```



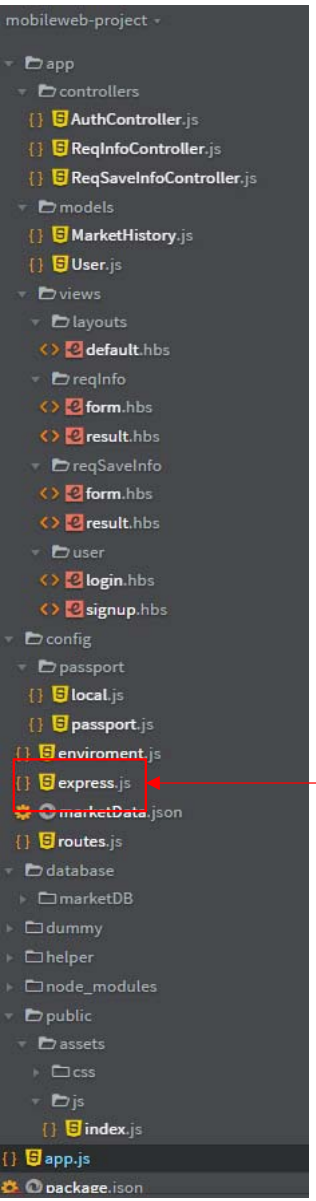
passport.js

```
/*  사용할 인증 방식 등록
  - 인증 후 사용자 정보를 세션에 저장하거나 사용자 정보를 복원하는 모듈 */

/* 인증 방식 설정 파일 참조
   - 인증 방식별로 설정 파일을 만들어 스트래티지를 선언해야함
*/
const local = require('./local');//로컬 스트래티지
//const facebook = require('./facebook');//페이스북 스트래티지
//const twitter = require('./twitter');//트위터 스트래티지
const LocalStrategy = require('passport-local').Strategy;

console.log('call : /config/passport/passport.js');

module.exports = function (passport) {

    /* 사용자 인증에 성공했을 때 호출되는 serializeUser() 콜백 함수는
       전달받은 user.email 객체의 정보를 콘솔에 출력하고 done() 메소드를 호출함
       - done() 메소드의 두번째 파라미터로 user.email 객체를 전달하면,
         이 객체는 그 다음에 처리할 함수쪽으로 전달됨
       - 즉, 인증을 통해 로그인에 성공하였다면 이쪽에서 어떤 사용자 정보를 세션에 저장할지 설정해줍니다.*/
    passport.serializeUser(function (user, done) {
        console.log('passport.serializeUser()  호출 : ' + user.email);
        done(null, user.email)
    });

    /* 사용자 인증 이후 사용자 요청이 있을 때마다 호출됨 - 로그인 상태인 경우 */
    /* 사용자로부터 받은 세션정보와 실제 DB의 데이터와 비교하여줍니다.*/
    /* 세션값과 나의 값을 체크해줌으로써 보안을 위한 기능입니다. */
    passport.deserializeUser(function (email, done) {
        console.log('passport.deserializeUser()  호출 : ' + email);
        const profile = {email: email};
        done(null, profile);
    });

    // 사용하기 위해 참조한 인증방식을 미들웨어로 등록
    passport.use(local);
    //passport.use(facebook);
    //passport.use( twitter);
};
```



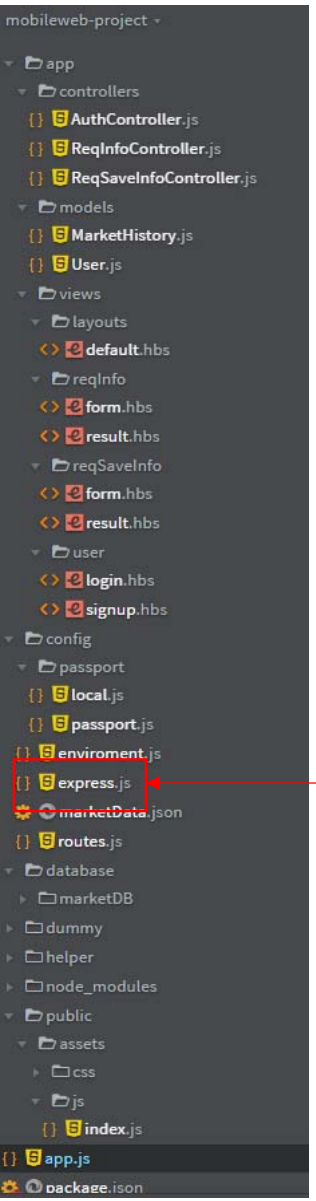
express.js

```
/* express 서버 구축 관련 설정 모듈 */

const express = require('express');
const path = require('path');
const bodyParser = require('body-parser');
const exphbs = require('express-handlebars');
const hbsHelper = require('handlebars-helpers');
const ENV = require("../config/enviroment");

//쿠키모듈
const cookieParser = require('cookie-parser');
const cookieSession = require('cookie-session');

//세션모듈
const session = require('express-session');
const mongoStore = require('connect-mongo')(session);
//const FileStore = require('session-file-store')(session); 파일스토리지사용시

// ===== flash 메시지를 사용하기 위한 'connect-flash' 모듈 참조(2) ====//
/* - connect-flash 모듈은 요청 객체에 메시지를 넣어 둘 수 있는 기능을 제공하는데,
     보통 웹 서버 안의 다른 함수에 메시지를 전달하거나 뷰 템플릿을 처리하는 함수에 메시지를
     전달하기 위해 사용
   - 'connect-flash' 모듈은 'cookie-parser'와 'cookie-session'을 사용함으로
     이들보다 뒤에서 참조해야 함
*/
const flash = require('connect-flash');
```



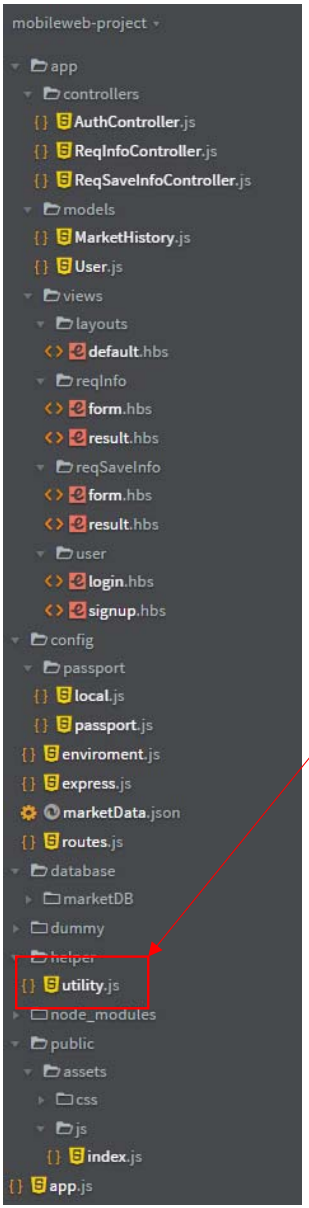
express.js

```
28 module.exports = function (app, passport) {
29
30     console.log('call : /config/express.js');
31
32     app.use(bodyParser.json());
33     app.use(bodyParser.urlencoded({extended: true}));
34
35     //Resource 용으로 사용할 static 폴더(public) 정의
36     app.use(express.static('public'));
37
38     //ejs,hbs등 사용할 view template을 설정합니다.
39     const hbs = exphbs.create({
40         extname: '.hbs', //사용할 확장자
41         partialsDir: __dirname + '/../app/views/partials', // sidebar등 부분적으로 사용할 디렉토리
42         defaultLayout: __dirname + '/../app/views/layouts/default.hbs',
43         layoutsDir: __dirname + '/../app/views/layouts', // 사용할 view들의 위치
44         helpers: hbsHelper,
45         /* 핸들바에서 사용하는 helper 클래스들의 모음으로
46            https://github.com/helpers/handlebars-helpers 플러그인을 사용*/
47     });
48     /* ==== 핸들바 사용을 위한 설정 ==== */
49     require('handlebars-helpers')(hbs);
50     app.engine('.hbs', hbs.engine); //사용할 뷰 엔진(hbs.engine) 설정
51
```



express.js

```
52      /* 쿠기설정 */
53      app.use(cookieParser()); //쿠키사용을 설정합니다.
54      app.use(cookieSession({secret: ENV.SESSION_SECRET})); //세션값을 쿠키에 담습니다.
55
56      /* 세션을 저장할 곳을 설정  */
57      /* 실무에서는 일반적으로 메모리DB(etc. redis) 사용하지만 실습에선 mongodb를 이용 */
58
59      /* 몽고디비를 이용 */
60      app.use(session({
61          secret: ENV.SESSION_SECRET, //자신에 맞게 secret을 설정
62          store: new mongoStore({ //세션데이터를 몽고에 저장하기때문에 mongo 접속정보
63              url: ENV.DATABASE,
64              collection: ENV.MONGO_SESSION_COLLECTION_NAME,
65          }),
66      }));
67
68      //======= passport 초기화(3) ============//
69      /* passport 사용을 위한 설정
70         - passport 모듈의 initialize()함수와 session() 함수를 호출했을 때
71           반환되는 객체를 미들웨어로 사용하도록 설정
72      */
73      app.use(passport.initialize()); //패스포트 초기화
74      app.use(passport.session()); //패스포트 세션설정
75      //flash 메세지 미들웨어 등록
76      app.use(flash());
77
78      app.set('view engine', '.hbs'); //핸들바 뷰 파일의 확장자를 설정(.hbs)
79      app.set('views', path.join(__dirname, '/../app/views')); //뷰가 있는 디렉토리를 정의
80  };
```

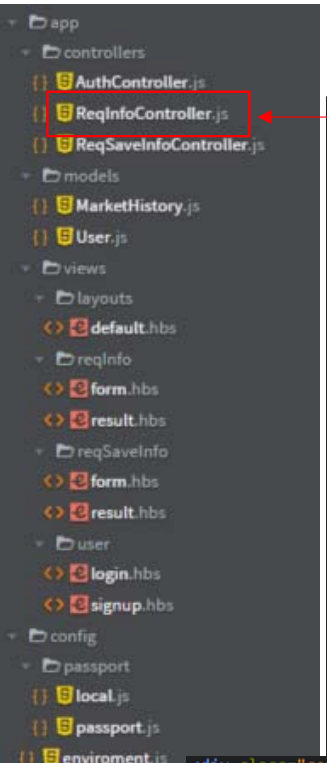



```
/* 클라이언트에서 조회하고자 하는 과일을 선택하면 코드명으로 과일명을 찾아 반환하는 모듈 */
//마켓데이터를 가져옵니다(marketData.json)
const marketList = require('../config/marketData');
/* 코드명을 넣어 과일 이름을 반환합니다.*/
// 단축형으로 사용한다면
/* module.exports.getNameByCode = code =>
        marketList.filter(data => data.mcode == code)[0].name; */

console.log('call : /helper/utlity.js');

module.exports.getNameByCode = function(code){
    return marketList.filter(function(data){
        return data.mcode == code;
    })[0].name;
};
```

utlity.js

marketData.json

ReqInfoController.js (경락정보 조회)




```
/* 경락정보조회 Controller(라우팅 함수 모듈 선언) */

// marketData.json 참조 */
const marketList = require('../../config/marketData');
// utility.js에서 getNameByCode 함수 참조
const {getNameByCode} = require('../../helper/utility');
// enviroment.js.에서 API_URL과 SERVICEKEY 참조
const {API_URL, SERVICEKEY} = require('../../config/enviroment');
// const API_URL = require('../../config/enviroment').API_URL
// const SERVICEKEY =  require('../../config/enviroment').SERVICEKEY
// 비동기 통신을 위한 Request.js 라이브러리 참조(request-promise 모듈)
const request = require('request-promise');
const pageTitle = "경락정보 조회";

console.log('call : /controllers/ReqInfoController.js');

// 라우팅 함수 설정(localhost:3000/reqInfo)
module.exports.index = async function (req, res) {
    //isAuthenticated():세션에 저장된 user 데이터가 있으면 true 없으면 false
    const isLogin = req.isAuthenticated();
    //reqInfo/form.hbs 뷰 템플릿 랜더링
    res.render("reqInfo/form", {marketList: marketList, pageTitle: pageTitle,
                                isUserLogedIn: isLogin});
};
```

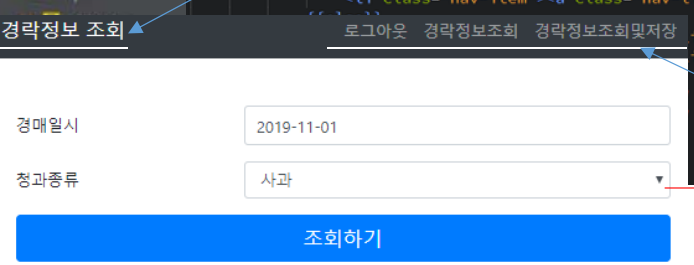
```
module.exports.getNameByCode = function(code){
    return marketList.filter(function(data){
        return data.mcode == code;
    })[0].name;
};
```

```
module.exports = {

    PORT:3000,//포트번호
    DATABASE:"mongodb://localhost:2
    SERVICEKEY:'zMtFz5%2F0QfkhM90Jc
    API에서 발급받은 server key 값
    MONGO_SESSION_COLLECTION_NAME:"
    SESSION_SECRET:"your_secret", /
    API_URL:"http://openapi.epis.or
    PAGINATION:{
        PAGE_SIZE:10
    }
};
```

```
{
  "name": "사과",
  "lcode":"06",
  "mcode": "0601"
},
```

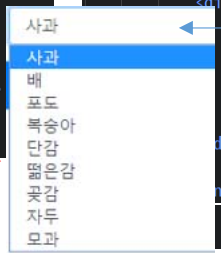
```
<div class="container">
    <a class="navbar-brand" href="/">{{pageTitle}}</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarCollapse"
            aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>

    <div id="navbarCollapse" class="collapse navbar-collapse">
        <ul class="navbar-nav mr-auto"></ul>
        <ul class="navbar-nav">
            {{#if isUserLogedIn}}
                <li class="nav-item"><a class="nav-link" href="/logout">로그아웃</a></li>
            {{else}}
                <li class="nav-item"><a class="nav-link" href="/signup">회원가입</a></li>
                <li class="nav-item"><a class="nav-link" href="/login">로그인</a></li>
            {{/if}}
            <li class="nav-item"><a class="nav-link" href="/reqInfo">경락정보조회</a></li>
            <li class="nav-item"><a class="nav-link" href="/reqSaveInfo">경락정보조회및저장</a></li>
        </ul>
    </div>
</div>
```

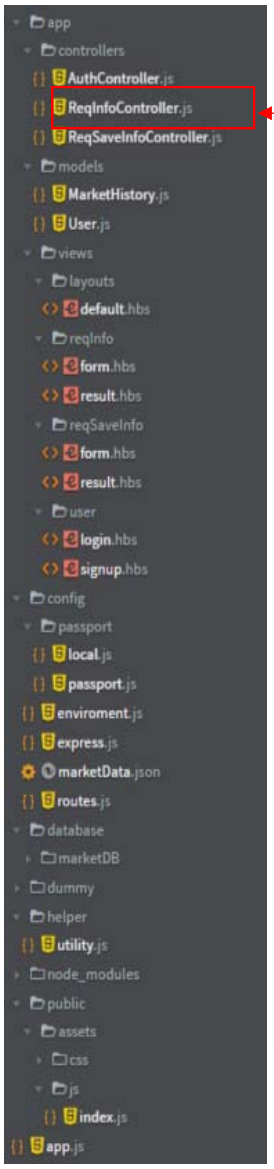
default.hbs




```
<div class="col-md-12">
    <form method="get" action="/reqInfo/result">
        <input type="hidden" name="lcode" value="06">
        <div class="form-group row">
            <label for="example-date-input" class="col-4 col-form-label">경매일시</label>
            <div class="col-8">
                <input name="date" class="form-control" type="date" value="2019-11-01" id="example-date-input"/>
            </div>
        </div>
        <div class="form-group row">
            <label for="example-date-input" class="col-4 col-form-label">청과종류</label>
            <div class="col-8">
                <select class="form-control" name="mcode" id="exampleSelect1">
                    {{#each marketList}}
                    <option value="{{this.mcode}}">{{this.name}}</option>
                    {{/each}}
                </select>
            </div>

            <button type="submit" id="btn-market-search" class="btn btn-primary btn-lg btn-block">조회하기</button>
```

form.hbs





ReqInfoController.js (경락정보 조회)

```
/* 라우팅 함수 설정
   - localhost:3000/reqInfo에서 form을 통해 요청파라미터를 받아 처리
   - form에서 날짜와 과일을 선택한 후 "조회하기" 버튼을 클릭하면
    "/reqInfo/result" 요청 패스로 공공DB 서버에 요청하여, 그 결과 받아 처리 */
module.exports.result = async function (req, res) {

    /* Request.js 라이브러리의 request-promise 모듈을 이용하여 공공DB포털의
       경락가격정보서비스 서버에 Get 요청(요청 파라이터는 qs 옵션에 설정) */
    const marketResult = await request.get({
        url: API_URL,
        timeout: 10000,
        json: true,//리턴받을 데이터를 자동으로 Json.Parse
        qs: {//요청 파라미터 설정
            'ServiceKey': SERVICEKEY,
            //날짜는 2019-08-29 형식으로 들어오기때문에 20190829로 변경
            'dates': req.query.date.split("-").join(''),
            'lcode': req.query.lcode,//대분류
            'mcode': req.query.mcode,//중분류
            '_type': "json",
        },
    }).then((result) => {
        /*데이터를 성공적으로 가져온경우 필요한 items 리스트만 가져옵니다.*/
        console.log(result);
        return result.response.body
    }).catch(e => {
        /* 에러처리*/
        console.error("request Error : " + e)
    });

    /* 조회결과 응답(reqInfo/result.hbs 뷰 템플릿 랜더링) */
    res.render("reqInfo/result", {
        fruit: getNameByCode(req.query.mcode),//과일명
        marketResult: marketResult,//경락가격정보(응답결과)
        query: req.query,//query 객체
        pageTitle: pageTitle,//페이지 타이틀("경락정보 조회")
        isUserLogedIn: req.isAuthenticated(),//로그인인증정보
    });
};
```

```
module.exports.getNameByCode = function(code){
    return marketList.filter(function(data){
        return data.mcode == code;
    })[0].name;
};
```


```
[
  {
    "name": "사과",
    "lcode":"06",
    "mcode": "0601"
  },
  {
    "name": "배",
    "lcode":"06",
    "mcode": "0602"
  },
]
```


result [marketResult]


```
"response": {
    "header": {
        "resultCode": "00",
        "resultMsg": "NORMAL SERVICE."
    },
    "body": {
        "items": {
            "item": [
                {
                    "avgprice": 33000,
                    "coname": "남일청과",
                    "dates": "2018/08/29",
                    "gradename": "없음",
                    "marketname": "순천도매시장",
                    "maxprice": 33000,
                    "mclassname": "배",
                    "minprice": 33000,
                    "rnum": 1,
                    "sclassname": "신고",
                    "sumamt": 110,
                    "unitname": "15kg"
                }
            ]
        },
        "numOfRows": 10,
        "pageNo": 1,
        "totalCount": 361
    }
}
```


```
<div class="row">
    <div class="col-sm-12">
        <h4 class="mt-4 mb-4">{{query.date}}일 / {{fruit}} 도매경매 현황</h4>
        <table class="table table-bordered">
            <thead>
            <tr>
                <th>장소</th>
                <th>업체명</th>
                <th>평균가</th>
                <th>수량</th>
                <th>기준</th>
            </tr>
            </thead>
            <tbody>
            {{#each marketResult.items.item}}
                <tr>
                    <td>{{this.marketname}}</td>
                    <td>{{this.coname}}</td>
                    <td>{{this.avgprice}}</td>
                    <td>{{this.sumamt}}</td>
                    <td>{{this.unitname}}</td>
                </tr>
            {{/each}}
            </tbody>
        </table>
    </div>
</div>

<a href="/" class="btn btn-primary btn-lg btn-block">목록으로</a>
```

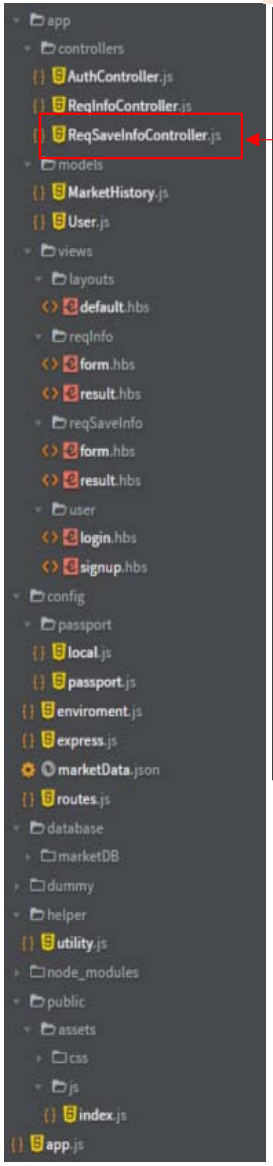
result.hbs

2019-11-01일 / 사과 도매경매 현황

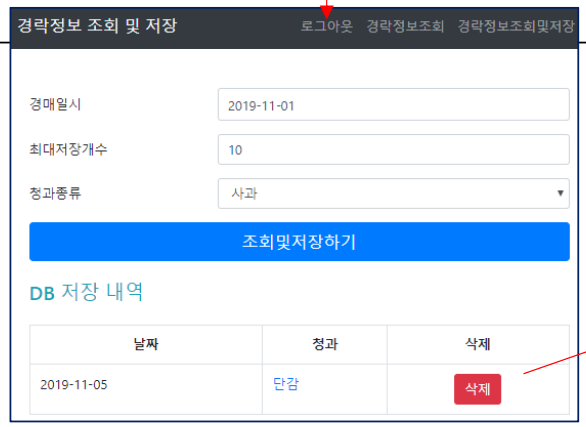
장소	업체명	평균가	수량	기준
서울강서도매	강서청과	19146	96	10kg
수원도매시장	수원청과	21102	265	10kg
원주도매시장	합동청과	15860	240	10kg
광주각화도매	광주중앙청과	12000	2	10kg 상자
구미도매시장	구미농협(공)	17425	32	10kg 상자
대전오정도매	대전청과	21406	3650	10kg 상자
부산반여도매	부산중앙청과	7591	22	10kg 상자
울산도매시장	울산중앙청과	21394	1386	10kg 상자
전주도매시장	전주원협(공)	12909	11	10kg 상자
청주도매시장	충북원협(청주)	3778	9	10kg 상자

목록으로

ReqSaveInfoController.js (경락정보조회및저장)




```js
/* 경락정보조회및저장 Controller
   - 라우팅 함수 모듈 선언 */

const marketList = require('../../config/marketData');
const {SERVICEKEY, API_URL} = require('../../config/enviroment');
const {getNameByCode} = require('../../helper/utility');
const mongoose = require("mongoose");
//DB(MarketHistory 컬렉션) 모델 가져오기
const MarketHistory = mongoose.model('MarketHistory');

// 비동기 통신을 위한 Request.js 라이브러리 참조(request-promise 모듈)
const request = require('request-promise');
const pageTitle = "경락정보 조회 및 저장"; //고정 상수 타이틀값

console.log('call : /controllers/ReqSaveInfoController.js');

/* 라우팅 함수 설정(localhost:3000/reqSaveInfo)
   - DB에 저장된 경락정보를 리스트로 표시하기 위한 index 모듈 */
module.exports.index = async function (req, res) {
    //DB(MarketHistory 컬렉션)의 데이터를 조회(전체)
    const marketHistories = await MarketHistory.find();

    //reqSaveInfo/form.hbs 뷰 템플릿 랜더링
    res.render("reqSaveInfo/form", {
        marketList: marketList,//marketData.json
        pageTitle: pageTitle,//페이지타이틀("경락정보 조회 및 저장")
        marketHistories: marketHistories,//MarketHistory 컬렉션
        isUserLogedIn:req.isAuthenticated()//로그인 인증정보
    });
};
```

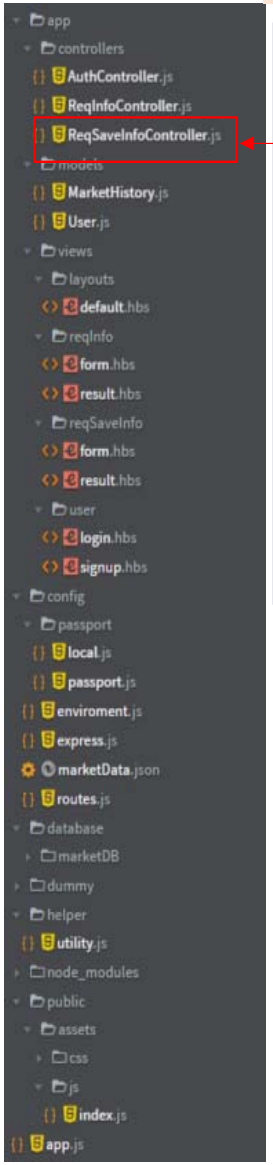


form.hbs

```
<div class="col-md-12">
    <input type="hidden" name="lcode" value="06">
    <div class="form-group row">
        <label for="example-date-input" class="col-4 col-form-label">경매일시</label>
        <div class="col-8">
            <input name="date" class="form-control" type="date" value="2018-11-22"/>
        </div>
    </div>
    <div class="form-group row">
        <label for="example-date-input" class="col-4 col-form-label">최대저장개수</label>
        <div class="col-8">
            <input name="count" class="form-control" type="number" value="10"
        </div>
    </div>
    <div class="form-group row">
        <label for="example-date-input" class="col-4 col-form-label">청과종류<
        <div class="col-8">
            <select class="form-control" name="mcode" id="exampleSelect1">
                {{#each marketList}}
                    <option value="{{this.mcode}}">{{this.name}}</option>
                {{/each}}
            </select>
        </div>
    </div>
    <button type="submit" id="btn-market-save" class="btn btn-primary btn-lg btn-block">저장하기</button>
    <div class="row">
        <div class="col-sm-12">
            <h4 class="mt-4 mb-4">저장된 내역</h4>
            <table class="table table-bordered">
                <thead>
                <tr>
                    <th>날짜</th>
                    <th>청과</th>
                    <th>삭제</th>
                </tr>
                </thead>
                <tbody>
                {{#each marketHistories}}
                    <tr>
                        <td>{{this.date}}</td>
                        <td><a href="/internal/show?id={{this._id}}">{{this.fruit}}</a></td>
                        <td style="text-align:center"><button type="button" data-id="{{this._id}}" class="bt
                        delete-history">삭제</button></td>
                    </tr>
                {{/each}}
                </tbody>

            </table>
        </div>
    </div>
</div>
```

```
[
  {
    "name": "사과",
    "lcode":"06",
    "mcode": "0601"
  },
  {
    "name": "배",
    "lcode":"06",
    "mcode": "0602"
  },
```

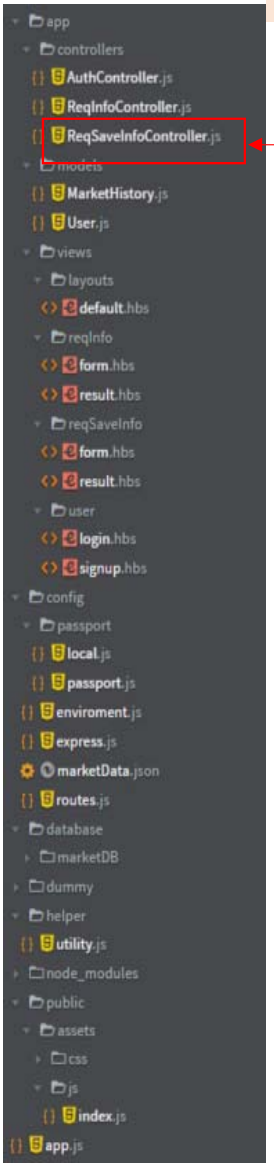
ReqSaveInfoController.js (경락정보조회및저장)




```
33 /* show 상세보기 라우팅 함수 설정 */
34 // http://localhost/reqSaveInfo/show?id={Object_ID}
35 module.exports.show = async function (req, res) {
36     //DB(MarketHistory 컬렉션)의 데이터를 조회(id값에 맞는 1개조회)
37     const marketHistory = await MarketHistory.findOne({_id: req.query.id});
38
39     //reqSaveInfo/result.hbs 뷰 템플릿 랜더링
40     res.render("reqSaveInfo/result", {
41         pageTitle: pageTitle,
42         marketHistory: marketHistory,
43         isUserLogedIn:req.isAuthenticated()
44     });
45 };
46
47 // DELETE 라우팅 함수 설정 - http://localhost/reqSaveInfo/delete
48 module.exports.deleteResult = async function (req, res) {
49     //DB(MarketHistory 컬렉션)에서 데이터 삭제(id)
50     MarketHistory.findOneAndRemove(req.body.id, (err) => {
51         if (err) res.status(500).send();
52         else res.status(200).send()
53     });
54 };
```

result.hbs

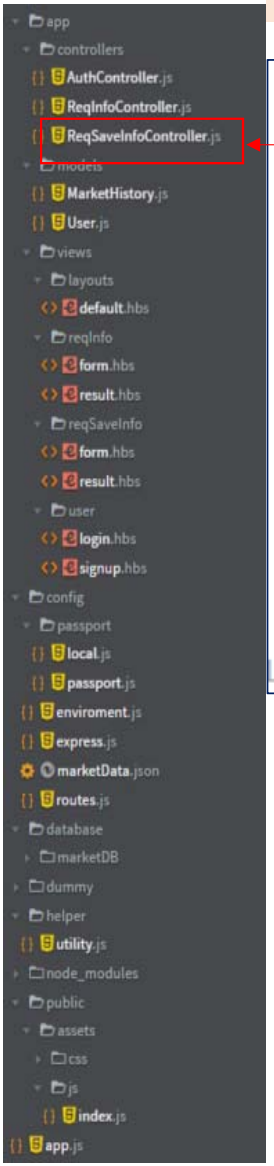
```
<div class="row">
    <div class="col-sm-12">
        <h4 class="mt-4 mb-4">{{marketHistory.date}}일 / {{marketHistory.fruit}} 도매경매 현황</h4>
        <table class="table table-bordered">
            <thead>
            <tr>
                <th>장소</th>
                <th>업체명</th>
                <th>평균가</th>
                <th>수량</th>
                <th>기준</th>
            </tr>
            </thead>
            <tbody>
            {{#each marketHistory.result}}
                <tr>
                    <td>{{this.marketname}}</td>
                    <td>{{this.coname}}</td>
                    <td>{{this.avgprice}}</td>
                    <td>{{this.sumamt}}</td>
                    <td>{{this.unitname}}</td>
                </tr>
            {{/each}}
```



ReqSaveInfoController.js (경락정보조회및저장)

```
/* 라우팅 함수 설정
   - localhost:3000/reqSaveInfo에서 form을 통해 요청파라미터를 받아 처리
   - form에서 날짜와 저장개수, 과일을 선택한 후 "조회및저장하기" 버튼을 클릭하면
    "/reqSaveInfo/result" 요청 패스로 공공DB 서버에 요청하여, 그 결과 받아 처리 */
module.exports.createResult = async function (req, res) {

    /* Request.js 라이브러리의 request-promise 모듈을 이용하여 공공DB포털의
       경락가격정보서비스 서버에 Get 요청(요청 파라이터는 qs 옵션에 설정)
       - 서버에서 응답한 경락가격정보를 DB에 저장 */
    const marketResult = await request.get({
        url: API_URL,
        timeout: 10000,
        json: true,
        qs: {
            'ServiceKey': SERVICEKEY,
            'dates': req.body.date.split("-").join(''),
            'lcode': req.body.lcode,//
            'mcode': req.body.mcode,
            'numOfRows': req.body.count,
            '_type': "json",
        },
    }).then((result) => {
        /*데이터를 성공적으로 가져온경우 필요한 item 리스트만 가져옵니다.*/
        return result.response.body
    }).catch(e => {
        /* 에러처리*/
        console.error("request Error : " + e);
        res.status(500).send();
    });
```


```
"response": {
    "header": {
        "resultCode": "00",
        "resultMsg": "NORMAL SERVICE."
    },
    "body": {
        "items": {
            "item": [
                {
                    "avgprice": 33000,
                    "coname": "남일청과",
                    "dates": "2018/08/29",
                    "gradename": "없음",
                    "marketname": "순천도매시장",
                    "maxprice": 33000,
                    "mclassname": "배",
                    "minprice": 33000,
                    "rnum": 1,
                    "sclassname": "신고",
                    "sumamt": 110,
                    "unitname": "15kg"
                }
            ]
        },
        "numOfRows": 10,
        "pageNo": 1,
        "totalCount": 361
    }
}
```

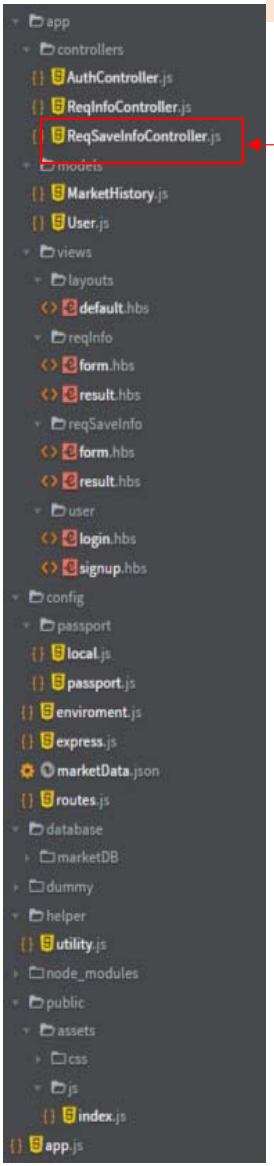


ReqSaveInfoController.js (경락정보조회및저장)

```
85
86      /* 0보다 많을경우 데이터베이스에 저장 */
87      if (marketResult.totalCount > 0) {
88          new MarketHistory({
89              fruit: getNameByCode(req.body.mcode),//과일명
90              date: req.body.date,//저장일자
91              result: marketResult.items.item,//조회된경락정보
92          }).save((err) => {
93              if (err) res.status(500).send(); //실패시 500리턴
94              else res.status(200).send() // 성공시 200리턴
95          })
96      } else {
97          /* 검색된 결과가 0개인경우*/
98          res.status(204).send("데이터가 없습니다.");
99      }
100 };
```

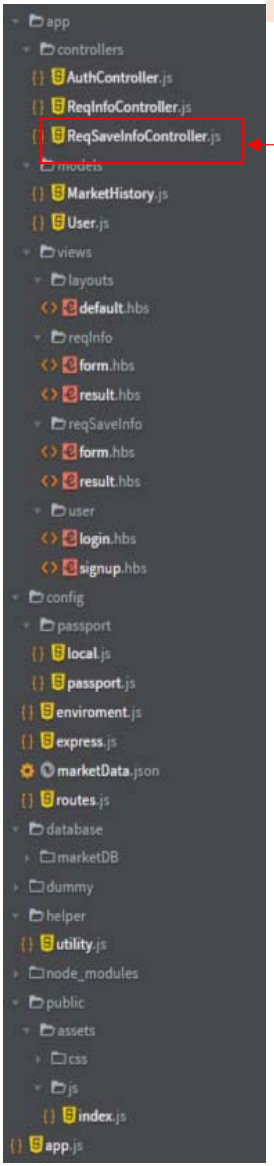
markethistories

	_id ObjectId	fruit String	date String	result Array	__v Int32
1	5dc619be5ae0e517f44f2c30	"단감"	"2019-11-05"	[] 10 elements	0



AuthController.js (회원가입 및 로그인 인증)

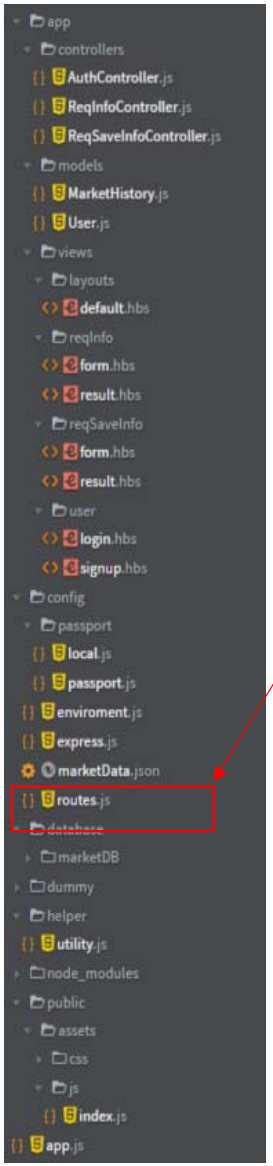
```
1  // 회원 가입 및 로그인 인증 */
2
3  const mongoose = require("mongoose");
4  //User 모델을 가져온다.
5  const User = mongoose.model('User');
6
7  console.log('call : /controllers/Authcontroller.js');
8
9  //login 라우팅 함수 설정 */
10 module.exports.login = async function (req, res) {
11     //성공했다면 req.flash().message가 담깁니다.
12     //실패했다면 자동으로 에러메세지가  req.flash().error로 세팅되어받습니다.
13     res.render("user/login", {pageTitle: "로그인", alert: req.flash()});
14 };
15
16 //signup 라우팅 함수 설정 */
17 module.exports.signup = async function (req, res) {
18     res.render("user/signup", {pageTitle: "회원가입", alert: req.flash()});
19 };
20
21 //logout 라우팅 함수 설정 */
22 module.exports.logout = async function (req, res) {
23     req.logout();
24     res.redirect('/login');
25 };
26
```



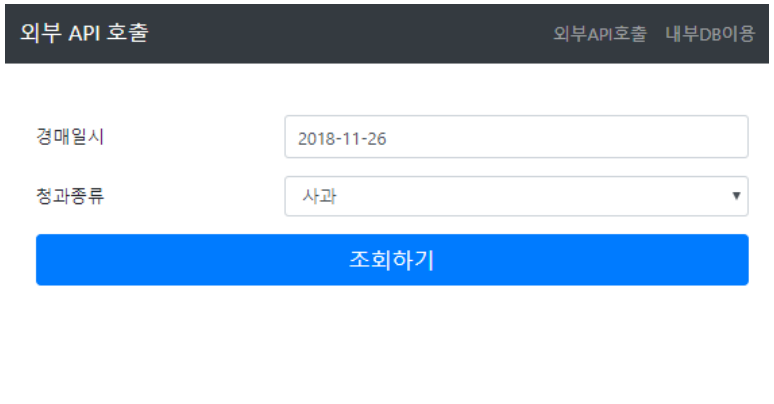
AuthController.js (회원가입 및 로그인 인증)

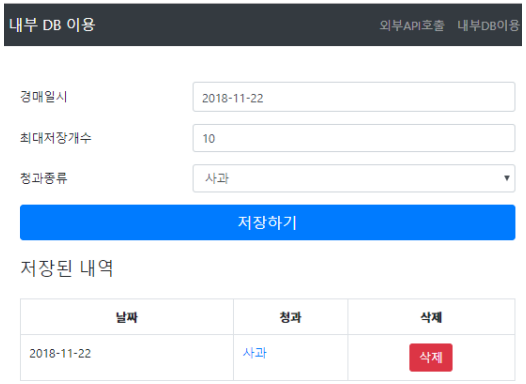
```
27 //DB(users 컬렉션)에 이메일과 비밀번호, salt 저장 라우팅 함수 설정 */
28 module.exports.create = async function (req, res) {
29
30     new User({
31         email: req.body.user_email,
32         password: req.body.user_password,
33     }).save((err) => {// DB에 저장
34         if (err) {
35             req.flash('message', err.message);
36             res.redirect('/signup');
37         } else {
38             req.flash('message', "회원가입성공");
39             res.redirect('/login');
40         }
41
42     });
43 };
44 //requiresLogin 라우팅 함수 설정 */
45 // 인증을 통해 로그인 되어 있어여 다음을 실행하고,
46 // 아니면 '/login'으로 리다이렉트
47 // 예) app.get("/reqInfo", auth.requiresLogin,marketReqInfo.index);
48 module.exports.requiresLogin = async (req, res, next) => {
49     if (req.isAuthenticated()) return next();
50     else res.redirect('/login');
51 };
52
53 //checkUserLogin 라우팅 함수 설정 ???? */
54 module.exports.checkUserLogin = async function (req, res) {
55     /* 세션에 redirect가 저장되어 있다면 해당 페이지를 보여줍니다.*/
56     /* 현재 샘플에선 구현이 되어있진 않지만 쇼핑몰의 경우 현재 보고 있는 상품을
57     장바구니에 누른 후 로그인 요청창이 뜨는경우 로그인을 한 후에도 계속 보고있던 상품을 보도록합니다.
58     아래 코드의 경우 sessions에 returnTo가 설정되어있다면 해당주소로, 없다면 rootpath('/')로 이동합니다.
59      */
60     const redirectTo = req.session.returnTo
61         ? req.session.returnTo
62         : '/';
63     delete req.session.returnTo;
64     res.redirect(redirectTo);
65 };
```

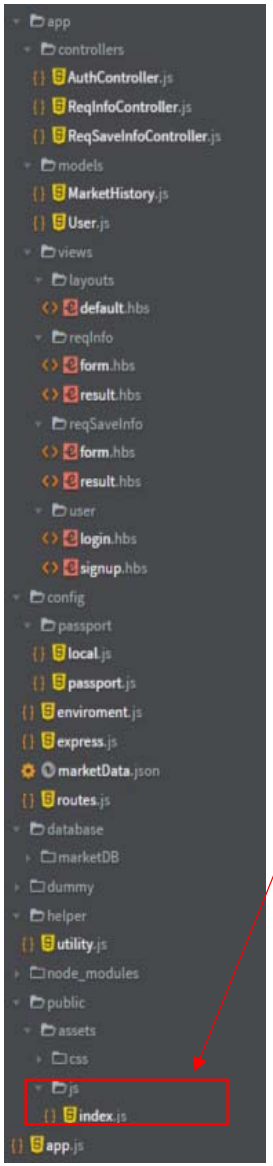
routes.js




```
/* 클라이언트 요청 패스에 따른 라우팅 함수 설정 모듈 */

const marketReqInfo = require("../app/controllers/reqInfoController");
const marketReqSaveInfo = require("../app/controllers/reqSaveInfoController");
const auth = require("../app/controllers/AuthController");

console.log('call : /config/routes.js');

//라우팅 함수 설정
module.exports = function (app, passport) {

    //comment: 공공데이터 포털에 요청한 경락정보를 그대로 파싱하여 리턴
    app.get("/", ((req, res) => res.redirect('/reqInfo'))); //메인화면(경락가격조회로 redirect)
    app.get("/reqInfo", auth.requiresLogin,marketReqInfo.index); //reqInfo의 form값
    app.get("/reqInfo/result", auth.requiresLogin,marketReqInfo.result);//reqSaveInfo의 결과값을 보여줍니다.
    app.get("/reqSaveInfo", auth.requiresLogin,marketReqSaveInfo.index);//reqSaveInfo form 화면
    app.get("/reqSaveInfo/show", auth.requiresLogin,marketReqSaveInfo.show); //reqSaveInfo 결과값
    app.delete("/reqSaveInfo/delete",auth.requiresLogin, marketReqSaveInfo.deleteResult);//reqSaveInfo DB 삭제
    app.post("/reqSaveInfo/result",auth.requiresLogin, marketReqSaveInfo.createResult);//요청결과 DB에 저장

    app.get('/login', auth.login);//로그인
    app.get('/signup', auth.signup);//회원가입
    app.get('/logout', auth.logout);//로그아웃

    /* 유저로그인 */
    app.post("/login_user", passport.authenticate('local', {
        successRedirect : '/',
        failureRedirect: '/login',
        failureFlash:true
    }),auth.checkUserLogin);

    /* 유저생성*/
    app.post('/create', auth.create);
};
```





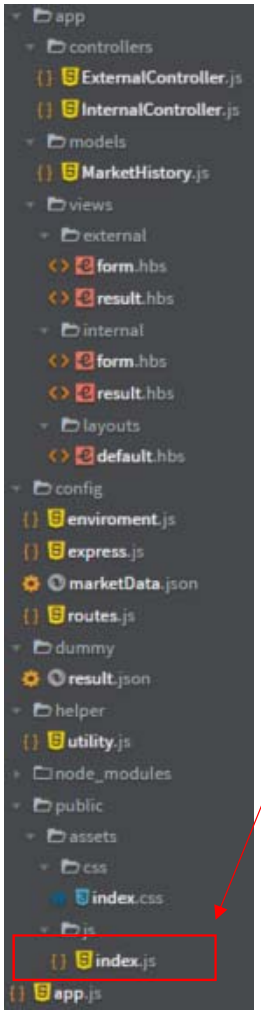


index.js

```
/* 서버에서 사용자에게 응답하는 기본 웹 문서(/view/layout/default.hbs)에서 사용하는 스크립트 */

$(document).ready(function () {

    console.log('call : /public/index.js');

    /* '내부DB이용'(views/internal/forms.hbs)화면에서 '경매일시','최대저장개수','청과종류'를 선택한 후
       저장하기버튼(#btn-market-save") 클릭하면 fetch API를 이용하여 '/internal/result' 요청 패스로
       데이터를 DB에 저장할 것을 요청하는 이벤트  */
    $("#btn-market-save").click(function () {

        var _this = $(this)//button의 this를 저장
        $(this).prop("disabled", true); //버튼을 또 못누르도록
        $(this).text("로딩중..."); //버튼 텍스트 바꾸어주기

        var date = $("input[name*='date']").val();
        var count = $("input[name*='count']").val();
        var mcode = $("select[name*='mcode']").val();
```

default.hbs

```
<div class="container mt-5">
    {{{body}}}
</div>

<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/limonte-sweetalert2/6.11.2/sweetalert2.all.min.js"></script>
<script src="/assets/js/index.js"></script>
</body>
</html>
```



index.js

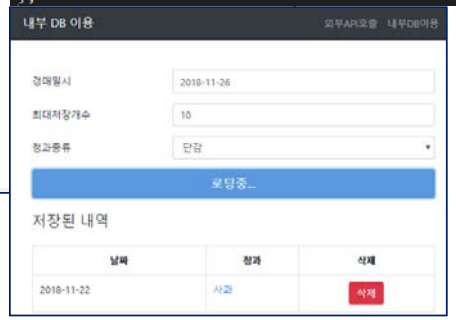
```
20          /* fetch API를 이용하여 '/internal/result' 요청 패스로 요청
21             - '/internal/result' 요청 패스로 요청하면 routes.js의
22                app.post("/internal/result", marketInternal.createResult) 라우팅에 의해
23                InternalController.js의 createResult() 함수 모듈에서 서버에 요청하고
24                그 결과를 DB에 저장한 후, 상태코드를 response 객체로 응답함 - then(function (response)
25          */
26          fetch('/internal/result', {//서버에 요청
27              //method, headers, body 속성을 설정하여 요청
28              method: 'POST',
29              headers: {
30                  'Accept': 'application/json',
31                  'Content-Type': 'application/json',
32              },
33              body: JSON.stringify({date:date, mcode:mcode, count:count}),//JSON 문자열로 변환
34          }).then(function (response) {
35              //fetch API를 통해 서버에 요청했을 때 서버의 응답 결과는 response 객체로 받음
36              // response 객체 : 서버와 통신한 상태 정보를 가지고 있음
37              /* 조회된 있으면 정상(200) */
38              if (response.status === 200) {
39                  location.reload();// 저장 결과 화면 새로고침(f5와 동일)
40              } else {//조회된 결과가 없으면
41                  swal({
42                      type: 'warning',
43                      title: '주의',
44                      text: "조회된 결과수가 0개이므로 저장되지 않았습니다.",
45                  })
46              }
47              _this.prop("disabled", false); //버튼을 원래 상태로 변경합니다.
48              _this.text("저장하기"); //버튼을 원래 상태로 변경합니다.
49          }).catch(function (data) {
50              _this.prop("disabled", false); //버튼을 원래 상태로 변경합니다.
51              _this.text("저장하기"); //버튼을 원래 상태로 변경합니다.
52              showError(msg)
53          });
54      });
```

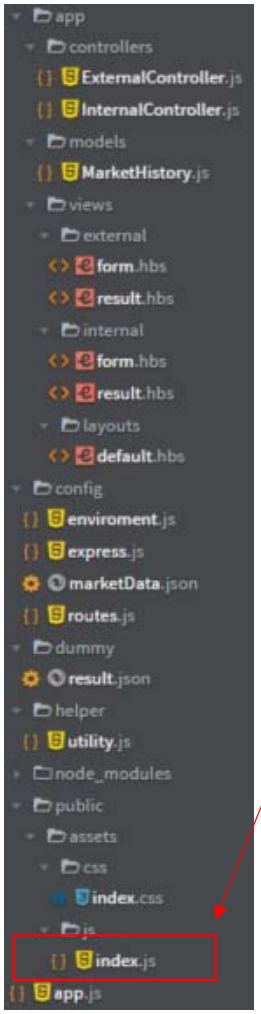
InternalController.js

```
55  module.exports.createResult = async function (req, res) {
56
57      //리턴받은 결과값을 marketResult 변수에 담습니다.
58      const marketResult = await request.get({
59          url: API_URL,
60          timeout: 10000,
61          json: true,
62          qs: {
63              'ServiceKey': SERVICEKEY,
64              'dates': req.body.date.split("-").join(''),
65              'lcode': req.body.mcode.substring(0, 2),
66              'mcode': req.body.mcode,
67              'numOfRows': req.body.count,
68              '_type': "json",
69          },
70      }).then((result) => {
```

routes.js

```
const marketExternal = require("../app/controllers/ExternalController");
const marketInternal = require("../app/controllers/InternalController");

module.exports = function (app) {

    console.log('call : /config/routes.js');

    /**
     * comment: 외부에 요청한 데이터를 그대로 파싱하여 리턴합니다.
     */
    app.get("/", ((req, res) => res.redirect('/external'))); //메인화면으로 외부API로 redirect합니다.
    app.get("/external",marketExternal.index); //외부 external의 form값
    app.get("/external/result", marketExternal.result); //외부 external의 결과값을 보여줍니다.
    app.get("/internal", marketInternal.index);//내부 DB 이용 form 화면
    app.get("/internal/show", marketInternal.show); //내부 DB 이용 결과값
    app.delete("/internal/delete", marketInternal.deleteResult);//내부 DB에서 삭제
    app.post("/internal/result", marketInternal.createResult);//서버에 요청하여 결과값을 DB에 저장

};
```





index.js

```
56      /* 데이터 삭제 */
57      /* '내부DB이용'(views/internal/forms.hbs)화면에서 저장된 내역의
58          '삭제' 버튼(".btn-delete-history") 클릭하면 ajax를 이용하여 '/internal/delete' 요청 패스로
59          데이터 삭제를 요청하는 이벤트  */
60      $(".btn-delete-history").click(function () {
61          /* ajax를 이용하여 'deleteResult' 요청 패스로 요청
62            - '/internal/delete' 요청 패스로 요청하면 routes.js의
63              app.delete("/internal/delete", marketInternal.deleteResult); 라우팅에 의해
64              InternalController.js의 deleteResult() 함수 모듈에서 해당 id 데이터를 삭제하고
65              그 결과를 응답함
66          */
67          $.ajax({
68              url: '/internal/delete',
69              type: "DELETE",
70              data: {
71                  'id': $(this).data("id"), //id값
72              },
73              success: function () {
74                  location.reload(); //성공한 경우 화면을 새로고침(f5와 동일)
75              },
76              error: (msg) => showError(msg),//Error 메시지 출력
77          })
78      });
79  });
80
81  /**
82   * 에러메세지를 띄웁니다.
83   */
84  function showError(msg) {
85      swal({
86          type: 'error',
87          title: '에러코드',
88          text: JSON.stringify(msg),
89      })
90  }
```