**Chapter 09. 패스포트로 사용자 인증하기**

* 패스포트 모듈 분리하기
* [config] 폴더 생성 후 그 하위에 [passport] 폴더 생성
* [/config/passport.js]

var local\_login = require('./passport/local\_login');

var local\_signup = require('./passport/local\_signup');

// 이 함수 실행 시에는 app, passport 객체를 사용하므로 호출 시 app과 passport 객체를 파라미터로 전달해줘야 정상적으로 실행이 가능하다.

module.exports = function(app, passport){

console.log('config/passport 호출됨');

// 사용자 인증에 성공했을 때 호출

passport.serializeUser(function(user, done){

console.log('serializeUser() 호출됨');

console.dir(user);

done(null, user);

});

// 사용자 인증 이후 사용자 인증 요청 있을 때 호출

passport.deserializeUser(function(user, done){

console.log('deserializeUser() 호출됨');

console.dir(user);

done(null, user);

});

// 인증방식 설정

passport.use('local-login', local\_login);

passport.use('local-signup', local\_signup);

};

* [config/passport/local\_login.js]

var LocalStrategy = require('passport-local').Strategy;

module.exports = new LocalStrategy({

usernameField: 'email',

passwordField: 'password',

passReqToCallback: true

}, function(req, email, password, done){

console.log('passport의 local-login 호출됨 : ' + email + ', ' + password);

var database = req.app.get('database');

database.UserModel.findOne({ 'email': email }, function(err, user){

if(err) { return done(err); }

// 등록된 사용자가 없는 경우

if(!user){

console.log('계정이 일치하지 않음');

return done(null, false, req.flash('loginMessage', '등록된 계정이 없습니다.'));

}

// 비밀번호 불일치 하는 경우

var authenticated = user.authenticate(password, user.\_doc.salt, user.\_doc.hashed\_password);

if(!authenticated){

consle.log('비밀번호 일치하지 않음');

return done(null, false, req.flash('loginMessage', '비밀번호가 일치하지 않습니다.'));

}

// 정상

console.log('계정과 비밀번호가 일치함');

return done(null, user);

});

});

* [/config/passport/local\_signup.js]

var LocalStrategy = require('passport-local').Strategy;

module.exports = new LocalStrategy({

        usernameField : 'email',

        passwordField : 'password',

        passReqToCallback : true // 이 옵션을 설정하면 아래 콜백 함수의 첫번째 파라미터로 req 객체 전달됨

    }, function(req, email, password, done) {

// 요청 파라미터 중 name 파라미터 확인

var paramName = req.body.name || req.query.name;

        console.log('passport의 local-signup 호출됨 : ' + email + ', ' + password + ', ' + paramName);

     // findOne 메소드가 blocking되지 않도록 하고 싶은 경우, async 방식으로 변경

     process.nextTick(function() {

         var database = req.app.get('database');

         database.UserModel.findOne({ 'email' : email }, function(err, user) {

         // 에러 발생 시

         if (err) {

         return done(err);

         }

         // 기존에 사용자 정보가 있는 경우

         if (user) {

             console.log('기존에 계정이 있음.');

         return done(null, false, req.flash('signupMessage', '계정이 이미 있습니다.')); // 검증 콜백에서 두 번째 파라미터의 값을 false로 하여 인증 실패한 것으로 처리

         } else {

             // 모델 인스턴스 객체 만들어 저장

             var user = new database.UserModel({'email':email, 'password':password, 'name':paramName});

             user.save(function(err) {

                 if (err) {

                     throw err;

                 }

              console.log("사용자 데이터 추가함.");

              return done(null, user); // 검증 콜백에서 두 번째 파라미터의 값을 user 객체로 넣어 인증 성공한 것으로 처리

             });

         }

         });

     });

    });

* [app.js]

내부에서 앞서 선언한 코드를 app.js에서 삭제하고, config 폴더의 passport를 불러오는 코드를 작성한다.

// 패스포트 설정

var configPassport = require('./config/passport');

configPassport(app, passport);

* [routes/user\_passport.js]

app.js 내의 라우팅 관련 함수를 모듈로 분리한다.