

Node.js 강좌



@IT Korea 강사 이정훈

Passport 란?



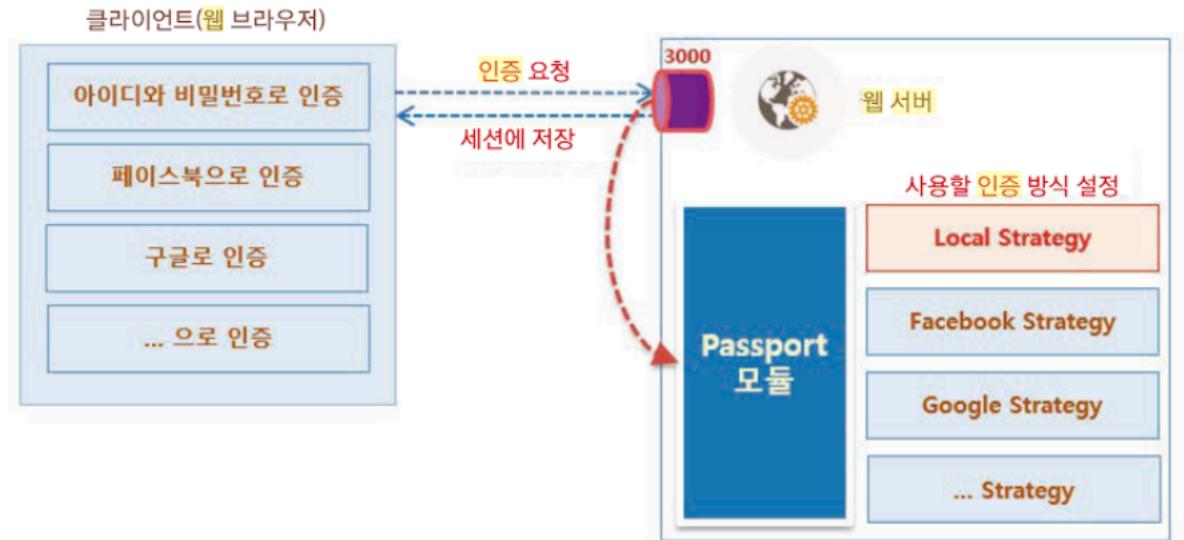
패스포트(passport)란 노드에서 사용하는 사용자 인증 모듈입니다.

패스포트 모듈의 목적은 클라이언트에서 요청한 인증정보로 사용자 인증을 하는 것입니다.

Strategy(스트래티지)

1. 로컬 인증 방식

2. OAuth 방식



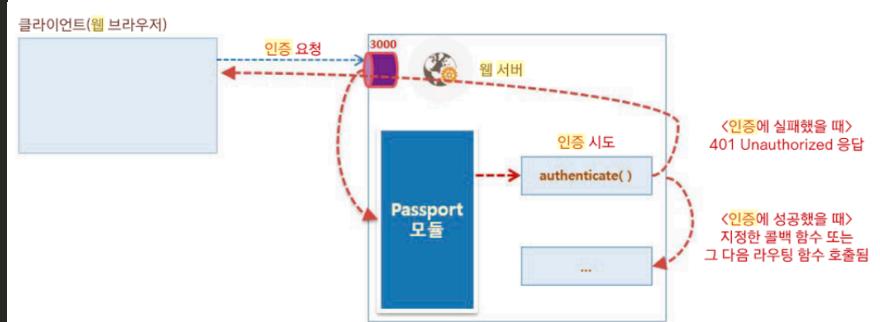
패스포트는 수백 가지 인증 방식을 제공하는데, 어떤 인증 방식을 사용하는지 결정하는 것이 스트래티지 입니다. 그리고 어떤 스트래티지를 사용하느냐에 따라서 인증방식이 달라집니다.

위 그림은 패스포트 미들웨어를 사용해 웹서버에서 인증하는 방식입니다.

Passport 사용 방법

```
router.route('/login').post(passport.authenticate('local',
{
  successRedirect: '/',
  failureRedirect: '/login'
});

router.route('/login').post(passport.authenticate('local',
  function(req, res){
    // 인증에 성공했을 때 호출됨
    // 'req.user' 는 인증된 사용자의 정보임
    res.redirect('/users' + req.user.username);
  }
));
```



Passport 인증 메소드의 local은 인증방식을 결정하는 스트래티지 이름입니다.

이 이름은 해당 스트래티지 설정시 지정합니다.

이 때 스트래티지는 라우팅 함수에 사용되기전에 설정해야합니다.

‘플래시 메세지’ 와 ‘커스텀콜백’

```
router.route('/login').post(passport.authenticate('local',
  {
    successRedirect : '/',
    failureRedirect : '/login'
    failureFlash : true
  });
}

router.route('/login').get(function(req,res,next) {
  passport.authenticate('local',function(err,user,info) {
    if(err){ return next(err);}
    if(!user){
      return res.redirect('/login');
    }
    //패스포트 인증결과에 따라 로그인 진행
    req.login(user,function(err){
      if(err){ return next(err);}
      return res.redirect('/users/' + user.username);
    });
  })(req,res,next);
});
```

리다이렉트로 응답을 보낼때 보통 플래시 메세지를 같이 사용합니다.

이 플래시 메세지는 스트레티지 설정시 검증콜백이 설정되어있다면 자동으로 설정됩니다.

커스텀 콜백은 인증을 성공했거나 실패할때 직접 함수로 지정한 것입니다.

스트래티지 설정과 검증 콜백

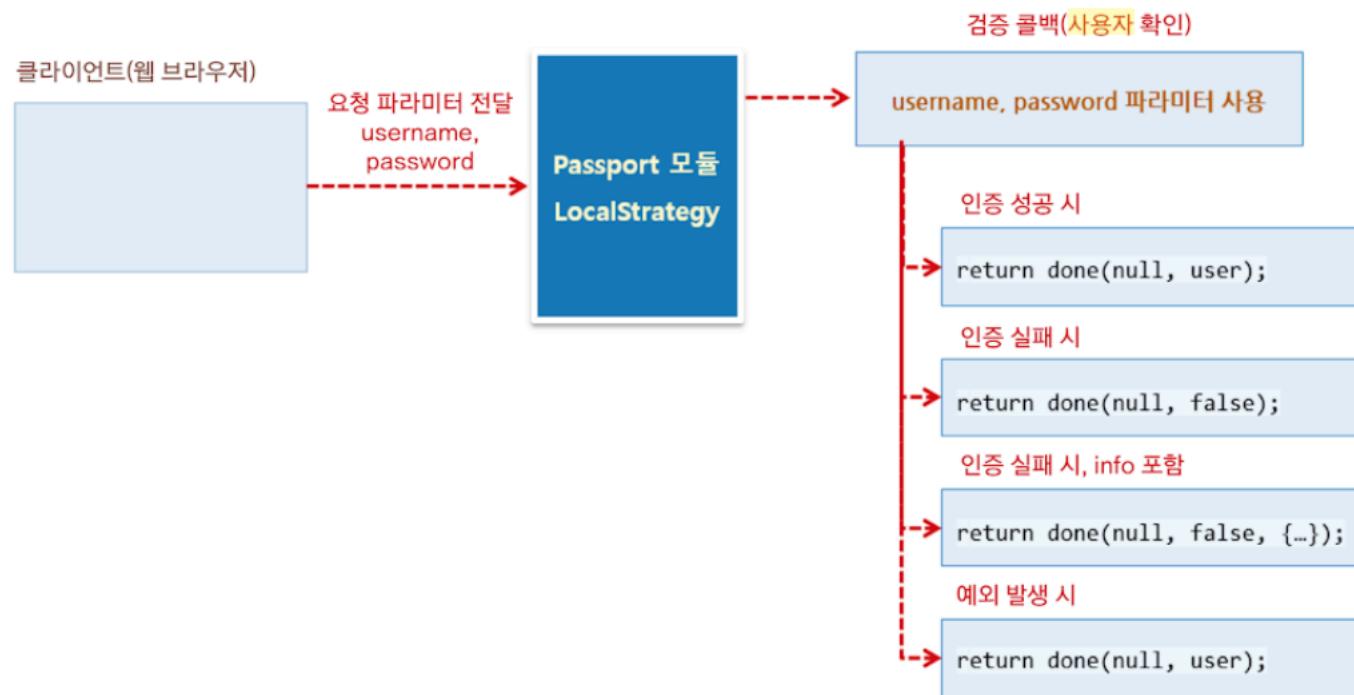
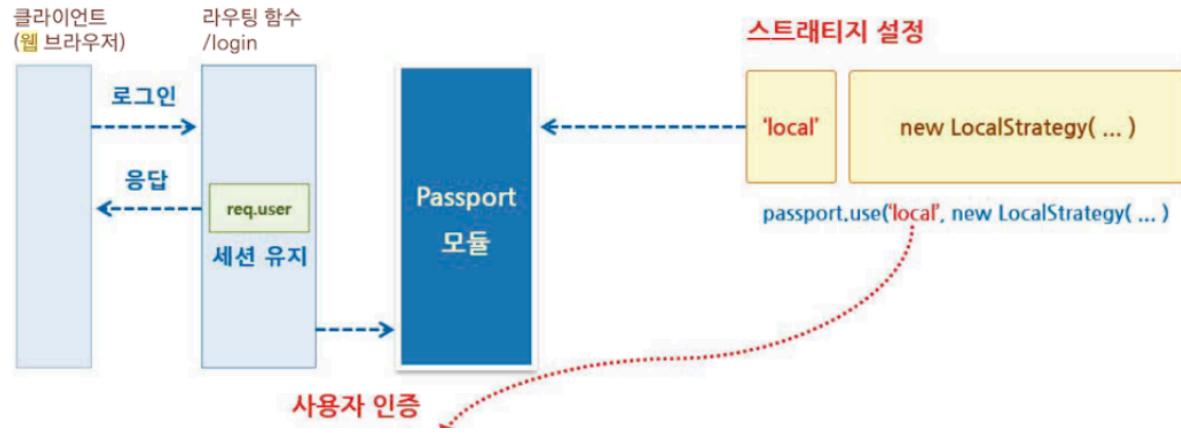
```
var passport = require('passport')
var LocalStrategy = require('passport-local').Strategy;

passport.use(new LocalStrategy(
  function(username, password, done){
    UserModel.findOne({username: username}, function(err,user) {
      if(err){ return done(err);}
      if(!user){
        return done(null,false,{message:'Incorrect username.'});
      }

      if(!user.validPassword(password)){
        return done(null,false,{message:'Incorrect password.'});
      }

      return done(null, user);
    });
  });
});
```

로컬 인증 방식인 LocalStrategy를 사용하는 전형적인 코드입니다.



콜백함수란?

친구들과 즐겁게 시내를 돌아다니다가, 집에 갈 때 사갈 떡볶이를 사가려고 한다.
그런데 이게 무슨일이람. 떡볶이가 너무 많이 밀려서 시간이 조금 걸린다고 한다!
그래서 나는 전화번호를 주고, 조리가 끝나면 받아갈테니 **전화를 다시 주라고 하였다!**

```
plus = function(a,b,callback){  
    var result = a+b;  
    callback(result);  
}  
  
plus(5,10,function(res){  
    console.log(res);  
})  
  
pm = function(a,b, callback){  
    callback(a+b, a-b);  
}  
  
pm(5,10, function(res1, res2){  
    console.log(res1);  
    console.log(res2);  
})
```

콜백 함수에 대해 잘 이해가 안되는 분을 위해
한번더 설명을 하고 넘어가겠습니다.
콜백, 자신을 다시 불러온다는 의미겠죠.
무엇인가 일을 다른 객체에 시키고,
그 일을 끝내는걸 기다리지 않고, 그 객체가
다시 나를 부를때까지 내 할일을 하는것.

데이터베이스 스키마와 패스포트 설정하기

```
app.js

// Express 기본 모듈 불러오기
var express = require('express')
, http = require('http')
, path = require('path');

// Express의 미들웨어 불러오기
var bodyParser = require('body-parser')
, cookieParser = require('cookie-parser')
, static = require('serve-static');

// Session 미들웨어 불러오기
var expressSession = require('express-session');

//===== Passport 사용 =====/
var passport = require('passport');
var flash = require('connect-flash');
```

```
//===== Passport 사용 설정 =====/
// Passport의 세션을 사용할 때는 그 전에 Express의 세션을 사용하는 코드가 있어야 함
app.use(passport.initialize());
app.use(passport.session());
app.use(flash());
```

먼저 app.js 에 passport모듈과, flash 모듈을 선언해줍니다.
그리고 우리는 로컬인증 방식이기에 npm passport-local 역시 설치해줍니다.

```
user_schema.js
```

```
var crypto = require('crypto');

var Schema = {};

Schema.createSchema = function(mongoose) {

    // 스키마 정의
    var UserSchema = mongoose.Schema({
        email: {type: String, 'default':''}
        , hashed_password: {type: String, required: true, 'default':''}
        , name: {type: String, index: 'hashed', 'default':''}
        , salt: {type:String, required:true}
        , created_at: {type: Date, index: {unique: false}, 'default': Date.now}
        , updated_at: {type: Date, index: {unique: false}, 'default': Date.now}
    });
}

// 입력된 칼럼의 값이 있는지 확인
UserSchema.path('email').validate(function (email) {
    return email.length;
}, 'email 칼럼의 값이 없습니다.');

UserSchema.path('hashed_password').validate(function (hashed_password) {
    return hashed_password.length;
}, 'hashed_password 칼럼의 값이 없습니다.');
```

그리고 패스포트를 초기화시 initialize() 미들웨어가 필요하며,
로그인 세션을 유지하기 위해 session() 미들웨어가 필요합니다.
여태 우리는 아이디와 비밀번호로 사용자 정보를 확인했습니다만,
이번에는 이메일 주소를 아이디로 사용하겠습니다. validate

```
// 모델 객체에서 사용할 수 있는 메소드 정의  
UserSchema.static('findByEmail', function(email, callback) {  
    return this.find({email:email}, callback);  
});
```

```
config.js  
  
module.exports = {  
  server_port: 3000,  
  db_url: 'mongodb://localhost:27017/local',  
  db_schemas: [  
    {file:'./user_schema', collection:'users5', schemaName:'UserSchema', modelName:'UserModel'}  
  ],  
  route_info: [  
  ]  
}
```

스키마 구성을 변경을 했으니, config파일 역시 수정을 해줍니다.

스키마 배열 객체에 들어있던 속성 중 collection을 user5로 변경을 합니다.

기존에 있던 route 정보는 지워주고 빈배열로 만들고, route 폴더의 user.js를 지워줍니다.

```
//===== Passport Strategy 설정 =====//  
  
var LocalStrategy = require('passport-local').Strategy;  
  
//패스포트 로그인 설정  
passport.use('local-login', new LocalStrategy({  
    usernameField : 'email',  
    passwordField : 'password',  
    passReqToCallback : true // 이 옵션을 설정하면 아래 콜백 함수의 첫번째 파라미터로 req 객체 전달됨  
}, function(req, email, password, done) {  
    console.log('passport의 local-login 호출됨 : ' + email + ', ' + password);  
  
    var database = app.get('database');  
    database.UserModel.findOne({ 'email' : email }, function(err, user) {  
        if (err) { return done(err); }  
  
        // 등록된 사용자 없는 경우  
        if (!user) {  
            console.log('계정이 일치하지 않음.');//  
            return done(null, false, req.flash('loginMessage', '등록된 계정이 없습니다.'));  
            | // 검증 콜백에서 두 번째 파라미터의 값을 false로 하여 인증 실패한 것으로 처리  
        }  
  
        // 비밀번호 비교하여 맞지 않는 경우  
        var authenticated = user.authenticate(password, user._doc.salt, user._doc.hashed_password);  
        if (!authenticated) {  
            console.log('비밀번호 일치하지 않음.');//  
            return done(null, false, req.flash('loginMessage', '비밀번호가 일치하지 않습니다.'));  
            | // 검증 콜백에서 두 번째 파라미터의 값을 false로 하여 인증 실패한 것으로 처리  
        }  
  
        // 정상인 경우  
        console.log('계정과 비밀번호가 일치함.');//  
        return done(null, user); // 검증 콜백에서 두 번째 파라미터의 값을 user 객체로 넣어 인증 성공한 것으로 처리  
    });  
});
```

```
// 패스포트 회원가입 설정
passport.use('local-signup', 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 = app.get('database');
        database.UserModel.findOne({ 'email' : email }, function(err, user) {
            // 예외 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.'));
            } 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
                });
            }
        });
    });
});
```

```
// 사용자 인증 성공 시 호출
// 사용자 정보를 이용해 세션을 만들
// 로그인 이후에 들어오는 요청은 deserializeUser 메소드 안에서 이 세션을 확인할 수 있음
passport.serializeUser(function(user, done) {
    console.log('serializeUser() 호출됨.');
    console.dir(user);

    done(null, user); // 이 인증 콜백에서 넘겨주는 user 객체의 정보를 이용해 세션 생성
});

// 사용자 인증 이후 사용자 요청 시마다 호출
// user -> 사용자 인증 성공 시 serializeUser 메소드를 이용해 만들었던 세션 정보가 파라미터로 넘어온 것임
passport.deserializeUser(function(user, done) {
    console.log('deserializeUser() 호출됨.');
    console.dir(user);

    // 사용자 정보 중 id나 email만 있는 경우 사용자 정보 조회 필요 – 여기에서는 user 객체 전체를 패스포트에서 관리
    // 두 번째 파라미터로 지정한 사용자 정보는 req.user 객체로 복원됨
    // 여기에서는 파라미터로 받은 user를 별도로 처리하지 않고 그대로 넘겨줌
    done(null, user);
});
```

위에 방식으로 스트래티지를 설정했으니, 인증방식이 결정되었죠.

웹 요청 처리과정에서는 인증정보가 로그인 요청시에만 전달됩니다.

로그인 이후에는 정보가 전달이 안되므로, 요청정보를 세션으로 확인해야합니다.

그래서 패스포트에서는 로그인 세션을 지원을 하는데, Serialize(저장), Deserialize(복원)이 바로 그 두가지입니다. 각각 사용자 정보를 세션에 저장하는 역할, 세션으로부터 사용자 정보를 복원해오는 역할을 합니다.

로그인과 회원가입 화면을 만들기 위한 라우팅함수 등록



클라이언트에서 사용하는 기능은 이 7가지 기능입니다. 각각 get과 post 방식으로 정보를 전달을 해줍니다. 단순 웹 문서 조회시 get방식, 사용자인증과 회원가입 진행 요청은 post 방식을 취했습니다.

```
// 로그인 화면 - login.ejs 템플릿을 이용해 로그인 화면이 보이도록 함
router.route('/login').get(function(req, res) {
    console.log('/login 패스 요청됨.');
    res.render('login.ejs', {message: req.flash('loginMessage')});
});

// 사용자 인증 - POST로 요청받으면 패스포트를 이용해 인증함
// 성공 시 /profile로 리다이렉트, 실패 시 /login으로 리다이렉트함
// 인증 실패 시 검증 콜백에서 설정한 플래시 메시지가 응답 페이지에 전달되도록 함
router.route('/login').post(passport.authenticate('local-login', {
    successRedirect : '/profile',
    failureRedirect : '/login',
// 회원가입 화면 - signup.ejs 템플릿을 이용해 회원가입 화면이 보이도록 함
router.route('/signup').get(function(req, res) {
    console.log('/signup 패스 요청됨.');
    res.render('signup.ejs', {message: req.flash('signupMessage')});
});

// 회원가입 - POST로 요청받으면 패스포트를 이용해 회원가입 유도함
// 인증 확인 후, 성공 시 /profile 리다이렉트, 실패 시 /signup으로 리다이렉트함
// 인증 실패 시 검증 콜백에서 설정한 플래시 메시지가 응답 페이지에 전달되도록 함
router.route('/signup').post(passport.authenticate('local-signup', {
    successRedirect : '/profile',
    failureRedirect : '/signup',
    failureFlash : true
}));
```

Post 방식을 사용시 authenticate메소드를 사용합니다.

```
// 프로필 화면 - 로그인 여부를 확인할 수 있도록 먼저 isLoggedIn 미들웨어 실행
router.route('/profile').get(function(req, res) {
    console.log('/profile 패스 요청됨.');

    // 인증된 경우, req.user 객체에 사용자 정보 있으며, 인증안된 경우 req.user는 false값임
    console.log('req.user 객체의 값');
    console.dir(req.user);

    // 인증 안된 경우
    if (!req.user) {
        console.log('사용자 인증 안된 상태임.');
        res.redirect('/');
        return;
    }

    // 인증된 경우
    console.log('사용자 인증된 상태임.');
    if (Array.isArray(req.user)) {
        res.render('profile.ejs', {user: req.user[0]._doc});
    } else {
        res.render('profile.ejs', {user: req.user});
    }
});
```

```
// 로그아웃 - 로그아웃 요청 시 req.logout() 호출함
router.route('/logout').get(function(req, res) {
    console.log('/logout 패스 요청됨.');

    req.logout();
    res.redirect('/');
});
```

프로필 화면과 로그아웃을 위한 코드입니다.

로그인과 회원가입 화면을 만들기 위한 뷰 템플릿 만들기

Examples



지난 시간에는 semantic-ui를 이용해보았습니다.
이번에는 부트스트랩을 이용하여 만들어보겠습니다.

index.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, height=device-height, initial-scale=1">
    <title>홈 화면</title>
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.2/css/bootstrap.min.css"> <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/font-awesome/4.0.3/css/font-awesome.min.css">
    <style>
      body { padding-top:80px; }
      @font-face {
        font-family: 'Nanum Gothic'; font-style: normal; font-weight: 400;
        src: url('../public/NanumGothic-Regular.eot');
        src: local('NanumGothic'), Show usages of Css Function Impl 'format('woff2')'
              url('../public/NanumGothic-Regular.eot?#iefix') format('embedded-opentype'),
              url('../public/NanumGothic-Regular.woff2') format('woff2'),
              url('../public/NanumGothic-Regular.woff') format('woff'),
              url('../public/NanumGothic-Regular.ttf') format('truetype');
      }
      h1, p, a { font-family: 'Nanum Gothic' }
    </style>
  </head>
  <body>
    <div class="container">
      <div class="jumbotron text-center">
        <h1><span class="fa fa-shopping-cart"></span> 쇼핑몰 홈</h1>
        <br>
        <p>로그인하세요.</p>
        <p>계정이 없으시면 회원가입하세요.</p>
        <br>
        <a href="/login" class="btn btn-default"><span class="fa fa-user"></span>로그인</a>
        <a href="/signup" class="btn btn-default"><span class="fa fa-user"></span>회원가입</a>
      </div>
    </div>
  </body>
</html>
```

login.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, height=device-height, initial-scale=1">
    <title>로그인 화면</title>
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.2/css/bootstrap.min.css"> <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/font-awesome/4.0.3/css/font-awesome.min.css">
    <style>
      body { padding-top:80px; }
    </style>
  </head>
  <body>
    <div class="container">
      <div class="col-sm-6 col-sm-offset-3">
        <h1><span class="fa fa-sign-in"></span> 로그인</h1>
        <!-- 인증 처리 후 메시지가 있으면 메시지 표시 -->
        <% if (message.length > 0) { %>
          <div class="alert alert-danger"><%= message %></div>
        <% } %>
        <form action="/login" method="post" >
          <div class="form-group">
            <label>이메일</label>
            <input type="text" class="form-control" name="email">
          </div>
          <div class="form-group">
            <label>비밀번호</label>
            <input type="password" class="form-control" name="password">
          </div>
          <button type="submit" class="btn btn-warning btn-lg">로그인</button>
        </form>
        <hr>
        <p>계정이 없으세요? <a href="/signup">회원가입하기</a></p>
        <p><a href="/">홈으로</a>.</p>
      </div>
    </div>
  </body>
</html>
```

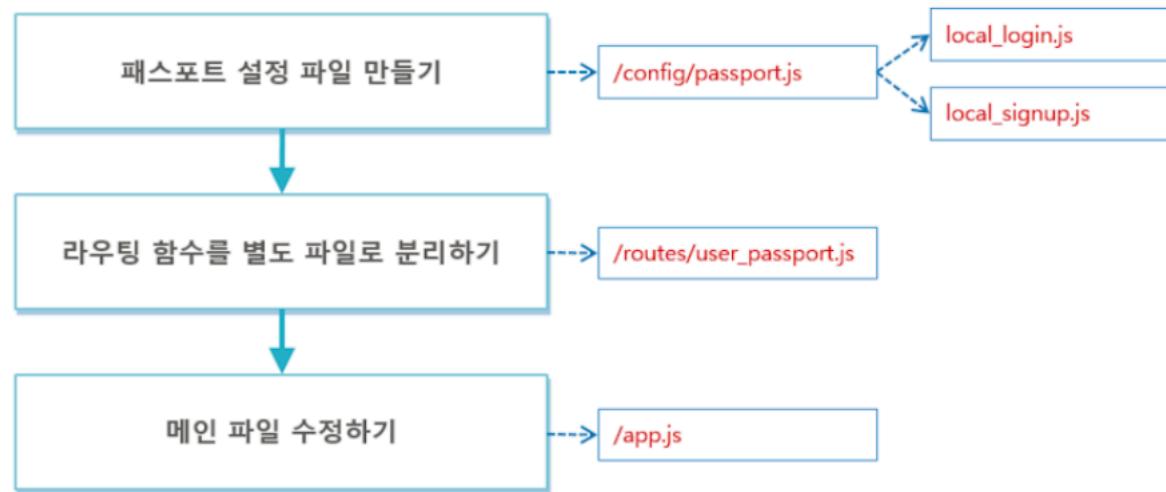
signup.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, height=device-height, initial-scale=1">
    <title>회원가입 화면</title>
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.2/css/bootstrap.min.css"> <!-- 브라우저 호환성 위해 -->
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/font-awesome/4.0.3/css/font-awesome.min.css">
    <style>
      body { padding-top:80px; }
    </style>
  </head>
  <body>
    <div class="container">
      <div class="col-sm-6 col-sm-offset-3">
        <h1><span class="fa fa-sign-in"></span> 회원가입</h1>
        <!-- 인증 처리 후 메시지가 있으면 메시지 표시 -->
        <% if (message.length > 0) { %>
          <div class="alert alert-danger"><%= message %></div>
        <% } %>
        <form action="/signup" method="post" >
          <div class="form-group">
            <label>이메일</label>
            <input type="text" class="form-control" name="email">
          </div>
          <div class="form-group">
            <label>비밀번호</label>
            <input type="password" class="form-control" name="password">
          </div>
          <div class="form-group">
            <label>별명</label>
            <input type="text" class="form-control" name="name">
          </div>
          <button type="submit" class="btn btn-warning btn-lg">회원가입</button>
        </form>
        <hr>
        <p>이미 계정이 있으신가요? <a href="/login">로그인하기</a></p>
        <p><a href="/">홈으로</a>.</p>
      </div>
    </div>
  </body>
```

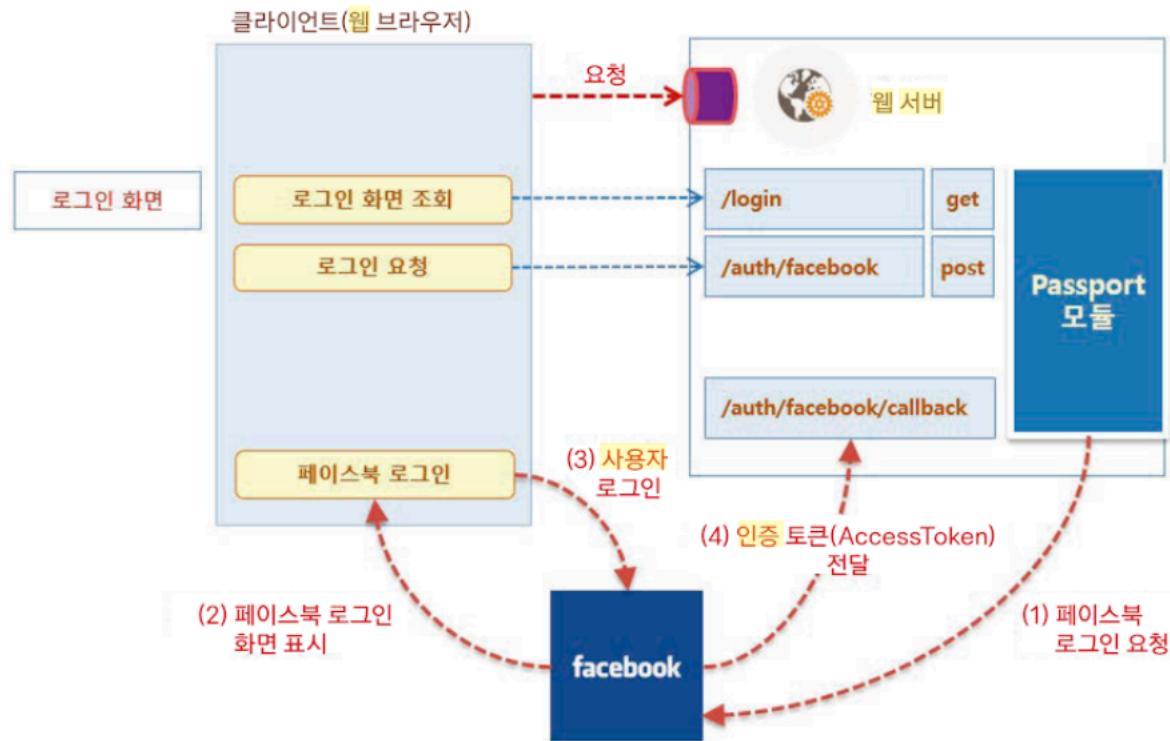
profile.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, height=device-height, initial-scale=1">
    <title>프로필 화면</title>
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.2/css/bootstrap.min.css"> <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/font-awesome/4.0.3/css/font-awesome.min.css">
    <style>
      body { padding-top:80px; }
    </style>
  </head>
  <body>
    <div class="container">
      <div class="page-header text-center">
        <h1><span class="fa fa-book"></span> 사용자 프로필</h1>
        <a href="/logout" class="btn btn-default btn-sm">로그아웃</a>
      </div>
      <br>
      <div class="row">
        <div class="col-sm-6">
          <div class="well">
            <h3><span class="fa fa-user"></span> 로컬 프로필 정보</h3>
            <br>
            <p>
              <strong>이메일</strong>: <%= user.email %>
              <br><br>
              <strong>별명</strong>: <%= user.name %>
            </p>
            <br>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

모듈화하기

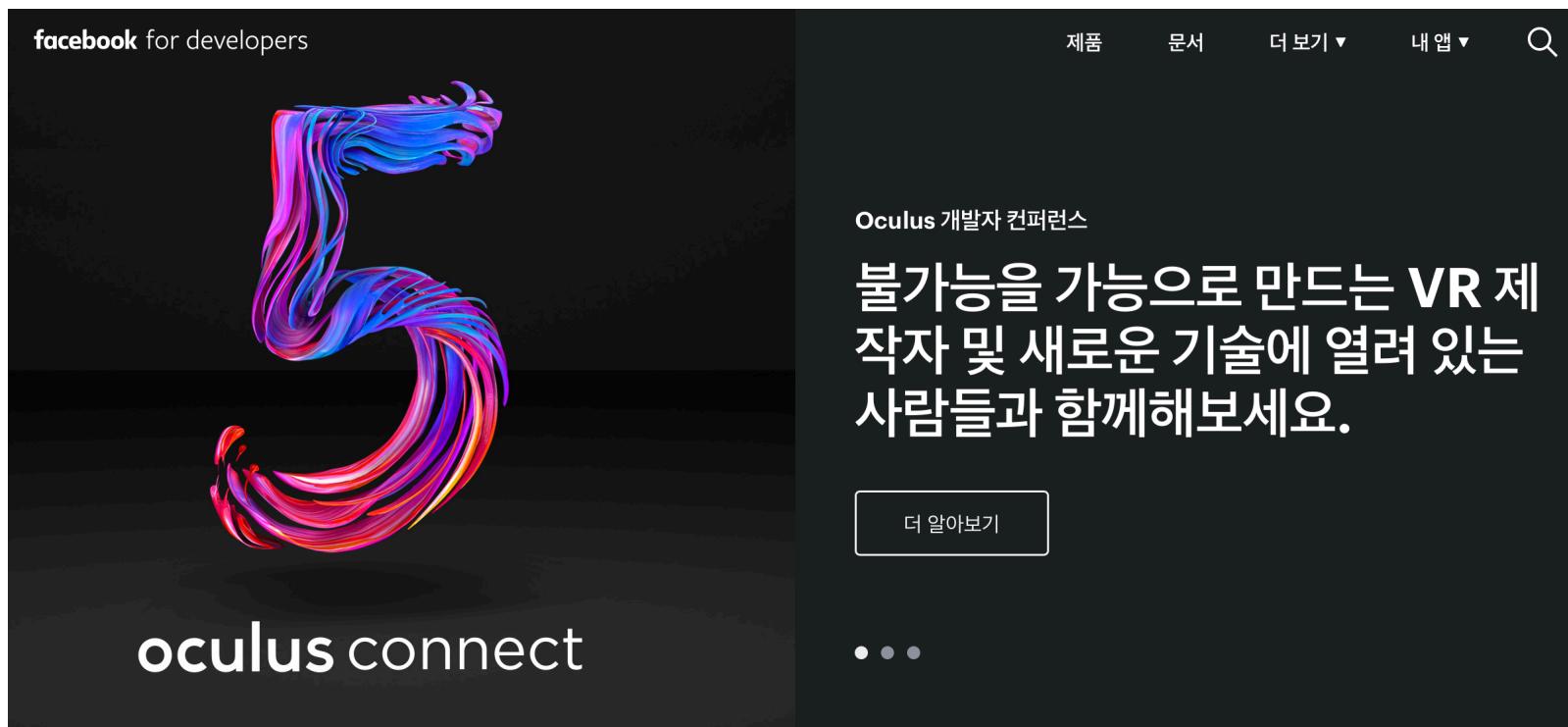


페이스북으로 로그인하기



이번에는 OAuth 인증으로 돌아가는 과정을 알아보겠습니다.

페이스북 개발자 홈페이지



먼저 우리는 developer.facebook.com으로 들어가서 사전준비를 합니다.

facebook.js

```
var FacebookStrategy = require('passport-facebook').Strategy;
var config = require('../config');

module.exports = function(app, passport) {
  return new FacebookStrategy({
    clientID: config.facebook.clientID,
    clientSecret: config.facebook.clientSecret,
    callbackURL: config.facebook.callbackURL,
    profileFields: ['id', 'emails', 'name']
  }, function(accessToken, refreshToken, profile, done) {
    console.log('passport의 facebook 호출됨.');
    console.dir(profile);

    var options = {
      criteria: { 'facebook.id': profile.id }
    };

    var database = app.get('database');
    database.UserModel.findOne(options, function (err, user) {
      if (err) return done(err);

      if (!user) {
        var user = new database.UserModel({
          name: profile.displayName,
          email: profile.emails[0].value,
          /* password : 'facebook', */
          provider: 'facebook',
          authToken: accessToken,
          facebook: profile._json
        });

        user.save(function (err) {
          if (err) console.log(err);
          return done(err, user);
        });
      } else {
        return done(err, user);
      }
    });
  });
};
```

passport.js

```
var local_login = require('./passport/local_login');
var local_signup = require('./passport/local_signup');
var facebook = require('./passport/facebook');

module.exports = function (app, passport) {
    console.log('config/passport 호출됨');
    // 사용자 인증 성공 시 호출
    // 사용자 정보를 이용해 세션을 만들
    // 로그인 이후에 들어오는 요청은 deserializeUser 메소드 안에서 이 세션을 확인할 수 있음
    passport.serializeUser(function (user, done) {
        console.log('serializeUser() 호출됨.');
        console.dir(user);

        done(null, user); // 이 인증 콜백에서 넘겨주는 user 객체의 정보를 이용해 세션 생성
    });

    // 사용자 인증 이후 사용자 요청 시마다 호출
    // user -> 사용자 인증 성공 시 serializeUser 메소드를 이용해 만들었던 세션 정보가 파라미터로 넘어온 것임
    passport.deserializeUser(function (user, done) {
        console.log('deserializeUser() 호출됨.');
        console.dir(user);

        // 사용자 정보 중 id나 email만 있는 경우 사용자 정보 조회 필요 - 여기에서는 user 객체 전체를 패스포트에서 관리
        // 두 번째 파라미터로 지정한 사용자 정보는 req.user 객체로 복원됨
        // 여기에서는 파라미터로 받은 user를 별도로 처리하지 않고 그대로 넘겨줌
        done(null, user);
    });

    passport.use('local-login', local_login);
    passport.use('local-signup', local_signup);
    passport.use('facebook', facebook(app, passport));
};

};|
```

config.js

```
module.exports = {
  server_port: 3000,
  db_url: 'mongodb://localhost:27017/local',
  db_schemas: [
    {file:'./user_schema', /*collection:'user5'*/ collection:'users6', schemaName:'UserSchema', modelName:'UserModel'}
  ],
  route_info: [
  ],
  facebook : {
    clientID: '0000000000000000',
    clientSecret: '0000000000000000000000000000000000000000000000000000000000000000',
    callbackURL: '/auth/facebook/callback'
  }
}
```

user_schema

```
// 스키마 정의
var UserSchema = mongoose.Schema({
  email: {type: String, 'default':''},
  hashed_password: {type: String, required: true, 'default':''},
  salt: {type:String, required:true},
  name: {type: String, index: 'hashed', 'default':''},
  created_at: {type: Date, index: {unique: false}, 'default': Date.now},
  updated_at: {type: Date, index: {unique: false}, 'default': Date.now},
  provider: {type: String, 'default' : ''},
  authToken: {type: String, 'default' : ''},
  facebook: {}
});
```

user_passport

```
// 패스포트 - 페이스북 인증 라우팅
router.route('/auth/facebook').get(passport.authenticate('facebook', {
    scope : 'email'
}));

// 패스포트 - 페이스북 인증 콜백 라우팅
router.route('/auth/facebook/callback').get(passport.authenticate('facebook', {
    successRedirect : '/profile',
    failureRedirect : '/'
}));
```

기존의 코드에 위의 코드들을 추가 시켜줍니다.

각각 페이스북에서 주어지는 client속성들과, 라우팅, 그리고 함수설정을 합니다.

이 모든 작업이 끝나면 HTML 파일에 페이스북 링크를 연동합니다.

실습문제

Q1. 위에 했던 실습내용에 페이스북이 아닌 구글로 로그인하기를 추가해봅니다.

hint) 구글로 로그인 버튼을 누르면 웹 서버 쪽으로 인증을 요청하며, 패스포트 모듈에서 구글로 로그인 처리를 진행합니다. 이를 위해 서버쪽에 미리 인증을 위한 함수를 만들어봅니다.

Q2. 위에 했던 실습내용에 페이스북이 아닌 트위터로 로그인하기를 추가해봅니다.

hint) 트위터로 로그인 버튼을 누르면 웹 서버 쪽으로 인증을 요청하며, 패스포트 모듈에서 트위터로 로그인 처리를 진행합니다. 이를 위해 서버쪽에 미리 인증을 위한 함수를 만들어봅니다.