

SNS

SNS

❖ 구현 기능

- ✓ 로그인
- ✓ 이미지 업로드
- ✓ 게시글 작성
- ✓ 해시태그 검색
- ✓ 팔로잉

프로젝트 구성

- ❖ 프로젝트 생성 – node_sns

- ❖ package.json 수정

```
"main": "app.js",
"scripts": {
  "start": "nodemon app",
  "test": "test"
}
```

- ❖ 필요한 패키지 설치

```
npm install express dotenv compression morgan file-stream-rotator multer cookie-parser  
express-session express-mysql-session mysql2 sequelize sequelize-cli nunjucks
```

```
npm install --save-dev nodemon
```

- ❖ sequelize 초기화

```
npx sequelize init
```

프로젝트 구성

- ❖ 프로젝트에 views 디렉토리, routes 디렉토리, public 디렉토리, passport 디렉토리 생성
- ❖ 시작 파일로 사용할 app.js 파일 생성
- ❖ 상수를 정의할 .env 파일을 생성
- ❖ 마지막 변경 날짜를 저장할 update.txt 파일을 생성

기본 설정

이메일

비밀번호

[로그인](#) [카카오톡](#) [회원가입](#)

Made by [ADAM](#)

기본 설정

- ❖ .env 파일에 작성

PORT=9000

COOKIE_SECRET=sns

HOST='localhost'

MYSQLPORT=3306

USERID='adam'

PASSWORD='wnddkd'

DATABASE='adam'

기본 설정

- ❖ app.js 파일을 작성

```
const express = require('express');

const dotenv = require('dotenv');
dotenv.config();

//서버 설정
const app = express();
app.set('port', process.env.PORT);

//로그 출력 설정
const fs = require('fs');
const path = require('path');

//static 파일의 경로 설정
app.use(express.static(path.join(__dirname, 'public')));

//view template 설정
const nunjucks = require('nunjucks');
app.set('view engine', 'html');
nunjucks.configure('views', {
  express:app,
  watch: true,
});
```

기본 설정

- ❖ app.js 파일을 작성

```
const morgan = require('morgan');
const FileStreamRotator = require('file-stream-rotator');

const logDirectory = path.join(__dirname, 'log');

// 로그 디렉토리 생성
fs.existsSync(logDirectory) || fs.mkdirSync(logDirectory);

// 로그 파일 옵션 설정
const accessLogStream = FileStreamRotator.getStream({
  date_format: 'YYYYMMDD',
  filename: path.join(logDirectory, 'access-%DATE%.log'),
  frequency: 'daily',
  verbose: false
});

// 로그 설정
app.use(morgan('combined', {stream: accessLogStream}));

// 출력하는 파일 압축해서 전송
const compression = require('compression');
app.use(compression());
```

기본 설정

- ❖ app.js 파일을 작성

```
//post 방식의 파라미터 읽기
var bodyParser = require('body-parser');
app.use(bodyParser.json());      // to support JSON-encoded bodies
app.use(bodyParser.urlencoded({    // to support URL-encoded bodies
  extended: true
}));

//쿠키 설정
const cookieParser = require('cookie-parser');
app.use(cookieParser(process.env.COOKIE_SECRET));
```

기본 설정

- ❖ app.js 파일을 작성

```
//세션 설정
const session = require("express-session");
var options = {
    host :process.env.HOST,
    port : process.env.MYSQLPORT,
    user : process.env.USERID,
    password : process.env.PASSWORD,
    database : process.env.DATABASE
};

const MySQLStore = require('express-mysql-session')(session);

app.use(
    session({
        secret: process.env.COOKIE_SECRET,
        resave: false,
        saveUninitialized: true,
        store : new MySQLStore(options)
    })
);
```

기본 설정

- ❖ app.js 파일을 작성

```
//라우터 설정
const pageRouter = require('./routes/page');
app.use ('/',pageRouter);

//에러가 발생한 경우 처리
app.use((req, res, next) => {
  const err = new Error(`${req.method} ${req.url} 라우터가 없습니다.`);
  err.status = 404;
  next(err);
});

//에러가 발생한 경우 처리
app.use((err, req, res, next) => {
  res.locals.message = err.message;
  res.locals.error = process.env.NODE_ENV !== 'production' ? err : {};
  res.status(err.status || 500);
  res.render('error');
});

app.listen(app.get('port'), () => {
  console.log(app.get('port'), '번 포트에서 대기 중');
});
```

기본 설정

- ❖ routes 디렉토리에 page.js 파일을 생성하고 작성

```
const express = require('express');

const router = express.Router();

router.use((req, res, next) => {
  res.locals.user = null;
  res.locals.followerCount = 0;
  res.locals.followingCount = 0;
  res.locals.followerIdList = [];
  next();
});
```

기본 설정

- ❖ routes 디렉토리에 page.js 파일을 생성하고 작성

```
router.get('/profile', (req, res) => {
  res.render('profile', { title: '내 정보 - NodeSNS' });
});

router.get('/join', (req, res) => {
  res.render('join', { title: '회원가입 - NodeSNS' });
});

router.get('/', (req, res, next) => {
  const twits = [];
  res.render('main', {
    title: 'NodeSNS',
    twits,
  });
});

module.exports = router;
```

기본 설정

- ❖ views 디렉토리에 공통된 레이아웃을 위한 layout.html 파일을 생성하고 작성

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>{{title}}</title>
    <meta name="viewport" content="width=device-width, user-scalable=no">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <link rel="stylesheet" href="/main.css">
  </head>
  <body>
    <div class="container">
      <div class="profile-wrap">
        <div class="profile">
          {%- if user and user.id %} 
            <div class="user-name">{{'안녕하세요! ' + user.nick + '님'}}</div>
            <div class="half">
              <div>팔로잉</div>
              <div class="count following-count">{{followingCount}}</div>
            </div>
            <div class="half">
              <div>팔로워</div>
              <div class="count follower-count">{{followerCount}}</div>
            </div>
          {%- endif %}
        </div>
      </div>
    </div>
  </body>
</html>
```

기본 설정

- ❖ views 디렉토리에 공통된 레이아웃을 위한 layout.html 파일을 생성하고 작성

```
<input id="my-id" type="hidden" value="{{user.id}}>
<a id="my-profile" href="/profile" class="btn">내 프로필</a>
<a id="logout" href="/auth/logout" class="btn">로그아웃</a>
{% else %}
    <form id="login-form" action="/auth/login" method="post">
        <div class="input-group">
            <label for="email">이메일</label>
            <input id="email" type="email" name="email" required autofocus>
        </div>
        <div class="input-group">
            <label for="password">비밀번호</label>
            <input id="password" type="password" name="password" required>
        </div>
        <a id="join" href="/join" class="btn">회원가입</a>
        <button id="login" type="submit" class="btn">로그인</button>
        <a id="kakao" href="/auth/kakao" class="btn">카카오톡</a>
    </form>
{% endif %}
</div>
```

기본 설정

- ❖ views 디렉토리에 공통된 레이아웃을 위한 layout.html 파일을 생성하고 작성

```
<footer>
    Made by ;
    <a href="https://ggangpae1.tistory.com/" target="_blank">ADAM</a>
</footer>
</div>
{% block content %}
{% endblock %}
</div>
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
window.onload = () => {
    if (new URL(location.href).searchParams.get('loginError')) {
        alert(new URL(location.href).searchParams.get('loginError'));
    }
};
</script>
{% block script %}
{% endblock %}
</body>
</html>
```

기본 설정

- ❖ views 디렉토리에 메인 화면을 위한 main.html 파일을 생성하고 작성

```
{% extends 'layout.html' %}

{% block content %}
<div class="timeline">
{% if user %}
<div>
<form id="twit-form" action="/post" method="post" enctype="multipart/form-data">
<div class="input-group">
<textarea id="twit" name="content" maxlength="140"></textarea>
</div>
<div class="img-preview">
<img id="img-preview" src="" style="display: none;" width="250" alt="미리보기">
<input id="img-url" type="hidden" name="url">
</div>
<div>
<label id="img-label" for="img">사진 업로드</label>
<input id="img" type="file" accept="image/*">
<button id="twit-btn" type="submit" class="btn">게시</button>
</div>
</form>
</div>
{% endif %}
```

기본 설정

- ❖ views 디렉토리에 메인 화면을 위한 main.html 파일을 생성하고 작성

```
<div class="twits">
    <form id="hashtag-form" action="/hashtag">
        <input type="text" name="hashtag" placeholder="태그 검색">
        <button class="btn">검색</button>
    </form>
    {% for twit in twits %}
        <div class="twit">
            <input type="hidden" value="{{twit.User.id}}" class="twit-user-id">
            <input type="hidden" value="{{twit.id}}" class="twit-id">
            <div class="twit-author">{{twit.User.nick}}</div>
            {% if not followerIdList.includes(twit.User.id) and twit.User.id !== user.id %}
                <button class="twit-follow">팔로우하기</button>
            {% endif %}
            <div class="twit-content">{{twit.content}}</div>
            {% if twit.img %}
                <div class="twit-img"></div>
            {% endif %}
        </div>
    {% endfor %}
</div>
{% endblock %}
```

기본 설정

- ❖ views 디렉토리에 메인 화면을 위한 main.html 파일을 생성하고 작성

```
% block script %}
<script>
if (document.getElementById('img')) {
  document.getElementById('img').addEventListener('change', function(e) {
    const formData = new FormData();
    console.log(this, this.files);
    formData.append('img', this.files[0]);
    axios.post('/post/img', formData)
      .then((res) => {
        document.getElementById('img-url').value = res.data.url;
        document.getElementById('img-preview').src = res.data.url;
        document.getElementById('img-preview').style.display = 'inline';
      })
      .catch((err) => {
        console.error(err);
      });
  });
}
```

기본 설정

- ❖ views 디렉토리에 메인 화면을 위한 main.html 파일을 생성하고 작성

```
document.querySelectorAll('.twit-follow').forEach(function(tag) {  
    tag.addEventListener('click', function() {  
        const myId = document.querySelector('#my-id');  
        if (myId) {  
            const userId = tag.parentNode.querySelector('.twit-user-id').value;  
            if (userId !== myId.value) {  
                if (confirm('팔로잉하시겠습니까?')) {  
                    axios.post(`/user/${userId}/follow`)  
                        .then(() => {  
                            location.reload();  
                        })  
                        .catch((err) => {  
                            console.error(err);  
                        });  
                }  
            }  
        }  
    });  
});  
</script>  
{% endblock %}
```

기본 설정

- ❖ views 디렉토리에 사용자의 팔로워 와 팔로잉 중인 목록을 출력하기 위한 profile.html 파일을 생성하고 작성

```
{% extends 'layout.html' %}

{% block content %}
<div class="timeline">
<div class="followings half">
<h2>팔로잉 목록</h2>
{% if user.Followings %}
    {% for following in user.Followings %}
        <div>{{following.nick}}</div>
    {% endfor %}
    {% endif %}
</div>
<div class="followers half">
<h2>팔로워 목록</h2>
{% if user.Followers %}
    {% for follower in user.Followers %}
        <div>{{follower.nick}}</div>
    {% endfor %}
    {% endif %}
</div>
</div>
{% endblock %}
```

기본 설정

- ❖ views 디렉토리에 회원가입을 위한 join.html 파일을 생성하고 작성

```
{% extends 'layout.html' %}

{% block content %}
<div class="timeline">
<form id="join-form" action="/auth/join" method="post">
<div class="input-group">
<label for="join-email">이메일</label>
<input id="join-email" type="email" name="email"></div>
<div class="input-group">
<label for="join-nick">닉네임</label>
<input id="join-nick" type="text" name="nick"></div>
<div class="input-group">
<label for="join-password">비밀번호</label>
<input id="join-password" type="password" name="password">
</div>
<button id="join-btn" type="submit" class="btn">회원가입</button>
</form>
</div>
{% endblock %}
```

기본 설정

- ❖ views 디렉토리에 회원 가입을 위한 join.html 파일을 생성하고 작성

```
{% block script %}  
  <script>  
    window.onload = () => {  
      if (new URL(location.href).searchParams.get('error')) {  
        alert('이미 존재하는 이메일입니다.');//  
      }  
    };  
  </script>  
{% endblock %}
```

기본 설정

- ❖ views 디렉토리에 에러 메시지를 출력하기 위한 error.html 파일을 생성하고 작성

```
{% extends 'layout.html' %}

{% block content %}
<h1>{{message}}</h1>
<h2>{{error.status}}</h2>
<pre>{{error.stack}}</pre>
{% endblock %}
```

기본 설정

- ❖ public 디렉토리에 스타일 설정을 위한 main.css 파일을 생성하고 작성

```
* { box-sizing: border-box; }
html, body { margin: 0; padding: 0; height: 100%; }
.btn {
    display: inline-block;
    padding: 0 5px;
    text-decoration: none;
    cursor: pointer;
    border-radius: 4px;
    background: white;
    border: 1px solid silver;
    color: crimson;
    height: 37px;
    line-height: 37px;
    vertical-align: top;
    font-size: 12px;
}
input[type='text'], input[type='email'], input[type='password'], textarea {
    border-radius: 4px;
    height: 37px;
    padding: 10px;
    border: 1px solid silver;
}
```

기본 설정

- ❖ public 디렉토리에 스타일 설정을 위한 main.css 파일을 생성하고 작성

```
.container { width: 100%; height: 100%; }
@media screen and (min-width: 800px) {
    .container { width: 800px; margin: 0 auto; }
}
.input-group { margin-bottom: 15px; }
.input-group label { width: 25%; display: inline-block; }
.input-group input { width: 70%; }
.half { float: left; width: 50%; margin: 10px 0; }
#join { float: right; }
.profile-wrap {
    width: 100%;
    display: inline-block;
    vertical-align: top;
    margin: 10px 0;
}
```

기본 설정

- ❖ public 디렉토리에 스타일 설정을 위한 main.css 파일을 생성하고 작성

```
@media screen and (min-width: 800px) {  
    .profile-wrap { width: 290px; margin-bottom: 0; }  
}  
  
.profile {  
    text-align: left;  
    padding: 10px;  
    margin-right: 10px;  
    border-radius: 4px;  
    border: 1px solid silver;  
    background: lightcoral;  
}  
  
.user-name { font-weight: bold; font-size: 18px; }  
.count { font-weight: bold; color: crimson; font-size: 18px; }  
.timeline {  
    margin-top: 10px;  
    width: 100%;  
    display: inline-block;  
    border-radius: 4px;  
    vertical-align: top;  
}
```

기본 설정

- ❖ public 디렉토리에 스타일 설정을 위한 main.css 파일을 생성하고 작성

```
@media screen and (min-width: 800px) { .timeline { width: 500px; } }

#twit-form {
    border-bottom: 1px solid silver;
    padding: 10px;
    background: lightcoral;
    overflow: hidden;
}

#img-preview { max-width: 100%; }

#img-label {
    float: left;
    cursor: pointer;
    border-radius: 4px;
    border: 1px solid crimson;
    padding: 0 10px;
    color: white;
    font-size: 12px;
    height: 37px;
    line-height: 37px;
}
```

기본 설정

- ❖ public 디렉토리에 스타일 설정을 위한 main.css 파일을 생성하고 작성

```
#img { display: none; }
#twit { width: 100%; min-height: 72px; }
#twit-btn {
    float: right;
    color: white;
    background: crimson;
    border: none;
}
.twit {
    border: 1px solid silver;
    border-radius: 4px;
    padding: 10px;
    position: relative;
    margin-bottom: 10px;
}
```

기본 설정

- ❖ public 디렉토리에 스타일 설정을 위한 main.css 파일을 생성하고 작성

```
.twit-author { display: inline-block; font-weight: bold; margin-right: 10px; }
.twit-follow {
    padding: 1px 5px;
    background: #fff;
    border: 1px solid silver;
    border-radius: 5px;
    color: crimson;
    font-size: 12px;
    cursor: pointer;
}
.twit-img { text-align: center; }
.twit-img img { max-width: 75%; }
.error-message { color: red; font-weight: bold; }
#search-form { text-align: right; }
#join-form { padding: 10px; text-align: center; }
#hashtag-form { text-align: right; }
footer { text-align: center; }
```

데이터베이스 설정

❖ 테이블 구조

- ✓ user 테이블
 - 이메일
 - 닉네임
 - 비밀번호
 - 로그인 방법: local 과 kakao로 구분
 - 카카오 아이디
 - 생성 시간
 - 수정 시간
 - 삭제 시간

데이터베이스 설정

❖ 테이블 구조

- ✓ Post 테이블
 - 게시글 내용
 - 이미지 파일의 경로
- ✓ HashTag 테이블
 - 태그 이름

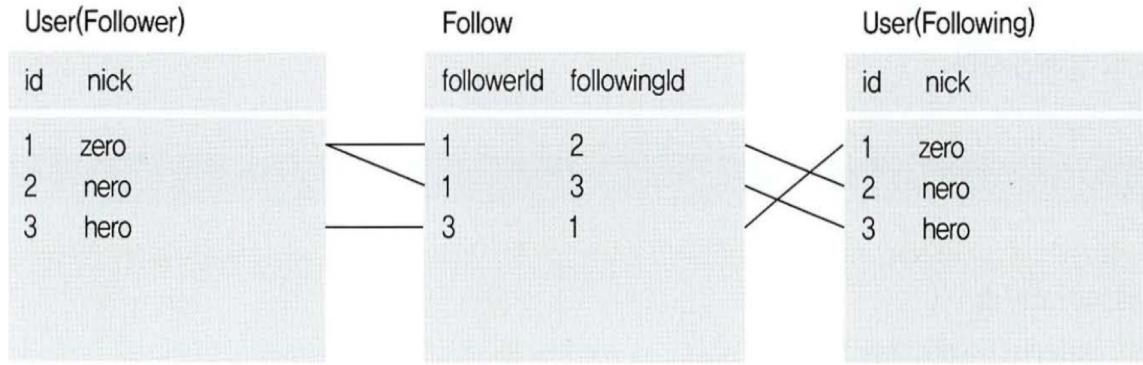
데이터베이스 설정

❖ 테이블 구조

✓ 테이블 사이의 관계

- User 와 Post는 1:N 관계
- User 와 User는 N:M 관계
- HashTag 와 Post는 N:M 관계

같은 테이블 간 N:M



데이터베이스 설정

- ❖ models 디렉토리에 user 테이블의 모델을 위한 user.js 파일을 생성하고 작성

```
const Sequelize = require('sequelize');

module.exports = class User extends Sequelize.Model {
  static init(sequelize) {
    return super.init({
      email: {
        type: Sequelize.STRING(40),
        allowNull: true,
        unique: true,
      },
      nick: {
        type: Sequelize.STRING(15),
        allowNull: false,
      },
      password: {
        type: Sequelize.STRING(100),
        allowNull: true,
      },
      provider: {
        type: Sequelize.STRING(10),
        allowNull: false,
        defaultValue: 'local',
      },
    });
  }
}
```

데이터베이스 설정

- ❖ models 디렉토리에 user 테이블의 모델을 위한 user.js 파일을 생성하고 작성

```
snsId: {  
    type: Sequelize.STRING(30),  
    allowNull: true,  
},  
, {  
    sequelize,  
    timestamps: true,  
    underscored: false,  
    modelName: 'User',  
    tableName: 'users',  
    paranoid: true,  
    charset: 'utf8',  
    collate: 'utf8_general_ci',  
});  
}
```

데이터베이스 설정

- ❖ models 디렉토리에 user 테이블의 모델을 위한 user.js 파일을 생성하고 작성

```
static associate(db) {  
    db.UserhasMany(db.Post);  
    db.User.belongsToMany(db.User, {  
        foreignKey: 'followingId',  
        as: 'Followers',  
        through: 'Follow',  
    });  
    db.User.belongsToMany(db.User, {  
        foreignKey: 'followerId',  
        as: 'Followings',  
        through: 'Follow',  
    });  
}
```

데이터베이스 설정

- ❖ models 디렉토리에 post 테이블의 모델을 위한 post.js 파일을 생성하고 작성

```
const Sequelize = require('sequelize');

module.exports = class Post extends Sequelize.Model {
  static init(sequelize) {
    return super.init({
      content: {
        type: Sequelize.STRING(140),
        allowNull: false,
      },
      img: {
        type: Sequelize.STRING(200),
        allowNull: true,
      },
    }, {
      sequelize,
      timestamps: true,
      underscored: false,
      modelName: 'Post',
      tableName: 'posts',
      paranoid: false,
      charset: 'utf8mb4',
      collate: 'utf8mb4_general_ci',
    });
  }
}
```

데이터베이스 설정

- ❖ models 디렉토리에 post 테이블의 모델을 위한 post.js 파일을 생성하고 작성

```
static associate(db) {  
    db.Post.belongsTo(db.User);  
    db.Post.belongsToMany(db.Hashtag, { through: 'PostHashtag' });  
}  
};
```

데이터베이스 설정

- ❖ models 디렉토리에 hashtag 테이블의 모델을 위한 hashtag.js 파일을 생성하고 작성

```
const Sequelize = require('sequelize');

module.exports = class Hashtag extends Sequelize.Model {
  static init(sequelize) {
    return super.init({
      title: {
        type: Sequelize.STRING(15),
        allowNull: false,
        unique: true,
      },
    }, {
      sequelize,
      timestamps: true,
      underscored: false,
      modelName: 'Hashtag',
      tableName: 'hashtags',
      paranoid: false,
      charset: 'utf8mb4',
      collate: 'utf8mb4_general_ci',
    });
  }
}
```

데이터베이스 설정

- ❖ models 디렉토리에 hashtag 테이블의 모델을 위한 hashtag.js 파일을 생성하고 작성

```
static associate(db) {  
    db.Hashtag.belongsToMany(db.Post, { through: 'PostHashtag' });  
}  
};
```

데이터베이스 설정

- ❖ models 디렉토리의 index.js 파일의 데이터베이스 설정을 수정

```
const Sequelize = require('sequelize');
const env = process.env.NODE_ENV || 'development';
const config = require('../config/config')[env];
const User = require('./user');
const Post = require('./post');
const Hashtag = require('./hashtag');

const db = {};
const sequelize = new Sequelize(
  config.database, config.username, config.password, config,
);

db.sequelize = sequelize;
db.User = User;
db.Post = Post;
db.Hashtag = Hashtag;
```

데이터베이스 설정

- ❖ models 디렉토리의 index.js 파일의 데이터베이스 설정을 수정

```
User.init(sequelize);
Post.init(sequelize);
Hashtag.init(sequelize);
```

```
User.associate(db);
Post.associate(db);
Hashtag.associate(db);
```

```
module.exports = db;
```

데이터베이스 설정

- ❖ config 디렉토리의 config.js 파일의 접속 정보 수정

```
{  
  "development": {  
    "username": "adam",  
    "password": "wnddkd",  
    "database": "adam",  
    "host": "127.0.0.1",  
    "dialect": "mysql"  
  },  
  "test": {  
    "username": "adam",  
    "password": "wnddkd",  
    "database": "adam",  
    "host": "127.0.0.1",  
    "dialect": "mysql"  
  },  
  "production": {  
    "username": "adam",  
    "password": "wnddkd",  
    "database": "adam",  
    "host": "127.0.0.1",  
    "dialect": "mysql"  
  }  
}
```

데이터베이스 설정

- ❖ config 디렉토리의 config.js 파일의 접속 정보 수정

```
{  
  "development": {  
    "username": "adam",  
    "password": "wnddkd",  
    "database": "adam",  
    "host": "127.0.0.1",  
    "dialect": "mysql"  
  },  
  "test": {  
    "username": "adam",  
    "password": "wnddkd",  
    "database": "adam",  
    "host": "127.0.0.1",  
    "dialect": "mysql"  
  },  
  "production": {  
    "username": "adam",  
    "password": "wnddkd",  
    "database": "adam",  
    "host": "127.0.0.1",  
    "dialect": "mysql"  
  }  
}
```

데이터베이스 설정

- ❖ 데이터베이스 생성 – 데이터베이스가 없는 경우에만 수행

```
npx sequelize-cli db:create
```

- ❖ 테이블 생성

```
npm start
```

로그인 구현

이메일 |

비밀번호

[로그인](#) [카카오톡](#) [회원가입](#)

Made by [ADAM](#)

안녕하세요! 아담님

팔로잉

0

팔로워

0

[내 프로필](#)

[로그아웃](#)

Made by [ADAM](#)

사진 업로드

태그 검색

검색

로그인 구현

- ❖ Passport 모듈

- ✓ 세션과 쿠키 처리와 같은 작업을 단순화 해줌
- ✓ Social 로그인 구현 처리 가능

- ❖ 모듈 설치

```
npm install passport passport-local passport-kakao bcrypt
```

로그인 구현

- ❖ app.js 파일에 Passport 모듈 연결하는 코드를 추가

```
const passport = require('passport');
const passportConfig = require('./passport');
passportConfig();
app.use(passport.initialize());
app.use(passport.session());
```

로그인 구현

- ❖ passport 디렉토리에 Passport 설정을 위한 index.js 파일을 생성하고 작성

```
const passport = require('passport');
const local = require('./localStrategy');
const kakao = require('./kakaoStrategy');
const User = require('../models/user');

module.exports = () => {
    //로그인 했을 때 사용자 정보 객체를 이용해서 사용자의 아이디를 deserializeUser에게 넘김
    passport.serializeUser((user, done) => {
        done(null, user.id);
    });

    passport.deserializeUser((id, done) => {
        //데이터베이스에서 id에 해당하는 값이 있는지 찾아서 세션에 저장
        User.findOne({ where: { id } })
            .then(user => done(null, user))
            .catch(err => done(err));
    });

    local();
    kakao();
};
```

로그인 구현

- ❖ routes 디렉토리에 필터링을 위한 middlewares.js 파일을 생성하고 작성

```
exports.isLoggedIn = (req, res, next) => {
    //로그인 되어 있으면 다음 라우터 처리를 수행하고 그렇지 않으면 에러 발생
    if (req.isAuthenticated()) {
        next();
    } else {
        res.status(403).send('로그인 필요');
    }
};

exports.isNotLoggedIn = (req, res, next) => {
    //로그인 되어 있지 않았다면 다음으로 넘어가고 그렇지 않으면 리다이렉트
    if (!req.isAuthenticated()) {
        next();
    } else {
        const message = encodeURIComponent('로그인한 상태입니다.');
        res.redirect(`/?error=${message}`);
    }
};
```

로그인 구현

- ❖ routes 디렉토리의 page.js 파일의 내용 수정

```
const express = require('express');

const {isLoggedIn, isNotLoggedIn} = require('./middlewares');

const router = express.Router();

router.use((req, res, next) => {
  res.locals.user = req.user;
  res.locals.followerCount = 0;
  res.locals.followingCount = 0;
  res.locals.followerIdList = [];
  next();
});

router.get('/profile', isLoggedIN, (req, res) => {
  res.render('profile', { title: '내 정보 - NodeSNS' });
});

router.get('/join', isNotLoggedIn, (req, res) => {
  res.render('join', { title: '회원가입 - NodeSNS' });
});
```

로그인 구현

- ❖ routes 디렉토리에 회원가입, 로그인, 로그아웃 처리를 위한 auth.js 파일을 생성하고 작성

```
const express = require('express');
const passport = require('passport');
const bcrypt = require('bcrypt');
const { isLoggedIn, isNotLoggedIn } = require('./middlewares');
const User = require('../models/user');

const router = express.Router();
```

로그인 구현

- ❖ routes 디렉토리에 회원가입, 로그인, 로그아웃 처리를 위한 auth.js 파일을 생성하고 작성

```
router.post('/join', isNotLoggedIn, async (req, res, next) => {
  const { email, nick, password } = req.body;
  try {
    const exUser = await User.findOne({ where: { email } });
    if (exUser) {
      return res.redirect('/join?error=exist');
    }
    const hash = await bcrypt.hash(password, 12);
    await User.create({
      email,
      nick,
      password: hash,
    });
    return res.redirect('/');
  } catch (error) {
    console.error(error);
    return next(error);
  }
});
```

로그인 구현

- ❖ routes 디렉토리에 회원가입, 로그인, 로그아웃 처리를 위한 auth.js 파일을 생성하고 작성

```
router.post('/login', isLoggedIn, (req, res, next) => {
  passport.authenticate('local', (authError, user, info) => {
    if (authError) {
      console.error(authError);
      return next(authError);
    }
    if (!user) {
      return res.redirect(`/?loginError=${info.message}`);
    }
    return req.login(user, (loginError) => {
      if (loginError) {
        console.error(loginError);
        return next(loginError);
      }
      return res.redirect('/');
    });
  })(req, res, next); // 미들웨어 내의 미들웨어에는 (req, res, next)를 붙입니다.
});
```

로그인 구현

- ❖ routes 디렉토리에 회원가입, 로그인, 로그아웃 처리를 위한 auth.js 파일을 생성하고 작성

```
router.get('/logout', isLoggedIn, (req, res) => {
  req.logout();
  req.session.destroy();
  res.redirect('/');
});

router.get('/kakao', passport.authenticate('kakao'));

router.get('/kakao/callback', passport.authenticate('kakao', {
  failureRedirect: '/',
}), (req, res) => {
  res.redirect('/');
});

module.exports = router;
```

로그인 구현

- ❖ passport 디렉토리에 로컬 로그인 전략을 위한 localStrategy.js 파일을 생성하고 작성

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

const User = require('../models/user');
```

로그인 구현

- ❖ passport 디렉토리에 로컬 로그인 전략을 위한 localStrategy.js 파일을 생성하고 작성

```
module.exports = () => {
  passport.use(new LocalStrategy({
    usernameField: 'email',
    passwordField: 'password',
  }, async (email, password, done) => {
    try {
      const exUser = await User.findOne({ where: { email } });
      if (exUser) {
        const result = await bcrypt.compare(password, exUser.password);
        if (result) {
          done(null, exUser);
        } else {
          done(null, false, { message: '비밀번호가 일치하지 않습니다.' });
        }
      } else {
        done(null, false, { message: '가입되지 않은 회원입니다.' });
      }
    } catch (error) {
      console.error(error);
      done(error);
    }
  }));
};
```

로그인 구현

- ❖ passport 디렉토리에 카카오 로그인 전략을 위한 kakaoStrategy.js 파일을 생성하고 작성

```
const passport = require('passport');
const KakaoStrategy = require('passport-kakao').Strategy;

const User = require('../models/user');
```

로그인 구현

- ❖ passport 디렉토리에 카카오 로그인 전략을 위한 kakaoStrategy.js 파일을 생성하고 작성

```
module.exports = () => {
  passport.use(new KakaoStrategy({
    clientID: process.env.KAKAO_ID,
    callbackURL: '/auth/kakao/callback',
  }, async (accessToken, refreshToken, profile, done) => {
    console.log('kakao profile', profile);
    try {
      const exUser = await User.findOne({
        where: { snsId: profile.id, provider: 'kakao' },
      });
      if (exUser) {
        done(null, exUser);
      } else {
        const newUser = await User.create({
          email: profile._json && profile._json.kaccount_email,
          nick: profile.displayName,
          snsId: profile.id,
          provider: 'kakao',
        });
        done(null, newUser);
      }
    }
  })
}
```

로그인 구현

- ❖ passport 디렉토리에 카카오 로그인 전략을 위한 kakaoStrategy.js 파일을 생성하고 작성

```
    catch (error) {  
        console.error(error);  
        done(error);  
    }  
});  
};
```

로그인 구현

- ❖ app.js 파일에 로그인 관련 라우터 등록

```
const authRouter = require('./routes/auth');
app.use ('/auth',authRouter);
```

로그인 구현

- ❖ 카카오 클라이언트 아이디 발급
 - ✓ 카카오 개발자 페이지에 로그인: <https://developers.kakao.com/>



로그인 구현

- ❖ 카카오 클라이언트 아이디 발급
- ✓ 애플리케이션 추가

The screenshot shows the Kakao Developers application management interface. At the top, there is a dark header bar with the "kakao developers" logo on the left and navigation links like "내 애플리케이션", "제품", "문서", "도구", "포럼", an email link "itstudy@kakao.com", a search icon "Q", and language options "KOR ENG". Below the header, a sub-header "내 애플리케이션" is visible. The main content area displays a list of applications. At the top of this list, there is a blue button with a white plus sign and the text "애플리케이션 추가하기". The first application listed is "nodelogin", which is categorized as an "APP". It has a thumbnail, the name "nodelogin", and a small box containing "ID 744322", "OWNER", and "Web". The second application listed is "pythonjsonparsing", also categorized as an "APP". It has a thumbnail, the name "pythonjsonparsing", and a small box containing "ID 701558", "OWNER", and "Web".

로그인 구현

- ❖ 카카오 클라이언트 아이디 발급
- ✓ 애플리케이션 추가

애플리케이션 추가하기

앱 아이콘

파일 선택

이미지
업로드

JPG, GIF, PNG

권장 사이즈 128px, 최대 250KB

앱 이름

내 애플리케이션 이름

사업자명

사업자 정보와 동일한 이름

- 입력된 정보는 사용자가 카카오 로그인을 할 때 표시됩니다.
- 정보가 정확하지 않은 경우 서비스 이용이 제한될 수 있습니다.

취소

저장

로그인 구현

- ❖ 카카오 클라이언트 아이디 발급
 - ✓ REST API 키 복사

앱 키

| | |
|--------------|----------------------------------|
| 네이티브 앱 키 | 1c27cd7585258f6fee08e0748a5b95ae |
| REST API 키 | 6a92e72defaeb1bd45b361490ae7f2b8 |
| JavaScript 키 | 001fb4c571ae0f8a89b993d5aa1c5bbb |
| Admin 키 | 4672367f605b189d64c90c7990142cf6 |

로그인 구현

- ❖ 카카오 클라이언트 아이디 발급
 - ✓ 플랫폼 등록: 사용할 도메인을 등록

사이트 도메인

JavaScript SDK, 카카오링크, 카카오맵, 메시지 API 사용시 등록이 필요합니다.

여러개의 도메인은 줄바꿈으로 추가해주세요. 최대 10까지 등록 가능합니다. 추가 등록은 포럼(데브톡)으로 문의주세요.

예시: (O) <https://example.com> (X) <https://www.example.com>

http://localhost:9000

기본 도메인

기본 도메인은 첫 번째 사이트 도메인으로, 카카오링크와 카카오톡 메시지 API를 통해 발송되는 메시지의 Web 링크 기본값으로 사용됩니다.

http://localhost:9000

취소

저장

로그인 구현

- ❖ 카카오 클라이언트 아이디 발급
 - ✓ 로그인 활성화

The screenshot shows the Kakao Developers Platform interface for the 'nodelogin' application. The left sidebar lists various settings like App Configuration, Basic Information, Business, API Key, Platforms, and Team Management. The 'Kakao Login' section is currently selected. The main content area displays the 'Kakao Login' status as 'ON'. Below it, the 'Activation Settings' section shows the 'Status' as 'ON'. A descriptive text explains that using the Kakao Login API allows users to log in through KakaoTalk or the service provider's website. It also notes that the activation status can be changed even if the status is OFF.

앱 설정
요약 정보
일반
비즈니스
앱 키
플랫폼
팀 관리

제품 설정
카카오 로그인
동의항목
간편가입
카카오톡 채널
개인정보 국외이전

APP nodelogin ID 744322 OWNER Web

카카오 로그인 **ON** 동의 화면 미리보기

활성화 설정

상태 **ON**

카카오 로그인 API를 활용하면 사용자들이 번거로운 회원 가입 절차 대신, 카카오톡으로 서비스를 시작할 수 있습니다.
상태가 OFF일 때도 카카오 로그인 설정 항목을 변경하고 서버에 저장할 수 있습니다.
상태가 ON일 때만 실제 서비스에서 카카오 로그인 화면이 연결됩니다.

로그인 구현

- ❖ 카카오 클라이언트 아이디 발급
 - ✓ Redirect URI 등록

Redirect URI

카카오 로그인에서 사용할 OAuth Redirect URI를 설정합니다.

여러개의 URI를 줄바꿈으로 추가해주세요. (최대 10개)

REST API로 개발하는 경우 필수로 설정해야 합니다.

예시: (O) <https://example.com/oauth> (X) <https://www.example.com/oauth>

```
http://localhost:9000/auth/kakao/callback
```

취소

저장

로그인 구현

- ❖ 카카오 클라이언트 아이디 발급
- ✓ 수집 정보 항목 설정

| 제품 설정 | 항목 이름 | ID | 상태 | 설정 |
|-----------|-------------------------------------|------------------|---|----|
| 카카오 로그인 | 닉네임 | profile.nickname | <input checked="" type="radio"/> 필수 동의 | 설정 |
| 동의항목 | 프로필 사진 | profile.image | <input type="radio"/> 사용 안함 | 설정 |
| 간편가입 | 카카오계정(이메일) | account_email | <input checked="" type="radio"/> 선택 동의 [수집] | 설정 |
| 카카오톡 채널 | 성별 | gender | <input type="radio"/> 사용 안함 | 설정 |
| 개인정보 국외이전 | 연령대 | age_range | <input type="radio"/> 사용 안함 | 설정 |
| 연결 끊기 | 생일 | birthday | <input type="radio"/> 사용 안함 | 설정 |
| 사용자 프로퍼티 | 출생 연도 | birthyear | <input type="radio"/> 권한 없음 | |
| 보안 | 카카오계정(전화번호) | phone_number | <input type="radio"/> 권한 없음 | |
| 고급 | CI(연계정보) | account_ci | <input type="radio"/> 권한 없음 | |
| 메시지 | 카카오 서비스 내 친구목록(프로필사진, 닉네임, 즐겨찾기 포함) | friends | <input type="radio"/> 사용 안함 | 설정 |

로그인 구현

- ❖ 카카오 클라이언트 아이디 발급

- ✓ 복사한 REST API 키를 .env 파일에 설정

```
KAKAO_ID=6a92e72defaeb1bd45b361490ae7f2b8
```

이미지 업로드

- ❖ routes 디렉토리에 게시글 관련 라우팅 처리를 위한 post.js 파일을 생성하고 작성

```
const express = require('express');
const multer = require('multer');
const path = require('path');
const fs = require('fs');

const { Post, Hashtag } = require('../models');
const { isLoggedIn } = require('../middlewares');

const router = express.Router();

try {
  fs.readdirSync('uploads');
} catch (error) {
  console.error('uploads 폴더가 없어 uploads 폴더를 생성합니다.');
  fs.mkdirSync('uploads');
}
```

이미지 업로드

- ❖ routes 디렉토리에 게시글 관련 라우팅 처리를 위한 post.js 파일을 생성하고 작성

```
const upload = multer({  
  storage: multer.diskStorage({  
    destination(req, file, cb) {  
      cb(null, 'uploads/');  
    },  
    filename(req, file, cb) {  
      const ext = path.extname(file.originalname);  
      cb(null, path.basename(file.originalname, ext) + Date.now() + ext);  
    },  
  }),  
  limits: { fileSize: 5 * 1024 * 1024 },  
});  
  
router.post('/img', isLoggedIn, upload.single('img'), (req, res) => {  
  console.log(req.file);  
  res.json({ url: `/img/${req.file.filename}` });  
});
```

이미지 업로드

- ❖ routes 디렉토리에 게시글 관련 라우팅 처리를 위한 post.js 파일을 생성하고 작성

```
const upload2 = multer();
router.post('/', isLoggedIn, upload2.none(), async (req, res, next) => {
  try {
    const post = await Post.create({
      content: req.body.content,
      img: req.body.url,
      UserId: req.user.id,
    });
    const hashtags = req.body.content.match(/#[^#]+#/g);
    if (hashtags) {
      const result = await Promise.all(
        hashtags.map(tag => {
          return Hashtag.findOrCreate({
            where: { title: tag.slice(1).toLowerCase() },
          })
        }),
      );
      await post.addHashtags(result.map(r => r[0]));
    }
    res.redirect('/');
  } catch (err) {
    console.error(err);
    res.status(500).send('Internal Server Error');
  }
});
```

이미지 업로드

- ❖ routes 디렉토리에 게시글 관련 라우팅 처리를 위한 post.js 파일을 생성하고 작성

```
    } catch (error) {  
      console.error(error);  
      next(error);  
    }  
  );  
  
module.exports = router;
```

이미지 업로드

- ❖ routes 디렉토리의 page.js 수정

```
const express = require('express');

const {isLoggedIn, isNotLoggedIn} = require('./middlewares');

const {Post, User} = require('../models')

const router = express.Router();

router.use((req, res, next) => {
  res.locals.user = req.user;
  res.locals.followerCount = 0;
  res.locals.followingCount = 0;
  res.locals.followerIdList = [];
  next();
});

router.get('/profile', isLoggedIn, (req, res) => {
  res.render('profile', { title: '내 정보 - NodeSNS' });
});

router.get('/join', isNotLoggedIn, (req, res) => {
  res.render('join', { title: '회원가입 - NodeSNS' });
});
```

이미지 업로드

- ❖ routes 디렉토리의 page.js 수정

```
router.get('/', async (req, res, next) => {
  try {
    const posts = await Post.findAll({
      include: [
        { model: User,
          attributes: ['id', 'nick'],
        },
        { order: [['createdAt', 'DESC']]},
      ],
    });
    res.render('main', {
      title: 'NodeSNS',
      twits: posts,
    });
  } catch (err) {
    console.error(err);
    next(err);
  }
});

module.exports = router;
```

팔로우

- ❖ routes 디렉토리에 팔로우 관련 라우팅 처리를 위한 users.js 파일을 생성하고 작성

```
const express = require('express');

const { isLoggedIn } = require('./middlewares');
const User = require('../models/user');

const router = express.Router();

router.post('/:id/follow', isLoggedIn, async (req, res, next) => {
  try {
    const user = await User.findOne({ where: { id: req.user.id } });
    if (user) {
      await user.addFollowing(parseInt(req.params.id, 10));
      res.send('success');
    } else {
      res.status(404).send('no user');
    }
  } catch (error) {
    console.error(error);
    next(error);
  }
});

module.exports = router;
```

팔로우

- ❖ passport 디렉토리의 index.js 파일에 로그인 할 때 팔로우 정보를 가져오도록 수정

```
const passport = require('passport');
const local = require('./localStrategy');
const kakao = require('./kakaoStrategy');
const User = require('../models/user');

module.exports = () => {
  passport.serializeUser((user, done) => {
    done(null, user.id);
  });
}
```

팔로우

- ❖ passport 디렉토리의 index.js 파일에 로그인 할 때 팔로우 정보를 가져오도록 수정

```
passport.deserializeUser((id, done) => {
  User.findOne({
    where: { id },
    include: [
      {
        model: User,
        attributes: ['id', 'nick'],
        as: 'Followers',
      }, {
        model: User,
        attributes: ['id', 'nick'],
        as: 'Followings',
      }],
  })
  .then(user => done(null, user))
  .catch(err => done(err));
});

local();
kakao();
};
```

팔로우

- ❖ routes 디렉토리의 page.js 파일 설정

```
const express = require('express');
const { isLoggedIn, isNotLoggedIn } = require('./middlewares');
const { Post, User, Hashtag } = require('../models');

const router = express.Router();

router.use((req, res, next) => {
  res.locals.user = req.user;
  res.locals.followerCount = req.user ? req.user.Followers.length : 0;
  res.locals.followingCount = req.user ? req.user.Followings.length : 0;
  res.locals.followerIdList = req.user ? req.user.Followings.map(f => f.id) : [];
  next();
});

router.get('/profile', isLoggedIn, (req, res) => {
  res.render('profile', { title: '내 정보 - NodeBird' });
});

router.get('/join', isNotLoggedIn, (req, res) => {
  res.render('join', { title: '회원가입 - NodeBird' });
});
```

팔로우

- ❖ routes 디렉토리의 page.js 파일 수정

```
router.get('/', async (req, res, next) => {
  try {
    const posts = await Post.findAll({
      include: {
        model: User,
        attributes: ['id', 'nick'],
      },
      order: [['createdAt', 'DESC']],
    });
    res.render('main', {
      title: 'NodeBird',
      twits: posts,
    });
  } catch (err) {
    console.error(err);
    next(err);
  }
});
```

팔로우

- ❖ routes 디렉토리의 page.js 파일 수정

```
router.get('/hashtag', async (req, res, next) => {
  const query = req.query.hashtag;
  if (!query) {
    return res.redirect('/');
  }
  try {
    const hashtag = await Hashtag.findOne({ where: { title: query } });
    let posts = [];
    if (hashtag) {
      posts = await hashtag.getPosts({ include: [{ model: User }] });
    }
    return res.render('main', {
      title: `${query} | NodeBird`,
      twits: posts,
    });
  } catch (error) {
    console.error(error);
    return next(error);
  }
});

module.exports = router;
```

라우터 연결

- ❖ app.js 파일에 라우터 연결

```
router.get('/hashtag', async (req, res, next) => {
  const query = req.query.hashtag;
  if (!query) {
    return res.redirect('/');
  }
  try {
    const hashtag = await Hashtag.findOne({ where: { title: query } });
    let posts = [];
    if (hashtag) {
      posts = await hashtag.getPosts({ include: [{ model: User }] });
    }
    return res.render('main', {
      title: `${query} | NodeBird`,
      twits: posts,
    });
  } catch (error) {
    console.error(error);
    return next(error);
  }
});

module.exports = router;
```