Node.js
SNS 구현하기

실습 진행하기

2021.02.02

장혜선

01

화면

구현한 기능 살펴보기

02

로그인

코드 자세히 보기

01 화면

이메일

비밀번호

로그인 회원가입

이메일

닉네임

비밀번호

회원가입

안녕하세요! hi님

팔로잉 0

팔로워 1

내 프로필 로그아웃

사진 업로드 확인

태그 검색 검색

test 팔로우하기
지금은 공부 중 #공부 #STUDY

test 팔로우하기
테스트용 글 #테스트 #해시태그

hi
내가 만든 배경

게시글 삭제

Passport

인증 절차에 대한 로직을 편하게
코딩할 수 있도록 도와주는 Node.js 미들웨어

```
app.use(passport.initialize());
app.use(passport.session());
```

req 객체에 설정
req.session에 passport 정보 저장

serializeUser

로그인 시 실행, 세션 객체에
어떤 데이터를 저장할 지 결정

deserializeUser

서버로 들어오는 요청마다
세션 정보가 유효한 지를 검사

사용자 정보

```
module.exports = () => {
  passport.serializeUser((user, done) => {
    done(null, user.id);
  });
  passport.deserializeUser((id, done) => {
    User.findOne({
      where: { id } })
      .then(user => done(null, user))
      .catch(err => done(err));
  });

  local();
};
```

에러 발생 시 저장할 데이터

req.user에 저장

※ user.id만 저장하는 이유 :
모두 저장하면 용량 ↑ 데이터 일관성 문제 발생

정리

serializeUser
전달받은 사용자 정보 객체를
세션에 아이디로 저장

deserializeUser
세션에 저장한 아이디를 통해
사용자 정보 객체 불러옴

02 로그인

```
exports.isLoggedIn = (req, res, next) => {
    if (req.isAuthenticated()) {
      next();                                  로그인 중 - true
    } else {
      res.status(403).send('로그인 필요');


    }
  };

  exports.isNotLoggedIn = (req, res, next) => {
    if (!req.isAuthenticated()) {
      next();                                  로그인 X - false
    } else {
      const message = encodeURIComponent('로그인한 상태입니다.');
      res.redirect(`/?error=${message}`);
    }
  };
```

※ req.isAuthenticated( ) : 서버에 요청을 보낸 사용자가 인증이 되어있는
상태인지를 확인 후 'Boolean' 으로 알려준다.

사용법

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

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

02 로그인 - 회원가입

```
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);
  }
});
```

1. 기존에 같은 이메일로 가입한 사용자가 있는지 조회

2-1. 있으면 회원가입 페이지로
단, 주소 뒤에 에러 쿼리스트링 표시

2-2. 없으면 비밀번호 암호화

3. 사용자 정보 저장

```
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);
    }
  }));
};
```

첫 번째 인수 : 전략에 관한 설정

일치하는 속성명 적어줌

두 번째 인수 : 실제 전략 수행

1. 사용자 db에서 일치하는 이메일 찾음

2-1. 있으면 bcrypt의 compare함수로 비밀번호 비교

3-1. 비밀번호도 일치하면 done함수 두번째 인수로 사용자 정보 넣어 보냄

3-2. 로그인 실패 – 두번째 인수 사용하지 않음

2-2. 로그인 실패 – 두번째 인수 사용하지 않음

세 번째 인수 :
done 함수 (passport.authenticate 콜백 함수)

```
router.post('/login', isNotLoggedIn, (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);
});
```

로그인 전략 수행

1 2

값이 있으면 실패

done(error)

done(null, false, 에러 메시지)

값이 있으면 성공

done(null, exUser)

인수 제공해 호출

02 로그인 과정

로그인 요청 들어옴 ▶ passport.authenticate ▶ 로그인 전략 수행

▼

로그인 성공 시

사용자 정보 객체 + req.login ◀ passport.serializeUser ◀ req.session 사용자 아이디만 저장

▼

로그인 완료

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

req.user 객체 제거

세션 제거

03

이미지 업로드

코드 자세히 보기

03 이미지


```
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 하나를 업로드 받은 뒤 이미지의 저장 경로를 클라이언트로 응답

03 이미지

이미지 데이터 없어서

```
const upload2 = multer();
router.post('/', isLoggedIn, upload2.none(), async (req, res, next) => {
  try {
    console.log(req.user);
    const post = await Post.create({
      content: req.body.content,
      img: req.body.url,
      UserId: req.user.id,
    });
    const hashtags = req.body.content.match(/#[^\s#]+/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 (error) {
    console.error(error);
    next(error);
  }
});
```

게시글 데이터베이스에 저장

게시물 내용에서 해시태그 추출

데이터베이스에 저장
#떼고 소문자로 바꿔줌

해시태그 모델과 게시글 연결
결과값으로 모델만

04

추가 기능

코드 자세히 보기

req.params.id

```
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);
  }
});
```

팔로우할 사용자 데이터베이스 조회

현재 사용자와의 관계 지정

```
passport.deserializeUser((id, done) => {
  User.findOne({
    where: { id } })
    .then(user => done(null, user))
    .catch(err => done(err));
});
```

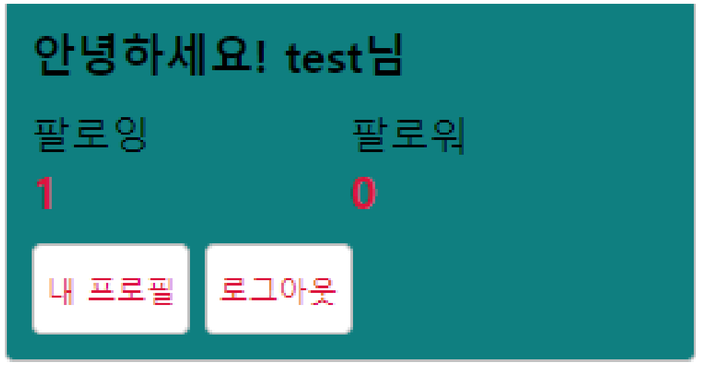
팔로잉, 팔로워 목록

```
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));
});
```

```
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();
});
```

값 있으면 해당 길이
아니면 0

게시물 작성자 아이디가 존재하지 않으면
팔로우 버튼 보여주기 위해서



04 추가 - 해시태그 조회

```
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} | sns`,
      twits: posts,
    });
  } catch (error) {
    console.error(error);
    return next(error);
  }
});
```

쿼리스트링으로 해시태그 이름 받음

값 없는 경우 => 메인 페이지

db에서 해시태그 검색

해시태그 존재 => 작성자 정보 합쳐서
모든 게시글 가져옴

조회 후 메인 페이지 렌더링 하면서
조회한 게시글만 twits에 넣어서 렌더링

```
router.delete('/:postId', isLoggedIn, async(req, res, next) =>{
  try{
    const post = await Post.destroy({where: {id: req.params.postId}});
    res.send('success');
  } catch (err){
    console.error(error);
    next(error);
  }
});
```

- 팔로우할 사용자 데이터베이스 조회
- 현재 사용자와의 관계 지정

발표를 들어주셔서

감사합니다

끝

2021.02.02

장혜선