오늘하루

당신의 하루를 기록하고 공유해보세요.

고급웹프로그래밍 기말 프로젝트

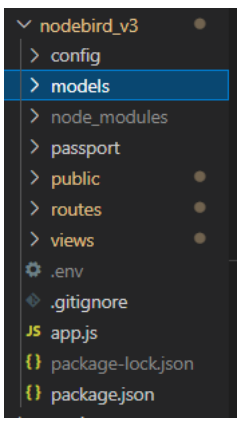
> 고급웹프로그래밍
> HW3 학기말 프로젝트
> 이은비
> 60191677

▼ 목차

What is <오늘 하루>

☝ 오늘 하루의 느낀 점 혹은 감정을 **한 마디로 기록하는 서비스**이다.
다른 유저들과 하루의 느낀 점을 공유할 수 있다.

Architecture



- config
- models
- passport
- public
- routes
- views
- app.js
- .env
- package.josn

How to start !

☝ npm start

http://localhost:8001 로 접속

Demo

메인 페이지

시나리오

사용자가 http://localhost:8001 로 접속한다.

시스템은 `메인 페이지`를 보여준다.

회원가입

시나리오

사용자가 [회원가입] 버튼을 누른다.

시스템은 `회원가입 페이지`를 보여준다.

사용자가 이메일, 닉네임, 비밀번호 입력창에 각각의 값을 입력한다.

사용자가 입력창 하단에 [회원가입] 버튼을 누른다.

시스템은 `메인페이지`로 이동한다.

로그인 - 일반

시나리오

사용자가 회원가입 한 이메일과 비밀번호를 각각의 입력창에 입력한다.

사용자가 [로그인] 버튼을 누른다.

시스템은 `메인 페이지`를 보여준다.

`메인 페이지`에 [내가 기록한 하루들], [로그아웃] , [홈으로] 버튼이 보여진다.

`메인 페이지`에 오늘 하루를 기록할 입력창과 [기록] 버튼이 보여진다.

`메인 페이지`에 다른 유저들이 남긴 기록이 보여지고, [수정하기], [삭제하기] 버튼이 비활성화된 채로 보여진다.

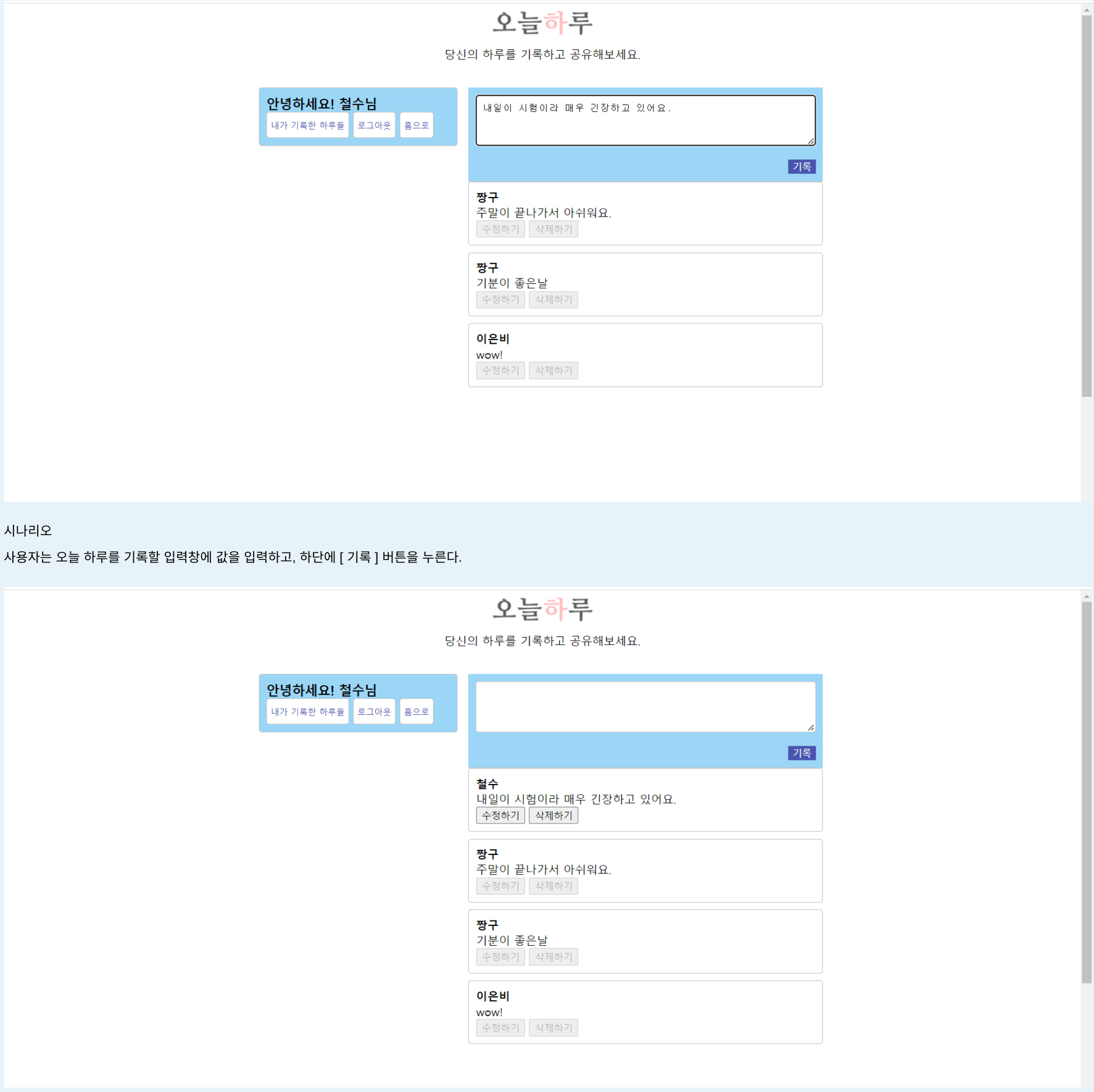
로그인 - 간편 (카카오톡)

사용자는 [카카오톡] 버튼을 누른다.

시스템은 카카오톡으로 로그인을 한다.

시스템은 `메인 페이지`로 이동한다.

메인 페이지 - 기록 남기기



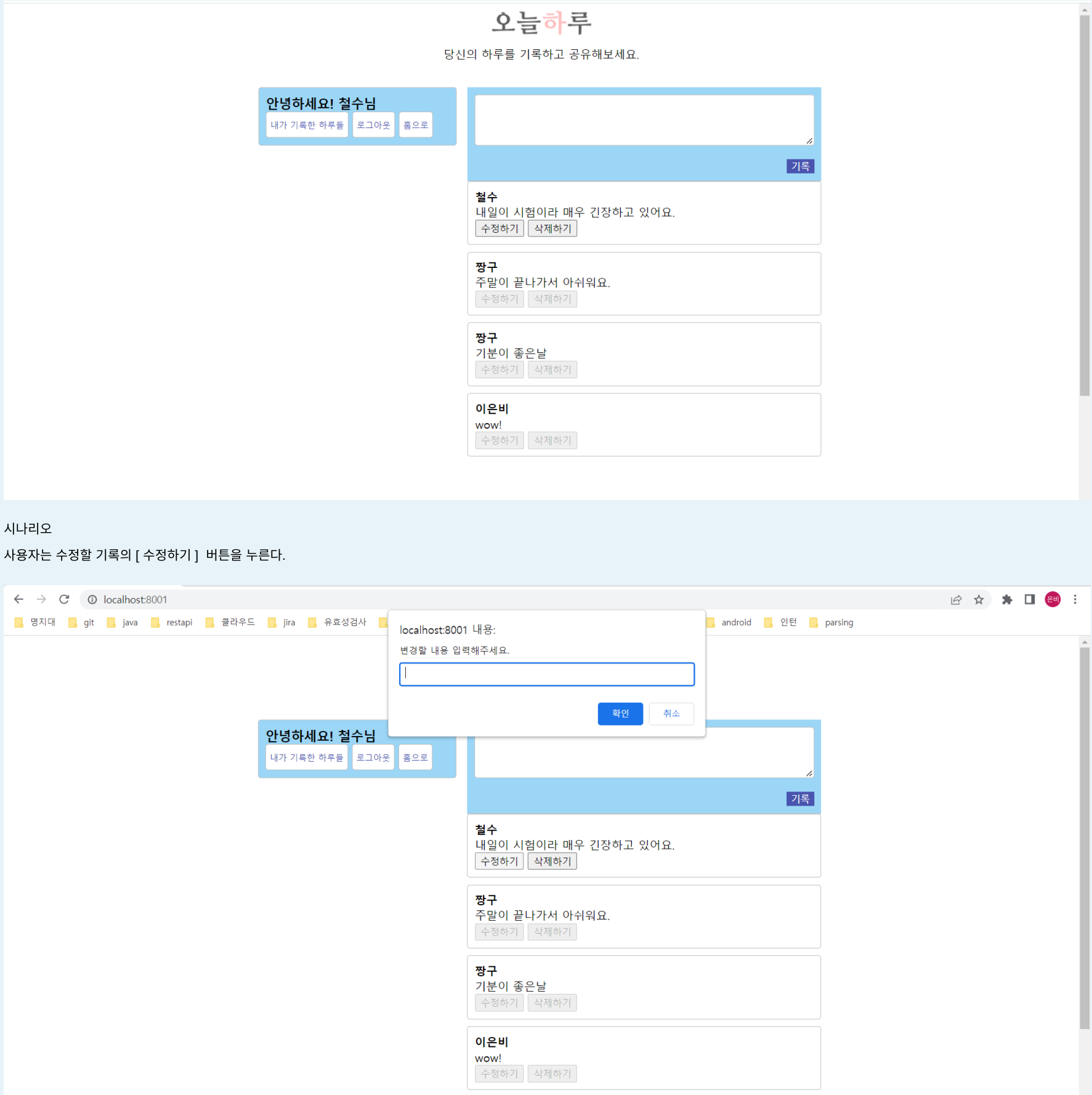
시나리오

사용자는 오늘 하루를 기록할 입력창에 값을 입력하고, 하단에 [기록] 버튼을 누른다.

시스템은 사용자가 남긴 기록을 입력창 하단 리스트에 보여준다.

사용자 닉네임, 사용자가 남긴 기록, [수정하기], [삭제하기] 버튼이 활성화 된 채 보여진다.

메인 페이지 - 기록 수정하기



시나리오

사용자는 수정할 기록의 [수정하기] 버튼을 누른다.

시스템은 prompt 창에 “변경할 내용 입력해주세요.” 라는 문구와 함께 입력창과 [확인], [취소] 버튼을 보여준다.

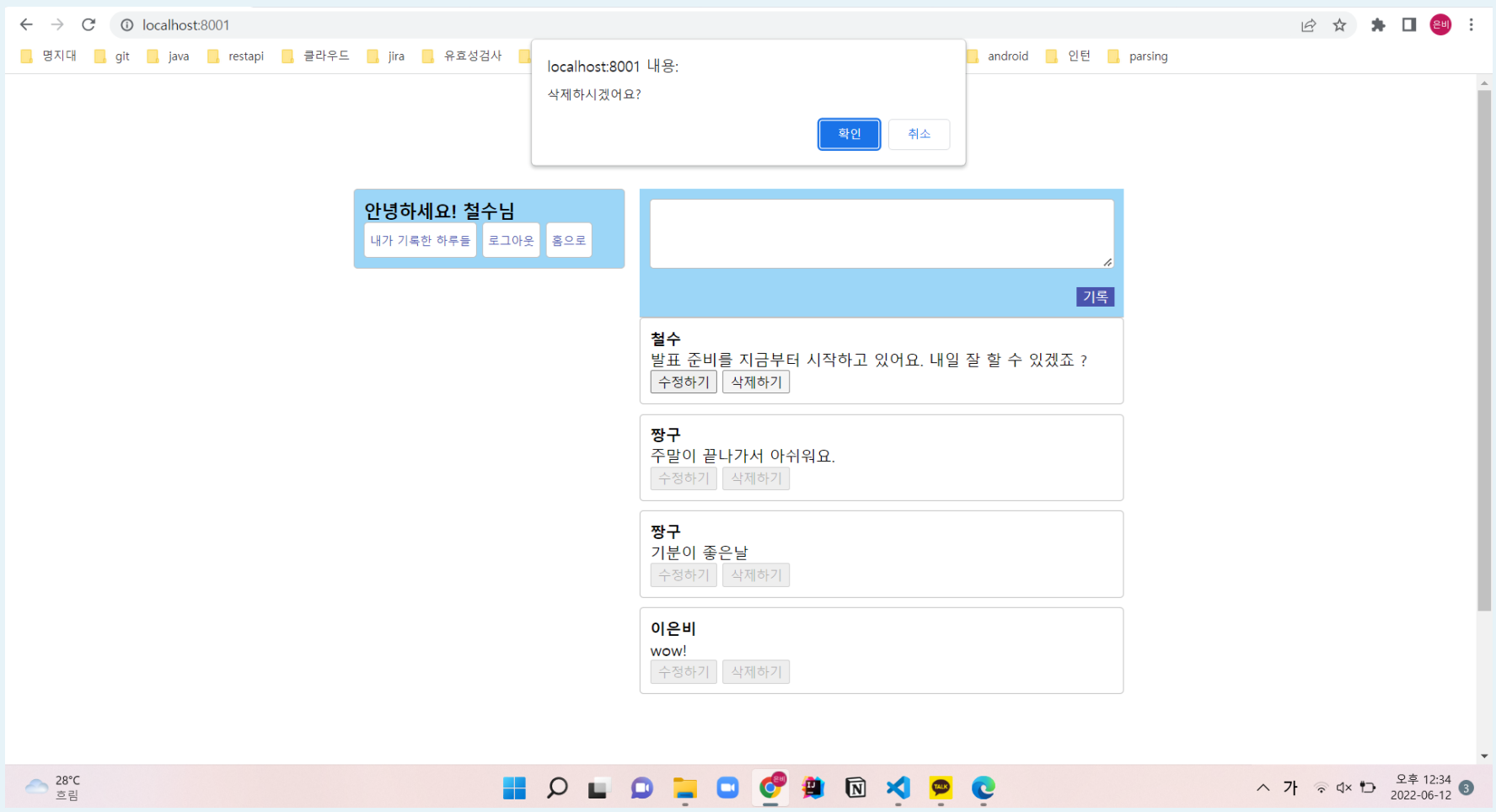
사용자는 변경할 내용을 입력하고 [확인] 버튼을 누른다.

시스템은 `메인 페이지`를 리다이렉트한다.

메인 페이지 - 기록 삭제하기

시나리오

사용자는 삭제할 기록의 [삭제하기] 버튼을 누른다.

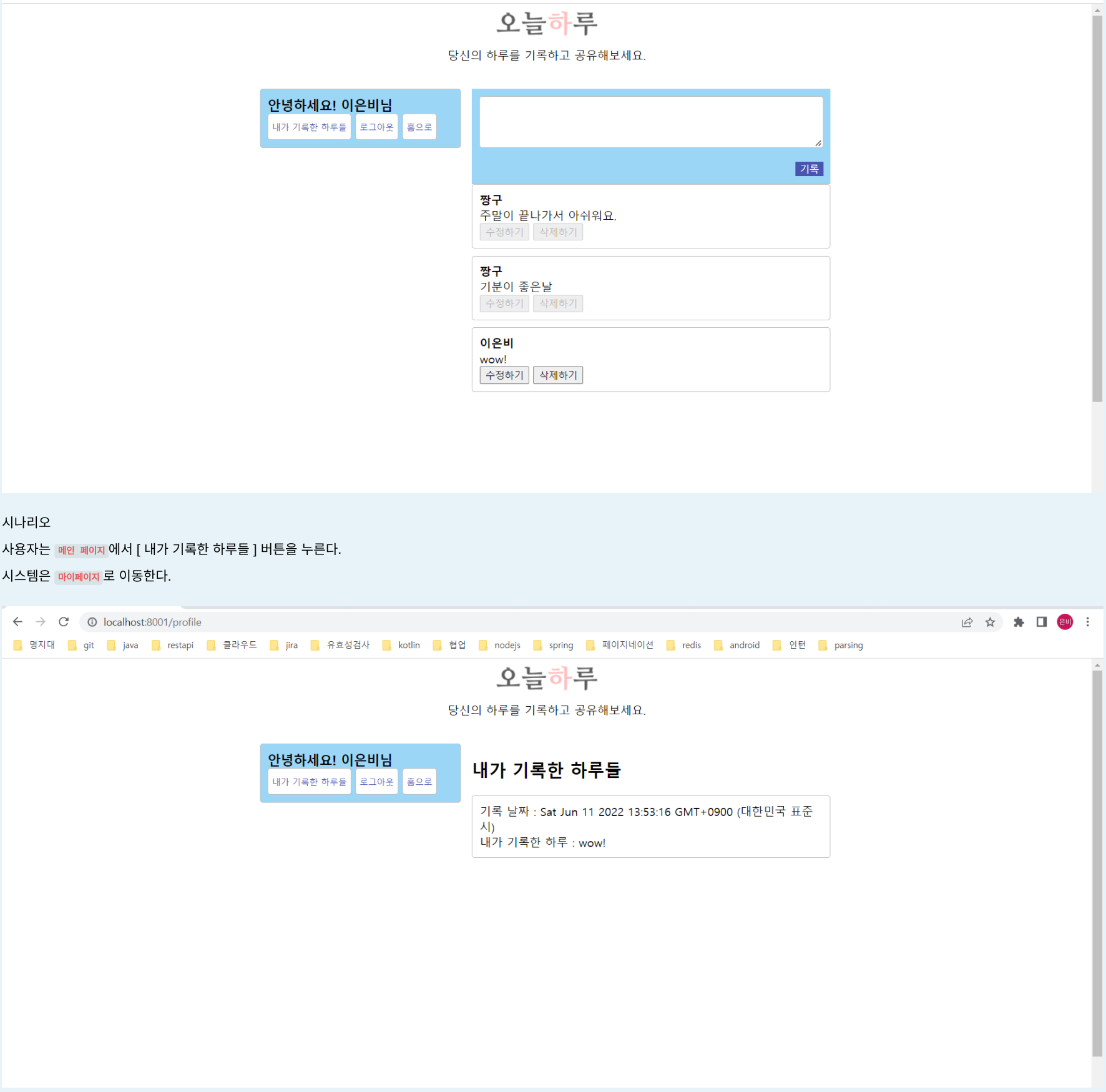


시스템은 confirm 창으로 “삭제하시겠어요?” 문구와 함께 [확인], [취소] 버튼을 보여준다.

사용자는 [확인] 버튼을 누른다.

시스템은 삭제 후, `메인 페이지`를 리다이렉트한다.

마이페이지 - 내가 기록한 하루들



시나리오

사용자는 `메인 페이지`에서 [내가 기록한 하루들] 버튼을 누른다.

시스템은 `마이페이지`로 이동한다.

시스템은 사용자가 기록한 기록에 대해서만 리스트를 보여준다.

각각의 기록에는 [기록 날짜] , [기록 내용] 이 보여진다.

사용자는 [홈으로] 버튼을 누른다.

시스템은 `메인 페이지`로 이동한다.

소스 코드

app.js

```
// 필요한 모듈
const express = require('express');
const morgan = require('morgan');
const nunjucks = require('nunjucks');
const cookieParser = require('cookie-parser');
const path = require('path');
const session = require('express-session');
const dotenv = require('dotenv');
const passport = require('passport');

dotenv.config();

// import routers
const pageRouter = require('./routes/page');
const authRouter = require('./routes/auth');
const postRouter = require('./routes/post');
const { sequelize } = require('./models');

const app = express();
app.set('port', process.env.PORT || 8001);
app.set('view engine', 'html');
```

```
nunjucks.configure('views', {
  express: app,
  autoescape: true,
});

sequelize
  .sync({ force: false })
  .then(() => {
    console.log('데이터베이스 연결 성공');
  })
  .catch((err) => {
    console.error(err);
  });

app.use(morgan('dev'));
app.use(express.static(path.join(__dirname, 'public')));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser(process.env.COOKIE_SECRET));
app.use(
  session({
    resave: false,
    saveUninitialized: false,
    secret: process.env.COOKIE_SECRET,
    cookie: {
      httpOnly: true,
      secure: false,
    },
  })
);

// passport 초기화 & 세션 미들웨어 실행
app.use(passport.initialize());
app.use(passport.session());

// 요청 경로에 따라 router 실행
app.use('/', pageRouter);
app.use('/auth', authRouter);
app.use('/post', postRouter);

// 오류 처리: 요청 경로가 없을 경우
app.use((req, res, next) => {
  const error = new Error(`${req.method} ${req.url} 라우터가 없습니다.`);
  error.status = 404;
  next(error);
});

// 서버 오류 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 대기중');
});
```

서버 실행 시, 필요한 모듈과 라우터 연결 및 오류 처리를 한다.

auth.js

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

const passport = require('../passport/index.js');
const { isLoggedIn, isNotLoggedIn } = require('./middlewares'); // 로그인 여부
const User = require('../models/user');

const router = express.Router();

// local 회원가입
/*
request.body
{
  "email": "asdf@asdf.com",
  "nick" : "nickname",
  "password" : "thisispassword"
}
*/
router.post('/join', isNotLoggedIn, async (req, res, next) => {
  const { email, nick, password } = req.body;
  try {
    const exUser = await User.findOne({ where: { email } });
    if (exUser) { // 유저가 존재하면, redirect
      return res.redirect('/join?error=exist');
    }
    console.info('___User.create(): ' + nick);
    const hash = await bcrypt.hash(password, 12); // pw 암호화
    await User.create({
      email,
      nick,
      password: hash,
    }); // 유저가 없으면, DB에 새로 생성
    return res.redirect('/');
  } catch (error) {
    console.error(error);
    return next(error);
  }
});

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

    console.info('___req.login()');
    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.
```

```
});

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

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

// kakao site login후 자동 redirect
// kakao 계정 정보를 이용하여 login or 회원가입/login
router.get(
  '/kakao/callback',
  passport.authenticate('kakao', {
    failureRedirect: '/',
  }),
  (req, res) => {
    res.redirect('/');
  }
);

module.exports = router;
```

일반 로그인 및 간편 로그인을 처리한다.

page.js

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

const router = express.Router();

// req.user의 사용자 데이터를 넌적스 템플릿에서 이용가능하도록 res.locals에 저장
router.use((req, res, next) => {
  res.locals.user = req.user;
  next();
});

// 사용자가 작성한 포스트 목록 조회
router.get('/profile', isLoggedIn, async (req, res, next) => {
  console.log(req.user.id);
  try {
    const posts = await Post.findAll({
      attributes : ['content', 'updatedAt'],
      include: {
        model: User,
        attributes: ['id', 'nick'],
        where : {id: req.user.id},
      },
      order: [['createdAt', 'DESC']],
    });
    res.render('profile', {
      title: '내 정보 - NodeBird' ,
      twits : posts
    });
  } catch (err) {
    console.error(err);
    next(err);
  }

});

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


// 포스트 전체 목록 조회
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);
  }
});



module.exports = router;
```

페이지에 보여지는 [전체 목록 조회], [내가 작성한 기록 목록 조회] api 요청을 처리한다.

> 내가 작성한 기록 목록 조회는 사용자 id로 post를 조회한다.

post.js

```
const express = require('express');
const Post = require('../models/post');
const { isLoggedIn } = require('./middlewares');

const router = express.Router();


// 포스트 생성
router
.post('/', isLoggedIn, async (req, res, next) => {
  try {
    const post = await Post.create({
      content: req.body.content,
```

```
      UserId: req.user.id,
    });
    console.log('Post:', post.content, post.UserId);
    res.redirect('/');
  } catch (error) {
    console.error(error);
    next(error);
  }
})
// 포스트 수정
.put('/:id', isLoggedIn, async (req, res, next) => {
  try {
    const postId = req.params.id;
    const content = req.body.content;
    const post = await Post.findOne({where:{id:postId}});
    console.log(post);
    if(post){
      Post.update({
        content : content,
      }, {
        where: {id: postId},
      });
      res.render('main', {
        title: 'NodeBird',
        twit: post,
      });
    } else {
      res.status(404).send('no post');
    }
  } catch (error) {
    console.error(error);
    next(error);
  }
})
// 포스트 삭제
.delete('/:id', isLoggedIn, async (req, res, next) => {
  try {
    const postId = req.params.id;
    await Post.destroy({
      where: {id: postId}
    })
    res.send("게시물 삭제 완료");
  } catch (error) {
    console.error(error);
    next(error);
  }
});

module.exports = router;
```

기록의 생성, 수정, 삭제 api 요청을 처리한다.

> 기록의 수정 및 삭제 시, postId를 pathvariable로 받아서, where 조건절에 사용한다.
> 수정 시, request body에 수정할 내용을 받아온다.

middlewares.js

```
// isLoggedIn 함수
// 로그인된 상태이면 다음 미들웨어로 연결
exports.isLoggedIn = (req, res, next) => {
  if (req.isAuthenticated()) {
    next();
  } else {
    res.status(403).send('로그인 필요');
  }
};

// isNotLoggedIn 함수
// 로그인 안된 상태이면 다음 미들웨어로 연결
exports.isNotLoggedIn = (req, res, next) => {
  if (!req.isAuthenticated()) {
    next();
  } else {
    const message = encodeURIComponent('로그인한 상태입니다.');
    res.redirect(`/?error=${message}`);
  }
};
```

로그인 상태 상태에 따른 로직을 처리한다.

models/index.js

```
const Sequelize = require('sequelize');
const User = require('./user');
const Post = require('./post');

const env = process.env.NODE_ENV || 'development';
const config = require('../config/config')[env];
const db = {};

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

db.sequelize = sequelize;

db.User = User;
db.Post = Post;

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

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

module.exports = db;
```

DB 기본 설정을 처리한다.

models/user.js

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

class User extends Sequelize.Model {
  static init(sequelize) {
    const userAttr = {
      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',
      },
      snsId: {
        type: Sequelize.STRING(30),
        allowNull: true,
      },
    };

    const userTbl = {
      sequelize,
      timestamps: true,
      underscored: false,
      modelName: 'User',
      tableName: 'users',
      paranoid: true,
      charset: 'utf8mb4',
      collate: 'utf8mb4_unicode_ci',
    };

    return super.init(userAttr, userTbl);
  }

  static associate(db) {
    db.User.hasMany(db.Post);

  }
}

module.exports = User;
```

user 테이블에 대한 정의와 관계 설정을 처리한다.

models/post.js

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

class Post extends Sequelize.Model {
  static init(sequelize) {
    const postAttr = {
      content: {
        type: Sequelize.STRING(140),
        allowNull: false,
      },
    };

    const postTbl = {
      sequelize,
      timestamps: true,
      underscored: false,
      modelName: 'Post',
      tableName: 'posts',
      paranoid: false,
      charset: 'utf8mb4',
      collate: 'utf8mb4_general_ci',
    };

    return super.init(postAttr, postTbl);
  }

  static associate(db) {
    db.Post.belongsTo(db.User);
  }
}

module.exports = Post;
```

post 테이블에 대한 정의와 관계 설정을 처리한다.

passport/index.js

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

passport.serializeUser((user, done) => {
  console.info('___passport.serializeUser()');
  done(null, user.id);
});

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

passport.use(localStrategy);
passport.use(kakaoStrategy);

module.exports = passport;
```

passport 설정을 담당한다.

passport/localstrategy.js

```
const LocalStrategy = require('passport-local').Strategy;
const bcrypt = require('bcrypt');
const User = require('../models/user');

module.exports = new LocalStrategy(
  {
    usernameField: 'email',
    passwordField: 'password',
  },
  async (email, password, done) => {
    console.info('___new LocalStrategy()');
    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);
    }
  }
);
```

일반 로그인 처리를 한다.

passport/kakaostrategy.js

```
const KakaoStrategy = require('passport-kakao').Strategy;
const User = require('../models/user');

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

카카오 간편 로그인 처리를 담당한다.

footer.html

```
<footer>
    <div class="writer">
        <p> <br/>
            2022학년도 1학기 고급웹프로그래밍 학기말 프로젝트<br/>
            Made by dldmsql

        </p>
    </div>
</footer>
```

join.html

```
{% 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 %}

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

error.html

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

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

layout.html

```
<!DOCTYPE html>
<html lang="kor">
  <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">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/js/bootstrap.bundle.min.js" integrity="sha384-pprn3073KE6tl6bjs2QrFaJGz5/SUsLqktiwsUTF55Jfv3qYSDhgCecCxMW52nD2" crossorigin="anonymous">
  </head>
  <body>
    <header>
      <img class="header-img" src="/오늘하루_가로.png"/>
      <p> 당신의 하루를 기록하고 공유해보세요. </p>
    </header>
    <div class="container">
      <div class="profile-wrap">
        <div class="profile">
          {% if user and user.id %}
            <div class="user-name">{{'안녕하세요! ' + user.nick + '님'}}</div>
          <input id="my-id" type="hidden" value="{{user.id}}">
          <a id="my-profile" href="/profile" class="btn">내가 기록한 하루들</a>
          <a id="logout" href="/auth/logout" class="btn">로그아웃</a>
          <a id="home" href="/" class="btn">홈으로</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>

      </div>
      {% block content %}
      {% endblock %}
    </div>
    {% include "footer.html" %}
    <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>
```

main.html

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

{% block content %}
    <div class="timeline">
      {% if user %}
        <div>
          <form id="twit-form" action="/post" method="post">
            <div class="input-group">
              <textarea id="twit" name="content" maxlength="140"></textarea>
            </div>
            <div>
              <button id="twit-btn" type="submit" class="btn-outline-primary">기록</button>
            </div>
          </form>
        </div>

      {% endif %}

      <div class="twits">
        {% 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>
            <div class="twit-content">{{twit.content}}</div>
            {% if user and twit.User.id === user.id %}
            <div class="twit-btn">
              <button class="twit-modify btn-outline-primary">수정하기</button>
              <button class="twit-delete btn-outline-primary">삭제하기</button>
```

```
            </div>
            {% else %}
            <div>
              <button class="twit-modify btn-outline-primary" disabled>수정하기</button>
              <button class="twit-delete btn-outline-primary" disabled>삭제하기</button>
            </div>
            {% endif %}
          </div>
        {% endfor %}
      </div>
    </div>
{% endblock %}

{% block script %}
<script>
  document.querySelectorAll('.twit-modify').forEach(function(tag) {
    console.log(tag);
    tag.addEventListener('click', function(){

      const id = document.querySelector('.twit-id').value;
      const input = prompt('변경할 내용 입력해주세요.');

      if(input) {
        axios.put(`/post/${id}`, {content : input})
        .then(()=>{

          location.reload();
        })
        .catch((err) => {
          console.error(err);
        });
      }
    });
  });
  document.querySelectorAll('.twit-delete').forEach(function(tag) {
    console.log(tag);
    tag.addEventListener('click', function(){
      const id = document.querySelector('.twit-id').value;
      const input = confirm('삭제하시겠어요?');
      if(input) {
        axios.delete(`/post/${id}`)
        .then(()=>{
          console.log('바꼈나요?');
          location.reload();
        })
        .catch((err) => {
          console.error(err);
        });
      }
    });
  });
</script>
{% endblock %}
```

profile.html

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

{% block content %}
  <div class="timeline">
    <div class="twits">
        <h2>내가 기록한 하루들</h2>
        {% 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-updatedAt">{{'기록 날짜 : ' +  twit.updatedAt}}</div>
            <div class="twit-content">{{'내가 기록한 하루 : ' + twit.content}}</div>

          </div>
        {% endfor %}
      </div>
  </div>
{% endblock %}
```