API Server

기본 설정

❖ 프로젝트 생성 – node_api

❖ 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 passport passport-kakao passport-local bcrypt uuid

npm install --save-dev nodemon

❖ 이전 프로젝트에서 config, models, passport 디렉토리를 복사해서 붙여넣기

기본 설정

- 프로젝트에 routes 디렉토리를 생성하고 이전 프로젝트에서 auth.js 와 middleware.js 를 복사
- .env 파일을 생성하고 작성

```
PORT=9001
COOKIE_SECRET=sns
KAKAO_ID=6a92e72defaeb1bd45b361490ae7f2b8

HOST='localhost'
MYSQLPORT=3306
USERID='adam'
PASSWORD='wnddkd'
DATABASE='adam'
```

- 프로젝트에 views 디렉토리를 생성하고 error.html 파일을 작성

```
<h1>{{message}}</h1>
<h2>{{error.status}}</h2>
<pre>{{error.stack}}</pre>
```

기본 설정

❖ 프로젝트에 app.js 파일을 생성하고 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

❖ 프로젝트에 app.js 파일을 생성하고 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

- 프로젝트에 app.js 파일을 생성하고 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

❖ 프로젝트에 app.js 파일을 생성하고 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

❖ 프로젝트에 app.js 파일을 생성하고 작성

```
const {sequelize} = require('./models');

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

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

기본 설정

- 프로젝트에 app.js 파일을 생성하고 작성

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

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

app.use('/img', express.static(path.join(__dirname, 'uploads')));
```

기본 설정

❖ 프로젝트에 app.js 파일을 생성하고 작성

```
//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

- 도메인 등록을 위한 테이블
  - ✓ 컬럼
    - host: 인터넷 주소
    - type: 도메인 종류(free, premium)
    - clientSecret: 클라이언트 비밀 키
  - ✓ 사용자 한 명이 여러 개의 도메인 소유 가능

기본 설정

❖ 도메인을 등록하기 위한 domain.js 파일을 models 디렉토리에 생성하고 작성

```
const Sequelize = require('sequelize');
module.exports = class Domain extends Sequelize.Model {
  static init(sequelize) {
    return super.init({
      host: {
        type: Sequelize.STRING(80),
        allowNull: false,
      },
      type: {
        type: Sequelize.ENUM('free', 'premium'),
        allowNull: false,
      },
      clientSecret: {
        type: Sequelize.STRING(36),
        allowNull: false,
      },
    }, {
      sequelize,
      timestamps: true,
      paranoid: true,
      modelName: 'Domain',
      tableName: 'domains',
    });
  }
```

기본 설정

- 도메인을 등록하기 위한 domain.js 파일을 models 디렉토리에 생성하고 작성

```
    static associate(db) {
      db.Domain.belongsTo(db.User);
    }
  };
```

기본 설정

❖ models 디렉토리의 index.js 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 Domain = require('./domain');

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;
db.Domain = Domain;

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

기본 설정

- models 디렉토리의 index.js 수정

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

module.exports = db;
```

기본 설정

❖ models 디렉토리의 user.js 수정

```
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',
      },
```

기본 설정

❖ models 디렉토리의 user.js 수정

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

기본 설정

❖ models 디렉토리의 user.js 수정

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

기본 설정

❖ views 디렉토리에 로그인을 위한 login.html 파일을 생성하고 작성

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>API 서버 로그인</title>
    <style>
      .input-group label { width: 200px; display: inline-block; }
    </style>
  </head>
  <body>
    {% if user and user.id %}
      <span class="user-name">안녕하세요! {{user.nick}}님</span>
      <a href="/auth/logout">
        <button>로그아웃</button>
      </a>
      <fieldset>
```

기본 설정

❖ views 디렉토리에 로그인을 위한 login.html 파일을 생성하고 작성

```
<legend>도메인 등록</legend>
<form action="/domain" method="post">
  <div>
    <label for="type-free">무료</label>
    <input type="radio" id="type-free" name="type" value="free">
    <label for="type-premium">프리미엄</label>
    <input type="radio" id="type-premium" name="type" value="premium">
  </div>
  <div>
    <label for="host">도메인</label>
    <input type="text" id="host" name="host" placeholder="ex) zerocho.com">
  </div>
  <button>저장</button>
</form>
</fieldset>
```

기본 설정

❖ views 디렉토리에 로그인을 위한 login.html 파일을 생성하고 작성

```
<table>
  <tr>
    <th>도메인 주소</th>
    <th>타입</th>
    <th>클라이언트 비밀키</th>
  </tr>
  {% for domain in domains %}
    <tr>
      <td>{{domain.host}}</td>
      <td>{{domain.type}}</td>
      <td>{{domain.clientSecret}}</td>
    </tr>
  {% endfor %}
</table>
```

기본 설정

❖ views 디렉토리에 로그인을 위한 login.html 파일을 생성하고 작성

```
{% else %}
  <form action="/auth/login" id="login-form" method="post">
    <h2>NodeSNS 계정으로 로그인하세요.</h2>
    <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>
      <button id="login" type="submit" class="btn">로그인</button>
      <a id="join" href="http://localhost:9000/join" class="btn">회원가입</a>
      <a id="kakao" href="http://localhost:9000/auth/kakao" class="btn">카카오톡</a>
  </form>
```

기본 설정

- views 디렉토리에 로그인을 위한 login.html 파일을 생성하고 작성

```
        <script>
          window.onload = () => {
            if (new URL(location.href).searchParams.get('loginError')) {
              alert(new URL(location.href).searchParams.get('loginError'));
            }
          };
        </script>
      {% endif %}
    </body>
  </html>
```

기본 설정

❖ routes 디렉토리에 index.js 파일을 생성하고 작성

```
const express = require('express');
const { v4: uuidv4 } = require('uuid');
const { User, Domain } = require('../models');
const { isLoggedIn } = require('./middlewares');

const router = express.Router();

router.get('/', async (req, res, next) => {
  try {
    const user = await User.findOne({
      where: { id: req.user && req.user.id || null },
      include: { model: Domain },
    });
    res.render('login', {
      user,
      domains: user && user.Domains,
    });
  } catch (err) {
    console.error(err);
    next(err);
  }
});
```

기본 설정

❖ routes 디렉토리에 index.js 파일을 생성하고 작성

```
router.post('/domain', isLoggedIn, async (req, res, next) => {
  try {
    await Domain.create({
      UserId: req.user.id,
      host: req.body.host,
      type: req.body.type,
      clientSecret: uuidv4(),
    });
    res.redirect('/');
  } catch (err) {
    console.error(err);
    next(err);
  }
});

module.exports = router;
```

JWT 인증

- JWT(JSON Web Token)
  - ✓ 전자 서명 된 URL-safe (URL로 이용할 수 있는 문자 만 구성된)의 JSON
  - ✓ 전자 서명은 JSON 의 변조를 체크 할 수 있게 되어 있음
  - ✓ JWT는 속성 정보를 JSON 데이터 구조로 표현한 토큰으로 RFC7519 표준
  - ✓ JWT는 서버와 클라이언트 간 정보를 주고 받을 때 Http Request Header에 JSON 토큰을 넣은 후 서버는 별도의 인증 과정없이 헤더에 포함되어 있는 JWT 정보를 통해 인증
  - ✓ JWT는 HMAC 알고리즘을 사용하여 비밀키 또는 RSA 를 이용한 Public Key/ Private Key 쌍으로 서명할 수 있음
  - ✓ 구성
    - HEADER: 토큰 종류 와 해시 알고리즘 정보
    - PAYLOAD: 토큰의 내용물이 인코딩 된 부분
    - SIGNATURE: 일련의 문자열로 토큰이 변조되었는지 여부를 확인할 수 있음

JWT 인증

- 설치

  npm install jsonwebtoken

- .env 파일에 암호화에 사용할 문자열을 등록

  JWT_SECRET=jwtSecret

JWT 인증

❖ routes 디렉토리의 middleware.js 파일에 인증 관련 기능을 추가

```
const jwt = require('jsonwebtoken');

exports.verifyToken = (req, res, next) => {
  try {
    req.decoded = jwt.verify(req.headers.authorization, process.env.JWT_SECRET);
    return next();
  } catch (error) {
    if (error.name === 'TokenExpiredError') { // 유효기간 초과
      return res.status(419).json({
        code: 419,
        message: '토큰이 만료되었습니다',
      });
    }
    return res.status(401).json({
      code: 401,
      message: '유효하지 않은 토큰입니다',
    });
  }
};
```

JWT 인증

❖ routes 디렉토리에 토큰의 내용을 확인해서 사용하기 위한 v1.js 파일을 생성하고 작성

```
const express = require('express');
const jwt = require('jsonwebtoken');
const { verifyToken } = require('./middlewares');
const { Domain, User } = require('../models');

const router = express.Router();

router.post('/token', async (req, res) => {
  const { clientSecret } = req.body;
  try {
    const domain = await Domain.findOne({
      where: { clientSecret },
      include: {
        model: User,
        attribute: ['nick', 'id'],
      },
    });
    if (!domain) {
      return res.status(401).json({
        code: 401,
        message: '등록되지 않은 도메인입니다. 먼저 도메인을 등록하세요',
      });
    }
```

JWT 인증

❖ routes 디렉토리에 토큰의 내용을 확인해서 사용하기 위한 v1.js 파일을 생성하고 작성

```
    const token = jwt.sign({
      id: domain.User.id,
      nick: domain.User.nick,
    }, process.env.JWT_SECRET, {
      expiresIn: '1m', // 1분
      issuer: 'nodebird',
    });
    return res.json({
      code: 200,
      message: '토큰이 발급되었습니다',
      token,
    });
  } catch (error) {
    console.error(error);
    return res.status(500).json({
      code: 500,
      message: '서버 에러',
    });
  }
});
```

JWT 인증

- routes 디렉토리에 토큰의 내용을 확인해서 사용하기 위한 v1.js 파일을 생성하고 작성

```
router.get('/test', verifyToken, (req, res) => {
  res.json(req.decoded);
});

module.exports = router;
```

JWT 인증

- app.js 파일에 v1 등록하는 코드를 추가

```
const v1 = require('./routes/v1');
app.use ('/v1',v1);
```

JWT 인증

❖ 프로젝트 실행

클라이언트 애플리케이션

❖ 프로젝트 생성 – node_client

❖ package.json 수정

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

❖ 필요한 패키지 설치

npm install express dotenv axios cookie-parser express-session morgan nunjucks

npm install --save-dev nodemon

클라이언트 애플리케이션

❖ 프로젝트에 app.js 파일을 생성하고 작성

```
const express = require('express');
const morgan = require('morgan');
const cookieParser = require('cookie-parser');
const session = require('express-session');
const nunjucks = require('nunjucks');
const dotenv = require('dotenv');

dotenv.config();
const indexRouter = require('./routes');

const app = express();
app.set('port', process.env.PORT || 4000);
app.set('view engine', 'html');
nunjucks.configure('views', {
  express: app,
  watch: true,
});
```

클라이언트 애플리케이션

- 프로젝트에 app.js 파일을 생성하고 작성

```
app.use(morgan('dev'));
app.use(cookieParser(process.env.COOKIE_SECRET));
app.use(session({
  resave: false,
  saveUninitialized: false,
  secret: process.env.COOKIE_SECRET,
  cookie: {
    httpOnly: true,
    secure: false,
  },
}));
```

클라이언트 애플리케이션

❖ 프로젝트에 app.js 파일을 생성하고 작성

```
app.use('/', indexRouter);

app.use((req, res, next) => {
  const error =  new Error(`${req.method} ${req.url} 라우터가 없습니다.`);
  error.status = 404;
  next(error);
});

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');
});
```

클라이언트 애플리케이션

- 프로젝트에 views 디렉토리를 생성하고 error.html 파일을 작성

```
<h1>{{message}}</h1>
<h2>{{error.status}}</h2>
<pre>{{error.stack}}</pre>
```

- 프로젝트에 .env 파일을 생성하고 작성

```
COOKIE_SECRET=nodeclient
CLIENT_SECRET=baf82034-6291-44bb-b690-09bb9f7eb86f
```

클라이언트 애플리케이션

❖ 프로젝트에 routes 디렉토리를 생성하고 index.js 파일을 생성 한 후 작성

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

const router = express.Router();

router.get('/test', async (req, res, next) => { // 토큰 테스트 라우터
  try {
    if (!req.session.jwt) { // 세션에 토큰이 없으면 토큰 발급 시도
      const tokenResult = await axios.post('http://localhost:9001/v1/token', {
        clientSecret: process.env.CLIENT_SECRET,
      });
      if (tokenResult.data && tokenResult.data.code === 200) { // 토큰 발급 성공
        req.session.jwt = tokenResult.data.token; // 세션에 토큰 저장
      } else { // 토큰 발급 실패
        return res.json(tokenResult.data); // 발급 실패 사유 응답
      }
    }
```

클라이언트 애플리케이션

❖ 프로젝트에 routes 디렉토리를 생성하고 index.js 파일을 생성 한 후 작성

```
    // 발급받은 토큰 테스트
    const result = await axios.get('http://localhost:9001/v1/test', {
      headers: { authorization: req.session.jwt },
    });
    return res.json(result.data);
  } catch (error) {
    console.error(error);
    if (error.response.status === 419) { // 토큰 만료 시
      return res.json(error.response.data);
    }
    return next(error);
  }
});

module.exports = router;
```

클라이언트 애플리케이션

API 구현

❖ API Server 프로젝트의 routes 디렉토리의 v1.js 수정

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

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

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

API 구현

❖ API Server 프로젝트의 routes 디렉토리의 v1.js 수정

```
router.post('/token', async (req, res) => {
  const { clientSecret } = req.body;
  try {
    const domain = await Domain.findOne({
      where: { clientSecret },
      include: {
        model: User,
        attribute: ['nick', 'id'],
      },
    });
    if (!domain) {
      return res.status(401).json({
        code: 401,
        message: '등록되지 않은 도메인입니다. 먼저 도메인을 등록하세요',
      });
    }
```

API 구현

❖ API Server 프로젝트의 routes 디렉토리의 v1.js 수정

```
      const token = jwt.sign({
        id: domain.User.id,
        nick: domain.User.nick,
      }, process.env.JWT_SECRET, {
        expiresIn: '1m', // 1분
        issuer: 'nodebird',
      });
      return res.json({
        code: 200,
        message: '토큰이 발급되었습니다',
        token,
      });
    } catch (error) {
      console.error(error);
      return res.status(500).json({
        code: 500,
        message: '서버 에러',
      });
    }
  });
```

API 구현

❖ API Server 프로젝트의 routes 디렉토리의 v1.js 수정

```
router.get('/test', verifyToken, (req, res) => {
  res.json(req.decoded);
});

router.get('/posts/my', verifyToken, (req, res) => {
  Post.findAll({ where: { userId: req.decoded.id } })
    .then((posts) => {
      console.log(posts);
      res.json({
        code: 200,
        payload: posts,
      });
    })
    .catch((error) => {
      console.error(error);
      return res.status(500).json({
        code: 500,
        message: '서버 에러',
      });
    });
});
```

API 구현

❖ API Server 프로젝트의 routes 디렉토리의 v1.js 수정

```
router.get('/posts/hashtag/:title', verifyToken, async (req, res) => {
  try {
    const hashtag = await Hashtag.findOne({ where: { title: req.params.title } });
    if (!hashtag) {
      return res.status(404).json({
        code: 404,
        message: '검색 결과가 없습니다',
      });
    }
    const posts = await hashtag.getPosts();
    return res.json({
      code: 200,
      payload: posts,
    });
  } catch (error) {
    console.error(error);
    return res.status(500).json({
      code: 500,
      message: '서버 에러',
    });
  }
});

module.exports = router;
```

API 구현

- API Client 프로젝트의 routes 디렉토리의 index.js 수정

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

const router = express.Router();
const URL = 'http://localhost:9001/v1';
```

API 구현

❖ API Client 프로젝트의 routes 디렉토리의 index.js 수정

```
axios.defaults.headers.origin = 'http://localhost:4000'; // origin 헤더 추가
const request = async (req, api) => {
  try {
    if (!req.session.jwt) { // 세션에 토큰이 없으면
      const tokenResult = await axios.post(`${URL}/token`, {
        clientSecret: process.env.CLIENT_SECRET,
      });
      req.session.jwt = tokenResult.data.token; // 세션에 토큰 저장
    }
    return await axios.get(`${URL}${api}`, {
      headers: { authorization: req.session.jwt },
    }); // API 요청
  } catch (error) {
    if (error.response.status === 419) { // 토큰 만료시 토큰 재발급 받기
      delete req.session.jwt;
      return request(req, api);
    } // 419 외의 다른 에러면
    return error.response;
  }
};
```

API 구현

❖ API Client 프로젝트의 routes 디렉토리의 index.js 수정

```
router.get('/mypost', async (req, res, next) => {
  try {
    const result = await request(req, '/posts/my');
    res.json(result.data);
  } catch (error) {
    console.error(error);
    next(error);
  }
});
```

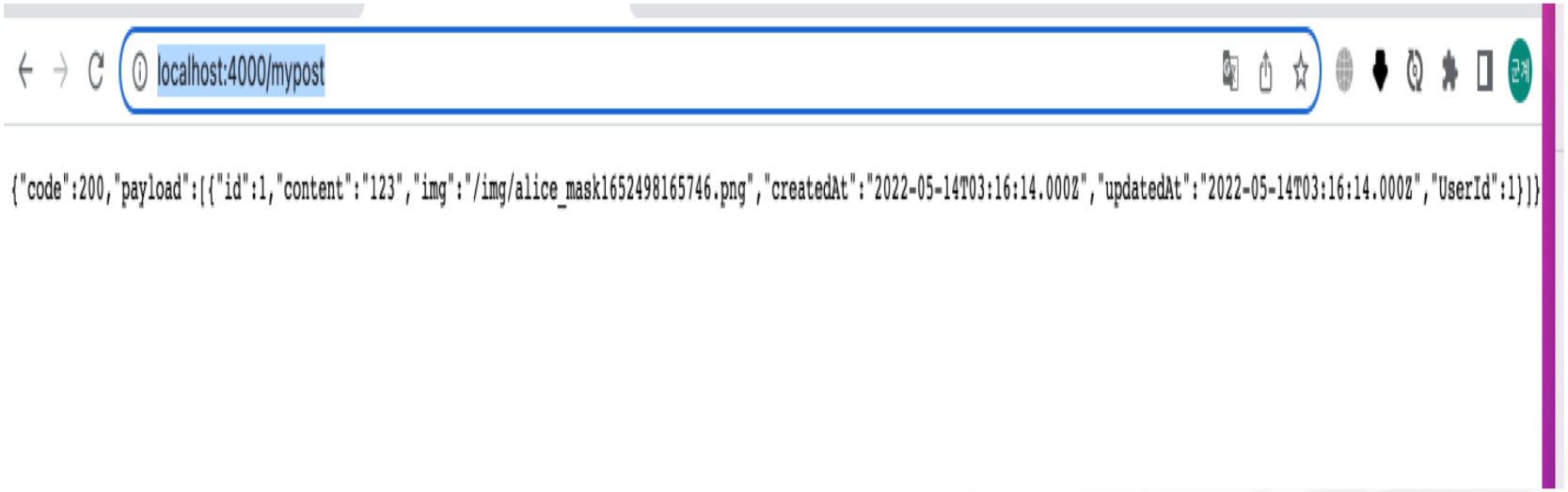
SNS API

❖ API Client 프로젝트의 routes 디렉토리의 index.js 수정

```
router.get('/search/:hashtag', async (req, res, next) => {
  try {
    const result = await request(
      req, `/posts/hashtag/${encodeURIComponent(req.params.hashtag)}`,
    );
    res.json(result.data);
  } catch (error) {
    if (error.code) {
      console.error(error);
      next(error);
    }
  }
});

module.exports = router;
```

SNS API


localhost:4000/mypost
{"code":200,"payload":[{"id":1,"content":"123","img":"/img/alice_mask1652498165746.png","createdAt":"2022-05-14T03:16:14.000Z","updatedAt":"2022-05-14T03:16:14.000Z","UserId":1}]}

사용량 제한

❖ 설치

npm install express-rate-limit

❖ 서버 프로젝트의 routes 디렉토리의 middleware.js 파일에 추가

```
const RateLimit = require('express-rate-limit');

exports.apiLimiter = new RateLimit({
  windowMs: 60 * 1000, // 1분
  max: 10,
  delayMs: 0,
  handler(req, res) {
    res.status(this.statusCode).json({
      code: this.statusCode, // 기본값 429
      message: '1분에 한 번만 요청할 수 있습니다.',
    });
  },
});

exports.deprecated = (req, res) => {
  res.status(410).json({
    code: 410,
    message: '새로운 버전이 나왔습니다. 새로운 버전을 사용하세요.',
  });
};
```

사용량 제한

❖ 서버 프로젝트의 routes 디렉토리에 v2.js 파일을 추가하고 작성

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

const { verifyToken, apiLimiter } = require('./middlewares');
const { Domain, User, Post, Hashtag } = require('../models');

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

사용량 제한

❖ 서버 프로젝트의 routes 디렉토리에 v2.js 파일을 추가하고 작성

```
router.post('/token', apiLimiter, async (req, res) => {
  const { clientSecret } = req.body;
  try {
    const domain = await Domain.findOne({
      where: { clientSecret },
      include: {
        model: User,
        attribute: ['nick', 'id'],
      },
    });
    if (!domain) {
      return res.status(401).json({
        code: 401,
        message: '등록되지 않은 도메인입니다. 먼저 도메인을 등록하세요',
      });
    }
```

사용량 제한

❖ 서버 프로젝트의 routes 디렉토리에 v2.js 파일을 추가하고 작성

```
    const token = jwt.sign({
      id: domain.User.id,
      nick: domain.User.nick,
    }, process.env.JWT_SECRET, {
      expiresIn: '30m', // 30분
      issuer: 'nodebird',
    });
    return res.json({
      code: 200,
      message: '토큰이 발급되었습니다',
      token,
    });
  } catch (error) {
    console.error(error);
    return res.status(500).json({
      code: 500,
      message: '서버 에러',
    });
  }
});
```

사용량 제한

❖ 서버 프로젝트의 routes 디렉토리에 v2.js 파일을 추가하고 작성

```
router.get('/test', verifyToken, apiLimiter, (req, res) => {
  res.json(req.decoded);
});

router.get('/posts/my', apiLimiter, verifyToken, (req, res) => {
  Post.findAll({ where: { userId: req.decoded.id } })
    .then((posts) => {
      console.log(posts);
      res.json({
        code: 200,
        payload: posts,
      });
    })
    .catch((error) => {
      console.error(error);
      return res.status(500).json({
        code: 500,
        message: '서버 에러',
      });
    });
});
```

사용량 제한

❖ 서버 프로젝트의 routes 디렉토리에 v2.js 파일을 추가하고 작성

```
router.get('/posts/hashtag/:title', verifyToken, apiLimiter, async (req, res) => {
  try {
    const hashtag = await Hashtag.findOne({ where: { title: req.params.title } });
    if (!hashtag) {
      return res.status(404).json({
        code: 404,
        message: '검색 결과가 없습니다',
      });
    }
    const posts = await hashtag.getPosts();
    return res.json({
      code: 200,
      payload: posts,
    });
  } catch (error) {
    console.error(error);
    return res.status(500).json({
      code: 500,
      message: '서버 에러',
    });
  }
});

module.exports = router;
```

사용량 제한

❖ 서버 프로젝트의 routes 디렉토리에 v1.js 파일 수정

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

const { verifyToken, deprecated } = require('./middlewares');
const { Domain, User, Post, Hashtag } = require('../models');

const router = express.Router();

router.use(deprecated);

router.post('/token', async (req, res) => {
```

사용량 제한

- 서버 프로젝트의 app.js 파일에 추가

```
const v2 = require('./routes/v2');
app.use ('/v2',v2);
```

사용량 제한

- 클라이언트 프로젝트의 routes 디렉토리의 index.js 파일에 추가

```
const URL = 'http://localhost:9001/v2';
```

CORS

- 교차 출처 리소스 공유 (CORS)
  - ✓ 교차 출처 리소스 공유(Cross-Origin Resource Sharing, CORS)는 추가 HTTP 헤더를 사용하여 한 출처에서 실행 중인 웹 애플리케이션이 다른 출처의 자원에 접근할 수 있는 권한을 부여하도록 브라우저에 알려주는 체제
  - ✓ 웹 애플리케이션은 리소스가 자신의 출처(도메인, 프로토콜, 포트)와 다를 때 교차 출처 HTTP 요청을 실행함
  - ✓ 보안 상의 이유로 브라우저는 스크립트에서 시작한 교차 출처 HTTP 요청을 제한함
  - ✓ XMLHttpRequest 와 Fetch API는 동일 출처 정책을 따르기 때문에 이 API 를 사용하는 웹 애플리케이션은 자신의 출처와 동일한 리소스만 불러올 수 있으며 다른 출처의 리소스를 불러오려면 그 출처에서 올바른 CORS 헤더를 포함한 응답을 반환해야 함

CORS

- SOP(Same Origin Policy - 동일 출처 정책)
  - ✓ 어떤 출처(origin)에서 불러온 문서나 스크립트가 다른 출처에서 가져온 리소스와 상호 작용하는 것을 제한하는 브라우저의 보안 방식
  - ✓ 다른 출처와 같은 출처를 구분하는 기준은 URI의 프로토콜 그리고 호스트 및 포트가 동일한지 여부
  - ✓ 모든 방식의 요청에 적용 되는 것은 아님
  - ✓ 적용되지 않는 경우
    - ❑ <img> 태그로 다른 도메인의 이미지 파일을 가져오는 경우
    - ❑ <link> 태그로 다른 도메인의 CSS를 가져오는 경우
    - ❑ <script> 태그로 다른 도메인의 javascript를 가져오는 경우
    - ❑ <video> <audio> <object> <embed> <applet> 태그
  - ✓ SOP는 script에서 XMLHttpRequest나 Fetch API 를 사용해 다른 출처에 리소스를 요청할 때 적용

CORS

- 클라이언트 프로젝트의 routes 디렉토리의 index.js 파일에 추가

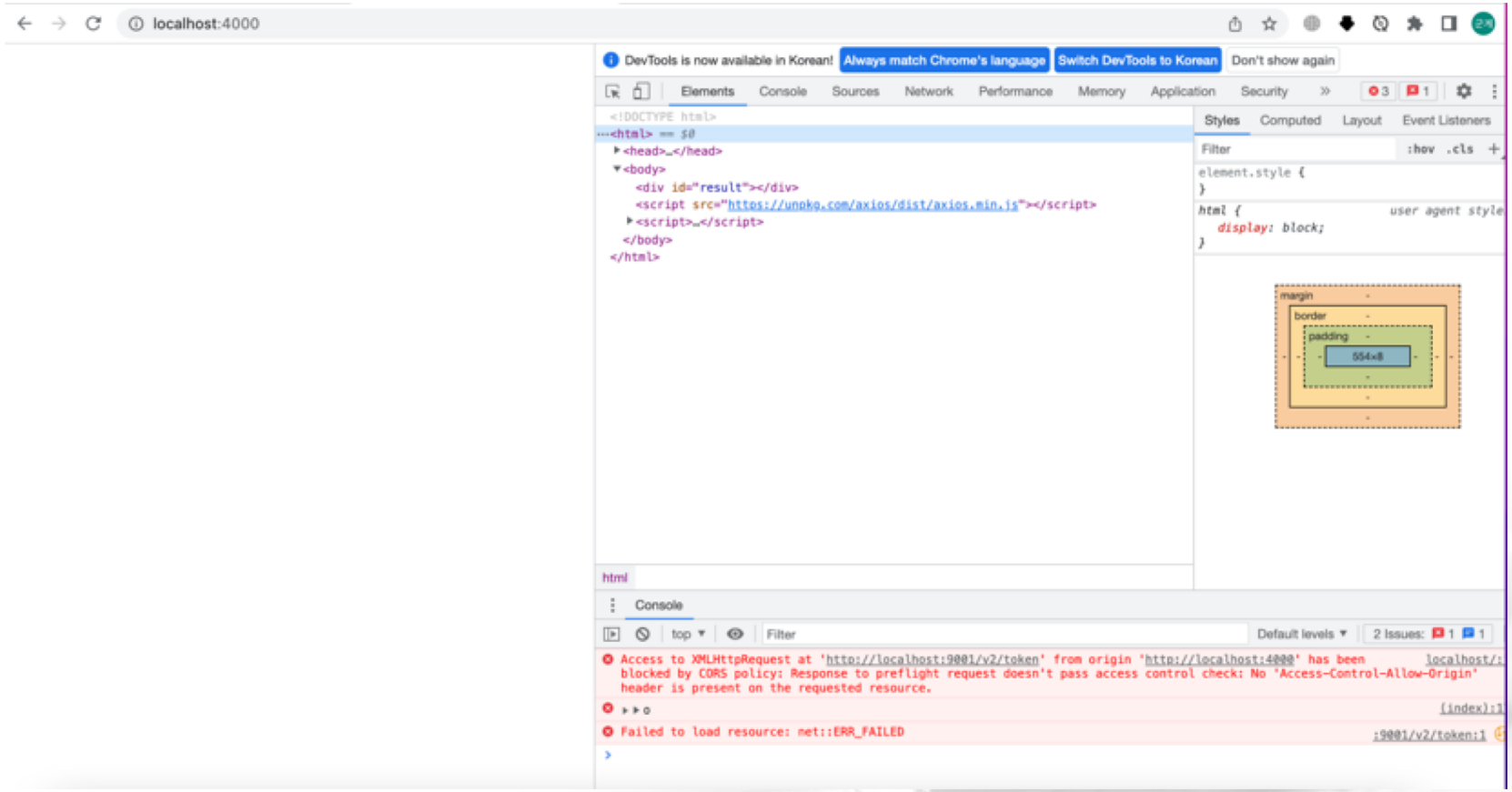
```
router.get('/', (req, res) => {
  res.render('main', { key: process.env.CLIENT_SECRET });
});
```

CORS

❖ 클라이언트 프로젝트의 views 디렉토리에 main.html 파일을 추가하고 작성

```
<!DOCTYPE html>
<html>
  <head>
    <title>프론트 API 요청</title>
  </head>
  <body>
  <div id="result"></div>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <script>
    axios.post('http://localhost:9001/v2/token', {
      clientSecret: '{{key}}',
    })
      .then((res) => {
        document.querySelector('#result').textContent = JSON.stringify(res.data);
      })
      .catch((err) => {
        console.error(err);
      });
  </script>
  </body>
</html>
```

CORS


localhost:4000
DevTools is now available in Korean!
Always match Chrome's language
Switch DevTools to Korean
Don't show again
Elements
Console
Sources
Network
Performance
Memory
Application
Security
<!DOCTYPE html>
<html> == $0
<head>…</head>
<body>

<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>…</script>
</body>
</html>
Styles
Computed
Layout
Event Listeners
Filter
:hov .cls
element.style {
}
html {
user agent style
display: block;
}
margin
border
padding
554×8
html
Console
top
Filter
Default levels
2 Issues:
Access to XMLHttpRequest at 'http://localhost:9001/v2/token' from origin 'http://localhost:4000' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.
localhost/:1
(index):1
Failed to load resource: net::ERR_FAILED
:9001/v2/token:1

CORS

- 서버 프로젝트에 CORS 구현을 위한 패키지 설치

  npm install cors

CORS

❖ 서버 프로젝트의 routes 디렉토리의 v2.js 파일 수정

```
const express = require('express');
const jwt = require('jsonwebtoken');
const cors = require('cors');
const url = require('url');

const { verifyToken, apiLimiter } = require('./middlewares');
const { Domain, User, Post, Hashtag } = require('../models');

const router = express.Router();

router.use(async (req, res, next) => {
  const domain = await Domain.findOne({
    where: { host: url.parse(req.get('origin')).host },
  });
  if (domain) {
    cors({
      origin: req.get('origin'),
      credentials: true,
    })(req, res, next);
  } else {
    next();
  }
});
```

CORS

{"code":200,"message":"토큰이 발급되었습니
다","token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmljayI6IuyVhOuLtCIsImlhdCI6MTY1MjU4MjQ4OSwiZXhwIjoxNjUyNTg(