Auth 기능과 로그아웃 기능

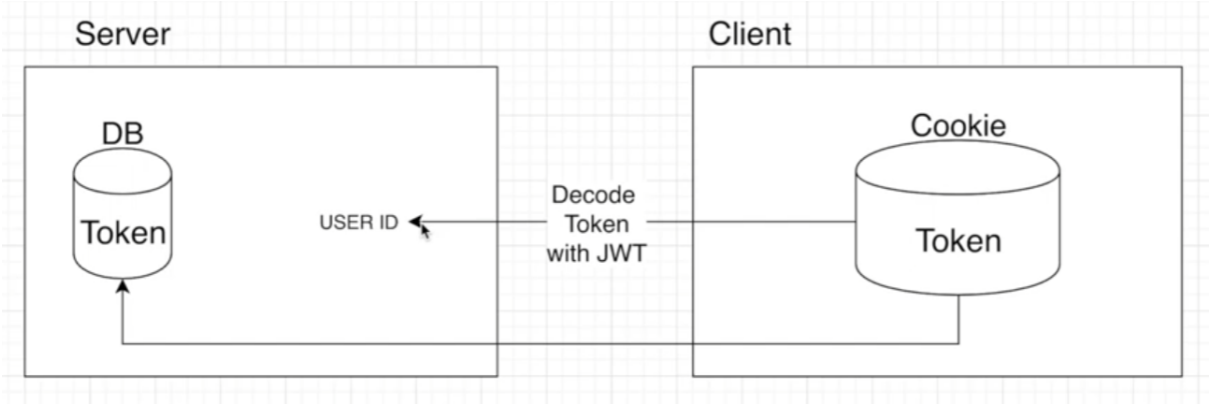
Authentication(인증) / Authorization(인가) 이란

- **인증** : 유저가 누구인지 확인하는 절차, 회원가입하고 로그인하는 것
- **인가** : 유저에 대한 권한을 허락하는 것

페이지를 이동 할 때 로그인 되어 있는지 안되어 있는지 체크

로그인 되어 있는 유저가 관리자인지 일반 유저인지 등을 체크

글을 쓸 때나 지울 때 권한이 있는지 체크



DB의 정보와 Cookie에 담겨있는 정보가 같은지 체크하는 것

1. Cookie에 저장된 Token을 Client Server에서 가져와서 복호화를 한다.
2. 복호화를 하면 User Id가 나오는데 그 User Id를 이용해서 데이터베이스 Users Collection의 유저를 찾은 후, 쿠키에서 받아온 token이 유저도 가지고 있는지 확인한다.
3. 쿠키가 일치하지 않으면 인증 실패
4. 쿠키가 일치하면 인증 성공 → 해당

auth route 만들기

index.js

```
const express = require('express')
const app = express()
const port = 5000
const cookieParser = require('cookie-parser')

const config = require('./config/key');
const { User } = require("./models/User")
//auth.js = 인증 처리를 하는 곳
const { auth } = require("./middleware/auth")

app.use(express.json())
app.use(cookieParser());

const mongoose = require('mongoose')
mongoose.connect(config.mongoURI).then(() => console.log('M
ongoDB Connected...'))
  .catch(err => console.log(err))

app.get('/', (req, res) => {
  res.send('Hello World')
})

app.post('/api/users/register', async (req, res) => {
  const user = new User(req.body);
  await user.save().then(() => {
    res.status(200).json({
      success: true
    })
  }).catch((err) => {
    res.json({ success: false, err })
  })
})
```

```
app.post('/api/users/login', (req, res) => {
  User.findOne({ email: req.body.email })
  .then(async (user) => {
    if(!user){
      throw new Error("제공된 이메일에 해당하는 유저가 없습니다.")
    }
    const isMatch = await user.comparePassword(req.body.pas
sword);
    return { isMatch, user };
  })
  .then(({ isMatch, user }) => {
    console.log(isMatch);
    if (!isMatch) {
      throw new Error("비밀번호가 틀렸습니다.")
    }
    return user.generateToken();
  })
  .then ((user) => {
    return res.cookie("x_auth", user.token)
      .status(200)
      .json({
        loginSuccess:true,
        userId: user._id
      })
  })
  .catch ((err) => {
    console.log(err);
    return res.status(400).json({
      loginSuccess: false,
      message: err.message
    })
  })
})

//auth미들웨어 = 콜백함수가 호출되기 전에 인증처리를 하는 메서드
app.get('/api/users/auth', auth ,(req,res)=>{

})
```

```
app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

미들웨어 = 양 쪽을 연결하여 데이터를 주고 받을 수 있도록 중간에서 매개 역할을 하는 소프트웨어

웹 브라우저에서 데이터베이스로부터 데이터를 저장하거나 읽어올 수 있게 중간에 미들웨어가 존재하게 된다

middleware 디렉터리를 생성하고 auth.js 생성

middleware > auth.js

```
//User.js를 불러오고
const { User }= require("../models/User")

let auth = (req, res, next)=> {
        //인증 처리를 하는곳

    // client 쿠키에서 토큰을 가져온다.
    let token = req.cookies.x_auth;

    //토큰이 잘 들어있는지 테스트
    console.log(token)

      //토큰을 복호화 한후 유저를 찾는다
      //User를 통해 findByToken이라는 메서드를 호출하여 복호화 이때 인자로 client의 토큰을 전달한다
    User.findByToken(token)

    //유저가 있으면 인증 Okay!
    //유저가 없으면 인증 No!
}

module.exports = { auth };
```

토큰 복호화

User.js

```
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');
const saltRounds = 10;
const jwt = require('jsonwebtoken');

const userSchema = mongoose.Schema({
  name: {
    type: String,
    maxlength: 50
  },
  email: {
    type: String,
    trim: true, //띄어쓰기(빈칸)을 제거하는 역할
    unique: 1
  },
  password: {
    type: String,
    minlength: 5
  },
  role: { // 예) 넘버가 1이면 관리자고 넘버가 0이면 일반유저
    type: Number,
    default: 0
  },
  image: String,
  token: { // 토큰을 이용해 나중에 유효성 관리를 할 수 있음
    type: String
  },
  tokenExp: { //토큰을 사용할 수 있는 기간
    type: Number
  }
})
```

```
userSchema.pre('save', function(next){
  const user = this;
  if(user.isModified('password')){
    bcrypt.genSalt(saltRounds, function(err, salt) {
      if(err) return next(err)
      bcrypt.hash(user.password, salt, function(err, hash)
{
        if(err) return next(err);
        user.password = hash;
        return next();
      });
    });
  } else {
    return next();
  }
})

userSchema.methods.comparePassword = function(plainPasswor
d) {
  return bcrypt.compare(plainPassword, this.password)
}

userSchema.methods.generateToken = function() {
  const token = jwt.sign(this._id.toJSON(), 'secretToken');
  this.token = token;
  return this.save();
}

//주어진 토큰을 검증하고 해당 토큰이 유효한 사용자인지 확인하는 기능을 수
행
userSchema.statics.findByToken = function(token, cb){//toke
n = 클라이언트로부터 받은 JWT토큰, cb는 콜백함수
  const user = this;

    //토큰 복호화(디코딩 = 암호화된 데이터를 원래의 형태로 되돌리는 과
정)
  jwt.verify(token,'secretToken', function(err, decoded){
      // token을 디코드(복호화)해서 userId를 사용하여 DB에서 유저를
```

```
찾은 후,
      // 클라이언트에서 가져온 token과 DB에 보관된 token이 일치하는
지 확인
      user.findOne({"_id": decoded, "token": token})
      .then((user)=>{ //token이 일치하면 err = null과 user정보
를 콜백함수로 전달
        cb(null, user);
      })
      .catch((err)=>{ //token이 일치하지 않으면 콜백함수로 에러를
전달
        return cb(err);
      })
  })
}

const User = mongoose.model('User', userSchema);

module.exports = { User }
```

토큰 복호화는 방법은 jwt공식문서를 보면 나온다.

https://www.npmjs.com/package/jsonwebtoken

```
// verify a token symmetric - synchronous
var decoded = jwt.verify(token, 'shhhhh');
console.log(decoded.foo) // bar

// verify a token symmetric
jwt.verify(token, 'shhhhh', function(err, decoded) {
  console.log(decoded.foo) // bar
});

// invalid token - synchronous
try {
  var decoded = jwt.verify(token, 'wrong-secret');
} catch(err) {
  // err
}
```

middleware > auth.js

```
const { User }= require("../models/User")

let auth = (req, res, next)=> {
    let token = req.cookies.x_auth;

        //콜백 함수의 파라미터로 err와 user를 전달 받고
    User.findByToken(token, (err,user)=>{
            //err와 user정보 확인
            console.log(err, user)

            //err가 있으면 에러를 출력
        if(err) throw err;

        //user가 없으면 isAuth: false로 error: true로 반환
        if(!user) return res.json({ isAuth: false, error: t
rue})

        //user와 token정보를 req에 넣어 줍니다.
        req.token = token;
        req.user= user;

        //next()를 호출하여 auth미들웨어를 빠져 나갑니다.
        next();
    })

}

module.exports = { auth };
```

index.js

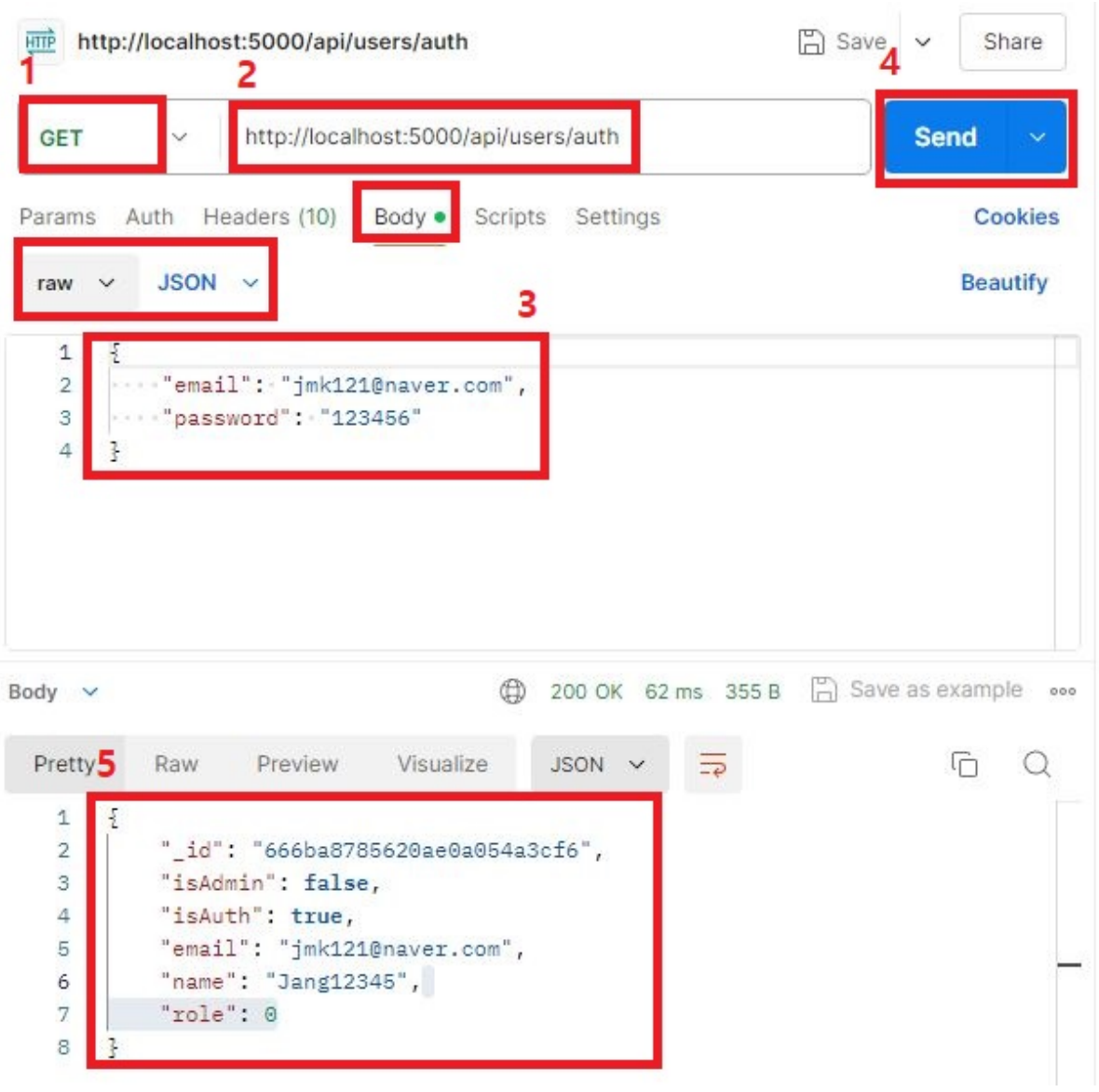
```
const express = require('express')
const app = express()
```

```
const port = 5000
const cookieParser = require('cookie-parser')

const config = require('./config/key');
const { User } = require("./models/User")
const { auth } = require("./middleware/auth")

app.use(express.json())

app.use(cookieParser());

const mongoose = require('mongoose')
mongoose.connect(config.mongoURI).then(() => console.log('M
ongoDB Connected...'))
  .catch(err => console.log(err))

app.get('/', (req, res) => {
  res.send('Hello World')
})

app.post('/api/users/register', async (req, res) => {
  const user = new User(req.body);
  await user.save().then(() => {
    res.status(200).json({
      success: true
    })
  }).catch((err) => {
    res.json({ success: false, err })
  })
})

app.post('/api/users/login', (req, res) => {
  User.findOne({ email: req.body.email })
  .then(async (user) => {
    if(!user){
      throw new Error("제공된 이메일에 해당하는 유저가 없습니다.")
    }
    const isMatch = await user.comparePassword(req.body.pas
```

```
sword);
    return { isMatch, user };
  })
  .then(({ isMatch, user }) => {
    console.log(isMatch);
    if (!isMatch) {
      throw new Error("비밀번호가 틀렸습니다.")
    }
    return user.generateToken();
  })
  .then ((user) => {
    return res.cookie("x_auth", user.token)
      .status(200)
      .json({
        loginSuccess:true,
        userId: user._id
      })
  })
  .catch ((err) => {
    console.log(err);
    return res.status(400).json({
      loginSuccess: false,
      message: err.message
    })
  })
})

//auth.js에서 next()가 호출되면 auth미들웨어에서 코드의 실행이 콜백함
수로 이동됩니다.
app.get('/api/users/auth', auth ,(req,res)=>{

    //여기까지 통과해 왔다는 의미는 Auth가 true라는 것
    res.status(200).json({
      _id: req.user._id,
      isAdmin: req.user.role === 0 ? false : true,
      isAuth: true,
      email: req.user.email,
      name: req.user.name,
```

```
      role: req.user.role,
      image: req.user.image
    })
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

테스트



로그아웃 기능

logout route 생성

index.js

```
const express = require('express')
const app = express()
const port = 5000
const bodyParser = require('body-parser')
const cookieParser = require('cookie-parser')

const config = require('./config/key');
const { User } = require("./models/User")
const { auth } = require("./middleware/auth")

app.use(bodyParser.urlencoded({extended: true}));
app.use(bodyParser.json())

app.use(cookieParser());

const mongoose = require('mongoose')
mongoose.connect(config.mongoURI).then(() => console.log('M
ongoDB Connected...'))
  .catch(err => console.log(err))

app.get('/', (req, res) => {
  res.send('Hello World')
})

app.post('/api/users/register', async (req, res) => {
  const user = new User(req.body);
  await user.save().then(() => {
    res.status(200).json({
      success: true
    })
  }).catch((err) => {
```

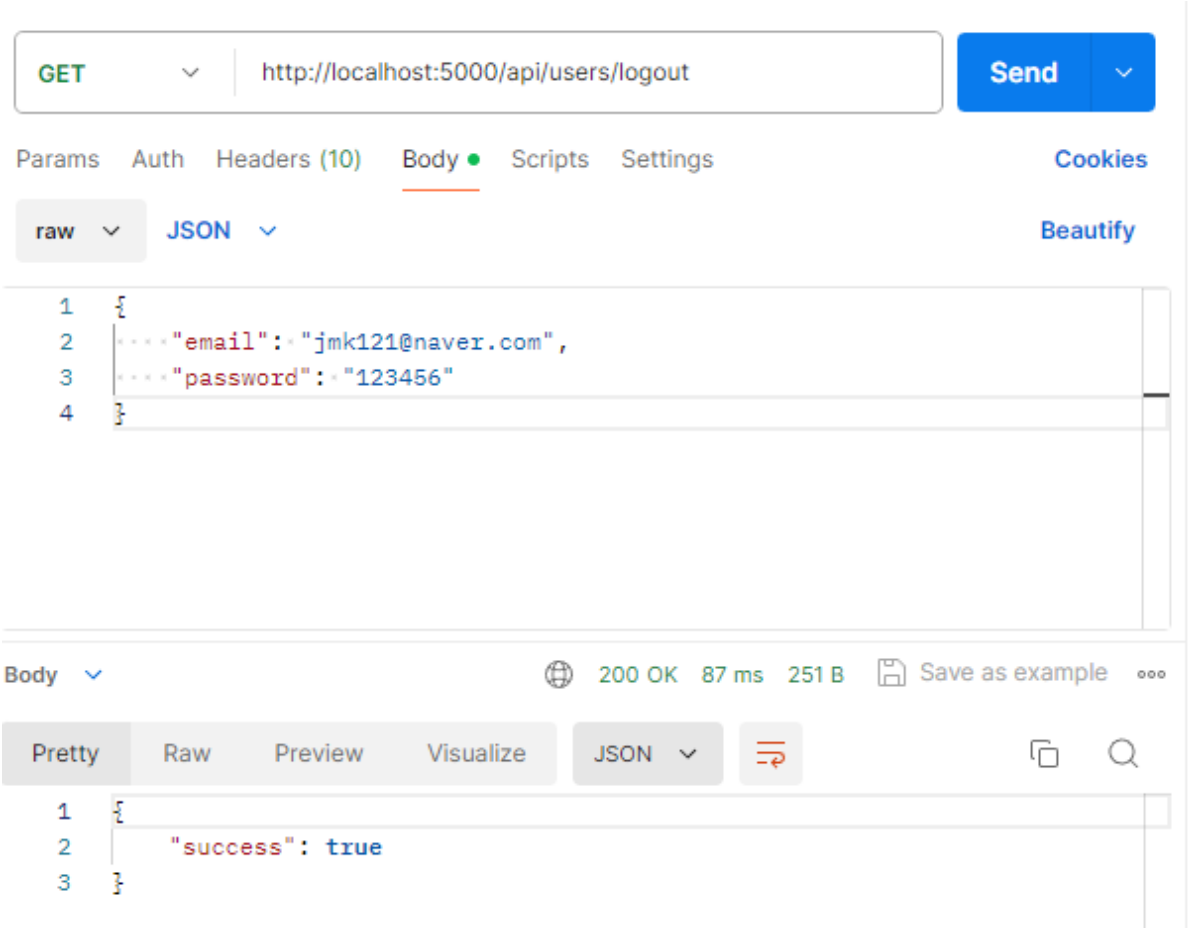
```
    res.json({ success: false, err })
  })
})

app.post('/api/users/login', (req, res) => {
  User.findOne({ email: req.body.email })
  .then(async (user) => {
    if(!user){
      throw new Error("제공된 이메일에 해당하는 유저가 없습니다.")
    }
    const isMatch = await user.comparePassword(req.body.pas
sword);
    return { isMatch, user };
  })
  .then(({ isMatch, user }) => {
    console.log(isMatch);
    if (!isMatch) {
      throw new Error("비밀번호가 틀렸습니다.")
    }
    return user.generateToken();
  })
  .then ((user) => {
    return res.cookie("x_auth", user.token)
    .status(200)
    .json({
      loginSuccess:true,
      userId: user._id
    })
  })
  .catch ((err) => {
    console.log(err);
    return res.status(400).json({
      loginSuccess: false,
      message: err.message
    })
  })
})
```

```
app.get('/api/users/auth', auth ,(req,res)=>{
  res.status(200).json({
      _id: req.user._id,
      isAdmin: req.user.role === 0 ? false : true,
      isAuth:true,
      email: req.user.email,
      name: req.user.name,
      role: req.user.role,
      image: req.user.image
  })
})

//로그 아웃
app.get('/api/users/logout', auth, (req, res) => {
  //DB에서 id로 user를 찾고, token을 초기화 시켜준다.
  User.findOneAndUpdate({ _id: req.user._id }, { token: ""
})
    .then(() => {
      console.log(req.user._id);
      res.status(200).send({ success: true }) //로그아웃에 성
공하면 success: true를 반환
    })
    .catch((err) => {
       //로그아웃에 실패하면 success:false와 에러 객체를 반환
      res.json({ success: false, err });
    })
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

User.findOneAndUpdate()

유저를 찾아서 업데이트 해주는 메서드

테스트



MongoDB에 토큰이 없어지면 로그아웃이 잘 된 것임

```
_id: ObjectId('666ba8785620ae0a054a3cf6')
name : "Jang12345"
email : "jmk121@naver.com"
password : "$2b$10$R/LvyUVGalzEYxHvJEQMe.GMmpVmu3NozsFvICEmKEPP4NyNoxaai"
role : 0
__v : 0
token : ""
```