로그인 기능 만들기(token생성)

login route 만들기

index.js

```
const express = require('express')
const app = express()
const port = 5000

const config = require('./config/dev');
const { User } = require("./models/User")

app.use(express.json())

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) => {
```

```
    //입력한 이메일과 같은 이메일 값을 가지는 데이터가 DB에 있는지 확인
    User.findOne({ email: req.body.email })

    //DB에 입력한 이메일 값과 일치하는 데이터가 있으면 파라미터로 입력
한 이메일과 일치하는 유저 정보를 받을 수 있다.
    .then(async (user) => {
    if(!user){
      throw new Error("요청받은 이메일에 해당하는 유저가 없습니
다.")
    }

    //입력되는 이메일과 일치하는 유저 정보가 있으면 comparePassword
메서드로 입력되는 비밀번호를 인자로 전달(이때 함수 이름은 바꿔도 된다.)
    const isMatch = await user.comparePassword(req.body.pas
sword);
  })
})

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

User.findOne(검색조건) = 하나의 문서를 찾는 역할을 한다.

User.findById(id) = _id를 기준으로 하나의 문서를 찾는다.

models > User.js

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

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(plainPassword) {
  // 사용자가 입력한 비밀번호와 DB에 암호화된 비밀번호가 같은지 체크
  // 비밀번호가 일치하면 true, 일치하지 않으면 false를 반환
  return bcrypt.compare(plainPassword, this.password)
}

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

module.exports = { User }
```

bcrypt.compare(사용자가 입력한 비밀번호, DB에서 검색한 데이터의 비밀번호)

DB에 실제 비밀번호가 아니라 해시값이 저장되어 있다.

그렇기 때문에 로그인할 때 실제 비밀번호가 아니라 해시값으로 비교를 해서 비밀번호 일치 여부를 따져봐야한다.

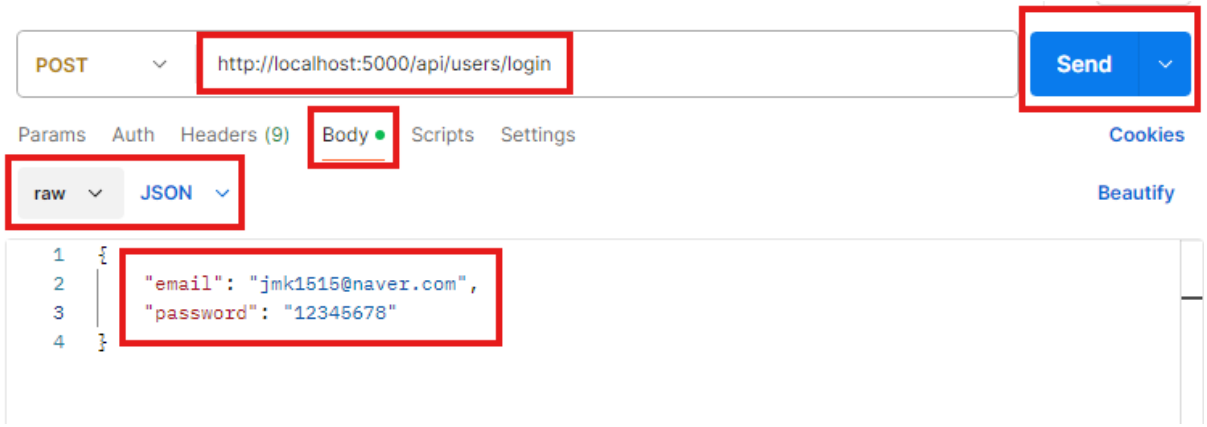
bcrypt.campare는 해시값으로 비밀번호를 비교한다.

Token 생성을 위해 로그인 성공 여부와 유저 정보를 추출

index.js

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

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

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

```
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/login', (req, res) => {
    User.findOne({ email: req.body.email })
    .then(async (user) => {
    if(!user){
      throw new Error("요청받은 이메일에 해당하는 유저가 없습니
다.")
    }
    const isMatch = await user.comparePassword(req.body.pas
sword);
    //isMatch와 user정보를 리턴
    return { isMatch, user };
  })

    .then(({ isMatch, user }) => {
    console.log(isMatch, user);
  })
})

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




```
MongoDB Connected...
true {
  _id: new ObjectId('672b7695e3cbcc4132e1339c'),
  name: 'JANG1234',
  email: 'jmk1515@naver.com',
  password: '$2b$10$YhJEQolOgTaiYe5v7agm9OyBFW6ugadWb6563W1rEi45gYCYnvYlW',
  role: 0,
  __v: 0
}
```

jwt API를 활용하여 토큰 생성

index.js

```
    ...

    .then(({ isMatch, user }) => {
    console.log(isMatch);
    if (!isMatch) { //비밀번호가 일치하지 않으면 에러 메시지를 출력
      throw new Error("비밀번호가 틀렸습니다.")
    }
    //토큰을 생성하기 위한 메서드로 user에 generateToken메서드를 호출
    return user.generateToken();
  })
})

app.listen(port, () => {
```

```
  console.log(`Example app listening on port ${port}`)
})
```

jwt 다운로드(토큰을 생성하기 위한 라이브러리)

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

```
npm i jsonwebtoken --save
```

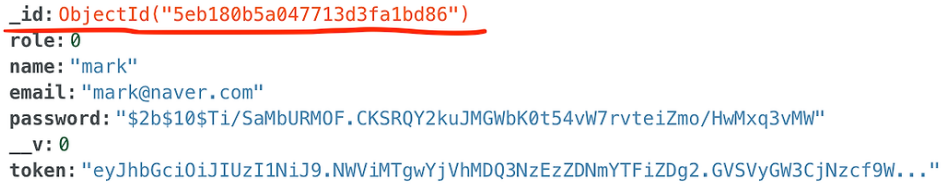
models > User.js

```
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)
}

//토큰 생성을 위해 generateToken메서드를 생성(메서드 이름은 변경해도
됨)
userSchema.methods.generateToken = function() {
    const user = this; //this = userSchema -> user = userSc
hema
  // jwt 생성
  const token = jwt.sign(user._id.toJSON(), 'secretToken');

  //생성된 토큰을 userSchema의 token 필드에 넣어줌
  this.token = token;
```

```
    console.log(this.token) //토큰을 출력
    //save메서드로 DB에 저장하고 리턴
  return this.save();
}

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

module.exports = { User }
```

ObjectId 클래스는 MongoDB 도큐먼트의 기본키이고, **_id 필드**에 해당된다



jwt.sign(payload, secretOrPrivateKey, [options, callback])

payload는 **Buffer**나 **string**이어야 하므로 _id 필드를 **payload**로 사용하려면 plain object로 바꿔주는 작업이 필요하다.

따라서 **toJSON()** 또는 **toHexString()**으로 plain object로 만들어 주어야 한다.

Buffer = Node.js 에서 제공하는 Binary 의 데이터를 담을 수 있는 Object 입니다.

생성된 토큰을 쿠키로 저장

cookie-parser 다운로드

https://www.npmjs.com/package/cookie-parser

```
npm i cookie-parser --save
```

index.js

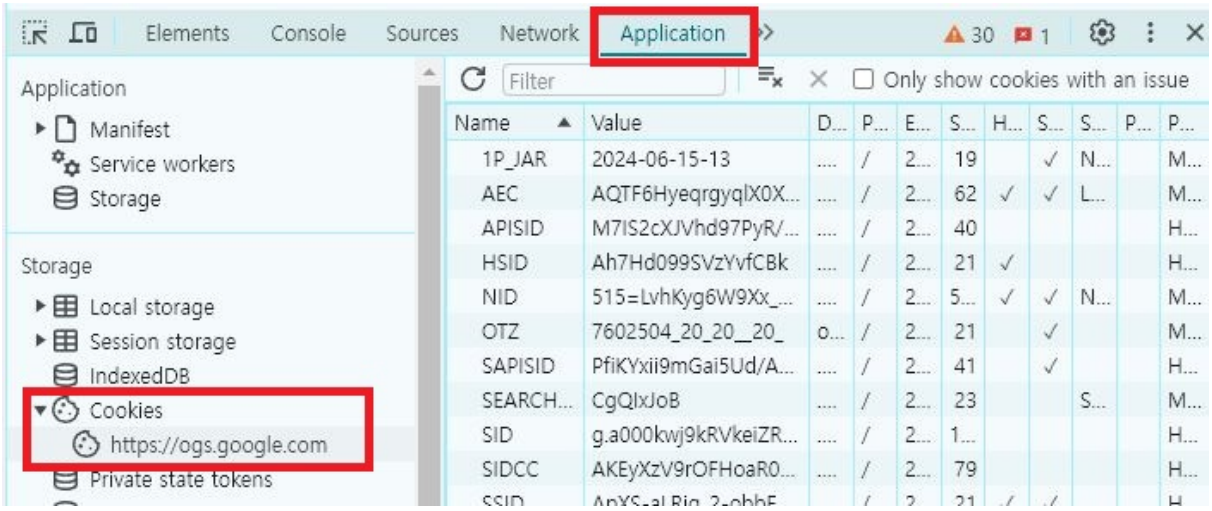
```
const express = require('express')
const app = express()
const port = 5000
const cookieParser = require('cookie-parser')

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

app.use(express.json())

//cookie-parser
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();
  })
    // generateToken메서드로 생성된 토큰을 user파라미터로 받음
  .then ((user) => {
    //토큰을 쿠키로 저장 res.cookie(쿠키 이름, 쿠키에 저장할 데이터
(토큰))
    return res.cookie("x_auth", user.token)
    //쿠키 저장이 성공하면 DB의 _id값을 전달
    .status(200)
    .json({
      loginSuccess:true,
      userId: user._id
    })
  })
  //에러가 발생하면 에러메시지 전달
  .catch ((err) => {
    console.log(err);
    return res.status(400).json({
      loginSuccess: false,
      message: err.message
    })
  })
})

app.listen(port, () => {
```

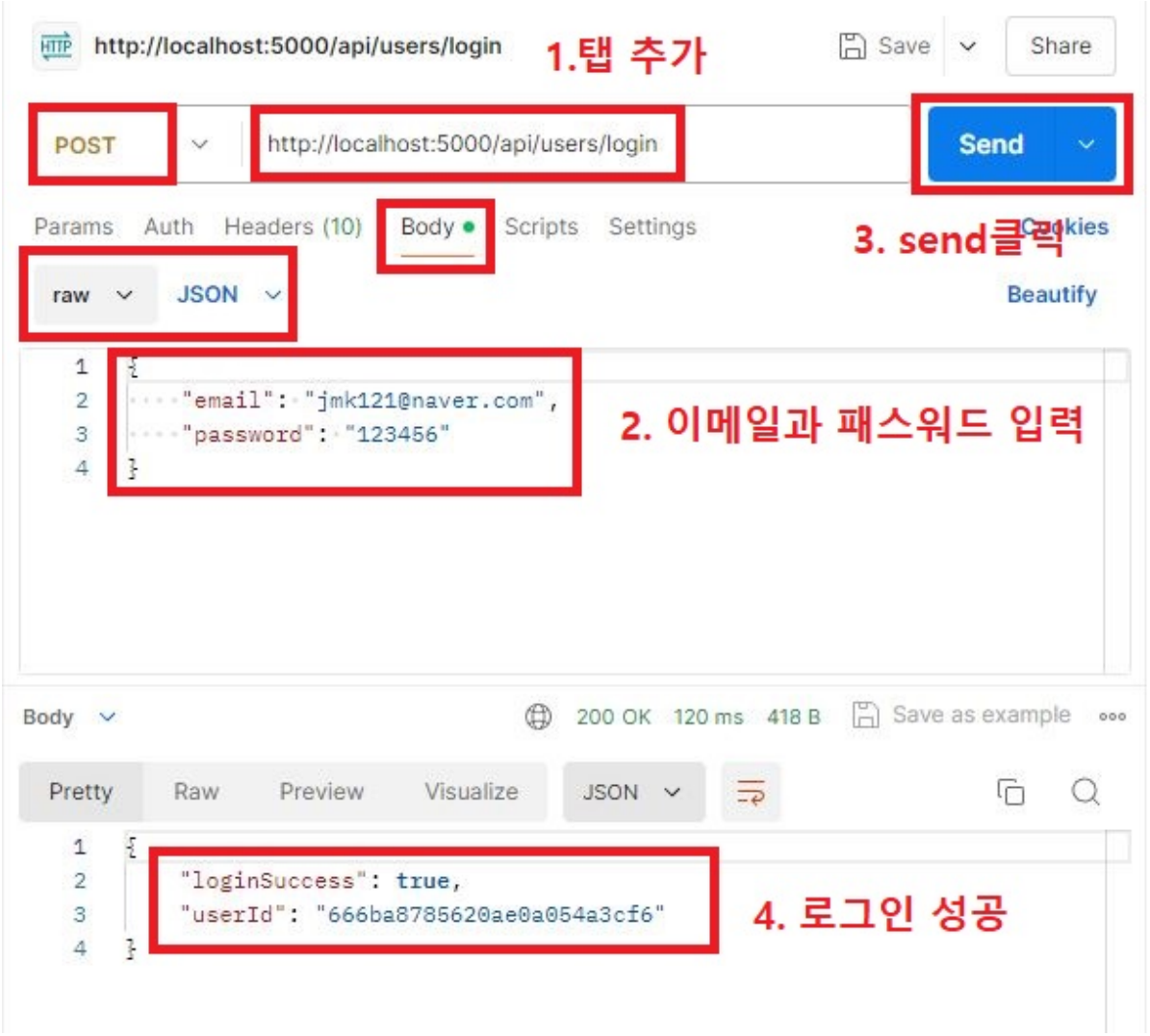
```
  console.log(`Example app listening on port ${port}`)
})
```

return res.cookie(쿠키 이름, 쿠키에 저장할 데이터(토큰))

아래는 예시로 쿠키가 저장되는 곳을 설명하기 위한 것



테스트



로그인이 성공하면 MongoDB의 해당 유저 아이디에 token이 생성됨

```
_id: ObjectId('666ba8785620ae0a054a3cf6')
name : "Jang12345"
email : "jmk121@naver.com"
password : "$2b$10$R/LvyUVGalzEYxHvJEQMe.GMmpVmu3NozsFvICEmKEPP4NyNoxaai"
role : 0
__v : 0
token : "eyJhbGciOiJIUzI1NiJ9.NjY2YmE4NzglNjIwYWUwYTAlNGEzY2Y2.OLRTVGwidZv0kijd…"
```