

Team Project 보고서 (3팀)

고급웹프로그래밍(Prof. 류은경)

ocrstagram



2016117285 이상민

2016111244 손승우

2016115631 안상준

2015116906 김도형

목차

제 1장 소개

1-1. 배경

1-2. 목표

제 2장 환경설정

2-1. 개발 환경

제 3장 기능

3-0. 기능 개요

3-1. 로그인

3-2. 회원가입

3-3. 사용자 화면

3-4. 포스팅

3-5. OCR

3-6. 번역

3-7. DB

Detail |

01

소 개

소개에서는 해당 프로젝트 주제 선정 배경, 목표와 프로젝트 수행방법에 대해 기술하였다.

제 1장 소개

1-1. 배경

- 기존의 식상한 SNS의 기능에서 벗어나서 나만의 기록을 남기기 위함
 - 현재 SNS는 단순히 사진을 올리고 글을 적는것에 그치는데 반해서 OCRstagram은 자동으로 OCR기능을 통해서 그 사진의 키워드를 자동으로 글자로 변환함으로써 색다른 SNS사용경험을 제공함.
- 문서 번역 과정에서의 불필요한 작업 최소화
 - OCR 기능을 통해 변환된 문서를 별다른 작업을 거치지 않고 파파고 API를 활용하여 보다 쉽고 빠르게 문서를 번역할 수 있음.

1-2. 목표

- 로그인 및 회원가입 기능 구현
 - 몽고DB를 통해서 회원가입시 계정정보를 저장하고 , DB상의 정보와 대조를 통해서 로그인 기능을 구현한다.
- 프로필별 게시물
 - 각 게시물별로 게시자의 ID정보를 통해서 구분을 할수있다
- 게시물 OCR기능 및 번역
 - 게시물 업로드시 OCR버튼을 클릭시 게시물을 OCR처리와 번역작업을 거쳐 글자로 나오게 된다.

Detail |

02

환경 설정

환경설정에서는 해당 프로젝트 서비스 환경 구축, 서버 자원에 대해 기술하였다.

제 2장 환경설정

2-1. 개발 환경



- **Visual Studio Code**

Node.js 개발에 사용하는 대표적인 에디터

개발의 편의성을 위해 사용



- **Node.js Ver : 15.12.2**

자바스크립트를 동일하게 사용해서 서버단 로직을 처리할 수 있는 Node.js 선택

다양한 모듈(패키지)를 제공하여 개발 속도와 효율성이 매우 높다



- **React.js Ver : 17.0.2**

사용자와 상호작용할 수 있는 UI를 손쉽게 만들 수 있는 라이브러리
다양한 프레임워크나 라이브러리와 혼용하여 사용하기 쉽다



- **AWS EC2**

AWS EC2를 사용하여 본 프로젝트 ocrstargram의 호스팅을 하였다

Detail |

03

기능

기능에서는 해당 프로젝트의 전체적인 흐름과 주요 기능에 대해 기술하였다.

3-0. 기능 개요



프론트엔드의 개발을 위해 익숙한 자바스크립트 프론트엔드 개발 라이브러리인 *React.js*를 이용하였다.

홈페이지를 SPA(Single Page Application)으로 작성할수 있는 장점이 있으며 사용자에게 빠른 화면전환과 모바일 환경에서의 확장성을 개발자에게 제공할수 있게 된다.

또한, 자바스크립트 기반의 라이브러리라고 언급했듯이, *Node.js*에 익숙할경우 동시에 개발을 진행할수 있으며 build하여 *Node.js*에 라우팅만 해주면 배포준비가 완료되기 때문에 빠른 속도로 작업할수있는 장점이 있다.



백엔드의 개발을 위해선 *Node.js*기반의 프레임워크 *express*를 사용하였다.

express 서버는 논블로킹 I/O, 싱글스레드 방식이기 때문에 이미지나 텍스트가 자주 오고가는 해당 프로젝트에 적합하다고 판단하였다. 또한 자바스크립트 기반이기 때문에 *React.js*와 협업하기 좋다는 장점이 있다.

개발 속도가 *Spring*에 비해 빠르고 *express*라는 강력하고 편리한 프레임워크가 존재하기 때문에 빠른속도로 개발을 할수있다는 장점이 있다.

또한 MVC 모델 기반의 서버를 구축하여 Mode, View, Controller를 구분하여 코드의 수정과 에러의 탐지, 유지보수를 간편하게 하였다.

3-1. 첫 화면 및 로그인




ocrstagram


이미지를 자유롭게 변환해 보세요

시작하기

<http://54.159.40.14:8080>으로 접속할시 나타나는 메인

ocrstagram

 ID

 Password

Login

New to us?

메인화면에서 시작하기 버튼을 누를 경우 로그인 페이지로 넘어간다.

SPA의 특성상 라우팅만 변경해주면 프론트단에서 렌더링을 해주기 때문에 특별히 서버의 요청 없이 렌더링 된다.

```
const onSubmitHandler = (event) => {
  axios.post(`http://54.159.40.14:8080/${ID}`, {
    id: ID,
    pw : Password
  }).then((result) => {
    if(result.data === true){
      console.log(result.data)
      const expires = new Date()
      expires.setDate(Date.now() + 1000 * 60 * 60 * 24 * 14)
      cookie.save('id', ID, {
        path: '/',
        expires,
      })
      window.location.href = `/${ID}`
    }
    else
      window.location.href = '/user/login'
  }).catch((err) => {
    console.error(err)
  })
}
```

로그인 form에 SubmitHandler를 달아서 서버에 post 방식으로 id와 pw를 보낸다.

true 응답을 받으면 쿠키를 생성하고 해당 브라우저의 경로를 <http://54.159.40.14:8080/id>로 연결해준다.

false 응답을 받게 된다면 /user/login으로 리다이렉트를 해준다.

서버에서는 MVC 방식을 사용하여 post에 대한 controller를 달아주고 해당 controller에서 로그인 처리를 해준다.

app.post('/:id', loginUserController)

```
const fs = require('fs')
const path = require('path')
const User = require('../models/User')
const bcrypt = require('bcrypt')

module.exports = (req, res, next) => {

  User.findOne({ userid: req.body.id }, (error, user) => {
    console.log(user)
    if (user) {
      console.log(req.body)
      bcrypt.compare(req.body.pw, user.pw, (error, same) => {
        console.log(same)
        if (same) {
          res.send(true)
          if (!fs.existsSync(`${path.resolve(__dirname, '..', '..', 'public/img')}/${req.body.id}`)) {
            console.log('create folder')
            fs.mkdirSync(`${path.resolve(__dirname, '..', '..', 'public/img')}/${req.body.id}`)
          }
        }
        else {
          res.send(false)
        }
      })
    }
    else {
      res.send(false)
    }
  })
}
```

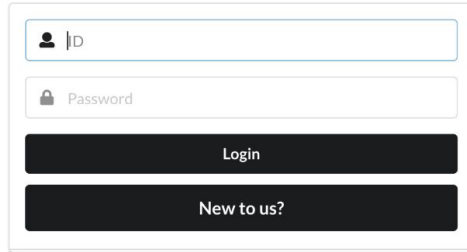
loginUserController에서는 mongoose와 bcrypt 방식을 이용하여 mongoose에서 입력받은 id에 해당되는 document를 찾고 bcrypt방식으로 해당 pw를 비교한다.

만약 pw가 같다면 포스팅 된 이미지의 저장을 위해 public 폴더 아래의 해당 아이디로 된 폴더를 생성해주고 true를 클라이언트 측으로 전송한다.

pw가 다르거나 id가 없다면 false를 전송해준다.

3-2. 회원가입

oerstagram

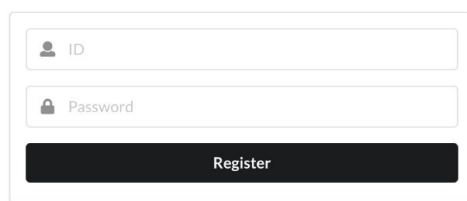


A login form with two input fields: 'ID' (with a user icon) and 'Password' (with a lock icon). Below the fields are two buttons: 'Login' and 'New to us?'.

로그인 화면에서 New to us?를 클릭할경우 회원가입 페이지로 넘어간다.

회원가입 페이지 또한 SPA의 특성상 프론트단에서 라우팅 처리를 해준다.

oerstagram



A registration form with two input fields: 'ID' (with a user icon) and 'Password' (with a lock icon). Below the fields is a single button: 'Register'.

해당 페이지에서 ID와 PW를 입력하고 Register button을 누르면 서버측의 DB에 저장을 요청한다.

```

onSubmitHandler = (event) => {
  event.preventDefault()
  axios.post(`http://54.159.40.14:8080/user/register`, {
    id : this.state.id,
    pw : this.state.pw
  }).then((result) => {
    console.log(result.data)
    window.location.href="/"
  }).catch((err) => {
    console.error(err)
  })
}
}

```

회원가입 form에 SubmitHandler를 달아서 서버에 post 방식으로 id와 pw를 보낸다.

```

app.post('/user/register', storeUserController);

```

서버에서는 MVC 방식을 사용하여 post에 대한 controller를 달아주고 해당 controller에서 회원가입 처리를 해준다.

앞으로의 나올 모든 요청을 받는 방식은 MVC 방식을 이용했기 때문에 언급을 생략하도록 하겠다.

```

module.exports = async (req, res) => {
  console.log(req.body)
  User.findOne({ userid: req.body.id }, (error, user) => {
    console.log(user)
    if (user!=null) {
      res.send(false)
    }
    else {
      User.create({ userid: req.body.id, pw: req.body.pw })
      res.redirect('/')
    }
  })
}
}

```

storeUserController에서는 전달받은 id가 있는지 없는지 살펴보고 없다면 UserShema를 이용하여 User를 생성해준다.

3-3. 사용자화면

ocrstagram

hello1



ENGLISH
EDUCATION

ENGLISH
EDUCATION

Miraculous Achievement #9

Grand Master Rhee conducted the first Martial Arts School Business and Philosophy Seminar to revive the industry in 1985.

Grand Master Rhee began his martial arts school business in 1962 in Washington, DC. He was so successful that he was able to purchase a home in Arlington, Virginia in five years for \$60,000.00 in 1967. It seemed like a mansion to him at that time. A few days before he bought the house, he showed a photograph to his assistant instructor, Jack Dutcher, a tough former US Marine, to see his reaction. Dutcher said, "It is too big for you. People will think you're a Chinese houseboy." Jack always loved to tease his master instructor.

Sure enough, when Grand Master Rhee was mowing the lawn one sunny Sunday morning, a gentleman passing by in a black Cadillac convertible stopped his car and shouted, "Say, boy! How much an hour do you charge?" Master Rhee replied: "Nothing, sir, but I get to share the master bedroom with the madam who owns the house." The man drove away quietly.

Grand Master Rhee was one of very few martial artists who believed in heavy advertising in the 1960s. He spent a lot on the TV Guide section of The Washington Post and also on television advertising using the theme "Nobody Bothers Me." The jingle is still recognized today and can be seen on YouTube. He was known as one of the most successful martial arts school operators in the country.

In the 1980s, Grand Master Rhee was very concerned with the declining popularity of martial arts. Nobody thought of providing martial arts business seminars for struggling school owners. Somehow, most instructors had been influenced by their old Asian masters to feel that making money with martial arts is committing some sort of a crime. They were proud to say, "I teach martial arts for free." Master Rhee always asked them, "Do you think what you teach has no value?" He was one of a few who made his teaching Taekwondo very professional and was proud to do so.

Grand Master Rhee announced the first Martial Arts Business Seminar to be conducted in Fort Worth, Texas from July 4-6, 1985. The seminar began at 2:00 p.m. on Friday, went until midnight, then ran from 6:00 a.m. to midnight on Saturday, and from 6:00 a.m. to 5:00 p.m. on Sunday. There were thirty students attending: Pat Burleson, the owner of the Studio, Al Garza, H. K. Lee, George Menshew, Steve Oliver, Ishmael Robles, Jose Santamaria, and other notables.

The seminars became so popular that Grand Master Rhee conducted monthly seminars for several years. Those who attended these seminars and are still active in the martial arts industry are Pat Burleson, Dennis Brown, Tom Callos, Joon Chung, Adahki Fariborz, Jim Harrison,

38

My strict parents were picky about giving me permission to join in activities away from home, but they usually let me go with the church youth group. I was

로그인에 성공했다면 해당 사용자의 화면이 나타나게 된다. 사용자의 화면에 해당되는 사용자가 로그인했다면 우측 상단의 버튼이 활성화 되어 Posting과 Logout을 할수있게 되지만 만약 다른 사람의 Posting을 확인하기 위해 브라우저 라우터의 경로를 다른 사람의 id로 할경우 우측 상단의 버튼이

ocrstagram

hello2



Our Project is very good Our Project is very good

Make me Crazy

Make me Crazy

Very very good

Very very good

Very hot project

Very hot project

My strict parents were picky about giving me permission to join in activities away from home, but they usually let me go with the church youth group. I was so excited when Mrs. Burks, the church youth leader who planned most of our activities, announced we were going to the skating rink. I'd never been roller-skating and really wanted to learn how to skate. It looked like so much fun!

When we got there, everyone scrambled out of the cars and rushed inside. Some of us rented our skates and soon began lacing them up. We made our way across the carpet, held onto the handrails, and stepped onto the slick floor. Then, everyone but me rolled away from the wall and started skating.

Holding onto the handrail, I pulled myself along the side as I tried moving my legs back and forth. I stiffened up and clung to the bar as I began to lose my balance. Too late! I was on the floor. After a few more failed attempts, Mrs. Burks skated over to me and stopped.

"You need to learn to fall," said Mrs. Burks.

What? I thought the idea was to go zooming around on these roller skates—not to fall down! Most of the kids in our youth group and some of the chaperones were skating around the busy rink

해당 사용자 화면은 hello1 으로 로그인했을때 브라우저를 통해 hello2의 사용자 화면을 요청했을때의 모습이다.

우측 상단의 버튼이 비활성화 된것을 확인할수있다.

또한 해당 아이디가 존재하지 않는 경우 404 not found 페이지가 랜더링 되도록 하였다.

```
componentDidMount = () => {  
  axios.post(`http://54.159.40.14:8080/${this.state.id}/getDB`,  
    {userid : this.props.props.match.params.id})  
    .then((result) => {  
      console.log(result.data)  
      let rs = result.data  
      result.data.map((value) => {  
        dataList.push([value.image, value.text])  
      })  
      console.log(dataList)  
      console.log(this.props.props.match.params.id)  
      this.setState({  
        dbLength : result.data.length  
      })  
    })  
    .catch((err) => {  
      console.error(err)  
    })  
}
```

사용자가 포스팅 한 게시물을 가져오기 위해서 서버에 요청을 하여 사용자의 아이디로 작성된 게시물의 db list를 응답 받는다.

응답받은 db에는 ocr로 변환한 text와 ocr로 변환한 이미지의 경로가 포함되어있다.

해당 db의 결과를 이미지에 onClick 리스너에 달아주어 이미지를 클릭할경우 해당 이미지와 텍스트가 화면에 출력되게 하였다.


```
renderText = (src, text) => {
  console.log(src)
  console.log(text)
  document.getElementsByClassName('Posted')[0].style.visibility='visible'
  document.getElementsByClassName('PostedImg')[0].style.width = '100%'
  document.getElementsByClassName('PostedImg')[0].style.height = '100%'
  document.getElementsByClassName('PostedImg')[0].src = src
  document.getElementsByClassName('PostedText')[0].innerHTML = text
  console.log(document.getElementsByClassName('PostedImg')[0])
}

render() {
  if(this.state.dbLength.length === 0)
    return(
      <div>loading</div>
    )
  else
    return (
      <div className="GridView" style={divStyle}>
        <div style={divStyle2}>
          <GridList cellHeight='auto' cols={3} style={{ 'width': '90vw' }}>
            {dataList.map((data) => (
              <GridListTile style={tileStyle}>
                { /* <a href={data.image}>
                  </a> */}
                <Image
                  style={{ 'width': '100%', 'height': '100%' }}
                  src={`http://54.159.40.14:8080/${this.props.props.match.params.id}/getImg?image=${data[0]}`}
                  onClick={this.renderText.bind(this, `http://54.159.40.14:8080/${this.props.props.match.params.id}/getImg?image=${data[0]}`, data[1])}
                ></Image>
              </GridListTile>
            ))}
          </GridList>
        </div>
      </div>
    )
}
```

해당 이미지의 src에는 서버에 요청을 보내 src를 서버에서 받게 해주었다.

또한 click 리스너에 해당 이미지의 src경로와 텍스트를 전달하여 클릭하면 보여주게 될 Posted 레이어에 데이터를 전달하게 하였다.

Posted 레이어는 visible로 바뀔과 동시에 해당 이미지와 텍스트 속성을 클릭한 이미지의 src와 text로



3-4. 포스팅

ocrstagram

 octest



로그인을 한 후 뜨는 사용자 화면에서 우측 상단의 + 버튼을 클릭하여 포스팅을 할 수 있다.

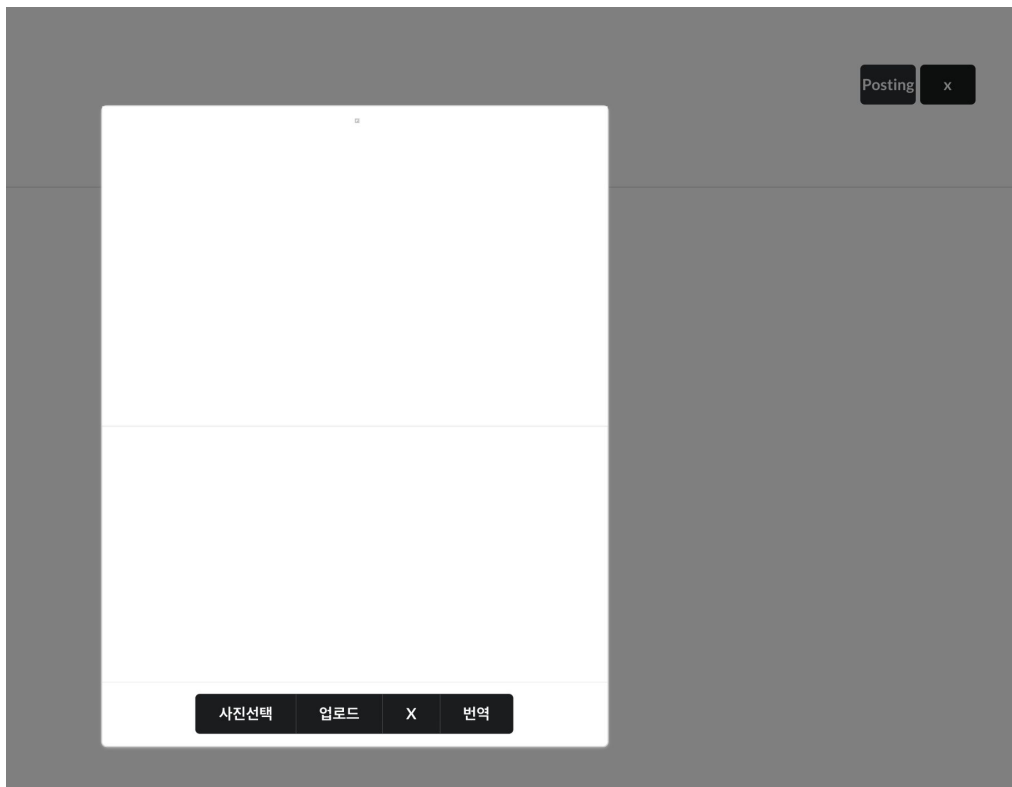


사진 선택을 통해 이미지를 선택한 후, OCR 또는 번역을 한 후에 업로드 버튼을 통해 서버로 업로드 할 내용을 전달할 수 있다.


```

handleUpload = (e) => {
  e.preventDefault()
  let fd = new FormData()

  fd.append('img', this.state.img)
  fd.append('text', document.getElementsByClassName('ocrResult')[0].innerHTML)

  axios.post(`http://54.159.40.14:8080/${this.state.id}/post`, fd, {
    headers: {
      'Content-Type': 'multipart/form-data'
    }
  })
  .then((result) => {
    console.log(result)
    window.location.href=`/${this.props.props.match.params.id}`
  })
  .catch((err) => {
    console.error(err)
  })
}

```

업로드 버튼을 누르면 파일의 전송이기 때문에 multipart 형식으로 서버에 요청을 한다.
그 후 포스팅된 이미지의 업데이트를 위해 해당 아이디의 경로로 redirect를 해준다.

```

let img = req.files.img
let id = req.params.id

let temp = new Date().getTime().toString()
let imgPath = path.resolve(__dirname, '..', '..', `public/img/${req.params.id}`, temp)

```

controller 중 storePost.js 에서 request를 통해 img를 받아오고, 현재 시간을 기준으로 하여 저장할 이미지의 path이름을 정해준다.

```

img.mv(imgPath, async(err) => {
  User.findOne({userid: id}, (error, user) => {
    let _id = user._id
    Post.create({
      text: req.body.text,
      image : '/img/' + id + '/' + temp,
      username : _id
    })
    res.send('ok')
  })
})

```

req를 통해 받아온 id를 토대로 User collection의 id를 검색하고, Post collection에 image 경로와 username을 포함하여 데이터베이스 포스팅을 저장할 수 있다.

```

ubuntu@ip-172-31-19-34:~/WebProgramming/ocrstagram/public/img$ ls
123123 Web hello1 hello2 otest son123
ubuntu@ip-172-31-19-34:~/WebProgramming/ocrstagram/public/img$ cd Web
ubuntu@ip-172-31-19-34:~/WebProgramming/ocrstagram/public/img/Web$ ls
1623488782891 1623488800847
ubuntu@ip-172-31-19-34:~/WebProgramming/ocrstagram/public/img/Web$ cd ../hello2
ubuntu@ip-172-31-19-34:~/WebProgramming/ocrstagram/public/img/hello2$ ls
1623488429433 1623488528434 1623488737783
1623488462748 1623488549753 1623489502876
ubuntu@ip-172-31-19-34:~/WebProgramming/ocrstagram/public/img/hello2$

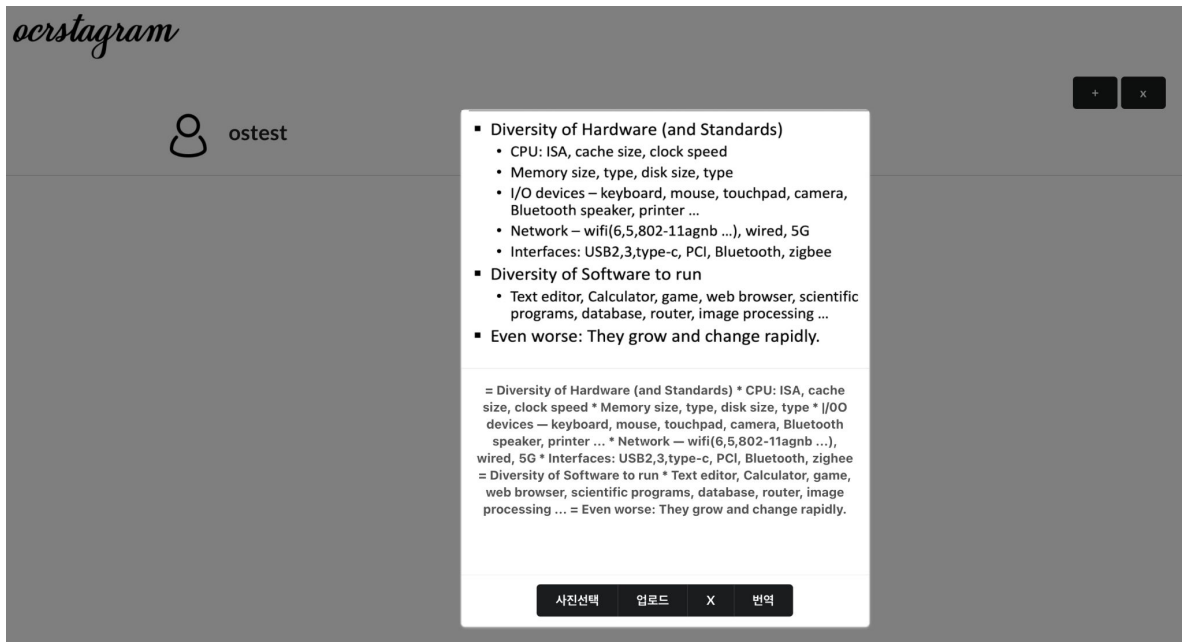
```

해당 사진은 실제 서버에서 생성된 아이디와 이미지가 저장되어있는 모습이다.

3-5. OCR

포스팅을 할 이미지를 업로드 한 후 이미지를 클릭하면 이미지의 텍스트를 OCR을 통해 추출할 수 있다.

OCR은 *tesseract.js*를 활용하여 텍스트를 추출하였다.



ocr 전환할 이미지를 클릭하면 *onClick* 리스너에 의해 서버에 요청하여 해당 이미지의 ocr 결과를 리턴받는다.

```
handleOcr = (e) => {
  e.preventDefault()
  let fd = new FormData()
  fd.append('img', this.state.img)
  document.getElementsByClassName('ocrLoading')[0].style.display = 'block'
  axios.post(`http://54.159.40.14:8080/${this.state.id}/ocr`, fd, {
    headers: {
      'Content-Type': 'multipart/form-data'
    }
  }, {withCredentials: true})
  .then((result) => {
    document.getElementsByClassName('ocrLoading')[0].style.display = 'none'
    document.getElementsByClassName('ocrResult')[0].innerHTML = result.data
    console.log(result.data)
  })
  .catch((err) => {
    console.error(err)
  })
}
```

이미지를 업로드 해줘야 하기 때문에 *multipart* 형식으로 이미지를 전달하고, 응답 받은 *data*를 결과화면에 출력한다.

```
const asyncHandler = require('express-async-handler');
const path = require('path')
const fs = require('fs')

module.exports = asyncHandler(async (req, res, next) => {
```

서버에서 ocr을 해서 전달을 해줄 때 비동기 처리를 해야하는데, express의 경우 router에서 비동기 함수를 쓸 때 에러 처리등이 자동으로 되지 않으므로 express-async-handler를 사용해서 비동기 처리를 정상적으로 작동하도록 할 수 있다.

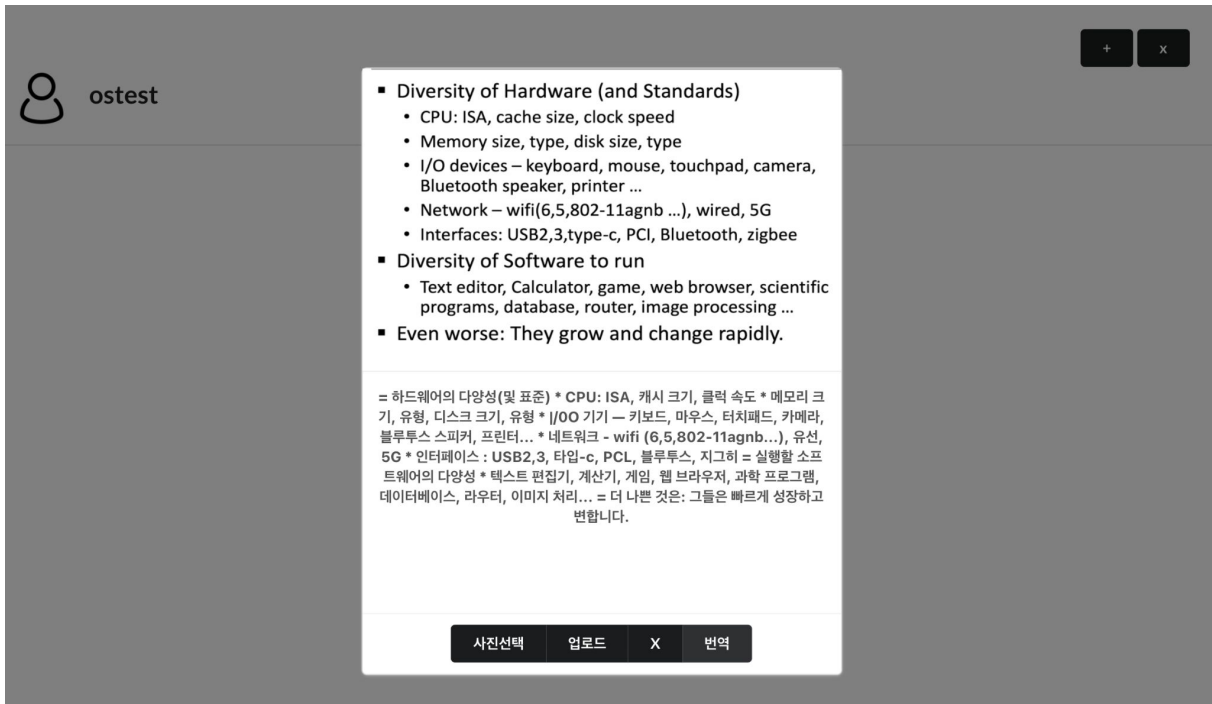
```
const { createWorker } = require('tesseract.js');
const asyncHandler = require('express-async-handler');
const path = require('path')
const fs = require('fs')

module.exports = asyncHandler(async (req, res, next) => {
  let img = req.files.img
  let temp = new Date().getTime().toString()
  let imgPath = path.resolve(__dirname, '..', '..', `public/img/${req.params.id}`, temp) // add getTime
  try {
    img.mv(imgPath, async(err) => {
      console.log('upload img to server')
    })

    const worker = createWorker()
    await worker.load();
    await worker.loadLanguage('eng');
    await worker.initialize('eng');
    // ocr img file
    const { data: { text } } = await worker.recognize(imgPath);
    console.log(text);
    // send ocr text
    // res.send('upload img to server\n' + text);
    res.send(text)
    await worker.terminate();
    fs.unlinkSync(imgPath)
  } catch(err) {
    console.error(err)
  }
})
```

사용자가 업로드한 이미지를 현재 시간에 따라 이름을 지정하여 저장해준다. 이미지를 받아서 저장한 후, tesseract의 createWorker를 통해 만든 worker의 recognize 함수를 통해 추출한 데이터를 res.send()를 통해 전송해준다.

3-6. 번역



OCR을 한 이후 번역 버튼을 통해 파파고 api를 사용하여 번역한 결과를 받을 수도 있다.
번역 또한 번역 버튼에 onClick 리스너를 달아 서버로 요청을 보낸다.
그 후 응답받은 결과를 화면에 출력해 준다.

```
var api_url = 'https://openapi.naver.com/v1/papago/n2mt'  
var request = require('request')
```

파파고 API를 호출하기 위한 url을 선언하고, post요청을 보내기 위해 request 라이브러리를 사용한다.

```
request.post(options, function (error, response, body) {  
  if (!error && response.statusCode == 200) {  
    res.send(JSON.parse(body).message.result.translatedText)  
  } else {  
    console.error(error)  
  }  
})
```

ocr을 했던 텍스트를 번역한 결과를 res.send를 통해 전송한다.

3-7. DB

데이터베이스는 *mongoDB*를 사용하였으며, *User* 와 *Post* 두 개의 컬렉션을 만들어 사용하였다.

먼저 *User*의 모델은 다음과 같다.

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema;
const bcrypt = require('bcryptjs')

const UserSchema = new Schema({
  userid: { // pass in config object. and put in validation rules
    type: String,
    required: true,
    unique: true
  },
  pw: {
    type: String,
  }
});

// note: no lambda func! (not work!)
UserSchema.pre('save', function(next){
  const user = this
  bcrypt.hash(user.pw, 10, (error, hash) => {
    user.pw = hash
    next()
  });
});

const User = mongoose.model('User', UserSchema);
module.exports = User
```

*User*의 *schema*는 *userid*와 *pw*로 구성돼있다.

*userid*의 경우 *required*로 설정해주었고, *unique* 한 *id*들만 저장되도록 했다.

*pw*의 경우 *bcrypt*를 통해 해싱한 정보를 *DB*에 저장하도록 했다.

Post의 모델은 다음과 같다.

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema

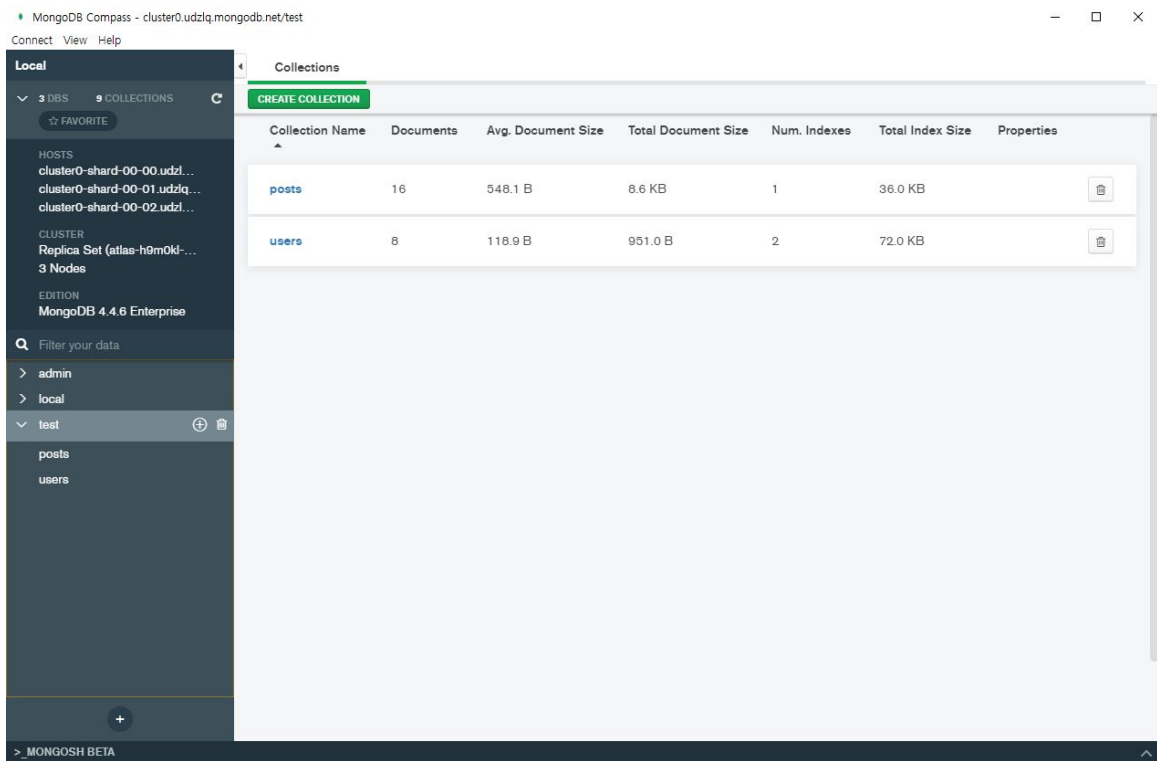
const PostSchema = new Schema({
  username: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  text: String,
  image: String
});

const BlogPost = mongoose.model('Post', PostSchema);
module.exports = BlogPost
```

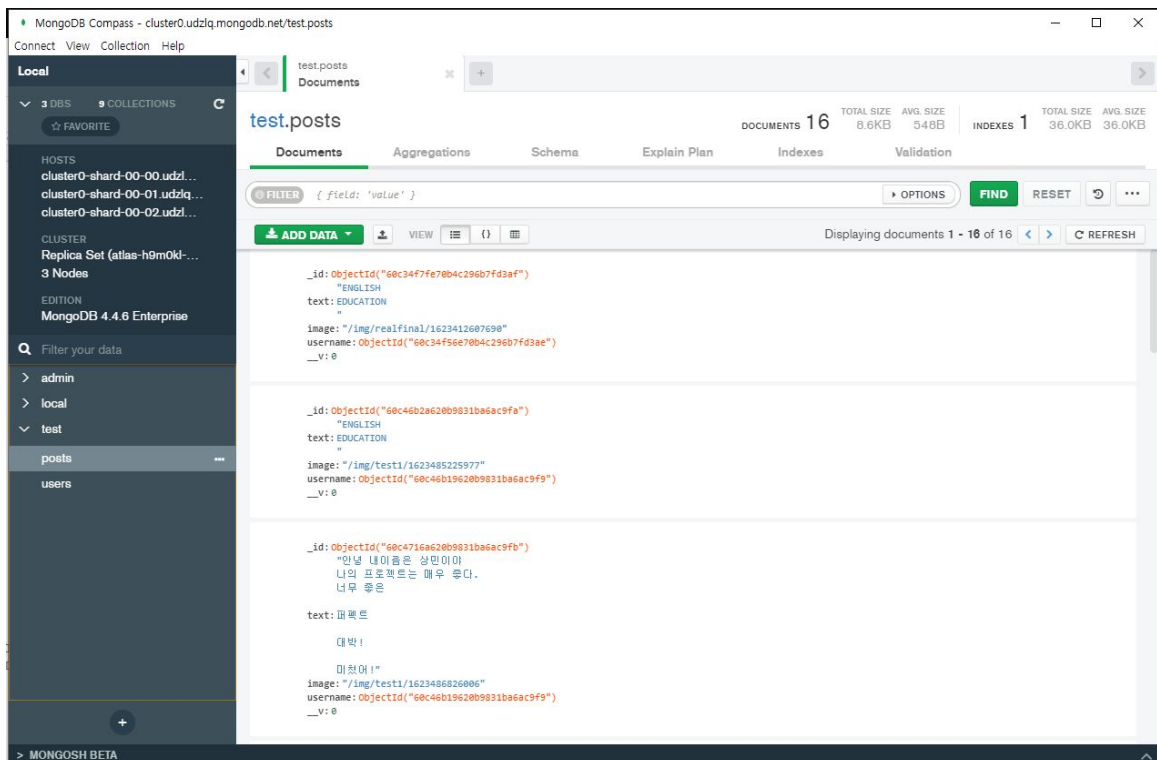
username의 경우 ref를 통해 user 모델에 저장한 것을 참조하도록 하였다.

text에는 이미지를 ocr하여 추출한 텍스트를 저장하였다.

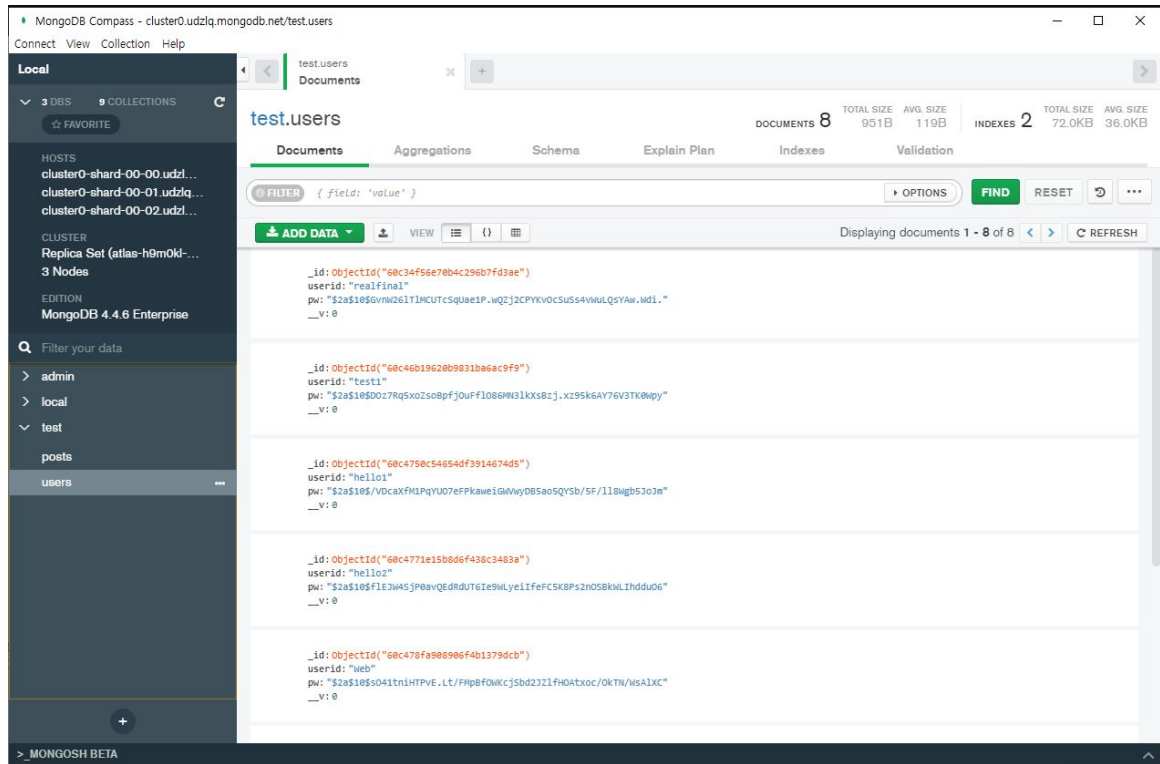
image에는 서버에 저장된 이미지의 경로를 저장하고 있고, 사용자 화면에서 업로드 한 이미지들을 보여줄 때 사용한다.



MongoDB compass를 통해 확인한 User와 Post 컬렉션이다.



실제로 저장된 Post 데이터이다.



실제로 저장된 User 데이터이다. pw가 해싱을 해서 저장된 것을 확인할 수 있다.