**Node.js 팀프로젝트***http 모듈로 서버만들기 프로그램 분석 보고서*

소프트웨어공학과 201732009 김윤서  
IT융합자율학부 201732028 이현복

각 파일의 역할과 실행 과정 및 파일간 관계를 기술했으며  
프로그램의 세부적인 내용은 해당 파일의 소스코드에 주석으로 기술함.   
가시성 및 보고서 내용 확인의 편의성 향상을 위해 펼치기/접기 가능한 목차를 활용했으며,  
**해당 목차 (파일명 CODE)** 좌측 화살표 클릭시 펼침/접힘.  
실행 결과 및 디버그 콘솔 화면 또한 문서 하단 실행 결과 목차 좌측 화살표 클릭 시 확인 가능.

# **restFront.css – 스타일시트**. 클라이언트에서 사용할 css 속성 정의 및 저장. 타 프로그램 파일에서 link로 참조

# **restFront.css CODE**

/\* a태그의 색을 나타내기 위한 css 파일 \*/

a {

  color: blue;

  text-decoration: none;

}

# **restFront.html – 프론트**. 각 페이지 하이퍼링크와 input/button 태그 정의. resFront 스크립트 사용. 등록 버튼 작동 시 인풋태그 내용 Get method로 서버로 전달 후 서버에서 핸들링 진행 후 전달된 div 표시

# **restFront.html CODE**

<!DOCTYPE html>

<html>

  <!-- head part -->

  <head>

    <!-- incodeing -->

    <meta charset="utf-8" />

    <!-- html page title -->

    <title>RESTful SERVER</title>

    <!-- reference css -->

    <link rel="stylesheet" href="./restFront.css" />

  </head>

  <!-- body part -->

  <body>

    <nav>

      <!-- href to about.html / itself -->

      <a href="/">Home</a>

      <a href="/about">About</a>

    </nav>

    <div>

      <!-- form to register new user -->

      <form id="form">

        <input type="text" id="username" />

        <button type="submit">등록</button>

      </form>

    </div>

    <div id="list"></div>

    <!-- use javascript -->

    <script src="https://unpkg.com/axios/dist/axios.min.js"></script>

    <!-- use restFront javascript file -->

    <script src="./restFront.js"></script>

  </body>

</html>

# **restFront.js – 백**.메인 메소드들과 액션이 진행되는 서버 스크립트. resFront 리스트 데이터 확인 후 해당 데이터로 새로운 태그 생성 후 restFront 페이지에서 표시. 유저 데이터 수정 메소드. 값 입력받고 데이터 restServer로 전달. PUT 유저 데이터 삭제 메소드. 데이터 restServer로 전달. DELETE

# **restFront.js CODE**

async function getUser() {

  // 로딩 시 사용자 정보를 가져오는 함수

  try {

    // /users에서 가져오는 json형식을 받아옵니다.

    // get 메서드 호출

    const res = await axios.get('/users')

    // users에 /users에서 받아온 data를 저장합니다.

    const users = res.data

    // id가 list인 태그를 변수 list로 저장

    const list = document.getElementById('list')

    // list의 내용은 빈칸이다.

    list.innerHTML = ''

    // 사용자마다 반복적으로 화면 표시 및 이벤트 연결

    Object.keys(users).map(function (key) {

      // div태그를 가진 userDiv 생성

      const userDiv = document.createElement('div')

      // span태그를 가진 span 생성

      const span = document.createElement('span')

      // span 안에 users의 값들을 넣는다.

      span.textContent = users[key]

      // button 태그를 가진 edit 생성

      const edit = document.createElement('button')

      // edit 안에 '수정' 내용을 넣는다.

      edit.textContent = '수정'

      // 수정 버튼 클릭

      edit.addEventListener('click', async () => {

        // 팝업?창으로 변경 이름 설정

        const name = prompt('바꿀 이름을 입력하세요')

        if (!name) {

          return alert('이름을 반드시 입력하셔야 합니다')

        }

        // /user/에 key값에 맞는 이름을 name으로 바꿔서 넣는다.

        // put 메서드 호출

        try {

          await axios.put('/user/' + key, { name })

          // 다시 함수 시작

          getUser()

        } catch (err) {

          console.error(err)

        }

      })

      // button 태그를 가진 remove 생성

      const remove = document.createElement('button')

      // edit 안에 '삭제' 내용을 넣는다.

      remove.textContent = '삭제'

      // 삭제 버튼 클릭

      remove.addEventListener('click', async () => {

        try {

          // /user/에 key값에 맞는 이름을 삭제

          // delete 메서드 호출

          await axios.delete('/user/' + key)

          // 함수 다시 시작

          getUser()

        } catch (err) {

          console.error(err)

        }

      })

      // list < userDiv < span, edit, remove 이러한 구조로 추가

      userDiv.appendChild(span)

      userDiv.appendChild(edit)

      userDiv.appendChild(remove)

      list.appendChild(userDiv)

      console.log(res.data)

    })

  } catch (err) {

    // 만일 오류가 나오면 catch문을 받아 err를 console에 출력한다.

    console.error(err)

  }

}

window.onload = getUser // 화면 로딩 시 getUser 호출

// 폼 제출(submit) 시 실행

document.getElementById('form').addEventListener('submit', async (e) => {

  e.preventDefault()

  const name = e.target.username.value

  if (!name) {

    return alert('이름을 입력하세요')

  }

  try {

    await axios.post('/user', { name })

    getUser()

  } catch (err) {

    console.error(err)

  }

  e.target.username.value = ''

})

# **about.html – 프론트.** 소개 페이지. 페이지 하이퍼링크와 안내 메시지 출력

# **about.html CODE**

<!DOCTYPE html>

<html>

  <!-- head part -->

  <head>

    <!-- incodeing -->

    <meta charset="utf-8" />

    <!-- html page title -->

    <title>RESTful SERVER</title>

    <!-- reference css -->

    <link rel="stylesheet" href="./restFront.css" />

  </head>

  <!-- body part -->

  <body>

    <nav>

      <!-- href to about.html itself / restFront.html -->

      <a href="/">Home</a>

      <a href="/about">About</a>

    </nav>

    <div>

      <!-- information of client. register new user -->

      <h2>소개 페이지입니다.</h2>

      <p>사용자 이름을 등록하세요</p>

    </div>

  </body>

</html>

# **restServer.js – 백**. 요청의 메소드 및 리퀘스트파라메터에 따라 동작 분화 및 응답 전송. POST - 유저 정보 body에 적재. 종료시 적재된 유저 데이터(JSON) 파싱 후 값으로 저장. PUT – reqParam 유저일 경우 스플릿 통해 ID값 추출. 전달된 값 body에 적재 후 파싱 / 추출한 ID의 유저 데이터 수정 DELETE - reqParam 유저일 경우 스플릿 통해 ID값 추출. 추출한 ID의 유저 데이터 삭제 이외의 메소드는 404.

# **restServer.js CODE**

const http = require('http')

const fs = require('fs').promises

// users의 내용을 저장하기 위한 변수 users

const users = {}

http

  .createServer(async (req, res) => {

    try {

      // 어떤 method로 실행되고 어디 url에서 왔는지 log 찍기

      console.log(req.method, req.url)

      // GET이면

      if (req.method === 'GET') {

        // / 주소일 때

        if (req.url === '/') {

          // restFront.html 읽어서 data에 넣는다.

          const data = await fs.readFile('./restFront.html')

          // 헤더 정보 내보내기

          res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' })

          // res data 넣고 출력, 컨텐츠 출력 완료(응답 종료)

          return res.end(data)

        }

        // /about 주소일 때

        else if (req.url === '/about') {

          const data = await fs.readFile('./about.html')

          res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' })

          return res.end(data)

        }

        // /users 주소일 때

        else if (req.url === '/users') {

          res.writeHead(200, { 'Content-Type': 'text/plain; charset=utf-8' })

          // res에 users 데이터 json 형식으로 넣고 출력, 컨텐츠 출력 완료(응답 종료)

          return res.end(JSON.stringify(users))

        }

        // 주소가 /도 /about도 /users도 아니면

        try {

          // req의 url을 data에 넣고 return

          const data = await fs.readFile(`.${req.url}`)

          return res.end(data)

        } catch (err) {

          //주소에 해당하는 라우트를 못 찾았다는 404 Not Found error 발생

        }

      }

      // POST이면

      else if (req.method === 'POST') {

        // /users 주소일 때

        if (req.url === '/user') {

          let body = ''

          // 요청의 body를 stream 형식으로 받음

          req.on('data', (data) => {

            body += data

          })

          // 요청의 body를 다 받은 후 실행됨

          return req.on('end', () => {

            // 요청 받은 body log 출력

            console.log('POST 본문(Body):', body)

            // body를 name 변수에 저장

            const { name } = JSON.parse(body)

            // id는 지금 시간

            const id = Date.now()

            // users의 id 번째에 name 저장

            users[id] = name

            res.writeHead(201)

            res.end('등록 성공')

          })

        }

      }

      // PUT이면

      else if (req.method === 'PUT') {

        // url이 /user/로 시작하면

        if (req.url.startsWith('/user/')) {

          // url을 '/' 기준으로 끊고 2번째 있는 값 key에 저장, 즉 user id 값을 key에 저장

          const key = req.url.split('/')[2]

          let body = ''

          // 요청의 body를 stream 형식으로 받음

          req.on('data', (data) => {

            body += data

          })

          // users[key]에 body의 name을 넣는다.

          return req.on('end', () => {

            console.log('PUT 본문(Body):', body)

            users[key] = JSON.parse(body).name

            // return users

            return res.end(JSON.stringify(users))

          })

        }

      }

      // DELETE이면

      else if (req.method === 'DELETE') {

        // url이 /user/로 시작하면

        if (req.url.startsWith('/user/')) {

          const key = req.url.split('/')[2]

          // user[key] 삭제

          delete users[key]

          return res.end(JSON.stringify(users))

        }

      }

      // get, push, put, delete 이중 아무것도 안걸리면

      res.writeHead(404)

      return res.end('NOT FOUND')

    } catch (err) {

      console.error(err)

      res.writeHead(500, { 'Content-Type': 'text/html; charset=utf-8' })

      res.end(err.message)

    }

  })

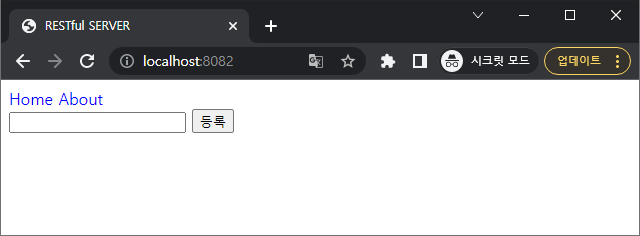
  .listen(8082, () => {

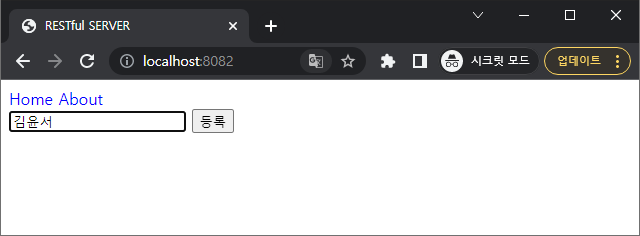
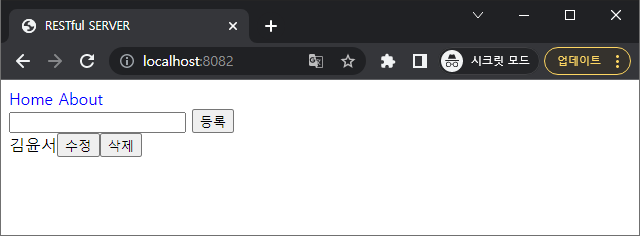
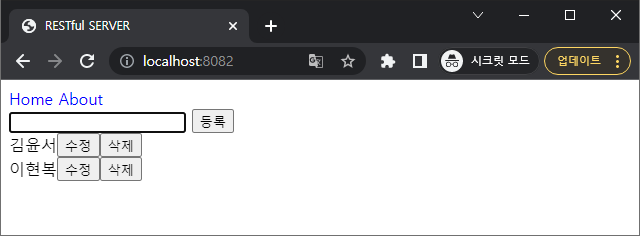
    // 8082 포트 시작시

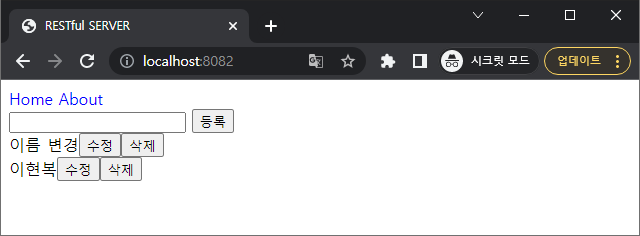
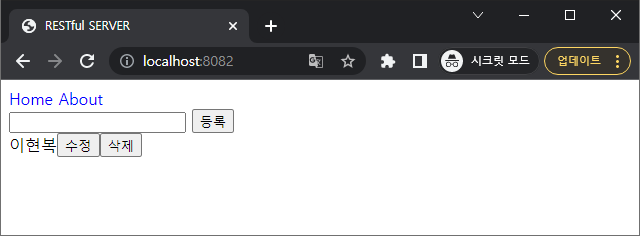
    console.log('8082번 포트에서 서버 대기 중입니다.')

  })

# **실행 결과**

 텍스트이(가) 표시된 사진

자동 생성된 설명    텍스트, 스크린샷, 모니터, 검은색이(가) 표시된 사진

자동 생성된 설명   텍스트이(가) 표시된 사진

자동 생성된 설명