# โครงงานการพัฒนา re-rank และ interface
# 01204453 Web Information Retrieval and Mining

**กลุ่ม :** ม้าโพนี่มหัศจรรย์

**รายชื่อสมาชิก :**

| | | |
|---|---|---|
| นางสาวศลิษา | อติวัฒนชัย | 5610500061 |
| นางสาวชุติกาญจน์ | น้อยกาญจนะ | 5610500222 |
| นายชุติพนธ์ | คงสมพรต | 5610501865 |
| นายคณาธิป | จิตตวิสุทธิวงศ์ | 5610503833 |
| นางสาวพิมวนัช | โกศิยะกุล | 5610503922 |


**GitHub:** https://github.com/chutiphon-k/webir-search

## Introduction

โปรเจคนี้เป็นโปรเจคเกี่ยวกับการสร้าง Web Search Engine โดยจะเริ่มตั้งแต่การสร้าง Web Crawler เพื่อไปเก็บข้อมูลของแต่ละเว็บมา การสร้าง Index File การทำ Ranking และการสร้าง Web Interface ซึ่งโปรเจคนี้จะถูกแบ่งออกเป็น 2 ส่วน ได้แก่ API Server และ Web Application โดยในส่วนของการเก็บข้อมูลด้วย Web Crawler การสร้าง Index File และการทำ Ranking นั้นจะอยู่ในส่วนของ API Server และในส่วนของ Web Application นั้นจะทำการส่ง Request เพื่อร้องขอข้อมูลต่างๆที่ผู้ใช้งานต้องการค้นหาไปยัง API Server ซึ่ง API Server จะนำข้อความนั้นไปค้นหาที่ Index File และ Response ผลลัพธ์ที่ได้กลับไปยัง Web Application เพื่อแสดงผลต่อไป

Package ที่ใช้

- API Server
    - cheerio : ใช้สำหรับตัด tag html
    - elasticlunr : ใช้สำหรับสร้าง Index File และ Search ข้อมูล
    - expressjs : ใช้สำหรับเปิด API Server
    - robotto : ใช้สำหรับนำลิ้งค์ไปตรวจสอบกับ robots.txt
    - pagerank.js : ใช้สำหรับคำนวณ pagerank
- Web Application
    - reactjs : ใช้สำหรับพัฒนาส่วนของ frontend
    - redux : ใช้สำหรับจัดการ state ใน reactjs
    - react-bootstrap : ใช้สำหรับตกแต่งหน้าเว็บ
    - expressjs : ใช้สำหรับเปิด Web Server
    - webpack : ใช้สำหรับสร้างไฟล์ bundle
    - babel : ใช้สำหรับแปลง ES6 เป็น ES5

## รายละเอียดการทำงาน

### 1. Web Crawler

- หลักการทำงาน

เริ่มต้น Web Crawler จะดึงลิ้งค์เริ่มต้นจาก queue ออกมา แล้วตรวจสอบว่าลิ้งค์นั้นมี domain ตรงกับ domain ของลิ้งค์ก่อนหน้าที่มีการเก็บข้อมูลไปหรือไม่ หลังจากนั้นก็จะนำลิ้งค์ดังกล่าวไปตรวจสอบกับ Robotto เพื่อนำลิ้งค์ดังกล่าวไปตรวจสอบข้อกำหนดในการเก็บข้อมูลกับ robots.txt หลังจากนั้นจึงนำลิ้งค์ดังกล่าวไปเข้ากระบวนการเก็บข้อมูล ซึ่งจะมีการตรวจสอบอีกว่าเว็บนั้นสามารถยิง Request เพื่อไปขอข้อมูลได้หรือไม่ Web Crawler นั้นจะหยุดการทำงานเมื่อเก็บข้อมูลได้ครบ 1000 pages แล้วหรือใน queue ไม่มีลิ้งค์เหลือแล้ว หลังจากหยุดการทำงาน ใน folder contents ก็จะมีหน้าเว็บทั้งหมด 1000 pages และข้อมูลต่างๆที่ได้จากการเก็บข้อมูล เช่น เส้นทางของเว็บต้นทางและปลายทาง, ชื่อหลักสูตรของคณะวิศวกรรมศาสตร์, ลิ้งค์ที่ไม่สามารถเข้าไปเก็บข้อมูลได้ และลิ้งค์ที่สามารถเข้าไปเก็บข้อมูลได้ เป็นต้น

- เก็บข้อมูลจากเว็บแบบเช็ค Domain

ในช่วงที่จะเริ่มการเก็บข้อมูลของลิ้งค์ถัดไป Web Crawler จะทำการนำ Domain ของลิ้งค์ก่อนหน้าซึ่งเก็บอยู่ที่ตัวแปร currentHost ไปเช็คใน queue เพื่อหาลิ้งค์ที่อยู่คนละ Domain แล้วนำตำแหน่งของลิ้งค์นั้นไปเก็บที่ position แล้วใช้คำสั่ง splice เพื่อทำการดึงลิ้งค์นั้นออกจาก queue ไปเก็บในตัวแปร url และทำการตั้งค่าให้เป็น currentHost ดังรูป

```
let setCurrentHost = (url) => {
    let urlParse = urlmodule.parse(url)
    if(urlParse.host){
        currentHost = urlParse.protocol + '//' + urlParse.host
    }
}
```

```
new Promise((resolve, reject) => {
    let position = q_urls.findIndex(url => url.indexOf(currentHost) == -1)
    let url = (q_urls.splice(position, 1))[0]
    if(url != undefined){
        setCurrentHost(url)
        resolve(url)
    }
})
```

- ตรวจสอบข้อกำหนดในการเก็บข้อมูลจาก robots.txt

หลังจาก Web Crawler ได้รับลิ้งค์มาแล้วก็จะทำการส่งต่อไปให้ robotto ซึ่งเป็น package สำหรับนำลิ้งค์ไปและ user agent ไปตรวจสอบกับ robots.txt ว่าเจ้าของเว็บไซน์นั้นๆอนุญาติให้เข้าไปเก็บข้อมูลจาก path ดังกล่าวหรือไม่ หลังจากตรวจสอบเสร็จก็จะเก็บค่าสถานะเอาไว้ที่ canCrawl ว่าลิ้งค์นี้สามารถเข้าไปเก็บข้อมูลได้หรือไม่ ดังรูป

```
robotto.canCrawl(userAgent, url, function(err, isAllowed) {
    if (err || isAllowed) {
        resolve({
            canCrawl: true,
            url
        })

    } else {
        resolve({
            canCrawl: false,
            url
        })
    }
})
```

- เก็บเส้นทางของเว็บต้นทางและปลายทาง

หลังจากทำการ Request ไปยังลิ้งค์ที่ผ่านการตรวจสอบต่างๆแล้ว หาก Request สำเร็จ Web Crawler ก็จะนำลิ้งค์ดังกล่าวไปเข้าฟังก์ชั่น getUrl() ซึ่งที่ฟังก์ชั่น getUrl() นั้นจะใช้ cheerio ซึ่งเป็น package สำหรับตัด tag ต่างๆของ html เพื่อดึงลิ้งค์ต่างๆที่อยู่ใน domain .ku.ac.th จากเนื้อหาของเว็บ ซึ่งค่าที่ได้จะถูกเก็บอยู่ที่ตัวแปร urls แล้วนำลิ้งค์เหล่านั้นไปเก็บที่ตัวแปร src_dest ซึ่งเป็น object อีกทีหนึ่ง โดนจะกำหนดให้ key เป็นลิ้งค์ต้นทางของลิ้งค์เหล่านั้น หลังจาก Web Crawler ทำงานเสร็จก็จะนำค่าที่เก็บอยู่ใน src_dest เขียนลงบนไฟล์ src_dest.json ซึ่งเก็บอยู่ที่โฟเดอร์ outputs ดังรูป

```
let getUrl = (data) => {
    let $ = cheerio.load(data)
    let urls = $('a').map(function(i, el) {
    let url = $(this).attr('href')
    if(url != undefined){
        if(url.indexOf('.ku.ac.th') != -1){
            return ((url.endsWith('/')) ? url.slice(0,-1) : url).trim()
            }
    }}).get()
    return urls
}


let urls = getUrl(res.body)
Object.assign(src_dest, {[url]: urls})

fs.writeFileSync(path.join(__dirname, 'outputs', 'src_dest.json'), JSON.stringify(src_dest, null, 2), 'utf8')
```

```
{
  "https://mike.cpe.ku.ac.th/seed/": [
    "http://www.eng.ku.ac.th"
  ],
  "http://www.eng.ku.ac.th": [
    "http://www.eng.ku.ac.th",
    "http://www.eng.ku.ac.th/?page_id=9688",
    "http://www.eng.ku.ac.th/?page_id=9906",
    "http://www.eng.ku.ac.th/?page_id=9690",
    "http://www.eng.ku.ac.th/?page_id=9726",
    "http://www.eng.ku.ac.th/?page_id=9758",
    "http://www.eng.ku.ac.th/?page_id=9762",
    "http://www.eng.ku.ac.th/?page_id=9937",
    "http://www.eng.ku.ac.th/?page_id=9939",
    "http://www.eng.ku.ac.th/?page_id=9694",
    "http://www.eng.ku.ac.th/?page_id=9696"
    ]
}
```


**ตัวอย่างการเก็บเส้นทางของเว็บต้นทางและปลายทาง**

- เก็บข้อมูลของเว็บที่ดึงข้อมูลได้สำเร็จและไม่สำเร็จ

Web Crawler จะทำการ Request แบบ GET เพื่อข้อดึงข้อมูลเว็บไซน์จากลิ้งค์ที่ได้ทำการตรวจสอบแล้ว โดยก่อนจะทำการเก็บข้อมูลนั้นจะตรวจสอบก่อนว่า Http Status Code ที่ได้รับกลับมาจากการ Request ซึ่งหากมีค่าน้อยกว่า 400 แสดงว่าสามารถเข้าไปเก็บข้อมูลได้ และทำการเก็บข้อมูลจากเว็บไซน์นั้นๆ และเก็บลิ้งค์ดังกล่าวลงในตัวแปร success_urls อีกด้วย แต่ถ้าหากมากกว่าหรือเท่ากับ 400 แสดงว่าที่ client หรือ server มีปัญหา หลังจากนั้นก็จะทำการเก็บลิ้งค์ดังกล่าวลงในตัวแปร error_urls ซึ่งหลังจาก Web Crawler จบการทำงานก็จะนำค่าในตัวแปร success_urls และ error_urls เขียนลงในไฟล์ชื่อ success.json และ error.json ตามลำดับ ซึ่งเก็บอยู่ที่โฟเดอร์ outputs ดังรูป

```
try {
    let res = request('GET',url,requestOption)
    if(res.statusCode < 400){
        currentUrl = url
        ++countSuccess
        success_urls.push(url) // <----------
        saveContent(url, res.getBody('utf8'))
        if(url == 'http://www.eng.ku.ac.th/?page_id=9690'){
            getCourse(res.getBody('utf8'))
        }
        let urls = getUrl(res.body)
        Object.assign(src_dest, {[url]: urls})
        pushUrl(urls)
    } else{
        error_urls.push({
            url,
            status_message: HTTPStatus[res.statusCode]
        }) // <----------
        console.log('status_message : ', HTTPStatus[res.statusCode])
    }
} catch(err) {
    error_urls.push({
        url,
        status_message: err.message
    }) // <----------
    console.log('status_message : ', err.message)
}


fs.writeFileSync(path.join(__dirname, 'outputs', 'error.json'), JSON.stringify(error_urls, null, 2), 'utf8') // <----------
fs.writeFileSync(path.join(__dirname, 'outputs', 'success.json'), JSON.stringify(success_urls, null, 2), 'utf8') // <----------
```

```
[
    "https://mike.cpe.ku.ac.th/seed/",
    "http://www.eng.ku.ac.th",
    "http://admission.eng.ku.ac.th",
    "http://iup.eng.ku.ac.th/index.php/en",
    "http://www.eng.ku.ac.th/?page_id=9688",
    "http://www.sa.ku.ac.th",
    "http://www.eng.ku.ac.th/?page_id=9906",
    "http://www.go.sa.ku.ac.th",
    "http://www.eng.ku.ac.th/?page_id=9690"
]
```


**ตัวอย่างเก็บลิ้งค์ที่เก็บข้อมูลสำเร็จ**


```
[
    {
        "url": "http://sa.eng.ku.ac.th/index.php",
        "status_message": "getaddrinfo ENOTFOUND sa.eng.ku.ac.th sa.eng.ku.ac.th:80"
    },
    {
        "url": "https://fac-meeting.ku.ac.th",
        "status_message": "connect ECONNREFUSED 158.108.219.102:443"
    },
    {
        "url": "http://mmec.eng.ku.ac.th",
        "status_message": "getaddrinfo ENOTFOUND mmec.eng.ku.ac.th mmec.eng.ku.ac.th:80"
    }
]
```


**ตัวอย่างเก็บลิ้งค์ที่เก็บข้อมูลไม่สำเร็จ**

- เก็บข้อมูลชื่อหลักสูตรของคณะวิศวกรรมศาสตร์

ในการเก็บชื่อหลักสูตรของคณะวิศวกรรมศาสตร์นั้น Web Crawler จะตรวจสอบว่าในเนื้อหาข้อเว็บนั้นมีคำว่า “หลักสูตร” อยู่หรือไม่ซึ่งถ้าหากมีจะเริ่มเก็บข้อมูลหลักสูตรโดยจะตรวจสอบว่ามีคำว่า “สาขา” อยู่ที่บรรทัดไหนแล้วจะทำการดึงข้อความใน tag ของบรรทัดนั้นออกมาแล้วเก็บใส่ตัวแปร courses ซึ่งข้อมูลในตัวแปร courses จะถูกเขียนลงไฟล์ courses.json หลังจาก Web Crawler จบการทำงาน

```
let getCourse = (data) => {
    if(data.indexOf('หลักสูตร')){
        let regx = /[A-Za-z0-9_./]/gi
        let first = data.indexOf('สาขา')
        while (first != -1) {
          let last = data.indexOf('<', first)
          let course = data.substring(first, last)
          if(!courses.includes(course) && !regx.test(course)){
            courses.push(course)
          }
          first = data.indexOf('สาขา', last)
        }
    }
}
```


ตัวอย่างชื่อหลักสูตรในคณะวิศวกรรมศาสตร์

## 2. Index File และ Search

- หลักการทำงาน

ในส่วนของ Index File และ Search จะใช้ package elasticlunr ในการทำงานโดยในการสร้าง Index File นั้นจะมีการทำ stopwordfilter และ stemmer โดยต้องทำการ add field ที่ต้องการจะใช้ในการค้นหาก่อน ซึ่งได้กำหนดให้เป็น title และ content หลังจากนั้นก็จะทำการดึงข้อมูลจากโฟเดอร์ contents ที่ได้จากการเก็บข้อมูลของ Web Crawler เพื่อนำไปทำ Index File ซึ่งผลลัพธ์ที่ได้คือไฟล์ indexFile.json ซึ่งถูกเก็บอยู่ในโฟเดอร์ outputs และในส่วนของการ Search นั้นจะทำการดึงข้อมูลจากไฟล์ indexFile.json แล้วโหลดข้อมูลจากไฟล์ดังกล่าวเข้าไปใน elasticlunr ด้วยคำสั่ง elasticlunr.Index.load(indexFile) ซึ่งจะถูกเก็บอยู่ที่ตัวแปร idx และนำ Index File ดังกล่าวไปใช้ในการ search ด้วยคำสั่ง idx.search(query) ซึ่งผลลัพธ์ที่จะได้อยู่ในรูปของ object หลังจากนั้นจึงนำเอา content ที่อยู่ใน object ดังกล่าวไปสร้างเป็น snippet โดยจะทำการค้นหาตำแหน่งของคำที่ค้นหาก่อนแล้วนำตำแหน่งนั้นไปลบ 50 และบวก 50 ซึ่งจะได้ผลลัพธ์ออกมาเป็นช่วงของ snippet

```
var idx = elasticlunr(function () {
  this.addField('title')
  this.addField('content')
  this.setRef('id')
});

idx.pipeline.remove(elasticlunr.trimmer)

try {
  let files = fs.readdirSync(path.join(__dirname, 'contents'))
  files.map((file, index) => {
      let data = fs.readFileSync(path.join(__dirname, 'contents', file), 'utf8')
      let getUrl = JSON.parse(fs.readFileSync(path.join(__dirname, 'outputs', 'success.json'), 'utf8'))
      try {
        let $ = cheerio.load(data, { normalizeWhitespace: true, ignoreWhitespace: true })
        $('script').remove()
        $('style').remove()
        let url = getUrl[file-1]
        console.log('[', index+1, '] : ', url)
        if(url != undefined){
          idx.addDoc({
              id: index+1,
              url,
              title: $('title').text().trim(),
              content: $('body').text().trim()
          })
        }
      } catch(err) {}
  })
} catch(err) {
  console.log('Read File Contents Or Success Error!!!')
}
```

```
exports.getSearch = (query) => {
  console.log('>>> Start Search <<<')
  let ans = idx.search(query,  {
      fields: {
          title: {boost: 1},
          content: {boost: 1}
      },
      bool: "OR",
      expand: true
  })

  let ansMaps = ans.map((value) => {
    let ansMap = idx.documentStore.getDoc(value.ref)
    let splitQuery = query.split(/(\s+)/).filter((str) => str.trim().length > 0 )
    let missing = splitQuery.filter((value) =>
      ansMap.content.toLowerCase().indexOf(value.toLowerCase()) == -1 && ansMap.title.toLowerCase().indexOf(value.toLowerCase()) == -1)
    let snippet = getSnippet(query.toLowerCase(), ansMap.content)
    if(snippet == -1){
      let snippetSplit = splitQuery.map((value) => {
        return getSnippet(value.toLowerCase(), ansMap.content)
      })

      snippet = snippetSplit.find((snippet) => snippet != -1)
      if(snippet == undefined){
        snippet = ansMap.content.substring(0, 50)
      }
    }
    Object.assign(ansMap, {snippet, missing, similarity_score: value.score})
    return ansMap
  })

  console.log('>>> Search Done <<<')

  return ansMaps.map((ansMap) => {
    let { id, url, title, snippet, missing, similarity_score } = ansMap
    return {
        id,
        url,
        title,
        snippet,
        missing,
        similarity_score
    }
  })
}
```

### 3. Ranking

- หลักการทำงาน

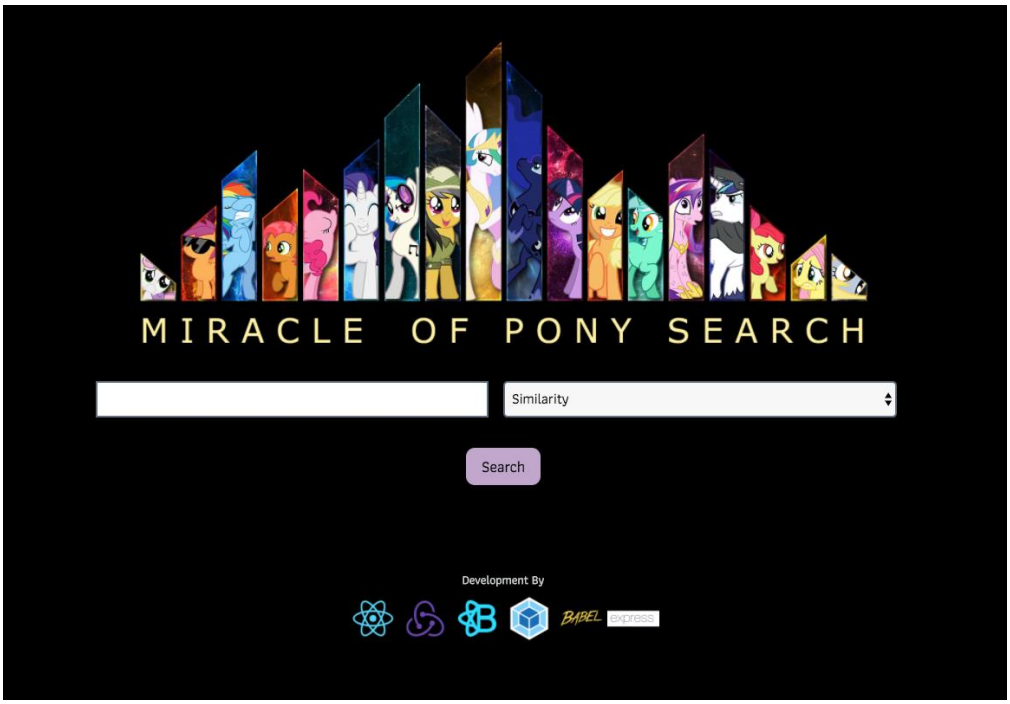
ในการ Ranking นั้นจะแบ่งออกเป็น 3 แบบได้แก่ Similarity, PageRank และ ReRank โดยตัว Similarity หลังจากที่ผ่านการ search ผ่าน elasticlunr ผลลัพธ์ที่ได้จะถูก Ranking แบบ Similarity มาอยู่แล้ว โดยจะมีค่า similarity_score ติดออกมากับผลลัพธ์จากการ search ด้วย ในส่วนของ PageRank นั้นจะใช้ package pagerank.js ในการคำนวณค่า PageRank Score โดยจะต้องกำหนดค่า DAMPING และ EPSILON ซึ่งได้กำหนดให้เป็น 0.85 และ 0.000001 ตามลำดับ และใส่ต้นทางและปลายทางของเว็บต่างๆ ให้แก่ pagerank.js ซึ่งจะได้ค่า PageRank Score ออกมาเก็บอยู่ใน pagerank_score สุดท้ายในส่วนของ ReRank ก็จะนำเอาค่า similarity_score และ pagerank_score มาถ่วงน้ำหนักด้วยค่า alpha ซึ่งกำหนดค่าตั้งต้นไว้เป็น 0.5 แต่สามารถเปลี่ยนได้โดยการส่ง query string มากับ url โดยในส่วนนี้จะถูกทำที่ API Server ด้วย package expressjs ทั้งหมดซึ่งผู้ใช้งานสามารถกำหนดค่า filter, page, limit, alpha ได้จากการยิง Request แล้วส่ง query string มากับ Request นั้นด้วย

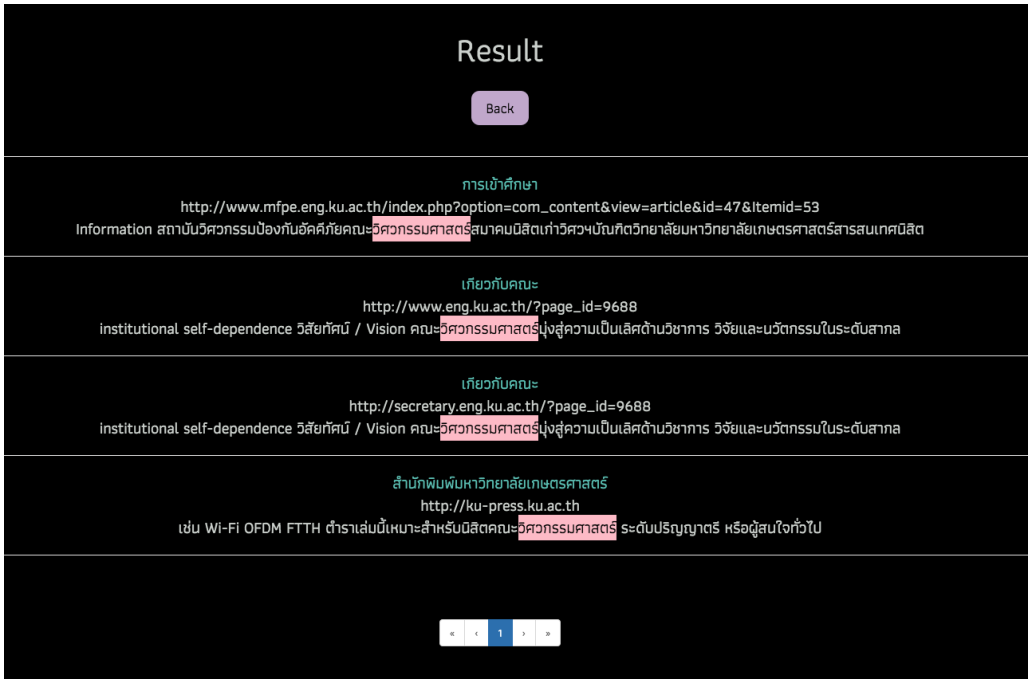
```
app.get('/api/search', (req, res) => {
    console.log(req.query.search)
    let { filter, page, limit, alpha } = req.query
    let data = search.getSearch(req.query.search)
    page = +page || 1
    limit = +limit || 10
    alpha = +alpha || 0.5
    switch(filter){
        case 'pagerank':
            data = ranking.getPageRank(data)
        case 'rerank':
            data = ranking.getReRank(data, alpha)
    }

    let pageCount = Math.ceil(data.length/limit)
    let pagination = {
        pageStart: 1,
        pagePrevious: ((page-1) <= 1) ? 1:page-1,
        pageCorrent: page,
        pageNext: ((page+1) >= pageCount) ? pageCount: page+1,
        pageCount,
        itemCount: data.length,
    }
    data = data.splice((page-1)*limit, limit)
    res.json({pagination, data})
})
```

## 4. Web Interface

- หลักการทำงาน

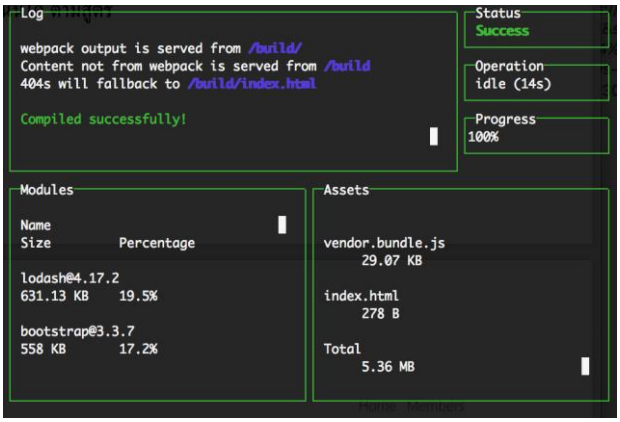
เว็บถูกพัฒนาเป็นแบบ Single Page Application ซึ่งจะถูกแบ่งออกเป็น 2 หน้าได้แก่หน้าแรกที่ path : / และหน้าที่สองที่ path: result เมื่อทำการพิมพ์ข้อความและเลือกรูปแบบการ Ranking แล้วกด search แล้วเว็บจะยิง Request ไปที่ API Server แล้ว redirect ไปที่หน้า result แล้วนำค่าที่ได้รับจาก API Server มาแสดงผลโดยตัวเว็บจะใช้ expressjs เป็น Web Server และใช้ ReactJS เป็น lib สำหรับจัดการหน้าเว็บ



- Tool ต่างๆที่ใช้
    - Webpack

ใช้สำหรับการ bundle ไฟล์ต่างๆที่ใช้ในการพัฒนา Web Application อาทิเช่น bootstrap, react, redux เป็นต้น ซึ่งโดยปกติหากใช้ lib หรือ framework ต่างๆจะต้องใส่ script ไว้ที่หน้า html ซึ่งทำให้มี script เป็นจำนวนมาก และ lib หรือ framework มีขนาดที่ใหญ่เกินไป Webpack จะทำหน้าที่ย่อไฟล์ต่างๆเป็นไฟล์ bundle เพียงไฟล์เดียวและทำให้มีขนาดเล็กลง โดยจะแบ่งออกเป็นแบบ Development และ Production ซึ่งแบบ Development นั้นจะเน้นที่การคอมไพล์ได้รวดเร็วและสะดวกต่อการพัฒนาเว็บ โดยจะมีการใช้ webpack-hot-loader เพื่อทำการ autorefresh เมื่อมีการเปลี่ยนแปลง code ในส่วนของ Production จะเน้นให้ไฟล์มีขนาดเล็กเพื่อให้ browser สามารถโหลดเว็บได้รวดเร็วยิ่งขึ้น ซึ่งขนาดไฟล์ bundle ที่ได้จาก Webpack แบบ Development กับ Production จะมีขนาดต่างกันประมาณ 50 เปอร์เซ็น




**คอมไพล์ไฟล์ bundle แบบ Development**


**คอมไพล์ไฟล์ bundle แบบ Production**