**2020-2**

**웹 시스템 설계**

**학습과제**

**Week 9**

**JavaScript Async Programming & Express.js Framework**

**Composed by:**

**WISE Research Lab Ajou University** 

**학습목표**

1. JavaScript의 Async-programming 방법에 대해 학습한다.
2. Node.js의 Express framework를 활용한 백엔드 개발에 대해 학습한다.

|  |
| --- |
| **JavaScript Async Programming** |

## Promise - <https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Promise>

* 여러 개의 Callback함수를 중첩해서 사용할 경우 코드의 depth가 증가해서 가독성을 해치는 경우(Callback Hell)를 방지하기 위한 패턴
* 모든 Callback을 1 depth로 처리하여 복잡한 코드를 개선할 수 있다. (Callback-Hell 극복)
* Promise 는 resolve, reject의 호출로 실행 결과를 반환하며, 실행의 성공과 실패에 따라 .then() 또는 .catch() 함수가 호출된다.
* .then() 함수에서 다시 Promise를 수행하여 여러 개의 비동기 함수를 순차적으로 호출하여 사용할 수 있다.

function doSomething(data, callback) {

callback(data\*2);

}

doSomething(1, (data) => {

doSomething(data, (data2) => {

doSomething(data2, (data3) => {

console.log(`This is callback hell::${data3}`)

*// => "This is callback hell::8"*

})

})

})

위의 Callback-Hell 코드를 Promise 기반으로 작성하면 아래와 같이 바뀐다

function doSomething(data) {

*return* Promise.resolve(data\*2)

}

doSomething(1).then((data) => {

*return* doSomething(data)

}).then((data2) => {

*return* doSomething(data2)

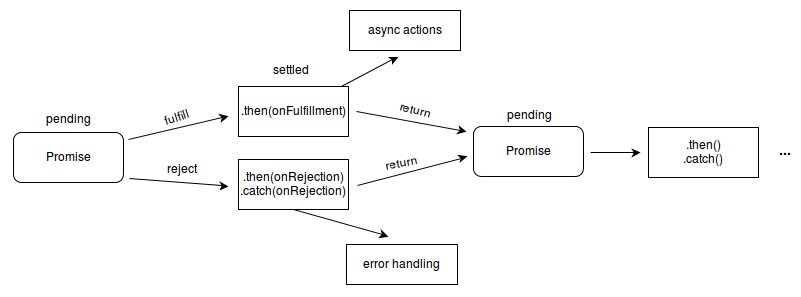
}).then((data3) => {

console.log(`Problem solved::${data3}`)

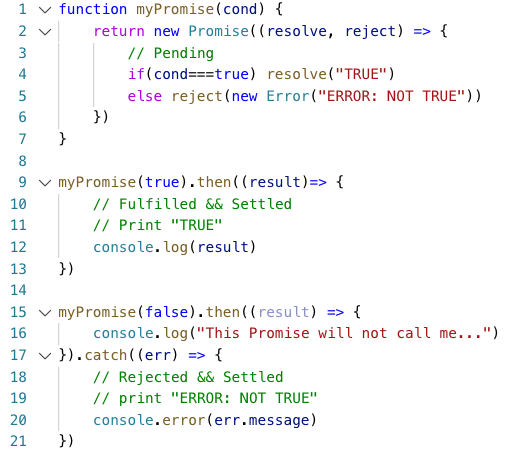
*// => "Problem solved::8"*

})

* Promise는 다음과 같은 4가지의 상태를 가진다.

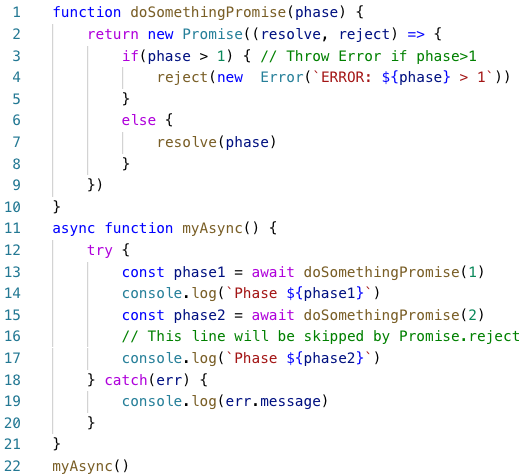


* + Pending
    - Fulfilled 또는 Reject 가 되기 전의 상태.
    - Promise가 아직 실행중일 경우를 의미함
  + Fulfilled
    - Promise가 성공적으로 실행된 상태
  + Rejected
    - 실행에 실패한 상태(에러 반환)
  + Settled
    - Resolve / Reject에 관계 없이 실행이 종료된 상태



## 2. Async / Await

* Promise와 마찬가지로 non-blocking이며, Promise 를 사용하면서도 전통적인 동기식 코드와 유사한 코드 구성이 가능하다.
* await 키워드를 사용하면 Promise.resolve에서 반환한 데이터에 바로 접근이 가능하다.
  + async 키워드로 선언된 function 안에서만 await 키워드의 사용이 가능함
* Async/Await를 사용하면 동기와 비동기 에러를 모두 try-catch로 핸들링이 가능하다.
  + try-catch에서 Promise.reject를 처리할 수 있음



[Promise 기반의 비동기 함수를 try-catch로 예외처리를 하는 예제 코드]

* await 키워드를 이용한 함수 호출은 Promsie를 반환하는 함수만 가능하다.
* Callback 기반으로 구현된 비동기 함수를 Promise 기반으로 사용하는 방법
  + 해당 코드를 Promise 기반의 함수로 리팩토링할 수 없는 상황이라면, 2가지 방법을 통해 함수를 변환하여 사용할 수 있다.

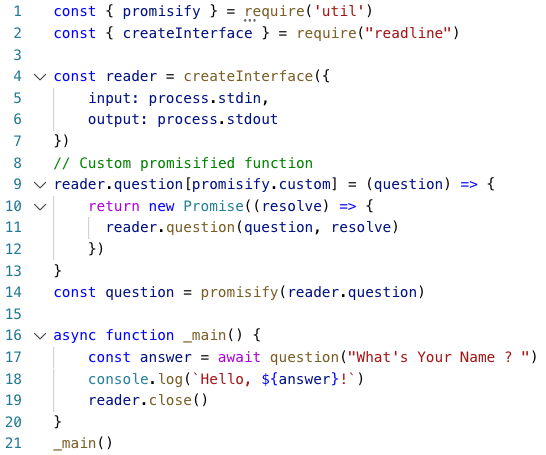


* Callback 코드를 래핑하는 래퍼(wrapper) 함수 구현



* Util 모듈의 promisify 함수를 이용하여 함수 변환

[API 문서 <https://nodejs.org/api/util.html#util_util_promisify_original>]



promisify.custom을 이용하면 promisify 함수를 오버라이드할 수 있다.

(즉, promisify 호출 시 반환되는 함수를 직접 정의할 수 있다)

|  |
| --- |
| **Express** |

1. **Express.js**

* Express.js는 웹과 모바일 애플리케이션을 위한 기능을 제공하는 Node.js 웹 애플리케이션 프레임워크이다.
* Express는 아래와 같은 문제를 해결해주는 동시에 웹 애플리케이션에 MVC 형태의 구조를 제공한다.
  + HTTP 요청 본문 파싱
  + 쿠키 파싱
  + 세션 관리
  + URL 경로와 HTTP 요청 method를 기반으로 한 복잡한 if 조건을 통해 routing구성
  + 데이터 타입을 토대로 한 적절한 응답 헤더 결정

1. **Express 프로젝트 구성**

* Express 모듈은 node package manager(npm) 를 통하여 설치할 수 있다.
* Express를 사용하는 구조를 가진 프로젝트를 다양한 방법으로 생성할 수 있다.
  + 빈 프로젝트에 express를 설치하고 관련된 부분을 직접 설정
  + express-generator를 사용하여 express 관련 설정이 적용된 프로젝트를 생성
* npm 빈 프로젝트에 express를 추가하는 방법

|  |
| --- |
| $ mkdir [폴더 이름]  $ cd [폴더 이름]  $ npm init  ㅈ |

* + package.json 파일에 Express가 추가된다.
* CLI 명령을 사용하여 프로젝트를 생성하는 방법

|  |
| --- |
| $ npm install express-generator -g  $ express [옵션 지정; --no-view] [프로젝트 이름; myapp]  $ cd myapp  $ npm install  $ npm start |

* + 즉시 실행해볼 수 있는 Express 앱 예제가 생성된다.
  + 자세한 정보는 다음을 참고한다.

<https://expressjs.com/ko/starter/generator.html>

* WebStorm에서 Express 프로젝트를 생성하는 방법
  + WebStorm 메뉴에서 File > New > Project… 선택
  + Node.js Express App를 선택
  + node interpreter와 express 버전, 그리고 View, Stylesheet Engine 을 선택한다.
  + Create를 선택하여 프로젝트를 생성한다.
* Express를 사용하는 방법
  + **expressServer.js**

|  |
| --- |
| *// express 모듈을 불러온다.* **var express** = require(**'express'**); **var** app = **express**();  *//어플리케이션의 루트 페이지* app.get(**'/'**, **function** (req,res){  *//페이지의 body에 Hello World! 출력* res.send(**'Hello World!'**); });  *//3000번 port에 대해서 listen* app.listen(3000, **function**(){  **console**.log(**'Express server is listening'**); }); |
|  |

1. **Routing in Express**

* Routing은 URI 및 특정한 HTTP 요청 method인 특정 endpoint에 대한 클라이언트 요청에 애플리케이션이 응답하는 방법을 결정하는 것을 말한다.
  + 각각의 Route는 하나 이상의 handler 함수를 가질 수 있으며, 이러한 함수는 route가 일치할 때 실행된다.
  + route 정의에는 다음과 같은 구조가 필요하다.

|  |  |
| --- | --- |
| app.METHOD(PATH, HANDLER) | |
| app | express의 인스턴스 | |
| METHOD | HTTP 요청 method | |
| PATH | 서버에서의 경로 | |
| HANDLER | route가 일치할 때 실행되는 함수 | |

* Route method는 HTTP method 중 하나로부터 파생되며, express 클래스의 인스턴스에 연결된다.
  + 애플리케이션의 루트에 대한 GET 및 POST method에 대해 정의된 route의 예시

|  |  |
| --- | --- |
| app.get(**'/'**, **function** (req,res){  res.send(**'Hello World!'**); }); | 애플리케이션의 홈 페이지인 루트 라우트에서 Hello World!로 응답 |
| app.post(**'/'**, **function** (req, res) {  res.send(**'Got a POST request'**); }); | 홈 페이지에서 POST 요청에 응답 |

* Route handler를 이용하면 미들웨어와 비슷하게 작동하는 여러 콜백 함수를 통해 요청을 처리할 수 있다. 요청을 핸들하는 콜백 함수로, 하나 이상의 함수가 연결될 수 있다.
* 아래의 Response 오브젝트에 대한 method들은 응답을 클라이언트에게 전송하고 request-response 주기를 종료할 수 있다. Route handler로부터 다음 method들 중 어느 하나도 호출되지 않은 경우 클라이언트 요청은 정지된 채로 방치된다.

|  |  |
| --- | --- |
| res.download() | 파일이 다운로드되도록 프롬프트 |
| res.end() | 응답 프로세스를 종료 |
| res.json() | JSON 응답을 전송 |
| res.jsonp() | JSONP 지원을 통해 JSON 응답을 전송 |
| res.redirect() | 요청의 경로를 재지정 |
| res.render() | View template를 렌더링 |
| res.send() | 다양한 유형의 응답 전송 |
| res.sendFile | 파일을 octet stream의 형태로 전송 |
| res.sendStatus() | 응답 상태 코드를 설정한 후 해당 코드를 문자열로 표현한 내용을 응답 본문으로서 전송 |

* Express의 routing과 관련한 자세한 사용법 및 예제는 공식 document문서를 참고한다.

<http://expressjs.com/en/starter/basic-routing.html>

<http://expressjs.com/en/guide/routing.html>

* **Pug Template Engine**
* Pug는 angle bracket (<) 으로 html element의 구조를 설계하는 기존의 html문법을 whitespace로 구분하는 것으로 대체하여 코드를 간소화할 수 있다.

|  |  |
| --- | --- |
| **HTML** | **Pug** |
| <!DOCTYPE html>  <html lang="en">  <head>  <title>Hello, World!</title>  </head>  <body>  <h1>Hello, World!</h1>  <div class="phrase">  <p>Pug rocks!!</p>  </div>  </body>  </html> | doctype html  html(lang='en')  head  title Hello, World!  body  h1 Hello, World!  div.phrase  p Pug rocks! |

* Pug 문법으로 작성된 문서는 .pug 확장자로 저장하며 이후 pug node 모듈을 통하여 html 문서로 렌더링이 수행된다.
* Pug는 block과 extends keyword를 사용하여 다른 template을 상속받는 것이 가능하다.

|  |  |
| --- | --- |
| **layout.pug** | **mypage.pug** |
| html  head  title My Site - #{title}  block scripts  script(src='/jquery.js')  body  block content | extends layout.pug  block scripts  script(src='/jquery.js')  script(src='/pets.js')  block content  h1= title  - var pets = ['cat', 'dog']  each petName in pets  include pet.pug |

* + - 위의 예제에서는 layout.pug의 내용을 상속받고 추가 작성한 코드를 지정 부분에 대입하는 것으로 mypage.pug를 구성하고 있다. 상속없이 작성한 mypage.pug의 모습은 다음과 같다.

|  |
| --- |
| **mypage.pug (Inheritance이후)** |
| html  head  title My Site - #{title}  script(src='/jquery.js')  script(src='/pets.js')  body  h1= title  - var pets = ['cat', 'dog']  each petName in pets  include pet.pug |

* + - 자세한 사용법은 pug의 공식 document를 참고하도록 한다.

<https://pugjs.org/api/getting-started.html>

* **Request Body Parsing**
* **<form>의 POST요청**은 <form>이 가지는 정보를 **key과 value의 pair**의 형태로 **req.body**의 parameter에 담아 전송한다.
* Server는 요청의 body를 다음과 같은 방식으로parsing하여 view에서 활용할 수 있다.

|  |
| --- |
| routes/index.js |
| router.post('/movies', function(req, res) {  res.render('views/index.pug', { data: req.body });  }); |
| views/index.pug |
| div  h1 #{data.name}  img(src=`${data.file}`) |

|  |
| --- |
| **학습과제** |

## 과제 개요

* JavaScript Async Programming(Promise 및 Async, Await)과 Express.js 프레임워크를 활용하여 웹을 통해 기입된 영화정보를 저장하는 웹 어플리케이션을 구축한다.

## 학습과제 파일 구성 및 과제 제출 양식

* 프로젝트의 폴더 구성은 express-generator를 통해 구성된 스켈레톤 프로젝트를 기반으로 구성한다.
* 아래의 구성 요소는 스켈레톤 프로젝트를 구성 후 추가로 구성해야하는 파일 요소이다.
  + public/images : 영화의 이미지 파일을 저장하고 있는 위치이다. 테스트를 위해 영화 포스터 파일을 저장하는 위치
    - poster\_1.jpg, poster\_2.jpg, poster\_3.jpg : 포스터의 이미지 파일
    - star.svg : 별점 표시를 위해 저장하는 별 이미지 파일
  + public/stylesheets : 렌더링 되는 웹페이지의 스타일시트를 저장하는 폴더
    - style.css : 렌더링되는 웹페이지의 스타일 시트
  + routes/index.js : “/” 경로에 대한 GET 요청을 처리하는 파일
  + routes/results.js : “/result” 경로에 대한 POST 요청을 처리하는 파일
  + views/result.pug : “/result” 경로에 대한 POST 요청 처리에 대한 결과를 렌더링하기 위한 템플릿
* 이외 express-generator를 통해 생성되는 파일들에 대해 수정이 필요한 경우, 수정을 수행한다.
* 압축 파일의 이름을 “Lab9\_자신의학번.zip”으로 지정하여 폴더(프로젝트)를 압축한다.

## 3. 제출방법 및 주의사항

* 설치된 node package들을 제외한 프로젝트 폴더를 압축해서 제출
  + 즉, node\_module 폴더는 압축파일에 포함하지 않도록 한다.
* 지각 제출 시, 0점으로 처리한다.
* 채점 시 완성도의 평가는 Google의 [Chrome 브라우저](https://www.google.com/intl/ko/chrome/)에서 렌더링 된 화면을 기준으로 함

## 4. 프로그램 설계

**4.1 템플릿 정의**

* 아래의 두 템플릿은 layout.pug을 상속받으며, Index.pug와 result.pug는 layout.pug의 content block을 정의한다.

**4.1.1 index.pug (영화 리뷰 입력 페이지)**

* 프로젝트 내의 index.pug 파일을 수정한다.
* **<Form>** element를 생성한 후, 영화 정보를 입력할 수 있도록 다음의 element들을 하위 element로 추가한다.
  + **type=’text’**인 **<input> 4개**
    - 영화 제목, 출연진, 장르, 영화 포스터(이미지) URL
    - 이미지 파일의 경로는 **로컬 파일의 경로를 입력하도록 한다.**
  + **type=’radio’**인 **<input> 6개**
    - 영화 평점 (0점 - 5점)
    - **6개 중 하나만** 선택할 수 있어야 한다.
  + **type=’submit’**인 **<input>**
    - submit을 클릭하면 영화 추가 요청을 생성하고 서버는 영화 정보 페이지를 render한다.
* 스타일을 지정하는 **style.css** 파일은 자유롭게 구현한다.

**4.1.2 result.pug (영화 리뷰 결과 페이지)**

* 프로젝트 내의 result.pug 파일을 수정한다.
* 영화 리뷰 입력 페이지에서 추가한 영화의 정보를 보여줄 수 있도록 다음의 element들을 추가한다.
  + <img>: 영화 포스터
  + <h1>: 영화제목
  + <p> 또는 <div>: 출연진, 장르
  + <img>: 별 그림
    - 예시와 같이 for문을 사용하여 영화에 주어진 평점과 동일한 개수로 생성한다.

|  |
| --- |
| views/result.pug |
| -for(var idx=0; idx<data.rating; idx++)  img(src='/images/star.svg' class='rating-star-img') |

* + <Button>: 뒤로 가기
    - 클릭하면 영화 리뷰 입력 페이지로 돌아갈 수 있도록 한다.
* 영화 포스터를 보여주는 <img>의 높이는 600px, 너비는 165px로 지정하도록 한다.
* 위에서 지정한 부분 이외의 스타일은 자유롭게 구현한다.

**4.2 서버 요청 처리 정의**

* **4.2.1 ㅈ**

**4.2.2 index.js, result.js**

* 페이지에 방문한 client가 server에 요청할 수 있는 **REST interface**는 다음과 같다.

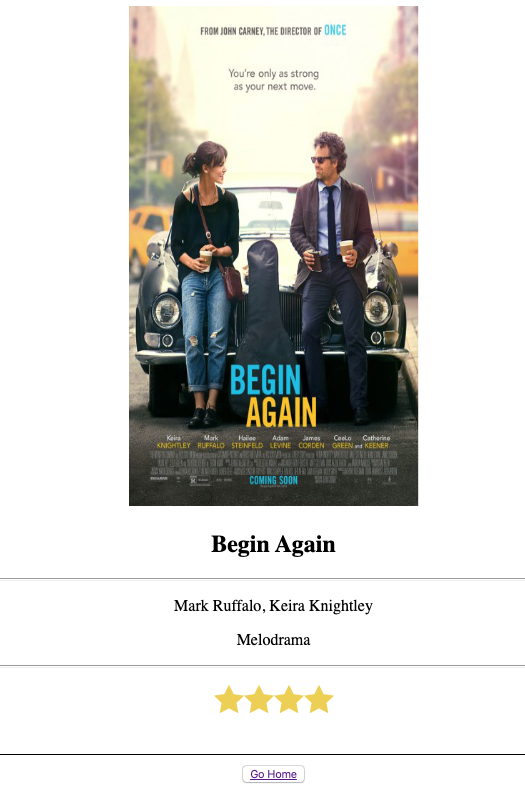
|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP req Method** | **URI** | **하는 일** | **in File** |
| GET | / | 영화 리뷰 입력 페이지 render | routes/index.js |
| POST | /result | 입력한 영화 정보를 페이지에 전달하고 목록 페이지 render | routes/result.js |

* **영화 리뷰 입력 페이지**에서 **submit <input>**을 클릭 시
  + **URI**가 ‘**/result’**, **method type**이 **POST**인 요청을 생성한다.
  + 요청과 함께 form 하위 element에 **입력한 data**를 전달한다.
  + 서버는 **영화 리뷰 결과 페이지를 render** 하며 전달받은 data를 넘겨준다.
    - 서버에서 이미지를 불러올 때, **반드시 Promise** 객체를 사용하거나 Node.js utils 패키지내 **Promistify**를 사용하여 fs모듈의 stat 메소드를 호출한다.
  + 이 과정에서 서버가 넘겨받는 **request data를 parsing**할 수 있다.
  + 단, 입력이 누락되거나 파일을 찾을 수 없는 경우 **HTTP 요청을 할 수 없다.**

**4.3 실행 예시**



[예시 1] 추가할 영화 정보를 입력한 모습



[예시 2] 예시 1에서 Submit을 클릭한 이후의 모습