2020-2

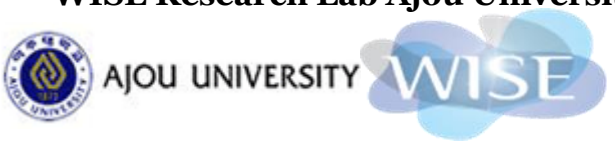
웹 시스템 설계

학습과제

Week 10

Express with Mongoose

Composed by:
WISE Research Lab Ajou University



학습목표

1. Express Framework 와 MongoDB 를 연동하여 웹서버를 구축할 수 있다.

I. MongoDB

1. MongoDB 란? (참고문서: https://docs.mongodb.com/manual/)

- Document 기반의 **No-SQL Database** 이다.
  - ✓ **Document** 는 MongoDB 의 데이터 저장 단위로, 한 개 이상의 Key-Value 쌍으로 이루어진 객체를 BSON 형태로 저장한다.
  - ✓ **SQL expression** 이 존재하지 않으며, 모든 데이터 조작은 mongoshell 또는 ODM(Object Data Mapping) 라이브러리에서 **메소드 호출 방식**으로 동작함
  - ✓ **Schema-free** 한 데이터 저장 방식 덕분에 같은 Collection 에 속한 Document 라도 다른 유형의 데이터 필드를 가질 수 있다.
  - ✓ 모든 Document 는 기본적으로 **_id** 라는 고유한 값을 가진 필드가 존재한다
- **Document**
  - ✓ MongoDB 의 **데이터 저장 단위**로, 한 개 이상의 Key-Value 쌍으로 이루어진 객체를 BSON 형태로 저장한다.
- **Collection**
  - ✓ 데이터를 용도에 따라 분류하기 위한 **Document 의 모음**이며, 모든 Document 는 특정 Collection 에 속한다. (RDBMS 의 Table 과 유사함)
- **Database**
  - ✓ 여러 Collection 들을 담고 있는 **물리적 저장소**이다.
  - ✓ MongoDB 서버는 여러 개의 Database 를 가질 수 있으며, "**System**" 은 서버 운영에 필요한 메타 정보를 저장하기 위해 예약되어 있으므로 사용 불가

II. Mongoose

1. Mongoose 란? (참고문서: https://mongoosejs.com/docs/guide.html)

- Node.js 기반의 MongoDB ODM 라이브러리이다.
- MongoDB Driver 가 기본적으로 제공하는 CRUD 기능을 포함하여 다양한 추가 기능 제공
  - ✓ Schema (https://mongoosejs.com/docs/guide.html)
    - Schema 기반의 Collection 데이터 모델링이 가능하고, 데이터를 DB 에 저장하기 전에 검사 가능
  - ✓ Population (https://mongoosejs.com/docs/populate.html)
    - Document B 를 참조(reference)하고 있는 Document A 를 쿼리할 때, 참조 위치에 실제 Document B 의 데이터가 치환되어 값이 반환되는 기능
  - ✓ Callback-base / Promise-base 방식을 모두 지원하는 유연한 API 제공

2. Mongoose Connect (https://mongoosejs.com/docs/api.html#Connection)

- 프로젝트 디렉토리에서 NPM 명령어로 Mongoose 설치

```
npm install mongoose –-save
```

- MongoDB 서버와 통신하기 위한 connection 객체 생성

Synchronous

```
const mongoose = require('mongoose')
// Database 이름은 "Lab10-자신의 학번"으로 지정한다.
mongoose.connect('mongodb://127.0.0.1:27017/Lab10-201812345',
{ useUnifiedTopology: true, useNewUrlParser: true })
const conn = mongoose.connection
```

Async/Await

```
const mongoose = require('mongoose')
// Database 이름은 "Lab13-자신의 학번"으로 지정한다.
const conn = await mongoose.connect('mongodb://127.0.0.1:27017/Lab10-201812345',
{ useUnifiedTopology: true, useNewUrlParser: true })
```

  - ✓ 서버 주소 작성방법: **"mongodb://ADDRESS:PORT/DATABASE"**
  - ✓ 생성된 connection 객체를 이용해서 Schema 정의 및 CRUD 작업 호출이 가능함

3. Mongoose API

■ Schema

✓ Collection 에 저장할 Document 의 데이터 구조를 모델링하는 기능

```
const userSchema = new mongoose.Schema({
        name: {
                first: String,
                last: String
        },
        createdAt: {
                type: Date,
                default: Date.now
        }
})
```

✓ **Unique**

- 해당 Collection 내에서 Document 간 중복될 수 없는 필드임을 명시
- Document 를 저장할 때, unique:true 인 필드가 이미 저장된 다른 Document 와 중복된다면 저장에 실패 (에러 발생)

```
username: {
                type:String,
               unique:true,
}
```

✓ **Required**

- 새로운 Document 를 저장할 때, 무조건 포함해야 하는 필드임을 명시
- required:true 인 field 가 비어 있다면 저장에 실패 (에러 발생)

```
userid: {
                type:String,
                required:true,
}
```

✓ **Default**

- Document 를 저장할 때, 해당 필드의 값이 존재하지 않는다면 기본값을 할당

```
createdAt: {
                type: Date,
                default: Date.now
}
```

✓ **Virtual**

- 해당 필드에 접근할 때에만 생성되는 가상의 필드 값으로, DB 에 저장 X

```
userSchema.virtual('fullName', () => {
        return `${this.firstName} ${this.lastName}`
})
```

■ **Model**

✓ 앞서 정의한 Schema 를 기반으로 Collection 에 CRUD 수행을 하기 위한 객체이다.

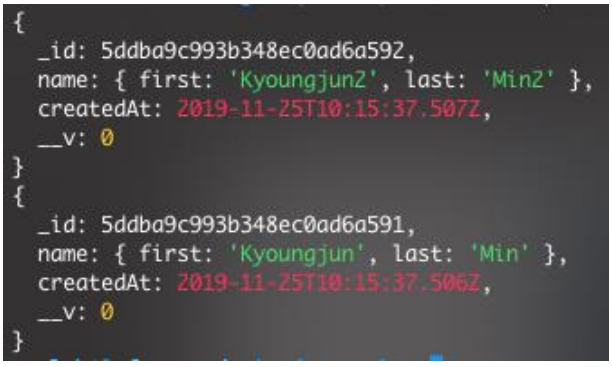
```
// "User" Collection 에 데이터 조작(CRUD)을 처리하기 위한 모델 생성
const userModel = mongoose.model('User', userSchema)
```

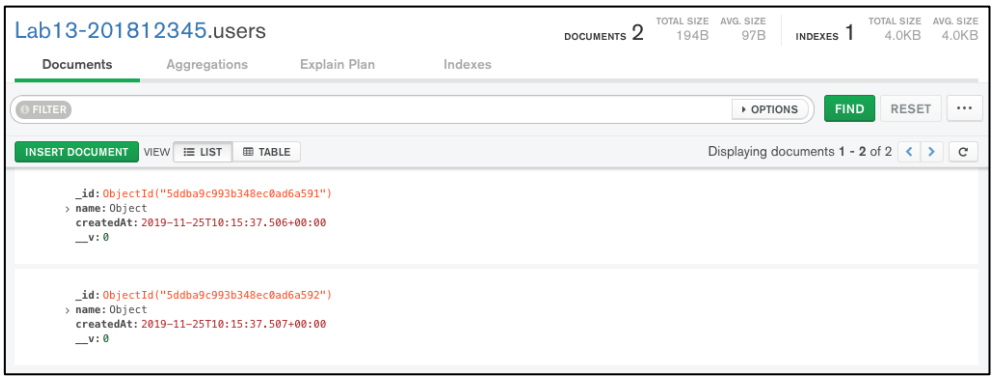
○ 생성한 model 을 이용해 Document 의 검색/수정/삭제 및 새로운 Document 생성이 가능함

✓ 새로운 Document 생성

```
// 새로운 Document 생성
const data1 = new userModel({ name: { first: "Kyoungjun", last: "Min" } })
const data2 = await userModel.create({ name: { first: "Kyoungjun2", last: "Min2" } })

data1.save(err => {
        if (err) console.error(err)
        else console.log(data1)
})
try {
        const res = await data2.save()
        console.log(res)
} catch (err) {
        console.error(err)
}
```

○ 새로운 Document 가 저장되었음을 콘솔 및 Compass 에서 확인할 수 있다.





■ Mongoose 의 주요 API

✓ create – Document 생성

- Model 스키마를 기반으로 새로운 Document 인스턴스를 생성

```
const new_data = await userModel.create({ name: {first: "Kyoungjun", last: "Min"} })
```

✓ save – Document 저장

- 생성한 인스턴스를 저장하기 위한 메소드로, 새로 생성 된 인스턴스들은 save 를 호출해야만 DB 내에 Document 로 저장 된다.

```
const document = await new_data.save()
{ name: { first: 'Kyoungjun', last: 'Min' }, _id: 0, __v: 0 }
```

✓ find – Document 검색

- 찾고자 하는 Document 의 조건을 매개변수에 객체 형태로 지정하며, 해당 조건과 일치하는 값을 포함하는 Document 의 목록을 반환

```
const result = await userModel.find({name: { first: "Kyoungjun", last:"Min" }})
// [
// { name: { first: 'Kyoungjun', last: 'Min' }, _id: 0, __v: 0 }
//     ...
// ]
```

  - find 메소드는 1 개 이상의 검색 결과가 반환되며, 한 개의 결과만 얻고 싶은 경우에는 findOne 메소드를 사용할 수 있다.

```
const result = await userModel.findOne({name: { first: "Kyoungjun", last:"Min" }})
// { name: { first: 'Kyoungjun', last: 'Min' }, _id: 0, __v: 0 }
```

✓ update – Document 수정

- 조건(첫번째 매개변수)에 일치하는 값을 포함하는 Document 의 일부 또는 전체를 수정

```
const result = await userModel.update({name: { first: "Kyoungjun", last:"Min" }}, {name:
{last: "Choi"}})
// { n: 1, nModified: 1, ok: 1 }
```

- n: 선택 된 Document 개수, nModified: 수정 된 Document 개수
- 한 번에 여러 개의 Document 를 수정하려면 updateMany 메소드를 사용할 수 있다.

✓ deleteOne – Document 삭제

○ 조건(첫번째 매개변수)에 일치하는 값을 포함하는 Document 를 DB 에서 삭제한다.

```
const result = await userModel.deleteOne({name: { first: "Kyoungjun", last:"Min" }})
// { n: 1, ok: 1, deletedCount: 1 }
```

- n: 선택 된 Document 개수, deletedCount: 삭제 된 Document 개수
- 한 번에 여러 개의 Document를 삭제하려면 deleteMany 메소드를 사용할 수 있다.

■ **Auto-increment Index**

✓ MongoDB 는 자체적으로 생성되는 고유한 객체 id 가 존재하기 때문에, Document 생성 시 index 값을 자동으로 부여하는 기능을 포함하고 있지 않다.

✓ 따라서, Document 생성 시 순차적인 index 값을 자동으로 부여하기 위해서는 해당 기능을 직접 구현하거나 외부 플러그인의 적용이 필요

➔ mongoose-auto-increment 사용 (https://www.npmjs.com/package/mongoose-auto-increment)

○ NPM 으로부터 플러그인 설치

```
npm i –save mongoose-auto-increment
```

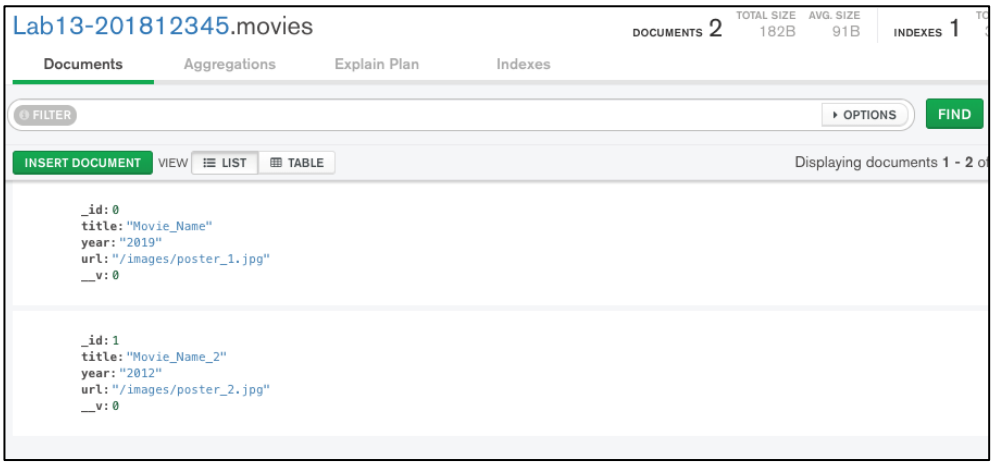
○ App.js 에서 mongoDB 연결 후 플러그인 등록

```
const mongoose = require('mongoose')
const autoInc = require('mongoose-auto-increment')

mongoose.connect("mongodb://localhost:27017/Lab13-201812345")
autoInc.initialize(mongoose.connection)
```

○ (DB 에 Model 을 등록하기 전에) Schema 에서 auto-increment 설정

```
const movieSchema = mongoose.Schema({ /* ... */ })

movieSchema.plugin(autoInc.plugin, 'Movie')
mongoose.model('Movie', movieSchema)
```

✓ mongoose-auto-increment 플러그인 설정을 마치고 나면, Movie Collection 에서 생성되는 Document 의 "_id" 필드에 0 부터 시작하는 index 값이 저장되는 것을 확인할 수 있다.



학습과제

1. 과제 개요

- NoSQL 기반인 MongoDB 의 collection 와 document 개념을 익힌다.
- Mongoose 를 사용하여 Node.js backend 에서 MongoDB 의 데이터를 조작하는 방법에 대해 학습한다.
- MongoDB 와 연동된 Node.js 백엔드 및 Pug(Jade) 템플릿 기반 프론트 엔드를 구성하여 영화 정보 관리 웹 어플리케이션을 구성한다.

2. 학습과제 파일 구성 및 과제 제출 양식

- 학습과제는 express-generator 를 기반으로 한 스켈레톤을 기반으로 구성되며, 추가되는 파일 구성은 아래와 같다.

상위 폴더 명	추가 파일 이름	설명
models	**movie.js**	영화 정보를 저장하는 스키마 및 모델을 정의
public/images	**poster_[1,2,3,4,5].jpg**	포스터 이미지 파일
public/stylesheets	**style.css**	프론트엔드 단의 CSS 파일
routes	**index.js**	기본적인 웹페이지 요청을 처리하는 모듈
	movie.js	Movie 정보에 대한 CRUD 요청을 처리하는 모듈
views	**index.pug**	영화 정보 웹 어플리케이션의 시작 웹 페이지
	layout.pug	Navbar 및 CSS, JavaScript 로드를 위한 레이아웃 템플릿
	newmovie.pug	신규 영화 등록을 웹페이지

- 이외 express-generator 를 통해 생성되는 파일들에 대해 수정이 필요한 경우, 수정을 수행한다.

3. 제출방법 및 주의사항

- 설치된 node package 들을 제외한 프로젝트 폴더를 압축해서 제출
  - ✓ 즉, node_module 폴더는 압축파일에 포함시키지 않는다.
- 압축 파일의 이름을 "Lab10_자신의학번.zip"으로 지정하여 폴더(프로젝트)를 압축한다.
- 압축 파일을 학습보고서 1~3 과 함께 제출한다.
- 지각 제출 시, 0점으로 처리한다.
- 채점 시 완성도의 평가는 Google의 Chrome 브라우저에서 렌더링 된 화면을 기준으로 함

※ **"4. 프로그램 설계"**를 참고하여 프로그램이 올바르게 동작하도록 구현한다.

4. 프로그램 설계

4.1 개요

- frontend는 Pug 템플릿 기반으로 백엔드 측에서 렌더링이 진행된다.
- backend는 영화 정보에 대한 CRUD(생성, 검색, 갱신, 삭제) 요청 및 웹페이지에 대한 요청을 처리한다.
- 예시들의 서버 포트 번호는 3000번으로 가정한다.

4.2 템플릿 및 프론트엔드 단 JavaScript 및 CSS

- 아래의 템플릿은 layout.pug를 상속받으며, layout.pug의 content 블록을 각각 정의한다.

4.2.1 layout.pug

- layout.pug는 본 웹 어플리케이션의 템플릿의 기반이 되는 템플릿으로써 이후 설명되는 템플릿은 layout.pug 를 확장한다. 이 템플릿은 다음과 같은 역할을 수행한다.

  1. 외부 CSS 정의 로드
  2. Javascript 파일 로드
  3. 메뉴 이동을 위한 Navbar 정의(Home, New Movie)
     - Home : 서버로 "/", Get 메소드 요청
     - New Movie : 서버로 "/newmovie", GET 메소드 요청

4.2.2. index.pug

- 사용자로부터 "/" 경로로 Get 메소드 요청이 올 때, 렌더링 되는 템플릿이다.

- 템플릿은 백엔드 측에서 영화 정보 리스트 및 인기 영화 리스트를 얻어와 두가지 과정을 수행한다.

  1. 영화 정보(영화 이름, 영화 개봉년도, 영화 포스터 이미지) 리스트 출력
  2. 인기 상영 영화 리스트(영화 이름, 영화 개봉년도, 영화 포스터 이미지) 출력

4.2.3. newmovie.pug

- 사용자로부터 "/newmovie" 경로로 GET 요청이 올 때 렌더링되는 템플릿이다.
- 이 템플릿은 새로운 영화 등록을 위한 템플릿을 제공하며, 영화 등록 UI 구성 및 영화등록 요청을 백엔드에 전달하기 위해 다음과 같은 형태의 form element을 구성한다.

Form 의 메소드	POST
Form 의 Action	/routes/movie/create
Label element	Text content 가 Title 인 element
Input element	name attribute 가 title 인 element
Label element	Text content 가 Year 인 element
Input element	name element 가 year 인 element
Label element	Text content 가 Image Path 인 element
Input element	name element 가 url 인 element

4.2.4. CSS(/public/stylesheets/style.css)

- 과제의 구현 난이도로 인해 별도의 CSS 요구사항은 존재하지 않으나, 직관성 있는 CSS를 구성할 경우 소폭 추가 점수 부여 예정.

4.3. 백엔드단 요청 처리

4.3.1 MongoDB 설정(/app.js)

- **Mongoose** 로 연결하는 MongoDB 의 주소는 각자 로컬에 설치한 서버 주소와 기본 포트를 사용한다
- 데이터를 저장하는 **Database** 의 이름은 **"Lab10-자신의 학번"** 으로 지정한다.
- 따라서 해당 데이터베이스에 접속하는 **url** 은 다음과 같다. **(mongodb://localhost:27017/Lab10-학번)**
- **Mongoose-auto-increment 패키지를 설치하고 Import 하여 id 의 중복이 발생하지 않도록 한다.**

```
const mongoose = require('mongoose');
const mongooseAutoInc = require('mongoose-auto-increment');

mongoose.connect('mongodb://localhost:27017/Lab10-201724525', {
```

```
  useFindAndModify: false,
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
mongooseAutoInc.initialize(mongoose.connection);
```

4.3.2. 영화 스키마 및 모델 정의(models/movie.js)

- Id의 자동 증가를 위해 mongoose-auto-increment를 Import 한다.

```
const mongoose = require('mongoose');
const mongooseAutoInc = require('mongoose-auto-increment');

const Schema = mongoose.Schema;

const movieSchema = new Schema({
  title: {
    type: String,
    required: true,
  },
  year: {
    type: Number,
    required: true,
  },
  url: {
    type: String,
    required: true,
  },
  trending: {
    type: Boolean,
    required: true,
    default: false,
  },
});

movieSchema.plugin(mongooseAutoInc.plugin, 'movie');

module.exports = mongoose.model('movie', movieSchema);
```

4.3.3 index.js

- Index.js 의 경우, 영화 정보 어플리케이션의 주요 웹페이지의 렌더링을 처리하는 모듈을 정의함.
- 요청 실패 상황은 고려하지 않는다.

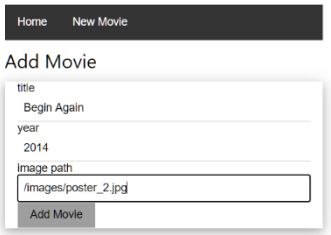
Method	URI	동작
GET	**/**	✓ MongoDB 내에 저장된 **모든 영화의 목록을 배열 형태로 반환한다.** ✓ 반환된 영화 목록을 활용하여 유행영화 목록 생성 ✓ 영화 목록 및 유행 영화 목록을 렌더링을 위한 local variable로 활용하여 index.pug를 렌더링한다..
GET	**/newmovie**	✓ MongoDB에 새로운 영화 등록을 위한 newmovie.pug 템플릿을 렌더링한다..

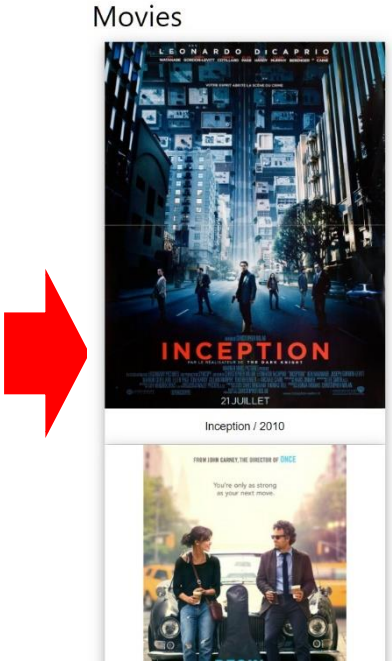
4.3.4 movie.js

- movie.js 의 경우, 영화 정보의 CRUD를 담당한다.
- 요청 실패 상황은 고려하지 않는다.

POST	**/routes/movie/create**	✓ 클라이언트가 전송한 영화 정보를 **MongoDB** 에 저장한다. ✓ 서버에 저장되는 영화 정보의 데이터 구조는 다음과 같다. **{ _id, title, year, url, trending }** ➔ **_id** : mongoose-auto-increment 에 의해 자동 생성된 index ➔ **title**: 클라이언트로부터 전송 받은 **영화 제목** ➔ **year**: 클라이언트로부터 전송 받은 **영화의 개봉 년도** ➔ **url**: 클라이언트로부터 전송 받은 **영화 포스터의 url** ➔ **trending**: 인기 영화를 분류하기 위한 **Boolean** 값으로, **영화 생성시 false 값**이 저장되어야 한다. ✓ 데이터 저장에 성공하면, http://**localhost:3000/** **로 redirect한다.**

5. 실행 예시





http://localhost:3000/ 접속 NavBar 의 New Movie 선택 ➡ 새로운 영화 등록 및 확인