Nodejs

1일차

교제 > 참고

2015 년도 ECMA5 부터 자바스크립트가 틀이 잡히기 시작함. >> 여기서부터 node가 ECMA5에 지원을 하기 시작함.

>> node가 Ryno를 만듬.

왜 node를 쓸까??? >> 노드에 특징은 자바스크립트와 동일함(비동기, 이벤트 기반, **non-blocking I/O**)

Non-blocking : 입출력 처리는 시작만 해둔 채 완료되지 않은 상태에서 다른 처리 작업을 계속 진행할 수 있도록 멈추지 않고 입출력 처리를 기다리는 방법을 말한다.(ex: youtube 같은 경우 동영상이 나오는 동안 댓글등 탐색이 가능함.)

아키텍처 같은 느낌으로는 queue : 데이터를 쓰고 빠진다.

>> 목적은 확장 가능한 네트워크 프로그램을 간단히 만드는 것.

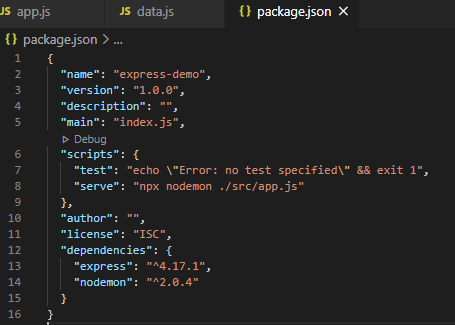
>> Nodejs 는 하나의 스레드(싱글Thread)임. 하나가 이벤트 루프를 구동

>> 고성능 서버의 쉬운 개발이 가능.

역사로 따지면 CPU 하나하나 리소스등이 중요했기 때문에 많이 리소스를 먹는 애들을 잘 안 썻음 그러나 성능이 좋아지기 시작하면서 많이 쓰기 시작함.

Reactive(반응성) ex) ajax > <http://sculove.github.io/blog/2016/06/22/Reactive-Programming/> 참고

Nodejs > 교제 참고 9page > c++을 사용> OS와 연결이 생각보다 밀접하다.



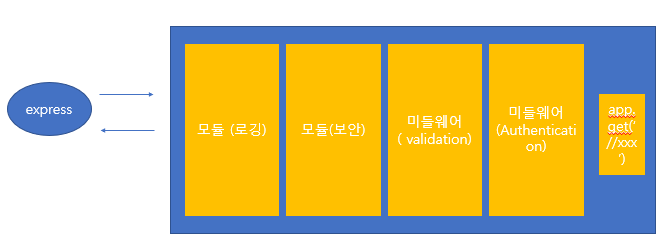
Serve 추가 > 실행시 : npm run serve

2일차

Node middleware (express, static, multer 등등) : 미들웨어란 기능, 특히 애플리케이션에 대한 http 요청에서 동작하는 기능을 캡슐화하는 방법

* **매개변수 세가지를 받는 함수(응답, 요청, next, 옵션 error)**
* next()를 호출하지 않으면 해당요청은 그 미들웨어에서 중단됨.
* 단계 단계 지나처서 진행

Next() 호출시 다음 미들웨어로 진행이 된다. 그러나 해당 미들웨어에서 send(응답)을 보내고 next() 호출한다면 진행이 되지 않는다. [이미 요청받은것에 대한 응답을 보낸것이기 때문(한프로세스가 진행이 된것)]



* 자주 쓰는 미들웨어들
* **Morgan : 로깅**
* **Body-parser : 데이터를 받음:**
* Cookie-parser : 쿠키 데이터를 받음
* Method-override :http 특정 메소드를 가능하게 해줌

글 등록(post)

글 조회(get)

글 수정(PUT, PATCH)

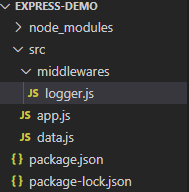
* Put : 전체 수정 (전부 다 주면 바로 엎어치기) –(backend 관점에서 다 주는게 정책상 편리)
* Patch : 부분 수정 (일부분을 찾아서 수정)

글 제거(delete)

node에서 on이 들어간 함수는 event listener on(‘test’, (a,b) => {}) :: test라는 이벤트가 왔을 때 해당 콜백함수를 실행



일반적인 디렉토리 구조



비즈니스 로직 찢기 전 app.js(main)

const mockData = require('./data');

const express = require('express');

const morgan = require('morgan');

const bodyParser = require('body-parser');

const logger = require('./middlewares/logger');

const app = express();

app.use(morgan('combined'));

app.use(bodyParser.json()); //이 미들웨어 없으면 post(articles) req.body안됨

//app.use(bodyParser.urlencoded());

app.use(logger);

app.get('/articles', (req, res) => {

    //DB에서 전체글을 조회해서 JSON 형태 만들어서 응답..

    res.json(mockData);

});

app.post('/articles', (req, res) => {

    const article = req.body;

    console.log(article);

    res.send('글이 잘 등록 외었습니다.');

});

app.get('/articles/:articleId', (req, res) => {

    const articleId = req.params.articleId;

    mockData.forEach(function(item){

        if (item.articleId == articleId){

            res.send(item);

        }

    });

});

app.put('/articles', (req, res) => {

    const article = req.body;

    res.send('글이 정상적으로 수정 완료');

});

app.delete('/articles/:articleId', (req, res) => {

    const keyForDelete = req.params.articleId;

    res.send(keyForDelete+'글이 정상적으로 삭제되었습니다.');

});

app.listen(5000, function(){

    console.log('service ready on port 5000');

});

Loger.js

module.exports = function (request, response, next){

    const start = +new Date();

    const stream = process.stdout;

    const url = request.url;

    const method = request.method;

    response.on('finish', function() {

        let duration = +new Date() - start;

        let message =  method+' to '+url+'\ntook '+duration+' ms \n\n';

        stream.write(message);

    });

    next();

};

App.js 찢기 [url 기준으로 api 찢기 (api/articles, api/members)]

1. 비즈니스 로직 부분 제거

const mockData = require('./data');

const express = require('express');

const morgan = require('morgan');

const bodyParser = require('body-parser');

const logger = require('./middlewares/logger');

const app = express();

app.use(morgan('combined'));

app.use(bodyParser.json()); //이 미들웨어 없으면 post(articles) req.body안됨

//app.use(bodyParser.urlencoded());

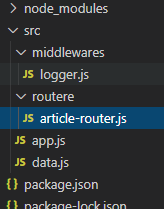
app.use(logger);

app.listen(5000, function(){

    console.log('service ready on port 5000');

});

1. Router 폴더 생성 (복수형 articles 같은 경우는 collection)



1. Article-router.js 에 app.js에 있던 비즈니스 로직 붙이기 (app 대신에 router를 불러서 생성)

const app = request('express');//express중 라우터 기능만 있는게 있음.

const router = app.Router;

router.get('/articles', (req, res) => {

    //DB에서 전체글을 조회해서 JSON 형태 만들어서 응답..

    res.json(mockData);

});

router.post('/articles', (req, res) => {

    const article = req.body;

    console.log(article);

    res.send('글이 잘 등록 외었습니다.');

});

router.get('/articles/:articleId', (req, res) => {

    const articleId = req.params.articleId;

    mockData.forEach(function(item){

        if (item.articleId == articleId){

            res.send(item);

        }

    });

});

router.put('/articles', (req, res) => {

    const article = req.body;

    res.send('글이 정상적으로 수정 완료');

});

router.delete('/articles/:articleId', (req, res) => {

    const keyForDelete = req.params.articleId;

    res.send(keyForDelete+'글이 정상적으로 삭제되었습니다.');

});

라우팅 경로[참고]

문자열을 기반으로 하는 라우트 경로의 몇 가지 예는 다음과 같습니다.

다음의 라우트 경로는 요청을 루트 라우트 /에 일치시킵니다.

app.get('/', function (req, res) {

res.send('root');

});

다음의 라우트 경로는 요청을 /about에 일치시킵니다.

app.get('/about', function (req, res) {

res.send('about');

});

다음의 라우트 경로는 요청을 /random.text에 일치시킵니다.

app.get('/random.text', function (req, res) {

res.send('random.text');

});

문자열 패턴을 기반으로 하는 라우트 경로의 몇 가지 예는 다음과 같습니다.

다음의 라우트 경로는 acd 및 abcd와 일치합니다.

app.get('/ab?cd', function(req, res) {

res.send('ab?cd');

});

다음의 라우트 경로는 abcd, abbcd 및 abbbcd 등과 일치합니다.

app.get('/ab+cd', function(req, res) {

res.send('ab+cd');

});

다음의 라우트 경로는 abcd, abxcd, abRABDOMcd 및 ab123cd 등과 일치합니다.

app.get('/ab\*cd', function(req, res) {

res.send('ab\*cd');

});

다음의 라우트 경로는 /abe 및 /abcde와 일치합니다.

app.get('/ab(cd)?e', function(req, res) {

res.send('ab(cd)?e');

});

?, +, \* 및 () 문자는 정규식 문자의 서브세트입니다. 하이픈(-) 및 점(.)은 문자열 기반 경로에 의해 문자 그대로 해석됩니다.

정규식을 기반으로 하는 라우트 경로의 예:

다음의 라우트 경로는 라우트 이름에 “a”가 포함된 모든 항목과 일치합니다.

app.get(/a/, function(req, res) {

res.send('/a/');

});

다음의 라우트 경로는 butterfly 및 dragonfly와 일치하지만, butterflyman 및 dragonfly man 등과 일치하지 않습니다.

app.get(/.\*fly$/, function(req, res) {

res.send('/.\*fly$/');

});

1. articleRouter app.js에 적용 (article-router.js) > app.js에서 root값 uri를 갖고 있음.

const app = require('express');//express중 라우터 기능만 있는게 있음.

const mockData = require('../data');

const router = app.Router();

router.get('/', (req, res) => {

    //DB에서 전체글을 조회해서 JSON 형태 만들어서 응답..

    res.json(mockData);

});

router.post('/', (req, res) => {

    const article = req.body;

    console.log(article);

    res.send('글이 잘 등록 외었습니다.');

});

router.get('/:articleId', (req, res) => {

    const articleId = req.params.articleId;

    mockData.forEach(function(item){

        if (item.articleId == articleId){

            res.send(item);

        }

    });

});

router.put('/, (req, res) => {

    const article = req.body;

    res.send('글이 정상적으로 수정 완료');

});

router.delete('/:articleId', (req, res) => {

    const keyForDelete = req.params.articleId;

    res.send(keyForDelete+'글이 정상적으로 삭제되었습니다.');

});

module.exports = router;

1. app.js

//const mockData = require('./data');

const express = require('express');

const morgan = require('morgan');

const bodyParser = require('body-parser');

const logger = require('./middlewares/logger');

const articleRouter = require('./router/article-router');

const app = express();

app.use(morgan('combined'));

app.use(bodyParser.json()); //이 미들웨어 없으면 post(articles) req.body안됨

//app.use(bodyParser.urlencoded());

app.use(logger);

app.use('/articles', articleRouter);

//app.use('/members', memberRouter);

app.listen(5000, function(){

    console.log('service ready on port 5000');

});

**스캐폴딩을 통한 프로젝트 생성**

Npm install -g express-generator

Express express-calculator (프로젝트루트 상위 디렉터리로 가서 다음 명령어 실행)

터미널에서 > set DEBUG=myapp:\* & npm start

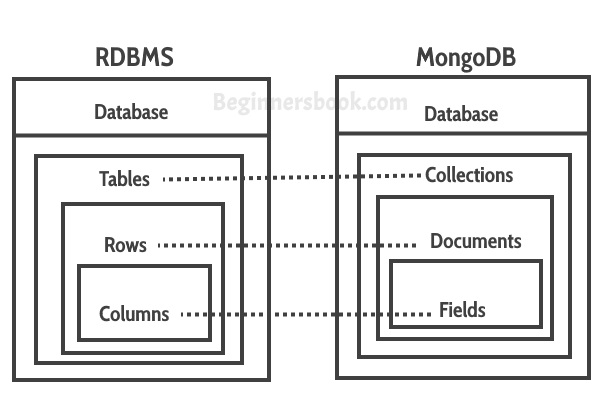
npm i >> package.json 파일에 있는 파일들 설치

실습때 하였던 명령어

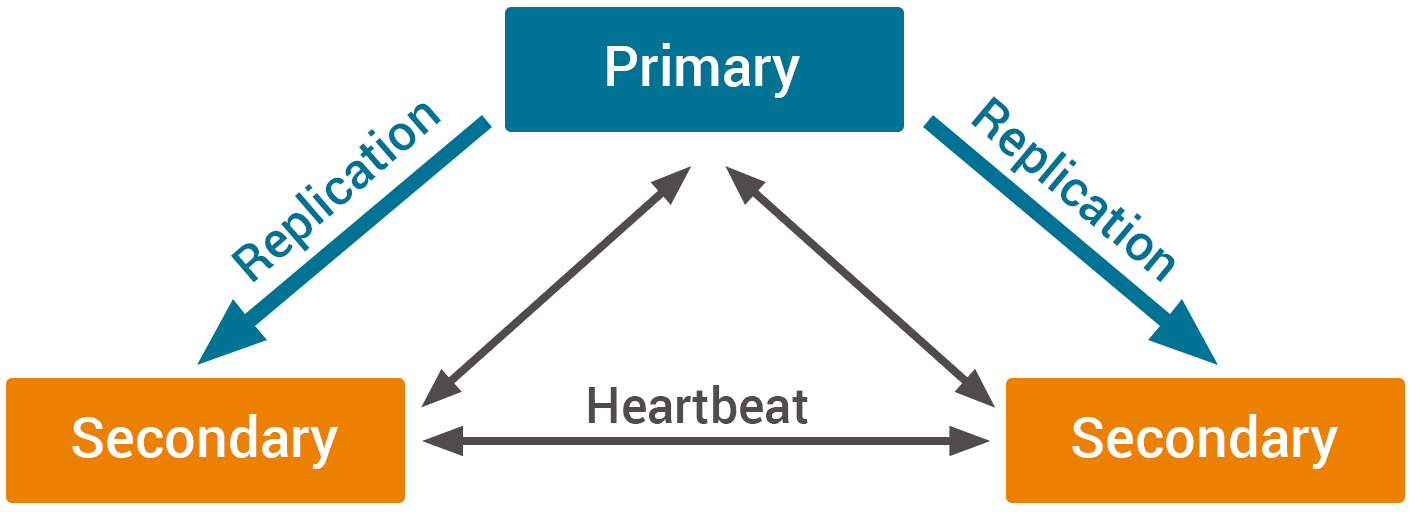
* npm install -g express-generator
* express ktds-sns >> 프로젝트가 생성이 됨.
* 예제 ) > 여기다 article 라우터 추가해서 작업 해보기.

Mongodb 연결

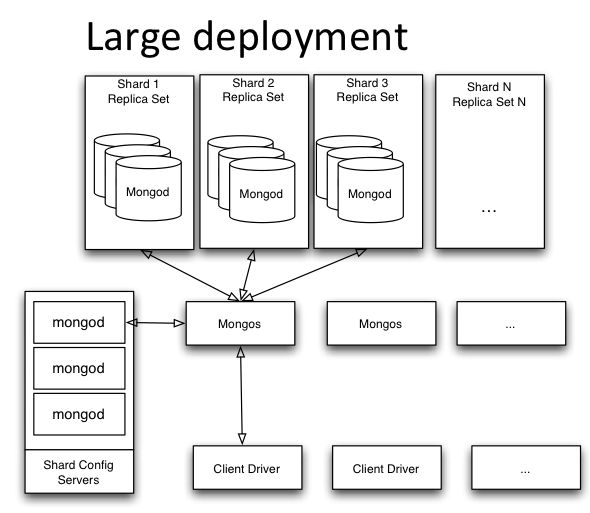
* Opensource noSQL 데이터베이스
* Key-value 기반
* 테이블 대신 표준 데이터 포맷을 사용
* 계층적 문서구조를 지원하는 JSON, XML 포맷을 저장
* 비 정형화 된 대용량의 데이터 처리에 좋다고 알려짐



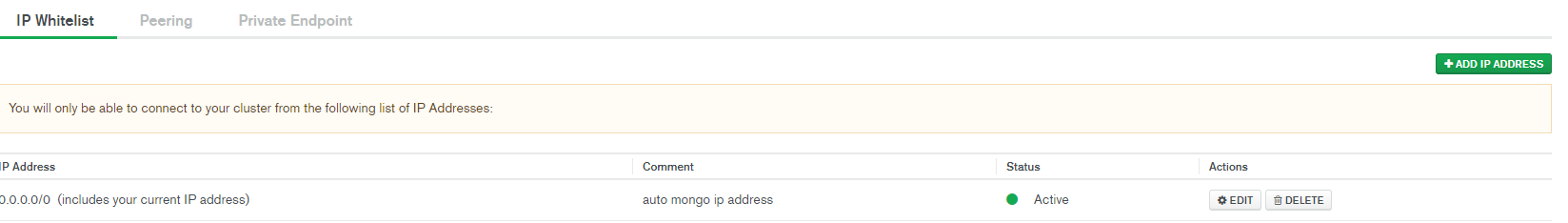
* BSON 데이터 포맷
  + Json과 유사하게 객체와 배열이 key-value 포맷
  + \_id : primary keyfh 사용, ObjectId라고 함, 자동생성
  + 날자 타입 : 에포크로부터 4바이트 숫자로 표현
* Ad hoc 쿼리
  + Select \* from posts where Title like ‘%mongo%’;
  + db.posts.find({title:/mongo/});
* replica set(기본적으로 3대가 운영이 되어야한다. 그래서 cloud나 docker에서 사용)

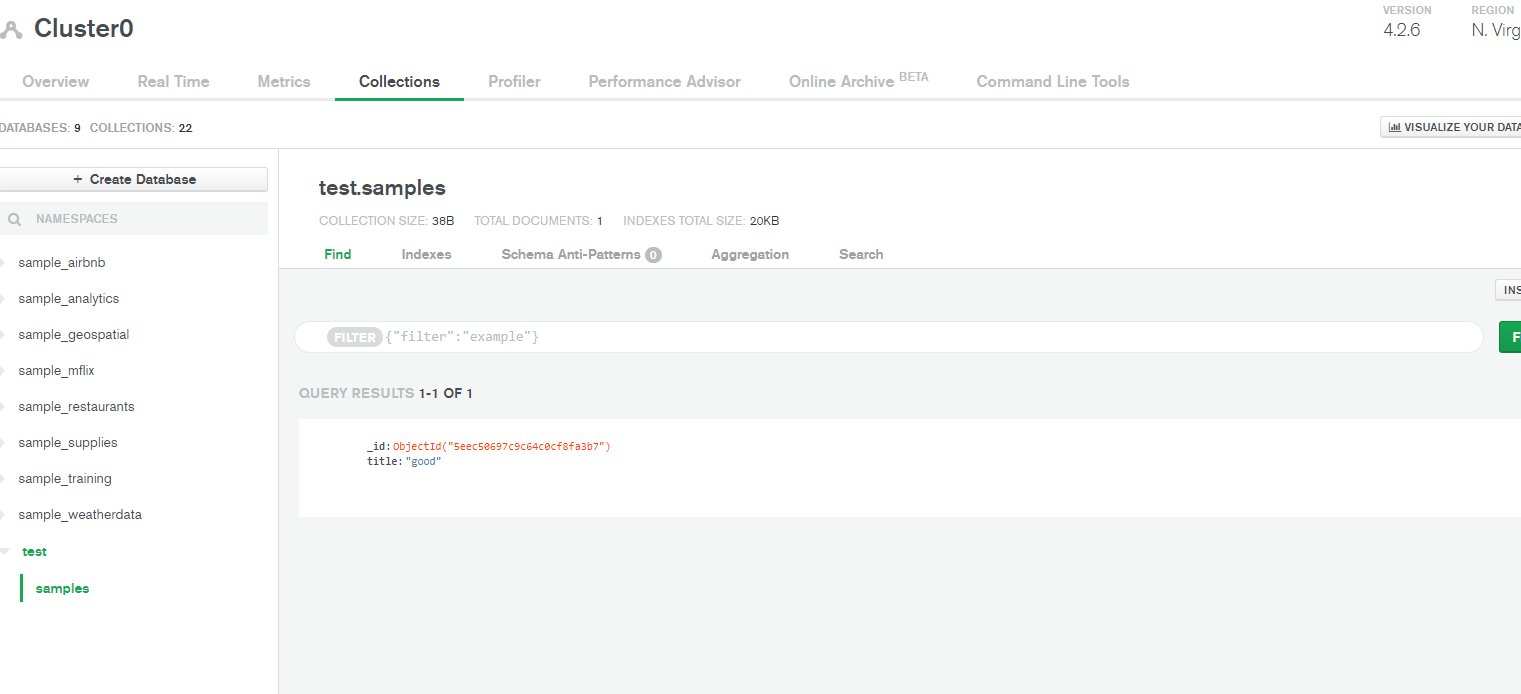


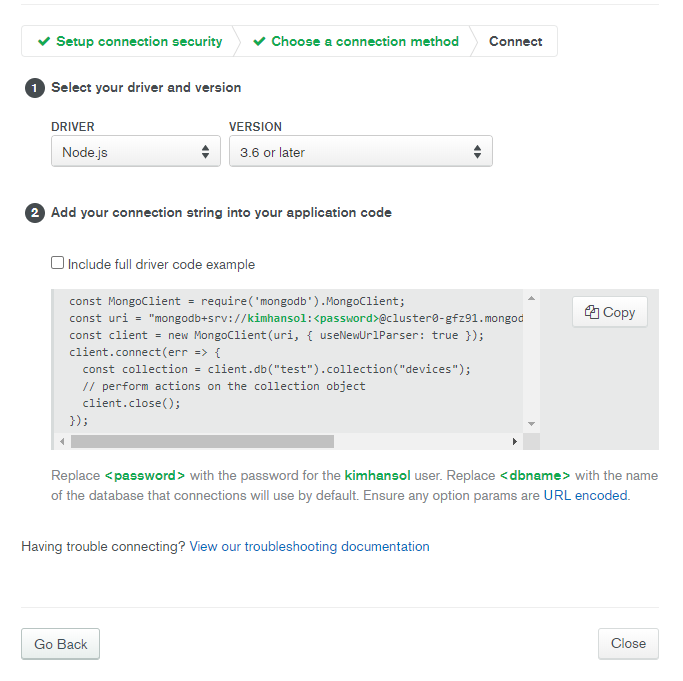
* Sharding



* Mongodb page
  + Mongo 페이지에 접근 > 로그인 > cluster 생성 > testdb 생성 > insertCollection >> connect >> connect your application
  + Network access > 0.0.0.0/0 / create user







* Npm install mongodb
* 접속은 위 내용 카피 앤 패스트

비동기식 처리

1. Callback
2. Promise (ES6)
3. Async ~ await (ES8)

* Mongoose 접속

mongoose.connect(

  'mongodb+srv://kimhansol:rlagksthf12!@cluster0-gfz91.mongodb.net/test?retryWrites=true&w=majority'

  ,{

    useNewUrlParser: true, useUnifiedTopology: true}, function(err){

      if (err){

        console.log(err);

      }else{

        console.log("mongodb connect ok");

      }

    }

  );

* 모델 만들기 ( /models/article.js)

const Mongoose = require('mongoose');

const Schema = Mongoose.Schema;

const ArticleSchema = new Mongoose.Schema({

    articleId: {

        type: String,

        required: true

    },

    title: String,

    content: String,

    author: {

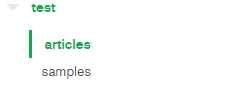
        type: String,

        required: 'noname'

    }

});

module.exports = Mongoose.model('Article', ArticleSchema); //모델명 Article => articles



* Insert 작업 (post)

var Article = require('../models/article');

router.post('/', (req, res) => {

    const article = req.body;

    console.log(article);

    const forSave = new Article(article);

    forSave.save()

    .then(function(saved){

        console.log('insert ok');

        res.send(saved);

    }).catch(function(err){

        console.log(err);

        res.send('error 발생');

    });

});

* Select \*

/\* GET articles listing. \*/

router.get('/', function(req, res, next) {

  //res.json(mockData);

  articles = Article.find({}).exec(function(err, doc){

    res.send(doc);

  });

});

* 상세 보기

router.get('/:articleId', (req, res) => {

    //const articleId = req.params.articleId;

    const aId = req.params.articleId;

    // mockData.forEach(function(item){

    //     if (item.articleId == articleId){

    //         res.send(item);

    //     }

    // });

    Article.find({articleId : aId}).exec(function(err, doc){

        if(err){

            res.send(err);

        }

        else{

            res.send(doc);

        }

    });