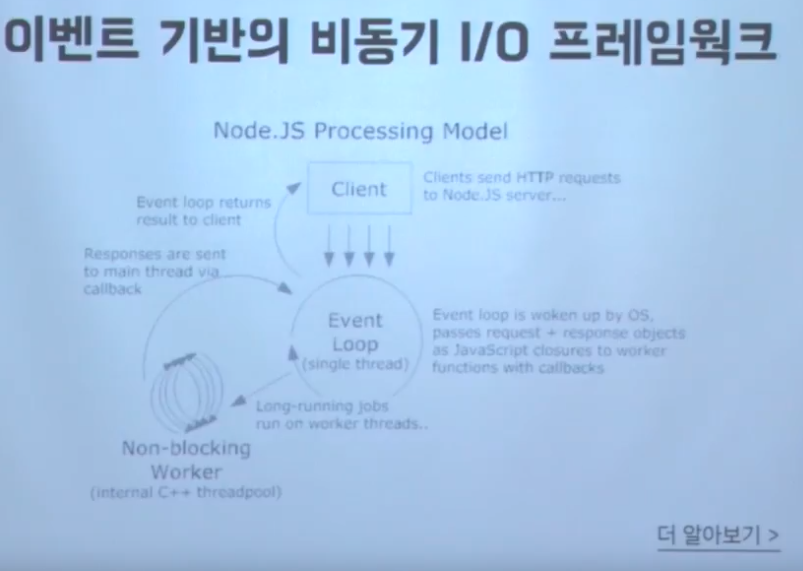
Node.JS

* 브라우저 밖에서 자바스크립트 코드를 실행할 수 있다,
* 크롬에서 사용하는 V8 엔진을 사용한다.
* 이벤트 기반의 비동기 I/O 프레임워크
* CommonJS를 구현한 모듈 시스템

브라우저 안에는 자바스크립트 코드를 해석해주는 엔진이 있는데, 크롬에서는 V8을 쓰고 있음

Node에서는 크롬에서 사용하는 V8 엔진을 사용한다.



노드는 main이되는 이벤트 loop가 하나만 동작하기에 single thread 라고도 한다.

Worker가 완료된 event를 보내주면, 메인 루프가 client에게 결과를 보내줌

무거운 job들은 worker에게 무거운 job들을 위임한다.

모듈 시스템

* 브라우저에서는 윈도우 컨텍스트를 사용하거나, RequireJS같은 의존성 로더를 사용함.
* 노드는 파일형태로 모듈을 관리할수 있는 CommonJS로 구현  
  기본 모듈  
  써드파티 모듈  
  사용자 정의 모듈

비동기 세계

* 노드는 기본적으로 비동기로 동작함
* readFile() vs readFileSync()
* readFileSync()의 경우 Event Loop가 Worker가 파일을 읽을때까지 멈춰서 기다려주지만(동기),

readFile일 경우 worker에게 파일 읽는 작업 위임하고 다음 이벤트를 처리한다(비동기).  
파일을 다 읽으면 Worker는 callback 함수를 호출함으로써 파일을 다 읽었다는 것을 Event loop에 알려준다.

노드의 실행 명령어

node ./index.js

윈도우는 postman 프로그램 설치해서 curl 명령어 안되는 것을 대체한다.

익스프레스JS 기초

노드에서 가장 많이 쓰는 웹 프레임워크가 Express.js이다.

* 어플리케이션
* 미들웨어
* 라우팅
* 요청/응답 객체

어플리케이션

* 익스프레스 인스턴스를 어플리케이션이라 한다.
* 서버에 필요한 기능인 미들웨어를 어플리케이션에 추가한다.
* 라우팅 설정을 할 수 있다.
* 서버를 요청 대기 상태로 만들 수 있다.

미들웨어

* 미들웨어는 함수들의 연속이다.
* 로깅 미들웨어를 만들어 보자
* 써드파티 미들웨어를 사용해 보자
* 일반 미들웨어 vs 에러 미들웨어

미들웨어는 순서가 중요하다.

미들웨어의 세팅순서에 따라서 서버의 동작이 바뀐다.

라우팅

* 요청 url에 대해 적절한 핸들러 함수로 연결해 주는 기능을 라우팅이라고 부른다.
* 어플리케이션의 get(), post() 메소드로 구현할 수 있다.
* 라우팅을 위한 전용 Router 클래스를 사용할 수도 있다.

요청 객체

* 클라이언트 요청 정보를 담은 객체를 요청(Request)객체라고 한다.
* http 모듈의 request 객체를 래핑한 것이다.
* Req.params(), req.query(), req.body() 메소드를 주로 사용한다.

응답 객체

* 클라이언트 응답 정보를 담은 객체를 응답(Response)객체라고 한다.
* http 모듈의 response 객체를 래핑한 것이다.
* Res.send(), res.status(), res.json() 메소드를 주로 사용한다.

3강

REST API

보통 api를 설계한다고 하면 누구나 다 REST API 라고 한다. (정말?)

HTTP 요청

* 모든 자원은 명사로 식별한다.
* HTTP 경로로 자원을 요청한다.
* 예

GET /users

GET /users/{id}

HTTP 메서드

* 서버 자원에 대한 행동을 나타낸다. (동사로 표현)

GET : 자원을 조회

POST : 자원을 생성

PUT : 자원을 갱신

DELETE : 자원을 삭제

* 이는 익스프레스 어플리케이션의 메소드로 구현되어 있다

HTTP 상태코드

200 : 성공(success), GET, PUT

201 : 작성된(created), POST

204 : 내용 없음(No Content), DELETE

첫 API 만들기

* GET /users
* 사용자 목록을 조회하는 기능

아래는 Npm init으로 만든 Package 정보

Dependencies에 node\_modules 폴더 안에 있는 라이브러리 정보들 포함되어 있음.

devDependencies 는 개발환경에 필요한 모듈임

Script에 명령어 추가하여 node 구동할 수 있음

* {
* "name": "node-api-server",
* "version": "1.0.0",
* "description": "",
* "main": "index.js",
* "dependencies": {
* "morgan": "^1.9.1",
* "express": "^4.17.1"
* },
* "devDependencies": {},
* "scripts": {
* "start" : "node ./ex\_test.js",
* "test": "echo \"Error: no test specified\" && exit 1"
* },
* "author": "",
* "license": "ISC"
* }

4강

테스트 주도 개발

* TDD로 개발하자
* Mocha, should, superTest

Mocha ( 설치 명령어 : npm install mocha –save-dev )

* 모카(mocha)는 테스트 코드를 돌려주는 테스트 러너
* 테스트 꾸러미 : 테스트 환경으로 모카에서는 describe()으로 구현한다.
* 테스트 케이스 : 실제 테스트를 말하며 모카에서는 it()으로 구현한다.
* Git checkout -f mocha

테스트 파일은 보통 **파일명.spec.js**으로 이름짓는다.

Should ( 설치 명령어 : npm install should –save-dev )

* 슈드(should)는 검증(assertion) 라이브러리다.
* 가독성 높은 테스트 코드를 만들 수 있다.

SuperTest ( 설치 명령어 : npm install supertest –save-dev)

* 단위 테스트 : 함수의 기능 테스트
* 통합 테스트 : API의 기능 테스트
* 슈퍼 테스트는 익스프레스 통합 테스트용 라이브러리이다.
* 내부적으로 익스프레스 서버를 구동시켜 실제 요청을 보낸 뒤 결과를 검증한다.

5강

TDD로 API 서버 개발

첫 API 테스트 만들기

-성공

- 유저 객체를 담은 배열로 응답한다.

- 최대 limit 개수만큼 응답한다.

-실패

- limit이 숫자형이 아니면 400을 응답한다,

- offset이 숫자형이 아니면 400을 응답한다.

6강 – 사용자목록 API 개발

GET /user/:id

Success

* Id가 1인 유저 객체를 반환한다.

Error

* Id가 숫자가 아닐경우 400으로 응답한다.
* Id로 유저를 찾을 수 없을 경우 404로 응답한다.

데이터를 전달하는 방법 3가지

1. Query string
2. Params
3. Body (POST나 PUT을 요청할 때는 body를 많이 쓴다.  
   test 에서 body는 send라는 함수로 보낼 수 있다.  
   express에서는 query string과 params만 제공하고 있다.  
   그래서 body를 추가하기 위해서는 body-parser 미들웨어를 사용해야한다.  
   npm install body-parser --save

Query와 param 사용법

.get('/users?limit=two')

app.get('/users', (req, res) => {

    req.query.limit = req.query.limit || 10

.get('/users/1')

app.get('/users/:id', (req, res) => {

    // id 값을 얻어낸다.

    const id = parseInt(req.params.id, 10)

DELETE /users/:id

Success

* 204를 응답한다. (rest 설계할 때 성공했는대 body가 없으면 204 응답함)

Error

* Id가 숫자가 아닐경우 400으로 응답한다.

POST /users

Success

* 201 상태코드를 반환한다. (create를 의미함. 어떤 자원이 추가되었다는)
* 생성된 유저 객체를 반환한다.
* 입력한 name을 반환한다.

Error

* Name 파라미터 누락시 400을 반환한다.
* Name이 중복일 경우 409를 반환한다. (409는 기존에 있던 리소스와 conflict가 났음을 의미함.)

7강 – 코드 리팩토링 및 데이터베이스 연동

Api/user/index.js 파일에 users 관련된 api들 일괄 이동하였음

Api/user/user.ctrl.js 파일에는 controller 코드들 따로 분리하여 저장하였음.

package

 "scripts": {

    "start": "node ./bin/www",

    "test": "mocha ./api/\*\*/\*.spec.js"

Test 항목에 와일드카드 부분 설명은, api 하위 모든 폴더에서 \*.spec.js 이름을 가진 모든 테스트 파일을 실행하겠다 라는 의미임

**각 파일의 역할 정리!!!!**

Ex\_test.js : 오로지 서버를 구성하는 기능만 수행하고 있음  
미들웨어 추가하는 부분이 핵심임  
Logger를 추가하고, bodyParser 추가  
경로에 대해서 라우팅 할 때는 user 라우터가 담당한다고 위임해놓음

Api/user/index.js : user 객체에 대한 라우팅을 설정하는 부분이 있음

Api/user/user.ctrl.js : 실제 controller에 갔을 때 어떤 동작이 일어나는지 함수별로 구현

Api/user/user.spec.js : 이전에 만들어 놓았던 testcode들 모두 user 객체의 테스트 코드이니 위의 경로로 이동 & 이름 변경

Bin/www.js : 서버를 응답 대기 상태로 만드는 부분

데이터베이스

SQL

* MySQL, PostgreSQL, Aurora, Sqlite

NoSQL

* MongoDB, DynamoDB

In Memory DB

* Redis, Memcashed
* 로그인해서 생기는 세션 정보같은 것을 DB에 저장하게되면 성능에 부하를 준다. 그래서 세션데이터를 메모리에 올려서 관리를 한다.

SQL 쿼리 기초

Insert users (‘name’) values (‘alice’);

Select \* from users;

Update users set name =’bek’ where id =1;

Delete from users where id = 1;

ORM

데이터베이스를 객체로 추상화해 논 것을 ORM(Object Relational Mapping)이라고 한다.

쿼리를 직접 작성하는 대신 ORM의 메소드로 데이터 관리할 수 있는 것이 장점이다.

노드에서 SQL ORM은 시퀄라이져(Sequelize)가 있다.

Insert users (‘name’) values (‘alice’);

* User.create({name:’alice’})

Select \* from users;

* User.findAll()

Update users set name =’bek’ where id =1;

* User.update({name:’bek’}, {where:{id:1}});

Delete from users where id = 1;

* User.destroy({where:{id:1}});