NODEJS

Dương Hữu Thành,
thanh.dh@ou.edu.vn,
Faculty of Information Technology,
Ho Chi Minh City Open University.

This slide is based on the below references

[1] https://nodejs.org/en/docs/
[2] https://www.tutorialspoint.com/nodejs/index.htm
[3] https://www.w3schools.com/nodejs/default.asp
[4] https://expressjs.com/

Agenda

- Introduction to NodeJS
- Callback
- File System
- URL module
- Global veriables
- Event Loop
- Working with MySQL
- Express framework
  - Introduction to express
  - Static files
  - Get request
  - Post request
- Working with MongoDB

Introduction to NodeJS

- NodeJS is developed by Ryan Dahl in 2009.
- It is an open-source and cross-platform to develop the server-side applications. These ones are written in Javascript and can be run within the NodeJS runtime on the operating system.
- NodeJS = Runtime environment + Javascipt libs

Introduction to NodeJS

- NodeJS uses an event-driven, non-blocking IO model that makes it lightweight and efficient, perfect for data-intensive real-time application that run across distributed devices.

Features of NodeJS

- Asynchronous and event-driven: NodeJS has based on server never waits for an API to return data, the server moves to the next API after calling it and a notification mechanism of events helps the server to get a response from the previous API call.
- NodeJS library is very fast in code execution.

Features of NodeJS

- Single Thread: NodeJS uses a single threaded model with event loop. Event mechanism helps the server to repond in a non-blocking way.
- No buffering: NodeJS applications have never buffered any data.

The first program

- A NodeJS application consists three important components
  - Import the required modules
  - Create Server
  - Read request and return response

The first program

- For example

```
var http = require("http")

http.createServer(function(request, response) {
  response.writeHead(200, {
    'Content-Type': 'text/plain'
  })

  response.end('Hello World\n')
}).listen(8081)

console.info('http://127.0.0.1:8081/')
```

Callback

- Callback function often used to continue some execution after an asynchronous operation has completed (calling asychronous callbacks).
- NodeJS makes heavy use of callbacks, all the APIs of NodeJS are written in such a way that they support callbacks.

File System

- The nodejs file system module allows to work with the file system on your computer.
- Every method in the fs module has synchronous as well as asynchronous.

```
var fs = require("fs")
```

File System

- fs.readFile(): reads files on your computer
- fs.appendFile(): appends the specified content to a file, if the file does not exist, it will be created.
- fs.writeFile(): write the specified content to a file. It will replace the content if the file exists, else a new file will be created with that content.
- fs.unlink(): delete the specified file.
- fs.stat(): view the stats of the specified file.

File System

- fs.mkdir(path[, mode], callback): create a directory.
- fs.readdir(path, callback): read a directory
- fs.rmdir(path, callback): remove a directory

File System

```
var http = require('http')
var fs = require('fs')

http.createServer(function(req, res) {
  fs.readFile('demo.html', function(err, data) {
    res.writeHead(200, {
        'Content-Type': 'text/html'
    })
    res.write(data)

    return res.end()
  })
}).listen(8080)
```

File System

```
var http = require('http')
var fs = require('fs')

http.createServer(function(req, res) {
    fs.appendFile("test.txt",
                  "Hello World!", function(err) {
        if (err)
            throw err
        console.info("saved")
    })
}).listen(8080)
```

File System

```
var http = require('http')
var fs = require('fs')

http.createServer(function(req, res) {
    fs.writeFile('demo.txt',
                 'Hello', function(err) {
        if (err)
            throw err
        console.info('saved!')
    })
}).listen(8080)
```

File System

```
var http = require('http')
var fs = require('fs')

http.createServer(function(req, res) {
    fs.unlink('demo.txt', function(err) {
        if (err)
            throw err
        console.info('deleted!')
    })
}).listen(8080)
```

URL Module

- The URL module splits up a web address into readable parts.
- Parsing an address with the url.parse() method, it will return a URL object with each part of the address as properties.

URL Module

```
var url = require('url')
var q =
url.parse('http://localhost:8080/t.html?y
ear=2020', true)

console.info(q.host)
console.info(q.pathname)
console.info(q.search)
console.info(q.query.year)
```

```
localhost:8080
/t.html
?year=2020
2020
```

URL Module

```
http.createServer(function(req, res) {
  var q = url.parse(req.url, true)
  fs.readFile(`.${q.pathname}`, function(err, data){
     if (err) {
       res.writeHead(404, {
          'Content-Type': 'text/html'
       })
       return res.end('404 - Not Found')
     }
     res.writeHead(200, {
        'Content-Type': 'text/html'
     })
     res.write(data)
     res.end()
 })
}).listen(8080)
```

Assignment

- Build the multiple choice test of english application
  - View all questions and search questions by name
  - Add question
  - Delete question
  - Practice with the random questions

Assignment

- Note: read body data of the post request

```
var parse = require('querystring')

let body = ''
req.on('data', chunk => {
   body += chunk.toString()
})

req.on('end', () => {
    let data = parse.parse(body)
})
```

Global Variables

- __filename: the filename of the code is being executed, this is the resolved absolute path.
- __dirname: the name of the directory that the currently executing script resides in.
- setTimeout()/clearTimeout()
- setInterval()/clearInterval()

Working with MySQL

- Install mysql module

```
npm install mysql
```

- Config the connection info

```
let mysql = require('mysql')

let conn = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: '12345678',
    database: 'saledb'
})
```

Working with MySQL

- Connect to the database

```
conn.connect(err => {
    if (err)
        throw err

    console.info("connected")
})
```

Working with MySQL

- Execute the query

```
conn.query('SELECT * FROM product',
        function(err, rows, fields) {
    if (err)
        throw err

    console.info(rows)
})
```

- Close the connection

```
conn.end()
```

Working with MySQL

- Select the name and price of the products which the name contains some keywords.

```
let sql = "SELECT * FROM product " +
          "WHERE name like '%iphone%'"
conn.query(sql, function(err, data) {
    if (err)
        throw err

    data.forEach(d => {
        console.info(`${d.name} - ${d.price}`)
    })
})
```

Working with MySQL

- Select products where their prices is greater than 15tr and less than 30tr.

```
let sql = "SELECT * FROM product " +
          "WHERE price between ? and ? "
conn.query(sql, [15000000, 30000000],
                  function(err, data) {
    if (err)
        throw err

    data.forEach(d => {
        console.info(`${d.name} - ${d.price}`)
    })
})
```

Event Loop

- NodeJS is a single-threaded application, but it can support concurrency via the concept of events and callbacks.
- Node thread keeps an event loop and whenever a task get completes, it fires the corresponding event.
- Event loop works on the observer pattern, whenever an event gets fired, its listener function begins to execute.

Event Loop

- The events module is a bulit-in module in NodeJS, it can create, fire, listen your own events.
- All event properties and methods are an instance of an EventEmitter object.
  - Using emit() method to fire an event.

Event Loop

```
var events = require('events')
var emitter = new events.EventEmitter()

emitter.on('connection', function() {
  console.info('connected...')
  emitter.emit('received')
})

emitter.on('received', function() {
  console.info('data received...')
})

emitter.emit('connection')
console.info('ended')
```

Express framework

- Express is a minimal and flexible, it provides a robust set of features to develope web and mobile application.
- It facilitates the rapid development of NodeJS based Web application.
- Installing the express framework

```
npm install express --save
```

Express framework

- Create the instance of the express application

```
var express = require('express')
var app = express()
```

- Binds and listens for connections on the specified host and port.

```
app.listen([port[, host]][, callback])
```

Express framework

- Other modules are important in express
  - body-parser: the middleware handles JSON, raw text, URL encoded form data.
  - cookie-parser: this parses cookie header and populate req.cookies with an object keyed by the cookie names.
  - multer: the middleware handles multipart/form-data

```
npm install body-parser --save
npm install cookie-parser --save
npm install multer --save
```

Request in Express App

- The req object represents the HTTP request and has properties for the request query string, parameters, body, HTTP header, ...
- req.app: the instance of the express application.
- req.baseUrl: the URL path which a router instance was mounted.
- req.method: a string is the HTTP method: GET, POST, PUT, ... in lowercase.

Request in Express App

- req.body: contains key-value pairs of data submitted in the request body. Notes that it only populated when you use body-parser middlewaer such as express.json() or express.urlencoded().
- req.params: contains the properties mapped to the named route parameters. For example, the route /user/:name, then the name property is req.params.name.
- req.query: contains a property for each query string parameter in the route.

Response in Express App

- The res object represents the HTTP response that an Express app sends when it gets an HTTP request.
- res.append(field [, value]): appends the specified value to the HTTP response header field.
- res.get(field): returns the HTTP response header specified by field.
- res.set(field [, value]): sets the response's HTTP header filed to value.

Response in Express App

- res.json([body]): send a JSON response that is the parameter converted to a JSON string using JSON.stringify().
- res.redirect([status,] path): redirects to the URL derived from the specified path.
- res.render(view [, locals][, callback]): renders a view and sends the rendered HTML string to the client.
- res.end(): ends the response process.

Response in Express App

- res.send([body]): sends the HTTP response, the body parameter can be a Buffer object, a String, an object, a Boolean or an array.
- res.sendFile(path [, options] [, fn]): transfers the file at the given path, sets the Content-Type respinse HTTP header filed based on the filename's extension.
- res.sendStatus(statusCode): sets the respponse HTTP status code and send its string representation as the response body.

Express framework

- Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI and a specific HTTP request method (GET, POST, ...).
- Route definition takes the following structure

```
app.method(PATH, HANDLER)
```

- Using all() method matches all HTTP request

```
app.all(PATH, HANDLER)
```

Express framework

- Returns an instance of a single route, which you can then use to handle HTTP verbs with optional middleware. Using app.route() to avoid duplicate route names.

```
var app = express()
app.route('/t').get(function(req, res, next) {

}).post(function(req, res, next) {

}).delete(function(req, res, next) {

})
```

Express framework

```
app.get('/', function(req, res) {
    res.send('get')
})
app.post('/', function(req, res) {
    res.send('post')
})
app.put('/', function(req, res) {
    res.send('put')
})
app.delete('/', function(req, res) {
    res.send('delete')
})
```

Express framework

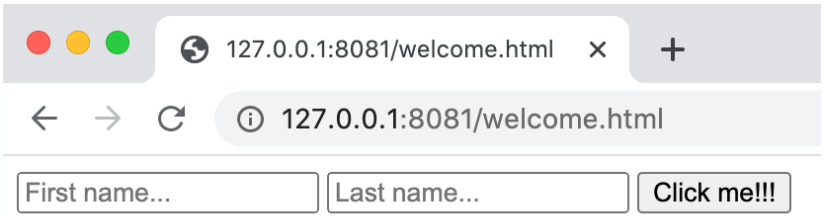
```
var express = require('express')
var app = express()

app.get('/', function(req, res) {
    res.send('Hello World')
})

var server = app.listen(8082, function() {
    var host = server.address().address
    var port = server.address().port

    console.info(`http://${host}:${port}`)
})
```

Get Request

```
<!DOCTYPE html>
<html>
  <body>
    <form method="get"
       action="http://127.0.0.1:8081/welcome">
      <input type="text" name="first_name"
             placeholder="First name..." />
      <input type="text" name="last_name"
             placeholder="Last name..." />
      <input type="submit"
             value="Click me!!!" />
    </form>
  </body>
</html>
```

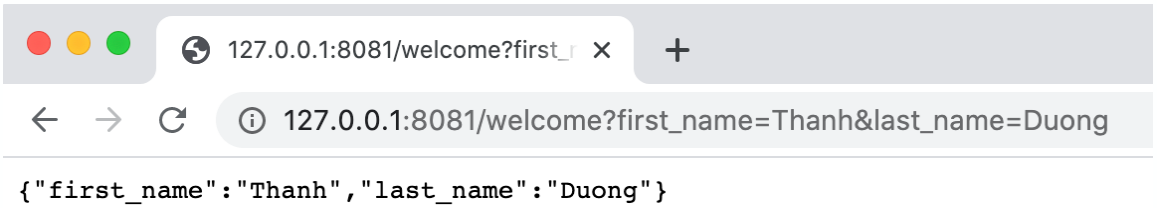
Get Request

```
var express = require('express')
var app = express()

app.get('/welcome.html', function(req, res) {
   res.sendFile(__dirname + '/welcome.html')
})

var server = app.listen(8081, function() {

})
```



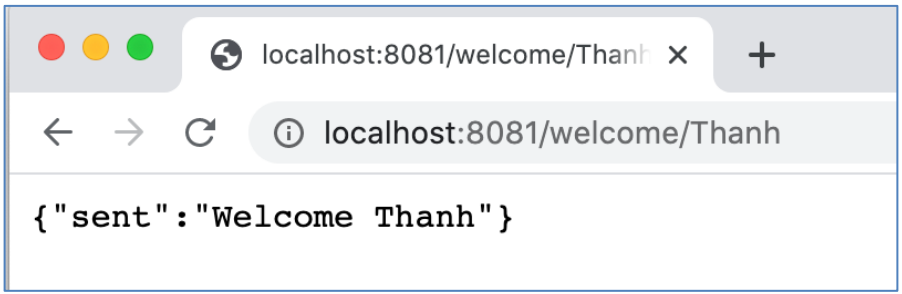
Get Request

```
app.get('/welcome', function(req, res) {
  data = {
      first_name: req.query.first_name,
      last_name: req.query.last_name
  }

  res.end(JSON.stringify(data))
})
```



Get Request

```
app.get('/welcome/:name', function(req, res) {
    res.end(JSON.stringify({
        'sent': `Welcome ${req.params.name}`
    }))
})
```

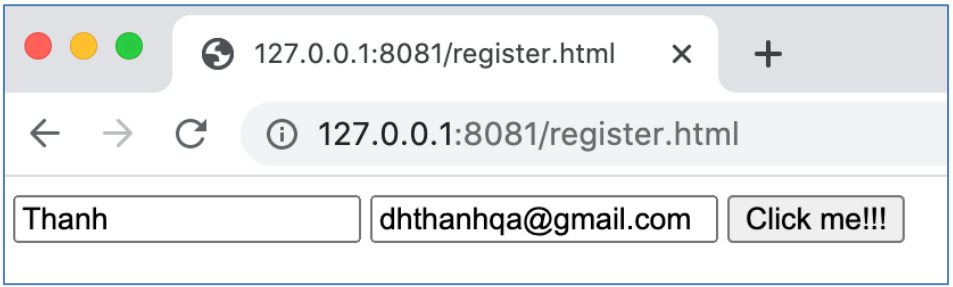


Post Request

```
<!DOCTYPE html>
<html>
  <body>
    <form method="post"
      action="http://127.0.0.1:8081/register">
       <input type="text" name="name"
              placeholder="Name..." />
       <input type="text" name="email"
              placeholder="Email..." />
       <input type="submit"
              value="Click me!!!" />
    </form>
  </body>
</html>
```

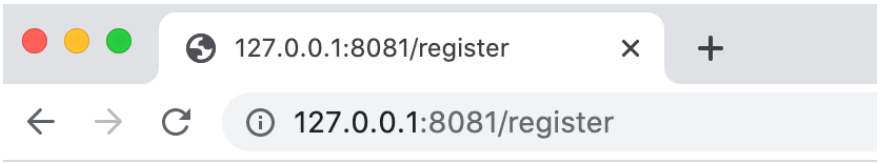
Post Request

```
var express = require('express')
var bodyParser = require('body-parser')
var app = express()
var urlParser
    = bodyParser.urlencoded({extended: false})

app.get('/register.html', function(req, res) {
    res.sendFile(__dirname + '/register.html')
})
```



Post Request

```
app.post('/register', urlParser,
          function(req, res) {
    data = {
        name: req.body.name,
        email: req.body.email
    }

    res.end(JSON.stringify(data))
})
```



Mounts the middleware

- Mounts the specified middleware function or functions at the specified path.

```
app.use([path, ] callback [, callback...])
```

- The middleware mounted without a path will be executed for every request to the app.

```
app.use(function(req, res, next) {
    console.info(Date.now())
    next()
})
```

Static Files

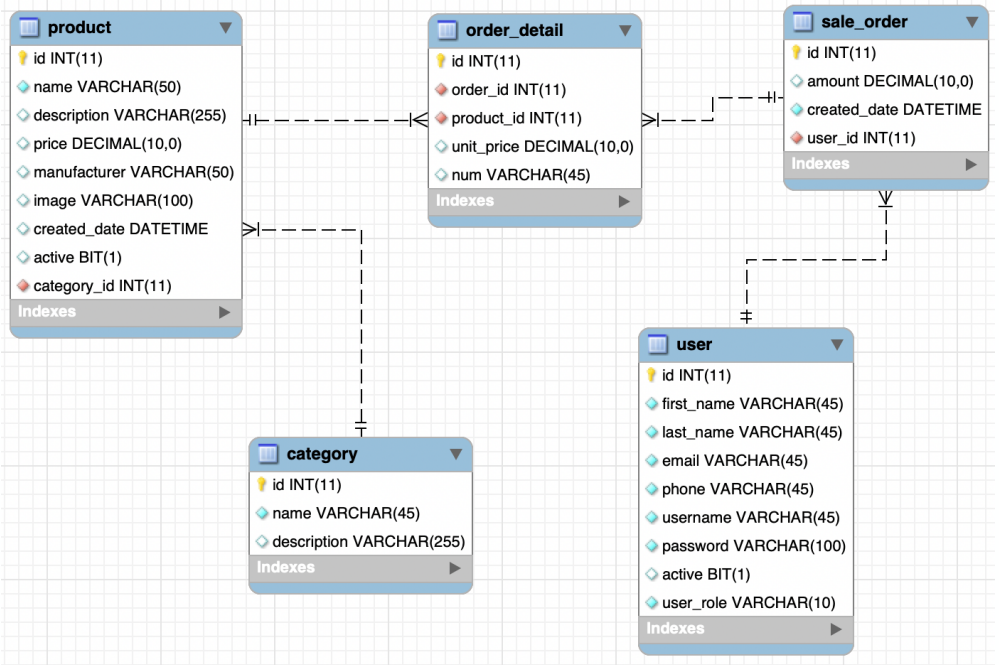
- Express provides a built-in middleware, namely express.static, to serve static files.
- For example: public >> images >> test.png

```
app.use(express.static("public"))
```

- We can access:
  - http://127.0.0.1:8080/images/test.png

Build Restful APIs

- Create a database has the below scheme

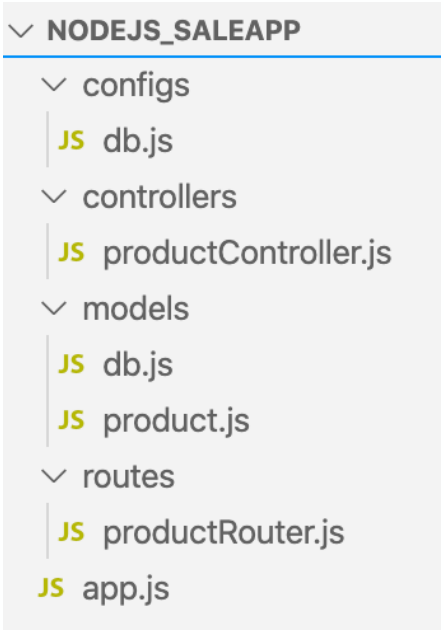


Build Restful APIs

- We will build the following APIs
  - /products?kw=&cateId=
  - /products/:id
  - /products/:id/edit
  - /products/:id/delete
  - /users/register
  - /orders
  - /orders/add

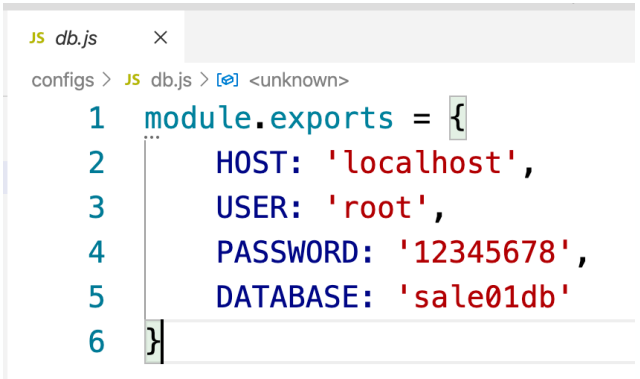
Build Restful APIs

- Create the project structure likes as follows:



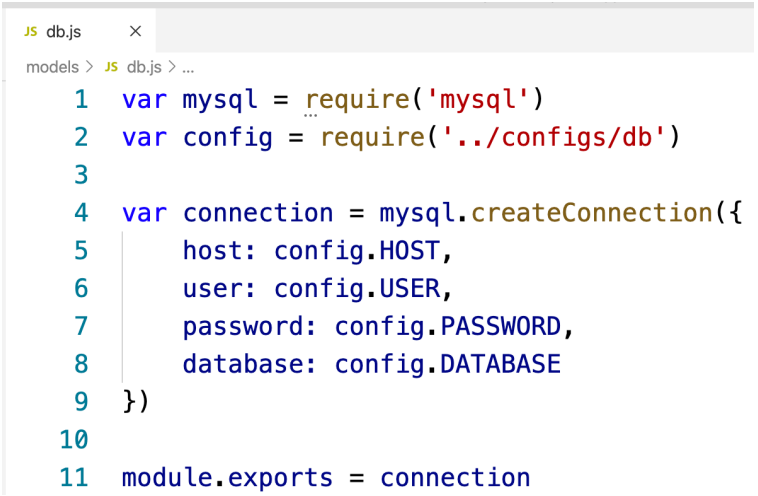
Build Restful APIs

- The configs/db.js contains the database config



Build Restful APIs

- The models folder contains the modules which interact with the database.
  - models/db.js



Build Restful APIs

- The models.product.js module

product.js

models > product.js > Product > getProductById

```
1  conn = require('./db')
2
3  class Product {
4      static getProducts = (result, kw=null) => {
5          let sql = "SELECT * FROM product"
6          if (kw != null)
7              sql += ` WHERE name like '%${kw}%'`
8
9          conn.query(sql, (err, products) => result(err, products))
10     }
11
12     static getProductById = (result, productId) => {
13         let sql = "SELECT * FROM product WHERE id=?"
14         conn.query(sql, [productId], (err, products) => {
15             if (err)
16                 result(err, null)
17             else
18                 result(err, products[0])
19         })
20     }
21 }
22
23 module.exports = Product
```

Build Restful APIs

- The controllers folder contains the actions which receive the HTTP request.
  - controllers/products.js

Build Restful APIs

- The routes folder contain the routers
  - routes/productRouter.js

productRouter.js

routes > productRouter.js > <unknown>

```
1  var router = require('express').Router()
2  var controller = require('../controllers/productController')
3
4  router.get('/', (req, res) => {
5      controller.getProducts(req, res)
6  })
7
8  router.get('/:id', (req, res) => {
9      controller.getProductById(req, res)
10 })
11
12 module.exports = router
```

Build Restful APIs

- The app.js module

app.js

```
1  var express = require('express')
2  var app = express()
3  var bodyParser = require('body-parser')
4  var productRouter = require('./routes/productRouter')
5  var port = 8081
6
7  app.use('/products', productRouter)
8  app.use(bodyParser.json()) // application/json
9  app.use(bodyParser.urlencoded({extended: true})) // application/x-www-form-urlencoded
10
11 app.listen(port, (err) => {
12     if (err)
13         console.error(err)
14     else
15         console.info(`http://127.0.0.1:${port}`)
16 })
```


Q&A