Full Stack Web Development: The Big Picture: Objectives and Outcomes

This lesson gives you a big picture view of the Full Stack Web Development and positions the current course in the overall structure of the specialization. This provides you the context for doing this course and sets the path forward in this course. The first video gives you an overview of full stack web development. You may skip this video if you have seen it already in the previous course. The second video gives you an overview of this course and the various topics to be covered in this course. The third video explains how to make the best use of the learning resources available in the course. You may skip this video if you have already seen it in the previous course. At the end of this lesson, you will be able to:

- Understand what is meant by full stack in the context of web development
- Distinguish between front-end, back-end and full stack web development
- Understand the position of this course in the context of this specialisation

✓ Complete

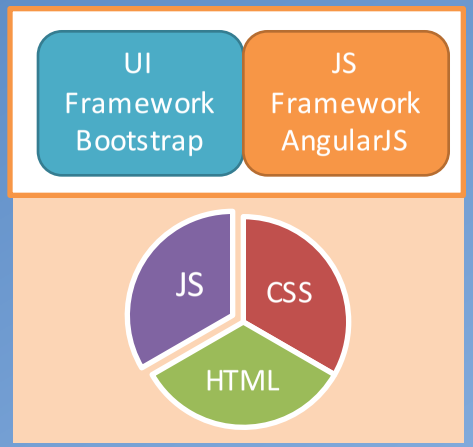
Server-side Development with NodeJS
Course Overview

Jogesh K. Muppala

THE DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

香港科技大學
THE HONG KONG UNIVERSITY OF
SCIENCE AND TECHNOLOGY

Full Stack Web Development



BaaS

Node.js Modules

Node.js

MongoDB

Presentation layer

Business Logic layer

Data Access layer

Course Outline

- Node.js and Node modules
- Express Framework
- MongoDB
- Backend as a Service (BaaS)

Module 1: Introduction to Server-side Development

- Full Stack Web Development: The Big Picture
- Introduction to Node.js and NPM
- Node Modules
- Node and HTTP
- Introduction to Express
- Assignment 1

Module 2: Data, Data, Where art Thou Data?

- Express Generator
- Introduction to MongoDB
- Node and MongoDB
- Mongoose ODM
- Assignment 2

Module 3: Halt! Who goes there?

- REST API with Express, MongoDB and Mongoose
- Basic Authentication
- Cookies, Tea and err ... Express Sessions
- User Authentication with Passport
- Assignment 3

Module 4: Backend as a Service (BaaS)

- Mongoose Population
- HTTPS and Secure Communication
- Backend as a Service (BaaS)
- Loopback, StrongLoop ARC and IBM Bluemix (Guest Lecture by Mr. Raymond Camden)
- Assignment 4

What is Full Stack Web Development?

Jogesh K. Muppala

THE DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

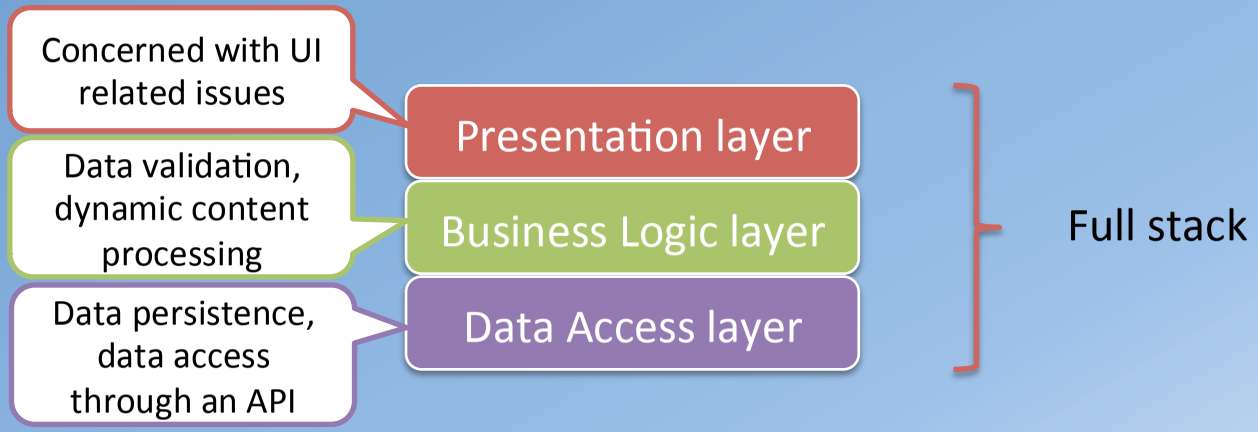
香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Front end and Back end

- Front end / Client-side
  - HTML, CSS and Javascript
- Back end / Server-side
  - Various technologies and approaches
  - PHP, Java, ASP.NET, Ruby, Python

https://en.wikipedia.org/wiki/Front_and_back_ends

Three Tier Architecture



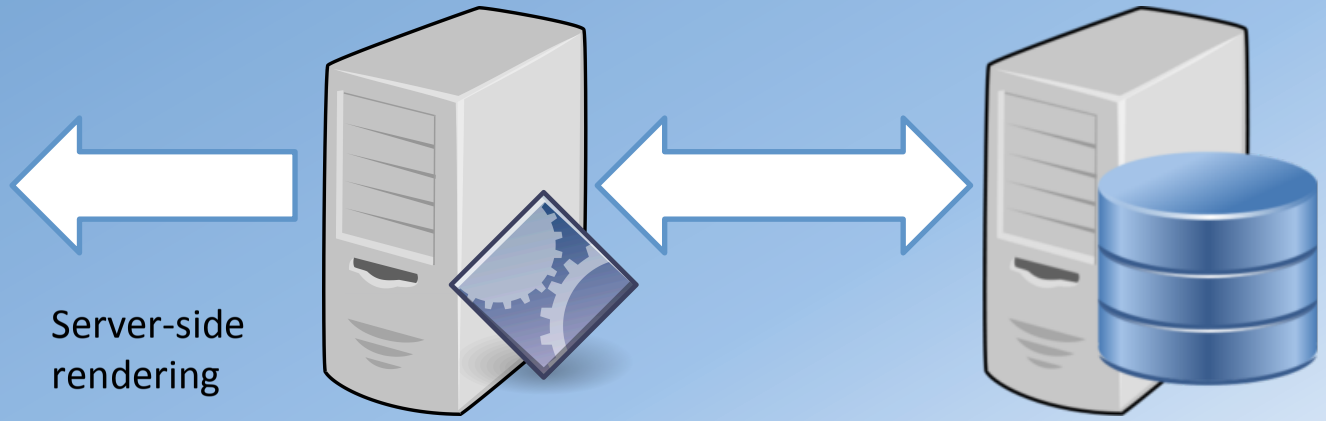
https://en.wikipedia.org/wiki/Multitier_architecture#Three-tier_architecture
http://www.tonymarston.net/php-mysql/3-tier-architecture.html

Traditional Web Development

HTML, CSS, JS

Ruby, Python, Java, C++, PHP

DBMS



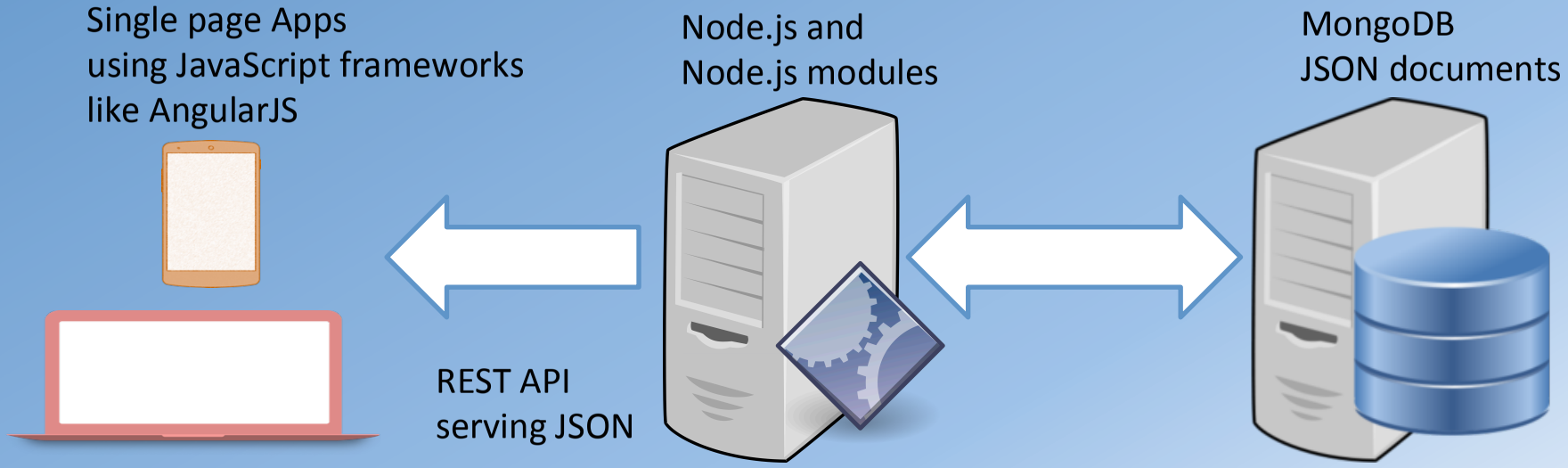
Server-side rendering

Presentation layer

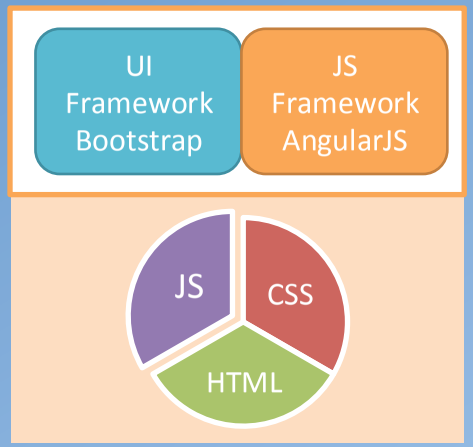
Business Logic layer

Data Access layer

Full Stack JavaScript Development



Full Stack Web Development



BaaS

Node.js Modules

Node.js

MongoDB

Presentation layer

Business Logic layer

Data Access layer

Full Stack Web Development

- Course 1: HTML, CSS and JavaScript
- Course 2: Front-End Web UI Frameworks and Tools
  - Bootstrap
  - Bower, Grunt, Gulp, Yo, Yeoman

Full Stack Web Development

- Course 3: Front-end JavaScript Frameworks: AngularJS
- Course 4: Multiplatform Mobile Application Development using HTML, CSS and JavaScript
  - Cordova
  - Ionic Framework

Full Stack Web Development

- Course 5: Server-side Development with NodeJS
  - Node JS
  - NodeJS modules
  - BaaS
- Course 6: Full Stack Web Development Capstone Project

Introduction to Node.js and NPM: Objectives and Outcomes

In this lesson you will learn the basics of Node.js and NPM. Thereafter, you will install Node.js and NPM on your machine so that you can start writing simple Node applications. At the end of this lesson, you should be able to:

- Download and install Node.js and NPM on your machine
- Verify that the installation was successful and your machine is ready for using Node.js and NPM.

✓ Complete

Node.js and NPM

Jogesh K. Muppala

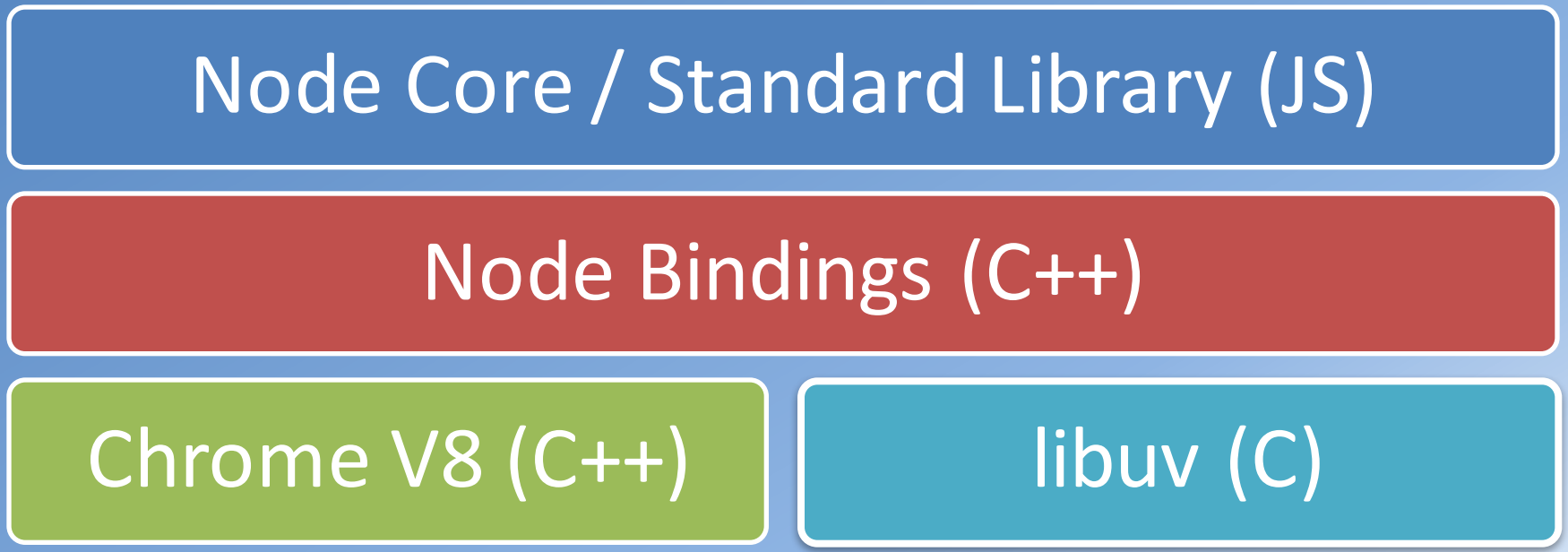
THE DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

香港科技大學
THE HONG KONG UNIVERSITY OF
SCIENCE AND TECHNOLOGY

What is Node.js?

- JavaScript runtime built on Chrome V8 JavaScript Engine
- Uses an event-driven, non-blocking I/O model
  - Makes it lightweight and efficient

Node Architecture



Node.js Use Cases

- Utilities written in JavaScript for web development:
  - Bower, Grunt, Gulp, Yeoman etc.
- Server-side Development
  - Web server, Business logic, Database access

Node Package Manager

- Node package manager (NPM): manages ecosystem of node modules / packages
- A package contains:
  - JS files
  - package.json (manifest)

Exercise: Setting up Node.js and NPM

- Complete the set up of Node.js and NPM on your machine
- Understand the basics of Node.js and NPM

Exercise (Instructions): Setting up Node.js and NPM

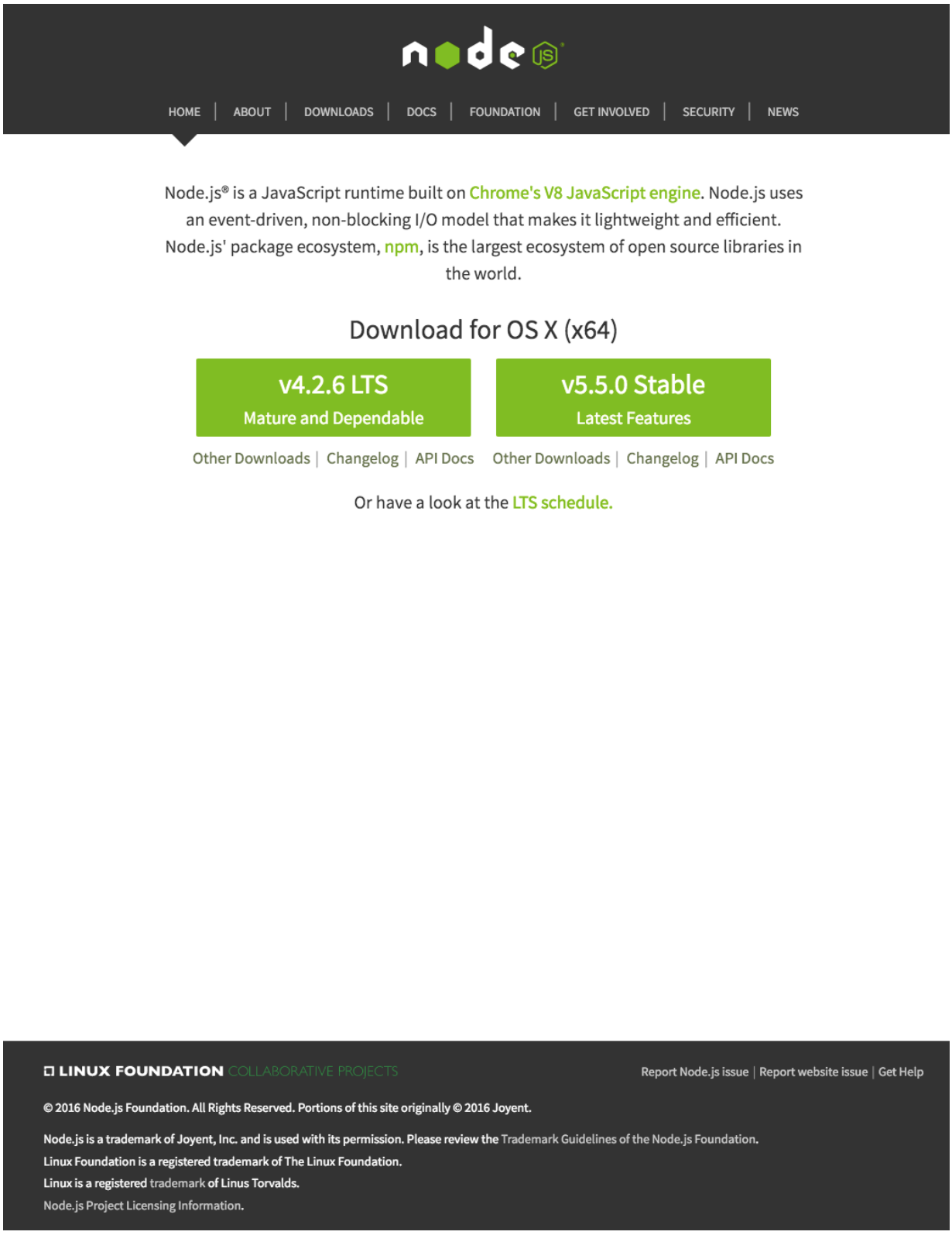
Objectives and Outcomes

In this exercise, you will learn to set up the Node.js environment, a popular Javascript based server framework, and node package manager (NPM) on your machine. To learn more about NodeJS, you can visit https://nodejs.org. For this course, you just need to install Node.js on your machine and make use of it for running some front-end tools. You will learn more about the server-side support through Node.js in a subsequent course. At the end of this exercise, you will be able to:

- Complete the set up of Node.js and NPM on your machine
- Understand the basics of Node.js and NPM

Installing Node

- To install Node on your machine, go to https://nodejs.org and click on the Download button. Depending on your computer's platform (Windows, MacOS or Linux), the appropriate installation package is downloaded.As an example, on a Mac, you will see the following web page. Click on the Download button. Follow along the instructions to install Node on your machine. (Note: Now Node gives you the option of installing a mature and dependable version and a more newer stable version. You can choose to install the mature and dependable version. I will continue to use this version in the course. You can choose to install the newer stable version if you wish. You may not see any perceptible differences between the two as users).



Note: On Windows machines, you may also need to install Git on your machine if you don't have it already installed. Some of the Node based tools that we use later will need Git to be installed. You can download the installer from here.

Note: On Windows machines, you may need to configure your PATH environmental variable in case you forgot to turn on the add to PATH during the installation steps.

Verifying the Node Installation

- Open a terminal window on your machine. If you are using a Windows machine, open a cmd window or PowerShell window with **admin** privileges.
- To ensure that your NodeJS setup is working correctly, type the following at the command prompt to check for the version of **Node** and **NPM**

```
1        node -v
2
3        npm -v
```

Conclusions

At the end of this exercise, your machine is now ready with the Node installed for further development. We will examine web development tools next.

✓ Complete

Node Modules: Objectives and Outcomes

In this lesson you will learn to write Node applications. You will learn about Node modules and how you can make use of them within your Node applications. At the end of this lesson, you will be able to:

- Write simple Node applications and run them using Node
- Develop Node modules and use them within your Node applications
- Learn about using callbacks and handling errors within your Node application

✓ Complete

Node Modules

Jogesh K. Muppala

THE DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

香港科技大學
THE HONG KONG UNIVERSITY OF
SCIENCE AND TECHNOLOGY

JavaScript Modules

- JavaScript does not define a standard library
- CommonJS API fills this gap by defining APIs for common application needs
  - It defines a module format
  - Node follows the CommonJS module specification

Node Modules

- Each file in Node is its own module
- The *module* variable gives access to the current module definition in a file
- The *module.exports* variable determines the export from the current module
- The *require* function is used to import a module

Module Example

- *rectangle* Module:

```
module.exports = function () {
        return {
                perimeter: function(x,y) { return (2*(x+y)); },
                area: function(x,y) { return ( x*y) )); }
        };
}
```

- Using this module:

```
var rect = require('./rectangle');
```

Module Example: Alternative

- rectangle Module:

```
exports.perimeter = function (x, y) {
        return (2*(x+y));
}

exports.area = function (x, y) {
        return (x*y);
}
```

  - *exports* is alias for *module.exports*

Exercise: Node Modules

- Writing a simple Node application
- Understanding Node modules

Node Modules: Further Details

Jogesh K. Muppala

THE DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

香港科技大學
THE HONG KONG UNIVERSITY OF
SCIENCE AND TECHNOLOGY

Node Modules

- File-based Modules
- Core Modules
  - Part of core Node
  - Examples: path, fs, os, util, . . .
- External Node modules
  - Third-party modules
  - Installed using NPM
  - node_modules folder in your Node application

Using Node Modules

- Include them using require function
- File-based modules:
  - require('./module_name')
  - Specify the relative path to the file
- Core and External modules:
  - require('module_name')
  - Looks for external modules in:
    - ./node_modules, ../node_modules, ../../node_modules, . . .
    - Up the folder hierarchy until the module is found

A Brief Tour of a Node Module

- Examine package.json file
- Semantic Versioning
  - <Major Version>.<Minor Version>.<Patch>
  - npm install can specify the acceptable package version:
    - Exact: npm install express@4.0.0
    - Patch acceptable: npm install express@”~4.0.0”
    - Minor version acceptable: npm install express@”^4.0.0”

Exercise: Node Modules: Callbacks and Error Handling

- Using Callbacks and error handling in Node applications
- Using external Node modules

Node Modules: Callbacks and Error Handling

Jogesh K. Muppala

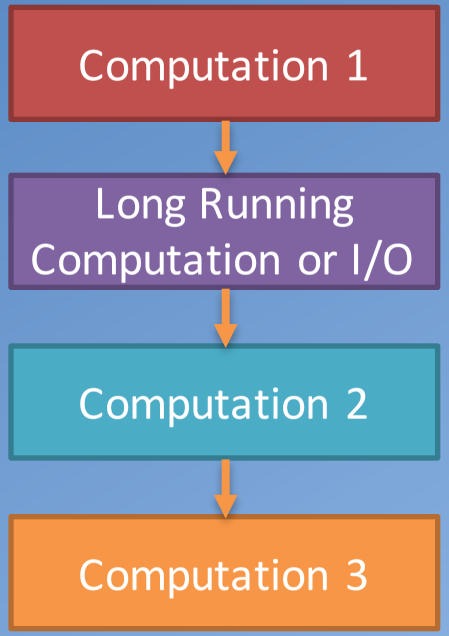
THE DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

香港科技大學
THE HONG KONG UNIVERSITY OF
SCIENCE AND TECHNOLOGY

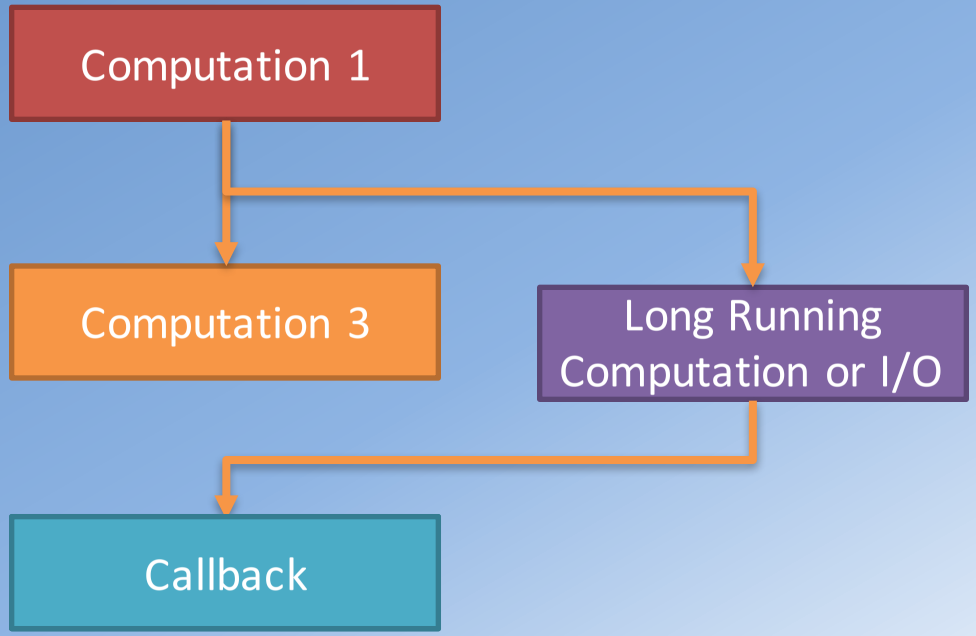
Two Salient Features of JavaScript

- First-class functions: A function can be treated the same way as any other variable
- Closures:
  - A function defined inside another function has access to all the variables declared in the outer function (outer scope)
  - The inner function will continue to have access to the variables from the outer scope even after the outer function has returned

Asynchronous Programming

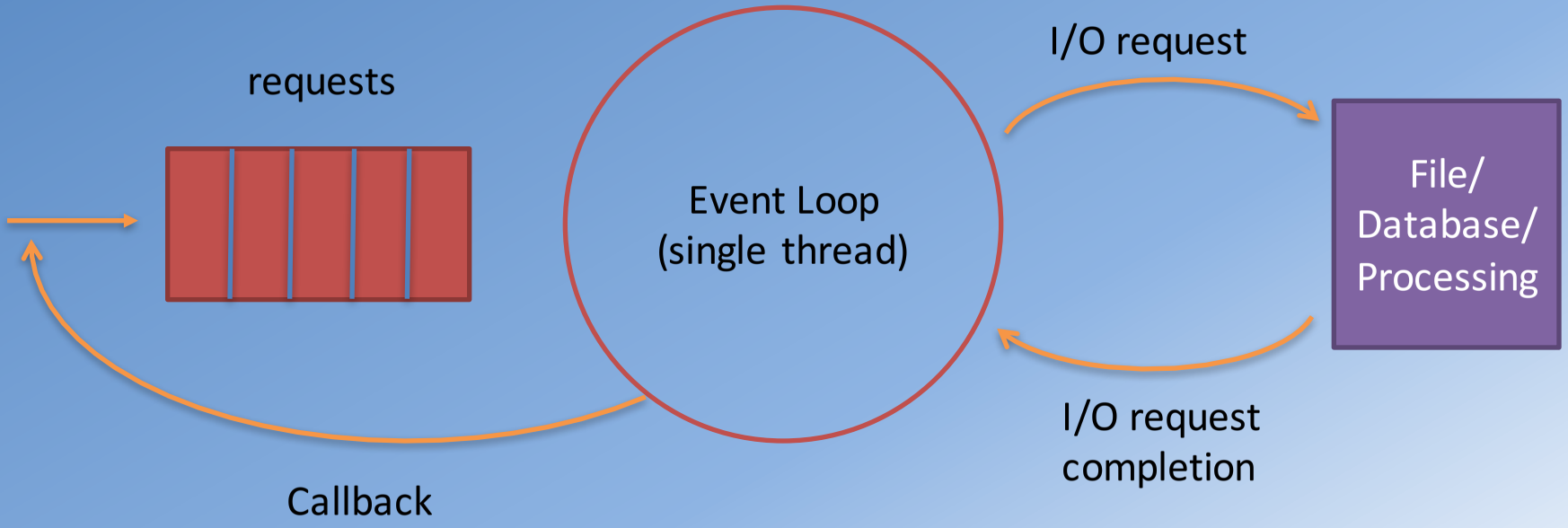


Synchronous Programming



Asynchronous Programming

Node, Async I/O and Callbacks



Callbacks and Error Handling

- rectangle module:

```
module.exports = function(x,y,callback) {
 try {
  if (x < 0 || y < 0) {
     throw new Error("Rectangle dimensions should be greater than zero: l = " + x + ", and b = " + y);
  }
  else
     callback(null, {
        perimeter: function () { return (2*(x+y)); },
        area:function () { return (x*y); }
     });
 }
 catch (error) { callback(error,null); }
}
```

Callbacks and Error Handling

- Calling the function:

```
rect(l,b, function(err,rectangle) {
        if (err) {
                console.log(err);
        }
        else {
                . . .
        }
});
```

Exercise: Node Modules: Callbacks and Error Handling

- Using Callbacks and error handling in Node applications
- Using external Node modules

Exercise (Instructions): Understanding Node Modules

Objectives and Outcomes

In this exercise, you will learn about writing Node applications using JavaScript and also learn about the basics of Node modules. At the end of this exercise, you will be able to:

- Write a simple Node application in JavaScript.
- Understand the basics of Node modules and write simple file-based Node modules

Starting a Node Application

- Create a folder named *node-examples* at a convenient location on your computer, and move to that folder.
- Create a file named *simplerect.js* and add the following code to this file:

```
var rect = {
  perimeter: function (x, y) {
            return (2*(x+y));
  },
  area: function (x, y) {
             return (x*y);
  }
};

function solveRect(l,b) {
    console.log("Solving for rectangle with l = " + l + " and b = " + b);

    if (l < 0 || b < 0) {
        console.log("Rectangle dimensions should be greater than zero:  l = "
               + l + ",  and b = " + b);
    }
    else {
  console.log("The area of a rectangle of dimensions length = "
               + l + " and breadth = " + b + " is " + rect.area(l,b));
  console.log("The perimeter of a rectangle of dimensions length = "
               + l + " and breadth = " + b + " is " + rect.perimeter(l,b));
    }
}

solveRect(2,4);
solveRect(3,5);
solveRect(-3,5);
```

- To run the Node application, type the following at the prompt:

```
node simplerect
```

A Simple Node Module

- Now, create a file named *rectangle-1.js*, and add the following code to it:

```
exports.perimeter = function (x, y) {
        return (2*(x+y));
}

exports.area = function (x, y) {
        return (x*y);
}
```

- Then, create another file named *solve-1.js* and add the following code to it:

```
var rect = require('./rectangle-1');

function solveRect(l,b) {
    console.log("Solving for rectangle with l = " + l + " and b = " + b);

    if (l < 0 || b < 0) {
        console.log("Rectangle dimensions should be greater than zero:  l = "
               + l + ",  and b = " + b);
    }
    else {
   console.log("The area of a rectangle of dimensions length = "
               + l + " and breadth = " + b + " is " + rect.area(l,b));
   console.log("The perimeter of a rectangle of dimensions length = "
               + l + " and breadth = " + b + " is " + rect.perimeter(l,b));
    }
}

solveRect(2,4);
solveRect(3,5);
solveRect(-3,5);
```

- To run the Node application, type the following at the prompt:

```
node solve-1
```

Conclusions

In this exercise, you learnt about writing simple Node applications in JavaScript. Thereafter you learnt about writing a basic Node module and use it within your Node application.

✓ Complete

Exercise (Instructions):Node Modules: Callbacks and Error Handling

Objectives and Outcomes

In this exercise, you will learn about callbacks and error handling in Node applications. At the end of this exercise, you will be able to:

- Using Callbacks and error handling in Node applications
- Using external Node modules

Using Callbacks and Error Handling

- Create a file named *rectangle-2.js* and add the following code to this file:

```
module.exports = function(x,y,callback) {
  try {
    if (x < 0 || y < 0) {
        throw new Error("Rectangle dimensions should be greater than zero: l = "
                        + x + ", and b = " + y);
    }
    else
         callback(null, {
            perimeter: function () {
               return (2*(x+y));
      },
            area:function () {
                return (x*y);
      }
    });
  }
  catch (error) {
        callback(error,null);
  }
}
```

- Then, create a file named *solve-2.js* and include the following code in there:

```
var rect = require('./rectangle-2');

function solveRect(l,b) {
    console.log("Solving for rectangle with l = "
                + l + " and b = " + b);
    rect(l,b, function(err,rectangle) {
        if (err) {
      console.log(err);
  }
  else {
      console.log("The area of a rectangle of dimensions length = "
                + l + " and breadth = " + b + " is " + rectangle.area());
            console.log("The perimeter of a rectangle of dimensions length = "
                + l + " and breadth = " + b + " is " + rectangle.perimeter());
  }
    });
};

solveRect(2,4);
solveRect(3,5);
solveRect(-3,5);
```

- To run the Node application, type the following at the prompt:

```
node solve-2
```

Using yargs External Node module

- Install the *yargs* Node module by typing the following at the prompt:

```
npm install yargs --save
```

- Then, create another file named *solve-3.js* and add the following code to it:

```
var argv = require('yargs')
    .usage('Usage: node $0 --l=[num] --b=[num]')
    .demand(['l','b'])
    .argv;

var rect = require('./rectangle-2');

function solveRect(l,b) {
    console.log("Solving for rectangle with l = "
                + l + " and b = " + b);
    rect(l,b, function(err,rectangle) {
  if (err) {
      console.log(err);
  }
  else {
            console.log("The area of a rectangle of dimensions length = "
                + l + " and breadth = " + b + " is " + rectangle.area());
      console.log("The perimeter of a rectangle of dimensions length = "
                + l + " and breadth = " + b + " is " + rectangle.perimeter());
  }
    });
};

solveRect(argv.l,argv.b);
```

- To run the Node application, type the following at the prompt:

```
node solve-3
```

Conclusions

In this exercise, you learnt about using Callbacks and error handling in Node applications. In addition you learnt about using external Node modules.

✓ Complete

Node and HTTP: Objectives and Outcomes

In this lesson you will learn more about the Node HTTP core module. You will create a simple HTTP server using the Node HTTP module and serve HTML files from a public folder. Along the way you will learn about the fs and path Node core modules. At the end of this lesson, you will be able to:

- Create a simple HTTP server using the Node HTTP core module
- Create a web server to serve static HTML files from a folder

✓ Complete

The HTTP Protocol

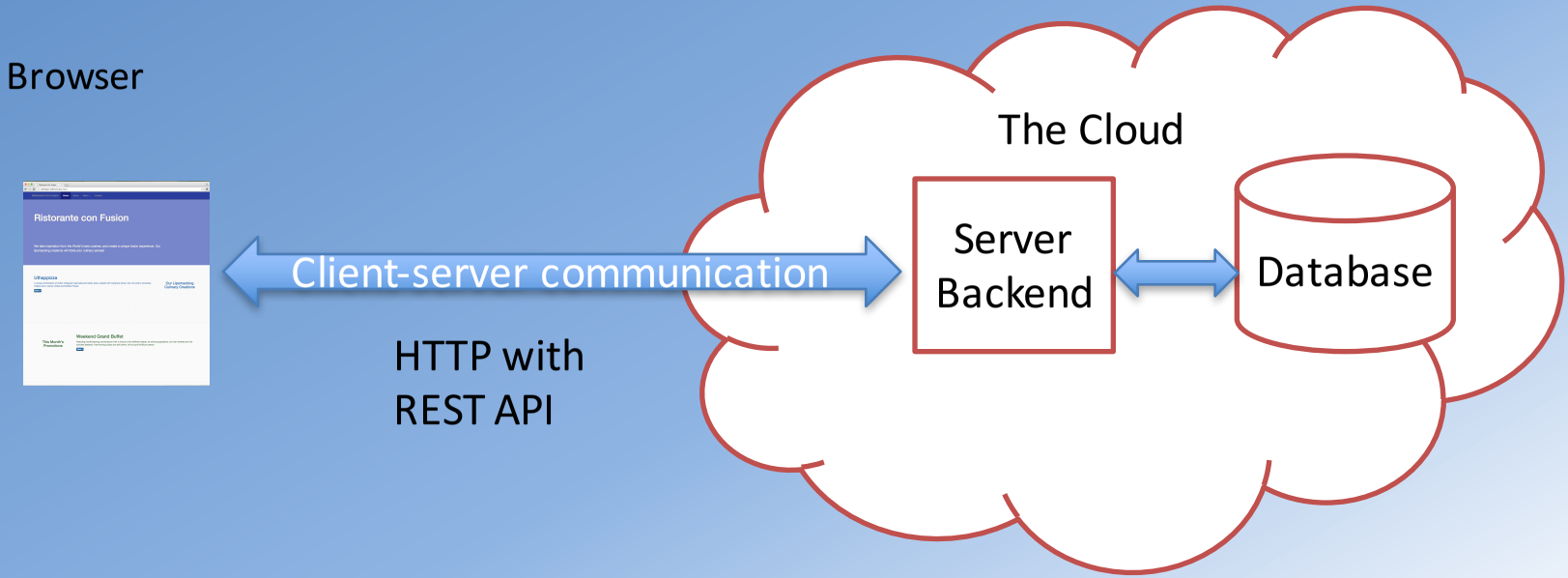
Jogesh K. Muppala

THE DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

香港科技大學
THE HONG KONG UNIVERSITY OF
SCIENCE AND TECHNOLOGY

Client and Server

- Web applications are not stand-alone
- Many of them have a “Cloud” backend



The Networking Alphabet Soup

Client-Server Communication

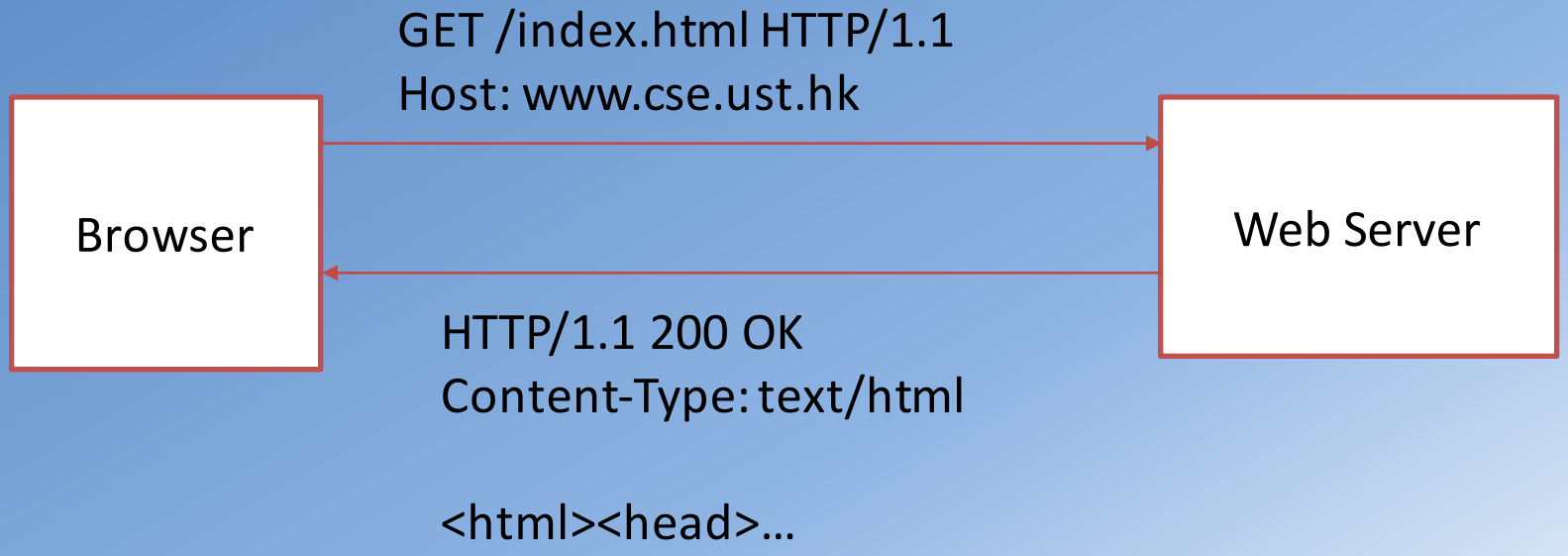
- Network operations cause unexpected delays
- You need to write applications recognizing the asynchronous nature of communication
  - Data is not instantaneously available

Hypertext Transfer Protocol (HTTP)

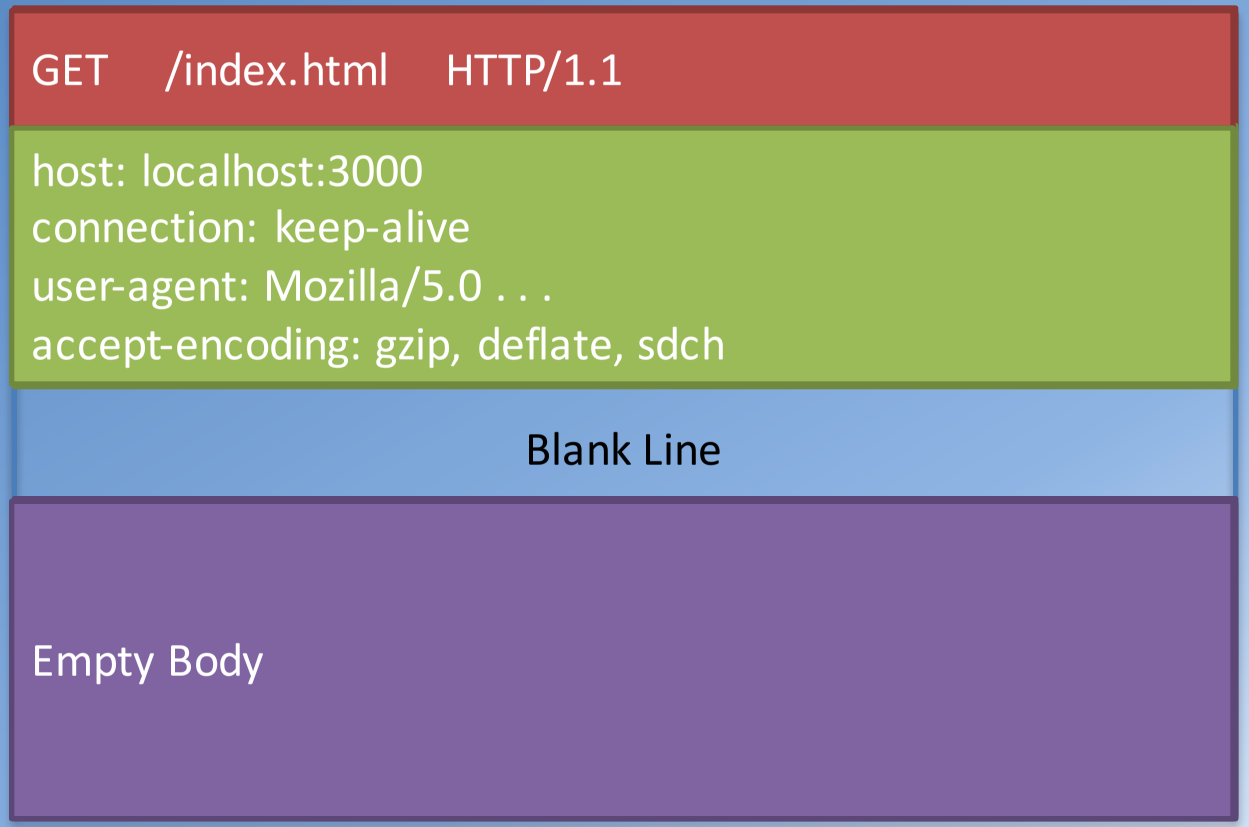
- A client-server communications protocol
- Allows retrieving inter-linked text documents (hypertext)
  - World Wide Web.

- HTTP Verbs
  - HEAD
  - GET
  - POST
  - PUT
  - DELETE
  - TRACE
  - OPTIONS
  - CONNECT

Hypertext Transfer Protocol (HTTP)



HTTP Request Message



HTTP Response Message

HTTP/1.1 200 OK

Connection: keep-alive
Content-Type: text/html
Date: Sun, 21 Feb 2016 06:01:43 GMT
Transfer-Encoding: chunked

Blank Line

<html><title>This is index.html</title><body><h1>Index.html</h1> This is the contents of this file </body></html>

HTTP Response Codes (Main ones)

Code	Meaning
200	OK
201	Created
301	Moved Permanently
304	Not Modified
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
422	Unprocessable Entry
500	Internal Server Error
505	HTTP Version Not Supported

HTTP Response

- Server may send back data in a specific format:
  - eXtensible Markup Language (XML)
  - Javascript Object Notation (JSON)

Node and the HTTP Module

Jogesh K. Muppala

THE DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

香港科技大學
THE HONG KONG UNIVERSITY OF
SCIENCE AND TECHNOLOGY

Node HTTP Module

- Core networking module supporting a high-performance foundation for a HTTP stack
- Using the module:

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

- Creating a server:

  ```
  var server = http.createServer(function(req, res){ . . . });
  ```

- Starting the server:

  ```
  server.listen(port, . . . );
  ```

Node HTTP Module

- Incoming request message information available through the first parameter “req”
  - req.headers, req.body, . . .
- Response message is constructed on the second parameter “res”
  - res.setHeader("Content-Type", "text/html");
  - res.statusCode = 200;
  - res.writeHead(200, { 'Content-Type': 'text/html' });
  - res.write(’Hello World!');
  - res.end('<html><body><h1>Hello World</h1></body></html>');

Node path Module

- Using path Module:

```
var path = require('path');
```

- Some example path methods:

```
path.resolve('./public'+fileUrl);
path.extname(filePath);
```

Node fs Module

- Use fs module in your application

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

- Some example fs methods:

```
fs.exists(filePath, function(exists) { . . . } );
fs.createReadStream(filePath).pipe(res);
```

Exercise: Node and the HTTP Module

- Constructing a simple HTTP Server
- Constructs a server that returns html files from a folder

Exercise (Instructions): Node and the HTTP Module

Objectives and Outcomes

In this exercise, you will explore three core Node modules: HTTP, fs and path. At the end of this exercise, you will be able to:

- Implement a simple HTTP Server
- Implement a server that returns html files from a folder

A Simple HTTP Server

- Create a folder named *node-http* at a convenient location and move into the folder.
- In the *node-http* folder, create a subfolder named *public*.
- Create a file named *server-1.js* and add the following code to it:

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

var hostname = 'localhost';
var port = 3000;

var server = http.createServer(function(req, res){
  console.log(req.headers);
    res.writeHead(200, { 'Content-Type': 'text/html' });
  res.end('<html><body><h1>Hello World</h1></body></html>');
  })
server.listen(port, hostname, function(){
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

- Start the server by typing the following at the prompt:

```
    node server-1
```

- Then you can type http://localhost:3000 in your browser address bar and see the result.
- You can also use postman chrome extension to send requests to the server and see the response.

Serving HTML Files

- In the *public* folder, create a file named *index.html* and add the following code to it:

```
<html>
<title>This is index.html</title>
<body>
<h1>Index.html</h1>
<p>This is the contents of this file</p>
</body>
</html>
```

- Similarly create an aboutus.html file and add the following code to it:

```
<html>
<title>This is aboutus.html</title><body>
<h1>Aboutus.html</h1><p>This is the contents of the aboutus.html file</p></body>
</html>
```

- Then create a file named *server-2.js* and add the following code to it:

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

var hostname = 'localhost';
var port = 3000;

var server = http.createServer(function(req, res){
  console.log('Request for ' + req.url + ' by method ' + req.method);
  if (req.method == 'GET') {
    var fileUrl;
        if (req.url == '/') fileUrl = '/index.html';
    else fileUrl = req.url;
        var filePath = path.resolve('./public'+fileUrl);
        var fileExt = path.extname(filePath);
        if (fileExt == '.html') {
       fs.exists(filePath, function(exists) {                 if (!exists) {
          res.writeHead(404, { 'Content-Type': 'text/html' });
          res.end('<html><body><h1>Error 404: ' + fileUrl +
                        ' not found</h1></body></html>');
          return;
          }
                    res.writeHead(200, { 'Content-Type': 'text/html' });
          fs.createReadStream(filePath).pipe(res);
              });
    }
    else {

        res.writeHead(404, { 'Content-Type': 'text/html' });
        res.end('<html><body><h1>Error 404: ' + fileUrl +
                ' not a HTML file</h1></body></html>');
    }
  }
  else {
        res.writeHead(404, { 'Content-Type': 'text/html' });
        res.end('<html><body><h1>Error 404: ' + req.method +
                ' not supported</h1></body></html>');
  }
})

server.listen(port, hostname, function(){
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

- Start the server, and send various requests to it and see the corresponding response.

Conclusions

In this exercise you learnt about using the Node HTTP module to implement a HTTP server.

✓ Complete

Introduction to Express: Objectives and Outcomes

In this lesson you will learn about the Express framework that enables implementing and deploying powerful web servers based on Node. At the end of this lesson, you will be able to:

- Implement a web server using the Express framework
- Develop a web server that supports a REST API
- Use Express router to implement support for the REST API

✓ Complete

Introduction to Express

Jogesh K. Muppala

THE DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

香港科技大學
THE HONG KONG UNIVERSITY OF
SCIENCE AND TECHNOLOGY

What is Express

- Express: Fast, unopinionated, minimalist web framework for Node.js (from expressjs.com)
- Web application framework that provides a robust set of features
- Many third-party *middleware* to extend functionality
- Installing Express:
  - In your project folder do: npm install express --save

Express Hello World Example

```
var express = require('express'),
    http = require('http');

var hostname = 'localhost';
var port = 3000;

var app = express();

app.use(function  (req, res, next) {
    res.writeHead(200, { 'Content-Type':  'text/html' });
    res.end('<html><body><h1>Hello  World</h1></body></html>');
});

var server = http.createServer(app);

server.listen(port, hostname, function(){
    console.log(`Server running at http://${hostname}:${port}/`);
});
```

Express Middleware

- Middleware provide a lot of plug-in functionality that can be used within your Express application
- Example: *morgan* for logging

```
var morgan = require('morgan');
app.use(morgan('dev'));
```

- Serving static web resources:

```
app.use(express.static(__dirname + '/public'));
```

  - Note: __filename and __dirname give you the full path to the file and directory for the current module

Serving Static Content

```
var express = require('express');
var morgan = require('morgan');

var hostname = 'localhost';
var port = 3000;

var app = express();

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

app.use(express.static(__dirname + '/public'));

app.listen(port, hostname, function(){
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Exercise: Introduction to Express

- Implement a simple web server
- Implement a web server that serves static content

Express Router

Jogesh K. Muppala

THE DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

香港科技大學
THE HONG KONG UNIVERSITY OF
SCIENCE AND TECHNOLOGY

Express Application Routes

- We examined REST in the previous lecture
- Identify an end point with a URI
- Apply the verb on the URI
- Express supports this through app.all, app.get, app.post, app.put, app.delete methods

Express Application Routes

- Application Routes:

```
app.all('/dishes', function(req,res,next) { . . . });
app.get('/dishes', function(req,res,next) { . . . });
app.post('/dishes', function(req,res,next) { . . . });
app.put('/dishes', function(req,res,next) { . . . });
app.delete('/dishes', function(req,res,next) { . . . });
```

Routes with Parameters

- Example:

```
app.get('/dishes/:dishId', function(req,res,next){
        res.end('Will send details of the dish: ’
            + req.params.dishId +' to you!')
});
```

Body Parser

- Middleware to parse the body of the message
- Using Body Parser:

```
var bodyParser = require('body-parser');
app.use(bodyParser.json()); // parse the JSON in the body
```

- Parses the body of the message and populates the *req.body* property

Express Router

- Express Router creates a mini-Express application:

```
var dishRouter = express.Router();
dishRouter.use(bodyParser.json());

dishRouter.route('/')
    .all(. . .);
    .get(. . .);
    . . .
```

Exercise: Introduction to Express Part 2

- Use application routes to support REST API
- Use the Express Router to support REST API

Brief
Representational State Transfer (REST)

Jogesh K. Muppala

THE DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

香港科技大學
THE HONG KONG UNIVERSITY OF
SCIENCE AND TECHNOLOGY

Web Services

- A system designed to support interoperability of systems connected over a network
  - Service oriented architecture (SOA)
  - A standardized way of integrating web-based applications using open standards operating over the Internet
- Two common approaches used in practice:
  - SOAP (Simple Object Access Protocol) based services
    - Uses WSDL (Web Services Description Language)
    - XML based
  - REST (Representational State Transfer)
    - Use Web standards
    - Exchange of data using either XML or JSON
    - Simpler compared to SOAP, WSDL etc.

Representational State Transfer (REST)

- A style of software architecture for distributed hypermedia systems such as the World Wide Web.
- Introduced in the doctoral dissertation of Roy Fielding
  - One of the principal authors of the HTTP specification.
- A collection of network architecture principles which outline how resources are defined and addressed

Representational State Transfer (REST)

- Four basic design principles:
  - Use HTTP methods explicitly
  - Be stateless
  - Expose directory structure-like URIs
  - Transfer using XML, JavaScript Object Notation (JSON), or both

REST and HTTP

- The motivation for REST was to capture the characteristics of the Web that made the Web successful
  - URI (Uniform Resource Indicator) Addressable resources
  - HTTP Protocol
  - Make a Request – Receive Response – Display Response
- Exploits the use of the HTTP protocol beyond HTTP POST and HTTP GET
  - HTTP PUT, HTTP DELETE
  - Preserve Idempotence

REST Concepts

Nouns (Resources)
unconstrained
i.e., http://www.conFusion.food/dishes/123

Verbs
constrained
i.e., GET, PUT, POST, DELETE

Representations
constrained
i.e., XML, JSON

Resources

- The key abstraction of information in REST is a resource.
- A resource is a conceptual mapping to a set of entities
  - Any information that can be named can be a resource: a document or image, a temporal service (e.g. "today's weather in Hong Kong"), a collection of other resources, a non-virtual object (e.g. a person), and so on
- Represented with a global identifier (URI in HTTP)
  - http://www.conFusion.food/dishes/123

Naming Resources

- REST uses URI to identify resources
  - http://www.conFusion.food/dishes/
  - http://www.conFusion.food/dishes/123
  - http://www.conFusion.food/promotions/
  - http://www.conFusion.food/leadership/
  - http://www.conFusion.food/leadership/456
- As you traverse the path from more generic to more specific, you are navigating the data
- Directory structure to identify resources

Verbs

- Represent the actions to be performed on resources
  - Corresponding to the CRUD operations
- HTTP GET ←→ READ
- HTTP POST ←→ CREATE
- HTTP PUT ←→ UPDATE
- HTTP DELETE ←→ DELETE

HTTP GET

- Used by clients to request for information
- Issuing a GET request transfers the data from the server to the client in some representation (XML, JSON)
  - GET http://www.conFusion.food/dishes/
    - Retrieve all dishes
  - GET http://www.conFusion.food/dishes/452
    - Retrieve information about the specific dish

HTTP PUT, HTTP POST, HTTP DELETE

- HTTP POST creates a resource
  - POST http://www.conFusion.food/feedback/
    - Content: {first name, last name, email, comment etc.}
    - Creates a new feedback with given properties
- HTTP PUT updates a resource
  - PUT http://www.conFusion.food/dishes/123
    - Content: {name, image, description, comments …}
    - Updates the information about the dish, e.g., comments
- HTTP DELETE removes the resource identified by the URI
  - DELETE http://www.conFusion.food/dishes/456
    - Delete the specified dish

Representations

- How data is represented or returned to the client for presentation
- Two main formats:
  - JavaScript Object Notation (JSON)
  - XML
- It is common to have multiple representations of the same data
  - Client can request the data in a specific format if supported

Stateless Server

- Server side should not track the client state:
  - Every request is a new request from the client
- Client side should track its own state:
  - E.g., using cookies, client side database
  - Every request must include sufficient information for server to serve up the requested information
  - Client-side MVC setup

REST?

- The REST is not history!

Exercise (Instructions): Introduction to Express Part 1

In this exercise, you will make use of the Express framework to implement similar functionality as implemented by the HTTP module based servers in the previous exercise. At the end of this exercise, you will be able to:

- Implement a simple web server using Express framework
- Implement a web server that serves static content

A Simple Server using Express

- Create a folder named *node-express* at a convenient location on your computer and move to that folder.
- Copy the *public* folder from *node-http* to this folder.
- Create a file named server-1.js and add the following code to it:

```
var express = require('express'),
     http = require('http');

var hostname = 'localhost';
var port = 3000;

var app = express();

app.use(function (req, res, next) {
  console.log(req.headers);
    res.writeHead(200, { 'Content-Type': 'text/html' });
  res.end('<html><body><h1>Hello World</h1></body></html>');

});

var server = http.createServer(app);

server.listen(port, hostname, function(){
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

- Then, install the Express framework in the folder by typing the following at the prompt:

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

- Start the server by typing the following at the prompt, and then interact with the server:

```
node server-1
```

Serving Static Files

- Create a file named server-2.js and add the following code to it:

```
var express = require('express');
var morgan = require('morgan');

var hostname = 'localhost';
var port = 3000;

var app = express();

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

app.use(express.static(__dirname + '/public'));

app.listen(port, hostname, function(){
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

- Then, install morgan by typing the following at the prompt:

```
npm install morgan --save
```

- Start the server and interact with it and observe the behavior.

Conclusions

In this exercise you learnt to use the Express framework to design and implement a web server.

✓ Complete

Exercise (Instructions): Introduction to Express Part 2

Objectives and Outcomes

In this exercise, you will develop a web server that exports a REST API. You will use the Express framework, and the Express router to implement the server. At the end of this exercise, you will be able to:

- Use application routes in the Express framework to support REST API
- Use the Express Router in Express framework to support REST API

Setting up a REST API

- Create a new file named *server-3.js* and add the following code to it:

```
var express = require('express');
var morgan = require('morgan');
var bodyParser = require('body-parser');

var hostname = 'localhost';
var port = 3000;

var app = express();

app.use(morgan('dev'));
app.use(bodyParser.json());

app.all('/dishes', function(req,res,next) {
      res.writeHead(200, { 'Content-Type': 'text/plain' });
      next();
});

app.get('/dishes', function(req,res,next){
        res.end('Will send all the dishes to you!');
});

app.post('/dishes', function(req, res, next){
     res.end('Will add the dish: ' + req.body.name + ' with details: ' + req
       .body.description);
});

app.delete('/dishes', function(req, res, next){
        res.end('Deleting all dishes');
});

app.get('/dishes/:dishId', function(req,res,next){
        res.end('Will send details of the dish: ' + req.params.dishId +' to
          you!');
});

app.put('/dishes/:dishId', function(req, res, next){
    res.write('Updating the dish: ' + req.params.dishId + '\n');
    res.end('Will update the dish: ' + req.body.name +
            ' with details: ' + req.body.description);
});

app.delete('/dishes/:dishId', function(req, res, next){
        res.end('Deleting dish: ' + req.params.dishId);
});

app.use(express.static(__dirname + '/public'));

app.listen(port, hostname, function(){
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

- Install body-parser by typing the following at the command prompt:

```
      npm install body-parser --save
```

- Start the server and interact with it from the browser/postman.

Using Express Router

- Create a new file named *server-4.js* and add the following code to it:

```
var express = require('express');
var morgan = require('morgan');
var bodyParser = require('body-parser');

var hostname = 'localhost';
var port = 3000;

var app = express();

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

var dishRouter = express.Router();

dishRouter.use(bodyParser.json());

dishRouter.route('/')
.all(function(req,res,next) {
      res.writeHead(200, { 'Content-Type': 'text/plain' });
      next();
})

.get(function(req,res,next){
        res.end('Will send all the dishes to you!');
})

.post(function(req, res, next){
    res.end('Will add the dish: ' + req.body.name + ' with details: ' + req.body
      .description);
})

.delete(function(req, res, next){
        res.end('Deleting all dishes');
});

dishRouter.route('/:dishId')
.all(function(req,res,next) {
      res.writeHead(200, { 'Content-Type': 'text/plain' });
      next();
})

.get(function(req,res,next){
        res.end('Will send details of the dish: ' + req.params.dishId +' to
          you!');
})

.put(function(req, res, next){
        res.write('Updating the dish: ' + req.params.dishId + '\n');
    res.end('Will update the dish: ' + req.body.name +
            ' with details: ' + req.body.description);
})

.delete(function(req, res, next){
        res.end('Deleting dish: ' + req.params.dishId);
});

app.use('/dishes',dishRouter);

app.use(express.static(__dirname + '/public'));

app.listen(port, hostname, function(){
  console.log(`Server running at http://${hostname}:${port}/`);
});

```

- Start the server and interact with it and see the result.

Conclusions

In this exercise, you used the Express framework and Express router to build a server supporting a REST API.

✓ Complete

Peer-graded Assignment: Node Modules, Express and REST API

You passed!
Congratulations. You earned 100 / 100 points. Review the feedback below and continue the course when you are ready. You can also help more classmates by reviewing their submissions.

Review Classmates' Work

Instructions

My submission

Discussions

Instructions

Assignment Overview

In this assignment you will continue the exploration of Node modules, Express and the REST API. At the end of this assignment, you should have completed the following tasks to update the server:

- Created a Node module using Express router to support the routes for the dishes REST API.
- Created a Node module using Express router to support the routes for the promotions REST API.
- Created a Node module using Express router to support the routes for the leadership REST API.

Assignment Requirements

The REST API for our Angular and Ionic/Cordova application that we built in the previous courses requires us to support the following REST API end points:

1. http://localhost:3000/dishes

2. http://localhost:3000/promotions
3. http://localhost:3000/leadership

We need to support GET, PUT, POST and DELETE operations on each of the three endpoints mentioned above, including supporting the use of route parameters to identify a specific dish, promotion and leader. We have already constructed the REST API for the dishes route in the previous exercise.

This assignment requires you to complete the following **three** tasks. Detailed instructions for each task are given below.

Task 1

In this task you will create a separate Node module implementing an Express router to support the REST API for the dishes. You can reuse all the code that you implemented in the previous exercise. To do this, you need to complete the following:

- Create a Node module named *dishRouter.js* that implements the Express router for the /dishes REST API end point.
- Require the Node module you create above within your Express application and mount it on the /dishes route.

Task 2

In this task you will create a separate Node module implementing an Express router to support the REST API for the promotions. To do this, you need to complete the following:

- Create a Node module named *promoRouter.js* that implements the Express router for the /promotions REST API end point.
- Require the Node module you create above within your Express application and mount it on the /promotions route.

Task 3

In this task you will create a separate Node module implementing an Express router to support the REST API for the leadership. To do this, you need to complete the following:

- Create a Node module named *leaderRouter.js* that implements the Express router for the /leadership REST API end point.
- Require the Node module you create above within your Express application and mount it on the /leadership route.

Review criteria **less**

Upon completion of the assignment, your submission will be reviewed based on the following criteria:

Task 1:

- The new Node module, *dishRouter* is implemented and used within your server to support the /dishes end point.
- The REST API supports GET, POST and DELETE operations on /dishes and GET, PUT and DELETE operations on /dishes/:id end points.

Task 2:

- The new Node module, *promoRouter* is implemented and used within your server to support the /promotions end point.
- The REST API supports GET, POST and DELETE operations on /promotions and GET, PUT and DELETE operations on /promotions/:id end points.

Task 3:

- The new Node module, *leaderRouter* is implemented and used within your server to support the /leadership end point.
- The REST API supports GET, POST and DELETE operations on /leadership and GET, PUT and DELETE operations on /leadership/:id end points.