Node.js

Assoc Prof. Dr. Kanda Runapongsa Saikaew
Department of Computer Engineering
Khon Kaen University
krunapon@kku.ac.th

Agenda (1/2)

- What is Node.js?
- Environment Setup
- First Application
- REPL Terminal
- Package Manager (NPM)
- Callbacks Concept
- Event loop

Agenda (2/2)

- Streams
- File System
- Global Objects
- Utility Modules
- Web Module
- Express Framework
- RESTFul API
- Express Framework

What is Node.js? (1/2)

- Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable work applications
- Node.js uses an event-driven, non-blocking I/O model that makes its lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices

What is Node.js? (2/2)

- Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications
- Node.js applications are written in JavaScript, and can run within Node.js runtime on OS X, Microsoft Windows, and Linux
- Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent

Node.js = Runtime Environment + JavaScript Library

Features of Node.js (1/3)

- Asynchronous and Event Driven
  - All APIs of Node.js library are asynchronous, that is non-blocking
  - A Node.js based 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 of Node.js helps the server to get a response from the previous API call

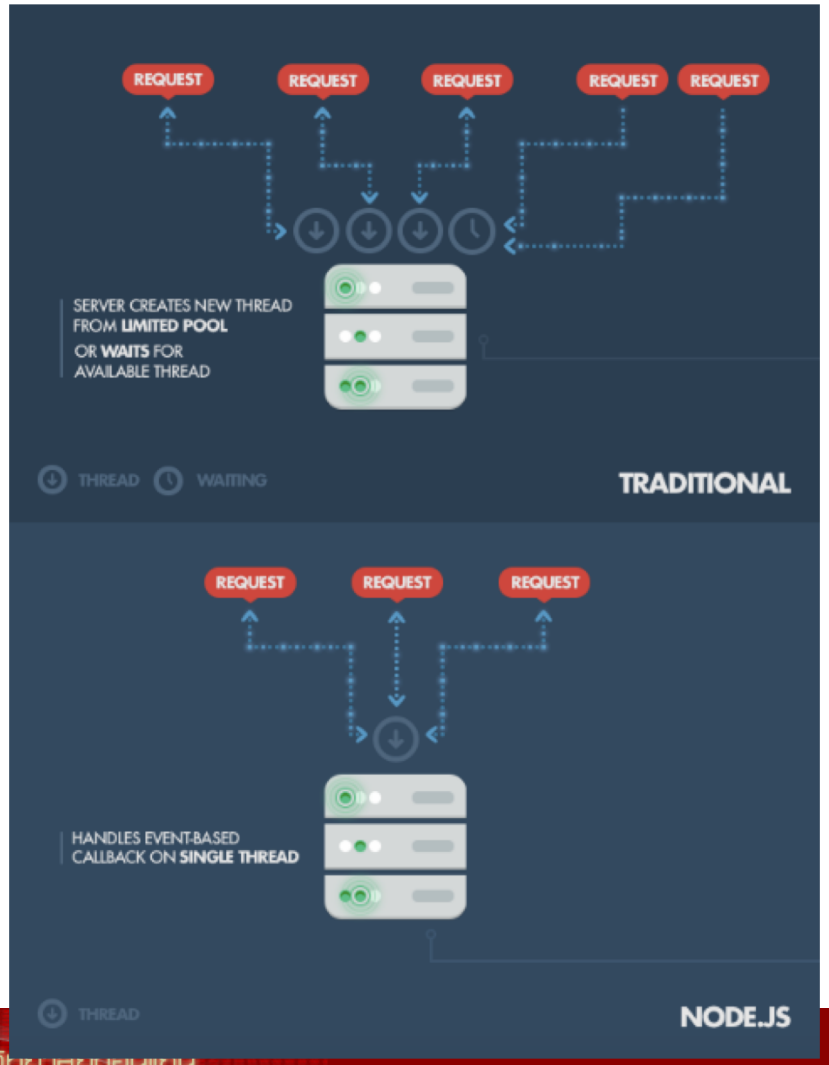
Features of Node.js (2/3)

- Single Threaded but Highly Scalable
  - Node.js uses a single threaded model with event looping
  - Event mechanism helps the server to respond in a non-blocking way
  - Traditional server created limited threads to single requests

Features of Node.js (3/3)

- Very Fast
  - Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution
- No Buffering
  - Node.js applications never buffer any data. These applications simply output the data in chunks
- License
  - Node.js is released under the MIT license

Traditional web-serving techniques vs Node.js



Who Uses Node.js?

- Several companies use Node.js
  - Netflix
  - Linkedin
  - Walmart
  - Trello
  - Uber
  - PayPal
  - Medium
  - eBay
  - NASA

Examples of where Node.js should be used (1/2)

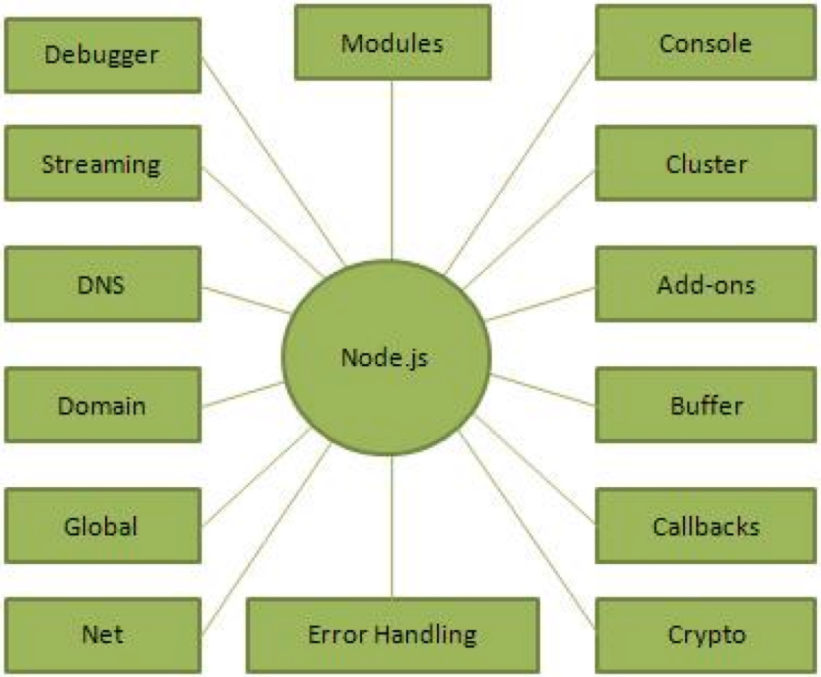
- Chat
  - Chat is the most typical real-time, multi-user application
  - It's a lightweight, high traffic, data-intensive (but low processing/computation) application that runs across distributed devices
- API on top of an object DB
  - It's quite a natural fit for exposing the data from object DBs (e.g. MongoDB)
  - JSON stored data allow Node.js to function without the impedance mismatch and data conversion.

Examples of where Node.js should be used (2/2)

- Data streaming
  - It's possible to process files while they're still being uploaded, as the data comes in through a stream
- Proxy
  - Node.js is easily employed as a server-side proxy where it can handle a large amount of simultaneous connections in a non-blocking manner
- Application monitoring dashboard
  - Another common use-case in which Node-with-web-sockets fits perfectly: tracking website visitors and visualizing their interactions in real-time.

Concepts

- The following diagram depicts some important parts of Node.js



Where to Use Node.js?

- Following are the areas where Node.js is proving itself as a perfect technology partner
  - I/O bound Applications
  - Data Streaming Applications
  - Data Intensive Real-time Applications (DIRT)
  - JSON APIs based Applications
  - Single Page Applications
- Where Not to Use Node.js?
  - It is not advisable to use Node.js for CPU intensive applications

Environment Setup

- Need to have (a) text editor (b) Node.js binary installables
- Download node.js from https://nodejs.org/en/download/
- Verify installation

```
[~/Dropbox/Sites/nodejs]$  more main.js
console.log("Hello, World!");
[~/Dropbox/Sites/nodejs]$  node main.js
Hello, World!
```

First Application

- A Node.js application consists of the following three important components
  - Import required modules – We use the require directive to load Node.js modules
  - Create server – A server which will listen to client's requests similar to Apache HTTP Server
  - Read request and return response – The server created in an earlier step will read the HTTP request made by the client which can be a browser or a console and return the response

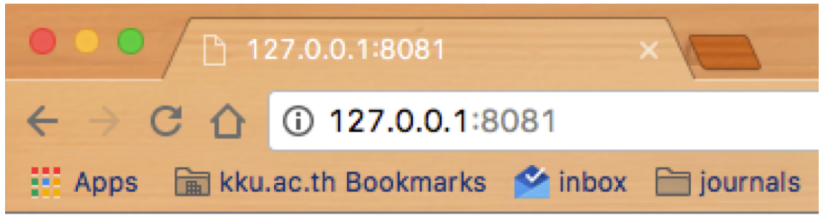
Creating Node.js Application (1/2)

- Step 1: Import Required Module
  - We use the require directive to load the http module and store the returned HTTP instance into an http variable
  - Example: var http = require(“http”);
- Step 2: Create Server
  - We use the created http instance and call http.createServer() method to create a server instance and then we bind it at port 8081 using the listen method associated with the server instance.
  - Pass it a function with parameters request and response
  - Write the sample implementation to always return “Hello World”

Creating Node.js Application (2/2)

- Step 3 – Testing Request & Response

```
[~/Dropbox/198371/labs/nodejs]more server.js
var http = require("http");

http.createServer(function (request, response) {

   // Send the HTTP header
   // HTTP Status: 200 : OK
   // Content Type: text/plain
   response.writeHead(200, {'Content-Type': 'text/plain'});

   // Send the response body as "Hello World"
   response.end('Hello World\n');
}).listen(8081);

// Console will print the message
console.log('Server running at http://127.0.0.1:8081/');
[~/Dropbox/198371/labs/nodejs]node server.js
Server running at http://127.0.0.1:8081/
```



Hello World

REPL

- REPL stands for Read Eval Print Loop
- Node.js comes bundled with a REPL environment
- It performs the following tasks
  - Read: reads user's input, parses the input into JavaScript data-structure, and stores in memory
  - Eval: takes and evaluates the data structure
  - Print: prints the result
  - Loop – loops the above command until the user pressed ctrl-c twice

Starting REPL

- REPL can be started by simply running node on shell/console without any arguments
- Simple Expression

```
[~/Dropbox/198371/labs/nodejs]node
> 1 + 3
4
> 1 + (2 * 3) - 4
3
```

PERL: Use Variables

- You can make use variables to store values and print later like any conventional script
- If var keyword is not used, then the value is store in the variable and printed
- If var keyword is used, then the value is stored but not printed. You can print variables using console.log()

```
[~/Dropbox/198371/labs/nodejs]node
> x = 10
10
> var y = 10
undefined
> x + y
20
> console.log("Hello World")
Hello World
undefined
```

PERL: Multiline Expression

- Node PERL supports multiline expression similar to JavaScript

```
[~/Dropbox/198371/labs/nodejs]node
> var x = 0
undefined
> do {
... x++
... console.log("x:" + x);
... } while (x < 5);
x:1
x:2
x:3
x:4
x:5
undefined
```

PERL: Underscore Variable

- You can use underscore (_) to get the last result

```
[~/Dropbox/198371/labs/nodejs]node
> var x = 10
undefined
> var y = 20
undefined
> x + y
30
> var sum = _
undefined
> console.log(sum)
30
undefined
>
```

REPL Commands

- Ctrl + c - terminate the current command
- Ctrl + c twice – terminate the Node REPL
- CtrL + d – terminate the Node REPL
- .help – list of all commands
- .break – exit from multiline expression
- .save filename - save the current session to a file
- .load filename – load file content in current session

Node.js - NPM

- Node Package Manager (NPM) provides
  - Online repositories for node.js packages/modules which are searchable on search.nodejs.org
  - Command line utility to install Node.js packaegs, do version management and dependency management of Node.js packages
- NPM comes bundled with Node.js installables
- Using “sudo npm install npm –g” to update it to the latest version

```
[~/Dropbox/Sites/nodejs]$  sudo npm install npm -g
Password:
/usr/local/bin/npm -> /usr/local/lib/node_modules/npm/bin/npm-cli.js
/usr/local/bin/npx -> /usr/local/lib/node_modules/npm/bin/npx-cli.js
+ npm@6.4.1
updated 1 package in 14.086s
[~/Dropbox/Sites/nodejs]$  npm --version
6.4.1
[~/Dropbox/Sites/nodejs]$
```

Installing Modules Using NPM

- There is a simple syntax to install any Node.js module
  - npm install <Module Name>
- For example, the command to install a famous Node.js web framework module called express
  - npm install express
- Now you can use this module in your js file as following
  - var express = require('express');

Local Installation (1/2)

- By default, NPM installs any dependency in the local mode
- Here local mode refers to the package installation in node_modules directory lying in the folder where Node application is present
- Locally deployed packages are accessible via require() method

Local Installation (2/2)

- We can use npm ls command to list down all the locally installed modules

```
krunapon@1.0.0 /Users/krunapon
├── @babel/standalone@7.0.0-beta.32
├── babel-preset-env@1.7.0
│   ├── babel-plugin-check-es2015-constants@6.22.0
│   │   └── babel-runtime@6.26.0
│   │       ├── core-js@2.5.7
│   │       └── regenerator-runtime@0.11.1
│   ├── babel-plugin-syntax-trailing-function-commas@6.22.0
│   ├── babel-plugin-transform-async-to-generator@6.24.1
│   │   ├── babel-helper-remap-async-to-generator@6.24.1
```

Global Installation

- Globally installed packages/dependencies are stored in system directory
- Such dependencies can be used in CLI (Command Line Interface) function of any node.js but cannot be imported using require() in Node application directly
- For example, using command “npm install express –g” to install express globally
  - Need to have permission as root/administrator

```
[~/Dropbox/Sites/nodejs]$  sudo npm install express -g
Password:
+ express@4.16.4
added 1 package from 1 contributor, removed 1 package and updated 19 packages i
n 4.454s
[~/Dropbox/Sites/nodejs]$
```

Learning by Doing

- Learn the basics of JavaScript. No previous programming experience required
  - sudo npm install –g javascripting
  - javascripting
- Learn Git and GitHub basics
  - sudo npm install –g git-it
  - git-it
- Learn the basic of node: asynchronous i/o, http
  - sudo npm install –g learnyounode
  - learnyonode

Agenda

- What is Node.js?
- Environment Setup
- First Application
- REPL Terminal
- Package Manager (NPM)
- **Callbacks Concept**
- Event loop
- Buffer
- RESTFul API

What is Callback?

- Callback is an asynchronous equivalent for a function
- A callback function is called at the completion for a given task
- Node makes heavy use of callbacks
- All the APIs of Node are written in such a way that they support callbacks

Why Callback?

- For example, a function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed
- Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter

Why Callback?

- Why Node.js highly scalable
- There is no blocking or wait for file I/O
- It can process a high number of requests without wait for any function to return results

Blocking Code vs. Non-Blocking Code

- Blocking Code

```
[~/Dropbox/Sites/nodejs]$  more block_code.js
var fs = require("fs");

var data = fs.readFileSync("input.txt");

console.log(data.toString());
console.log("Program Ended");
```

- Non-blocking Code

```
[~/Dropbox/Sites/nodejs]$  more non_block_code.js
var fs = require("fs");

fs.readFile("input.txt", function(err, data) {
        if (err)
                return console.error(err);
        console.log(data.toString());
});

console.log("Program Ended");
```

Blocking vs Non-Blocking Results

```
[~/Dropbox/Sites/nodejs]$  node block_code.js
The best preparation for tomorrow is doing your best today.
- H. Jackson Brown, Jr.

In order to succeed, we must first believe that we can.
- Nikos Kazantzakis

Program Ended
[~/Dropbox/Sites/nodejs]$  node non_block_code.js
Program Ended
The best preparation for tomorrow is doing your best today.
- H. Jackson Brown, Jr.

In order to succeed, we must first believe that we can.
- Nikos Kazantzakis
```

What We Learn from These Examples

- The first example shows that the program blocks until it reads the file and then only it proceeds to end the program
- The second example shows that the program does not wait for the reading and print “Program Ended” and at the same time, the program without blocking continues reading the file

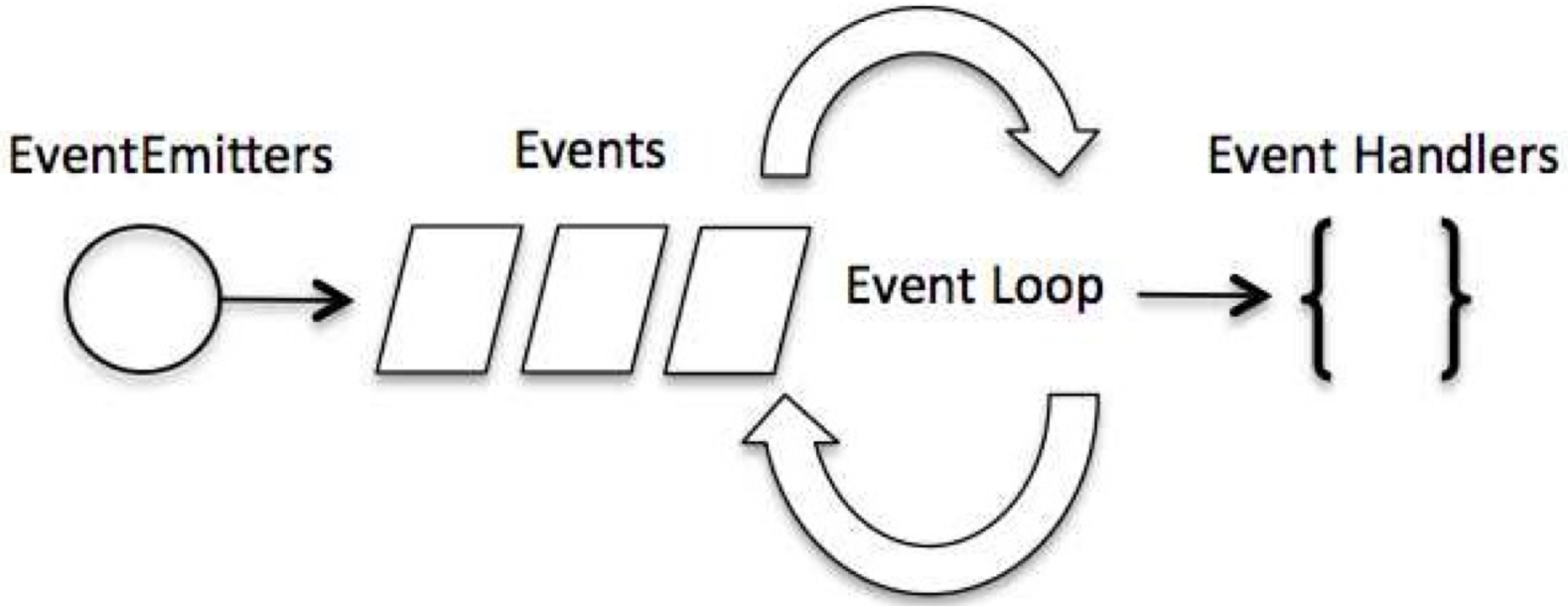
How Node.js Supports Concurrency

- Node.js is a single-threaded application
- It can support concurrency via the concept of event and callbacks
- Every API of Node.js is asynchronous and being single-threaded
- They use async function calls to maintain concurrency

Event-Driven Programming

- Node.js uses event heavily and it is also one of the reasons why Node.js is pretty fast compared to other technologies
- As soon as Node starts its server, it simply initiates its variables, declares functions and then simply wait for the event to occur
- There is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected

Event-Driven Programming



Events vs Callbacks

- Although events look quit similar to callbacks, the difference lies in the fact that callback functions are called when an asynchronous function returns its result, whereas event handling works on the observer pattern
- The functions that listen to events act as Observers
- Whenever an event gets fired, its listener function starts executing

Procedure in Firing an Event

```
// Import events module
var events = require('events');
// Create an eventEmitter object
var eventEmitter = new events.EventEmitter();
// Bind event and event handler as follows
eventEmitter.on('eventName', eventHandler);
// Fire an event
eventEmitter.emit('eventName');
```

Example: event.js (1/2)

```
// Import events module
var events = require('events');
// Create an eventEmitter object
var eventEmitter = new events.EventEmitter();
// Create an event handler as follows
var connectHandler = function connected() {
console.log('connection successful.');
// Fire the data_received event
eventEmitter.emit('data_received');
}
```

Example: event.js (2/2)

```
// Bind the connection event with the handler
eventEmitter.on('connection', connectHandler);
// Bind the data_received event with the anonymous function
eventEmitter.on('data_received', function() {
  console.log('data received successfully.');
});
// Fire the connection event
eventEmitter.emit('connection');
console.log('Program Ended.');
```

How Node Applications Work?

- In Node Application, any async function accepts a callback as the last parameter and a callback function accepts an error as the first parameter
- Example

```
fs.readFile('input.txt', function(err, data) {
 });
```

- function(err,data) is the callback function
- err is an error

Example: file.js

```
var fs = require("fs");
fs.readFile("input.txt", function(err, data) {
        if (err) {
                console.log(err.stack);
                return;
        }
        console.log(data.toString());
});
console.log("Program Ended");
```

Explanation about file.js

- Here fs.readFile() is a async function whose purpose is to read a file.
- If an error occurs during the read operation, then the **err object** will contain the corresponding error, else data will contain the contents of the file
- **readFile** passes err and data to the callback function after the read operation is complete, which finally prints the content

EventEmitter Class

- When an EventEmitter instance faces any error, it emits an ‘error’ event
- When a new listener is added, ‘newListener’ event is fired and when a listener is removed, 'removeListener' event is fired
- EventEmitter provides multiple properties like **on** and **emit**
  - on property is used to bind a function with the event
  - Emit is used to fire an event

Example: events.js (1/2)

```
var events = require('events');
var eventEmitter = new events.EventEmitter();

// listener #1
var listner1 = function listner1() {
    console.log('listener1 executed.');
}

// listner #2
var listner2 = function listner2() {
    console.log('listener2 executed.');
}

// Bind the connection event with the listner1 function
eventEmitter.addListener('connection', listner1);

// Bind the connection event with the listner2 function
eventEmitter.on('connection', listner2);

var eventListeners = require('events').EventEmitter.listenerCount
(eventEmitter, 'connection');
console.log(eventListeners + " Listener(s) listening to connection event");
```

Example: events.js (2/2)

```
// Fire the connection event
eventEmitter.emit('connection');

// Remove the binding of listner1 function
eventEmitter.removeListener('connection', listner1);
console.log("Listner1 will not listen now.");

// Fire the connection event
eventEmitter.emit('connection');

eventListners = require('events').EventEmitter.listenerCount(eventEmitter,
'connection');

console.log(eventListeners + " Listener(s) listening to connection event");

console.log("Program Ended.");
```

Running events.js

```
[~/Dropbox/Sites/nodejs]$  node events.js
2 Listener(s) listening to connection event
listener1 executed.
listener2 executed.
Listner1 will not listen now.
listener2 executed.
2 Listener(s) listening to connection event
Program Ended.
```

Agenda (2/2)

- **Streams**
- File System
- Web Module
- Express Framework
- RESTFul API

Streams

- Streams are objects that let you read data from a source or write data to a destination in continuous fashion
- There are four types of streams
  - Readable: Stream which is used for read operation
  - Writable: Stream which is used for write operation
  - Duplex: Stream which can be used for both read and write operation
  - Transform: A type of duplex stream where the output is computed based on input

Stream is an EventEmitter

- Each type of Stream is an EventEmitter instance and throws several events at different instance of times
- Some of the commonly used events
  - data: this event is fired when there is data is available to read
  - end: this event is fired when there is no more data to read
  - error: this event is fired when there is any error receiving or writing data
  - Finish: this event is fired when all the data has been flushed to underlying stream

Reading from a Stream

```
1  var fs = require("fs");
2  var data = '';
3
4  // Create a readable stream
5  var readerStream = fs.createReadStream('input.txt');
6
7  // Set the encoding to be utf8
8  readerStream.setEncoding('UTF8');
9
10 readerStream.on('data', function(chunk) {
11     data += chunk;
12 });
13
14 readerStream.on('end', function() {
15     console.log(data);
16 });
17
18 readerStream.on('error', function(err) {
19     console.log(err.stack);
20 });
21
22 console.log("Program Ended");
```

```
[~/Dropbox/Sites/nodejs]$  node readfile.js
Program Ended
The best preparation for tomorrow is doing your best today.
- H. Jackson Brown, Jr.

In order to succeed, we must first believe that we can.
- Nikos Kazantzakis
```

Writing to a Stream

```
1  var fs = require("fs");
2  var data = 'Keep learning';
3
4  // Create a readable stream
5  var writerStream = fs.createWriteStream('output.txt');
6
7  // Set the encoding to be utf8
8  writerStream.write(data, 'UTF8');
9
10 // Mark the end of file
11 writerStream.end();
12
13 // Handle stream events => finish and error
14 writerStream.on('finish', function() {
15     console.log("Write completed.");
16 });
17
18 writerStream.on('error', function(err) {
19     console.log(err.stack);
20 });
21
22 console.log("Program Ended");
```

```
[~/Dropbox/Sites/nodejs]$  node writefile.js
Program Ended
Write completed.
```

```
[~/Dropbox/Sites/nodejs]$  ls -l output.txt
-rw-r--r--@ 1 krunapon  staff  13 Oct 31 13:40 output.txt
[~/Dropbox/Sites/nodejs]$  more output.txt
Keep learning
```

Piping the Streams

- Piping is a mechanism where we provide the output of one stream as the input to another stream
- Sample

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

readerStream = fs.createReadStream('input.txt');

var writerStream = fs.createWriteStream('output.txt');
output.txt readerStream.pipe(writerStream);
console.log("Program Ended");
```

File System: Synchronous vs. Asynchronous

- Every method in the fs module has synchronous as well as asynchronous forms
- Asynchronous methods take the last parameter as the completion function callback and the first parameter of the callback function as error
- The asynchronous method never blocks a program whereas the second one does

Agenda (2/2)

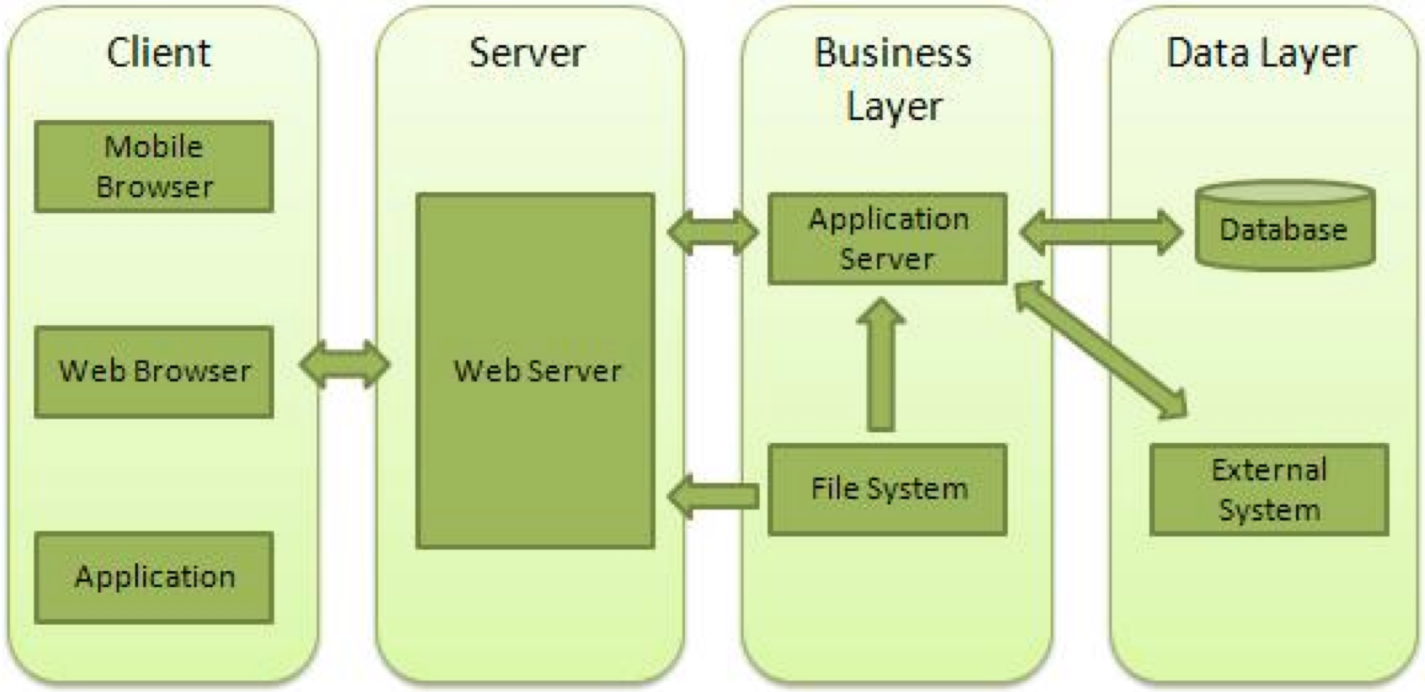
- Streams
- File System
- **Web Module**
- Express Framework
- RESTFul API

What is a Web Server?

- A Web Server is a software application which handles HTTP request sent by the HTTP client, like web browsers, and return web pages in response to the clients
- Most web servers support server-side scripts which retrieves data from a database and performs complex logic and ten sends a result to the HTTP client through the Web server
- Apache web server is one of the most commonly used web servers. It is an open source project

Web Application Architecture

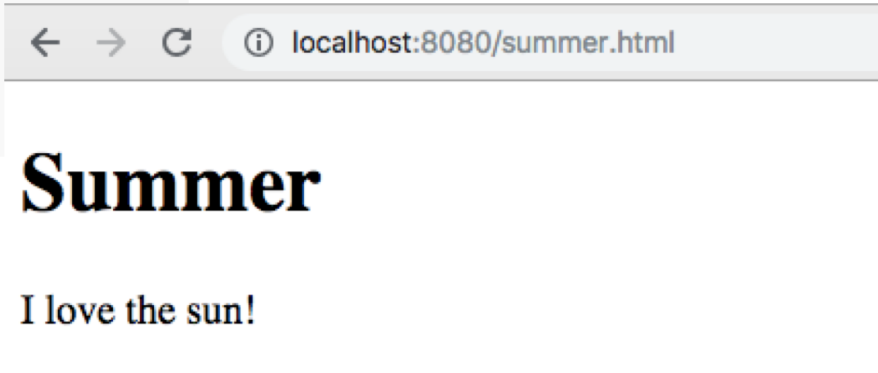
A Web application is usually divided into four layers –



Creating a Web Server (1/2)

```
1  var http = require('http');
2  var fs = require('fs');
3  var url = require('url');
4
5  // Create a server
6  http.createServer(function(request, response) {
7      // Parse the request containing the file name
8      var pathname = url.parse(request.url).pathname;
9
10     // Print the name of the file for which request is made
11     console.log("Request for " + pathname + " received.");
12
13     // Read the request file content from file system
14     fs.readFile(pathname.substr(1), function (err, data) {
15         if (err) {
16             console.log(err);
17             response.writeHead(404, {'Content-Type': 'text/html'});
18         } else {
19             response.writeHead(200, {'Content-Type': 'text/html'});
20
21             response.write(data.toString());
22         }
23         response.end();
24     });
25 }).listen(8080);
26
27 console.log('Server running at http://localhost:8080/');
```

```
[~/Dropbox/Sites/nodejs]$  node server.js
Server running at http://localhost:8080/
Request for /summer.html received.
```



Agenda (2/2)

- Streams
- File System
- Web Module
- **Express Framework**
- RESTFul API

Express Overview (1/2)

- Express is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications
- It facilitates the rapid development of Node based Web applications

Express Overview (2/2)

- Core features of Express framework
  - Allows to set up middlewares to respond to HTTP requests
  - Defines a routing table which is used to perform different actions based on HTTP Method and URL
  - Allows to dynamically render HTML Pages based on passing arguments to templates

Installing Express

- Firstly, install the Express framework globally using NPM so that it can be used to create a web application using node terminal

  % npm install express –save

- The above command saves the installation locally in the node_modules directory and create a directory express inside node_modules

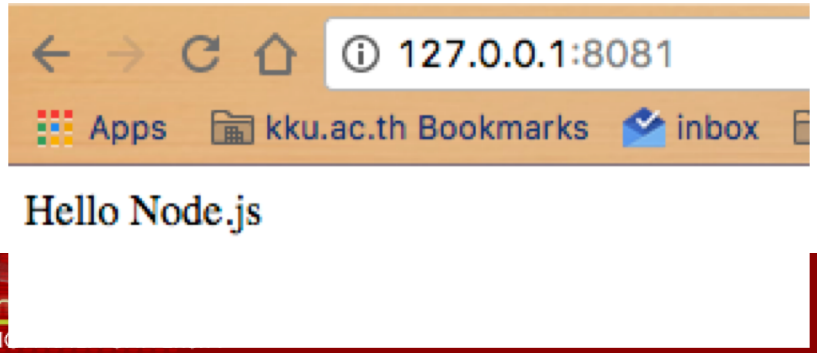
Important Modules Along with Express

- body-parser – This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data
  - npm install body-parser --save
- cookie-parrser – Parse Cookie header and populate req.cookies with an object keyed
  - npm install cookie-parser --save
- multer – This is a node.js middleware for handing multipart/form-data
  - npm install multer --save

Sample: express.js

```
1  var express = require('express');
2  var app = express();
3
4  app.get('/', function(req,res) {
5      res.send('Hello Node.js');
6  });
7
8  var server = app.listen(8081, function () {
9
10     console.log("Example app listening at http://127.0.0.1:8081");
11 })
```

```
[~/Dropbox/Sites/nodejs]$  node express.js
Example app listening at http://127.0.0.1:8081
```



Agenda (2/2)

- Streams
- File System
- Web Module
- Express Framework
- **RESTFul API**

HTTP Methods

- GET – This is used to provide a read only access to a resource
- PUT – This is used to create a new resource
- DELETE – This is used to remove a resource
- POST – This is used to update an existing resource or create a new resource

Sample File users.json

```
{
    "user1" : {
        "name" : "kanda",
        "password" : "pwd1",
        "profession" : "teacher",
        "id" : 1
    },
    "user2" : {
        "name" : "hillary",
        "password" : "pwd2",
        "profession" : "politician",
        "id" : 2
    }
}
```

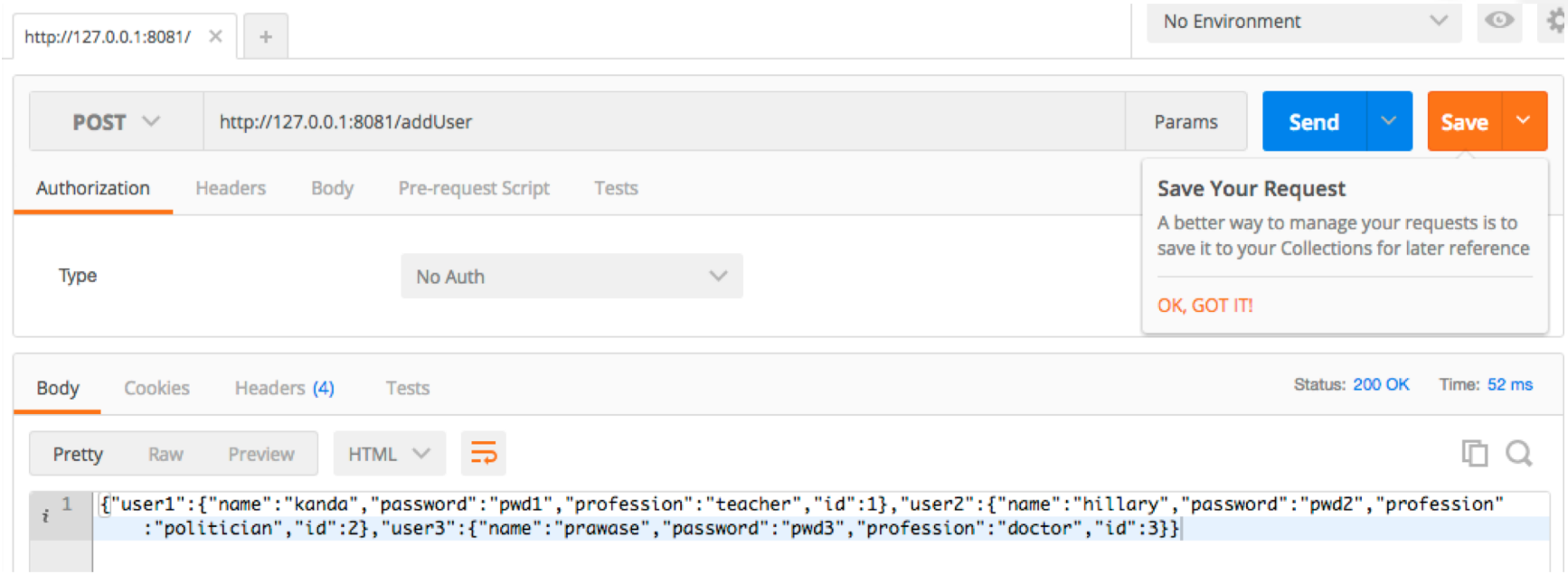
URIs for RESTful APIs

URI	HTTP Method	POST body	Result
listUsers	GET	empty	Show list of all the users
addUser	POST	JSON String	Add details of a new user
deleteUser	DELETE	JSON String	Delete an existing user
:id	GET	empty	Show details of a user

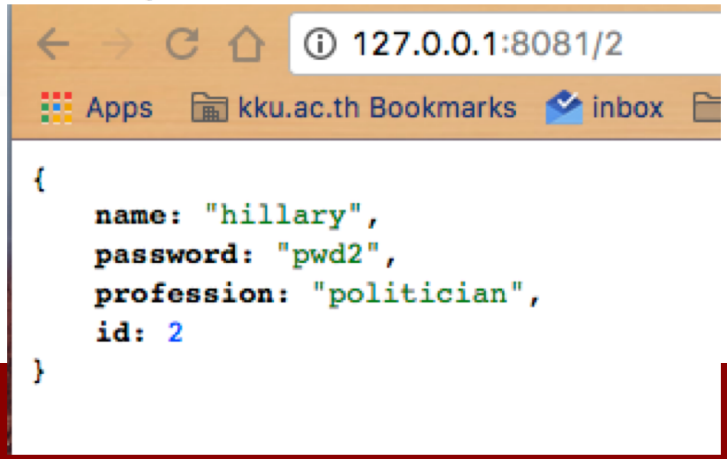
```
1  var express = require('express');
2  var app = express();
3  var fs = require("fs");
4
5  app.get('/listUsers', function(req,res) {
6      fs.readFile(__dirname + "/" + "users.json", "utf8",
7          function(err,data) {
8              console.log(data);
9              res.end(data);
10     });
11 })
12
13 var server = app.listen(8081, function () {
14     console.log("Example app listening at http://127.0.0.1:8081");
15 })
```

```
[~/Dropbox/198371/labs/nodejs]more adduser.js
var express = require('express');
var app = express();
var fs = require("fs");

var user = {
        "user3" : {
                "name" : "prawase",
                "password" : "pwd3",
                "profession" : "doctor",
                "id" : 3
        }
}

app.post('/addUser', function(req,res) {
        fs.readFile(__dirname + "/" + "users.json", "utf8",
                function(err,data) {
                        data = JSON.parse(data);
                        data["user3"] = user["user3"];
                        console.log(data);
                        res.end(JSON.stringify(data));
        });
})

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

        console.log("Example app listening at http://%s:%s",
                host, port)
})
```

Result of Method addUser with HTTP POST

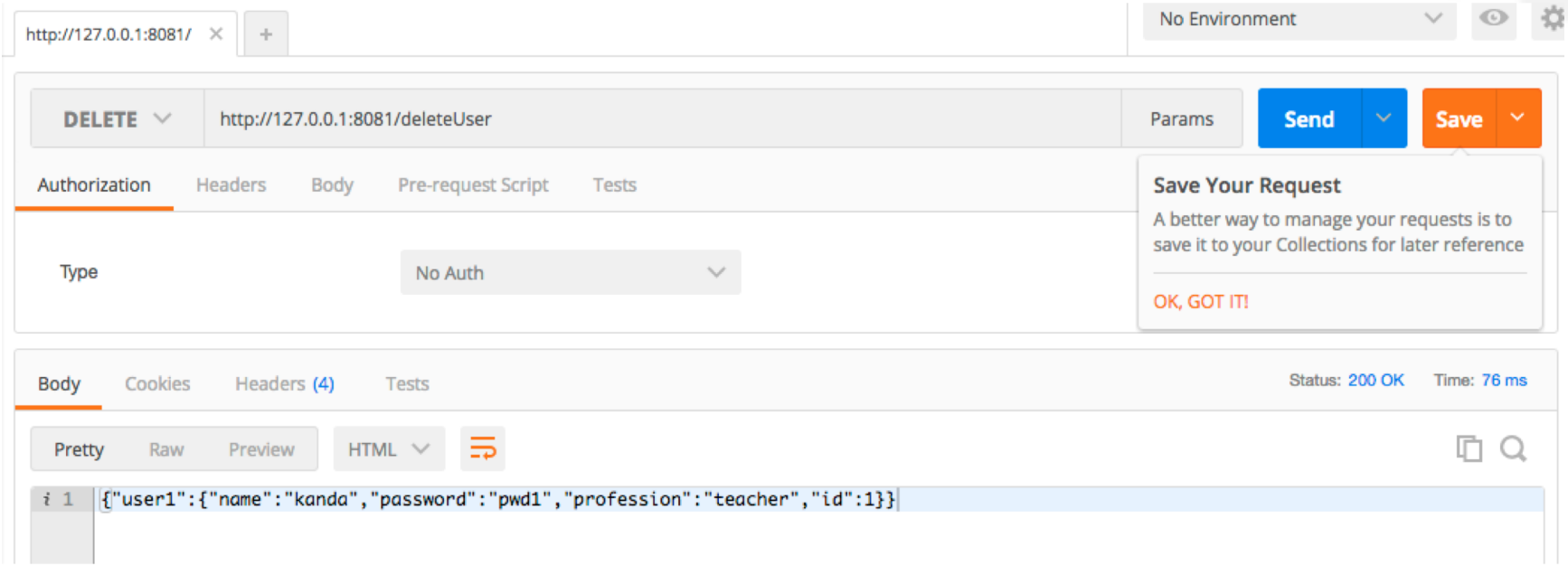


```
[~/Dropbox/198371/labs/nodejs]more userdetail.js
var express = require('express');
var app = express();
var fs = require("fs");

app.get('/:id', function(req,res) {
        // first read existing users
        fs.readFile(__dirname + "/" + "users.json", "utf8",
                function(err,data) {
                        users = JSON.parse(data);
                        var user = users["user" + req.params.id]
                        console.log(data);
                        res.end(JSON.stringify(user));
        });
})

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

        console.log("Example app listening at http://%s:%s",
                host, port)
})
```



```
[~/Dropbox/198371/labs/nodejs]more deleteuser.js
var express = require('express');
var app = express();
var fs = require("fs");

app.delete('/deleteUser', function(req,res) {
        // first read existing users
        fs.readFile(__dirname + "/" + "users.json", "utf8",
                function(err,data) {
                        data = JSON.parse(data);
                        delete data["user" + 2];
                        console.log(data);
                        res.end(JSON.stringify(data));
        });
})

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

        console.log("Example app listening at http://%s:%s",
                host, port)
})
```

Result of Method deleteUser with HTTP DELETE



References

- https://www.tutorialspoint.com/nodejs/index.htm