

# NODE.JS WEB SERVICES

Stephen Schaub

# Hello World Server

2

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

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.write('Hello World\n');
  res.end();
}).listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');
```

# http.createServer

3

## □ Callback function

```
http.createServer(function (request, response) { ... })
```

- request – http.IncomingMessage object
  - Cookies, request headers, query string, ...
- response – http.ServerResponse object
  - Response code, headers, body
- See Node.js API for details

# Serving Up Files

4

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

http.createServer(function(request, response) {
  console.log("Serving: " + request.url);
  fs.readFile(__dirname + request.url, function(err, data) {
    if (err) {
      response.statusCode = 500;
      response.end(String(err));
    } else {
      response.end(data);
    }
  });
}).listen(8888);
```

## Notes:

- `request.url` is the path and query string of the request
- `response.end(data)` transmits *data* to client and closes connection

# Server with Streams and Pipes

5

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

http.createServer(function(request, response) {
  console.log("Serving: " + request.url);
  fs.readFileStream(__dirname + request.url)
    .on('error', function(err) {
      response.statusCode = 500;
      response.end(String(err));
    })
    .pipe(response);
});
}).listen(8888);
```

# A Node.js Web App

6

```
var http = require("http");
http.createServer(function(request, response) {
  if (request.url === "/") {
    response.end("Hello <strong>home page</strong>");
  } else if (request.url === "/foo") {
    response.end("Hello <strong>foo</strong>");
  } else if (request.url === "/bar") {
    response.end("Hello <strong>bar</strong>");
  } else {
    response.end("404 Not Found");
  }
}).listen(8001);
```

7

# Express

A framework for Node.js web applications

# An Express Web App with Routes

8

```
var express = require("express");
var app = express();
app.get("/", function(req, res, next) {
  res.send("Hello <strong>home page</strong>");
});
app.get("/foo", function(req, res, next) {
  res.send("Hello <strong>foo</strong>");
});
app.get("/bar", function(req, res, next) {
  res.send("Hello <strong>bar</strong>");
});
app.listen(8000);
```



# App Basics

9

- Instantiate express application object

```
var express = require("express");  
var app = express();
```

- Define routes using `get()/post()`

- ▣ Maps URLs to functions handling those requests

```
app.get("/foo", function(req, res, next) { ... });  
app.post("/foo", function(req, res, next) { ... });
```

- Begin listening for requests

```
app.listen(8000);
```

# Routes with Parameters

10

- Routes can match with wildcard patterns

```
app.get('/ab*cd', function(...) { ... })
```

- Or regular expressions

```
app.get(/\/products\/(?:\w+)?$/ , function(...) {  
  var productId = req.params[0];  
})
```

- URLs can include parameter markers

```
app.get("/books/:bookId", function(req, res, next) {  
  res.send("You requested bookId: " + req.params.bookId);  
});
```

# Express Middleware

11

- Express uses **middleware**, components that make up a request processing pipeline, to provide important functionality
- Add middleware to request processing pipeline with `app.use( )` to:
  - ▣ Decode POST submissions
    - `app.use(bodyParser.json());`
    - `app.use(bodyParser.urlencoded({ extended: false }));`
  - ▣ Do Logging
    - `app.use(logger('dev'));`
  - ▣ Serve static content
    - `app.use(express.static(path.join(__dirname, 'public')));`
  - ▣ Do error handling

# Adding Your Own Middleware

12

```
app.use(function(request, response, next) {  
  // ... do some processing  
  next(); // continue with next Middleware component  
});
```

Notes:

- Omit the call to `next()` if this function has handled the request (called `response.send()`)

# Error Handling

13

- Define an error-handling middleware function
  - ▣ After all other middleware
  - ▣ Using four arguments instead of three

```
app.use(function(err, req, res, next) {  
  console.error(err.stack);  
  res.status(500).send('Something broke!');  
});
```

# Express API

14

- Available at <http://expressjs.com/4x/api.html>
- API Classes:
  - ▣ Application – methods and properties to configure application, render HTML views, manage state
  - ▣ Request – represents incoming request
  - ▣ Response – generate response
  - ▣ Router – map URL's to processing functions

# Request API

15

- req.query
  - ▣ Query string parameter values
- req.body
  - ▣ Submitted form fields (requires body-parser)
- req.cookies
  - ▣ Cookie values (requires cookie-parser)
- See also
  - ▣ <http://expressjs.com/4x/api.html#req>

# Response API

16

- `res.send()`
  - ▣ Send HTML, JSON, or binary response
- `res.write()`
  - ▣ Send text
- `res.end()`
  - ▣ End response
- `res.set()`
  - ▣ Set HTTP header
- `res.redirect()`
  - ▣ Redirect to another route
- See also
  - ▣ <http://expressjs.com/4x/api.html#res>



# Case Study

17

- Hello, World with Express
  - ▣ Make it Interactive
  - ▣ Add a Form
  - ▣ Add POST processing

# Modular Application Organization

18

- Express applications are often organized into modules that handle different URL prefixes

- In `app.js`:

```
var users = require('./routes/users');  
app.use('/users', users);
```

- In `routes/users.js`:

```
var express = require('express');  
var router = express.Router();  
// handle /users/boo:  
router.get('/boo', function(req, res, next) {  
  res.send('blah blah');  
});  
module.exports = router;
```