# COMP 3322 Modern Technologies on World Wide Web

2nd semester 2017-2018

Node.js (O2)

Dr. C Wu

Department of Computer Science
The University of Hong Kong

## Roadmap

- Technologies for creating dynamic, interactive web pages
  - PHP [server side]
  - JavaScript (AJAX, JSON, jQuery) [client side]
  - HTML5 [client side]
  - Node.js [server side and client side]
  - AngularJS [client side]
  - React [client side]

## Overview of Node.js

- Node.js is a powerful JavaScript-based platform running on V8 (JavaScript execution engine built for Google Chrome) for *easily* building fast network applications
  - Node.js server programs are written in JavaScript and can be run within the Node.js runtime (v8) on various operating systems
  - Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient for data-intensive real-time applications, e.g., web applications such as video streaming sites

    it does not wait for an API to return data; uses an event mechanism to get and process response from the previous API call
  - Node.js operates on a single thread, so it is not recommended for CPU intensive application, since it cannot make use of multiple CPU cores to expedite the application
  - open source, free, used by IBM, Microsoft, Walmart, Groupon, LinkedIn, PayPal, etc.

## Overview of Node.js (cont'd)

- Node.js is primarily used to build web applications, making it similar to PHP
  - *main difference*

    PHP is a blocking language, where commands execute only after the previous command has completed

    Node.js is a non-blocking language, implements event-driven programming using JavaScript, where commands execute in parallel and use callbacks to signal completion

- Node.js can be used to build other network applications as well, e.g., TCP server and TCP client

## Overview of Node.js (cont'd)

- Node.js creates network applications using a collection of "modules" that handle various core functionalities
  - e.g., modules handle file system I/O, networking (HTTP, TCP, UDP, DNS, or TLS/SSL), binary data (buffers), cryptography functions, data streams, etc.
  - thousands of open-source libraries have been built for Node.js
- There are many frameworks built on Node.js used to accelerate the development of web applications
  - Express.js (most popular Node.js web application framework)
  - Restify.js (built specifically for building correct REST web services)
  - Hapi.js (another rich framework for building applications and services)
  - etc.

Node.js = Runtime Environment + JavaScript Library

## An exmple Node.js application

- A simple Web server implemented by main.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/');
```

return the built-in HTTP module

a callback function, called after server created

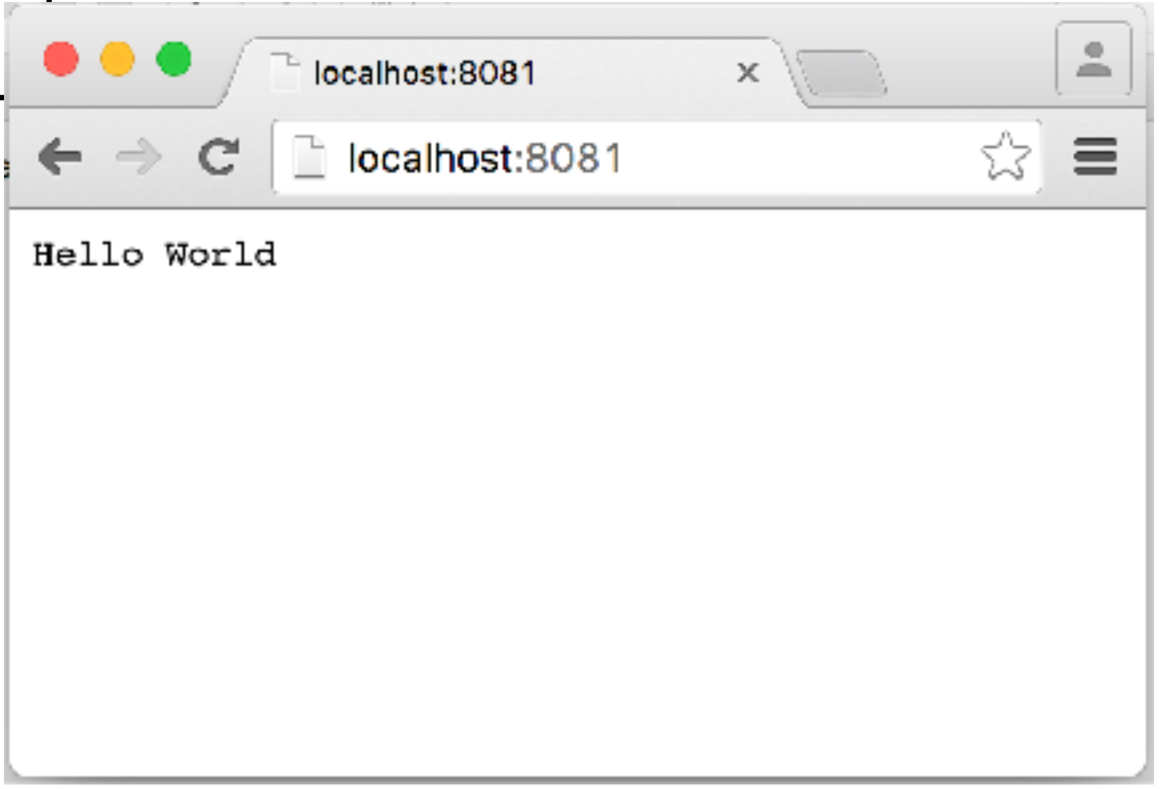
Node.js makes heavy use of callbacks

on console:

```
$ node main.js
```

execute main.js to start the server

- Web browser display when accessing http://127.0.0.1:8081



```
Hello World
```

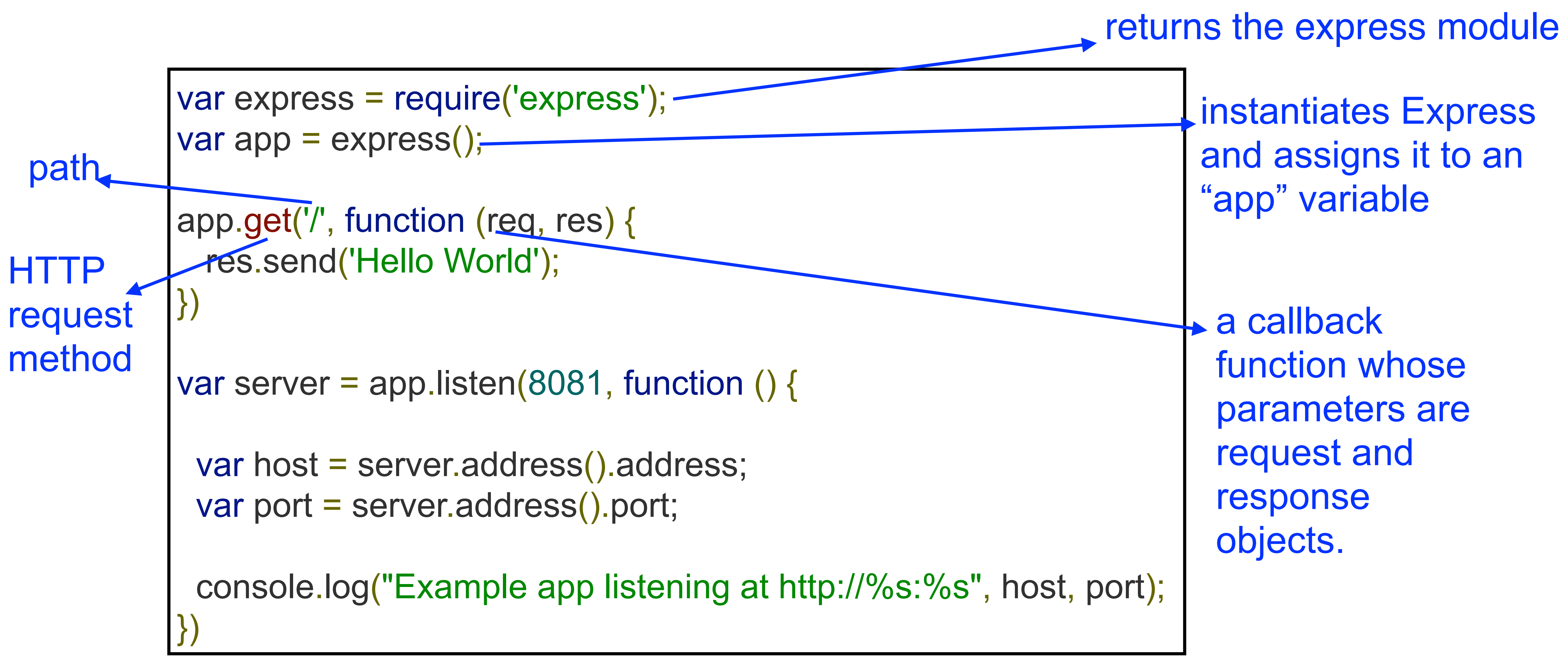
## Express.js

- Express.js is the most popular Node.js web application framework, which provides a robust set of features for *easy* development of web applications
  - Set up middlewares to respond to HTTP requests
  - Define routes which are used to perform different actions based on HTTP Method and URL
  - Allow to dynamically render HTML pages based on passing arguments (data) to templates

    works with template engines such as Pug, EJS, Hogan, etc.

# A basic Express app

- An Express app which starts a server and listens on port 8081 for connection



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

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

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);
})
```

returns the express module
instantiates Express and assigns it to an "app" variable
path
HTTP request method
a callback function whose parameters are request and response objects.

**app.js** (used as the main file for your Express app)

## Routing

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

a route and its handler function which handles GET requests to “/“

- An endpoint is combination of a URI (or path) and a specific HTTP request method (GET, POST, etc.)
- A route is a combination of a URI, a HTTP request method, and one or more handlers for the endpoint, which takes the following structure

  app.METHOD(path, [callback..], callback)

  where app is an instance of express, METHOD is an HTTP request method (in lower case), path is a path on the server, and callback is the function executed when the route is matched
- Routing refers to determining how an application responds to a client request to a particular endpoint

## Routing (cont'd)

Express apps can respond to various HTTP request methods, e.g., GET, POST, PUT, DELETE, etc.

```
// This responds to a POST request for the homepage
app.post('/', function (req, res) {
   res.send('Got a POST request');
})

// This responds to a PUT request for the /user page
app.put('/user', function (req, res) {
  res.send('Got a PUT request at /user');
});

// This responds to a DELETE request for the /del_user page
app.delete('/del_user', function (req, res) {
   res.send('Got a DELETE request at /user');
})
```

different routes

## Routing (cont'd)

- The express.Router class can be used to create modular mountable route handlers
  - a Router instance created for this class corresponds to a complete routing system

1. Create birds.js in the same directory as app.js, as follows:

```
var express = require('express');
var router = express.Router();

// middleware specific to this router
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next();
});

// define the home page route
router.get('/', function(req, res) {
  res.send('Birds home page');
});

// define the about route
router.get('/about', function(req, res) {
  res.send('About birds');
});

module.exports = router;
```

create a router instance

function carried out for all requests, before their respective handler is executed

pass control to the next handler

export this router as a module

## Routing (cont'd)

- The express.Router class can be used to create modular mountable route handlers
  - a Router instance created for this class corresponds to a complete routing system

1. Create birds.js in the same directory as app.js, as follows:

```
var express = require('express');
var router = express.Router();

// middleware specific to this router
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next();
});

// define the home page route
router.get('/', function(req, res) {
  res.send('Birds home page');
});

// define the about route
router.get('/about', function(req, res) {
  res.send('About birds');
});

module.exports = router;
```

2. Then in app.js, add:

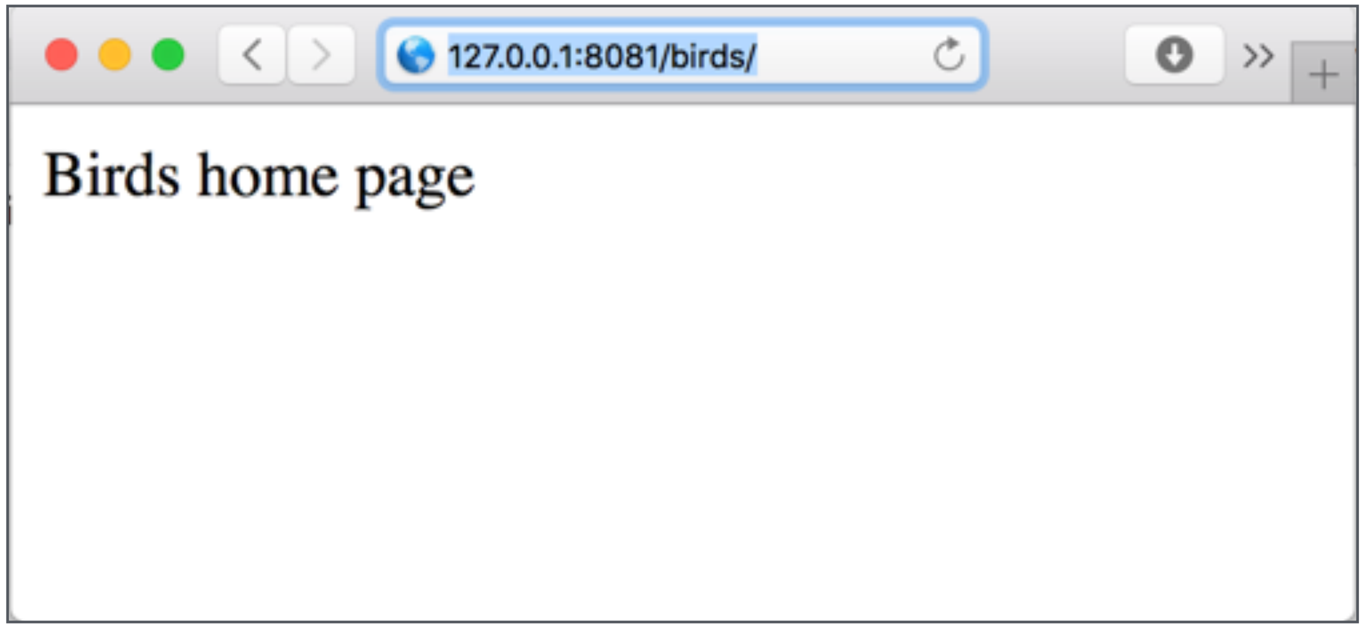
```
var birds_router = require('./birds');
...
app.use('/birds', birds_router);
```

load the router module defined in birds.js

3. The app will now be able to handle requests to "/birds/" and "/birds/about" using routes defined in the router module

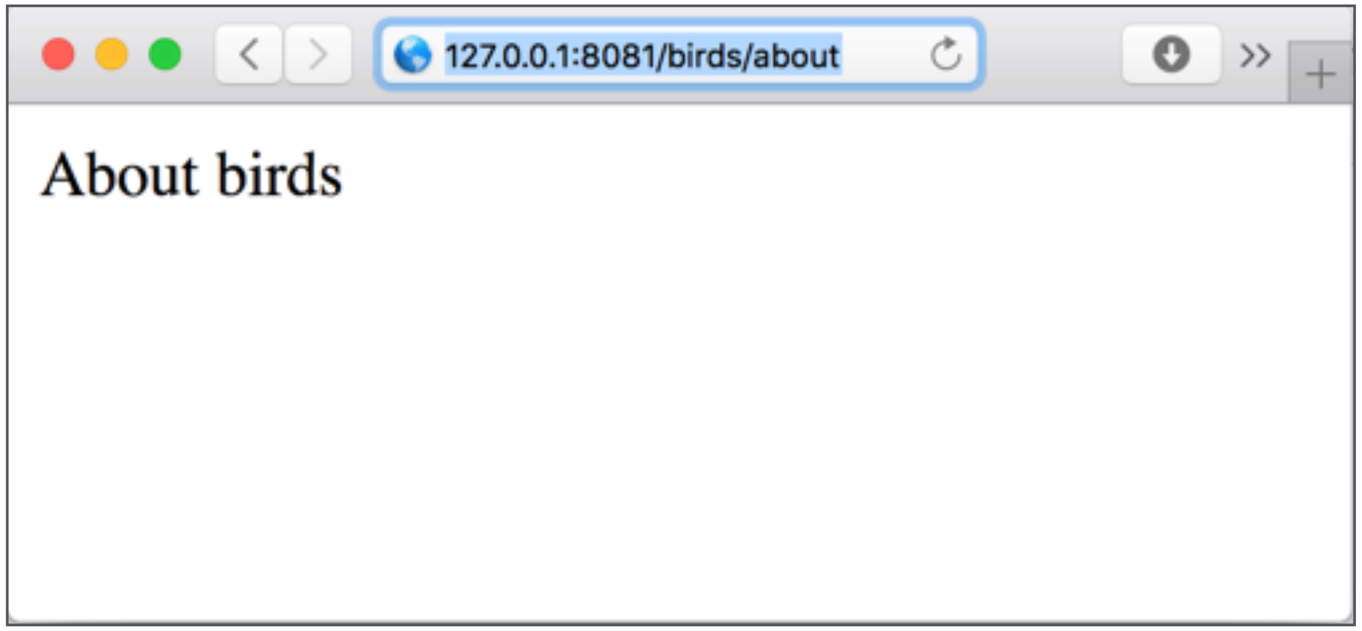
See more about routing at:
http://expressjs.com/guide/routing.html

## Routing (cont'd)



127.0.0.1:8081/birds/

Birds home page



127.0.0.1:8081/birds/about

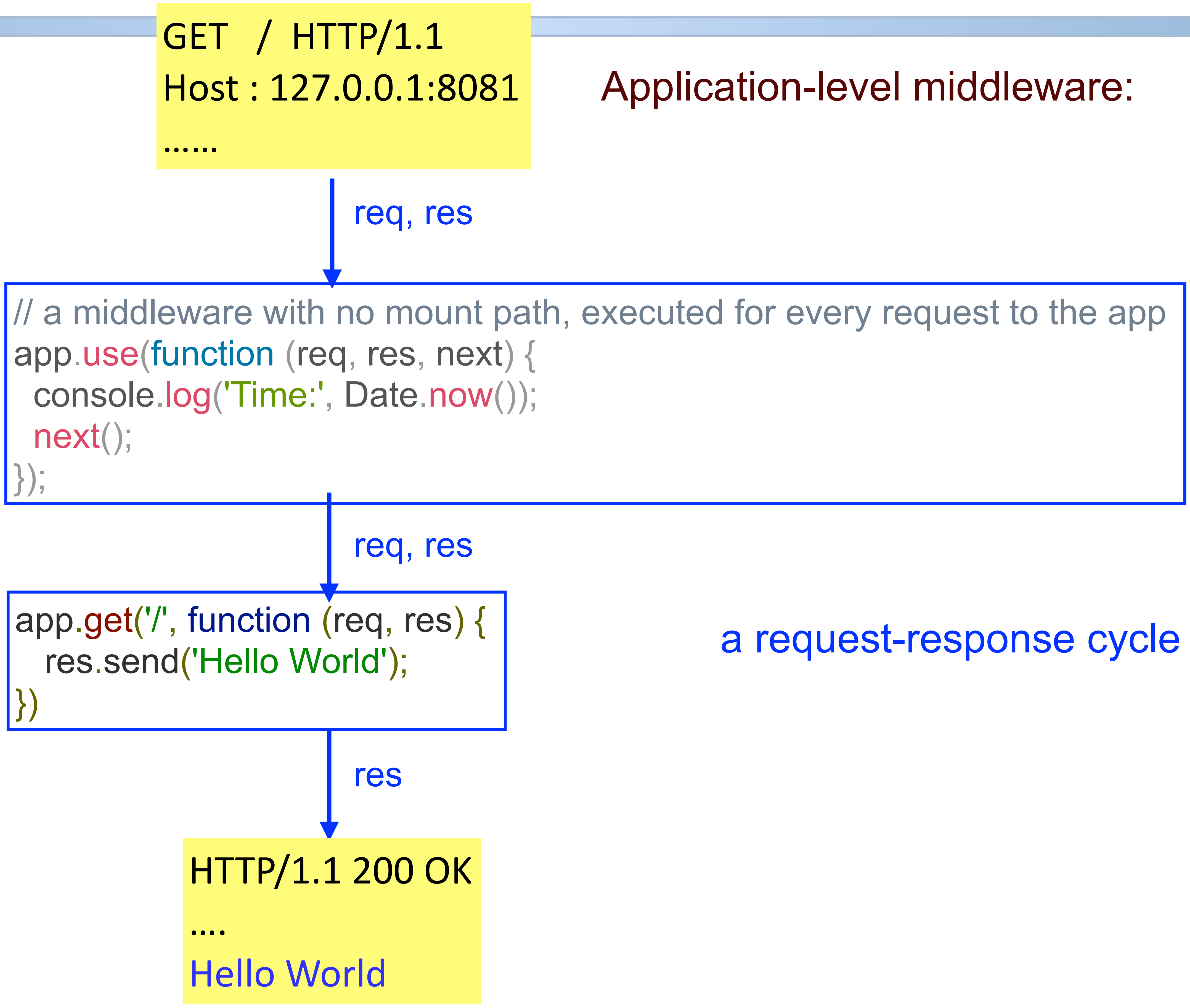
About birds

## Middleware

- A middleware is a function with access to the request object (req), the response object (res), and the next middleware in the application’s request-response cycle
- Middleware can:
  - execute any code
  - make changes to the request and the response objects
  - call the next middleware in the stack through next()
  - end the request-response cycle (if it does not call next())

# Example middleware

Application-level middleware:



```
GET   /  HTTP/1.1
Host : 127.0.0.1:8081
……
```

```
// a middleware with no mount path, executed for every request to the app
app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});
```

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

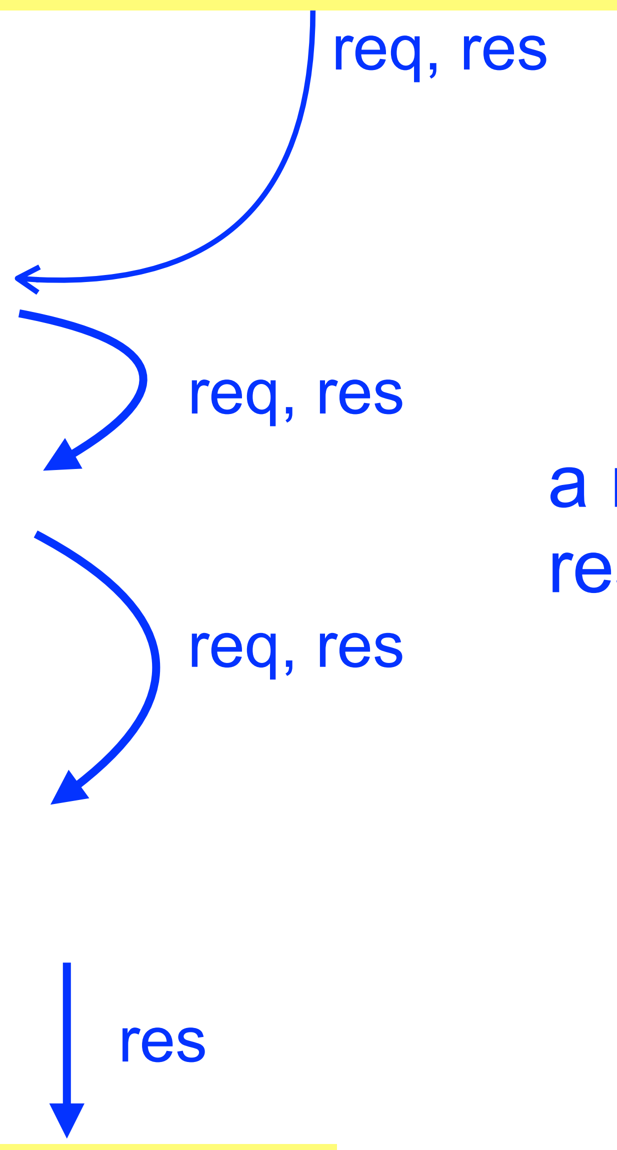
```
HTTP/1.1 200 OK
….
Hello World
```

a request-response cycle

# Example middlewares (cont’d)

Application-level middleware:

```
GET   /user/39  HTTP/1.1
Host : 127.0.0.1:8081
……
```



```
app.get('/user/:id', function (req, res, next) {
  console.log('ID:', req.params.id)
  next()
}, function (req, res, next) {
  res.write('User Info: ’)
  next()
})

app.get('/user/:id', function (req, res, next) {
  res.end(req.params.id)
})
```

end the response process

a request-response cycle

```
HTTP/1.1 200 OK
….
User Info: 39
```

## Example middlewares (cont'd)

Application-level middleware:

```
app.get('/user/:id', function (req, res, next) {
  console.log('ID:', req.params.id)
  next()
}, function (req, res, next) {
  res.write('User Info')
  next()
})

app.get('/user/:id', function (req, res, next) {
  res.end(req.params.id)
})
```

route parameter: named URL segment used to capture the value specified at its position in the URL

- the captured value is populated in the `req.params` object, with the name of the route parameter as the respective key
- e.g., if request URL is http://localhost:3000/user/39, then req.params.id = 39

## Example middlewares (cont’d)

Router-level middleware:

```
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next();
});
```

## Example middlewares (cont'd)

Error-handling middleware:

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

Error-handling middleware always takes **four** arguments. Even if you don't need to use the next object, you must specify it.

Error-handling middlewares are defined last, after other app.use() and routes calls in app.js

## Example middlewares (cont'd)

Built-in middleware:

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

the root directory from which to serve static assets

express.static is the only built-in middleware function in Express.js, responsible for serving static assets such as HTML files, images, etc.

## Example middlewares (cont’d)

Third-party middleware:

```
var express = require('express')
var app = express()
var cookieParser = require('cookie-parser')

// load the cookie-parsing middleware
app.use(cookieParser())
```

An Express application is essentially a series of middleware calls!

## An Express app serving static files in a directory

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

app.use(express.static('public'));

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)
})
```

app.js

Pass the name of the directory, which is to be marked as the location of static files, to the express.static middleware

Then the files in the directory can be retrieved directly, such as:

```
http://localhost:8081/images/kitten.jpg
http://localhost:8081/css/style.css
http://localhost:8081/hello.html
```

# An Express app serving static file and handling form data sent by GET

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

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

app.get('/process_get', function (req, res) {
   // Prepare output in JSON format
   response = {
      first_name:req.query.first_name,
      last_name:req.query.last_name
   };
   console.log(response);
   res.json(response);
})

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)
})
```

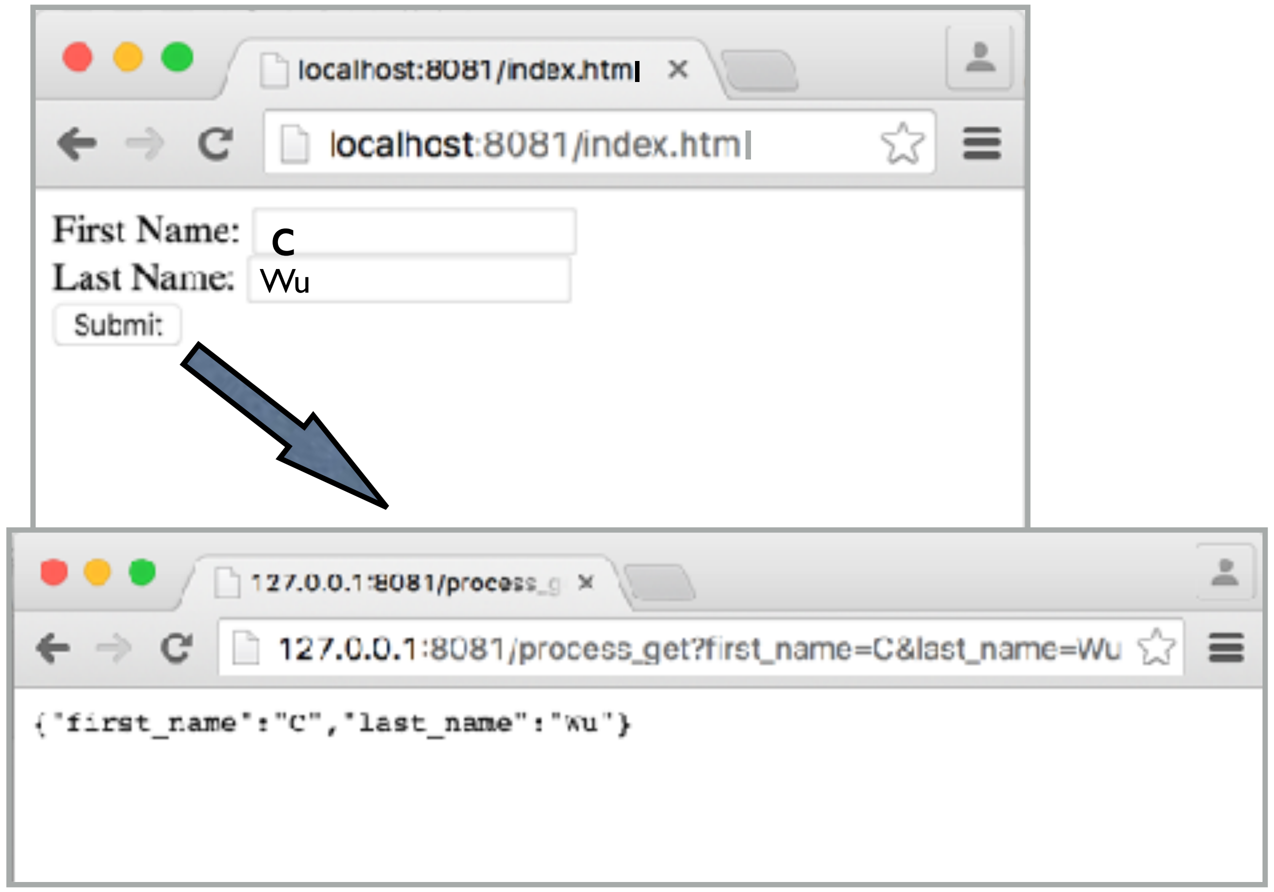
app.js

the directory in which the currently executing script resides

send response in JSON

```
<html>
<body>
<form action="http://127.0.0.1:8081/process_get" method="GET">
First Name: <input type="text" name="first_name"> <br>
Last Name: <input type="text" name=“last_name"> <br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

index.html


localhost:8081/index.html
localhost:8081/index.html
Submit
127.0.0.1:8081/process_g
127.0.0.1:8081/process_get?first_name=C&last_name=Wu

First Name: C
Last Name: Wu

```
{"first_name":"C","last_name":"Wu"}
```

# An Express app serving static file and handling form data sent by POST

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

// Create application/x-www-form-urlencoded parser
var urlencodedParser =
bodyParser.urlencoded({ extended: false })

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

app.post('/process_post', urlencodedParser, function
(req, res) {
   // Prepare output in JSON format
   response = {
       first_name:req.body.first_name,
       last_name:req.body.last_name
   };
   console.log(response);
   res.json(response);
})

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)
})
```

app.js

load body-parser module

The extended option allows to choose between parsing the URL-encoded data with the querystring library (when false) or the qs library (when true)

a middleware that parses the request body as URL encoded data and exposes the resulting object (containing the keys and values) on `req.body`

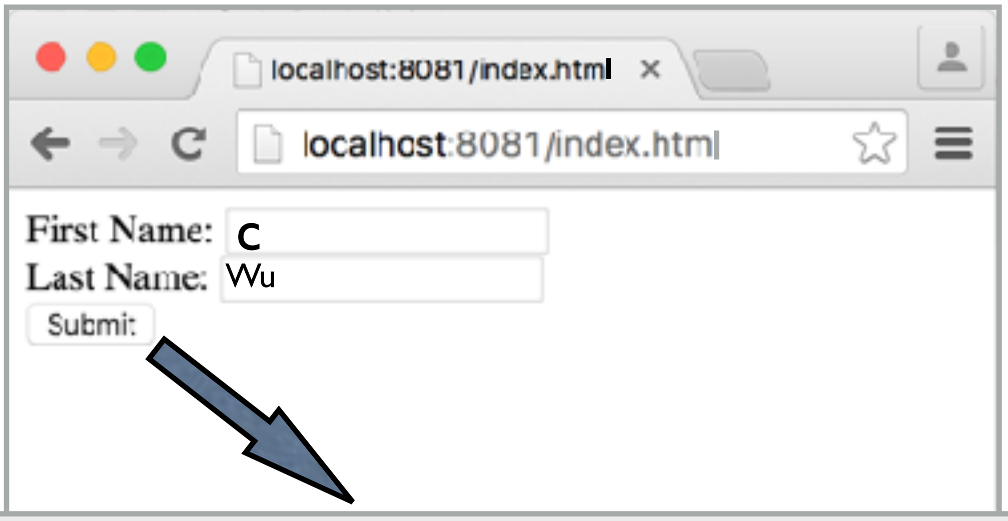
# An Express app serving static file and handling form data sent by POST (cont'd)

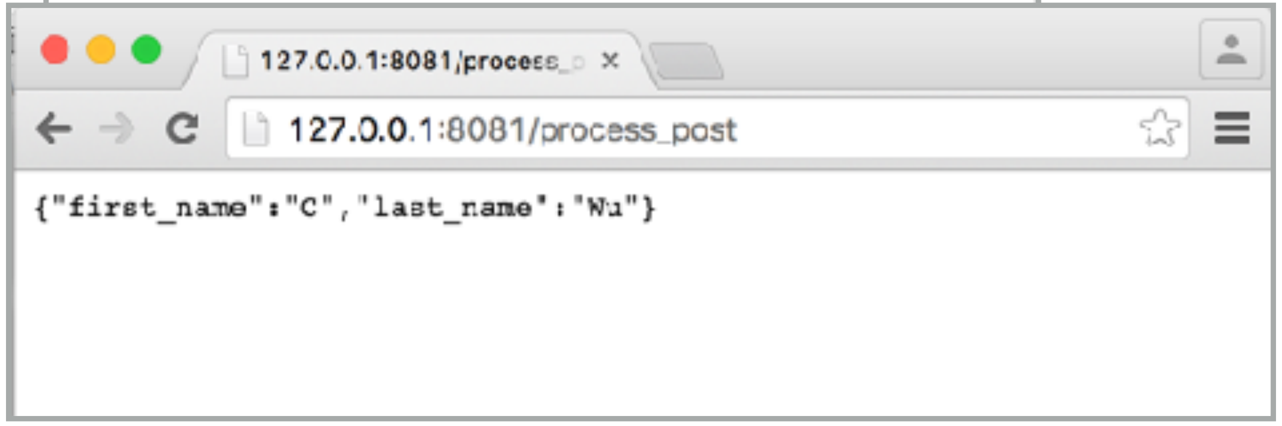
```
var express = require('express');
var app = express();
var bodyParser = require('body-parser');

// Create application/x-www-form-urlencoded parser
var urlencodedParser =
bodyParser.urlencoded({ extended: false })

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

app.post('/process_post', urlencodedParser, function
(req, res) {
   // Prepare output in JSON format
   response = {
       first_name:req.body.first_name,
       last_name:req.body.last_name
   };
   console.log(response);
   res.json(response);
})

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)
})
```

app.js

```
<html>
<body>
<form action="http://127.0.0.1:8081/process_post" method="POST">
First Name: <input type="text" name="first_name"> <br>
Last Name: <input type="text" name="last_name"> <br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

index.html





```
{"first_name":"C","last_name":"Wu"}
```

# Cookie and session

- Cookie management

```
var express  = require('express');
var app = express();
var cookieParser = require('cookie-parser');

app.use(cookieParser());

app.get('/', function(req, res){
  if (req.cookies.remember) {
    res.send('Click to <a href="/forget">forget</a>!');
  } else {
   var milliseconds = 60 * 1000;
    res.cookie('remember', 1, { maxAge: milliseconds });
    res.sendFile( __dirname + "/" + "index.html" );
  }
});

app.get('/forget', function(req, res){
  res.clearCookie('remember');
  res.redirect('back');
});

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)
})
```

load cookie-parser module

use cookieParser middleware to parse cookies in the requests

test if the cookie “remember” has been set

set cookie “remember”

unset a cookie

redirect to the previous page

# Cookie and sessions

- Session management

```
var express  = require('express');
var app = express();
var session = require('express-session');

app.use(session({secret: 'random_string_goes_here'}));

app.get('/', function(req, res){
  if (req.session.remember) {
    res.send('Click to <a href="/forget">forget</a>!');
  } else {
    req.session.remember = 1;
    res.sendFile( __dirname + "/" + "index.html" );
  }
});

app.get('/forget', function(req, res){
  req.session.remember = null;
  res.redirect('back');
});

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)
})
```

load express-session module

use session middleware to retrieve session (secret is the secret used to sign the session ID cookie)

test if the session variable “remember” has been set

set session variable “remember”

unset a session variable

## Generate dynamic pages using template engine

- A template engine produces customized web pages by combining web templates and some data source
  - a template controls the *view* of the produced web page
- Express.js can work with many template engines
  - Pug (previously named "Jade") is a commonly used template engine

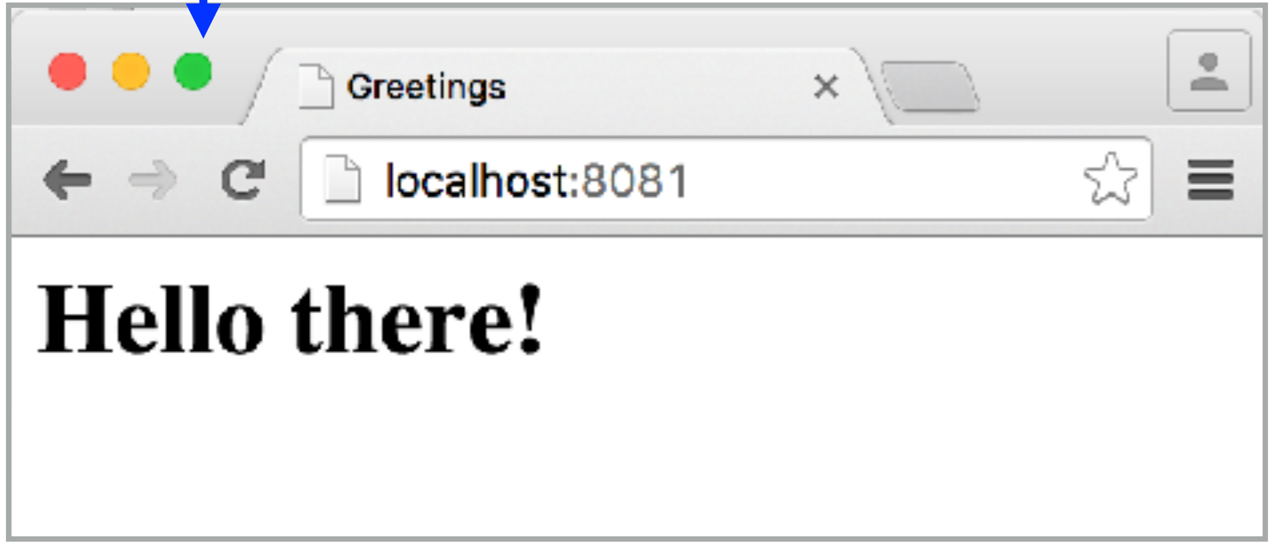
index.pug

```
html
  head
    title= title
  body
    h1= message
```

in app.js:

```
app.set('view engine', 'pug');

app.get('/', function (req, res) {
  res.render('index', { title: 'Greetings', message: 'Hello there!'});
});
```

the HTML page rendered



Hello there!

# A “Hello World” Express app with Pug

app.js

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

var index = require('./routes/index');

app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

app.use('/', index);

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)
})
```

The path module contains several helper functions to make path manipulation easier

load the router module implemented by index.js in “routes” directory

set the path to the tempates (in the “views” directory)

all requests to “/” will be handled by the “index” router module

# A “Hello World” Express app with Pug (cont’d)

app.js

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

var index = require('./routes/index');

app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

app.use('/', index);

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)
})
```

index.js

```
var express = require('express');
var router = express.Router();

router.get('/', function(req, res) {
    res.render('index', { title: 'Index', message: 'Hello
There!'});
});

router.get('/helloworld', function(req, res) {
    res.render('helloworld', { title: 'Hello World', message:
'Hello World!'});
});

module.exports = router;
```

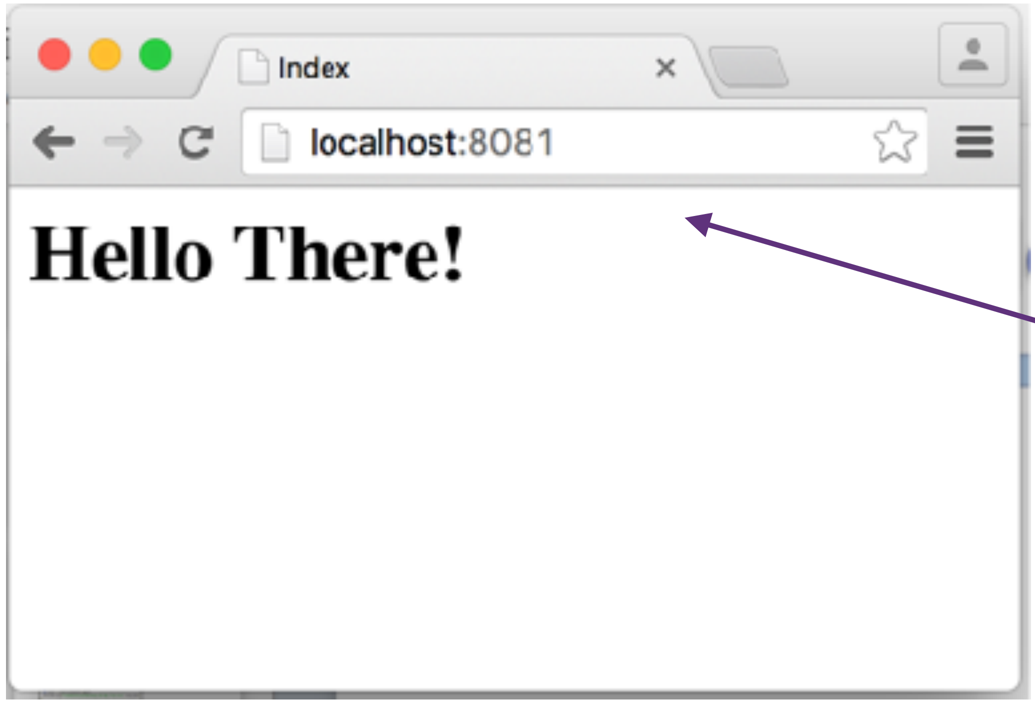
index.pug

```
html
  head
    title= title
  body
    h1= message
```

helloworld.pug

```
html
  head
    title= title
  body
    h1= message
    p Hello World! Welcome to #{title}
```

# A "Hello World" Express app with Pug (cont'd)

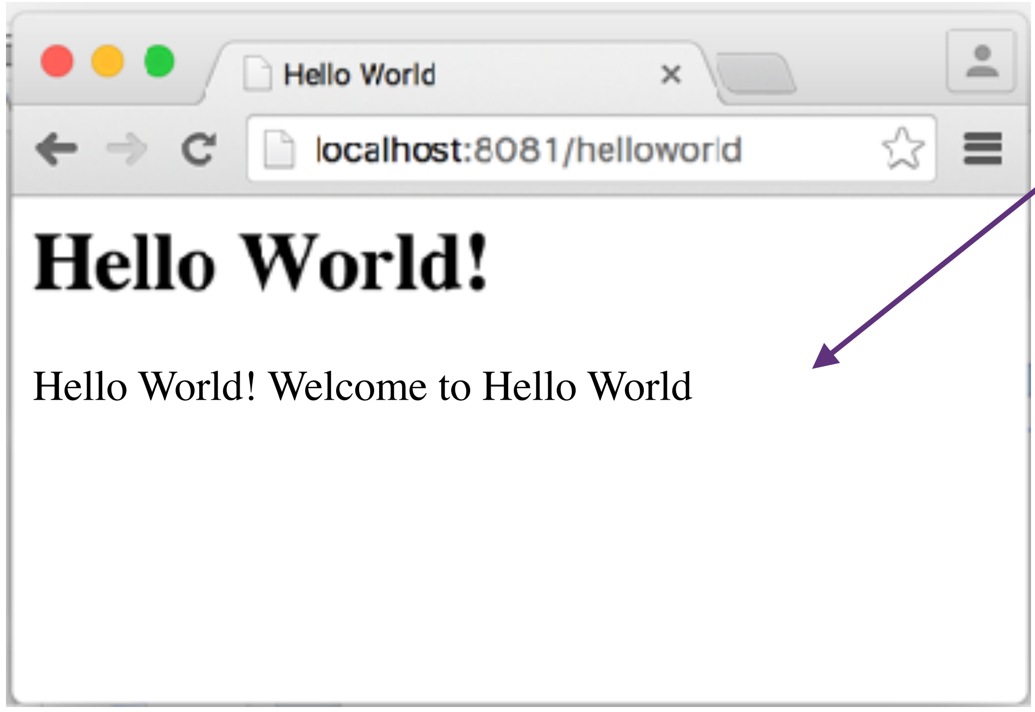

Index

localhost:8081

Hello There!

index.js

```
var express = require('express');
var router = express.Router();

router.get('/', function(req, res) {
    res.render('index', { title: 'Index', message: 'Hello
There!'});
});

router.get('/helloworld', function(req, res) {
    res.render('helloworld', { title: 'Hello World', message:
'Hello World!'});
});

module.exports = router;
```


Hello World

localhost:8081/helloworld

Hello World!

Hello World! Welcome to Hello World

index.pug

```
html
  head
    title= title
  body
    h1= message
```

helloworld.pug

```
html
  head
    title= title
  body
    h1= message
    p Hello World! Welcome to #{title}
```

## Database integration

- We can connect databases to an Express app by loading an appropriate Node.js driver for the database in the app

- Database systems that an Express app can connect to
  - Cassandra, MySQL, MongoDB, Oracle, PostgreSQL, Redis, SQL Server, SQLite, ElasticSearch, etc.

## MongoDB

- A free and open-source cross-platform document-oriented database
- A NoSQL database, which uses JSON-like documents
  - a MongoDB server typically has multiple databases, and each database is a container for collections
  - a collection is a group of MongoDB documents, similar to a table in a relational database system (e.g., MySQL).
  - a document is a set of key-value pairs
  - documents within a collection can have different fields, and typically all documents in a collection are of similar or related purpose

## MongoDB (cont’d)

- Example MongoDB document

```
{
   _id: ObjectId(7df78ad8902c),
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by: ‘example.inc',
   url: 'http://www.exampleinc.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100,
   comments: [
      {
         user:'user1',
         message: 'My first comment',
         dateCreated: new Date(2011,1,20,2,15),
         like: 0
      },
      {
         user:'user2',
         message: 'My second comments',
         dateCreated: new Date(2011,1,25,7,45),
         like: 5
      }
   ]
}
```

# A RESTful Web service by Express.js

app.js

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

// Database
var mongo = require('mongodb');
var monk = require('monk');
var db = monk('localhost:27017/test1');

var index = require('./routes/index');
var users = require('./routes/users');

var app = express();

app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

// Make db accessible to router
app.use(function(req,res,next){
    req.db = db;
    next();
});

app.use('/', index);
app.use('/users', users);

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)
})
```

load the MongoDB module

monk is a layer that provides simple yet substantial usability improvements for MongoDB usage within Node.js

get the database instance: the MongoDB is running on localhost at port 27017; the database name is “test1”

a middleware that parses JSON (to support parsing of application/json type data in request body), and the parsed data is exposed on req.body

a middleware that parses URL encoded data (to support parsing of application/x-www-form-urlencoded type data) and exposes the parsed data on req.body

# A RESTful Web service by Express.js (cont’d)

index.js

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: ‘RESTful Web Service’ });
});

module.exports = router;
```

index.pug specifies the page layout/content, and possibly links to external files (e.g., .js, .css)

# A RESTful Web service by Express.js (cont'd)

users.js

```
var express = require('express');
var router = express.Router();

/*
 * GET userlist.
 */
router.get('/userlist', function(req, res) {
    var db = req.db;
    var collection = db.get('userlist');
    collection.find({},{},function(e,docs){
        res.json(docs);
    });
});

/*
 * POST to adduser.
 */
router.post('/adduser', function(req, res) {
    var db = req.db;
    var collection = db.get('userlist');
    collection.insert(req.body, function(err, result){
        res.send(
            (err === null) ? { msg: '' } : { msg: err }
        );
    });
});
```

```
/*
 * DELETE to deleteuser.
 */
router.delete('/deleteuser/:id', function(req, res) {
    var db = req.db;
    var collection = db.get('userlist');
    var userToDelete = req.params.id;
    collection.remove({ '_id' : userToDelete }, function(err) {
        res.send((err === null) ? { msg: '' } : { msg:'error: ' + err });
    });
});

module.exports = router;
```

returns all records in the collection (first two parameters of collection.find give record and field to be retrieved; empty then the whole collection)

https://mongodb.github.io/node-mongodb-native/api-generated/collection.html

# A RESTful Web service by Express.js (cont'd)

Example client-side jQuery code for accessing the web service (e.g., contained in a .js file that index.pug links to):

```
 $.getJSON( '/users/userlist', function( data )
{
      // process or display the data
      ….
    });
```

```
$.ajax({
            type: 'POST',
            data: newUser,
            url: '/users/adduser',
            dataType: 'JSON'
        }).done(function( response ) {
             // actions upon receiving response
            …
        });
```

```
$.ajax({
           type: 'DELETE',
           url: '/users/deleteuser/' + $(this).attr('id')
       }).done(function( response ) {
     // actions upon receiving response
       });
```

***We will practise more about creating REST Web service using Express.js in the lab exercise***

- References
  - http://expressjs.com
  - https://nodejs.org/en/
  - https://pugjs.org/api/getting-started.html
  - https://www.mongodb.org